Коммерциялық емес акционерлік қоғам

АЛМАТЫ ЭНЕРГЕТИКА ЖӘНЕ БАЙЛАНЫС УНИВЕРСИТЕТІ

Ақпараттық жүйелер
кафедрасы

АССЕМБЛЕРДЕ ПРОГРАММАЛАУ

5В060200 – Информатика мамандығы студенттеріне арналған дәрістер жинағы

Алматы, 2017

2017 ж. баспа жоспары, реті 225

Айжан Токжумаевна Купарова
Гулшат Бимуратовна Альмуратова

АССЕМБЛЕРДЕ ПРОГРАММАЛАУ

5В060200 – Информатика мамандығының студенттеріне арналған дәрістер жинағы

Редактор Ж.Н. Изтелеуова
Стандарттау бойынша маман Н.Қ. Молдабекова

Басуға_____ қол қойылды	Пішімі 60х84 1/16
Таралымы 15 дана.	Баспаханалық қағаз №1
Көлемі 4,75 есептік-баспа табақ	Бағасы 2375 тг Тапсырыс_____

«Алматы энергетика және байланыс университеті»
коммерциялық емес акционерлік қоғамының
көшірмелі-көбейткіш бюросы
050013, Алматы, Байтұрсынұлы көшесі, 126

ҚҰРАСТЫРУШЫЛАР: А.Т. Купарова, Г.Б. Альмуратова. «Ассемблерде программалау». 5В060200 – Информатика мамандығының студенттері үшін дәрістер жинағы. – Алматы: АЭжБУ, 2016. – 74 б.

Бұл дәрістер жинағы «Ассемблерде программалау» курсының программасына сәйкес 5В060200 – Информатика мамандығы студенттері үшін құрастырылған- Алматы: АЭжБУ, 2015 – 74 б.

Пікір беруші: т.ғ.к., доцент Матаев У.М.

«Алматы энергетика және байланыс университеті» коммерциялық емес акционерлік қоғамының 2017 жылғы жоспары бойынша басылады.

Мазмұны

Кіріспе

Ассемблер бағдарламалардың төменгі деңгейдегі тілі болып табылады. Ол үйренуде біршама қиындау, ал, екінші жағынан алғанда көптеген және әртүрлі барлық программа тілдерінің ішіндегі ең жылдамы болып табылады. Машиналы-бағытталған программа электронды есептегіш машиналарды жасаумен қатар пайда болды. Бастапқыда бұл машиналық кодтар еді, содан кейін Assembler (Автокод) программалау тілі пайда болды. Программалардың бұл түрі ЭЕМ архитектурасын және басқару жүйесін нақты егжей-тегжейлі білудің мүмкіндіктерін жобалайды, және басқалары дәрменсіз болып қалғанға дейін қолданылады, немесе осы ЭЕМ архитектурасының деректерін қолданумен осы және басқа басқару жүйелерінің шеңберінде жоғары деңгейдегі жедел әрекеттерді алуы тиіс. Компьютердің барлық мүмкіндіктерін тиімді пайдалану үшін машина тілінің символдық аналогы ретінде - ассемблер тілі қолданылады. Есептегіш техникалардың даму тарихы жүйелі программаларды қамтамасыз етудің дамыту тарихымен нық байланысты. Заманауи компьютерлік жүйелер қолданбалы программаларды қамтамасыз етумен бірге, әрқашанда есептеу барысын ұйымдастыруды қамтасыз ететін жүйеллерді қатар ұстайды.

«Ассемблерде бағдарламалау» курсының мақсаты ЭЕМ қалай жұмыс істейтінін айқын түсініп және нақты міндеттерді талқылауға икемді болу, кез келген алгоритмдік тілде сауатты бағдараламалар жасау үшін ЭЕМ–нің қызмет етуін және ұйымдастыру ерекшеліктерін үйрену; ассемблер тілінің негізгі түсінігін, Intel процессордың негізінде компьютердің архитектурасын, DOS және Windows үшін қолданбалы және жүйелік программаларды қоса алғанда ассемблерде бағдарламалаудың заманауи негізгі аспектілерін үйрену болып табылады.

Бұл әдістеме көрсеткіштерінде ассемблер тілі – машина тілінің символикалық түрі ретінде қарастырылады. Машинадағы барлық процестер, машина тілінің ең төменгі аппараттық деңгейінде тек командалармен (нұсқаулықтармен) жүзеге асырылатыны келтіріледі. Бұдан ассемблер тілінің компьютердің қай түріне болмасын сай келетіні түсінікті. Бұл ассемблерде жазылған бағдарламаның осы тіл көрінісі болып табылатын сыртқы түріне де, идеясына да қатысты.

1 Дәріс №1. ЭЕМ архитектурасы

Дәрістің мақсаты: IBM PC-ның архитектуралық ерекшеліктерін оқу.

Дәрістің мазмұны: ЭЕМ архитектурасының түсінігі; адрестік кеңістікке бөлу.

1.1 ЭЕМ архитектурасының түсінігі

ЭЕМ архитектурасы – ол ЭЕМ-ді оның құрылысын, тізбетехникалық және логикалық ұйымдастыруын көрсететін абстракті түрде көру. ЭЕМ архитектурасының түсінігі, өзіне:

- ЭЕМ-нің құрылымдық тізбесін;
- ұйымдастыру және ЭЕМ интефейстерінің дәрежелігін;
- теру және регистрлердің қолжетімділігін;
- ұйымдастыру және жадыны бағыттау тәсілдерін;
- ұсынудың тәсілдері және ЭВМ-ның деректер форматын;
- ЭЕМ-де машиналық командаларды теруін;
- машина командаларының форматтарын;
- кездейсоқ жағдайларды өңдеуді (үзуді) қосатын жиынтық болып табылады.

Демек, архитектура түсінігі программистер үшін қажет компьютер туралы ақпараттардың барлығының жиынтығы деп түсінуге болады. Заманауи ЭЕМ-дердің барлығы архитектураның кейбір жалпы және дара қасиеттеріне ие болып отыр. Оны ірілі-ұсақты туыстастарынан ерекшелендіретін компьютердің тек нақты үлгісіне ғана тән дара қасиеттер. Онда жалпы архитектуралық өзіне тән ерекшеліктердің болуы, бар машиналардың түрлерінің көпшілігі ЭЕМ-нің фон-нейман архитектурасы деп аталатын 4 және 5 буындарына жатқаны себеп болып отыр. Жалпы архитектуралық ерекшеліктердің және қағидалардың қатарына:

а) сақталатын бағдаралама қағидасын;

б) микробағдарламалау қағидасын;

в) жадының сызықтық кеңістігін;

г) бағдарламаны орындау бірізділігін;

д) деректерді мақсатты белгілеуге талғаусыздығын кіргізуге болады.

Компьютердің көмегімен ақпарат өңдеудің толық функционалдық іскер құралын құрастыратын өзара байланысқан барлық блоктардың жиынтығы компьютерлік жүйе деп аталады.

Стандартты компьютерлік жүйенің негізгі компоненттері: жүйелік блок, бейнемонитор, пернетақта, баспа құралы, дисководтар және әртүрлі асинхронды байланыс және ойын бағдарламаларды басқару үшін әртүрлі қаражаттар. 1 суретте микропроцессорларының негізінде Intel P6 санатына жататын Pentium Pro/II, III компьютерлер тізбесі көрсетілген. Тізбеде: орталық процессор, жедел сақтау жадысы, сыртқы құрылғылар. Жүйелі шина арқылы барлық компоненттер өзара байланысқан. Жүйелі шинаның қосымша

шинасы - кеңейменің шинасы болады. Микропроцессордың негізін микропрограммалық басқаруды блок, атқарушы құрылғы, регистрлер құрайды. Микропроцессордың қалған құрамдастары көмекші функцияларды орындайды. ЭЕМ ақпараттарды түрлендіргіш болып табылады.

Ондағы бастапқы деректердің міндеттер оның шешімінің нәтижесіне айналады. Машинаның ақпаратын ұсынудың пайдаланылатын пішінімен сәйкес: үздіксіз іс-әрекет - аналогты және дискретті іс-әрекет – сандық болып екі сыныпта бөлінеді. Ақпаратты ұсынудың сандық нысаны әмбебап болуының себебінен электронды есептеуіш машиналар өз алдына ақпаратты өңдеу құралының анағұрлым әмбебап түрін көрсетеді. ЭЕМ-нің негізгі қызметі – бағдараламалық басқарудың негізінде есептеу үрдісін автоматтандыру, арифметикалық және логикалық операцияларды орындаудағы үлкен жылдамдық, әртүрлі ақпараттардың көп мөлшерін сақтау мүмкіндігі, математикалық есептер мен ақпаратты өңдеу есептерінің үлкен аумағын шешу мүмкіндігі. Басқарушы ЭЕМ объектіні және өндірістік үрдісті басқаруға арналады. Объектімен байланыстыру үшін оларды датчиктермен қамтамасыз етеді. Датчиктерден түсетін сигналдардың тоқтаусыз түсініктері ұқсас сандық түрлендірушілердің көмегімен басқару алгоритміне сәйкес ЭЕМ-ге енгізілетін сандық сигналдарға түрлендіріледі. Сигналдардың талдауынан кейін ұқсас сандық түрлендірушілердің көмегімен ұқсас сигналдарға түрлендірілетін басқарушы әсерлер қалыптасады. Атқарушы механизмдер арқылы объектінің күйі өзгереді.

Әмбебап ЭЕМ-дердің құрылымдары ЭЕМ-дерді әзірлеу кезінде нақтыланбайтын есептердің үлкен аумағын шешуге арналған. ЭЕМ-дердің заманауи архитектуралық салаларының мысалы: дербес ЭЕМ (IBM ОС және Apple Macintosh – үйлесетін машиналар), айрықша ақпаратты өңдеуге арналған машиналар (Targa, Silicon Graphics графикалық станциялары), үлкен ЭЕМ (IBM, Cray, ЕС ЭЕМ мэйнфреймдері). Жүйелік БҚ-ның жалпы міндеті – программист немесе пайдаланушы мен ЭЕМ-нің аппараттық бөлігі арасындағы интерфейсті қамтамасыз ету (операциялық жүйе, қамтама-бағдарламалар) және қосалқы қызметтерді орындау (утилита-бағдарламалар).

Заманауи операциялық жүйе келесілерді қамтамасыз етеді:

1) Бағдарламаларға басқаруды тапсыру жолымен үрдісті басқару.

2) Үзілістерді қайта өңдеу, ресурстарға рұқсатты синхрондау.

3) Жадыны басқару.

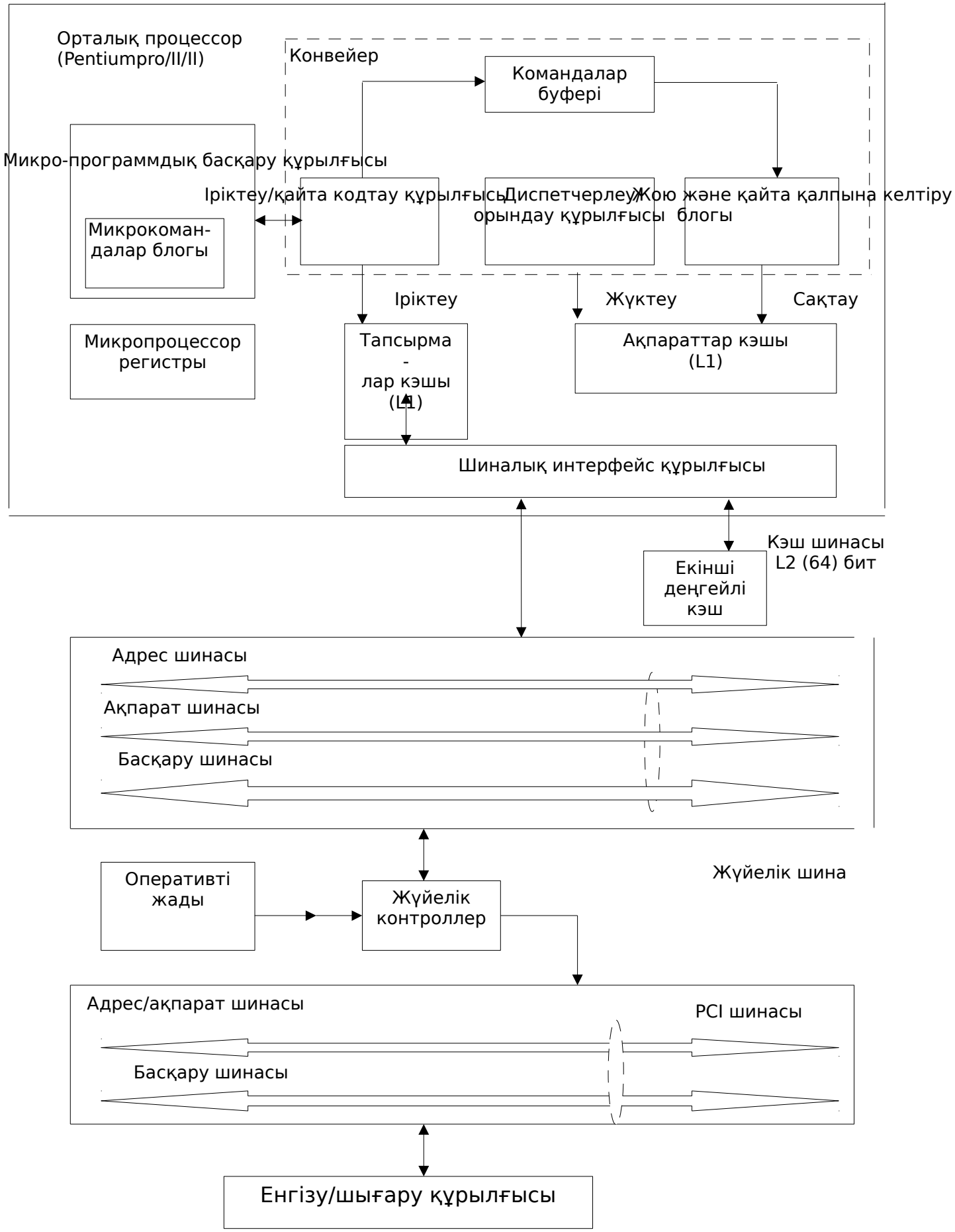
4) Енгізу-шығару құрылғыларын басқару.

5) Бағдарламалар инициализациясын басқару, программааралық байланыстар.

6) Файлдық жүйені қолдау арқылы ұзақ мерзімді сақтаушылардағы ақпаратты басқару.

Архитектура – ЭЕМ-нің жүзеге асуына көмектесетін техникалық құралдар мен олардың конфигурацияларының жиынтығы. 5-буын ЭЕМ-дері, әдетте, шиналық архитектураға ие, ол барлық құрылғылардың шина деп аталатын жалғыз электрлік желіге қосылатынын білдіреді. Егер құрылғы

шинаға сигнал шығарса басқалары оны оқи алады. Бұл сипаты ақпараттар көлемін ұйымдастыру үшін қолданылады. Осы мақсатта шина 3 адреске бөлінген – адрестің шинасы, ақпараттар шинасы және басқарушы сигналдың шинасы. Сонымен қатар, барлық заманауи ЭЕМ-дер әртүрлі құрылғыларда шинадағы ресурстардың орын алу тәртібін анықтайтын шинаның төрешісі деп аталатын құрылғыдан тұрады. PC-да ISA, EISA, PCI, VLB шиналары кең таралған.



