F. Sh. Jo'rayev

ASSEMBLER TILI VA KOMPYUTERDAGI JARAYONLAR

Toshkent – 2012

UDK: 004.431.4
KBK 32.973
I-98

I-98 **F.Sh. Jo'rayev. Assembler tili va kompyuterdagi jarayonlar.**
O'quv qo'llanma.–T.: «Fan va texnologiya». 2012, 272 bet.

Mazkur o'quv qo'llanma Assembler dasturlash tilining mazmun va mohiyatini atroflicha bayon etishga qaratilgan. Undagi mavzularda Assemblerning eng oddiy tushunchalaridan tortib nozik tomonlari va jarayonlarigacha qamrab olingan. Shuningdek, kitobda kompyuterdagi jarayon va operatsion tizimning Assembler tiliga taalluqli jihatlari ham yoritilgan.

O'quv qo'llanma axborot texnologiyalari sohasida mutaxassis bo'lib chiqadigan oliy o'quv yurti talabalari, magistrantlari, kasb-hunar kollejlari o'quvchilari hamda kompyuter va dasturlash sohasini mustaqil o'rganuvchilarga mo'ljallangan.

O'quv qo'llanmadan kompyuter muhandislari, dasturchilari, dasturchi-muhandislar, axborot xavfsizligi xodimlari, antivirus ishlab chiqaruvchilar va kompyuterda ishlash bilan qiziquvchilar foydalanishlari mumkin.

UDK: 004.431.4
KBK 32.973

Taqrizchilar:
Sh. Qayumov – fizika-matematika fanlari nomzodi, dotsent;
B. Usmonov – texnika fanlari nomzodi, dotsent.

Ushbu o'quv qo'llanmani muallifning yozma ruxsatisiz qayta nashr qilish, tarqatish, tarjima qilish yoki uning biror qismini fotonusxa, har qanday elektron yoki mexanik yozish va saqlash uskunalarida tarqatish qat'iyan taqiqlanadi.

ISBN 978–9943–10–697–0

© F.Sh.Jo‘rayev, 2012.
© «Fan va texnologiya» nashriyoti, 2012.

Mundarija

Muqaddima 7
Kitob tuzilishi 9
Kitobdagi shartli belgilar 9
Aloqalar 9

I bob. Dastlabki tushunchalar
1.1. Assembler nima uchun kerak? 11
1.2. NASM nima? 12
1.3. Operatsion tizim nima? 13
1.3.1. Linux tizimi 14
1.3.2. Windows tizimi 16
1.3.3. Mac OS X tizimi 16

II bob. Tarixiy ma'lumotlar
2.1. Birinchi hisob mashinalari 17
2.2. Charls Bebbij tajribalari 17
2.3. Birinchi avlod EHMlari: elektron lampalar va elektr o'zgartiruvchi devorchalar 18
2.4. Ikkinchi avlod EHMlari: tranzistorlar va to'plamli qayta ishlash tizimlari . . . 18
2.5. Uchinchi avlod EHMlari: ko'p vazifalik 19
2.6. To'rtinchi avlod EHMlari: shaxsiy kompyuterlar 20

III bob. Sanoq tizimlari
3.1. Ikkilik sanoq tizimi 21
3.2. Sakkizlik sanoq tizimi 22
3.3. O'nlik sanoq tizimi 22
3.4. O'n oltilik sanoq tizimi 22
3.5. Bir sanoq tizimidan ikkinchisiga o'tish usullari 23
3.5.1. O'nlik sanoq tizimiga o'tish 24
3.5.2. Ikkilik sanoq tizimiga o'tish 24
3.5.3. Sakkizlik sanoq tizimiga o'tish 25
3.5.4. O'n oltilik sanoq tizimiga o'tish 26
3.6. Har xil sanoq tizimlarida kasr sonlar 26

IV bob. Kompyuter tuzilishi
4.1. Markaziy protsessor 28
4.2. Xotira 29
4.3. O'quv/yozuv moslamalari 32
4.4. Axborot tashish kanallari 34
4.5. Intel protsessor turkumlari 36
4.6. Markaziy protsessor registrlari 37

4.6.1. 16 bitli registrlar 37
4.6.2. 32 bitli registrlar 38
4.7. Sonli qo'shma protsessor registrlari 40
4.8. Xotirani manzillash usullari 40
4.8.1. Haqiqiy usul 41
4.8.2. 16 bitli himoyalangan usul 42
4.8.3. 32 bitli himoyalangan usul 43
4.9. Markaziy protsessorda uzilishlar tushunchasi 43

V bob. Assembler tilida dasturlash
5.1. Mashina tili 45
5.2. Assembler tili 45
5.3. O'zgaruvchilarni e'lon qilish 46
5.4. O'zgarmaslar, ifodalar va direktivalar 49
5.5. Asosiy amallar va ular qabul qiladigan qiymatlar 54
5.6. Assemblerda o'quv/yozuv 58
5.7. Birinchi dastur 61
5.8. Dasturni kompilyator orqali yig'ish 63
5.9. Obyekt fayllarni ulash 64
5.10. Buyruqlar qatorida NASM buyrug'i kalitlari 65

VI bob. Assembler tili asoslari
6.1. Ishorali va ishorasiz butun sonlar 69
6.2. Butun sonlar ustida arifmetik amallar 71
6.2.1. Qo'shish va ayirish 71
6.2.2. Ko'paytirish va bo'lish 74
6.3. Qiymatlar o'lchamini kattalashtirish va kichiklashtirish 77
6.4. Boshqaruv buyruqlari 79
6.4.1. Taqqoslash buyrug'i 80
6.4.2. Tarmoqlash buyruqlari 80
6.4.3. `LOOP` buyrug'i 85
6.5. Amaliy dastur: Tub sonlarni topish 86

VII bob. Mantiqiy amallar va bitlarni siljitish buyruqlari
7.1. Bitlarni siljitish buyruqlari 89
7.1.1. Mantiqiy siljitish 90
7.1.2. Arifmetik siljitish 90
7.1.3. Halqasimon siljitish 91
7.2. Dasturlashda siljitish buyruqlaridan foydalanish 93
7.3. Bitlar ustida mantiqiy amallar 94
7.3.1. `AND` amali 94
7.3.2. `OR` amali 95
7.3.3. `XOR` amali 96
7.3.4. `NOT` amali 96
7.3.5. `TEST` buyrug'i 97
7.4. Mavzular yuzasidan mashqlar 97

VIII bob. Qismli dasturlash
8.1. Vositali manzillash 101
8.2. Qismli dastur tuzish 102
8.3. Stack 107
8.4. `CALL` va `RET` buyruqlari 109
8.5. Mahalliy nishonlar 110
8.6. Stack orqali qiymatlar jo'natish 112
8.7. `ENTER` va `LEAVE` buyruqlari 117
8.8. `LEA` buyrug'i 119
8.9. Assembler va C dasturlarini bog'lash 124
8.10. Qism dasturlarida qayta kirish va o'z-o'zini chaqirish tushunchalari 128
8.11. Chuqur o'rganuvchilar uchun amaliy dastur 132

IX bob. Makro vositalar
9.1. Bir satrli makrolar 136
9.1.1. `%DEFINE` makro direktivasi: Umumiy uslub 136
9.1.2. `%XDEFINE` makro direktivasi: Kengaytirilgan uslub 139
9.1.3. `%UNDEF` makro direktivasi: Aniqlangan makrolarni bekor qilish . . . 140
9.1.4. `%ASSIGN` direktivasi: Makro o'zgaruvchilar 140
9.1.5. `%DEFSTR`, `%STRLEN` va `%SUBSTR` makro direktivalari: Qatorli o'zgarmaslar bilan ishlash 140
9.1.6. `%INCLUDE` makro direktivasi: Boshlang'ich fayllarni dasturga qo'shish 141
9.2. Makro takrorlanish 141
9.3. Ko'p satrli makrolar 143
9.3.1. Qiymat berish va qabul qilish usullari 145
9.3.2. `%ROTATE` makro direktivasi 147
9.3.3. Makrolarda mahalliy nishonlar 149
9.4. Dasturni shartli yig'ish 151
9.4.1. Makroni aniqlanganligini tekshirish 153
9.4.2. Matn tengligini tekshirish 154
9.4.3. Qiymat turlarini tekshirish 155
9.5. Mezoniy makrolar 156
9.6. Amaliy dastur: Lotindan kirillga 159

X bob. Assemblerda qoliplar va jadvallar
10.1. Qoliplarni e'lon qilish 164
10.2. Qolip kataklariga murojaat etish 165
10.3. Jadvallar 167
10.4. Jadvallarni e'lon qilish 170
10.5. Jadval kataklariga murojaat qilish 171
10.6. Qoliplapga mo'ljallangan buyruqlar 173
10.7. Amaliy dasturlar: Saralash 179

XI bob. O'nli kasrlar
11.1. Sonli qo'shma protsessor 183

11.2. O'nli kasr sonlar andozasi 183
11.3. Sonli qo'shma protsessor registrlari 185
11.4. Sonli qo'shma protsessor asosiy buyruqlari 187
11.4.1. Qiymatni yuklash va o'qish buyruqlari 187
11.4.2. Qo'shish va ayirish buyruqlari 189
11.4.3. Ko'paytirish va bo'lish buyruqlari 190
11.4.4. Taqqoslash buyruqlari 192
11.5. Qo'shimcha buyruqlar 193
11.6. Amaliy dastur: Kvadrat ildiz 195

XII bob. Fayllar bilan ishlash
12.1. Operatsion tizimda fayllar tushunchasi 198
12.2. Tizim chaqiriqlari 202
12.3. Fayllar bilan assemblerda ishlash 204
12.3.1. Xususiy holat: Linux 205
12.3.2. Xususiy holat: Windows 212
12.4. Amaliy dastur: Lotindan kirillga 217

Ilova A. Buyruqlar qatorida ishlash
A.1. Buyruqlar umumiy ro'yxati 222
A.2. Linux tizim buyruqlari 223
A.2. Windows tizim buyruqlari 225

Ilova B. Nasmni o'rnatish
B.1. NASMni Linuxda o'rnatish 228
B.1.1. Debian yoki Ubuntu foydalanuvchilari uchun 228
B.1.2. Fedora, SUSE, Mandrive, Cent OS, Doppix yoki Red Hat foydalanuvchilari uchun 229
B.1.3. Umumiy usul 229
B.2. NASMni Windowsda o'rnatish 230
B.2.1. Birinchi usul 230
B.2.2. Ikkinchi usul 231
B.2.3. Umumiy usul 231

Ilova C. C kompilyatorlari va obyekt fayllarni ular orqali ulash
C.1 Linuxdagi mavjud C kompilyatorlari 232
C.1.1. GCC kompilyatori 232
C.2. Windowsdagi mavjud C kompilyatorlari 232
C.2.1. GCC kompilyatori 232
C.2.2. Microsoft Visual Studio 233
C.2.3. LCC-WIN32 kompilyatori 234

Ilova D. ***nasm-io.inc*** **fayli** 235
Ilova E. Buyruqlar ro'yxati 238
Ilova F. Atamalar 266
Ilova G. Foydalanilgan adabiyotlar 270

Muqaddima

Bugungi kunga kelib elektrotexnika vositalarining shiddat bilan rivojlanayotgani va tobora kundalik hayotimizning ajralmas qismiga aylanib borayotgani hech kimga sir emas. Bunda axborot texnologiyalarining o'rni beqiyos, chunki elektrotexnik vositalarni boshqarish, ularni inson foydalanishi uchun qulay holga keltirishda axborot texnologiyalaridan keng foydalaniladi. Dunyoda axborot texnologiyalari taraqqiy etib borgan sari uni takomillashtiradigan mutaxassislarga bo'lgan talab ham ortib boraveradi. XXI asrni axborot texnologiyalari asri deyilayotgani ham bejiz emas. O'zbekiston Prezidentining «Kompyuterlashtirishni yanada rivojlantirish va axborot - kommunikatsiya texnologiyalarini joriy etish to'g'risida» gi Farmoni va Vazirlar Mahkamasining «Kompyuterlashtirishni yanada rivojlantirish va axborot – kommunikatsiya texnologiyalarini joriy etish chora-tadbirlari to'g'risida»gi Qarorida bu sohadagi vazifalar belgilab berilgan. Bu vazifalarni amalga oshirish maqsadida mamlakatimizda ko'plab litseylar, kollejlar, institutlar va ularning bo'limlari tashkil topdi, jamiyatning kompyuter savodxonligini oshirishga e'tibor yanada kuchaytirildi.

Mazkur kitobning asosiy maqsadi ham yuqorida aytib o'tilgan ta'lim maskanlarida tahsil olayotgan talabalarga va o'quvchilarga axborot texnologiyalarining dasturlash yo'nalishi bo'yicha keng va batafsil ma'lumot berishdan iborat bo'lib, unda dasturlash tillarining quyi tabaqasiga mansub bo'lgan *Assembler tilining mazmuni va mohiyati* yoritiladi.

Hozirgi kunga kelib kompyuterning behisob imkoniyatlari ma'lum bo'lmoqda. Undagi dasturlar orqali bajariladigan ishlar kishini hayratga solmoqda. Bularning barchasiga o'ta yuqori darajadagi *mavhumlik asosida* erishilgan. Masalan, kundalik hayotimizda ko'p duch keladigan oddiy rasmlarni olaylik. Biz ularni osonlik bilan kompyuterga yuklaymiz va u yerda tomosha qilamiz, shuningdek, maxsus dasturlar yordamida o'zgartiramiz. Ammo fizikaviy nuqtai nazardan qaralganda kompyuter ichida hech qanday rasm yo'q! Kompyuter elektr zanjirlari-yu yarim o'tkazgichlar birlashmasi xolos va u yerda umuman rasm haqida gap ham bo'lishi mumkin emas. Shunday ekan, kompyuterda saqlaydigan barcha narsalarimiz (rasmlar, filmlar, kitoblar, hujjat matnlari, dasturlar, umuman olganda fayllar) mavhum holda kompyuter xotirasida mavjuddir, ya'ni ular *fizikaviy kattalikka ega emas*. Agar kompyuterni ta'kidlanganidek yarim o'tkazgich-u va shunga o'xshash «temir-tersak» deb hisoblasak, barcha o'ziga xos mavhumliklar temirga, ya'ni kompyuterga faqatgina *mashina tilida yozilgan dasturlar yordamida «tushuntiriladi»*. Boshqacha qilib aytganda, *barcha kompyuter qurilmalari mashina tilida tuzilgan dastur orqali boshqariladi*. Demak, mavhumlik amalda kompyuterda aynan qanday ifodalanishini yaxshi anglash uchun birinchi navbatda mashina tilini o'rganish kerak. Ammo *mashina tili ikkilik sanoq tizimidagi sonlar ketma-ketligidan iborat* bo'lib, uni inson tushunishi yoki unda dasturlashi juda qiyin. Assembler tili esa mana shu *muammoni yechish maqsadida* chiqarilgan.

Assembler dasturlash tili mashina tiliga juda yaqin (o'xshash) bo'lib, unda faqat tushunarsiz sonlar o'rnida inson uchun tushunarli bo'lgan harfli ifodalar va nomlar ishlatiladi. Assembler tilida yozilgan dastur esa kompilyator nomli maxsus dastur yordamida *mashina tiliga o'giriladi*.

Assembler quyi tabaqa dasturlash tili ekanligi aytildi, ammo quyi so'zini past saviyali degan ma'noda tushunmaslik kerak! Gap shundaki, dasturlash tillarining o'z qatlamlari bo'lib, unda mashina tiliga yaqin tillar quyi tabaqa tillari deb yuritiladi. Aksincha, dasturlash olamida Assembler tili asos sifatida tan olinadi. Deyarli barcha *qolgan dasturlash tillari Assemblerda «qurilgan»*. Shunday ekan, Assembler tilini o'rganish dasturchi uchun poydevor vazifasini o'taydi.

Kompyuterda dastur ishga tushganda uning ichida nimalar sodir bo'lishi haqida tushunchaga ega bo'lish dasturchi uchun juda muhim, chunki shundagina u masalaning tub mohiyatini anglashi mumkin. Bunday holatni esa faqat assembler tilida uchratishimiz mumkin. Yuqori tabaqa dasturlash tillariga mansub bo'lmish C/C++, Paskal, PHP, Perl, Python yoki Java tillaridan farqli o'laroq, Assemblerda dasturchi protsessorda bo'layotgan ichki jarayonlarga o'zi guvoh bo'ladi. Xulosa qilish mumkinki, *kuchli dasturchi bo'lishni istagan va uni ilmiy asosda egallamoqchi bo'lgan har bir dasturchi Assembler tilini tushunishi shart!*

Mazkur kitob Assembler dasturlash tili imkoniyatlarini hisobga olgan holda va ushbu sohada o'zbek tilida manbalar deyarli yo'qligi sababli yozildi. Umuman olganda Assembler bo'yicha butun dunyodagi mavjud adabiyotlarining ham ko'pchiligi ancha eskirgan, chunki ular ancha avvalgi rusumlardagi kompyuterlarga mo'ljallab yozilgan. Masalan, Assembler haqidagi kitoblarning asosiy qismi eski 16 bitli tartibda ishlaydigan protsessorlarga mo'ljallangan MS-DOS operatsion tizimida dasturlashni qamrab olgan. Boshqalari esa tor yo'nalishda bo'lib, biror operatsion tizim uchun yozilgan. Ushbu *kitob esa eng so'nggi protsessor rusumida ishlaydigan eng zamonaviy operatsion tizimlarning imkoniyatlarini hisobga olib yozilgan*.

Kitob *Assembler dasturlash tilini o'quvchiga chuqurroq o'rgatishga qaratilgan*. Kitobdagi mavzularda Assemblerning eng oddiy tushunchalaridan boshlab, dasturlashning nozik tomonlari va murakkab jarayonlarigacha qamrab olingan. Dasturlashdan tashqari kitobda kompyuterdagi jarayonlari va operatsion tizimning Assembler tiliga taalluqli jihatlari ham yoritilgan. Dasturlash tili xususiyatlari, imkoniyatlari shunchaki sanab o'tilmay, balki muammoni o'quvchi oldiga qo'yish, so'ng uning yechimni taqdim etish tartibida tushuntirilgan. Kitob muallifning dunyodagi ilg'or oliy ta'lim maskanlarida ushbu soha qanday o'qitilishini va bu sohada yozilgan turli tillardagi adabiyotlarni o'rganib chiqishi asosida izchil *didaktik qonuniyatlar va mezonlarga amal qilgan* holda yozilgan.

Hozirgi davrda bir qancha protsessor oilalari mavjud bo'lib, quyida x86 oilasiga mansub Intel va AMD yoki shu oilaga mansub boshqa protsessorga mo'ljallangan Assembler tili yoritiladi. Bunga sabab bu turdagi protsessorlar hozirgi kunda nisbatan arzon va juda keng tarqalgan hisoblanadi. Operatsion tizimga kelsak, Assembler tili unga unchalik ham bog'liq emas. Deyarli barcha operatsion tizimlar Assembler tilida dasturlash imkoniyatini beradi. Ushbu kitobdagi amaliy dasturlar asosan umumiy ko'rinishda keltiriladi va ularning Windows, Linux va Mac OS X tizimlarda ishlashi nazarda tutiladi. Ma'lum bir operatsion tizim bilan bo'ladigan xususiy holatlar esa alohida ogohlantirish orqali beriladi.

Assembler tili qaysi kompilyatorga asoslanganiga qarab har xil ko'rinishda bo'ladi. Ammo bir necha xil assembler tillari mavjud bo'lsada, ularning hammasining asosi bir xildir va bir-biridan kam farq qiladi. Asosiy farq dasturni yozish qoidalarida bo'ladi. Mazkur kitobda NASM kompilyatoriga asoslangan Assembler tili o'rgatiladi. Chunki ushbu kompilyator juda mashhur va dasturchilar tomonidan hozirgi kungacha yangilanib qo'shimcha imkoniyatlari orttirilib kelinadi va u barcha operatsion tizimlarda ishlaydi. Lekin kompilyator tanlash imkoniyati o'quvchining o'ziga havola. Chunki *kitobdagi go'yalarni yaxshi o'zlashtirgan o'quvchi istalgan kompilyatorga mo'ljallangan assembler tilini ortiqcha qiyinchiliklarsiz tushuna oladi*.

Kitob qo'lyozmasini tayyorlash chog'ida ba'zi tushuncha va fikrlarni aniqlash va takomillash-tirishdagi qimmatli maslahatlari uchun taqrizchilar fizika-matematika fanlari nomzodi, dotsent Sh. Qayumovga, texnika fanlari nomzodi, dotsent B. Sh. Usmonovga minnatdorchilik bildiraman. Shuningdek, dasturchi Sh. Rahmatovning foydali maslahatlarini ham ta'kidlamoqchiman.

Kitob tuzilishi

Kitobdagi tushunchalar oddiydan murakkabga qarab yo'nalishida berilgan. Har mavzu boshida masalaning nazariy tomonlari tushuntirilib, so'ng amaliy yechimlarga o'tilgan. Mavzular esa o'zaro bog'liq qilib tuzilgan, ya'ni *bir mavzu bilan tanishmay turib keyingisini anglash mushkul*.

Kitob 12 bobdan iborat. Birinchi bobda Assembler tili bilan bevosita bog'liq bo'lmagan, ammo dasturchi xabardor bo'lishi talab qilinadigan ma'lumotlar bayon etilgan. Ikkinchi bobda axborot texnologiyalaridan tarixiy ma'lumotlar berilgan. Uchinchi bobda sanoq tizimlari tushuntirilgan. To'rtinchi bobda kompyuterning tuzilishi va unda dasturlarning ishlashi yoritilgan. Beshinchi bobda Assembler tili bilan yaqindan tanishilgan va unda birinchi dastur tuzib ko'rsatilgan. Oltinchi bobda Assembler tilining asosiy qonuniyatlari berilgan. Yettinchi bobda mantiqiy amallar bilan ish olib borilgan. Sakkizinchidan o'ninchigacha bo'lgan boblarda dasturlashda qo'llanadigan asosiy algoritmlar tushuntirilgan. O'n birinchi bobda o'nli kasr sonlar va ular ustida matematik amallarning Assemblerdagi ifodasi berilgan. O'n ikkinchi bobda esa dastur orqali qattiq diskdagi fayllar bilan ishlash o'rgatilgan.

Kitob so'ngida ancha keng ko'lamli ilova berilgan bo'lib, u yerda tashkiliy ishlarga molik masalalar tushuntirilgan. Masalan, qanday qilib tuzilgan dasturlarni ishga tushirish, Assembler kompilyatorini o'rnatish va boshqa shunga o'xshash tomonlar shular jumlasidandir.

Bundan tashqari kitobdagi barcha namunaviy dasturlar va ularni ishga tushirishda kerak bo'ladigan boshqa dasturlar fayllari ushbu kitobga bag'ishlangan assembler.zn.uz saytida berilgan.

Kitobdagi dasturiy namunalar Windows XP, Windows Vista, Windows 7, Debian, Ubuntu va Fedora operatsion tizimlarida sinab ko'rilgan.

Kitobdagi shartli belgilar

Kitobda o'quvchi uchun turli xil shartli belgilar orqali muntazam yo'llanmalar va sharhlar berib boriladi. Quyidagi shartli belgilar foydalanilgan.

paging unit	Chet el atamasi yoki maxsus tushuncha birinchi marta uchraganda yoki ma'lum iboraga urg'u berishda qo'llanadi.
`mov eax , 10`	Dastur kodini yoki buyruqlar qatoridagi amallarni bildiradi.
Eslatma: ...	Bilish muhim va eslab qolish foydali bo'lgan matn.
Start→Control Panel→ ...	Operatsion tizimda ma'lum harakatlar ketma-ketligini bajarishni ko'rsatadi. Masalan, ushbu holatda avval «Start» tugmasi bosilib, so'ng u yerdan «Control Panel» tanlanadi.
←	O'zgaruvchiga qiymat o'zlashtirilishini bildiradi.

Aloqalar

Barcha joyda bo'lgani kabi ushbu kitobda ham xato va kamchiliklar bo'lishi mimkin. Ayniqsa, amaliy mashg'ulotlar uchun berilgan dasturiy namunalarda uchraydigan xatolardan butunlay qutulishning iloji yo'q. Ammo o'quvchi uchun bu xatolarga amaliyotda duch kelishning foydasi

ham bor. Chunki bu xatolarni tuzatish o'rganuvchi uchun katta tajriba maktabi hisoblanadi. Lekin biz bu bilan kitobdagi bo'lishi mumkin bo'lgan xatolarni oqlashdan yiroqmiz. Shunday holatlarga duch keladigan hushyor o'quvchidan biz bilan aloqa o'rnatib xato va kamchiliklar to'g'risida xabar berishlarini juda xohlar edik. Bizning elektron manzilimiz siz aziz mutolaachidan keladigan xatlar xizmatida bo'ladi:

assembler.uzbek@yahoo.com

Qo'shimcha tarzda ushbu kitob uchun ochilgan internet sayt mavjud bo'lib, u yerdan siz Assembler va dasturlash haqida muallif tomonidan yozib boriladigan maqola va yangiliklarni topishingiz mumkin. Web sahifamiz quyidagicha:

http://www.assembler.zn.uz

Sim qoqib ulanuvchilar uchun raqamimiz:

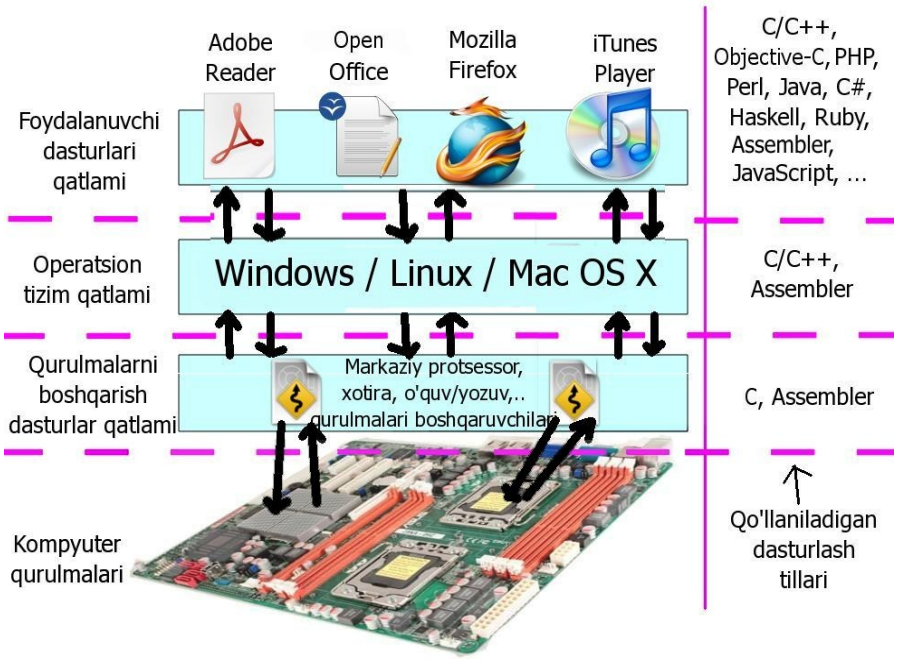
+998-97-708-02-35

I bob
Dastlabki tushunchalar

Mazkur bobda biz ushbu fanni o'zlashtirishimiz uchun oldindan kerak bo'ladigan dastlabki tushunchalar bilan tanishib chiqamiz.

1.1. Assembler nima uchun kerak?

Assembler tili dasturlashda alohida o'ringa ega. Assembler tilidagi dastur to'g'ridan-to'g'ri markaziy protsessor buyruqlaridan foydalangan holda tuziladi. Bu esa dasturchiga protsessor imkoniyatlarini chinakam bilishga yordam beradi. Yuqorida ta'kidlanganidek, Assembler tilidan asosan kompyuter qurilmalari bilan to'g'ridan-to'g'ri ishlaydigan va ularni boshqaradigan dasturlar (drayverlar) tuzishda foydalaniladi. Masalan, mashina ishlab chiqaradigan korxonalardagi avtomatlashgan robotlardan boshlab elektron meditsina asboblarigacha barchasi uchun yoziladigan boshqaradigan dasturlar shular jumlasidandir. Ammo hozirgi mavjud barcha qurilmalar uchun boshqaruvchi dasturlar yozilgan-ku, endi Assemblerni o'rganish nimaga kerak degan savol tug'ilishi mumkin. To'g'ri, lekin hayot hozirda mavjud kompyuter imkoniyatlari bilan to'xtab qolmaydi. Kundan-kunga yangi qurilmalar ishlab chiqilayapti va ularni boshqaruvchi dasturlarni tuzish uchun Assembler dasturchilari kerak bo'ladi.



1-rasm.

Bundan tashqari dasturchiga Assemblerda dasturlash ehtiyoji bo'lmaganda ham uni bilib qo'yishning yana bir ustunlik jihati bor. Gap shundaki, kodi yashirin bo'lgan mavjud dasturlarning kodini faqat Assembler tilida qayta tiklash mumkin. Assemblerni bilish esa tiklangan kodni o'rganishga va o'zgartirish kiritishga yordam beradi.

Assemblerning yana bir yutug'i shundaki, unda yozilgan dastur boshqa tillardagiga qaraganda juda tez ishlaydi. Shunday ekan, Assemblerdan faqat kompyuter qurilmalarini

boshqaradigan dasturlar emas, balki umumiy masalalarni yechadigan dasturlarni tuzishda ham foydalansa bo'ladi. Demak, ko'pincha quyidagi holatlarda dastur assemblerda yoziladi:

- Kompyuter qurilmalarini boshqaruvchi dastur kerak bo'lganda.
- Tez ishlashi kerak bo'lgan dastur qismini yozishda.
- Operatsion tizimning qurilmalar bilan ishlaydigan qismlarini yozishda.
- Kompyuter viruslarini tuzishda, ularni o'rganishda va ularga qarshi antiviruslarni yozishda.

1-rasmda Assembler va boshqa dasturlash tillarining qayerlarda ishlatilishi tasvirlangan.

Xulosa qilib aytsak, dasturlash nuqtai nazaridan Assembler ham boshqa dasturash tillari kabi o'z ahamiyatiga egadir.

1.2. NASM nima?

Har qanday operatsion tizim bir necha yuz, balki undan ham ko'proq dasturlar birlashmasidan tashkil topgan bo'ladi. Dastur esa o'z navbatida ma'lum bir dasturlash tilda yozilgan buyruqlar ketma-ketligidir. Hozirgi kunga kelib, dunyoda 400 ga yaqin dasturlash tillari mavjud bo'lib, ulardan dastur tuzishda foydalaniladi. Bu tillar inson tushuna oladigan va ularni boshqara oladigan darajada osonlashtirilib foydalanishga joriy etilgan. Ammo bu tillarning birortasini, shu jumladan, Assemblerni ham mashina tushuna olmaydi. Mashina uchun esa alohida *mashina tili* mavjud bo'lib, u faqat ikkilik sanoq tizimidagi (faqat 0 va 1 dan tuziladigan sonlar tizimi) sonlar oqrali ifodalanadigan buyruqlar ketma-ketligi ko'rinishida bo'ladi va juda ham sodda, ammo yodda saqlash murakkabdir. Unda to'g'ridan-to'g'ri dasturlash juda qiyin. Shu sababli inson uchun mo'ljallangan dasturlash tilida yozilgan dasturni mashina tiliga o'girib beradigan vosita kerak bo'ladi. Bu vosita dasturlashda *kompilyator* deyiladi. Demak, har bir til o'zining kompilyatoriga ega. Assembler tili uchun ham bir qator kompilyatorlar mavjud.

Har bir qurilma (mikroprotsessorga ega istalgan moslama) o'z Assembler tiliga ega va o'sha qurilmada ishlaydigan operatsion tizimlar (agar qurilma operatsion tizimga ega bo'lsa, albatta) qurilmaga mo'ljallangan Assembler tilida dasturlash imkoniyatini beradi[1]. Masalan, Intel protsessorlarida ishlaydigan Windows, Linux, Mac OS X va boshqalar shu protsessor uchun mo'ljallangan Assemblerni taqdim etishadi. Intel protsessorlari juda ham mashhur bo'lganligi sababli, ko'pgina dasturiy ta'minot ishlab chiqaruvchi firmalar Intel Assembleri qoidalariga mo'ljallangan o'z kompilyatorlarini ishlab chiqishgan. Assembler dasturi protsessordan tashqari qaysi kompilyatorga mo'ljallab yozilayotgani ham hisobga olinadi. Chunki bir kompilyator uchun yozilgan dastur ikkinchisiga to'g'ri kelmasligi mumkin. Farq juda katta bo'lmasada, qoidalar bir oz farq qiladi.

Yuqorida aytganimizdek, kitobda NASM kompilyatoriga asoslangan Intel assembleri o'rganiladi. NASM (**N**etwide **As**se**m**bler – Butun Tarmoq Assembleri) Assembler kompilyatori dunyoda keng foydalanishda bo'lib, uni tanlashdan maqsad undagi Assembler qoidalari ancha osonlashtirilgan, ya'ni bosh qotirishi kerak bo'lmagan har xil sozlashlar dasturchi zimmasidan olib tashlangan va kompilyator tomonidan o'z-o'zidan bajariladi, qolaversa, NASM Assembleri tushunishga juda oson. Ushbu xususiyatlar, ayniqsa, o'rganish maqsadlarida qo'l keladi. Bundan tashqari NASM barcha operatsion tizimlarda ishlaydi.

NASM kompilyatori LGPL(**L**esser **G**eneral **P**ublic **L**icense) guvohnomasi bilan chiqariladi, ya'ni dastur kodini ochiq holda erkin (to'lovsiz) tarqatiladi. Mahsulotni olib o'zgartirish va qayta tarqatish mumkin (Batafsil ma'lumot uchun NASM rasmiy guvohnomasiga qarang).

[1] Operatsion tizimsiz ishlaydigan qurilmalar ham mavjud bo'lib, ularga mashina tiliga o'girilgan kod to'g'ridan-to'g'ri yuklanadi.

NASM dan tashqari yana bir qancha kompilyatorlar mavjud va ularni qisqacha ko'rib chiqamiz:

- MASM (**M**icrosoft **As**se**m**bler) Assembleri dasturlarining tuzilish uslubi birmuncha chalkash va protsessor buyruqlaridan ham ko'ra ko'proq o'zining ichki atamalarini yozishga to'g'ri keladi. Buning ustiga yana pulli mahsulotdir.
- TASM (Borland Assembler) kompilyatori birmuncha yaxshiroq, ammo MASM ga qarab yaratilgan va unga juda o'xshaydi. TASM ham pulli mahsulot.
- GAS (**G**NU **As**se**m**bler) Assembleri qoidalari juda «dahshatli» va u inson dasturlashiga mo'ljallanmagan. GAS Assembleri asosan boshqa dasturlar tomonidan ishlatiladi. Ushbu mahsulot bepul va Windows, Linuxda ishlaydi.
- AS Assembleri faqat Linux va MINIXda ishlaydi.
- FASM kompilyatori NASMning yaqin «og'aynisi» bo'lib, NASM uchun mo'ljallab yozilgan dasturlarni tushunadi va NASM kabi oson dasturlash qoidalarga ega.

NASMni operatsion tizimda o'rnatish bo'yicha ma'lumot uchun kitob so'ngidagi «NASMni o'rnatish» ilovasiga murojaat qiling.

1.3. Operatsion tizim nima?

Har bir kompyuter bir qacha bo'laklardan iborat bo'lgan asosiy dastur to'plamiga ega. Bu dastur to'plami *operatsion tizim* deb yuritiladi. Operatsion tizimning kompyuterni ishga soladigan muhim ahamiyat kasb etuvchi bo'lagi *o'zak* deb ataladi. Kompyuter ishga tushirilganda birinchi navbatda operatsion tizim xotiraga yuklanadi va u keyinchalik bo'ladigan ish jarayonini ta'minlaydi. Operatsion tizim kompyuterning risoladagidek ishlashini ta'minlaydigan eng zarur amallarni o'z ichiga oladi.

Operatsion tizim asosan quyidagi vazifalarni bajaradi:

- Kompyuterning asosiy qurilmalarini (xotira, o'quv/yozuv moslamalari, markaziy protsessor va boshqalar[1]) boshqarish.
- Foydalanuvchi dasturlariga ishlash muhitini yaratib berish.

Foydalanuvchi dasturi deganda operatsion tizimning ishlashiga kerak bo'lmagan, faqatgina foydalanuvchi uchun tuzilgan dastur nazarda tutiladi. Masalan, kompyuter qurilmasini boshqaruvchi dastur foydalanuvchi dasturi emas, chunki u operatsion tizim bo'lagi hisoblanadi. Foydalanuvchi dasturlariga Microsoft Office, Adobe Reader, iTunes Music Player, Mozilla Firefox Web Browser va boshqalarni misol qilib keltirishimiz mumkin.

Zamonaviy operatsion tizim quyidagi muhim xususiyatlarga ega bo'lishi lozim:

- bir vaqtning o'zida bir nechta foydalanuvchilarga xizmat ko'rsata olishi, har bir foydalanuvchi o'z maxfiy so'ziga ega bo'lib, uning kompyuterdagi ma'lumotlari boshqa foydalanuvchilardan himoyalangan bo'lishi kerak;
- barcha foydalanuvchi dasturlari markaziy protsessorda *jarayon* sifatida bajarilishi kerak. Jarayon ancha mavhum tushuncha bo'lib, u markaziy protsessorda bajarilayotgan buyruqlar ketma-ketligini (dasturni) bildiradi;
- operatsion tizim dasturlari bir butun emas, balki alohida bo'laklardan tashkil topgan bo'lishi kerak. Ushbu dastur bo'laklari asosiy xotiraga lozim paytda yuklanib, zaruriyat bo'lmaganda qattiq diskda turaveradi;

[1] «Kompyuter tuzilishi» bobiga qarang.

- axborotlarni saqlash uchun ishonchli *fayl tizimini* taqdim etishi kerak. *Fayl* axborot saqlaydigan asosiy obyekt bo'lib, odatda ikki xil turda uchraydi: oddiy fayl va *direktoriya*. Barcha fayllar direktoriyalarda saqlanadi, shu jumladan, boshqa direktoriyalar ham.

Hozirgi kunga kelib, bir qancha tashkilotlar va firmalar operatsion tizim ishlab chiqarish bilan shug'ullanishadi. Quyidagi mavzularda shularning eng taniqlilari haqida qisqacha so'z yuritiladi.

1.3.1. Linux tizimi

Unix erkin (bepul) tarqatiladigan operatsion tizim bo'lib, ko'pgina kompyuter tarmoqlarida foydalaniladi. Tizimlar ichida u o'z zamonaviyligi va xatosiz ishlashi bilan ajralib turadi. Windows kabi tizimlarda foydalanuvchidan butunlay yashirin o'tadigan tizim jarayonlari Unixda ancha ochiq holda bo'ladi va foydalanuvchi bevosita o'zi ham buni boshqarish imkoniyatiga ega bo'ladi. Qisqasini aytganda dasturchi uchun haqiqiy muhit yaratilgan. Unix ishonchli tizim bo'lib, undan katta-katta serverlarda, bundan tashqari qo'l kompyuterlaridan boshlab super kompyuterlargacha foydalaniladi. Oxirgi paytlarda bunday kuchli tizimlar qatoriga *Linux* ham qo'shilib bormoqda. Linuxni o'zini oladigan bo'lsak, u ham Unix tizimining bir turkumi bo'lib, Unixsimonlar oilasiga kiradi. Unix tizimi ishlab chiqarilganidan beri ko'p vaqt o'tdi va bu vaqt davomida u bir necha turkumlarga bo'linib ketgan. AIX, BSD, NextSTEP, HP-UX, Linux, MINIX, OSF/1, SCO UNIX, System V, Solaris, XEMIX va boshqa ko'pgina turkumlar shular jumlasidandir. O'z navbatida bu turkumlar ham yana o'zining qism turkumlariga bo'linadi. Masalan, Linux tizimi yana Slackware, Debian, Fedora, Red Hat, Cent OS, Suse, Mandriva, Arch Linux, Ubuntu, Linux Mint va Android kabi turkumlarga bo'linadi. Unixning bunday turlari ko'p bo'lishiga sabab uning ochiq manba ekanligidir, ya'ni tizim dasturlari yashirin emas va xohlagan dasturchi uni o'ziga moslab olishi mumkin.

Unix tarixiga nazar solsak, 50–60-yillarda AQSHning Massachusetts Texnologiya Instituti tomonidan yaratilgan CTSS tizimi o'sha davr nazarida mukammal edi va katta shov-shuvga sabab bo'ldi. Shundan so'ng Massachusetts texnologlari Bell Labs labaratoriyasi va General Electronic[1] korxonasi bilan hamkorlik qilishni boshlaydi. Bundan ko'zlangan asosiy maqsad esa MULTICS (**Mult**iplexet **I**nformation and **C**omputing **S**ervice - ko'p qirrali axborot va hisob xizmati) nomli tizimining ikkinchi avlodini ishlab chiqish edi. Lekin ba'zi sabablarga ko'ra Bell Labs laboratoriyasi hamkorlikdan bosh tortib loyihadan chiqadi. Shunga qaramay Ken Tompson, Bell Labs laboratoriyasi xodimi, o'z izlanishlarini davom ettiraveradi va natijada o'zi assembler tilida MULTICS tizimini yozadi. Shu tariqa Tompson izlanishlarini davom ettirib yanada mukammalroq va yangilangan MULTICS turkumlari chiqariladi. Uning hamkasbi Brain Kernigan bu tizimga UNICS (**Un**iplexed **I**nformation and **C**omputing **S**ervice – Sodda axborot va hisob xizmati) deb nom beradi. Keyinchalik esa bu tizimni osonroq qilib UNIX deb atashgan.

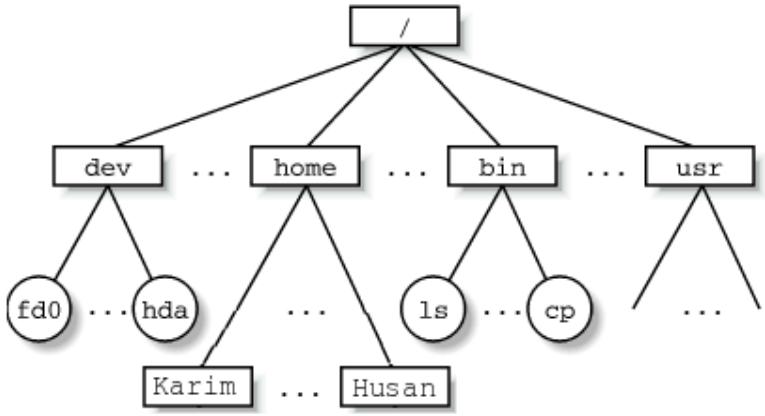
Linux ancha keyinroq 1991-yilda fin talabasi Linus Torvalds tomonidan yaratiladi. 1994-yilda esa Linuxning yangi turkumi chiqariladi. Bu tizim 165000 qator dastur kodidan[2] iborat bo'lib o'z ichga yangi fayllar tizimini va TCP/IP soketlari bilan tarmoqli dasturlar ta'minotini jamlagan edi. Bundan tashqari bir qancha yangi qurilmalarni boshqaruvchi dasturlar[3] ham shular jumlasidandir. Navbatdagi 2.0 Linux turkumi 1996-yilda dunyo yuzini ko'radi. Bu tizimning 470000 qatori C dasturlash tilida va 8000 qatori assemblerda yozilgan edi. Hozirgi kunda ham Linux ustida minglab dasturchilar ishlamoqda. Tarmoqli dasturlar, X WINDOW oyna tizimi va Unix dasturiy ta'minotining katta qismi Linuxga o'girilgani shunga misoldir. Linuxda birqancha

[1] O'sha davrlarda bu firma kompyuter ishlab chiqarish bilan shug'ullangan.
[2] Dasturni tashkil etuvchi dasturlash tilidagi buyruqlar to'plami tushuniladi.
[3] Kompyuter qurilmalarini (fleshka, monitor, modem) ilg'aydigan va boshqaradigan dastur. Ingliz tilida *driver* deb yuritiladi.

GUI (**G**raphical **U**ser **I**nterface), ya'ni tasvirli ishlash muhitlari mavjud: Gnome, KDE, XFCE va LXDE. Umuman olganda, Unix muxlislari uchun Linux loyiq tizim bo'lib shakllandi.

Birinchi navbatda shuni ta'kidlash lozimki, Linux yuz foiz viruslardan xoli tizimdir. Bundan tashqari Linuxning eng katta yutuqlaridan yana biri uning ochiq tizimligidir. Operatsion tizim kodi erkin tarqatiladi. Bu esa operatsion tizim qanday tuzilishini o'rganish uchun eng qulay imkoniyatdir. Chunki bir soha bo'yicha mutaxassis bo'lish uchun, o'sha soha haqida chuqur bilimga ega bo'lish kerak. Fikrimizning yaqqol isboti sifatida Linux asosida yaratilgan Doppix, UzMOT, EastLinux kabi O'zbek milliy operatsion tizimlarini keltirishimiz mumkin.

Linux tizimidagi ba'zi tushunchalar haqida batafsil to'xtalamiz. Linuxda barcha fayl va direktoriyalar bitta umumiy direktoriyada saqlanadi. Bu direktoriya *root*, ya'ni «ildiz» deb ataladi va qiya chiziq (*/*) belgisi orqali belgilanadi. Ildiz deb atalishining sababi tizimdagi direktoriyalarini bir katta daraxtga o'xshatishimiz mumkin (rasmga qarang). Root direktoriyasi esa daraxt ildizida joylashgan bo'ladi.



2-rasm.

Linux tizimida foydalanuvchilar asosan ikkiga bo'linadi. Bular oddiy va imkoniyati kengaytirilgan foydalanuvchilar. Birinchi turdagi foydalanuvchining imkoniyatlari ancha cheklangan bo'lib, uning ishlashi uchun ma'lum bir direktoriya beriladi va bu direktoriya uning uy direktoriyasi hisoblanadi. Linuxda oddiy foydalanuvchilar uchun ko'pincha */home* direktoriyasidan joy ajratiladi, masalan */home/Karim*.

Eslatma: */home/Karim* uy direktoriyasining mutlaq yo'lidir. Ildiz direktoriyadan boshlab to fayl yoki direktoriyagacha bo'lgan barcha direktoriyalarning tartibli ro'yxati shu fayl yoki direktoriyagacha bo'lgan mutlaq yo'l deb ataladi. Mutlaq yo'l Linuxda doim qiya chiziq (/) belgisidan boshlanadi. Bu qiya chiziq direktoriya nomlarini bir-biridan ajratish uchun ham ular orasiga qo'yiladi. Bundan tashqari nisbiy yo'l ham mavjud bo'lib, bu yo'l o'zingiz ishlayotgan direktoriyaga nisbatan beriladi. Masalan, */home* direktoriyasida turgan bo'lsangiz nisbiy yo'l quyidagicha beriladi: *Karim/salom.asm*.

Oddiy foydalanuvchi uy direktoriyasidan tashqarida ishlash huquqiga ega bo'lmaydi. Tizim fayllariga o'zgartirish kirita olmaydi, tizimda dastur ham o'rnata olmaydi. Ikkinchi turdagi foydalanuvchining imkoniyatlari katta bo'lib, unga berilgan huquqlar deyarli barcha amallarni bajarishga imkoniyat yaratadi, jumladan tizimda dasturlar o'rnatishda ham. Imkoniyati kengaytirilgan foydalanuvchi Linuxda root[1] nomini oladi. Tizimda barcha foydalanuvchilar maxfiy so'zlar bilan himoyalangan. Yanada ko'proq ma'lumot uchun «Buyruqlar qatorida ishlash» ilovasiga qarang.

[1] Buni tizimdagi *root* direktoriyasi bilan chalkashtirmang.

1.3.2. Windows tizimi

Windows operatsion tizimi AQSHning Washington shtatida joylashgan Microsoft korporatsiyasining mahsulotidir. Microsoft asosan operatsion tizim va ofis dasturlarini ishlab chiqarish bilan shug'ullanadi. Birinchi Windows tizimi 1985-yilda chiqarilgan va oldingi MS-DOS tizimidan uning farqi tasvirli oynalar orqali ishlashi edi.

Hozirda dunyoda Windows oddiy foydalanuvchilar ichida eng keng tarqalgan tizim bo'lib, u zamonaviy tizim talablariga javob beradi. Tizim dasturlari C, C#, C++ va Assembler dasturlash tillarida tuzilgan va ular ochiq holda tarqatilmaydi. Chunki Windows yopiq tizimdir va u pulli mahsulot.

Windows ham ko'p foydalanuvchili tizim bo'lib, foydalanuvchilar bir-birlaridan ma'lum darajada himoyalangan. Foydalanuvchilarning uy direktoriyalari *C:\Documents and Settings* da saqlanadi.

Yanada ko'proq ma'lumot uchun «Buyruqlar qatorida ishlash» ilovasiga qarang.

1.3.3. Mac OS X tizimi

Mac OS X operatsion tizimi jahonning nomdor firmalaridan biri bo'lmish Apple mahsulotidir. Apple faqat dasturiy ta'minot bilan emas, balki kompyuter qurilmarini ishlab chiqarish bilan ham shug'ullanadi. Shulardan Macintosh kompyuterlari, iPhone telefonlari va iPad kundalik rejalashtirgichlari kishilar orasida katta qiziqish uyg'otgan.

Unix tizimlari turkumiga mansub Mac OS X (tizim o'zagi Darwin deb nomlanib, u BSD va NextSTEP asosida yaratilgan) o'zining foydalanishda osonligi, ishonchliligi va tashqi ko'rinishining o'ziga xosligi bilan boshqa tizimlardan ajralib turadi. Umuman olganda, Apple mahsulotlari barcha zamonlarda namunali hisoblangan va qolgan ishlab chiqaruvchilar undagi sifat darajasiga erishishga intilganlar.

Mac OS X yarim yopiq va yarim ochiq operatsion tizim bo'lib, undagi dasturlar boshqa tizimlardagi dasturlarga nisbatan qimmatroq turadi.

II bob
Tarixiy ma'lumotlar

2.1. Birinchi hisob mashinalari

Hisob-kitob ishlari insonning ibtidosidan boshlab hozirgacha uning zaruriy ehtiyoji bo'lgan. Hatto eng qadimgi davrlarda ham odamlar hisob-kitob ishlarini osonlashtirish uchun turli vositalardan foydalanganlar. Masalan, bundan 1500 yillar avval ixtiro qilingan hisob taxtachasini[1] dalil qilib keltirish mumkin. Hisob taxtachasi bir necha pog'onali simlardan tashkil topgan bo'lib, simlardan bir xil o'lchamli toshchalar o'tkazilgan. Shu toshchalarni uyoq bu yoqqa surish orqali qo'shish va ayirish amallari bajarilgan. Lekin yuqoridagi moslama bilan bir oz murakkabroq amallarni bajarishning iloji bo'lmagan. Shu tufayli insoniyat taraqqiy topishi bilan uning hisoblash ishlariga ham ehtiyoji orta borgan.

Tarixdan bizgacha yetib kelgan ma'lumotlarga ko'ra birinchi hisob mashinasi yaratishga bo'lgan harakat 1623-yilda qayd etilgan. Uni Vilhelm Shikkard ismli kishi amalga oshirgan. To'g'ri, tarixda bundan ham oldinroq yaratilgan hisob mashinalari bo'lgan degan qarashlar ham bor. Chunki Leonardo da Vinchi qoralamalari ichida ham shunga o'xshash mashina chizmalari topilgan. Lekin bu qurilmaning qanday ishlagani bizga qorong'u. Vilgelm Shikard mashinasi esa aniq ishlagani haqida ma'lumotlar yetarli va bu moslama qo'shish va ayirish amallarini ham bajargani aniq. Mashinaga *arifmometr* deb nom berishgan.

Keyingi sana esa bir oz vaqt o'tgach, 1655-yilda qayd etilgan. Fransiyalik soliq yig'uvchining o'g'li bo'lmish Bleiz Paskal otasining hisob-kitob ishlarini osonlashtirish maqsadida o'z arifmometrini ixtiro qiladi. Qo'l harakati yordamida aylantirilib ishga solingan bu moslama ustida qilingan 3 yillik izlanish uning vositasida qo'shish va ayirish amallarini bajarish imkonini berdi. Arifmometr ishlash tartibi o'qqa kiygizilgan tishli g'ildiraklarning aylanishiga asoslangan edi.

1685-yilda esa matematik olim Gotfried Vilhelm Leibniz yanada mukammalroq mashina yaratadi. Bu qurilma qo'shish, ayirish amallaridan tashqari ko'paytirish va bo'lishni ham uddalar edi. Ushbu hisob mashinasi birinchi bo'lib ommaviy ishlab chiqarilganligi bilan ajralib turadi va hozirgacha dunyoning ko'pgina muzeylarida saqlanmoqda.

1821-yilda esa Charls Xavyer Tomas ismli kishi Leibniz arifmometriga o'xshagan moslama yaratadi. Uning avvalgilardan farqi bir nechta funksiyani ketma-ket bajara olishida edi. 1852 yilga kelib ixtirochi o'z mahsulotidan 1000 donaga yaqin ishlab chiqaradi va uning asosiy xaridorlari hukumat idoralari, banklar va sug'urta korxonalari bo'lgan.

2.2. Charls Bebbij tajribalari

O'z davrigacha ma'lum bo'lgan hisob mashinalari tuzilishiga tubdan o'zgartirish kiritishga intilgan ingliz matematigi Charls Bebbijning (1792–1871) kompyuterlar rivojiga qo'shgan hissasi beqiyosdir. Intilgan, harakat qilgan deganimizning sababi shuki, u juda ko'p harakat qilgan, ammo o'zi ko'zlagan dastur asosida ishlashi kerak bo'lgan hisob mashinasini ishga tushira olmagan. To'g'rirog'i umri yetmagan. Shunday bo'lsada, Charls tajribalari birinchi bo'lib dastur asosida ishlaydigan kompyuterni yasashga bo'lgan harakat sifatida tarixda qolgan. U yuqorida

[1] Bu taxtacha ovropada *abacus* deb yuritilgan, rus tilida esa *счет* deyiladi.

aytgan arifmometrlarga o'xshagan, lekin kattaligi 23 raqamli sonlar bilan ishlay oladigan kuchli qurilma ixtiro qilmoqchi bo'ladi. Bunday hisob mashinasi hukumat ishlarida ham juda qo'l kelgan bo'lar edi. Shuning uchun Angliya qirolligi bu loyihani moliyaviy qo'llab-quvvatlaydi. Ammo bu loyihani oxiriga yetkazmay turib Charls Bebbij boshqa mashina haqida o'ylay boshlaydi: «Dunyoda nima ko'p, masala ko'p, nima endi har bir shunaqa masala uchun yana shuncha temir-tersak yig'ish kerakmi? Yo'q shunday hisob mashinasi bo'lsinki xohlagan hisobni amalga oshira olsin» deya xayolidan o'tkazadi. Bu fikrni qo'llab-quvvatlagan holda Ada Lavleis ismli ayol yaratilishi kerak bo'lgan bu mashina uchun dastur ham yozadi. Shu tariqa Ada Lavleis birinchi dasturchi sifatida, Charls Bebbij esa «kompyuter otasi» degan nom bilan tarixda qoladi.

2.3. Birinchi avlod EHMlari[1]: elektron lampalar va elektr o'zgartiruvchi devorchalar[2]

Charls Bebbij orzu qilgan hisob mashinasi keyinchalik ikkinchi jahon urushi davrigagina kelib yaratiladi. 1941-yilda germaniyalik Konrad Zuse ismli kishi raqamli hisob mashinasini yaratadi.

Birinchi elektr hodisalari asosida ishlaydigan kompyuterlar esa Herman Holleriz tomonidan asos solingan IBM firmasi tomonidan ishlab chiqarila boshlandi. Mark I nomini olgan bu kompyuterning uzunligi 19.5 m, balandligi esa 3 m va og'irligi 5 tonna bo'lgan. Mashina 750000 ta detaldan tashkil topgan bo'lib, 23 ta raqamli ikkita sonni 4 soniyada ko'paytira olgan.

1946-yilda ishlashi to'liq elektr hodisalariga asoslangan ENIAC nomli EHM J. Presper Ekert va Jon Mauchly tomonidan yaratilgan. Kompyuter Mark I ga qaraganda 1000 marta tez ishlar va $2.4 \times 0.9 \times 30$ m^3 o'lchamga ega edi.

1949-yilda esa birinchi bo'lib Jon von Neuman dasturlarni kompyuter xotirasida saqlash g'oyasini ilgari suradi va ushbu g'oya asosida o'zi tuzgan algoritm bo'yicha ishlaydigan ikkilik koddan iborat dasturni ENIAC kompyuterida sinab ko'radi. Ammo u yaratgan asosiy g'oya bu bilan emas, balki kompyuter tuzilishi bilan bog'liq. Bunga ko'ra kompyuterning ishlash jarayoni uchta asosiy qurilmaga asoslangan bo'lishi kerak. Bular markaziy protsessor, xotira va o'quv/yozuv qurilmasidir. Hozirgi zamonaviy kompyuterlarning tuzilishi ham shunga asoslangan.

O'sha davr kompyuterlarida mexanik relelardan[3] foydalanilgan. Bu ishni juda sekinlashtirar edi. Keyinchalik ularni elektron lampalarga almashtirishgan. Birinchi avlod kompyuterlari uchun dasturlar to'g'ridan-to'g'ri mashina tilida yozilgan(u paytlarda hali dasturlash tillari bo'lmagan). Dasturlar esa maxsus injenerlar tomonidan elektr o'zgartiruvchi devorchalarni almashtirish orqali o'rnatilar edi. 50-yillarga kelib, elektr o'zgartiruvchi devorchalar o'rniga perifiral varaqlar dunyo yuzini ko'rdi.

2.4. Ikkinchi avlod EHMlari: tranzistorlar va to'plamli qayta ishlash tizimlari

50-yillar o'rtalariga kelib, tranzistorlar texnikaning hamma sohalarida ishlatila boshlandi. Natijada u kompyuterlar sohasiga ham kirib keldi. Bunday kompyuterlar *Mainframe* nomini olib, juda qimmat turardi. Faqatgina katta korxona va muassasalargina million-million dollarlar evaziga bunday kompyuterlarni xarid qilishar edi. Ishlash tizimida fortran, assembler kabi dasturlash tillari vujudga kelgan edi. Dasturlar esa hali ham perifiral varaqlarda alohida-alohida yozilib kompyuter oldida o'tiradigan operator xodimlarga keltirib berilar edi. Bir dastur varag'i

---

[1] EHM – elektron hisoblash mashinasi.

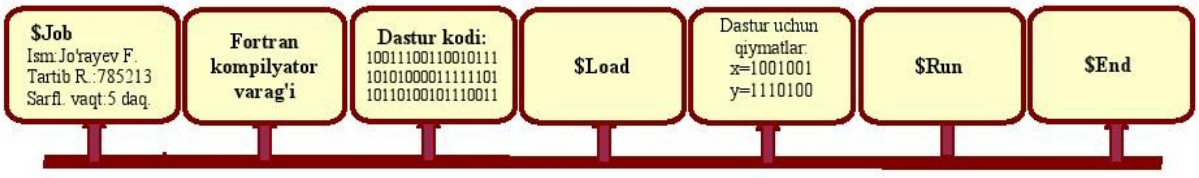
[2] Ingliz tilida *commutation panel* deb nomlanib elektr impulslarini o'zgartirib turish orqali turli xil signallar jo'natishga xizmat qiladi.

[3] Rele (fransuzcha *relayer* – almashtirmoq) biror qurilma yoki jarayon ko'rsatkichlarini qiymati yoxud yo'nalish o'zgarishini sezadigan va kichik quvvatli impuls bilan ijro etish mexanizmiga ta'sir qiladigan qurilma.

bajarilib bo'lgach, operator uni ikkinchisiga almashtirguncha ancha vaqt ketar, bu esa ish samaradorligini pasaytirar edi.

Shu sababli to'plamli qayta ishlash tizimi isloh qilinadi. Bunda avval barcha kerakli varaqlar to'planib imkoniyatlari kichikroq bo'lgan kompyuterda, masalan IBM 1401, magnit lentalariga o'girilar edi. Keyin magnit lentadagi dastur kuchliroq bo'lgan IBM 7094 kompyuterlarida ishga tushirilar edi.

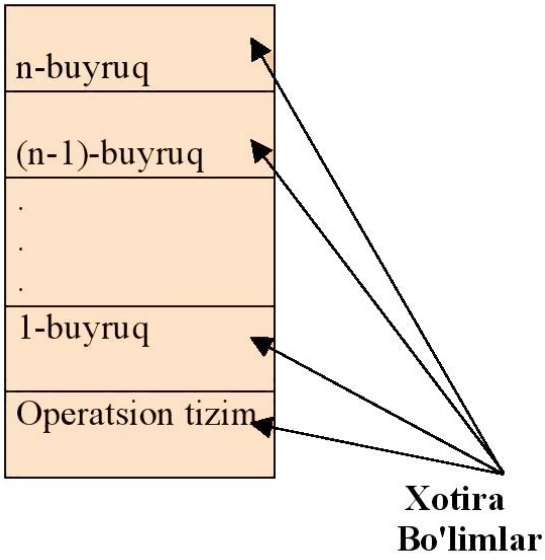
To'plamda birinchi bo'lib **`$Job`** varag'i qo'yilar edi. Unda dasturchi nomi tartib raqami va dasturga ajratilgan vaqt ko'rsatilgan, ikkinchi bo'lib Fortran kompilyator varag'i keyin dasturning o'zi, uchinchi bo'lib natijaviy dasturni yuklash varag'i **`$Load`** qo'yilgan, so'ng **`$Run`** varag'i ishga tushirish uchun qo'yilgan va oxirgi bo'lib **`$End`** varag'i kelgan (3-rasmga qarang).



3-rasm.

2.5. Uchinchi avlod EHMlari: ko'p vazifalik

Uchinchi avlod kompyuterlarida esa haqiqiy ma'noda operatsion tizim nomini olgan dasturlar paydo bo'la boshlagan. Bunga misol qilib 60-yillar o'rtasiga borib yaratilgan Multics[1] tizimini aytishimiz mumkin. Bu tizimdan 10 yilcha katta muvaffaqiyat bilan foydalanilgan, hatto 90-yillargacha General Motors, Ford kabi ulkan korxonalarda iste'molda bo'lgan. Keyin esa ikkita erkin tarqatiladigan yirik tizim dunyoga keldi. System V va BSD (**B**erkeley **S**oftware **D**istribution – Berkeley Dasturiy Ta'minoti). 1987 yilda UNIX mualliflari tomonidan MINIX turkum tizimi chiqariladi. Bu tizim hozirda ham internetda erkin tarqatiladi.



4-rasm.

Bu turkum protsessorlarining birinchi yoxud ikkinchi avlod protsessorlaridan asosiy farqi shunda ediki, ular ko'p, ya'ni bir necha vazifani «parallel» tarzda juda tez bajara olardi (4-rasmga qarang). Parallel deganda ma'joziy ma'no ko'zda tutiladi, ya'ni protsessor aslida

[1] Bu tizim haqida ko'proq ma'lumotga ega bo'lish uchun «Linux tizimi» mavzusiga qarang.

buyruqlarni ketma-ket bajaradi. Ammo buni u juda tez bajargani uchun xuddi bir necha dastur bir vaqtning o'zida bajarilayotgandek tushuncha hosil bo'ladi. Tizim to'xtovsiz tarzda bir buyruqni tugatgach darhol keyingisiga o'tar edi. Buni *spooling* (**S**imultaneous **P**eripheral **O**peration **O**n **L**ine – Jonli tarzda bir vaqtning o'zida perifiral operatsiyalar) deb ham atashadi.

Uchinchi avlod EHMlari ichida tarmoqli ishlaydigan tizimlar bir vaqtning o'zida 10-15 foydalanuvchilarga xizmat ko'rsata olar edi.

2.6. To'rtinchi avlod EHMlari: shaxsiy kompyuterlar

1980-yildan boshlab to'rtinchi avlod kompyuterlari ishlab chiqarila boshlangan. Bu kompyuterlar hozirda ko'pchiligimiz foydalanadigan juda kuchli kompyuterlar bo'lib, ular sichqoncha, tugmachalar taxtasi, ekran, printer va shunga o'xshagan juda ko'p tashqi moslamalar bilan ta'minlangan. Katta integral sxemalar bilan ishlaydigan bu kompyuterlarda kremniyli mikrosxemalar keng foydalaniladi.

Bu turkum kompyuterlar uchun operatsion tizim sifatida Microsoft firmasining Windows tizimini qo'llash juda urf bo'lgan. Bill Gates tomonidan asos solingan bu kompaniya hozirgacha juda sifatli tizim ishlab chiqarib keladi. Yana shuni ta'kidlash lozimki, bu tizimlarda foydalanuvchi uchun oson va qulay bo'lgan tasvirli ishlash muhiti taqdim etilgan.

III bob
Sanoq tizimlari

Dasturlash ilmida, ma'lumki, sonlarning va raqamlarning o'rni katta. Shu boisdan kitobda sanoq tizimlari uchun alohida bob ajratilgan. Sanoq tizimlarining ahamiyati shundaki, dasturchi ular haqida bilmay turib yaxshi dastur tuza olmaydi. Buning ustiga kompyuterning butun ishlash tartibi biz kundalik hayotimiz davomida ishlatadigan o'nlik sanoq tizimiga emas, balki to'laligicha ikkilik sanoq tizimiga asoslangan. Qisqasini aytganda sanoq tizimlarini bilmaydigan *dasturchi sanashni bilmagan matematikka* o'xshaydi.

3.1. Ikkilik sanoq tizimi

Ikkilik sanoq tizimidagi sonlar faqat 0 va 1 raqamlari majmuasidan tuziladi. Sanoq tizimining bunday nomlanishiga sabab ham uning faqat ikkita raqamdan iborat ekanligidadir, 0 va 1. Tushunarli bo'lishi uchun 1-jadvalda ikkilik sanoq tizimi o'nlik sanoq tizimi bilan taqqoslangan.

1-jadval

Ikkilik sanoq tizimi	0	1	10	11	100	101	110	111	1000	1001
O'nlik sanoq tizimi	0	1	2	3	4	5	6	7	8	9

Belgilanishi: $(100101)_2$.

Kompyuter qurilmalari ham axborotni aynan shu sanoq tizimida qabul qiladi, axborot yarim o'tkazgichlarda tok kuchi ma'lum chegaradan oshganda 1 deb, aks holda 0 deb qabul qilinadi. Axborot o'lchamga ega bo'ib, eng kichik o'lchov birligi *bit* deb qabul qilingan. Bir bit axborot 0 yoki 1 ni qabul qilishi mumkin. Keyingi kattaroq o'lchov birligi *bayt* bo'lib, 8 bit 1 baytga teng.

Ikkilik sanoq tizimidagi sonlar ustida arifmetik amallar xuddi o'nli sonlarga o'xshash bajariladi. Faqat ikkilik sanoq tizimida 0 va 1 dan boshqa raqam yo'qligini unutmaslik kerak, xolos.

1) Oddiy amallar:

0+0=0	0+1=1	1+1+1=11
1+0=1	1+1=10	100-10=10

2) Qo'shishda yodda saqlash, ayirishda esa qarz olish amallarini ko'rsatamiz:

$$\begin{array}{r} {\scriptstyle y\ y\ y} \\ 101 \\ +\ 011 \\ \hline 1000 \end{array} \qquad \begin{array}{r} {\scriptstyle q\ q} \\ 1101 \\ -\ 0111 \\ \hline 110 \end{array}$$

Raqamlar ustidagi «y» yodda saqlashni, «q» qarz olishni bildiradi.

3) O'nli sonlar bilan ham yoritamiz:



Assembler tilida sonning ikkilik son ekanini kompilyatorga bildirish uchun son oxirida `b` harfi qo'yiladi. Masalan: `1011101b`[1] – to'g'ri
`b1011101` – **noto'g'ri**
`1011102b` – **noto'g'ri**

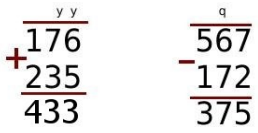
3.2. Sakkizlik sanoq tizimi

Sakkizlik sanoq tizimidagi sonlar quyidagi raqamlardan tuziladi: 0, 1, 2, 3, 4, 5, 6, 7. Taqqoslash uchun 2-jadvalda sakkizlik va o'nlik sanoq tizimidagi mos sonlar keltirilgan.

2-jadval

Sakkizlik sanoq tizimi	0	1	2	3	4	5	6	7	10	11	...	143	144
O'nlik sanoq tizimi	0	1	2	3	4	5	6	7	8	9	...	99	100

Ba'zi arifmetik amallarni keltiramiz:



Assembler tilida son sakkizlik sanoq tizimida ekanligini bildirish uchun son oxirida `q` yoki `o` lotin harfi qo'yiladi. Masalan: `77142q` = `111111001100010b`.
`712O`[2] = `458` (o'nlik sanoq tizimida).

3.3. O'nlik sanoq tizimi

O'nlik sanoq tizimiga ortiqcha izohning hojati yo'q deb o'ylaymiz. Bu kundalik hayotimizda qo'llanadigan 0,1,2,3,4,5,6,7,8,9 raqamlaridan tuziladigan sonlardir. Dasturlashda ushbu sanoq tizimidan ham keng foydalaniladi.

3.4. O'n oltilik sanoq tizimi

Dasturlashda o'n oltilik sanoq tizimi alohida o'ringa ega. Tizim raqamlari quyidagilar: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, a, b, c, d, e, f. Kichik harflar o'ringa bosh harflardan foydalansa ham bo'ladi. Assembler tilida son o'n oltilik sanoq tizimida ekanligini bildirish uchun son oxirida `h` lotin harfi qo'yiladi va doim 0….9 raqamlaridan birortasi bilan boshlanishi shart. Masalan: `0F3Dh`, `5afh`[3]. Assemblerda son o'n oltilik sanoq tizimida ekanligini bildirishning yana bir usuli son oldiga dollar

[1] Ingliz tilidagi *binary* so'zidan olingan bo'lib «ikkilik» ma'nosini beradi.
[2] Ingliz tilidagi *quad* va *octal* so'zlaridan olingan bo'lib «sakkizlik» degan ma'nolarni beradi.
[3] Ingliz tilidagi *hexadecimal* so'zidan olingan bo'lib «o'n oltilik» degan ma'noni beradi.

($) belgisini qo'yishdir. Masalan: `$0F3Dh`, `$5af`. Yana bir usul son oldiga `0x` qo'yishdan iborat. Bunda son 0...9 raqamlari bilan boshlanishi shart emas. Masalan: `0xF3Dh`, `0x5af`.

4-jadval orqali barcha sanoq tizimlarini taqqoslashimiz mumkin. Keling endi bir nechta amallar bajaraylik.

1) Oddiy amallar:

1 + 10 = 11	10 - B = 5
2 + E = 10	A - F = -5
A + D = 17	C - 1 = B

2) Ustma-ust ayirish va qo'shishni ko'rib chiqsak:



Yanada tushunarliroq bo'lishi uchun o'n oltilik sanoq tizimida ko'paytirish jadvali 3-jadvalda keltirilgan.

3-jadval

x	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	10
1	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	10
2	2	4	6	8	A	C	E	10	12	14	16	18	1A	1C	1E	20
3	3	6	9	C	F	12	15	18	1B	1E	21	24	27	2A	2D	30
4	4	8	C	10	14	18	1C	20	24	28	2C	30	34	38	3C	40
5	5	A	F	14	19	1E	23	28	2D	32	37	3C	41	46	4B	50
6	6	C	12	18	1E	24	2A	30	36	3C	42	48	4E	54	5A	60
7	7	E	15	1C	23	2A	31	38	3F	46	4D	54	5B	62	69	70
8	8	10	18	20	28	30	38	40	48	50	58	60	68	70	78	80
9	9	12	1B	24	2D	36	3F	48	51	5A	63	6C	75	7E	87	90
A	A	14	1E	28	32	3C	46	50	5A	64	6E	78	82	8C	96	A0
B	B	16	21	2C	37	42	4D	58	63	6E	79	84	8F	9A	A5	B0
C	C	18	24	30	3C	48	54	60	6C	78	84	90	9C	A8	B4	C0
D	D	1A	27	34	41	4E	5B	68	75	82	8F	9C	A9	B6	C3	D0
E	E	1C	2A	38	46	54	62	70	7E	8C	9A	A8	B6	C4	D2	E0
F	F	1E	2D	3C	4B	5A	69	78	87	96	A5	B4	C3	D2	E1	F0
10	10	20	30	40	50	60	70	80	90	A0	B0	C0	D0	E0	F0	100

3.5. Bir sanoq tizimidan ikkinchisiga o'tish usullari

Sanoq tizimlari bilan ishlaganda biridan ikkinchisiga o'tish usullarini bilish juda muhim. Shu sababdan biz quyida bir nechta kerakli formulalarni ko'rib chiqamiz.

3.5.1. O'nlik sanoq tizimiga o'tish

Bu eng oson algoritmlardan biri bo'lib, istalgan sanoq tizimidan quyidagi formula orqali o'giriladi:

$$(a_0 a_1 a_2 \dots a_n)_p = a_0 * p^{(n)} + a_1 * p^{(n-1)} + a_2 * p^{(n-2)} + \dots + a_n * p^0 = N$$

Bu yerda, $a_0 a_1 a_2 \dots a_n$ ma'lum bir sanoq tizimidagi sonning raqamlar ketma-ketligi, p bu mos ravishda o'sha sanoq tizimi asosi, masalan, ikkilik sanoq tizimi uchun p ikkiga teng, N bu o'nlik sanoq tizimidagi natija, natural son. Shu formula bo'yicha bir nechta misollar ko'rib chiqamiz. O'n oltilik f84a51c, ikkilik 1101001 va sakkizlik 1234567 sonlarini o'nlik sanoq tizimiga o'tkazish kerak bo'lsin:

1) $(f84a51c)_{16} = 15*16^6 + 8*16^5 + 4*16^4 + 10*16^3 + 5*16^2 + 1*16^1 + 12*16^0 = N$
2) $(1101001)_2 = 1*2^6 + 1*2^5 + 0*2^4 + 1*2^3 + 0*2^2 + 0*2^1 + 1*2^0 = N$
3) $(1234567)_8 = 1*8^6 + 2*8^5 + 3*8^4 + 4*8^3 + 5*8^2 + 6*8^1 + 7*8^0 = N$

Natijaviy N natural sonini hisoblash o'quvchiga havola.

Eslatma: Qavs pastidagi son sanoq tizimining asosini bildiradi. Agar o'sha son qo'yilmasa, son o'nlik sanoq tizimida deb tushuniladi. Masalan:125467.

4-jadval

O'nlik sanoq tizimi	O'n oltilik sanoq tizimi	Sakkizlik sanoq tizimi	Ikkilik sanoq tizimi
0	0	0	0000
1	1	1	0001
2	2	2	0010
3	3	3	0011
4	4	4	0100
5	5	5	0101
6	6	6	0110
7	7	7	0111
8	8	10	1000
9	9	11	1001
10	A	12	1010
11	B	13	1011
12	C	14	1100
13	D	15	1101
14	E	16	1110
15	F	17	1111

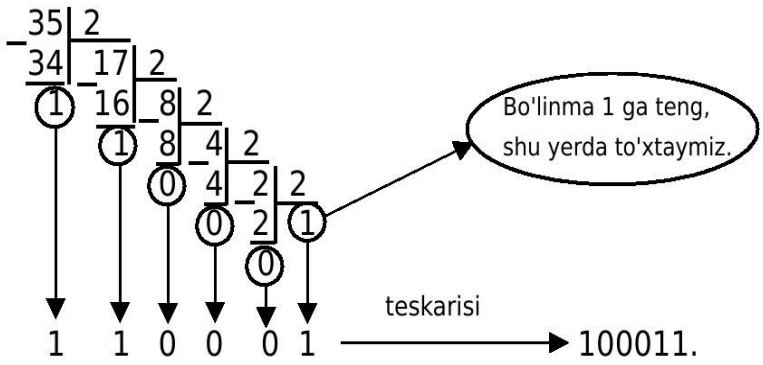
3.5.2. Ikkilik sanoq tizimiga o'tish

Agar siz o'nlik sanoq tizimidan ikkilik sanoq tizimiga o'tmoqchi bo'lsangiz, quyidagi amallarni bajarasiz:

1. Sonni 2 ga bo'lamiz va qoldiqni yodda saqlaymiz.
2. Hosil bo'lgan bo'linmani yana bo'linuvchi sifatida qabul qilib, 1-amalni qaytaramiz va bu ishni to bo'linma 1 yoki 0 ga teng bo'lguncha takrorlaymiz.

3. Bo'linma 1 yoki 0 ga teng bo'lganda to'xtaymiz va bo'linmani o'zini ham yodda saqlaymiz.
4. Yodda saqlagan raqamlarimizni teskari tartibda yozib chiqamiz.

Masalan 35 sonini ko'rib chiqamiz:



Demak, 35 = $(10011)_2$. Bu usulni biz shartli ravishda *zinama-zina bo'lish usuli* deb ataymiz.

Endi sakkizlik sanoq tizimidan ikkilikka o'tishni ko'rib chiqamiz. Bu yerda ham zinama-zina usulni qo'llasa bo'ladi, lekin osonroq usullar bor. Barchamizga ma'lumki, ikkining uchinchi darajasi sakizga teng va sakkizlik sanoq tizimidagi eng katta raqamning ikkilik ko'rinishi uchta birdan iborat, $(7)_8 = (111)_2$. Bu bog'liqlikning sonlarni o'girishda ham qatnashishini kuzatamiz. Gap shundaki, sakkizlik sonning har bir raqamini uning uchta raqamdan iborat ikkilik ko'rinishiga almashtirsak, shu sonning ikkilik ko'rinishiga ega bo'lamiz. Masalan: $(751623)_8$

$$(751623)_8=(111\ 101\ 001\ 110\ 010\ 011)_2$$
$$7\quad 5\quad 1\quad 6\quad 2\quad 3$$

Eslatma: Agar raqamning ikkilik ko'rinishi uchta raqamdan kam bo'lsa chapdan nollar bilan to'ldiriladi. Masalan : $(3)_8 = (011)_2$.

Mazkur usulni *raqamli almashtirish* deb nomlasak bo'ladi.

O'n oltilik tizimdan ikklikka o'tish ham shunga o'xshash. Lekin bu safar har bir o'n oltilik raqam to'rttalik ikkilik ko'rinishga almashtiriladi. Chunki ikkining to'rtinchi darajasi o'n oltiga teng, $2^4 = 16$. Masalan:

$$(B24A3F)_{16}=(1011\ 0010\ 0100\ 1010\ 0011\ 1111)_2$$
$$B\quad 2\quad 4\quad A\quad 3\quad F$$

Bu yerda, ham zinama-zina usulini qo'llasa bo'ladi. Ammo shuni unutmaslik kerakki, bo'lish va ayirish amallari o'n oltilik sanoq tizimida bajariladi.

3.5.3. Sakkizlik sanoq tizimiga o'tish

Ikkilik sanoq tizimidagi sonni sakkizlikka o'girishda raqamli almashtirish usulining teskarisi qo'llaniladi. Demak, ikkilik sonni oxiridan boshlab chapga qarab uchta-uchta raqamdan qilib beligilab olamiz va shu uchliklarni sakkizllik tizimidagi mos raqamlarga o'tkazib yana o'sha tartibda yozsak, sakkizlik son kelib chiqadi. Misol tariqasida 1010101 sonini ko'rib chiqamiz:

$$(1010101)_2 \longrightarrow 001\ .\ 010\ .\ 101 \longrightarrow (1\ 2\ 5)_8 \quad .$$

O'nlik sanoq tizimidan sakizllikka o'tishda esa yuqorida ko'rsatilgan zinama-zina bo'lish usulidan foydalaniladi. Bu safar 2 ga emas, balki 8 ga bo'lamiz va bo'linma 1 dan 7 gacha bo'lgan sonlarga teng bo'lmaguncha davom ettiramiz. Agar raqamli almashtirish usuli orqali o'girmoqchi bo'lsangiz, ikki o'girishni amalga oshirishingizga to'g'ri keladi. Birinchi, o'n oltilik

sonni «Ikkilik sanoq tizimiga o'tish» mavzusida ko'rsatilgani kabi ikkilik ko'rinishga keltirasiz. Keyin esa yuqorida ko'rsatilgan raqamli almashtirish usuli bilan ikkilikdan sakkizlikka o'tkazasiz. Masalan: $(E43FD)_{16}$

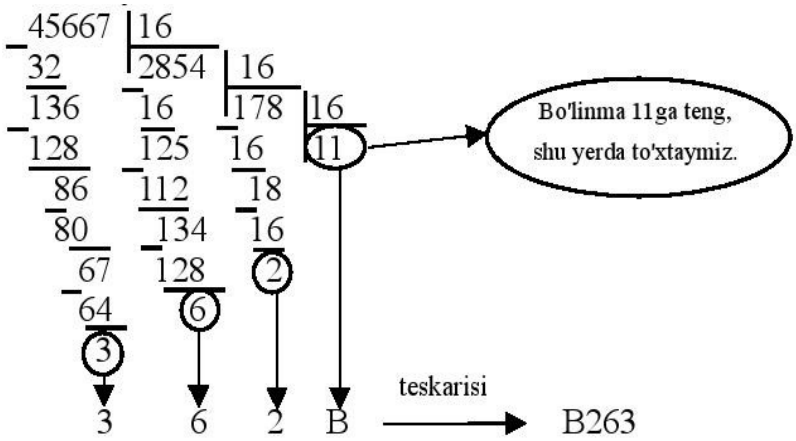
$$\begin{matrix} & E & 4 & 3 & F & D \\ (E43FD)_{16} & = & (1110 & 0100 & 0011 & 1111 & 1101)_2 \end{matrix}$$

$$(011.100.100.001.111.111.101)_2 = (3441775)_8$$

3.5.4. O'n oltilik sanoq tizimiga o'tish

Bu tizimga sonlarni o'girish chamasi qanday yuz berishini ziyrak o'quvchi bilib olgan bo'lsa kerak. Chunki endi yuqorida qo'llangan usullar asosan takrorlanadi. Agar sizda ikkilik son bo'lsa, uni o'ngdan boshlab chapga to'rtta-to'rtta qilib guruhlaysiz va shu guruhlarni o'n oltilik mos sonlarga almashtirasiz. Qisqasini aytganda, bu raqamli almashtirish usulining o'zginasi bo'ladi.

O'nlik sonlar esa o'n oltilikka zinama-zina bo'lish usuli bilan keltiriladi. Faqat shuni unutmaslik kerakki, bu safar siz 16 ga bo'lasiz va bo'linma 1 dan 15 gacha bo'lgan sonlarga teng bo'lmaguncha davom ettiraverasiz. Yodda saqlaydigan soningiz 9 dan katta bo'lsa, uni harflar bilan belgilaysiz. Masalan, 45667 ni olaylik:



Sakkizlik sanoq tizimidan o'tishni esa o'quvchining o'ziga qoldiramiz. Faqat shuni aytishimiz mumkinki, bu yerda o'n oltilikdan sakkizlikka o'tish amallari teskarisiga bajariladi.

3.6. Har xil sanoq tizimlarida kasr sonlar

Barcha sanoq tizimlarida kasr sonlar bo'lib, ular xuddi o'nlik sanoq tizimidagi kasr sonlar kabi butun va kasr qismiga ega. Masalan:

1) 56.45 2) $(11001.101)_2$
3) $(B5A1.78F)_{16}$ 4) $(12345.67)_8$

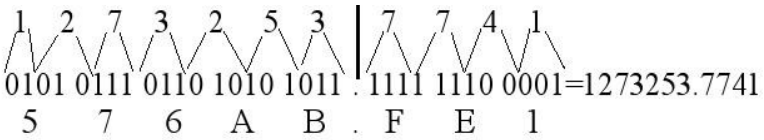
Istalgan sanoq tizimidagi kasr sonlarni o'nlik sanoq tizimiga o'girish uchun quyidagi formulalardan foydalanamiz:

$$(a_n \dots a_2 a_1 a_0 . b_1 b_2 b_3 \dots b_m)_p = a_n * p^{(n)} + \dots + a_2 * p^{(2)} + a_1 * p^{(1)} + a_0 * p^0 + b_1 * p^{-1} + b_2 * p^{-2} + b_3 * p^{-3} + \dots$$
$$\dots + b_m * p^{-m} = R$$

Bu yerda p sanoq tizimi asosi, $a_n \ldots a_2 a_1 a_0$ sonning butun, $b_1 b_2 b_3 \ldots b_m$ esa kasr qism raqamlari va nihoyat R mos ravishda o'nlik tizimdagi natijaviy ratsional son. Misol tariqasida $(10101.01)_2$ sonini ko'rib chiqamiz:

$$1*2^4+0*2^3+1*2^2+0*2^1+1*2^0+0*2^{-1}+1*2^{-2}=16+0+4+0+1+0+0.25=21.25$$

Agar ish sakkizlik, ikkilik yoki o'n oltilik sanoq tizimlari bilan bo'ladigan bo'lsa biz asosan sanoq tizimiga qarab uchli yoki to'rtli guruhlash usullaridan foydalanamiz, ya'ni raqamli almashish usuli. Masalan $(576AB.FE1)_{16} = (?)_8$ ni yechib ko'ramiz:

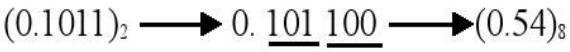


Eslatma: Shuni unutmaslik kerakki, nuqtadan chap tomondagi sonning butun qismi nuqtadan boshlab chapga qarab, kasr qismi esa nuqtadan o'nga qarab guruhlanadi.

Biz to'rtlik shaklga keltirganda doim chap tamondan to'ldiruvchi nollar kiritamiz, e'tibor bering:



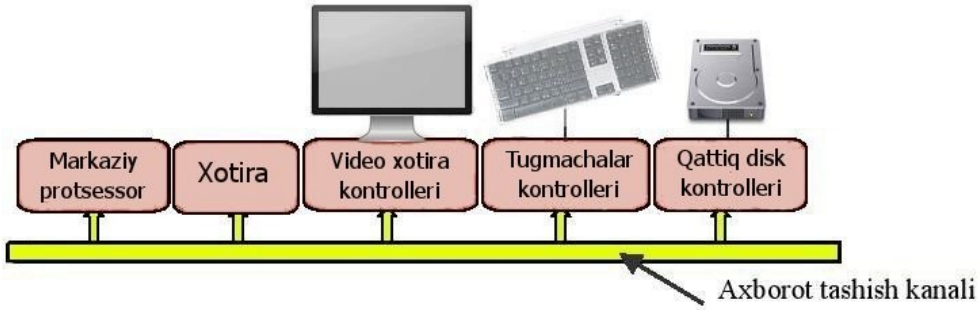
Bu usulning asosiy tomoni shundaki, biz ikkilik sonning kasr qismini o'nga qarab uchtadan yoki to'rttadan qilib guruhlaganimizda yetmagan raqamlarni mos ravishda o'ngdan nollar bilan to'ldiramiz. Masalan:



Mazkur bobda sanoq tizimlari haqida asosiy ma'lumotlar berildi. Ammo ushbu kitob aynan sanoq tizimlariga bag'ishlanmagan va sizga batafsilroq ma'lumot kerak bo'lsa, informatika bo'yicha maktab darsliklariga qarang.

IV bob
Kompyuter tuzilishi

Dasturchi faoliyati davomida har safar kompyuter bilan ish ko'rar ekan uning tuzilishi bilan ham tanish bo'lishi zarur. Ushbu bobda biz kompyuter tuzilishining umumiy ko'rinishini bilan ma'lum darajada tanishib chiqamiz.



4.1. Markaziy protsessor

Markaziy protsessor kompyuterning ajralmas qurilmasi bo'lib, asosiy hisoblashlarni aynan shu moslama amalga oshiradi. Barcha zamonaviy kompyuterlarning asosiy qisimlari markaziy protsessor, xotira va o'quv/yozuv[1] qurilmalaridan tashkil topgan bo'lib, bu von Neuman yaratgan g'oyaning mutlaqo o'zidir.

Markaziy protsessor rus tilida *центральный процессор*, ingliz tilida *cpu* (**C**entral **P**rocessing **U**nit) deb yuritiladi. Uning asosiy vazifasi berilgan manbalardan buyruqlarni o'qib ularni bajarishdan iborat bo'lib, kompyuter «miyasi» hisoblanadi. Hozirda eng so'nggi hisoblangan Pentium IV protsessori uchun mezoniy x86 buyruqlar to'plami belgilangan bo'lib, ushbu buyruqlarni bajara oladigan protsessorlar x86 oilasiga mansub hisoblanadi. x86 oilasiga misol qilib Intel va AMD protsessorlarini keltirish mumkin. Ammo har bir firma o'z protsessorlariga x86 dan tashqari qo'shimcha buyruqlar ham kiritishi mumkin.

Protsessor asosan *jarayonlar* bilan ishlaydi. Protsessor so'zi ham anyan inglizcha *process* so'zidan olingan bo'lib, jarayon ma'nosini beradi. Jarayon aslini olganda mavhum tushuncha bo'lib, ko'p vazifali operatsion tizimlarda juda katta ahamyatga ega. Jarayon deb ma'lum bir dasturning protsessor tomonidan bajarilayotgan vaziyatga aytiladi. Dastur bir marta yoziladi va shundayligicha mavjud bo'ladi, jarayon esa boshlanish, davom etish va tugash vaqtiga ega. Jarayonning dasturdan farqi ham shunda. Umuman olganda dastur ishga tushirilganda uning protsessor orqali bajarilishi uchun jarayon yaratiladi. Bitta dastur bajarilishi uchun bir nechta jarayon yaratilishi yoki bitta jarayon o'z ichiga bir nechta dasturni olishi ham mumkin.

Markaziy protsessorning ishlash tartibi quyidagicha: ma'lum bir vazifani o'quv/yozuv moslamasidan yoki xotiradan qabul qiladi va uni bajaradi, bu tugagach boshqasini boshlaydi. Lekin hozirgi qo'sh yadroli superprotsessorlar bir vaqtning o'zida bir qancha buyruqlarni bajarish qobilyatiga ega. Buning asosiy sababi vaqtni tejashdir. Bundan tashqari vaqtni yo'qotmaslik uchun ko'pgina usullar o'ylab topilgan. Masalan, xotiradan axborot olish uchun ham ko'p vaqt kerak, buni oldini olish uchun to'g'ridan-to'g'ri protsessorga yopishtiriladigan *registrlar* vaqtinchalik xotira vazifasini bajarish maqsadida ixtiro qilingan. Registrlar sanoqli va xotira

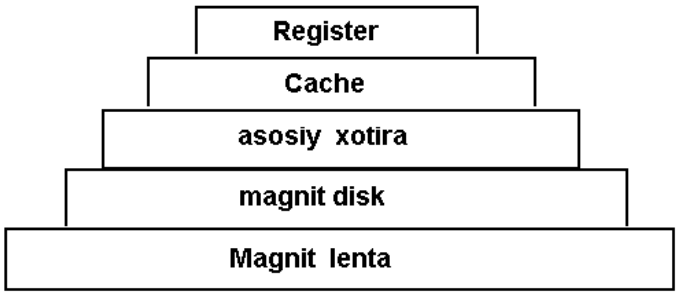
[1] To'liq ma'lumot uchun «O'quv/yozuv moslamalari» mavzusiga qarang.

sig'imi chegaralangan bo'lsada, vaqtinchalik axborot saqlashda juda qo'l keladi. Registrlar yaratilganidan buyon takomillashib axborot saqlash sig'imi ham kattalashib kelayapti. Ularning 8, 16, 32 bitlilari protsessor turkumiga qarab muomalada bo'ladi. Hozirgi kunda pentium IV protsessorlarida 32 yoki 64 bitli registrlar joylashtiriladi. Shunga qarab protsessorlar ham 32 yoki 64 bitli deyiladi.

Bundan tashqari protsessorlar qanchalik tez ishlay olishiga qarab ham baholanadi. Bu ular ichida joylashgan *tizim soati* deb ataluvchi moslamaga bog'liq. Mazkur moslama protsessorning ishlashini boshqarib turadi. Tizim soati MP (Markaziy Protsessor)ga bir xil vaqt oraliqlarida 1 yoki 0 ga teng bo'lgan signallarni yuborish orqali bir buyruqdan ikkinchisiga o'tishni ko'rsatib turadi. Bitta buyruqni bajarish uchun bir yoki undan ko'p signallar kerak bo'lishini hisobga olsak, protsessor vaqt birligi ichida qancha ko'p tizim soatidan signallar qabul qila olsa, shuncha tez ishlay oladi degan xulosaga kelsak bo'ladi. Hozirgi pentium IV protsessorlari shunday signallarning ikki milliardchasini bir soniya ichida qabul qila oladi. Boshqacha qilib aytganda, protsessor chastotasi 2 GHz[1] ga teng. Agar kompyuter xarid qilgan bo'lsangiz, protsessor ko'rsatgichlari ichida uning chastotasi ham berilgan bo'ladi. O'sha yerda, masalan, shunday yozuv bo'ladi: CPU 2.5 GHz.

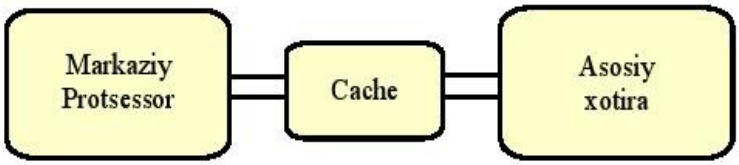
4.2. Xotira

Xotira kompyuterning asosiy axborot ombori bo'lib, ma'lumot saqlashda foydalaniladigan qurilmadir. Xotira rus tilida *память,* ingliz tilida *memory* deb yuritiladi. Xotira qatlamlari tez ishlashiga qarab bir nechta pog'onalardan iborat.



Rasmda tasvirlangan registrlar MP ga yaqin joylashgan qurilmalar bo'lib ular nihoyatda tez ishlaydi. Registrlar har biri 32 yoki 64 bitgacha axborot saqlay oladi va ularning soni sanoqli bo'lib, ularda faqat protsessor tomonidan eng ko'p ishlatiladigan zaruriy axborotlar saqlanadi. Har bir registr o'z nomiga ega bo'lib, ularning ichida umumiy foydalanish uchun mo'ljallanganlari va maxsus vazifaga ega bo'lganlari mavjud. Boshqa tillardan farqli o'laroq assembler tili registrlar bilan to'g'ridan-to'g'ri ishlash imkoniyatini beradi. Dasturlashda qiymatlarni saqlashda eng ko'p registrlardan foydalaniladi. Ular bilan keyinroq batafsil tanishib chiqamiz.

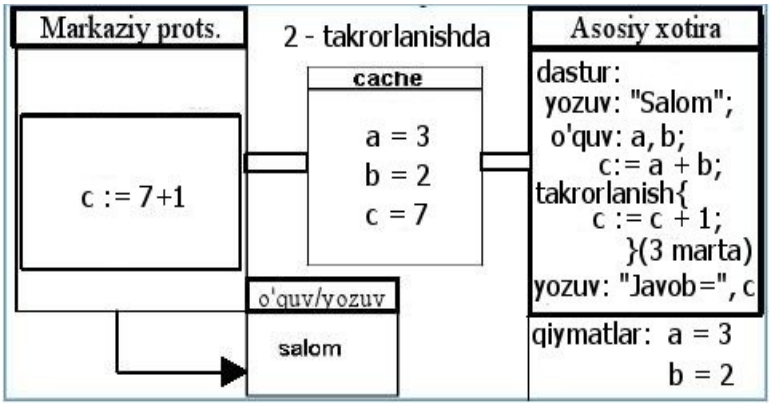
Rasmda xotira turlari bejizga tartib bilan ustma-ust qo'yilmagan. Ular pastga borgan sari katta ko'lamda axborot saqlash imkoniyati ortib boraveradi, ammo ularga murojaat etishga sarf bo'ladigan vaqt ham ortib boraveradi.



Shunday qilib registrlardan keyin *cache* xotira keladi. Cache xotira qatlami markaziy protsessor va asosiy xotira orasida joylashgan bo'lib protsessorga yaqin joylashgani tufayli

[1] Giga Hertz vaqt birligi ichida tebranishlar sonini o'lchash birligi. Mashhur nemis fizigi Henrih Hertz sharafiga qo'yilgan.

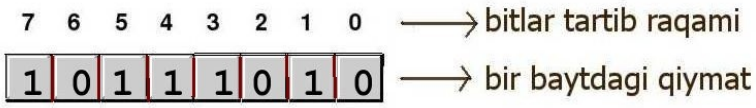
asosiy xotiradan tezroq muntazirdir. Registrlardan farqli o'laroq, cacheda dasturning buyruqlar yoki qiymatlar bo'limi ham saqlanishi mumkin. Protsessor asosiy xotiraga yuzlanishidan oldin cacheni qarab ko'radi. Agar u yerda kerakli ma'lumot topilsa, ish asosiy xotiraga murojaat qilishgacha yetib bormaydi (5-rasmga qarang). Cacheda 1-5 Mb gacha ma'lumot saqlanishi mumkin.



5-rasm.

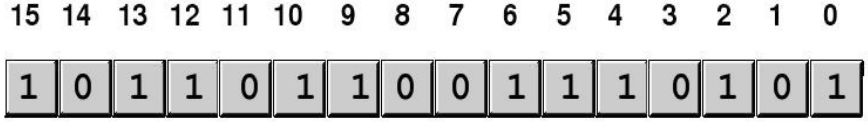
Keling endi *asosiy xotira* imkoniyatlarini o'rganaylik. Bu xotira rus tilida *оперативная память*, ingliz tilida *RAM* (**R**andom **A**ccess **M**emory – Tavakkal tanlanadigan xotira) deb ataladi. Bu xotira dasturlar ishlashi uchun eng kerakli va asosiysi bo'lib axborotni o'zida vaqtinchalik saqlab turadi. Vaqtinchalik deyilishiga sabab kompyuter yoqilib dasturlar ishga tushgandagina asosiy xotira kerak bo'ladi, ya'ni protsessor ishlamayotganida asosiy xotira kerak bo'lmaydi. Registrlar, cache va asosiy xotiraning qattiq diskdan farqi ham shunda. Qattiq diskda axborot, soddaroq qilib aytganda fayllar, doimiy saqlanadi.

Asosiy xotirani chiziqli jadvalga o'xshatishimiz mumkin. Har bir jadval katakchasida esa bir baytga teng ma'lumot saqlanadi. Ikkilik sanoq tizimidan ma'lumki, bir bit ma'lumot 1 yoki 0 ga teng bo'lishi mumkin. Bitdan keyingi kattaroq o'lchov birligi bayt bo'lib u sakkiz bitdan iborat. Bir bayt axborotni ko'z oldimizga keltiraylik:



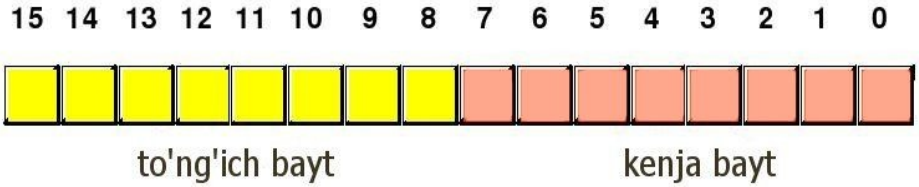
Ko'rinib turibdiki, bir bayt axborot eng ko'pi bilan $(11111111)_2$ sonini o'zida sig'dirishi mumkin. Bu son o'n oltilik sanoq tizimida $(FF)_{16}$ ga teng, ya'ni 255 soni. Xotiradagi qiymatni ifodalashda o'n oltilik sanoq tizimi juda qulay bo'lib, sonning o'lchamini ko'z bilan chamalab tezda aniqlash imkoniyatini beradi. Chunki, ikki xonali o'n oltilik sonni bir bayt sifatida qabul qilish mumkin. Bundan buyon biz xotiradagi axborot haqida gapirganimizda asosan o'n oltilik sanoq tizimidan foydalanamiz. Chunki bu sanoq tizimida sonlar ancha ixcham ko'rinishga ega.

Yana bir e'tiborga molik jihat bitlarni raqamlash tartibidir. Ular o'ngdan chapga qarab noldan boshlab raqamlanadi. Eng chapdagi bit eng katta(yuqori) bit hisoblanadi va shartli ravishda *to'ng'ich bit* deb nomlanadi. Eng o'ngdagi bit esa eng kichik(quyi) bit hisoblanadi va shartli ravishda *kenja bit* deb nomlanadi. Bir baytda 7-bit to'ngi'ch bit hisoblanadi. To'chgi'ch bit tartib raqami axborot o'lchamiga qarab har xil bo'ladi. Masalan, ikki bayt axborotni olaylik:

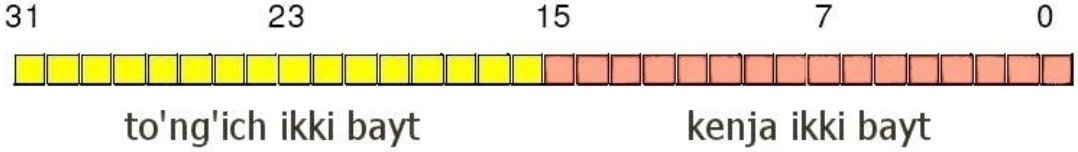


Bu yerda to'ng'ich bit 15-bit hisoblanadi. Kenja bit esa doim 0-bitga to'g'ri keladi.

Gap katta o'lchamlar haqida borganda nafaqat bitlar, balki baytlarga nisbatan ham to'ng'ich yoki kenja atamalari ishlatilishi mumkin. Masalan, ikki baytni olsak, unda to'ng'ich va kenja baytlar bo'ladi:

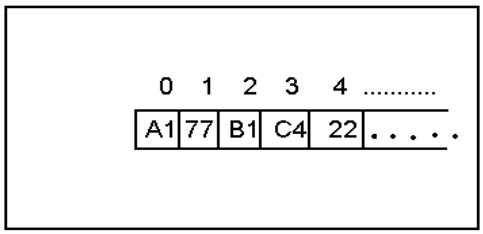


Dasturlashda ikki baytli axborot o'lchamidan keyin ko'p uchrashi bo'yicha to'rt baytli o'lcham turadi. Chunki asosiy registrlar va stackning[1] ishlashida shu o'lcham qo'llaniladi. To'rt baytda to'ng'ich bit 31-bit bo'ladi:



Xotirada har bir bayt o'z tartib raqamiga ega. Bu raqam xotiradagi baytning *manzili* deyiladi. Biz o'zimizga kerakli axborotni xotiradan uning manzili orqali qo'lga kiritamiz. Manzil tushunchasining bo'lishiga sabab shundaki, xotiraning har bir katakchasidagi axborotni aynanlashtirish kerak. Shuning uchun operatsion tizim bu ma'lumotlarni manzillab chiqadi va aytilgan vaqtda ulardan foydalanadi. Biz faqat manzilni berish orqaligina ma'lum bir axborotni o'qishimiz mumkin. Oddiy hayotiy misol keltiraylik. Muzeylarda, teatrlarda kirish joylarida yechinish xonalarini ko'rgansiz. U yerdagi kiyimlarni iluvchi xizmatchilar kiyimingizni qabul qilib o'rniga raqamlangan jeton berishadi. Bu yerda kiyim saqlash xonasini xotiraga, undagi xizmatchilarni operatsion tizimga o'xshatishimiz mumkin. Kiyimingiz bo'lsa xotiraga yuklanadigan axborot, oladigan raqamingiz bo'lsa manzil hisobiga o'tadi.

Yuqorida ta'kidlaganimizdek xotirada har bir bayt o'z manziliga ega. Xotiradagi birinchi bayt nol soni bilan belgilanadi va hokazo. 7-rasmda dastlabki 5 bayt qanday manzillanishi ko'rsatilgan. Katakcha ichidagi o'n oltilik sanoq tizimidagi sonlar bu xotirada saqlanayotgan axborotlar, tepadagi sonlar esa ularning manzillari:



7-rasm.

Baytdan tashqari yanada kattaroq o'lchov birliklari mavjud:

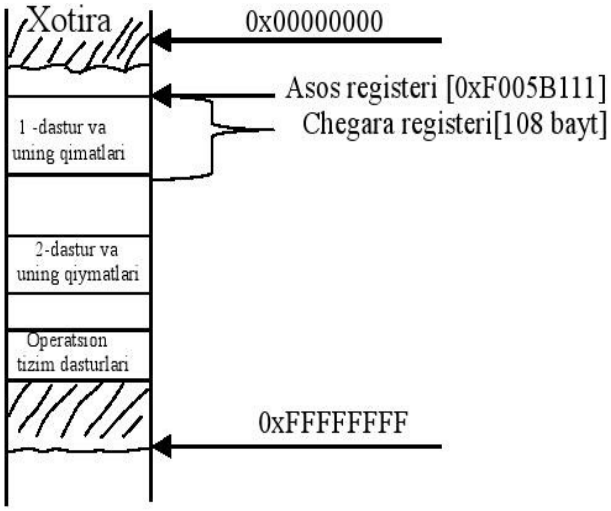
1Kb = 1024b = 2^{10}b (Kilo bayt),
1Mb = 1024Kb = 2^{20}b (Mega bayt),
1Gb = 1024Mb = 2^{30}b (Giga bayt) va hokazo.

Bir narsani eslatib o'tish lozimki, xotiradan farqli o'laroq registrlarda saqlanadigan axborot manzillanmaydi. Chunki ular aniq bir registr ichida turgan bo'ladi. O'sha registr nomini bilish kifoya. Cache xotiraning ham o'z manzillash mezoni bor.

Asosiy xotira hajmi 64Mb dan 64Gb gacha bo'lishi mumkin. Xotira faoliyati davrida turli xil muammolar bo'ladi va shularning ba'zilari e'tiborga molikdir. Muammolarning biri xotiradagi

[1] Batafsil ma'lumot uchun «Stack» mavzusiga qarang.

dasturlarning bir-birlari bilan yoki operatsion tizimning qiymatlari bilan aralashib ketishidan saqlashdir. Shunday muammolardan yana biri dasturlarni u yoqdan bu yoqqa ko'chirganda manzillarni boshqarishdir. Ikkinchi muammoni yechish uchun ikkita qo'shimcha registr o'ylab topilgan. Birinchisi asos registri, ikkinchisi chegara registri. Asos registrida asosiy xotirada joriy dastur boshlangan joyning manzili saqlanadi. Chegara registrida esa dastur va uning qiymatlari hajmi haqida ma'lumot saqlanadi. Protsessor xotiraga murojaat qilishidan avval mazkur registrlarni tekshirib, so'ng buyruqlarni bajara boshlaydi (bu rasmda yaqqol aks ettirilgan).



6-rasm.

Asosiy xotiradan so'ng magnit disk, ya'ni qattiq disk keladi. Bu xotira asosiy xotiradan chamasi 3-4 barobar sekinroq ishlaydi va ancha arzonroq ham. Vazifasiga ko'ra bu xotira axborotni doimiy saqlashni amalga oshiradi. Kompyuteringizdagi dasturlar fayllari va o'zingizning hujjatlaringiz hammasi xotiraning shu qatlamida saqlanadi. Qattiq disk bir necha plastinkalardan tashkil topgan bo'lib, ularning daqiqasiga 5400–10800 martagacha aylanishi hisobiga yodda saqlashni amalga oshiradi. Hajmi esa 5Gb dan 500Gb gacha bo'ladi. Keyingi pog'ona xotira esa magnit lentalari bo'lib, bu hamma kompyuterlarda ham bo'lavermaydi. Chunki bunday xotira tizimi ancha katta bo'lib, ulardan yirik ma'lumotlar omborlarini tuzishda foydalaniladi.

4.3. O'quv/yozuv moslamalari

O'quv/yozuv moslamalari kompyuterdan tashqarida joylashgan va uni tashqi olam bilan axborot almashinishini ta'minlaydigan qurilmalardir. O'quv/yozuv moslamalari rus tilida *устройства ввода и вывода*, ingliz tilida esa *input/output devices* deb yuritiladi. Bu turdagi moslamalar aynan protsessor ish jarayoniga keragi yo'q bo'lib, lekin ular biz kompyuter bilan muloqot qilishimiz uchun zarurdir. Birinchi navbatda o'quv/yozuv moslamalariga monitor, tugmachalar taxtasi, sichqoncha, printer, skanner, diskdon va shunga o'xshagan axborot manbalari kiradi. Kompyuterga kiruvchi axborotlar *o'quv*, undan chiquvchi axborotlar *yozuv* hisoblanadi. Misol qilib monitorni yozuv, tugmachalar taxtasini esa o'quv shaklida ko'rsatishimiz mumkin.

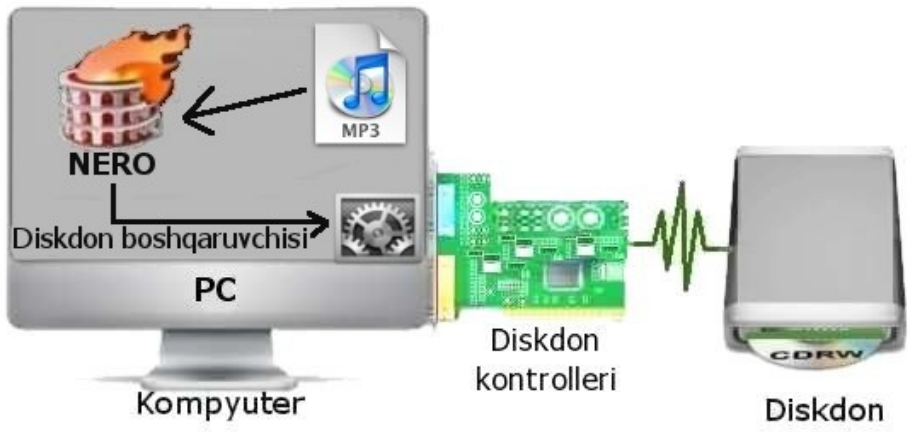
Sanab o'tgan moslamalar to'g'ri protsessorga ulanmaydi. Umuman olganda protsessor ularning mavjud ekanligi haqida ham bilmaydi. Protsessor asosan ularga asosiy xotira orqali murojaat qiladi. Masalan, monitordagi tasvirni yozuv deb hisoblasak, protsessor tomonidan qilinadigan ish bor yo'g'i video xotiraga tasvir nuqtalarini raqamli tarzda yozishdan iborat bo'ladi. Xotiradagi axborot ekranda paydo bo'lguncha ko'p bosqichli jarayonlar yuz beradi. Shuning uchun kompyuterdagi o'quv/yozuv masalalari bir necha pog'onalarga bo'lingan.

Eng yuqoridagi pog'onada foydalanuvchi dasturlari turadi. Ushbu dasturlar yuqori daraja tillari taqdim etadigan `write`, `read` va `puts` kabi o'quv/yozuv funksiyalaridan foydalanadi. Bu funksiyalar qurilma shaklida emas, balki dastur sifatida mavjud bo'lib, dasturchiga juda qulay bo'lgan o'quv/yozuv imkoniyatlarini beradi. Dastur ko'rinishidagi so'nggi pog'ona bu qurilmalarni boshqaruvchi dasturlar bo'lib, ular ma'lum bir tashqi moslama uchun mo'ljallab yozilgan bo'ladi. Masalan, tugmachalar taxtasini boshqaruvchisi qurilma simi orqali keluvchi elektr impulslarini operatsion tizimga bosilgan tugma belgisiga aylantirib beradi.

Dastur qatlamidan pastda moslamalar qatlami joylashgan. Bu qatlamda barcha jarayonlar elektr impulslari ko'rinishida yuz beradi. Mazkur qatlamda o'quv/yozuv har xil murakkab ko'rinishda bo'ladi. Chunki har bir tashqi qurilma ishlab chiqaradigan korxona o'z usulida moslama yaratadi va uni tuzishda turli xil murakkab usullaridan foydalanadi. Moslamaning bunday imkoniyatlarini elektr impulslari tilida ishga solish juda ham mushkul. Shu sababdan murakkab tuzilishga ega bo'lgan moslamalar uchun yana qo'shimcha tarzda *kontroller* qurilmasi ishlatiladi. Kontroller quyidagi vazifalarni bajaradi:

- dasturiy o'quv/yozuv qatlamidan yuqori darajali buyruqlarni qabul qilib, ularni tashqi moslamaga tushunarli bo'lgan elektr impulslariga aylantirib beradi;
- tashqi moslamadan elektr impulslarini qabul qilib, ularni yuqori darajali buyruqlarga aylantiradi.

Misol qilib kompyuteringizdagi diskdon kontrollerini keltirish mumkin. Ko'pchilik diskka axborot yozish maqsadida mashhur NERO dasturidan foydalanadi. Diskka yozilishi kerak bo'lgan ma'lum fayl NERO ga ko'rsatilgach, u axborotni bo'laklarga bo'lib diskdon boshqaruvchi dasturiga yubora boshlaydi. Boshqaruvchi dastur esa o'z navbatida qabul qilingan axborot bo'laklarini diskning qaysi yo'laklariga yozish kerakligini va shunga o'xshash fizik ko'rsatmalar to'plamini diskdon kontrolleriga bildiradi. Kontroller ushbu buyruqlarni *«diskning bosh yo'lagiga o't»* va *«tanlangan yo'lakka yozuvni bajar»* kabi oddiyroq bo'lgan buyruqlarga aylantirib, ularni elektr impulslari ko'rinishida diskdonga yetkazadi (rasmga qarang).



7-rasm.

Yana bir muhim tushuncha o'quv/yozuv portlaridir. Har bir axborot tashish kanallariga[1] ulangan moslama o'z manzillar to'plamiga ega. Bu manzillar to'plami odatda o'quv/yozuv *portlari* deb ataladi. Markaziy protsessor o'quv/yozuv moslamalariga asosiy xotira orqali murojaat qilishida portlardan foydalanadi. Assembler tilida portlar bilan to'g'ridan-to'g'ri ishlash uchun `IN`, `INS`, `OUT` va `OUTS` buyruqlari mavjud.

[1] «Axborot tashish kanallari» mavzusiga qarang.

4.4. Axborot tashish kanallari

Biz shu paytgacha markaziy protsessor, xotira va o'quv/yozuv moslamalari o'rtasida axborot almashinishi haqida ko'p gapirdik. Lekin ular o'rtasida axborot yetkazilishi aynan qanday amalga oshirilishi haqida umuman fikr yuritmadik. Bu vazifani axborot tashish kanallari amalga oshiradi. Bu vosita rus tilida *шина,* ingliz tilida *bus* deb ataladi.

ATK (**A**xborot **T**ashish **K**anallari) yuqori darajali cho'g'lanmas elektr sim bog'lamidan iborat bo'lib, har bir simdan bir bitga teng axborot jo'natiladi, masalan 0 va 1. Asosan uch xil ATK mavjud: *manzil ATKasi, qiymat ATKasi* va *boshqaruv ATKasi*.

Qiymat ATKasi xotira, markaziy protsessor va o'quv/yozuv moslamalari o'rtasida axborot tashiydi. Markaziy protsessor qiymat ATKsi orqali o'ziga kerakli qiymatlarni xotiradan o'qiydi va yangi ma'lumotlarni u yerga yozadi. Masalan, biror dasturni ishga tushirganingizda uning qiymatlari va buyruqlari qiymat ATKasi orqali protsessor, xotira va o'quv/yozuv moslamasi o'rtasida tashib turiladi. Qiymat ATKlari 8086, 8088, 80186 va 80288 turkumli protsessorlarda 20 bitli bo'lgan. Bu 20 ta simdan iborat bo'lgan bog'dan bir deganda 20 ta har xil elektr impulslari uzatiladi deganidir. Elektr kuchi kattaroq bo'lgan simda axborot miqdorini 1 ga teng deb, kamroq bo'lganida esa 0 ga teng deb olinadi. Keyinchalik muomalaga 24, 32 va 64 bitli ATKlar kirib keldi.

Asosiy ahamiyat kasb etadigan ATK bu manzil ATKsidir. Gap shundaki, markaziy protsessormi yo boshqasimi, kerakli ma'lumotni olish uchun qiymat ATKsini safarbar etar ekan, u o'sha axborot qayerda ekanligini ham ko'rsatishi kerak. Yuqorida ta'kidlanganidek bu axborot manzili bo'ladi. Yana o'sha hayotiy misolga qaytaylik. Tasavvur qiling siz teatr tamom bo'lgach, kiyim tashuvchi xodimga yuzlanib: «Mening kamzulimni olib bering!» - deysizda, na kamzul ilingan ilmoq raqami, na jeton to'g'risida so'z ochasiz. Shuning uchun qiymat manzili juda muhim.

20 bitli ATKalar juda jo'n bo'lib, ular yordamida faqatgina 1Mb ga teng axborot manzillari ifodalanib bo'lgan xolos. Buning sababini diskret matematikasidan xabari bo'lgan o'quvchi tezda anglab olgan bo'lsa kerak. Masalani oydinlashtirish uchun eslatib o'tish zarurki, 20 bit axborot 20 ta 1 yoki 0 lar ketma-ketligidan tashkil topgan, ikkilik sanoq tizimidagi sondir. E'tibor bergan bo'lsangiz bu ketma-ketlikda 1 va 0 lar har xil joylashishi mumkin.

Baytlar	**Ularning manzillari**
1-bayt [10111011]	00000000000000000000
2-bayt [00010110]	00000000000000000001
3-bayt [01101101]	00000000000000000010
. . .	. . .
2^{18}-bayt [11111101]	11111111111111111101
2^{19}-bayt [00101101]	11111111111111111110
2^{20}-bayt [10101010]	11111111111111111111

8-rasm.

Har bir joylashuv (kombinatsiya) bir bayt axborotning manzilini beradi. Demak, necha xil shunday joylashuv mavjud bo'lsa, shuncha baytni manzillash mumkin. Ko'rinib turibdiki, bunday kombinatsiyalar soni 2^{20} ga teng, yani 2^{20} ta bayt manzilini manzil ATKasi orqali berishimiz

mumkin. Bu degani 1 MB axborotdir. 8-rasmga qaraganda hammasi tushunarli bo'ladi. Rasmdagi o'ng ustunda manzil ATKasi orqali tashiladigan manzillar berilgan. Chap ustunda esa qiymat ATKasida tashiladigan mos manzillardagi baytlar xotirada joylashuv tartibi bilan berilgan. Burchakli qavslar ichida bayt bo'lib saqlanayotgan 8 bit axborotning o'zi misol tariqasida ko'rsatilgan. Demak, qiymat ATKasi orqali aynan o'sha 8 bit axborot tashiladi.

Markaziy protsessor ma'lum bir xotira manzilini ko'rsatdi deylik. U yerdan qiymat ATKasi orqali axborotni o'qimoqchimi yoki u yerga axborotni yozmoqchimi degan savolga boshqaruv ATKasi javob beradi. Qo'pol qilib aytganda uning ikkita simi bo'lib, ularning biri o'quv ikkinchisi yozuv deyiladi. Yozuv=0 va o'quv=1 bo'lganida yozuv yuz beradi, va aksincha. Quyidagi jadvalda ATKalarning ba'zi ko'rsatgichlari aks ettirilgan.

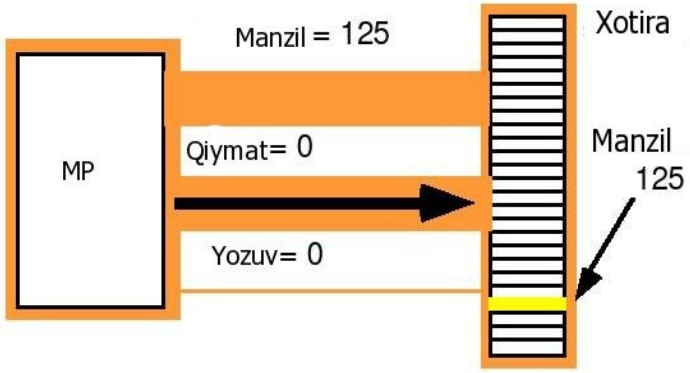
5-jadval

Protsessor turkumlari	**ATK o'lchamlari (bitlarda)**	**Tashilishi mumkin bo'lgan eng ko'p axborot hajmi (baytlarda)**	**Ixchamroq ko'rinishda**
8086, 8088, 80186, 80188	20	1 048 576	1 MB
80286, 80386sx	24	16 777 216	6 MB
80386dx	32	4 294 976 296	4 GB
804486, Pentium	32	4 294 976 296	4 GB
Pentium PRO, II, III, IV	64	64 719 476 736	64 GB

Keling endi an'anaviy dasturlash tillarida yozilgan bir misolni ATKlar yordamida qanday yechilishini ko'rib chiqsak. Masala shundan iboratki, xotiraning 125 katakchasiga bir bayt kattalikdagi 0 sonini yozish kerak. Bir bayt kattalikdagi 0 soni quyidagicha ko'rinishga ega:

0	0	0	0	0	0	0	0

Masalaning turli xil tillardagi yechimi quyidagicha bo'ladi.



9-rasm.

Paskaldagi ko'rinishi:

```
xotira[125]:=0;
```

C dagi ko'rinishi:

```
xotira[125]=0;
```

Assemblerdagi ko'rinishi:

```
mov byte[xotira+125] , 0
```

Bu jarayon ATKlar yordamida qanday yuz berishi 9-rasmda ko'rsatilgan.

4.5. Intel protsessor turkumlari

Kompyuterlarning bugungi kundagi salohiyatiga erishgunga qadar bir qancha bosqichlardan o'tilgan. Biz shu bosqichlarni Intel firmasi tomonidan yaratib kelingan 80x86 li protsessorlar oilasi misolida ko'rib chiqamiz.

8086, 8088	Bu turkum protsessorlari ilk yaratilgan xotirani haqiqiy manzillash usulida[1] ishlagan. Asosiy xotira hajmi 1 Mb gacha borgan. Shu bilan bir qatorda ularda 16 bitli bir nechta registrlar bo'lgan. Bular AX, BX, CX, DX, SI, DI, BP, SP, CS, DS, SS, ES, IP va FLAGS. Bu registrlardan AX, BX, CX va DX larini 8 bitdan kenja va to'ng'ich qismlarga bo'lib alohida ishlatish imkoniyati bor. Bu turkum protsessorlarida dastur bir necha segmentga, ya'ni **bo'limga** bo'lingan va segment o'lchami eng ko'pi bilan 64 Kb gacha bo'lgan.
80286	Bu protsessorlarga kelib xotirani manzillashning himoyalangan usuli foydalanila boshlangan. Registrlar esa o'sha 16 bitligicha qolgan. Shu jumladan bo'limlar o'lchami ham 64 Kb ligicha qolgan.
80386	Ushbu protsessorlarda ko'zga tashlanarli o'zgarishlar ro'y bergan. Asosiy o'zgarish bu 32 bitli registrlarning paydo bo'lishidir. Bunday registrlar nomlari oldiga E (**E**xtended - kengaytirilgan) old qo'shimcha qo'shilgan, ya'ni EAX, EBX, EDX, ESP, EBP, EIP, EDI, ESI va EFLAGS. CS, DS va SS registrlari 16 bitligicha qolgan va ularga qo'shimcha tarzda yangi FS va GS registrlari qo'shilgan. Bu protsessor ham himoyalangan usulda ishlagan. Lekin himoyalangan usulda 32 bitli ko'rinishda amalga oshirilgan, shunga yarasha barcha ATKlar ham 32 bitli qilib o'zgartirilgan. Bu turkum protsessorlarida segment o'lchami 4 Gb gacha borgan.
80486 **Pentium** **Pentium PRO**	Shu kabi keyingi chiqqan protsessorlarda esa o'zgarishlar kam edi. Lekin tezlik borasida ular ancha ilg'or bo'lgan.
Pentium MMX	Protsessorida ko'p vazifalik g'oyasi qo'llanilgan. G'oyaning asosiy mazmuni tasvirli dasturlarning tez bajarilishiga qaratilgan bo'lgan.
Pentium II	Bu mahsulot aslini olganda Pentium PRO va MMX ning birlashtirilgani edi. Bundan keyin chiqarilgan Pentium III, Pentium IV, Core, Core 2, Celeron, Xeon protsessorlariga kelsak, ularning oldingilardan asosan ishlash tezligi va xotirasining ulkanlashtirilganligi bilan ajralib turishini aytib o'tish mumkin.

[1] «Xotirani manzillash usullari» mavzusiga qarang.

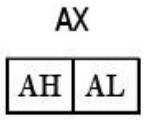
4.6. Markaziy protsessor registrlari

Registr juda katta tezlikda xotira vazifasini bajaradigan protsessorga juda yaqin joylashgan qurilma ekanligi shu paytgacha bo'lgan mavzularda ta'kidlab o'tilgan edi. Keling endi ularni batafsilroq o'rganamiz.

4.6.1. 16 bitli registrlar

8086 turkum protsessorlarida registrlar asosan 16 bitli bo'lgan. Registrlar vazifasiga ko'ra quyidagicha guruhlanadi:

- Umumiy maqsad registrlar: AX/AH/AL, BX/BH/BL, CX/CH/CL, DX/DH/DL, BP, SP, SI va DI. Birinchi to'rtta registrning kenja va to'ng'ich baytlariga alohida murojaat qilish imkoniyati bor. Masalan, AX registrini oladigan bo'lsak, u bir baytli AH (**H**igh - Yuqori) va AL (**L**ow – Quyi) qism registrlaridan iborat. Ammo quyi va yuqori qism registrlari AX dan mustaqil emas, ya'ni AL yoki AH ni qiymatini o'zgartirganda AX ning ham qiymati o'zgaradi. Umumiy maqsadlarda ishlatiladigan bu turdagi registrlar dasturchi ixtiyorida bo'ladi. Dasturlashda ular tez-tez murojaat qilinadigan qiymatlarni saqlashda foydalaniladi.



- Segment registrlari: CS, DS, SS va ES. Dastur bo'limlari boshlanish manzillari *selector* deb atalib, ularni saqlash oson bo'lishi uchun protsessor segment registrlarini taqdim etadi. Demak, segment registrlarida selectorlar, ya'ni bo'lim boshlanish manzillari saqlanadi. Ushbu registrlar vazifasi quyidagicha:
  - CS registrida dastur buyruqlaridan iborat bo'lgan bo'lim boshlanish manzili saqlanadi.
  - SS registrida joriy dastur stacki joylashgan bo'lim boshlanish manzili saqlanadi.
  - DS registrida dasturning tashqi[1] va ommaviy[2] qiymatlari joylashgan bo'limning boshlanish manzili saqlanadi.

  Bu turdagi registrlar operatsion tizim tomonidan boshqariladi. Ammo dasturchi ham ular qiymatlariga murojaat qilishi mumkin. Bular haqida «Haqiqiy usul» va «16 bitli himoyalangan usul» mavzularidan to'liq ma'lumot olishingiz mumkin.
- Boshqarish va holatni belgilash registrlari: IP va FLAGS. IP da doim navbatdagi bajarilishi lozim bo'lgan buyruqning dastur buyruqlar bo'limi boshiga nisbatan olingan manzili saqlanadi. FLAGS da esa eng oxirgi bajarilgan buyruq natijasi haqidagi ma'lumotlar alohida bit sifatida saqlanadi.

Sanab o'tilgan registr nomlarining kengaytirilgan holdagi ma'nolari:

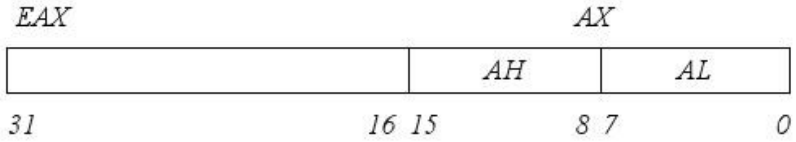
AX (**A**ccumulator register) – Akkumulator registri.
BX (**B**ase register) – Asos registri.
CX (**C**ounter register) – Hisob registri.
DX (**D**ata register) – Qiymat registri.
SI (**S**ource **I**ndex) – Manba tartib raqami registri.

[1] «Qismli dastur tuzish» mavzusiga qarang.
[2] «Birinchi dastur» mavzusiga qarang.

DI (**D**estination **I**ndex) – Maqsad tartib raqami registri.
IP (**I**nstruction **P**ointer) – Buyruq ko'rsatgichi registri.
SP (**S**tack **P**ointer) – Stack ko'rsatgichi registri.
BP (**B**ase **P**ointer) – Asos ko'rsatgichi registri.
CS (**C**ode **S**egment) – Buyruqlar bo'limi registri.
DS (**D**ata **S**egment) – Qiymat bo'limi registri.
SS (**S**tack **S**egment) – Stack bo'limi registri.
ES (**E**xtra **S**egment) – Qo'shimcha bo'limi registri.
FLAGS – Bayroqlar registri.

4.6.2. 32 bitli registrlar

80386 protessorlari davriga kelib 32 bitli registrlar ishlab chiqarila boshlandi. Tabiiyki, xotira hajmi kattalashgan registrlar ancha katta imkoniyatlarni ochib beradi. 10-rasmda umumiy maqsad registrlar tasvirlangan. Quyidagi rasmda registrlarning alohida bo'laklaridan foydalanganda ularning qanday ko'rinish hosil qilishi aks ettirilgan. Agar EAX registrini bitlar ko'rinishida tasavvur qilsak, u quyidagi ko'rinishga ega bo'ladi:

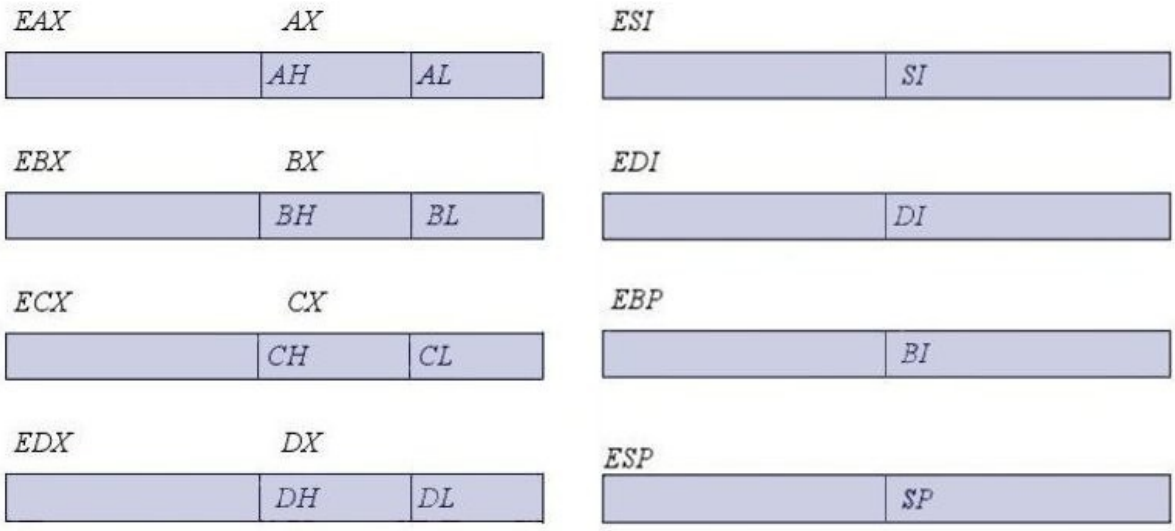


AX registri kengaytirilib, unga EAX nomi berilgan va AX shu registrning qism registriga aylanib qolgan. Bu AX dan ham ikki baytli registr sifatida foydalanishingiz mumkin degani. AL qism registri ham foydalanishda qolgan bo'lib, u EAX dagi 0 dan 7 gacha bo'lgan bitlarni tashkil etadi, ya'ni AX ning kenja(quyi) bayti. 8 dan 15 gacha bo'lgan bitlar to'plami esa, ya'ni AX ning to'ng'ich(yuqori) bayti, AH orqali muomalada bo'ladi. AL va AH qism registrlaridan siz bir baytga teng axborotni saqlashda foydalanishingiz mumkin. Shunday qilib, AX bu AH va AL ning yaxlit ko'rinishi ekanligi tushunarli bo'lsa kerak. EAX ning kenja ikki baytiga AX orqali murojaat qilish imkoni bo'lsada, uning to'ng'ich, ya'ni 16 dan 31 gacha bo'lgan bitlariga, ikki baytiga murojaat qilishga mo'ljallangan qism registri mavjud emas. Va nihoyat EAX ning o'ziga kelsak, uni to'rt baytli registr sifatida to'liqligicha ishlatishingiz mumkin. Umuman olganda, dasturlashda bir baytli axborotni saqlashda ham EAX ning o'zidan foydalanishingiz mumkin, chunki bir, ikki yoki uch baytli axborot to'rt baytli EAX ga muammosiz sig'adi. Faqat saqlayotgan qiymatingiz registr o'lchamidan katta bo'lmasligi darkor.

Yana bir bor ta'kidlash lozimki, EAX, AX, AH va AL bir-biridan mustaqil tarzda ishlatilsada, ular alohida tuzilgan registrlar emas, balki 32 bitdan tuzilgan yagona registr. Faqat masala shundaki, assembler tili bizga EAX ning kichik qismlaridan ham alohida foydalanish imkoniyatini beradi. Masalan, agar `0x8b` va `0x5a` sonlari AH va AL da mos ravishda saqlanayotgan bo'lsa, u holda AX ning qiymati `0x8b5a` soniga teng bo'ladi. Yuqoridagi barcha mulohazalar EBX, ECX va EDX registrlariga ham taalluqlidir.

Tartib raqam registrlari bo'lmish ESI/SI, EDI/DI esa faqat ikkiga bo'linadi, ya'ni faqat kenja ikki baytga alohida murojaat qilish mumkin. Ular asosan katta xotira bo'laklarining ichki manzillarini saqlashda qo'llaniladi. Ammo ulardan xohlagan maqsadda foydalanish mumkin. Ushbu registrlardan qanday unumli foydalanish «Assemblerda qoliplar va jadvallar» bobida tushuntirilgan.

ESP/SP va EBP/BP ko'rsatgich registrlari dinamik xotira bo'lmish stackni manzillashda qo'llanilib, ulardan o'z qiymatlaringizni saqlashda foydalanmaganingiz ma'qul. ESP va EBP ham faqat kenja ikki baytga alohida murojaat qilish imkoniyatini beradi xolos. Ular haqidagi to'liq ma'lumotni biz «Qismli dasturlash» bobida ko'rib chiqamiz.



10-rasm.

Segment registrlari o'sha 16 bitligicha qolgan. Ular qatoriga yana yangi FS va GS registrlari qo'shilgan. Bu registrlar ham o'ziga yarasha vazifaga ega ekanligini taxmin qilishimiz mumkin. Lekin ularni atroflicha muhokama qilish kitob ko'lamidan chetga chiqadi.

Boshqarish va holatni belgilash registrlari ham o'z navbatida kengaytirildi. Xususan IP registri EIP ga, FLAGS esa EFLAGS ga o'zgartirilgan. EFLAGS registri biroz boshqacharoq ma'noga ega. Bu registr bir butun ko'rinishda hech narsani anglatmaydi. Biroq uning har bir biti eng oxirgi bo'lib bajarilgan buyruq natijasi haqida ma'lumotlarni saqlaydi. Shuning uchun uning har bir bitiga alohida nom berilgan. Har bir bit *bayroq* deb nomlanib, agar u 1 bo'lsa bayroq o'rnatilgan hisoblanadi, aks holda o'rnatilmagan bo'lib chiqadi. Masalan, ayirish amalidan so'ng natija 0 ga teng bo'lgan bo'lsa tenglik bayrog'i (biti) o'rnatiladi, ya'ni u 1 ga teng bo'ladi. Bu bayroqlar protsessor tomonidan o'rnatiladi va dasturchi ularning qiymatini tekshirish orqali oxirgi buyruq natijasi qanchalik «to'g'ri» bo'lganini tekshirishi mumkin. Albatta, protsessor amallarni bajarayotganda hech qanday arifmetik xatoga yo'l qo'ymaydi, ammo u dasturchi hisobga olishi kerak bo'lgan mantiqiy xatolarga javob bermaydi. Bu kabi muammolar «Assembler tili asoslari» bobida atroflicha yoritilgan. Keling endi shu bayroqlarning eng muhimlari bilan tanishib chiqaylik. Ularning umumiy chizmasi rasmda tasvirlangan.

...	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
X	ID	VIP	VIF	AC	VM	RF	**X**	NT	IOPL		OF	DF	IF	TF	SF	ZF	**X**	AF	**X**	PF	**X**	CF

11-rasm.

- CF (**C**arry **F**lag) – ishorasiz sonlar ustida arifmetik amallar bajarganda natija noto'g'ri chiqsa, o'rnatiladi. Natijaning noto'g'ri chiqishiga asosan ikkita sabab bo'lishi mumkin:
  1. Qo'shish amalini bajarganda yig'indi ko'rsatilgan o'lchamli xotira sohasiga sig'masa, ya'ni tagma-tag qo'shishda to'ng'ich bitlarga kelganda bir-yodda yuz bersa.
  2. Ayirganda kamayuvchining to'ng'ich biti uchun qarz olinsa.
- PF (**P**arity **F**lag) – natijaning kenja baytidagi birga teng bitlar soni juft bo'lsa o'rnatiladi.
- ZF (**Z**ero **F**lag) – natija nolga teng bo'lgan bo'lsa o'rnatiladi.
- SF (**S**ign **F**lag) – natija manfiy bo'lgan bo'lsa o'rnatiladi. Bu ishorali arifmetikaga taalluqlidir.

- OF (**O**verflow **F**lag) – CF ga o'xshaydi, lekin bu ishorali sonlar bilan bog'liq.
- IOPL (**I**nput/**O**utput **P**rivilege **L**evel) – himoyalangan usulda[1] ishlaganda, o'quv yoki yozuv vazifalarini qay birining bajarilishi muhimligi aniqlab beradi.
- NF (**N**ested **F**lag) - himoyalangan usulda ishlaganda, bir vazifasining ikkinchisiga bog'liq yoki bog'liq emasligini aniqlab beradi.

4.7. Sonli qo'shma protsessor registrlari

Kompyuterda markaziy protsessordan tashqari o'nli kasr sonlar bilan ishlashga mo'ljallangan sonli qo'shma protsessor bo'lib, u ham o'z registrlariga ega. Sonli qo'shma protsessor registrlari: ST0, ST1, ST2, ST3, ST4, ST5, ST6 va ST7. Har biri 80 bitli bo'lgan bu registrlar o'nli kasr sonlarni saqlashda foydalaniladi va ular dasturchi ixtiyorida bo'ladi. To'liq ma'lumot uchun «O'nli kasrlar» bobini qarang.

4.8. Xotirani manzillash usullari

Dastur bajarilishi uchun qattiq diskdan xotiraga yuklangach, operatsion tizim oldida dasturning xotiradagi joylashuv manzillarini boshqarish vazifasi turadi. Kompyuterlar yaratilgandan buyon bu masala turlicha yechilib, protsessor rusumlari bilan birga takomillashib borgan.

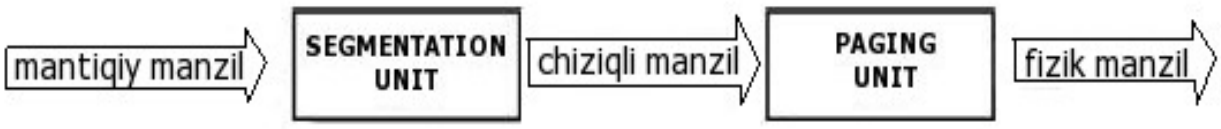
Mazkur kitobda biz Intel korporatsiyasi tomonidan ishlab chiqilgan va 8086 protsessor rusumidan boshlab hozirgi kungacha muomalaga bo'lib kelayotgan ikki xil xotirani manzillash usullari bilan tanishib chiqamiz. Keyingi uchta mavzuda shu usullar haqida gap ketadi.

Manzillar bilan ishlar ekanmiz, ularning qanday turlari borligini bilishimiz lozim. Asosan uch xil manzil turlari farqlanadi:

Mantiqiy manzil	Kompilyator tomonidan mashina tiliga o'girilgan buyruqlar yoki ular qiymatlarining manzillari. Har bir mantiqiy manzil *selector:offset* ko'rinishida bo'ladi. Bu yerda offset buyruq yoki uning qiymatlarining dastur bo'limi boshiga nisbatan olingan manzillarini beradi. Demak, har doim bo'lim boshi uchun offset nolga teng!
Chiziqli manzil	32 bitli ishorasiz butun son orqali ifodalanadigan manzil bo'lib, 4 Gb gacha xotirani manzillash imkoniyatini beradi, ya'ni 2^{32} yoki 4 294 967 296 ta xotira katakchalarini manzillay oladi. Bu manzil dastur bajarilish jarayoni paytida hisoblanib, dastur asosiy xotiraning aynan qay sohasiga joylashtirilishini aniqlaydi. Shuning uchun dastur asosiy xotiraga yuklangach undagi mantiqiy manzil chiziqli manzilga o'giriladi. Dasturchiga chiziqli manzillar bilan ishlash juda kam hollarda kerak bo'lib qoladi. Ushbu turdagi manzillar bilan ko'pincha operatsion tizim ish olib boradi. Kitobda xotira manzillari to'g'risida gap ketganda mantiqiy manzillar ko'zda tutiladi.
Fizik manzil	Kompyuterdagi xotira chipining katakchalarini manzillashda foydalaniladi. Ular MP tomonidan manzil ATKasi orqali jo'natiladigan manzillar bo'lib, 32-bitli ishorasiz butun son orqali ifodalanadi.

[1] «16 bitli himoyalangan usul» va «32 bitli himoyalangan usul» mavzulariga qarang.

MP mantiqiy manzilni chiziqli manzilga o'girishda *segmentation unit* qurilmasidan, chiziqli manzilni fizik manzilga o'girishda esa *paging unit* qurilmasidan foydalaniladi.



12-rasm.

Dasturni segmentlarga, ya'ni bo'limlarga ajratish g'oyasining yuzaga kelish sababi dasturchiga o'z dasturini bir-biri bilan mantiqiy bog'liq bo'lgan qism dasturlari yoki ommaviy va mahalliy qiymatlar sohalari kabi mag'zlarga bo'lish imkoniyatini berishdan iboratdir.

4.8.1. Haqiqiy usul

Haqiqiy usul bu xotirani manzillashning ilk ko'rinishilaridan biri bo'lib, 8086 protsessorlaridan boshlab 80286 protsessorlarigacha juda keng qo'llanib kelingan. Bu usul ancha oddiy bo'lib, oldingi protsessor rusumlari bilan ham ishlash imkoniyatini bergan.

O'sha davrda asosiy xotira 1 Mb gacha cheklangan bo'lib, ravshanki, undagi baytlarni manzillash uchun `0x00000` dan `0xFFFFF` gacha bo'lgan sonlar kerak bo'ladi, ya'ni xotiraning fizik manzili 0 va 2^{20} orasida o'zgaradi. Ko'rinib turibdiki, bunda manzil 20 bit o'lchamga ega bo'ladi. 8088 protsessorlarida dasturni bo'limlarga ajratib xotiraga yuklash nazariyasi qo'llangani tufayli segment registrlaridagi selector va buyruq yoki qiymatning nisbiy manzili offset, ya'ni quyidagicha fizik manzilga bitta qiymat sifatida o'giriladi:

$$fizik\ manzil = 16 * selector + offset$$

Demak, segment registrlarida bo'lim boshlanish manzilining 16 ga bo'lingan qiymati saqlanar ekan. Masalan, mantiqiy manzil selector=`0x12A5`, offset=`0x0015` bo'lsin, unda fizik manzil quyidagicha hisoblanadi: `0x12A5*0x10 + 0x0015 = 0x12A50 + 0x0015 = 0x12A65`. Ifodadagi `0x10` soni o'nlik sanoq tizimidagi 16 ga teng va shuning uchun ham o'n oltilik sanoq tizimida sonni 16 ga ko'paytirish qiyin emas, shunchaki son oxiriga bitta nol qo'shiladi xolos. Biz bu namunada `0x12A5:0x0015` mantiqiy manzili qanday qilib haqiqiy usul orqali fizik manzilga o'tkazilishini guvohi bo'ldik.

Umuman olganda, dastur bo'limining o'lchami offsetga bog'liq. Chunki bo'lim ichidagi baytlar offset orqali manzillanadi. Masalan, buyruqlar bo'limini oladigan bo'lsak, u yerda doim navbatdagi bajarilishi kerak bo'lgan buyruqning nisbiy manzili 16 bitli IP registrida saqlanadi. Bundan kelib chiqadiki, bo'lim o'lchami 2^{16} baytdan ortiq bo'la olmaydi, ya'ni 2^{16} b=2^6 * 2^{10} b=64 Kb. Ma'lumot uchun shuni aytib o'tishimiz kerakki, buyruqlar bo'limidagi buyruqning mantiqiy manzili registrlar orqali CS:IP ko'rinishida beriladi va haqiqiy usul yordamida fizik manzili CS*16+IP tarzda hisoblanadi.

Haqiqiy usulning o'ziga yarasha kamchiliklari ham bor:

- Xotiradagi baytning fizik manzilini yagona mantiqiy manzil bilan ifodalab bo'lmaydi. Masalan, 12A65 manzilini har xil mantiqiy manzillar orqali berish mumkin: 12A4:0025, 12A5:0015, 1295:0115... Vaholanki, xotiradagi har bir bayt manzili takrorlanmas tarzda aniqlangani ma'qul. Aks holda ancha chalkashliklar kelib chiqishi mumkin.

- Dastur bo'limi o'lchami ortig'i bilan 64 Kb gacha bo'ladi. Bo'lim 64 Kb dan katta bo'lsachi?
- Asosiy xotiradagi dastur bo'limlari bir-biridan unchalik yaxshi himoyalanmagan. Bir buyruqlar bo'limidagi buyruq boshqa bo'limdagi o'ziga taalluqli bo'lmagan qiymatlarni o'zgartish imkoniyatiga ega.

Haqiqiy usul ancha oldin ishlab chiqilgan bo'lsada, u yetarlicha oddiy bo'lib, hozirgi zamonaviy kompyuterlarda ham ba'zi maqsadda ishlatiladi. Masalan, kompyuteringizni yoqqaningizda birinchi bo'lib *BIOS* (**B**asic **I**nput/**O**utput **S**ystem – Asosiy O'quv/Yozuv Tizimi) ishga tushadi. BIOS bu maxsus xotira chipida saqlanadigan dasturlar toplami bo'lib, uning vazifasi operatsion tuzimning boshalang'ich bir bo'lagini asosiy xotiraga yuklashdan iborat, shundan keyingina Linux yoki Windows o'zini «eplab keta oladi». Mazkur bosqichda hali tizimning himoyalangan usulni amalga oshiradigan murakkab dasturlari ishga tushmagan bo'ladi va shuning uchun BIOSning o'zi haqiqiy usulda ishlaydi. Bunday olib qaraganda, hatto himoyalangan usulni amalga oshiradigan dasturlarning o'zi haqiqiy usulda ishlaydi.

4.8.2. 16 bitli himoyalangan usul

80286 protsessorlariga kelib yo'lga qo'yilgan xotirani manzillashning himoyalangan usuli haqiqiy usuldagi nuqsonlarni ma'lum darajada bartaraf etdi va bu usul hozirgacha barcha zamonaviy operatsion tizimlarda ishlatib kelinadi.

Himoyalangan usulda har bir dastur bo'limi u haqidagi 8 baytli axborot orqali aniqlanadi. Xotiradagi barcha bo'limlar uchun bo'lgan bu turgadi axborotlar *Descriptor Table* (Xususiyatlar Jadvali) jadvalida saqlanadi. Demak, jadvalning har bir qatori xotiraga yuklangan ma'lum bir bo'limning xususiyatlari haqidagi axborotni o'zida saqlaydi. Bo'lim xususiyatlari uning boshlanish manzili, o'lchami, turi va shunga o'xshash ma'lumotlardan iborat. Jadval esa xotiraning maxsus sohasida saqlanadi.

Himoyalangan usulda selector qiymati butunlay boshqacha ma'no kasb etadi. Haqiqiy usulda u bo'lim boshlanish manzilining 16 ga bo'lingan qiymatini o'zida saqlasa, mazkur usulda esa uning 16 ta biti quyidagicha qabul qilinadi:

- 15–3 bitlari xususiyatlar jadvalida joylashgan qator tartib raqamini beradi. Qator deganda selector aniqlayotgan bo'lim haqidagi axborotlar saqlanayotgan qator nazarda tutilmoqda.
- 2-bit jadval turini aniqlovchi bayroq. Xususiyatlar jadvalining mahalliy va ommaviy turlari mavjud.
- 1–0 bitlari bo'lim selectori CS segment registriga yuklanganda, bo'limdan bo'ladigan tashqi murojaatlarning muhimlik darajasini aniqlaydi.

Mazkur usulda offset o'zining avvalgi vazifasida qolgan. Manzillash 80286 protsessorida amalga oshirilgani tufayli offset 16 bitli qiymatligicha qolgan. Shunga qarab bo'lim o'lchami ham haqiqiy usulda bo'lgani kabi 64Kb gacha borgan xolos. Usulning 16 bitli deb nomlanishi ham shundan.

Haqiqiy usulda dastur bo'limlari u boshdan oyoq bajarilib bo'lguncha doim xotiraning aniq bir sohasida joylashtiriladi, ya'ni segment registrlari qiymati dastur yakunlanmaguncha o'zgarmaydi. Himoyalangan usulda esa virtual xotira tushunchasi kirib kelgan. Bunda faqat oniy vaqtda bajarilayotgan bo'lim qattiq diskdan asosiy xotiraga yuklanadi. Protsessor boshqa bo'limni bajarishga o'tganida esa oldingisi yana qattiq diskka qaytariladi va shunday qilib, jarayon tugaguncha bir bo'lim asosiy xotira va qattiq disk orasida ko'chib yuraveradi.

Virtual xotira tushunchasi faqat asosiy xotirada bo'limlar uchun joy yetishmagandagina ishga tushiriladi. Mazkur yondashuvning afzallik tomoni xotiraning ancha tejalishidir. Virtual xotira operatsion tizim tomonidan juda ustalik bilan amalga oshirilib, dasturlarni aynan shu usulda ishlashi uchun qayta yozishning keragi bo'lmaydi. Ma'lumot uchun shuni aytish lozimki, qattiq diskning bo'limlarni vaqtincha saqlaydigan sohasi *swap* deb ataladi.

Himoyalangan usulning ustunlik tomoni shundaki, dastur kodi va qiymatlari boshqa dasturlarnikidan himoyalangan bo'lib, ular o'zlariga tegishli xotira chegarasidan chiqishmaydi.

4.8.3. 32 bitli himoyalangan usul

Bu usul 32 bitli deb nomlanishining sababi 80386 protsessorlar oilasi davridan boshlab qo'llanila boshlanganidir. Bu protsessorlarga kelib 32 bitli registrlar muomalaga kiritilgan va offset o'lchamini ham 32 bitgacha oshirish imkoniyati bo'lgan. Dastur bo'limining o'lchami esa 4 Gb gacha yetgan. Yana bir katta farq esa virtual xotiraning ixchamlashtirilganidir. Endi asosiy xotiraga butun bir bo'lim emas, balki uning 4 Kb li bo'lakchasi yuklanadigan bo'lgan. Masalan, agar bo'lim o'lchami juda katta bo'lsa, uni to'liqligicha asosiy xotiraga yuklash dastur bajarilish vaqtini ancha sekinlashtirishi va noqulayliklarni keltirib chiqarishi mumkin. Endi esa uning kichik bo'lakchasi yuklanadi xolos. Bu 4 Kbli bo'lakcha *page* deb ataladi. Zamonaviy operatsion tizimlarning deyarli barchasi 32 bitli himoyalangan usulda ishlaydi. Windows NT/2000/XP, OS/2, Mac OS X va Linux shular jumlasidandir.

4.9. Markaziy protsessorda uzilishlar tushunchasi

Ma'lum bir jarayonning bajarilishi to'xtatilib, boshqasini bajarishga o'tish holati uzilish deb ataladi. Uzilishlar protsessorga signal sifatida maxsus qurilmalar orqali yetkaziladi va MP ularni tahlil qiladi.

Uzilishlar ikki turga, ya'ni ichki va tashqiga, bo'linadi:

- Ichki uzilishlar buyruqlarni bajarayotganda markaziy protsessor tomonidan yuzaga keltiriladi va joriy buyruq bajarilib bo'lingandan keyingina uzilish tahlil qilinadi. Bu kabi uzilishlar *istisno* ham deb ataladi.
- Tashqi uzilishlar boshqa kompyuter moslamalari tomonidan yuzaga keltiriladi va ular istalgan paytda sodir bo'lishi mumkin. Tashqi uzilishlar shunchaki uzilish deb yuritiladi.

Uzilishlarning bu kabi ikki turga bo'linishining sababi ularni sodir etuvchi omillarning turlicha ekanligidadir. Istisnoli holatlar asosan dasturdagi xatolar yoki operatsion tizimdagi noxushliklar evaziga yuzaga kelishi mumkin. Dasturda uchrashi mumkin bo'lgan xatolarga birinchi navbatda misol qilib, sonni nolga bo'lish holatini keltirish mumkin. Tizimdagi noxushliklarga esa oddiy misol etib dastur bo'limlarini xotiraga joylashtirishdagi «anglashilmovchiliklarni» keltirishimiz mumkin.

Tashqi uzilishlar yoki oddiy qilib aytganda, uzilishlar asosan o'quv/yozuv moslamalariga MP tomonidan e'tibor qaratilishi kerak bo'lganda yuzaga keltiriladi. Masalan, protsessor ma'lum bir jarayonni bajarib turibdi deylik, shu paytda foydalanuvchi sichqonchani qimirlatib qo'ydi. Mana shu yerda uzilish yuz beradi. MP bajarilib turgan jarayonini to'xtatadi va uning o'sha holatini belgilaydigan barcha qiymatlarni saqlab qo'yadi va barcha e'tibor sichqonchaga qaratiladi. Chunki sichqoncha siljitilganda ekrandagi sichqoncha ko'rsatgichini ham kerakli koordinataga burish lozim. Shuning uchun protsessor shu amallarni bajaruvchi dasturni yangi jarayon sifatida ishga tushirishi kerak. Bu kabi uzilishlarga olib keladigan juda ko'plab misollarni keltirish mumkin. Masalan, diskdonga diskning kiritilishi, tugmachlar taxtasidagi tugmani bosilishi va

hokazo. Bundan boshqa ham tizim ichidagi son mingta dasturlar va kompyuter ichki qurilmalari uzilishlarni amalga oshirib turadi.

Uzilish yuz berganda MP bajarilayotgan jarayonini to'xtatishdan oldin birinchi bo'lib jarayonning o'sha paytdagi holatini belgilovchi qiymatlarni (masalan, EIP va CS kabi registrlar qiymati) xotiraning maxsus sohasida saqlab qo'yadi. Bundan maqsad shuki, keyinchalik protsessor yana shu jarayonni davom ettirishga qaytganda bajarishning qayerda to'xtab qolganini aniqlay olishi kerak. Demak, uzilishni talab qilayotgan yangi jarayon eskisi to'liqligicha asrab qo'yilgandan so'nggina boshlanar ekan.

V bob
Assembler tilida dasturlash

Mazkur bobda biz assembler tilining asosiy buyruq va amallari bilan tanishamiz va ushbu tilda dastur qanday tuzilib ishga tushirilishini ko'rib chiqamiz.

5.1. Mashina tili

Yuqoridagi boblarda aytib o'tilganidek mashina tili faqat markaziy protsessor tushunadigan ikkilik sanoq tizimida ifodalanadigan buyruqlar ketma-ketligidir. Har bir buyruq o'zining maxsus raqamli kodiga ega bo'lib, u *buyruq kodi* deb ataladi. Har qanday dasturlash tilida yozilgan dastur ham ishga tushirilishi uchun mashina kodiga (tiliga) o'giriladi. Bu ishni kompilyator bajaradi. U orqali o'girilgan dastur *bajaruvchi kod* hisoblanadi, ya'ni uni markaziy protsessorda ishga tushirish mumkin. Bajaruvchi kod joylashgan fayl *bajaruvchi fayl* deb yuritiladi. Bundan buyon biz dasturni kompilyator orqali mashina tiliga o'girish jarayonini dasturni yig'ish deb ataymiz.

To'g'ri, hech qanday dasturlash tilidan foydalanmay turib to'g'ridan-to'g'ri mashina tilida dastur kodini yozishimiz mumkin, lekin bu nihoyatda murakkab. Chunki har bir buyruqning raqamli kodini eslab qolib, undan dasturlashda foydalanish inson uchun mushkul mashg'ulotdir. Fikrlarimizni tasdiqlash uchun misollar keltiramiz. Keling EBX registrida saqlanayotgan qiymatni EAX registriga ko'chirish kerak bo'ldi deylik, bu ish mashina tilidagi quyidagi buyruq yordamida bajariladi:

```
89D8
```

Bu yerda 89D8 o'n oltilik sanoq tizimidagi son bo'lib, mashina tilida EAX registriga EBX registrining qiymatini o'zlashtirish buyrug'i kodini anglatadi. Endi esa EAX registriga 45 sonini o'zlashtirish (ko'chirish, saqlash) kerak bo'ldi deylik:

```
B82D000000
```

Bu safar B8 soni EAX registriga o'zgarmas qiymatini ko'chirish buyrug'i kodi bo'lib kelayapti. 0000002D[1] esa 45 sonining o'n oltilik ko'rinishidir. Ko'rinib turibdiki, birgina EAX registri bilan bog'liq bo'lgan qiymat ko'chirish buyrug'ining o'zi ko'chirilayotgan manbaning turiga qarab har xil bo'lar ekan. Shuning o'ziyoq mashina tilida dasturlashdan hech qanday manfaat yo'qligini ko'rsatadi.

5.2. Assembler tili

Assembler so'zi ingliz tilidan olingan bo'lib, *yig'uvchi*, *jamlovchi* ma'nolarini anglatadi. Assemblerda yoziladigan dasturlar hamma dasturlash tillarida bo'lgani kabi, oddiy matn sifatida matn muharrirlari orqali yoziladi. Aslini olganda assembler ham bir dastur bo'lib, u berilgan matndagi assembler buyruqlarini mashina buyruqlariga o'giradi. Bu til quyi darajali til deb atalishining sababi shuki, assembler tili mashina tiliga juda yaqin ko'rinishga ega. Undagi har bir

[1] Bu yerda protsessor little endian ekanligi nazarda tutilgan. To'liq ma'lumot uchun «O'zgaruvchilarni e'lon qilish» mavzusiga qarang.

vazifa, topshiriq, amal (amal deganda, arifmetik amallarni bajaradigan buyruqlar ko'zda tutiladi) yoki buyruq aniq bir mashina darajasidagi buyruqqa to'g'ri keladi, ya'ni assemblerdagi har bir buyruq mos ravishda o'z yagona mashina kodiga ega. Masalan, yuqori darajali dasturlash tillaridagi ma'lum bir ifoda yoki buyruq mashina tilidagi 10-15 ta yaqin mashina buyruqlari bilan ifodalanadi. Shuning uchun ham assembler boshqa dasturlash tillari kompilyatorlaridan soddaroqligi bilan ajralib turadi.

> **Eslatma**: Ushbu kitobda bundan buyon «Assembler tilida yozilgan dastur» va «Assembler dasturi» iboralari bir xil ma'noda ishlatiladi.

5.3. O'zgaruvchilarni e'lon qilish

O'zgaruvchilar dasturlashda katta ahamiyatga ega. Matematika kursidan ma'lumki, funksiya ma'lum bir o'zgaruvchilarga bog'liq bo'ladi. O'zgaruvchi esa har xil qiymatlarni qabul qilib, doim o'zgarib turadi. Unga bog'liq holda funksiya qiymati ham o'zgarib turadi. Masalan, $f(x)=\frac{x+2}{4}$ funksiyasini oladigan bo'lsak, uning o'zgaruvchisi x hisoblanadi. Shu tariqa dasturni ham funksiya deb tushunsak, uning ham o'z o'zgaruvchilari mavjud. Lekin biz daftarda yechadigan tenglama yoki funksiya o'zgaruvchilaridan dasturdagi o'zgaruvchilarning farqi shundaki, ular uchun alohida qiymat saqlash maqsadida xotiradan joy ajratiladi. Qancha va qanday joy ajratish bu dasturchi vazifasiga kiradi. Demak, dasturdagi *o'zgaruvchilar* bu dasturchining kerakli qiymatlarini dasturning boshidan oxirigacha bo'lgan ish jarayonida saqlaydigan va xotirada ma'lum bir joyni ajratish yo'li bilan amalga oshiradigan obyektlardir.

Yuqoridagi boblarda xotira haqida ma'lumot olgan bo'lsangiz, ravshanki, xotiraga murojaat etganda kerakli xotira sohasining manzili orqali aloqaga kirishiladi. O'zgaruvchilarda ham shu narsa yuz beradi. Biz ularni birinchi marta e'lon qilganimizda, ya'ni dasturga tanishtirganimizda, xotiradan keraklicha joy ajratiladi va bu joy manzili o'zgaruvchi sifatida dasturga jo'natiladi. Biz dasturda o'zgaruvchiga murojaat qilish uchun berilgan manzildan foydalanamiz. Manzilning o'zi ham bunday olib qaraganda qandaydir bir son bo'lib, unga keyinchalik murojaat qilish uchun dasturchi eslab qolishiga to'g'ri keladi. Dasturlash davomida ba'zida shunday o'zgaruvchilarning 30-40 tasi kerak bo'lishi mumkin. Bu har bir o'zgaruvchidan foydalanish uchun uning manzilini eslab qolishi zarur deganidir. Inson ruhiyatidan ma'lumki, kishi katta-katta sonlardan ko'ra harflardan, belgilardan tashkil topgan nomlarni yoki hech bo'lmaganda bo'g'inlarni ancha yaxshi eslab qoladi. Shuni hisobga olgan holda dasturlashda ham matematikada bo'lgani kabi o'zgaruvchilarga ularni e'lon qilganda nom beriladi.

O'zgaruvchi nomi assembler tili nuqtai nazarida xotiradan ajratilgan joy manziliga qo'yilgan *nishondir*. Demak, nishon bu o'zgaruvchi uchun berilgan nom ekan. Dasturlash davomida o'zgaruvchiga murojaat qilish kerak bo'lgan joylarda nishonlar yoziladi. Nishonlar faqat dasturchi uchun ahamiyatga ega bo'lib, assembler dasturni yig'ish chog'ida ularni hammasini kerakli mantiqiy manzillarga almashtirib chiqadi.

O'zgaruvchi, ya'ni nishon nomlari harflardan, raqamlardan va `_`, `$`, `#`, `@`, `~`, `.`, `?` kabi maxsus belgilardan tashkil topishi mumkin. Nomlashda faqat lotin alifbosidagi harflar ishlatiladi. Nishon nomi harf bilan boshlanishi shart! Ammo ba'zi, maxsus hollarda nishon `.`, `_` va `?` belgilari bilan ham boshlanishi mumkin. Bu hollar haqida birozdan so'ng so'z yuritamiz. O'zgaruvchi nomi 4095 ta belgidan ortiq bo'lmasligi kerak.

Nishonga qanday nom berish dasturchiga havola. Asosiysi o'zgaruvchiga shunday nom berish kerakki, nom unda qanday qiymat saqlanayotgani haqida bildirib tursin. Masalan: `x`, `y`, `abc`, `F7`, `eng_katta_b`, `EKUK` va hokazo. NASM kompilyatori bosh harflarni kichik harflardan farqlaydi. Ya'ni o'zgaruvchilar nomi bir xil bo'lsada, faqat bosh yoki kichik harflarga

farq qilsa ular boshqa-boshqa o'zgaruvchilar hisoblanadi. Masalan: `hisob`, `Hisob`, `HISOB` va `hiSOB` boshqa-boshqa o'zaruvchilar hisoblanadi.

Endi bevosita o'zgaruvchi uchun xotiradan joy ajratishga to'xtalsak. Buni biz dasturlash tilida o'zgaruvchini e'lon qilish deb ataymiz. Buning asosan ikki xil yo'li bor. Birinchisida faqat keraklicha joy ajratiladi va manzil nishonga o'zlashtiriladi. Ikkinchisida esa joy ajratilishi bilan birga o'zgaruvchiga boshlang'ich qiymat ham beriladi, ya'ni o'zgaruvchi e'lon qilinishi bilan birga u *aniqlanadi* ham. Birinchi usulda e'lon qilinadigan o'zgaruvchilar dasturning `.bss` bo'limida yoziladi. Yozish qoidasida esa biz `RESX` direktivasidan foydalanamiz. `X` o'rnida 6-jadvalda ko'rsatilgan harflardan biri bo'lishi mumkin. Bu harflar qancha xotira ajratilishini aniqlaydi. *Direktiva* iborasiga kelsak, bu faqat assembler tushunadigan va umuman mashina tiliga o'girilmaydigan maxsus buyruqlar. Assembler `RESX` ni ko'rganida o'zgaruvchi uchun joy ajratilishi kerakligini anglaydi.

6-jadval

X	**To'liq nomi**	**O'lchami**
`B`	Byte	1 bayt
`W`	Word	2 bayt
`D`	Double word	4 bayt
`Q`	Quad	8 bayt
`O`	Octal	8 bayt (IEEE[1])
`T`	Ten	10 bayt

Qoidasi:

```
nishon_nomi     resx      n
```

Bu yerda `nishon_nomi` bu dasturchi o'zgaruvchiga beradigan nom, `RESX` - o'zgaruvchi o'lchami, `n` - natural son bo'lib, nechta *X* kattaligida joy ajratilishi kerakligini ko'rsatadi, ya'ni $x*n$ bayt joy ajratiladi. O'quvchiga yanada tushunarli bo'lishi uchun shuni eslatib o'tamizki, masalan, xotiradan bir bayt joy ajratsangiz u yerda 0 dan 2^8 gacha, ya'ni 255 sonigacha bo'lgan sonlarni sig'dira olasiz. Misollar:

```
x     resb      1
```

Bir bayt joy ajratib uning manzilini `x` o'zgaruvchisida (nishonida) saqlaymiz.

```
y     resb      2
```

`y` o'zgaruvchisi uchun ikki bayt joy ajratamiz, ya'ni u yerda 0 dan 2^{16}-1 gacha bo'lgan sonlarni sig'dira olamiz.

```
y2    resw      1
```

`y2` o'zgaruvchisining o'lchami bu yerda `y` bilan teng. `x`, `y` va `y2` o'zgaruvchilarida harfli va sonli qiymatlarni saqlashingiz mumkin. Shuni aytish lozimki, bitta harf yoki bitta belgi uchun bir bayt joy kerak bo'ladi.

```
kasr      resq      1     ;; o'nli kasr son uchun 8 bayt ajratiladi
q2        reso      2     ;; q2 uchun 8 baytdan 16 bayt ajratiladi
```

[1] IEEE (**I**nstitute of **E**lectrical and **E**lectronic **E**ngineers – Elektr va Elektron Muhandislar Instituti) tomonidan ishlab chiqilgan usul bo'yicha kasr sonlarni xotirada saqlashda kerak bo'ladigan o'lcham.

```
katta_kasr rest     1    ;; katta_kasr uchun 10 bayt ajratiladi
```

Eslatma: Assembler tilida nuqta verguldan (;) so'ng o'sha qator oxirigacha bo'lgan yozuvlar izoh deb tushuniladi va ularning dasturga hech qanday aloqasi yo'q.

Ikkinchi usulda e'lon qilinganida o'zgaruvchilar dasturning `.data` bo'limida yoziladi. Bunda biz `DX` direktivasidan foydalanamiz. *X* harfi `RESX` dagi kabi ma'noga ega.

Qoidasi:

```
nishon_nomi     dx    boshlang'ich_qiymat
```

Bu yerda oldingi usuldan farqli o'laroq o'zgaruvchiga `boshlang'ich_qiymat` ham o'zlashtiriladi. Misollar:

```
son   db    0
```

`son` o'zgaruvchisiga bir bayt joy ajratamiz va u yerga 0 sonini yozamiz.

```
eng_katta_2_baytli_son   dw   65535
```

Bu yerda o'zgaruvchi nomining o'zi unda nima saqlanayotgani haqida ma'lumot berib turibdi. Boshlang'ich qiymat ikki baytga sig'ishi mumkin bo'lgan eng katta ishorasiz son bo'lib, u $(1111111111111111)_2$ soniga teng.

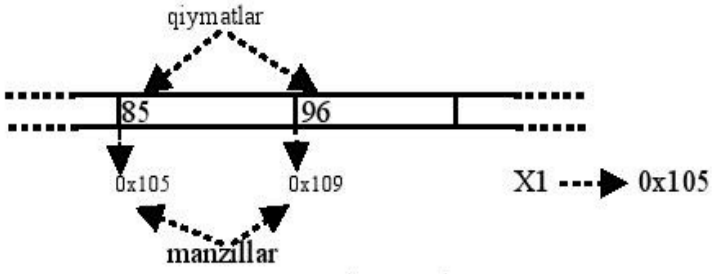
Eslatma: Agarda boshlang'ich qiymat 65535 dan hatto bittaga katta bo'lsa ham u `eng_katta_2_baytli_son` o'zgaruvchisiga sig'maydi va NASM kompilyatori dasturni yig'ish davomida xato yuz bergani haqida xabar beradi. Chunki biz o'zgaruvchini e'lon qilganimizda uni ikki baytdan iborat ekanligini `DW` direktivasi orqali kompilyatorga aytgan edik.

```
a     dd    -789           ;; to'rt bayt ajratiladi.
b     dq    5.2244e+14     ;; qo'sh aniqlikdagi kasr son.
c     dt    5.2244e+14     ;; kengaytirilgan aniqlikdagi kasr son.
```

Agar ikki yoki undan ko'p qiymatlar bo'lsa, ularni vergul orqali berish mumkin. Bu holda har bir qiymat uchun ketma-ket ravishda *X* orqali ko'rsatilgan kattalikda joy ajratiladi. Masalan:

```
X1    dd    85, 96
```

4 baytdan 2 marta joy ajratiladi va birinchisiga 85, ikkinchisiga 96 yoziladi. `X1` ga qaysi birining manzili yoziladi degan savol tug'iladi. Gap shundaki, manzillar bu yerda faqat 4 soniga farq qiladigan ikkita ketma-ket son bo'ladi va birinchisi `X1` ga yoziladi. Bu 13-rasmda tushunarli aks etgan.



13-rasm.

O'zgaruvchilarda belgili (harfli) yoki qatorli qiymatlarni ham saqlash mumkin. Belgili va qatorli qiymatlar nishon nomlari va assembler buyruqlaridan farqlanishi uchun qo'shtirnoqlar ichida yoziladi. Ma'lumot uchun shuni aytib o'tish kerak-ki, *IISC* (**I**nstitude of **I**nternational **S**tandards of **C**oding – Xalqaro Kodlash mezoni Instituti) mezoni bo'yicha bitta belgi yoki harf uchun eng kamida bir bayt joy kerak bo'ladi. Masalan, o'nta harfdan iborat so'zni xotirada saqlash uchun o'n baytli o'zgaruvchi kerak bo'ladi. Misollar:

```
qator     db    'S', 'a', 'l', 'o', "m", 0
```

`qator` o'zgaruvchisi uchun 6 bayt joy ajratiladi va u yerga `Salom` so'zi yoziladi va oxirida nol soni qo'yiladi. Nolni doim qatorlar oxirida qo'yish lozim, bu qator so'ngini bildiradi.

```
qator2    db    "Salom",0       ;; qator o'zgaruvchisi singari.
Haft      db    'G'             ;; bir baytli Harf o'zgaruvchisiga
                                ;; bosh G harfini boshlang'ich
                                ;; qiymat qilib beramiz.
```

Assembler tilida `"..."`, `'...'` yoki `` `...` `` turdagi qo'shtirnoqlar farqlanmaydi.

Ba'zida fayllarda saqlanayotgan katta-katta matnlar yoki musiqiy axborotlarni o'zgaruvchiga boshlang'ich qiymat sifatida o'zlashtirish ehtiyoji tug'uladi. `INCBIN` direktivasi shunday imkoniyatni bizga beradi. Unga boshlang'ich qiymat sifatida fayl nomi beriladi. Misollar:

```
Xat    incbin "matn.txt"              ;; Fayl to'liqligicha yuklanadi.
Musiqa incbin "madhiya.mp3",500       ;; Birinchi 500 bayt yuklanmaydi.
film   incbin "Buxoro.mpg", 500, 10   ;; 501-baytdan boshlab faqat 10
                                      ;; bayt yukalanadi.
```

Xulosa qilib aytganda, biz mazkur mavzuda ajratilgan xotira sohalariga qanday qilib nishonlar qo'yilishini o'rganib oldik. Dasturning `.data` va `.bss` bo'limlarida ko'rib chiqqanimiz kabi e'lon qilingan nishonlar *ommaviy* deb ataladi. Ommaviy nishonlarga dastur yozilayotgan faylning istalgan yeridan murojaat qilish mumkin.

5.4. O'zgarmaslar, ifodalar va direktivalar

Dasturlashda o'zgaruvchilar bilan bir qatorda o'zgarmaslar ham keng qo'llaniladi. O'zgarmas bu dastur kodida to'g'ridan-to'g'ri yozib ketiladigan qiymat. Masalan, yuqorida ko'rib chiqqan $f(x)=\frac{x+2}{4}$ matematik funksiyamizning o'zgarmas qiymatlari 2 va 4 hisoblanadi. Shunga o'xshab dasturning ham o'z o'zgarmaslari bo'ladi. Oldingi mavzuda biz o'zgarmaslar bilan ish ko'rib ulgurgan edik. `X1 dd 85, 96` ni oladigan bo'lsak, bu yerda 85 va 96 o'zgarmas hisoblanadi.

O'zgarmaslar to'rt turga bo'linadi: sonli, belgili(harfli), qatorli va kasr sonli. Sonli o'zgarmaslar har xil sanoq tizimidagi oddiy tarzda beriladigan sonlardir. Agar son maxsus qo'shimchalarsiz berilsa NASM ularni o'nli son sifatida qabul qiladi. Son ma'lum bir sanoq tizimiga tegishliligini bildirish uchun son oxirida quyidagi qo'shimchalardan birini qo'shish kerak bo'ladi: `H`, `Q` yoki `O`, `B`. Ular mos ravishda son o'n oltilik, sakkizlik va ikkilik sanoq tizimida ekanligini NASMga bildiradi. O'n oltilik son `0x` old qo'shimchasi orqali ham berilishi mumkin. Ammo, bir narsani unutmaslik kerakki, o'n oltilik son oxirida `H` qo'shimchasi bilan berilgan doim 0....9 raqamlaridan birortasi bilan boshlanishi shart! Aks holda ba'zan o'n oltilik sonni nishon nomidan farqlash qiyin bo'ladi. Masalan, `FADH` ni oladigan bo'lsak, bu nishon nomi yoki 4013 soni ekanligini anglash qiyin. Muammo ikki xil usulda yechilishi mumkin: `0FADH` yoki `0xFAD`.

Sonli o'zgarmas tarkibida pastki chiziq (_) ham ishlatilishi mumkin. Bunday imkoniyatning foydasi shuki, o'zgarmas qiymat uzunroq bo'lganda siz uni o'qish oson bo'lishi uchun bo'laklarga ajratish imkoniyatiga ega bo'lasiz. Misollar:

```
mov  eax , 456      ;; o'nlik sanoq tizimidagi 456 soni.
mov  eax , 0fff4dh  ;; o'n oltilik sanoq tizimidagi fff4d soni.
mov  eax , 4ah      ;; o'n oltilik sanoq tizimidagi 4a soni.
mov  eax , 0xfff4d  ;; o'n oltilik sanoq tizimidagi fff4d soni.
mov  eax , 567o     ;; sakkizlik sanoq tizimidagi 567 soni.
mov  eax , 567q     ;; sakkizlik sanoq tizimidagi 567 soni.
mov  eax , 110010100111b      ;; ikkilik son.
mov  eax , 1100_1010_0111b    ;; tushunarliroq ko'rinishda.
```

Yuqoridagi misollarning barchasida o'zgarmaslar EAX registriga yuklanadi.

> **Eslatma**: Boshlang'ich qiymat sifatida sonli o'zgarmaslar faqat `DB`, `DW`, `DD` va `DQ` direktivalariga berilishi mumkin.

Dasturlashda yana bir keng ishlatiladigan o'zgarmas turi belgili o'zgarmaslardir. *Belgi* deganda harf, raqam va tugmachalar taxtasidagi maxsus belgilar tushuniladi. Tugmachalar taxtasidagi barcha tugmalar raqamlab chiqilgan bo'lib, ular tizim uchun ma'lum bir sonni anglatadi. Bunday raqamlash usullarining ASCII, KOE-8, UTF-8 kabi har xil turlari mavjud bo'lib, ular 8 bitli mezon bo'yicha belgilarni son orqali ifodalashga xizmat qiladi. Sizning dasturlaringizda aynan qaysi raqamlash turi ishlatilayotganini operatsion tizimning sozlashlar bo'limidan aniqlashingiz mumkin.

Assemblerda sakkiz baytgacha bo'lgan belgilar ketma-ketligi belgili o'zgarmaslar hisoblanadi va ularga tizim oddiy son sifatida qaraydi. Lekin faqat dasturchi uchun ular belgi ko'rinishida namoyish etiladi. Misollar:

```
mov  ah , 't'       ;; AH ← t, ya'ni AH ← 116
mov  al , '2'       ;; AL ← 50
```

Misoldagi `'2'` belgili o'zgarmasni 2 soni bilan chalkashtirmaslik kerak. Tugmachalar taxtasidagi raqamlar 48 dan boshlab raqamlanadi va shunga ko'ra `'2'` ning tartib raqami 50 bo'ladi. Ushbu buyruqlardan so'ng AX registrida `'t2'` belgili o'zgarmasi bo'ladi.

```
mov  ebx , 'Havo'   ;; EBX ← 0x4861766F
```

Belgilarni son sifatida o'n oltilik sanoq tizimida ifodalash juda qulay, chunki barcha belgilarni ikki xonali o'n oltilik son bilan raqamlab chiqish mumkin. Shuning uchun misolda yuklanayotgan son sifatida o'n oltilik sonni keltirdik: `H=0x48, a=0x61, v=0x76` va `o=0x6F`.

Biz belgili o'zgarmaslarni `'...'` qo'shtirnog'i bilan chegaralashga kelishib olamiz. Ammo, ba'zida o'zgarmas tarkibida qo'shtirnoq bo'lishi kabi istisnoli holatlar yuzaga keladi. Masalan, quyidagi so'zlarni olaylik: A'lo, Ra'no, "iPhone",... Bunday holatlarda chegaralovchi qo'shtirnoqlar sifatida o'zgarmas tarkibidagidan boshqacha bo'lganlarini qo'llash kifoya:

```
mov  eax , "A'lo"       ;; EAX ← A'lo
mov  ax , "O'"          ;; AX ← O'
mov  eax , '"AB"'       ;; EAX ← "AB"
```

Yana shunday holatlar bo'ladiki, o'zgarmas tarkibida barcha qo'shtirnoq turlari qatnashadi. Masalan, "O'zbek" so'zini olaylik. Bunday holatlar uchun yuqoridagi masalaning ikkinchi xil yechimi mavjud. Qo'shtirnoqqa o'xshagan maxsus belgilar maxsus ma'noga ega emasligini

assemblerga bildirish uchun belgi oldidan qiya chiziqni (\) ishlatsa bo'ladi: `'\"O\'zbek\"'`. Mazkur usul qo'llanganda o'zgarmas `'...'` qo'shtirnog'i bilan chegaralanishi zarur.

Quyida shunga o'xshagan maxsus belgilar berilgan:

```
\'          tutuq belgisi (')
\"          qo'shtirnoq (")
\`          teskari tutuq (`)
\\          qiya chiziq (\)
\?          so'roq belgisi (?)
\t          satr boshi (ASCII 9)
\n          yangi satrga o'tish (ASCII 10)
\r          satr boshiga qaytish (ASCII 13)
\e          ESCAPE (ASCII 27)
\u263a      kulib turgan chehra, UTF-8 da
```

Qatorli o'zgarmas bu `INCBIN` yoki `DX` direktivalar oilasiga boshlang'ich qiymat qilib berilgan bir yoki undan ortiq belgili o'zgarmaslardan iborat bo'lgan ketma-ketlikdir. Biz qatorli o'zgarmaslarni `"..."` qo'shtirnog'i bilan chegaralashga kelishib olamiz. Misollar:

```
db    "Toshkent - O'zbekiston poytaxti",0     ;; 0 qator so'nggi.
db    'Toshkent - O'zbekiston poytaxti\0'     ;; Ikkinchi usuli.
db    "@#!%$^&()-={}",'\`\"\?\'\'        ;; 17 baytli o'zgarmas.
```

Kasr sonli o'zgarmaslarga ortiqcha ta'rif berish shart bo'lmasa kerak. Assemblerda bu turdagi o'zgarmaslar faqat `DB`, `DW`, `DD`, `DQ`, `DT` va `DO` direktivalari yoki `__float8__`, `__float16__`, `__float32__`, `__float64__`, `__float80m__`, `__float80e__`, `__float128l__` va `__float128h__` maxsus buyruqlariga qiymat sifatida berilishi mumkin.

```
db    7.8                 ;; Chorak aniqlikdagi kasr son.
dw    7.8                 ;; Yarim aniqlikdagi kasr son.
dd    7.888888888         ;; Birlamchi aniqlikdagi kasr son.
dd    7.888_888_888       ;; Tushunarliroq ko'rinish.
dq    7.8e+10             ;; 7.8 * 10^10, qo'sh aniqlik.
dq    7.8e-10             ;; 7.8 * 10^-10.
```

Kasr sonlarni o'n oltilik sanoq tizimida ham yozish mumkin:

```
mov   eax , __float32__(0x7.8)      ;; 32 bitli o'zgarmas sifatida.
mov   ax , __float16__(-0x7.8p+2)   ;; -0x7.8p+2=-0x7.8*2^2=-0x1E.0
```

O'nlik sanoq tizimidagi kasr sonlarda ishlatilgan `E` belgisi o'n oltilik kasr sonlar bilan ishlatilishi mumkin emas. Aks holda `0xE` soni bilan chalkashib ketishi mumkin. O'n oltilik kasr sonlar bilan ishlatiladigan `P` belgisini ham o'nlik o'zgarmaslar bilan ishlatib bo'lmaydi (Kasr sonlar haqida batafsilroq ma'lumot uchun «O'nli kasrlar» bobiga qarang).

Assembler dasturlash tilida ham boshqa tillarda bo'lgani kabi ifodalardan foydalanish imkoniyati bor. Mantiqiy yoki arifmetik amallar orqali bog'langan o'zgarmaslar ketma-ketligi *ifoda* deyiladi. NASM dasturni yig'ish chog'ida bu ifodalarni hisoblab, ularni o'zgarmas sonlarga almashtirib chiqadi, ya'ni bajaruvchi kodgacha ifodalar yetib bormaydi.

Quyida amallar birinchi bo'lib bajarilish tartibi bo'yicha kichikdan kattaga qarab berilgan:

- **Mantiqiy amallar**
  - `|` : Bitma-bit mantiqiy qo'shish amali. Assemblerdagi `OR` buyrug'i bilan ishlash tartibi bir xil.

- `^` : Bitma-bit mantiqiy istisnoli qo'shish amali. Assemblerdagi `XOR` buyrug'i bilan ishlash tartibi bir xil.
- `&` : Bitma-bit mantiqiy ko'paytirish amali. Assemblerdagi `AND` buyrug'i bilan ishlash tartibi bir xil.
- `<<` va `>>` : Bitlarni siljitish amali. Assemblerdagi `SHL` va `SHR` buyruqlari bilan mos ravishda ishlash tartiblari bir xil.

- **Arifmetik amallar**
  - `+` va `-` : Oddiy qo'shish va ayirish amallari.
  - `*`, `/`, `//`, `%` va `%%` : Ko'paytirish va bo'lish amallari. Bu yerda `*` ko'paytirish, `/` va `//` mos ravishda ishorasiz va ishorali sonlarni bo'lish, `%` va `%%` esa mos ravishda ishorasiz va ishorali sonlarni bo'lishdan qoladigan qoldig'ini topish amallari.
- **Bir qiymatli amallar**
  - `-`, `~` va `!` : Bu amallar faqat bitta o'zgarmasga ta'sir qilib, eng birinchi bo'lib bajariladi. `-` manfiy qiymatga aylantiradi, `~` qiymatning birlamchi to'ldiruvchisini[1] hisoblaydi va `!` mantiqiy inkor. Mantiqiy inkorda agar qiymat noldan farqli bo'lsa, natija nolga teng va aksincha, agar qiymat nolga teng bo'lsa natija birga teng.

Ko'rib chiqilgan bitlar ustida amalga oshiriladigan mantiqiy va siljitish amallari «Mantiqiy amallar va bitlarni siljitish buyruqlari» bobida keng yoritilgan. Ammo u yerdagi assembler buyruqlari bilan ifodalarda qatnashadigan mazkur amallarni chalkashtirmaslik kerak. Ular ma'no jihatdan bir xil bo'lsada, ifodalardagi mantiqiy va arifmetik amallarning mashina tiliga o'giriladigan assembler buyruqlariga hech qanday aloqasi yo'q! Misollar:

```
a     db    100 + 4*10          ;; a = 140
b     db    210/4 - 2           ;; b = 50
c     dw    -100 + !100         ;; c = -100
d     dd    ~1111_0000b         ;; d = 0000_1111b
k     dq    -0x7A // -0xE       ;; k = 0x8
```

Ifodalarda kasr sonli o'zgarmaslar qatnashishi mumkin emas. Chunki NASM kasr sonlarni hisoblay olmaydi. Shuning uchun ham ifodalarda bo'linma yaxlitlab olinadi. Masalan, 210/4=52.5 emas, balki 50 ga teng.

```
mov   eax , 0000_0001b << 2     ;; EAX ← 0000_0100b, x<<y = x * 2^y
mov   eax , 0000_0100b >> 2     ;; EAX ← 0000_0001b, x>>y = x // 2^y
mov   al , 110b | 101b          ;; AL ← 111b
mov   al , 110b & 101b          ;; AL ← 100b
mov   al , 110b ^ 101b          ;; AL ← 011b
```

Ifodalarda qatnashishi mumkin bo'lgan yana bir maxsus belgi mavjud: `$`. Bu belgi o'zi ishlatilayotgan dastur qatorining boshlanish offset manzilini beradi.

Ba'zida dastur davomida o'zgarmas qiymatlarni o'zimiz uchun belgilab olishimiz juda foydali bo'ladi. Bunday imkoniyatni bizga `EQU` direktivasi beradi:

```
belgilash_nomi   equ   ifoda
```

Masalan:

```
equ   25
```

[1] «Ishorali va ishorasiz butun sonlar» mavzusiga qarang.

Shu yozuvdan so'ng dasturda `uzunlik` bu 25 sonini anglatuvchi kattalik bo'ladi. Bu yerda hech qanday xotiradan joy ajratilmaydi. Faqat dasturning barcha `uzunlik` uchraydigan joylari assembler tomonidan 25 soniga almashtirib chiqiladi. O'zgaruvchilarni e'lon qilishdan farqli o'laroq o'zgarmas kattaliklarni bu usulda aniqlash dasturning istalgan bo'limida amalga oshirilishi mumkin.

Assembler tilida ko'pincha qatorli qiymatlar bilan ishlaganda ularning uzunligini eslab qolish talab etiladi. Shuning uchun qatorli o'zgaruvchilarni e'lon qilganda, ularning uzunligini o'zgarmas sifatida tezda belgilab qo'ygan ma'qul.

```
qator      db    "Salom",0
qator_uzunligi equ  $-qator
```

`qator` nishonida `'S'` belgisining manzili saqlanilayotganini va `$` keyin qator boshlanish manzilini berishini hisobga olsak, `$-qator` ifodasi `qator` uchun ajratilgan baytlar sonini beradi, ya'ni 6. O'zgarmaslarni bu kabi belgilab qo'yish juda ham qulay bo'lib, agar qator o'zgaruvchining boshlang'ich qiymatini keyinchalik o'zgartirgan taqdiringizda ham `qator_uzunligi` to'g'ri uzunlikni ko'rsatib turadi.

Yana bir foydali direktivalardan biri `TIMES` direktivasidir. Bu direktiva o'zidan keyin keladigan buyruqlarni berilgan son marta takror bajarilishini ta'minlaydi. Masalan:

```
chiziq     times      25    resb 1
```

`chiziq` o'zgaruvchisi uchun 25 marta bir baytdan joy ajratib chiqiladi. Bu ishni, albatta, quyidagicha ham amalga oshirish mumkin:

```
chiziq     resb       25
```

Bu ikki usul o'rtasida to'g'ri ishlash nuqtai nazaridan hech qanday farq yo'q. Ammo `TIMES` bilan bo'lgan holat kompilyator tomonidan 25 marta sekinroq yig'iladi. Bu direktiva ishni sekinlashtirsada, uning ham kerakli tomonlari bor. Masalan, 25 bayt ajaratish bilan birga har bir baytga 1 sonini boshlang'ich qiymat sifatida berish kerak bo'lsa, `TIMES` asqatadi:

```
chiziq     times      25    db    1
```

`TIMES` ga takrorlanishlar soni ifoda orqali ham berilishi mumkin. Masalan, xotiradan 100 bayt joy ajratib, u yerga "`Hammaga salom`" qatorini boshlang'ich qiymat sifatida yozish va qolgan baytlarni nol bilan to'ldirish kerak bo'lsin. Ko'rinib turibdiki, nol o'zlashtirilishi kerak bo'lgan baytlar soni $100-k$ ga teng, bu yerda k "`Hammaga salom`" qatori uzunligiga teng, ya'ni 13. Demak:

```
chiziq     db    "Hammaga salom"
           times      100-($-chiziq) db    0
```

Birinchi qatorda `chiziq` o'zgaruvchisi uchun 13 bayt ajratilib, u yerga berilgan "`Hammaga salom`" qiymati yozilishi tushunarli bo'lsa kerak. Ikkinchi qatorda esa ajratilgan baytlar ketidan yana `100-($-chiziq)` marta bir baytdan joy ajratilib, u yerga 0 yoziladi. Bizning holatimizda `100-($-chiziq)` ifodasi 87 ga teng.

Eslatma: Ifodalarda qavslar ham ishlatilishi mumkin.

5.5. Asosiy amallar va ular qabul qiladigan qiymatlar

Kitobda amallar va buyruqlar degan atamalar ko'p ishlatilgan. Gap shundaki, bularning ikkalasi ham mashina uchun bir xil ma'noga ega va o'z mashina kodiga ega. Lekin biz kitobda amallar deganda qo'shish ayirish kabi arifmetik buyruqlarni nazarda tutganmiz.

Buyruqlar va amallar ma'lum bir qiymatlar ustida har xil vazifalarni bajaradigan obyektlardir. Ularni biz matematikadagi oddiy funksiyalarga o'xshatishimiz mumkin. Masalan $f(x,y)=x+y$ funksiyasini olaylik. U o'ziga beriladigan x va y o'zgaruvchilari qiymatlarini yig'indisini aniqlaydi. Shunga o'xshash assembler tilida ham mazkur funksiyalarni amalga oshiruvchi buyruqlar mavjud. Ularga qiymat sifatida o'zgaruvchilar, registrlar yoki o'zgarmaslar berilishi mumkin. Assamblerda buyruqlar dasturning `.text` bo'limida yoziladi. Har bir buyruq alohida qatorda yoziladi. Buyruqdan so'ng boshqa dasturlash tillaridan farqli o'laroq hech qanday maxsus belgi qo'yilmaydi. `.text` bo'limidagi qator quyidagicha umumiy ko'rinishga ega:

```
nishon:   buyruq    qiymatlar       ; izohlar
```

Bu yerda `buyruq` dan boshqa hamma obyektlar bo'lishi majburiy bo'lmagan obyektlardir. Quyida assembler buyruqlarining eng keraklilari izohlangan.

> **Eslatma**: Assembler tili bo'limlari bo'lmish `.bss`, `.data`, va `.text` yuqorida ko'rib chiqilgan operatsion tizim nuqtai nazaridagi dastur bo'limlarining soddalashtirilgan ko'rinishidir. CS registrida saqlanadigan dastur buyruqlaridan iborat bo'lgan bo'limga `.text` to'g'ri kelsa, DS registrida saqlanadigan dasturning tashqi va ommaviy qiymatlari joylashgan bo'limga esa `.bss` va `.data` lar mos keladi.

Bizga tanish bo'lgan `MOV` buyrug'i berilgan birinchi o'zgaruvchiga ikkinchi bo'lib berilgan qiymatni o'zlashtiradi(ko'chiradi).

Qoidasi:

```
mov  maqsad , manba
```

Bu yerda `manba` qiymati `maqsad` ga nusxa qilib ko'chiriladi va `maqsad` ning oldingi qiymati o'chib ketadi. Lekin ba'zi cheklovlar mavjud bo'lib, masalan, `manba` ham `maqsad` ham bir vaqtning o'zida xotira bo'la olmaydi, ya'ni bir nishon qiymatini ikkinchisiga to'g'ridan-to'g'ri o'zlashtirib bo'lmaydi. `manba` va `maqsad` qanday obyektlar bo'lishi mumkinligi 7-jadvalda batafsil ko'rsatilgan. Yana shuni ta'kidlash lozimki, `manba` va `maqsad` o'lchami bir xil bo'lishi kerak. Masalan, to'rt baytli o'zgaruvchining qiymatini bir baytli o'zgaruvchiga ko'chira olmaysiz.

7-jadval

Maqsad	Manba
Registr	Registr
Registr	Xotira
Registr	O'zgarmas
Xotira	Registr
Xotira	O'zgarmas

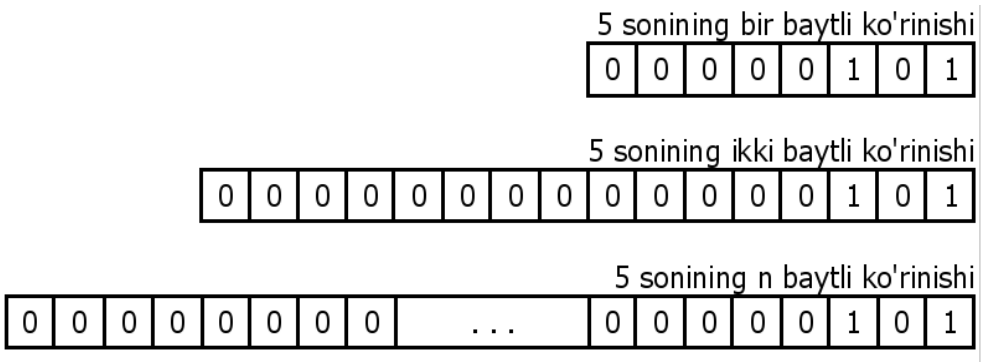
Misollar:

```
mov  eax , 0        ;; EAX registri qiymati nol qilinadi, EAX ← 0.
mov  eax , edx      ;; EDX qiymati EAX ga ko'chiriladi, EAX ← EDX.
```

O'zgaruvchilarni e'lon qilish mavzusida `X1` nomli o'zgaruvchini e'lon qilgan edik. Keling undagi 85 sonini ECX registriga ko'chirishga harakat qilib ko'ramiz:

```
mov  ecx , X1       ;; e'tibor bering, ECX ← 0x105 !!!
```

Bir qarashda hammasi joyidagidek, lekin qanchalik «kutulmagan» bo'lmasin, ECX da `0x105` soni ko'chiriladi. Chunki `X1` nishoni bor yog'i `.data` bo'limidagi `0x105` ga teng bo'lgan mantiqiy manzildir. NASM `mov ecx , X1` buyrug'ini ko'rgach uni `mov ecx , 0x105` ga almashtiradi.



14-rasm.

Assembler buyruqlariga qiymat sifatida beriladigan manzil *foydali* manzil deyiladi va qiymat shunchaki qiymat emas, balki aynan manzil ekanligini NASMga bildirish uchun u burchakli qavsga (`[]`) olinadi. Assemblerda burchakli qavsga olingan qiymat manzil sifatida qabul qilinadi va berilgan buyruqqa qarab o'sha manzildagi qiymatga murojaat amalga oshiriladi. Burchakli qavs ichida nishon, sonli o'zgasmaslar va registrlar ifoda ko'rinishida berilishi mumkin. Ifoda assembler tomonidan dasturni yig'ish chog'ida hisoblanib, natijaviy manzilga almashtiriladi[1]. Demak, yuqoridagi muammo quyidagicha hal etiladi:

```
mov  ecx , [X1]     ;; ECX ← 85 va nihoyat!
```

Endigi vazifa `X1` dagi ikkinchi 4 baytda saqlanayotgan 96 sonini EDX ga ko'chirish bo'lsin. Bizga ma'lumki, `X1` da 85 saqlanayotgan to'rt baytning birinchi baytining manzili, ya'ni `0x105` turibdi. 96 saqlanayotgan sohaning manzili esa 4 baytdan keyin boshlanadi. 13-Rasmda ko'rsatilganidek, bu `0x109` bo'ladi. Demak, bizga kerakli manzil `X1` ga 4 ni qo'shish orqali topiladi:

```
mov  edx , [X1+4]   ;; EDX ← 96
```

Farqlang:

```
mov  edx , X1+4     ;; EDX ← 0x109
```

Keling endi avval e'lon qilgan `son` nomli o'zgaruvchimizga besh sonini o'zlashtirishga harakat qilib ko'ramiz:

```
mov  [son] , 5      ;; Xato !!!
```

[1] Burchakli qavs ichida uchrashi mumkin bo'lgan ifodalarning barchasi «Vositali manzillash» va «`LEA` buyrug'i» mavzularida to'liq yoritilgan.

Dasturni kompilyatorga berganda mana shu qatorda xato borligi haqida xabar olasiz. Assembler `.data` yoki `.bss` dastur bo'limlarida ko'rsatilgan o'zgaruvchilarga mos ravishda joy ajratishga ajratadi-yu, so'ng aynan qancha baytdan ajratganini dastur davomida eslab qolmaydi. Buyruqni bajarishga kelganda esa 5 sonini `son` o'zgaruvchisiga qanday kattalikda yozishni bilmaydi. Chunki `son` o'zgaruvchisining o'lchamini assembler bilmaydi va 5 sonini u xohlagan o'lchamida «tasavvur» qilishi mumkin. Unga chap tomondan qo'shimcha nollarni biriktirish qiyin emas (14-rasmga qarang). Shuning uchun `maqsad` oldida uning manbani qanday o'lchamda qabul qilishi ko'rsatilishi kerak. Bizning holatimzda esa bu bir bayt bo'lib, nishondan oldin keladigan `byte` aniqlovchisi orqali kompilyatorga bildiriladi:

```
mov  byte[son] , 5            ;; son ← 5
```

8-jadvalda barcha o'lchamlar uchun aniqlovchilar berilgan. O'zgaruvchi oldiga uning o'lchamidan kichik o'lchamni beruvchi aniqlovchi qo'yish mumkin, ammo kattasini qo'yish mumkin emas. Masalan, ikki baytli `y2` o'zgaruvchisi oldiga `byte` qo'yilsa, ko'chiriladigan qiymat o'zgaruvchining birinchi baytiga yoziladi. Lekin uning oldiga `dword` ni qo'yib bo'lmaydi.

8-jadval

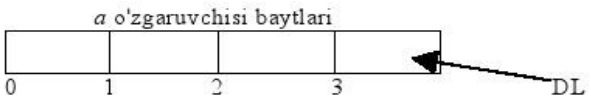
Aniqlovchi	O'lcham
`byte`	1 bayt
`word`	2 bayt
`dword`	4 bayt
`qword`[1]	8 bayt
`tword`[1]	10 bayt

Misollar:

```
mov  word[y2] , 0x999         ;; y2 ← 0x999
mov  dword[a] , 11101b        ;; a ← 1110b
```

Sinchkov o'quvchi registrlar bilan bo'lgan holatlarda biz old aniqlovchisidan foydalanmaganligimizni payqab qolgan bo'lsa ajab emas. Biz u yerda ham bu aniqlovchilardan foydalanishimiz mumkin edi. Lekin bu shart emas. Chunki assembler registr nomidan uning qanday o'lchamga ega ekanligini biladi va shu o'lcham asosida buyruqni amalga oshiradi. Masalan:

```
mov  eax , 0          ;; yoki
mov  dword eax , 0
mov  al , [c]         ;; c ning birinchi bayti AL ga ko'chiriladi
mov  byte al , [c]    ;; Yuqoridagi bilan bir xil.
mov  [a] , bx         ;; a ning birinchi 2 baytiga BX
                      ;; o'zlashtriladi.
mov  [a+3] , dl       ;; a ning oxirgi 4-baytiga DL yoziladi.
```

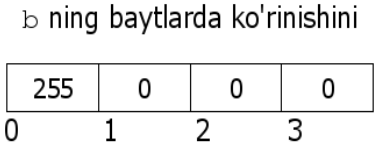


[1] `qword`, `tword` aniqlovchilari sonli qo'shma protsessor buyruqlariga tegishli («O'nli kasrlar» bobga qarang).

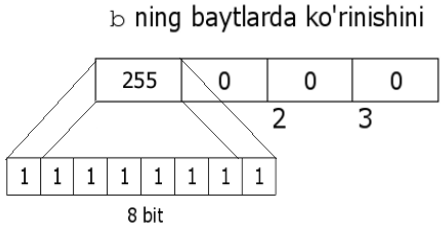
Qiymatlar xotira katakchalarida aynan qay tarzda saqlanishini bilish dasturlashda ancha asqatadi. Shuning uchun bu haqida ham bir og'iz so'z. Quyidagi buyruqni ko'rib chiqaylik:

```
mov  dword[b] , 255
```

Bir bayt joyga 255 gacha bo'lgan sonlar siqqani tufayli 255 soni `b` ning birinchi baytiga yozib qo'yiladi (rasmga qarang).



Keling birinchi baytni ikkilik ko'rinishida tasavvur qilib ko'raylik:



15-rasm.