1 сурет - Дербес компьютердің құрылымдық сызбасы

1.2 Адрес кеңістігін бөлу

Дербес компьютердің түрленуіне және оның перифериялық бөлу құрамына өзгерістер енгізуге байланысты адрестік кеңістік сәл өзгеше болуы мүмкін.Дегенмен, негізгі құрамдас компоненттерді орналастыру жүйесі қатаң түрде бірыңғайландырылған.

Негізгі жады шеңберіне, бірінші 640 Кбайт адрестік кеңістіктің 00000h тан 9FFFFh қа дейінгі адрестер жатады, оларды қазіргі күнге дейін стандартты (conventional) деп атайды.

Жедел жадының бастапқы килобайты үзу векторларымен толтырылған (4 байтты 256 векторлар) . Үзілу векторларының кейін деректер аймағы BIOS орналасады, ол 00400h тан 004FFh қа дейінгі адрестерді алады . Бұл аймақта BIOS бағдарламасын перифериялық құрылғылар көмегімен түрлі деректерді сақтау барысында, векторлардан кейін орналасатындарға жататындар:

- көрсеткіштер жүйесі арқылы жасалған буферлік енгізу пернетақтасы;
- параллель және тізбекті порттардың адрестері;
- видео реттеу жүйесін сипаттайтын деректер (курсордың формасы, және оның экрандағы ағымда орналасуы,ағымдағы бейне режимі, экранның ені және т.б.)
- уақыт санау үшін ағымдағы тор;
- есепаралық байланыстар аумағы және т.б.

BIOS деректер аумағы бастапқы компьютер және қажетттілік жағдайына байланысты динамикалық жүйенің түрленуінің және жүктелуінің арқасында ақпаратпен толтырылады; көптеген қолданбалы программалар бұл аумаққа ондағы қамтылған ақпаратты оқу немесе түрлендіру мақсатымен қолданылады.

Жад аймағында, 500H адресінен бастап, кейбір жүйелік DOS деректерін қамтиды. DOS деректер аумағынан кейін, IO. SYS и MS DOS.SYS (IBMBIO.COM және IBMDOS.COM PC-DOS жүйесі үшін) файлдары арасынан жүктелетін операциялық жүйелері орналасады. Жүйе әдетте бірнеше ондаған килобайт алады.

Операциялық жүйенің жоғарыда көрсетілген компоненттері, әдетте 60-90 Кбайт орын алады.Барлық қалған жад640 килобайт шегіне дейінгі (кейде транзиттік аумағы деп аталады) кез келген жүйелер немесе қолданбаларды жүктеу үшін еркін болып табылады.Адрес кеңістігін қалған 384 Кбайт, жоғарғы (upper) жад бастапқыда үздіксіз есте сақтау құрылғысын (ҮЕҚ) орналастыру үшін әзірленген.Іс жүзіндеүздіксіз есте сақтау құрылғысының тек адрес бөлігі ғана бос. Адрестік кеңістіктің ең соңында, F0000h…FFFFFh (немесе E0000h…FFFFFh) аумағында, BIOS үздіксіз еске сақтау құрылғысы, ал C0000h адресінен бастап BIOS кеңейтулерінің графикалық адаптерлері мен дискілеріне қызмет етуге арналған аталмыш ҮЕҚ орналастырылады. Суретте көрсетілген видеобуферлердің орналасуы EGA адаптеріне тікелей

байланысты;басқа адаптерлерге ол өзгеше болуы мүмкін (мысалы, қарапайым монохромдыMDA адаптерінің видеобуфері 4 Кбайт орын алады және B0000h адресінен бастап орналасады.)

80386/486 – 4 Гбайт процессорын қолдану барысында максимальды ауданның адресті процессор шинасына байланысты, стандартты жады (640 Кбайт) PC/AT компьютерлер құрамына енеді.100000h адресінен бастап, бұл жады мекенжай кеңістігін бірінші мегабайтынан тыс орналасқан. Кеңейтілген жадыны пайдалану қызметі «Кеңейтілген жадтың техникалық сипаттамаларына» бағынған(Extended Memory Specification, қысқартылған түрі XMS), кейде жадының өзін де XMS – жады деп атайды. Жоғарыда атап өтілгендей, кеңейтілген жадыға қосылу қорғалған режимінде орындалады, сондықтан тек нақты режимде жұмыс істейтін MS DOS үшін кеңейтілген жад қол жетімсіз болып табылады. Ең бірінші 64 Кбайт кеңейтілген жадының, нақтылап айтқанда,100000h тан 10FFEFh қа дейінгі байт адрестерімен сипатталатын, арнаулы жоғарғы жады(High Memory Area, HMA) атауымен қолданылады. Бұл аймақтың негізгі ерекшелігі, бірінші мегабайт шегінен тыс болады,оған микропроцессордың нақты режиміндегі күйімен қатынасуға болады.

2 Дәріс №2. Ассемблер бағдарламалау тілі және оның конструкциясының микропроцессор архитектурасымен байланысы

Дәрістің мақсаты: микропроцессордың ішкі архитектурасын зерттеу.

Дәрістің мазмұны: ЭЕМ - нің қасиеттері, сегментті адресация, жұмыс режимдері.

2.1 ЭЕМнің қасиеттері

ЭЕМ ақпаратты түрлендіргіштер болып табылады. Мәннің бастапқы деректері оның шешіміне түрлендіреді. Қолданбалы формасына байланысты машиналы ақпараттың көрсетілуі екі класка бөлінеді: үздіксіз қызмет ететін-аналогты және дискретті қызмет ететін- сандық. Сандық форманың жан-жақтылық әсерінен ақпараттың, электронды есептеуіш машинаның көрсетілуі ақпаратты өңдеудің әмбебап түрі болып табылады. ЭЕМнің негізгі қасиеттері – программаны басқару барысында есептеу үдерісін автоматтандыру, жоғары жылдамдықта арифметикалық және логикалық операцияларды орындау, әртүрлі мағлұматтарды сақтау мүмкіндігі, математикалық есептер мен деректерді өңдеу есебін шығару мұмкіндігі. ЭЕМнің айрықша мәнділігі, олардың ең бірінші пайда болғанынан бастап адам баласы автоматтандыру барысында мәліметтерді өңдеуге үлкен мүмкіндік алды. Нысанмен байланысу барысында оларды құрылғымен жабдықтайды.

Датчиктерден түсетін сигналдардың үздіксіз түсініктері ұқсас сандық түрлендірушілер арқылы басқару тәртібіне сәйкес ЭЕМ-ге енгізілетін сандық сигналдарға түрлендіріледі. Сигналдардың талдауынан кейін ұқсас сандық

түрлендірушілердің көмегімен ұқсас сигналдарға түрлендірілетін басқарушы әсерлер қалыптасады. Атқарушы механизмдер арқылы объектінің күйі өзгереді.

Әмбебап ЭЕМ-дер құрамы ЭЕМ-ді әзірлеу кезінде нақтыланбайтын тапсырмалардың үлкен көлемін шешуге арналған. ЭЕМ-дердің заманауи архитектуралық салаларының мысалы: дербес ЭЕМ (IBM ОС және Apple Macintosh – үйлесетін машиналар), айрықша ақпаратты өңдеуге арналған машиналар (Targa, Silicon Graphics графикалық станциялары), үлкен ЭЕМ (IBM, Cray, ЕС ЭЕМ мэйнфреймдері).

Кез келген микропроцессордың ең маңызды сипаттамасы болып оның ішкі тіркелімдерінің, сонымен қатар сыртқы адрес пен ақпарат шиналарының разрядтылығы табылады. i8086 микропроцессорында 16-разрядты ішкі архитектура және дәл осындай разрядты ақпарат шинасы бар. Осылайша, микропроцессор қызмет ете алатын максималды бүтін сан (берілген немесе адрес) құрайтыны: 2^{16}-1=65535 (64К-1). Алайда i8086 микропроцессорының адрестік шинасы 2^{20}=1 Мбайт адрестік кеңістігіне сәйкес келетін 20 желіден тұрады. 16-разрядты адрестер арқылы 20-разрядты адрестік кеңістіктің кез келген нүктесіне қатынаса алу үшін микропроцессорда төрт сегментті регистрлердің көмегімен жүзеге асатын жадының сегменттік адрестеуі қарастырылған.

Сегменттік бағдарланудың маңызы келесіде: жадының кез келген ұяшығының атқарушы 20-разрядты адресі әдетте қатыстық адрес деп атайтын, бастапқы сегменттен оған қарай жылжытылған (байттарда), осы ұяшық орналасқан жады сегментінің бастапқы адресін қосу арқылы есептеледі. Төрт кішкентай битсіз сегменттік адрес, яғни 16-ға бөлінген, сегменттік регистрлердің бірінде сақталады. Атқарушы адресті есептеу кезінде процессор сегменттік регистрдің құрамын 16-ға көбейтеді (солға қарай 4 екілік разрядқа жылжыту арқылы) және шыққан 20-разрядты адреске қатыстық адресті қосады. Негізге адресті 16-ға көбейту адрестелуші ұяшықтардың аумағын 64 Кбайт · 16 =1 Мбайт шамасына дейін көбейтеді.

IBMPC/AT компьютерлерінің орталық процессоры ретінде қолданылатын i80286 микропроцессоры i8086-тің жадыны басқару және оны қорғау сызбаларымен толықтырылған, жетілдірілген нұсқасы болып табылады. i80286 микропроцессоры 16-разрядты операндылармен жұмыс істейді, бірақ 2^{24}=16 Мбайт адрестік кеңістігіне сәйкес келетін 24-разрядты адрестік шинаға ие. Алайда жоғарыда сипатталған жадыны сегменттік адрестеу әдісі 1 Мбайт шегінен асуға мүмкіндік бермейді. Бұл шектеуден өту үшін i80286 микропроцессорында (сондай-ақ, i80386 микропроцессорындағыдай) жұмыстың екі режимі қолданылады: нақты адрес және виртуалды қорғалған адрестің немесе қарапайым қорғалған режим. Іс жүзіндегі режимде i80286 микропроцессоры шынында i8086 микропроцессорындай жоғарғы әрекет етумен қызмет етеді және адрестік кеңістіктің 1 Мбайтына ғана қатынаса алады. Жадының қалған 15 Мбайты олар компьютерде орнатылған болса да қолданыла алмайды.

Қорғаныс режимінде сол қалпынша сегменттер мен олардағы орын ауыстырулар қолданылады, бірақ бастапқы адрестер сегменттік регистрлерді 16-ға көбейту арқылы есептелінбейді, ал сол сегменттік регистрлердің өздерімен индекстелетін сегменттік дескрипторлар кестесінен алынады. Әрбір сегменттік дескриптор 6 байт орын алады, оның ішіндегі 3 байты (24 екілік разрядты) сегментастылық адреске жатады, осылайша 24-разрядты адрестік кеңістіктің толық қолданылуын қамтамасыз етеді.

Әрбір сегменттік регистрда сегменттік дескрипторлар кестесінің индексі үшін 14 екілік разряд бөлінеді. Адрестелетін ұяшықтың толық логикалық адресі сегмент нөмірінің 14-разрядты индексі және 16-разрядты қатыстық адрестен тұрады. Бұл әрбір программаға 2^{30}=1 Гбайт-қа дейін логикалық немесе виртуалды кеңістікті қолдануға мүмкіндік береді, ал ол осылайша өзінің көмегімен физикалық жадының максималды мүмкін болатын көлемінен 64 есеге асады. Виртуалды жадының операциялық жүйесі оперативті жадыға қажетінше қандай да бір сегменттерді автоматты түрде жүктеу арқылы үлкен дискілік кеңістікте орындалатын программалардың барлық сегменттерін сақтайды.

i80386 және i80486 микропроцессорлары 32-разрядты ақпарат шиналары және 32-разрядты ішкі архитектуралары бар жоғарғы өндірістік процессорлар болып табылады. Соңғы айтылғандар алдыңғы модельді процессорларға қарағанда бұл процессорлардың ішкі регистрларының ұзындығы 31-битті дегенді білдіреді. Сондықтан микропроцессор жұмыс істей алатын мксималды нақты сан 2^{32}-1 = 4294967296 (4 Гбайт)-ты құрайтынын білдіреді. Көп жағдайда 32-битті операндыларды қолдану есептеулерді жеңілдете және жылдамдата алады. Сонымен қатар, i80386 және i80486 микропроцессорларында регистрлер құрамы кеңейтілген, ол да өз алдында программистке айтарлықтай жеңілдіктер көрсетеді. Сонымен, процессорлардың жаңа модельдерінде көп тапсырмалық режимді, сондай-ақ көп процессорлық жүйелерді қолдаудың ішкі құралдары бар. Әрине, бұл процессорлар да i80286 микропроцессоры сияқты нақты және қорғалған режимде қызмет ете алады. Соңғы айтылған жағдайда микропроцессор 2^{32}=4 Гбайтқа дейінгі физикалық және 2^{46}=64 Гбайтқа дейінгі виртуалды жадыны адрестеуге мүмкіндік береді. Сонымен бірге өндірушілер процессорлардың жаңа және ескі моделдерінің толық үйлестігін қамтамасыз еткендігін ескерте кеткен жөн, яғни 16-битті операндылардыларды қолдану арқылы i8086-i80286 процессорлары үшін жазылған программалар жаңа процессорларда ешбір түзелтулерсіз орындала береді.

3 Дәріс №3. Ассемблердегі бағдарламаның өмірлік циклі. Ассемблердегі бағдарлама құрылымы

Дәрістің мақсаты: ассемблердегі бағдарламаны әзірлеу кезеңдерін зерттеу.

Дәрістің мазмұны: енгізу, трансляция, программаны орындау және редакциялау.

3.1 Ассемблердегі бағдарламаны әзірлеу кезеңдері

Ассемблер тілі машиналы–жабдықталған тілдер қатарына жатады. Бұл тілдегі бағдарлама ақпаратты басқарады,бұл бағдарламамен жұмыс істейтін дерексіздік деңгейі ЭЕМ архитектурасына сәйкес келеді:

- ақпаратты өңдеу процесін басқару шарты, ақпаратпен қамтамасыз ету жүйесі және оны қолдану ережелері, ЭЕМ-де орындалатын ақпараттың көптеген түрімен;
- бағдарламаларда мәліметтерді бағыттау жолымен, мәліметтермен жабдықтау жүйесі;
- ЭЕМ мен сыртқы орта арасында ақпарат алмасуды ұйымдастыру, ЭЕМ жұмысын басқару процестерін синхрондау тәсілдері.

Бағдарламалармен ассемблер тілінде жұмыс істеу барысында, ассемблирлеу жүйесін, бағдарламаны орындау және топтауды білу қажет.

3.2 Бағдарламаның негізгі мәтінін енгізу

Бағдарламаның негізгі мәтінін енгізу үшін, кез келген мәтінді өңдеу редакторы қолданылады, сондай-ақ бастапқы модульде кеңейтілген asm. болуы қажет. Мысалы, NORTONEDITOR-ды қолдануға болады.

Shift+F4

Ассембрлер тілінде жазылған бағдарламалар келесі түрде жазылады: атауы -*бағдарлама. asm* кіші немесе үлкен әріптермен теріледі. Операторлар жолдарының арасында бір пробел қалдыру міндетті. Бағдарлама листингін оқуды ыңғайлы ету үшін, бағандарға сәйкес ескертпелер, сілтемелер және анықтамалар жазып отыру керек. Бағдарламада сілтемелерді қолдану оның анықтылығын жақсартады. Сілтеме әрқашан негізгі модульдің кез келген жолынан «нүктелі үтір» (;) белгісінен басталады. Сілтеме барлық жолды алуы мүмкін немесе бастапқы немесе келесі жолда тобымен қатар жүруі мүмкін.

3.3 Бағдарламаны көрсету

Негізгі бағдарламаны енгізу үшін, басты екі қадам жасау қажет. Ең бірінші бағдарламаны ассембрлеу керек, ал содан кейін тұтастыру қажет. Ассембрлеу негізгі модульді көліктік объективті кодта көрсетуді және OBJ-модулінің генерациясын қамтиды. Бұл қадамды орындау барысында, көліктік кодтардағы негізгі бағдарламалар туралы мәліметті, сонымен қатар оны басқа модульдармен тұтастыру үшін қажет ақпараттарды қамтитын объективті модуль тұжырымдалады.

TASM.EXE-ні іске енгізу үшін командалық жолдардың келесі форматын орындау керек:

TASM.EXE негізгі құжат атауы. asm

Экранда пайда болады:

- source filename (.ASM): с: құжат атауы м[ENTER];
- object filename (filename.OBJ): с: [ENTER];
- source listing (NUL.LST): с: [ENTER];
- cross-reference (NUL.CRF): с: [ENTER].

LST кеңейтілген файлы жинақталатын машина кодын және жол нөмірін көрсетеді. CRF кеңейтілген программасы - бұл қандай команда қай жердегі деректерге сілтеме жасап тұрғанын көрсететін қиылысатын сілтемелер файлы. Одан басқа ассемблер LST-файлда CRF файлда пайдалынатын жолдардың нөмірлерін жинақтайды.

Листингтің мазмұны үш бағана түрінде беріледі:

- 1 бағана –сегменттің басынан бастап адресті жылжытуы төрт буынды он алтылық мағынасы орнын алу;
- 2 бағана – әр команданың объективті кодын көрсетеді. Сегмент үшін стектің және деректердің – бұл он алтылық мағынасына сәйкес константт;
- 3 бағана – бастапқы программаның мәтіні.

3.4 Бағдарламаны тұтастыру

Модульдің OBJ форматы - тіпті атқарушы формаға жақын, бірақ орындауға дайын емес. Тұтастыру қадамы OBJ- модулі EXE (орындалатын) модулінде түрленуді қосады.

TLINK.EXE іске қосу үшін командалық форматтың жолдары

TLINK.EXE_ бағдарламаның атауы. obj

Тұтастырушының сұранысы	Жауап
Object Modules (.OBJ)	С: программа - атауы
Run file (прогр.-атауы, .EXE)	С:
List file (NUL.MAP):	CON
Libraries (.LIB)	[ENTER]

MAP-пен кеңейтілген файл сегменттердің атауы мен көлемімен кестені, сондай-ақ TLINK табатын қателерді көрсетеді.

3.5 Бағдарламаны орындау

Бағдарламасын іске қосу үшін орындау жеткілікті шақыруға жүктелуші модуль: аты....exe

Ассемблер синтаксисі. Ұсыныстар құрайтын бағдарламасын мүмкіндігінше синтаксистикалық конструкциясына, тиісті командада макрокоманде, директивасына немесе комментариясын көрсете алады. Ассемблер трансляторы тануы үшін, олар белгілі бір синтаксистикалық ережелерді қалыптастыруы тиіс. Ассемблер тілінің жалпы сипаттамасы. Деректер типтері. Ресімдеу. Жазу командасының форматы.

Ассемблер - машиналық-бағдарланған тіл, екі негізгі артықшылықтары бар:

1) Командалар процессоры деңгейінде бағдарлама жазуға мүмкіндік береді.

2) Командаларды білуді талап етпейді, әрқайсысы ауыстырылатын ыңғайлы есте қаларлық мнемонико - қысқартылған ағылшын сөздерінен тұрады. Трансляторы мнемоникаларды өздерініңың сандық баламаларына аударады. Тілдің элементтері: операторлар (ассемблер командалары + макроассемблердің псевдооператорлары), операндалар, өрнектер, тұрақтылар, белгілер, түсініктемелер. Ассемблердің жеке командалар процессоры – операндаларсыз, бір немесе бірнеше операндалармен қолдвнылуы мүмкін, сонымен қатар түрлі адресацияларды қолданады. Псевооператорлардың 5 түрі бар: идентификаторларды табу (EQU), мәліметтер (DB), сыртқы сілтемелер (PUBLIC, EXTRN), сегментерді табу және ішкі программалар (SEGMENT, PROC), трансляцияларды басқару (END).

Тұрақтылар сандық және литералды болуы мүмкін (әріптердің реті, апострофтардың аяқталуы).

Комментариялар таңбалардан басталады, бағдарламаның оқылуын жақсартылуы үшін тағайындалған.

Таңбалар өтуілерді ұйымдастыру үшін тағайындалған. Олар жергілікті және ғаламдық болуы мүмкін. Деректер типтері. Бүтін типтері.

BYTE - байт (бір байтты бүтін сан, символ коды, жолдар элементі).

WORD - сөз (бүтін сан белгісі немесе белгісі жоқ).

DWORD - қос сөз, ұзын бүтін.

Көрсеткіштер толық 32-биттік көрсеткіш немесе 16-битовое ығысуы. Заттай типтері (математикалық сопроцессордың түрлері) - нақты санның ұзындығы 32, 64, 80 бит. Жиымдар. Ассемблердегі мүмкін хабарландыру массивтерді сандар.

RECORD – биттік жазбалар, олардың әрқайсысы ұзындығы белгілі бір сандық бит болады және кейбір мәндерімен инициализацияланады.

STRUC – құрылым, элементі 1 немесе одан да көп типті мәліментерден тұрады, мүшелер құрылымы деп аталады.

UNION (біріктіру) - дәл сондай, құрылымы да, қоспағанда, барлық бірлестік мүшелері 1 және сол учаскесіндегі сақталған орынды алады.

Форматы командалардың тілі:

[Таңба:] мнемокод [операнд] [;комментариялар]

Әдепкі бойынша, бас және бас әріптер тіл ішінде ерекшеленбейді.

Бағдарламалауды ресімдеу:

```
;Регистрдің сегменттермен сәйкестігін көрсетейік
Assume CS:code, DS:data
;Сегмент кодын суреттейміз
;Сегмент кодын ашамыз

Code segment
```

```
Begin:
; мәліменттер сегментін DS-қа дұрыстаймыз
Mov AX, data
Mov DS, AX
; Мұнда бағдарламаның денесі қойылады
; Бағдарламаның аяқталуы
; DOS функциясы бағдарламаның аяқталуы
Mov AX, 4C00h
; DOS  үзілуінің шақырылуы
Int 21h
Code ends
; Мәліменттер сегменттін суреттейміз
; Мәліменттер сегменттін ашамыз
Data segment
; Мұнда мәліменттерді қосамыз
, Мәліменттер сегментін жабамыз
Data ends
; стек сегментін суреттейміз
; стек сегментін ашамыз
Stak segment stack
: Стек астына 256 байтты жазамыз
Db 256 dup(?)
; Стек сегментін жабамыз
Stak ends
;  Кіру нүктесі арқылы бағдарламаның соңы
End begin
```

Тұрақтылар, белгілер, шартты компиляция. Тұрақтылар сандық болуы мүмкін (ондық, екілік, оналтылық) ten EQU 10, antiten EQU - 10, bitmask EQU 10001001b, video EQU 0A000h және литералы - символдық, EQU "string data".

Тұрақтылар ассемблер тілі командасына меншіктеу үшін қызмет етеді. Ұйымдастыру үшін тағайындалған өтуілер үшін арналған. Таңбада мынадай символдар болуы мүмкін:

- A дан Z және a дан z дейінгі әріптер;
- 0 ден 9 ға дейінгі сандар;
- арнайы символдар ?; нүкте (.); астын сызу(_); коммерциялық эт (@); доллар белгісі ($).

Жаһандық таңбалар бүкіл бағдарламада жұмыс істейді. Жергілікті – тек бағдарлама ішінде жұмыс істейді. Директивалар шартты трансляциялауға арналған белгілеулері бар блоктың бағдарламалық кодқа қосылады және объектілік файл бар кезде ғана берілген шарт орындалады.

Синтаксис: Ifxxx;операторлар, ELSE шарты орындалғанда файлға көшіріледі;операторлар, ENDIF шартты орындалмағанда файлға көшірілмейді.

4 Дәріс №4. Бағдарлама жадысының бейнесі және құрылымы .EXE және . COM

Дәрістің мақсаты: бағдарлама құрылымын зерттеу.

Дәрістің мазмұны: құрылымды типтік бағдарлама типті .EXE және .COM, директиваның операнды SEGMENT.

4.1Бағдарлама құрылымының типі .EXE және .COM

Бағдарламаның басқаруымен орындалатын MS-DOS, тиесілі болуы мүмкін бір екі түрі бар: .COM және .EXE. Бағдарламаларда типті .EXE бағдарламаның коды, деректер және стек алады жекелеген сегменттеріне, ал .COM - бір сегменті.

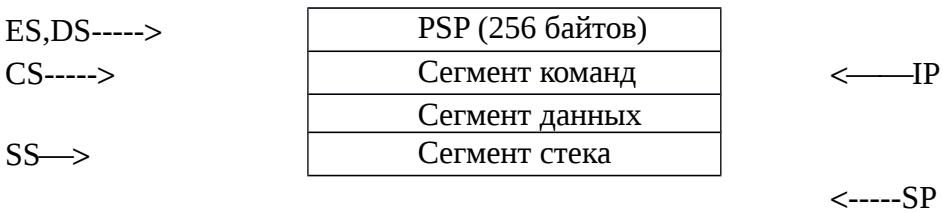
Осылайша, .COM бағдарламасының типінің мөлшері 64 Кбайттан аспауы тиіс, ал EXE бағдарламасының типінің мөлшері іс жүзінде шектелмеген, өйткені оған кез келген саны сегменттерінің бағдарламалар мен деректер кіруі мүмкін.

Жүктеу кезінде бағдарламаның сегменттері 4.1 кестеде көрсетілгендей жадыда орналастырылады.

Бағдарлама жадының суреттелуі префикс бағдарламалық сегментінен басталады. (Program Segment Prefics, PSP), құрылатын және толтырылатын жүйесінен тұрады.

PSP әрқашан көлемі 256 байт болады, құрамында кестелер және деректер өрісі бар, олар жүйесінде бағдарламаның орындау барысында пайдаланылады. PSP артынан бағдарлама сегменті орналасады. Сегментті регистрлер автоматты түрде осылайша инициализияцияланады: ES және DS PSP басында көрсетіледі, CS - сегментінің команда басында көрсетіледі, ал SS – стек сегменті басында көрсетіледі. Көрсеткішке команда IP жүктеледі салыстырмалы мекен-жайы кіру нүктесіне бағдарламасына (операнда директивасының END), стек көрсеткіші SP - стек сегментінің соңына ығысуы. Осылайша, жүктелгеннен кейін бағдарламаның жадына барлық сегменттердің, мәліметтер сегментінен басқа, адрестелгенін көрсетіледі. DS регистірінің инициализациялануы алғашқы жолдарында адрестеу және бұл сегмент бағдарламасын жасауға мүмкіндік береді.

4.1 кесте -. EXE есте сақтау бағдарламасының суреттелуі:



4.2 .COM бағдарламасының құрылымы мен жадыға шолу жасау

.COM бағдарламасының типті .EXE бағдарламасының типінен, құрамында тек бір сегментімен барлық компоненттерін қамтитын: PSP, бағдарламалық код (т. е оттранслированные машиналық кодтар бағдарламалық-жолдан), деректер және стек болуымен ерекшеленеді. .COM типінің бағдарламалық құрылымы ассемблер тіліндегі келесі түрде көрінеді (4.2-кесте).

4.2 кесте - Бағдарлама мәтіні

```
  title     Бағдарлама      типі        .COM
  text      segment         «code»
            assume          CS, DS
              org           100h
Myproc      pro
          ….  ;мәтін
Myproc        Endp
          …  ; деректер
  text      Ends
              End           Myproc
```

Бағдарлама құрамында жалғыз text сегменті бар және оған "CODE" класы байланысқан. ASSUME операторында, сегментті регистрлер CS және DS, жалғыз осы сегментке көрсетілетіні көрсетілген.

ORG 100h операторы PSP үшін 256 байтты береді. PSP бұрынғы жүйе бойынша толтырылады, бірақ оның орнына бастапқы сегментті бағдарламашы алып шығуы тиіс. Бағдарламада DS сегментті регистрін инициализациялау қажет емес, себебі басқа сегментті регистерлер тәрізді оны жүйе инициализациялайды. Деректерді бағдарлама рәсімінен кейін (суретте көрсетілгендей), немесе ішінде, тіпті болмаса алдында орналастыруға болады. Тек ескеру тиеу кезінде бағдарламаның типті .COM тіркелімі IP әрқашан инициализацияланатын саны 100h, сондықтан бірден артынан операторы ORGI 100h тұруы тиіс бірінші орындалатын жол. Егер деректер дұрыс орналастыру бағдарламаның басында, олардың алдында орналастыру керек операторы көшу нақты кіру нүктесі, мысалы, JMP Entry.

Бейнесі есте бағдарламасы типті .COM 4.3 кестесінде көрсетілген. Бағдарлама жүктелгеннен кейін барлық сегментті регистрлер басындағы жалғыз сегментті көрсетеді, яғни нақты басындағы PSP ны көрсетеді. Стек көрсеткіші автоматты түрде FFFEh саны арқылы инициализияцанады. Осылайша, бағдарламаның нақты мөлшеріне қарамастан, оған 64 Кбайт мекенжай кеңістігінен бөлінеді, бүкіл төменгі бөлігін стек алып жатады.

4.3 кесте - . COM программасының жад үлгісі

CS, DS, ES, SS—>	PSP (256 байтов)	
	Программа және деректер	<—IP
	Стек	
		<—SP = FFFEh

Сегменттің синтаксистік сипаттамасы ассемблерде конструкциясын көрсетеді, ол 2-суретте көрсетілген. Айта кету керек, функционалдық мақсаты сегменті біразы кеңірек, жай бөлу бағдарламасының блоктар код, мәліметтер және стекке қарағанда. Сегменттеу жалпы тетігін байланысты тұжырымдама, модульдік бағдарламалаудың бір бөлігі болып табылады. Ол біріздендіруді ресімдеу объектілік модульден құрылатын компилятормен, соның ішінде әртүрлі тілдер бағдарламалауды көздейді. Бұл әр тілде жазылған бағдарламаны біріктіруге мүмкіндік береді. Түрлі нұсқаларын осындай бірлестіктер мен тағайындалған операнды " SEGMENT директивасы дәл осы үшін арналған. Оларды жан-жақты қарастырайық.