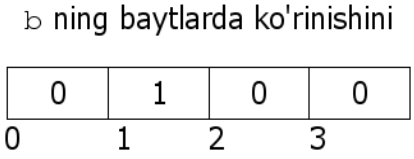
15-rasmdan ko'rinib turibdiki, 255 dan har qanday katta son birinchi baytga sig'magan bo'lar edi. Agar biz 256 ni `b` ga o'zlashtirganimizda edi, uning ikkinchi bayti ham ishga solingan bo'lar edi:

```
mov  dword[b] , 256
```

Bu holda ikkinchi bayt ishga solinadi va unga ortib qolgan bit yoziladi. Ya'ni:

```
            11111111   (255)
+
            00000001   (1)
=
00000001 00000000   (256)
2 - bayt   1 - bayt
```

16-rasmda ham bu tushunarli qilib tasvirlangan.

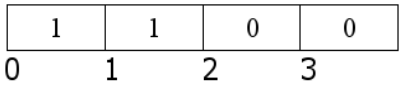


16-rasm.

Qo'shimcha tarzda 257 sonini ko'rib chiqamiz. Bir baytga eng ko'pi bilan 255 sig'ishini hisobga olib 257 ni 255+2 ko'rinishida tasavvur qilamiz:

```
            11111111   (255)
+
            00000010   (2)
=
00000001  00000001   (257)
 2 - bayt     1 - bayt
```

To'rt baytli ko'rinishda quyidagicha bo'ladi:



Xulosa qilib shuni aytish mumkinki, ketma-ket olingan bir qancha baytlar qiymati yaxlit son sifatida quyidagi formula orqali hisoblanadi: $N = bayt_1 * (2^8)^0 + bayt_2 * (2^8)^1 + \ldots + bayt_n * (2^8)^{(n-1)}$

Yuqorida tasvirlangan xotiraning qiymatlar bilan to'ldirilishi chapdan o'ngga qarab amalga oshirilayotganiga hayron bo'layotgandirsiz. Masalan, nega `b` ning ko'rinishi unga 255 ni o'zlashtirgandan so'ng quyidagicha:

0	0	0	255

emas-u, aksincha 15-rasmdagidek? Buning sababi Intel kompyuterlarida xotirani qiymatlar bilan to'ldirish shunday amalga oshirilishidir va bunday kompyuterlar *little endian* deb ataladi. Biroq *big endian* turkumli kompyuterlar ham bor. Ularda hammasi biz xohlaganimizdek amalga oshiriladi. Bunday kompyuterlar ko'pincha katta-katta tizimlarda ishlatiladi va ularni *mainframe* deb ham atashadi.

`MOV` dan so'ng yana ikkita muhim amallar, ya'ni qo'shish va ayirish amallari bilan tanishamiz.

Qo'shish amali qoidasi:

```
add  maqsad , manba
```

Ayirish amali qoidasi:

```
sub  maqsad , manba
```

`ADD` amali `maqsad` ga `manba` ni qo'shib, natijani yana `maqsad` ga yozadi, ya'ni:

$$maqsaq = maqsad + manba$$

`SUB` amali `maqsad` dan `manba` ni ayirib, natijani yana `maqsad` ga yozadi, ya'ni:

$$maqsaq = maqsad - manba$$

`ADD` va `SUB` amallariga `MOV` amali haqida aytilgan barcha cheklovlar taalluqlidir. `maqsad` va `manba` qanday obyektlar bo'lishi mumkinligini bilish uchun 7-jadvalga qarang.

Misollar:

```
mov  eax , 10                          ;; EAX ← 10
add  eax , 5                           ;; EAX ← EAX+5, EAX = 15
sub  eax , 7                           ;; EAX = 8
mov  dword[a] , 17                     ;; a ← 17
add  eax , [a]                         ;; EAX = 25
sub  dword[a] , 7                      ;; a = 10
mov  dword[a] , __float32__(0x45.45)   ;; a ← 0x45.45
```

5.6. Assemblerda o'quv/yozuv

Barcha yuqori darajali dasturlash tillari dastur natijalarini ekranga chiqarish yoki tugmachalar taxtasidan kiritiladigan qiymatlarni o'qish uchun tayyor qism dasturlarga ega. Masalan, C da `printf` va `scanf`, Paskalda `write` va `read`, Rubyda `gets` va `puts` va hokazo. Ammo assembler tilida hech qanday o'quv/yozuv uchun mo'ljallangan tayyor qism dasturlar yo'q. Har

bir dasturchi o'ziga kerakligini o'zi yaratib olishiga to'g'ri keladi. Bunday qism dasturlar assemblerda aynan qanday tuzilishi «Qismli dasturlash» va «Fayllar bilan ishlash» boblarida to'liq yoritilgan. Lekin o'sha boblargacha foydalanib turish uchun kitobda C dasturlash tilining mezoniy kutubxonasidagi `printf` va `scanf` o'quv/yozuv qism dasturlaridan foydalaniladi. Dasturlashda *kutubxona* deganda qism dasturlar to'plami tushuniladi. Kutubxonada qism dasturlarning bajaruvchi kodi joylashgan bo'ladi va kutubxona fayl ko'rinishida bo'ladi. Ammo kutubxona bajaruvchi fayl hisoblanmaydi. Kutubxonaga misol qilib Linuxdagi *libc6.so, libstdc++.a* yoki Windowsdagi *kernel32.dll, libcmt.lib* fayllarini keltirish mumkin.

Boshqa dasturlash tilidagi imkonitaylardan Assemblerda foydalanish bir oz qiyinroq. Shuning uchun `printf` va `scanf` qism dasturlaridan foydalanishni osonlashtiradigan tayyor `chop_et` va `qabul_qil` makro vositalar kitob muallifi tomonidan tuzilgan va kitobdagi barcha dasturlarda o'shalar ishlatiladi[1].

Quyida makrolardan dasturingizda qanday foydalanishingiz ko'rsatilgan.

---

chop_et O'ziga berilgan qiymatlarni andoza bo'yicha ekranga chiqaradigan makro.

Qoidasi:

```
chop_et   `andoza`, qiymat_1, qiymat_2, … , qiymat_n
```

Bu yerda `andoza` qatorli o'zgarmas bo'lib, undagi harflar ketma-ket ekranga chiqariladi. Masalan, `Salom` so'zi quyidagicha chop etiladi:

```
chop_et   `Salom`
```

`chop_et` yordamida nafaqat qatorli o'zgarmaslarni, balki o'zgaruvchilarda saqlanayotgan istalgan turdagi qiymatlarni va o'zgarmaslarni chop etish mumkin. Masalan, butun sonlarni, o'nli kasrlarni va hokazo. Buning uchun andoza tarkibida `%X` belgisi ishlatiladi. `%X` belgisi qiymatni qay andozada chop etilishini aniqlaydi. `%X` dagi `X` o'rnida quyidagi harflar bo'lishi mumkin:

- **i, d** – (**i**nteger, **d**ecimal) ishorali o'nlik son sifatida chop etiladi.
- **o** – (**o**ctal) ishorasiz sakkizlik son sifatida chop etiladi.
- **u** – (**u**nsigned) ishorasiz o'nlik son sifatida chop etiladi.
- **x,X** – (he**x**adecimal) ishorasiz o'n oltilik son sifatida chop etiladi.
- **f, g, e** – (**f**loat, **e**xtended) ishorali o'nli kasr son sifatida chop etiladi.
- **s** – (**s**tring) qatorli qiymat sifatida chop etiladi.
- **c** – (**c**har) bitta belgi chop etiladi.

Demak, `%X` ning o'zi chop etilmay, u aniqlab kelayotgan andozada qiymatlar chop etiladi. Andozada birinchi uchragan `%X` uchun `andoza` dan keyingi birinchi qiymat, ikkinchi uchragan `%X` uchun esa `andoza` dan keyingi ikkinchi qiymat chop etiladi.

Andozadan keyin vergul bilan berilgan qiymatlar o'zgarmaslar, registrlar yoki xotiradagi o'zgaruvchilar bo'lishi mumkin. Lekin bu yerda ba'zi cheklamalar bor. Masalan, nishon nomi burchakli qavs ichida berilishi kerak: `[a]`. Shunday qilinmasa o'zgaruvchi qiymati emas, balki uning manzili ekranda paydo bo'ladi.

---

[1] Makrolar uchun «Makro vositalar» bobiga qarang.

Bundan tashqari agar qiymat butun son sifatida chop etilayotgan bo'lsa, u 4 baytli bo'lishi kerak. Masalan, EAX=`1234o`, EBX=4, EDX=`0xff7` bo'lsin, unda:

```
chop_et  `EAX=%o, EBX=%i, EDX=%X.`, eax, ebx, edx
```

Dasturni ishga tushirganingizda ekranda quyidagi narsa paydo bo'ladi:

```
EAX=1234, EBX=4, EDX=FF7.
```

Andoza ichida `%X` dan tashqari `\n` va `\t` kabi maxsus ma'noli belgilardan ham foydalanish mumkin. Bu yerda `\n` yangi satrga o'tishni, `\t` esa satr boshini anglatadi. Masalan:

```
chop_et   `Salom,\n\tMen keldim!\nXayr.\n`
```

quyidagicha chop etiladi:

```
Salom,
     Men keldim!
Xayr.
```

Qatorli o'zgaruvchilar bilan masala boshqacha bo'ladi. Agar siz `%s` orqali o'zgaruvchidagi qiymatni chop etmoqchi bo'lsangiz, `andoza` dan so'ng o'zgaruvchining manzilini ko'rsatishingiz kerak bo'ladi. Demak, o'zgaruvchi nomi burchakli qavslarsiz beriladi. Masalan, `abs` o'zgaruvchisi quyidagicha e'lon qilingan bo'lsin:

```
abs  db   "Ahillik - bu yaxshi",0
```

Undagi qiymatni `chop_et` orqali chop etishga harakat qilib ko'ramiz:

```
chop_et   `abs=%s`, abs
```

Yoki avval `abs` manzilini biror bir registrga yuklagan holda `chop_et` ga o'sha registrni berish mumkin:

```
mov  eax , abs
chop_et  `abs = EAX = "%s"`, eax
```

Natija:

```
abs = EAX = "Ahillik - bu yaxshi"
```

`%s` orqali chop etiladigan qatorli qiymatlar **nol bilan tugashi shart!**

---

qabul_qil Tugmachalar taxtasidan o'quvni amalga oshirib, o'qilgan qiymatlarni unga berilgan o'zgaruvchilarga yuklaydigan makro.

Qoidasi:

```
qabul_qil   `andoza`, o'zgaruvchi_1, o'zgaruvchi_2, …
```

Yuqorida aytilgan barcha eslatmalar `qabul_qil` uchun ham taalluqlidir. Faqat ushbu makroga o'zgaruvchi qiymati emas, balki manzili beriladi. Demak, nishon

nomi burchakli qavslarsiz beriladi. Masalan:

```
qabul_qil  `%i %i`, x, y
```

Ikkita son o'qiladi va ular berilgan tartibda `x` va `y` larga mos ravishda yoziladi. Ushbu makroga registr ham berish mumkin. Bunday holda registr qiymati xotira manzili sifatida qabul qilinadi, ya'ni registrga allaqachon nishon manzili yuklangan bo'lishi kerak. Masalan:

```
mov  ebx , abs
qabul_qil  `%s`, ebx
```

`qabul_qil` makrosi `%s` andozasi berilganda belgilar ketma-ketligini o'qiydi, ya'ni qatorli qiymatli qabul qilinadi. Qator to boshliq yoki satr boshi yoki tasdiq tugmalari bosilgunga qadar o'qiladi.

Ushbu makro vositalar *nasm-io.inc* faylida joylashgan bo'lib, ulardan foydalanish uchun ushbu fayl `%INCLUDE` maxsus makro direktivasi yordamida dastur boshida quyidagicha qo'shib olinadi:

```
%include  "nasm-io.inc"
```

nasm-io.inc fayli kitob oxiridagi «D» ilovasida berilgan. Bundan tashqari ushbu kitobga bag'ishlangan assembler.zn.uz saytidan ham *nasm-io.inc* fayli yozib olinishi mumkin. Ushbu fayl dasturingiz joylashgan direktoriyada bo'lishi shart!

Eslatma: Dasturlashda *nasm-io.inc* kabi fayllar *boshlang'ich fayl* deb ataladi va bunday fayllarda asosan makro vositalar saqlanadi.

5.7. Birinchi dastur

Shu paytgacha o'rgangan bilimlarimiz endilikda assemblerda birinchi oddiy dasturini tuzishga yetadi. Shundan kelib chiqqan holda ikkita sonni yig'indisini hisoblaydigan amaliy dastur tuzishga harakat qilib ko'ramiz. Ikkita son foydalanuvchi tomonidan kiritiladi. Mazkur dasturda `chop_et` va `qabul_qil` makrolaridan foydalaniladi va dasturni to'g'ridan-to'g'ri ishga tushira olish uchun *nasm-io.inc* fayli kerak bo'ladi.

1 - namuna: Yig'indini top.

```
(1)  ;; Maqsad:
(2)  ;;   Ushbu dastur beriladigan ikkita sonni qabul
(3)  ;;   qilib, ularning yig'indisini hisoblab, natijani
(4)  ;;   chop etadi.
(5)  ;;
(6)  ;; O'zgaruvchilar:
(7)  ;;   a va b - ikkita qo'shiluvchi, EAX - yig'indi.
(8)
(9)  %include "nasm-io.inc"
(10)
(11) section   .bss
(12) a    resd 1
(13) b    resd 1
(14)
(15) section   .data
```

```
(16) xayr      db   "Yana uchrashguncha, xayr.",0
(17)
(18) section   .text
(19) global    _main
(20)
(21) _main:
(22)      chop_et   `Salom, bugun havo yaxshi! \n`
(23)      chop_et   `Iltimos, ikkita son kiriting: `
(24)      qabul_qil `%i %i`, a, b
(25)      mov  eax , [a]
(26)      add  eax , [b]
(27)      chop_et   `Natija: %i + %i = %i \n`, [a], [b], eax
(28)      chop_et   `%s \n`, xayr
(29)
(30) ret
```

Yuqorida ko'rib chiqilganidek dasturimiz uch bo'limdan iborat: `.bss`, `.data` va `.text`. 9-qatorda `%INCLUDE` orqali boshlang'ich *nasm-io.inc* faylini dasturimizga «qo'shamiz». U yerda `chop_et` va `qabul_qil` makrolari joylashgan. 11-qatordan `.bss`, ya'ni boshlang'ich qiymatisiz e'lon qilinadigan o'zgaruvchilar bo'limi boshlanadi. 16-qatorda esa boshlang'ich qiymatga ega bo'lgan o'zgaruvchi sifatida qatorli o'zgaruvchi e'lon qilingan. U dastur oxirida xayrlashuv qatorini chop etishda qo'llaniladi. 19-qatordagi `global` kalit so'zi undan keyin keladigan nishon ommaviy ekanligini va unga tashqaridan turib murojaat qilish mumkin ekanligini bildiradi. Dastur ishga tushganda operatsion tizim dasturning boshlanish (kirish) nuqtasini axtara boshlaydi va uni topgach dastur buyruqlarini shu nuqtadan boshlab protsessorga bajarish uchun bera boshlaydi (bu nuqtadan oldin yozilgan har qanqay buyruq bajarilmaydi). Boshlanish nuqtasi nishon orqali belgilanadi va `_main` nishoni dasturning boshlanish nuqtasi hisoblanadi.

> **Eslatma**: Aslini olganda dastur boshlanish nuqtasi `_main` emas, balki `_start` nishoni hisoblanadi. Ammo, ushbu kitobda ba'zi murakkabliklarni chetlab o'tish maqsadida `_main` nishonidan foydalaniladi va u dasturning kirish nuqtasi sifatida qabul qilinadi. `_main` nishoniga ega dastur *asosiy dastur* deyiladi.

Ushbu nishonga `global` kalit so'zi yordamida ommaviy va tashqi tus beriladi. *Tashqi* deganda boshqa tashqi dasturlarning ushbu nishonga murojaat etishi mumkinligi nazarda tutiladi. Dasturning boshlanish nuqtasiga esa operatsion tizim murojaat eta olishi kerak. `_main` nishoni asosiy dasturning `.text` bo'limi boshida e'lon qilinishi shart!

Agar siz Linuxda dasturlayotgan bo'lsangiz `_main` boshidagi pastki chiziq (_) belgisini tashlab keting:

```
(31) global    main
(32)
(33) main:
```

Dasturimiz birinchi bo'lib ikkita son kiritishni so'raydi. 24-qatorga kelib dasturimiz bajarilishdan to'xtab, to foydalanuvchi ikkita son kiritmaguncha kutub turaveradi, chunki `qabul_qil` makrosi tugmachalar taxtasidan qiymatlarni o'qishga moslashtirilgan. 30-qatordagi `ret` buyrug'i dasturni yakunlaydi va boshqaruvni operatsion tizimga topshiradi.

5.8. Dasturni kompilyator orqali yig'ish

Dasturni NASM kompilyatori orqali yig'ish uchun buyruqlar qatoriga quyidagi yoziladi[1]:

```
$ nasm -f obyekt-fayl-andozasi dastur_fayli_nomi.asm
```

Assembler dasturi joylashgan faylning kengaytmasi *asm* bo'lishi kerak. Dasturni yig'ish natijasida *obyekt fayl* vujudga keladi, ammo bu fayl natijaviy bajarilivchi fayl hisoblanmaydi, ya'ni faylni hali ishga tushira olmaymiz. Obyekt faylda mashina tiliga o'girilgan kod bo'lib, bu kod haqiqiy bajarilivchi kod bo'lishi uchun unga yana ba'zi «narsalar» ulanishi kerak bo'ladi. Assembler obyekt fayldagi kodni maxsus andoza bo'yicha yaratiladi. Bunday obyekt fayl andozalari ishlatilayotgan kompilyator va operatsion tizim turiga qarab har xil bo'ladi. NASM hammabop kompilyator bo'lgani uchun andozalarning ko'plab turlari bilan ishlay oladi va unga aynan qaysi andoza bo'yicha obyekt fayl yaratish kerakligi, uning `-f` kaliti bilan beriladi. Quyida eng keng qo'llaniladigan obyekt fayl andozalari keltirilgan.

bin	Eng oddiy andoza bo'lib, dastur qanday yozilgan bo'lsa, shundayligicha obyekt kodga o'giriladi (unga qo'shimcha hech narsa qo'shilmaydi). Asosan MS-DOS tizimida foydalaniladi. `bin` obyekt fayllari kengaytmasiz bo'ladi.
obj	Asosan MASM va TASM assembler kompilyatorlari ishlatadigan Microsoftning OMF andozasi bo'lib, `.EXE` kengaytmali bajaruvchi fayllarni yaratishda foydalaniladi. `obj` obyekt fayllari *obj* kengaytmali bo'ladi.
win32, **win64**	Microsoftning 32 bitli va 64 bitli dasturlariga mo'ljallangan andozalar bo'lib, hozirda Visual C++ ulovchilari tomonidan keng qo'llaniladi. `win32` yoki `win64` obyekt fayllari *obj* kengaytmali bo'ladi.
coff	DJGPP ulovchisi ishlatadigan andoza. `coff` (**C**ommon **O**bject **F**ile **F**ormat – Umumiy Obyekt Fayl Andozasi) obyekt fayllari *o* kengaytmali bo'ladi.
elf	Linux, shuningdek, Unix System V turkumiga mansub Solaris x86, UnixWare va SCO Unix tizimlarida juda keng qo'llaniladigan andoza turi. `elf` (**E**xecutable and **L**inkable **F**ormat – Bajariladigan va Ulanadigan Andoza) obyekt fayllari *o* kengaytmali bo'ladi.

Yuqoridagi andoza turlari NASMning `-f` kalitiga qiymat sifatida beriladi.
Linux tizimida yuqorida ko'rib chiqilgan birinchi dastur quyidagicha yig'iladi:

```
$ nasm -f elf birinchi_dastur.asm
```

Shundan so'ng joriy direktoriyada *birinchi_dastur.o* ismli obyekt fayl paydo bo'ladi. Windowsda esa asosan `win32` andozasi qo'llaniladi:

```
> nasm -f win32 birinchi_dastur.asm
```

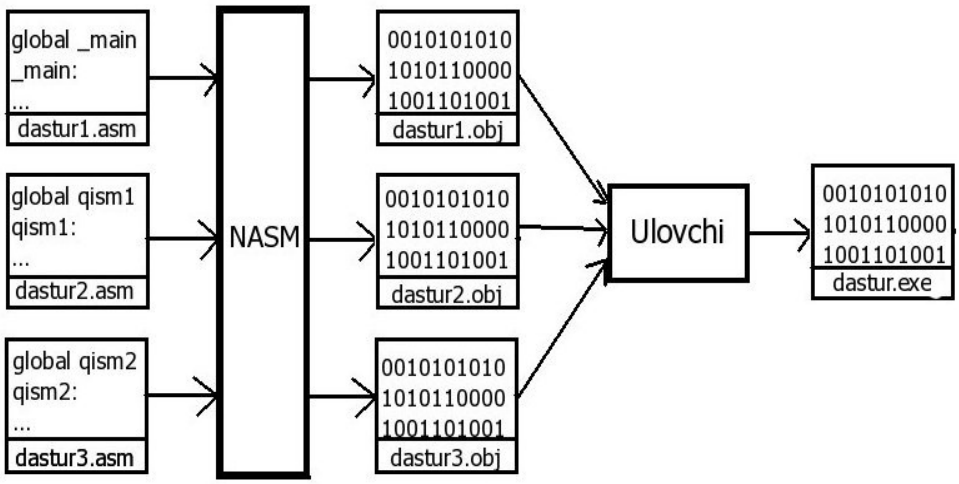
[1] NASM kompilyatorini operatsion tizimda qanday o'rnatish «NASMni o'rnarish» ilovasida keltirilgan. Buyruqlar qatorida ishlash yuzasidan savollar bo'lsa «Buyruqlar qatorida ishlash» ilovasiga qarang.

Shundan so'ng joriy direktoriyada *birinchi_dastur.obj* ismli obyekt fayl paydo bo'ladi. Obyekt fayldan to'laqonli bajaruvchi kodga ega bo'lish uchun yana bir bosqich mavjud va u keyingi mavzuda yoritiladi.

Dastur yig'ilayotgan chog'da `warning` yoki `error` kabi xabarlar buyruqlar qatorida paydo bo'lib qolishi mumkin. Kompilyator `warning` orqali dasturda shubhali ifoda borligi va u keyinchalik kutilmagan natijaga olib kelishi mumkinligi haqida ogohlantiradi. Ammo bu holatda kompilyator o'z ishini to'xtatmaydi va obyekt faylni yaratadi. `error` esa dastur tuzilishida xato borligini bildiradi va bu holatda kompilyator o'z ishini to'xtatadi. Hech qanday obyekt fayl ham yaratilmaydi. Bu vaziyatda xato dasturning nechanchi qatorida yuz bergani e'lon qilinadi va dasturchi xatoni tuzatib, kompilyatorni qayta ishga tushirishi kerak.

5.9. Obyekt fayllarni ulash

Obyekt fayllarni ulash dasturni mashina tiliga o'girishning oxirgi bosqichi bo'lib, bunday vazifani bajaradigan dastur *ulovchi* deyiladi. Ulovchi dasturining vazifasi bir yoki bir necha obyekt fayllarni bitta bajaruvchi faylga ulashdan iborat. Bu jarayon 17-rasmda tasvirlangan.



17-rasm.

Ulovchiga beriladigan obyekt fayllarning albatta bittasi dasturning boshlanish nuqtasiga ega bo'lishi kerak (agar dasturiy kutubxona yaratmayotgan bo'lsangiz, albatta). Ulovchi bunday nuqtani uchratmasa yoki ikki marta uchratsa xato yuz beradi. Dasturning boshlanish nuqtasini `main` nishoni bo'lib, `elf` andozasidan tashqari barcha obyekt fayl andozalarida ushbu nishon pastki chiziq (_) belgisi bilan boshlanadi. Demak, dasturchi qaysi obyekt fayl andozasidan foydalanayotganiga qarab doim dastur boshlanish nuqtasini moslab borishi kerak. Ushbu kitob maxsus bir obyekt fayl andozasiga emas, balki barcha operatsion tizimlarga mo'ljallangani uchun keyingi mavzulardan boshlab dastur kirish nuqtasini berish umumiy usulni qo'llagan holda amalga oshiriladi. Gap shundaki, *nasm-io.inc* boshlang'ich faylida kitob muallifi tomonidan `tizim_global` makrosi berilgan. Ushbu makro juda qulay bo'lib, uni oddiy assemblerdagi `global` kalit so'zining o'rnida ishlatish orqali obyekt fayl andozasi haqida unutish mumkin. `tizim_global` makrosi unga berilgan nishonni o'zi obyekt fayl andozasiga moslaydi. Demak, dastur kirish nuqtasi `tizim_global` makrosi yordamida operatsion tizim yoki obyekt fayl turidan qat'i nazar quyidagicha e'lon qilinadi:

```
tizim_global main
main:
```

Agar sizni ushbu makro qanday tuzilgani qiziqtirsa, «Makro vositalar» bobiga qarang.

Mazkur kitobdagi deyarli barcha amaliy dasturlar C mezoniy kutubxonasidagi `printf`, `scanf` kabi qism dasturlaridan foydalanadi. Demak, NASM yaratgan obyekt fayllarga C mezoniy kutubxonasi ham ulanishi kerak. Ammo buni oddiy ulovchi dasturi bilan amalga oshirish bir oz murakkabroq. Shuning uchun ushbu kitobda obyekt fayllarni ulash C kompilyatori yordamida bajariladi. Assembler kompilyatorlaridan farqli o'laroq barcha C kompilyatorlari o'z ulovchilariga ega va ular ortiqcha mehnat talab qilmasdan ulashni o'z-o'zidan bajarishadi.

Juda ko'p C kompilyatorlari mavjud bo'lib, shulardan eng mashhuri GCC (**G**NU **C C**ompiler) hisoblanadi. Ushbu mahsulot GNU loyihasiga tegishli bo'lib, barcha UNIX oilasiga mansub tizimlardan tortib iPhone telefonlaridagi dasturlargacha hammasi GCC orqali yig'iladi. Undan tashqari GCC erkin tarqatiladigan dastur bo'lib, o'quv muassasalariga juda mos keladi. Shularni hisobga olgan holda obyekt fayllarni ulash GCC misolida keltiriladi. Ammo ulovchi yoki C kompilyatorini tanlash o'quvchi ixtiyorida. «C kompilyatorlari va obyekt fayllarni ular orqali ulash» ilovasida GCC kompilyatorini o'rnatish va undan tashqari Microsoft Visual Studio kabi boshqa kompilyatorlardan foydalanishga molik tashkiliy masalalar yoritiladi.

Shunday qilib yuqorida NASM tomonidan yaratilgan obyekt fayl Linuxda quyidagicha ulanadi:

```
$ gcc birinchi_dastur.o –o natija
```

Demak, bajaruvchi kod *natija* nomli yangi faylga joylashtiriladi. Ushbu fayl joriy direktoriyada yaratiladi va u quyidagicha ishga tushiriladi:

```
$ ./natija
```

Bu yerdagi nuqtqa (.) belgisi joriy direktoriyani anglatadi. Agar hammasi joyida bo'lsa, ekranda quyidagilarni korishimiz mumkin:

```
$ ./natija
Salom, bugun havo yaxshi
Iltimos, ikkita son kiriting :
```

Shundan so'ng ikkita son kiritilishi kutiladi va natija ekranga chiqariladi.

Windowsda esa yuqoridagi amallar quyidagicha bajariladi:

```
> gcc birinchi_dastur.obj –o natija.exe
```

Endi *natija.exe* bajaruvchi faylini ishga tushiramiz:

```
> natija.exe
Salom, bugun havo yaxshi
Iltimos, ikkita son kiriting :
```

5.10. Buyruqlar qatorida NASM buyrug'i kalitlari

Nasm kompilyatori ishlash tartibini unga qo'shimcha tarzda kalitlar berib o'zgartirish mumkin. Shu sababdan kalitlarning eng keraklilarini ko'rib chiqamiz.

NASM buyrug'i kalitlari:

–f ANDOZA	Bu kalitdan keyin siz NASM yig'uvchisi uchun o'zingiz xohlagan obyekt fayl andozasini berishingiz mumkin. **ANDOZA** o'rnida quyidagilar bo'lishi mumkin: `coff, obj, win32, ...`

	Misol: `$ nasm -f coff dastur.asm`
-o OBYEKT FAYL NOMI	Ushbu kalitdan so'ng obyekt fayl uchun nom ko'rsatiladi. Bu kalitsiz NASM obyekt faylga o'zi nom beradi, ya'ni berilgan dastur fayli nomi bilan `-f` kalitiga qarab mos kengaytmali obyekt fayl yaratadi. Masalalan, `dastur.asm` faylidan `dastur.o` ismli obyekt fayl yaratiladi. Agar obyekt fayl nomi, masalalan, `natija.o` bo'lishini istasangiz quyidagi buyruq orqali dasturni yig'asiz: `$ nasm -f obj dastur.asm -o natija.obj`
-a	Preprotsessor ishga tushirilmagan holda dastur yig'iladi («Makro vositalar» bobiga qarang). NASMga bu kalit berilganda hech qaysi makro ishlamaydi. Agar biz birinchi dasturimizni ushbu kalitni qo'llagan holda yig'sak, `chop_et`, `qabul_qil` makro vositalar ishlamaydi.
-e	Dasturni faqat preprotsessordan o'tkazish va natijani ekranga chiqarish (hech qanday dasturni yig'ish amalga oshirilmaydi va obyekt fayl yaratilmaydi). Agar birinchi dasturimizni NASMga shu kalit bilan bersak, ekranda o'zimiz tuzgan dasturni ko'ramiz. Faqat makro vositalar o'rnida preprotsessor ularni nimalarga almashtirganini ko'rishimiz mumkin.
-E FAYL NOMI	Yig'ish chog'ida kelib chiqadigan xatolar ro'yxati ekranga emas, balki berilgan faylga yoziladi. Bu kalit yuzaga kelgan xatolarni keyinchalik ham tahlil qilishda qo'l kelishi mumkin.
-i,-I DIREKTORIYA NOMI	Boshlang'ich fayllarni axtarish uchun qo'shimcha direktoriya ko'rsatish. Odatda `%INCLUDE` makrosiga berilgan boshlang'ich fayllar NASM buyrug'i ishga tushirilgan direktoriyadan qidiriladi. Ushbu kalit orqali esa NASMning boshlang'ich fayllarni axtaradigan direktoriyalariga boshqa direktoriyalarni qo'shish mumkin. Masalan, `nasm-io.inc` fayli dastur joylashgan direktoriyada emas, balki shu direktoriya ichidagi `etc` direktoriyasida bo'lsin, u holda: `$ nasm -f elf birinchi_dastur.asm -I etc/` Direktoriya nomidan so'ng (`/`) ni qo'yishni unutmang! Chunki faylning to'liq yo'lini aniqlashda NASM shunchaki direktoriya nomiga boshlang'ich fayl nomini qo'shadi.

Eslatma: Albatta bu kalitsiz ham `%INCLUDE` makrosiga boshlang'ich faylning to'liq yo'lini ko'rsatib qo'yish mumkin. Ammo bu holda har safar boshlang'ich faylni boshqa yoqqa ko'chirganingizda kompilyatsiyadan oldin dastur kodiga o'zgartirish kiritishga to'g'ri keladi.

`-p BOSHLANG'ICH FAYL NOMI`

Dasturga ko'rsatilgan boshlang'ich faylni qo'shadi. Bu orqali dasturga `%INCLUDE` makrosisiz boshlang'ich fayllarni qo'shish mumkin, shunchaki `-p` kaliti orqali fayllar NASMga beriladi. Masalan, dasturimizdagi birinchi qatorni o'chirib tashlangda, yig'ish jarayonida quyidagini yozing:

```
$ nasm -f elf dastur.asm -p nasm-io.inc
```

`-d MAKRO[=qiymat]`

Bir qatorli makroni kompilyatsiya jarayonida aniqlaydi. Bu kalitdan dasturni shartli yig'ishda foydalaniladi («Makro vositalar» bobiga qarang). Bu yerda to'rt burchak qavs uni ichidagi narsa bo'lishi ham, bo'lmasligi ham mumkinligini bildiradi.

`-l TAHLIL FAYL NOMI`

Dasturni yig'ish natijasida vujudga keladigan mashina tilidagi kodni maxsus ko'rinishda berilgan faylga yozib beradi. Mashina tilidagi kod va manzillar inson tushunishi qiyin bo'lgan ikkilik sanoq tizimida emas, balki ixchamroq bo'lgan o'n oltilik sanoq tizimida beriladi. Faylda siz tuzgan assembler dasturi va unga qo'shilgan boshlang'ich fayldagi dastur o'ng tomonda, unga to'g'ri keladigan mashina kodi esa chap tomonda joylashtiriladi. Bu juda ham foydali kalit bo'lib, tuzgan dasturingizni mashina tilida qanday bo'lishini ko'rishga imkon beradi. Masalan, qiziqish uchun birinchi dasturimizni sinab ko'rishimiz mumkin:

```
$ nasm -f elf dastur.asm -l tahlil.txt
```

Endi *tahlil.txt* faylini biror bir matn muharriri orqali ochib ko'ramiz. Qiziquvchilar uchun dasturning ba'zi qatorlarini tahlil qilamiz.

Birinchi dasturning 11–16 qatorlari:

```
112                                 section     .bss
113 00000000 <res 00000004>         a resd 1
114 00000004 <res 00000004>         b resd 1
115
116                                 section .data
117 00000000 59616E612075636872- xayr db "Yana uchr-
118 00000009 61736867756E636861-          ashguncha-
119 00000012 2C20786179722E0              , xayr.",0
```

25–26 qatorlar:

```
215 00000094 A1[00000000]           mov    eax , [a]
216 00000099 0305[04000000]         add    eax , [b]
```

Birinchi ustunda tahlil faylidagi har bir satrning tartib raqami joylashgan va bu ustun sizda boshqacha bo'lishi mumkin. Ikkinchi ustunda dasturning har bir qatorining foydali manzillari 4 baytli qiymat

sifatida berilgan, ya'ni offsetlar (o'n oltilik sanoq tizimida). Uchinchi ustunda esa bevosita qiymatlar va buyruqlarning mashina tilidagi kodlari joylashgan. E'tibor bergan bo'lsangiz 11 va 15-qatorlardagi `section` kabi assembler tiliga xos direktivalar mashina tiliga o'girilmagan. Barcha bo'limlarning offset manzillari noldan boshlangan. `a` va `xayr` nishonlari bo'lim boshida joylashgani uchun ularning ham manzillari `0x00000000` ga teng. `xayr` ning 25 baytli boshlang'ich qiymati nishon oldida o'n oltilik sanoq tizimida berilgan. `a` o'zgaruvchisi 4 baytli bo'lgani sababli `b` ning manzili shunga qarab `0x00000004` dan boshlangan. (25) qatordagi `MOV` buyrug'i xotiradagi o'zgaruvchi qiymatini EAX ga yuklashni anglatadigan `0xA1` mashina kodiga o'girilgan va yonida xotira manzili burchakli qavs ichida berilgan. Ushbu buyruq qiymati bilan birga 5 bayt o'lchamga ega bo'lgani tufayli keyingi buyruq `0x00000099` manzilidan boshlangan: `0x95 + 0x5 = 0x99`.

Eslatma: Windowsda NASM kalitlariga beriladigan fayl nomi yoki mutlaq yo'llar tarkibida bo'shliq ( ) belgisi bo'lmasligi kerak. Masalan:

```
> nasm -f win32 dastur.asm -IC:\Program files
```

ishlamaydi.

VI bob
Assembler tili asoslari

Butun sonlar kompyuter xotirasida ikkilik ko'rinishida saqlanishi III bobda tushuntirib o'tilgan edi. Shu paytgacha bo'lgan mavsulardan siz ikkilik sanoq tizimi haqida yetarlicha ma'lumotga ega bo'ldingiz deb o'ylaymiz. Ushbu bobda biz ular ustuda bajariladigan asosiy arifmetik amallar kompyuterda aynan qanday yuz berishi ustida to'xtalamiz.

6.1. Ishorali va ishorasiz butun sonlar

Kompyuterda har bir qiymat o'zining qat'iy o'lchamiga ega. Dasturlashda ikkita bir baytli o'zgaruvchi qiymatlarini qo'shib, yana bir baytli natija olinadi, ya'ni arifmetik amallar bajarilayotganida hadlar o'lchami teng bo'lishi kerak. Bundan buyog'iga qiymat deganda ma'lum bir o'lchamda ega bo'lgan o'zgaruvchi ichidagi qiymat tushuniladi.

Butun sonlarni kompyuterga taqdim qilishda bir qancha muammolarga duch kelinadi. Masalan, biz uchun ishorali va isharasiz sonlarni farqlash juda oson bo'lishi mumkin. Kompyuter uchun esa son bu 1 va 0 lar ketma-ketligidan boshqa narsa emas. Shuning uchun ishorali va ishorasiz sonlarni xotirada saqlashning bir qancha usullari o'ylab topilgan. Asosiy g'oya shundaki, son ishorali deb qaralganda uning to'ng'ich biti qiymati ishora sifatida qabul qilinadi. Qolgan bitlar esa bevosita qiymatni beradi. Masalan, 7 sonini bir bayt ko'rinishida tasavvur qilaylik: $(\underline{0}0000111)_2$. Bu yerda to'ng'ich, ya'ni 7-bit sonning ishorasini bildiradi. 0 musbat ishorani bildirsa, 1 manfiy ishorani bildiradi. Shunga qarab -7 ni topish mumkin: $(\underline{1}0000111)_2$. Son ishorasiz qaralganda esa barcha bitlar qiymati sifatida qabul qilinadi. Sonni qanday qabul qilish dasturchiga bog'liq. Ta'kidlash kerakki, kompyuter uchun buning hech qanday farqi yo'q. Protsessor ishorali yoki ishorasiz sonlarni farqlamaydi va ular ustida amallarni birdek bajaradi. Bunda asosiysi natija ham to'g'ri hisoblanadi. Bunday ajoyib hisob usulini amalga oshirish uchun bir qancha murakkabliklarni yechishga to'g'ri kelgan.

Murakkablikning bir ko'rinishi qilib nol sonini olaylik. Ko'rib o'tgan usulimiz bo'yicha nol sonining ikki xil shakli mavjud bo'lib qolayapti: +0 $(\underline{0}0000000)_2$ va -0 $(\underline{1}0000000)_2$, vaholanki, nolning ishorasi bo'lmaydi. Shunga o'xshash yana bir qancha o'ng'aysizliklar bo'lib, ular markaziy protsessor ishini chigallashtirishi mumkin.

Sonni xotirada saqlashning ikkinchi usuli birlamchi to'ldiruvchi (one's complement) deb nomlanadi. Bu usulga ko'ra sonning qarama-qarshi ishoralisi o'sha sonning barcha bitlari qarama-qarshisiga almashtirish orqali qo'lga kiritiladi. Masalan 7 $(00000111)_2$ sonining birlamchi to'ldiruvchisi 11111000 ko'rinishga ega bo'ladi. Demak, -7 xotirada 11111000 ko'rinishda saqlanadi. Ammo bu usulda ham nol ikkita ko'rinishga ega (00000000) va (11111111).

Uchinchi usul barcha hozirgi zamonaviy komyuterlarda qo'llaniladigan ikkilamchi to'ldiruvchi(two's complement) usuli bo'lib, uni hisoblash ikki bosqichga ega:

1) Avval sonning birlamchi to'ldiruvchisi topiladi,
2) Keyin shu natijaga 1 qo'shiladi.

Masalan, +7 $(00000111)_2$ uchun bu quyidagicha bo'ladi:

1) $\underline{0}0000111 \rightarrow 11111000$

2) 11111000 → birlamchi to'ldiruvchi.

```
 +_______1
  11111001  =  -7
```

Ishora bitining ham o'z-o'zidan qarama-qarshisiga o'zgarayotganini sezayotgandirsiz. Bu usulda nol faqat bitta ko'rinishga ega, ya'ni 0 $(00000000)_2$ uchun:

```
   11111111  →  birlamchi to'ldiruvchi
 +_______1
 1 00000000  =  0
```

Yuzaga kelgan bir-yodda qiymat natijaga yozilmaydi. Chunki, qo'shish bir baytli qiymatlar ustida amalga oshirilayapti va natija ham bir bayt bo'lishi kerak. To'ng'ich bitda yuzaga kelgan bir-yodda uchun esa 8-bit kerak bo'ladi, bu esa bir bayt chegarasidan chiqib ketadi. Bu holat CF bayrog'ining o'rnatilishiga sabab bo'ladi.

Yanada tushunarliroq bo'lishi uchun keling -1 sonini bir baytli ko'rinishini aniqlaylik. Buning uchun 1 $(00000001)_2$ sonining ikkilamchi to'ldiruvchisini topish kifoya:

```
   11111110 → birlamchi to'ldiruvchi
 +_______1
  11111111  = - 1
```

Demak, o'n oltilik ko'rinishdagi `0xFF` son -1 ni berar ekan.

Ishorasiz sonlarni taqdim etish hech qanday o'zgarishni taqozo etmaydi. Ularda barcha bitlar qiymat sifatida qabul qilinadi. Masalan, xotiradagi -1 ni beruvchi 11111111 son ishorasiz deb qaralganda 255 ni, ya'ni bir baytga sig'adigan eng katta sonni beradi. 9-jadvalda har xil o'lchamlarda ishorali va ishorasiz sonlarning sig'ish chegaralari aks ettirilgan.

9-jadval

O'lchamlar	Ishorasiz	Ishorali
1 bayt	0…255	-128…+127
2 bayt	0…65535	-32768…+32767
4 bayt	0…4294967295	-2147483648…+2147283647

10-jadvalda ba'zi sonlarning ishorasiz ko'rinishi o'n oltilik sanoq tizimida keltirilgan.

10-jadval

O'nlik	O'n oltilik
0	00
1	01
127	7F
-128	80
-127	81
-2	FE
-1	FF

6.2. Butun sonlar ustida arifmetik amallar

Ushbu bo'limda qo'shish, ayirish, bo'lish va ko'paytirish amallari assemblerda qanday amalga oshirilishini ko'rib chiqamiz. Yuqorida ta'kidlanganidek protsessor uchun qiymat ishorasining ahamiyati yo'q. Lekin dasturlash davomida siz sonlarga bemalol ishorali yoki ishorasiz qiymat sifatida muomala qilishingiz mumkin. Assembler dasturchi qiymatga nisbatan ishlatayotgan buyruqlariga qarab qiymatning ishorali yoki ishorasiz ekanligini bilib oladi. Aslida qiymatga nisbatan birinchi ishlatilgan buyruq o'sha qiymat ishorali yoki ishorasiz deb qaralishini aniqlab beradi. Lekin hamma buyruq ham ishorali yoki ishorasiz sonlar uchun har xil bo'lavermaydi.

6.2.1. Qo'shish va ayirish

Ikki son qo'shilganida ularning ikkilik ko'rinishidagi nusxalari protsessor tomonidan tagma-tag usulda qo'shiladi.

Gap ishorasiz qiymatlar ustida ketganda natija bajarilayotgan amal o'lchamidan oshib ketmasligi haqida qayg'urish kerak. Masalan, bir baytli qiymatlar bilan ishlaganda natija 255 dan oshib ketmasligi kerak. 254 ga 5 ni bir bayt sifatiga qo'shib ko'ramiz:

```
        11111110 = 254
      +
        00000101 = 5
CF← |1| 00000011 = 3 ?
```

Bu yerda natija ham bir baytli bo'lishi kerak, ya'ni unga sakkizinchi bit sig'maydi va oshib ketgan bit CF bayrog'iga o'rnatiladi. Haqiqiy natija sifatida esa o'ngdan birinchi 8 bit olinadi. Albatta bu son noto'g'ri natija hisoblanadi, ya'ni 3 $(00000011)_2$ soni. Ishorasiz sonlar bilan ishlaganda dasturchi buyruqdan so'ng shunday ko'ngilsiz hollarni yuz berganini tekshirish uchun CF qiymatiga murojaat qilishi kerak. Masalan, quyida `ADD` amalidan so'ng CF o'rnatiladi:

```
mov  al , 74
add  al , 200
```

Asosan qo'shish amalini bajaradigan uchta buyruq bor. Ular:

1) `INC` (**Inc**rement) – o'zgaruvchining qiymatini bittaga oshiradi.

```
inc  maqsad
```

2) `ADD` (**Add**ition) – ikkita qiymatni oddiy qo'shish.

```
add  maqsad , manba
```

3) `ADC` (**Ad**d **C**arry) – CF ni hisobga olgan holda qo'shish.

```
adc  maqsad , manba
```

Natijaga CF ning qiymati ham maxsus tarzda qo'shiladi, ya'ni: $maqsad = maqsad + CF + manba$.

Oxirgi amal biroz tushunarsiz bo'lsa, bir misol keltiramiz. Deylik, siz ikkita 8 baytli katta qiymatlarni 4 baytli registrlar orqali qo'shmoqchisiz. Assemblerda yaxlit 8 baytli qiymatni saqlashda ikkita 4 baytli registrlardan foydalanish mumkin. Bunda qiymatga murojaat qilish uchun registr nomlari ketma-ket qo'yib ular o'rtasiga ikki nuqta qo'yiladi. EDX:EAX ni olsak, bu yerda sonning to'ng'ich 32 biti EDX da, kenja 32 biti esa EAX da saqlanadi. Qiymatlarimizni saqlashda EDX:EAX va EBX:ECX registrlaridan foydalansak, birinchi navbatda kenja 32 bitlar alohida qo'shiladi:

```
add  eax , ecx
```

Agar yig'indi EAX ga sig'masa, chapdan ortib qolgan bit CF ga o'rnatiladi, chunki yuqorida aytilganidek `ADD` amali CF ni qiymatini zaruriyat bo'lganda o'zgartiradi. Endigi qilinadigan ish to'ng'ich 32 bitni maktabda o'rgatilganidek bir yodda usuli bilan CF ni hisobga olgan holda qo'shishdan iboratdir:

```
adc  edx , ebx
```

Natija bundan so'ng albatta EDX:EAX da saqlanadi.

Endi ishorali sonlarni qoshishga to'xtalamiz. Sizga ma'lumki, qiymatga ishorali son sifatida qaralganda uning to'ng'ich biti ishora sifatida qaraladi. Masalan, 2 baytli son bilan ishlaganimizda natija 15-bitga ta'sir qilmasligini, ya'ni bir-yodda qiymat 15-bitga kelib qo'shilmasligini ta'minlashimiz kerak. Amal bajarilgandan so'ng buni OF (**O**verflow **F**lag) bayroqchasi yordamida tekshirish mumkin. EFLAGS registrining 11-bitini tashkil etuvchi OF faqat ishorali sonlar arifmetikasida ma'noga ega. Qolgan hollarda o'rnatilgani bilan u bizni qiziqtirmaydi. Fikrimizni asoslash uchun bir qancha misollarni ko'rib chiqamiz (bu yerda sonlar 2 baytli o'lchmga ega deb hisoblangan).

Birinchi holat:

```
0111010000110110 = 29750
+
0000000101110001 = 369
=
0111010110100111 = 30119
```

Ko'rinib turibdiki, natija to'g'ri va hech qanday bir-yodda natijaning ishora bitiga tahdid solmayapti. Demak, OF o'rnatilmagan.

Ikkinchi holat:

```
0111111111111111 = 32767
+
0000000000000001 = 1
=
1000000000000000 = -32768
```

Navbatdagi misolda 14-bitdan bir-yodda yig'indining ishora bitiga qo'shilib uni manfiy songa aylantirib qo'ymoqda, ya'ni natijaga ishorali son sifatida qaraganda u umuman noto'g'ri bo'lgan manfiy sonni beradi, -32768. Bu yerda to'ng'ich bitgacha bo'lgan qiymat bitlarida o'ziga xos toshish(sig'maslik) yuz berayapti. Bunday hollarda OF o'rnatiladi. Ammo uni ishorasiz qiymat sifatida qabul qilsak, ya'ni barcha 16 bitni qiymat sifatida olsak 32768 soni kelib chiqadi. Shundan ham ko'rinib turibdiki, ikkilamchi to'ldiruvchi usulida amallar qanday ishorali bo'lmasin natija to'g'ri chiqadi.

Uchinchi holat:

```
 1010011101100110 = -22682
 +
 1000000000000110 = -32762
 =
|1| 0010011101101100 = 10092
```

15-bitda bir-yodda kuzatildi. Natija ham noto'g'ri. Chunki ikki manfiy sonning yig'indisi musbat bo'lmaydi. Qiymat bitlarida hech qanday toshish ro'y bermagan bo'lsada, OF o'rnatiladi. Buning sababi natija ishorali sonlar arifmetikasi nuqtai nazaridan noto'g'ridir.

To'rtinchi holat:

```
 1110000011110000 = -7952
 +
 1110000000001111 = -8177
 =
|1| 1100000011111111 = -16129
```

14 va 15 bitlarda oshib ketish kuzatilgan bo'lsada natija to'g'ri. Demak, OF o'rnatilmagan.

Shunday qilib, quyidagi hollarda OF o'rnatilar ekan:

- agar qo'shiluvchilar musbat bo'lsa, ya'ni ishora biti nolga teng va yig'indining ishora bitida oshish kuzatilsa, bunday holda OF o'rnatiladi (ikkinchi holat);
- agar qo'shiluvchilar manfiy bo'lsa, ya'ni ishora biti birga teng va qiymat bitlaridan ishora bitiga oshish kuzatilmasa-yu faqat ishora bitida oshish kuzatilsa, u holda OF o'rnatiladi (uchinchi holat).

Ko'rinib turibdiki, agar to'ng'ich va undan bitta oldingi bitda bir-yodda yuz bermasa yoki ikkalasida ham bir vaqtning o'zida bir-yodda yuz bersa OF o'rnatilmas ekan (birinchi va to'rtinchi holat).

Ayirish amali bilan ham shunga o'xshash muammolar yuzaga kelishi mumkin. Ishorasiz sonlar bilan ishlaganda, masalan, kichik sondan katta sonni ayirib bo'lmaydi. Agar shunday holat talab qilinsa, protsessor to'ng'ich bitdan ham kattaroq bo'lgan g'oyibona o'rindan bir-qarz oladi va buni bildirish uchun CF bayrog'ini o'rnatadi. Masalan, ishorasiz arifmetikada 2 dan 8 ni ayirib bo'lmaydi. Buni bir baytli ko'rinishda protsessorda qanday amalga oshirilishini kuzatamiz:

```
CF → |1| 00000010 = 2
         -
         00001000 = 8
         =
         11111010 = 250
```

Javob ishorasiz son sifatida qaralganda albatta noto'g'ri, 250. Lekin ayirmaga ishorali son sifatida qaralganda -6 kelib chiqadi . Bu yerda CF bayrog'i o'rnatiladi, chunki kamayuvchining to'ng'ich biti uchun bir-qarz olindi va shuni evaziga natija ham noto'g'ri, ya'ni kamayuvchi 00000010 emas, balki 100000010 bo'ldi. Mazkur misolda ayirma nima sababdan aynan 11111010 ga teng bo'lgani tushunarsiz bo'lsa, ayirishni qo'shishga aylantirish mumkin, 2 + (-8):

```
 00000010
 +
 11111000
 =
 11111010
```

Ayirish amalini bajaradigan asosan uchta buyruq bor. Ular:

1) `DEC` (**Dec**rement) - o'zgaruvchining qiymatini bittaga kamaytiradi.

```
dec  maqsad
```

2) `SUB` (**Sub**traction) - oddiy ayirish amali

```
sub  maqsad , manba
```

3) `SBB` (Subtraction with carry) - CF bayroqchasi hisobga olgan holda ayirish. Natijadan CF ning qiymati ham maxsus tarzda ayiriladi, ya'ni: $maqsad = maqsda - CF - manba$.

```
sbb  maqsad , manba
```

Ishorali sonlar bilan ayirish amalini bajarganda esa bizni asosan OF bayrog'ining qiymati qiziqtiradi. Masalan, 112 - (-126).

```
01110000 = 112
-
10000010 = -126
=
11101110 = -18
```

Ko'rinib turibdiki, hadlarning ishora bitlari o'zaro ayirilib ayirmaning ishora bitiga ta'sir etdi. Bunday hollarda OF o'rnatiladi. -18 noto'g'ri natija bo'lsada, ammo 11101110 sonini ishorasiz deb qabul qilsak u 238 ni beradi.

Yana bir misolni ko'rib chiqamiz: -54 - 59 = -54 + (-59) = -113.

```
 11001010 = -54
 +
 11000101 = -59
 =
|1| 10001111 = -113
```

Natija to'g'ri va hech qanday OF o'rnatilmaydi.

Aslini olganda protsessor uchun aynan ayirish amali mavjud bo'lmasligi ham mumkin edi. Chunki ayirish uchun kamayuvchiga ayriluvchining ikkilamchi to'ldiruvchisini qo'shish kifoyadir.

Yana bir ishora bilan bog'liq bo'lgan buyruq `NEG` bo'lib, u o'zgaruvchining ishorasini qarama-qarshisiga o'zgartiradi. Masalan EAX da 44 bo'lsa:

```
neg  eax
```

dan so'ng EAX da -44 soni bo'ladi.

Xulosa qilib aytganda, protsessor har qanday holda ham to'ng'ich bitda oshish kuzatilganda CF ni o'rnatadi. Chunki protsessor ishorali yoki ishorasiz sonlarni farqiga yetmaydi. Barcha mas'uliyat dasturchi zimmasiga yuklanadi. U OF va CF qiymatlarni tahlil qilgan holda ish ko'rishi kerak. CF va OF larning tahlil usullari «Taqqoslash buyrug'i» mavzusida keltirilgan.

6.2.2. Ko'paytirish va bo'lish

Ko'paytirish va bo'lishda ham ishorasiz va ishorali sonlar farq qiladi. Ammo qo'shish va ayirishdan farqli ravishda bu yerda buyruqlar ishorali yoki ishorasiz qiymatlarga qarab har xil bo'ladi. Chunki ikkilamchi to'ldiruvchi usuli ko'paytirish yoki bo'lishga kelganda protsessorga ikkala holda ham bir xil ishlash imkonini bermaydi.

Shunday qilib, ishorasiz ko'paytirishda biz `MUL` buyrug'idan foydalanamiz.

Qoidasi:

```
mul manba
```

Bu yerda manba birinchi ko'paytuvchi sifatida keladi va uning o'rnida registr yoki xotira kelishi mumkin. Ikkinchi ko'paytuvchi esa yashirin keladi va uning vazifasini har doim EAX registri yoki uning qismlari bajaradi. Ko'paytma ham EAX registrida saqlanadi. MUL ga berilgan manbaning o'lchamiga qarab mos EAX qismi ko'paytiriladi. Agar manba bir baytli bo'lsa, u holda ikkinchi ko'paytuvchi sifatida AL qism registri qiymati olinadi. Natija esa 2 baytli AX ga yoziladi. Bir baytli ko'paytuvchilarning natijasi 2 baytli registrga yozilishining sababi ko'paytirishda boshqa amallarga nisbatan ancha katta son kelib chiqishidir. Agar natija bir baytga sig'sa u AL ning o'ziga yoziladi va AH ning qiymati nolga teng bo'lib qoladi, ya'ni natija = AX = AL, aks holda ko'paytma to'liqligicha AX ga joylashtirilgan bo'ladi, ya'ni natija = AX. Umuman olganda ikkala holda ham agar qo'shimcha bosh og'rig'iga ehtiyoj bo'lmasa, EAX yoki AX ni natija sifatida ishlataverish kerak. Natija AL dan oshganini CF va OF bayroqlari o'rnatilgan yoki o'rnatilmaganligiga qarab aniqlash mumkin. Agar OF=0 va CF=0 bo'lsa, demak, natija AL dan oshib ketmagan, aks holda natija 2 baytli sondan iborat.

Agar manba 2 baytli bo'lsa, u AX registridagi qiymatga ko'paytirilib, natija 4 baytli DX:AX registriga yoziladi. Bu yerda DX:AX bitta butun 4 baytli registr sifatida qaraladi. Agar ko'paytma 2 baytdan oshib ketmagan bo'lsa, natijangizni AX dan topasiz, ya'ni DX ga e'tibor qaratmasangiz ham bo'ladi. Agar manba 4 baytli bo'lsa, u holda ikkinchi ko'paytuvchi vazifasini EAX bajaradi va natija EDX:EAX ga yoziladi.

Ishorali sonlarni ko'paytirish uchun IMUL buyrug'idan foydalanamiz.
Qoidasi:

```
imul manba
```

Ushbu holda ham MUL uchun aytilgan fikrlar o'z kuchida qoladi. MUL dan farqli o'laroq IMUL dasturchiga maqsadni ko'rsatishga ham imkon beradi.
Qoidasi:

```
imul maqsad , manba
```

Bu yerda maqsad va manba ko'paytuvchilar vazifasini bajarib, natija ham maqsad ga yoziladi. IMUL buyrug'i hatto ikkinchi ko'paytuvchini alohida ko'rsatishga imkon beradi:

```
imul maqsad , manba1 , manba2
```

11-jadvalda maqsad va manba o'rnida aynan qanday o'lchamdagi o'zgaruvchi va o'zgarmaslar kelishi batafsil ko'rsatilgan. Agar jadvaldagi belgilash usullari tushunarsiz bo'lsa, «Buyruqlar ro'yxati» ilovasiga qarang.

11-jadval

maqsad	manba1	manba2	Bajariladigan amal
	reg_xotira8		AX = AL * manba1
	reg_xotira16		DX:AX = AX * manba1
	reg_xotira32		EDX:EAX = EAX * manba1
reg16	reg_xotira16		maqsad = maqsad * manba1
reg32	reg_xotira32		maqsad = maqsad * manba1
reg16	o'zgarmas16		maqsad = maqsad * manba1
reg32	o'zgarmas32		maqsad = maqsad * manba1
reg16	reg_xotira16	o'zgarmas16	maqsad = manba1 * manba2

maqsad	manba1	manba2	Bajariladigan amal
reg32	reg_xotira32	o'zgarmas32	maqsad = manba1 * manba2

Bo'lishda ham ishorali va ishorasiz sonlar uchun ikki xil buyruq ishlatiladi. Ishorasiz qiymatlar uchun `DIV` buyrug'i mavjud bo'lib, unga manba sifatida faqat bo'luvchi ko'rsatiladi:

```
div  manba
```

Xuddi ko'paytirishda bo'lgani kabi bu yerda ham bo'linuvchi yashirin keladi. `manba` o'lchamiga qarab bo'linuvchi sifatida mos ravishda AX, DX:AX yoki EDX:EAX juftligi olinadi. Agar bo'luvchi 1 baytli bo'lsa, u holda AX registri bo'linuvchi vazifasini bajaradi. Bo'linma AL da, qoldiq esa AH da saqlanadi. Agar bo'luvchi ikki baytli bo'lsa, u holda DX:AX bo'linuvchi sifatida olinadi va bo'linma AX ga, qoldiq esa DX ga o'zlashtiriladi. Agar faqat AX dagi sonni ikki baytli bo'luvchiga bo'lish kerak bo'lsa, u holda DX registriga tegishli qiymat berish lozim. Aks holda bo'lish jarayonida DX dagi qiymat ham bo'linuvchining to'ng'ich ikki bayti sifatida qaraladi va qandaydir tushunarsiz bo'linma kelib chiqadi. Tegishli qiymat deganimizning sababi shuki, bo'linmaga DX ning qiymati ta'sir ko'rsatmasligi uchun unga maxsus qiymat yuklash kerak. Agar ishorasiz bo'lishni amalga oshirayotgan bo'lsangiz, DX qiymatini nolga tenglashtirish kerak. Ishorali bo'lishda esa DX bitlarini barchasini AX ning ishora biti qiymatiga tenglashtirish kerak[1], ya'ni agar AX>0 bo'lsa, DX ← 0 va agar AX<0 bo'lsa, DX ← -1. Quyidagi misolda shunga o'xshash holat tasvirlangan:

```
mov  cx , 2
mov  ax , 20
div  cx
```

Misolda AX dagi 20 son CX dagi 2 ga bo'lindi, ya'ni $AX = \frac{DX:AX}{CX}$. Ammo assembler dastur boshlanishidan oldin o'zgaruvchi va registr qiymatlarini nolga teng qilib olmaydi, ya'ni o'zgaruvchilarga o'zingiz xohlagan qiymatlarni o'zlashtirmas ekansiz, registr va xotira sohalarida oldingi bajarilgan dastur qiymatlari saqlanib turaveradi. Bizning holatimizda ham `DIV` buyrug'i bajarilishi oldidan DX ning qanday qiymatga ega bo'lgani qorong'u va uning qiymati albatta natijaga ta'sir qilib, noto'g'ri bo'linma yuzaga keladi. Masalaning yechimi to'rt baytli DX:AX bo'linuvchisining to'ng'ich ikki baytini nol qilishdan iboratdir, ya'ni `DIV` buyrug'idan oldin DX ning qiymatini nollaymiz:

```
mov  dx , 0
```

Bo'luvchi 4 baytli bo'lganda ham xuddi shunday vaziyat yuzaga keladi. Faqat bu holatda bo'linuvchi EDX:EAX bo'ladi va bo'lishdan so'ng bo'linma EAX da, qoldiq esa EDX da bo'ladi.

Ishorali sonlarni bo'lishda `IDIV` buyrug'i ishlatiladi:

```
idiv manba
```

Ushbu buyruqqa ham `DIV` bilan bo'lgan barcha mulohazalar taalluqli.

Eslatma: Agar dastur bajarilayotgan paytda `manba`, ya'ni bo'luvchi nolga teng bo'lib qolsa yoki bo'linma `maqsad` ga sig'masa, o'sha yerga yetgach operatsion tizim dasturni to'xtatadi va buning iloji yo'qligi haqida xabar beradi.

[1] O'zgaruvchi bitlari bilan ishlash uchun «Qiymatlar o'lchamini kattalashtirish va kichiklashtirish» mavzusiga va «Mantiqiy amallar va bitlarni siljitish buyruqlari» bobiga qarang.

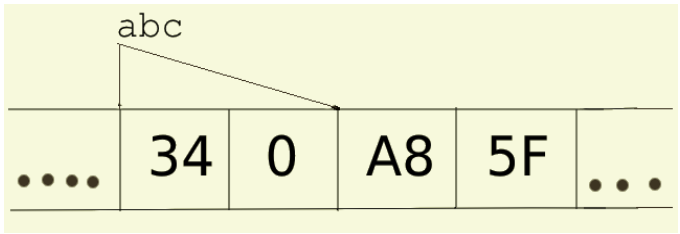
6.3. Qiymatlar o'lchamini kattalashtirish va kichiklashtirish

Butun qiymatlar bilan arifmetik amallarni ko'rib chiqar ekanmiz, o'zgaruvchi o'lchami qanday hal qiluvchi ahamiyatga ega ekanligi ko'zga tashlanadi. Misol uchun 5 soni EAX yoki CL registrlarida saqlanayotganiga qarab ikki xil o'lchamga ega bo'ladi. Shunday ekan qiymat o'lchamini kattalashtirish yoki kichiklashtirish imkoniyatiga ega bo'lish juda muhimdir. Birgina `chop_et` makrosini misol qilib olsak, u orqali o'zgaruvchi qiymatini ishorali butun son sifatida ekranga chop etish uchun o'zgaruvchi 4 baytli bo'lishi shart. Biroq hamma vaqt ham o'zgaruvchilar o'lchami 4 baytli bo'lavermaydi. Bu muammoni yechish uchun avval qiymat o'lchamini 4 baytgacha kattalashtirib, so'ng `chop_et` ga berishga to'g'ri keladi. Yuqorida `DIV` buyrug'i bilan ko'rib chiqqan dasturimizda ham AX dagi natijani `chop_et` orqali chop etib bo'lmas edi.

Avval muhimroq bo'lgan o'lchamni kattalashtirish usullarini o'rganishdan boshlaymiz. 2 baytli `abc` o'zgaruvchisidagi `0x34` sonini EAX registriga 4 bayt sifatida o'zlashtirish kerak bo'lsin. Bir tomondan olib qaraganda buni oddiy `MOV` buyrug'i orqali ham amalga oshirsa bo'ladigandek:

```
mov  eax , [abc]
```

Lekin bu *noto'g'ri*, chunki EAX 4 baytli o'zgaruvchi bo'lgani uchun unga xotira sohasidagi manzili `abc` dan boshlanadigan 4 baytli qiymat o'zlashtiriladi. Bu bilan biz `abc` o'zgaruvchisining chegarasidan chiqib, xotira sohasidagi keyingi 2 bayt qiymatni ham olamiz. EAX ning kenja 2 bayti 0x34 sonini bersada, uning to'ng'ich 2 baytida qandaydir noma'lum qiymat joylashtirilgan bo'ladi va agar biz EAX ni bir butun son sifatida `chop_et` orqali chop etsak, umuman, kutilmagan qiymatni ekranda ko'ramiz.



18-rasm.

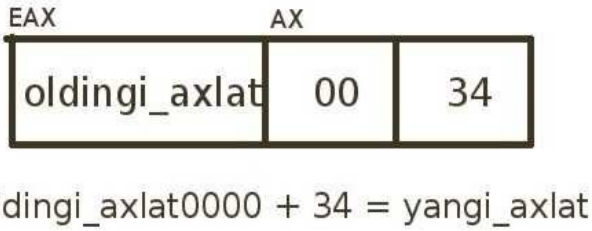
18-rasmdagi holatda EAX dagi kutilmagan qiymat `0x5FA80034` ga teng ekanligi ko'rsatilgan. Demak, `abc` dagi qiymatni aniq 2 bayt sifatida boshqa joyga ko'chirish uchun `maqsad` ham 2 baytli bo'lishi kerak. Bu muammoni yechimi EAX ning 2 baytli qism registri AX ni ishga solishda bo'lib ko'rinayotgandek:

```
mov  ax , [abc]
```

Mana endi AX dan biz xohlagan qiymat joy oldi, ya'ni `0x34`. Bu buyruqdan so'ng EAX ning faqat kenja 2 bayti o'zgartiriladi va EAX da qiymatimiz 4 baytli bo'lib turgandek. Ammo EAX ning to'ngi'ch 2 bayti o'zgarishsiz qoladi, ya'ni bu buyruqqacha qanday qiymat turgan bo'lsa, o'sha hali ham saqlanib turibdi. EAX ning oldingi qiymatining to'ng'ich 2 bayti nollardan iborat bo'lgan bo'lsa, bu yaxshi, ya'ni ko'chirgan sonimizga ta'sir qilmaydi. Agar nol bo'lmasa-chi? Bu vaziyatda agar biz EAX ga 4 baytli yaxlit son sifatida qaraydigan bo'lsak, butunlay noto'g'ri qiymat kelib chiqadi (rasmga qarang). Muammoning yakuniy yechimi avval EAX ning qiymatini nollab, so'ng AX ga `abc` ni ko'chirishdan iboratdir:

```
mov  eax , 0
mov  ax , [abc]
```

Oldin ko'rib chiqqan mavzularimizdan ham ma'lumki qiymatni o'lchamini kattalashtirish uchun uni chap tomondan nollar bilan to'ldirish kifoyadir (19-rasmga qarang). Lekin bu faqat *ishorasiz* sonlarga tegishli. Ishorali sonlar bilan esa masala bir oz qiyinlashadi. Ishorali sonning o'lchamini kattalashtirish uchun chapdan nol bilan emas, balki ishora bitining qiymati bilan to'ldirib chiqiladi.



19-rasm.

Protsessorda maxsus kattalashtirish buyruqlari mavjud bo'lib, ular bizga yuqoridagi ishlarni bitta buyruq orqali bajarish imkoniyatini beradi. Ishorasiz sonlar uchun `MOVZX` buyrug'i mavjud:

```
movzx       maqsad , manba
```

`manba` bo'lib kelgan 8 yoki 16 bitli qiymatni mos ravishda 16 yoki 32 bitli o'lchamlargacha kengaytirib, `maqsad` ga ko'chiradi.

Eslatma: `maqsad` faqat registr bo'lishi mumkin.

Ishorali sonlar uchun 8086 protsessorlar oilasi asosida qurilgan `CBW` (**C**onvert **B**yte to **W**ord) va `CWD` (**C**onvert **W**ord to **D**ouble **W**ord) buyruqlari mavjud.

- `CBW` buyrug'i AL registridagi bir baytini AX registriga 2 bayt ko'rinishida o'giradi.
- `CWD` buyrug'i AX registridagi 2 baytni DX:AX registriga 4 bayt ko'rinishida o'giradi.

Eslatma: Bu buyruqlar 8086 protsessorlar oilasiga tegishli ekanligiga urg'u berilganining sababi shuki, bu protsessorlar davrida hali kengaytirilgan 4 baytli EAX ga o'xshagan registrlar bo'lmagan. Shuning uchun ham u vaqtda 4 baytli registr sifatida DX:AX ga o'xshagan juftliklar ishlatilgan. Shunga o'xshagan juftliklarga duch kelinganda o'sha buyruq eski protsessorlar oilasiga mansub ekanini anglash lozim.

Keyingi 80386 protsessorlar oilasi bizga ancha qulay bo'lgan `CWDE` (**C**onvert **W**ord to **D**ouble word **E**xtended) va `CDQ` (**C**onvert **D**ouble word to **Q**uarter word) buyruqlarini taqdim etdi.

- `CWDE` buyrug'i AX dagi 2 baytni EAX ga 4 bayt ko'rinishida o'giradi.
- `CDQ` buyrug'i EAX dagi 4 baytni EDX:EAX ga 8 bayt ko'rinishida o'giradi.

Yuqorida ko'rilgan bu to'rt buyruq asosan EAX registri bilan ishlagani uchun ularga manba sifatida hech qanday obyekt ko'rsatilmaydi, ya'ni dasturda ularning o'zidan boshqa hech narsa yozilmaydi.

Va nihoyat, ishorali qiymatlar uchun `MOVZX` ga o'xshagan to'laqonli buyruq yaratilgan:

```
movsx       maqsad , manba
```

`MOVZX` uchun aytilgan barcha qoidalar `MOVSX` ga ham taalluqlidir.

Misol uchun $y=\frac{a+b}{c}$ qiymatini hisoblaymiz, bu yerda `a`, `b`, `c` bir baytli `y` esa 4 baytli o'zgaruvchilardir.

2-namuna: (a + b) / c ni top.

```
(1)    ;; Maqsad:
(2)    ;;  Ushbu dastur y = (a + b)/c ni hisoblaydi va
(3)    ;;  y ni natija sifatida chop etadi.
(4)
(5)    %include "nasm-io.inc"
(6)
(7)    section  .bss
(8)    y   resd 1
(9)
(10)   section  .data
(11)   a   db   6
(12)   b   db   10
(13)   c   db   2
(14)
(15)   section  .text
(16)   tizim_global  main
(17)
(18)   main:
(19)       mov  al , [a]
(20)       cbw                        ;; AX <- AL
(21)       movsx     bx , [b]         ;; BX <- b
(22)       add  ax , bx               ;; AX <- a + b
(23)       idiv byte[c]          ;; AL <- AX / c, qoldiq AH da bo'ladi.
(24)       cbw                        ;; AX <- AL
(25)       cwde                       ;; EAX <- AX
(26)       mov  [y] , eax             ;; y <- (a+b)/c
(27)       chop_et   `Natija: y = %i\n`, [y]
(28)
(29)   ret
```

Bu yerda barcha o'lchamni kattalashtirishlar ishorali sonlar uchun bajarilgan.

O'lchamni kichiklashtirish usullari unchalik ko'p foydalanilmaydi. Masalan, o'lchami to'g'ri kelmaydigan ikkita o'zgaruvchidagi qiymatlarni qo'shmoqchi bo'lsangiz, ko'p hollarda kichik o'lchamni kattalashtirishga harakat qilasiz. Chunki katta o'lchamli o'zgaruvchi keng imkoniyatlar yaratib beradi. Bundan tashqari natija oshib ketmadimikan degan xavotirdan qutulasiz. Yuqorida yuritgan fikrlarimizdan kerakli xulosa chiqargan holda o'lchamni kichiklashtirish usullarini o'quvchi o'zi o'ylab topadi degan umidda bunga keng to'xtalib o'tirmaymiz. Faqat shuni aytishimiz mumkinki, o'lchamni kichiklashtirish uchun o'zgaruvchining kenja baytlariga murojaat qilish kifoya. Bu usul ishorali yoki ishorasiz sonlar bilan ham birdek to'g'ri ishlaydi. Masalan BX da `0x0078` bo'lsin, biz uni 1 baytli DL ga ko'chirmoqchimiz:

```
mov  dl , bl          ;; DL = 0x78
```

Agar, masalan, BX da `0x0178` bo'lganda, o'lchamni 1 baytgacha kichraytirib bo'lmas edi.

6.4. Boshqaruv buyruqlari

Dasturlashning yana bir eng muhim tushunchalaridan biri bu boshqaruv buyruqlaridir. Boshqaruv deganda biz ma'lum bir shartlar bajarilishiga qarab dasturning turli bo'laklari ishga tushishini tushunamiz. Masalan, ikki sonni katta-kichikligini taqqoslash natijasiga ko'ra dasturni tarmoqlash shular jumlasidandir.

Asosiy gap shundaki, qandaydir shart bajarilishi yoki bajarilmasligiga qarab dasturning qaysi bo'lagini ishlatish boshqarib turiladi. Buni falsafiy tomondan quyidagicha izohlash mumkin:

– Agar mantiqiy shart haqiqat bo'lsa, u holda birinchi amallar ketma-ketligi
– Aks holda ikkkinchi amallar ketma-ketligi bajariladi.

Yuqori daraja tillarida bunday holatlarni amalga oshirish uchun `if ... then ... else` yoki `if ... else` kabi ifodalardan foydalaniladi. Ammo assemblerda bunday «murakkab» buyruqlar yo'q. Buning o'rniga mantiqiy shart yoki ifodani haqiqat yoki yolg'on ekanligini tekshirish uchun qiymatlar oddiy usulda taqqoslanadi.

6.4.1. Taqqoslash buyrug'i

Ikki o'zgaruvchi qiymatini taqqoslashda `CMP` (**Cо*mp*are** - Taqqoslamoq) buyrug'idan foydalaniladi:

```
cmp  chap_manba , o'ng_manba
```

Taqqoslash uchun ushbu buyruq `chap_manba` dan `o'ng_manba` ni ayiradi va agar natija musbat chiqsa, `chap_manba` ni katta deb topadi va aksincha. Lekin `CMP` ning ayirish buyrug'idan farqi shundaki, u ayirmani biror yerda saqlamaydi, ya'ni `maqsad` talab qilinmaydi. Agar sizni ayirma ham qiziqtirsa, unda `CMP` ning o'rnida bemalol `SUB` dan foydalanishingiz mumkin. `SUB` buyrug'i uchun aytilgan barcha qoidalar `CMP` ga ham taalluqlidir. Taqqoslash bajarilgandan so'ng `CMP` natijasi turli xil bayroqlar o'rnatilgan yoki o'rnatilmaganligiga qarab aniqlanadi.

Ishorasiz qiymatlarni taqqoslaganda faqat ZF va CF bayroqlari tekshiriladi. Agar `chap_manba = o'ng_manba` bo'lsa, u holda ZF bayrog'i o'rnatiladi, CF esa o'rnatilmaydi, ZF=1 va CF=0. ZF bayrog'i qanday buyruq bo'lishidan qat'i nazar, agar natija nolga teng bo'lsa, o'rnatiladi. Agar `chap_manba > o'ng_manba` bo'lsa, u holda ZF=0 va CF=0. Chunki bu yerda to'ng'ich bit uchun hech qanday bir-qarz olinmaydi, natija ham nolga teng emas. Agar `chap_manba < o'ng_manba` bo'lsa, u holda ZF=0 va CF=1. Bu yerda to'ng'ich bit uchun bir-qarz olishga to'g'ri keladi va buni protsessor CF da belgilab qo'yadi.

Ishorali sonlarni taqqoslaganda ZF, OF va SF bayroqlariga murojaat qilinadi. Agar natija manfiy bo'ladigan bo'lsa SF bayrog'i o'rnatiladi. OF bayrog'i esa ishora bitlari bilan har xil «kimyoviy» hodisalar ro'y berganida o'rnatiladi. Agar `chap_manba = o'ng_manba` bo'lsa, u holda ZF=1. Agar `chap_manba > o'ng_manba` bo'lsa, u holda ZF bayrog'i o'rnatilmaydi va SF=OF. Agar `chap_manba < o'ng_manba` bo'lsa, u holda ZF=0 va SF≠OF. Nega bunday bo'lshini topish kitobxon vasifasiga yuklanadi.

> **Eslatma**: `chap_manba > o'ng_manba` bo'lganda SF=OF bo'lishiga sabab shuki, agar natija musbat bo'lsa, demak, SF o'rnatilmaydi va ishora bitidan qarz olish ham kuzatilmaydi, ya'ni SF=OF=0. Agar ikkala manbada ham manfiy sonlar bo'lsa, natija ham manfiy bo'ladi va SF o'rnatiladi. Bu holda OF ham albatta o'rnatiladi, chunki ishora bitidan qarz olinadi, ya'ni SF=OF=1.

6.4.2. Tarmoqlash buyruqlari

Tarmoqlash buyruqlari turli bayroqlar o'rnatilgan yoki o'rnatilmaganligiga qarab dasturni turli tarmoqlar bo'yicha harakat qilishini ta'minlaydi. Masalan, `CMP` orqali ikki qiymatni o'zaro taqqoslab tekshirgan shartimiz bajarilgan yoki bajarilmaganligidan kelib chiqib dasturning

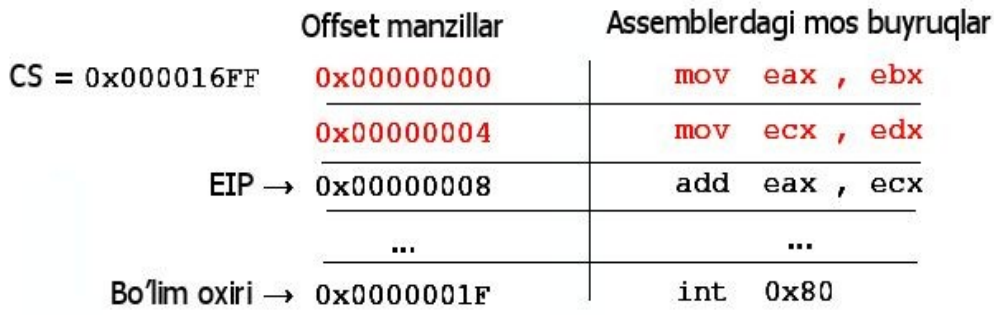
qolgan qismini turlicha ishlatishimiz mumkin. Buni amalga oshirish uchun dasturning ma'lum bir qismida shart haqiqat bo'lganda bajarilishi kerak bo'lgan buyruqlar ketma-ketligini joylashtirsak, dasturning boshqa qismida esa shart yolg'on bo'lgandagi bajarilishi kerak bo'lgan buyruqlar ketma-ketligini joylashtiriladi. Taqqoslashdan so'ng esa kerakli dastur qismiga «sakrash» orqali o'tamiz, ya'ni kerakli tarmoqga ko'chamiz.

Shu paytgacha ko'rib chiqqan dasturlarda buyruqlar ketma-ket tarzda bir boshdan bajarilar edi. Endi esa tarmoqlash buyruqlari orqali dasturning xohlagan qatoriga o'tish mumkin. Ikki xil, ya'ni shartli va shartsiz o'tishlar mavjud. Shartsiz o'tishda hech qanday bayroq qiymati tekshirilmaydi va so'zsiz ko'rsatilgan nuqtaga dastur boshqaruvi topshiriladi. Shartli o'tishda esa ma'lum bayroqlar o'rnatilgan bo'lsa, shundagina o'tish yuz beradi. Aks holda hech qanday o'tish bo'lmaydi va tarmoqlash buyrug'idan so'ng keladigan navbatdagi buyruq bajariladi. Shartsiz o'tish `JMP` (**Ju**mp – Sakrash) buyrug'i orqali amalga oshiriladi:

```
jmp  [aniqlovchi]  o'tish_manzili
```

Bu yerda `aniqlovchi` ning burchakli qavs ichiga olinishiga sabab uni dasturchi berishi ham, bermasligi ham mumkin. Aniqlovchi haqida sal keyinroq to'xtalamiz. O'tish manzili esa `JMP` buyrug'iga dasturning qaysi qatoriga o'tib o'sha yerdan buyruqlarni bajarishni boshlashni ko'rsatadi.

`JMP` buyrug'i ishlash tartibiga batafsilroq to'xtalamiz. Yodingizda bo'lsa, biz «Kompyuter tuzilishi» bobida EIP boshqaruv registri bilan tanishgan edik. Bu registrda dasturning kod bo'limidagi bajarilish bo'yicha navbatda turgan buyruqning offset manzili saqlanadi. 20-rasmda bu yaqqol tasvirlangan.



20-rasm.

Rasmda bajarib bo'lingan buyruqlar och rang bilan bo'yalgan. Demak, EIP da `add eax , ecx` buyrug'ining foydali manzili turibdi, ya'ni `0x00000008`. Bu manzillar mantiqiy bo'lib, bo'lim asosiy xotiraga yuklanganda haqiqiy manzillar CS:EIP juftligi orqali hisoblanadi. Bizning misolimizda EIP ko'rsatib turgan buyruqning haqiqiy manzili 000016FF:00000008 ga teng bo'ladi.

Sinchkov o'quvchi `JMP` buyrug'i qanday tuzulganini anglagan bo'lsa kerak. U qiladigan ish bor yog'i EIP registriga biz ko'rsatgan o'tish manzilini yuklashdan iboratdek:

```
mov  eip , o'tish_manzili
```

Lekin biz EIP registrini to'g'ridan-to'g'ri ishlata olmaymiz. Bu misolimiz ham ramziy ma'noga ega, uni dasturda ishlatib bo'lmaydi.

`JMP` bilan ishlaganda o'tish manzilini son sifatida berish dasturchi uchun juda o'ng'aysiz. Chunki o'tish kerak bo'lgan buyruqning segmentdagi joylashuv manzilini hisoblab chiqishga to'g'ri keladi. Assemblerda bu masala o'tish kerak bo'lgan qatorga nishon qo'yish bilan yechiladi. «O'zgaruvchilarni e'lon qilish» mavzusida nishonlar bilan tanishgan edik. Faqat o'shanda

nishonlar o'zgaruvchilarni e'lon qilishda `.bss` yoki `.data` bo'limlarida ishlatilgan edi. Bu safar esa biz nishonlarni o'zgaruvchilar uchun ajratilgan manzillarga emas, balki kod bo'limidagi foydali manzillarga qo'yamiz.

Assemblerda nishonlar birinchi marta e'lon qilinganda, ulardan keyin ikki nuqta (`:`) qo'yish mumkin. Bu belgining nishon nomiga hech qanday aloqasi yo'q. Bundan keyin tarmoqlash maqsadida qo'yilgan nishonlardan keyin ikki nuqtadan foydalanishga kelishib olsak:

```
nishon:
     buyruq_1
     ...
     buyruq_n
```

Bu yerda `buyruq_1` ning foydali manziliga `nishon` ko'rsatgich qilib olinadi. Dasturning xohlagan yeridan tarmoqlash buyruqlariga shu foydali manzilni, ya'ni nishon nomini bergan holda `buyruq_1` ni bajarishiga o'tishimiz mumkin. Misol:

```
boshla:
     mov  eax , 10
     chop_et `Salom\n`
     jmp  boshla
```

Bu dastur bo'lagi cheksiz ko'p marta bajariladi. Chunki EAX ga 10 o'zlashtirilib `Salom` chop etilgandan so'ng dastur so'zsiz yana `boshla` nishoniga o'tadi va bu ish har safar takrorlanadi.

Tarmoqlash buyruqlariga nishondan oldin qo'yilishi mumkin bo'lgan aniqlovchilar shu nishonga necha baytli ishorali son sifatida qarash kerak ekanligini bildiradi. NASM dasturdagi barcha nishonlarni mos sonlarga almashtirib chiqadi, chunki nishonlar ko'rsatgich bo'lib turgan manzillar ham aslini olganda sonlardir. Quyidagi aniqlovchilar berilishi mumkin:

SHORT	Nishon bir baytli son sifatida almashtiriladi. Demak, ko'pi bilan 128 bayt yuqoriga yoki pastga sakrashni amalga oshirish imkoniyatini beradi. Manzil ishorali son sifatida hisoblanib, EIP ga qo'shiladi.
NEAR	Agar hech qanday aniqlovchi qo'yilmasa, NASM ushbu aniqlovchi bor deb qabul qiladi. Protsessor rusumiga qarab nishon to'rt yoki ikki baytli son sifatida almashtiriladi. Masalan, ikki baytli holatda taxminan 32000 baytgacha pastga yoki tepaga sakrash imkoniyati bo'ladi. Agar siz aniq ikki baytli sakrashni `NEAR` bilan amalga oshimoqchi bo'lsangiz nishondan oldin `WORD` aniqlovchisini qo'shimcha tarzda qo'yishingizga to'g'ri keladi.
FAR	Kod bo'limlari aro o'tishda, ya'ni uzoqqa sakrashda foydalaniladi. Bunda ham EIP registri, ham CS registri o'zgartiriladi.

Endi shartli o'tishlar bilan tanishamiz. Ular ma'lum bir bayroqlar o'rnatilgan bo'lsagina o'tishni sodir etadi, aks holda navbatdagi buyruq bajarilaveradi. Buyruqlar:

`JZ` – (**J**ump if **Z**ero) agar ZF bayrog'i o'rnatilgan bo'lsa, o'tish yuz beradi.
`JNZ` – (**J**ump if **N**ot **Z**ero) agar ZF o'rnatilmagan bo'lsa, o'tish yuz beradi.
`JO` – (**J**ump if **O**verflow) agar OF bayrog'i o'rnatilgan bo'lsa, o'tish yuz beradi.
`JNO` – (**J**ump if **N**ot **O**verflow) agar OF bayrog'i o'rnatilmagan bo'lsa, o'tish yuz beradi.
`JS` – (**J**ump if **S**ign) agar SF bayrog'i o'rnatilgan bo'lsa, o'tish yuz beradi.
`JNS` – (**J**ump if **N**ot **S**ign) agar SF bayrog'i o'rnatilmagan bo'lsa, o'tish yuz beradi.

`JC` – (**J**ump if **C**arry) agar CF bayrog'i o'rnatilgan bo'lsa, o'tish yuz beradi.
`JNC` – (**J**ump if **N**ot **C**arry) agar CF bayrog'i o'rnatilmagan bo'lsa, o'tish yuz beradi.
`JP` – (**J**ump if **P**arity) agar PF bayrog'i o'rnatilgan bo'lsa, o'tish yuz beradi.
`JNP` – (**J**ump if **N**ot **P**arity) agar PF bayrog'i o'rnatilmagan bo'lsa, o'tish yuz beradi.

Misol tariqasida quyidagi dasturni ko'rib chiqamiz: Agar EAX ning qiymati 20 ga teng bo'lsa, EBX ga 100 ni o'zlashtiramiz, aks holda EBX ga 200 ni o'zlashtiramiz. C dasturlash tilida bu quyidagicha ko'rinishga ega:

```
If (EAX==20) EBX=100; else EBX=200;
```

Assembler esa:

```
(1)       cmp  eax , 20       ;; EAX – 20 ?
(2)       jz   u_holda        ;; Agar EAX=20, ya'ni ZF=1 bo'lsa
(3)       mov  ebx , 200      ;; Demak ZF=0, ya'ni EAX≠0
(4)       jmp  davom          ;; u_holda ni chetlab o'tamiz
(5)  u_holda:
(6)       mov  ebx , 100      ;; EBX ← 100, chunki EAX=20
(7)  davom:                   ;; Dastur bajarilishi davom etadi
```

2-qatorda agar EAX 20 ga teng bo'lmasa, o'tish yuz bermaydi va 3-qatordagi buyruq bajariladi. Agar 4-qatorda shartsiz o'tish qo'yilmasa va EAX 20 ga teng bo'lmay qolsa, avvaliga `mov ebx, 200` bajariladi so'ng dastur 6-qatorga tushib `mov ebx, 100` ni bajaradi. Natijada EAX da qanday son bo'lishidan qat'i nazar EBX ga 100 soni yuklanadi.

Navbatdagi misol biroz qiyinroq. Endi EAX ni 20 dan katta yoki tengligini tekshiramiz. EAX dagi qiymatga ishorali son sifatida qaraymiz. Agar ZF=1 yoki OF=SF bo'lsa, demak, EAX≥20, aks holda EAX<20. C dagi ko'rinishi:

```
if (EAX>=20) EBX = 100; else EBX = 200;
```

Assemblerda esa:

```
(1)       cmp  eax , 20       ;; EAX – 20 ?
(2)       js   ishora         ;; agar SF = 1 bo'lsa
(3)       jo   aks_holda      ;; agar OF = 1 va SF = 0 bo'lsa
(4)       jmp  u_holda        ;; agar SF = 0 va OF = 0 bo'lsa
(5)  ishora:
(6)       jo   u_holda        ;; agar SF = 1 va OF = 1
(7)  aks_holda:
(8)       mov  ebx , 200      ;; EBX ← 200
(9)       jmp  davom          ;; u_holda ni chetlab o'tamiz
(10) u_holda:
(11)      mov  ebx , 100      ;; EBX ← 100
(12) davom:
```

Biz 4-qatorgacha faqat OF=SF=0 bo'lgandagina yetib borishimiz mumkin, ya'ni EAX≥20. Shuning uchun ham 4-qatordan so'zsiz `u_holda` tarmog'iga o'tamiz. 6-qatordan `.u_holda` tarmog'iga faqat SF=OF=1 bo'lganda o'ta olamiz. Bu degani EAX>5. (SF=0 va OF=1) yoki (SF=1 va OF=0) bo'lganda biz `aks_holda` tarmog'iga o'tamiz.

Ko'rinib turibdiki, masala ancha chigal. Taqqoslashning natijasini tahlil qilish uchun bayroqlar bilan murakkab «jarrohlik» ishlari amalga oshirilayapti. Buning oldini olish uchun assemblerda boshqa turdagi tarmoqlash buyruqlari mavjud. Bu buyruqlar ma'lum bir bayroq o'rnatilganligiga

qarab emas balki yuz bergan holatga qarab o'tishni amalga oshiradi. Bu buyruqlar oilasi ishorali va ishorasiz sonlar uchun ikki guruhga bo'linadi. Taqqoslashni ramziy x va y uchun keltiramiz:

```
cmp  x , y
```

Umumiy buyruqlar:

JE – (**J**ump if **E**qual) Agar $x=y$ bo'lsa, shunda o'tish amalga oshiriladi. JZ kabi.
JNE – (**j**ump if **N**ot **E**qual) Agar $x \neq y$ bo'lsa, o'tish amalga oshiriladi.

Ishorali sonlarni taqqoslaganda:

JL – (**J**ump if **L**ess than) Agar $x<y$ bo'lsa, o'tishni amalga oshiradi.
JLE – (**J**ump if **L**ess than or **e**qual) Agar $x \leq y$ bo'lsa, o'tishni amalga oshiradi.
JG – (**J**ump if **G**reater than) Agar $x>y$ bo'lsa, o'tishni amalga oshiradi.
JNLE – (**J**ump if **N**ot **L**ess than or **E**qual) JG kabi.
JGE – (**J**ump if **G**reater than or **E**qual), Agar $x \geq y$ bo'lsa, o'tishni amalga oshiradi.
JNL – (**J**ump if **N**ot **L**ess than) JGE kabi.

Ishorasiz sonlarni taqqoslaganda:

JB – (**J**ump if **B**elow) Agar $x<y$ bo'lsa, o'tishni amalga oshiradi.
JNAE – (**J**ump if **N**ot **A**bove or **E**qual) JB kabi.
JBE – (**J**ump if **B**elow or **E**qual) Agar $x \leq y$ bo'lsa, o'tishni amalga oshiradi.
JNA – (**J**ump if **N**ot **A**bove) JBE kabi.
JA – (**J**ump if **A**bove) Agar $x>y$ bo'lsa, o'tishni amalga oshiradi.
JNBE – (**J**ump if **N**ot **B**elow or **E**qual) JA kabi.
JAE – (**J**ump if **A**bove or **E**qual) Agar $x \geq y$ bo'lsa, o'tishni amalga oshiradi.
JNB – (**J**ump if **N**ot **B**elow) JAE kabi.

Ikki xil ishorali sonlar uchun mo'ljallangan, ammo bitta maqsadda foydalaniladigan buyruqlar bir-biridan ajralib turishi uchun ikki xil atamalardan foydalanilgan. Masalan, «katta» ma'nosini berishda «greater» va «above» so'zlari ishlatilgan.

Endi yuqoridagi misolimiz bu buyruqlar bilan qanday oson kechishini ko'rib o'tamiz:

```
(1)       cmp  eax , 20
(2)       jge  u_holda
(3)       mov  ebx , 200
(4)       jmp  davom
(5)   u_holda:
(6)       mov  ebx , 100
(7)   davom:
```

Shunday qilib, biz assembler tilida tarmoqlash, ya'ni *agar … u holda … aks holda …* ko'rinishidagi ifodalarni qanday tuzishni o'rganib oldik. Bunday ifodalarni umumiy tuzilishini quyidagicha yozish mumkin. C dasturlash tilida:

```
if (mantiqiy_shart)
     u_holda tarmog'i
else
     aks_holda tarmog'i
```

Assemblerda:

```
cmp  mantiqiy_shart
jxx  aks_holda      ;; xx ning o'rniga keraklisini tanlang.
```

```
;; u_holda tarmog'ining dastur kodi
jmp  cmp_oxiri
aks_holda:
;; aks_holda tarmog'ining dastur kodi
cmp_oxiri:
```

Asosiy tushunchalardan yana biri takrorlanish tushunchasidir. Bu tushuncha ingliz tilida *loop, cycle* yoki *iteration* deb, rus tilida esa *цикл* deb yuritiladi. Agar ma'lum bir buyruqlar ketma-ketligining bajarilishi ko'p marta takrorlanishi kerak bo'lsa, undan foydalaniladi. Takrorlanish qandaydir bir shart bajarilmaguncha yoki talab qilingan marta qadar takrorlanmaguncha qaytalanaveradi.

C kabi yuqori daraja tillarida `while ... , do ... while` yoki `for ...` kabi boshqaruv ifodalari berilgan. Assemblerda biz buning uchun oddiy tarmoqlash buyruqlaridan foydalanamiz.

Masalan, siz ekranga 10 marta "Salom" so'zini chop etmoqchi bo'lsangiz, 10 marta `chop_et` makrosini yozib o'tirmaysizda, uning o'rniga bir marta yozib buni takrorlanish halqasi ichiga olasiz. Takrorlanish necha marta amalga oshirilganini sanab turish uchun qo'shimcha o'zgaruvchidan foydalanishga to'g'ri keladi. Bu ECX registri bo'la qolsin.
C dagi ko'rinishi:

```
ecx = 0;
while (ecx != 10)
{
          printf ("Salom\n");
          ecx++;
}
```

Assemblerdagi ko'rinishi:

```
(1)       mov  ecx , 0          ;; ECX ← 1
(2)  halqa_boshi:               ;; while
(3)       cmp  ecx , 10         ;; ECX - 10 ?
(4)       jz   halqa_oxiri      ;; takrorlanish tugadi
(5)       chop_et  `Salom\n`
(6)       inc  ecx              ;; ECX++
(7)       jmp  halqa_boshi
(8)  halqa_oxiri:
```

Eslatma: C da MP registrlari bilan to'g'ridan-to'g'ri ishlash imkoniyati yo'q. C da keltirilgan dastur bo'lagidagi `ecx` oddiy o'zgaruvchi bo'lib, uning faqat nomi registrni eslatadi.

6.4.3. LOOP buyrug'i

Ushbu buyruq tarmoqlash buyrug'i bo'lib takrorlanishlarni oson amalga oshirish uchun yaratilgan. `LOOP` buyrug'i ECX qiymati nolga teng bo'lmaguncha takrorlanishni davom ettiradi. Har safar qaytalaganda ECX ni bittaga kamaytiradi. Masalan, noldan o'ngacha bo'lgan sonlar arifmetik progressiyasining hadlar yig'indisini hisoblamoqchi bo'ldik deylik.

C dagi ko'rinish:

```
ecx = 10;
eax = 0;
while (ecx != 0)
{
```

```
eax = eax + ecx;
ecx--;
}
```

Assemblerdagi ko'rinish:

```
(1)             mov  eax , 0
(2)             mov  ecx , 10
(3)     halqa_boshi:
(4)             add  eax , ecx
(5)             loop halqa_boshi
```

`LOOPE` yoki `LOOPZ` buyrug'i `LOOP` dan farqli o'laroq ZF bayrog'i qiymatini ham hisobga oladi. Agar ECX≠0 va ZF=1 bo'lsa, takrorlanish davom etadi.

`LOOPNE` yoki `LOOPZE` buyrug'i, agar ECX≠0 va ZF=0 bo'lsa, takrorlanish davom etadi. Mazkur buyruqlarning bayroqlarga, ya'ni EFLAGS registriga ta'siri yo'q.

6.5. Amaliy dastur: Tub sonlarni topish

Mazkur bobda o'rgangan bilimlarimizni mustahkamlash maqsadida amaliy dasturni ko'rib chiqamiz. Dastur vazifasi foydalanuvchi ko'rsatgan songacha (chegaragacha) bo'lgan barcha tub sonlarni topib ularni ekranga chop etishdan iborat. Sonning tub yoki tub emasligini aniqlash usullari ko'p. Biz quyidagi nazariyadan foydalanamiz: agar son o'zining yarmigacha bo'lgan sonlarning birortasiga ham qoldiqsiz bo'linmasa, demak, u tub sondir. Dasturimiz quyidagi algoritm asosida ishlaydi. Avval foydalanuvchi kiritgan chegara son ikkidan kichik bo'lsa, unda dasturni davom ettirishdan hech qanday foyda yo'q va biz dasturni tark etamiz. Agar hammasi joyida bo'lsa, 2 sonini chop etamiz, chunki uning tub son ekanligini aniq o'zimiz ham bilamiz. Shunday qilsak, dastur ikkini tub yoki tub emasligini aniqlashga vaqt sarflamaydi va tez ishlaydi. Shundan keyin asosiy algoritm ishga tushadi. U uchdan chegara sonigacha bo'lgan barcha toq sonlarni tub yoki tub emasligini tekshirib chiqadi: $x \in \{2n+1 \ldots chegara\}, n \in \{1,2,3 \ldots \infty\}$. Dastur tez ishlashi uchun biz faqat toq sonlarni tekshiramiz.

C dagi ko'rinish:

3-namuna: tub_sonlar.c

```
(1)   #include <stdio.h>
(2)   int main()
(3)   {
(4)         unsigned int chegara, yarmi, x, maxraj = 2, tubmi = 1;
(5)         printf("Nechagacha bo'lgan tub sonlar topilsin: ");
(6)         scanf("%i",&chegara);
(7)
(8)         if(chegara < 2) goto tamom;
(9)
(10)        printf("2 \n");
(11)
(12)        for(x=3; x<=chegara; x+=2)
(13)        {
(14)              yarmi = (int) x / 2;
(15)
(16)              while(yarmi >= maxraj)
(17)              {
```

```
(18)                 if((x % maxraj)==0)
(19)                 {
(20)                      tubmi = 0;
(21)                      break;
(22)                  }
(23)                  maxraj++;
(24)            }
(25)            if(tubmi)
(26)                 printf("%i\n",x);
(27)
(28)            tubmi = 1;
(29)            maxraj = 2;
(30)      }
(31)
(32) tamom:
(33)
(34) return 0;
(35) }
```

Assemblerdagi ko'rinish:

4-namuna: tub_sonlar.asm

```
(1)  ;; Maqsad:
(2)  ;;Ushbu dastur berilgan songacha bo'lgan
(3)  ;;barch tub sonlarni chop etadi.
(4)
(5)  %include "nasm-io.inc"
(6)
(7)  section .data
(8)  maxraj dd   2
(9)  ikki   dd   2
(10) tubmi  db   1
(11)
(12) section .bss
(13) chegara resd 1
(14) yarmi  resd 1
(15)
(16) section .text
(17) tizim_globalmain
(18)
(19) main:
(20)   chop_et   `Nechagacha bo'lgan tub sonlar topilsin? : `
(21)   qabul_qil `%i`, chegara
(22)   cmp  dword[chegara] , 2
(23)   jb   near tamom
(24)   chop_et   `2\n`          ;; Ikkining tub ekanligi aniq
(25)   mov  ecx , 3             ;; x o'rnida ECX
(26)
(27) asosiy_halqa:              ;; C dagi "for"
(28)   cmp  [chegara] , ecx
(29)   jb   tamom
(30)   mov  edx , 0             ;; EDX:EAX ning to'ng'ich to'rt
(31)                            ;; baytini nollaymiz.
(32)   mov  eax , ecx           ;; ECX ni bo'lishga tayyorlash.
```

```
(33)    div  dword[ikki]
(34)    mov  ebx , eax           ;; yarmi o'rnida EBX
(35)
(36) ichki_halqa:                ;; C dagi "while"
(37)    cmp  ebx , [maxraj]
(38)    jb   ichki_halqa_oxiri
(39)    mov  edx , 0             ;; EDX:EAX ning to'ng'ich to'rt
(40)                             ;; baytini nollaymiz.
(41)    mov  eax , ecx           ;; ECX ni bo'lishga tayyorlash.
(42)    div  dword [maxraj]
(43)    inc  dword [maxraj]      ;; C dagi "maxraj++"
(44)    cmp  edx , 0             ;; Qoldiqni tekshiramiz
(45)    jne  ichki_halqa
(46)    mov  byte [tubmi] , 0
(47)
(48) ichki_halqa_oxiri:
(49)    cmp  byte [tubmi] , 1    ;; C dagi "if(tubmi)"
(50)    jne  tub_emas
(51)    chop_et   `%i\n`, ecx
(52)
(53) tub_emas:
(54)    mov  byte [tubmi] , 1
(55)    mov  dword [maxraj] , 2
(56)    inc  ecx
(57)    jmp  asosiy_halqa
(58)
(59) tamom:
(60)
(61) ret
```

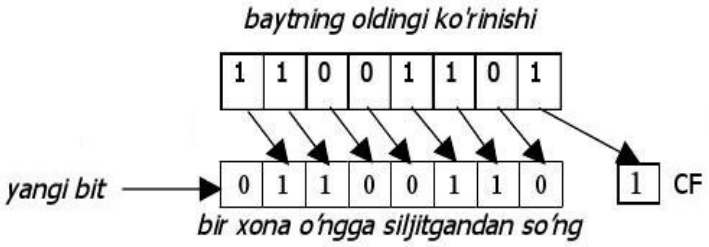
VII bob
Mantiqiy amallar va bitlarni siljitish buyruqlari

Dasturlashda baytlar bilan bir qatorda ularni tashkil qiluvchi bitlar bilan ishlashga ko'nikma hosil qilish muhim ahamiyatga ega. Shu paytgacha ko'rib chiqilgan buyruqlar qiymatni baytlarda berilgan yaxlit son sifatida qayta ishlar edi. Ushbu bobda ko'rib chiqiladigan buyruqlar ularga berilgan o'zgaruvchilarning bitlari ustida alohida ish yuritish imkoniyatini beradi. Bunday buyruqlarning mantiqiy deb atalishiga sabab shuki, ularning natijalari mantiq fanida o'rganiladigan haqiqat va yolg'on tushunchalariga asoslangan.

Bir bit ma'lumot 1 yoki 0 ga teng bo'lishi mumkinligi ma'lum. Bular mos ravishga haqiqat yoki yolg'on deb tushuniladi. Mantiqda bu ikki tushuncha ustida har xil amallar bajarish mumkin. Bu amallar dasturlashda ham mavjud bo'lib, uning asosini tashkil etadi. Kompyuter protsessori juda sodda bo'lgan elektr zanjirlardan iborat. Bu zanjirlar oddiy bit ustida ish bajarishi mumkin. Masalan, `ADD` buyrug'iga ikki 32 bitli o'zgaruvchi qo'shish uchun berilganida, protsessor hech qachon ular ichidagi ma'lumotni butun qiymat sifatida qo'sha olmaydi. Buning o'rniga u amalni *bitma-bit* bajaradi.

7.1. Bitlarni siljitish buyruqlari

Bitlarni siljitish deganda biz o'zgaruvchining har bir bitini o'ngga yoki chapga bir yoki bir necha xonaga siljitishni tushunamiz. Masalan, bizda bir bayt axborot bo'lsin, 11001101. Bu ishorasiz olinganda 205 sonini beradi. Bu bayt bitlarini bir xona o'ngga siljitib ko'ramiz. Siljitishda chap tomondan to'ng'ich bit o'rniga yangi bit sifatida 0 kiradi. O'ng tomondan chiqib ketayotgan kenja bit qiymati CF bayrog'iga o'zlashtiriladi (21-rasmga qarang).



21-rasm.

Endi 11001101 ni bir xona chapga siljitib ko'ramiz. Siljitishda chiqib ketayotgan to'ng'ich bit qiymati CF ga o'tadi. O'ngdan yangi bit sifatida 0 kirib keladi (22-rasmga qarang).



22-rasm.

Siljitishda doim necha xonaga bo'lmasin oxirgi chiqib ketayotgan bit qiymati CF bayrog'iga o'zlashtiriladi. Yangi kirayotgan bitlar esa 0 ga teng.

7.1.1. Mantiqiy siljitish

Yuqorida izohlangan siljitishlar mantiqiy siljitish deb ataladi va ular SHL (**S**hift **L**eft) va SHR (**S**hift **R**ight) buyruqlari orqali amalga oshiriladi.

```
shl  maqsaq , siljitish_miqdori
```

Bu buyruq chap tomonga siljishni amalga oshiradi.

```
shr  maqsad , siljitish_miqdori
```

Bu buyruq o'ng tomonga siljishni amalga oshiradi.

Siljish miqdori bu necha xonaga siljishni ko'rsatuvchi son bo'lib, maqsad bitlari shuncha xonaga siljitiladi. Siljitish miqdori o'rnida siz o'zgarmas son yoki CL qism registrini berishingiz mumkin. CL registri berilganda siljish miqdorini hisoblash uchun kenja besh bit olinadi. Chunki besh bitga sig'adigan eng katta qiymat 31 ta xonagacha bo'lgan siljitishlarni amalga oshirish imkoniyatini beradi. Undan ko'proq xonaga siljitishning hech qanday ma'nosi yo'q.

Misollar:

```
mov  al , 1000_1001b     ;; AL ← 137
shl  al , 1    ;; Chapga bir xona siljitamiz: CF = 1, AL = 10010b.
shr  al , 1    ;; O'ngga bir xona siljitish: CF = 0, AL = 1001b.
shr  al , 1    ;; O'ngga bir xona siljitish: CF = 1, AL = 100b.
mov  al , 1000_1001b
shl  al , 2    ;; Chapga ikki xona siljitish: CF = 0,AL = 100100b.
mov  cl , 3
shr  al , cl   ;; O'ngga bir xona siljitish: CF = 1, AL = 100b.
```

Sizda bu buyruqlar nimaga kerak degan savol tug'ilganligi tabiiy. Biz kundalik hayotimizda sonlarni 10 ga oson tarzda ko'paytira olamiz yoki bo'la olamiz. Ko'paytirishda faqat son oxiriga nolni qo'yish kifoya. Ikkilik sanoq tizimida ham sonni osha sanoq tizimi asosiga, ya'ni 2 ga, ko'paytirganda faqat son oxiriga nol qo'shib qo'yiladi: $(10101)_2 * (10)_2 = (101010)_2$, ya'ni 21*2=42. Demak, o'zgaruvchi qiymati chapga n xona siljitilganda uning qiymati 2^n ga ko'paytirilar ekan. O'ngga n xona siljitganda esa qiymat 2^n ga bo'linadi. Siljitish buyruqlari DIV va MUL ga qaraganda ancha tez ishlaydi. Bo'luvchi 2 ga karrali bo'lganda siljitish buyruqlaridan foydalangan ma'qul. Lekin ishorali sonlar bilan ish olib borganda natija noto'g'ri bo'ladi. Chunki ishora biti ham siljitiladi.

7.1.2. Arifmetik siljitish

Yuqorida bo'lgan ishorali sonlar muammosi arifmetik siljitish buyruqlari orqali yechiladi. Arifmetik deb nomlanishi ham shundan. Mantiqiy siljitishda bitlar oddiy axborot sifatida qabul qilingan bo'lsa, bu buyruqlarda axborotga arifmetik yondashiladi. Arifmetik siljitish SAL (**S**hift **A**rithmetically **L**eft) va SAR (**S**hift **A**rithmetically **R**ight) buyruqlar orqali amalga oshiriladi.

```
sal  maqsad , siljish_miqdori
```

SHL dan farqli o'laroq ishora bitiga tegmaydi.

```
sar  maqsad , siljish_miqdori
```

SHR dan farqli o'laroq chap tomondan nollar emas, balki ishora biti qiymatiga teng qiymatlar kirgiziladi. Masalan, manfiy son bilan ishlaganda chapdan 1 kiritiladi.

Misol tariqasida -16 ni 4 ga bo'lib ko'ramiz:

```
mov  eax , -16
sar  eax , 2   ;; Ikki xona o'ngga siljitish, ya'ni -16 / 2² = -4.
```

Bu kabi siljitishlar juda xilma-xil maqsadlarda ishlatiladi. Ba'zan bizni sonning ikkilik ko'rinishidagi birga teng bo'lgan bitlar miqdori qiziqtiradi. Bunda siljitish buyruqlari qo'l keladi. Misol tariqasida EAX dagi qiymat ikkilik ko'rinishining nechta biri bor ekanini hisoblash dasturini ko'rib chiqamiz.

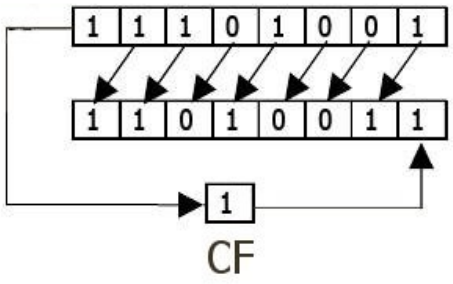
```
(1)     mov  bl , 0
(2)     mov  ecx , 32
(3)  loop_boshi:
(4)     shl  eax , 1
(5)     adc  bl , 0          ;; BL = BL+0+CF
(6)     loop loop_boshi
```

Bu yerda biz birlar yig'indisini BL qism registrida saqlaymiz. ECX ga 32 ni o'zlashtirishimizning boisi EAX 32 bitdan iborat va LOOP buyrug'i 32 marta qaytalanishi kerak. Har bir takrorlanishda EAX ning bitta biti CF bayrog'iga o'tkaziladi. CF bayrog'ining qiymatini ADC amali orqali BL ga qo'shamiz.

7.1.3. Halqasimon siljitish

Bu turdagi buyruqlarning oldingi siljitish buyruqlaridan farqi shundaki, chiqib ketayotgan bit yangi bit sifatida boshqa tomondan kirib keladi. Halqasimon deb atalishining sababi ham shunda. Ipga munchoqlarni o'tkazib uni halqa qilib bog'langan holatini ko'z oldingizga keltiring. Agar munchoqlarni bitlar deb hisoblasak, juda o'xshash manzara yuzaga keladi. Eng chetdagi munchoqni ip bo'ylab sursak, u halqa bo'ylab aylanib kelib boshqa tomondan qo'shiladi.

Halqasimon siljitish buyruqlarida ham chiqib ketayotgan bit qiymati CF bayrog'iga o'zlashtiriladi. Rasmda bir bayt axborotni chap tomonga bir xona halqasimon siljitish tasvirlangan.



23-rasm.

Bu turdagi siljitishlarni biz ROL (**Ro**tate **L**eft) va ROR (**Ro**tate **R**ight) buyruqlari orqali amalga oshiramiz.

```
rol  maqsad , siljish_miqdori
ror  maqsad , siljish_miqdori
```

Misollar:

```
mov  al , 1000_1001b
rol  al , 1          ;; AL = 10011b, CF = 1
rol  al , 1          ;; AL = 100110b, CF = 0
rol  al , 1          ;; AL = 1001100b, CF = 0
ror  al , 2          ;; AL = 10011b, CF = 0
ror  al , 1          ;; AL = 1000_1001b, CF = 1
```

Avvaliga AL bitlari uch xona chapga, keyin esa qayta uch xona o'ngga siljitildi. Natijada AL ning qiymati o'zgarmay qoldi. Bu holatni oldingi siljitish buyruqlari bilan solishtirib ko'ring. Masalan, halqasimon siljitish bizga 32 bitli registrning to'ng'ich 16 biti va kenja 16 biti o'rnini almashtirish imkoniyatini beradi:

```
rol  eax , 16
```

Qo'shimcha tarzda `RCL` (**R**otate with **C**F **L**eft) va `RCR` (**R**otate with **C**F **R**ight) buyruqlari mavjud bo'lib, ular halqasimon siljitishda CF buyrug'ini ham qo'shib siljitishadi. Masalan AX ni shu buyruqlar orqali siljitsangiz AX va CF bitlari siljitiladi. Kirayotgan bit CF bayrog'i qiymatiga teng bo'ladi, chiqayotgan bit esa CF bayrog'iga o'zlashtiriladi:

```
rcl  eax , 1         ;; EAX va CF bir xona chapga siljitiladi.
```

Biz bu buyruqlardan ko'pgina maqsadlarda foydalanishimiz mumkin. Masalan, EAX ning to'ng'ich 16 bitini BX ga o'tkazish kerak bo'lsin. Agar EAX ning kenja 16 bitini o'tkazish kerak bo'lganda bu juda oson bo'lar edi, ya'ni AX orqali:

```
mov  bx , ax
```

Lekin EAX ning to'ng'ich 16 bitini beradigan qism registri yo'q. Bu masala `RCL` buyrug'i orqali hal qilinadi:

```
     mov  ecx , 16       ;; takrorlanish miqdori
loop_boshi:
     rcl  eax , 1        ;; EAX ning to'ng'ich biti CF ga yuklanadi
     rcl  bx , 1         ;; BX ning kenja bitiga CF yuklanadi
     loop loop_boshi     ;; LOOP 16 marta takrorlanadi
```

«Arifmetik siljitish» mavzusida biz qanday qilib o'zgaruvchidagi qiymati birga teng bo'lgan bitlar sonini hisoblash dasturini tuzgan edik. Ammo e'tibor bergan bo'lsangiz u yerda EAX ning bitlari sanalib bo'lgandan so'ng, uning qiymati nolga teng bo'lib qolgan edi, ya'ni uning avvalgi qiymati yo'qotilgan edi. Chunki EAX ga o'ng tomondan 32 marta nol kirgiziladi. Agar birga teng bitlar sonini hisoblamoqchi bo'lsangiz va shu bilan birga EAX ning qiymatini saqlab qolmoqchi bo'lsangiz, `ROL` yoki `ROR` buyruqlaridan foydalaning:

```
mov  bl , 0
mov  ecx , 32
loop_boshi:
rol  eax , 1
adc  bl , 0
loop loop_boshi
```

7.2. Dasturlashda siljitish buyruqlaridan foydalanish

Oliy matemetikada siljitish yo'li bilan oson yechilishli mumkin bo'lgan masalalar juda ko'p. O'tilgan mavzularni mustahkamlash uchun shunday masalalarning birini assemblerda yechishini ko'rib chiqamiz.

Diskret matematikada *Heming masofasi* degan tushuncha bor. Bu ikki son o'rtasidagi masofa sifatida tushunilib, Heming algoritmi yordamida topiladi. Axborotlar katta-katta ma'lumotlar omborida saqlanganida sonlarni bunday o'zaro munosabati ularni tizimga solishda katta yordam beradi.

Heming algoritmi bo'yicha ikki son o'rtasidagi masofa ularning farq qiladigan mos bitlari soniga teng. Demak, algoritmni qo'llash uchun birinchi navbatda bu ikki sonning ikkilik ko'rinishi topilishi kerak. So'ng bu ikkilik sonlarni tagma-tag qo'yib mos xonalardagi farqli bitlar soni sanab chiqiladi. Masalan 32 va 8 o'rtasidagi masofa 2 ga teng:

00**1**0 **0**000 = 32
00**0**0 **1**000 = 8

3- va 5-bitlar o'zaro teng emas.

Biz ko'rib chiqadigan dastur berilgan ikki son o'rtasidagi masofani topishi kerak. Dasturda qo'llaniladigan usul shundan iboratki, sonlarni navbatma-navbat siljitamiz va siljib chiqqan bitlarni qo'shamiz. Agar yig'indi birga teng bo'lsa, demak, bitlar har xil va bu bitlarni hisobga olamiz. Dasturda farqli bitlar sonini DL qism registrida saqlab boramiz.

5-namuna: Heming masofasi.

```
(1)  ;; Maqsad:
(2)  ;;     Ushbu dastur beriladigan ikkita sonni qabul
(3)  ;;     qilib, ular o'rtasidagi masofani Heming algoritmi
(4)  ;;     bo'yicha hisoblaydi.
(5)  ;;
(6)  ;; O'zgaruvchilar:
(7)  ;;     son1 va son2 - ikki son, DL - masofa.
(8)
(9)  %include "nasm-io.inc"
(10)
(11) section     .bss
(12) son1   resd 1
(13) son2   resd 1
(14)
(15) section .text
(16) tizim_global     main
(17)
(18) main:
(19)   chop_et    `Ikkita son kiriting: `
(20)   qabul_qil `%i %i`, son1, son2
(21)   mov  edx , 0
(22)   mov  ecx , 32
(23)
(24) loop_boshi:
(25)   clc
(26)   mov  bl , 0
(27)   rol  dword[son1] , 1
(28)   adc  bl , 0
(29)   rol  dword [son2] , 1
```

```
(30)    adc  bl , 0
(31)    cmp  bl , 1
(32)    jnz  bir_xil
(33)    add  dl , bl
(34) bir_xil:
(35)    loop loop_boshi
(36)
(37)    chop_et   `%i va %i o'rtasidagi masofa %i ga teng \n`, \
(38)    [son1], [son2], edx
(39)
(40) ret
```

12, 13-qatorlarda beriladigan ikki son uchun 4 baytli o'zgaruvchilar e'lon qilamiz. 21-qatorda DL qism registrida bo'lgan oldingi qiymatni nolga aylantiramiz. Aks holda EDX dagi oldingi, ma'noga ega bo'lmagan axborotlar natijaga qo'shilib ketadi. O'zgaruvchilar 32 bitli bo'lgani uchun ularni to'liq siljitish 32 marta takrorlanishni talab qiladi. Shuning uchun ECX ga 32 ni o'zlashtiramiz. Takrorlanish halqasi ichidagi 25-qatorda CF bayrog'i qiymati har safar `CLC` buyrug'i orqali nolga tenglashtiriladi. Har takrorlanish boshida BL qism registri nolga teng bo'lishi shart. Chunki 28 va 30-qatorlarda unga CF bayrog'ini qo'shganda, oldingi takrorlanishlardagi qiymatlar bo'lmasligi kerak. 31-qatorda ikki mos bit yig'indisi 1 bilan taqqoslangan. Agar BL 1 ga teng bo'lmasa, demak, bitlar bir xil va bu bitlarni hisobga oladigan 33-qatorni chetlab o'tamiz.

O'zingizni sinash uchun dasturni o'zgartirib ko'ring. Bu safar dasturga bitta son va masofa berilsin, keyin yana uchta son berilsin. Dastur birinchi berilgan son bilan o'rtadagi masofasi berilgan masofaga teng bo'lgan sonlarni keyin berilgan sonlar ichidan topishi kerak.

7.3. Bitlar ustida mantiqiy amallar

O'zgaruvchi qiymatlari ustida amallar bajaradigan `ADD`, `SUB` yoki `DIV` kabi buyruqlar bilan bir qatorda o'zgaruvchining har bir biti ustida amal bajaradigan buyruqlar ham mavjud. Bunday amallar mantiqiy amallar deb yuritilishining boisi ularning haqiqat (1) va yolg'on (0) nisbati ustida ish olib borishidir.

7.3.1. AND amali

Mantiqiy ko'paytirish nomini olgan mazkur amal natijasi ikki bit ko'paytmasi bo'ladi. Agar ikkala bit ham 1 ga teng bo'lsa, natija 1 ga teng bo'ladi. Aks holda natija 0 ga teng. Quyida barcha holatlar uchun amal qanday bajarilishi keltirilgan:

$$X \ and \ Y$$
$$0 * 0 = 0$$
$$0 * 1 = 0$$
$$1 * 0 = 0$$
$$1 * 1 = 1$$

Misol tariqasida ikkita bir baytli axborotni `AND` amaliga berib ko'ramiz:

	1	0	1	1	1	0	1	0
AND								
	0	0	0	1	0	1	1	0
=								
	0	0	0	1	0	0	1	0

Bu amal mantiqiy ma'nosi jihatdan «va» so'z bog'lovchisi ma'nosini beradi. Agar uyingizda televizor borligi haqiqat bo'lsa (1) «va» elektr toki borligi haqiqat bo'lsa (1), u holda televizor ko'rishingiz ham haqiqat bo'ladi (1). Aks holda agar shulardan birortasi yolg'on bo'lsa (0) televizor ko'rishingiz ham yolg'on bo'ladi (0).

```
and  maqsad , manba
```

Bu yerda `maqsad` va `manba` o'rtasida mantiqiy ko'paytirish amalga oshiriladi va natija `maqsad` ga yoziladi. Shuni anglash muhimki, `AND` amali o'zgaruvchining mos bitlari ustida alohida-alohida mantiqiy ko'paytirishni bajaradi. Masalan, EAX=4 bo'lsin.

Oddiy ko'paytirish:

```
mul  3              ;; Shundan so'ng, EAX = 12
```

Mantiqiy ko'paytirish:

```
and  eax , 3        ;; Shundan so'ng, EAX = 0
```

Misollar:

```
mov  ax , 0xE5A3
and  ax , 0x9FE2    ;; AX = 0x85A2
```

7.3.2. OR amali

Mantiqiy qo'shish nomi bilan mashhur bo'lgan ushbu amalda agar qo'shiluvchilardan birortasi birga teng bo'lsa, u holda natija ham birga teng bo'ladi. Protsessorda ushbu maqsadda yaratilgan `OR` buyrug'i bo'lib, u berilgan o'zgaruvchilarning har bir biti ustida mazkur amalni bajaradi.

X	or	Y
0	+	0 = 0
0	+	1 = 1
1	+	0 = 1
1	+	1 = 1

1+1 ning yig'indisi ham 1 ga teng bo'lishining sababi shundaki, mantiqiy amallar bir bitli qiymatlar ustida bajarilayapti va bu yerda eng katta son birdir.

```
or   maqsad , manba
```

Misol tariqasida ikkita bir baytli axborotni `OR` amaliga berib ko'ramiz:

	1	0	1	1	1	0	1	0
OR								
	0	0	0	1	0	1	1	0
=								
	1	0	1	1	1	1	1	0

Misollar:

```
mov  ax , 0xE5A3
or   ax , 0x9FE2    ;; AX = 0xFFE3
```

7.3.3. XOR amali

Bu amal ingliz tilida *exclusive OR* deb yuritiladi. Uni biz o'zbek tiliga *istisnoli* qo'shish deb tarjima qilishimiz mumkin. Agar qo'shiluvchilar bir xil bo'lsa natija 0 ga teng, aks holda 1 ga teng.

X	*xor*	*Y*		
0	⊕	0	=	0
0	⊕	1	=	1
1	⊕	0	=	1
1	⊕	1	=	0

Qoidasi:

```
xor  maqsad , manba
```

Payqagan bo'lsangiz bu amal orqali farqli bitlarni aniqlash juda qulay. «Dasturlashda siljitish buyruqlaridan foydalanish» mavzusida ko'rib chiqilgan Heming masofasi bu amal orqali juda oson bajariladi. Buning uchun ikki son istisnoli qo'shiladi, so'ngra natijadagi birga teng bitlar «Arifmetik sijitish» mavzusida ko'rib chiqilgani singari sanab chiqiladi:

```
(1)    mov  eax , [son1]
(2)    xor  eax , [son2]
(3)    mov  ecx , 32
(4)  loop_boshi:
(5)    shl  eax , 1
(6)    adc  bl , 0
(7)    loop loop_boshi
```

XOR amalidan dasturlashda ba'zan «ayyorona» foydalaniladi. Aytib o'tganimizdek bitlar bilan ishlaydigan buyruqlar boshqa buyruqlardan ancha tez ishlaydi. Endi XOR ga ikkita qo'shiluvchi sifatida bitta o'zgaruvchini qayta berishni o'ylab ko'ring:

```
xor  eax , eax
```

Bu amaldan song EAX qiymati nolga teng bo'lib qoladi. Lekin bu buyruq

```
mov  eax , 0
```

dan ko'p marta tez ishlaydi.

7.3.4. NOT amali

Mazkur amal inkor buyrug'i nomi bilan tanilgan. Boshqa «do'stlaridan» farqli olaroq bitta qiymat oladi va uning har bir biti qiymatini teskarisiga aylantiradi.

not 1 = 0
not 0 = 1

Qoidasi:

```
not  maqsad
```

NOT amali aslini olganda birlamchi to'ldiruvchini aniqlab beradi.
Misollar:

```
mov  al , 10111011b
not  al                 ;; AL = 01000100b
```

7.3.5. TEST buyrug'i

AND amali bilan bir xil, lekin natija hech qayerda saqlanmaydi, faqat bayroqlar o'rnatiladi xolos. AND va TEST buyruqlari farqi xuddi yuqorida ko'rib o'tilgan SUB va CMP buyruqlari farqiga o'xshaydi. TEST ning natijasini ZF bayrog'i orqali aniqlash mumkin. Agar ZF=1 bo'lsa, demak, natija nolga teng bo'lgan, ya'ni ikkala o'zgaruvchida birga teng bo'lgan mos bitlar chiqmagan.

TEST buyrug'ini ham XOR ga o'xshab «ayyorona» ishlatish mumkin. Agar ikkita qiymat o'rniga ham bitta o'zgaruvchi berilsa va bu o'zgaruvchi qiymati nolga teng bo'lsa, u holda ZF bayrog'i o'rnatiladi. Sonni nol bilan taqqoslash kerak bo'lsa:

```
cmp  eax , 0
```

dan ko'ra

```
test eax , eax
```

buyrug'i ancha tezroq ishlaydi.

7.4. Mavzular yuzasidan mashqlar

Dasturlashda mantiqiy amallardan samarali foydalanish uchun o'zgaruvchining qiymati yaxlit son sifatida emas, balki bitlar ketma-ketligi deb tasavvur qilish zarur. Ba'zan o'zgaruvchining ma'lum bir xonasidagi bitning qiymatini bilish yoki uni o'zgartirish kerak bo'lib qoladi. Mantiqiy amallar esa bunday hollarda ayni muddao bo'ladi.

Ma'lum bir xonadagi bitning qiymatini 1 ga teng qilib qo'yishda OR amalidan foydalaniladi:

```
or   edx , 1000b    ;; EDX ning 3-bitini 1 ga tenglashtiradi
or   ecx , 8        ;; ECX ning 3-bitini 1 ga tenglashtiradi
or   ax , 0xF00     ;; AX ning 8-bitidan boshlab 4 ta
                    ;; bitini 1 ga tenglashtiradi
```

Ma'lum bir xonadagi bitning qiymatini 0 ga tenglashtirish uchun AND amalidan foydalanamiz:

```
and  eax , 0xFFFFFFF7         ;; 3-bitni 0 ga aylantiradi
and  eax , 0xFFFF0FFF         ;; 12-bitdan boshlab 4 ta bitni
                              ;; qiymatini 0 ga tenglashtiradi.
```

Ma'lum xonadagi bitlarni teskarisiga ogirishda XOR amali qo'l ketadi:

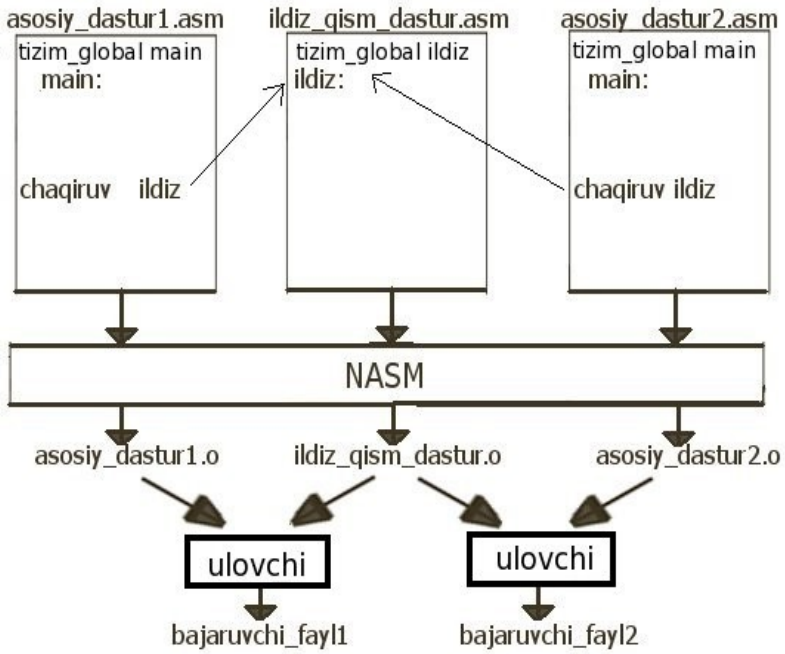
```
xor  bl , 100b          ;; BL dagi 2-bitni teskarisiga o'tkazish
xor  eax , 0xFFFFFFFF   ;; EAX dagi qiymatning birlamchi
                        ;; to'ldiruvchisini topish
```

AND amali orqali sonni 2^n ga bo'lganda hosil bo'ladigan qoldig'ini hisoblash mumkin. Bunda bo'luvchi 2^n-1 ko'rinishida AND ga ikkinchi qiymat qilib beriladi. Birinchi qiymatda esa bo'linuvchi bo'lishi kerak. Masalan 40 ni 16, ya'ni 4^2 ga bo'lganda hosil bo'ladigan qoldiqni topish uchun:

```
mov  eax , 40
mov  ebx , 15       ;; 15 = 2^4 - 1
and  eax , ebx      ;; EAX = 8
```

VIII bob
Qismli dasturlash

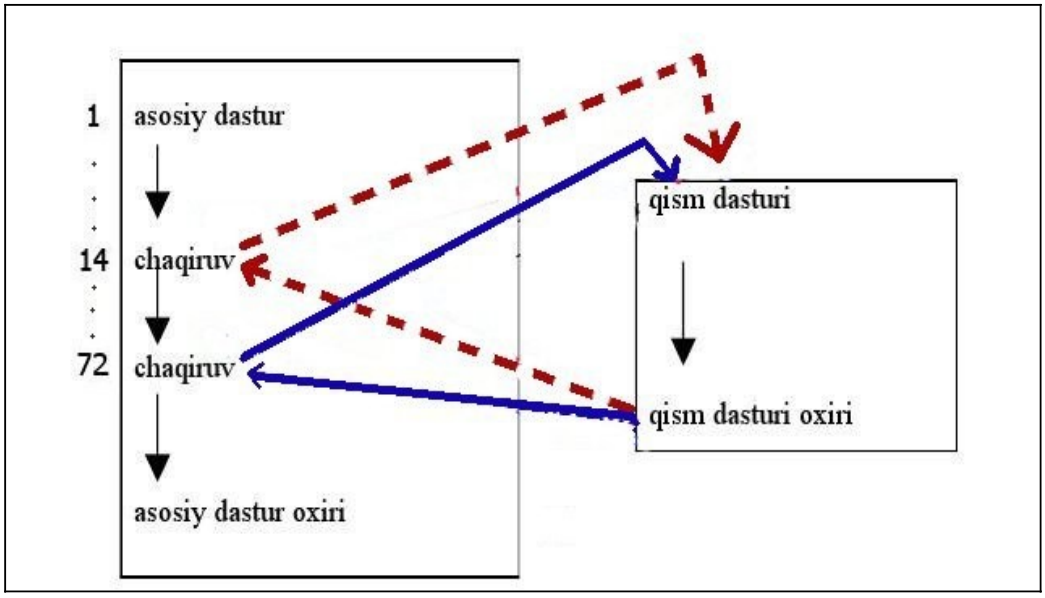
Dasturlashning asosiy tushunchalaridan yana biri qismli dasturlashdir. Takrorlanish va tarmoqlash tushunchalari bilan bir qatorda turuvchi ushbu jarayon dasturni qism-qism qilib tuzishga asoslangan. Bunda dasturning ko'p marta bajarilishi kerak bo'lgan qismini alohida mayda dastur sifatida tuziladi va shu dastur bo'lagiga istalgan paytda va istalgan joydan turib murojaat qilinadi. Masalan, assemblerda butun sonlar bilan ishlaydigan buyruqlar ichida sonning kvadrat ildizini hisoblaydigani yo'q. Agar shunday dasturni alohida faylda tuzsangiz, undan qism dasturi sifatida foydalanishingiz mumkin. Ildizni hisoblash dasturingizdan nafaqat o'zingiz, balki do'stlaringiz ham foydalanishlari mumkin. Buning uchun asosiy va qism dasturi alohida NASM orqali yig'iladi, keyin esa ulovchi dastur orqali ulanadi[1]. Bu holat rasmda yaqqol tasvirlangan. Ikkita alohida asosiy dasturning bitta ildizni hisoblovchisi qism dasturidan foydalanishi ko'rsatilgan. Buning aynan qanday amalga oshirilishi keyingi mavzularda ko'rib chiqiladi.



24-rasm.

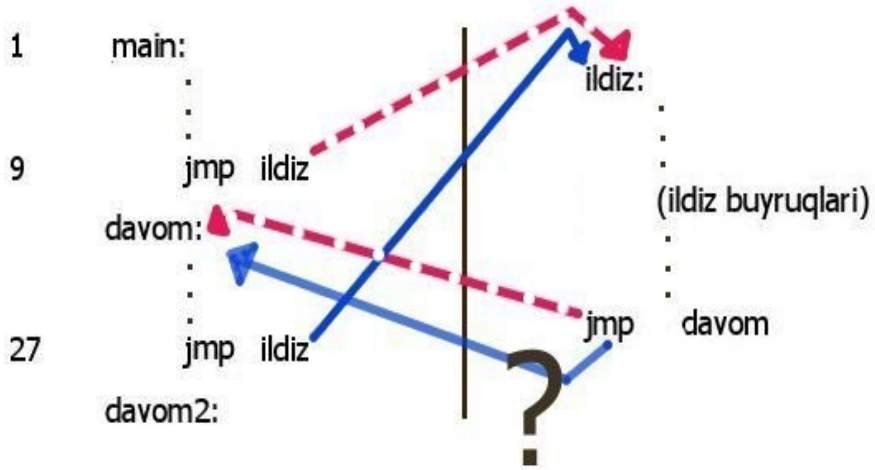
Qism dasturning ishlash usuli shunga asoslanganki, unga murojaat qilinganda, ya'ni chaqirilganda asosiy dastur buyruqlari bajarilishi chaqiruv amalga oshirilgan joydan to'xtatiladi va boshqaruv qism dasturi buyruqlariga topshiriladi. Qism dasturi bajarilib bo'lgach, boshqaruv yana asosiy dasturdan chaqiruv bo'lgan joyga qaytariladi, ya'ni qism dasturning oxirgi buyrug'i bajarilgach asosiy dasturdagi qolgan buyruqlar bajarilishi davom etadi. Qism dasturi qayerdan turib chaqirilmasin, u bajarilib bo'lgach, boshqaruv yana o'sha chaqiriq amalga oshirilgan joyga qaytariladi.

[1] «Obyekt fayllarni ulash» mavzusiga qarang.



25-rasm.

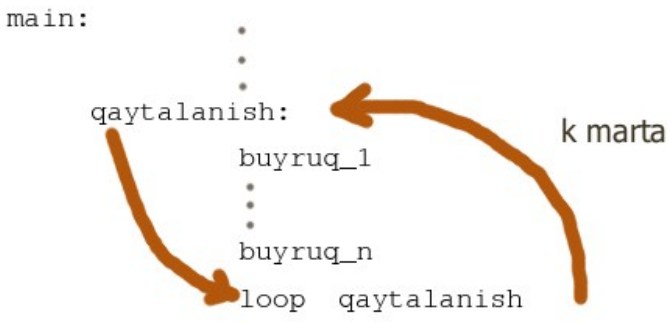
Rasmda ko'rsatilganidek, asosiy dasturning 14- va 72-qatorida chaqiruv amalga oshirilayapti. Bunda chaqiruv amalga oshirilgan joyda asosiy dastur to qism dasturi bajarilib bo'lguncha to'xtatib qo'yiladi va shundan keyin asosiy dastur chaqiruv amalga oshirilgan joydan davom etaveradi. Lekin qismli dasturlashni oldin ko'rib o'tilgan takrorlanish yoki tarmoqlanish tushunchalari bilan chalkashtirmaslik kerak. Masalan, tarmoqlash buyruqlari orqali qism dasturiga o'tishni amalga oshirganimiz bilan chaqirilgan joyga qaytib borish imkoni bo'lmaydi. Chunki o'tish buyruqlariga doim aniq manzil ko'rsatiladi. Ammo qism dasturi so'ngida har safar aniq bitta manzilga emas, balki qayerdan chaqirilgan bo'lsa o'sha yer manzilga qaytiladi (26-rasmga qarang).



26-rasm.

Rasmda ko'rinib turibdiki, ildiz 9-qatordan chaqirilganda qaytish `davom` nishoniga bo'ladi. Bu albatta yaxshi, chunki 9-qatordan keyingi buyruqlar shu nishondan boshlanadi. Ammo chaqiruv 27-qatordan amalga oshirilganda qism dasturidan `davom2` nishoniga o'tish yuz bermaydi. Chunki qism dasturi oxirida qat'iy qilib `jmp davom` yozib qo'yilgan.

Takrorlanishda esa dasturning ma'lum bir qismi berilgan son marta qayta-qayta takrorlanadi (27-rasmga qarang). Misolda `buyrug_1` dan `buyruq_n` gacha bo'lgan buyruqlar k marta takrorlanadi.



27-rasm.

8.1. Vositali manzillash

Manzillar bilan ishlash o'tgan mavzularda ancha keng yoritilgan bo'lsada, ba'zi muhim tomonlarni yana bir bor ko'rib chiqishimizga to'g'ri keladi. Ma'lumki, registrlar axborot saqlash uchun qo'llanilib, ular hech qanday manzilga ega emas. Ajratilgan xotira uchun ko'rsatgich qilib o'rnatiladigan nishonlar esa manzillarga ega, ya'ni nishon o'zi ko'rsatgich bo'lib turgan xotira bo'lagining manzilidir. Nishon manzilida saqlanayotgan qiymatga murojaat qilish uchun esa nishon burchakli qavs ichiga olinadi. Masalan, `son` o'zgaruvchisida 194 saqlanayapti deylik, u holda:

```
mov  eax , son          ;; EAX = 0XFF046A45
mov  eax , [son]        ;; EAX = 194
```

Birinchi buyruqda EAX ga `son` o'zgaruvchisining manzili yuklanadi. Ikkinchi buyruqda esa `son` o'zgaruvchisida saqlanayotgan qiymat EAX ga yuklanadi. Assemblerda registrlar orqali ham manzillar bilan ishlash imkoniyati bor. Buning uchun registr burchakli qavsga olinadi. Bunday hollarda assembler registr qiymatiga xotiradagi joy manzili sifatida qaraydi. Lekin registrga qanday munosabatda bo'lish, unga nisbatan qo'llanilayotgan buyruqqa ham bog'liq. Masalan, `JMP` buyrug'iga o'tish manzili sifatida registr burchakli qavslarsiz berilsa ham, u registr qiymatini manzil sifatida qabul qilib, o'sha manzilga o'tishni amalga oshiradi.

Keling biz `son` o'zgaruvchisiga 5 ni o'zlashtirmoqchi bo'laylik. Buning uchun osongina qilib `MOV` buyrug'idan foydalanamiz:

```
mov  dword [son] , 5
```

Endi esa bu ishni EAX orqali amalga oshiramiz, ya'ni `son` o'zgaruvchisining manzilini unga yuklab, 5 sonini EAX orqali o'zlashtiramiz:

```
mov  eax , son          ;; O'zgaruvchi manzilini yuklaymiz.
mov  dword [eax] , 5    ;; son = 5
```

Shundan keyin `son` o'zgaruvchisida 5 soni bo'lib qoladi.

Murakkab manzillarni hisoblashda burchakli qavs ichida bir nechta hadlar orasida (+) yoki (-) belgilarini va hadni ko'paytma sifatida (*) belgisi orqali ifodalash mumkin. Buning qulaylik tomoni shundaki, bu amallarni `ADD`, `SUB`, `MUL` buyruqlari yordamida bajarib o'tirmaysiz. Masalan, EAX dagi manzildan 8 bayt keyingi manzilda joylashgan to'rt baytli qiymatni EBX ga o'zlashtirmoqchi bo'ldingiz deylik:

```
mov  ebx , [eax+8]
```

Bu yerda EBX to'rt baytli registr ekanligi aniq bo'lgani uchun `dword` kalit so'zidan foydalanmaymiz. Agar, masalan, EDX registrida 4 soni bo'lsa, yuqoridagi ifodani quyidagicha yozish ham mumkin:

```
mov  ebx , [eax + edx*2]
```

Ifodani yanada murakkablashtirib EAX dan 7 bayt keyingi bir baytni BL ga yuklaymiz:

```
mov  bl , [eax + edx*2 - 1]
```

Manzillarni barcha 32 bitli umumiy foydalanishdagi registrlarda va EDI , ESI registrlarida saqlash mumkin. Manzillar 32 bitli kompyuterda 4 bayt kattalikda saqlanadi.

Burchakli qavs ichida beriladigan foydali manzil andozasi haqida to'liq ma'lumot uchun «`LEA` buyrug'i» mavzusiga qarang.

8.2. Qismli dastur tuzish

Qism dasturi o'zining kod va qiymatlar segmentiga ega bo'lishi mumkin yoki boshqa kod segmenti ichida ham bo'lishi mumkin. Qism dasturi boshlanadigan yerga oddiy nishon qo'yiladi va bu nishon shu qism dasturi nomi bo'lib xizmat qiladi. Qism dasturi faylda asosiy dasturdan oldin yoki asosiy dasturdan keyin pastda yozilishi mumkin.

Birinchi ko'rinish:

```
;; Qism dasturi
segment .data             ;; Qism dastur qiymatlari
…
segment .bss
…
segment .text
qism_dastur_nishoni:
…
;; Qism dastur kodi
…
ret                       ;; Qism dasturi oxiri
;; Asosiy dastur
segment .data
…
segment .bss
…
segment .text
global    main:
main:
…
;; qism dasturini chaqirish
…
ret                       ;; Asosiy dasturi oxiri
```

Ikkinchi ko'rinish:

```
;; Asosiy dastur
segment .data
…
segment .bss
…
```

```
      segment .text
      global    main:
      main:
      …
      ;; qism dasturini chaqirish
      …
      ret
      ;; Qism dasturi
      segment .data
      …
      segment .bss
      …
      segment .text
      qism_dastur_nishoni:
      …
      ;; qism dastur kodi
      …
      ret                       ;; Qism dasturi oxiri
```

Qism dasturi asosiy dastur kodi ichida ham joylashtirilishi mumkin. Ammo bunda qism dasturi buyruqlari asosiy dastur buyruqlari bilan aralashib ketmasligi uchun qism dasturi o'rin olgan joy ustidan shartsiz o'tish amalga oshirilishi kerak.

Uchinchi ko'rinish:

```
      …
      tizim_global main
main:
      …
      jmp  chetlash             ;; Qism dasturni chetlab o'tamiz.
      ;; qiymat segmentlari
      qism dasturi:             ;; Qism dasturi boshi.
      …
      ret
chetlash:                       ;; Asosiy dastur davomi.
      …
```

Biz qism dasturlarini asosiy dasturdan pastda joylashtirishni maslahat beramiz.

Endilikda haqiqiy ishlaydigan ildizni hisoblovchi qism dasturini tuzishga harakat qilib ko'rsak bo'ladi. Asosiy dastur berilgan ikki sonining kvadrat ildizlarining yig'indisini hisoblashi kerak bo'lsin. Demak, asosiy dastur har bir sonning kvadrat ildizini hisoblash uchun qism dasturiga ikki marta murojaat qiladi. Qism dasturdan foydalanishda yana bir masala qiymatlarni unga qanday jo'natish muammosidir. Kvadrat ildizi topilishi kerak bo'lgan sonlar qism dasturiga qanday yetkaziladi? Buning yechimi sifatida registrlardan foydalaniladi. Qism dasturini chaqirishdan oldin unga yetkazmoqchi bo'lgan qiymatlarni ma'lum bir registrga o'zlashtiramiz. Qism dasturiga o'tilganida esa xuddi shu registr qiymati jo'natilgan qiymat sifatida qabul qilinadi. Qism dasturi o'z hisob-kitoblarini bajarib bo'lgach, asosiy dasturga qaytishni amalga oshirish bilan bir qatorda natijani ham jo'natishi kerak. Bunda ham registrlardan foydalanish mumkin. Qism dasturi chaqirilishi oldidan unga qaytish manzilini ham jo'natish lozim. Qaytish manzili bu qism dasturga o'tish buyrug'idan keyin keladigan qatorning kod segmentidagi manzilidir. Bu joyning manzilini hisoblab o'tirmasdan, shunchaki o'sha yerga nishon qo'yamiz.

6-namuna: $\sqrt{a}+\sqrt{b}$ *ni hisoblaymiz.*

```
(1)  ;; Asosiy dastur.
(2)  ;;
(3)  ;; Maqsad:
(4)  ;;   Ikki sonning kvadrat ildizining yig'indisini hisoblaydi.
(5)  ;;   Ildiz qism dasturidan foydalanadi.
(6)  ;;
(7)  ;; O'zgaruvchilar:
(8)  ;;   a va b - ikkita qo'shiluvchi, EDI - yig'indi.
(9)
(10) %include "nasm-io.inc"
(11)
(12) section .bss
(13) son1 resd 1
(14) son2 resd 1
(15)
(16) section .text
(17) tizim_global   main
(18)
(19) main:
(20)      chop_et    `Ikkita son kiriting : `
(21)      qabul_qil  `%i %i`, son1, son2
(22)      mov  ebx , [son1]        ;; Son ildiz qism dasturiga EBX da
(23)                               ;; jo'natalida.
(24)      mov  ecx , davom1   ;; Qaytish manzili ECX da jo'natiladi.
(25)      jmp  ildiz
(26)
(27) davom1:
(28)      mov  edi , eax           ;; son1 ning kvadrat ildizini EDI da
(29)                               ;; saqlaymiz.
(30)
(31)      mov  ebx , [son2]        ;; Son ildiz qism dasturiga EBX da
(32)                               ;; jo'natalida.
(33)      mov  ecx , davom2   ;; Qaytish manzili ECX da jo'natiladi.
(34)      jmp  ildiz
(35)
(36) davom2:
(37)      add  edi , eax
(38)      chop_et    `Javob = %i \n`, edi
(39)
(40) ret
(41)
(42) ;; Ildiz qism dasturi.
(43) ;;
(44) ;; Maqsad:
(45) ;;   Ushbu qism dasturi sonning kvadrat ildizini hisoblaydi.
(46) ;;
(47) ;; Qiymatlar:
(48) ;;   EBX registri orqali ildizi hisoblanish kerak bo'lgan
(49) ;;   son qabul qilinadi. ECX da qaytish manzili bo'lishi kerak.
(50) ;;
(51) ;; Natija:
(52) ;;   Hisoblangan kvadrat ildiz EAX da qaytariladi.
(53)
```

```
(54) section .bss
(55) xn   resd 1                    ;; xn ↔ x_n, n = {1, ..., n}
(56)
(57) section   .text
(58)
(59) ildiz:
(60)      chop_et   `Ildiz qism dasturi ishga tushdi \n`
(61)
(62)      mov  eax , ebx            ;; Jo'natilgan qiymatni EAX ga
(63)                                ;; o'zlashtiramiz, ya'ni EAX <- a.
(64)      mov  [xn] , eax
(65)      shr  dword [xn] , 1       ;; x_1 <- a / 2.
(66)
(67) takrorlanish:
(68)      xor  edx , edx            ;; EDX:EAX ning to'ng'ich 4 baytini
(69)                                ;; nollaymiz.
(70)      div  dword [xn]           ;; EAX <- a / x_n.
(71)      add  eax , [xn]           ;; EAX <- x_n + a/x_n.
(72)      shr  eax , 1              ;; EAX <- (x_n + a/x_n)/2, bu
(73)                                ;; yerda EAX = x_n+1.
(74)      cmp  eax , [xn]           ;; (EAX = x_n) ?
(75)      jz   topildi
(76)      mov  [xn] , eax           ;; x_n <- x_n+1.
(77)      mov  eax , ebx            ;; Jo'natilgan qiymatni tiklaymiz.
(78)      jmp  takrorlanish
(79)
(80) topildi:
(81)      jmp  ecx                  ;; Orqaga qaytamiz.
```

22-qatorda `son1` o'zgaruvchisining qiymati EBX registriga o'zlashtirilgan. Qism dasturida esa EBX registridagi manzil qiymati bilan ishlaymiz. 24-qatorda qaytish manzilini ECX ga o'zlashtiramiz. Qism dasturi bajarilib bo'lgach, ECX dagi qiymat manzil sifatida qabul qilinadi va o'sha nishonga o'tish amalga oshiriladi. 25-qatorda qism dasturiga shartsiz o'tamiz. 54-qatordan ildiz qism dasturi boshlanadi. Uning ham o'z o'zgaruvchilari hamda kod bo'limi bor. Agar qism dasturining `.bss` va/yoki `.data` bo'limlari bo'lmaganda edi, 57-qatordagi `.text` bo'limini e'lon qilmasak ham bo'lar edi. Assemblerda bo'lim nomi e'lon qilingandan so'ng undan pastdagi barcha kod to fayl oxirigacha yoki boshqa bo'lim boshlanmaguncha o'sha bo'limga tegishli hisoblanadi. 55-qatorda faqat qism dasturiga tegishli bo'lgan o'zgaruvchi e'lon qilingan. 59-qatordan esa `ildiz` qism dasturining kodi boshlangan. Qism dasturi chaqirilganda shu nishonga o'tish amalga oshiriladi. Ildiz qism dasturimiz berilgan sonning aniq kvadrat ildizini hisoblay olmaydi. Chunki hali assemblerda kasr sonlar bilan ishlaydigan buyruqlarni ko'rib chiqqanimiz yo'q. Shuning uchun kvadrat ildiz sifatida butun songacha yaxlitlangan qiymat olinadi. Masalan, qism dasturi 25 dan ham, 30 dan ham 5 sonini ildiz sifatida oladi.

Aynan qaysi usulda sonning kvadrat ildizi topilgani sizga tushunarli bo'lmasa kerak. Buni batafsilroq ko'rib chiqamiz. Elektron hisoblash mashinalari yordamida musbat a sonidan kvadrat ildiz chiqarishda $\{x_n\}$ rekurrent ketma-ketligidan[1] foydalaniladi. Ushbu ketma-ketlik quyidagi formula bilan aniqlanadi:

$$x_{n+1}=\frac{1}{2}\left(x_n+\frac{a}{x_n}\right),\ n=1,2,3\ldots \qquad (*)$$

[1] Rekurrent (lotincha *recurretis* – qaytuvchi) ketma-ketlik deb har bir hadi o'zidan avvalgi hadlar bilan aniqlanadigan ketma-ketlikka aytiladi. Ketma-ketlik haqida to'liq ma'lumot olish uchun oliy matematika kursiga qarang.

Bu yerda x_1 sifatida ixtiyoriy musbat son olinishi mumkin. Biz ildiz tezroq topilishi uchun x_1 ni a ning yarmiga teng qilib oldik, `shr dword [xn] , 1`. Kvadrat ildizni topishga mo'ljallangan (*) rekurrent formulani takror-takror qo'llash orqali ildiz topiladi. Birinchi qo'llanishda x_1 orqali x_2 topiladi, keyingi safar esa x_2 orqali x_3 topiladi va hokazo. Nechanchidir takrorlanishda x_n ketma-ketlik a'zosi a ning kvadrat ildiziga teng bo'lib qoladi. Shundan keyin formula necha marta takrorlansa ham navbatdagi x_{n+m} lar x_n ga taqriban teng bo'ladi:

$$x_n \approx x_{n+m} \qquad n,\ m = 1,\ 2,...,$$

Shuning uchun biz dasturdagi takrorlanishda doim x_{n+1} va x_n larni taqqoslab turamiz, 74-qator. Ular teng bo'lib qolganda esa `topildi` nishoniga o'tiladi. Dasturda qiymatlar butun songacha yaxlitlangani uchun ildiz topilganda x_{n+1} va x_n lar taqriban emas, balki aniq teng bo'ladi. 81-qatorda esa asosiy dasturga qaytish amalga oshiriladi. Ko'rib chiqilgan misolda asosiy dastur ham, qism dasturi ham bitta faylda ketma-ket joylashgan. Holbuki, qism dasturi alohida boshqa faylda joylashishi ham mumkin.

Eslatma: Dasturda barcha bo'lish amallarida bo'linmaning faqat butun qismi olingani uchun ildiz hisoblash algoritmi ba'zi hollarda ishlamaydi. Xususan, 1 son va kvadrat ildizi butun son bo'lgan sonlardan bitta oldingi sonlarda dastur ishlamay qoladi. Masalan: 24, 35, 48 va hokazo.

Agar asosiy dasturdan chaqirilishi kerak bo'lgan nishonlar boshqa fayllarda joylashgan bo'lsa, bu nishonlar asosiy dastur kodi boshida `extern` maxsus so'zi orqali e'lon qilinadi. Xuddi shu nishonlar o'zlari joylashgan faylda `global` maxsus so'zi bilan e'lon qilingan bo'lishlari shart. Assembler tilida barcha nishonlar odatda ichki, ya'ni faqat o'zi joylashgan fayl ichida ma'noga ega bo'ladi. `global` va `extern` maxsus so'zlari nishonning fayllararo ahamiyatga ega ekanligini bildiradi. Masalan, ildiz qism dasturimizni yangi *qism.asm* fayliga o'tkazdik deylik. Bu holda quyidagi o'zgartirishlarni kiritamiz:

```
...
section .text
extern  ildiz
tizim_global main

main:
      ...
      jmp ildiz
      ...
      ret
```

asosiy.asm

```
...
section .text
global  ildiz

ildiz:
      ...
      ;; Ildiz qism dasturi kodi
      ...
      jmp  ecx
```

qism.asm

Agar dastur bir emas, bir nechta qism dasturiga ega bo'lsa, barcha qism dasturlarini bitta faylga, asosiy dasturni esa boshqa faylda tuzgan ma'qul. Bunday hollarda `extern` va `global` so'zlaridan so'ng nishon nomlari vergul bilan ajratib beriladi:

```
...
section .text
extern ildiz, kub_ildiz
tizim_global main

main:
        ...
        jmp   ildiz
        ...
        jmp   kub_ildiz
        ...
        ret
```

asosiy.asm

```
...
section .text
global  ildiz, kub_ildiz

ildiz:
        ...
        jmp   qaytish_manzili

kub_ildiz:
        ...
        jmp   qaytish_manzili
```

qism.asm

Bu yerda asosiy dastur uchun `ildiz` va `kub_ildiz` nishonlari tashqi hisoblanadi. Yana bir narsaga e'tibor berish lozimki, *qism.asm* faylida nishonlar `global` orqali e'lon qilindi, `tizim_global` bilan emas! Umuman olganda `tizim_global` ga o'xshagan `tizim_extern` makro vositasi ham *nasm-io.inc* faylida mavjud bo'lib, undan qanday foydalanish «Assembler va C dasturlarini bog'lash» mavzusida keltirilgan.

8.3. Stack

Stack bu dasturlash davomida qiymatlarni saqlash mumkin bo'lgan xotira bo'lagi. Stack xotiraning ma'lum bir maydonidan o'rin oladigan narsa bo'lsada, protsessorda u bilan ishlashni ta'minlovchi qurilmalar bo'ladi. Stack o'zgaruvchilar singari dastur boshida e'lon qilinmaydi. Dasturchi undan xohlagan vaqtida foydalanishi mumkin. Stack asosan vaqtinchalik qiymatlarni saqlashda ishlatiladi. O'zgaruvchilardan asosiy farqi shundaki, stack uchun xotiradan joy faqat unga murojaat qilinganda ajratiladi. Bunday xotira ajratilishi *dinamik* xotira ajratilishi deyiladi. O'zgaruvchilar e'lon qilinganda dastur davomida ishlatilish yoki ishlatilmasligidan qat'i nazar ular uchun xotiradan joy ajratiladi. Bu turdagi joy ajratilishi *statik* joy ajratish deyiladi. Demak, qiymatlar segmenti har bir dastur uchun o'zgarmas kattalikka ega bo'ladi. Stack segmentining o'lchami esa aksincha undan foydalanish yoki foydalanmasligimizga qarab o'zgarib turadi. Stackda qiymatlar joylashtirilishi va u yerdan o'zgaruvchilarga yuklanishi mumkin. Stackda qiymatlar LIFO (**L**ast **I**n **F**irst **O**ut – Oxirgi kirgan birinchi chiqadi) qonuniyati bo'yicha ketma-ket joylashtiriladi. Oxirgi bo'lib joylashtirilgan qiymat, u yerdan olinishda birinchi bo'lib chiqadi. Bu holatni bir tomoni berk quvurga koptokchalarni joylashtirishga o'xshatish mumkin (rasmga qarang).



28-rasm.

Rasmdan ko'rinib turibdiki, quvurga oxirgi bo'lib B harfli koptokcha kiritilgan va navbatda kiritilish uchun C harfli koptokcha turibdi. U yerdan A harfli koptokchani chiqarib olish uchun esa

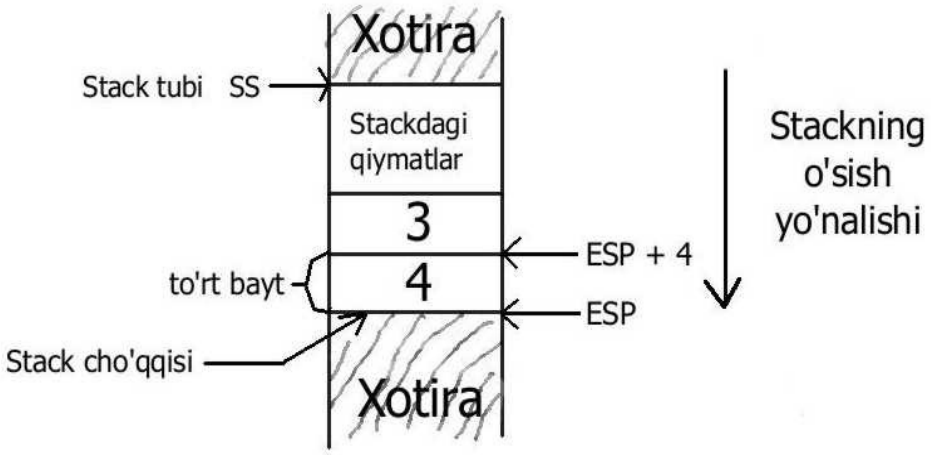
birinchi bo'lib B harfli koptokchani chiqarishga to'g'ri keladi. Stack ham xuddi shunday tuzilishga ega. Faqat u yerda koptokchalar o'rniga qiymatlar keladi.

Stackka qiymatlarni joylashtirish va u yerdan olish mos ravishda `PUSH` va `POP` buyruqlari orqali amalga oshiriladi. Qiymatlar o'lchami to'rt baytli bo'lgani ma'qul[1]. Stack segmenti boshlanish manzili SS registrida saqlanadi. Oxirgi bo'lib joylashtirilgan qiymat manzili ESP registrida saqlanadi. `PUSH` buyrug'i orqali yangi qiymat stackka joylashtirilganda, yangi axborot manzili ESP registridan axborot o'lchamiga teng sonni ayirish bilan hisoblanadi, ya'ni:

$$kiritilayotgan_qiymat_manzili = ESP - kiritilayotgan_qiymat_o'lchami$$

Masalan, 3 va 4 sonlarini stackka to'rt baytli qiymat sifatida kiritdik deylik:

```
push dword 3
push dword 4
```



29-rasm.

29-rasmdan ham ko'rinib turibdiki, `PUSH` buyrug'idan so'ng ESP oxirgi bo'lib joylashtirilgan 4 soniga ko'rsatgich[2] bo'lib turibdi. E'tibor bergan bo'lsangiz stackka yangi qiymat qo'shgan sari, u pastga qarab o'saveradi, ya'ni ESP qiymati 4 tadan kamayaveradi. `POP` buyrug'i bilan qiymat qaytarib olinganda esa, aksincha, eng oxirgi qo'shilgan qiymat stackdan olinadida ESP ga 4 soni qo'shiladi. Chunki qiymat olingandan so'ng, undan oldinroq joylashtirilgan qiymatga ESP ko'rsatgich bo'lib turishi kerak. `POP` orqali chiqarib olinayotgan qiymat qayerda saqlanishi buyruqqa ko'rsatilishi kerak. Bu to'rt baytli o'zgaruvchi yoki registr bo'lishi mumkin. Ilgarigi sonlarni chiqarib olamiz:

```
pop  eax        ;; EAX = 4
pop  ebx        ;; EBX = 3
```

4 soni eng oxirgi bo'lib kiritilgani tufayli birinchi bo'lib chiqadi, ya'ni EAX ga ko'chiriladi. EBX ga esa 3 soni o'zlashtiriladi. Shundan so'ng stack yana o'zining boshlang'ich holatiga qaytadi.

Dasturchi ESP registrining qiymatini qo'shish yoki ayirish amallari orqali o'zgartirishi mumkin. Lekin juda ehtiyot bo'lish kerak. Chunki dasturning butun ishlash jarayoni stackka ko'pincha bog'liq bo'ladi. Birgina noto'g'ri ESP ning o'zgartirilishi dasturning «qulashiga» olib kelishi

[1] Mazkur kitobda qiymatlar o'lchami bir yoki ikki baytli bo'lish hollari ko'rib chiqilmaydi.
[2] Ko'rsatgich deb xotiradagi axborot manzilini saqlab turgan obyektga aytiladi.

mumkin. Masalan, oldingi misolda ko'rsatilgan stackdagi sonlarni EAX va EBX registrlariga POP buyrug'i yordamisiz ham olib o'tishimiz mumkin edi:

```
mov  eax , [esp]
mov  ebx , [esp+4]
add  esp , 8
```

Mavzu boshida aytganimizdek, stackdan asosan vaqtinchalik qiymatlarni saqlashda foydalanish mumkin. Masalan EAX da kerakli qiymat saqlanayapti deylik. To'satdan EBX dagi qiymatni 7 ga bo'lish kerak bo'lib qoldi. Ammo `DIV` buyrug'i faqat EAX dagi qiymatni bo'lishga mo'ljallangan. EBX ning qiymatini EAX ga o'tkazishga to'g'ri keladi. Yaxshi yechim, lekin EAX ning o'zining qiymati o'chib ketadi-ku. Bu holatda avval EAX qiymati stackga kiritiladi, bo'luv amalga oshirilgach EAX ning qiymati yana stackdan qaytarib olinadi:

```
push eax            ;; [ESP] = EAX
mov  eax  ,  ebx
div  dword 7        ;; EAX = EDX:EAX / 7
mov  ebx , eax
pop  eax            ;; EAX = [ESP]
```

Stack bilan ishlash uchun yana `PUSHA` va `POPA` buyruqlari mavjud. Bu buyruqlar bir varakayiga EAX, EBX, ECX, EDX, ESI, EDI va EBP registr qiymatlarini ma'lum bir ketma-ketlikda stackka mos ravishda kiritadi va chiqarib oladi. Demak, `PUSHA` buyrug'idan so'ng ESP qiymati 28 taga kamayadi. `POPA` buyrug'idan so'ng esa aksincha. Bu buyruqlar asosan qismli dasturlashda qo'l keladi. Umuman olganda qismli dasturlashda stack hal qiluvchi ahamiyatga ega. Bu haqida keyingi mavzularda kengroq fikr yuritilgan.

8.4. `CALL` va `RET` buyruqlari

Qismli dasturlashda qism dasturiga o'tish amalga oshirilganda yana xuddi chaqirilgan joyga qaytish muammosi ko'rib chiqilgan edi. Bu muammoning yechimi sifatida qaytish kerak bo'lgan joylarga nishon qo'yib chiqilgan edi. Buning ustiga bu nishonlar manzili registrlarga o'zlashtirilib qism dasturiga ortiqcha qiymat sifatida jo'natilgan edi. Bu murakkabliklar `CALL` va `RET` buyruqlari orqali oson hal qilinadi. `CALL` buyrug'i qism dasturni chorlashga mo'ljallangan bo'lib, bunda dasturchi qaytish manzili haqida o'ylab o'tirmaydi. Qism dasturi oxirida esa `RET` buyrug'i qo'yiladi. Bu buyruqlar yordamida ko'rib chiqilgan 6-namuna quyidagicha ko'rinishga ega bo'ladi:

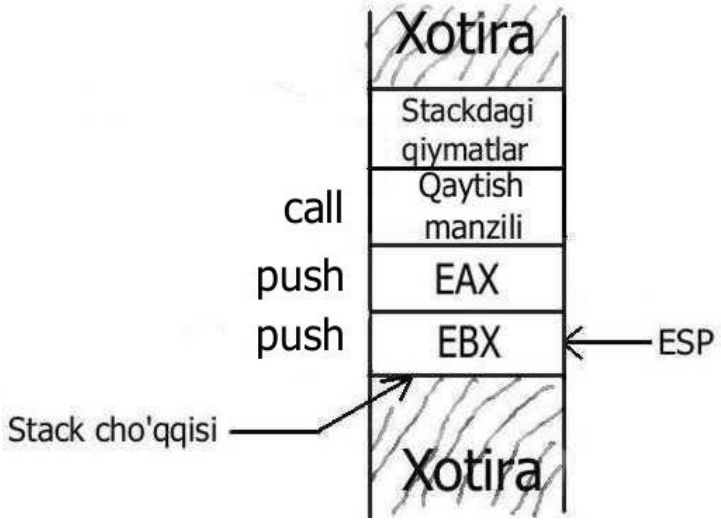
```
(1)  ;; Asosiy dastur
     …
(22) mov  ebx , [son1]
(23) call ildiz     ;; 6-namunadagi 24-27 qatorlarning keragi yo'q.
(24) mov  edi , eax
(25) mov  ebx , [son2]
(26) call ildiz     ;; 6-namunadagi 33-36 qatorlarning keragi yo'q.
(27) add  edi , eax
     …
(42) ;; Ildiz qism dasturi.
     …
(80) topildi:
(81) ret            ;; jmp ecx ning o'rnida
```

CALL buyrug'i JMP singari shartsiz o'tishni amalga oshiradi. Lekin uning asosiy farqi shundaki, o'tishni amalga oshirishdan oldin qaytish manzilini, ya'ni EIP registri qiymatini stackka kiritadi. Demak, CALL buyrug'idan foydalanganimizda ESP ning qiymati 4 ga kamayadi. RET buyrug'i esa qaytish manzilini stackdan chiqarib olib, shu manzilga o'tishni amalga oshiradi. Qism dasturi ichida qancha qiymat stackga kiritilsa, hammasi RET buyrug'idan oldin stackdan chiqarib olinishi kerak. Chunki RET buyrug'i hech qanday javobgarlikni o'z bo'yniga olmaydi. U stack cho'qqisidagi qiymatni qaytish manzili sifatida qabul qiladi. Bu qiymat CALL buyrug'i kiritgan EIP registri qiymati bo'lishi shart. Quyidagi namunani ko'rib chiqamiz:

```
;; asosiy dastur
…
call qism_dastur
…

;; qism dasturi
qism_dasturi:
     push eax
     push ebx
     ret
```

CALL buyrug'i o'z navbatida EIP ni qiymatini stackka yuklaydi. Qism dasturi ishga tushganda esa EAX hamda EBX qiymatlari stackka kiritiladi. RET buyrug'i ko'r-ko'rona EBX qiymatini chiqarib olib, umuman noto'g'ri bo'lgan manzilga o'tmoqchi bo'ladi. Shuning uchun RET dan oldin ikki marta POP orqali kiritilgan qiymatlarni chiqarib tashlash kerak yoki ESP ga 8 ni qo'shish kerak (rasmga qarang).



30-rasm.

8.5. Mahalliy nishonlar

Biz shu paytgacha assemblerning barcha bo'limlarida foydalanib kelgan nishonlar ommaviy ma'noga ega edi. Qismli dasturlar bilan ish ko'rganda esa masala biroz murakkablashadi va nishonlardan ham ko'proq foydalanishga to'g'ri keladi. O'rgangan tashqi va ommaviy nishon turlarimiz bizga hamma vaqt ham xohlagan ishimizni qilishga imkoniyat bermaydi. Masalan, bitta faylda joylashgan ikkita qism dasturi o'z kod bo'limlarida bir xil nomli nishondan foydalanmoqchi bo'lishdi deylik. Lekin qaysidir bir nishonga o'tish kerak bo'lsa, ikkilanish yuzaga keladi.

```
ildiz:              ;; birinchi qism dasturi
     ...
davom:              ;; davom nishoni
     ...
     ret

kub_ildiz:          ;; ikkinchi qism dasturi
     ...
     jz davom       ;; qaysi davom ga?!
     ...
davom:              ;; yana birta davom?
     ...
     ret
```

Bu dasturni yig'ish paytida assembler xato yuz bergani haqida xabar beradi. Chunki bitta nishon ikki marta ikki xil joyda e'lon qilingan.

Bunday muammolar mahalliy nishonlar orqali hal etiladi. Mahalliy nishonlar o'zidan oldin keladigan ommaviy nishonga ergashib kelib, faqat mahalliy, ya'ni qism dasturi ichida ma'noga ega. Odatda qism dasturini boshlaydigan nishon ommaviy e'lon qilinadi va uning oxirigacha bo'lgan barcha nishonlar mahalliy qilib e'lon qilinadi.

Assemblerda mahalliy nishonlar nomi nuqta (`.`) bilan boshlanadi.

```
ildiz:              ;; birinchi qism dasturi
     ...
     jz   .davom    ;; hammasi joyida
     ...
.davom:             ;; mahalliy nishon
     ...
     ret

kub_ildiz:          ;; ikkinchi qism dasturi
     ...
     jz   .davom    ;; hammasi joyida
     ...
.davom:             ;; yuqoridagi davom ga aloqasi yo'q
     ...
     ret
```

Bu yerda chalkashlik kelib chiqmasligining sababi shundaki, NASM barcha mahalliy nishon nomlari oldiga ular ergashib kelayotgan ommaviy nishon nomini qo'shib chiqadi. Masalan, birinchi `.davom` nishoni `ildiz.davom` ga, ikkinchisi esa `kub_ildiz.davom` ga almashtiriladi. Ammo bir qism dasturida turib boshqa qism dasturdagi mahalliy nishonlarga murojaat etish mumkin. Buning uchun o'sha nishon nomi to'liq ko'rsatiladi. Masalan, uchinchi qism dasturi bo'lsin:

```
boshqa_nishon:                ;; Ommaviy nishon.
     ...
.davom:                       ;; Mahalliy nishon.
     jmp  ildiz.davom         ;; Yuqoridagi qism dasturlaridagi.
     jmp  kub_ildiz.davom     ;; Mahalliy nishonlarga o'tish.
     jmp  .davom              ;; boshqa_nishon.davom ga o'tish.
```

8.6. Stack orqali qiymatlar jo'natish

Qismli dasturlash haqida shu paytgacha o'rganganlarimiz allaqachon to'liq qism dasturlarini tuzishga imkon beradi. Olgan bilimlarimiz orasida stack katta ahamiyatga ega. Chunki stack qismli dasturlar bilan ishlashni ancha osonlashtiradi. Bu mavzuda biz stackning yana bir foydali xususiyatini ko'rib chiqamiz.

Qism dasturini chaqirish va unga qiymatlar jo'natishning o'z me'yor va mezonlari bo'lib, ularga doim amal qilgan ma'qul. Chunki katta-katta dasturlarni ko'pchilik tuzganda ma'lum bir kelishuv bo'lishi kerak. Boshqa dasturlash tillaridagi dasturlardan assemblerda yozilgan qism dasturini chaqirishda ham qat'iy kelishuv asosida qism dasturi chaqiriladi va unga qiymatlar jo'natiladi. Bu kelishuvlarning birinchisi doim qism dasturini `CALL` orqali chaqirishdir. Ikkinchisi doim qiymatlar qism dasturiga stack orqali jo'natilishidir. Yodingizda bo'lsa, ildiz qism dasturiga qiymatlar registrlar orqali jo'natilgan edi. Ammo buning bir nechta o'ng'aysizliklari bor:

1. Jo'natilishi kerak bo'lgan qiymatlar soni registrlar sonidan ko'p bo'lsachi?
2. Registrlar hammasi boshqa kerakli qiymatlar bilan band bo'lsachi? Qism dasturi ham o'z navbatida registrlardan boshqa maqsadlarda foydalanishi kerak bo'lsachi?
3. Doim qaysi registrda aynan qaysi qiymat jo'natilganini hisobini olib turishga to'g'ri keladi.
4. Chaqiruvchi dasturning registrlardan foydalanish imkoniyati bo'lmasachi?

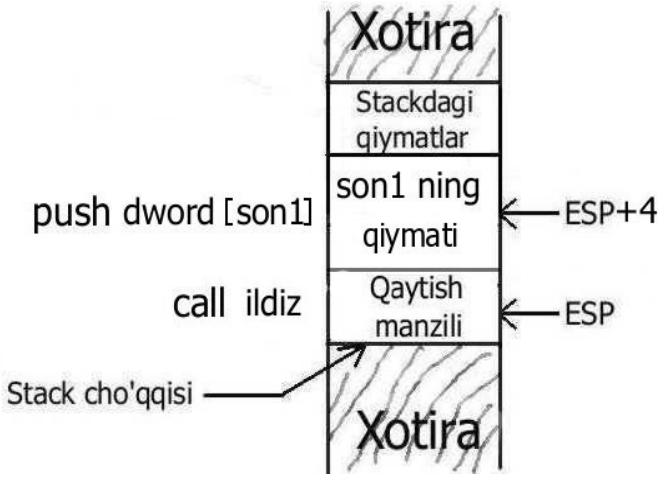
Shuning uchun dasturlash mezoniga ko'ra qiymatlar qism dasturiga stack orqali jo'natiladi. Bu juda ham oson bo'lib kerakli qiymatlarni stackka kiritasizda qism dasturini chaqirasiz. Demak, 6-namunadagi dasturni takomillashtirishni davom ettirib, dasturlash mezonlarini qo'llaymiz:

6.2-namuna

```
(1)  %include "nasm-io.inc"
(2)
(3)  section .bss
(4)  son1 resd 1
(5)  son2 resd 1
(6)
(7)  section .text
(8)  tizim_global   main
(9)
(10) main:
(11)      chop_et   `Ikkita son kiriting : `
(12)      qabul_qil `%i %i`, son1, son2
(13)      push dword [son1] ;;Qiymat EBX emas,stack orqali jo'natiladi.
(14)      call ildiz          ;; ildiz qism dasturini chaqirish.
(15)      mov  edi , eax      ;; son1 ning kvadrat ildizini
(16)                          ;; EDI da saqlaymiz.
(17)      push dword [son2] ;;Qiymat EBX emas,stack orqali jo'natiladi.
(18)      call ildiz          ;; ildiz qism dasturini chaqirish.
(19)      add esp , 8         ;; Kiritilgan ikkita qiymatni stackdan
(20)                          ;; boshatamiz, bu juda muhim.
(21)      add  edi , eax
(22)      chop_et   `Javob = %i \n`, edi
(23)
(24) ret
(25)
(26) ;; Ildiz qism dasturi.
(27) section .bss
(28) xn   resd 1              ;; x_n, n = {1, ..., n}
```

```
(29)
(30) section   .text
(31) ildiz:
(32)      chop_et   `Ildiz qism dasturi ishga tushdi \n`
(33)      mov eax , [esp + 4] ;; Jo'natilgan qiymatni EBX dan emas,
(34)                          ;; balki stackdan EAX ga o'zlashtiramiz.
(35)      mov  [xn] , eax
(36)      shr  dword [xn] , 1       ;; x1 <- a / 2
(37)
(38) .takrorlanish:
(39)      xor  edx , edx            ;; EDX:EAX ning to'ng'ich 4 baytini
(40)                                ;; nollaymiz .
(41)      div  dword [xn]           ;; EAX <- a / xn
(42)      add  eax , [xn]           ;; EAX <- xn + a/xn
(43)      shr  eax , 1              ;; EAX <- (xn + a/xn)/2,
(44)                                ;; bu yerda EAX = xn+1
(45)      cmp  eax , [xn]           ;; (EAX = xn) ?
(46)      jz   .topildi
(47)      mov  [xn] , eax           ;; xn <- xn+1
(48)      mov  eax , [esp + 4]      ;; Jo'natilgan qiymatni tiklaymiz.
(49)      jmp  .takrorlanish
(50)
(51) .topildi:
(52) ret                            ;; jmp ecx ning o'rnida
```

Qism dasturi chaqirilishi oldindan `son1` o'zgaruvchisining qiymati stackka yuklanadi. `son1` 4 baytli bo'lgani uchun `dword` dan foydalanamiz. `CALL` buyrug'i esa qaytish manzilini ham stackka yuklaydi. Shuning uchun qism dasturi ichida biz stack cho'qqisidagi emas, balki bitta oldin kiritilgan qiymatni EAX ga o'zlashtiramiz, ya'ni `[ESP+4]` ni.



31-rasm.

31-rasmda stack qanday ko'rinishga ega bo'lishi ko'rsatilgan.

Qism dasturga qiymatlarni jo'natishning ikki xil usuli bor. Birinchi usul *qiymat orqali jo'natish* bo'lib, bunda stack orqali o'zgaruvchi qiymatning o'zi jo'natiladi. Agar o'zgaruvchi qiymati qism dasturi tomonidan o'zgartirilishi kerak bo'lmasa, bu usul ma'qul hisoblanadi. Masalan, `ildiz` qism dasturiga ham ushbu yo'sinda qiymatlar jo'natiladi, chunki qism dasturini faqat qiymatning o'zi qiziqtiradi. U qiymatni olib, uning kvadrat ildizini hisoblaydi va natijani EAX da qaytaradi. Qism dasturini ushbu usulda amalga oshirish matematikadagi funksiyalar tushunchasidan olingan: $y = f(x)$.

Ikkinchi usul *manzil orqali jo'natish* bo'lib, bunda stack orqali o'zgaruvchining qiymati emas, balki manzili jo'natiladi. Agar o'zgaruvchi qiymati qism dasturi tomonidan o'zgartirilishi kerak bo'lsa, bu usul ma'qul hisoblanadi. Barcha dasturlash tillarida ham o'zgaruvchi qiymatiga o'zgartirish kiritish zarur bo'lsa, uning manzili qism dasturiga jo'natiladi. Masalan, ikki o'zgaruvchi qiymatini o'zaro almashtiradigan qism dasturini olaylik. Ushbu qism dasturi o'zgaruvchi qiymatiga o'zgartirishi kerak, ya'ni ularni almashtirishi kerak. Shuning uchun almashtirish qism dasturiga o'zgaruvchi manzillari yuborilishi kerak.

7-namuna: Almash qism dasturi

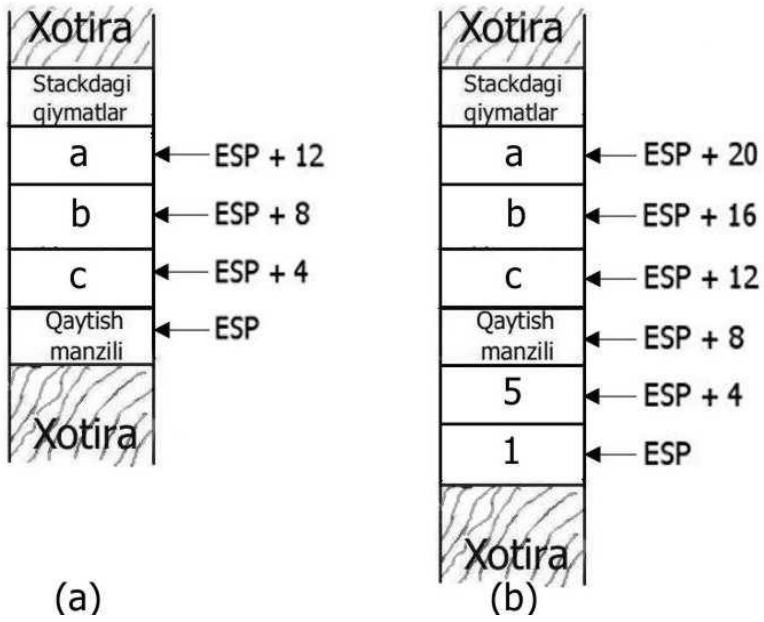
```
(1)   ;; Asosiy dastur.
(2)   %include "nasm-io.inc"
(3)
(4)   section .data
(5)   a   dd   9
(6)   b   dd   21
(7)
(8)   section .text
(9)   tizim_global  main
(10)
(11)  main:
(12)      chop_et   `a = %i, b = %i \n`, [a], [b]
(13)      push dword b              ;; b ning manzili stackka yuklandi.
(14)      push dword a              ;; a ning manzili stackka yuklandi.
(15)      call almash
(16)      add  esp , 8   ;; Jo'natilgan qiymatlar stackdan o'chiriladi.
(17)      chop_et   `a = %i, b = %i \n`, [a], [b]
(18)
(19)  ret
(20)
(21)  ;; Almash qism dasturi.
(22)  ;;
(23)  ;; Maqsad:
(24)  ;;  Ikki o'zgaruvchi qiymatini o'zaro almashtirish.
(25)  ;;
(26)  ;; Qiymatlar:
(27)  ;;  O'zgaruvchilar manzillari stack orqali jo'natiladi.
(28)
(29)  almash:
(30)      mov  eax , [esp+4]        ;; a ning manzili stackdan olinadi.
(31)      mov  ebx , [esp+8]        ;; b ning manzili stackdan olinadi.
(32)      push dword[eax]
(33)      push dword[ebx]
(34)      pop  dword[eax]
(35)      pop  dword[ebx]
(36)  ret
```

Dasturning 13-, 14- qatorlarida o'zgaruvchilarning manzillari jo'natiladi. Agar shunday qilinmasa, almash qism dasturi qiymatlarni almashtirgani bilan asosiy dasturga qaytganda o'zgaruvchilar qiymati o'zgarishsiz qoladi. 32–35-qatorlarda qiymatlarni o'zaro almashtirishning an'anaviy usuli qo'llanilgan.

Yana bir muammo stackdan qism dasturi ichida foydalanganda kelib chiqadigan noqulaylik bilan bog'liqdir. Masalan, qism dasturiga `a`, `b`, va `c` o'zgaruvchilari qiymatlarini yuborish kerak bo'ldi deylik:

```
  push dword [a]
  push dword [b]
  push dword [c]
  call qism_dasturi
```

Qism dasturida qaytish manzilini hisobga olgan holda `a`, `b` va `c` larning qiymatlariga `ESP+12`, `ESP+8` va `ESP+4` lar orqali murojaat qilishimiz mumkin.



32-rasm.

Tasavvur qiling qism dasturi ichida stackdan foydalanish kerak bo'lib qoldi va siz u yerga yana 2 ta qiymat kiritasiz:

```
qism_dastur:
  push dword 5
  push dword 1
  …
```

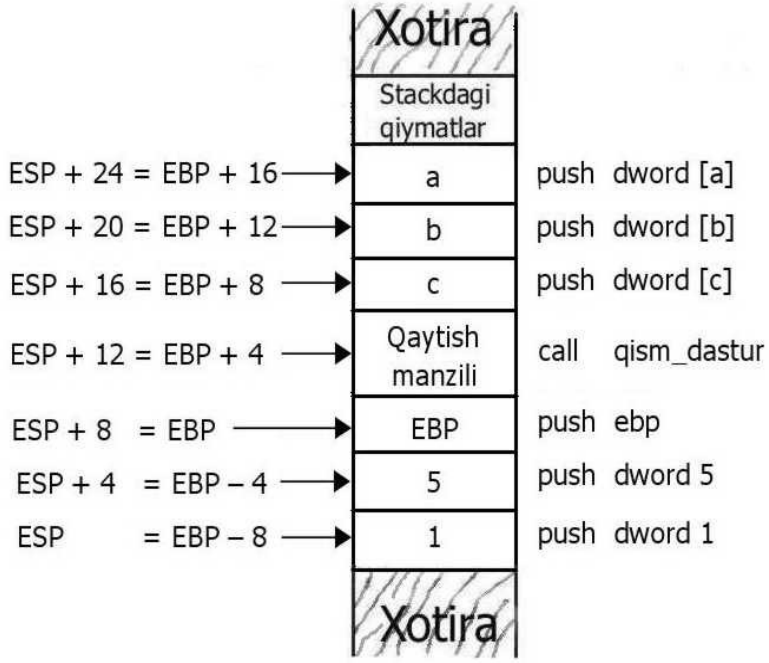
Endilikda `a` ni `ESP+12` manzili orqali emas, balki `ESP+20` orqali hisoblashga to'g'ri keladi. (a) rasmda stackning qism dasturi undan foydalanmasidan oldingi holati tasvirlangan. (b) rasmda esa 5 va 1 stackka kiritilgandan so'ng yuzaga kelgan holat ko'rsatilgan. Shunday qilib stack holati o'zgarishi bilan qism dasturiga jo'natilgan qiymatlarning manzili ham o'zgarib boradi. Bu chalkashlikka olib kelishi mumkin. Mazkur masalani yechish uchun EBP registri taqdim etilgan. Yechim shundaki siz qism dasturi boshida ESP ning qiymatini EBP ga o'zlashtirasiz. Shundan keyin ESP qancha o'zgarsa ham EBP uning odingi qiymatini o'zida saqlab qoladi. Jo'natilgan qiymatlarga esa EBP orqali murojaat qilasiz, lekin EBP dan foydalanganda uning avvalgi qiymatini yo'qotib yubormaslik kerak. Chunki undagi qiymat operatsion tizim uchun muhim hisoblanadi. Shuning uchun birinchi uning qiymati stackka yuklanadi, so'ngra ESP qiymati EBP ga o'zlashtiriladi. Qism dasturi oxirida esa EBP ning oldingi qiymati stackdan `POP` orqali tiklanadi.

Qiyidagi misolda `a`, `b`, va `c` o'zgaruvchilari jo'natiladigan qism dasturida EBP registridan foydalanilgan holda yoritilgan:

```
(1)  qism_dasturi:
(2)    push ebp
(3)    mov  ebp , esp
```

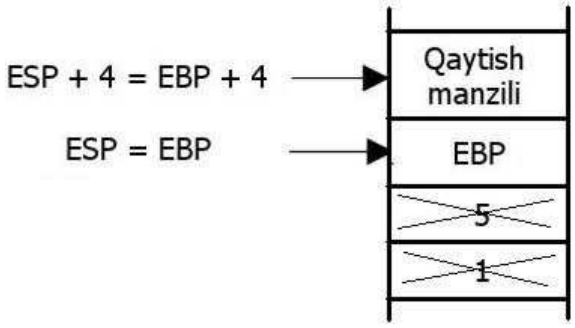
```
(4)     push dword 5
(5)     push dword 1
(6)     mov  ecx , [ebp+8]       ;; c ning qiymatini ECX ga yuklash.
(7)     mov  ebx , [ebp+12]      ;; b ning qiymatini EBX ga yuklash.
(8)     mov  eax , [ebp+16]      ;; a ning qiymatini EAX ga yuklash.
(9)     ;; qism dasturi yakuni
(10)    mov   esp , ebp
(11)    pop  ebp
(12)    ret
```

2-qatorda EBP ning shu paytgacha bo'lgan qiymatini kelishuvga ko'ra vaqtinchalik stackka kiritib turamiz. Keyin esa ESP ning o'sha holatini EBP ga ko'chiramiz. Stack bilan bo'ladigan keyingi o'zgarishlardan qat'i nazar bundan buyogi'ga jo'natilgan qiymatlarga EBP orqali murojaat qilinadi. Lekin ularning manzillarini hisoblashda qaytish manzili bilan bir qatorda stackka kiritilgan EBP ning qiymatini ham hisobga olish kerak. Demak, qism dasturiga jo'natilgan qiymatlardan keyin ham yana ikkita qiymat stackka kiritilgani sababli `EBP+8` va hokazolar orqali qiymatlarga murojaat qilamiz. Quyidagi rasmda stackning 5-qatoridan so'nggi holati tasvirlangan.



33-rasm.

10-qatorda esa ESP ga uning qism dasturi boshidagi holati qaytariladi, ya'ni ESP=EBP (34-rasmga qarang).



34-rasm.

Bu degani qism dasturi ichida stackka qanday o'zgarishlar kiritilgan bo'lmasin ESP dastur yakunida kerakli joyga borib turishini anglatadi demakdir. Kerakli joy deyilishiga sabab, keyingi buyruq yordamida EBP ning eski qiymati chiqarib olinadi va ESP qaytish manziliga ko'rsatgich bo'lib turadi. `RET` buyrug'i esa stack cho'qqisida nima turgan bo'lsa, o'shani chiqarib olib, o'tishni amalga oshiradi.

Yana bir o'ylantiradigan masala stackni keraksiz qiymatlardan tozalab turishdir. Shu joyda qism dasturi yakunida asosiy dasturdan unga jo'natilgan qiymatlarni stackdan bo'shatish kerakmi degan savol tug'iladi. Kelishuv mezonlarida qism dasturi ichida unga jo'natilgan qiymatlarga tegmaslik maslahat beriladi. Chunki ularni chiqarib tashlash uchun avval qaytish manzilini chiqarib olish va uni vaqtinchalik qayerdadir saqlash kerak bo'ladi. So'ngra qiymatlar chiqarilgach yana qaytadan qaytish manzilini stackka kiritish kerak bo'ladi. Aks holda `RET` buyrug'i to'g'ri ishlamaydi. Shuning uchun qiymatlarni jo'natgan dasturning o'zi keyinchalik chiqarib tashlashi kerak.

Shunday qilib, sizga ma'lum bo'lganidek, stackni bo'shatishning ikki xil usuli bor. Agar sizga stackdagi qiymatlar qandaydir ma'noni anglatsa, ya'ni kerak bo'lsa, u holda `POP` buyrug'i orqali ularni kerakli o'zgaruvchilarga yuklab olasiz. Agar stackdagi qiymatlarni shunchaki o'chirib tashlash kerak bo'lsa, ESP ga o'chirilishi kerak bo'lgan qiymatlar o'lchami qo'shlib uni orqaga qaytariladi.

Yuqoridagi misolda asosiy dasturga qaytilganda `a`, `b` va `c` larni quyidagicha stackdan bo'shatish mumkin:

```
;; Agar qiymatlar kerak bo'lsa
pop  ecx            ;; ECX = c
pop  ebx            ;; EBX = b
pop  eax            ;; EAX = a
;; Agar qiymatlar kerak bo'lmasa
add  esp , 12  ;; a, b va c umumiy o'n ikki baytni tashkil etadi.
```

8.7. ENTER va LEAVE buyruqlari

Qism dasturining ichki o'zgaruvchilari haqida aytib o'tgan edik. Qism dasturi faqat o'ziga kerak bo'ladigan o'zgaruvchilarni o'z `.bss` va `.data` bo'limida e'lon qilishi mumkin. Lekin bu kabi ichki o'zgaruvchilar uchun ham stackdan joy ajratilishi udumga kirgan. Buning uchun qism dasturiga kirishda EBP ni stackka yuklagach ichki qiymatlar o'lchami ESP dan ayiriladi. Bu kabi ajratilgan joyga keyinchalik EBP orqali murojaat qilinadi:

```
qism_dasturi:
  push ebp
  mov  ebp , esp
  sub  esp , ichki_qiymatlar_o'lchami
  ;; qism dasturi kodi
  mov  esp , ebp          ;; Ichki qiymatlarni stackdan bo'shatish.
  pop  ebp                ;; EBP ning oldingi qiymatini tiklash
  ret
```

Namunadagi birinchi uchta qator dastur muqaddimasi, oxirgi ikki qator esa xotimasi deyiladi. Masalan, qism dasturiga ichki ikkita to'rt baytli o'zgaruvchi kerak bo'ldi deylik:

```
push ebp
mov  ebp , esp
sub  esp , 8
```

Bu qiymatlarga `EBP-8` va `EBP-4` orqali murojaat qilinadi. Nega bunday ekanligini 33-rasmga qarab bilib olish mumkin. Farq shundaki, u yerda qiymatlar oldindan ma'lum bo'lgani uchun `PUSH` buyrug'idan foydalanilgan. Agar ichki o'zgaruvchilar qiymati hali noaniq bo'lsa, `SUB` buyrug'idan foydalanasiz.

> **Eslatma**: ESP dan ayiradigan soningiz 4 ga karrali bo'lishiga harakat qilining. Albatta, bir baytli yoki ikki baytli qiymatlar uchun ham stackdan joy ajratish mumkin. Ammo bu katta chalkashliklarga olib kelishi mumkin.

`ENTER` buyrug'i dastur muqaddimasini osonlashtirish uchun mo'ljallangan bo'lib keltirilgan uchta buyruq o'rnida foydalanasiz. Bu buyruqqa ikkita qiymat beriladi. Birinchi qiymat bu ichki dastur o'zgaruvchilari uchun stackdan ajratiladigan joy o'lchamidir. Ikkinchi qiymat nol bo'ladi. `LEAVE` buyrug'i esa xotimani amalga oshiradi.

```
qism_dasturi:
   enter ichki_qiymatlar_o'lchami , 0
   ;; qism dasturi kodi
   leave
   ret
```

Qism dasturining qiymatlari uchun stackdan yoki `.bss`, `.data` bo'limlaridan joy ajratish dasturchi tanloviga havola. Biroq stackdan foydalanmagan taqdiringizda ham muqaddima va xotimani ishlatgan ma'qul. Buning bir nechta sabablari bor. Masalan, asosiy dastur registrlarida o'zining kerakli qiymatlarini saqlab turib qism dasturini chaqirib qolishi mumkin. Qism dasturi ishga tushgach u ham registrlardan o'z manfaati uchun foydalana boshlaydi. Bu holatda asosiy dasturning registrlardagi qiymatlari o'chib ketishi mumkin. Boshqaruv yana qaytib asosiy dasturga topshirilganida esa registrdagi o'zgarib ketgan qiymatlar dasturining noto'g'ri bajarilishiga olib keladi.

Bunday holatlarni oldini olish uchun qism dasturi muqaddimasidan so'ng barcha registrlardagi asosiy dastur qiymatlarini stackka yuklab qo'yish kerak. Bu ish `PUSHA` buyrug'i bilan osonroq bajariladi. Ammo mazkur buyruqdan so'ng ESP qanchaga o'zgarishini aniqlash qiyin. ESP noma'lum tarzda o'zgarib ketgach stack orqali jo'natilgan qiymatlarning manzilini hisoblash mumkin bo'lmay qoladi. Agar muqaddimadan foydalanilgan bo'lsa, bu holda ESP ning holatidan qat'i nazar EBP orqali jo'natilgan qiymatlarni qo'lga kiritamiz. Yuritgan fikrlarimizdan kelib chiqadiki, agar qism dasturiga stack orqali qiymat jo'natilgan bo'lsa yoki ichki qiymatlar uchun stackdan joy ajratilgan bo'lsa, bunda muqaddima va xotimadan foydalanish ma'noga ega bo'ladi.

Dastur oxirida esa xotimadan oldin `POPA` buyrug'i yordamida barcha registrlar qiymati tiklanadi:

```
qism_dastur:
   enter ichki_qiymatlar_o'lchami , 0
   pusha          ;; Barcha registrlarni stackka yuklash.
   ;; qism dasturi kodi
   popa           ;; Barcha registrlar qiymatlarini stackdan tiklash.
   leave
   ret
```

Bu mezonlarga amal qilib dastur tuzish doim samarali bo'ladi. Mezonlarni yana bir bor sanab chiqamiz:

- `CALL` va `RET` buyruqlari orqali qism dasturini chiqarish va u yerdan qaytish.
- Doim muqaddima va xotimadan foydalanish.
- Qiymatlarni stack orqali jo'natish.
- Qism dasturi natijalarini asosiy dasturga registrlar orqali yuborish (asosan EAX registri orqali)

8.8. `LEA` buyrug'i

Qism dasturiga qiymatlarning manzillari stack orqali jo'natilganda yoki uning ichki o'zgaruvchilari uchun stackdan joy ajratilganda shu manzillar bilan vositali ishlash talab qilinishi mumkin. Bu degani ma'lum bir hisoblangan manzilni biror o'zgaruvchiga, xususan registrga yuklab, so'ng uning qiymatiga manzil sifatida qarashdir. Masalan, qism dasturi o'zining ichki o'zgaruvchilari `x` va `y` uchun stackdan joy ajratgan bo'lsin:

```
enter     8 , 0          ;; to'rt baytdan
```

`EBP - 4` ni `x`, `EBP - 8` ni esa `y` deb tasavvur qilamiz. `x` bilan vositali ishlash uchun uning manzilini biror registrga, masalan, EAX ga o'zlashtirish kerak bo'ladi:

```
mov  eax , ebp-4
```

Lekin bu assembler qoidalariga to'g'ri kelmaydi. Chunki o'zlashtirilayotgan obyekt yoki o'zgarmas, yoki o'zgaruvchi bo'lishi kerak. EBP - 4 kabi arifmetik ifodalarni assembler hisoblay olmadi. Agar burchakli qavs qo'llansa:

```
mov  eax , [ebp-4]
```

EAX ga `x` ning manzili emas, balki qiymati ko'chiriladi. Albatta, bu muammoni yechish uchun EAX ga EBP ni o'zlashtirib keyin undan 4 ni ayirsa bo'ladi:

```
mov  eax , ebp
sub  eax , 4
```

Biroq bu ikkita buyruqni bajarishni talab qiladi va bundan tashqari hisoblamoqchi bo'lgan manzilingiz murakkabroq ko'rinishga ega bo'lishi ham mumkin, masalan, EBP + EBX*4 - 8.

`LEA` (**L**oad **E**ffective **A**ddress - Foydali Manzilni Yuklash) buyrug'i shu ishni oson bajarishga mo'ljallangan. Uning `MOV` dan farqi shundaki, `LEA` doim qiymatni emas, balki manzilni o'zlashtiradi. Demak, `EBP-4` manzilini EAX ga yuklash uchun quyidagi qo'l keladi:

```
lea  eax , [ebp-4]
```

`LEA` buyrug'i `MOV` dan farq qilsada, u bilan ham `MOV` bajaradigan ishni amalga oshirsa bo'ladi. Masalan, EAX ga EBX ni o'zlashtirish kerak bo'lsin:

```
mov  eax , ebx
```

Buni `LEA` orqali qilsa ham bo'ladi:

```
lea  eax , [ebx]
```

Shuni anglash zarurki, `LEA` hech qachon xotiradagi qiymatga murojaat qilmaydi. U shunchaki nishon manzilini yoki registr qiymatini hisoblaydi va natijani `maqsad` ga yuklaydi. Bundan kelib chiqadiki, `LEA` yordamida o'zingizni manzillar bilan ishlayotganday tutib, aslida ko'p vaqtni oladigan arifmetik hisoblashlarni `ADD`, `SUB` va `MUL` buyruqlarisiz amalga oshirishingiz mumkin bo'ladi. Masalan, EAX + EBX*4 ni hisoblash va yig'indini EDX ga saqlash kerak bo'lsa:

```
lea  edx , [eax + ebx*4]
```

Ammo burchakli qavs ichida beriladigan ifoda ixtiyoriy bo'la olmaydi. Uning tuzilish shartlari bor:

- Ifoda ko'pi bilan uchta haddan iborat bo'lishi mumkin.
- Uch hadli ifoda berilsa, hadlardan bittasi o'zgarmas bo'lishi shart!
- Ko'paytma ko'rinishida beriladigan hadlar faqat registrlar va/yoki o'zgarmaslar orqali berilishi mumkin, nishonlar bilan emas. Masalan: 7*8, EAX*4 va hokazo. Registrga ko'paytiriladigan o'zgarmas sonlar bo'lib faqat 1, 2, 4 va 8 lar kelishi mumkin.

Misollar:

```
lea  eax , [nishon + ebx*4 - 12]
lea  eax , [edi+eax]
lea  eax , [edx*2 + 18]
```

Amaliy mashq sifatida $ax^2+bx+c=0$ ko'rinishidagi tenglamani yechuvchi dasturni ko'rib chiqamiz. Dasturga a, b va c koeffitsiyentlari beriladi. Dastur esa tenglama ildizlari x_1 va x_2 larni topishi kerak. Maktab dasturidan bilamizki, bu ko'rinishdagi tenglamalar yechimi quyidagi formula yordamida topiladi:

$$x_{1,2}=\frac{-b\pm\sqrt{b^2-4\mathrm{ac}}}{2\mathrm{a}}$$

Ildiz ostidagi ifoda diskriminant atalib, u lotincha D harfi bilan belgilanadi. Agar diskriminant noldan kichik bo'lsa, tenglama yechimga ega bo'lmaydi. Agar diskriminant nolga teng bo'lsa, tenglama bitta yechimga, noldan katta bo'lsa, ikkita yechimga ega bo'ladi. Diskriminantni ildizdan chiqarishda ildiz qism dasturidan foydalanamiz. Bundan tashqari diskriminantni o'zini hisoblash uchun alohida qism dasturi tuzamiz. Dasturimiz tenglamani har doim ham aniq yechmaydi. Chunki $x_{1,2}$ larni hisoblashda D dan ildiz olishga to'g'ri keladi. Ildiz qism dasturimiz esa faqat yaxlitlangan kvadrat ildizni hisoblaydi. Bundan tashqari formulada kasr chizig'i mavjud va bo'luvdan har safar ham butun son chiqavermaydi. Demak, barcha bo'lish amallarida qoldiqlar e'tibordan chetda qoldiriladi. Aytishimiz mumkinki, natijalar taqribiy bo'ladi, ammo bizning birinchi maqsadimiz matematik aniqlikka erishish emas, balki dasturlash asoslarini o'rganishdir. Shu jihatdan kichik kamchiligimizni oqlashimiz mumkin. Lekin siz dasturga javobi aniq hisoblanadigan tenglama koeffitsiyentlarini berishingiz mumkin. Masalan, $x^2+3x-4=0$ da $x_1=1$ va $x_2=-4$ yechimlarini olasiz. Asosiy dasturni *kvadrat_tenglama.asm* faylida saqlaymiz. Qism dasturlarini esa alohida *qism.asm* faylida saqlaymiz.