Атрибут теңестіру сегментінің түрі (теңестірудін түрі) деп компоновщике бастау сегментіне арналған, берілген шекарада орналастыруды қамтамасыз етсін деп хабарлайды. Бұл өте маңызды, себебі дұрыс теңестіруді деректерге процессорларында i80x86 тезірек орындалады. Рұқсат етілген мәндері осы атрибутта мынадай:

1) BYTE - тегістеу орындалмаса, сегмент кезкелген мекен-жай жадынан басталуы мүмкін.

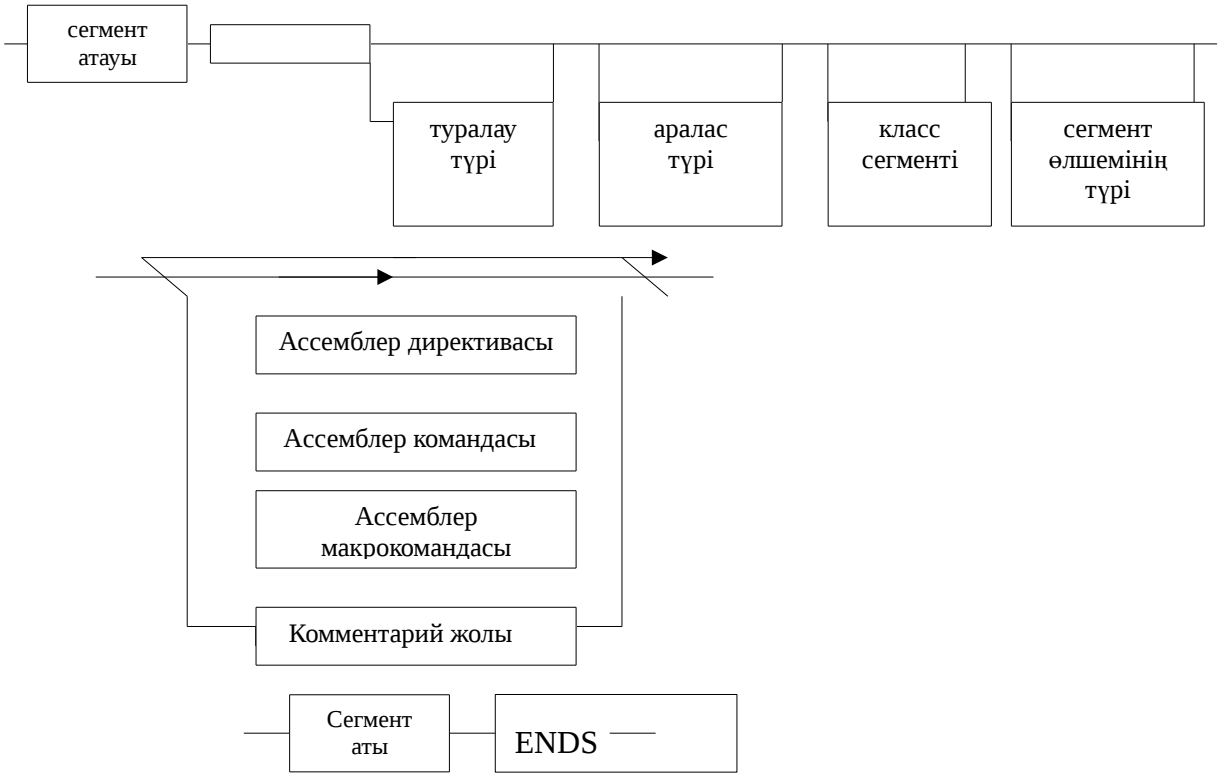
2) WORD-сегменті басталады мекен-жайы бойынша, екі есе, яғни соңғы (кіші) мәнді биттік физикалық мекен-жайы 0 тең болғанда басталады. (тегістеу шекарасы екілік сөздер).

3) DWORD - сегменті мекен-жайы бойынша еселік төрт, яғни соңғы екі (кіші) бит 0 тең болғанда басталады (тегістеу арқылы қосарланған екілік сөздер).

4) PARA-сегменті мекен-жайы бойынша 16 есе, яғни соңғы оналтылық санның мекен-жайы oh болуы тиіс, сонда басталады (параграфтың тегістеу шекарасында).

5) PAGE - сегменті мекен-жайы бойынша еселік 256, яғни соңғы екі оналтылық сандар ooh болуы тиіс, сонда басталады (тегістеу шекара беттер көлемі 256 байт).

6) MEMPAGE - сегменті мекен-жайы бойынша еселік 4 Кбайт, яғни, үш соңғы сандар oooh болуы тиіс, сонда басталады (мекен-жайы келесі беттің жады көлемі 4 Кбайт).



Үнсіздік бойынша теңестіру типі PARA мәнін қабылдайды. Сегменттерді комбинациялау атрибуты (комбинаторлық тип) жинақтаушыға әртүрлі модульдегі бір атаудан тұратын сегменттерді қалай комбинациялау керектігін хабарлайды. Үнсіздік бойынша сегменттерді комбинациялау атрибуты PRIVATE мәнін қабылдайды. Сегментті комбинациялау атрибутының мәндері келесілер болуы мүмкін:

1) PRIVATE (жеке) - сегмент аттас модуль беріледі бірақ басқа сегменттерін ұштастыру мүмкін емес болады;

2) PUBLIC (қоғамдық) - аттас барлық сегменттер компоновщик арқылы іске косылады. Жаңа аралас сегменті қауіпсіз және үздіксіз болады. Барлық нысандары, мекенжайлары (ығысулар), және ол сегментің түрі командаларға немесе деректерге байланысты болады жаңа сегментке қатысты есептелінеді.

3) COMMON (ортақ) - сол мекен-жай бойынша аттас барлық сегменттер.

Белгілі бір тақырып берілген сегменттер, жұмыла отырып жадыны қолданады.

Қорытынды бойынша алынған сегмент мөлшері, ең үлкен сегмент санына тең болады.

STACK- стек сегментінің анықтамасы. Ол бір тақырыптағы сегменттер мен олардың тақырыбын есептеуге арналған.

Үйлесімделінген STACK типі PUBLIC типіне ұқсас болып келеді және бұл ss тізіміне қоспағанда, стек сегменті тіркелімі сегменттер үшін стандартты болып табылады.

Егер бір де бір сегмент торы көрсетілмесе, компоновті стек сегментінің табылмағанын ескеретеді. Ал егер сегмент торы құрылып STACK

қолданылмайтын болса онда бағдарламашы STACK сегментінің мекен-жайын тіркеу тізіміне жүктеп алуы кажет.

Атрибут класс сегменті (класс түрі) – бұл жинақтаушыға белгілі бір тізім бойынша сегменттер мен бірнеше бағдарламаларды және модульдерді анықтауға, зерттеуге көмектеседі.

Жинақтаушыға барлық сегменттерді жадында бірдей тақырыптағы кластарды біріктіре алады (жалпы алғанда, кез келген болуы мүмкін класс атауы, бірақ ол сегменттер атауы функционалдық аудару болса жақсы болар еді).

Мысал ретінде топқа барлық бағдарлама кодтарын біріктіруді айтсақ болады. Әдете ол үшін «code» класы қолданылады.

Механизмнің көмегімен негізі кластын типтерін инициалданған немесе инициалданған еместі деп қарастыруға болады

5 Дәріс №5. Машинаның программалық қолжетімді элементтері

Дәріс мақсаты: IBM PC XT тізілімдерді зерттеу.

Дәрістің мазмұны: жалпы мақсаттағы тізілімдер, сегмент тізілімдері.

5.1 Қолданылатын IBM PC XT тізілімдер.

Құрастыру тілі бағдарламаларында тізілімдер өте қарқынды пайдаланылады. Олардың көпшілігінің нақты анықталған функционалдық мәні бар. Микропроцестердің бағдарламалық моделі бағдарламаларды қолдануға арналған бірнеше топтан тұрады.

Жалпы мақсаттағы қолданылатын тізімдер тобы :ax/ax/ah/al,ebx/bx/bh/bl,edx/dx/dh/dl,ecx/cx/ch/cl,ebp/bp,esi/si,edi/di,

esp/sp.

Бұл тізімдегі кодтар мәліметтермен, мекен-жайларды сақтау үшін қолданылады.

- тізілім сегменттері :fs, cs, ds, ss, es , gs. Осы топтың тізілімдер тізімі жадта мекенжайлардың сегменттерін сақтау үшін пайдаланылады;

- сопроцессор тізілімдер : St (0), ЖК (1), ЖК (2), ЖК (3), ЖК (4), ЖК (5), ЖК (6), ЖК (7). Осы топтың өзгермелі деректер түрін пайдаланатын жазу бағдарламалар үшін арналған.

Жағдайлық тізілімдер және бақылау тізілімдері-бұл микропроцессорлық жағдай туралы ақпаратты қамтиды, орындалатын бағдарламалық жағдайды өзгертуге мүмкүндік береді.

- жалаулар тізілімдері: eflags/flags;

- бағыттаушы тізілімдер тобы: eip/ip.

Тіркелу жүйесі - бұл әртүрлі жұмыс режимдерін, қызмет көрсету функцияларын, сондай-ақ белгілі бір тізілімдер мен микропроцессорлық моделдеу жүйесі.

5.2 Жалпы мақсаттағы тіркелімдер

Жалпы мақсаттағы тізілімдер бағдарламаларды сақтау үшін пайдаланылады:

- логикалық және арифметикалық операндты операциялар;
- компоненттерді шешу;
- жады ұяшығын көрсетушілер;

EAX / AX / AH / AЛ тізілімдер- аккумуляторы, барлық енгізу/шығару операциялары мен кейбір сызықтар мен арифметикалық операцияларды (қосу, көбейту) іске қосу үшін пайдалынады.

ebx / BX / BH / BL тізілімдер – базалық тізілімдер, жады объектінің базалық мекенжайын сақтау үшін пайдаланылады.

ECX / CX / CH / CL тізілімдер – тізілімдер санының циклдарының қайталану санын бақылау және солға немесе оңға операцияларды жылжыту үшін пайдаланылады. Сондай-ақ, есептеулерде қолданылады.

EDX / DX / DH / DL тізілімдер - белгілі бір енгізу/шығару операциялары үшін пайдаланылады және үлкен сандар көбейту және бөлу командалары қолданылады.

5.3 Сегмент тізілімдер

Негізгі сегмент тізілімдер саны - төрт. Барлық тізілімдер сегменттеріне сайкес сегменттерді анықтау үшін пайдаланылады. Олардың болуы Intel микропроцессорларының нақты ұйымдастыру және пайдалану, жадына байланысты және микропроцессорлық аппараттық сегменттері құрылымы үш бөлімнен тұратын бағдарламаны қамтиды. Тиісінше, осындай жады ұйым сегменті деп аталады. Микропроцессорлық бағдарламалық қамтамасыз ету моделі мынадай сегмент тізілімдерінен тұрады : CS, DS, SS, ES.

- CS (Code Сегмент) тізілім - тізілім коды сегментінің бастапқы мекенжайын қамтиды. Бұл команданың (тізілім IP) мекенжайын анықтайды және соны орындау үшін таңдалыну керек. Қалыпты бағдарламалар үшін CS тізілімдер сілтеме жасаудың қажеті жоқ.
- DS (Data Segment) тізілім –мәліметтер сегменті. Бастапқы мекенжайды қамтиды.

Командада анықталған, деректер сегментінде белгілі бір ұяшықты көрсетеді.

- CC (Stack Сегмент) тізілім - стек сегментінде бастап мекенжайын қамтиды
- ES (Extra Segment) тізілім - тіркеу тізімнің кеңейту сегменті.

Кейбір жолдарға арналған операцияларды шешуге, тізілімдегі ES қосымша сегменті пайдаланылады және мекенжайлық мәліметтерді, жады қадағалайды.

Бұл жағдайда ES тіркеуі DS тікеуімен байланысты. Әдетте бұл байланысты <ES:DI> мағынасын білдіреді.

Егер ES тіркеу тізімін қолдану қажет болса, ассемблер бағдарламасы оны баптандыру керек.

5.4 Тізілім көрсеткіштері мен индекстері

eip/ip тізілім - бағдар беруші командасы. EIP / IP тізілім - 32/16 бит ені бар және cs ағымдағы сегмент командалар сараланымдық тізіліміне мазмұны келесі орындалуы тиіс ауысым команданы құрайды. Бұл тізілім бағдарламашы үшін тікелей қол жетімді емес, бірақ жүктеу және пәрмендерін шартты және шартсыз секіру қамтиды, түрлі басқару командалары, өндірілген мәндерін, қоңырау рәсімдері мен рәсімдерін қайта өзгертеді.

Үзілістің пайда болуы EIP / IP тізілім модификациясына әкеліп соғады.

- ESI / Si тізілімнің бастапқы коды. Бұл тізілім операциялардың жолдық ағымдағы мекенжайын қамтиды;

- edi/di тізілім - қабылдағыш индексі (алушы). Бұл тізілім операциялар тізбегі ағымындағы мекенжайын қамтитын пайда көзі.

esp/sp тізілім - стек меңзер тізілімі. Ол ағымдағы стек сегментінде стек сілтегіштін қамтиды.

- ebp/bp – базаға бағыттаушы тізілім. Стектер үшін әзірленген кездейсоқ деректер ұйымы.

5.5 Тізілім жағдайы мен басқарылуы.

Микропроцессорларға бірнеше тізілім қосылған: қазіргі уақытта конвейер жүктелген процессор және бағдарламаның екі мәртебесі туралы ақпаратты қамтамасыз етеді.

Бұл тізілімге мыналар кіреді:

-тулар тізілім eflags/flags;

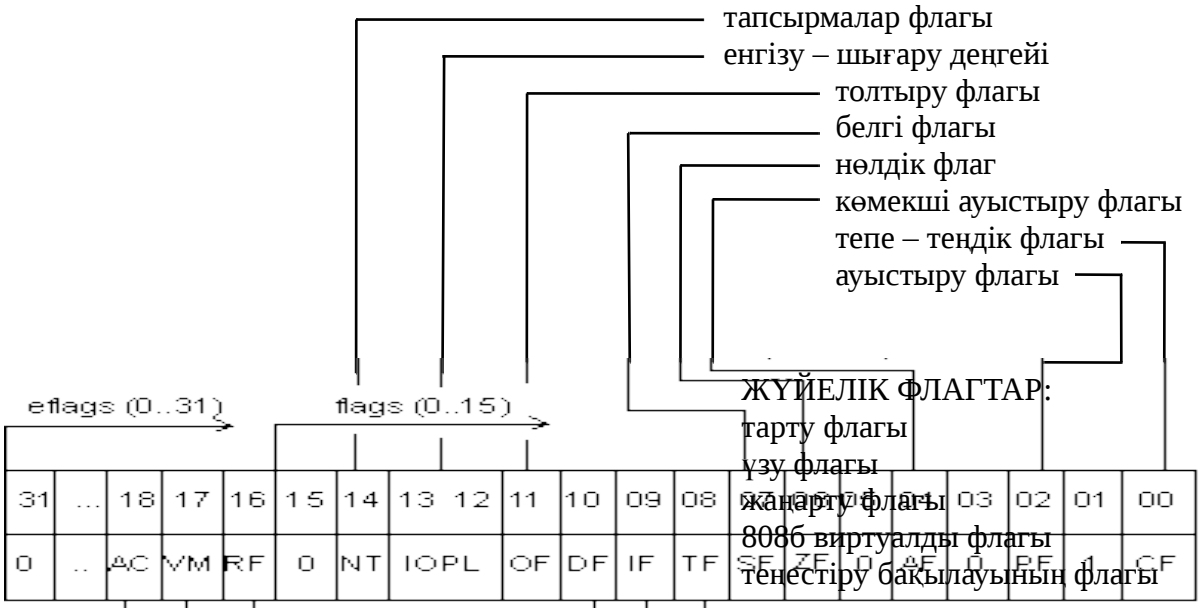
- тізілім көрсетулер командасы *eip/ip.*

Бұл тізілімді пайдалану арқылы командалар нәтижелері және микропроцессор мемлекет әсері туралы хабар сақтауға болады.

Бұл тізілімдердің құрамын және берілуін толығырақ қарастырайық:

eflags/flags – белгілер тізілімі. Разрядтылығы eflags/flags – 32/16 бит. Берілген тізілімнің әрбір битінің анықталған функционалдық тағайындамалары бар, оларды белгілер деп атайды. Бұл тізілімнің кіші бөлігі i8680 үшін арналған flags тізілімнің жұмысына ұқсас болып келеді. Eflags тізілімінің құрылымы 5-суретте көрсетілген:

Флаг және жай – күйі



Қолдану ерекшеліктеріне байланысты eflags/flags тізілімдер белгілерін үш топқа бөлуге болады:

- 8 белгілер күйі. Бұл белгілер машина командасының орындалуына байланысты өзгеріп отырады. Белгілер күйінің тізілімі eflags негізінен арифметикалық және логикалық операциялар орындалған кездегі нәтиже ерекшеліктерін белгілейді. Бұл есептеу процесінің күйіне анализ жасауға және оған шартты өту командалары немесе бағыныңқы программаларды шақыру арқылы өзгеріс енгізуге мүмкіндік береді;

- 1 басқару белгісі df деп белгіленеді. Ол M eflags тізілім 10-шы битінде болады да, негізінен тізбектеген командалармен қолданылады. df белгісінің мәні бұл операциялардағы өңделетін элементтердің бағытын анықтайды: жолдың басынан соңына қарай (df =0) немесе керісінше, жолдың соңынан басына қарай (df=1).

Df белгісімен жұмыс істеу үшін арнайы командалар бар: cld (df белгісін нөлдеу) және std (df белгісін тағайындау). Бұл командаларды қолдану df белгісін алгоритмімен сәйкестендіруге және кішірейтіп отыруға мүмкіндік береді;

- 5 жүйелік белгілер, енгізу/шығаруды, үзулерді маскалауды, аудармалауды, 8086 виртуалды режимімен тапсырмалар арасындағы ауысуларды басқарады. Қолданбалы программаларға бұл белгілерді мофикациялауға болмайды, өйткені ол көп жағдайда программа жұмысын үзуге әкеліп соғады.

Күйлер белгісі:

1) CF- тізілім белгісі (Carry Flag). Егер арифметикалық операция нәтижесінде үлкен биттен тасымал болған болса, 1-ге орналастырады. Үлкен болып операдтың өлшеміне байланысты 7-,15- немесе 31-ші биттер алынады. Егер тасымал болмаса, онда CF=0.

2) PF – жұптық белгі (Parity Flag). Егер нәтиженің кіші бірлік байттарының саны жұп болса, 1-ге орналастырылады, әйтпесе – 0.

3) AF – қосымша тасымал белгісі (Auxiliary carry Flag). Егер алдыңғы орындалған операция нәтиже кезінде 3 және 4 разрядтар арасында тасымал болған кезде, 1-ге орналастырылады, әйтпесе – 0.

4) ZF – нөлдік белгі (Zero Flag). Егер нөлдік нәтиже алынған болса, 1-ге орналастырылады, әйтпесе – 0.

5) SF – таңба белгісі (Sign Flag). Бұл арқашанда нәтиженің үлкен битіне тең (7-,15- және 31- биттері 8 үшін, сәйкесінше 16- немесе 31-разрядтық операдтар алынады).

6) OF – аса толу белгісі (Overflow Flag). Егер таңбалы сандармен арифметикалық операцияларды орындау кезінде нәтиже белгіленген жағдайдан асып кетсе, 1-ге орналастырылады.

7) IOPL – енгізу-шығару үстемділік деңгейі (Input/Output Privilege Level). Микропроцессордың қорғалған режимде енгізу-шығару командаларындағы есептің ерекшеліктеріне байланысты жұмысына қатынауды бақылау үшін қолданылады.

8) NT – енгізілген есеп белгісі (Nested Task).

Жүйелік белгілер:

1) TF – жүйелік үзу белгісі(Trace Flag). Микропроцессор жұмысын қадамдап орындау үшін арналған. Егер TF=1 болса – микропроцессор әрбір машина командасын орындап болғаннан соң 1-ге тең үзулерді генерациялайды. Программаларды аудармалау кезінде де, аудармалағыштармен қолданылады. Ал егер TF=0 – қалыпты жұмыс.

2) IF – үзу белгісі (Interrupt eneble Flag). Ақпараттық үзулердің орындалуына рұқсат немесе рұқсат еместігі (маскировка) орнату белгісі (INTR кірісі бойынша үзу). 1- ақпараттық үзулерге рұқсат етілген; 0- ақпараттық үзулерге рұқсат жоқ.

3) RF – қалыпқа келтіру белгісі (Resume Flag). Аудармалау тізілімі келіп түсетін түзулерді өңдеу үшін қолданылады.

4) VM – виртуальді режим белгісі (Virtual 8086 Mode). 8086 виртуальді режимде микропроцессордың жұмыс істеу белгісі. 1- процессор 8086 виртуальді режимде жұмыс істеп жатыр; 0- процессор нақты немесе қорғалған режимде жұмыс істеп жатыр.

5) AC – теңестіруін бақылау белгісі (Alignment Check). Жадыға қатынаған кездегі теңестіруді бақылау үшін арналған. CR0 жүйелік тізілімі AM белгісімен бірге қолданылады. Мысалы, Pentium кезкелген мекенжайдан командаларды және деректерді орналастыруға рұқсат етеді. Егер командалар мен деректердің мекенжайларының 2 немесе 4 бөлінетіндерінің теңестіруін бақылау керек болса, онда берілген бөлінбейтін адрестерді шақырған кезде олар қолайсыз жағжай туғызады.

Eip/ip (Instraction Pointer register) – команда тізілім-көрсеткіші. Eip/ip

Тізілімінің 32/16 биттік разрядтылығы болады және ағымдағы команда сегментіндегі cs сегменттік тізілімнің құрамына байланысты келесі

орындалатын командaның ығысуынан тұрады. Бұл тізілім тікелей программалаушымен қолданылмайды, бірақ оның мәнін жүктеу және өзгерту әртүрлі басқару командалары арқылы орындалады, оларға шатты және шартсыз өту командалары, процедураларды шақыру және қайта қайту командалары жатады. Үзудің пайда болуы да осы Еір/ір інің модификациясына әкеп соғады.

6 Дәріс. №6 Микропроцессордың жүйелік тізілімдері

Бұл тізілім атының өзі айтып тұрғандай, олар жүйедегі функцияларды орындайды. Жүйелік тізілімдерді қолдану қатты қадағаланады. Осы тізілім қорғалған режимде жұмыс жасауды қамтамасыздандарады және де бұларды микропроцессор архитектурасы ретінде қарауға болады, өйткені билікті жүйелік программалаушы төменгі деңгейдегі операцияларды орындай алу үшін әдейі қалдырылған.

Жүйелік Тізілімді үш топқа бөлуге болады:

1) Төрт басқару тізілімі.
2) Төрт жүйелік басқару тізілімі і.
3) Сегіз жөндеу тізілімі і.

Басқару тізілімі і.

Бұл топтағы басқару тізіліміне келесі 4 тізілімі жатады: cr0,cr1,cr2,cr3. Бұл тізілімідер жүйені жалпы басқару үшін арналған. Басқару тізілімнің үстем деңгейі 0-ге тең программаларда ғана қолданылады. Дегенмен микропроцессорда төрт басқару тізілімі і бар болғанымен, ол оның үшеуін ғана қолдана алады - cr1 қолданылмайды, оның функциялары анықталмаған (ол келешекте қолданылатын тізілімдерге жатады).

Cr0 тізілімі жүйелік белгілерден тұрады, микропроцессор жұмыс режимін басқарады және оның күйін нақты бір орындалатын есептен тәуелсіз ауқымды түрде белгілеп береді.

Жүйелік белгілердің атқаратын жұмысы:

- pe (Protect Enable), 0 бит – қорғалған режим жұмысына рұқсат. Бұл белгінің мәні сол уакытта микропроцессор екі режимнің – нақты pe=0 немесе қорғалған pe=1 қайсысына жұмыс істеп жатқандығын көрсетеді;

- Mp (Math Present), 1 бит – қосымша процессордың бар болуы, ол әрқашанда 1 болады;

- Ts (Task Switched), 3 бит, - есептен есепке ауысу. Процессор бір есептен екінші есепке ауысқан кезде бұл битті автоматты түрде орнатады;

- am (Aligment Mask), 18 бит – қалыптастыру мақсаты. Am=1 болса, бұл битке рұқсат немесе am=0 болса, бұл битке рұқсат жоқ яғни қалыптастыруға болмайды;

- cd (Cache Disable), 30 бит – кэш-жадыға рұқсат жоқ. Бұл биттің көмегімен қолданылатын ішкі кэш-жадыны қолданылуына cd=1 болса, рұқсат жоқ немесе cd=0 болса рұқсат бар (бірінші деңгейдегі кэш жады);

-pg (PaGing), 31 бит – беттік өзгеруге рұқсат бар, егер pg=1 болса немесе рұқсат жоқ, егер pg=0. Бұл белгі жадыны ұйымдастырудың беттік моделын қолданған кезде қолданылады;

- cr2 тізілімі – ағымдағы қоманда сол кезекте жадыда болмаған, беттік жадыда орналасқан, адреске сұраныс жасаған кезде жағдайлардағы тізілім үшін жедел жадының беттік ұйымдастырылуы кезінде қолданылады. Бұл жағдайда микропроцессорда осы шектеуді шақырған, 14 нөмірмен және 32 биттік тізбекті адреспен ерекше шектеу cr2 тізілімдеде жазылады. Бұл ақпаратты 14 шектеу өңдегеуіші қолдана отырып, бетті анықтайды және оның жадыға жүктелуін қамтамасыздандырады да, әрі қарай программаның дұрыс жұмыс істеуін қалыптастырады;

- Cr3 тізілімі – бұл да жадыны беттік ұйымдастыру кезінде қолданылады. Бұл тізілімі де тағы да бірнеше деңгейлі беттік каталог тізілімі деп те атайды. Ол ағымдағы есептің 20 биттік каталогтың физикалық базалық адресінен тұрады. Бұл каталог 1024 32-биттік дескриптор болады, олардың арқайсысы екінші деңгейлі беттік кесте адресінен тұрады. Ал, ол өз кезегінде, беттік кадрларды жадыға адрестейтін, әрбір кесте екінші деңгейлі 1024 32-биттік дескрипторы бар беттер кестесінен тұрады. Беттік кадр өлшемі – 4 Кбайт.

Жүйелік адрес тізілімі.

Бұл тізілімдерді тағы да жадыны басқару тізілімдері деп те атайды. Олар микропроцессор жұмысының көпесептік режимі кезінде программалар мен деректерді қорғау үшін қолданылады.

Микропроцессордың жұмысының қорғалған режимінде адрестік кеңістік келесі топтарға бөлінеді:

1) Ауқымды-барлық есептер үшін жалпы.
2) Жергілікті-әрбір есеп үшін дара.

Микропроцессордың архитектурасында келесі бөліктерге бөлінген жүйелік тізілімдер бар:

- GDTR ауқымды дескрипторлар кестесінің тізілімі і (Global Descriptor Table Register), өлшемі 48 бит және GDT ауқымды дескрипторлар кестесінің 32 биттік (16-47 биттер) ауқымды дискрипторлар кестесінің базалық адресінен және 16 биттік (0-15 биттер) GDT кестесінің байтпен берілген шектеу мәнінен тұрады;

- *тізілім кестесінің локальды дескрипторлары* LDTR (Lokal Descriptor Table Register), өлшемі 16 бит және локальдық дескрипторлық кестелерінің LDT селектор дескрипторы. Бұл селектор кестеде белгі болып саналады GDT, ол сегментті сипаттайды, оның ішінде локальдық дескрипторлық кесте LDT бар;

- *тізілімдер кестесінің үзіліс дескрипторлары* IDTR (Interrupt Descriptor Table Register), өлшемі 48 бит және құрамында 32 биттік (16 -47 биттер) дескрипторлық үзілістің базалық адресі IDT және 16 биттік (0-15 биттер) шектік мәні, IDT кестесінде өлшемдері байтпен беріледі.

- 16 – биттік есеп тізілімі i TR (Task Register), ол Idtr тізіліміне ұқсас; құрылымында селектор бар, яғни GDT кестесінде дескрипторға нұсқауышы бар. Ол дескриптор ағымдағы сегмент есеп күйін (TSS- Task Segment Status). Бұл сегмент жүйедегі әрбір есеп үшін жасалынады, құрылымы қатаң реттелген және есептің мәліметтерінен құралады (ағымдағы жағдайы). TSS сегментінің негізі – ауысу басқа есепке көшу барысында есептің ағымдағы күйін сақтау.

6.2 Жөндеу тізілімі

Бұл аппараттық қателерге арналған тізілімдердің тобы. Бірінші аппараттық қателердің ортасы i486 микропроцессорларында пайда болды. Аппараттық микропроцессорде сегіз жөндеу тізілімі бар, бірақ көбінесе олардың 6-уы ғана қолданылады.

DR0, DR1, DR2, DR3 тізілімдердің разрядтары 32 бит және төрт үзіліс нүктелерінің сызықтық адресін беруге арналған. Сол кездегі қолданатын механизм келесі: ағымдағы бағдарламамен құралатын кез келген мекенжай DR0…DR3 тізілімідегі мекенжаймен салыстырылады, егер сәйкес мекенжайлары табылса нөмірі 1-ші жөндеу тізілімі құрылады. DR6 тізілімі жөндеу тізілімінің күй тізілімі деп аталады. Бұл тізілімдердің биттері соңғы 1 жөндеу тізілімнің себептерімен сәйкес орнатылады.

Осы биттерді және олардың атап өтейік:

1) b0- бұл бит 1-ге тең болса, онда соңғы шектеу (үзу) dr0 тізілімінде анықталғандай бақылау нүктесіне жеткендігін, болғандығын білдіреді.
2) b1- b0 ұқсас, бірақ dr1 тізілімдегі бақылау нүктесі.
3) b2 - b0 ұқсас, бірақ dr2 тізілімдегі бақылау нүктесі.
4) b3 - b0 ұқсас, бірақ dr3 тізілімідегі бақылау нүктесі.
5) bd (13 бит) – жөндеу тізілімін қорғау үшін қолданылады.
6) bs (14 бит) – егер 1 шектеу eflags тізіліміндегі tf=1 белгі күйімен шақырылған болса, 1-ге орнатылады.
7) bt (15бит) – егер TSS қақпанындағы t=1 битімен есептен есепке ауысқан кезде шақырылған болса, 1-ге орнатылады.