8-namuna: kvadrat_tenglama.asm

```
(1)  ;; Qism dasturlari: ildiz, diskriminant
(2)  ;; O'zgaruvchilar: a, b va c.
(3)
(4)  %include "nasm-io.inc"
(5)
(6)  section .bss
```

```
(7)  a    resd 1
(8)  b    resd 1
(9)  c    resd 1
(10)
(11) section .text
(12) extern ildiz, diskriminant
(13) tizim_global  main
(14)
(15) main:
(16)      enter     0 , 0
(17)      chop_et   `Tenglama koeffitsiyentlarini kiriting (a,b,c) : `
(18)      qabul_qil `%i %i %i`, a, b, c
(19)
(20)      push dword[c]       ;; a,b va c lar diskriminant qism
(21)      push dword[b]       ;; dasturiga jo'natiladi.
(22)      push dword[a]
(23)      call diskriminant
(24)      add  esp , 12       ;; stackni bo'shatamiz
(25)
(26)      neg  dword[b]       ;; b → -b
(27)      sal  dword[a] , 1   ;; a → 2a
(28)
(29)      cmp  eax , 0
(30)      jge  .manfiy_emas
(31)      chop_et   `Tenglama ildizga ega emas\n`
(32)      jmp  .tamom
(33)
(34) .manfiy_emas:
(35)      jnz  .ikkita_yechim
(36)      mov  eax , [b]      ;; Demak, tenglama bitta yechimga ega
(37)      cdq                 ;; EAX → EDX:EAX, ishorali son sifatida
(38)      idiv dword[a]       ;; EAX <- -b / 2a
(39)      chop_et   `X = %i\n`, eax
(40)      jmp  .tamom
(41)
(42) .ikkita_yechim:
(43)      push eax            ;; Diskriminantning ildizni
(44)      call ildiz          ;; hisoblaymiz.
(45)      add  esp , 4
(46)      mov  ecx , eax      ;; Ildiz natijasini saqlab qo'yamiz
(47)      add  eax , [b]      ;; EAX ← -b + ildiz(b*b - 4ac)
(48)      cdq
(49)      idiv dword[a]       ;; EAX <- ( -b + ildiz(b*b - 4ac) ) / 2a
(50)      chop_et   `X1 = %i, `, eax
(51)      mov  eax , [b]      ;; EAX <- -b
(52)      sub  eax , ecx      ;; EAX <- -b - ildiz(b*b - 4ac)
(53)      cdq
(54)      idiv dword[a]       ;; EAX <- ( -b - ildiz(b*b - 4ac) ) / 2a
(55)      chop_et   `X2 = %i\n`, eax
(56)
(57) .tamom:
(58)      leave
(59)
(60) ret
```

qism.asm

```
(1)  %include "nasm-io.inc"
(2)
(3)  section .text
(4)  global ildiz, diskriminant
(5)
(6)  ;; Diskriminant qism dasturi.
(7)  ;;
(8)  ;; Maqsad:
(9)  ;;  Stack orqali uchta qiymat qabul qiladi: a = [ebp + 8],
(10) ;;  b = [ebp + 12], c = [ebp + 16] va b*b - 4*a*c ni hisoblaydi.
(11) ;;
(12) ;; C tilidagi e'lon qilinishi:
(13) ;;  int diskriminant(int a, int b, int c);
(14) ;;
(15) ;; Natija:
(16) ;;  Hisoblangan diskriminant EAX da qaytariladi.
(17)
(18) diskriminant:
(19)      enter     0 , 0
(20)      push ecx                ;; Qism dasturda ishlatiladigan
(21)      push edx                ;; registrlarning oldingi
(22)      push ebx                ;; qiymatlarini stackka yuklaymiz.
(23)
(24)      mov  ecx , [ebp+16]     ;; ECX <- c
(25)      mov  ebx , [ebp+12]     ;; EBX <- b
(26)      mov  eax , [ebp+8]      ;; EAX <- a
(27)      cdq                     ;; EAX → EDX:EAX
(28)      imul ecx                ;; EAX <- a*c
(29)      mov  ecx , eax          ;; ECX <- a*c
(30)      mov  eax , ebx
(31)      cdq                     ;; EAX → EDX:EAX
(32)      imul ebx                ;; EAX ← b*b
(33)      neg  ecx
(34)      lea  eax , [eax + 4*ecx] ;; EAX <- b2 - 4ac
(35)
(36)      pop  ebx                ;; Ishlatiladigan registrlarning
(37)      pop  edx                ;; oldingi qiymalari tiklanayapti.
(38)      pop  ecx
(39)      leave
(40)      ret
(41)
(42) ;; Ildiz qism dasturi.
(43) ;;
(44) ;; Maqsad:
(45) ;;  Ushbu qism dasturi sonning kvadrat ildizini hisoblaydi.
(46) ;;
(47) ;; Qiymatlar:
(48) ;;  Ildizi hisoblanish kerak bo'lgan
(49) ;;  son stack orqali qabul qilinadi.
(50) ;;
(51) ;; Natija:
(52) ;;  Hisoblangan kvadrat ildiz EAX da qaytariladi.
(53)
```

```
(54) section  .text
(55)
(56) ildiz:
(57)      enter     4 , 0          ;; xn uchun
(58)      push edx                 ;; Qism dasturda EDX ishlatiladi
(59)
(60)      mov  eax , [ebp+8]       ;; Jo'natilgan qiymatni EAX ga
(61)                               ;; o'zlashtiramiz, ya'ni EAX <- a
(62)      mov  [ebp-4] , eax
(63)      shr  dword [ebp-4] , 1   ;; x1 <- a / 2
(64)
(65) .takrorlanish:
(66)      xor  edx , edx           ;; EDX:EAX ning to'ng'ich 4
(67)                               ;; baytini nollaymiz
(68)      div  dword [ebp-4]       ;; EAX <- a / xn
(69)      add  eax , [ebp-4]       ;; EAX <- xn + a/xn
(70)      shr  eax , 1             ;; EAX <- (xn + a/xn)/2,
(71)                               ;; bu yerda EAX = xn+1
(72)      cmp  eax , [ebp-4]       ;; (EAX = xn) ?
(73)      jz   .topildi
(74)      mov  [ebp-4] , eax       ;; xn <- xn+1
(75)      mov  eax , [ebp+8]       ;; Jo'natilgan qiymatni tiklaymiz
(76)      jmp  .takrorlanish
(77)
(78) .topildi:
(79)      pop  edx
(80)
(81)      leave
(82)      ret                      ;; Orqaga qaytamiz
```

Biz bu yerda asosiy dasturda ham muqaddima va xotimadan foydalandik. Xotima bizni dastur so'ngida stackni tozalash majburiyatidan ozod qiladi. Ikkala qism dasturi muqaddimasida ham ishlatiladigan registrlarning oldingi qiymatlari bitta-bitta `PUSH` yordamida stackka vaqtinchalik saqlandi va xotimada u yerdan `POP` yordamida tiklandi. Ammo qism dasturlarda `PUSH` o'rnida `PUSHA` dan foydalansa ham bo'lar edi. Bu, albatta, dasturchi tanloviga bog'liq. Faqat shuni ta'kidlash lozimki, `PUSHA` qism dasturda ishlatilish yoki ishlatilmasligidan qat'i nazar barcha registrlar qiymatini stackka yuklaydi. Bu esa dastur ishlashini sekinlashtiradi. Masalan, `ildiz` qism dasturini oladigan bo'lsak, unda faqat EDX registrini stackda saqlash kerak xolos. Dasturlash mezoni bo'yicha qism dasturining natijasi EAX da qaytarilgani uchun, ushbu registrning oldingi qiymatini stackda saqlashdan hech qanday ma'no yo'q.

Bu dasturda yana bir e'tiborga molik narsa `CDQ` buyrug'i orqali EAX ning ishorali son sifatida EDX:EAX gacha kattalashtirilganidir. Oldingi dasturlarda biz `DIV` amalidan oldin shunchaki EDX ning qiymatini nolga tenglashtirar edik. Ammo bu safar biz ishorali bo'lishni amalga oshirdik. Gap shundaki, to'rt baytli bo'lishda bo'linuvchining ishora biti EDX da joylashgan bo'ladi. Shuning uchun EAX ning qiymatini ishorali kattalashtirishga to'g'ri keladi.

Xulosa qilib shuni aytishimiz mumkinki, biz bu dastur orqali qismli dasturlash mezonlariga rioya qilgan holda qasturlashni o'rgandik. To'g'ri, dasturdagi ba'zi qatorlar sizga erish va ishni qiyinlashtiradigan bo'lib tuyulgandir. Hatto, «Bundan osonroq yo'li bor-ku» degan fikrni ham o'ylagandirsiz. Biroq mezonlariga rioya qilib tuzilgan dasturlar ulardan keyinchalik foydalanmoqchi bo'lgan boshqa dasturchilar tomonidan alohida olqish bilan qabul qilinadi. Nima qilganda ham aynan qanday yo'sinda dastur tuzish o'z ixtiyoringizga havola!

8.9. Assembler va C dasturlarini bog'lash

Assembler quyi daraja dasturlash tili bo'lganligi tufayli kelajakda dasturni boshidan oxirigacha faqat unda yozishingizga to'g'ri kelmaydi. Katta dasturlar uchun qulay va mukammal bo'lgan yuqori daraja dasturlash tillari bor. Lekin assemblerning registrlar va boshqa kompyuter uskunalari bilan to'g'ridan-to'g'ri ishlash imkoniyati bor. Shuning uchun yuqori daraja tillarida dasturlaganda, kerakli joyda assemblerda tuzilgan qism dasturi chaqiriladi. Shunday yuqori daraja tillaridan biri C. Bu mavzuda C dagi dasturdan turib assemblerda tuzilgan qism dasturni qanday chaqirishni ko'rib chiqamiz va aksincha.

C dasturlash tilida assemblerdan ikki xil foydalanish usuli bor. Birinchi usulda assembler kodi to'g'ridan-to'g'ri C dasturi ichida qism dasturi sifatida yoziladi. Ammo ko'p hollarda C ichida yoziladigan assembler kodi biz o'rganayotgan NASM ko'rinishida bo'lmaydi. Asosan bu turdagi qism dasturlar GAS ko'rinishida yoziladi. GAS ko'rinishi ham yagona emas. Aynan qanday assembler ko'rinishi ishlatilishi foydalanayotgan C kompilyatoringizga bog'liq.

Ikkinchi usul esa biz o'rgangan qism dasturlari bilan ishlash usulidan iborat. Bunda boshqa alohida faylda tuzilgan assembler qism dasturi C dasturidan turib chaqiriladi. Ushbu usuldan foydalanganda barcha ta'kidlab o'tgan qismli dasturlash kelishuvlari o'z kuchida qoladi.

C dasturlash tilida qism dasturi chaqirilganda, qism dasturi quyidagi registrlar qiymatiga o'zgartirish kiritmaydi deb hisoblanadi: EBX, ESI, EDI, EBP, CS, BS, SS va ES. Demak, qism dasturi ichida yuqoridagi registrlardan foydalanmoqchi bo'lsangiz, ularning oldingi qiymatlarini stackda asrab qo'yishingizga to'g'ri keladi.

C dan chaqiriladigan assembler qism dasturi nomi, ya'ni nishon nomi, assemblerda pastki chiziq (_) bilan boshlanishi kerak. Chaqirishda esa C da pastki chiziqsiz murojaat qilinadi. Masalan, ildiz qism dasturi assemblerda `_ildiz` deb nomlanishi kerak. C dasturidan esa `ildiz(...)` sifatida chaqiriladi. Faqat `elf` obyekt fayl andozasi bundan mustasno. Ushbu andozada assemblerdagi nishon nomlari qanday bo'lsa, C da ham shundayligicha ishlatiladi. Demak, `elf` obyekt fayl andozasidan foydalanganda pastki chiziq shart emas.

Endi qiymatlarni C dan assemblerga jo'natish haqida so'z yuritamiz. Bu narsa, albatta, stack orqali amalga oshiriladi:

```
qism_dastur_nomi(jo'natiladigan_qiymat1, ... , n);
```

Qavs ichidagi jo'natiladigan qiymatlar assembler qism dasturi ichida stack orqali qo'lga kiritiladi. Bu qiymatlar stackda teskari tarzda yuklanadi. Birinchi bo'lib n chi qiymat stackka kiritiladi, keyin n-1 va hokazo. Masalan, diskriminant qism dasturini chaqirishdan oldin qiymatlarni quyidagicha stackka yuklagan edik:

```
push dword [c]
push dword [b]
push dword [a]
```

Shu qism dasturini C dasturidan chaqirish uchun quyidagicha yozamiz:

```
diskriminant(a, b, c);
```

Qism dasturi natijasi AL/AX/EAX yoki EDX:EAX registrida qaytarilishi kerak. Agar natija kasr son bo'lsa, u holda ST0 registrida qaytariladi[1]. Bizning diskriminant dasturimiz ham natijani EAX registrida qaytaradi.

```
natija = diskriminant(a, b, c);
```

[1] Bu registr «O'nli kasrlar» bobida muhokama qilinadi.

C dasturlash tilida tenglik (=) belgisi assemblerdagi `MOV` buyrug'i bajaradigan o'zlashtirishni amalga oshiradi. Qism dasturlari esa funksiyalar deb yuritiladi. Shuning uchun ham qism dasturimizning C dagi chaqirilishi matematika kursidagi funksiyalarning berilishiga o'xshaydi, $z=f(x,y)$.

Sanab o'tgan kelishuv mezonlarimiz ko'pgina C kompilyatorlari uchun odatiy hisoblanadi. Lekin C kompilyatorlari har xil firmalar tomonidan ishlab chiqarilgani sababli mezonlari farq qilishi mumkin. GCC kompilyatorini oladigan bo'lsak, u bizga har xil mezonlarni ishlatish imkonini beradi. C dasturlarida kelshuv mezoni funksiya e'lon qilinishida `__attribute__` kalit so'zi orqali beriladi.

> **Eslatma**: C dasturlash tilidagi funksiyalar ular tuzilishi va chaqirilishi oldidan dastur boshida e'lon qilinadi. Buni biz assemblerdagi o'zgaruvchilarni e'lon qilishga o'xshatishimiz mumkin. C da o'zgaruvchilar ma'lum bir turga mansub bo'lishi bilan assemblerdagi o'zgaruvchilardan farq qiladi. Shuning uchun funksiya e'lon qilinganda uning qanday turdagi qiymatni qaytarishi va qanday turdagi qiymatlar unga jo'natilish kerakligi aniq qilib e'lon qilinadi.

Masalan, diskriminant qism dasturidan C da foydalanish uchun uni oldindan quyidagicha e'lon qilamiz:

```
int diskriminant(int, int, int) __attribute__ ((cdecl));
```

Funksiya oldidagi `int`[1] uning qanday turdagi qiymat qaytarishini bildiradi. Qavs ichidagi uchta `int` esa unga jo'natiladigan qiymatlar soni va turini bildiradi. Kalit so'z `__attribute__` dan so'ng esa mazkur qism dasturi uchun qanday kelishuv mezoni qo'llanishi kerakligini bildiruvchi qiymat beriladi. Masalan, `cdecl` qiymati shu paytgacha ko'rib chiqqan mezonlarimizdan foydalanish kerakligini bildiradi. Bundan tashqari `stdcall` qiymati ham mavjud bo'lib, uning `cdecl` dan farqi shundaki, u stack orqali jo'natilgan qiymatlarni qaytish amalga oshirilishidan oldin qism dasturining o'zi stackdan bo'shatishini talab qiladi.

> **Eslatma**: Ba'zi C kompilyatorlarida `__attribute__` kalit so'zi ishlamasligi mumkin. Agar siz ishlatayotgan C kompilyatori ushbu ifodani xato deb e'lon qilsa, `__attribute__ ((cdecl))` ni dasturdan olib tashlang.

Misol tariqasida tenglama dasturimizni qayta ko'rib chiqamiz. Asosiy dasturni C tilida yozamiz va u yerdan turib assemblerda yozilgan qism dasturlarini chaqiramiz. Asosiy dastur *tenglama.c* faylida saqlansin deylik. Qism dasturlari esa oldingi namunada ko'rsatilganidek *qism.asm* faylida turibdi. Agar siz Windowsda dasturlayotgan bo'lsangiz `ildiz` va `diskriminant` nishon nomlarini `_ildiz` va `_diskriminant` ga o'zgartiring yoki `ildiz` va `diskriminant` nishonlarini oddiy `global` bilan emas, balki `tizim_global` yordamida e'lon qiling.

9-namuna: tenglama.c

```
(1)  #include <stdio.h>
(2)
(3)  int diskriminant(int, int, int) __attribute__ ((cdecl));
(4)  unsigned ildiz(unsigned) __attribute__ ((cdecl));
(5)
(6)  int main()
(7)  {
(8)    int a, b, c, x, D;
```

[1] C dasturlash tilida o'zgaruvchi butun son qiymatlarni qabul qilishini bildiradi.

```
(9)    printf("Tenglama koeffitsiyentlarini kiriting (a,b,c) : ");
(10)   scanf("%i %i %i", &a, &b, &c);
(11)
(12)   D = diskriminant(a,b,c);
(13)   if(D < 0)
(14)   {
(15)        printf("Tenglama ildizga ega emas.\n");
(16)        goto tamom;
(17)   }
(18)   if(D == 0)
(19)   {
(20)        x = (int) -b / (2*a);
(21)        printf("X = %i.\n", x);
(22)   }
(23)   else
(24)   {
(25)        D = ildiz(D);
(26)        x = (int) (-b + D) / (2*a);
(27)        printf("X1 = %i, ", x);
(28)        x = (int) (-b - D) / (2*a);
(29)        printf("X2 = %i.\n", x);
(30)   }
(31)
(32) tamom:
(33) return 0;
(34) }
```

Dastur Linuxda quyidagicha yig'iladi va ishga tushiriladi:

```
$ nasm -f elf qism.asm
$ gcc tenglama.c qism.o -o tenglama
$ ./tenglama
```

Windowsda esa:

```
> nasm -f win32 qism.asm
> gcc tenglama.c qism.obj -o tenglama.exe
> tenglama.exe
```

Agar GCC dan boshqa kompilyatordan foydalanayotgan bo'lsangiz, «C kompilyatorlari va obyekt fayllarni ular orqali ulash» ilovasiga qarang.

Endi C da tuzilgan qism dasturlarini assemblerdagi dasturdan chaqirishni ko'rib chiqamiz. Misol tariqasida yuqorida bajargan ishimizni aksini qilamiz, ya'ni ildiz va diskriminant qism dasturlarini C da yozib, ularni assemblerda yozilgan *kvadrat_tenglama.asm* faylidan chaqiramiz. Agar siz Windowsda dasturlayotgan bo'lsangiz, *kvadrat_tenglama.asm* faylidagi barcha `ildiz` va `diskriminant` nishon nomlarini `_ildiz` va `_diskriminant` ga o'zgartiring yoki `ildiz` va `diskriminant` nishonlarini oddiy `extern` bilan emas, balki `tizim_extern` yordamida e'lon qiling. `tizim_extern` makrosi ham `tizim_global` ga o'xshab har xil muammolarning oldini olish masadida kitob muallifi tomonidan tuzulgan va *nasm-io.inc* faylida joylashtirilgan[1].

[1] Batafsil ma'lumot uchun «Mezoniy makrolar» mavzusiga qarang.

10-namuna: qism.c

```
(1)  int diskriminant(int, int, int) __attribute__ ((cdecl));
(2)  unsigned ildiz(unsigned) __attribute__ ((cdecl));
(3)
(4)  unsigned ildiz(unsigned a)
(5)  {
(6)    unsigned xn = 0,formula;
(7)    formula = a / 2;
(8)    while(formula != xn)
(9)    {
(10)        xn = formula;
(11)        formula = (xn + a/xn)/2;
(12)   }
(13) return xn;
(14) }
(15)
(16) int diskriminant(int a, int b, int c)
(17) {
(18) return b*b-4*a*c;
(19) }
```

Dastur Linuxda quyidagicha yig'iladi va ishga tushiriladi:

```
$ nasm -f elf kvadrat_tenglama.asm
$ gcc kvadrat_tenglama.o qism.c -o tenglama
$ ./tenglama
```

Windowsda esa:

```
> nasm -f win32 kvadrat_tenglama.asm
> gcc kvadrat_tenglama.obj qism.c -o tenglama.exe
> tenglama.exe
```

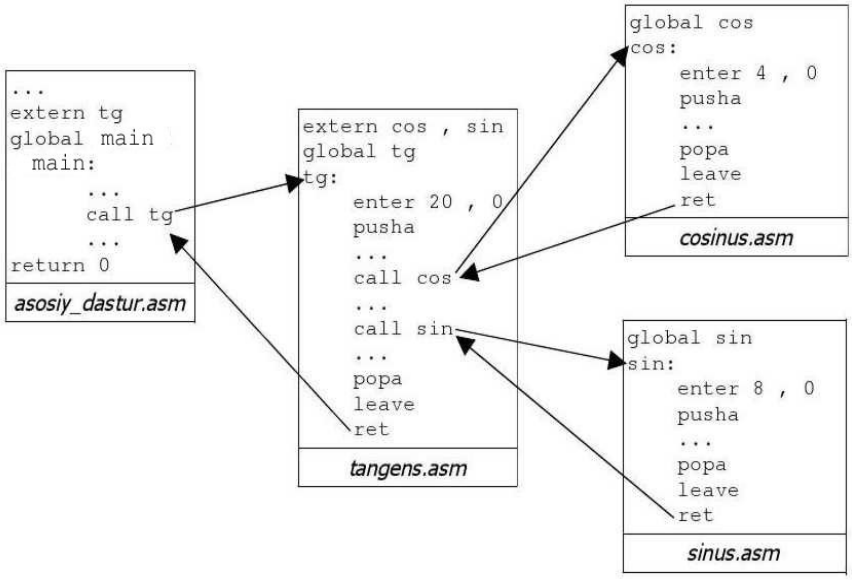
Shunday qilib, C va Assemblerda tuzilgan dasturlar bir-biri bilan qanday hamkorlikda ishlashi batafsil ko'rib chiqildi. Bundan buyog'iga assemblerdagi dasturlaringizda nafaqat o'zingiz tuzgan C qism dasturlaridan foydalanishingiz mumkin, balki C tilining tayyor mezoniy funksiyalarini ham bemalol assemblerda ishlatishingiz mumkin. Bu dasturlashni ancha osonlashtiradi. Misol tariqasida shu paytgacha `chop_et` va `qabul_qil` makro vositalari tomonidan ishlatib kelinayotgan C dagi `printf` va `scanf` qism dasturlarini makrolarning yordamisiz chaqirib ko'ramiz. Masalan `a` o'zgaruvchisining qiymatini chop etish kerak bo'lsin:

```
;; misol.asm
...
andoza    db   "Javob = %i"
...
tizim_global main
tizim_extern printf, scanf
main:
     …
     push dword [a]           ;; a ning qiymati
     push dword andoza        ;; andoza ning manzili
     call printf
     add  esp , 8
     …
     ret
```

8.10. Qism dasturlarida qayta kirish va o'z-o'zini chaqirish tushunchalari

Qismli dasturlashda o'z o'rniga ega bo'lgan qayta kira olish va o'z-o'zini chaqirish tushunchalari ingliz tilida *reentrant* va *recursive* deb ataladi. Bu tushunchalar qism dasturini turli xil holatlarda chaqirilishi bilan bog'liq bo'lib, ba'zi murakkabliklarni keltirib chiqaradi. Shuning uchun ham biz mazkur tushunchalarga batafsil to'xtalishni joiz deb topdik.

Biz shu paytgacha qism dasturlarini asosiy dasturdan turib chaqirar edik. Ammo, chaqirilgan qism dasturi o'z navbatida boshqa qism dasturini chaqirib qolishi mumkin va bu holat n marta ichma-ich sodir bo'lishi mumkin. Masalan, tangensning berilgan burchakdagi qiymatini hisoblaydigan `tg` qism dasturi bo'lsin. `tg` esa o'z navbatida `cos` va `sin` qism dasturlarini chaqiradi, chunki $tg(a)=\frac{\sin(a)}{\cos(a)}$. Rasmda bu qanday ro'y berishi tasvirlangan.



35-rasm.

Agar qism dasturi quyidagi shartlarni bajarsa, u *qayta kira oladigan* qism dastur hisoblanadi:

- Dastur buyruqlari kod bo'limi qiymatlarini o'zgartirmasligi kerak, ya'ni u yerdan qancha ma'lumot kerak bo'lsa o'qishi mumkin, lekin o'zgartirish kiritilishi mumkin emas. Masalan quyidagi dastur bo'lagi qayta kira olish tushunchasiga zid:

```
    jmp   .chetlash
    x     resd  1
.chetlash
    mov   dword[x] , 14
```

  Oxirgi buyruq kod bo'limida e'lon qilingan `x` o'zgaruvchisining qiymatini o'zgartirayapti. Himoyalangan usulda bunga yo'l qo'yilmaydi. Dastur ishga tushirilganida protsessor tomonidan uzib qo'yiladi. Bu shartning mazmuni shundaki, qayta kirishni ta'minlaydigan dasturlar ishga tushganda ular asosiy xotiraga bir nusxada yuklanadi va nechta jarayon bir vaqtning o'zida bu dasturdan foydalanmoqchi bo'lsa ham ular bitta xotira bo'lagiga murojaat qilishadi. Bu yondashish kompyuter ishlash paytida asosiy xotirani ancha tejashga xizmat qiladi. Masalan, bir vaqtning o'zida yigirmatacha dastur `printf` qism

dasturini chaqirsada, `printf` xotiraga bir marta yuklanadi. Dasturlar esa bundan bexabar holda bitta xotira bo'lagiga *qayta-qayta kirishadi*.

- Qism dasturi `.bss` yoki `.data` bo'limlarida e'lon qilinadigan ommaviy o'zgaruvchilardan foydalanmasligi kerak. Ichki qiymatlar uchun stackdan joy ajratilishi kerak.

Qism dasturi *o'z-o'zini chaqirish* imkoniyatiga ega bo'lishi uchun u qayta kira oladigan qism dasturi bo'lishi kerak. Bunday dasturlar qism dasturi chaqirilish kelishuvlariga qat'iy amal qilishi kerak. Bu kabi dasturlash usullari o'zaro bo'lishiladigan kutubxonalar (*dll* fayllar) yoki katta-katta dasturlar tuzishda ishlatiladi. Unixsimon va boshqa zamonaviy operatsion tizimlarda qayta kirish tushunchasi keng qo'llanilgan.

O'z-o'zini chaqirish qism dasturlarida har xil bo'lishi mumkin. Masalan, bir qism dasturi ikkinchisini chaqirganda ikkinchisi ishga tushgach yana birinchisini chaqirishi mumkin:

```
birinchi_dastur:
   …
   call ikkinchi_dastur
   …
   ret

ikkinchi_dastur:
   …
   call birinchi_dastur
   …
   ret
```

Bundan tashqari qism dasturi o'z kodi ichida yana o'zini chaqirishi mumkin:

```
qism_dasturi:
     …
     call qism_dasturi
     …
     ret
```

Bizni asosan oxirgi holat ko'proq qiziqtiradi.

O'z-o'zini chaqiradigan qism dasturlarida bu kabi ichma-ich chaqiruvlarni qachondir nihoyasiga yetkazish uchun `CALL` buyrug'idan oldin biror bir shartni tekshirib turib, shart bajarilganda esa chaqiruvni tarmoqlash buyruqlari orqali chetlab o'tish lozim. Aks holda, dastur bajarilishi hech qachon tugamaydi.

Odatda o'z-o'zini chaqiradigan qism dasturlarini misol orqali yoritishda faktorialni hisoblash masalasi ko'rib chiqiladi. Biz ham an'analarga sodiq qolgan holda o'z-o'zini chaqiradigan faktorial qism dasturini tuzishga harakat qilib ko'ramiz.

Faktorial $n!$ bilan belgilanib, 1 dan to n gacha bo'lgan sonlar ko'paytmasiga aytiladi, ya'ni $n!=1*2*3*...*n$.

11-namuna: faktorial.asm

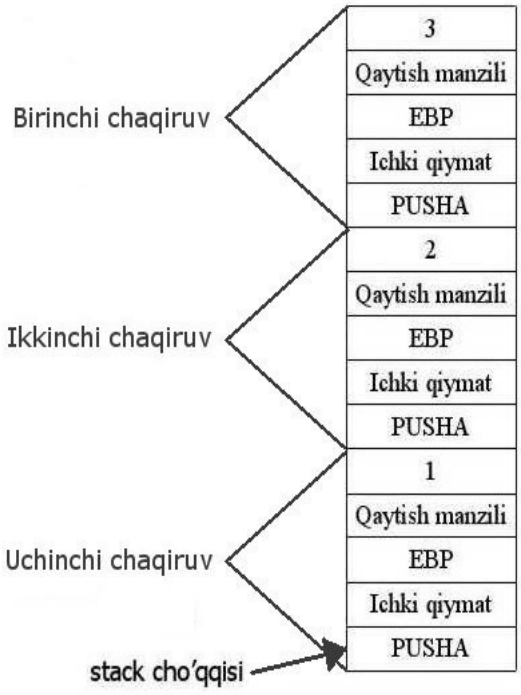
```
(1)  ;; Asosiy dastur
(2)  %include "nasm-io.inc"
(3)
(4)   section .bss
(5)   n   resd 1
(6)
(7)  section .text
```

```
(8)  tizim_global  main
(9)
(10) main:
(11)      chop_et   `n sonini kiriting: `
(12)      qabul_qil `%u`, n
(13)
(14)      push dword [n]
(15)      call faktorial
(16)      add  esp , 4
(17)
(18)      chop_et   `Javob: %u! = %u \n`, [n] , eax
(19)      ret
(20)
(21) ;; Faktorial qism dasturi.
(22) ;;
(23) ;; Maqsad:
(24) ;;  Jo'natilgan qiymatgacha faktorialni hisoblaydigan
(25) ;;  qism dasturi.
(26) ;; Natija:
(27) ;;  Hisoblangan faktorial EAX da qaytariladi.
(28) faktorial:
(29)      enter     4 , 0          ;; Natija uchun vaqtinchalik joy.
(30)      pusha
(31)      mov  eax , [ebp+8]       ;; EAX ← Jo'natilgan qiymat.
(32)      cmp  eax , 1             ;; Agar EAX birga teng bo'lsa,
(33)      jbe  .tamom              ;; O'z-o'zini chaqirish tugatiladi.
(34)      dec  eax
(35)      push eax
(36)      call faktorial
(37)      add  esp , 4
(38)      xor  edx , edx
(39)      mul  dword [ebp+8]       ;; EDX:EAX ← t * (t-1) , t=1...n
(40)
(41)  .tamom:
(42)      mov  [ebp-4] , eax
(43)      popa
(44)      mov  eax , [ebp-4]
(45)      leave
(46)      ret
```

Namunadagi qism dasturi bir oz tushunarsiz bo'lishi mumkin. Undagi 36-qator anchagina o'ylashni talab qilishi aniq. Lekin qism dasturi chaqirilganda stackda qanday hodisalar yuz berishini yaxshi bilgan kishi uchun buni tushunish ancha oson bo'ladi. Boshqa dasturlash tillarida ham o'z-o'zini chaqiruvchi qism dasturlarini tuzish imkoniyati bo'lib, ammo u tillarda stack nimaligi umuman yoritilmaydi va o'sha tillarda dasturlovchi dasturchilar uchun bu kabi dasturlarni tushunish yanada qiyinroq kechadi. Assemblerda esa bu tushuncha stack orqali tushuntirilgani uchun *asl mohiyat* yaqqol namoyon bo'ladi. Shuning uchun assembler tili dasturlash tillari asosi hisoblanadi va uni o'rganish kuchli dasturchi bo'lishni xohlagan kishining birinchi vazifasidir.

Qo'llanilgan g'oya shundaki, qism dasturi har safar jo'natilgan qiymatni bittaga kamaytirib yana qayta o'ziga jo'natadi. Bu holat toki jo'natilayotgan qiymat birga teng bo'lmaguncha takrorlanaveradi, ya'ni dastur bajarilishi 36-qatordan pastga o'tmaydi. 32-qatordagi shart bajarilganda esa `.tamom` nishoniga o'tish yuz beradi va u yerdagi `RET` buyrug'i qism dasturi

qayerdan turib chaqirilgan bo'lsa, o'sha yerga qaytadi. Chaqirilgan joy qism dasturining 36-qatori bo'lganligi sababli, 37-qatorga qaytiladi va undan pastdagi buyruqlar bajariladi. Umuman olganda 37-qatorga qaytish n-1 marta, 16-qatorga esa 1 marta amalga oshiriladi. Shunday qilib n marta takror chaqirilgan qism dasturi stackni rasmda ko'rsatilgan ko'rinishga olib keladi. Rasmda 3 soni bilan chaqirilgan `faktorial` qism dasturining 32-qatordagi shart bajarilgandagi stack holati tasvirlangan. Shart bajarilib dastur qaytishni boshlagandan so'ng stack ham `RET`, `POPA` va `LEAVE` buyruqlari yordamida bo'shatila boshlanadi. Shu orada biz stack bo'ylab yoyilgan 3, 2 va 1 sonlarini 39-qatorda ko'paytirib olamiz. Ko'paytma natija sifatida qaytariladi. Birinchi chaqiruv 1 sonini qaytaradi, ikkinchi chaqiruv 2 sonini qaytadi va uchinchisi esa 6 ni.



36-rasm.

Dasturda qo'llanilgan `PUSHA` va `POPA` buyruqlari ortiqchadek tuyulishi mumkin. Chunki bir qaraganda qism dasturi EAX dan boshqa registrlarning qiymatini o'zgartirmayotganga o'xshaydi. Lekin 39-qatordagi ko'paytirish EDX ga ta'sir ko'satishi mumkin. `PUSHA` va `POPA` dan foydalanganimiz sababli, `enter 4 , 0` orqali vaqtinchalik EAX ning qiymatini saqlash uchun kerak bo'ladigan ichki o'zgaruvchi ajratishimizga to'g'ri keldi. EDX ning qiymati asosiy dasturga kerak bo'lmaganini hisobga olsak, uning o'zgartirilishi yoki shundayligicha qoldirilishining hech qanday ta'siri yo'qdek. Shunday bo'lsada, biz barcha mezon va me'yorlarga to'g'ri keladigan dastur tuzdik va undan nafaqat assemblerda, balki istalgan boshqa dasturlash tilida foydalanish mumkin. Masalan, C da ham faktorial qism dasturini chaqirish mumkin:

```
unsigned n;
scanf("%u",&n);
n = faktorial(n);
printf("%u\n",n);
```

8.11. Chuqur o'rganuvchilar uchun amaliy dastur

Mazkur bobda o'rgangan bilimlaringiz sizga qism dasturlari haqida to'liq ma'lumot beradi. Bu mavzuda ko'riladigan masalalar esa faqat chuqur o'rganuvchilar uchun mo'ljallangan bo'lib, matn ortiqcha izohlarsiz berilgan. O'qish esa o'z tanlovingizga bog'liq.

Qism dasturlariga ancha katta e'tibor qaratayotganimizning sababi undagi g'oyalar dasturlashda hal qiluvchi ahamiyatga ega. Stack va manzillar bilan vositali ishlash shular jumlasidan. Aslini olganda kompilyator orqali yig'iladigan barcha yuqori daraja tillaridagi dasturlar avval assembler tiliga o'giriladi, so'ngra esa assemblerdan mashina tiliga o'giriladi. Shuning uchun assembler tilini bilish dasturchiga katta imkoniyat beradi.

Linux tizimida stack bilan bog'liq bir muammo ancha ko'p munozaralarga sabab bo'lgan edi. Bu muammo hozirgi kunga kelib bartaraf etilgan bo'lsada, biz buni ko'rib chiqishni lozim topdik. Gap shundaki, tizimdagi «yomon» niyatli foydalanuvchi o'zining «yomon» dasturlarini root foydalanuvchisiga tegishli dasturlar orqali ishga tushirish imkoniga ega bo'lgan. Sizga ma'lumki, root foydalanuvchisi katta imkoniyatlarga ega va unga tegishli bo'lgan fayllar ham shular jumlasidandir. Root nomidan g'arazli niyatda ishga tushirilgan dastur esa ko'ngliga kelgan ishni qila olgan. Buni aynan qanday amalga oshirilishini bir kichkina misol tariqasida ko'rib chiqamiz.

C da tuzilgan oddiy dasturga nazar solamiz:

```
int oddiy_qism_dastur(int a, int b)
{
     char qator[10];
     scanf("%s", qator);
     printf("%s\n", qator);
return 1;
}

int main()
{
oddiy_qism_dastur(1, 2);
return 0;
}
```

Bu dastur foydalanuvchidan qatorli o'zgarmas kiritishini talab qiladi va u kiritgan qiymatni yana qayta chop etadi. Bu yerda qism dasturining `qator` o'zgaruvchisi uchun stackdan 10 bayt joy ajratiladi.

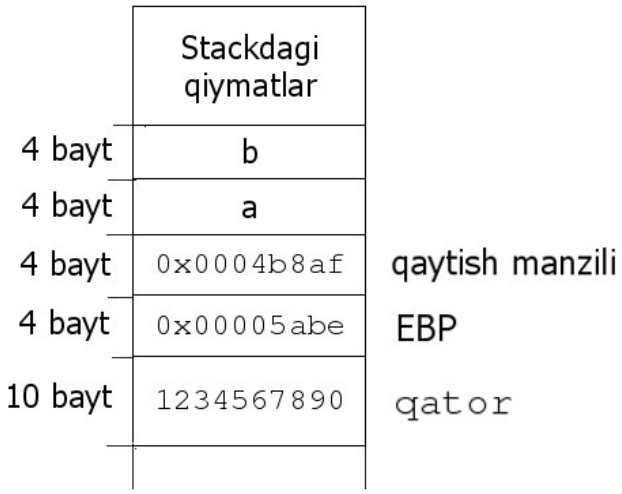
Dasturimizni ishga tushiramiz va 10 baytli qiymat kiritamiz:

```
$ ./dastur
1234567890
1234567890
```

Nima kiritsak, o'sha muammosiz chop etildi. Rasmda kiritilgan qiymat `qator` ga yuklangandan so'ng bo'ladigan stack holati tasvirlangan.

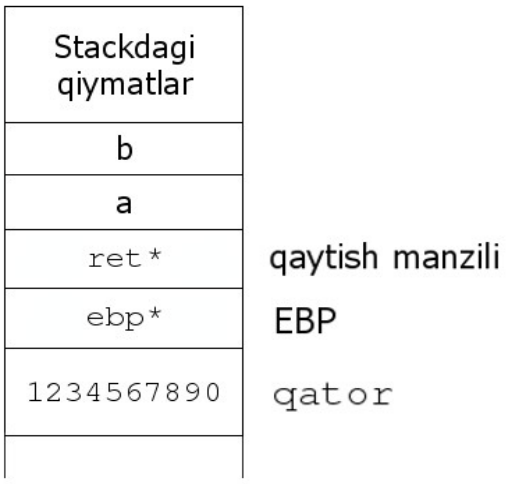
Endi esa dasturni yana ishga tushirib, 18 baytli o'zgarmas kiritamiz:

```
$ ./dastur
1234567890ebp*ret*
1234567890ebp*ret*
Segmentation fault
```



37-rasm.

Ko'rib turganingizdek xato yuz berdi. Buning sababi qiymat `qator` uchun ajratilgan joydan oshib ketdi. Chunki `qator` ga 10 baytli qiymat sig'adi xolos. Qolgan 8 bayt esa stackning boshqa qiymatlari ustiga yoziladi. Ortib qolgan birinchi 4 bayt stackda saqlanayotgan EBP ning qiymati ustiga, ikkinchi 4 bayt esa qaytish manzili ustiga yoziladi, ya'ni [EBP]=`'ebp*'` va [EBP+4]=`'ret*'`. Qism dasturi oxiriga kelib esa `RET` buyrug'i `'ret*'` ni manzil sifatida qabul qilib, unga o'tishni amalga oshirmoqchi bo'ldi. Albatta bu noto'g'ri manzil. 38-rasmda kiritilgan qiymat `qator` ga yuklangandan so'ng bo'ladigan stack holati tasvirlangan.



38-rasm.

Shu o'rinda bir savol tug'iladi. Agar oxirgi 4 bayt sifatida to'g'ri manzil berilsachi?! Yomon niyatlarga mo'ljallangan qism dasturining manzili berilsachi?! Afsuski, yomon dasturchilar bunday imkoniyatga ega bo'lishgan. Bunda qism dasturidan qaytish chaqirilgan joyga emas, balki boshqa manzilga amalga oshirilgan. Buni sinash uchun yuqoridagi dastur fayliga yana bir qism dastur qo'shamiz:

```
void yomon_dastur()
{
     printf("Yomon dastur ishga tushdi\n");
}
```

Endigi vazifa `yomon_dastur` nishonining foydali manzilini aniqlashdan iborat. Buning uchun bajaruvchi faylni GDB[1] tahlilchisiga yuklaymiz va tahlilchining `disas` buyrug'iga funksiya nomini berib, uning assembler kodi va manzillarni qayta tiklaymiz.

```
$ gdb dastur
...
(gdb) disas yomon_dastur
Dump of assembler code for function yomon_dastur:
0x080483ff <yomon_dastur+0>:       push   %ebp
0x08048400 <yomon_dastur+1>:       mov    %esp,%ebp
0x08048402 <yomon_dastur+3>:       sub    $0x8,%esp
0x08048405 <yomon_dastur+6>:       movl   $0x8048513,(%esp)
0x0804840c <yomon_dastur+13>:      call   0x8048308 <puts@plt>
0x08048411 <yomon_dastur+18>:      leave
0x08048412 <yomon_dastur+19>:      ret
End of assembler dump.
```

Ko'rinib turibdiki, qism dasturining boshlanish manzili `0x080483ff` ga teng. Agar dastur ishga tushganda oxirgi 4 bayt sifatida shu manzil kiritilsa, `oddiy_qism_dastur` dan `main` dagi chaqiruv joyiga emas, balki `yomon_dastur` ga qaytish amalga oshiriladi.

> **Eslatma**: Sizda bu manzillar boshqacha bo'lishi mumkin. Muhimi, manzilni GDB orqali aniq belgilash kerak. Aks holda «nayrangimiz» o'tmay qoladi.

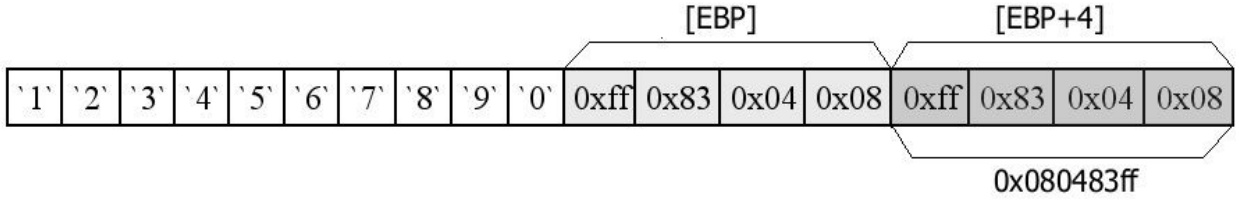
`0x080483ff` manzilini 4 ta belgi sifatida tugmachalar taxtasi orqali kiritish qiyin. Chunki manzildagi raqamlarni beradigan tugmalar bo'lmasligi mumkin. Shuning uchun sonni belgi sifatida chop etuvchi yana bir dastur tuzamiz va uning chop etadigan belgilarni yomon dasturimizga yo'naltiramiz, ya'ni dasturimiz tugmachalar taxtasidan emas, balki boshqa dastur yozuvdan o'qishni amalga oshiradi.

sondan_belgiga.asm

```
(1)  section .data
(2)  yomon_qator    db   '1234567890'   ;; 18 bayt ajratamiz
(3)       times 8   db   0
(4)  uzunlik   equ  $-yomon_qator
(5)  section .text
(6)  global    main
(7)  main:
(8)       mov  dword[yomon_qator+10] , 0x080483ff      ;; EBP uchun
(9)       mov  dword[yomon_qator+14] , 0x080483ff      ;; RET uchun
(10)
(11)      ;; Linuxda xotira bo'lagini andozasiz chop etish.
(12)      mov  eax , 4                  ;; Chop etish.
(13)      mov  ebx , 1                  ;; Qaysi faylga (1 - ekranga).
(14)      mov  ecx , yomon_qator     ;; Chop etiladigan qator manzili.
(15)      mov  edx , uzunlik            ;; Belgilar soni.
(16)      int  0x80                     ;; Linuxga murojaat
(17)
(18)      ret
```

[1] GDB (**G**NU **D**e**b**ugger – GNU tahlilchisi) mashina tilidagi kodni tahlil qiluvchi dastur. Barcha Linux tizimlarida C kompilyatorlari bilan birga o'rnatilgan bo'ladi. Agar sizda o'rnatilmagan bo'lsa, o'rnatish xuddi C kompilyatori o'rnatilganiga o'xshab bajariladi.

Har ehtimolga qarshi EBP ning qiymati ustiga ham `0x080483ff` manzilini yozamiz. `yomon_qator` dagi qiymat rasmda ko'rsatilgandek bo'ladi.



39-rasm.

Rasmda qiymatlar little endian usulida joylashganligi nazarda tutilgan.

Bir dasturning chop etgan qiymati ikkinchisining qabul qilishiga tik chiziq (`|`) orqali yo'naltiriladi.

Qoidasi:

```
$ chop_etuvchi_dastur | qabul_qiluvchi_dastur
```

Demak, kerakli foydali manzil yomon dasturga quyidagicha yo'naltiriladi:

```
$ ./sondan_belgiga | ./yomon_dastur
1234567890????#
Yomon dastur ishga tushdi
Segmentation fault
```

Ish uddalandi! Garchi `Segmentation fault` xatosi yana yuz bergan bo'lsada, `yomon_dastur` o'z «shumligini» bajarib bo'ldi. E'tibor bering, bu qism dasturi dasturning hech qayeridan chaqirilmadi. Uni biz tashqaridan turib `sondan_belgiga` yordamida ishga tushirdik.

IX bob
Makro vositalar

Ko'pgina dasturlash tillarida bo'lgani kabi assemblerda ham makro vositalar taqdim etilgan. Dasturlash jarayonini osonlashtirishga mo'ljallangan o'zgarmaslar va o'zgarmas ifodalar bilan ishlovchi direktivalar *makro vositalar* deb yuritiladi. Makro direktivalar yuqorida ko'rib chiqilgan `DB` yoki `EQU` kabi assembler direktivalar bilan chalkashtirilmasligi kerak. Chunki makro vositalar dastur yig'ilishining ilk bosqichi bo'lmish *preprotsessorda* ishga tushadi. Assembler direktivalari esa dastur yig'ilishining ikkinchi, ya'ni asosiy bosqichida NASM tomonidan ishga tushiriladi. Ammo makro va assembler direktivalari vazifasining o'xshash bo'lganligi va mashina tiligacha yetib bormasligi (mashina tiliga o'girilmasligi, ya'ni protsessor buyruqlariga hech qanday aloqasi yo'qligi), ularning bir xil nomlanganligiga sabab bo'lsa ajab emas.

Makro direktivalar orqali istalgan ifodani belgilab olish mumkin. Bunda ifodaga nom berib, so'ngra shu nom dastur davomida qo'llanadi. Ifoda nomi *makro* deb yuritiladi va dastur yig'ilayotganida barcha makrolar mos ifodalarga preprotsessor tomonidan «ko'r-ko'rona» almashtirib chiqiladi. Ko'r-ko'rona deyilishiga sabab preprotsessor hech qanday hisoblashlarni ifoda ustida bajarmaydi va ifodadagi xatolarga ham e'tibor bermaydi (makro vositalar berilish qoidasida yo'l qo'yilgan xatolardan tashqari, albatta).

Makrodan dasturda foydalanish uchun uni avval aniqlash kerak. Makro dasturning istalgan yerida aniqlanishi mumkin. Makrolar faqatgina ular aniqlangan dastur satridan so'ng qo'llanishi mumkin. Shu tomondan ular nishonlardan farq qiladi. Makrolarga beriladigan nomlar nishonlarni nomlash qoidasiga amal qiladi. Ko'pincha dasturchilar makrolarni nomlashda ular nishon nomlaridan ajralib turishi uchun katta harflardan foydalanadilar.

Assemblerda makro direktivalar doim foiz (`%`) belgisi bilan boshlanadi.

Assemblerda juda ko'p makro direktivalar mavjud bo'lib, ushbu bobda ulardan eng zarurlari o'rganiladi.

9.1. Bir satrli makrolar

Bir satrli makrolar eng oddiy turdagi makrolar sanalib, ular dasturning faqat bitta satrida aniqlanadi. Ammo satr oxirida teskari qiya chiziq `(\)` qo'ygan holda keyingi satrda aniqlashni davom ettirish mumkin. Preprotsessor uni bitta satr sifatida qabul qiladi.

9.1.1. `%DEFINE` makro direktivasi: Umumiy uslub

Makrolarni aniqlashning asosiy direktivasi bo'lib, quyidagicha ishlatiladi:

```
%define   makro_nomi     ifoda
```

Buning o'ng'aylik tomoni shundaki, katta va eslash qiyin bo'lgan ifodalarga oddiygina qilib nom berib, shu nomdan keyinchalik foydalanish mumkin. Masalan:

```
%define   UZUNLIK   100
```

Bu yerda UZUNLIK makrosi 100 soni bilan aniqlandi yoki boshqacha qilib aytilganda UZUNLIK makro, 100 esa uning qiymati hisoblanadi. Bu yerda sonli o'zgarmas ifoda bo'lib kelayapti. %DEFINE makro direktivasining EQU assembler direktivasidan nima farqi bor degan savol tug'ilgan bo'lishi mumkin. Gap shundaki, EQU ga qiymat sifatida faqat o'zgarmas yoki o'zgarmas ifoda berilishi mumkin. EQU berilgan ifodaning qiymatini o'sha yerning o'zida hisoblaydi va natijani son sifatida belgilab oladi. Masalan:

```
qator     db   "salom",0
UZUNLIK   equ  $-qator
…
mov  eax , UZUNLIK
```

Oxirgi buyruq quyidagicha talqin qilinadi:

```
mov  eax , 6
```

Demak, EAX ga qator ning uzunligi yuklanadi. Endi %DEFINE orqali sinab ko'ramiz:

```
qator     db   "Salom",0
%define   UZUNLIK   $-qator
…
mov  eax , UZUNLIK
```

Oxirgi buyruq quyidagicha talqin qilinadi:

```
mov  eax , $-qator
```

Bunda ham EAX ga qandaydir qiymat yuklanadi, ammo bu qiymat qator ning uzunligini bermaydi. Nega shunday ekanligini o'zingiz toping.

Misollar:

```
%define   OMMAVIY   global
%define   BOSHLANISH     _main
…
OMMAVIY BOSHLANISH
BOSHLANISH:
…
```

Quyidagiga almashtiriladi:

```
global _main
_main:
…
```

Agar xohlasangiz assemberdagi barcha buyruqlarni o'zbekcha nomlab chiqishingiz mumkin. Masalan:

```
%define   YUKLA     mov
%define   GA   ,
%define   NI
…
YUKLA eax GA ebx NI
```

Quyidagiga almashtiriladi:

```
mov  eax , ebx
```

Agar makro nomi katta yoki kichik harflar bilan yozilishini farqlanmasligini xohlasangiz `%DEFINE` o'rniga `%IDEFINE` dan foydalaning.

Makro vositalardan qismli dasturlashda ham unumli foydalanish mumkin. Masalan, qism dasturga jo'natilgan qiymatlarga yoki qism dasturning ichki qiymatlariga EBP registri orqali murojaat qilinadi. Ammo qiymatlarni bu kabi foydali manzillarni hisoblash evaziga qo'lga kiritish murakkab va dasturga chalkashliklarni olib kirishi mumkin. Buning o'rnida qiymatlarga inson uchun tushunarli bo'lgan nomlar orqali murojaat qilish osonroq. Bu borada bizga makro vositalar yordam beradi, ya'ni EBP orqali beriladigan foydali manzil ifodalarini makrolar bilan aniqlab chiqiladi. «Qismli dasturlash» bobining 8-namunasidagi qism dasturni olaylik. U yerda dastur boshida quyidagi makrolarni aniqlash mumkin:

```
%define   c    [ebp+16]
%define   b    [ebp+12]
%define   a    [ebp+8]
```

Shundan so'ng dasturning istalgan yerida makrolardan foydalansa bo'ladi. Masalan, *qism.asm* faylidagi 22–24 sartlarni quyidagicha o'zgartirish mumkin:

```
mov  ecx , c
mov  ebx , b
mov  eax , a
```

Makrolarni qiymat qabul qila oladigan qilib ham aniqlash mumkin.

Qoidasi:

```
%define   makro_nomi(qiymat1, … )  ifoda
```

Masalan, doim $y=x^2+4x-7$ ning ma'lum bir nuqtadagi qiymatini hisoblash kerak bo'ldi deylik.

```
%define   y(x) x*x + 4*x - 7
```

Bu yerda `y` makroning nomi, `x` - u qabul qiladigan qiymat , ifoda bo'lib esa `x*x + 4*x - 7` kelmoqda. Ushbu aniqlashdan so'ng dasturning istalgan yerida `y` makrosidan foydalanish mumkin.

```
mov  eax , y(5)
mov  ebx , y(-9) + 10
```

Bular preprotsessor tomonidan quyidagiga almashtiriladi:

```
mov  eax , 5*5 + 4*5 - 7
mov  ebx , -9*-9 + 4*-9 - 7 + 10
```

Yuqoridagi o'zgarmas ifodalar esa dasturni yig'ish jarayonida NASM tomonidan hisoblanib natijaviy sonlarga almashtiriladi. Masalan, 5*5 + 4*5 - 7 ifodasi 38 ga almashtiriladi. Shu joyda yana bir narsani ta'kidlash lozimki, NASM kasr sonli ifodalarni hisoblay olmaydi!

Eslatma: Preprotsessor hech qanday arifmetik yoki mantiqiy hisoblashlarni amalga oshira olmaydi. U faqat mos almashtirishlarni bajaradi xolos.

Qiymat qabul qila oladigan makrolar qismli dasturlashda ham foydali bo'lishi mumkin. Masalan, qism dasturiga jo'natiladigan qiymatlarning soni juda ko'p bo'lsa, ularning har biriga nom berib makro orqali aniqlab bo'lmaydi. Bunday qism dasturiga qiymatlarni chop etadigan qism dasturini misol qilib keltirish mumkin. Ushbu qism dasturi chop etish uchun bitta ham, o'nta ham, yuzta ham qiymat qabul qilishi mumkin, ya'ni unga qancha qiymat jo'natilishi oldindan ma'lum emas. Bunday holatda qism dasturi unga jo'natilgan qiymatlarga [ebp+x*4+8] ifodasi yordamida murojaat qiladi ($x \in [0,\dots,n]$). Har safar bunday katta ifodadan dasturda foydalanmaslik uchun kichkina makro yozish kifoya.

```
    %define  Jqiymat(x) dword[ebp + x*4 + 8] ;; Jo'natilgan qiymatlar
    …
    xor  ebx , ebx
takror:
    mov  eax , JQiymat(ebx)
    ;; jo'natilgan qiymat ustida amallar kodi
    inc  ebx
    jmp  takror
```

Shu kabi makrodan ichki qiymatlar uchun ham aniqlasa bo'ladi.

```
    %define   IQiymat(x)     dword[ebp- x*4 - 4]   ;; Ichki qiymatlar
```

Faqat bu holda `IQiymat` o'zgarmas bilan ishlatilishi mumkin[1].

Qiymat qabul qiladigan makrolar bir xil nom bilan ko'p marta qayta e'lon qilinishi mumkin. Bunda ular qabul qiladigan qiymatlar soniga qarab farqlanadi. Masalan, quyidagi makro `IQiymat` dan boshqa hisoblanadi.

```
    %define   IQiymat(x,y)   [ebp - x*y - 4]
    …
    mov  di , IQiymat(0,2)        ;; [ebp - 0*2 - 4]
    mov  si , IQiymat(1,2)        ;; [ebp - 1*2 – 4]
    mov  eax , IQiymat(4)         ;; dword[ebp - 4*4 - 4]
```

Bir marta aniqlangan makroni boshqa qiymat bilan qayta aniqlash mumkin:

```
    %define   IKKI 2
    son  db   IKKI                ;; son    db   2
    %define   IKKI "ikkinchi",0
    qator     db   IKKI           ;; qator  db   "ikkinchi",0
```

9.1.2. `%XDEFINE` makro direktivasi: Kengaytirilgan uslub

Ushbu makro direktivasi `%DEFINE` ga o'xshaydi. Yagona farq shundaki, agar ifoda o'rnida ham makro kelsa, `%XDEFINE` ifoda bo'lib kelgan boshqa makroning qiymatini oladi. Masalan:

```
    %define  QIYMAT1    1
    %define  QIYMAT2    QIYMAT1 ;; QIYMAT2 va QIYMAT1 bir narsa bo'ldi
```

[1] Nega bunday bo'lishini bilish uchun «Qismli dasturlash» bobining «Vositali manzillash» mavzusiga qarang.

```
%define  QIYMAT1    2
…
mov  eax , QIYMAT2
```

Bu yerda preprotsessor ikki marta almashtirishni amalga oshiradi. Birinchisida `QIYMAT2` makrosini `QIYMAT1` ga almashtiriladi, ikkinchisida esa `QIYMAT1` ni 2 ga. Endi `%XDEFINE` bilan sinab ko'ramiz:

```
%define   A    100
%xdefine  B    A          ;; B = 100
%define   A    50
…
mov  eax , B              ;; EAX ← 100
```

`%XDEFINE` u orqali aniqlanayotgan `B` ning qiymati aniqlash uchun darhol o'zi almashtirishni amalga oshiradi.

9.1.3. `%UNDEF` makro direktivasi: Aniqlangan makrolarni bekor qilish

`%UNDEF` makro direktivasi undan oldin aniqlangan makrolarni bekor qiladi. Masalan:

```
%define   A    "abcdef"
…
%undef    A
qator     db   A          ;; Preprotsessor ishlamaydi.
```

9.1.4. `%ASSIGN` direktivasi: Makro o'zgaruvchilar

Ushbu makro direktiva ham `%DEFINE` va `%XDEFINE` kabi bir satrli makroni aniqlaydi. Ammo uning farqi shundaki, makroning qiymati butun son yoki sonli ifoda ko'rinishida berilishi shart. Ifoda esa aniqlashning o'zidayoq tezda hisoblanib natijaviy songa almashtiriladi. Makro vositalardan foydalanayotganda `%ASSIGN` sonli qiymatlarni o'zlashtira oladigan o'zgaruvchi bilan ishlash muhitini yaratib beradi. Masalan:

```
%assign   x    5          ;; x=5
%assign   x    x+10       ;; x=x+10=5+10=15
```

Mazkur direktiva vazifasi assemblerdagi `MOV` buyrug'iga o'xshaydi. Lekin makro vositalarda hech qanday xotiradan joy ajratiladigan haqiqiy o'zgaruvchi haqida gap bo'lishi mumkin emas. Ushbu direktivaning foydali tomonlari «Ko'p satrli makrolar» mavzusida ko'rib chiqiladi.

9.1.5. `%DEFSTR`, `%STRLEN` va `%SUBSTR` makro direktivalari: Qatorli o'zgarmaslar bilan ishlash

Ma'lumki, makroga qiymat sifatida qatorli o'zgarmas ham berish mumkin.

```
%define   QATOR     "Dunyo"
```

`%DEFSTR` direktivasi ham `%DEFINE` ga o'xshaydi, ammo u faqat makroni qatorli o'zgarmas orqali aniqlashga mo'ljallangan. Shunnig uchun unga qatorli o'zgarmas qo'shtirnoqlarsiz beriladi.

```
%defstr   QATOR     Dunyo
```

NASM da operatsion tizim o'zgaruvchilarining qiymatiga murojaat qilish imkoniyati mavjud. Bu o'zgaruvchi nomi oldiga foiz va undov (`%!`) belgilarini qo'yish bilan amalga oshiriladi. Masalan, Windowsda `WINDIR` degan o'zgaruvchi bo'lib, unda doim tizim o'rnatilgan direktoriyaning mutlaq yo'li saqlanadi. Masalan, `WINDIR="C:/Windows"`.

Agar dastur uchun shunga o'xshash qiymatlar kerak bo'lsa, `%DEFSTR` yordamida ularni belgilab olish mumkin.

```
%defstr   WinXP     %!WINDIR
```

`%STRLEN` direktivasi makroni qatorli o'zgarmasning uzunligi bilan aniqlaydi. Masalan:

```
%strlen   UZUNLIK   "Dunyo va Men"
```

`UZUNLIK` 12 ga teng. Qatorli o'zgarmas o'rniga boshqa makro ham bo'lishi mumkin. U holda boshqa makroning qiymati qatorli o'zgarmas bo'lishi kerak. Masalan:

```
%strlen   UZUNLIK   QATOR
```

`%SUBSTR` direktivasi berilgan qatorli o'zgarmasini ma'lum qismini ajratib oladi. Masalan:

```
%substr   QISM "Dunyo", 1          ;; QISM = 'D'
%substr   QISM "Dunyo", 2          ;; QISM = 'u'
%substr   QISM "Dunyo", 3, 3       ;; QISM = 'nyo'
```

Qatorli o'zgarmasdan keyin keladigan birinchi son nechanchi belgidan boshlab ajratish kerakligini bildiradi. `%SUBSTR` uchun birinchi belgining tartib raqami birga teng. Ikkinchi son esa aynan nechta belgi ajratish kerakligini bildiradi. Agar u berilmasa bitta belgi ajratiladi. Bu yerda ham qatorli o'zgarmas o'rnida boshqa makro kelishi mumkin, faqat boshqa makroning qiymati ham qatorli o'zgarmas bo'lishi shart.

9.1.6. `%INCLUDE` makro direktivasi: Boshlang'ich fayllarni dasturga qo'shish

`%INCLUDE` direktivasi berilgan boshlang'ich faylni o'zi yozilgan dastur fayliga qo'shadi. Fayl nomi qatorli o'zgarmas sifatida beriladi. Propreprotsessor ushbu direktiva yozilgan satrdan boshlab berilgan boshlang'ish fayldagi kodni o'sha yerga shunchaki ko'chirib qo'yadi.

Boshlang'ish fayl dastur yoziladigan oddiy fayl bo'lib, keyinchalik `%INCLUDE` orqali boshqa dastur fayllariga qo'shishga mo'ljallangan bo'ladi. Odatda boshlang'ish faylda makro aniqlashlar joylashgan bo'ladi. Boshlang'ich faylning kengaytmasi `.inc` bo'lishi odat tusiga kirgan. Biz shu paytgacha foydalangan `nasm-io.inc` fayli ham shular jumlasidandir.

Ushbu direktivaga tegishli NASM yig'uvchisining `-p` va `-I` kalitlari bo'lib, ular haqidagi batafsil ma'lumot uchun «Buyruqlar qatorida NASM buyrug'i kalitlari» mavzusiga qarang.

9.2. Makro takrorlanish

«Assembler tili asoslari» bobida takrorlanishlarni ko'rib chiqilgan edi. Dasturlashda takrorlanish deganda ma'lum bir dastur qismining ko'rsatilgan son marta takror-takror bajarilishi tushunilishi ma'lum. Preprotsessor ham bizga shunga o'xshash imkoniyatni beradi. Gap shundaki, `%REP` va `%ENDREP` makro direktivalari bilan chegaralangan dastur qismi preprotsessor orqali ko'rsatilgan son marta takror-takror dasturga qo'yib chiqiladi. Takrorlanishlar soni `%REP`

ga qiymat sifatida beriladi. Bu qiymat oldin aniqlangan bir satrli makro, sonli o'zgarmas yoki o'zgarmas ifoda bo'lishi mumkin. Masalan: EAX=0 va ECX=3 bo'lsin, unda

```
%rep 3
add  eax , ecx
dec  ecx
%endrep
```

Preprotsessor tomonidan quyidagicha almashtiriladi:

```
add eax, ecx
dec ecx
add eax, ecx
dec ecx
add eax, ecx
dec ecx
```

Ushbu buyruqlar $a_1=1, d=1$ bo'lgandagi arifmetik progressiyaning dastlabki 3 ta hadining yig'indisini hisoblaydi, ya'ni EAX=3+2+1=6. Albatta, buni makro direktivalar yordamisiz ham assemblerdagi takrorlanish buyruqlari bilan bajarilsa bo'ladi:

```
loop_boshi:
     add  eax , ecx
     loop loop_boshi
```

Natijada hech qanday farq bo'lmaydi, ya'ni EAX=6. Ammo, `%REP` bilan bo'lgan holda dastur kodi bir qancha kattaroq bo'ladi va u xotiradan ko'proq joy egallaydi. Chunki ikkita buyruq oltitaga almashtiriladi. `LOOP` bilan bo'lgan holda esa ikkita buyruq ikkitaligicha qolaveradi. Har ikkita yechimning ham o'ziga yarasha ustunlik va kamchilik tomonlari mavjud. Masalan, protsessor ketma-ket qo'yib chiqilgan oltita buyruqni bajarishga uch marta tarorlanishni amalga oshirishdan ko'ra kamroq vaqt sarflaydi. Chunki `ADD` buyrug'i bajarib bo'lingach, `LOOP` oldin `ECX` ning qiymatini bittaga kamaytiradi, so'ng o'tishni amalga oshirish uchun EIP ga `loop_boshi` manzilini yuklaydi. Undan tashqari CS registri bilan ham bir qator tekshiruv ishlari olib boriladi. Chunki o'tish boshqa dastur bo'limga yo'naltirilgan bo'lishi ham mumkin. Protsessorlar hali bir qaraganda qaysi bo'limdagi nishon ko'zda tutilayotganini aniqlash darajasida aqlli emas. Qisqasini aytganda `%REP` ham, tarmoqlash va takrorlash buyruqlari ham vaziyatga qarab tanlanadi. Bu dasturchiga bo'g'liq. Muhimi mohiyatini tushunib olishda.

Boshqa tomondan `%REP` makro direktivasini `TIMES` assembler direktivasiga ham o'xshatish mumkin. Masalan:

```
qator     times 26  db   'A'
```

Bu yerda `qator` uchun 26 bayt joy ajratilib, har bir baytga `'A'` harfi o'zlashtiriladi. Agar har bir baytga mos ravishda lotin alifbosidagi harflar ketma-ket o'zlashtirilishi kerak bo'lsa `TIMES` yordam bera olmaydi. `%REP` orqali esa bu quyidagicha amalga oshiriladi:

```
%assign   harf 'A'
     qator:
%rep 26
     db   harf
%assign   harf harf + 1
%endrep
```

Bu preprotsessor tomonidan quyidagiga almashtiriladi:

```
qator:
     db   'A'        ;; harf = 'A'
     db   'B'        ;; harf = 'A' + 1 = 65 + 1 = 66 = 'B'
     db   'C'        ;; harf = 'B' + 1 = 'C'
     …
```

Shundan so'ng `qator` = "ABCDEFIGNKLMNOPQRSTUVWYZ" bo'ladi.

Xulosa qilib shuni aytish mumkinki, `%REP` asosan preprotsessor jarayonida bajarilishi kerak bo'lgan ishlarni amalga oshirishda foydalaniladi. Ushbu makro direktivaga doir misollar «`%ROTATE` makro direktivasi» mavzusida ko'proq keltirilgan.

9.3. Ko'p satrli makrolar

Ko'p satrli makrolarni dasturlashdagi qism dasturlariga o'xshatish mumkin. Bu yerda farq shundaki, ko'p satrli marko chaqirilganda, makro joylashgan yerga o'tish amalga oshirilmaydi, aksincha preprotsessor makro chaqiriq o'rniga ko'p satrli makro kodini almashtirib qo'yadi.

Qoidasi:

```
%macro    makro_nomi     qabul_qilinadigan_qiymatlar_soni
…
;; ko'p satrli makro kodi
…
%endmacro
```

`%MACRO` va `%ENDMACRO` bilan chegaralangan satrlar ko'p satrli makro dastur kodi hisoblanib, u yerda istalgan dastur bo'limida uchraydigan kod yoki bir satrli makrolar bo'lishi mumkin. Masalan, tuzayotgan dasturingizning ko'p yerida nuqul EAX va EBX qiymatlarini bir-biri bilan almashtiradigan kod ishlatilayapti deylik:

```
…
push eax
mov  eax , ebx
pop  ebx
…
push eax
mov  eax , ebx
pop  ebx
…              ;; va shunaqa yana ko'p
```

Biroq har safar kerakli joyda ushbu buyruqlar ketma-ketligini qayta-qayta yozish ko'p vaqtni oladi. Bu ko'p satrli makro orqali quyidagicha hal qilinadi:

```
%makro    almash 0
     push eax
     mov  eax , ebx
     pop  ebx
%endmacro
```

Bu yerda `almash` ko'p satrli makroning nomi, undan keyingi 0 esa u hech qanday qiymat qabul qilmaydi degani. Ushbu aniqlashdan so'ng, dasturning istalgan yerida `almash` ni ishlatishimiz mumkin:

```
…
almash
…
almash
…
```

`almash` qaysi registrlar bilan ishlashi aniq bo'lgani uchun hech qanday qiymat qabul qilmaydi. Endi vazifani bir oz murakkablashtiramiz, ya'ni `almash` faqat EAX va EBX ni emas, balki istalgan ikkita registrning qiymatlarini bir-biri bilan almashtirsin. Registrlar unga qiymat sifatida beriladi, ya'ni u ikkita qiymat qabul qiladi.

```
%macro    almash 2
     push %1
     mov  %1 , %2
     pop  %2
%endmacro
```

Ko'p satrli makrolarga jo'natilgan qiymatlar makroning ichida `%1`, `%2` va hokazolar orqali qo'lga kiritiladi. Preprotsessor ularni shunchaki jo'natilgan qiymatlarga almashtirib chiqadi. `almash` esa quyidagicha chaqiriladi:

```
almash    esi , edi      ;; (1)
almash    ecx , edx      ;; (2)
almash                   ;; (3)
```

Oxirgi `almash` ning ishlatilishida birinchi ko'rib chiqqan ko'rinish nazarda tutilgan. Bir satrli makrolarda bo'lgani kabi bu yerda ham bir xil nomga ega bo'lgan makrolar har xil aniqlanishi mumkin. Preprotsessor ularni qabul qiladigan qiymatlar soniga qarab farqlaydi. Shunday qilib yuqoridagi makro qo'llanishlar quyidagiga almashtiriladi:

```
push esi                 ;; (1)
mov  esi , edi           ;; (1)
pop  edi                 ;; (1)
push ecx                 ;; (2)
mov  ecx , edx           ;; (2)
pop  edx                 ;; (2)
push eax                 ;; (3)
mov  eax , ebx           ;; (3)
pop  ebx                 ;; (3)
```

Agar registrlar bilan qo'shib 4 baytli nishonlarni ham `almash` ga berilishi kerak bo'lsa, u holda yana bir oz o'zgartirish kiritishga to'g'ri keladi:

```
%macro    almash 2
     push dword %1
     mov  dword %1 , %2
     pop  dword %2
%endmacro
```

9.3.1. Qiymat berish va qabul qilish usullari

Ko'p satrli makrolarga qiymatlar vergul (,) orqali ajratib beriladi. Masalan:

```
almash    [son] , edx
```

Ammo ba'zan makro berilgan ikkita qiymatni bitta qiymat sifatida qabul qilishi kerak bo'lib qolishi mumkin. Qatorli o'zgaruvchilarni e'lon qilishni osonlashtiruvchi makrolarni ko'rib chiqamiz.

```
%macro    E_LON_QIL 2
%1   db   %2
%endmacro
…
E_LON_QIL abs, "Bu qator"
```

Bu quyidagiga almashtiriladi:

```
abs  db   "Bu qator"
```

Qatorli o'zgarmas oxiriga nol qo'ymoqchi bo'lsak yoki o'zgaruvchiga ikki-uchta boshlang'ich qiymat bermoqchi bo'lsak, makro ularni uchinchi yoki to'rtinchi qiymatlar sifatida qabul qiladi va ularga `%3` va `%4` orqali murojaat qilishga to'g'ri keladi. Biz esa bir nechta qabul qilingan qiymat bitta qiymat sifatida kelishini istaymiz. Buning ikki xil yechimi bor. Birinchi usul vergul bilan ajratilgan, lekin bitta qiymat sifatida qabul qilinishi kerak bo'lgan qiymatlarni shaklli qavs (`{}`) ichiga olishdan iborat. Masalan:

```
E_LON_QIL abs, {"Bu qator",0}
E_LON_QIL bayt, {65,87,92}
```

Ikkinchi usul makroning aniqlanishiga o'zgartirish kiritishdan iborat. Bunda qabul qilinadigan qiymatlar sonidan so'ng qo'shuv (`+`) belgisi qo'yiladi.

```
%macro    E_LON_QIL 2+
%1   db   %2
%endmacro
```

Oxirgi bo'lib berilishi kerak bo'lgan qiymatdan keyin beriladigan barcha qiymatlar oxirgisiga qo'shib olinadi. Boshqacha qilib aytilganda, barcha ortiqcha kelgan qiymatlar ajratib turuvchi vergullari bilan birga oxirgi qiymatga yelimlanadi.

Agar preprotsessor makrolar va makro direktivalarni ko'r-ko'rona haqiqiy qiymatlarga almashtirishini hisobga olsak, ba'zi foydali imkoniyatlar kelib chiqadi. Ko'pincha qatorli o'zgaruvchi e'lon qilinganda, uning uzunligi ham `EQU` orqali belgilab qo'yilar edi. Shu imkoniyatni `E_LON_QIL` makrosiga qo'llasak bo'ladi, ammo uzunlik uchun ham har safar belgilash nomini berishga to'g'ri keladi. Uzunlik esa ba'zida kerak, ba'zida esa kerak emas. Shuning uchun uzunlikka nom berishni `E_LON_QIL` makrosi o'z zimmasiga olgani ma'qul, ya'ni u har doim bir xil qoida asosida uzunlikni belgilovchi o'zgarmasga nom beradi. Masalan, bu nom quyidagicha berilishi mumkin: *nishon_nomi*_UZN.

```
%macro    E_LON_QIL 2+
     %1   db   %2
     %1_UZN    equ  $-%1
%endmacro
…
E_LON_QIL abs, "Bu qator",0
```

Endi istalgan paytda `abs_UZN` orqali `abs` ning uzunligiga murojaat qilsak bo'ladi.

Yana bir qabul qilinadigan qiymatlar sonini berish usuli *noaniq usul* bo'lib, agar makro nechta qiymat qabul qilishi oldindan ma'lum bo'lmasa, eng kam va eng ko'p qabul qilishi mumkin bo'lgan qiymatlar chegarasi chiziqcha (`-`) bilan ajratib beriladi.

```
%macro    yakun     0-1
;; makro kodi
%endmacro
```

Demak, bu makroga qiymat jo'natish ham jo'natmaslik ham mumkin. Yuqori chegarani yulduzcha (`*`) bilan bersa ham bo'ladi. Bu makroning qabul qiladigan qiymatlar soni cheqaralanmaganligini bildiradi. Masalan, sizga ma'lum bo'lgan `chop_et` va `qabul_qil` ko'p satrli makrolar ham shu tarzda e'lon qilingan.

```
%macro    chop_et 1-*
;; makro kodi
%endmacro
```

Ushbu aniqlashdan keyin `chop_et` dan foydalanganda unga eng kamida bitta qiymat jo'natish kerak bo'ladi. Birinchi qiymat esa har doim qatorli o'zgarmas sifatida beriladigan andoza hisoblanadi. Andozadan so'ng chop etish uchun istalgancha qiymat berilishi mumkin. Shuning uchun `chop_et` ni noaniq usulda aniqlashga to'g'ri keladi.

Ko'p satrli makrolarda ular qabul qiladigan qiymatlar sonidan kamroq qiymat jo'natilgan taqdirda, berilmagan qiymatlar uchun yashirin qiymatlarni ko'rsatish imkoniyati bor. Bunda yashirin qiymatlar makroni aniqlash paytida qabul qilinadigan qiymatlar sonidan keyin beriladi. Masalan:

```
%macro    chaqiruv  1-4  eax, ebx, ecx
;; makro kodi
%endmacro
```

Ushbu makro bittadan to'rttagacha qiymat qabul qiladi va ikkinchi, uchinchi yoki to'rtinchi qiymatlar berilmagan taqdirda ular o'rnida mos ravishda EAX, EBX va ECX ni qo'llaydi.

Qismli dasturlash bobidan ma'lumki, assemblerda qism dasturini chaqirish bir oz ko'p mehnatni talab qiladi. Shundan kelib chiqib `chaqiruv` ni uchta qiymat qabul qiladigan istalgan qism dasturni chaqiradigan makro tarzida tuzish mumkin. U birinchi qiymat sifatida qism dasturi nomini qabul qiladi, qolgan uchta qiymatlar esa qism dasturiga jo'natiladigan qiymatlar bo'ladi.

```
%macro    chaqiruv 1-4   eax, ebx, ecx
     push dword %4
     push dword %3
     push dword %2
     call %1
     add  esp , 12
%endmacro
```

Ushbu makrodan, masalan, diskriminant qism dasturini chaqirishda foydalanish mumkin.

```
chaqiruv  diskriminant, a, b ,c    ;; diskriminant, a,  b,  c
chaqiruv  diskriminant, esi, edi   ;; diskriminant, esi, edi, ecx
chaqiruv  diskriminant, edx        ;; diskriminant, edx, ebx, ecx
chaqiruv  diskriminant             ;; diskriminant, eax, ebx, ecx
```

9.3.2. %ROTATE makro direktivasi

Istalgan miqdorda qiymat jo'natish mumkin bo'lgan ko'p satrli makrolarga ma'lum bir chaqiriqda aynan qancha qiymat jo'natilganini aniqlash juda muhim. Chunki makro jo'natilgan barcha qiymatlar ustida ishlashi kerak. Masalan:

```
chop_et   `Salom, Bugun havo yaxshi`
chop_et   `%i, %i, %i`, eax, ebx, ecx
chop_et   `Javob=%i`, [natija]
```

Birinchi holatda `chop_et` makrosiga 1 ta qiymat jo'natilmoqda, ikkinchi holatda 4 ta va uchunchi holatda esa 2 ta.

Ko'p satrli makrolarda doim `%0` orqali ularga nechta qiymat jo'natilganligini aniqlash mumkin. Agar makro noaniq usul orqali aniqlangan bo'lsa, bu imkoniyat juda muhim hisoblanadi. Chunki `%0` marta makro takrorlanishni amalga oshirgan holda har bir qabul qilingan qiymat bilan ishlash mumkin. Masalan:

```
%makro    chop_et   1-*
     …
     %rep %0
     ;; har bir qabul qilingan qiymat ustida amallar
     %endrep
     …
%endmacro
```

Jo'natilgan qiymatlarga `%1, %2` va hokazolar orqali murojaat qilinishi ma'lum. Lekin har bir takrorlanishda aniq bir qiymat ko'rsatilishi kerak yoki biror usul orqali takrorlanishning har safarida keyingi qiymatni berish kerak. Bunda `%ROTATE` makro direktivasi yordam beradi.

`%ROTATE` makro direktivasi bitta sonli qiymat qabul qilib, agar son musbat bo'lsa, makro qabul qilgan qiymatlarni chap tomonga, manfiy bo'lsa, o'ng tomonga `%1, %2,…` larga nisbatan siljitadi. `%ROTATE` ga beriladigan son necha xona o'ngga yoki chapga siljitish kerakligini beradi. Masalan, `chop_et` quyidagicha ishlatildi deylik:

```
chop_et   `%i, %i, %i`, eax, ebx, ecx
```

unda `%1 = `%i, %i, %i`, %2` = EAX, `%3` = EBX, `%4` = ECX bo'ladi.

```
%rotate 1
```

dan so'ng esa `%1` = EAX, `%2` = EBX, `%3` = ECX, `%4 = `%i, %i, %i`` bo'ladi. Ko'rinib turibdiki, `%ROTATE` halqasimon siljitishni amalga oshiradi.

```
%rotate 2          ;; %1=ECX, %2 = `%i, %i, %i`, %3=EAX, %4=EBX
%rotate -1         ;; %1=EBX, %2=ECX, %3 = `%i, %i, %i`, %4=EAX
```

`%ROTATE` ning foydali tomoni shundaki, uni qo'llagan holda makro takrorlanishda faqat `%1` orqali barcha jo'natilgan qiymatlarga murojaat qilish mumkin.

Shu paytgacha ishlatib kelingan `chop_et` makrosida ham `%ROTATE` dan juda unumli foydalanilgan. Umuman olganda, `chop_et` ning asosiy vazifasi unga berilgan qiymatlarni stackka yuklab C dagi mezoniy qism dasturi bo'lmish `printf` ni chaqirib berishdan iborat.

```
%macro    chop_et   1-*
…
%rep %0-1           ;; Andozadan tashqari barcha qiymatlar uchun.
     %rotate -1          ;; Andozani chetlab o'tamiz.
     push dword %1
%endrep

push dword %%andoza   ;; %%andoza keyingi mavzuda tushuntiriladi.
call printf
add esp , %0 * 4      ;; stackni bo'shatamiz
…
%endmacro
```

`%ROTATE` yordamida yuqorida ko'rilgan `chaqiruv` makrosini yanada takomillashtirish mumkin. Endi makro faqat uchta qiymat qabul qiladigan qism dasturlarini emas, balki istalgan qism dasturni chaqiradigan qilinadi. Unga birinchi qiymat sifatida qism dasturi nomi beriladi.

```
%macro    chaqir 1-*
%assign   qiymatlar_soni %0-1   ;; Qism dastur nomi chiqariladi.
%xdefine  qism_dastur    %1
%rep qiymatlar_soni
     %rotate -1            ;; Qism dastur nomini chetlab o'tamiz.
     push dword %1
%endrep

call qism_dastur
add esp , qiymatlar_soni * 4       ;; Stackni bo'shatamiz.
%endmacro
```

Avval `%ASSIGN` orqali qism dasturga jo'natiladigan qiymatlar sonini belgilab olamiz. Keyin esa qism dastur nomini `%XDEFINE` orqali o'zimiz uchun aniqlab olamiz. Chunki siljitishlardan so'ng qism dastur nomi aynan qaysi raqamda bo'lishini bilish qiyin. Endi esa bemalol ushbu makrodan foydalanish mumkin.

```
chaqir    diskriminant , [a] , [b] , [c]
chaqir    ildiz , eax
```

Bu preprotsessor tomonidan quyidagiga almashtiriladi:

```
push dword [c]
push dword [b]
push dword [a]
call diskriminant
add  esp , 3 * 4

push dword 25
call ildiz
add  esp , 1 * 4
```

Misol tariqasida yana bir makroni ko'rib chiqamiz. Dasturlashda ko'pincha dastur boshida ishlatiladigan o'zgaruvchilar qiymatini nolga tenglashtirish kerak bo'ladi. Agar berilgan qiymatlarning barchasini nollaydigan makro bo'lsa juda o'ng'ay bo'ladi.

```
%macro    nolla 1-*
%rep %0
     mov  %1 , 0
     %rotate 1
     %endrep
%endmacro
…
nolla eax , dword[son] , edx , byte[harf] , ecx
```

Stack bilan bog'liq bo'lgan ko'pgina foydali makrolar o'ylab topish mumkin. Masalan, stackka birdaniga bir qancha qiymatlarni yuklaydigan yoki teskarisiga, ya'ni stackdan qiymatlarni chiqarib oladigan makrolar tuzish mumkin. Ushbu makrolarni tuzishni o'quvchiga qoldiramiz. Yordam sifatida shuni aytish mumkinki, makrolarni bemalol `PUSH` va `POP` deb nomlash mumkin. Chunki bitta qiymat bilan ishlatilganda assemblerdagi `PUSH` va `POP` buyruqlari, ko'p qiymatlar bilan ishlatilganda esa sizning makrolaringiz ishga tushadi.

Ushbu bobda ko'rib chiqilayotgan makrolar dasturlash jarayonida juda kerak bo'lishi mumkin. Shuning uchun ularning keraklilarini boshlang'ich fayl yaratib u yerga saqlab qo'ygan ma'qul. Keyin esa o'sha faylni `%INCLUDE` orqali qo'shib olib istalgan dasturingizda foydalanishingiz mumkin. Masalan:

```
%include  "mening_makrolarim.inc"
```

9.3.3. Makrolarda mahalliy nishonlar

Ba'zida ko'p satrli makrolarda ham tarmoqlash buyruqlaridan foydalanishga to'g'ri keladi. O'tishlar amalga oshiriladigan joyga esa nishonlar qo'yiladi. Masalan, `chop_et` makrosi andoza uchun xotiradan joy ajratib, `printf` ga shu joy manzilini jo'natadi[1]. Lekin `chop_et` dasturning `.text` bo'limida ishlatilgani sababli andoza uchun ham joy o'sha bo'limdan ajratilishiga to'g'ri keladi.

```
%macro    chop_et 1-*
…
jmp  chetlash
andoza    db   %1 , 0
chetlash:
…
push dword andoza
call printf
…
%endmacro
…
chop_et   `Salom`
```

Bu quyidagiga almashtiriladi:

```
     jmp  chetlash
     andoza    db   `Salom`
chetlash:
     …
     push dword andoza
     call printf
```

[1] Nega bunday qilinishi «C va Assembler dasturlarini bog'lash» mavzusida tushuntirilgan.

Eslatma: Ba'zida ko'p satrli makrolarda o'zgarmas qiymatlar uchun xotiradan joy ajratishga to'g'ri keladi. Shuning uchun ham NASM dasturning `.text` bo'limida `dX` direktivalar oilasidan foydalanishga ruxsat beradi. Ammo protsessor buyruqlarni bajarayotganida bunday o'zgarmas qiymatlarga duch kelmasligi kerak. Shuning uchun mazkur dastur satri shartsiz o'tish orqali chetlab o'tiladi.

Ammo bu holda yangi muammo paydo bo'ladi. Gap shundaki, `chop_et` makrosi dasturda ko'p marta ishlatilishi mumkin. Preprotsessor esa shunchaki almashtirishni amalga oshirib boraveradi. Qarabsizki, bitta nishon bir necha marta dasturda qo'yib chiqiladi. Masalan:

```
chop_et   `Salom`
chop_et   `Xayr`
```

Bu quyidagiga almashtiriladi:

```
     jmp  chetlash
     andoza    db   `Salom`
chetlash:
     …
     push dword andoza
     call printf

     jmp  chetlash
     andoza    db   `Xayr`
chetlash:
     …
     push dword andoza
     call printf
```

Ko'rinib turibdiki, `chetlash` va `andoza` nishonlari ikki martadan e'lon qilindi va shubhasiz NASM buni xato deb topadi. Bunday ikkilanishlarni oldini olish maqsadida preprotsessor makrolarga mo'ljallangan mahalliy nishonlarni taqdim etadi. Bunday nishonlarning qo'llanishi shu paytgacha ishlatgan nishonlarimizdan farq qilmaydi, faqat ular ikkita foiz belgisi (`%%`) bilan boshlanadi.

```
%macro    chop_et 1-*
     …
     jmp  %%chetlash
%%andoza  db   %1 , 0
%%chetlash:
     …
     push dword %%andoza
     call printf
     …
%endmacro
```

Bu kabi makro nishonlar preprotsessor tomonidan maxsus qoidaga binoan almashtiriladi. Masalan, `chop_et` ning birinchi ishlatilishida `%%chetlash` va `%%andoza` nishonlari mos ravishda `..@6.chetlash` va `..@6.andoza` ga almashtiriladi, keyingi ishlatilishida esa 6 soni oshirib borilaveradi. Bu esa har safar almashtirish amalga oshirilganda nishon o'z-o'zi bilan chalkashib ketmasligini ta'minlaydi. Masalan, oldingi holat quyidagiga almashtiriladi:

```
        jmp  ..@.6chetlash
        ..@.6andoza    db   `Salom`
..@6.chetlash:
        …
        push dword ..@6.andoza
        call printf

        jmp  ..@7.chetlash
        ..@7.andoza    db   `Xayr`
..@7.chetlash:
        …
        push dword ..@7.andoza
        call printf
```

Nishon nomini `..@` bilan boshlashdan maqsad shuki, ular boshqa foydalanuvchi nishonlari bilan bir xil bo'lib qolmasligidir.

9.4. Dasturni shartli yig'ish

Dasturni shartli yig'ish deganda uning ma'lum bir qismini ma'lum bir shart bajarilgandagina yig'ish tushuniladi. Masalan, tuzilayotgan dasturning ham Windowsda, ham Linuxda ishlashi talab qilinsin. Ammo dasturning ma'lum bir qatori Windows uchun boshqacha, Linux uchun boshqacha ko'rinishga ega bo'lishi mumkin. Shunda maxsus makro direktivalardan foydalanib, agar Windows bo'lsa dastur qatori quyidagicha ko'rinishga ega bo'lsin, Linux bo'lsa boshqacha ko'rinishga ega bo'lsin deb ko'rsatish, shundan so'ng dasturga hech qanday o'zgartirish kiritmasdan, uni Windows uchun alohida, Linux uchun ham alohida yig'ish mumkin.

Preprotsessor bunday imkoniyatni yuqori daraja tillaridagi kabi `%IF`, `%ELIF` va `%ELSE` direktivalari orqali taqdim etadi.

Qoidasi:

```
%if shart_1
        ;; shart_1 haqiqat bo'lganda bajarilishi kerak
        ;; bo'lgan dastur qismi.
%elif shart_2
        ;; shart_1 yolg'on va shart_2 haqiqat bo'lganda
        ;; bajarilishi kerak bo'lgan dastur qismi.
%else
        ;; birorta ham shart haqiqat bo'lib chiqmaganda
        ;; bajariladigan dastur bo'lagi.
%endif
```

Bu yerda `%ELIF` va `%ELSE` bo'laklarining bo'lishi ixtiyoriy. `%ELIF` bo'lagidan istalgancha ko'p qo'yish mumkin.

Haqiqat yoki yolg'onligi tekshiriladigan shart o'zgarmas ifoda bo'lishi kerak. Ifoda makrolardan, o'zgarmas qiymatlardan va taqqoslash belgilaridan iborat bo'lishi mumkin. Taqqoslash belgilari `==`, `<`, `>`, `<=`, `>=` va `!=` mos ravishda tenglik, kichik, katta, kichik yoki teng, katta yoki teng va teng emaslardan iborat. O'zgarmas ifodada barcha arifmetik va mantiqiy amallar qatnashishi mumkin. O'zgarmas ifodalar tuzilishi haqida «O'zgarmaslar, ifodalar va direktivalar» mavzusida to'liq ma'lumot berilgan.

Ma'lumki, `&`, `^` va `|` kabi mantiqiy amallarni bitma-bit bajaradigan belgilar mavjud. Ulardan tashqari preprotsessor taqdim etadigan `&&`, `^^` va `||` kabi amallar mavjud bo'lib, ular

qiymatlarga bitma-bit emas, balki butun tarzda munosabatda bo'ladi. Masalan, qiymat nolga teng bo'lsa, u yolg'on deb, boshqa barcha hollarda haqiqat deb olinadi. Misol:

```
%define   A    11110000b      ;; 240
%define   B    00001111b      ;; 15
%if (A && B)
     echo_et   `Bu haqiqat`
%endif

%if (A & B)
echo_et   `Bu yolg'on, shuning uchun hech qachon chop etilmaydi`
%endif
```

Berilgan ifoda preprotsessor tomonidan hisoblab chiqiladi va natija noldan farqli bo'lsa, shart haqiqat deb olinadi, aks holda shart bajarilmagan hisoblanadi, ya'ni yolg'on.

Dasturni shartli yig'ish makro direktivalari vazifa jihatdan assemblerdagi taqqoslash va tarmoqlash buyruqlariga o'xshaydi. Ammo makro direktivalarining assembler buyruqlariga hech qanday aloqasi yo'q va ular dastur qanday yig'ilishini nazorat qiladi xolos.

Misol tariqasida har xil operatsion tizimga mo'ljallangan dastur tuzilishini ko'rib chiqamiz.

```
%define   WINDOWS 1
%define   LINUX 2
…
%if  TIZIM == LINUX
     %include "/usr/include/asm/fcntl.inc"
%elif TIZIM == WINDOWS
     %include "C:\Windows\system32\winnasm.inc"
%else
     %error xato, tizim aniqlanmadi.
%endif
```

Ushbu dastur Windows uchun alohida va Linux uchun alohida yig'iladi. `TIZIM` makrosini tekshirishlardan oldin yoki dasturni yig'ish paytida NASM ning `-d` kaliti orqali aniqlashingiz mumkin. Masalan, Windows uchun TIZIM quyidagicha aniqlanadi:

```
%define   TIZIM     1
```

Yoki buyruqlar qatorida dasturni yig'ish chog'ida:

```
> nasm -f win32 -dTIZIM=1 dastur.asm
```

Preprotsessor yig'ish boshlanishidan oldin shartlarni tekshira boshlaydi va birinchi haqiqat bo'lib chiqqan shart uchun berilgan dastur bo'lagini yig'ish uchun qoldiradi. Qolgan bo'laklarni olib tashlaydi. Bizning holatimizda yuqoridagi misol quyidagiga almashtiriladi:

```
%include  "C:\Windows\system32\winnasm.inc"
```

Agar dastur yanada ko'proq tizimlarga mo'ljallanmoqchi bo'linsa, `%ELIF` bo'laklarini qo'shish kifoya. Ammo, oldindan `TIZIM` makrosi aniqlanishi shart! Aks holda `%ELSE` bo'lagi ishga tushadi. Biz u yerda `%ERROR` makro direktivasidan foydalandik. Bu makro direktiva dastur yig'ilayotgan paytda xato yuz bergani haqida xabar beradi va dasturning yig'ilishi to'xtatiladi. U o'zidan keyin keladigan va satr oxirigacha bo'lgan harflarni chop etadi. U yerda xato yuz berish

sabablarini ko'rsatish mumkin. %ERROR ni chop etishni amalga oshiradigan qism dasturlar bilan chalkashtirmaslik kerak! %ERROR chop etishni dastur yig'ish chog'ida amalga oshiradi.

%IF va %ELIF larga teskari bo'lgan %IFN va %ELIFN makro direktivalari ham mavjud bo'lib, ular shartning haqiqat emasligini tekshiradi, ya'ni shart yolg'on bo'lganda berilgan bo'lak bajariladi.

Dasturni shartli yig'ish makro direktivalarning juda ko'p turlari bo'lib, keyingi mavzularda eng keraklilari ko'rib chiqiladi.

9.4.1. Makroni aniqlanganligini tekshirish

Ba'zida ma'lum bir makro aniqlangan yoki aniqlanmaganligini tekshirishga to'g'ri kelib qoladi. %IFDEF va %IFMACRO makro direktivalari mos ravishda bir va ko'p satrli makrolar uchun shunday tekshiruvni amalga oshiradi. Ushbu makro direktivalari ularga berilgan makroni oldin aniqlangan yoki aniqlanmaganligini tekshiradi. Masalan, dastur tuzayotganda, hisob kitob to'g'ri kelayotganini tekshirish uchun registrlarning qiymatlarini dasturining har yerida chop etib borish kerak bo'ldi deylik:

```
…
chop_et   `EAX=%i, EBX=%i, ECX=%i, EDX=%i`, eax, ebx, ecx, edx
…
chop_et   `EAX=%i, EBX=%i, ECX=%i, EDX=%i`, eax, ebx, ecx, edx
…
```

Dastur to'g'ri ishlab maromiga yetgach, bu chop etishlar kerak bo'lmay qoladi. Demak, barcha chop_et larni o'chirishga to'g'ri keladi. Agar dastur katta bo'lsa, ish yanada ko'payadi. Buning yechimi sifatida %IFDEF dan foydalanish mumkin.

```
…
%ifdef    TEKSHIR
chop_et   `EAX=%i, EBX=%i, ECX=%i, EDX=%i`, eax, ebx, ecx, edx
%endif
…
%ifdef    TEKSHIR
chop_et   `EAX=%i, EBX=%i, ECX=%i, EDX=%i`, eax, ebx, ecx, edx
%endif
…
```

Dastur hali oxiriga yetmagancha TEKSHIR makrosi dastur boshida aniqlanadi. Dastur tayyor bo'lgach, TEKSHIR aniqlangan satr olib tashlanadi.

%IFDEF bilan birga %ELIFDEF va %ELSE larni ham ishlatsa bo'ladi. Ushbu makro direktivalar oilasi uchun ularga teskari bo'lgan %IFNDEF va %ELIFNDEF makro direktivalari mavjud.

%IFMACRO ham xuddi %IFDEF kabi, faqat u ko'p satrli makrolarning aniqlanganligini tekshirishga mo'ljallangan. Masalan, KIRILLCHA degan ko'p satrli makroni aniqlamoqchisiz. Ammo bunday makro dasturga qo'shib olingan birorta bo'shlang'ich faylda ham aniqlangan bo'lishi mumkin. Ikkita bir xil makro aniqlanishdan kelishmovchilik chiqishining oldini quyidagicha olish mumkin:

```
%ifmacro  KIRILLCHA 2
     %error 2 ta qiymat qabul qiladigan KIRILLCHA \
     makrosi allaqachon aniqlangan ekan!
```

```
%else
     %macro    KIRILLCHA 2
     ;; KIRILLCHA makrosining kodi
     %endmacro
%endif
```

`%IFMACRO` ga birinchi qiymat sifatida makroning nomi, so'ng u qabul qiladigan qiymatlar soni beriladi. Agar, masalan, `KIRILLCHA` makrosi dasturining biror yerda faqat bitta qiymat qabul qiladigan bo'lib aniqlangan bo'lsa, `%IFMACRO` uni inobatga olmaydi. Faqat aniq mosliklar haqiqat hisoblanadi.

`%IFMACRO` bilan birga `%ELIFMACRO` va `%ELSE` makro direktivalari ham ishlatilishi mumkin. Ushbu makro direktivalar oilasiga teskari bo'lgan `%IFNMACRO` va `%ELIFNMACRO` direktivalari ham mavjud.

9.4.2. Matn tengligini tekshirish

Asosan ko'p satrli makrolar bilan ishlaganda ularga yuborilgan qiymat aynan qanday nomga yoki agar qiymat bir satrli makro bo'lsa, qanday qiymatga ega ekanligini aniqlashga to'g'ri keladi. Misol tariqasida `nolla` makrosini eslasak. Yodingizda bo'lsa, ushbu makro orqali o'zgaruvchilar qiymatini nollagan edik. Ammo ushbu makroga ESP yoki EBP registrlari ham berilishi mumkin. «Qismli dasturlash» bobidan ma'lumki, bu registrlarning qiymatini o'zgartirmagan ma'qul. Kutulmaganda `nolla` ga ESP yoki EBP berilgan vaziyatda, ularni payqash kerak bo'ladi. Buning eng maqbul yo'li jo'natilgan qiymatlarni ESP yoki EBP emasligini tekshirishdan iboratdir.

`%IFIDN` marko direktivasi o'ziga berilgan ikkita qiymatni matn sifatida aynanligini tekshiradi. `%IFIDN` bilan birga `%ELIFIDN` va `%ELSE` makro direktivalari ham ishlatilishi mumkin. Masalan:

```
%macro    nolla 1-*

%rep %0
     %ifidn    %1 , esp
          %error    ESP ning qiymati o'zgartirilmaydi!
     %elifidn  %1 , ebp
          %error    EBP ning qiymati o'zgartirilmaydi!
     %else
          mov  %1 , 0
     %rotate 1
%endrep

%endmacro
```

Har holda endi ushbu makro EBP yoki ESP ning qiymatiga tegmaydigan bo'ldi.

`%IFIDN` juda ham foydali qurol bo'lib, u yordamida `nolla` ni yanada mukammallashtirilsa bo'ladi. O'zgaruvchi qiymatini `XOR` buyrug'i bilan ham nollash mumkin. Ammo buning uchun o'zgaruvchi registr bo'lishi shart. Demak, tekshiruvga har bir registr uchun `%ELIFIDN` bo'lagi qo'shiladi.

```
…
%elifidn  %1, eax
     xor  eax, eax
%elifidn  %1, ebx
```

```
        xor  ebx, ebx
%elifidn  %1, ecx
        xor  ecx, ecx
;; va hokazo
%else
        mov  %1 , 0
```

Nega `MOV` ning o'rnida ishni qiyinlashtirib `XOR` ni registrlar uchun qo'llash mukammallashtirish deb hisoblanganini o'zingiz topishga harakat qilib ko'ring.

`%IFIDN` tomonidan ikkita matn tekshirilayotganda matnlardagi boshliq ( ) belgisi hisobga olinmaydi. Yana shuni ta'kidlash lozimki, ushbu makro direktiva berilgan o'zgaruvchilarning qiymatini emas, balki ularning nomlanishidagi belgilarni tekshiradi. Aks holda `%IFIDN` ning o'rnida oddiy `%IF` marko direktivasidan yoki `CMP` buyrug'idan foydalansa bo'ladi.

`%IFIDNI` makro direktivasi ham mavjud bo'lib, u `%IFIDN` dan farqli o'laroq matndagi harflarning katta yoki kichikligini inobatga olmaydi.

Ushbu makro direktivalar oilasiga teskari bo'lgan `%IFNIDN` va `%ELIFNIDN` makro direktivalari mavjud.

9.4.3. Qiymat turlarini tekshirish

`%IFID` makro direktivasiga berilgan qiymat faqat o'zgaruvchi bo'lsagina, shart haqiqat hisoblanadi.

`%IFNUM` makro direktivasiga berilgan qiymat faqat sonli o'zgarmas bo'lsagina, shart haqiqat hisoblanadi.

`%IFSTR` makro direktivasiga berilgan qiymat faqat qatorli o'zgarmas bo'lsagina, shart haqiqat hisoblanadi.

Ushbu makro direktivalar oilasi uchun quyidagi qo'shimcha shartli yig'ish makro direktivalar mavjud: `%ELIFID`, `%IFNID`, `%ELIFNID`, `%ELIFNUM`, `%IFNNUM`, `%ELIFNNUM`, `%ELIFSTR`, `%IFNSTR`, `%ELIFNSTR`.

Bu kabi makro direktivalar asosan ko'p satrli makrolarga jo'natilgan qiymatlarning turini tekshirishda ishlatiladi. Masalan, `chop_et` makrosiga birinchi bo'lib qatorli o'zgarmas jo'natilishi kerak. Buni esa `%IFSTR` orqali tekshirish mumkin.

```
%macro    chop_et 1-*
%ifnstr %1
        %error    chop_et ning birinchi qiymati \
                  qatorli o'zgarmas bo'lishi kerak!
        jmp %%tamom
%endif
…
%%tamom:
%endmacro
```

Bu yerda birinchi jo'natilgan qiymat, ya'ni `%1`, qatorli o'zgarmas yoki yo'qligi tekshiriladi. Agar u rostdan ham qatorli o'zgarmas bo'lsa, u uchun xotiradan joy ajratiladi. Aks holda xato yuz bergani haqida xabar beriladi.

Qiymat turlarini tekshirish makro direktivalari ham boshqa shartli yig'ish makro vositalari kabi ancha foydali hisoblanadi. Misol uchun ko'p satrli makrolarda quyidagicha buyruq uchrashi mumkin.

```
mov  %2 , esi
```

Bu yerda `%2` albatta o'zgaruvchi bo'lishi shart, aks holda agar u o'zgarmas yoki boshqa bir narsa bo'lsa, NASM xato yuz bergani haqida xabar berishi turgan gap. Shuning uchun `%2` ni oldindan nima ekanligini tekshirgan ma'qul.

```
%ifid     %2
     mov  %2 , esi
%else
     %error    %2  o'zgaruvchi bo'lishi shart!
%endif
```

9.5. Mezoniy makrolar

Preprotsessor almashtirishlarni amalga oshirishdan oldin birinchi bo'lib tashkiliy masalalarga doir qiymatlarni bir satrli makrolar sifatida aniqlab qo'yadi. Bunday makrolar mezoniy deb yuritiladi. Ulardan dasturda bemalol foydalanish mumkin.

Quyida shunday makrolarning eng keraklilari keltirilgan.

- Obyekt fayl andozasiga tegishli bo'lgan makrolar:

__BITS__	Dastur necha bitli usulda ishlayotgani sonli o'zgarmas sifatida beradi. Masalan, protsessorga qarab usullar 16, 32 va 64 bitli bo'lishi mumkin.
__OUTPUT_FORMAT__	Obyekt fayl andozasini beradi. Uning qiymati NASM ga `-f` kaliti orqali berilgan obyekt fayl andozasiga teng bo'ladi. Masalan, `__OUTPUT_FORMAT__` ni preprotsessor quyidagilarga alamashtirilishi mumkin: `elf`, `win32`, `macho` va hokazo[1].

Ushbu turdagi makrolarni qo'llashga doir misollar keltirsak. Ba'zida dastur necha bitli usulda ishlayotgani bilish muhim bo'ladi. Masalan, 16 bitli usulda EAX registri bo'lmaydi, ya'ni 32 bitli kengaytirilgan registrlar mavjud emas. Agar dastur 16 va 32 bitli usullarda o'zgartirishlarsiz ishlashi kerak bo'lsa, quyidagi makrolar aniqlanadi:

```
%if  __BITS__ == 16
     %define   AX_reg ax
     %define   BX_reg bx
     …
     %define   D_BUTUN dw
     %define   D_YARIM db
     %define   BUTUN word
     %define   YARIM byte
%elif  __BITS__ == 32
     %define   AX_reg eax
     %define   BX_reg ebx
     …
     %define   BUTUN dd
     %define   YARIM dw
     %define   BUTUN dword
     %define   YARIM word
```

[1] «Obyekt fayllarni ulash» mavzusiga qarang.

```
%else
     %error NECHA BITLI USULDA ISHLAYAPSIZ?
%endif
```

Shundan so'ng registrlarga `AX_reg`, `BX_reg` va hokazolar orqali murojaat qilasiz. Ular bilan ishlaydigan o'zgaruvchilarni esa `D_BUTUN, D_YARIM` lar bilan e'lon qilasiz. Masalan:

```
section .data
x    D_BUTUN   100
y    D_YARIM   0
…
mov  AX_reg , BX_reg
add  AX_reg , [x]
mov  BUTUN[x] , 200
sub  YARIM[y] , 300
```

Obyekt fayl andozasiga kelsak, uning ham kerakli tomonlari bor. «Qismli dasturlash» bobidan sizga ma'lumki, C va Assemblerda tuzilgan dasturlar o'zaro bog'lanishi mumkin. Ammo `elf` obyekt fayl andozasidan tashqari barcha andozalarda C dasturida ishlatiladigan Assemblerdagi nishon nomi pastki chiziq (_) bilan boshlanishi kerak. Bu rangba-ranglikni umumiy ko'rinishga olib kelish uchun `tizim_global` va `tizim_extern` makrolari keltirilgan edi. Ushbu makrolar qaysi obyekt fayl andozasi ishlatilayotganini aynan `__OUTPUT_FORMAT__` makrosi orqali aniqlab nishon nomi oldiga pastki chiziq qo'yish yoki qo'ymaslikni hal qiladi.

12-namuna: Obyekt fayl andozasi

```
(1)    %define elf32     1
(2)    %define elf       elf32
(3)    %define coff      2
(4)    %define win32     3
(5)    %define win64     4
(6)    %define bin       5
(7)    %define obj       6
(8)
(9)    %if __OUTPUT_FORMAT__ == elf
(10)   ;; Linux uchun
(11)
(12)   %define tizim_global global
(13)   %define tizim_extern extern
(14)
(15)   %else
(16)   ;; Qolgan barcha andozalar uchun
(17)
(18)   %macro tizim_global 1-*
(19)      %rep %0
(20)           global _%1
(21)           %define %1 _%1
(22)           %rotate 1
(23)      %endrep
(24)   %endmacro
(25)
(26)   %macro tizim_extern 1-*
(27)      %rep %0
(28)           extern _%1
(29)           %define %1 _%1
```

```
(30)            %rotate 1
(31)        %endrep
(32)    %endmacro
(33)
(34)    %endif
```

Shundan so'ng qanday obyekt fayl andozasidan foydalanayotganimiz haqida xavotirlanmasdan `extern` va `global` assembler direktivalari o'rnida `tizim_extern` va `tizim_global` makro laridan foydalanishingiz mumkin.

```
section .text
tizim_extern printf, scanf
tizim_global main
main:
        …
        call printf
        …
        call scanf
```

`__OUTPUT_FORMAT__` makrosini yana foydaliroq maqsadlarda ishlatish mumkin. Masalan, obyekt fayl andozasiga ko'ra assembler direktivalarining berilish usullari ham bir o'z o'zgaradi. `__OUTPUT_FORMAT__` makrosi orqali esa iloji boricha dasturingizni ko'proq andozalarga to'g'ri keladigan qilib tuzishingiz mumkin. Misol tariqasida `obj` obyekt fayl andozasini oladigan bo'lsak, unda dastur bo'limlarining nomi nuqtasiz boshlanishi kerak: `text`, `bss` va `data`.

- Vaqt va sana makrolari:

`__DATE__`	Dastur yig'ilayotgan vaqtdagi sanani qatorli o'zgarmas sifatida *"YYYY-OO-SS"* andozasida beradi.
`__TIME__`	Dastur yig'ilayotgan vaqtdagi vaqtni qatorli o'zgarmas sifatida *"SS:DD:SS"* andozasida beradi.
`__DATE_NUM__`	`__DATE__` kabi, ammo sana sonli o'zgarmas sifatida *YYYYOOSS* andozasida beriladi.
`__TIME_NUM__`	`__TIME__` kabi, ammo vaqt sonli o'zgarmas sifatida *SSDDSS* andozasida beriladi.
`__POSIX_TIME__`	Birinchi yanvar 1970-yildan toki dastur yig'ilgan chog'gacha bo'lgan soniyalar sonini sonli oz'garmas sifatida beradi.

- Dastur yozilayotgan fayl bilan bog'liq bo'lgan makrolar:

`__FILE__`	O'zi yozilgan fayl nomini qatorli o'zgarmas sifatida beradi.
`__LINE__`	O'zi yozilgan satrning fayldagi tartib raqamini beradi.

Ushbu makrolardan dasturdagi xatolarni topishda foydalanish mumkin. Masalan, dasturning ma'lum bir qatorining to'g'ri ishlayotgani shubhali tuyulsa, qator raqamini __LINE__ yordamida chop etishingiz mumkin.

```
chop_et   `Qator:%i, dastur shu yergacha ishlayapti`, __LINE__
```

9.6. Amaliy dastur: Lotindan kirillga

Amaliy dastur sifatida qatorli o'zgaruvchini berilgan boshlang'ich qiymati bilan e'lon qiladigan makroni ko'rib chiqamiz. Makroning asosiy vazifasi qatorli o'zgarmas tarkibidagi lotincha harflarni kirillchadagi mos harflarga o'girishdan iborat bo'ladi. Bunday makro har tomonlama kerakli bo'lib, kirillcha yozishga qiynalsangiz yoki tizimda tugmachalar taxtasi orqali kirillcha harflarni kiritish imkoniyati bo'lmasa juda asqatadi.

Har xil alifbodagi harflar bilan ishlashdan oldin tizimdagi belgilarni raqamlash turlari bilan yaxshi tanish bo'lish kerak. Turli xil ASCII, UNICODE, UTF-8 kabi ko'pgina raqamlash turlari mavjud. Ammo barcha raqamlash turlari ham kirillcha harflarni o'z ichiga olmaydi. Lotindan kirillga o'giradigan dasturda esa kirill alifbosiga mo'ljallangan raqamlash turlaridan foydalanish kerak.

Kirill alifbosini o'z ichiga oladigan bir qancha raqamlash turlari bo'lib, shulardan eng ko'p qo'llaniladigani KOE8-R va Windows-1251 lardir. 12-jadvalda ushbu raqamlash turlarida lotincha va kirillcha harflar qanday ifodalanishi ko'rsatilgan.

12-jadval

	Lotin										**Kirill**									
Harflar	A	B	...	Y	Z	a	b	...	y	z	А	Б	...	Ы	З	а	б	...	ы	з
KOI8-R	65	66	...	89	90	97	98	...	121	122	225	226	...	249	250	193	194	...	217	218
Windows-1251	65	66	...	89	90	97	98	...	121	122	192	193	...	219	199	224	225	...	251	231
ASCII	65	66	...	89	90	97	98	...	121	122	ASCII da kirill harflari yo'q									

Dasturlashda belgi son sifatida qabul qilinishini hisobga olsak, lotin harfi uchun uning mos kirillchasi oradagi farqni qo'shish orqali topiladi. Ammo buning uchun kirill harflari raqamlash turida aynan lotin alifbosidagi tartibda qo'yib chiqilgan bo'lishi kerak. Bunday imkoniyat, masalan, Windows-1251 da mavjud emas, chunki unda kirill harflari haqiqiy kirill alifbosi bo'yicha joylashtirib chiqilgan. KOE8-R da esa biz istayotgan holat mujassam, ya'ni unda asosan barcha mos lotincha va kirillcha harflar o'rtasidagi farq bir xil. 12-jadvaldan ko'rinib turibdiki, KOE8-R da ushbu farq bosh harflar uchun 160 ga va kichik harflar uchun esa 96 ga teng.

KOE8-R raqamlash turida kirill harflari iloji boricha lotinchaga mos keladigan ketma-ketlikda joylashtirishga harakat qilingan. Albatta, alifbolar o'rtasida mutlaq moslikka erishib bo'lmaydi, chunki faqat alifbogagina xos bo'lgan harflarning borligi va alifbodagi harflar sonining har xilligi bunga yo'l qo'ymaydi. Shuning uchun KOE8-R raqamlash turidan foydalanilgan holda har qalay matn asliga yaqin bo'lgan o'girishni amalga oshirish mumkin.

Quyida keltiriladigan lotinchadan kirillchaga o'giradigan dasturda KOE8-R raqamlash turi qo'llaniladi. Ammo siz kirillchani o'z ichiga oladigan boshqa bir raqamlash turidan ham foydalanishingiz mumkin. U holda farqni ham o'zingiz aniqlashingizga to'g'ri keladi. Buning uchun sizga raqamlash turining jadvali kerak bo'ladi. Internetdan istalgan raqamlash jadvalini axtarib topish mumkin.

Istalgan raqamlash turining jadvalini qo'lga kiritishning dasturchilarga xos bo'lgan yo'li ham mavjud. Buning uchun 0 dan 255 gacha bo'lgan sonlarni belgi sifatida chop etuvchi dasturni tuzib, uni ishga tushirish kifoya. Demak, asosiy o'giruvchi makroni ko'rib chiqishdan oldin bir

baytga sig'adigan barcha sonlarni tartib bilan chop etib ko'ramiz. Bunday dasturni makro vositalardan unumli foydalangan holda tuzamiz.

13-namuna: raqamlash turini chop etish

```
(1)   %include "nasm-io.inc"
(2)
(3)   section  .data
(4)   belgilar:
(5)       %assign   harf 1    ;; 0 emas, 1 dan boshlaymiz
(6)       %rep 255
(7)           %defstr   raqam     harf
(8)           db   `\t`, raqam, `) `, harf, `\n`
(9)           %assign   harf harf+1
(10)      %endrep
(11)      db   0              ;; qator so'ngi
(12)
(13)  section  .text
(14)  tizim_global  main
(15)
(16)  main:
(17)      chop_et   `%s`, belgilar
(18)  ret
```

Bu yerda `raqam` makrosidan belgilarni raqamlangan holda chop etishda foydalaniladi. `%REP` esa 255 marta har bir belgi uchun `DB` direktivasini qo'yib chiqadi. `%REP` orqali ajratiladigan xotira bo'lagining manzili esa `belgilar` nishonidan boshlanadi. Proprotsessordan so'ng `.data` bo'limi quyidagicha ko'rinish oladi:

```
belgilar:
     db `\t`, '1', `) `, 1, `\n`
     db `\t`, '2', `) `, 2, `\n`
     …
     db `\t`, '65', `) `, 65, `\n`      ;; 'A'
     db `\t`, '66', `) `, 66, `\n`      ;; 'B', ya'ni 'A'+1
     db `\t`, '67', `) `, 67, `\n`      ;; 'C', ya'ni 'B'+1
     …
```

Bu yerda `%DEFSTR` ning vazifasi sonli o'zgarmasni qatorli o'zgarmasga o'tkazishdan iborat, masalan, 201 ni "201" ga.

Dasturni ishga tushirganda quyidagiga o'xshagan narsa buyruqlar qatorida paydo bo'ladi.

```
1) #
2) #
3) #
…
65) A
66) B
67) C
…
```

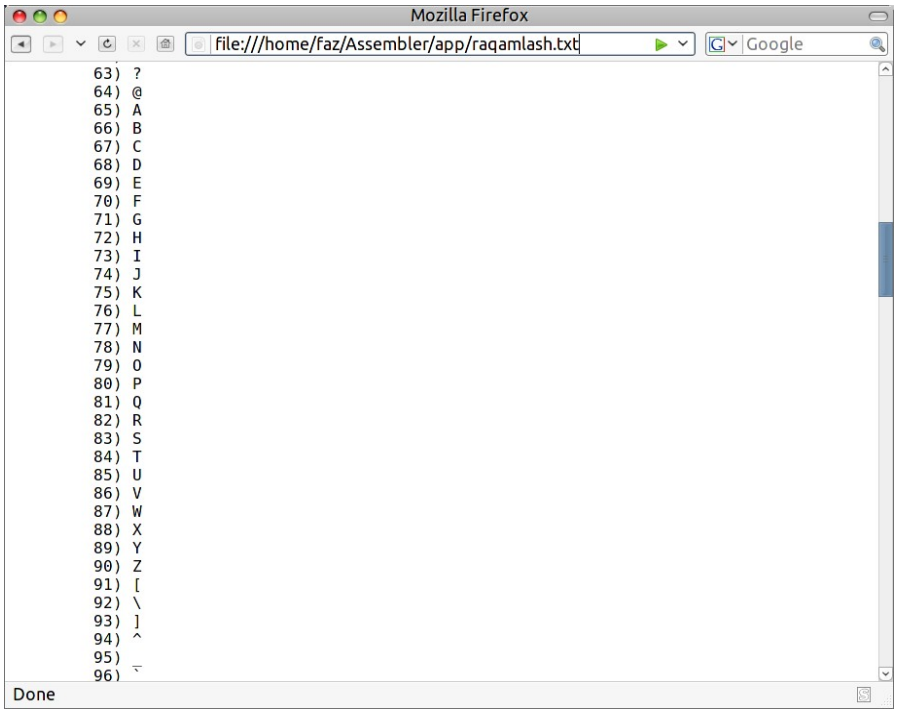
Agar buyruqlar qatori ishlatayotgan raqamlash turida kirill harflari bo'lsa, unda kirill harflari ham ko'rinadi. Windowsda asosan Windows-1251 usuli ishlatiladi, Linuxda esa UTF-8. Dastur natijasini istalgan raqamlash turida ko'rish uchun belgilarni buyruqlar qatorida emas, balki biror

raqamlash turlari bo'yicha mutaxassis dastur orqali kuzatgan ma'qul, masalan, web browserda. Buning uchun dastur natijasini buyruqlar qatoriga emas, balki biror faylga yo'naltiramiz va faylni web browserda ochamiz.

Natijani buyruqlar qatorida faylga yo'naltirish katta (>) belgisi orqali bajariladi.

```
$ dastur > fayl_nomi.txt
```

Web browserda esa istalgan raqamlash usulini tanlashingiz mumkin. Masalan, Mozilla Firefox web browserida View ro'yxatidan Character Encoding bandini tanlash orqali raqamlash usullari o'zgartiriladi (rasmga qarang).



40-rasm.

Shunday qilib raqamlash turlari tushunarli bo'lsa, endi rejalashtirilgan dasturingizni tuzishga kirishsak ham bo'ladi. Dasturlash jarayonida barcha makro aniqlashlarni bitta boshlang'ich faylga joylashtiramiz. Asosiy dasturda esa bu fayl `%INCLUDE` orqali qo'shib olinadi.

Lotinchadan kirillchaga o'giradigan makro ikkita qiymat qabul qiladi. Birinchi qiymat nishon nomi, ikkinchi qiymat esa qatorli o'zgarmas bo'ladi. Makroning vazifasi berilgan qatorli o'zgarmasdagi lotin harflarni topib, ularni kirillchaga almashtirish va natijaviy qatorli o'zgarmasni berilgan nishon uchun boshlang'ich qiymat sifatida e'lon qilish.

14-namuna: lotindan_kirillga.inc

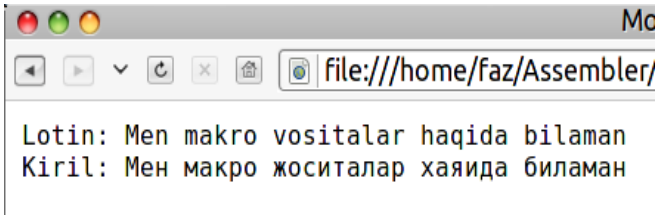
```
(1)    %macro lotindan_kirillga 2
(2)
(3)    %ifnid %1
(4)       error lotindan_kirillga beriladigan birinchi qiymat\
(5)             nishon nomi bo'lishi kerak.
(6)    %elifnstr %2
(7)       error lotindan_kirillga beriladigan ikkinchi qiymat qatorli\
(8)             o'zgarmas bo'lishi kerak.
(9)    %endif
(10)
(11)   %strlen uzunlik   %2
```

```
(12)   %assign joriy_belgi_tr 1
(13)   %define bosh_harf_farqi     160
(14)   %define kichik_harf_farqi   96
(15)
(16)   %1:
(17)   %rep    uzunlik
(18)      %substr   harf %2 joriy_belgi_tr
(19)
(20)      %if (harf >= 65) && (harf <= 90)
(21)           db   harf + bosh_harf_farqi
(22)      %elif (harf >= 97) && (harf <= 122)
(23)           db   harf + kichik_harf_farqi
(24)      %else
(25)           db   harf
(26)      %endif
(27)
(28)      %assign   joriy_belgi_tr joriy_belgi_tr+1
(29)   %endrep
(30)      db   0         ;; Qator so'ngi
(31)
(32)   %endmacro
```

asosiy_dastur.asm

```
(33)   %include "nasm-io.inc"
(34)   %include "lotindan_kirillga.inc"
(35)
(36)   section .data
(37)   qator1  db   "Men makro vositalar haqida bilaman",0
(38)   lotindan_kirillga qator2, "Men makro vositalar haqida bilaman"
(39)
(40)   section .text
(41)   tizim_global main
(42)
(43)   main:
(44)      chop_et   `Lotin: %s \nKirill: %s \n`, qator1, qator2
(45)   ret
```

Dastur natijasi 41-rasmda tasvirlangan.



41-rasm.

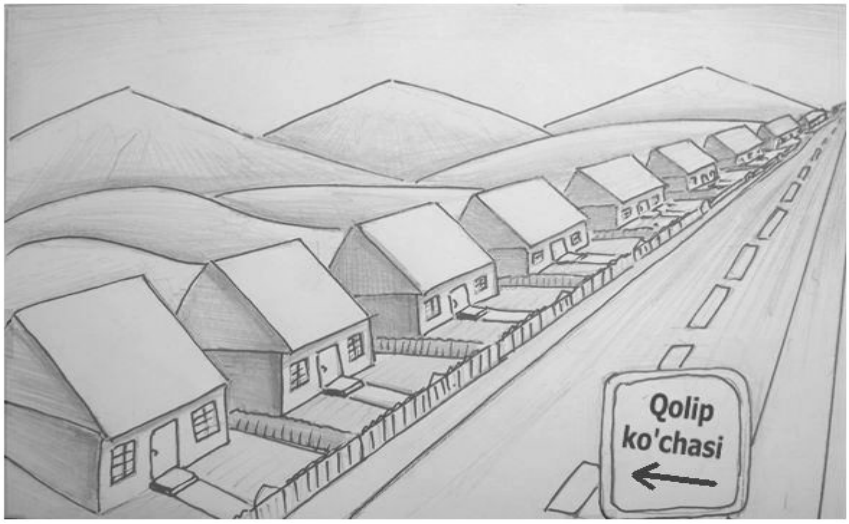
Albatta, ko'rib chiqilgan makro bir qancha xatolar bilan o'girishni amalga oshiradi. Shu sababdan mukammalroq o'giruvchi makro tuzish masalasi yechilmagancha qoldi. Shunday makroni dasturlashni o'quvchining o'ziga qoldiramiz.

X bob
Assemblerda qoliplar va jadvallar

Ushbu bobda biz dasturlashning yana bir asosiy tushunchasi – qoliplar bilan tanishamiz. Ushbu tushuncha ingliz tilida *array* deb, rus tilida esa *массив* deb yuritiladi. Barcha dasturlash tillarida bo'lgani kabi assembler ham qoliplar bilan ishlash imkoniyatini beradi. Assemblerda qoliplar bilan ishlash boshqa tillardagiga qaraganda ancha oddiy ko'rinishda amalga oshiriladi. Bu esa qoliplar aslida qanday tuzilishga ega ekanligini tushunishga yordam beradi.

Qolip deb, bir xil o'lchamda ketma-ket (davomiy) ajratilgan xotira bo'laklariga aytiladi. Qolip ham o'zgaruvchi sanaladi, ammo uning o'lchami shu paytgacha ko'rib chiqilgan o'zgaruvchilardan kattaroq bo'ladi. Boshqacha aytganda, qolip bir xil o'lchamli o'zgaruvchilar zanjiridir. Xotiradan ketma-ket ajratilgan bo'laklar qolip *kataklari* deb yuritiladi. Qolipdagi kataklar soni *qolip uzunligi* deyiladi. Qolipning xotiradan egallagan joyining baytlar soni *qolip o'lchami* deyiladi. Agar qolip bitta katagining o'lchami bir bayt bo'lsa, qolip o'lchami va uning uzunligi bir xil bo'ladi.

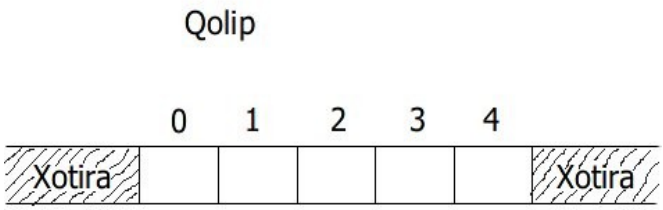
Qoliplarga hayotiy misol qilib qishloq ko'chasida joylashgan uylar ketma-ketligini keltirish mumkin (rasmga qarang). Bunda qishloq asosiy xotira bo'lsa, undagi ma'lum bir ko'cha qolip, ko'chadagi uylar esa qolip kataklari bo'ladi.



Asosiy xotira qishlog'i

42-rasm.

Qolip kataklari o'z tartib raqamiga ega bo'lib, raqamlash noldan boshlanadi. Rasmda uzunligi 5 ga teng bo'lgan qolip keltirilgan.

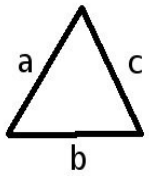


43-rasm.

Ko'rinib turibdiki, n uzunlikdagi qolipning eng oxirgi katagining tartib raqami n-1 bo'ladi. Qolipning i-chi katagi quyidagicha belgilanadi: `Qolip[i]`. Qoliplarga yaqqol misol qilib qatorli o'zgaruvchilarni keltirish mumkin, bunda qatorning o'zi qolip hisoblanib, undagi belgilar qolip kataklarida saqlanadigan qiymatlar hisoblanadi. Qatorli o'zgaruvchilar bilan ancha ishlagan bo'lsakda, ular aslida qolip hisoblanishi haqida gap ochmagan edik. Chunki, qoliplarni yaxshi anglash uchun avval xotira manzillari haqida to'liq tushunchaga ega bo'lish kerak. Mazkur bobga kelib esa biz xotira manzillari bilan yetarlicha tanishib bo'ldik.

Qoliplar dasturlashda ancha keng qo'llaniladi. Masalan, o'lcham jihatdan bir xil, ammo juda ko'p o'zgaruvchilar kerak bo'lib qolsa, ularning har bittasini alohida e'lon qilib, keyin ishlatish ko'p vaqtni oladi. Buning o'rniga xuddi shunday uzunlikka ega bo'lgan qolipdan foydalangan ma'qul. Masalan, geometrik shakllarga doir dastur tuzmoqchi bo'ldingiz deylik, uchburchak tomonlari uzunligini saqlash uchun 3 ta o'zgaruvchi kerak bo'lishi turgan gap. Siz buni osonlik bilan uddalashingiz mumkin, albatta (rasmga qarang).

```
a     resb 1
b     resb 1
c     resb 1
```



44-rasm.

Ammo ko'pburchak bilan ishlashga to'g'ri kelsachi? Masalan, o'n besh burchakni olaylik yoki bir paytning o'zida shunday shakllarning bir nechtasi bilan ishlashga to'g'ri kelsachi? Bunday vaziyatlarda har bir shakl uchun bittadan qolip belgilansa, bu ish ancha osonlashadi.

10.1. Qoliplarni e'lon qilish

Qoliplarni e'lon qilish assemblerda juda oddiy ko'rinishga ega. Siz oldin ham qatorli o'zgaruvchilarni e'lon qilganda buning guvohi bo'lgansiz.

Qolip uchun shunchaki xotiradan joy ajratish dasturning `.bss` bo'limida amalga oshiriladi.

Qoidasi:

```
qolip_nomi      resX      qolip_uzunligi
```

Bu yerda X 6-jadvalda ko'rsatilgan harflardan birortasini qabul qilib, qolipning bitta katagini o'lchamini beradi. `qolip_uzunligi` o'rnida sonli yoki ifodali o'zgarmas kelishi mumkin. `qolip_nomi` xotiradan ajratilgan joy manziliga oddiy nishon bo'lib xizmat qiladi. Demak, qolip nomida uning birinchi katagining manzili saqlanadi.

Qolip boshlang'ich qiymatlari bilan esa dasturning `.data` bo'limida e'lon qilinadi.

Qoidasi:

```
qolip_nomi      dX   qiymat_1, qiymat_2, …, qiymat_n
```

Bu yerda har bir katakning o'lchami X baytdan bo'lgan n uzunlikdagi qolip e'lon qilinadi va mos ravishda kataklarga `qiymat_1, qiymat_2` va hokazolar o'zlashtirib chiqiladi.

Misollar:

```
qolip1    resb 100        ;; 100 bayt joy ajratiladi.
```

`qolip1` dan har bir katagi bir baytdan bo'lgan 100 uzunlikdagi qolip sifatida foydalanish mumkin yoki har bir katagi ikki baytdan bo'lgan 50 uzunlikdagi qolip sifatida foydalanish ham mumkin va hokazo. Katak o'lchamlari qolip e'lon qilinishidayoq assembler direktivalari orqali aniq qilib berilishi ham mumkin:

```
qolip2    resw 50             ;; uzunlik – 50, o'lcham – 100.
qolip3    resd 25             ;; uzunlik – 25, o'lcham – 100.
```

Boshlang'ich qiymatlarni bergan holda qoliplarni e'lon qilishning bir qancha yo'llari mavjud.

```
uchburchak     db   5, 6, 7 ;; Uchta katakli qolip.
qator     times 100 db 'A'  ;;Har birida 'A' belgisi  saqlanadigan
                            ;; 100 uzunlikdagi qolip.
```

Boshlang'ich qiymatga ega bo'lgan qoliplarni e'lon qilishda makro vositalardan unumli foydalanish mumkin. Masalan, uzunligi 150 bo'lgan va 2 dan boshlab bo'lgan juft sonlarni boshlang'ich qiymat qilib oladigan qolip kerak bo'ldi deylik.

```
%assign   juft 2
juft_sonlar:
     %rep 150
          dd   juft
     %assign   juft  juft+2
     %endrep
```

Yana bir e'tiborga molik qolip turi bu qatorli o'zgaruvchilardir. Bu turdagi qoliplarni e'lon qilish usullari bilan yaxshi tanish bo'lsangizda yana bir bor ko'rib chiqamiz[1].

```
Matn db   "Salom, Men assemblerni o'rganiyapman!",0
```

10.2. Qolip kataklariga murojaat etish

Qolip nomi ham nishon bo'lgani sababli unga foydali manzil sifatida qarash mumkin. Demak, qolip nomini burchakli qavs (`[]`) ichiga olgan holda, unda saqlanayotgan qiymatlarga murojaat etish mumkin. Aniqlik uchun shuni ta'kidlash lozimki, qolip qiymatlari deganda uning kataklarida saqlanayotgan qiymatlar tushuniladi. Ma'lum bir katak manzili (1) formula orqali hisoblanadi.

$$katak\ manzili = qolip\ asosi + katak\ tartib\ raqami * bitta\ katak\ o'lchami \quad (1)$$

Bu yerda qolip asosi qolipning boshlanish manzili hisoblanadi.

Misol tariqasida, avval e'lon qilingan `juft_sonlar` qolipi bilan ishlab ko'ramiz. Ushbu qolipning har bir katak o'lchami 4 baytga tengligini hisobga olib, (1) formula bo'yicha uning nolinchi katagiga murojaat qilamiz:

```
mov  eax , [juft_sonlar + 0*4]          ;; EAX ← juft_sonlar[0]
```

Yoki har qanday sonning nol bilan ko'paytmasi nolga teng ekanidan kelib chiqib yuqoridagi buyruqni quyidagicha yozish mumkin.

```
mov  eax , [juft_sonlar]
```

[1] Qatorli o'zgaruvchilar haqida to'liq ma'lumot uchun «O'zgaruvchilarni e'lon qilish» mavzusiga qarang.

Assemblerda boshqa dasturlash tillaridan farqli o'laroq har bir katak manzilini dasturchining o'zi hisoblashiga to'g'ri keladi. Shunday qilib, qolipning nolinchi katakning manzili uning boshiga nisbatan olganda nolga teng bo'ladi. Birinchisining manzili esa, masalan, `juft_sonlar` uchun 4 ga teng bo'ladi. Ikkinchi katak manzili esa 8 ga teng va h.k. Agar `uchburchak` qolipini oladigan bo'lsak, ushbu qolip uchun kataklar manzili boshqacha bo'ladi, ya'ni birinchi katak manzili odatdagidek nol bo'lsada, ikkinchisiniki 1 ga teng bo'ladi va h.k.

Eslatma: Dasturlashda qolip kataklarining tartib raqamlari noldan boshlanganligi sababli ushbu bobda kataklar nolinchi, birinchi va hokazo deb yuritiladi. Vaholanki, nolinchi deganda qolipning ilk, ya'ni biz uchun birinchi bo'lgan katak nazarda tutiladi.

Misollar:

```
    mov eax , [juft_sonlar]
    mov ebx , [juft_sonlar + 1*4]      ;; EBX ← juft_sonlar[1]
    mov ebx , [juft_sonlar + 4]        ;; Yuqoridagi bilan bir xil.
```

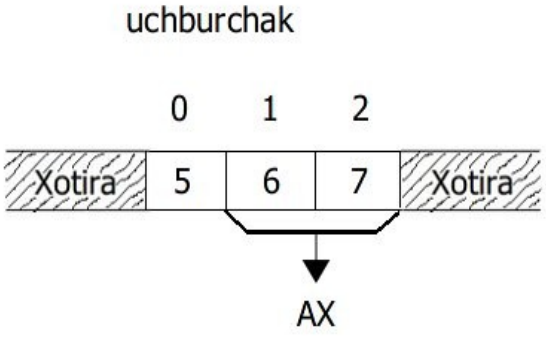
Katak tartib raqami sifatida registrlardan ham foydalansa bo'ladi. Masalan, `juft_sonlar` qolipini takrorlanish orqali barcha qiymatlarini chop etish kerak bo'ldi deylik:

```
    mov  ecx, 150
    xor  eax , eax
lp:
    chop_et   `juft_sonlar[%i] = %i \n`, eax, [juft_sonlar + eax*4]
    inc  eax
    loop lp
```

Bu yerda EAX da doim kataklar tartib raqami saqlanadi. `[juft_sonlar + eax*4]` ifodasidagi 4 soni qolip katagi o'lchamidir. O'lcham berilayotganda juda e'tiborli bo'lish talab etiladi. Quyidagi misollarni ko'rib chiqamiz:

```
    mov  al , [uchburchak + 1]         ;; AL ← uchburchak[1]
    mov  ax , [uchburchak + 1]         ;; ?, tushunarsiz qiymat.
    mov  ebx , [juft_sonlar + 12]      ;; EBX ← juft_sonlar[3]
    mov  ebx , [juft_sonlar + 3*4]     ;; Yuqoridagi bilan bir xil.
    mov  bx , [juft_sonlar + 12]       ;; ?, tushunarsiz qiymat
```

Vaziyatni yaqqol anglash uchun `uchburchak` qolipini ko'z oldimizga keltiramiz (rasmga qarang).

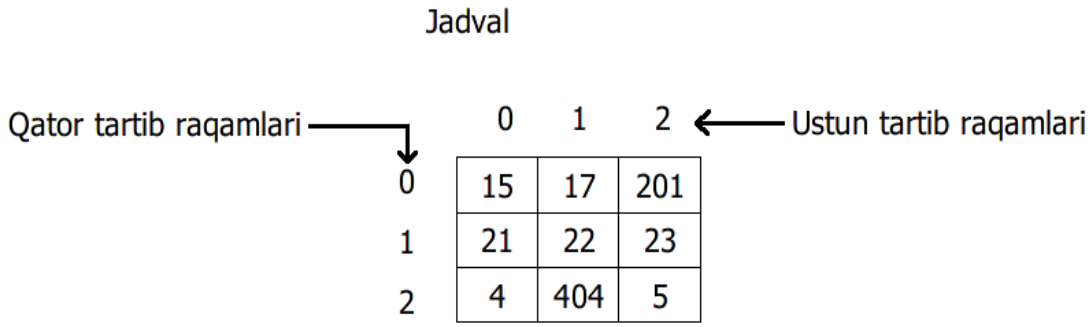


45-rasm.

`mov ax , [uchburchak + 1]` buyrug'i AX ga `uchburchak + 1` manzilidan boshlab 2 baytni yuklaydi, ya'ni birinchi va ikkinchi kataklardagi qiymati yaxlit qiymat sifatida yuklanadi. Natijada AX da tushunarsiz qiymat paydo bo'ladi. `mov bx , [juft_sonlar + 12]` buyrug'i ham shunga o'xshash, ya'ni BX ga `juft_sonlar[3]` ning kenja 2 baytini yuklaydi.

10.3. Jadvallar

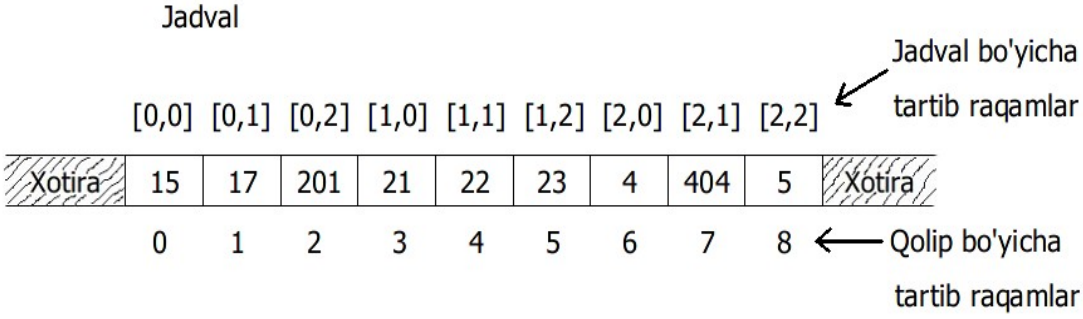
Ko'rib chiqilgan qoliplar matematik nuqtai nazardan vektor, ya'ni chiziqli kattalikdir. Matematika va dasturlashda ikki va undan ortiq o'lchamli qoliplar bilan ham ish ko'riladi. Amaliyotda esa ko'pi bilan ikki o'lchamli qoliplar qo'llaniladi. Bunday qoliplar *jadvallar* deb ataladi. Jadval bir yoki bir nechta qoliplardan iborat bo'lib, undagi har bir qolip *jadval qatori* deb yuritiladi. Jadvaldagi istalgan qatorning kataklari *jadval ustuni* orqali aniqlanadi. *Jadval uzunligi* deganda uning qatorlar va ustunlar sonining ko'paytmasi tushuniladi va quyidagicha belgilanadi: n x m. *Jadval o'lchami* deb uning uzunligi bilan bitta katagining o'lchami ko'paytmasiga aytiladi. Agar jadvalning bitta katagi o'lchami bir bayt bo'lsa, unda jadval o'lchami va uning uzunligi bir xil bo'ladi. Misol tariqasida 46-rasmda 3x3 uzunlikdagi jadval keltirilgan.



46-rasm.

Jadval har bir katagining ikkita tartib raqami bo'ladi. Tartib raqamlarning birinchisi katakning joylashgan qatori bo'yicha, ikkinchisi esa ustuni bo'yicha bo'ladi. Biz jadvalning ma'lum bir katagini quyidagicha belgilaymiz: `Jadval[i,j]`. Masalan, `Jadval[0,2]`=201, `Jadval[2,2]`=5, `Jadval[0,0]`=15 va h.k.

Jadvallar ikki yoki undan ortiq o'lchamli bo'lgani bilan ular kompyuter xotirasida bir qator, ya'ni qolip sifatida joylashtiriladi. Chunki xotira jadval singari ikki o'lchamli emas, balki avval ta'kidlangani kabi chiziqli tarzda joylashtirilgan baytlar ketma-ketligidir. Jadval xotirada uning qatorlari ketma-ket qo'yilgan holda joylashtiriladi. 47-rasmda yuqoridagi jadval xotirada qanday ko'rinishda bo'lishi tasvirlangan.



47-rasm.

Bunday olib qaraganda, kompyuter uchun qolip ham jadval ham bir narsa. Tuzilish jihatdan xotirada ular farqlanmaydi. Jadvallar yoki ko'p o'lchamli qoliplar faqat inson, ya'ni dasturchi uchun alohida ma'noga ega xolos. Demak, masalan, 3x3 bo'lgan jadval xotirada uzunligi 9 bo'lgan oddiy qolip sifatida saqlanar ekan.

Jadval kataklarining tartib raqami unga mos bo'lgan qolipnikiga nisbatan qanday bo'lishi 13-jadvalda keltirilgan.

13-jadval

Qator tartib raqami	Ustun tartib raqami	Qolip tartib raqami
0	0	0
0	1	1
0	2	2
1	0	3
1	1	4
1	2	5
2	0	6
2	1	7
2	2	8

Yuqoridagi misol 3x3 uzunlikdagi jadval uchun keltirildi, ammo 13-jadval istalgan o'lchamli qolip uchun ham bir xil qo'llanilaveriladi. Muhimi, shuni tushunish kerakki, ustun tartib raqami to'liq bir doira aylanganida qator tartib raqami bir marta o'zgaradi. Buni har 30 (yoki 31) kunda 1 oy almashinishiga o'xshatishimiz mumkin.

Jadvaldagi ma'lum bir katakning manzili quyidagi formula orqali hisoblanadi:

$$katak\ manzili = jadval\ asosi + (qator\ t.\ r. * ustunlar\ soni + ustun\ t.\ r.) * bitta\ katak\ o'lchami \quad (2)$$

Bu yerda *jadval asosi* jadvalning boshlanish manzilidir, *qator t. r.* katak joylashgan qatorning tartib raqami, *ustunlar soni* esa jadvaldagi ustunlar soni, bu jadvalning istalgan qatoridagi kataklar soniga teng. *ustun t. r.* katak joyloshgan ustun tartib raqamidir.

Aytish joizki, (2) formula qoliplar uchun berilgan (1) formula bilan bir xil kuchga ega. Masalan, `Jadval[i,j]` katagini olaylik, jadval o'lchami $n \times m$ bo'lsin. Unda (2) formulani quyidagicha yozish mumkin:

$$katak\ manzili = jadval\ asosi + (i * m + j) * bitta\ katak\ o'lchami \quad (3)$$

Jadval xotirada uzunligi $n \times m$ bo'lgan qolip sifatida joylashtirilishini hisobga olsak, unda i va j uchun shunday t topiladiki, `Jadval[i,j]=Qolip[t]` o'rinli bo'ladi. Ushbu mulohazalardan kelib chiqib (1) formulani quyidagicha yozish mumkin:

$$katak\ manzili = qolip\ asosi + t * bitta\ katak\ o'lchami \quad (4)$$

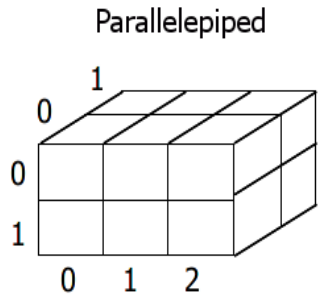
(3) va (4) formulalarini birlashtirgan holda, jadval va qolip asoslari bir narsa ekanligini hisobga olib, quyidagiga ega bo'lamiz:

$$i*m + j = t \quad (5)$$

Kelib chiqqan natijani istalgan jadval uchun tekshirib ko'rishingiz mumkin. Masalan, yuqorida keltirilgan 3×3 bo'lgan jadvalni olaylik. Undagi `Jadval[1,2]` katak tartib raqami qolip bo'yicha quyidagiga teng bo'ladi $t = 1×3 + 2 = 5$. Buni 13-jadval ham tasdiqlaydi.

Shunday qilib (1) va (2) formulalari teng kuchli ekanligi isbotlandi va jadval tartib raqamlaridan mos jadval tartib raqamiga o'tish (5) formula bo'yicha amalga oshirilishi ayon bo'ldi.

Yuqoridagi mulohazalar ko'p o'lchamli qoliplar uchun ham tegishlidir. Misol tariqasida ikki o'lchamli qolipdan kattaroq o'lchamli va amaliyotda qo'llanish bo'yicha jadvallardan so'ng keyingi o'rinda turuvchi uch o'lchamli qolipni, ya'ni parallelepipedni qisqacha ko'rib chiqamiz (48-rasmga qarang).



48-rasm.

Parallelepipedning uzunligi uning eni, bo'yi va balandligi ko'paytmasiga teng, ya'ni $n×m×p$. Umuman olganda parallelepipedni har birining uzunligi $m×p$ bo'lgan jadvallardan tashkil topgan n ta katakli qolip deb tasavvur qilishimiz mumkin. Parallelepipedning ma'lum bir katagi `Parallelepiped[i,j,k]` ko'rinishida belgilanadi. Parallelepipedga misol qilib kataklarida qatorli o'zgaruvchilar joylashgan jadvalni keltirishimiz mumkin.

`"Men"`	`"Sen"`
`"Biz"`	`"Siz"`

49-rasm.

49-rasmda 2x2x3 uzunlikdagi parallelepiped tasvirlangan. Ushbu parallelepiped qiymalari quyidagicha berilishi mumkin: `Parallelepiped[0,0]="Men", Parallelepiped[0,0,0]==='M', Parallelepiped[0,0,1]='e', Parallelepiped[1,1]="Siz".`

`Parallelepiped[i,j,k]` katagining xotiradagi manzili quyidagicha hisoblanadi:

$$katak\ manzili = parallelepiped\ asosi + (m*i + p*j + k)*\ bitta\ katak\ o'lchami \qquad (6)$$

Jadval va parallelepipeddan tashqari dasturlashda istalgan o'lchamli qoliplardan foydalanish mumkin. Masalan, n o'lchamli qolipni olaylik. Uning uzunligini quyidagicha belgilaymiz $m_1 × m_2 × … × m_n$. Ma'lum bir katagi esa `Qolip[t1,t2,…,tn]` ko'rinishida belgilanadi. Katak manzili esa quyidagi formula orqali hisoblanadi:

$$katak\ manzili = qolip\ asosi + T *\ bitta\ katak\ o'lchami \qquad (7)$$

Bu yerda:

$$T = (m_2 * m_3 * … * m_n * t_1) + (m_3 * m_4 * … * m_n * t_2) + … + (m_n * t_{n-1}) + t_n$$

(7) formulani ixchamroq bo'lgan matematik ko'rinishda ham yozish mumkin:

$$katak\ manzili\ =\ qolip\ asosi\ +\ (\sum_{i=1}^{n}[(\prod_{j=i+1}^{n} m_j)\ *\ t_i])\ *\ bitta\ katak\ o'lchami \qquad (8)$$

(8) formula istalgan o'lchamli qolipning kataklar manzilini hisoblash imkoniyatini beradi.

10.4. Jadvallarni e'lon qilish

Assemblerda jadvallar maxsus usulda e'lon qilinmaydi. Ular qoliplar qanday e'lon qilinsa, shunday e'lon qilinadi. Boshlang'ich qiymatsiz e'lon qilinadigan jadvallar dasturning `.bss` bo'limida uzunliklari berilgan holda e'lon qilinadi.

Qoidasi:

```
jadval_nomi     resX qatorlar_soni * ustunlar_soni
```

Bu yerda *X* jadvalning bitta katagining o'lchamini beradi. *X* o'rnida qanday qiymatlar kelishi 6-jadvalda ko'rsatilgan. `qatorlar_soni` o'rnida istalgan sonli o'zgarmas yoki ifoda kelishi mumkin va u jadvalda qancha qator bo'lishini anglatadi. `ustunlar_soni` esa jadvaldagi ustunlar sonini beradi.

Misollar:

```
vector     resd 10*3       ;; 10*3*4=120 bayt joy ajratiladi.
UstunY     resb 10*1       ;; Uzunligi ham, o'lchami ham 10
                           ;; bo'lgan ustun.
QatorX     resb 1*10       ;; UstunY kabi, ammo biz buni qator
                           ;; sifatida tasavvur qilishimiz mumkin.
Oddiy_qolip     resb 10    ;; UstunY va QatorX kabi.
```

Jadval kataklarining soni ikkita ko'paytuvchi sifatida berilishining sababi tushunish oson bo'lishi uchundir. Masalan, 7*3 ni ko'rgan zahotiyoq 7 ta qator va 3 ta ustundan iborat jadval ekanini anglab olish osonroq. Ammo kataklar sonining bu kabi berilishi majburiy emas, ya'ni uni siz bitta son orqali ham berishingiz mumkin. Yuqorida ta'kidlangani kabi kompyuter uchun buning hech qanday farqi yo'q.

```
parallelepiped_xyz  resw 3*10*5     ;; 300 bayt joy ajratiladi.
parallelepiped_xyz2 resw 150        ;; parallelepiped_xyz kabi.
```

Jadvallarni boshlang'ich qiymat bilan e'lon qilish, albatta, dasturning `.data` bo'limida amalga oshiriladi.

```
UstunY     db   1,2,3,4,5
QatorX     db   1,2,3,4,5
```

Jadval qiymatlarini tushunarliroq bo'lgan ko'rinishda dasturning bir nechta satrida ham berish mumkin.

```
jadval_xy dd    1,2,3,4         ;; 3x4 bo'lgan jadval
          dd    5,6,7,8
          dd    9,10,11,12
```

Yoki shuni bitta satrning o'zida ham e'lon qilish mumkin:

```
jadval_xy dd   1,2,3,4,5,6,7,8,9,10,11,12
```

Qismli dasturlash bobida ta'kidlangani kabi qism dasturning ichki qiymatlari uchun stackdan joy ajratgan ma'qul. Shunday ekan qolip uchun ham stackdan joy ajratsa bo'ladi. Buning uchun qism dasturi boshida ESP registridan qolip o'lchami ayriladi. Masalan, yuqorida keltirilgan `jadval_xy` qolipi uchun stackdan quyidagicha joy ajratiladi:

```
sub  esp , 48       ;; 4 baytdan 12 katak uchun
```

Yoki qolip o'lchamini `ENTER` buyrug'iga birinchi qiymat sifatida berish mumkin.

```
enter     48 , 0
```

10.5. Jadval kataklariga murojaat qilish

Jadval kataklariga murojaat qilish qoliplardagidan hech ham farq qilmaydi. Buning uchun burchakli qavslardan (`[]`) foydalanamiz. Ammo jadval kataklarining manzillari bir oz murakkab ko'rinishga ega bo'ladi. Chunki jadvalga bir o'lchamli qolip sifatida murojaat qilinadi.

Misollar:

```
mov  eax , [jadval_xy]                  ;; EAX ← jadval_xy[0,0]
mov  ebx , [jadval_xy+(1*4+3)*4]        ;; EBX ← jadval_xy[1,3]
mov  ecx , [jadval_xy+(2*4+2)*4]        ;; ECX ← jadval_xy[2,2]
```

Bu yerda katak tartib raqamlari (2) formula orqali hisoblandi. Ammo yuqoridagi foydali manzillar (5) formula yordamida to'g'ridan-to'g'ri hisoblanishi mumkin.

```
mov  ebx , [jadval_xy+7*4]          ;; EBX ← jadval_xy[7], ya'ni 8
```

Umuman olganda, `jadval_xy` dan 3x4 bo'lgan jadval yoki uzunligi 12 bo'lgan bir o'lchamli qolip sifatida foydalanish mumkin. Aynan qaysi usulda foydali manzillarni berish vaziyat va dasturchining o'ziga bog'liq.

Yodingizga bo'lsa foydali manzillarni berishda registrlardan ham foydalanish mumkin edi. Ushbu imkoniyatdan qoliplar bilan ishlaganda unumli foydalanish mumkin. Gap shundaki, har doim ham yuqorida ko'rsatilgani kabi tartib raqamlarning sonli o'zgarmas sifatida berilishi qulay bo'lavermaydi. Ba'zida tartib raqam o'zgaruvchi orqali berilishi kerak bo'lib qolishi mumkin. Shunday holatlarda tartib raqam qilib registrlarni ishlatgan maqsadga muvofiq sanaladi. Masalan, `jadval_xy` ning barcha kataklaridagi qiymatlarni chop etish kerak bo'lsa, 12 marta `chop_et` ni yozib chiqqandan ko'ra oddiy takrorlanishdan foydalangan ustundir.

Foydali manzillar tarkibida registrlar «`LEA` buyrug'i» mavzusida ko'rsatilgan andoza bo'yicha qo'llanishi kerak. Quyida foydali manzil andozasi qoliplar bilan ishlaganda asosan qanday ko'rinishga ega bo'lishi ko'rsatilgan.

$$[qolip\ asos + tartib\ raqam\ registri * bitta\ katak\ o'lchami \pm o'zgarmas\ ifoda] \quad (9)$$

Bu yerda asos manzilidan boshqa barcha hadlarning ishlatilishi majburiy emas. Asos manzili bo'lib nishon yoki qolipning istalgan katagining manzili yuklangan 4 baytli registr kelishi mumkin. Qolip manzillari asosan EDI va ESI registrlarida beriladi. *tartib raqam registrlari* bo'lib esa EAX, EBX, ECX, EDX, EBP, ESI va EDI lar kelishi mumkin. *bitta katak o'lchami* o'rnida faqat 2, 4 yoki 8

sonlari kelishi mumkin. O'zgarmas ifoda ba'zida mo'ljalni o'ngga yoki chapga surish uchun kerak bo'ladi.

Keltirilgan foydali manzil andozasi har doim xuddi shu ko'rinishda bo'lishi shart emas. (9) andozasida faqat eng ixcham ko'rinish keltirilgan. Amaliyotda qo'llanadigan ko'rinishlar esa ixchamlanganda, (9) andozaga mos kelishining o'zi kifoyadir. Masalan, assembler ifodalarida qavslardan foydalanish mumkin va quyidagi misol garchi (9) ga mos kelmasada to'g'ri hisoblanadi.

```
lea  edi , [jadval_xy + (ecx*4 + 2)*4]
```

Chunki qavs ochilib ifoda ixchamlashtirilganda `[jadval_xy + ecx*16 + 8]` kelib chiqayapti.

Qoliplar bilan ishlaganda yana bir kerakli anjom `LEA` buyrug'idir. Jadvallardan takrorlanishlarda foydalanganda, har safar jadvalning navbatdagi qatorining manzili biror registrga `LEA` orqali yuklab olib, shundan so'ng takrorlanadigan dastur bo'lagida o'sha qator bilan ish ko'rgan juda qulay hisoblanadi.

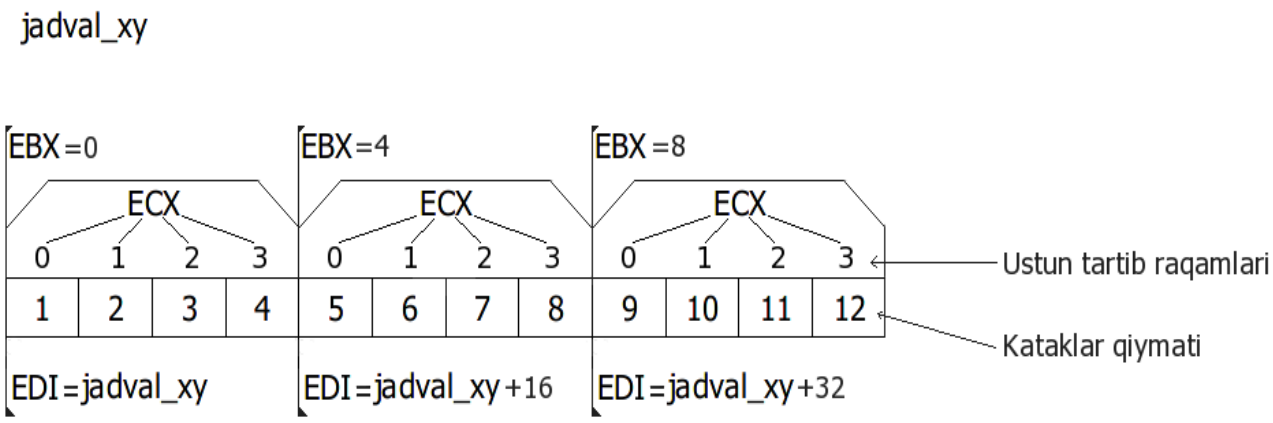
Eslatma: Foydali manzil tarkibidagi barcha registrlar 4 baytli bo'lishi shart. Chunki `LEA` manzillar bilan ishlaydi, manzillar esa 4 baytli qiymat sifatida qaraladi.

Misol tariqasida `jadval_xy` ning barcha qiymatlariga 2 ni qo'shib, ularni jadval ko'rinishida chop etamiz. Buning uchun bizga ikkita ichma-ich joylashtirilgan takrorlanishlar kerak bo'ladi. Birinchi takrorlanish qator tartib raqamini oshirib borsa, uning ichidagi ikkinchi takrorlanish ustun tartib raqamini oshirib boradi.

```
(1)       %define   bitta_katak_o  4    ;; Bitta katak o'lchami.
(2)       %define   qatorlar_soni  3
(3)       %define   ustunlar_soni  4
(4)       %define   jadval_uzunligi ustunlar_soni*qatorlar_soni
(5)       ;;...
(6)       mov  ebx , 0                  ;; Qolip bo'yicha tartib raqam.
(7)       mov  ecx , 0                  ;; Ustun tartib raqami.
(8)
(9)   .qator_tak:
(10)      cmp  ebx , jadval_uzunligi
(11)      jz   .yakun
(12)      lea  edi,[jadval_xy+ebx*bitta_katak_o]  ;;(1) formula asosida
(13)
(14)  .ustun_tak:
(15)      cmp  ecx , ustunlar_soni
(16)      jz   .keyingi_qator
(17)      add  dword[edi + ecx*bitta_katak_o] , 2
(18)      chop_et   `%i `, [edi + ecx*bitta_katak_o]
(19)      inc  ecx
(20)      jmp  .ustun_tak
(21)
(22)  .keyingi_qator:
(23)      add  ebx , ustunlar_soni
(24)      xor  ecx , ecx
(25)      chop_et   ` \n`
(26)      jmp  .qator_tak
(27)
(28)  .yakun:
```

Bu yerda qo'llanilgan usulning asosiy g'oyasi shundaki, qatorlar bo'yicha bo'lgan katta takrorlanishda joriy qator boshlanish manzili EDI ga yuklanib, so'ng o'sha registr bilan xuddi oddiy bir o'lchamli qolip singari ish ko'rilayapti.

EBX da har doim navbatdagi qatorning birinchi katak tartib raqami (5) formula bo'yicha hisoblanib saqlanadi. Masalan, birinchi takrorlanishda, `jadval_xy[0,0]` uchun EBX=0*4+0=0 bo'ladi, ikkinchi takrorlanishda esa, `jadval_xy[1,0]` uchun EBX=1*4+0=4 bo'ladi va hokazo. To'liq jarayon 50-rasmda tushunarli qilib tasvirlangan.



50-rasm.

Ushbu misolda makro vositalardan keng foydalanildi. Qolipga tegishli o'zgarmaslarni shu kabi aniqlab, so'ng dasturda faqat makrolarni ishlatgan ma'qul. Chunki keyinchalik, masalan, jadval o'lchamlarini o'zgartirishga to'g'ri kelsa, makro aniqlashlarga o'zgartirish kiritiladi xolos (dastur esa o'zgartirishlarsiz qoladi).

Ikki va undan ortiq o'lchamli qoliplar uchun ham masala shu kabi yechiladi. Masalan, uch o'lchamli parallelepiped kataklariga bir boshdan murojaatni amalga oshirish uchun uchta ichma-ich joylashtirilgan takrorlanishlar qo'llaniladi.

Stackdan ajratilgan qoliplarga murojaat etishga kelsak, «Qismli dasturlash» bobida aytib o'tilganidek ular bilan EBP registri yordamida ishlanadi. Masalan, qism dasturida `jadval_xy` va yana bir 4 baytli `son` mahalliy o'zgaruvchisi uchun stackdan joy ajratilgan deylik:

```
enter     12*4 + 4 , 0
```

Shundan so'ng `[EBP-4]` ga son o'zgaruvchisi sifatida qaraladi, `jadval_xy` esa `EBP-52` manzilidan boshlanadi. Ushbu manzillarni tezda nomlab qo'ygan ma'qul.

```
%define   son   [ebp-4]
%define   jadval_xy   ebp-52
```

Yuqoridagi dasturni mahalliy qiymatlar bilan ishlaydigan qilib o'zgartirishni o'quvchining o'ziga qoldiramiz.

10.6. Qoliplapga mo'ljallangan buyruqlar

O'zingiz guvohi bo'lganingiz kabi qoliplar bilan ishlaganda juda ko'p takrorlanishlardan foydalanishga to'g'ri keladi. Bu esa dasturlashni birmuncha qiyinlashtiradi. Shuni hisobga olgan holda aynan qoliplar bilan ishlashga mo'ljallangan assembler buyruqlari mavjud. Bu buyruqlar

asosan tartib raqam registrlari bo'lmish EDI va ESI lar bilan ish ko'radi. Tartib raqam degan bilan EDI va ESI da katak tartib raqami emas, balki uning manzili saqlanadi.

`LODSX` buyruqlar oilasi manba tartib raqami registri hisoblanuvchi ESI dagi manzilda saqlanayotgan qiymatni mos ravishda AL/AX/EAX ga yuklaydi va ESI dagi manzilni keyingi katak manzilga surib qo'yadi. Qolipning bitta katagining o'lchami necha baytligiga qarab `X` o'rnida 6-jadvaldagi `B`, `W` yoki `D` harflaridan biri kelishi mumkin (bundan keyin beriladigan buyruqlarga ham ushbu qoida tegishli). Masalan, `DW` orqali e'lon qilingan qolip uchun `LODSW` dan foydalaniladi.

`STOSX` buyruqlar oilasi mos ravishda AL/AX/EAX dagi qiymatni maqsad tartib raqami registri hisoblanuvchi EDI dagi manzilga yuklaydi va EDI dagi manzilni keyingi katak manzilga surib qo'yadi.

Ushbu buyruqlar oilasi quyidagi a'zolardan iborat:

STOSB	AL dagi qiymatni `[EDI]` ga yuklaydi va EDI ning qiymatini bittaga oshiradi. Qiymat bittaga oshirilishining sababi EDI qolipning keyingi katagining manziliga o'tishi kerak.
STOSW	AX dagi qiymatni `[EDI]` ga yuklaydi va EDI ning qiymatini ikkitaga oshiradi.
STOSD	EAX dagi qiymatni `[EDI]` ga yuklaydi va EDI ning qiymatini to'rttaga oshiradi.

Qoliplar uchun mo'ljallangan ushbu buyruqlardan takrorlanishda foydalanish juda qulay. Chunki qolip katagining o'lchami haqida bosh qotirib turilmaydi. `LODSX` va `STOSX` buyruqlari bir qolip qiymatlarini ikkinchisiga ko'chirishga juda qo'l keladi. Keyingi misolda boshlang'ich qiymatlarga ega bo'lgan `qolip1` qolipining qiymatlari `qolip2` ga yuklanadi.

15-namuna: Qolipdan qolipga

```
(1)  %define   KATAKLAR_SONI  10
(2)  %include "nasm-io.inc"
(3)
(4)  section .data
(5)  qolip1    dw   1,3,5,7,11,13,17,19,23,29
(6)
(7)  section .bss
(8)  qolip2    resw 10
(9)
(10) section .text
(11) tizim_global   main
(12)
(13) main:
(14)      cld                       ;; Juda muhim!
(15)      lea  esi , [qolip1]
(16)      lea  edi , [qolip2]
(17)      mov  ecx , KATAKLAR_SONI
(18)
(19) .lp_boshi:
(20)      lodsw                     ;; AX ← qolip1[i]
(21)      stosw                     ;; qolip2[i] ← AX
(22)      loop .lp_boshi
```

```
(23)
(24)        mov  ecx , KATAKLAR_SONI
(25)        xor  eax , eax
(26)        xor  ebx , ebx
(27)        lea  esi , [qolip2]
(28)
(29)  .lp_boshi2:
(30)        lodsw
(31)        chop_et   `qolip2[%i] = %i \n`, ebx, eax
(32)        inc  ebx
(33)        loop .lp_boshi2
(34)
(35)  ret
```

Dasturning 14-qatoridagi `CLD` buyrug'i juda muhim o'ringa ega. `LODSX` va `STOSX` buyruqlari tanishtirilganida, ular yuklashlarni amalga oshirishgach, ESI va EDI larni oldinga, ya'ni keyingi katak manziliga suradi deyilgan edi. Ammo bu faqat EFLAGS registridagi DF (**D**irection **F**lag - Yo'nalish bayrogi) o'rnatilmagandagina to'g'ri. Agar DF o'rnatilgan bo'lsa, tartib raqam registrlarining qiymatlari oldinga emas, balki bitta katak orqaga suriladi. `CLD` buyrug'i esa DF=0 bo'lishini ta'minlaydi. Demak, DF ning qiymati orqali `LODSX` va `STOSX` buyruqlarini boshqarish mumkin ekan.

Ba'zida qolipning oxiriga borib qolganda orqa tomonga yuritish ham kerak bo'lib qoladi. Buning uchun DF ning qiymatini o'rnatish kerak. Bu esa `STD` buyrug'i orqali bajariladi.

Eslatma: `LODSX` va `STOSX` buyruqlar oilasidan foydalanayotganda, albatta, DF bayrog'ining qiymatini hisobga olish kerak. Boshqacha qilib aytganda, yo `CLD` dan, yo `STD` dan foydalanish kerak.

Yana bir keyinchalik muomalaga kirgan buyruq `MOVSX` bo'lib, u `LODSX` va `STOSX` larni o'rnini bosadi. 15-namunadagi 20- va 21-qatorlar quyidagiga almashtirilishi mumkin:

```
movsw
```

`MOVSX` buyruqlar oilasi ham DF qiymatiga ko'ra ish ko'radi.

Yana bir foydali buyruq `REP` bo'lib, u o'ziga qiymat qilib berilgan boshqa bir buyruqni ma'lum son marta takror bajaradi. Takrorlanishlar soni sifatida ECX qiymati olinadi. Demak, 15-namunadagi 19 dan 22 gacha bo'lgan satrlar quyidagicha almashtirilishi mumkin:

```
rep movsw
```

`REPX` buyruqlar oilasiga mansub yana ikkita qo'shimcha `REPZ` va `REPNZ` buyruqlari mavjud bo'lib, ular ECX ning qiymati bilan bir qatorda mos ravishda ZF bayrog'ining o'rnatilgan yoki o'rnatilmaganligini ham hisobga oladi. `REPZ` buyrug'ining oddiy `REP` dan farqi shundaki, u ECX qiymatidan tashqari ZF bayrog'i o'rnatilganligini ham tekshiradi. Agar takrorlanish chog'ida ZF=0 bo'lib qolsa, ECX qiymatidan qat'i nazar takrorlanish to'xtatiladi. `REPNZ` esa `REPZ` ning teskari bo'lib, ZF ning o'rnatilmaganligini tekshiradi.

Eslatma: Har qanday buyruqning bajarilishi EFLAGS registrining qiymatiga ta'sir qilishi mumkin.

Ushbu buyruqlarning nimaga kerakligi tushunarsiz bo'lsa kerak. Quyida yoritiladigan buyruqlar oilasi ularni yaxshiroq tushunishingizga yordam beradi.

CMPSX buyruqlar oilasi 2 ta qolip kataklarini bir-biri bilan taqqoslaydi. Ushbu buyruqlar oilasi oldin ko'rib o'tganimiz CMP taqqoslash buyrug'iga o'xshab natijani EFLAGS registriga o'rnatadi, ya'ni taqqoslashdan so'ng tegishli bayroqlar o'rnatiladi. CMPSX buyruqlari bilan ishlaganda ham qolip kataklari o'lchamlari haqida dasturchi o'ylamasa ham bo'ladi.

Eslatma: Taqqoslanayotgan qoliplarning katak o'lchamlari bir xil bo'lishi kerak.

CMPSX buyrug'i orqali 2 ta bir xil uzunlikdagi qatorli o'zgaruvchilarni taqqoslash juda qulay.

```
        cld
        lea  esi , [fayl_nomi1]
        lea  edi , [fayl_nomi2]
        mov  ecx , FAYL_NOMI_UZUNLIGI
        repz cmpsb
        jz   .teng
        chop_et   `Qatorlar teng emas \n`
        jmp  .chetlash

    .teng:
        chop_et   `Qatorlar o'zaro teng \n`

    .chetlash:
```

Qatorlarni taqqoslaganda takrorlanish, ya'ni REPZ buyrug'i qanday natija bilan tugaganini aniqlash juda muhim. Chunki bu yerda 2 xil vaziyat yuz berishi mumkin. REPZ buyrug'i ma'lum bir kataklarning qiymatlari teng chiqmaganligi evaziga takrorlanishni to'xtatgan bo'lishi mumkin yoki qatorlarning barcha kataklari qiymatlari teng chiqib, ya'ni doim ZF o'rnatilgan bo'lib, REPZ buyrug'i ECX marta takrorlanishni amalga oshirgan bo'lishi mumkin. Shuning uchun ham REPZ buyrug'idan so'ng JZ orqali ZF o'rnatilgan yoki o'rnatilmaganligini tekshiramiz.

Agar qolip kataklari ichidan qandaydir qiymatni qidirish kerak bo'lsa, SCASX buyruqlar oilasidan foydalanish mumkin. Qidiriladigan qiymat o'lchamiga qarab mos ravishda AL/AX/EAX registrlarining birortasiga joylashtirasiz. Qidiruv amalga oshiriladigan qolip manzili EDI da bo'lishi kerak. Misol tariqasida jadvalda 100 soni bor yoki yo'qligini tekshirib ko'ramiz.

16-namuna: Qolipdan 100 ni qidirish

```
(1)    %define  JADVAL_UZUNLIGI     6
(2)    %include "nasm-io.inc"
(3)    section .data
(4)    jadval   db   99, 98, 100, 2, 55, 56
(5)
(6)    section  text
(7)    tizim_global  main
(8)
(9)    main:
(10)       xor  eax , eax      ;; Har ehtimolga qarshi EAX ni nollaymiz
(11)       lea  edi , [jadval]
(12)       mov  ecx , JADVAL_UZUNLIGI
(13)       mov  al , 100       ;; Qidiriladigan qiymat
(14)       repnz     scasb
(15)       jz   .topildi
```

```
(16)      chop_et   `Jadvalda %i soni yo'q \n` , eax
(17)      jmp  .chetlash
(18)
(19)  .topildi:
(20)      dec  edi
(21)  chop_et  `%i soni jadvalda topildi. Uning manzili: %p`,eax,edi
(22)
(23)  .chetlash:
(24)
(25)  ret
```

Qidirilayotgan qiymat topilgan taqdirda ham `SCASB` buyrug'i EDI ning qiymatini keyingi katakga surishga ulguradi. Shuning uchun 20-qatorda EDI ning qiymati bitta katak orqaga siljitiladi.

Yuqorida ko'rib chiqilgan misollar dasturlashda juda kerakli bo'lib, ulardan qism dasturlari tuzing va keyingi dasturlaringizda ulardan foydalanib boring. Demak, quyidagi qism dasturlari yuzaga kelishi kerak:

- Qolip qiymatlarini boshqa qolipga ko'chirib beradigan qism dasturi. Ushbu qism dasturi C da quyidagicha e'lon qilinadi:

  ```
  void ko_chirish(int * qolip1, int * qolip2, int uzunlik);
  ```

  `qolip1` qiymatlaridan `qolip2` ga nusxa olinadi(ko'chiriladi), `uzunlik` - `qolip1` ning uzunligi.
- Ikkita qatorni bir-biri bilan taqqoslaydigan qism dasturi. Natija sifatida agar qatorlar o'zaro teng bo'lsa 0, teng bo'lmasa 1 qaytariladi.

  ```
  int  qatorni_taqqosla(char * qator1, char * qator2, int uzunlik);
  ```

  `uzunlik` - `qator1` ning uzunligi.
- Berilgan qiymatni qolipdan qidiradigan qism dasturi. Natija sifatida agar shunday qiymat topilsa, uning qolipdagi tartib raqami, aks holda -1 qaytariladi.

  ```
  int  qolipdan_qidir(int * qolip, int qiymat, int uzunlik);
  ```

  `uzunlik` - `qolip` ning uzunligi.

Tuzgan qism dasturlaringizni C dagi dastarlardan chaqirishga harakat qilib ko'ring. Bunig uchun ommaviy va tashqi nishonlarni e'lon qilishda odatiy `global` va `extern` direktivalari o'rnida «Makro vositalar» bobidan ko'rib chiqilgan `tizim_global` va `tizim_extern` makrolaridan foydalaning.

Misol tariqasida oxirgi aytib o'tilgan qism dasturini tuzib ko'rsatamiz.

17-namuna: qolipdan_qiymat_qidirish_qism_dasturi.asm

```
(1)   %include "nasm-io.inc"       ;; tizim_global uchun
(2)
(3)   section .text
(4)   tizim_global qolipdan_qidir
(5)
(6)   ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
(7)   ;; Qolipda qidirish qism dasturi.
(8)   ;;
(9)   ;; Maqsad:
```

```
(10)  ;;  Berilgan qiymatni qolipdan qidiradigan qism dasturi.
(11)  ;;  C dagi e'lon qilinishi:
(12)  ;;  int qolipdan_qidir(char * qolip, char qiymat, int uzunlik);
(13)  ;;
(14)  ;; Qiymatlar:
(15)  ;;  qolip - qidiruv amalga oshiriladigan qolip manzili
(16)  ;;  qiymat - qidiriladigan qiymat
(17)  ;;  uzunlik - qolip uzunligi
(18)  ;;
(19)  ;; Natija:
(20)  ;;  Agar qiymat topilsa uning qolipdagi tartib raqami,
(21)  ;;  aks holda -1 qaytariladi.
(22)  ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
(23)
(24)  %define  qolip     ebp+8
(25)  %define  qiymat    [ebp+12]
(26)  %define  uzunlik   [ebp+16]
(27)
(28)  qolipdan_qidir:
(29)      enter     0 , 0
(30)      push edi            ;; Ishlatiladigan registrlarning
(31)      push ecx            ;; qiymatini stackda saqlab qo'yamiz.
(32)
(33)      mov  edi , [qolip]
(34)      mov  al , qiymat
(35)      mov  ecx , uzunlik
(36)
(37)      repnz     scasb
(38)      jz   .topildi
(39)      mov  eax , -1
(40)      jmp  .chetlash
(41)
(42)  .topildi:
(43)      mov  eax , uzunlik           ;; EAX <- uzunlik
(44)      neg  ecx                     ;; ECX <- -ECX
(45)      lea  eax , [eax + ecx - 1]   ;; EAX <- EAX + (-ECX) -1
(46)
(47)  .chetlash:
(48)      pop  ecx
(49)      pop  edi
(50)      leave
(51)
(52)  ret
```

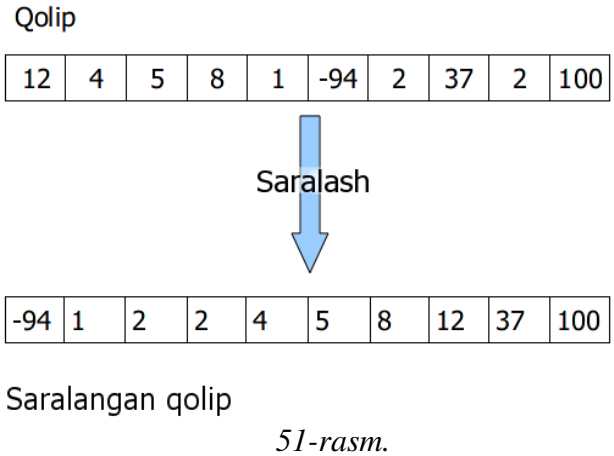
asosiy_dastur.c

```
(1)   #include <stdio.h>
(2)
(3)   int qolipdan_qidir(char * qolip, char qiymat, int uzunlik)
(4)    __attribute__ ((cdecl));
(5)
(6)   void main()
(7)   {
(8)       char *qator = "abcdefght", qiymat1 = 'c';
(9)       char qolip[6] = {99, 98, 100, 2, 55, 56}, qiymat2 = 65;
```

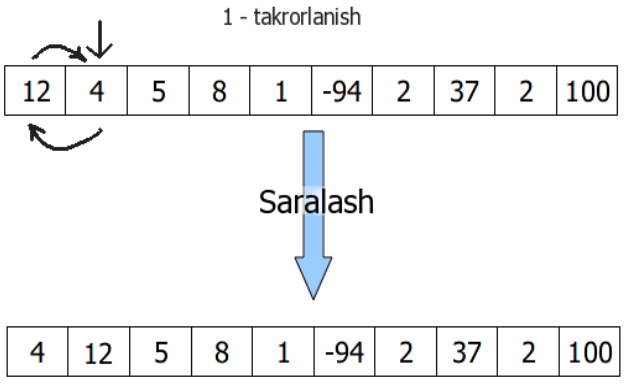
```
(10)        int  natija;
(11)
(12)        natija = qolipdan_qidir(qator, qiymat1, 9);
(13)
(14)        if(natija>=0)
(15)        printf("Topildi: qator[%i] = %c. \n", natija, qator[natija]);
(16)        else printf("Qatordan %c topilmadi \n", qiymat1);
(17)
(18)        natija = qolipdan_qidir(qolip, qiymat2, 6);
(19)
(20)        if(natija>=0)
(21)        printf("Topildi: qolip[%i] = %c. \n", natija, qolip[natija]);
(22)        else printf("Qolipdan %c topilmadi. \n", qiymat2);
(23)   }
```

10.7. Amaliy dasturlar: Saralash

Qoliplar bilan ishlashga doir juda ham ko'p misollar keltirish mumkin. Ushbu mavzuda shunday misollarning eng mashhuri bo'lgan qolipni saralash dasturi ko'rib chiqiladi. Saralash deganda qolipdagi tartibsiz joylashgan qiymatlarni kichikdan kattasiga yoki aksincha kattasidan kichigiga qarab tartiblab joylashtirish tushuniladi. Biz ushbu mavzuda kichikdan kattaga qarab saralab boradigan dasturni ko'rib chiqamiz. 51-rasmda qolip va uning saralashdan keyingi ko'rinishi keltirilgan.



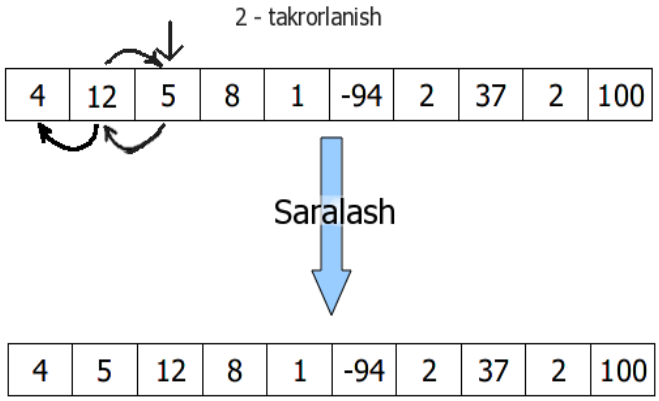
51-rasm.



52-rasm, a.

Saralashning bir qancha yo'llari bo'lib, biz ancha keng qo'llanadigan *pufakchalar usulidan* foydalanamiz. Ushbu usul ingliz tilida *bubble sort*, rus tilida esa *метод пузырки* deb yuritiladi. Ushbu usulda qolipning birinchidan boshlab oxirigacha bo'lgan har bir katagi barcha o'zidan oldingi kataklar bilan taqqoslanib chiqiladi.

Agar katak qiymati o'zidan oldin joylashgan kataknikidan kichik bo'lsa, ushbu kataklar o'rtasida o'zaro qiymat almashinuvi yuz beradi. 52-a va 52-b rasmlarda birinchi va ikkinchi kataklar qanday saralanishi ko'rsatilgan.



52-rasm, b.

Saralash qism dasturi sifatida tuziladi. C dasturchilari uchun qism qasturining C dagi ko'rinishi ham keltirilgan.

18-namuna: Qolipni saralash.

```
(1)   %include "nasm-io.inc"
(2)
(3)   %define  qolip_uzunligi 10
(4)
(5)   section .data
(6)   saralanadigan_qolip     dd   12, 4, 5, 8, 1, -94, 2, 37, 2, -100
(7)
(8)   section  .text
(9)   tizim_global  main
(10)  main:
(11)
(12)      push dword qolip_uzunligi
(13)      push dword saralanadigan_qolip
(14)      call saralash
(15)      add  esp , 8
(16)
(17)      mov  ecx , qolip_uzunligi
(18)      xor  eax , eax
(19)
(20)  .lp_boshi2:
(21)      chop_et   `saralanadigan_qolip[%i] = %i\n`, eax,\
(22)                [saralanadigan_qolip + eax*4]
(23)      inc  eax
(24)      loop .lp_boshi2
(25)
(26)  ret
```

```
(27)
(28)  ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
(29)  ;; Saralash qism dasturi.
(30)  ;;
(31)  ;; Maqsad:
(32)  ;;  Qolip qiymatlari kichikdan kattaga qarab saralanadi.
(33)  ;;  C dagi e'lon qilinishi:
(34)  ;;       void saralash(int * qolip, unsigned uzunlik);
(35)  ;;
(36)  ;; Qiymatlar:
(37)  ;;       qolip - saralanadigan qolip manzili
(38)  ;;       uzunlik - qolip uzunligi
(39)  ;;
(40)  ;; Qism dasturining C dagi ko'rinishi:
(41)  ;;
(42)  ;;  void saralash(int * qolip, unsigned uzunlik)
(43)  ;;  {
(44)  ;;       unsigned i,j;
(45)  ;;       int vaqtincha;
(46)  ;;
(47)  ;;       for(i = 1; i < uzunlik; i++)
(48)  ;;            for(j = 0; j < i; j++)
(49)  ;;            {
(50)  ;;                 if(qolip[j] > qolip[i])
(51)  ;;                 {
(52)  ;;                      vaqtincha = qolip[j];
(53)  ;;                      qolip[j] = qolip[i];
(54)  ;;                      qolip[i] = vaqtincha;
(55)  ;;                 }
(56)  ;;            }
(57)  ;;  }
(58)  ;;
(59)  ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
(60)
(61)  %define  qolip     ebp+8
(62)  %define  uzunlik   [ebp+12]
(63)
(64)  saralash:
(65)      enter     0 , 0
(66)      mov  edi , [qolip]
(67)      xor  ecx , ecx                    ;; i
(68)
(69)  .lp_katta:
(70)      inc  ecx
(71)      cmp  ecx , uzunlik
(72)      jae  .tamom
(73)      xor  ebx , ebx                    ;; j
(74)
(75)  .lp_kichik:
(76)      cmp  ebx , ecx
(77)      jae  .lp_katta
(78)      mov  eax , [edi + ecx*4]          ;; EAX <- Qolip[i]
(79)      cmp  [edi+ebx*4] , eax            ;; if(qolip[j] > qolip[i])
(80)      jle  .almashtirish_kerak_emas
```

```
(81)
(82)        mov  edx , dword[edi + ebx*4]      ;; EDX <- Qolip[j]
(83)        mov  dword[edi + ebx*4] , eax      ;; Qolip[j] <- Qolip[i]
(84)        mov  dword[edi + ecx*4] , edx      ;; Qolip[i] <- EDX
(85)
(86)    .almashtirish_kerak_emas:
(87)        inc  ebx
(88)        jmp  .lp_kichik
(89)
(90)    .tamom:
(91)        leave
(92)    ret
```

XI bob
O'nli kasrlar

Ushbu bobda biz o'nli kasrlarni kompyuter xotirasida qanday qilib saqlanishi va assember tilida ular bilan ishlashni o'rganamiz. «Sanoq tizimlari» bobida o'nli kasrlarni har xil sanoq tizimlarida qanday ko'rinishga ega ekanligi ko'rsatilgan bo'lsa, endi ularni dasturlashda aynan qanday tatbiq etish haqida so'z yuritiladi.

11.1. Sonli qo'shma protsessor

Dasturlash taraqqiy topib borishi bilan protsessorlarni ham mukammallashtirish talab qila boshlandi. Xususan, o'nli kasr sonlar bilan ishlaydigan protsessorlarga ehtiyoj vujudga keldi. Shuning yuzasidan turli xil protsessorlar ishlab chiqaradigan korxonalar o'zlarining yechimlarini taklif eta boshlashdi. Gap o'nli kasr sonning kompyuter xotirasida qanday andoza bo'yicha saqlanishi va u bilan ishlay oladigan buyruqlar to'plami haqida borayapti. Yechimlar har xil va bir-biriga to'g'ri kelmaydigan edi. Bir protsessorga mo'ljallab yozilgan dastur ikkinchisiga to'g'ri kelmas edi. Qisqasini aytganda, marra iloji boricha ko'proq andozalar bilan ishlaydigan protsessor ishlab chiqqanniki bo'lgan. Bu borada Intel korxonasi barcha raqobatchilarini dog'da qoldiradi. U 8086 protsessor turkumidan boshlab o'nli kasr sonlar bilan ishlashga mo'ljallangan *sonli qo'shma protsessorni* ishlab chiqadi. Sonli qo'shma protsessor markaziy protsessordan alohida joylashgan bo'lib, o'z buyruqlar va registrlar to'plamiga ega.

Intel ishlab chiqqan yechim o'sha paytda mavjud bo'lgan bir qancha kasr sonlar andozasi bilan ishlay olgan. Bu esa unga yaqin ta'qibchisi IBM dan ancha o'zib ketishga imkon yaratadi. Ushbu bobda ham aynan Intel tomonidan ishlab chiqilgan SQP(**S**onli **Q**o'shma **P**rotsessor) ga mo'ljallangan assembler buyruqlari ko'rib chiqiladi. Ushbu protsessor ingliz tilida *numeric coprocessor* yoki qisqartirilgan ko'rinishda *FPU* deb yuritiladi.

11.2. O'nli kasr sonlar andozasi

Hozirda IEEE tomonidan ishlab chiqilgan o'nli kasr sonlarni xotirada saqlash andozasi umumiy me'yor sifatida asosiy protsessor ishlab chiqaruvchilar tomonidan qo'llaniladi. Intel protsessorlari ham aynan shu andozadan foydalanadi. Quyida foydalaniladigan barcha tushunchalar ham shu mezonga asosalanadi.

Ushbu bob o'nli kasr sonlarga bag'ishlangani sababli, quyida o'nli kasr sonlar shunchaki son atamasi bilan yuritiladi. Umuman olganda, o'nli kasr sonlar dasturlashda biroz boshqacha nomlanadi. Masalan, ingliz tilida *floating point number*, rus tilida esa *число с плавающей точкой* deb ataladi. Bu *suzuvchi nuqtali son* degan ma'noni beradi, chunki sondagi kasr nuqtasi oldinga yoki orqaga istalgancha surilishi mumkin. Bu suzib yurgan nuqta tasavvurini beradi. Masalan, 37.512 sonini quyidagi ko'rinishlarda yozish mumkin: 37.512, $3.7512*10^{1}$, $0.37512*10^{2}$, $3751.2*10^{-2}$.

Ma'lumki, kompyuterda sonlar ikkilik sanoq tizimida saqlanadi. Masalan, 37.512 sonining ikkilik ko'rinishi 100101.10000011_2 bo'ladi. Ammo sonni bu ko'rinishda kompyuter xotirasida saqlab bo'lmaydi, chunki kasr nuqtasi ham bor.

Bu masalani hal etishda sonni *maromlashtirish* tushunchasidan foydalanilgan. Agar son quyidagi ko'rinishda ifodalangan bo'lsa, u maromlashtirilgan deyiladi:

$$1.abcds...* p^d$$

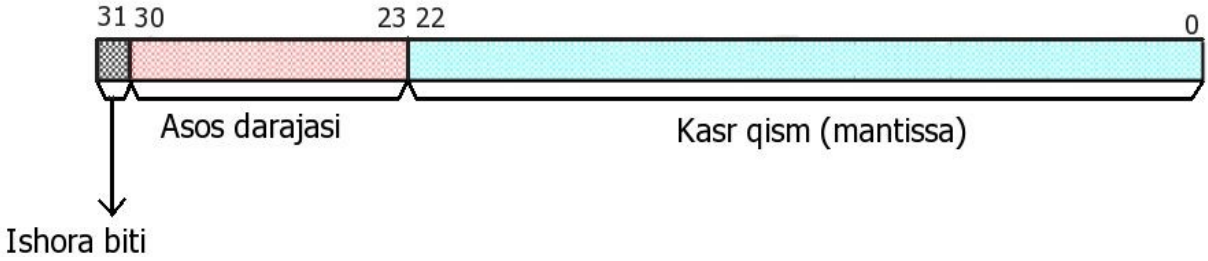
Bu yerda *abcds...* sonning kasr qismi, p sanoq tizimi asos, d esa asos darajasi. 37.512 sonini maromlashtirilgan ko'rinishda yozib ko'ramiz:

$$1.0010110000011 * 10^{101}$$

Ifodadagi barcha sonlar ikkilik sanoq tizimida berilgan.

O'nli kasr son uning kasr qismining aniqlik darajasiga qarab IEEE mezoni bo'yicha 3 xil guruhga bo'linadi. Bular: birlamchi aniqlik, qo'sh aniqlik va kengaytirilgan aniqlik.

Eng kichik aniqlik *birlamchi aniqlik* bo'lib, uning o'lchami 32 bitga teng (rasmga qarang).



53-rasm.

To'ng'ich, ya'ni 31-bit ishorani beradi. Manfiy kasr sonlar xotirada ikkilamchi to'ldiruvchi ko'rinishida saqlanmaydi. Agar ishora biti 1 bo'lsa demak, son manfiy, aks holda musbat. 23 dan 30 gacha bo'lgan bitlarda asos darajasi saqlanadi. U yerda haqiqiy d emas, balki asos darajasi va 127 ning yig'indisi saqlanadi. Buning sababi shundaki, IEEE me'yori bo'yicha xotiraning asos darajasi qismida manfiy bo'lishi mumkin emas. 0 dan 22 gacha bo'lgan bitlar esa *mantissa* deb ataladi va u yerda sonning kasr qismi saqlanadi. Xotirada son doim maromlashtirilgan holda saqlanadi. Mantissada esa maromlashtirilgan sonning nuqtadan keyingi qismi o'rin oladi. Butun qism doim birga teng bo'lgani uchun u hech qayerda saqlanmaydi.

Misol tariqasida 37.512 sonining IEEE bo'yicha birlamchi aniqlikdagi ko'rinishi xotirada qanday ifodalanishini ko'rib chiqamiz:

Ishora biti = 0
Asos darajasi = 1000 0100
Mantissa = 000 0000 0000 0101 1000 0011

Ba'zi hollarda sonning asos darajasi -127 dan kichik bo'ladi. Bu holda son xotirada maromlashtirilmagan holda saqlanadi. Masalan, $10.001 * 10^{-130}$ ni olaylik (bu yerda daraja o'nlik sanoq tizimida). Uning maromlashtirilgan ko'rinishi $1.0001 * 10^{-129}$ kabi bo'ladi. Ko'rinib turibdiki, uning asos darajasi -127 dan kichik. Lekin ushbu son maromlashtirilmagan holda xotirada saqlanishi mumkin, ya'ni $0.010001 * 10^{-127}$. Demak, bu holda IEEE bo'yicha asos darajasi 0 bo'ladi.

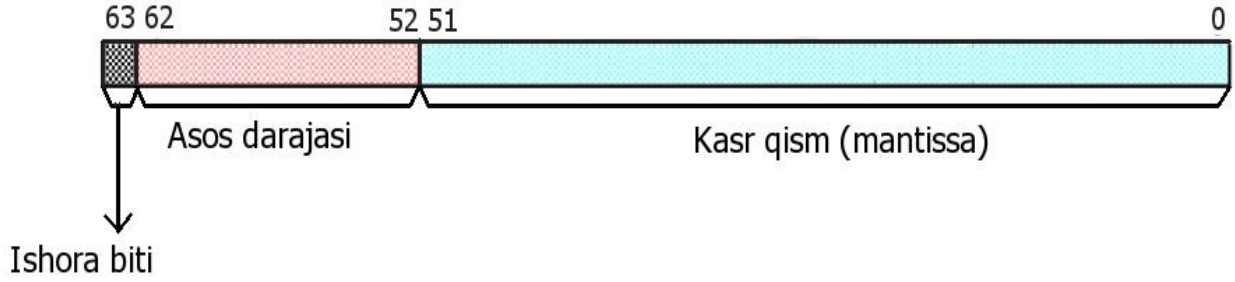
Xotiradagi asos darajasi va mantissaning ba'zi qiymatlarni qabul qilishi maxsus holatlar sifatida qaraladi. 14-jadvalda shunday holatlar qanday nomlanishi berilgan.

14-jadval

$Asos\ darajasi = 0\ va\ mantissa = 0$	0 sonini beradi, ishora bitining ahamiyati yo'q.
$Asos\ darajasi = 0\ va\ mantissa \neq 0$	Maromlashtirilmagan sonni anglatadi.
$Asos\ darajasi = 255\ va\ mantissa = 0$	Cheksizlikni anglatadi.
$Asos\ darajasi = 255\ va\ mantissa \neq 0$	Mavhum sonni anglatadi.

Birlamchi aniqlikdagi sonning aniqlik darajasi $1.0_2 * 2^{-126}$ dan $1.111...111_2 * 2^{127}$ gacha bo'ladi.

Dasturlashda eng ko'p qo'llaniladigan sonning kasr qismi aniqligi *qo'sh aniqlik* bo'lib, bunday sonlarning o'lchami 64 bitga teng bo'ladi. 54-rasmda bitlar joylashuvi ko'rsatilgan.



54-rasm.

Qo'sh aniqlikdagi son hajmi birlamchi aniqliknikidan ikki baravar kattaroqdir. Ular o'rtasidagi asosiy farqlar quyidagilar:

- Asos darajasiga 127 emas, balki 1023 qo'shib olinadi.
- Kasr qismning aniqlik darajasi taxminan $1.0_2 * 2^{-308}$ dan $1.111..._2 * 10^{308}$ gacha bo'lishi mumkin.
- Maromlashtirilmagan sonlar asos darajasi -1023 qilib olinadi.

Kengaytirilgan aniqlikdagi o'nli kasr sonlar 80 bitli bo'ladi va ushbu andozadan asosan SQP ning qiymat registrida foydalaniladi.

11.3. Sonli qo'shma protsessor registrlari

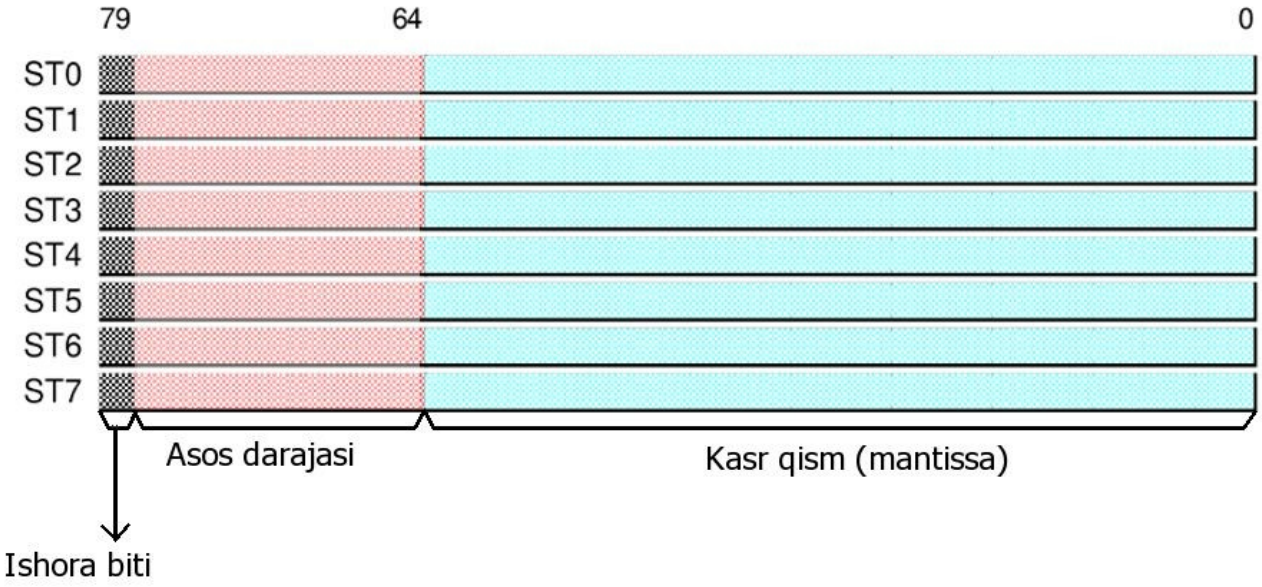
SQPda o'nli kasr sonlarni saqlashga mo'ljallangan 80 bitli 8 ta qiymat registrlarga ega. Ushbu registrlar umumiy foydalanishga mo'ljallangan bo'lib, ulardan dasturchi o'z qiymatlarini saqlashda foydalanishi mumkin. SQP qiymat registrlarini MP dagi dasturchi ixtiyorida bo'lgan 32 bitli (EAX, EBX, ...) registrlarga o'xshatish mumkin.

Qiymat registrlaridan tashqari SQP da yana 5 ta maxsus maqsadlarda ishlatiladigan registrlar bo'lib, ularda asosan tashkiliy ishlarga molik qiymatlar saqlanadi. Ushbu maxsus registrlar ko'pincha SQP va operatsion tizim tomonidan boshqariladi. Holbuki, dasturchi ham ularning qiymatlariga murojaat qilishi mumkin. Bizni asosan qiymat registrlari qiziqtirsada, maxsus registrlarning ham kerakli tomonlari quyida qisqacha ko'rib chiqiladi.

Qiymat registrlari ST0, ST1, ST2, ST3, ST4, ST5, ST6 va ST7 lardan iborat bo'lib, ulardagi qiymatni saqlash andozasi IEEE andozasidan farq qiladi. SQP qiymat registrlari tuzilish bo'yicha ham biroz boshqacharoq tashkillashtirilgan bo'lib, ular stack ko'rinishida ustma-ust qilib joylashtirilgan.

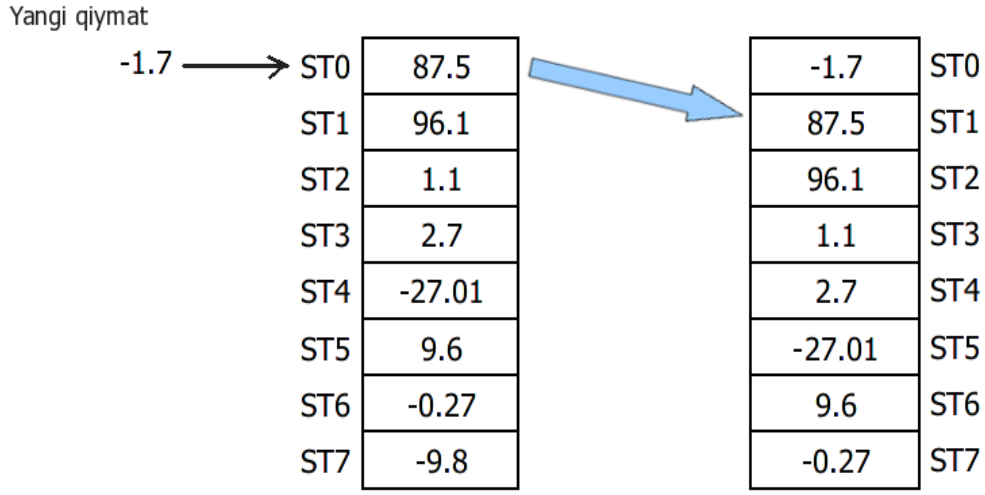
ST0 registri doim stack cho'qqisi, ST7 esa stack tubi hisoblanadi. Registrlar asosida qurilgan ushbu stackning ishlash tarzi «Qismli dasturlash» bobida o'rganilgan MP stacki bilan bir xil.