Қалған биттер нөлмен толтырылады. 1-ші шектеу өндегіштері dr6 құрамына байланысты, шектеу болған себепті анықтап, соған байланысты шараларды қолдану керек.

DR7 тізілімді жөндеуді басқару тізілімі деп аталады. Мұнда жөндеуді бақылау нүктесі бар әрбір төрт тізілім үшін орын қалдырған, олардың көмегімен үзуді генерациялауға және келесі болатын шарттарды анықтауға болады: бақылау нүктесінің тізілім орыны тек ағымдағы есепте немесе кезкелген есепте. Бұл биттер dr7 тізілімінің кіші сегіз битін алады (әрбір бақылау нүктесіне екі биттен (факт жүзінде үзу нүктесі), сәйкесінше dr0, dr1, dr2, dr3 тізілімімен беріледі.)

Әрбір жұптағы бірінші бит – жергілікті рұқсат деп аталады; бұл биттің орнатылуы, үзу нүктесінің орындалуы адрестік кеңістіктің, осы ағымдағы есепте болын талап етеді.

Ал әрбір жұптағы екінші бит- ауқымды рұқсат белгісі деп аталады; бұл биттің орнатылуы, үзу нүктесінің орындалуы адрестік кеңістік, осы ағымдағы есепте ғана емес жүйеде болуына да рұқсат етеді.Үзу инициализацияланатын қатынау типі деректерді жазу немесе жазу/оқу кезінде, командаларды таңдау кезінде қолданылады. Бұндай үзулердің пайда болатынын анықтайтын биттер берілген тізілімнің үлкен бөлігінде локализацияланады. Көптеген жүйелік тізілімді программада қолдануға болады.

7 Дәріс №7. Операнд командасын беру тәсілдері . Деректерді жіберу командасы

Дәрістің мақсаты: программа құру үшін адрестеудің режимдерін үйрену.

Дәрістің мазмұны: адрестеудің режимдері, деректерді жіберу командалары

7.1 Адрестеу режимдері

Операндтар тізілімінде, жадыда және командалардың өзінде болуы мүмкін. Операндты табу тәсілі адрестеу режимдерімен анықталады. Ассемблердің командаларында келесі: тізілімі, тікелей, жанама, төте, ығысумен база бойынша, масштабтаумен жанама, индекстеумен база бойынша, индекстелумен және маштабталумен база бойынша адрестеу режимдері қолданылады.

1) тізіліміді адрестеу режимі.

Операнд тізілімде орналасады:

MOV AX, BX

2) Төте адрестеу режимі.

Операнд командада орналасады:

MOV AX,10 ;10-тікелей операнд;

MOV BX, OFFSET A ; OFFSET A- тікелей операнд ;

MOV AX, K ; K-тікелей операнд, егер K EQU немесе = дерективалары арқылы анықталған болса, онда K EQU 10 деп беріледі.

3) Жанама адрестеу режимі.

Операнд жадыда орналасады, мысалы: MOV AX,[BX]; BX –те операнд адресі орналасады. Тік жақшалар тізілімінде мекенжай орналасқанын көрсетеді. Операндтың адресін көрсету үшін BX, BP, SI, DI, SP тізілімі қолдануы мүмкін.

4) Тікелей адрестеу режимі.

Операнд жадыда орналасады. Командада операнд адресі көрсетіледі: MOV AX,A; A- операнд адресі, ол деректер сегментінде келесі (жол) оператор түрінде берілген:

A DW 10,20,30

MOV AX,A+2; AX=20

5) База бойынша адрестеу режимі.

Операнд жадыда орналасады, операндтың адресі BP немесе BX базалық тізілімдердің бірімен беріледі:

MOV AL,[BX]

MOV DX, [BP]

MOV DX, [BX+2]

6) Тікелей индекстелумен адрестеу режимі:

Операнд жадыда орналасады, операндтың адресі SI немесе DI базалық тізілімінің қосындысының жылжытуымен беріледі:

MOV AL,[SI]

MOV B [DI], BX.

7) Индекстелумен база бойынша адрестеу режимі:

Операнд жадыда орналасады, операндтың адресі құрамында бір индекстік және бір базалық тізілімі бар, жылжытудың қосындысымен беріледі:

MOV AX, TAB [BX][SI]

MOV A [BP][DI],AX.

Аталып өткен адрестеу режимдері үшін операндтар ақпараттар сегменттің ішінде орналасады, тек BP тізілімін адрестеу кезінде операнд стек сегментінде орналасады.

Операндтардың адрестері тізілімге командалардың көмегімен жүктеледі:

MOV BX, OFFSET TAB немесе LEA BX, TAB.

7.2 Деректерді жіберу командалары

MOV AX, BX

2) Төте адрестеу режимі.

Операнд командада орналасады:

MOV AX,10 ;10-тікелей операнд;

MOV BX, OFFSET A ; OFFSET A- тікелей операнд ;

MOV AX, K ; K-тікелей операнд, егер K EQU немесе = дерективалары арқылы анықталған болса, онда K EQU 10 деп беріледі.

3) Жанама адрестеу режимі.

Операнд жадыда орналасады,мысалы: MOV AX,[BX]; BX –те операнд адресі орналасады. Тік жақшалар тізілімі мекенжай орналасқанын көрсетеді. Операндтың адресін көрсету үшін BX, BP, SI, DI, SP тізілімдері қолдануы мүмкін.

4) Тікелей адрестеу режимі.

Операнд жадыда орналасады. Командада операнд мекенжай көрсетіледі: MOV AX,A; A- операнд мекенжай, ол деректер сегментінде келесі (жол) оператор түрінде берілген:

A DW 10,20,30

MOV AX,A+2; AX=20

5) База бойынша адрестеу режимі.

Операнд жадыда орналасады, операндтың адресі BP немесе BX базалық тізілім бірімен беріледі:

MOV AL,[BX]

MOV DX, [BP]

MOV DX, [BX+2]

6) Тікелей индекстелумен адрестеу режимі:

Операнд жадыда орналасады, операндтың адресі SI немесе DI базалық тізілімінің қосындысының жылжытуымен беріледі:

MOV AL,[SI]

MOV B [DI], BX.

7) Индекстелумен база бойынша адрестеу режимі:

Операнд жадыда орналасады, операндтың адресі құрамында бір идекстік және бір базалық тізілімі бар, жылжытудың қосындысымен беріледі:

MOV AX, TAB [BX][SI]

MOV A [BP][DI],AX.

Аталып өткен адрестеу режимдері үшін операндтар ақпараттар сегменттің ішінде орналасады, тек BP тізілімін адрестеу кезінде операнд стек сегментінде орналасады.

Операндтардың адрестері тізілімге командалардың көмегімен жүктеледі:

MOV BX, OFFSET TAB немесе LEA BX, TAB.

7.2 Деректерді жіберу командалары

Деректерді жіберу командалары белгілерді, адрестерді, деректерді жіберу үшін қолданылады.

Жалпы міндетті жіберу командалары.

MOV – жіберу командасы. Команданың форматы: MOV қабылдағыш, таратқыш.

MOV командасы таратқыш операндын (байт, сөз немесе екілік сөз) қабылдағыш операнд орнына жібереді. Қабылдағыш ретінде тізілім немесе жады ұяшығы қолдауы мүмкін, ал таратқыш ретінде тізілім, жады ұяшығы және тұрақты қолданылады.

MOV командасын келесі жағдайларда қолдануы мүмкін емес:

- бір жады ұяшығының мәнін екіншіге жіберу;
- бір жады ұяшығының мәнін сегменттік тізілімге немесе керісінше жіберу;
- бір жады ұяшығының мәнін екінші сегментті тізілімге жіберу;

Бұл көшіруді аралық тізілім арқылы орындауға болады. Аралық тізілім ретінде SP-дан басқа жалпы міндетті тізілім қолданылады.

MOV командасында рұқсат етілген жіберулер: тізілім- тізілім, тізілімі-жады, тұрақты- тізілім, тұрақты- жадыға, жады- тізілім. Мысалы:

mov AX, DX

mov AX, FLDA[SI]

mov FLDA, AX

mov AL,22h

mov FLDA[BP][SI],323h

PUSH- сөзді стекке жазу командасы. Команданың форматы: PUSH таратқыш.

PUSH командасы стекке сөзді немесе екілік сөзді жазады. Таратқыш ретінде тізілім немесе жады ұяшығы қолданылады. Мысалы:

push CX

push TABL

POP- сөзді стектен оқу командасы. Команданың форматы: POP қабылдағыш. Мысалы:

pop BX

pop TABL

POP командасы стектен сөзді немесе екілік сөзді оқиды. Қабылдағыш ретінде тізілім немесе жады ұяшығы қолданылады.

XCHG – сөздерді немсе байттарды ауыстыру командасы. Команданың форматы:

XCHG операнд_1, операнд_2

Команда байттардың, сөздердің немесе екілік сөздердің орынын ауыстырады. Операндр ретінде тізілімді және жады ұяшығын келесі түрде алуға болады: тізілім- тізілім, тізілім – жады. Мысалы:

xchg AX,DX

xchg BX,A[SI]

7.3 Қайта кодтау командалары

Команда XLAT – қайта кодтау командасы.

Командаың форматы: XLAT таратқыш_кесте.

Команда ds:x+(al) адресті есептейді, сосын al тізілімдеггі байтты адрес бойынша есептегіш байтқа ауыстырады.

XLAT командасын пайдаланғанда, қайта кодталатын элементтің нөмірі AL тізіліміне енгізіледі, ал таратқыш кестесінің адресі BX тізіліміне енгізіледі. XLAT командасы орындалғаннан кейін нәтиже AL тізіліміне енгізіледі. Мысалы:

; деректер сегментінде

ASCII DB ‘0123456789’

; кодтар сегментінде

Mov BX,offset ASCII

```
Mov AL,5
Xlat ; AL=35h
```

7.4 Енгізу шығару командалары

IN- порттан оерандты енгізу.

Команданың форматы: IN аккумулятор, порт

IN – командасы байтты, сөзді және екілік сөзді порттан микропроцессорға алып орналастырады.

OUT- порттан операндты шығару.

Команданың форматы: OUT порт, аккумулятор.

OUT– командасы байтты, сөзді және екілік сөзді микропроцессордан портқа алып орналастырады.

Аккумулятор есебінде AL тізілімі (байттарды қайта тасымалду үшін) және AX тізілімі (сөзді қайта тасымалду үшін) қолданылады. Порт операторы есебінде 0-ден 255-ке дейін порттар нөмірі және DX тізілімі қолданылады. Мысалы:

```
In AL,60h
Out 20h,AL
In AX,DX
Out DX,AX
```

7.5 Адресті қайта тасымалдау командалары

LEA- тиімді адресті жүктеу командасы.

Команданың форматы: LEA қабылдағыш, таратқыш.

Команда орындалған кезде қабылдағыш тізілімге 16 биттік немесе 32 биттік таратқыш операндынан түскен мән жүктеледі. Қабылдағыш операнды DW немесе DD дерективасымен анықталуы керек. Мысалы:

```
TAB DW 10h,20h,30h
Lea BX,TAB
```

Берілген команда ассемблердегі offset операторына ұқсас. Оның offset операторынан айырмашылығы, Lea командасы операндты индекстеуді орындайды, ол арқылы операндты адрестеуді ұйымдастыру жеңіл болады. Мысалы:

```
; bx тізілім массивінің бесінші элементін жүктеу керек болсын
.data
Mas db 10 dup (0)
.code …
Mov di,4
Lea bx,mas[di]
; немесе
Lea bx, mas[4]
; немесе
```

Lea bx,mas+4

LDS/LES/LFS/LGS/LSS командалары жадыдан көрсетілген сегменттік тізілімдерді ds/es/fs/gs/ss жүктейді.

Команданың форматы: LDS қабылдағыш, таратқыш. Команда көрсетілген тізілімге кішкене сөзді, ал DS сегменттік тізіліміне үлкен сөзді жүктейді.Мысалы:

main DD 10000100h

lds BX,main

LDS командасы MOV ауыстырады:

mov BX,offsey main

mov AX,seg main

mov DS, AX

LES, LEF, LGS,LSS командалары LDS командасына ұқсас, бірақ мекенжай сегменті сәйкесінше ES,FS,GS,SS тізіліміне жүктейді.

Белшілерді ауыстыру командалары:

а) LAHF- AH тізіліміне белгілерді жүктеу.

LAHF командасы белгілер тізілімінің кіші битін AH тізіліміне жүктейді;

б) SAFH – AH тізілімдегі белгілерді орнату.

SAFH командасы AH тізілімдегі 0, 2, 4, 6, 7 разрядтарды белгілер тізілімінің кіші байтына қайта жазады.

в) PUSHF- стекке белгілерді орналастыру.

PUSHF командасы белгілер тізілімдегі мәндерді стекте сақтайды.

POPF- стектен белгілерді шығару.

POPF командасы стектен сөзді оқиды да, оны белгілер тізіліміне орналастырады.

8 Дәріс №8 Басқару тізгінін ауыстыру командасы

Дәрістің мақсаты: сызықтық емес алгоритмдерді программалау үшін командаларды үйрену.

Дәрістің мазмұны: басқару тізгінін шартсыз, шартты, циклді түрде ауыстыру.

Бұл топтағы командалар программаның бір жерінен екінші жнріне көшуге мүмкіндік береді. Бұл командалар жұмыс принципіне байланысты үш токқа бөлінеді:

1) Басқару тізгінін шартсыз ауыстыру командалары:

- JMP белгі –шартсыз өту командасы;

- CALL процедураның аты және RET – процедураны шақыру және процедурадан қайту;

- INT үзудің нөмірі – программалық үзілістерді шақыру және программалық үзілістен қайту.

2) Басқаруды ауыстыру командалары

- jxx өту_белгісі, мұнда xx командамен сұрыпталатын, нақты шартты анықтайды;

- CMP *операнд 1, операнд 2* – салыстыру командасының нәтижесімен ауысу командасы;

- Белгілі бір белгінің күйіне байланысты өту командасы: JC,(JNC); JP, (JNP); JZ, (JNZ); JS, (JNS); JO, (JNO);

- тізілімнің құрамына байланысты өту CX: JCXZ< өту – белгісі >

3) Циклді басқару командалары.

- LOOP өту_белгісі – циклді қайталау;

- санауышы бар циклді ұйымдастыру командалары.

Ассемблер тілінде басқару тізгінін ауыстыру үшін белгілер қолданылады. Белгі- символдық есім, жадыда белгілі бір ұшықты білдіреді, басқару тізгінін ауыстыру кезінде операнд ретінде қолдануға арналған.

8.2 Басқару тізгінін шартсыз ауыстыру командалары

JMP белгі –шартсыз өту командасы.

Команданың форматы: jmp [модификатор] ауысу_адресі.

Ауысу_адресі ауысу көрсеткіші орналасатын жады аймағы немсе таңба түріндегі адресті көрсетеді jmp шартсыз ауысу машиналық командаларының бірнеше кодтары кездеседі. Ауысу_адресі ағымдағы сегмент кодының ішінде немсе басқа сегментте орналасуы мүмкін. Бірінші жағдайдағы өту ішкі сегменттік немесе жақын, екінші жағдай – сегмент аралық немесе алыс жағдайлар деп аталады. Ішкі сегменттік ауысу кезінде eip/ip регистрлерінің құрамы ғана ауысады. jmp командасымен қолданылатын үш ішкі сегменттік ауыстыруларды атап өтуге болады: тікелей қысқа; тікелей; жанама.

Модификатор келесі мәндерді қабылдай алады:

а) near ptr – ағымдағы сегмент кодының ішіндегі таңбаға тікелей ауыстыру. Тек eip/ip (use16 немесе use32) тізілімі ғана командадағы көрсетілген адрес (таңба) бойынша модификацияланады;

б) far ptr – басқа сегмент кодынан таңбаға тікелей ауысу. Ауысу адресі 16 биттік селектор және 16/32 биттік ығысудан құралған адрестен (таңбадан) немесе тәуелсіз ортадан тұрады, олар сәйкесінше cs және ip/ei тізіліміне жүктеледі;

в) word ptr – ағымдағы сегмент кодынан таңбаға жаңа ауысу. Тек eip/ip модификацияланады. Ығысу өлшемі 16 немесе 32 бит;

г) dword ptr басқа сегмент кодынан таңбаға жанама ауысу. cs және ip/eip тізілімі модификацияланады. Бұл адрестің бірінші сөзі/екілік сөзі ығысуды көрсетеді және ip/eip жүктейді; екінші/үшінші сөздер cs жүктейді. CALL – процедураны шақыру. Команданың форматы: CALL процедура_аты RET - проецедцрадан қайту.

Процедура командалар тізбегі, ол бір рет қана жазылады, қажетіне қарай программада кезкелген жерден бірнеше рет шақырылады.

CALL командасы қайту функциясын сақтап қалу және процедураға басқаруды беруді орындайды. Ол CS команда сегментінің адресі, процедура NEAR атрибутымен анықталған болса, қайту адресінің ығысуын стекке орналастырады және FAR атрибутымен анықталған болса, онда ығысу адресін стекке орналастырады. NEAR атрибуты процедуралар өзі орналасқан сегменттен ғана шақырылады, ал FAR атрибутындағы сегменттер басқа сегменттерден де шақырыла береді.

Қайту адресін сақтағаннан кейін CALL командасы таңбаның сегменттік адресінің ығысуын «процедура_аты» IP (EIP) команда көрсеткішіне жүктейді. Егер процедураның атрибуты FAR болса, онда CALL командасы таңбаның сегменттік адресінің ығысуын «процедура_аты» CS тізіліміне жүктейді.

RET командасы стектен қайту адресін шығарады. Егер процедураның атрибуты NEAR болса, онда RET командасы стектен бір сөзді (екілік сөзді) шығарады және оны IP (EIP) команда көрсеткішіне жүктейді. Егер процедураның атрибуты FAR болса, онда RET командасы стектен екі (үш) сөзді шығарады: бірінші IP (EIP) команда көрсеткішіне орналастыру үшін адрестің ығысуын, содан кейін CS регистріне жүктеу үшін сегмент адресін шығарады.

8.3 Басқару тізгінін шартқа байланысты ауыстыру командалары

Басқару тізгінін шартқа байланысты ауыстыру командаларының жалпы форматы келесі түрде болады: jx ауысу_таңбасы, мұндағы x- модификатор, ол бір немесе бірнеше әріптерден тұрады, ол келесі мәндерді қабылдауы мүмкін:

1) E(equal) – тең.
2) N (not) – немесе.
3) G (greater) – үлкен.
4) L (less) – кіші.
5) A (above) – жоғары.
6) B (below) – төмен.

E, N модификаторлары кез келген типтегі операндтар G және L – таңбасыз сандар үшін; A мен B – таңбалы сандар үшін қолданылады.

Шартқа байланысты ауыстыру командаларын программаның орындалуы кезінде пайда болатын әртүрлі шарттарды тексеру үшін қолданған ыңғайлы. Көп командалар орындалу нәтижелерін flags(eflags) тізілімінде қалыптастырылады. Төменде 8.1 кестеде шартқа байланысты ауысу командалары және олар орындалған кездегі белгілердің өзгеруі, сондай-ақ оларға сәйкес логикалық ауысулар көрсетеді.

8.1 кесте - Шартты өту командалары

Команда	Тексерілетін белгілер күйі	Ауысу шарты
JA	CF=0 және ZF=0	Егер жоғары болса

JAE	CF=0	Егер жоғары болса немесе тең болса
JB	CF=1	Егер төмен болса
JBE	CF=1 немесе ZF=1	Егер төмен болса немесе тең болса
JC	CF=1	Егер тасымал болса
JE	ZF=1	Егер тең болса
JZ	ZF=1	Егер 0 болса
JG	ZF=0 және SF=OF	Егер үлкен болса
JGE	SF=OF	Егер үлкен немесе тең болса
JL	SF<>OF	Егер кіші болса
JLE	ZF=1 немесе SF<>OF	Егер кіші немесе тең болса
JNA	CF=1 және ZF=1	Егер жоғары болмаса
JNAE	CF=1	Егер жоғары немесе тең болса
JNB	CF=0	Егер төмен болмаса
JNBE	CF=1 және ZF=0	Егер жоғары немесе тең болса
JNC	CF=0	Егер тасымал болмаса
JNE	ZF=0	Егер тең болмаса
JNG	ZF=1 немесе SF<>OF	Егер үлкен болмаса
JNGE	SF<>OF	Егер үлкен емес немесе тең болса
JNL	SF=OF	Егер кіші болмаса
JNLE	ZF=0 және SF=OF	Егер кіші емес немесе тең болса
JNO	OF=0	Егер асатолу болмаса
JNP	PF=0	Егер нәтиже битерінің қосындысы тақ болса (тақтық паритет)
JNS	SF=0	Егер таңба оң болса (таңбалық (үлкен) бит нәтижесі 0 тең)
JNZ	ZF=0	Егер 0 болмаса
JO	OF=1	Егер асатолу болса
JP	PF=1	Егер нәтиже биттерінің қосындысы жұп болса (жұптық паритет)
JPE	PF=1	JP сияқты, яғни тақтық паритет
JPO	PF=0	JNP сияқты, яғни тақтық паритет
JS	SF=1	Егер таңба теріс болса (таңбалық (үлкен) бит нәтижесі 1 тең)
JCXZ	Әсер етпейді	Егер регистр CX=0 болса
JECXZ	Әсер етпейді	Егер регистр ECX=0 болса

«Үлкен» және «кіші» логикалық шарттары таңбалы бүтін сан мәндері салыстыру үшін, ал «жоғары» және «төмен» - логикалық шарттары таңбасыз бүтін сан мәндерін салыстыру үшін қолданылады. Бірінші кездегі i8086 микропроцессорларындағы шартқа байланысты ауысу командалары қысқа ауысуларды ғана жасай алатын, ал қазіргі i386 микропроцессорларына қойылған шарт жоқ, олар ағымдағы сегменттің кезкелген жерінен ауысу жасай

береді. Ал сегмент аралық ауысулар жасау үшін шартты ауысу командалары мен jmp шартсыз ауысу командаларын қосып қолданған дұрыс.

Басқаруды шартқа байланысты ауыстыру командаларына кез келген белгінің күйін өзгерте алатын командалар кедергі келтіре алады, олар әдетте СМР салыстыру командасымен бірге қолданылады.

8.4 Циклді басқару командалары

Циклді басқару командалары цикл ұйымдастыру кезінде шартты ауысу үшін қолданылады. Әрбір команда СХ регистріндегі санауыш мәнін 1-ге кемітіп отырады, ал одан кейін оның жаңа мәніне байланысты оны тоқтату керек пен немесе әрі қарай жалғастыру жайындағы шешім қабылдайды.

LOOP- циклді санауыш біткенге дейін қайталайды. Команданың форматы: LOOP таңба

Бұл команда СХ регистр санауыштың мәнін 1-ге азайтады және СХ регистрінің мәні 0-ге тең болғанға дейін басқаруы таңбаға береді. LOOP командасы циклды орындауын СХ регистрі 0-ге тең болған кезде ғана аяқтайды. Бірақ көптеген қосымшаларда белгілі бір шартты орындаса болғаны СХ регистрінің мәнін 0-ге тең болғанша күтпей-ақ аяқтау керек болатын жағдайлар кездеседі. Мұндай циклды аяқтау командаларына LOOPE (егер тең болса, циклды қайтала) және LOOPNE (егер тең болмаса, циклды қайтала).

LOOPE (LOOPZ) – циклді (нөлге) тең болғанша қайталайды. Команда СХ регистрінің мәнін 1-ге азайтады, одан кейін СХ регистрі 0-ге және ZF 1-ге тең болған кезде ауысу жасайды. Сонымен цикл өз жұмысын СХ регистрі 0-ге және ZF регистрі 0-ге тең болған кезде немесе олардың екеуі де 0-ге тең болған кезде аяқтайды. Әдетте LOOPE командасы нөлдік нәтиже іздеу операцияларында қолданылады.

LOOPNE (LOOPZ) – циклды (нөлге) тең болмағанша қайталайды. Команда СХ регистрінің мәнін 1-ге азайтады, одан кейін СХ регистрі 0-ге тең емес және ZF 0-ге тең болған кезде ауысу жасайды. Сонымен цикл өз жұмысын СХ регистрі 0-ге және ZF регистрі 1-ге тең болған кезде немесе олардың екеуі де орындалған кезде аяқтайды. Әдетте LOOPNE командасы бірінші нөлдік нәтиже іздеу операцияларында қолданылады.

8.5 Үзу командалары

Бұл процедураны шақыру операциясымен бірдей, үзу микропроцессорды қайта қайту үшін стекте ақпаратты сақтауға мәжбүрлейді, одан кейін үзуді өңдеу программасын орындайды.

Үзу барлық өз программасын өңдеу кезінде жанама өтуді үзу векторының 32-биттік адресін алу үшін пайдаланады. Стекте үзу адресін және белгілерді сақтайды. Үзулер жүйенің сытрқы құрылғысынан немесе арнайы программада қолданылатын үзулерден пайда болуы мүмкін. Үзу

командаларының үш түрі кездеседі- екі шақыру командасы және бір қайту командасы.

INT-үзу командасы. Команданың форматы: INT үзу_типі. Үзу типі бұл нөмір, жадыда орналасқан 256 әртүрлі векторларды өңдейді.

INT командасын орындаған кезде микропроцессор келесі әрекеттерді орындайды:

а) белгілер тізілімінің мәнін стекке орналастырады;

б) OF трассировка белгісін және IF үзуді өшіру/қосу белгісін нөлдейді;

в) CS тізілімінің мәнін стекке орналастырады;

г) үзу векторының адресін, үзу типін 4 көбейту арқылы анықтайды;

д) үзу векторының екінші сөзін CS регистріне жүктейді;

е) IP команда көрсеткіші тізілімінің мәнін стекке орналастырады;

ж) IP команда көрсеткіші тізіліміне үзу векторының бірінші сөзін жүктейді.

INT командасы орындалғаннан кейін IP бергісінің және CS тізілімінің мәндері стекте болады. TF жүйелік үзу белгісі және IF үзуі 0-ге тең болады. CS және IP регистрлер тобы үзуді өңдеу программасының бастапқы адресін көрсетеді. Одан кейін микропроцессор ол программаны орындауды жүргізелді.

INTO- аса толу болған кездегі үзу командасы. Ол шартты үзу командасы. Бұл команда үзуді тек OF асатолу белгісі 1-ге тең болған кезде орындайды.

IRET – үзуден кейін қайту командасы. Бұл команданың да жұмыс принципі RET процедурадан қайту командасына ұқсас. Сондықтан ол микропроцессормен үзуді өңдеу программасы орындалып болғаннан кейін қолданылады. IRET командасы стектен 16 биттік үш мәнді шығарады және оларды IP команда команда көрсеткіші тізіліміне, CS тізіліміне және белгілер тізіліміне орналастырады.

8.6 Микропроцессорды басқару командалары

Бұл командалар программадан микропроцессор жұмысын басқаруды орындайды. Олар үш топқа бөлінеді.

Белгілерді басқару командалары.

Микропроцессордың CF тасымал белгісін DF бағыт белгісін және IF узу белгісін өзгеретін жеті командасы.

STC- тасымал белгісін орналастыру командасы яғни CF белгісі 1-ге тең болады.

CLC- тасымал белгісін нөлдеу командасы яғни CF белгісі 0-ге тең болады.

Бұлар керекті CF белгісінің күйін орнату үшін немесе осы тасымал белгісінің күйін орнату үшін немесе тасымал белгісін пайдаланатын RCL және RCR командаларын қолданғанда және циклдық жылжыту командаларын қолданғанда орындалады.

CMC- тасымал белгісін терістеу командасы, яғни CF белгісі 0-ге тең болса, 1-ге ге немесе керісінше ауысады.

STD- Бағыт белгісін орнату командасы яғни DF белгісі 1-ге тең болады.

CLD- Бағыт белгісін нөлдеу командасы яғни DF белгісі 0-ге тең болады.

CLI- үзу белгісін нөлдеу командасы. IF үзу белгісін 0-ге теңестіреді, жүйенің сыртқы құрылғысынан түсетін үзулерді қалқалап, микропроцессор ол узулерге көңіл бөлмеу үшін қолданылады. Бірақ қалқаланбаған үзулер өңделе береді. Бұл үзулер жады ұяшығындағы қателерді беретін хабарламадан тұрады.

STI- үзу белгісін орнату командасы. IF үзу белгісін 1-ге теңестіреді, жүйенің сыртқы құрылғысынан түсетін үзулерді микропроцессор өңдейді.

Сыртқы синхронизация командасы

Бұл командалар негізінен микропроцессор жұмысының сыртқы жағдайларымен синхронизациясы үшін қолданлады.

XLAT-тоқтату командасы. Микропроцессорды тоқтату күйіне орнатады. WAIT- Күту командасы. Микропроцессорды бос жүру күйіне орнатады.

ESC-«қашу» командасы. Микропроцессорды онда орналасқан операнд мәнін алып, оны деректер машинасына беруді орындайды, ол арқылы ол басқа микропроцессорлаға өз командалар ағымынан командаларды қолдануға мүмкіндік береді. Команданың форматы: ESC сыртқы_код, таратқыш

Сыртқы_код бұл 6 биттік тәуелсіз операнд, таратқыш- тізілім немесе айнымалы.

Жалқы жүре командасы.

NOP- операция жоқ. Бұл команда белгіде де, тізілімде де, жады ұяшығына да ешқандай әсер етпейді, тек IP команда көрсеткішінің мәнін өзгертеді.

NOP командасы тізбектелген командаларды тестілеуде қолданған дұрыс. Оны тестіленетін программада соңғы етіп қойып жүйелік үзу жасауға болады.

9 Дәріс №9. Арифметикалық команда

Дәрістің мақсаты: арифметикалық операцияларды орындайтын командаларды оқып үйрену.

Дәрістің мазмұны: арифметикалық командалар, қалпына келтіру командалары.