Faqat SQP stackida bir vaqtning o'zida 8 tadan ortiq qiymat saqlab bo'lmaydi. Eng so'nggi yuklangan qiymat ST0 da bo'ladi. Agar yana qiymat yuklansa, ST0 qiymati ST1 ga o'tadi va yangi qiymat stack cho'qqisidan, ya'ni ST0 dan o'rin oladi.



55-rasm.

SQP stackida registrlarning joylashish tartibi o'zgarmas bo'lib, faqat yuklanayotgan qiymatlargina ularga nisbatdan pastga qarab siljib boradi. Har doim yangi qiymat stackka yuklanganda ST7 dagi oldingi qiymat oshib ketadi (56-rasmga qarang).



56-rasm.

Maxsus registrlardan bizni faqat boshqaruv va holatni belgilash registrlari qiziqtiradi. Boshqaruv registrining qiymati yaxlit holda hech narsani anglatmasada, uning ma'lum bitlarining qiymatlari ma'noga ega. Shu tomondan u MP ning EFLAGS registriga o'xshab ketadi. Boshqaruv registri orqali SQP buyruqlarining ish faoliyati nazorat qilinadi. Gap shundaki, boshqaruv registrining tegishli bitlari o'rnatilgan yoki o'rnatilmaganligiga qarab navbatdagi bajarilayotgan SQP buyrug'i turli xil natijani berishi mumkin.

Boshqaruv registrining 10- va 11-bitlari *yaxlitlash boshqaruvi* deb nomlanadi va ularning qiymatiga qarab SQP ning yaxlitlash buyruqlari turli xil yaxlitlashlarni amalga oshiradi. Ushbu ikki bit hammasi bo'lib to'rt xil qiymat qabul qiladi va bu qiymatlar o'z ma'nosiga ega. 15-jadvalda qiymatlar ma'nosi berilgan.

15-jadval

Yaxlitlash boshqaruvi	Ularning ma'nosi
00	Eng yaqin butun songa qarab yaxlitlash
01	Pastga qarab yaxlitlash
10	Yuqoriga qarab yaxlitlash
11	Butun qismni olish

Boshqaruv registrining boshqa bitlari bizni qiziqtirmaydi va ular ushbu kitobda yoritilmaydi.

Holatni belgilash registriga kelsak, uning bitlari EFLAGS da bo'lgani kabi bayroqlar sifatida qabul qilinadi. Unda asosan o'nli kasr sonlar arifmetikasida yuz berishi mumkin bo'lgan holatlarni aniqlaydigan bayroqlar o'rin olgan. Masalan, natija maromlashtirilmagan son bo'lganida, holat registrining birinchi biti o'rnatiladi. Ushbu bit *maromlashtirilmaganlik biti* deb nomlanadi. Bundan tashqari holatni belgilash registrida ZF, CF, OF ga o'xshagan bayroqlar mavjud.

Ushbu kitobda SQPning maxsus registrlari haqida batafsil to'xtalmaymiz. Agar sizni aynan o'sha registrlar qiziqtirsa tegishli manbalarga murojaat qilishingiz mumkin.

11.4. Sonli qo'shma protsessor asosiy buyruqlari

MP buyruqlaridan farqlanishi uchun barcha SQP buyruqlari `F` harfi bilan boshlanadi. Buyruqlarga qiymat sifatida birlamchi, qo'sh yoki kengaytirilgan aniqlikdagi sonlar beriladi. Agar xususiy holatlar bo'lsa, alohida ogohlantiriladi.

Kasr sonli o'zgarmas va o'zgaruvchilar haqida «O'zgarmaslar, ifodalar va direktivalar» hamda «O'zgaruvchilarni e'lon qilish» mavzularida yetarlicha ma'lumot berilgan. Shunday bo'lsada quyidagi o'zgaruvchilarni e'lon qilinishini ko'rib chiqamiz:

```
birlamchi_aniqlikdagi_son          dd   1.123456789
q_osh_aniqlikdagi_son              dq   2.123456789
kengaytirilgan_aniqlikdagi_son     dt   3.123456789
kasr dq   -15.999807
```

11.4.1. Qiymatni yuklash va o'qish buyruqlari

Ushbu mavzuda SQP registrlariga qiymatlarni yuklash va u yerdan o'qishni amalga oshirish buyruqlari o'rganiladi.

`FLD` buyrug'i o'zgaruvchidagi qiymatni o'nli kasr son sifatida stack cho'qqisiga yuklaydi.

Qoidasi:

```
fld  manba
```

Bu yerda `manba` o'rnida xotiradagi o'zgaruvchi yoki SQP registri bo'lishi mumkin.

Misollar:

```
fld  qword[kasr]          ;; ST0 ← -15.999807
fld  ST2                  ;; ST0 ← ST2, ST1 ← -15.999807
fld  eax                  ;; Xato
fld  7.8                  ;; Xato
fld  qword 7.8            ;; Xato
fld  __float32__(7.8)     ;; Xato
```

Shunga o'xshash yana FILD buyrug'i bo'lib, u berilgan butun qiymatni stack cho'qqisiga yuklaydi.

Qoidasi:

```
fild    manba
```

manba o'rnida 2, 4 yoki 8 baytli xotiradagi o'zgaruvchi kelishi mumkin. Butun qiymat avval o'nli kasr songa o'giriladi, so'ngra yuklanadi.

Dasturlashda ayni muddao bo'lgan FLDZ va FLD1 buyruqlari bo'lib, ular mos ravishda 0 va 1 ni stack cho'qqisiga yuklaydi. Ushbu buyruqlar hech qanday qiymat qabul qilmaydi.

Stack cho'qqisidagi qiymatni FST buyrug'i orqali o'qishimiz mumkin.

Qoidasi:

```
fst     maqsad
```

maqsad o'rnida 4 yoki 8 baytli xotira yoki SQP registri kelishi mumkin. Masalan:

```
fld   qword[kasr]        ;; ST0 ← -15.999807
fldz                     ;; ST0 ← 0, ST1 ← -15.999807
fst   qword[kasr]        ;; kasr ← 0
```

Agar qiymatni o'qish barobarida uni stack cho'qqisidan o'chirish ham kerak bo'lsa, FSTP buyrug'idan foydalaniladi. Ushbu buyruq vazifasi jihatdan MP ning POP buyrug'iga o'xshaydi. FST ga qo'shimcha ravishda FIST buyrug'i ham mavjud bo'lib, u qiymatni birinchi butun songa yaxlitlab, so'ngra maqsadga yuklaydi. Maqsad bo'lib 2 yoki 4 baytli xotira kelishi mumkin.

FSTP mavjud bo'lgani kabi FISTP buyrug'i ham bor. FISTP yaxlitlangan butun sonni maqsad ga yuklagach, uni stackdan ham o'chiriladi.

Aynan qanday yaxlitlash usuli qo'llanishi yaxlitlash boshqaruvi bitlarining qiymatiga bog'liq. Ular doim «00» ga teng bo'ladi. Ammo maxsus buyruqlar orqali boshqaruv registrining qiymatini o'zgartirish mumkin. FSTCW buyrug'i boshqaruv registrining qiymatini xotiradagi o'zgaruvchiga yuklab beradi. FLDCW buyrug'i esa xotiradagi o'zgaruvchi qiymatini boshqaruv registriga o'zlashtiradi.

Qoidasi:

```
fstcw     maqsad
fldcw     manba
```

Bu yerda maqsad ham, manba ham ikki baytli o'zgaruvchi bo'lishi kerak.

Ba'zida ma'lum registrlar qiymatlarini o'zaro almashtirish kerak bo'lib qolishi mumkin. FXCH buyrig'i ham aynan shu vazifani bajaradi. U o'ziga berilgan istalgan registr qiymatini ST0 bilan almashtiradi.

Qoidasi:

```
fxch  stn
```

Masalan, kasr o'zgaruvchisiga ST1 ning quymatini yuklash kerak bo'lib qoldi deylik:

```
fxch  st1                ;; ST0 ↔ ST1
fstp  qword[kasr]        ;; kasr ← ST0
```

11.4.2. Qo'shish va ayirish buyruqlari

Qo'shishni amalga oshirish uchun asosan `FADD`, `FADDP` va `FIADD` buyruqlari ishlatiladi.

Qoidasi:

```
fadd    manba              ;; ST0 ← ST0 + manba
fadd    maqsad , st0       ;; maqsad ← maqsad + ST0
faddp   maqsad , st0       ;; FADD kabi,ammo qo'shishdan so'ng stack
                           ;; cho'qqisi bo'shatiladi.
fiadd   manba              ;; FADD kabi, ammo manba butun qiymat
                           ;; sifatida qabul qilinadi.
```

Bu yerda `maqsad` o'rnida SQP registridan birortasi bo'lishi kerak. `FADD` va `FIADD` ga beriladigan manba mos ravishda `FLD` va `FILD` larga beriladigan manbalar kabi bo'ladi.

Ayirish asosan `FSUB` buyrug'i orqali bajariladi.

Qoidasi:

```
fsub    manba              ;; ST0 ← ST0 – manba
```

Bu yerda `manba` ayriluvchi bo'lib xizmat qiladi, kamayuvchi esa har doimgidek stack cho'qqisi hisoblanadi, ayirma ham stack cho'qqisiga yuklanadi.

Ayirishda hadlarning o'rni ahamiyatga ega bo'lgani sababli deyarli barcha ayirish buyruqlarining teskarisi mavjud. Teskari buyruqlar odatda `R` harfi bilan tugaydi. Demak, agar `FSUB` buyrug'idagi manba kamayuvchi sifatida qubul qilinishi kerak bo'lsa, `FSUBR` buyrug'idan foydalaniladi. Ikkala holda ham manba o'rnida yoki SQP registri, yoki 4, 8 baytli xotira kelishi mumkin.

Agar ayirmani stack cho'qqisiga emas, balki boshqa bir o'zgaruvchiga yuklash kerak bo'lsa, buni ushbu buyruqlarga 2 ta qiymat berish oqrali amalga oshirish mumkin. Ikkinchi qiymat doim ST0 bo'ladi.

Qoidasi:

```
fsub    maqsad , st0       ;; maqsad ← maqsad – ST0
fsubr   maqsad , st0       ;; maqsad ← ST0 – maqsad
```

`maqsad` bo'lib faqat SQP registri kelishi mumkin. Masalan:

```
fsub    st6 , st0          ;; ST6 ← ST6 – ST0
```

`FSUB` va `FSUBR` buyruqlariga qo'shimcha ravishda `FSUBP` va `FSUBRP` buyruqlari bo'lib, ular ayirishdan so'ng stack cho'qqisini ham tozalashadi.

Agar kamayuvchi yoki ayriluvchi xotiradagi butun qiymat bo'lsa, mos ravishda `FISUB` va `FISUBR` buyruqlari orqali ayirish amalga oshiriladi.

Qoidasi:

```
fisub    manba             ;; ST0 ← ST0 – manba
fisubr   manba             ;; ST0 ← manba – ST0
```

Bu yerda `manba` 2 yoki 4 baytli xotiradagi o'zgaruvchi bo'lishi mumkin.

Eslatma: Butun qiymatlar bilan ishlaydigan SQP buyruqlari berilgan manba yoki maqsadga ikkilamchi to'ldiruvchi andozasi bo'yicha qaraydi, IEEE andozasi bo'yicha emas.

11.4.3. Ko'paytirish va bo'lish buyruqlari

O'nli kasr sonlar ustida ko'paytirish amallari `FMUL`, `FMULP` va `FIMUL` buyruqlari yordamida bajariladi. `FMUL` o'ziga berilgan qiymatni stack cho'qqisidagi qiymatga ko'paytirib ko'paytmani yana stack cho'qqisiga yuklaydi.

Qoidasi:

```
fmul    manba               ;; ST0 ← ST0 * manba
```

Agar `FMUL` ga ikkita qiymat berilsa, ko'paytma stack cho'qqisiga emas, balki berilgan birinchi o'zgaruvchiga yuklanadi.

Qoidasi:

```
fmul    maqsad , st0        ;; maqsad ← maqsad * ST0
```

Ikkinchi qiymat faqat ST0 bo'ladi.

`FMULP` ham xuddi `FMUL` ga o'xshaydi, ammo ko'paytirishdan so'ng stack cho'qqisini tozalanadi. Agar ko'paytiruvchilardan biri butun qiymat ko'rinishda bo'lsa, `FIMUL` buyrug'i qo'llanadi.

Qoidasi:

```
fimul    manba              ;; ST0 ← ST0 * manba
```

Bo'lish buyruqlari bir oz ko'proq, chunki bo'lishda bo'linuvchi va bo'luvchining o'rinlari ahamiyatga ega. Bo'lish asosan `FDIV` buyrug'i orqali amalga oshiriladi.

Qoidasi:

```
fdiv    manba               ;; ST0 ← ST0 / manba
```

`manba` 4 yoki 8 baytli xotiradagi o'zgaruvchi yoki SQP registri bo'lishi mumkin. `manba` ni ST0 ga bo'lish uchun `FDIVR` buyrug'i mavjud.

Qoidasi:

```
fdivr    manba              ;; ST0 ← manba / ST0
```

`FDIV` va `FDIVR` buyruqlariga ikkita qiymat berish orqali bo'linmani ST0 ga emas, balki istalgan boshqa o'zgaruvchiga yuklash mumkin.

Qoidasi:

```
fdiv    maqsad , st0        ;; maqsad ← maqsad / ST0
fdivr   maqsad , st0        ;; maqsad ← ST0 / maqsad
```

Albatta bo'linma boshqa o'zgaruvchiga yuklangach, ST0 ning qiymati stackda kerak bo'lmay qolishi mumkin. Shunday holatlar uchun mos ravishda `FDIVP` va `FDIVRP` buyruqlari mavjud bo'lib, ular bo'lishdan so'ng stack cho'qqisini tozalaydi.

Bo'linuvchi yoki bo'luvchi butun qiymat bo'lsa, `FIDIV` va `FIDIVR` buyruqlari ish beradi.

Qoidasi:

```
fidiv   manba            ;; ST0 ← ST0 / manba
fidivr  manba , st0      ;; ST0 ← manba / ST0
```

Bu yerda `manba` 2 yoki 4 baytli xotiradagi butun qiymatga ega o'zgaruvchi bo'lishi kerak.

Misol tariqasida a, b, c berilganda $d = b^2 - 4*a*c$ ni hisoblovchi dasturni ko'rib chiqmiz.

19-namuna: Diskriminantni hisoblaymiz.

```
(1)   %include "nasm-io.inc"
(2)
(3)   section .bss
(4)   a  resq 1          ;; Qo'sh aniqlikdagi sonlarni
(5)   b  resq 1          ;; saqlash uchun o'zgaruvchilar.
(6)   c  resq 1
(7)   d  resq 1
(8)
(9)   section .data
(10)  koef dw  4
(11)
(12)  section text
(13)  tizim_global main
(14)
(15)  main:
(16)          chop_et   `a, b va c ni kiriting: `
(17)          qabul_qil `%lf %lf %lf`, a, b, c
(18)
(19)          fild  word[koef]    ;; ST0 ← 4
(20)          fmul  qword[a]      ;; ST0 ← 4*a
(21)          fmul  qword[c]      ;; ST0 ← 4*a*c
(22)
(23)          fld   qword[b]      ;; ST0 ← b , ST1 ← ST0
(24)          fmul  qword[b]      ;; ST0 ← b * b
(25)
(26)          fsub  st1           ;; ST0 ← b*b - 4*a*c
(27)          fstp  qword[d]      ;; d ← ST0, ST0 ← ST1
(28)
(29)          chop_et   `b*b - 4*a*c = %g \n`, [d], [d+4]
(30)
(31)  ret
```

27-qatorda `FSTP` o'rnida `FST` ni qo'llasa ham bo'lar edi, ammo stackni ishlatib bo'lgach, uni tozalab qo'ygan maqul. MPdagi stackka qiymatlar 4 baytdan yuklanishi sababli, qo'sh aniqlikdagi o'zgaruvchilar kenja 4 bayt va to'ng'ich 4 baytga ajratilgan holda, ikki martada `chop_et` makrosiga beriladi. `qabul_qil` makrosi o'zgaruvchi manzillari bilan ishlagani tufayli, unga faqat nishon nomlari beriladi xolos. Nishonlar esa 4 baytli manzillarni taqdim etadi.

11.4.4. Taqqoslash buyruqlari

Oldingi boblarda butun sonlarni taqqoslashga mo'ljallangan `CMP` buyrug'i ko'rib chiqilgan edi. Ammo ushbu buyruq o'nli kasr sonlarini o'zaro taqqoslashga yaramaydi. Sababi o'zingizga ma'lum, ya'ni butun va kasr sonlar xotirada har xil andozalarda saqlanadi.

Kasr sonlarda taqqoslash xuddi qo'shish, ayirish, ko'paytirish va bo'lishda bo'lgani kabi bir emas, balki bir necha buyruqlar orqali amalga oshiriladi. Taqqoslash natijalari ham EFLAGS registriga emas, balki SQP ning holat registriga o'rnatiladi. Taqqoslash asosan `FCOM` buyrug'i orqali bajariladi.

Qoidasi:

```
fcom  manba
```

Ushbu buyruq ST0 va `manba` ni taqqoslaydi. `manba` SQP registri yoki 4 yoki 8 baytli xotira bo'lishi mumkin. Taqqoslashdan so'ng ST0 ning qiymatini stackdan o'chirish kerak bo'lsa, uning uchun `FCOMP` buyrug'i mavjud. Yana bir juda qulay bo'lgan buyruq `FCOMPP` bo'lib, u hech qanday qiymat qabul qilmaydi va ST0 bilan ST1 ni taqqoslaydi. Ushbu buyruq taqqoslashdan so'ng stack cho'qqisini ikki marta tozalaydi.

O'nli kasr sonni butun son bilan ham taqqoslash imkoni bo'lib, bunda butun son avval o'nli kasr songa o'giriladi va so'ngra taqqoslash amalga oshiriladi. `FICOM` va `FICOMP` buyruqlari aynan shu ishni amalga oshiradi.

Qoidasi:

```
ficom   manba   ;; ST0 va manba taqqoslanadi.
ficomp  manba   ;; Taqqoslashdan so'ng stack cho'qqisi tozalanadi
```

Taqqoslash va uning natijasini tahlil qilish o'nli kasr sonlarida xuddi butun sonlarda bo'lgani kabi amalga oshiriladi, ya'ni taqqoslashdan so'ng kerakli bayroqlar o'rnatiladi. Ammo yuqorida ta'kidlanganidek, bayroqlar odatdagidek EFLAGS registriga emas, balki SQP ning holat registriga o'rnatiladi. Shartli o'tishlarni amalga oshiradigan buyruqlar oilasiga mansub barcha buyruqlar esa EFLAGS registri bilan ishlaydi. Masalaning yechimi sifatida Pentium Pro protsessoridan oldin bo'lgan protsessorlarda ikkita qo'shimcha buyruq ishlab chiqilgan. Biri SQP holat registrining kenja ikki baytini berilgan o'zgaruvchiga yuklab beradi, ikkinchisi esa berilgan qiymatni EFLAGS registriga yuklaydi. Bu ikki buyruq orqali holat registrining bayroqlari EFLAGS ga olib o'tilishi mumkin. Bu buyruqlar `FSTSW` va `SAHF` deb nomlanadi.

Qoidasi:

```
fstsw     maqsad
```

Ushbu buyruq SQP holat registrining kenja ikki baytini `maqsad` ga yuklaydi. `maqsad` o'rnida 2 baytli xotiradagi o'zgaruvchi yoki AX registri bo'lishi mumkin.

```
sahf
```

Ushbu buyruq AH ni EFLAGS ga yuklaydi. Agar EFLAGS dagi kenja baytni AH ga yuklash kerak bo'lsa, bu uchun mo'ljallangan LAHF buyrug'i ham mavjud.

Qoidasi:

```
lahf
```

Misol tariqasida ikki o'nli kasr sonning ichidan kattasini topadigan dasturni ko'rib chiqamiz. Sonlar 4 baytli `a` va `b` o'zgaruvchilarida saqlanayotgan bo'lsin deylik.

```
        fld  dword[a]
        fcomp     dword[b]
        fstsw     ax
        sahf
        ja   a_katta
        chop_et   `b katta`
        jmp  davom
a_katta:
        chop_et   `a katta`
davom:
        …
```

Pentium Pro va undan keyin chiqqan protsessorlarda o'nli kasrlarni taqqoslash uchun yana ikkita buyruq `FCOMI` va `FCOMIP` qo'shildi. Ularning asosiy farqi shundaki, taqqoslash bajarilgach kerakli bayroqlar SQP ning holat registriga emas, balki to'g'ridan-to'g'ri EFLAGS ga o'rnatiladi. Boshqacha qilib aytganda, dasturchi `FSTSW` va `SAHF` buyruqlari haqida o'ylamasa ham bo'ladi.

Qoidasi:

```
fcomi     manba
```

ST0 va `manba` taqqoslanadi, `manba` o'rnida SQP registridan birortasi bo'lishi mumkin. `FCOMIP` buyrug'i `FCOMI` dan farqli o'laroq taqqoslashdan so'ng stack cho'qqisini ham tozalaydi.

Yuqoridagi misolni ushbu buyruqlar yordamida ko'rib chiqamiz.

```
        fld  dword[b]       ;; ST0 ← b
        fld  dword[a]       ;; ST0 ← a, ST1 ← b
        fcomip    st1       ;; a ? b
        ja   a_katta
        chop_et   `b katta`
        jmp  davom
a_katta:
        chop_et   `a katta`
davom:
        …
```

11.5. Qo'shimcha buyruqlar

Yuqorida ko'rib chiqilgan buyruqlardan tashqari SQP da ko'plab o'nli kasr sonlar bilan ishlashga mo'ljallangan buyruqlar mavjud. Ushbu mavzuda biz ularning eng muhimlari bilan tanishib chiqamiz.

Yuqoridagi boblarda qiymat ishorasini qarama-qarshisiga o'zgartiradigan `NEG` buyrug'i bilan tanishgan edik. Bu ba'zida juda qo'l keladigan buyruq bo'lib, uning SQP dagi o'xshashi mavjud. `FCHS` (**Ch**ange **S**ign - Ishorani O'zgartir) aynan shu vazifani faqat o'nli kasr sonlar uchun bajaradi. Ushbu buyruq hech qanday qiymat olmaydi va faqat ST0 dagi qiymatga ta'sir qiladi.

Qoidasi:

```
fchs            ;; ST0 ← -1 * ST0
```

Asosan matematik hisob-kitoblar kasr sonlar bilan bo'lishini hisobga olgan holda SQP ning ko'pgina buyruqlari matematik amallarni bajarishga mo'ljallangan. Shunday buyruqlarning yana biri bo'lmish `FABS` sonning modulini[1] hisoblaydi. U ham `FCHS` ga o'xshab ST0 bilan ishlaydi.

Qoidasi:

```
fabs            ;; ST0 ← |ST0|
```

`FSIN` va `FCOS` buyruqlari mos ravishda sonning sinusi va kosinusini hisoblaydi. Ular ham oldingi buyruqlar kabi ST0 ni manba va maqsad sifatida qabul qiladi.

Qoidasi:

```
fsin            ;; ST0 ← sin(ST0)
fcos            ;; ST0 ← cos(ST0)
```

Eslatma: `FCOS` va `FSIN` burchakni, ya'ni ST0 ning qiymatini gradus emas, balki radian sifatida qabul qiladi. 1 radian $\frac{180^0}{\pi}$ ga teng ekanligini eslatamiz.

Misol tariqasida 90^0 ning sinusi va kosinusini hisoblaymiz.

```
rad  dd   1.5707          ;;  1.5707 rad ≈ 90^0
…
fld  dword[rad]           ;; ST0 ← rad
fsin                      ;; ST0 ← sin(1.5707 rad), ya'ni sin(90)
fld  dword[rad]           ;; ST0 ← rad, ST1 ← ST0
fcos                      ;; ST0 ← cos( 1.5707 rad), ya'ni cos(90)
```

Albatta matematikani kvadrat ildizsiz tasavvur qilish qiyin va shu vajdan SQP da bunday ishni amalga oshiradigan `FSQRT` (**Sq**uare **R**oo**t** - Kvadrat ildiz) buyrug'i mavjud.

Qoidasi:

```
fsqrt                     ;;  ST0 ← √ST0
```

Ma'lumki, qiymat bitlarini o'ngga yoki chapga siljitish orqali, qiymatni tez yo'sinda 2 ga karrali bo'lgan songa bo'lish yoki ko'paytirish mumkin edi. O'nli kasr sonlarda ham buning iloji bor. `FSCALE` buyrug'i ST0 dagi qiymatga ikki darajasi ST1 ni ko'paytiradi. ST1 dagi qiymat yaxlitlab olinadi.

Qoidasi:

```
fscale                    ;;  ST0 ← ST0 * 2^[ST1]
```

Misol tariqasida 7.1*8 hisoblaymiz:

```
daraja    dd   3
son  dd   7.1
…
```

[1] Sonning moduli deb uning mutlaq qiymatiga aytiladi va $|a|$ bilan belgilandi. Masalan, -7 ning ham, 7 ning ham moduli 7 ga teng.

```
fild dword[daraja]        ;; ST0 ← 3
fld  dword[son]           ;; ST0 ← 7.1, ST1 ← 3
fscale                    ;;  ST0←7.1∗2^3
```

11.6. Amaliy dastur: Kvadrat ildiz

Qismli dasturlash bobida butun sonning kvadrat ildizini hisoblovchi dastur ko'rib chiqilgan edi. Ammo o'shanda assemblerda o'nli kasrlar bilan ishlashni bilmaganimiz tufayli faqatgina ildizning yaxlit qismini hisoblay olgan edik. Endi esa bemalol o'shanda qo'llangan usul orqali sonning kvadrat ildizini aniq hisoblaydigan qism dasturi tuzishimiz mumkin. Garchi SQP da sonning kvadrat ildizini to'g'ridan-to'g'ri hisoblovchi buyruq bo'lsada, misol tariqasida biz o'zimizning ildiz qism dasturimizni yozishga harakat qilib ko'ramiz. Qo'llaniladigan usul o'sha-o'sha, agar unutgan bo'lsangiz, «Qismli dasturlash» bobiga qarab olishingiz mumkin. Faqat o'nli kasr sonlar bilan ishlaganda $\{x_n\}$ ketma-ketlikning x_n va x_{n+1} a'zolari ortasidagi farq doim nolga intilib boraveradi, ammo u amaliyotda kam hollarda nolga teng bo'lib qoladi.

$$\lim_{n\to\infty} |x_n - x_{n+1}| = 0 \qquad (*)$$

(*) ifodada ushbu hol limit yordamida ko'rsatilgan. Bu yerda n qancha katta son bo'lsa, natija ham shuncha aniq bo'ladi. Dasturlashda esa nazariyadan farqli o'laroq juda katta aniqlikka erishish shart emas, ya'ni n ko'pi bilan 10-15 gacha borsa yetarli hisoblanadi. Buning uchun ketma-ketlikning a'zolari farqi nol bilan emas, balki biror kichikroq bo'lgan epsilon (ε) soni bilan taqqoslanadi.

$$|x_n - x_{n+1}| < \varepsilon$$

ε qancha kichik son bo'lsa, sonning kvadrat ildizi ham shunchalik aniqroq hisoblanadi. Masalan, ε=0.0001 bo'lganda, $\sqrt{26.02}=5.10098$ bo'ladi va ε=0.000001 bo'lganda esa $\sqrt{26.02}=5.1009802$ bo'ladi.

20-namuna: Kasr ildiz.

```
(1)    %include "nasm-io.inc"
(2)
(3)    section .bss
(4)    a  resq 1
(5)    b  resq 1
(6)
(7)    section .text
(8)    tizim_global main
(9)
(10)   main:
(11)       chop_et   `a ni kiriting: `
(12)       qabul_qil `%lf`, a
(13)
(14)       push dword[a + 4]
(15)       push dword[a]
(16)       call kasr_ildiz
(17)       add  esp , 8
(18)
(19)       fstp qword[b]
(20)       chop_et   `kasr_ildiz(a) = %g \n`, [b], [b + 4]
(21)
```

```
(22)   ret
(23)
(24)   ;; Kasr ildiz qism dasturi.
(25)   ;;
(26)   ;; Maqsad:
(27)   ;;  Sonning kvadrat ildizi qo'sh aniqlikdagi kasr son
(28)   ;;  sifatida hisoblanadi.
(29)   ;;  C dagi e'lon qilinishi:
(30)   ;;       float kasr_ildiz(float a);
(31)   ;;
(32)   ;; Qiymatlar:
(33)   ;;  a - kvadrat ildizi hisoblanadigan qiymat.
(34)   ;;
(35)   ;; Natija:
(36)   ;;  kvadrat ildiz ST0 registrida qaytariladi.
(37)
(38)   ;; Jo'natilgan qiymat
(39)   %define  a     [ebp + 8]
(40)
(41)   ;; Mahalliy o'zgaruvchilar
(42)   %define  ikki word[ebp - 8]
(43)   %define  eps  dword[ebp - 4]
(44)
(45)   kasr_ildiz:
(46)       enter     8 , 0
(47)       mov  eps , __float32__(0.000001)
(48)       fld  eps                 ;; ST0 <- eps
(49)       fld  qword a
(50)       mov  ikki , 2            ;; x1 = a/2
(51)       fidiv     ikki           ;; ST0 <- xn
(52)       fld  qword a             ;; ST0 <- a , ST1 <- xn
(53)
(54)   .takrorlanish:
(55)       fdiv st1                 ;; ST0 <- a/xn
(56)       fadd st1                 ;; ST0 <- xn + a/xn
(57)       fidiv     ikki           ;; ST0 <- (xn + a/xn) / 2
(58)                                ;; Demak, ST0 = xn+1, ST1 = xn
(59)       fxch st1                 ;; ST0 <- xn, ST1 <- xn+1
(60)       fsub st1                 ;; ST0 <- xn - xn+1
(61)       fabs                     ;; ST0 <- |xn - xn+1|,
(62)                                ;; oradagi farqning moduli.
(63)       fcomip    st2            ;; |xn - xn+1| < eps ?
(64)       jbe  .topildi
(65)
(66)       fld  qword a             ;; ST0 <- a, ST1 <- xn+1
(67)       jmp  .takrorlanish
(68)
(69)   .topildi:
(70)       fxch st1                 ;; ST0 <- eps, ST1 <- √a
(71)       fstp qword[ebp - 8]      ;; ST0 <- √a
(72)       leave
(73)   ret
```

Qism dasturini chaqirish mezoni bo'yicha agar natijaviy qiymat kasr son bo'lsa, u ST0 registriga chaqiruvchi dasturga qaytariladi. `kasr_ildiz` qism dasturida ham shunga rioya qilingan. Qism dasturining 70- va 71-qatorlari aslini olganda ortiqcha bo'lib, ular faqat dastur so'ngida SQP stackini tozalashga qaratilgan.

XII bob
Fayllar bilan ishlash

Dasturlashda ko'pincha fayllar bilan ishlashga to'g'ri keladi. Ishlash deganda, faylni yaratish yoki uni o'chirish, faylga biror axborotni yozish yoki u yerdan o'qib olish nazarda tutiladi. Ko'pgina dasturlash tillari ular bilan ishlashga mo'ljallangan yuqori darajali qism dasturlaridan tashkil topgan kutubxonalarga ega. Assemblerda esa odatdagidek barchasi oddiy usulda amalga oshiriladi.

Ushbu bobda biz assembler tilida fayllar bilan ishlash qanday amalga oshirilishini ko'rib chiqamiz. Ammo bundan oldin operatsion tizimda fayllar qanday boshqarilishi bilan tanishish ma'qulroq. Binobarin quyidagi mavzu shunga bag'ishlangan.

12.1. Operatsion tizimda fayllar tushunchasi

Kompyuterning qattiq diskida baytlar ketma-ketligi ko'rinishida saqlanadigan obyekt *fayl* deb yuritiladi. Fayllar axborot saqlashda qo'llaniladi. Turli operatsion tizimlarda fayllar ham turlicha g'oyalar asosida tuzilgan. Shunday bo'lsada quyida fayllar tushunchasi maxsus bir operatsion tizim uchun emas, balki umumiy mezonlarga ko'ra yoritiladi. Shunday qilib, operatsion tizim nuqtai nazarida fayllar asosan quyidagi turlarga bo'linadi:

- Oddiy fayl
- Direktoriya ko'rinishidagi fayl
- Ramziy yorliq fayli
- O'quv/yozuv moslamalarini taqdim etuvchi fayl
- Aloqa fayli

Oddiy fayllar ingliz tilida *regular files* deb yuritiladi va ularda matn, rasm, musiqa yoki ikkilik sanoq tizimidagi dastur kodi kabi istalgan ko'rinishdagi axborotni saqlash mumkin. Oddiy turdagi fayllar asosiy hisoblanib, ushbu bobda aynan shu fayllar bilan ish ko'riladi.

Direktoriyalar ham faylning bir turi bo'lib, ularda fayllar nomining ro'yxati turadi (o'sha direktoriyada saqlanayotgan fayllarni ro'yxati albatta).

Ramziy yorliq fayllari ancha keng foydalaniladigan fayl turi bo'lib, ular ingliz tilida *symbolic links* deb yuritiladi. Bu kabi fayllarda hech qanday katta axborot saqlanmaydi va ular boshqa faylga ko'rsatgich sifatida qo'llanadi. Demak, ramziy yorliq faylida o'zi ko'rsatgich bo'lib turgan haqiqiy faylning mutlaq yo'li saqlanadi. Masalan, *A* direktoriyadagi *hujjatlar.doc* fayli bilan *B* direktoriyada turib ham ishlash kerak bo'lsa, *B* da *hujjatlar.doc* fayliga ramziy yorliq yaratiladi.

O'quv/yozuv moslamalari fayllari operatsion tizim tomonidan foydalanuvchi dasturlariga tashqi moslamalar billan axborot almashish uchun taqdim etiladi. Masalan, printer orqali biror axborotni chop etish uchun tizimdagi *printer* fayliga yozish amalga oshiriladi.

Aloqa fayllari ingliz tilida *sockets and pipes*[1] deb yuritiladi va ulardan protsessorda bajarilayotgan jarayonlar o'zaro axborot almashish maqsadida foydalanadi.

[1] Pipe fayllar FIFO(**F**irst **I**n **F**irst **O**ut – Birinchi kirgan – Birinchi chiqadi) ham deb yuritiladi.

Operatsion tizimda fayllar haqida to'liq ma'lumot beradigan maxsus buyruqlar bo'ladi. Ammo tizim turiga qarab bunday buyruqlar har xil bo'lishi mumkin. Masalan, Linuxda buyruqlar qatorida `ls` yoki `dir` buyrug'ini `-l` kaliti bilan berish kifoyadir:

```
faz@fnok /dev $ ls -l
-rw-r--r-- 1 root root        489 May  6 13:51 daemon.log
drwxr-xr-x 6 root root        120 Jun  5 11:23 disk
crw-rw---- 1 root dialout 4,   64 Jun  5 11:23 modem
srw-rw---- 1 root lp            0 Jun  5 11:23 printer
brw-rw---- 1 root disk    8,    0 Jun  5 11:23 sda
lrwxrwxrwx 1 root root         15 Jun  5 11:23 stderr->/proc/self/fd/2
lrwxrwxrwx 1 root root         15 Jun  5 11:23 stdin->/proc/self/fd/0
lrwxrwxrwx 1 root root         15 Jun  5 11:23 stdout->/proc/self/fd/1
prw-r----- 1 root adm           0 Jun  5 11:24 xconsole
```

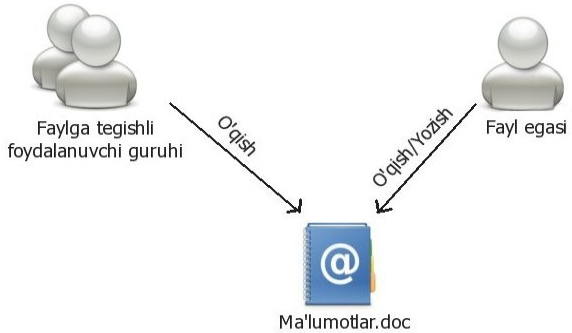
Natijaning birinchi ustunining birinchi belgisi fayl turini beradi. Qaysi belgi nima ma'noni anglatishi 16-jadvalda berilgan.

16-jadval

Belgi	Ma'nosi
-	Oddiy fayl
d	Direktoriya
b/c	O'quv/yozuv moslamalari
l	Ramziy yorliq
p/s	Aloqa fayli

Eslatma: Agar siz Windowsda ishlayotgan bo'lsangiz u yerda `dir` buyrug'i `/A` kaliti bilan ishlatiladi. To'liq ma'lumot uchun `help dir` buyrug'iga murojaat qiling.

Fayllar bilan ishlaganda yana bir e'tiborga molik xususiyat ularga beriladigan ruxsatlardir. Har bir fayl tizimdagi ma'lum bir foydalanuvchiga tegishli bo'lib, o'sha foydalanuvchi *fayl egasi* hisoblanadi. Asosan ko'pgina operatsion tizimlarda faylni kim yaratsa, o'sha fayl egasi bo'lib qoladi. Fayl egasining faylga nisbatan bir necha xil huquqlari bo'ladi. Faylga bo'lgan huquqlar asosan uch xil bo'ladi: *o'qish*, *yozish*, *bajarish*. Masalan, agar fayl egasining faqat fayldan o'qish huquqi bo'lsa, o'sha faylga u hech qanday o'zgartirish kirita olmaydi. Faylga biror bir o'zgartirish kiritish kerak bo'lsa, albatta, yozish huquqiga ega bo'lish kerak. Bajarish huquqiga kelsak, u bajaruvchi dastur kodi joylashgan fayllarga qo'yiladi. Masalan, *firefox.exe, explorer.exe, word.exe* kabi fayllarda bajarish huquqi bo'lishi shart.



57-rasm.

Tizimda foydalanuvchidan tashqari yana foydalanuvchi guruhlari mavjuddir. Foydalanuvchi guruhi unga a'zo bo'lgan foydalanuvchilardan tashkil topgan bo'lib, ushbu foydalanuvchilar o'zaro fayllar bilan almashinishlari mumkin. Masalan, fayl egasining faylga yozish va o'qish huquqlari bo'lsa, guruhdagi qolgan a'zolarda faqat o'qish huquqi bo'lishi mumkin (57-rasmga qarang). Guruhdagi a'zolarga qanday huquqlar berilishini fayl egasi hal qiladi.

Umuman olganda fayl foydalanuvchilari uch xil toifaga bo'linadi:

- Fayl egasi
- Faylga tegishli bo'lgan foydalanuvchi guruhlari
- Qolgan hamma

Ushbu toifa foydalanuvchilarning har biri ham yuqorida sanab o'tilgan uch xil huquqqa ega bo'lishi mumkin.

Yuqorida berilgan `ls` buyrug'ining natijasida fayl foydalanuvchilarining huquqlari yaqqol ko'rsatilgan. Natijadagi birinchi ustunining birinchi belgidan keyingi bo'lgan 9 ta belgi huquqlarni ifodalaydi. Birinchi uchlik fayl egasining huquqlarini bersa, keyingi uchliklar mos ravishda ikkinchi va uchunchi toifa foydalanuvchilarining huquqlarini aks ettirgan. Ustundagi `r`, `w` va `x` harflari mos ravishda o'qishga, yozishga va bajarishga bo'lgan huquqlardir. Agar huquq berilmagan bo'lsa, chiziqcha (`-`) belgisi qo'yilgan.

Dasturlashda huquqlar ikkilik sanoq tizimida beriladi. 1 yoki 0 mos ravishda huquq bor yoki yo'qligini bildiradi. Raqam o'rniga qarab esa kimning va qanday huquqi aniqlanayotganini bilish mumkin. Demak, barcha huquqlar 9 xonali ikkilik son sifatida beriladi. Masalan, $(110100100)_2$ soni fayl egasida o'qish/yozish, qolgan barchada faqat o'qish huquqi borligini bildiradi.

Fayllarda saqlanayotgan axborotlar foydalanuvchiga tegishli bo'lsada, fayllar operatsion tizim tomonidan boshqariladi. Boshqarish deganda quyidagi vazifalar nazarda tutiladi:

- Fayllarni qattiq diskda joylashtirish, qaysi fayl diskning nechanchi sektoriga yozilganini nazorat qilish.
- Foydalanuvchiga uning fayllarini istagan paytda va istalgan tartibda saqlash imkoniyatini berish, fayllarga murojaat etish va ularga o'zgartirish kiritish imkoniyatlari ham shular jumlasidan.
- Foydalanuvchi dasturlariga fayllar bilan yuqori darajada, ya'ni qattiq disk bilan bo'ladigan murakkab fizik jarayonlarga aralashmagan holda ishlash imkoniyatini berish.

17-jadval

Operatsion tizim	**Fayl tizimi nomi**	**Ishlab chiqaruvchi**
DOS	FAT (**F**ile **A**lloction **T**able – Fayllarni Joylashtirish Jadvali)	Microsoft
Windows (98, NT, XP)	NTFS (**N**ew **T**echnology **F**ile **S**ystem – Yangi Texnologiya Fayl Tizimi)	Microsoft
Linux	EXT (**Ext**ended File System – Kengaytirilgan Fayl Tizimi)	Remy Card[1]
UNIX	UFS (**U**NIX **F**ile **S**ystem – UNIX Fayl Tizimi)	CSRG[2]
Mac OS X	HFS (**H**ierarchical **F**ile **S**ystem – Darajama-daraja joylashtirish Fayl Tizimi)	Apple

[1] Fransuz dasturchisi. Linux operatsion tizimi yaratilishiga katta hissa qo'shgan.

[2] Computer System Research Group, ya'ni Kompyuter Tizimlari Tadqiqot Guruhi. Professor Bob Fabry tomonidan asos solingan AQSH ning Kalifornia shtatidagi Berkeley universitetidagi tadqiqot guruhi.

Ushbu ishlarni operatsion tizmning bir qismi sanaluvchi *fayl tizimi* dasturi amalga oshiradi. Har bir operatsion tizmning o'z fayl tizimi mavjud va ular bir biridan farq qiladi. 17-jadvalda bir nechta eng mashhur operatsion tizmlarning fayl tizmlari berilgan.

Albatta biz ushbu kitobda jadvaldagi barcha fayl tizimlari bilan batafsil tanishib chiqa olmaymiz. Hozircha faqat fayl tizimlari juda ko'pligi va ular bir-biridan farq qilishini bilish kifoya.

Fayl tizimlari turli-tuman bo'lgani bilan ularning hammasi asosiy bir g'oya asnosida qurilgan. Bu esa dasturlashni ancha osonlashtiradi. Masalan, bir fayl tizimi uchun yozilgan dasturni boshqa fayl tizimida ishlaydigan qilib o'zgartirish juda katta mehnat talab qilmaydi. Quyida keltiriladigan amaliy dasturlarda o'zingiz buning guvohi bo'lasiz.

Fayl tizimlarida qo'llaniladigan asosiy g'oya shundan iboratki, fayllar jadval shaklidagi tuzilmalar orqali boshqariladi. Jadvalning har bir qatori bitta fayl bilan bog'liq bo'lib, u yerda tashkiliy ishlarga molik ma'lumotlar saqlanadi. Ushbu ma'lumotlar turlari umumiy mezon bo'yicha taxminan quyidagilarga bo'linadi:

- Fayl turi
- Faylga qo'yilgan ramziy yorliqlar soni
- Fayl o'lchami (baytlarda)
- Fayl joylashgan qattiq disk bo'limining nomi (Masalan, *C:\, D:*)
- Fayl tizimidagi faylning tartib raqami
- Fayl egasi
- Faylga tegishli foydalanuvchi guruhi
- Fayl qachon yaratilgani, oxirgi o'zgartishlar kiritilgani haqidagi va boshqa sanalar
- Faylga berilgan huquqlar

Masalan, fayl tizimi jadvalida Windowsning *C:* disk bo'limidagi direktoriya va fayllar quyidagicha ko'rinishga ega bo'lishi mumkin.

18-jadval

Fayl tartib raqami	Fayl nomi	Fayl turi	Fayl o'lchami	Disk bo'limi	Fayl egasi	Huquqlar
...	...	...	...	...	...	...
21	*Program files*	Direktoriya	2 552 b	*C:*	Admin	111 111 000
22	*Document and Settings*	Direktoriya	363 b	*C:*	Fazliddin	110 100 100
23	*pagefile.sys*	Tizim fayli[1]	684 Mb	*C:*	System	111 100 100
24	*IO.SYS*	Tizim fayli	814 b	*C:*	System	111 100 100
...	...	...	...	...	...	...

Jadvaldagi birinchi ustun faylning tartib raqamini beradi. Bu tartib raqam inglizchada *file descriptor* deb yuritiladi. Qisqasini aytganda, ma'lum bir faylga murojaat qilganda uning jadvaldagi tartib raqamidan foydalaniladi. Fayl va uning fayl tizimidagi tartib raqamini xotiradagi o'zgaruvchi va uning xotiradagi manziliga o'xshatishimiz mumkin. Ammo fayl va o'zgaruvchi o'rtasidagi ushbu o'xshashlik shu bilan yakun topadi. Chunki fayl qattiq diskda toki uni foydalanuvchi o'chirib yubormaguncha doimiy saqlanadi, o'zgaruvchi esa dastur ishga tushgandagina yaratiladi va dastur yakunlangach o'chib ketadi.

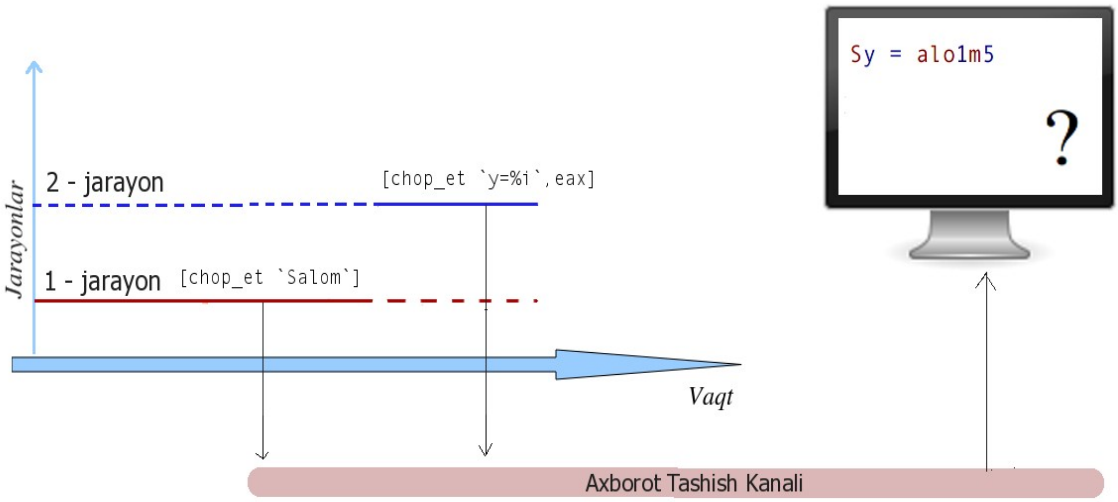
[1]Sanab o'tilgan fayl turlaridan tashqari Windowsda yana operatsion tizimga tegishli fayllar bo'lib, ularni o'quv/yozuv moslamalari yoki aloqa fayllariga o'xshatishimiz mumkin.

Ushbu mavzuda keltirilgan ma'lumotlar ma'lum darajada osonlashtirilgan va mavhumlashtirilgan, ya'ni aynan amaliyotda hammasi bir oz boshqacharoq bo'lishi mumkin. Bizning asosiy maqsadimiz fayl tizimlarini chuqur o'rganish emas, balki ular haqida tushunchaga ega bo'lishdir.

12.2. Tizim chaqiriqlari

Hozirgi zamonaviy operatsion tizimlarda barcha foydalanuvchi dasturlari *foydalanuvchi tartibida* bajariladi. Bunda bajarilayotgan dastur operatsion tizim tomonidan boshqariladi va dastur huquqlari ancha chegaralangan bo'ladi. Masalan, dastur to'g'ridan-to'g'ri kompyuter qurilmalari bilan axborot almasha olmaydi. Buning uchun operatsion tizimdan ruxsat so'rash kerak bo'ladi.

Foydalanuvchi tartibida ishlayotgan dasturga operatsion tizim o'zining bir qancha xizmatlarini taklif qiladi. Foydalanuvchi dasturi esa ushbu xizmatlardan foydalangan holda turli amallarni bajarishi mumkin. Operatsion tizim xizmatlari asosan kompyuter qurilmalari bilan muloqot qilish uchun mo'ljallangan.



58-rasm.

Kompyuter qurilmalariga murojaat operatsion tizim orqali amalga oshirilishining o'z sabablari mavjud. Masalan, bir vaqtning o'zida MP da ikkita jarayon[1] bajarilayotgan bo'lishi mumkin va ular bir vaqtning o'zida mos ravishda `Salom` va `y=15` ni monitorga chop etmoqchi bo'ldi deylik. Agar dasturlarda bu ishni to'g'ridan-to'gri amalga oshirish imkoni bo'lsa, monitorda tushunarsiz yozuvlar paydo bo'lishi mumkin. Chunki ikki chop etish ham bir-biridan bexabar tarzda amalga oshiriladi. 58-rasmda tasvirlanganidek qaysi yozuv qaysi dasturga tegishli ekanligi tushunarsiz bo'lib qoladi.

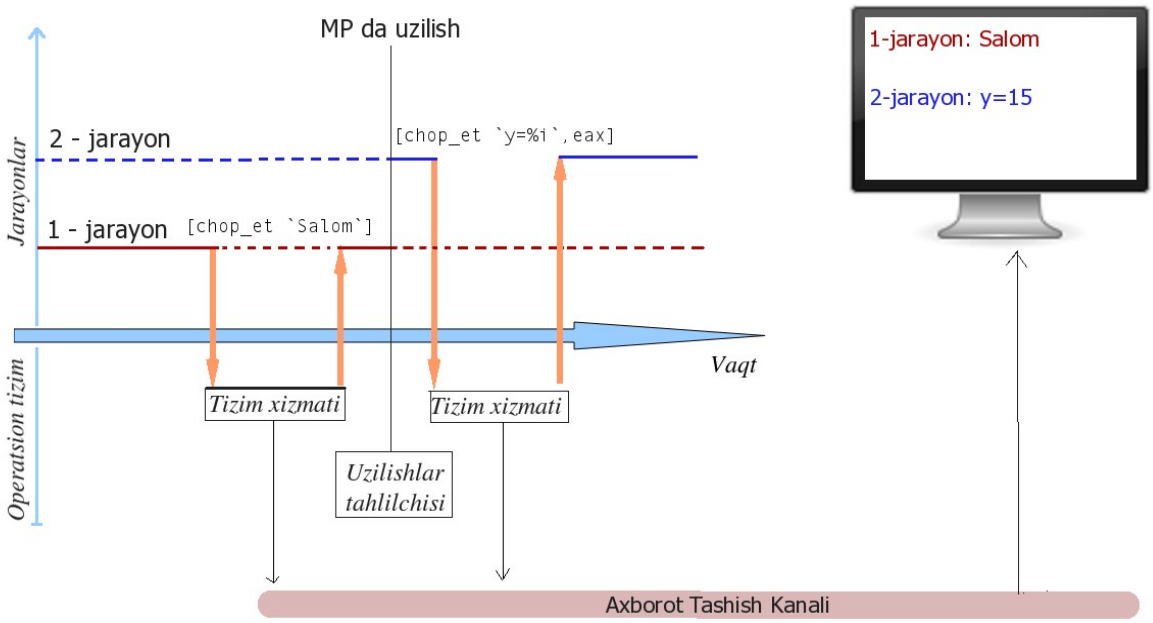
Shuning uchun masala barcha boshqaruvni operatsion tizim zimmasiga yuklash orqali yechilgan. Bunda operatsion tizim yo'l chirog'iga o'xshagan tushunchadan foydalanadi. Agar ikkita jarayon bitta moslamaga murojaat qilsa (bir vaqtning o'zida), biri uchun yashil chiroq, ikkinchisi uchun esa qizil chiroq yonadi. Biri ishini bajarib bo'lgach, ikkinchisi uchun yashil chiroq yonadi.

Operatsion tizim xizmatlariga murojaat qilish *tizim chaqirig'i* deyiladi. Operatsion tizim xizmatlari qism dasturi ko'rinishida tuzilgan. Ammo ular «Qismli dasturlash» bobida o'rganilgan

[1] Bu yerda jarayon deb foydalanuvchi dasturlari nazarda tutiliyapti. To'liq ma'lumot uchun «Markaziy protsessor» mavzusiga qarang.

oddiy qism dasturlaridan bir oz farq qiladi. Gap shundaki, MP da foydalanuvchi dasturlari foydalanuvchi tartibida bajarilsa, operatsion tizim dasturlari esa *operatsion tizim* tartibida bajariladi. MP bir vaqtning o'zida faqat bir tartibda ishlashi mumkin.

Tizim xizmatlari operatsion tizim tartibida ishlashi kerak, ya'ni ular foydalanuvchi dasturidan chaqirilganda, MP foydalanuvchi tartibidan operatsion tizim tartibiga o'tishi kerak. Bu ish MP da uzilish e'lon qilish orqali bajariladi. Assemblerda uzilish e'lon qiladigan maxsus `INT` (**Int**eruption – Uzilish) buyrug'i mavjud. Foydalanuvchi tartibida bajarilayotgan jarayon uzilish yordamida MP ga o'zini bajarishni to'xtatib turib, operatsion tizim tartibiga o'tish keragligini buyuradi. Uzilish yuzaga kelishi bilanoq operatsion tizimning *uzilishlar tahlilchisi* dasturi ishga tushadi. Uzilishlar tahlilchisi bir qancha tekshiruvlarni amalga oshirgach o'z navbatida tizim *chaqiriqlari tahlilchisini* chaqiradi. Tizim chaqiriqlari tahlilchisi esa foydalanuvchi dasturi qaysi tizim xizmatini so'ragan bo'lsa, o'shani ishga tushiradi va uning natijasini yana qaytib foydalanuvchi dasturiga jo'natadi. 58-rasmdagi holat tizim xizmatlari orqali qanday amalga oshirilishi 59-rasmda tasvirlangan.



59-rasm.

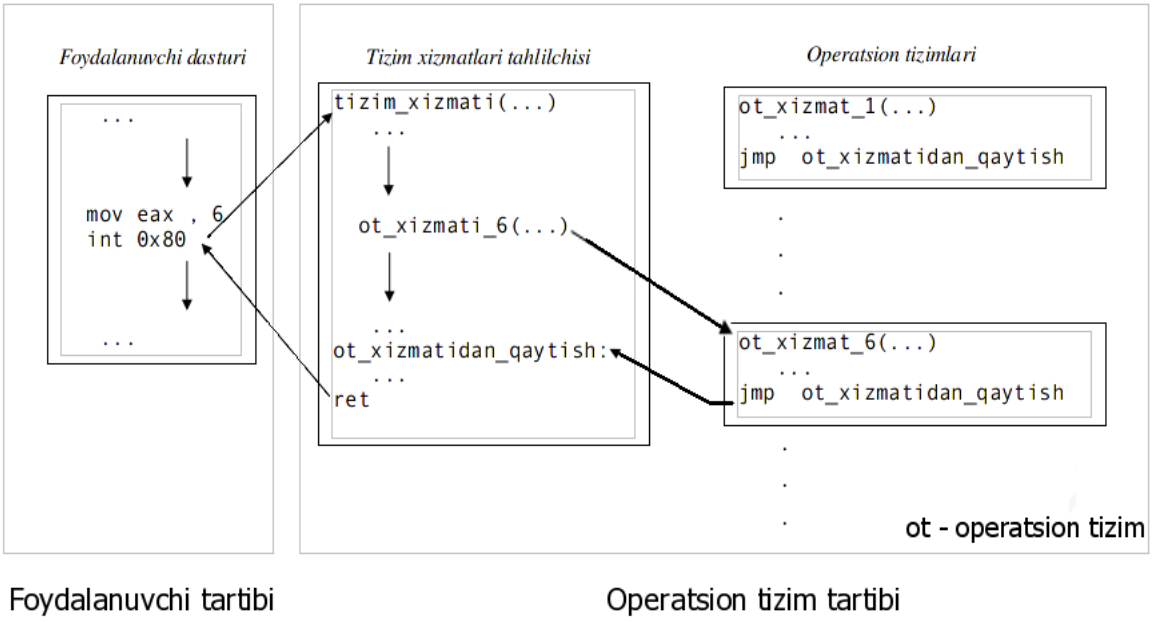
Har bir tizim xizmati o'z tartib raqamiga ega bo'lib, foydalanuvchi dasturi uzilish e'lon qilishidan oldin unga qaysi xizmat kerak bo'lsa, o'shaning raqamini EAX orqali jo'natishi kerak. Tizim xizmati turiga qarab yana qo'shimcha qiymatlar EBX, ECX va EDX registrlari orqali ham jo'natilishi mumkin. Dasturlash nuqtai nazaridan qaraganda tizim xizmatlari oddiy qism dasturlaridan tuzilish jihatdan farq qilmaydi. Ularga ham qiymatlar stack orqali jo'natiladi va natija EAX orqali qabul qilinadi. Tizim xizmatlari ko'pincha muvaffaqiyatli yakunlanganda musbat yoki nol sonini, biror xato yuz berganda esa manfiy qiymat qaytaradi. 60-rasmda dasturlar bir-biri bilan qanday aloqa qilishi tasvirlangan.

Ushbu bobda tizim chaqiriqlariga to'xtalganimizning sababi shundaki, fayllar ham kompyuterning qattiq disk qurilmasida saqlanadi va ular bilan ishlash uchun operatsion tizim xizmatlariga murojaat qilishga to'g'ri keladi. Bizni asosan fayllar bilan bog'liq bo'lgan tizim chaqiriqlari qiziqtiradi. Barcha zamonaviy operatsion tizimlar fayllarni boshqaradigan xizmatlarni taklif qiladi va ularning eng asosiylari quyidagilardan iborat:

- Faylni ochish (fayl bilan ishlashdan oldin uni ochish kerak).
- Fayldan axborot o'qish.
- Faylga yangi axborot yozish.

- Faylni yopish.
- Faylni bir direktoriyadan boshqasiga ko'chirish.
- Faylni o'chirib tashlash.

Quyidagi mavzularda ushbu tizim xizmatlari Linux va Windowsda yana qanday ishlatilishi haqida so'z boradi.



60-rasm.

12.3. Fayllar bilan assemblerda ishlash

Dasturlashda fayl bilan ishlash asosan uch bosqichga bo'linadi:

1) Avval fayl u bilan biror amal bajarish uchun tizim chaqirig'i orqali ochiladi. Fayl turli xil maqsadlarda ochilishi mumkin:
   - O'qish maqsadida
   - Yozish maqsadida

   Agar fayl yozish maqsadida ochilsa, yana qo'shimcha maqsadlar berilishi mumkin. Ular quyidagilar:
   - Yaratish maqsadi – agar ochilayotgan fayl mavjud bo'lmasa, u yaratiladi.
   - Tozalash maqsadi – agar faylda allaqachon qandaydir axborot yozilgan bo'lsa, ular o'chirib tashlanadi va yozish faylning boshidan boshlanadi.
   - Qo'shish maqsadi – faylga yozish fayl oxiridan davom ettiriladi.
2) Ma'lum maqsadda ochilgan fayl bilan ishlash, ya'ni unga yozish yoki undan o'qishni amalga oshirish. Ushbu bosqich asosiy hisoblanadi.
3) Barcha ishlar bajarib bo'lingach fayl yopiladi.

Yuqorida aytib o'tilganidek har bir tizim xizmati o'z tartib raqamiga ega va qism dasturi sifatida kerakli qiymatlar qabul qiladi. Bunday omillar operatsion tizim turiga qarab har xil bo'ladi. Shu sababdan quyida xususiy hollar yoritilgan.

12.3.1. Xususiy holat: Linux

Avval aytib o'tilganidek, tizim chaqiriqlariga MP da uzilish e'lon qilish yordamida murojaat qilinadi. Buning uchun assemblerda `INT` buyrug'i mavjud bo'lib, u MP da uzilishni yuzaga keltiradi.

Qoidasi:

```
int  uzilish_turi
```

Bu yerda `uzilish_turi` o'rnida o'zgarmas son kelishi mumkin va bu son uzilish turini aniqlaydi. Har xil uzilish turlari mavjud va ular nimaga yo'naltirilganiga qarab farq qiladi. Masalan, uzilish BIOS yoki operatsion tizim uchun bo'lishi mumkin. Bizni faqat operatsion tizimga mo'ljallangan uzilishlar qiziqtiradi. Linux uchun uzilish turi 128 ga teng, ammo uzilish turi ko'pincha o'n oltilik sanoq tizimimda beriladi, ya'ni `0x80`.

Uzilish e'lon qilishdan oldin EAX, EBX,ECX va EDX registrlariga kerakli qiymatlar yuklangan bo'lishi kerak. EAX da tizim xizmati tartib raqami saqlanishi lozim. Qolgan registrlarda esa tizim xizmati turiga qarab har xil zarur qiymatlar bo'lishi kerak.

Linuxning barcha xizmatlar ro'yxati */usr/include/asm-generic/unistd.h* faylida berilgan. Xizmatlarga to'g'ri keladigan barcha qism dasturlarining nomi `sys_` bilan boshlanadi. Linuxdagi barcha mavjud xizmatlar tartib raqamlarining ro'yxati */usr/include/asm/unistd_32.h* faylida taqdim etilgan.

Ishni faylni ochish tizim xizmatidan boshlaymiz. Uning tartib raqami 5 ga teng va C da quyidagicha e'lon qilingan:

```
int  open(
     char * fayl_yo'li,
     int ochish_maqsadlari,
     int fayl_huquqlari
     );
```

Jo'natiladigan qiymatlar quyidagi ma'nolarga ega:

fayl_yo'li	Faylning mutlaq yoki nisbiy yo'lini o'zida qatorli o'zgaruvchi sifatida saqlayotgan qolip manzili. Assemblerda ushbu qiymat EBX registri orqali beriladi.
ochish_maqsadlari	Faylni qay maqsadda ochish butun son sifatida beriladi. Har bir maqsad son sifatida aniqlanadi. Agar ochish maqsadlari bittadan ortiq bo'lsa, ular mantiqiy qo'shish orqali beriladi. 19-jadvalda faylni har xil ochish maqsadlari sonli qiymatlar ko'rinishida keltirilgan. Assemblerda ushbu qiymat ECX registri orqali beriladi.
fayl_huquqlari	Faqat fayl yaratilayotgandagina ma'noga ega butun son bo'lib, yangi faylga shu huquqlar beriladi. Assemblerda ushbu qiymat EDX registri orqali beriladi.

Agar faylni ochish muvaffaqiyatli yuz bergan bo'lsa, ochilgan fayl tartib raqami EAX da qaytariladi.

Eslatma: Faylni yozish maqsadida ochish uchun faylda yozish ruxsati bo'lishi kerak. Aks holda faylni ochish muvaffaqiyatsiz tugaydi, ya'ni EAX<0 bo'ladi.

Yaratish maqsadiga batafsilroq to'xtaymiz. Bu maqsad berilganda agar fayl mavjud bo'lmasa, shunday fayl yaratiladi. Ravshanki, yaratish maqsadi bilan birgalikda yozish maqsadi ham berilishi kerak. Chunki, endigina yaratilayotgan fayldan biror narsa o'qishning iloji yo'q. Fayl yaratilayotganda unga beriladigan huquqlar ham ko'rsatilishi kerak.

19-jadval

Madsaq	Maqsad soni
O'qish	0x000
Yozish	0x001
Yaratish	0x040
Tozalash	0x200
Qo'shish	0x400

Masalan, quyidagi faylni ochish kerak bo'lsin:

```
fayl db   "mening_matnim.txt",0
```

Uni o'qish uchun ochamiz:

```
mov  eax , 5              ;; Xizmat raqami.
mov  ebx , fayl           ;; Fayl nomi.
mov  ecx , 0x000          ;; O'qish maqsadi
int  0x80                 ;; Uzilish
```

Bu yerda mavjud fayl o'qish uchun ochilgani sababli EDX ning qiymati ahamyatsizdir. Uzilishdan so'ng ochilgan fayl tartib raqami EAX da qaytariladi. Agar faylni ochishda biror muammo bo'lganda, EAX da manfiy son bo'ladi. Aynan qanday xato yuz berganini EAX ning qiymatiga qarab aniqlash mumkin. Barcha xato turlari */usr/include/asm-generic/errno-base.h* faylida musbat sonlar ko'rinishida keltirilgan. Fayl ochilgan yoki ochilmaganligini quyidagicha tekshirishimiz mumkin.

```
     cmp  eax , 0
     ja   .fayl_ochildi
     chop_et   `%s ni ochib bo'lmadi`, fayl
     jmp  yakun
fayl_ochildi:
     ;; fayl ustida ishlash
yakun:
```

Ochilgan fayl tartib raqamini darhol biror doimiy o'zgaruvchida saqlab qo'ygan ma'qul:

```
mov  [fayl_tartib_raqami] , eax
```

Eslatma: *mening_matnim.txt* fayli muvaffaqiyatli ochilishi uchun dastur ishga tushirilgan direktoriyada fayl ham allaqachon mavjud bo'lishi kerak.

Barcha dasturlar ishga tushganda ularga operatsion tizim tomonidan uchta mezoniy fayllar ochib beriladi va dastur davomida ushbu fayllarni ishlatishdan oldin ularni ochish talab qilinmaydi. Bu fayllarning biri tugmachalar taxtasi bo'lib, u doim o'qish uchun ochiladi, tartib raqami esa 0 ga teng. Ushbu fayl ingliz tilida *stdin* deb yuritiladi. Ikkinchi fayl monitor bo'lib, u doim yozish uchun ochiladi, tartib raqami esa 1 ga teng. Ushbu fayl ingliz tilida *stdout* deb yuritiladi va nihoyat uchunchi fayl xatolar haqidagi ma'lumotlar fayli bo'lib, u ham monitorga ulangan, tartib raqami esa 2 ga teng. Ushbu fayl ingliz tilida *stderr* deb yuritiladi.

Bu uch fayl orqali barcha chop etilishlar va tugmachalar taxtasudan qiymat o'qishlar bajariladi. Shu paytgacha foydalanilgan C kutubxonasidagi `printf` va `scanf` qism dasturlari ham aynan shu mezoniy fayllar orqali o'quv/yozuvni amalga oshiradi.

Faylda saqlanayotgan axborotni baytlar ketma-ketligi deb hisoblasak, uni kataklarining o'lchami bir baytdan bo'lgan qolipga o'xshatish mumkin. Fayldagi baytlarni o'qish uning istalgan yeridan yoki boshidan amalga oshirilishi mumkin. Fayldan o'qish tizim xizmatining tartib raqami 3 ga teng va uning C dagi e'lon qilinishi quyidagicha:

```
int  read(int fayl_tartib_raqami, char * qolip, int n);
```

Jo'natiladigan qiymatlar quyidagi ma'nolarga ega:

fayl_tartib_raqami	O'qish amalga oshiriladigan fayl tartib raqami. Ushbu butun son faylni ochish tizim xizmati natijasi bo'lishi kerak. Agar tugmachalar taxtasidan o'qish amalga oshirilayotgan bo'lsa, `STDIN` fayli beriladi, ya'ni nol. Assemblerda ushbu qiymat EBX registri orqali beriladi.
qolip	Fayldan o'qilgan baytlarni xotiraga saqlash manzili. Assemblerda ushbu qiymat ECX registri orqali beriladi.
n	Fayldan necha bayt o'qilishi kerakligini anglatadi. `qolip` uzunligi n dan kam bo'lmasligi kerak. Assemblerda ushbu qiymat EDX registri orqali beriladi.

Ushbu xizmat fayldan *n* bayt axborot o'qib uni qolipga joylashtiradi. O'qish bir boshdan amalga oshiriladi, ya'ni birinchi marta fayl boshidan, masalan, 100 bayt o'qilsa, keyingi chaqiriqda o'qish faylning qolgan yeridan davom ettiriladi. Natija sifatida haqiqiy o'qilgan baytlar soni qaytariladi. Amalda o'qilgan baytlar soni tizim chaqirig'idan oldin berilgan o'qilishi kerak bo'lgan baytlar sonidan kam bo'lishi mumkin. Agar nol qaytarilsa, demak, fayl oxiriga yetilgan bo'ladi.

Misol tariqasida *mening_matnim.txt* faylidagi birinchi 100 bayt axborotni o'qishga harakat qilib ko'ramiz. `qator` o'zgaruvchisi quyidagicha e'lon qilingan bo'lsin:

```
qator     resb 100
qator_uzunligi equ  $-qator
```

Yuqoridagi dasturni davom ettiramiz:

```
fayl_ochildi:
     mov  eax , 3                       ;; O'qish tizim xizmati.
     mov  ebx , [fayl_tartib_raqami]    ;; Ochilgan fayl tartib raqami.
```

```
mov  ecx , qator
mov  edx , qator_uzunligi
int  0x80                          ;; Uzilish.
```

Shundan so'ng `qator` da o'qilgan baytlar yuklangan bo'ladi. EAX da esa o'qilgan baytlar soni bo'ladi.

Endi `qator` dagi fayldan o'qilgan baytlarni `chop_et` makrosi yordamisiz chop etamiz. Buning uchun `qator` dagi baytlar yozish tizim chaqirig'i orqali `STDOUT` ga yoziladi. Faylga yozish tizim xizmatining tartib raqami 4 ga teng va uning C dagi e'lon qilinishi quyidagicha:

```
int  write(int fayl_tartib_raqami, char * qolip, int n);
```

Ushbu xizmat qiymatlarining ma'nolari `read` bilan bir xil. Faqat bunda `qolip` dagi `n` bayt faylga yoziladi. Natija sifatida muvaffaqiyatli yozilgan baytlar soni qaytariladi.

Demak, `qator` dagi baytlar quyidagicha chop etiladi:

```
mov  edx , eax      ;; Necha bayt o‘qilgan bo'lsa, shuncha
                    ;; chop etamiz.
mov  eax , 4        ;; Yozish tizim xizmati.
mov  ebx , 1        ;; STDOUT tartib raqami.
int  0x80           ;; ECX da shundoq ham qatorning manzili
                    ;; turibdi.
```

Dastur ishga tushganda fayldagi birinchi 100 baytli matn chop etilganini ko'rasiz. Ochilgan fayl bilan bo'lgan barcha ishlar yakunlangach uni yopish lozim. Faylni yopish tizim xizmatining tartib raqami 6 ga teng va uning C dagi e'lon qilinishi quyidagicha:

```
int  close(int fayl_tartib_raqami);
```

Assemblerda fayl tartib raqami EBX registri orqali beriladi. Demak, yuqorida o'qish uchun ochilgan fayl quyidagicha yopiladi:

```
mov  eax , 6
mov  ebx , [fayl_tartib_raqami]
int  0x80
```

Ushbu bobga unchalik aloqasi bo'lmasada, yana bir tizim xizmatini ko'rib chiqamiz. Dasturni tugatish ham tizim xizmati orqali bajariladi. Dasturni yakunlash xizmati tartib raqami 1 ga teng. EBX da esa muvaffaqiyatli yoki muvaffaqiyatsiz tugallanganligi haqida ma'lumot bo'lishi kerak. Demak, `RET` buyrug'idan foydalanmasdan ham dasturni yakunlash imkoniyati mavjud.

```
mov  eax , 1
mov  ebx , 0        ;; Muvaffaqiyatli yakun
int  0x80
```

Shu paytgacha olgan bilimlarimiz operatsion tizim xizmatlaridan foydalangan holda amaliy dastur tuzishga yetadi. Misol tariqasida bir fayldan ikkinchisiga nusxa oladigan (ko'chiradigan) dasturni ko'rib chiqamiz. Ushbu dasturda biz marko vositalardan keng foydalanamiz. Buning bosh sababi har xil raqamlarning ko'payib ketganligidir. Tizim xizmatlari tartib raqamlari, faylni ochish maqsadlari, mezoniy fayllarning tartib raqamlari va hokazolar shular jumlasidandir. Aytib o'tilgan sonlarga nom berilib, bir satrli makro ko'rinishga keltirilsa, dasturni tushunish ancha osonlashadi. Bundan tashqari tizim chaqirig'ini amalga oshirishning o'zi ham kamida 4-5 ta buyruq yozishni talab qilyapti. Har bir tizim chaqirig'i uchun bittadan ko'p satrli makro aniqlansa,

dastur yanada ixcham ko'rinishga keladi. Barcha makro aniqlashlarni bitta boshlang'ich faylga joylashtiramiz.

21-namuna: linux_tizim_xizmatlari.inc

```
(1)   %ifndef LINUX_TIZIM_XIZMATLARI_INC
(2)   %define LINUX_TIZIM_XIZMATLARI_INC
(3)
(4)   ;; Mezoniy fayllar
(5)   %define  STDIN     0
(6)   %define  STDOUT    1
(7)   %define  STDERR    2
(8)
(9)   ;; Tizim xizmatlari
(10)  %define  FAYLNI_OCHISH_TCH  5
(11)  %define  O_QISH_TCH     3
(12)  %define  YOZISH_TCH     4
(13)  %define  FAYLNI_YOPISH_TCH   6
(14)  %define  DASTURNI_YAKUNLASH_TCH   1
(15)
(16)  ;; Faylni ochish maqsadlari
(17)  %define  O_QISH_MAQ     0x000
(18)  %define  YOZISH_MAQ     0x001
(19)  %define  YARATISH_MAQ   0x040
(20)  %define  TOZALASH_MAQ   0x200
(21)  %define  QO_SHISH_MAQ   0x400
(22)
(23)
(24)  ;;LINUX TIZIM XIZMATLARIGA MUROJAAT ETADIGAN MAKRO
(25)  ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
(26)  ;; * Ushbu makroga qiymatlar REGISTERLARDA berilishi
(27)  ;;   shart emas!
(28)  ;;
(29)  ;; * qiymatlari
(30)  ;;  %1 - tizim xizmatining tartib raqami
(31)  ;;  %2-%4 - tizim xizmatiga beriladigan
(32)  ;;       qo'shimcha qiymatlar
(33)  ;; (agar %3 va %4 berilmasa, ular 0 ga teng qilib olinadi)
(34)  %macro tizim_xizmati 2-4 0, 0
(35)      push edx
(36)      push ecx
(37)      push ebx
(38)      push dword %1
(39)      push dword %2
(40)      push dword %3
(41)      push dword %4
(42)      pop edx
(43)      pop ecx
(44)      pop ebx
(45)      pop eax
(46)      int 0x80
(47)      pop ebx
(48)      pop ecx
(49)      pop edx
(50)  %endmacro
```

```
(51)  ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
(52)
(53)  ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
(54)  ;; * Faylni ochish tizim chaqiruvi
(55)  ;;
(56)  ;; * qiymatlari
(57)  ;;  %1 - fayl yo'li
(58)  ;;  %2 - faylni ochish maqsadlari
(59)  ;;  %3 - faylga beriladigan ruxsatlar
(60)  ;; (agar %3 berilmasa, u 111000000b ga teng qilib olinadi)
(61)  %macro faylni_ochish 2-3 111000000b
(62)
(63)      tizim_xizmati FAYLNI_OCHISH_TCH, %1, %2, %3
(64)
(65)  %endmacro
(66)  ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
(67)
(68)  ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
(69)  ;; * O'qish tizim chaqiruvi
(70)  ;;
(71)  ;; * qiymatlari
(72)  ;;  %1 - fayl tartib raqami
(73)  ;;  %2 - o'qilgan baytlarni saqlash manzili
(74)  ;;  %3 - necha bayt o'qilishi
(75)  %macro fayldan_o_qish 3
(76)
(77)      tizim_xizmati O_QISH_TCH, %1, %2, %3
(78)
(79)  %endmacro
(80)  ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
(81)
(82)  ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
(83)  ;; * Yozish tizim chaqiruvi
(84)  ;;
(85)  ;; * qiymatlari
(86)  ;;  %1 - fayl tartib raqami
(87)  ;;  %2 - yoziladigan baytlarning xotiradagi manzili
(88)  ;;  %3 - necha bayt yozilishi
(89)  %macro faylga_yozish 3
(90)
(91)      tizim_xizmati YOZISH_TCH, %1, %2, %3
(92)
(93)  %endmacro
(94)  ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
(95)
(96)  ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
(97)  ;; * Faylni yopish tizim chaqiruvi
(98)  ;;
(99)  ;; * qiymatlari
(100) ;;  %1 - fayl tartib raqami
(101) %macro faylni_yopish 1
(102)
(103)     tizim_xizmati FAYLNI_YOPISH_TCH, %1
(104)
```

```
(105) %endmacro
(106) ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
(107)
(108) ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
(109) ;; * Dasturni yakunlash tizim chaqiruvi
(110) ;;
(111) ;; * qiymatlari
(112) ;;  %1 - Dastur qanday tugaganligi haqida xabar
(113) %macro dasturni_yakunlash 1
(114)
(115)     tizim_xizmati DASTURNI_YAKUNLASH_TCH, %1
(116)
(117) %endmacro
(118) ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
(119)
(120) %endif        ;; LINUX_TIZIM_XIZMATLARI_INC faylining oxiri
```

nusxa.asm

```
(1)   %include "linux_tizim_xizmatlari.inc"
(2)
(3)   section  .bss
(4)   yoziladigan_fayl_tr     resd 1   ;;Yoziladigan fayl tartib raqami
(5)   o_qiladigan_fayl_tr     resd 1   ;;O'qiladigan fayl tartib raqami
(6)
(7)   %define  fayl_nomi_uzn  200
(8)   o_qiladigan_fayl_nomi   resb fayl_nomi_uzn
(9)   yoziladigan_fayl_nomi   resb fayl_nomi_uzn
(10)
(11)  %define  qator_uzunligi 100       ;; 100 baytdan o'qiymiz
(12)  qator    resb qator_uzunligi
(13)
(14)  n   resd 1
(15)
(16)  section  .data
(17)  taklif1  db   "Nusxa olinadigan fayl nomi: ",0
(18)  taklif1_uzunligi   equ  $-taklif1
(19)
(20)  taklif2  db   "Yangi fayl nomi: ",0
(21)  taklif2_uzunligi   equ  $-taklif2
(22)
(23)  xatodb   "Faylni ochib bo'lmadi",0
(24)  xato_uzunligi equ  $-xato
(25)
(26)  section  .text
(27)  global   main
(28)
(29)  main:
(30)      faylga_yozish STDOUT, taklif1, taklif1_uzunligi
(31)      fayldan_o_qish STDIN, o_qiladigan_fayl_nomi, fayl_nomi_uzn
(32)      mov  byte[o_qiladigan_fayl_nomi+eax-1] , 0
(33)
(34)      faylga_yozish STDOUT, taklif2, taklif2_uzunligi
(35)      fayldan_o_qish STDIN, yoziladigan_fayl_nomi, fayl_nomi_uzn
(36)      mov  byte[yoziladigan_fayl_nomi+eax-1] , 0
```

```
(37)
(38)      faylni_ochish o_qiladigan_fayl_nomi, O_QISH_MAQ
(39)      cmp  eax , 0
(40)      ja   o_qiladigan_fayl_ochildi
(41)      faylga_yozish STDOUT, xato, xato_uzunligi
(42)      jmp  yakun
(43)
(44)  o_qiladigan_fayl_ochildi:
(45)      mov  [o_qiladigan_fayl_tr] , eax
(46)
(47)      faylni_ochish    yoziladigan_fayl_nomi,    (YARATISH_MAQ    |
          YOZISH_MAQ | TOZALASH_MAQ)
(48)      cmp  eax , 0
(49)      ja   yoziladigan_fayl_ochildi
(50)      faylga_yozish STDOUT, xato, xato_uzunligi
(51)      jmp  yakun
(52)
(53)  yoziladigan_fayl_ochildi:
(54)      mov  [yoziladigan_fayl_tr] , eax
(55)
(56)  ko_chirish:
(57)      fayldan_o_qish [o_qiladigan_fayl_tr], qator, qator_uzunligi
(58)      mov  [n] , eax
(59)      faylga_yozish  [yoziladigan_fayl_tr], qator, eax
(60)
(61)      cmp  dword[n] , 0
(62)      jnz  ko_chirish
(63)
(64)      faylni_yopish [o_qiladigan_fayl_tr]
(65)      faylni_yopish [yoziladigan_fayl_tr]
(66)
(67)  yakun:
(68)      dasturni_yakunlash 0
```

Ushbu dasturning ustunlik tomoni uning juda tez ishlashidadir. U orqali o'lchami katta fayllardan bemalol nusxa olish mumkin.

12.3.2. Xususiy holat: Windows

Windowsda masala ancha chigal. Chunki, UNIX operatsion tizimi oilasiga mansub bo'lgan, xususan Linux, FreeBSD, Mac OS X va boshqa tizimlarda chaqiriqlar umumiy bir mezonga keltirilgan. Biroq Microsoft firmasi mahsulotlari bunday imkoniyatga ega emas. Chunki Windowsning tizim chaqiriqlari tartib raqamlari hatto uning har bir turkumida har xildir. Masalan, tizim xizmatlaridan to'g'ridan-to'g'ri foydalangan holda Windows 95 uchun mo'ljallab yozilgan dastur Windows XP yoki Windows Vistada ishlamaydi. 61-rasmda faylni ochish tizim xizmatining bir nechta Windows turkumlaridagi tartib raqamlari misol tariqasida berilgan.

	Windows NT				Windows 2000					Windows XP			Windows 2003 Server	
	SP3	SP4	SP5	SP6	SP0	SP1	SP2	SP3	SP4	SP0	SP1	SP2	SP0	SP1
NtOpenFile	0x004f	0x004f	0x004f	0x004f	0x0064	0x0064	0x0064	0x0064	0x0064	0x0074	0x0074	0x0074	0x007a	0x007a

61-rasm.

Bundan tashqari Windows tijorat mahsuloti bo'lib, u yopiq tizim hisoblanadi. Microsoft firmasi o'z sirlarini osonlikcha oshkor qilavermaydi, bundan maqsad foydalanuvchi dasturlarini operatsion tizim ishlariga aralashishdan uzoqroq tutishdir. Buning natijasida MS-DOS turkumidan so'ng chiqarilgan tizimlarda foydalanuvchi dasturidan MP da uzilish e'lon qilish huquqi olib tashlangan. Tizim xizmatlariga murojaat esa Windowsning maxsus qism dasturlarini chaqirish orqali amalga oshiriladi. Ushbu qism dasturlari *kernel32.dll* va *user32.dll* kutubxonalaridan o'rin olgan.

Sanab o'tilgan kutubxonalarda Windowsning barcha xizmatlarini ishga soladigan qism dasturlari mavjud. Masalan faylni ochishni `CreateFile` qism dasturi bajaradi va uning C dagi e'lon qilinishi quyidagicha:

```
HANDLE WINAPI CreateFile(
  __in          LPCTSTR lpFileName,
  __in          DWORD dwDesiredAccess,
  __in          DWORD dwShareMode,
  __in          LPSECURITY_ATTRIBUTES lpSecurityAttributes,
  __in          DWORD dwCreationDisposition,
  __in          DWORD dwFlagsAndAttributes,
  __in          HANDLE hTemplateFile
);
```

Bu yerda jo'natiladigan qiymatlarning bir ikkitasini hisobga olmaganda, boshqalari unchalik katta ahamiyat kasb etmaydi. Umuman olganda Windows xizmatlarining ko'pchiligi shunga o'xshash bir qator ortiqcha qiymatlar jo'natilishini talab qiladi. Buning ustiga bu qiymatlar oddiy o'zgaruvchi yoki qolip manzili emas, balki murakkab tuzilmali obyektlardir. Bunday obyektlarni assembler tilida yaratish ushbu kitob vazifasiga kirmaydi. Agar siz aynan Windows xizmatlaridan foydalanib fayllar bilan ishlamoqchi bo'lsangiz, msdn.microsoft.com saytidagi kerakli hujjatlarga murojaat qiling.

Yuqorida sanab o'tilgan sabablarga ko'ra ushbu mavzuda fayllar bilan ishlash C dasturlash tilida mavjud bo'lgan mezoniy qism dasturlar orqali ko'rsatiladi. Aynan C tilidagi qism dasturlarining tanlanishiga sabab shuki, ularni assemblerdan turib chaqirish oson va C kutubxonalari barcha operatsion tizimlarda bir xil ishlaydi.

C dagi fayllar bilan ishlaydigan qism dasturlari Linux xizmatlariga juda o'xshaydi va chalkashlik kelib chiqmasligi uchun barchasi `f` harfi bilan boshlanadi. Demak, faylni ochishni `fopen` funksiyasi bajaradi va uning C dagi e'lon qilinishi quyidagicha:

```
FILE * fopen(
          const char * fayl_nomi,
          const char * faylni_ochish_maqsadlari
            );
```

Linuxdagi `open` dan farqli o'laroq bu yerda faylni ochish maqsadlari harflar ko'rinishida beriladi va ular quyidagilar:

r	O'qish maqsadi.
w	Yozish maqsadi, agar fayl mavjud bo'lsa, u tozalanadi; mavjud bo'lmasa, yangi fayl yaratiladi.
a	Qo'shish maqsadi, yozish maqsadi bilan birga berilishi kerak.

Agar faylni ochishda biror muammo yuz bersa, nol qaytariladi.

Fayldan o'qish `fread` orqali amalga oshiriladi va uning C dagi e'lon qilinishi quyidagicha:

```
int fread(
          char * qolip,
          int baytlar_soni,
          int bo'laklar_soni,
          FILE * fayl
          );
```

Qabul qilinadigan qiymatlarning berilish tartiblaridagi o'zgarishni hisobga olmasak, `fread` Linuxdagi `read` bilan yuz foiz mos tushadi. Faqat `baytlar_soni` hisobiga qiymatlar bittaga ko'paygan. Ushbu funksiya fayldan axborotni bo'lak-bo'lak qilib o'qiydi va `bo'laklar_soni` nechta bo'lak o'qilishi kerakligini beradi. `baytlar_soni` esa bitta bo'lak necha baytdan iborat ekanligini bildiradi. Demak, umumiy o'qilgan baytlar soni `bo'laklar_soni*baytlar_soni` ga teng bo'ladi. Biz faqat matnli fayllar bilan ishlayotganimiz uchun `baytlar_soni` ni doim 1 ga tenglashtirib olsak bo'ladi.

Faylga yozish qism dasturi `fwrite` bo'lib, uning e'lon qilinishi `fread` bilan bir xildir.

Faylni yopish `fclose` orqali bajariladi. Unga qiymat sifatida, albatta, `fopen` ning natijasi beriladi.

Misol tariqasida oldingi mavzuda keltirilgan fayldan nusxa olish amaliy dasturning C qism dasturlari orqali tuzilishini ko'rib chiqamiz. Asosiy o'zgartirishlar makro aniqlanishlarda bo'ladi. Ular shunday o'zgartiriladiki, asosiy dastur deyarli avvalgi shaklida qoladi.

22-namuna: c_qism_dasturlari.inc

```
(1)   %ifndef C_QISM_DASTURLARI_INC
(2)   %define C_QISM_DASTURLARI_INC
(3)
(4)   %include "nasm-io.inc"
(5)
(6)   ;; C mezoniy kutubxonasidagi fayllar bilan ishlovchi qism
(7)   ;; dasturlari.
(8)   tizim_extern  fopen, fread, fwrite, fclose
(9)   tizim_extern  exit      ;; Dasturni yakunlash Windowsda ham exit
(10)
(11)  ;; Tizim xizmatlari
(12)  %define  FAYLNI_OCHISH_TCH   fopen
(13)  %define  O_QISH_TCH         fread
(14)  %define  YOZISH_TCH         fwrite
(15)  %define  FAYLNI_YOPISH_TCH   fclose
(16)  %define  DASTURNI_YAKUNLASH_TCH   exit
(17)
(18)  ;; Faylni ochish maqsadlari
(19)  %define  O_QISH_MAQ     "r"
(20)  %define  YOZISH_MAQ     "w"
(21)  %define  QO_SHISH_MAQ   "wa"
(22)
(23)  ;; C QISM DASTURLARINI CHAQIRADIGAN MAKRO
(24)  ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
(25)  ;; * Ushbu makro qism dasturlarni chaqirishni osonlashtiradi.
(26)  ;;
(27)  ;; * qiymatlari
(28)  ;;  %1 - tizim xizmatini amalga oshiradigan qism dasturi nomi
```

```
(29)  ;;  %2-* - tizim xizmatiga beriladigan qo'shimcha qiymatlar
(30)  %macro tizim_xizmati 2-*
(31)
(32)  %xdefine qism_dastur    %1
(33)  %assign       qiymatlar_soni %0-1
(34)
(35)      %rep qiymatlar_soni
(36)           %rotate -1
(37)           push dword %1
(38)      %endrep
(39)
(40)      call qism_dastur
(41)      add       esp , qiymatlar_soni * 4
(42)
(43)  %endmacro
(44)  ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
(45)
(46)  ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
(47)  ;; * Faylni ochish qism dasturi
(48)  ;;
(49)  ;; * qiymatlari
(50)  ;;  %1 - fayl nomi
(51)  ;;  %2 - fayli ochish maqsadlari
(52)  %macro faylni_ochish 2
(53)
(54)      jmp  %%chetlash
(55)      %%fayl_ochish_maq   db   %2,0
(56)
(57)  %%chetlash:
(58)
(59)      tizim_xizmati FAYLNI_OCHISH_TCH, %1, %%fayl_ochish_maq
(60)
(61)  %endmacro
(62)  ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
(63)
(64)  ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
(65)  ;; * O'qish qism dasturi
(66)  ;;
(67)  ;; * qiymatlari
(68)  ;;  %1 - fayl
(69)  ;;  %2 - o'qilgan baytlarni saqlash manzili
(70)  ;;  %3 - necha bayt o'qilishi
(71)  %macro fayldan_o_qish 3
(72)
(73)      tizim_xizmati O_QISH_TCH, %2, 1, %3, %1
(74)
(75)  %endmacro
(76)  ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
(77)
(78)  ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
(79)  ;; * Yozish qism dasturi
(80)  ;;
(81)  ;; * qiymatlari
(82)  ;;  %1 - fayl
```

```
(83)  ;;  %2 - yoziladigan baytlarning xotiradagi manzili
(84)  ;;  %3 - necha bayt yozilishi
(85)  %macro faylga_yozish 3
(86)
(87)      tizim_xizmati YOZISH_TCH, %2, 1, %3, %1
(88)
(89)  %endmacro
(90)  ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
(91)
(92)  ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
(93)  ;; * Faylni yopish qism dasturi
(94)  ;;
(95)  ;; * qiymatlari
(96)  ;;  %1 - fayl tartib raqami
(97)  %macro faylni_yopish 1
(98)
(99)      tizim_xizmati FAYLNI_YOPISH_TCH, %1
(100)
(101) %endmacro
(102) ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
(103)
(104) ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
(105) ;; * Dasturni yakunlash qism dasturi
(106) ;;
(107) ;; * qiymatlari
(108) ;;  %1 - Dastur qanday tugaganligi haqida xabar
(109) %macro dasturni_yakunlash 1
(110)
(111)     tizim_xizmati DASTURNI_YAKUNLASH_TCH, %1
(112)
(113) %endmacro
(114) ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
(115)
(116) %endif
```

nusxa.asm

```
(1)   %include "c_qism_dasturlari.inc"
(2)   %include "nasm-io.inc"
(3)
(4)   section  .bss
(5)   yoziladigan_fayl_tr     resd 1  ;; Yoziladigan fayl tartib raqami
(6)   o_qiladigan_fayl_tr     resd 1  ;; O'qiladigan fayl tartib raqami
(7)
(8)   %define  fayl_nomi_uzn  200
(9)   o_qiladigan_fayl_nomi   resb fayl_nomi_uzn
(10)  yoziladigan_fayl_nomi   resb fayl_nomi_uzn
(11)
(12)  %define  qator_uzunligi 100       ;; 100 baytdan o'qiymiz
(13)  qator    resb qator_uzunligi
(14)
(15)  n   resd 1
(16)
(17)  section  .text
(18)  tizim_global  main
```

```
(19)
(20)  main:
(21)      chop_et    `Nusxa olinadigan fayl nomi: `
(22)      qabul_qil  `%s`, o_qiladigan_fayl_nomi
(23)
(24)      chop_et    `Yangi fayl nomi: `
(25)      qabul_qil  `%s`, yoziladigan_fayl_nomi
(26)
(27)      faylni_ochish o_qiladigan_fayl_nomi, O_QISH_MAQ
(28)      cmp  eax , 0
(29)      ja   o_qiladigan_fayl_ochildi
(30)      chop_et   `%s      faylini      ochib      bo'lmadi      \n`,
          o_qiladigan_fayl_nomi
(31)      jmp  yakun
(32)
(33)  o_qiladigan_fayl_ochildi:
(34)      mov  [o_qiladigan_fayl_tr] , eax
(35)
(36)      faylni_ochish yoziladigan_fayl_nomi, YOZISH_MAQ
(37)      cmp  eax , 0
(38)      ja   yoziladigan_fayl_ochildi
(39)      chop_et   `%s      faylini      ochib      bo'lmadi      \n`,
          o_qiladigan_fayl_nomi
(40)      jmp  yakun
(41)
(42)  yoziladigan_fayl_ochildi:
(43)      mov  [yoziladigan_fayl_tr] , eax
(44)
(45)  ko_chirish:
(46)      fayldan_o_qish [o_qiladigan_fayl_tr], qator, qator_uzunligi
(47)      mov  [n] , eax
(48)      faylga_yozish  [yoziladigan_fayl_tr], qator, eax
(49)
(50)      cmp  dword[n] , 0
(51)      jnz  ko_chirish
(52)
(53)      faylni_yopish [o_qiladigan_fayl_tr]
(54)      faylni_yopish [yoziladigan_fayl_tr]
(55)
(56)  yakun:
(57)      dasturni_yakunlash 0
```

Ushbu nusxa olish dasturi barcha operatsion tizimlarda birdek ishlaydi.

12.4. Amaliy dastur: Lotindan kirillga

Yakuniy amaliy dastur sifatida «Makro vositalar» bobida ko'rib chiqilgan lotin alifbosidagi harflarni kirillchaga o'giradigan dasturni tuzamiz. Haqiqiy matn ma'lum faylda joylashgan bo'ladi. Dastur faylni o'qish uchun ochib u yerdagi barcha lotin harflarini kirillchaga o'girib natijani yangi faylga yozadi. Dasturning aynan o'giruvchi qismi avval bo'lgani kabi ko'p satrli makro sifatida emas, balki qism dasturi ko'rinishida tuziladi. Ushbu qism dasturi o'girilishi kerak bo'lgan baytlar joylashgan qolipni va uning uzunligini qabul qiladi. Natijani ham yana o'sha qolipga yozadi.

Dasturda oldin yaratilgan *c_qism_dasturlari.inc* boshlang'ich faylidan foydalanamiz. Asosiy dastur kodi ham oldingi namunadagi koddan unchalik farq qilmaydi. Faqat fayldan o'qilgan baytlarni yangi faylga yozishdan oldin o'giruvchi qism dasturiga jo'natiladi.

23-namuna: lotindan_kirillga_faylni.asm

```
(1)   %include "c_qism_dasturlari.inc"
(2)   %include "nasm-io.inc"
(3)
(4)   section  .bss
(5)   yoziladigan_fayl_tr     resd 1
(6)   o_qiladigan_fayl_tr     resd 1
(7)
(8)   %define  fayl_nomi_uzn  200
(9)   o_qi_fayl_nomi     resb fayl_nomi_uzn
(10)  yoz_fayl_nomi resb fayl_nomi_uzn
(11)
(12)  %define  qator_uzunligi 100
(13)  qator    resb qator_uzunligi
(14)
(15)  n   resd 1
(16)
(17)  section  .text
(18)  tizim_global  main
(19)
(20)  main:
(21)      chop_et    `O'girilishi kerak bo'lgan fayl nomi: `
(22)      qabul_qil  `%s`, o_qi_fayl_nomi
(23)
(24)      chop_et    `Yangi fayl nomi: `
(25)      qabul_qil  `%s`, yoz_fayl_nomi
(26)
(27)      faylni_ochish o_qi_fayl_nomi, O_QISH_MAQ
(28)      cmp  eax , 0
(29)      ja   .o_qiladigan_fayl_ochildi
(30)      chop_et   `%s faylini ochib bo'lmadi \n`, o_qi_fayl_nomi
(31)      jmp  .yakun
(32)
(33)  .o_qiladigan_fayl_ochildi:
(34)      mov  [o_qiladigan_fayl_tr] , eax
(35)
(36)      faylni_ochish yoz_fayl_nomi, YOZISH_MAQ
(37)      cmp  eax , 0
(38)      ja   .yoziladigan_fayl_ochildi
(39)      chop_et   `%s faylini ochib bo'lmadi \n`, yoz_fayl_nomi
(40)      jmp  .yakun
(41)
(42)  .yoziladigan_fayl_ochildi:
(43)      mov  [yoziladigan_fayl_tr] , eax
(44)
(45)  .ko_chirish:
(46)      fayldan_o_qish [o_qiladigan_fayl_tr],qator, qator_uzunligi
(47)      mov  [n] , eax
(48)
(49)      push dword [n]
```

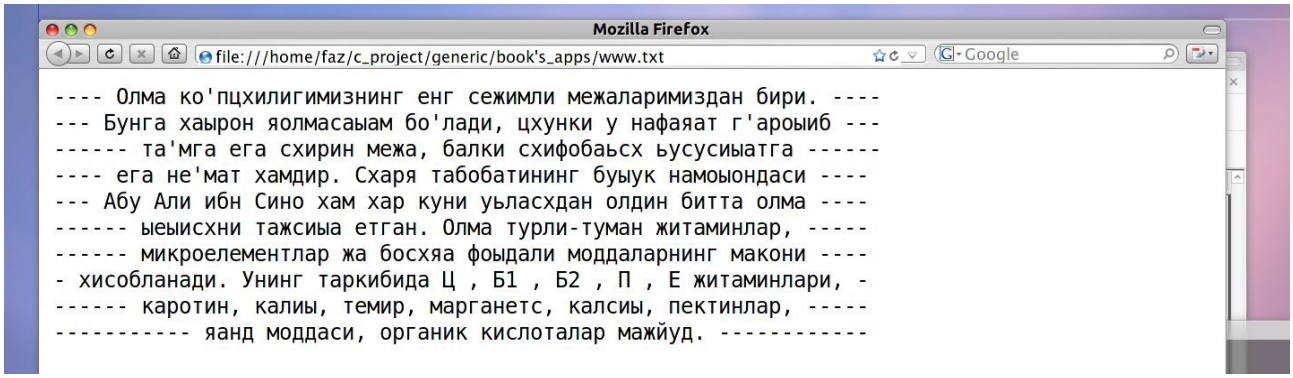
```
(50)      push dword qator
(51)      call o_girish
(52)      add  esp , 2 * 4
(53)
(54)      faylga_yozish  [yoziladigan_fayl_tr], qator, [n]
(55)
(56)      cmp  dword[n] , 0
(57)      jnz  .ko_chirish
(58)
(59)      faylni_yopish [o_qiladigan_fayl_tr]
(60)      faylni_yopish [yoziladigan_fayl_tr]
(61)
(62)  .yakun:
(63)      dasturni_yakunlash 0
(64)
(65)  ;; o_girish qism dasturi.
(66)  ;;
(67)  ;; Maqsad:
(68)  ;;  qolipdagi lotin harflarini mos kirillchasiga o'girish.
(69)  ;;  KOE8-R raqamlash turidan foydalaniladi.
(70)  ;;  C dagi e'lon qilinishi:
(71)  ;;  int o_girish(char * qolip, int uzunlik);
(72)  ;;
(73)  ;; Qiymatlar:
(74)  ;;  qolip - qolip manzili
(75)  ;;  uzunlik - qolip uzunligi
(76)  ;;
(77)
(78)  %define  qolip     [ebp+8]
(79)  %define  n         [ebp+12]
(80)
(81)  %define  lotin_a   97
(82)  %define  lotin_z   122
(83)
(84)  %define  lotin_A   65
(85)  %define  lotin_Z   90
(86)
(87)  %define  kichik_harflar_farqi     96
(88)  %define  bosh_harflar_farqi       160
(89)
(90)  o_girish:
(91)      enter     0 , 0
(92)      push edi
(93)      push esi
(94)      push ecx
(95)
(96)      mov  edi , qolip
(97)      mov  esi , qolip
(98)      mov  ecx , n
(99)
(100)     test ecx , ecx
(101)     jz   .yakun
(102)
(103) .lp_boshi:
```

```
(104)     lodsb
(105)     cmp  al , lotin_a
(106)     jb   .kichik_harf_emas        ;;%if (al < lotin_a)
(107)     cmp  al , lotin_z
(108)     ja   .kichik_harf_emas        ;;%if (al > lotin_z)
(109)     add  al , kichik_harflar_farqi
(110)
(111)     jmp  .bosh_harf_emas
(112)
(113) .kichik_harf_emas:
(114)     cmp  al , lotin_A
(115)     jb   .bosh_harf_emas              ;;%if (al < lotin_A)
(116)     cmp  al , lotin_Z
(117)     ja   .bosh_harf_emas              ;;%if (al > lotin_Z)
(118)     add  al , bosh_harflar_farqi
(119)
(120) .bosh_harf_emas:
(121)     stosb
(122)     loop .lp_boshi
(123)
(124) .yakun:
(125)     pop  ecx
(126)     pop  esi
(127)     pop  edi
(128)     leave
(129) ret
```

O'girish qanday amalga oshirilishi, qaysi raqamlash turidan foydalanilgani va dastur natijasini qanday ko'rish «Makro vositalar» bobida yoritilgan. Misol uchun bizda quyidagi matn saqlanayotgan fayl bo'lsin:

```
---- Olma ko'pchiligimizning eng sevimli mevalarimizdan biri. ----
--- Bunga hayron qolmasayam bo'ladi, chunki u nafaqat g'aroyib ---
------ ta'mga ega shirin meva, balki shifobaxsh xususiyatga ------
---- ega ne'mat hamdir. Sharq tabobatining buyuk namoyondasi -----
--- Abu Ali ibn Sino ham har kuni uxlashdan oldin bitta olma -----
------ yeyishni tavsiya etgan. Olma turli-tuman vitaminlar, ------
------ mikroelementlar va boshqa foydali moddalarning makoni -----
- hisoblanadi. Uning tarkibida C , B1 , B2 , P , E vitaminlari, --
------ karotin, kaliy, temir, marganets, kalsiy, pektinlar, ------
---------- qand moddasi, organik kislotalar mavjud. ------------
```

O'girilgan matn esa web browserda 62-rasmda tasvirlanganidek bo'ladi.



62-rasm.

Ilovalar

Ilova A. Buyruqlar qatorida ishlash

Buyruqlar qatori dasturlashda kerak bo'ladigan asosiy anjomlaridan biri hisoblanib, u ingliz tilida asosan `terminal` yoki `console` deb yuritiladi. Terminal ham o'zi dastur bo'lib, u bizga tasvirsiz ishlaydigan dasturlar bilan ishlash imkoniyatini beradi. Dasturlashda ishlatiladigan kompilyator, ulovchi yoki dastur tahlilchilari kabi barcha anjomlar tasvirsiz tarzda ishlaydi va shuning uchun ham dasturlash jarayonida buyruqlar qatoriga ko'p murojaat qilishga to'g'ri keladi.

Har bir operatsion tizimning o'z buyruqlar to'plami bo'lib, ushbu buyruqlar terminal orqali ishlatiladi. Tizim buyruqlari har xil vazifalarni amalga oshirishga qaratilgan bo'lib, ular orqali, masalan, fayl yaratish, direktoriyadagi fayllar ro'yxatini ko'rish, faylni bir joydan ikkinchi joyga ko'chirishga o'xshagan ishlarni bajarish mumkin. Umuman olganda operatsion tizim buyruqlarining ko'pchiligini tasvirli dasturlar orqali ham ishga tushirish iloji bor. Ammo har bir dasturchi buyruqlar qatorida ishlashni o'rganishi lozim. Shuning uchun kitobning ushbu ilovasi tizimdagi buyruqlar qatoridagi buyruqlarni yoritishga qaratilgan.

A.1. Buyruqlar umumiy ro'yxati

Quyidagi jadvalda eng asosiy buyruqlarning Linux va Windowsdagi nomlari va qisqacha ma'nosi keltirilgan.

Linux	Windows	Ma'nosi
`MAN` *buyruq nomi*	`HELP` *buyruq nomi*	Buyruq haqida ma'lumot (yordam) olish.
`CD`	`CD`	Joriy direktoriyadan boshqasiga o'tish.
`PWD`	`CHDIR`	Joriy direktoriyaning mutlaq yo'lini chop etish
`CLEAR`	`CLS`	Terminalni chop etilgan belgilardan tozalash.
`CP`	`COPY`	Faylni bir direktoriyadan boshqasiga ko'chirish (nusxa olish).
`RM`	`DEL`	Faylni o'chirish.
`LS` yoki `DIR`	`DIR`	Ma'lum direktoriyadagi fayllar ro'yxatini chiqarish.
`ECHO`	`ECHO`	Tizim o'zgaruvchisining qiymatini chop etish.
`VIM` yoki boshqasi	`EDIT`	Matnli fayl muharriri.
`DIFF`	`FC`	Ikkita faylni baytma-bayt taqqoslash.
`MKDIR`	`MKDIR`	Yangi direktoriya yaratish.
`MV`	`MOVE`	Faylni qayta nomlash yoki uni bir direktoriyadan boshqasiga olib o'tish.
`CAT`	`TYPE`	Fayl ichidagi matnni ko'rish.
`UNAME`	`VER`	Operatsion tizim haqida ma'lumot olish.

Agar sizni boshqa buyruqlar ham qiziqtirsa, internetdagi kerakli manbalarga murojaat qiling. Quyida buyruqlar batafsilroq yoritiladi.

A.2. Linux tizim buyruqlari

Linuxda buyruqlar qatorini ishga tushirish uchun ekranning chap yuqori qismidagi *Applications* boshlovchisini sichqoncha yordamida bosamiz va pastga yoyilgan ro'yxatdagi *Accessories* bandiga kirib *Terminal* ni tanlaymiz (63-rasmga qarang).



63-rasm.

Shundan so'ng ekranda qora yoki oq oyna paydo bo'ladi va shu yerga o'zingiz xohlagan buyruqni yozib uni tasdiqlashingiz mumkin (64-rasmga qarang).



64-rasm.

Eslatma: Keltirilgan xatti-harakatlar GNOME foydalanuvchi muhitida tasvirlangan. Agar siz KDE yoki boshqa bir shunga o'xshash muhitidan foydalanayotgan bo'lsangiz, u yerda terminalni ishga tushirish bir oz boshqacharoq bo'lishi mumkin.

Yanada tushunarliroq bo'lishi uchun eng ko'p foydalanadigan buyruqlar haqida batafsilroq to'xtalamiz.

`MAN` (**Man**ual – Qo'llanma) buyrug'i berilgan boshqa istalgan birorta buyruq haqidagi barcha ma'lumotlarni qo'llanma sifatida e'tiboringizga havola etadi. Masalan:

```
$ man ls
```

`LS` buyrug'i haqidagi barcha ma'lumotlar taqdim etiladi.

`LS` (**Lis**t – Ro'yxat) Joriy yoki berilgan direktoriyadagi barcha fayl va direktoriyalar ro'yxatini namoyish etadi. Masalan:

```
$ ls
dasturlar  hello.asm hujjatlar  mening_matnim.txt
```

Demak, bizning holatimizda buyruq berilgan direktoriyada to'rtta fayl bor ekan. Agar `LS` buyrug'iga qo'shimcha tarzda nisbiy yoki mutlaq direktoriya yo'li berilsa, u aynan berilgan direktoriyadagi fayllar ro'yxatini beradi. Agar fayllar haqida to'liqroq ma'lumot kerak bo'lsa, `LS` buyrug'iga `-l` kaliti qo'shib beriladi. Masalan:

```
$ ls -l dasturlar/
total 40764
drwxr-xr-x 2 faz faz     4096 Mar  4 12:53 bluefish
-rwxrwxrwx 1 faz faz   333772 Dec 23 18:34 cairo_clock_0.3_i386.deb
-rw-r--r-- 1 faz faz  1033200 Mar  4 12:53 nasm_2.08.01-1_i386.deb
-rwxrwxrwx 1 faz faz   506198 Oct 25  2009 rar_3.7b1-2_i386.deb
```

`LS` dan tashqari `DIR` buyrug'i ham mavjud bo'lib, u `LS` bilan bir xil ishlaydi.

Eslatma: Barcha buyruqalarga beriladigan faylning yo'li nisbiy yoki mutlaq bo'lishi mumkin, ya'ni fayl yo'lining qanday tarzda berilishiga hech qanday cheklovlar yo'q.

`PWD` (**P**rint Name of Current/**W**orking **D**irectory – Ishlayotgan/Joriy Direktoriya Mutlaq Yo'lini Namoyish Etish) Ishlayotgan direktoriyangizni mutlaq yo'lini taqdim etadi. Masalan:

```
$ pwd
/home/faz/dasturlar
```

`CD` (**C**hange **D**irectory – Direktoriyani O'zgartirish) Ishlayotgan direktoriyadan berilgan boshqa direktoriyaga o'tishni amalga oshiradi. Masalan, deylik siz A direktoriyada turibsiz, uning ichidagi B direktoriyaga quyidacha o'tiladi:

```
$ cd B/
```

Bitta direktoriya orqaga quyidagicha qaytiladi:

```
$ cd ../
```

Ushbu buyruqqa qiymat sifatida o'tish kerak bo'lgan direktoriyaning nisbiy yoki mutlaq yo'li berilishi mumkin.

`CP` (**Co**p**y** – Nusxa olish) Berilgan fayldan nusxa olib, nusxani ko'rsatilgan yerga ko'chiradi. Masalan, *dasturlar* direktoriyasidagi *nasm_2.08.01-1_i386.deb* fayldan *hujjatlar* direktoriyasiga nusxa olish kerak bo'lsa, quyidagi buyruq beriladi:

```
$ cp dasturlar/nasm_2.08.01-1_i386.deb hujjatlar/
```

Bu ishlar *dasturlar* direktoriyasi joylashgan direktorida turib bajarilayapti!

`MV` (**Mov**e – Ko'chirish) Berilgan faylni berilgan direktoriyaga ko'chiradi yoki fayl nomini o'sha yerning o'zida berilgan yangi nomga o'zgartiradi. Masalan, *hello.asm* faylini o'sha direktoriyadagi dasturlar direktoriyasiga o'tkazish uchun quyidagi kifoya:

```
$ mv hello.asm dasturlar/
```

Shundan so'ng ishlayotgan direktoriyangizda bu fayl bo'lmaydi. Buni `LS` buyrug'i orqali tekshirib ko'ramiz:

```
$ ls
dasturlar  hujjatlar  mening_matnim.txt
```

Agar *dasturlar* direktoriyasidagi fayllar ro'yxatini qarasak, bizning ko'chirgan faylimiz o'sha yerda ekanligining guvohi bo'lamiz:

```
$ ls dasturlar/
…  hello.asm  …
```

Agar ikkinchi berilgan qiymat ham fayl nomi bo'lsa, birinchi fayl nomi ikkinchisiga o'zgartiriladi. Masalan, *hello.asm* faylini nomini *salom.asm* ga o'zgartirish uchun quyidagini yozamiz:

```
$ mv hello.asm salom.asm
```

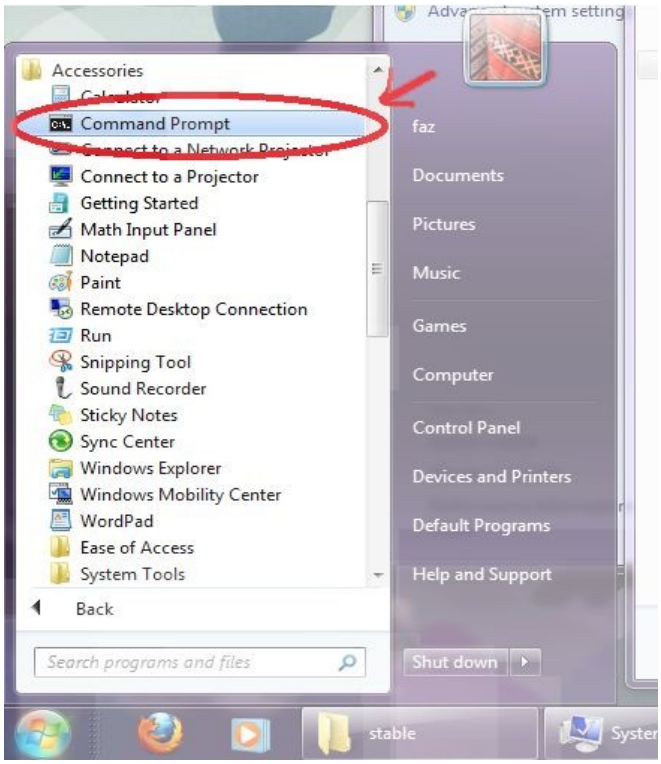
Linuxda tizim buyruqlaridan tashqari o'rnatilgan foydalanuvchi dasturlarini ham terminaldan buyruq sifatida ishga tushirish mumkin. Chunki Linux har doim yangi o'rnatilgan dasturni o'z buyruqlar to'plamiga qo'shib qo'yadi. Masalan, NASM kompilyatori tizim buyrug'iga kirmasada, uni bemalol buyruqlar qatoridan ishga tushirish mumkin:

```
$ nasm -f elf salom.asm
```

A.2. Windows tizim buyruqlari

Windowsda buyruqlar qatorini ishga tushirish uchun ekranning chap pastgi qismidagi boshlovchi yordamida *Start→All Programs→Accessories→Command Promt* ni tanlaymiz (65-rasmga qarang). Shundan so'ng 66-rasmda tasvirlangani kabi ekranda terminal oynasi paydo bo'ladi.

«Buyruqlar umumiy ro'yxati» ilovasida Windowsning eng muhim buyruqlariga ta'riflar keltirib o'tilgan. Umuman olganda Windows va Linux tizim buyruqlarining nomlanishidagi farqni hisobga olmasak, ularning ishlash tarzi deyarli bir xil. Shuning uchun «Linux tizim buyruqlari» ilovasida ta'kidlangan barcha mulohazalar Windows buyruqlariga ham amal qiladi. Shuning uchun quyida buyruqlar ijrosida faqat misollar ko'rib chiqiladi.



65-rasm.

Quyidagi buyruq `DIR` buyrug'i haqidagi barcha ma'lumotlarni qo'llanma sifatida terminalda chop etadi:

```
> help dir
```

C:\Documents and Settings direktoriyasidagi barcha fayl va direktoriyalar ro'yxatini quyidagicha ko'rish mumkin:

```
> dir C:\Documents and Settings
Volume in drive C is
Volume Serial Number is 0000-0000

Directory of C:\Documents and Settings

29/06/2011     21:30  <DIR>         .
29/06/2011     21:30  <DIR>         ..
29/06/2011     21:30  <DIR>         All Users
29/06/2011     21:30  <DIR>         Default User
29/06/2011     21:30  <DIR>         faz
29/06/2011     21:30  <DIR>         LocalService
29/06/2011     21:30  <DIR>         NetworkService
```

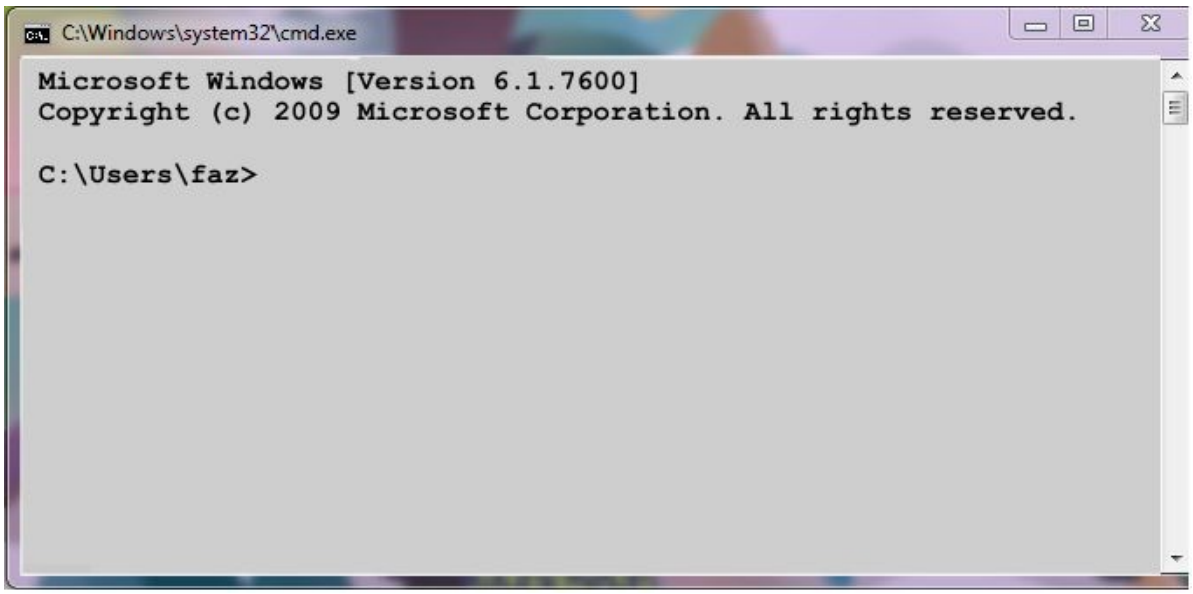
Ishlayotgan direktoriyangizning mutlaq yo'lini quyidagicha ko'rish mumkin:

```
> chdir
C:\Documents and Settings
```

Windowsda o'rnatilgan foydalanuvchi dasturlarining buyruqlarini bir-ikkita amallarni bajarmay turib terminaldan ishga tushirib bo'lmaydi. Ammo agar o'rnatilgan dastur buyruqlar qatorida ishlatishga mo'ljallangan anjomlarga ega bo'lsa, u o'z terminalini taqdim etishi mumkin. Gap shundaki, dastur tizim terminalini ishga tushirgach ba'zi sozlashlarni amalga oshiradi va shuning natijasida Windows buyruqlariga foydalanuvchi dasturi buyruqlari ham qo'shiladi.

Masalan, Microsoft Visual Studio dasturlash muhiti o'zining CL deb nomlanuvchi C kompilyatoriga ega. Lekin yuqorida ko'rsatilgan usul orqali ishga tushirilgan buyruqlar qatorida ushbu kompilyatorni ishlatib bo'lmaydi. CL dan ko'ngildagidek foydalanish uchun albatta Visual Studioning o'z terminalini ishga solish kerak.

Biror dasturni Windows buyruqlar qatoridan ishga tushirib bo'ladigan qilib sozlash tizimning Path o'zgaruvchisi orqali bajariladi. Ushbu o'zgaruvchi qiymat sifatida doim buyruqlar joylashgan direktoriyalarning mutlaq yo'llarini o'zida saqlaydi. Istalgan terminaldan turib dasturni buyruq sifatida ishga tushirish uchun o'sha dastur joylashgan direktoriyaning mutlaq yo'lini Path o'zgaruvchisi qiymatiga qo'shish kifoya.



66-rasm.

Ilova B. Nasmni o'rnatish

Kitobda beriladigan mashqlarni yechishda kerak bo'ladigan anjomlarning asosiysi NASM kompilyatori bo'lib, ushbu ilovada uni qanday o'rnatish yoritiladi.

NASM erkin tarqatiladigan dastur bo'lib, uning barcha operatsion tizimlarga mo'ljallangan eng oxirgi turkumini quyidagi saytlardan yozib olish mumkin:

- www.nasm.us – Windows, shuningdek Fedora, SUSE, Mandrive, Cent OS, Doppix va Red Hat uchun. Ushbu NASM ning rasmiy saytida kompilyatorning ochiq kodi ham mavjud.
- www.packages.debian.org – Debian, Linux Mint va Ubuntu uchun.
- www.packages.ubuntu.com – Ubuntu uchun
- www.pkgs.org – Yuqoridagilar va barcha mashhur Linux turkumlari uchun.

Bundan tashqari ushbu kitobga bag'ishlangan assembler.zn.uz saytidan ham kompilyatorning Windows va Linux tizimlariga mo'ljallangan nusxalarini yozib olish mumkin.

B.1. NASMni Linuxda o'rnatish

Avvalam bor dasturni Linuxda o'rnatish uchun sizda «Ildiz», ya'ni root imkoniyatlari bo'lishi kerak, aks holda dasturni o'rnatib bo'lmaydi. Buning uchun tizimga root foydalanuvchisi sifatida kirish kerak. Agar oddiy foydalanuvchi bo'lib ro'yxatdan o'tilgan bo'lsa, buyruqlar qatorida quyidagi buyruqni yozib root maxfiy so'zini kiriting:

```
$ su
Password:
```

Eslatma: Buyruqlar qatorida oddiy foydalanuvchi uchun dollar (`$`), root uchun esa panjara (`#`) belgisi ko'rsatib turiladi.

NASMni o'rnatish usuli aynan qaysi Linux turkumi ishlatilayotganiga qarab farqlanadi.

B.1.1. Debian yoki Ubuntu foydalanuvchilari uchun

Debian va Ubuntu tizimlarida dastur o'rnatishning turli xil yo'llari mavjud bo'lib, quyida eng oson usullar keltirilgan.

a) Agar kompyuteringiz internetga ulangan bo'lsa yoki tizim disklariga ega bo'lsangiz, quyidagi buyruq NASMni o'rnatadi:

```
# apt-get install nasm
```

b) Debian va Ubuntu tizimlarida o'z-o'zini o'rnatuvchi dasturlar *deb* kengaytmali bo'lib, bunday dastur fayllari `DPKG` buyrug'i orqali o'rnatiladi. O'rnatuvchi dastur fayliga ega bo'lgach, o'sha fayl joylashgan direktoriyaga o'tib, NASM kompilyatori quyidagicha o'rnatiladi:

```
# dpkg -i nasm.xxx-xx.deb
```

B.1.2. Fedora, SUSE, Mandrive, Cent OS, Doppix yoki Red Hat foydalanuvchilari uchun

Fedora, SUSE, Mandrive, Cent OS, Doppix va Red Hat tizimlarida dastur o'rnatishning turli xil yo'llari mavjud bo'lib, quyida eng oson usullar keltirilgan.

a) Agar kompyuteringiz internetga ulangan bo'lsa yoki tizim disklariga ega bo'lsangiz, quyidagi buyruq NASMni o'rnatadi:

```
# yum install nasm
```

b) Fedora, SUSE, Mandrive, Cent OS, Doppix va Red Hat tizimlarida o'z-o'zini o'rnatuvchi dasturlar *rpm* kengaytmali bo'lib, bunday dastur fayllari `RPM` buyrug'i orqali o'rnatiladi. O'rnatuvchi dastur fayliga ega bo'lgach, o'sha fayl joylashgan direktoriyaga o'tib, NASM kompilyatori quyidagicha o'rnatiladi:

```
# rpm –ivh nasm.xxx–xx.rpm
```

B.1.3. Umumiy usul

Yuqorida ko'rsatilgan usullardan tashqari Linuxda yana bir dasturni o'rnatish yo'li bo'lib, unda tizim turkimidan qat'i nazar o'rnatish bir xil kechadi. Ma'lumki, NASM kompilyatori LGPL ruxsatnomasi ostida erkin tarqatiladi. Bundan kelib chiqadiki, NASMning dastur kodini ochiq holda qo'lga kiritish mumkin. Dasturni o'rnatishning umumiy usuli shundan iboratki, NASM dastur kodini C kompilyatori orqali oldin yig'ib, so'ng vujudga kelgan bajaruvchi fayllarni kerakli direktoriyalarga joylashtiriladi.

NASM kompilyator kodi asosan *zip* kengaytmali siqilgan faylda bo'ladi. Birinchi navbatda siqilgan fayl ochiladi. Barcha tizimlarda siqilgan fayllarni ochishga mo'ljallangan tasvirli dasturlar bo'ladi, ammo bu ishni buyruqlar qatorida ham quyidagicha bajarish mumkin:

```
$ unzip nasm.xxx–xx.zip
```

Siqilgan fayl ochilgach *nasm.xxx-xx*[1] ismli yangi direktoriya paydo bo'ladi va ushbu direktoriyaga kiriladi:

```
$ cd nasm.xxx–xx
```

Bu direktoriyada kompilyator kodi joylashgan fayllar bo'lib, ular ichida maxsus *configure,* ya'ni so'zlash fayli berilgan. Bu faylning vazifasi NASMni yig'ishda kerak bo'ladigan qo'shimcha dasturlar tizimda o'rnatilgan yoki yo'qligini aniqlashdir. Demak, birinchi bo'lib *configure* fayli ishga tushiriladi:

```
$ ./configure
```

Shundan so'ng agar biror muammo yuzaga kelmasa, natija sifatida joriy direktoriyada *Makefile* fayli yaratiladi. *Makefile* ning vazifasi NASM dastur kodini maxsus kompilyatorlar orqali yig'ishdir. Ushbu fayl quyidagicha ishga tushiriladi:

```
$ make
```

Agar NASM dastur kodi muvaffaqiyatli yig'ilsa, endi uni quyidagicha o'rnatsa bo'ladi:

```
# make install
```

[1] Bu yerda *xxx-xx* o'rnida dastur turkumning tartib raqamlari bo'ladi.

Ushbu buyruq */usr/bin* direktoriyasida *nasm* va *ndisasm* ismli ikkilik fayllarni yaratadi. Mana NASM kompilyatorini o'rnatish jarayonini batafsil ko'rib chiqdik. Endi terminalga kirganda istalgan vaqtda va istalgan direktoriyadan turib NASM kompilyatorini buyruq sifatida ishga solish mumkin. Masalan *mening_dasturim.asm*[1] faylidagi assemblerda tuzilgan dasturimizni NASM orqali yig'amiz:

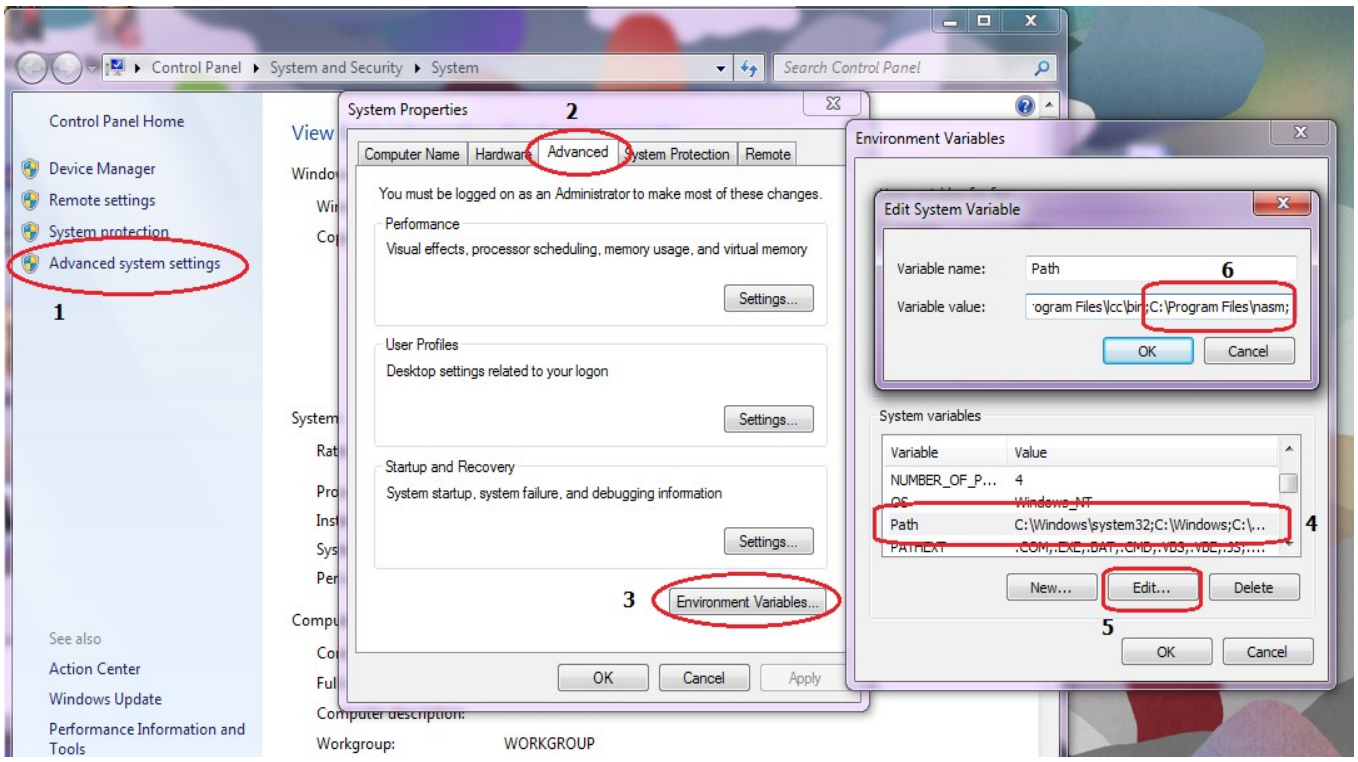
```
$ nasm -f elf mening_dasturim.asm
```

B.2. NASMni Windowsda o'rnatish

Windowsda NASMni o'rnatish juda oson va bir necha xil usullari bor.

B.2.1. Birinchi usul

Agar o'z-o'zini o'rnatuvchi *nasm-x.xx.xx-installer.exe* faylini olgan bo'lsangiz, uni ustiga sichqoncha yordamida ikki marta bosib ishga tushirasiz va beriladigan bir nechta savollarga javob berib o'rnatishni amalga oshirasiz. O'rnatishdan so'ng NASM o'zining buyruqlar qatorini taqdim etadi va uni *Start→All Programs→Netwide Assembler→Command Line* ni tanlash orqali ishga tushirish mumkin. NASMni tizim yoki boshqa dasturlar taqdim etadigan terminallardan ishga tushirish uchun esa u o'rnatildan direktoriyaning mutlaq yo'lini `Path` tizim o'zgaruvchisi qiymatiga qo'shamiz.



67-rasm.

`Path` o'zgaruvchisining qiymatiga yangi mutlaq yo'l qo'shish:

1. *Start→Control Panel→System* tanlanadi;
2. Paydo bo'lgan oynaning *Advanced* bo'limi tanlanadi;
3. U yerdagi *Environment Variables* tugmasi bosiladi;

[1] Assemler dasturlari joylashgan fayllar kengaytmasi *asm* bo'ladi.

4. Paydo bo'lgan oynaning *System variables* qismidan `Path` bilan boshlanadigan qator tanlanadi;
5. Pastdagi *Edit...* tugmasi bosiladi;
6. Paydo bo'lgan yangi oynaning *Variable value* qatorida joylashgan `Path` ning oldingi qiymati davomidan nuqta vergul (`;`) belgisini qo'ygan holda yangi mutlaq yo'l qo'shiladi. Masalan, NASM *C:\Program files\nasm* direktoriyasida o'rnatilgan bo'lsa, vaziyat 67-rasmda tasvirlanganidek ko'rinish oladi.

Eslatma: NASM o'rnatilgan direktoriya deganda *nasm.exe* va *ndisasm.exe* fayllari joylashgan direktoriya nazarda tutiladi. Umuman olganda ma'lum dastur o'rnatilgan direktoriya deganda o'sha dasturning bajaruvchi *exe* fayllari joylashgan direktoriya tushuniladi.

Mana NASM o'rnatildi. Endi uni xohlagan terminaldan ishga tushirish mumkin:

```
> nasm –f win32 mening_dasturim.asm
```

B.2.2. Ikkinchi usul

Agar *nasm.exe* va *ndisasm.exe* fayllarini o'z ichiga olgan *nasm-x.xx.xx-win32.zip* siqilgan faylini yozib olgan bo'lsangiz, uni ichidagi direktoriya shunchaki ochiladi va o'rnatilishi kerak bo'lgan joyga ko'chiriladi, masalan, *C:\Program files* ga. Shundan so'ng NASM o'rnatilgan direktoriyaning mutlaq yo'li tizimning `Path` o'zgaruvchisi qiymatiga qo'shiladi. Buni qanday amalga oshirish «Birinchi usul» ilovasida keltirilgan.

B.2.3. Umumiy usul

Agar NASM dastur kodi joylashgan *nasm.xxx-xx.zip* siqilgan faylni olgan bo'lsangiz, unda kodni yig'ish uchun Microsoft Visual Studio kompilyatori kerak bo'ladi. Visual Studio terminali *Start→All Programs→Visual Studio→Visual C++ Command Shell* ni tanlash orqali ishga tushiriladi. So'ng siqilgan fayl ochilib, vujudga kelgan yangi direktoriyaga kiriladi. NASM dastur kodi esa quyidagi buyruq orqali yig'iladi:

```
\> nmake /f Mkfiles/msvc.mak
```

Agar hammasi muvaffaqiyatli yakunlansa, ushbu direktoriya mutlaq yo'li tizimning `Path` o'zgaruvchisi qiymatiga qo'shiladi. Buni qanday amalga oshirish «Birinchi usul» ilovasida keltirilgan.

Ilova C. C kompilyatorlari va obyekt fayllarni ular orqali ulash

Ushbu kitobda ba'zi sabablarga ko'ra NASM tomonidan yaratilgan obyekt fayllar C kompilyatori oqrali ulanadi. Shuning uchun mazkur ilovada mavjud C kompilyatorlari va ular orqali obyekt fayllarni qanday ulash ko'rib chiqiladi.

C.1 Linuxdagi mavjud C kompilyatorlari

Linux tizimi uchun mo'ljallangan bir qator C kompilyatorlari mavjud bo'lib, ular orasida GCC alohida ahamiyatga ega. GCC nafaqat Linuxda, balki barcha zamonaviy operatsion tizimlarda ishlaydi.

C.1.1. GCC kompilyatori

GCC kompilyatori ko'pgina Linux turkumlarida tizim bilan birga o'z-o'zidan o'rnatiladi, ya'ni tizim terminalidan uni bemalol ishga tushirish mumkin. Agar o'rnatilmagan bo'lsa, GCC ni tizim disklaridan yoki «NASMni o'rnatish» ilovasida keltirilgan internetdagi saytlardan yozib olib o'rnatish mumkin.

NASM tomonidan yaratilgan obyekt fayllar GCC orqali quyidagicha ulanadi:

```
$ gcc obyekt_fayl_1.o ... obyekt_fayl_n.o –o bajaruvchi_fayl
```

GCC ga `–o` kaliti orqali bajaruvchi fayl nomi beriladi. Agar ushbu kalit berilmasa natija *a.out* fayliga yoziladi.

«Qismli dasturlash» bobida ko'rib chiqiladigan qisman C da va qisman Assemblerda yozilgan dasturlar GCC orqali quyidagicha yig'iladi va ulanadi:

```
$ gcc c_dasturi.c nasm_yaratgan_obyekt_fayl.o –o bajaruvchi_fayl
```

C.2. Windowsdagi mavjud C kompilyatorlari

Windows tizimi uchun mo'ljallangan C kompilyatorlari juda ko'p sonli bo'lib, ularning asosiy qismi tijorat mahsulotlaridir. Quyida shunday komlyatorlarning bir qanchasi ko'rib chiqiladi.

C.2.1. GCC kompilyatori

Avval ta'kidlanganidek GCC kompilyatori barcha operatsion tizimlarga ishlaydi. Windows ham bundan mustasno emas. GCC kompilyatori Windowsda MinGW (**Min**imal **G**NU for **W**indows) deb ataladi va uni www.mingw.org rasmiy saytidan yozib olib o'rnatish mumkin. Agar sayt orqali o'rnatish qiyinlik qilsa, GCC ni o'rnatishning yana bir oson yo'li mavjud. Gap shundaki, Nokia firmasining Qt dasturlash muhiti GCC ni o'z ichiga oladi va uni o'rnatish orqali GCC ga ham ega bo'lish mumkin. Qt ham Microsoft Visual Studio ga o'xshagan dasturlash muhiti bo'lib, internetdagi www.qt.nokia.com/products/ rasmiy saytidan uning o'z-o'zini o'rnatuvchi *qt-sdk-win-opensource-xxxx.xx.exe* faylini erkin yozib olish mumkin. Bundan tashqari ushbu kitobga bag'ishlangan assembler.zn.uz saytidan ham Qt ni yozib olish mumkin.

Qt o'z terminaliga ega bo'lib, GCC ham ana shu terminaldan ishga tushiriladi. Qt terminali *Start→ All Programs→Qt Open Source→Command Promt* ni tanlash orqali ishga tushiriladi. NASM tomonidan yaratilgan obyekt fayllar Windowsda ham xuddi «Linuxdagi mavjud C kompilyatorlari» ilovasida ko'rsatilganidek ulanadi. Faqat Windowsda asosan `win32` andozasi qo'llanilgani sababli obyekt fayllar kengaymasi *obj* bo'ladi. Bajaruvchi fayl esa *exe* kengaytmali bo'ladi.

C.2.2. Microsoft Visual Studio

Windows uchun mo'ljallangan asosiy dasturlash muhiti Microsoft Visual Studio hisoblanib, u o'zining Visual C++ kompilyatoriga ega. NASM yaratgan obyekt fayllarni Microsoft Visual Studio orqali ham ulash mumkin. Albatta Microsoft Visual Studio anjomlarini faqat uning terminalida ishga tushirish mumkin. Demak, NASM tomonidan yaratilgan obyekt fayllar Visual C++ orqali quyidagicha ulanadi:

```
> cl dastur_1.obj … dastur_n.obj /link libcmt.lib /out:bajaruvchi.exe
```

`/link` kaliti orqali qo'shimcha ulanishi kerak bo'lgan kutubxonalar beriladi, `libcmt.lib` kutubxonasida esa C mezoniy qism dasturlari joylashgan. `/out` kaliti orqali natija sifatida yaratiladigan bajaruvchi fayl nomi beriladi.

«Qismli dasturlash» bobida ko'rib chiqiladigan qisman C da va qisman Assemblerda yozilgan dasturlar Visual C++ orqali quyidagicha yig'iladi va ulanadi:

```
> cl c_dasturi.c nasm_yar_ob_fayl.obj /link libcmt.lib /out:bajar.exe
```

Agar siz uchun buyruqlar qatorida ishlash noqulay bo'lsa, Visual Studioning o'zida ham assemblerda dasturlash iloji bor. Buning uchun quyidagi amallarni bajarish kerak.

1. Avvalam bor NASM o'rnatiladi va u o'rnatilgan direktoriyadagi *contrib/VSrules/* ga boriladi.
2. U yerdagi *nasm.rules* fayli *C:\Program Files\Microsoft Visual Studio 2008\VC\VCProjectDefaults* direktoriyasiga ko'chiriladi.
3. Visual Studio ishga tushiriladi.
4. Dasturdagi *Tools→Options→VC++ Directories* tanlanadi.
5. Yuqoridagi ro'yxatdan «Show Directories for Executables» bandi tanlanadi.
6. Pastdagi ro'yxatga NASM o'rnatilgan direktoriya mutlaq yo'li qo'shiladi. Masalan, *C:\Program Files\nasm*.
7. Assemblerda dasturlash uchun yangi loyiha ochiladi.
8. Sichqonchani loyiha nomi ustiga etib, uning o'ng tomoni bosiladi va pastga yoyilgan ro'yxatdagi «Custom Build Rules» bandi tanlanadi.
9. NASM joylashgan qator belgilanadi.
10. Dasturni C mezoniy qism dasturlari bilan birga ulash uchun 8-qadamni takrorlab, bu safar «Properties» bandi tanlanadi.
11. U yerda «Configuration Properties» bo'limi yoyilib, *Linker→Input* tanlanadi.
12. «Additional Dependies» ga *libcmt.lib* qo'shiladi va barcha harakatlar tasdiqlanadi.
10. Endi loyihaga *asm* kengaytmali fayllar qo'shib, dastur kodi yozilsa bo'ladi.
11. Dasturni yig'ish uchun «Build» tugmasi bosiladi.

Mazkur kitobda keltirilgan deyarli barcha amaliy dasturlar *nasm-io.inc* boshlang'ich faylidan foydalanadi. Shuning uchun Visual Studio loyihasi yaratilgan direktoriyada ushbu fayl ham

bo'lishi kerak. Demak, har safar yangi loyiha yaratilganida *nasm-io.inc* faylidan ham u yerga nusxa ko'chirilishi kerak. Har safar *nasm-io.inc* faylini ko'chirib yurmaslik uchun uni muqim bir joyda, masalan *C:\dasturlar* da saqlab, u yerning mutlaq yo'lini NASMga `-I` kaliti orqali ma'lum qilgan ma'qul. Buning uchun *C:\Program Files\Microsoft Visual Studio 2008\VC\VCProjectDefaults* direktoriyasiga ko'chirilgan *nasm.rules* faylini ochib, u yerdagi

```
CommandLine="nasm.exe -f win32 -Xvc [AllOptions] [AdditionalOptions]
[Inputs]"
```

satri quyidagiga o'zgartiriladi:

```
CommandLine="nasm.exe -f win32 -Xvc [AllOptions] [AdditionalOptions]
[Inputs] -IC:\dasturlar\".
```

Eslatma: *nasm-io.inc* fayli joylashgan direktoriyaning mutlaq yo'li tarkibida boshliq ( ) belgisi bo'lmasligi kerak.

C.2.3. LCC-WIN32 kompilyatori

Visual Studio yoki Qt ga o'xshagan dasturlar o'lcham jihatdan ancha katta bo'lib, ular o'rnida kichikroq va yengil ishlaydigan C kompilyatorlari mavjud. Shunday kompilyatorlardan biri LCC-WIN32 hisoblanib, u www.cs.virginia.edu/~lcc-win32/ rasmiy saytidan erkin yozib olinishi mumkin. Ushbu mahsulot kompilyatordan tashqari dasturlashga mo'ljallangan matn muharririga ham ega bo'lib, bularning hammasi 8 Mb dan oshmaydi.

LCC-WIN32 o'z terminaliga ega emas. Shunday ekan Windows tizimining terminalidan foydalanishga to'g'ri keladi. Buning uchun LCC-WIN32 ning bajaruvchi *exe* fayllari joylashgan direktoriya mutlaq yo'li `Path` o'zgaruvchisining qiymatiga qo'shiladi. Ushbu fayllar kompilyator o'rnatilgan direktoriya ichidagi *bin* direktoriyasida joylashgan. Masalan, LCC-WIN32 *C:\Program Files\lcc* da o'natilgan bo'lsa, `Path` ga *C:\Program Files\lcc\bin* qo'shiladi.

NASM tomonidan yaratilgan obyekt fayllar LCC-WIN32 orqali quyidagicha ulanadi:

```
> lc obyekt_fayl_1.obj … obyekt_fayl_n.obj -o bajaruvchi_fayl.exe
```

«Qismli dasturlash» bobida ko'rib chiqiladigan qisman C dasturlash tilida va qisman Assemblerda yozilgan dasturlar LCC-WIN32 orqali quyidagicha yig'iladi va ulanadi:

```
> lc c_dasturi.c nasm_yaratgan_obyekt_fayl.obj -o bajaruvchi.exe
```

Eslatma: LCC-WIN32 kompilyatorida `__attribute__` kalit so'zi ishlamaydi. Shuning uchun namunalardagi C dasturlaridan `__attribute__ ((cdecl))` ni olib tashlang.

Ilova D. *nasm-io.inc* fayli

nasm-io.inc boshlang'ich faylida C mezoniy kutubxonasidagi o'quv/yozuvni amalga oshiradigan qism dasturlarini assemblerdan turib chaqirishni osonlashtiradigan makrolar aniqlangan. Mazkur fayl dasturga `%INCLUDE` makro direktivasi orqali qo'shiladi. *nasm-io.inc* dagi `chop_et` va `qabul_qil` makrolari e'tiborga molik bo'lib, ular mos ravishda axborotni monitorga yozishga va tugmachalar taxtasidan o'qishga qaratilgan.

nasm-io.inc

```
(1)   %ifndef NASM_IO_INC
(2)   %define NASM_IO_INC
(3)
(4)   %define  elf32     1
(5)   %define  elf       elf32
(6)   %define  coff      2
(7)   %define  win32     3
(8)   %define  win64     4
(9)   %define bin        5
(10)  %define obj        6
(11)
(12)  %if __OUTPUT_FORMAT__ == elf
(13)  ;; Linux uchun
(14)
(15)  %define tizim_global global
(16)  %define tizim_extern extern
(17)
(18)  %else
(19)  ;; Qolgan barcha andozalar uchun
(20)
(21)  %macro tizim_global 1-*
(22)      %rep %0
(23)           global _%1
(24)           %define %1 _%1
(25)           %rotate 1
(26)      %endrep
(27)  %endmacro
(28)
(29)  %macro tizim_extern 1-*
(30)      %rep %0
(31)           extern _%1
(32)           %define %1 _%1
(33)           %rotate 1
(34)      %endrep
(35)  %endmacro
(36)
(37)  %endif
(38)
(39)  tizim_extern printf, scanf
(40)
(41)  ;;;;;;;;;;;;;;;;;;; Asosiy makrolar ;;;;;;;;;;;;;;;;;;;;;;;
(42)  ;; YOZUV
```

```
(43)  %macro chop_et 1-*
(44)
(45)  %ifnstr %1
(46)      %error ": chop_et ning birinchi qiymati qatorli\
(47)      o'zgarmas bo'lishi kerak"
(48)      jmp %%tamom
(49)  %endif
(50)
(51)  %assign  qiymatlar_soni %0-1
(52)
(53)      jmp %%chetlash
(54)  %%andoza db   %1, 0
(55)
(56)  %%chetlash:
(57)      pushf
(58)      pusha
(59)
(60)      %rep qiymatlar_soni
(61)           %rotate -1
(62)           push dword %1
(63)      %endrep
(64)      push dword %%andoza
(65)      call printf
(66)
(67)      add esp , %0 * 4
(68)      popa
(69)      popf
(70)
(71)  %%tamom:
(72)  %endmacro
(73)
(74)
(75)  ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
(76)  ;; O'QUV
(77)  %macro qabul_qil 1-*
(78)
(79)  %ifnstr %1
(80)      %error ": qabul_qil ning birinchi qiymati qatorli o'zgarmas\
(81)           bo'lishi kerak"
(82)
(83)      jmp %%tamom
(84)  %endif
(85)
(86)  %assign  qiymatlar_soni %0-1
(87)
(88)      jmp %%chetlash
(89)  %%andoza db   %1, 0
(90)
(91)  %%chetlash:
(92)      pushf
(93)      %rep qiymatlar_soni
(94)           %rotate -1
(95)           push dword %1
(96)      %endrep
```

```
(97)      push dword %%andoza
(98)      call scanf
(99)
(100)     add esp , %0 * 4
(101)     popf
(102) %%tamom:
(103) %endmacro
(104)
(105) %endif
```

Ilova E. Buyruqlar ro'yxati

Quyida NASM assemblerida mavjud buyruqlarning ro'yxati keltirilgan. Ammo ro'yxatda barcha buyruqlar berilmagan. Shuni aytish joizki, ushbu buyruqlar Intel protsessori buyruqlari bilan mutlaqo mos keladi. Jadvalning birinchi ustunida buyruq nomlari, ikkinchi ustunda ular qabul qiladigan qiymatlar va uchinchi ustunda esa buyruqning birinchi bo'lib qaysi protsessor rusumida paydo bo'lgani berilgan.

Jadvalda foydalanilgan atamalar ma'nosi:

o'zgarmas	O'zgarmas qiymat. Masalan: 8, `0xFF`, `'abc'`, ...
xotira	Xotiradagi o'zgaruvchi. Masalan: `[son]`, `[x]`, `y`, `z`, ...
reg	Istalgan registr. Masalan: EAX, BX, CL, EDI, ...
reg_xotira	Registr yoki xotiradagi o'zgaruvchi.
fpureg	Sonli qo'shma protsessor registri.

Atama so'ngida berilgan son obyekt o'lchamini bildiradi. O'lcham bitlarda berilgan. Masalan, `reg8` sakkiz bitli registrni anglatadi: AL, AH, CL, … . Ba'zi buyruqlar faqat ma'lum bir registr bilan ishlasa, registr nomi yaqqol berilgan.

Maxsus belgilar:

qiymat|aniqlovchi	Buyruqqa berilayotgan qiymat oldidan aniqlovchi qo'yilishi mumkinligini bildiradi. Masalan
	
	`CALL` `o'zgarmas32|near` `386`
	
	quyidagicha ishlatilishi mumkin:
	
	`call near qism_dastur`

xx	EFLAGS dagi ma'lum holatni tekshiruvchi harflarga almashtirilishi kerak. Masalan, `Jxx` buyrug'i aslida `JNB` yoki `JNZ` bo'lishi mumkin.

Buyruqlar ro'yxati

Buyruq	Qiymatlar	Protsessor rusumi
AAA		8086,NOLONG
AAD		8086,NOLONG
AAD	o'zgarmas	8086,NOLONG
AAM		8086,NOLONG
AAM	o'zgarmas	8086,NOLONG
AAS		8086,NOLONG
ADC	xotira , reg8	8086
ADC	reg8 , reg8	8086
ADC	xotira , reg16	8086
ADC	reg16 , reg16	8086
ADC	xotira , reg32	386
ADC	reg32 , reg32	386
ADC	reg8 , reg8	8086
ADC	reg16 , xotira	8086
ADC	reg16 , reg16	8086
ADC	reg32 , xotira	386
ADC	reg32 , reg32	386
ADC	reg_xotira16 , o'zgarmas8	8086
ADC	reg_xotira32 , o'zgarmas8	386
ADC	al , o'zgarmas	8086
ADC	ax , o'zgarmas	8086
ADC	eax , o'zgarmas	386
ADC	reg_xotira8 , o'zgarmas	8086
ADC	reg_xotira16 , o'zgarmas	8086
ADC	reg_xotira32 , o'zgarmas	386
ADC	xotira , o'zgarmas8	8086
ADC	xotira , o'zgarmas16	8086
ADC	xotira , o'zgarmas32	386
ADD	xotira , reg8	8086
ADD	reg8 , reg8	8086
ADD	xotira , reg16	8086
ADD	reg16 , reg16	8086
ADD	xotira , reg32	386
ADD	reg32 , reg32	386
ADD	reg8 , xotira	8086

Buyruq	Qiymatlar	Protsessor rusumi
ADD	reg8 , reg8	8086
ADD	reg16 , xotira	8086
ADD	reg16 , reg16	8086
ADD	reg32 , xotira	386
ADD	reg32 , reg32	386
ADD	reg_xotira16 , o'zgarmas8	8086
ADD	reg_xotira32 , o'zgarmas8	386
ADD	al , o'zgarmas	8086
ADD	ax , o'zgarmas	8086
ADD	eax , o'zgarmas	386
ADD	reg_xotira8 , o'zgarmas	8086
ADD	reg_xotira16 , o'zgarmas	8086
ADD	reg_xotira32 , o'zgarmas	386
ADD	xotira , o'zgarmas8	8086
ADD	xotira , o'zgarmas16	8086
ADD	xotira , o'zgarmas32	386
AND	xotira , reg8	8086
AND	reg8 , reg8	8086
AND	xotira , reg16	8086
AND	reg16 , reg16	8086
AND	xotira , reg32	386
AND	reg32 , reg32	386
AND	reg8 , xotira	8086
AND	reg8 , reg8	8086
AND	reg16 , xotira	8086
AND	reg16 , reg16	8086
AND	reg32 , xotira	386
AND	reg32 , reg32	386
AND	reg_xotira16 , o'zgarmas8	8086
AND	reg_xotira32 , o'zgarmas8	386
AND	al , o'zgarmas	8086
AND	ax , o'zgarmas	8086
AND	eax , o'zgarmas	386
AND	reg_xotira8 , o'zgarmas	8086
AND	reg_xotira16 , o'zgarmas	8086
AND	reg_xotira32 , o'zgarmas	386
AND	xotira , o'zgarmas8	8086

Buyruq	Qiymatlar	Protsessor rusumi
AND	xotira , o'zgarmas16	8086
AND	xotira , o'zgarmas32	386
ARPL	xotira , reg16	286,PROT,NOLONG
ARPL	reg16 , reg16	286,PROT,NOLONG
BB0_RESET		PENT,CYRIX,ND
BB1_RESET		PENT,CYRIX,ND
BOUND	reg16 , xotira	186,NOLONG
BOUND	reg32 , xotira	386,NOLONG
BSF	reg16 , xotira	386
BSF	reg16 , reg16	386
BSF	reg32 , xotira	386
BSF	reg32 , reg32	386
BSR	reg16 , xotira	386
BSR	reg16 , reg16	386
BSR	reg32 , xotira	386
BSR	reg32 , reg32	386
BSWAP	reg32	486
BT	xotira , reg16	386
BT	reg16 , reg16	386
BT	xotira , reg32	386
BT	reg32 , reg32	386
BT	reg_xotira16 , o'zgarmas	386
BT	reg_xotira32 , o'zgarmas	386
BTC	xotira , reg16	386
BTC	reg16 , reg16	386
BTC	xotira , reg32	386
BTC	reg32 , reg32	386
BTC	reg_xotira16 , o'zgarmas	386
BTC	reg_xotira32 , o'zgarmas	386
BTR	xotira , reg16	386
BTR	reg16 , reg16	386
BTR	xotira , reg32	386
BTR	reg32 , reg32	386
BTR	reg_xotira16 , o'zgarmas	386
BTR	reg_xotira32 , o'zgarmas	386
BTS	xotira , reg16	386
BTS	reg16 , reg16	386

Buyruq	Qiymatlar	Protsessor rusumi
BTS	xotira , reg32	386
BTS	reg32 , reg32	386
BTS	reg_xotira16 , o'zgarmas	386
BTS	reg_xotira32 , o'zgarmas	386
CALL	o'zgarmas	8086
CALL	o'zgarmas|near	8086
CALL	o'zgarmas|far	8086,ND,NOLONG
CALL	o'zgarmas16	8086
CALL	o'zgarmas16|near	8086
CALL	o'zgarmas16|far	8086,ND,NOLONG
CALL	o'zgarmas32	386
CALL	o'zgarmas32|near	386
CALL	o'zgarmas32|far	386,ND,NOLONG
CALL	xotira|far	8086
CALL	xotira16|far	8086
CALL	xotira32|far	386
CALL	xotira|near	8086
CALL	xotira16|near	8086
CALL	xotira32|near	386,NOLONG
CALL	reg16	8086
CALL	reg32	386,NOLONG
CALL	xotira	8086
CALL	xotira16	8086
CALL	xotira32	386,NOLONG
CBW		8086
CDQ		386
CLC		8086
CLD		8086
CLI		8086
CLTS		286,PRIV
CMC		8086
CMP	xotira , reg8	8086
CMP	reg8 , reg8	8086
CMP	xotira , reg16	8086
CMP	reg16 , reg16	8086
CMP	xotira , reg32	386
CMP	reg32 , reg32	386

Buyruq	Qiymatlar	Protsessor rusumi
CMP	reg8 , xotira	8086
CMP	reg8 , reg8	8086
CMP	reg16 , xotira	8086
CMP	reg16 , reg16	8086
CMP	reg32 , xotira	386
CMP	reg32 , reg32	386
CMP	reg_xotira16 , o'zgarmas8	8086
CMP	reg_xotira32 , o'zgarmas8	386
CMP	al , o'zgarmas	8086
CMP	ax , o'zgarmas	8086
CMP	eax , o'zgarmas	386
CMP	reg_xotira8 , o'zgarmas	8086
CMP	reg_xotira16 , o'zgarmas	8086
CMP	reg_xotira32 , o'zgarmas	386
CMP	xotira , o'zgarmas8	8086
CMP	xotira , o'zgarmas16	8086
CMP	xotira , o'zgarmas32	386
CMPSB		8086
CMPSD		386
CMPSW		8086
CMPXCHG	xotira , reg8	PENT
CMPXCHG	reg8 , reg8	PENT
CMPXCHG	xotira , reg16	PENT
CMPXCHG	reg16 , reg16	PENT
CMPXCHG	xotira , reg32	PENT
CMPXCHG	reg32 , reg32	PENT
CMPXCHG486	xotira , reg8	486,UNDOC,ND
CMPXCHG486	reg8 , reg8	486,UNDOC,ND
CMPXCHG486	xotira , reg16	486,UNDOC,ND
CMPXCHG486	reg16 , reg16	486,UNDOC,ND
CMPXCHG486	xotira , reg32	486,UNDOC,ND
CMPXCHG486	reg32 , reg32	486,UNDOC,ND
CMPXCHG8B	xotira	PENT
CPUID		PENT
CPU_READ		PENT,CYRIX
CPU_WRITE		PENT,CYRIX
CWD		8086

Buyruq	Qiymatlar	Protsessor rusumi
CWDE		386
DAA		8086,NOLONG
DAS		8086,NOLONG
DEC	reg16	8086,NOLONG
DEC	reg32	386,NOLONG
DEC	reg_xotira8	8086
DEC	reg_xotira16	8086
DEC	reg_xotira32	386
DIV	reg_xotira8	8086
DIV	reg_xotira16	8086
DIV	reg_xotira32	386
DMINT		P6,CYRIX
ENTER	o'zgarmas , o'zgarmas	186
EQU	o'zgarmas	8086
F2XM1		8086,FPU
FABS		8086,FPU
FADD	xotira32	8086,FPU
FADD	xotira64	8086,FPU
FADD	fpureg	8086,FPU
FADD	fpureg	8086,FPU
FADD	fpureg , fpu0	8086,FPU
FADD	fpu0 , fpureg	8086,FPU
FADD		8086,FPU,ND
FADDP	fpureg	8086,FPU
FADDP	fpureg , fpu0	8086,FPU
FADDP		8086,FPU,ND
FBLD	xotira80	8086,FPU
FBLD	xotira	8086,FPU
FBSTP	xotira80	8086,FPU
FBSTP	xotira	8086,FPU
FCHS		8086,FPU
FCLEX		8086,FPU
FCMOVB	fpureg	P6,FPU
FCMOVB	fpu0 , fpureg	P6,FPU
FCMOVB		P6,FPU,ND
FCMOVBE	fpureg	P6,FPU
FCMOVBE	fpu0 , fpureg	P6,FPU

Buyruq	Qiymatlar	Protsessor rusumi
FCMOVBE		P6,FPU,ND
FCMOVE	fpureg	P6,FPU
FCMOVE	fpu0 , fpureg	P6,FPU
FCMOVE		P6,FPU,ND
FCMOVNB	fpureg	P6,FPU
FCMOVNB	fpu0 , fpureg	P6,FPU
FCMOVNB		P6,FPU,ND
FCMOVNBE	fpureg	P6,FPU
FCMOVNBE	fpu0 , fpureg	P6,FPU
FCMOVNBE		P6,FPU,ND
FCMOVNE	fpureg	P6,FPU
FCMOVNE	fpu0 , fpureg	P6,FPU
FCMOVNE		P6,FPU,ND
FCMOVNU	fpureg	P6,FPU
FCMOVNU	fpu0 , fpureg	P6,FPU
FCMOVNU		P6,FPU,ND
FCMOVU	fpureg	P6,FPU
FCMOVU	fpu0 , fpureg	P6,FPU
FCMOVU		P6,FPU,ND
FCOM	xotira32	8086,FPU
FCOM	xotira64	8086,FPU
FCOM	fpureg	8086,FPU
FCOM	fpu0 , fpureg	8086,FPU
FCOM		8086,FPU,ND
FCOMI	fpureg	P6,FPU
FCOMI	fpu0 , fpureg	P6,FPU
FCOMI		P6,FPU,ND
FCOMIP	fpureg	P6,FPU
FCOMIP	fpu0 , fpureg	P6,FPU
FCOMIP		P6,FPU,ND
FCOMP	xotira32	8086,FPU
FCOMP	xotira64	8086,FPU
FCOMP	fpureg	8086,FPU
FCOMP	fpu0 , fpureg	8086,FPU
FCOMP		8086,FPU,ND
FCOMPP		8086,FPU
FCOS		386,FPU

Buyruq	Qiymatlar	Protsessor rusumi
FDECSTP		8086,FPU
FDISI		8086,FPU
FDIV	xotira32	8086,FPU
FDIV	xotira64	8086,FPU
FDIV	fpureg	8086,FPU
FDIV	fpureg	8086,FPU
FDIV	fpureg , fpu0	8086,FPU
FDIV	fpu0 , fpureg	8086,FPU
FDIV		8086,FPU,ND
FDIVP	fpureg	8086,FPU
FDIVP	fpureg , fpu0	8086,FPU
FDIVP		8086,FPU,ND
FDIVR	xotira32	8086,FPU
FDIVR	xotira64	8086,FPU
FDIVR	fpureg	8086,FPU
FDIVR	fpureg , fpu0	8086,FPU
FDIVR	fpureg	8086,FPU
FDIVR	fpu0 , fpureg	8086,FPU
FDIVR		8086,FPU,ND
FDIVRP	fpureg	8086,FPU
FDIVRP	fpureg , fpu0	8086,FPU
FDIVRP		8086,FPU,ND
FEMMS		PENT,3DNOW
FENI		8086,FPU
FFREE	fpureg	8086,FPU
FFREE		8086,FPU
FFREEP	fpureg	286,FPU,UNDOC
FFREEP		286,FPU,UNDOC
FIADD	xotira32	8086,FPU
FIADD	xotira16	8086,FPU
FICOM	xotira32	8086,FPU
FICOM	xotira16	8086,FPU
FICOMP	xotira32	8086,FPU
FICOMP	xotira16	8086,FPU
FIDIV	xotira32	8086,FPU
FIDIV	xotira16	8086,FPU
FIDIVR	xotira32	8086,FPU

Buyruq	Qiymatlar	Protsessor rusumi
FIDIVR	xotira16	8086,FPU
FILD	xotira32	8086,FPU
FILD	xotira16	8086,FPU
FILD	xotira64	8086,FPU
FIMUL	xotira32	8086,FPU
FIMUL	xotira16	8086,FPU
FINCSTP		8086,FPU
FINIT		8086,FPU
FIST	xotira32	8086,FPU
FIST	xotira16	8086,FPU
FISTP	xotira32	8086,FPU
FISTP	xotira16	8086,FPU
FISTP	xotira64	8086,FPU
FISTTP	xotira16	PRESCOTT,FPU
FISTTP	xotira32	PRESCOTT,FPU
FISTTP	xotira64	PRESCOTT,FPU
FISUB	xotira32	8086,FPU
FISUB	xotira16	8086,FPU
FISUBR	xotira32	8086,FPU
FISUBR	xotira16	8086,FPU
FLD	xotira32	8086,FPU
FLD	xotira64	8086,FPU
FLD	xotira80	8086,FPU
FLD	fpureg	8086,FPU
FLD		8086,FPU,ND
FLD1		8086,FPU
FLDCW	xotira	8086,FPU,SW
FLDENV	xotira	8086,FPU
FLDL2E		8086,FPU
FLDL2T		8086,FPU
FLDLG2		8086,FPU
FLDLN2		8086,FPU
FLDPI		8086,FPU
FLDZ		8086,FPU
FMUL	xotira32	8086,FPU
FMUL	xotira64	8086,FPU
FMUL	fpureg	8086,FPU

Buyruq	Qiymatlar	Protsessor rusumi
FMUL	fpureg , fpu0	8086,FPU
FMUL	fpureg	8086,FPU
FMUL	fpu0 , fpureg	8086,FPU
FMUL		8086,FPU,ND
FMULP	fpureg	8086,FPU
FMULP	fpureg , fpu0	8086,FPU
FMULP		8086,FPU,ND
FNCLEX		8086,FPU
FNDISI		8086,FPU
FNENI		8086,FPU
FNINIT		8086,FPU
FNOP		8086,FPU
FNSAVE	xotira	8086,FPU
FNSTCW	xotira	8086,FPU,SW
FNSTENV	xotira	8086,FPU
FNSTSW	xotira	8086,FPU,SW
FNSTSW	ax	286,FPU
FPATAN		8086,FPU
FPREM		8086,FPU
FPREM1		386,FPU
FPTAN		8086,FPU
FRNDINT		8086,FPU
FRSTOR	xotira	8086,FPU
FSAVE	xotira	8086,FPU
FSCALE		8086,FPU
FSETPM		286,FPU
FSIN		386,FPU
FSINCOS		386,FPU
FSQRT		8086,FPU
FST	xotira32	8086,FPU
FST	xotira64	8086,FPU
FST	fpureg	8086,FPU
FST		8086,FPU,ND
FSTCW	xotira	8086,FPU,SW
FSTENV	xotira	8086,FPU
FSTP	xotira32	8086,FPU

Buyruq	Qiymatlar	Protsessor rusumi
FSTP	xotira64	8086,FPU
FSTP	xotira80	8086,FPU
FSTP	fpureg	8086,FPU
FSTP		8086,FPU,ND
FSTSW	xotira	8086,FPU,SW
FSTSW	ax	286,FPU
FSUB	xotira32	8086,FPU
FSUB	xotira64	8086,FPU
FSUB	fpureg	8086,FPU
FSUB	fpureg , fpu0	8086,FPU
FSUB	fpureg	8086,FPU
FSUB	fpu0 , fpureg	8086,FPU
FSUB		8086,FPU,ND
FSUBP	fpureg	8086,FPU
FSUBP	fpureg , fpu0	8086,FPU
FSUBP		8086,FPU,ND
FSUBR	xotira32	8086,FPU
FSUBR	xotira64	8086,FPU
FSUBR	fpureg	8086,FPU
FSUBR	fpureg , fpu0	8086,FPU
FSUBR	fpureg	8086,FPU
FSUBR	fpu0 , fpureg	8086,FPU
FSUBR		8086,FPU,ND
FSUBRP	fpureg	8086,FPU
FSUBRP	fpureg , fpu0	8086,FPU
FSUBRP		8086,FPU,ND
FTST		8086,FPU
FUCOM	fpureg	386,FPU
FUCOM	fpu0 , fpureg	386,FPU
FUCOM		386,FPU,ND
FUCOMI	fpureg	P6,FPU
FUCOMI	fpu0 , fpureg	P6,FPU
FUCOMI		P6,FPU,ND
FUCOMIP	fpureg	P6,FPU
FUCOMIP	fpu0 , fpureg	P6,FPU
FUCOMIP		P6,FPU,ND

Buyruq	Qiymatlar	Protsessor rusumi
FUCOMP	fpureg	386,FPU
FUCOMP	fpu0 , fpureg	386,FPU
FUCOMP		386,FPU,ND
FUCOMPP		386,FPU
FXAM		8086,FPU
FXCH	fpureg	8086,FPU
FXCH	fpureg , fpu0	8086,FPU
FXCH	fpu0 , fpureg	8086,FPU
FXCH		8086,FPU,ND
FXTRACT		8086,FPU
FYL2X		8086,FPU
FYL2XP1		8086,FPU
HLT		8086,PRIV
IBTS	xotira , reg16	386,SW,UNDOC,ND
IBTS	reg16 , reg16	386,UNDOC,ND
IBTS	xotira , reg32	386,SD,UNDOC,ND
IBTS	reg32 , reg32	386,UNDOC,ND
ICEBP		386,ND
IDIV	reg_xotira8	8086
IDIV	reg_xotira16	8086
IDIV	reg_xotira32	386
IMUL	reg_xotira8	8086
IMUL	reg_xotira16	8086
IMUL	reg_xotira32	386
IMUL	reg16 , xotira	386
IMUL	reg16 , reg16	386
IMUL	reg32 , xotira	386
IMUL	reg32 , reg32	386
IMUL	reg16 , xotira , o'zgarmas8	186
IMUL	reg16 , xotira , o'zgarmas16	186
IMUL	reg16 , xotira , o'zgarmas	186,ND
IMUL	reg16 , reg16 , o'zgarmas8	186
IMUL	reg16 , reg16 , o'zgarmas16	186
IMUL	reg16 , reg16 , o'zgarmas	186,ND
IMUL	reg32 , xotira , o'zgarmas8	386
IMUL	reg32 , xotira , o'zgarmas32	386

Buyruq	Qiymatlar	Protsessor rusumi
IMUL	reg32 , xotira , o'zgarmas	386,ND
IMUL	reg32 , reg32 , o'zgarmas8	386
IMUL	reg32 , reg32 , o'zgarmas32	386
IMUL	reg32 , reg32 , o'zgarmas	386,ND
IMUL	reg16 , o'zgarmas8	186
IMUL	reg16 , o'zgarmas16	186
IMUL	reg16 , o'zgarmas	186,ND
IMUL	reg32 , o'zgarmas8	386
IMUL	reg32 , o'zgarmas32	386
IMUL	reg32 , o'zgarmas	386,ND
IN	al , o'zgarmas	8086
IN	ax , o'zgarmas	8086
IN	eax , o'zgarmas	386
IN	al , dx	8086
IN	ax , dx	8086
IN	eax , dx	386
INC	reg16	8086,NOLONG
INC	reg32	386,NOLONG
INC	reg_xotira8	8086
INC	reg_xotira16	8086
INC	reg_xotira32	386
INCBIN		
INSB		186
INSD		386
INSW		186
INT	o'zgarmas	8086
INT01		386,ND
INT1		386
INT03		8086,ND
INT3		8086
INTO		8086,NOLONG
INVD		486,PRIV
INVLPG	xotira	486,PRIV
INVLPGA	ax , ecx	X86_64,AMD,NOLONG
INVLPGA	eax , ecx	X86_64,AMD
INVLPGA		X86_64,AMD

Buyruq	Qiymatlar	Protsessor rusumi
IRET		8086
IRETD		386
IRETW		8086
JCXZ	o'zgarmas	8086,NOLONG
JECXZ	o'zgarmas	386
JMP	o'zgarmas|short	8086
JMP	o'zgarmas	8086,ND
JMP	o'zgarmas	8086
JMP	o'zgarmas|near	8086,ND
JMP	o'zgarmas|far	8086,ND,NOLONG
JMP	o'zgarmas16	8086
JMP	o'zgarmas16|near	8086,ND
JMP	o'zgarmas16|far	8086,ND,NOLONG
JMP	o'zgarmas32	386
JMP	o'zgarmas32|near	386,ND
JMP	o'zgarmas32|far	386,ND,NOLONG
JMP	xotira|far	8086
JMP	xotira16|far	8086
JMP	xotira32|far	386
JMP	xotira|near	8086
JMP	xotira16|near	8086
JMP	xotira32|near	386,NOLONG
JMP	reg16	8086
JMP	reg32	386,NOLONG
JMP	xotira	8086
JMP	xotira16	8086
JMP	xotira32	386,NOLONG
JMPE	o'zgarmas	IA64
JMPE	o'zgarmas16	IA64
JMPE	o'zgarmas32	IA64
JMPE	reg_xotira16	IA64
JMPE	reg_xotira32	IA64
LAHF		8086
LAR	reg16 , xotira	286,PROT,SW
LAR	reg16 , reg16	286,PROT
LAR	reg16 , reg32	386,PROT

Buyruq	Qiymatlar	Protsessor rusumi
LAR	reg32 , xotira	386,PROT,SW
LAR	reg32 , reg16	386,PROT
LAR	reg32 , reg32	386,PROT
LDS	reg16 , xotira	8086,NOLONG
LDS	reg32 , xotira	386,NOLONG
LEA	reg16 , xotira	8086
LEA	reg32 , xotira	386
LEAVE		186
LES	reg16 , xotira	8086,NOLONG
LES	reg32 , xotira	386,NOLONG
LFS	reg16 , xotira	386
LFS	reg32 , xotira	386
LGDT	xotira	286,PRIV
LGS	reg16 , xotira	386
LGS	reg32 , xotira	386
LIDT	xotira	286,PRIV
LLDT	xotira	286,PROT,PRIV
LLDT	xotira16	286,PROT,PRIV
LLDT	reg16	286,PROT,PRIV
LMSW	xotira	286,PRIV
LMSW	xotira16	286,PRIV
LMSW	reg16	286,PRIV
LOADALL		386,UNDOC
LOADALL286		286,UNDOC
LODSB		8086
LODSD		386
LODSW		8086
LOOP	o'zgarmas	8086
LOOP	o'zgarmas , cx	8086,NOLONG
LOOP	o'zgarmas , ecx	386
LOOPE	o'zgarmas	8086
LOOPE	o'zgarmas , cx	8086,NOLONG
LOOPE	o'zgarmas , ecx	386
LOOPNE	o'zgarmas	8086
LOOPNE	o'zgarmas , cx	8086,NOLONG
LOOPNE	o'zgarmas , ecx	386

Buyruq	Qiymatlar	Protsessor rusumi
LOOPNZ	o'zgarmas	8086
LOOPNZ	o'zgarmas , cx	8086,NOLONG
LOOPNZ	o'zgarmas , ecx	386
LOOPZ	o'zgarmas	8086
LOOPZ	o'zgarmas , cx	8086,NOLONG
LOOPZ	o'zgarmas , ecx	386
LSL	reg16 , xotira	286,PROT,SW
LSL	reg16 , reg16	286,PROT
LSL	reg16 , reg32	386,PROT
LSL	reg32 , xotira	386,PROT,SW
LSL	reg32 , reg16	386,PROT
LSL	reg32 , reg32	386,PROT
LSS	reg16 , xotira	386
LSS	reg32 , xotira	386
LTR	xotira	286,PROT,PRIV
LTR	xotira16	286,PROT,PRIV,NOLONG
LTR	reg16	286,PROT,PRIV,NOLONG
MONITOR		PRESCOTT
MONITOR	eax , ecx , edx	PRESCOTT,ND
MOV	xotira , reg	8086
MOV	reg16 , reg	8086
MOV	reg32 , reg	386
MOV	reg , xotira	8086
MOV	reg , reg16	8086
MOV	reg , reg32	386
MOV	al , xotira	8086
MOV	ax , xotira	8086
MOV	eax , xotira	386
MOV	xotira , al	8086
MOV	xotira , ax	8086
MOV	xotira , eax	386
MOV	reg32 , reg	386,PRIV,NOLONG
MOV	reg , reg32	386,PRIV,NOLONG
MOV	reg32 , reg	386,PRIV
MOV	reg , reg32	386,PRIV
MOV	reg32 , reg	386,NOLONG,ND

Buyruq	Qiymatlar	Protsessor rusumi
MOV	reg , reg32	386,NOLONG,ND
MOV	xotira , reg8	8086
MOV	reg8 , reg8	8086
MOV	xotira , reg16	8086
MOV	reg16 , reg16	8086
MOV	xotira , reg32	386
MOV	reg32 , reg32	386
MOV	reg8 , xotira	8086
MOV	reg8 , reg8	8086
MOV	reg16 , xotira	8086
MOV	reg16 , reg16	8086
MOV	reg32 , xotira	386
MOV	reg32 , reg32	386
MOV	reg8 , o'zgarmas	8086
MOV	reg16 , o'zgarmas	8086
MOV	reg32 , o'zgarmas	386
MOV	reg_xotira8 , o'zgarmas	8086
MOV	reg_xotira16 , o'zgarmas	8086
MOV	reg_xotira32 , o'zgarmas	386
MOV	xotira , o'zgarmas8	8086
MOV	xotira , o'zgarmas16	8086
MOV	xotira , o'zgarmas32	386
MOVSB		8086
MOVSD		386
MOVSW		8086
MOVSX	reg16 , xotira	386
MOVSX	reg16 , reg8	386
MOVSX	reg32 , reg_xotira8	386
MOVSX	reg32 , reg_xotira16	386
MOVZX	reg16 , xotira	386
MOVZX	reg16 , reg8	386
MOVZX	reg32 , reg_xotira8	386
MOVZX	reg32 , reg_xotira16	386
MUL	reg_xotira8	8086
MUL	reg_xotira16	8086
MUL	reg_xotira32	386

Buyruq	Qiymatlar	Protsessor rusumi
MWAIT		PRESCOTT
MWAIT	eax , ecx	PRESCOTT,ND
NEG	reg_xotira8	8086
NEG	reg_xotira16	8086
NEG	reg_xotira32	386
NOP		8086
NOP	reg_xotira16	P6
NOP	reg_xotira32	P6
NOT	reg_xotira8	8086
NOT	reg_xotira16	8086
NOT	reg_xotira32	386
OR	xotira , reg8	8086
OR	reg8 , reg8	8086
OR	xotira , reg16	8086
OR	reg16 , reg16	8086
OR	xotira , reg32	386
OR	reg32 , reg32	386
OR	reg8 , xotira	8086
OR	reg8 , reg8	8086
OR	reg16 , xotira	8086
OR	reg16 , reg16	8086
OR	reg32 , xotira	386
OR	reg32 , reg32	386
OR	reg_xotira16 , o'zgarmas8	8086
OR	reg_xotira32 , o'zgarmas8	386
OR	al , o'zgarmas	8086
OR	ax , o'zgarmas	8086
OR	eax , o'zgarmas	386
OR	reg_xotira8 , o'zgarmas	8086
OR	reg_xotira16 , o'zgarmas	8086
OR	reg_xotira32 , o'zgarmas	386
OR	xotira , o'zgarmas8	8086
OR	xotira , o'zgarmas16	8086
OR	xotira , o'zgarmas32	386
OUT	o'zgarmas , al	8086
OUT	o'zgarmas , ax	8086

Buyruq	Qiymatlar	Protsessor rusumi
OUT	o'zgarmas , eax	386
OUT	dx , al	8086
OUT	dx , ax	8086
OUT	dx , eax	386
OUTSB		186
OUTSD		386
OUTSW		186
PAUSE		8086
POP	reg16	8086
POP	reg32	386,NOLONG
POP	reg_xotira16	8086
POP	reg_xotira32	386,NOLONG
POPA		186,NOLONG
POPAD		386,NOLONG
POPAW		186,NOLONG
POPF		8086
POPFD		386,NOLONG
POPFW		8086
PREFETCH	xotira	PENT,3DNOW
PREFETCHW	xotira	PENT,3DNOW
PUSH	reg16	8086
PUSH	reg32	386,NOLONG
PUSH	reg_xotira16	8086
PUSH	reg_xotira32	386,NOLONG
PUSH	o'zgarmas8	186
PUSH	o'zgarmas16	186,AR0,SZ
PUSH	o'zgarmas32	386,NOLONG,AR0,SZ
PUSH	o'zgarmas32	386,NOLONG,SD
PUSHA		186,NOLONG
PUSHAD		386,NOLONG
PUSHAW		186,NOLONG
PUSHF		8086
PUSHFD		386,NOLONG
PUSHFW		8086
RCL	reg_xotira8 , cl	8086
RCL	reg_xotira8 , o'zgarmas	186

Buyruq	Qiymatlar	Protsessor rusumi
RCL	reg_xotira16 , cl	8086
RCL	reg_xotira16 , o'zgarmas	186
RCL	reg_xotira32 , cl	386
RCL	reg_xotira32 , o'zgarmas	386
RCR	reg_xotira8 , cl	8086
RCR	reg_xotira8 , o'zgarmas	186
RCR	reg_xotira16 , cl	8086
RCR	reg_xotira16 , o'zgarmas	186
RCR	reg_xotira32 , cl	386
RCR	reg_xotira32 , o'zgarmas	386
RDSHR	reg_xotira32	P6,CYRIXM
RDMSR		PENT,PRIV
RDPMC		P6
RDTSC		PENT
RDTSCP		X86_64
RET		8086
RET	o'zgarmas	8086,SW
RETF		8086
RETF	o'zgarmas	8086,SW
RETN		8086
RETN	o'zgarmas	8086,SW
ROL	reg_xotira8 , cl	8086
ROL	reg_xotira8 , o'zgarmas	186
ROL	reg_xotira16 , cl	8086
ROL	reg_xotira16 , o'zgarmas	186
ROL	reg_xotira32 , cl	386
ROL	reg_xotira32 , o'zgarmas	386
ROR	reg_xotira8 , cl	8086
ROR	reg_xotira8 , o'zgarmas	186
ROR	reg_xotira16 , cl	8086
ROR	reg_xotira16 , o'zgarmas	186
ROR	reg_xotira32 , cl	386
ROR	reg_xotira32 , o'zgarmas	386
RDM		P6,CYRIX,ND
RSDC	reg , xotira80	486,CYRIXM
RSLDT	xotira80	486,CYRIXM

Buyruq	Qiymatlar	Protsessor rusumi
RSM		PENTM
RSTS	xotira80	486,CYRIXM
SAHF		8086
SAL	reg_xotira8 , cl	8086,ND
SAL	reg_xotira8 , o'zgarmas	186,ND
SAL	reg_xotira16 , cl	8086,ND
SAL	reg_xotira16 , o'zgarmas	186,ND
SAL	reg_xotira32 , cl	386,ND
SAL	reg_xotira32 , o'zgarmas	386,ND
SALC		8086,UNDOC
SAR	reg_xotira8 , cl	8086
SAR	reg_xotira8 , o'zgarmas	186
SAR	reg_xotira16 , cl	8086
SAR	reg_xotira16 , o'zgarmas	186
SAR	reg_xotira32 , cl	386
SAR	reg_xotira32 , o'zgarmas	386
SBB	xotira , reg8	8086
SBB	reg8 , reg8	8086
SBB	xotira , reg16	8086
SBB	reg16 , reg16	8086
SBB	xotira , reg32	386
SBB	reg32 , reg32	386
SBB	reg8 , xotira	8086
SBB	reg8 , reg8	8086
SBB	reg16 , xotira	8086
SBB	reg16 , reg16	8086
SBB	reg32 , xotira	386
SBB	reg32 , reg32	386
SBB	reg_xotira16 , o'zgarmas8	8086
SBB	reg_xotira32 , o'zgarmas8	386
SBB	al , o'zgarmas	8086
SBB	ax , o'zgarmas	8086
SBB	eax , o'zgarmas	386
SBB	reg_xotira8 , o'zgarmas	8086
SBB	reg_xotira16 , o'zgarmas	8086
SBB	reg_xotira32 , o'zgarmas	386

Buyruq	Qiymatlar	Protsessor rusumi
SBB	xotira , o'zgarmas8	8086
SBB	xotira , o'zgarmas16	8086
SBB	xotira , o'zgarmas32	386
SCASB		8086
SCASD		386
SCASW		8086
SGDT	xotira	286
SHL	reg_xotira8 , cl	8086
SHL	reg_xotira8 , o'zgarmas	186
SHL	reg_xotira16 , cl	8086
SHL	reg_xotira16 , o'zgarmas	186
SHL	reg_xotira32 , cl	386
SHL	reg_xotira32 , o'zgarmas	386
SHLD	xotira , reg16 , o'zgarmas	3862
SHLD	reg16 , reg16 , o'zgarmas	3862
SHLD	xotira , reg32 , o'zgarmas	3862
SHLD	reg32 , reg32 , o'zgarmas	3862
SHLD	xotira , reg16 , cl	386
SHLD	reg16 , reg16 , cl	386
SHLD	xotira , reg32 , cl	386
SHLD	reg32 , reg32 , cl	386
SHR	reg_xotira8 , cl	8086
SHR	reg_xotira8 , o'zgarmas	186
SHR	reg_xotira16 , cl	8086
SHR	reg_xotira16 , o'zgarmas	186
SHR	reg_xotira32 , cl	386
SHR	reg_xotira32 , o'zgarmas	386
SHRD	xotira , reg16 , o'zgarmas	3862
SHRD	reg16 , reg16 , o'zgarmas	3862
SHRD	xotira , reg32 , o'zgarmas	3862
SHRD	reg32 , reg32 , o'zgarmas	3862
SHRD	xotira , reg16 , cl	386
SHRD	reg16 , reg16 , cl	386
SHRD	xotira , reg32 , cl	386
SHRD	reg32 , reg32 , cl	386
SIDT	xotira	286

Buyruq	Qiymatlar	Protsessor rusumi
SLDT	xotira	286
SLDT	xotira16	286
SLDT	reg16	286
SLDT	reg32	386
SMI		386,UNDOC
SMINT		P6,CYRIX,ND
SMINTOLD		486,CYRIX,ND
SMSW	xotira	286
SMSW	xotira16	286
SMSW	reg16	286
SMSW	reg32	386
STC		8086
STD		8086
STI		8086
STOSB		8086
STOSD		386
STOSW		8086
STR	xotira	286,PROT
STR	xotira16	286,PROT
STR	reg16	286,PROT
STR	reg32	386,PROT
SUB	xotira , reg8	8086
SUB	reg8 , reg8	8086
SUB	xotira , reg16	8086
SUB	reg16 , reg16	8086
SUB	xotira , reg32	386
SUB	reg32 , reg32	386
SUB	reg8 , xotira	8086
SUB	reg8 , reg8	8086
SUB	reg16 , xotira	8086
SUB	reg16 , reg16	8086
SUB	reg32 , xotira	386
SUB	reg32 , reg32	386
SUB	reg_xotira16 , o'zgarmas8	8086
SUB	reg_xotira32 , o'zgarmas8	386
SUB	al , o'zgarmas	8086

Buyruq	Qiymatlar	Protsessor rusumi
SUB	ax , o'zgarmas	8086
SUB	eax , o'zgarmas	386
SUB	reg_xotira8 , o'zgarmas	8086
SUB	reg_xotira16 , o'zgarmas	8086
SUB	reg_xotira32 , o'zgarmas	386
SUB	xotira , o'zgarmas8	8086
SUB	xotira , o'zgarmas16	8086
SUB	xotira , o'zgarmas32	386
SVDC	xotira80 , reg	486,CYRIXM
SVLDT	xotira80	486,CYRIXM,ND
SVTS	xotira80	486,CYRIXM
SYSCALL		P6,AMD
SYSENTER		P6
SYSEXIT		P6,PRIV
SYSRET		P6,PRIV,AMD
TEST	xotira , reg8	8086
TEST	reg8 , reg8	8086
TEST	xotira , reg16	8086
TEST	reg16 , reg16	8086
TEST	xotira , reg32	386
TEST	reg32 , reg32	386
TEST	reg8 , xotira	8086
TEST	reg16 , xotira	8086
TEST	reg32 , xotira	386
TEST	al , o'zgarmas	8086
TEST	ax , o'zgarmas	8086
TEST	eax , o'zgarmas	386
TEST	reg_xotira8 , o'zgarmas	8086
TEST	reg_xotira16 , o'zgarmas	8086
TEST	reg_xotira32 , o'zgarmas	386
TEST	xotira , o'zgarmas8	8086
TEST	xotira , o'zgarmas16	8086
TEST	xotira , o'zgarmas32	386
UD0		186,UNDOC
UD1		186,UNDOC
UD2B		186,UNDOC,ND

Buyruq	Qiymatlar	Protsessor rusumi
UD2		186
UD2A		186,ND
UMOV	xotira , reg8	386,UNDOC,ND
UMOV	reg8 , reg8	386,UNDOC,ND
UMOV	xotira , reg16	386,UNDOC,ND
UMOV	reg16 , reg16	386,UNDOC,ND
UMOV	xotira , reg32	386,UNDOC,ND
UMOV	reg32 , reg32	386,UNDOC,ND
UMOV	reg8 , xotira	386,UNDOC,ND
UMOV	reg8 , reg8	386,UNDOC,ND
UMOV	reg16 , xotira	386,UNDOC,ND
UMOV	reg16 , reg16	386,UNDOC,ND
UMOV	reg32 , xotira	386,UNDOC,ND
UMOV	reg32 , reg32	386,UNDOC,ND
VERR	xotira	286,PROT
VERR	xotira16	286,PROT
VERR	reg16	286,PROT
VERW	xotira	286,PROT
VERW	xotira16	286,PROT
VERW	reg16	286,PROT
WAIT		8086
FWAIT		8086
WBINVD		486,PRIV
WRSHR	reg_xotira32	P6,CYRIXM
WRMSR		PENT,PRIV
XADD	xotira , reg8	486
XADD	reg8 , reg8	486
XADD	xotira , reg16	486
XADD	reg16 , reg16	486
XADD	xotira , reg32	486
XADD	reg32 , reg32	486
XBTS	reg16 , xotira	386,SW,UNDOC,ND
XBTS	reg16 , reg16	386,UNDOC,ND
XBTS	reg32 , xotira	386,SD,UNDOC,ND
XBTS	reg32 , reg32	386,UNDOC,ND
XCHG	ax , reg16	8086

Buyruq	Qiymatlar	Protsessor rusumi
XCHG	reg16 , ax	8086
XCHG	eax , eax	386,NOLONG
XCHG	reg8 , xotira	8086
XCHG	reg8 , reg8	8086
XCHG	reg16 , xotira	8086
XCHG	reg16 , reg16	8086
XCHG	reg32 , xotira	386
XCHG	reg32 , reg32	386
XCHG	xotira , reg8	8086
XCHG	reg8 , reg8	8086
XCHG	xotira , reg16	8086
XCHG	reg16 , reg16	8086
XCHG	xotira , reg32	386
XCHG	reg32 , reg32	386
XOR	xotira , reg8	8086
XOR	reg8 , reg8	8086
XOR	xotira , reg16	8086
XOR	reg16 , reg16	8086
XOR	xotira , reg32	386
XOR	reg32 , reg32	386
XOR	reg8 , xotira	8086
XOR	reg8 , reg8	8086
XOR	reg16 , xotira	8086
XOR	reg16 , reg16	8086
XOR	reg32 , xotira	386
XOR	reg32 , reg32	386
XOR	reg_xotira16 , o'zgarmas8	8086
XOR	reg_xotira32 , o'zgarmas8	386
XOR	al , o'zgarmas	8086
XOR	ax , o'zgarmas	8086
XOR	eax , o'zgarmas	386
XOR	reg_xotira8 , o'zgarmas	8086
XOR	reg_xotira16 , o'zgarmas	8086
XOR	reg_xotira32 , o'zgarmas	386
XOR	xotira , o'zgarmas8	8086
XOR	xotira , o'zgarmas16	8086

Buyruq	Qiymatlar	Protsessor rusumi
XOR	xotira , o'zgarmas32	386
CMOV*xx*	reg16 , xotira	P6
CMOV*xx*	reg16 , reg16	P6
CMOV*xx*	reg32 , xotira	P6
CMOV*xx*	reg32 , reg32	P6
J*xx*	o'zgarmas|near	386
J*xx*	o'zgarmas16|near	386
J*xx*	o'zgarmas32|near	386
J*xx*	o'zgarmas|short	8086,ND
J*xx*	o'zgarmas	8086,ND
J*xx*	o'zgarmas	386,ND
J*xx*	o'zgarmas	8086,ND
J*xx*	o'zgarmas	8086
SET*xx*	xotira	386
SET*xx*	reg8	386

Ilova F. Atamalar

Dasturlashda ishlatiladigan ko'pgina atamalar inglizcha bo'lib, ular boshqa tillarda gohida tarjima qilib, gohida esa o'z holicha ishlatiladi. Mazkur kitobda atamalar imkon boricha o'zbekchalashtirilgan va bu ba'zida tushunmovchiliklarga olib kelishi mumkin. Shuni hisobga olgan holda ushbu ilovada barcha ishlatilgan atamalarning inglizcha va ruscha tarjimalari keltirilgan.

O’zbekcha	Inglizcha	Ruscha
aloqa fayli	socket and pipes	сокеты и каналы
andoza	format	формат
anjom	tool	инструмент
asos darajasi	exponent	экспонент
bajaruvchi fayl	executable file	исполняемый файл
bayroq	flag	флаг
belgilarni raqamlash turlari	character encoding types	типы кодировка символов
bir satrli makro	single line macro	однострочный макрос
birlamchi aniqlik	single precision	одинарная точность
bitma-bit	bitwise	побитовый
bo'lim, segment	segment	сегмент
boshlang'ich fayl	header file	заголовочный файл
boshqaruv buyruqlari	control statements	операторы управления
buyruq qabul qiladigan qiymatlar	command arguments	аргументы команды
buyruqlar qatori	command line, terminal, console, prompt	командная строка
chaqirish	call	вызов
chiqarib olish/tashlash	pop	удаление
chiziqli	linear	линейный
dastur bo'lagi	code snippet; module	фрагмент кода; модуль
dasturni shartli yig'ish	conditional assembly	условное ассемблирование
direktoriya	directory, catalogue, folder	папка
diskdon	cd-rom	cd-rom
e'lon qilish	to declare	объявить
elektr o'zgartiruvchi devorchalar	commutation panel	коммутационные панели
erkin	free	свободный
fayl egasi	file owner	владелец файла

O’zbekcha	Inglizcha	Ruscha
fayl tizimi	file system	файловая система
foydalanuvchi dasturi	user program, application	программа пользователя, приложение
foydalanuvchi tartibi	user mode	непривилегированный режим
foydali manzil	effective address	эффективный адрес
guvohnoma	license	лицензия
halqasimon siljitish	rotate	циклический сдвиг
haqiqiy usul	real mode	реальный режим
himoyalangan usul	protected mode	защищенный режим
hisob taxtachasi	abacus	абак
holat registri	status register	регистр состояния
ikkilik	binary	двоичный
ildiz foydalanuvchisi	root user	корневой пользователь
jadval	table; two dimensional array, matrix	таблица; двухмерный массив, матрица
jadval qatori	row	строка
jarayon	process	процесс
kalit	option	опция
kengaytirilgan aniqlik	extended precision	расширенная точность
kenja, quyi	rightmost, low	младший
kiritish, joylashtirish	push	добавление
ko'p satrli makro	multi line macro	многострочный макрос
ko'p vazifalik	multimedia	мультимедиа
ko'rsatgich	pointer	указатель
kompilyator	compiler	компилятор
mahalliy	local	локальный
makro vosita	macro processor	макропроцессор
makroga qiymat berish	invoke macro with value	вызов макроса со значением
makroni aniqlamoq	to define macro	определить макрос
manba	source	источник
manzil	address	адрес
manzil orqali jo'natish	pass by address	передача по адресу
maqsad	destination	приемник
maromlashtirish	normalization	нормализация

O’zbekcha	Inglizcha	Ruscha
mashina tili	machine language	машинный язык
maxfiy so'z	password	пароль
mezoniy kutubxona	standard library	стандартная библиотека
moslama, qurilma	device	устройство
moslama, qurilmani boshqaruvchi dastur	driver	драйвер
mutlaq yo'l	absolute path	абсолютный путь
nisbiy yo'l	relative path	относительный путь
nishon	label	метка
o'n oltilik	hexadecimal	шестнадцатеричный
o'nlik	decimal	десятичный
o'quv/yozuv	input/output	ввод-вывод
o'z-o'zini chaqiradigan	recursive	рекурсивный
o'zak	kernel, core	ядро
ochiq kod	open source	открытый исходный код
oddiy fayl	regular file	регулярный файл
ommaviy	global	глобальный
operatsion tizim	operating system	операционная система
operatsion tizim tartibi	kernel mode	привилегированный режим
protsessor	processor	процессор
qator	string	строка
qatorli o'zgaruvchi/o'zgarmas	string variable/constant	строковая переменная/константа
qayta kirish	reenterability	реентерабельность
qism dastur	subprogram, subroutine, procedure, function	подпрограмма, процедура, функция
qiymat orqali jo'natish	pass by value	передача по значению
qo'sh aniqlik	double precision	двойная точность
qolip	array	массив
qolip kataklari	array cell, array element	элемент массива
qolip o'lchami	array size in bytes	размер массива в байтах
qolip uzunligi	array length	длина массива
ramziy yorliq fayli	symbolic link file	символическая ссылка на файл
sakkizlik	octal	восьмеричный
sakrash, o'tish	jump	переход

O’zbekcha	Inglizcha	Ruscha
sanoq tizimi	number system	система счисления
satr	line	строка
siljitish	shift	сдвиг
sonli qo'shma protsessor	numeric coprocessor	числовой сопроцессор
stack	stack	стек
stack cho'qqisi	top of the stack	вершина стека
tahlilchi dastur	debugger	отладчик
takrorlanish	loop, cycle, iteration	цикл, итерация
tarmoq	net	сеть
tarmoqlanish	branching	ветвление
tartib raqam	index, offset	индекс, смещение
tizim chaqirig'i	system call	системный вызов
tizim soati	system clock	системные часы
to'ng'ich, yuqori	leftmost, high, upper	старший
tugmachalar taxtasi	keyboard	клавиатура
turkum	version	версия
ulovchi dastur	linker	компоновщик
unixsimon	unix-like	unix-подобные
uzilish	interrupt	прерывание
vositali manzillash	indirect addressing	косвенная адресация
yig'uvchi	assembler	ассемблер
yuzaga keltirmoq	cause, generate, issue	генерировать

Ilova G. Foydalanilgan adabiyotlar

- *Paul A. Carter*. PC Assembly Language. 2006. Oklahoma.
- *The NASM Development Team*. NASM — The Netwide Assembler. 2008.
- *Randall Hyde*. The Art of Assembly Language. No Starch Press, 2010. California.
- *Jonathan Bartlett*. Programming from the Ground Up. Bartlett Publishing, 2004. Oklahoma.
- *В. И. Юров*. Assembler. Учебник для вузов. Питер, 2003. Санкт-Петербург.
- *Питер Абель*. Ассемблер и программирование для IBM PC. Питер, 1999. Санкт-Петербург.
- *Пирогов Владислав Юрьевич*. Ассемблер для Windows. Издатель Молгачева С.В., 2002.
- *Зубков С.В*. Assembler. Язык неограниченных возможностей. Издательство «ДМК Пресс», 1999.
- *Randall Hyde*. Writing Linux Device Drivers in Assembly Language. O'Reilly Press, 2001. California.
- *Daniel P. Bovet , Marco Cesati.* Understanding the Linux Kernel. O'Reilly Press, 2000. California.
- *Эндрю Таненбаум*. Современные операционные системы. Питер, 2002. Санкт-Петербург.
- *Столяров А. В., Головин И. Г., Волкова И. А.* Операционная система UNIX. МГУ им. Ломоносова, факультет ВМК, 2006. Москва.
- *Староверов, Корнев.* Работа в ОС UNIX. МГУ им. Ломоносова, факультет ВМК, 1999. Москва.
- *Jack Koziol, David Litchfield Dave Aitel, Chris Andley, Sinan Eren, Neel Mehta, Riley Hassel.* The Shellcoder's Handbook: Discovering and Exploiting Security Holes. John Wiley & Sons, 2004.
- *Machtelt Garrels.* Introduction to Linux. A Hands on Guide. Fultus Corporation, 2004.
- *Greg Kroah-Hartman.* Linux Kernel in a Nutshell. O'Reilly Press, 2009. California.
- *Mike Banahan, Declan Brady and Mark Doran*. The C Book: Featuring the ANSI C Standard. Addison-Wesley, 1991. Massachusetts.
- *Brian Kernighan, Dennis Ritchie*. The C Programming Language. Prentice Hall, 1988. New Jersey.
- *Sharam Hekmat.* C++ Essentials. PragSoft Corporation, 2005. Ahmedabad.
- *David Salomon*. Assemblers And Loaders. Prentice Hall, 1993. New Jersey.
- *Richard Mansfield*. Machine Language for Beginners. Compute! Publications, 1987.
- *Richard Mansfield*. The Second Book of Machine Language. Compute! Publications, 1984.
- *Henry Takeuchi*. Win32 Programming for x86 Assembly Language Programmers. OOPWeb.com, 2007.
- Axborot-Kommunikatsiya Texnologiyalari Izohli Lug'ati. BMTTDning O'zbekistondagi vakolatxonasi, 2010. Toshkent.

Internetdagi manbalar:

- IA-32 Intel® Architecture Software Developer's Manual. Volume 1: Basic Architecture www.intel.com/products/processor/manuals/ Intel Corporation.
- Microsoft MSDN Library Documentation: http://msdn.microsoft.com/library/default.aspx Microsoft Corporation.

- The Netwide Assembler: NASM: http://developer.apple.com/library/mac/#documentation/DeveloperTools/nasm/nasmdoc0.html Apple Inc.
- The Netwide Assembler: http://www.nasm.us/
- GCC online documentation: http://gcc.gnu.org/onlinedocs/
- Assembler tutorial: Hello world with NASM and CL.EXE or LINK.EXE: http://www.art0.org/techtips/reversing/assembler-tutorial-hello-world-with-nasm-and-cl-exe-or-link-exe
- Win32NASM the unofficial homepage: http://rs1.szif.hu/~tomcat/win32/
- Расширенный ассемблер: NASM: http://www.codenet.ru/progr/asm/nasm/ AsmOS group.
- Assembly Tutorial: http://www.geocities.com/SiliconValley/6112/asm0100.htm Ferdi Smit.
- FreeBSD Assembly Language Programming: http://www.int80h.org/ G. Adam Stanislav.
- Linux Assembly Tutorial Step-by-Step Guide: http://www.cin.ufpe.br/~if817/arquivos/asmtut/ Derick Swanepoel.
- Writing A Useful Program With NASM: http://leto.net/ Jonathan Leto.
- 80x86 Assembly with Masm: http://www25.brinkster.com/cheeseball Ferguson Travis.
- LINUX System Call Quick Reference: http://www.bigfoot.com/~jialong_he Jialong He.
- Open a file in MS-Dos: http://codeunivers.com/source-codes/assembler/open_file_nasm_dos
- Linux Programmer's Manual. OPEN (2): http://www.kernel.org/doc/man-pages/online/pages/man2/open.2.html
- Windows System Call Table: http://dev.metasploit.com/users/opcode/syscalls.html
- IA-32 Assembly Language Reference Manual: http://download.oracle.com/docs/cd/E19455-01/806-3773/index.html Oracle Corporation.

Jo'rayev Fazliddin Shahriyevich

ASSEMBLER TILI VA KOMPYUTERDAGI JARAYONLAR

Toshkent – «Fan va texnologiya» – 2012

Muharrir: *F. Ismoilova*
Texnik muharrir: *M. Xolmuhamedov*
Musavvirlar: *H. G'ulomov, S. Jo'rayeva*
Musahhiha: *M. Hayitova*
Kompyuterda sahifalovchi: *N. Hasanova*

Nashr.lits. AI№149, 14.08.09. Bosishga ruhsat etildi: 16.07.2012-yil.
Bichimi 60x84 $^{1}/_{8}$. «Tahoma» garniturasi. Ofset usulida bosildi.
Shartli bosma tabog'i 21,5. Nashr bosma tabog'i 21,75.
Tiraji 500. Buyurtma № 69.

«Fan va texnologiyalar Markazining bosmaxonasi» da chop etildi.
100066, Toshkent shahri, Olmazor ko'chasi, 171-uy.