9.1 Қосу командасы

Қосу командасы ADD.

ADD *операнд_1, операнд_2*

операнд_1= операнд_1+ операнд_2

ADD AL, M, BYTE ; тізілім мазмұны + жад ұяшығы;

ADD A, BX ; (A) жад ұяшығының сөзі + BX тізілім мазмұны;

ADD BX, 10 ; BX+10 тізілімі мазмұны;

ADD A, 200 ; (A)+200 жад ұяшығының сөзі;

ADC *операнд_1, операнд_2* – CF тасымал белгісі бар қосу командасы: операнд_1= операнд_1+ операнд_2+мән CF.

Қалпына келтіру командаларының қосу нәтижелері.

AAA: Қосу үшін ASCII – форматын қосу.

AL тізілімінде екі ASCII – байттық сомманы реттейді. Егер оң жақ төрт байтты AL тізілімінің мәні 9-дан үлкен болса немесе белгі 1-ге орнатылса, онда AAA командасы өзіне AH тізілімін қосады және AF пен CF белгілерін орнатады. Команда әрқашан AL тізіліміндегі төрт сол жақ битті тазалайды.

Мысалы: ADD AL, BL

AAA

INC операнд – инкремент операциясы, операнд мәнін 1-ге өсіруді орындайтын команда;

DAA – қосу нәтижесін ондық түрде беру үшін қалпына келтіру.

Екі BCD ондық жинақталған элементтердің қосу нәтижесін AL тізілімінде қалпына келтіреді, яғни AL тізіліміндегі 2 дұрыс ондық сандардың қосу нәтижесін қайта құрады.Егер төрт оң байт 9-дан үлкен мәнге ие болса немесе белгі AF = 1, онда DAA командасы 6 AL тізіліміне қосады және AF белгісін орнатады.Егер AL > 9F немесе CF= 1, онда команда AL тізіліміне 60 H қосады.

ADD AL, DL

DAA

9.2 Азайту командасы

Азайту командасы SUB

SUB *операнд_1, операнд_2*

операнд_1= операнд_1- операнд_2

SBB *операнд_1, операнд_2* – несиені ескеретін азайту командасы

Мысалы:

SUB BX, DX ; CF = 1

SBB AX, CX ; AX BX –тегі нәтиже

Команда мына белгілерге әсер етеді

CF, PF, AF, ZF, SF, OF

Азайту нәтижесінің қалпына келтіру командалары.

AAS: Азайту үшін ASCII – форматын қалпына келтіру. Әртүрлі екі ASCII -байтты AL регистрінде қалпына келтіреді. Егер оң жақ төрт байтты AL тізілімінің мәні 9-дан үлкен болса немесе белгі 1 AH тізілімінен 1-ге орнатылса, онда AAS командасы AL тізілімінен 6- алынады және AH

тізілімінен 1.Команда әрқашан AL регистріндегі төрт сол жақ битті тазалайды.

DAS**:** Азайту үшін ондық түзету.

AL тізіліміндегі екі BCD (ондық жинақталған) азайтудың нәтижесін түзетеді. Төрт оң 9 бит немесе белгі AF=1-ден үлкен мәні бар болса, онда тізілім AL-ден DAS 60 H шегеріледі және CF белгісін.

Операнд AL тізілімінде орналасады:

1) SUB AL , BL
   AAS
2) SUB AL , DL
   DAS
3) SUB BX,DX
   SBB AX,CX

DEC операнд – декремент операциясы, операнд мәнін 1-ге кемітуді орындайды;

DEC ұйымда цикл санауыш азайту және индекстік тізілім (ішекті бойынша операциялар) азайту үшін пайдаланылуы мүмкін. NEG: теріс белгісін өзгерту,

Мысалы:

NEG
ADD AL , 100

9.3 Көбейту командасы

Көбейту командасы *MUL*.

MUL *көбейткіш_1*

Командада бір ғана операнд көрсетіледі - *көбейткіш*. Екінші *операнд* - *көбейткіш 2* көрсетілмей беріледі. Оның орны жасырын көбейткіш өлшеміне байланысты болады. Көбейткіш өлшемдерінің нұсқалары және екінші операндтың орналасуы мен нәтижелері 9.1 кестеде көрсетілген.

Белгісіз көбейгішті белгісіз көбейткішке көбейтеді. Сол жақ деректер биті бірыңғай бит болып қарастырылады. 8 - биттік көбейту көбейткіші AL тізілімінде орналасуы керек, ал көбейткіш тізілім немесе жадта болуы мүмкін, мысалы MUL CL.Көбейтінді AX тізілімінде орналасады.

16- биттік көбейту үшін көбейткіш де немесе жадта болуы тиіс,AX регистрінде орналасуы керек, мысалы, MUL BX

9.1 кесте

Көбейткіш_1	Көбейткіш_2	Нәтиже
Байт	al	16 бит ax :al –нәтиженің кіші бөлігі;ah-нәтиженің үлкен бөлігі.
Сөз	ax	32 бит dx:ax: ax – нәтиженің кіші бөлігі; dx – нәтиженің үлкен бөлігі.

Екілік сөз	eax	64 бит в паре edx:eax: eax- нәтиженің кіші бөлігі; edx- нәтиженің үлкен бөлігі.

IMUL *операнд_1*- таңбалы сандарды көбейту, таңбалы көбейткішке көбейтуді орындайды . Сол бірлік бит теріс сандар үшін минус белгісі ретінде қарастырылады.

Түзету командасының көбейту нәтижесі.

AAM: Көбейту үшін түзету ASCII - форматы.

Команда AAM екі жинақталмаған ондық сандарды қалпына келтіру үшін қолданылады, AL-ді 10-ға бөледі нәтижені AH қалдық регистріне жазады,ал бөліндіні AL-ге.

9.4 Бөлу командасы

Бөлу командасы DIV.

DIV бөлінгіш **–** таңбасыз сандарды бөлу командасы. IDIV бөлінгіш – таңбалы сандарды бөлу командасы.

Көбейту және бөлу командаларында дереккөз ретінде тікелей операндты қолдануға болмайды. Бөлінгіш жады ұяшығында немесе жалпы міндетті регистрдің бірінде орналасса, оның өлшемі 8,16 немесе 32 бит болады. Бөлгіштің орны жасырын және операнд өлшеміне байланысты болады (9.2 кесте).

9.2 кесте

Бөлінгіш	Бөлгіш	Бөлінді	Қалдық
Ах-гі 16 бит сөз	Байт – регистр немесе жады ұяшығы	Al регистріндегі байт	Ah регистріндегі байт
32 бит dx-үлкен бөлігі ах-кіші бөлігі	16 бит регистр немесе жады ұяшығы	Ах регистріндегі 16 бит сөз	Dx регистріндегі 16 бит сөз
64 бит edx- үлкен бөлігі eax- кіші бөлігі	Екілік сөз 32 бит регистр немесе жады ұяшығы	Eax регистріндегі 32 бит екілік сөз	Edx регистріндегі 32 бит екілік сөз

Қалпына келтіру командасының бөлу нәтижесі.

AAD – бөлу үшін ASCII - форматын қалпына келтіру.

Қалпына келтіру командасының бөлу нәтижесі.

AAD – бөлу үшін ASCII - форматын қалпына келтіру.

Алдын-ала AX регистріне бөлінгіштің екі жинақталмаған сандарын орындайтын команда. Бұл команда алынған екілік санды AL регистріне келесі екілік бөлу үшін орналастырады, одан кейін регистр мәнін көбейтеді, одан

кейін AH=0, яғни AH регистрін нөлдейді. AAD жинақталмаған бөлінді екілік санды ауыстырып, оны AL регистріне орналастырады.

Мысалы:

AAD

DIV BL

Белгі шығарылады SF, IF, PF.

10 Дәріс №10. Логикалық командалардың топтарын сипаттау

Дәрістің мақсаты: логикалық операцияларды орындайтын командаларды оқып үйрену.

Дәрістің мазмұны: команда битін өңдеу, жылжыту және циклдық жылжыту командалары.

10.1 Логикалық командалар

Логикалық командалар формальді логика тәртібі бойынша жұмыс істейді. Логикалық командалар операндтар биті айлалы болғандықтан, ол операндтар мәндер жазу кезінде әдетте он алтылық белгілерді пайдаланады. Логикалық командаларға AND (логикалық көбейту), OR (логикалық қосу), XOR (модуль екі бойынша қосу), NOT (логикалық терістеу), TEST (логикалық тексеру) (10.1 сурет)

AND, OR, XOR командаларының операндтары ретінде байт,сөз немесе екілік сөз болады. Бұл командаларда екі регистрді, регистр мен жады ұяшықтарын, тәуелсіз мән мен регистр құрамдарын қолдануға болады.

AND - логикалық «И». Команда форматы: AND қабылдағыш, таратқыш.

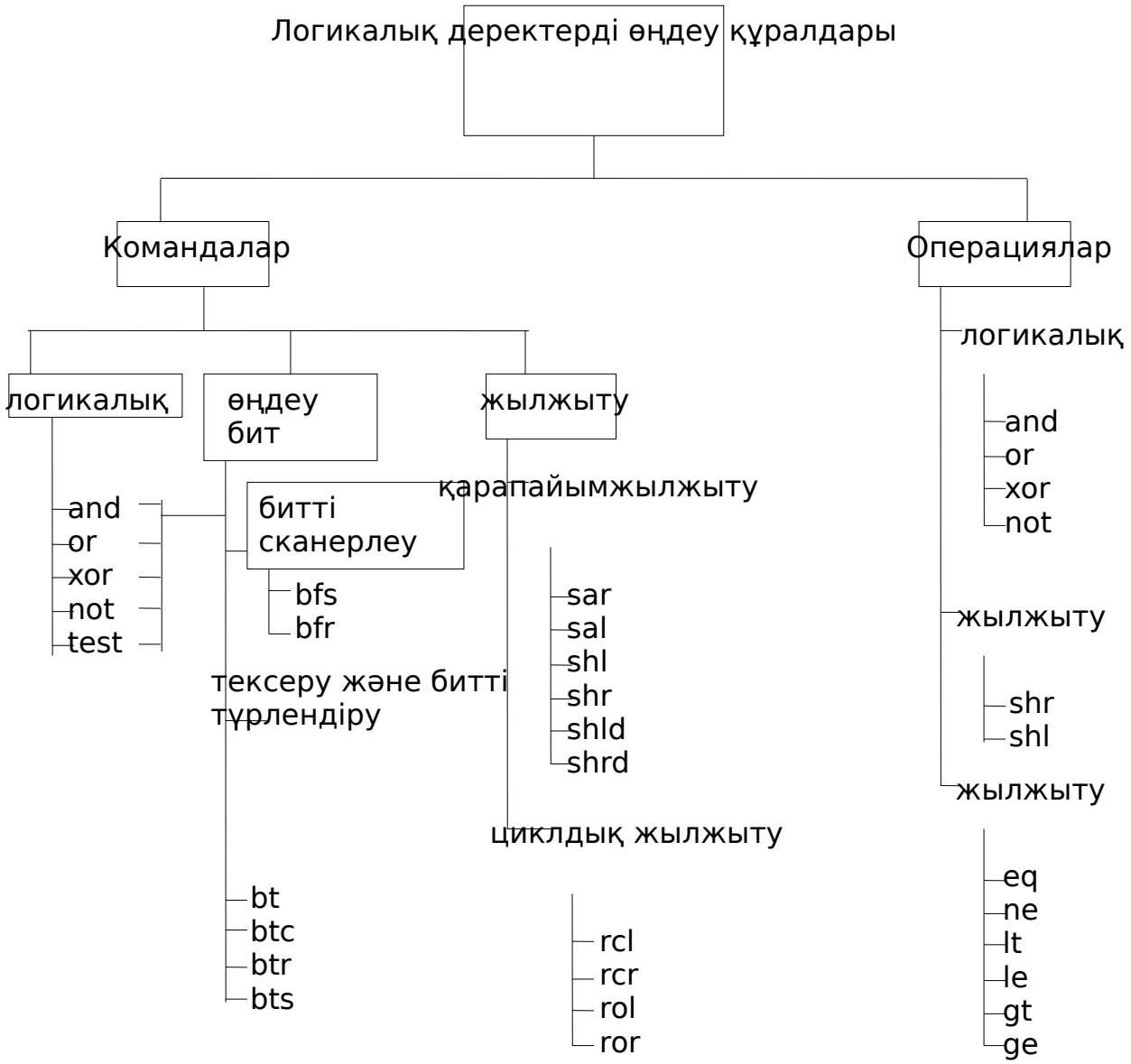
Нәтиженің әр разряды 1-ге тең болады, егер осыған сәйкес операндтар разряды 1-ге тең болса, қалған жағдайда нәтиже разряды нөлге тең, операция нәтижесі қабылдағышқа жазылады.

Команда орындалғаннан кейін: OF = CF = 0, SF, ZF және PF өз мәнін өзгертеді.

OR – логикалық «немесе». Команда форматы: OR қабылдағыш, таратқыш.

Командалық операндтар кем дегенде бір бірлігін қамтиды, оның мәндері, қабылдағыш операнд сол бит бірлігін белгілейді.

Логикалық деректерді өңдеу құралдары



10.1 сурет - Логикалық деректер үшін микропроцессордың жұмыс істеу құрылымы

Команда орындалғаннан кейін: OF = CF = 0, SF, ZF және PF өз мәнін өзгертеді.

XOR - логикалық «ТЕРІСКЕ ШЫҒАРУ».

Команда форматы: XOR қабылдағыш, таратқыш

Команда операнд түрлі мәні бар орнатылады, яғни, қабылдағыштың барлық бит бірлігін белгілейді, бірлігі - операндтар бір нөл мәні және басқа да бит орнатылады . Егер екі операнд осы жағдайда нөлдік немесе бір бар болса, онда команда қабылдағыш битті тазалайды.

NOT - логикалық «ТЕРІСТЕУ». Команда форматы: NOT операнд.

Команда таратқыш операндының барлық биттерін терістеу:1-ден 0-ге, 0-ден 1-ге ауыстыруды орындайды. Команданың орындалуы ешқандай белгілерге әсер етпейді.

TEST- логикалық тексеру командасы.

Команда форматы: TEST қабылдағыш, таратқыш.

Команда операндтармен AND операциясын орындайды және тек белгілерді ғана өзгертеді. Команда орындалғаннан кейін OF = CF = 0, SF, ZF және PF өз мәнін өзгертеді.

10.2 Жылжыту және циклдық жылжыту командалары

Осы топтағы команда операндтар жеке бит түрлендіруге мүмкіндік береді және мынадай қасиеттерге:

- байтты немесе сөзді өңдеу;
- тізілімге немесе жадқа рұқсаты бар;
- оңға және солға жылжыту;
- динамикалық немесе логикалық жылжыту.

Сызықтық жылжыту.

Келесі алгоритм бойынша жылжытуды жүзеге асыру командасын осы түр командасы ⟩ қамтиды:

- басқа «ұзарту » бит cf белгісін орнатады;
- басқа соңында биттік енгізу операнд, 0 мәні бар;
- ығысу кезінде келесі бит cf белгісіне өтеді, алдыңғы ығысу битінің мәні *жоғалады.*

Сызықтық жылжыту командасы екі типке бөлінеді: логикалық және арифметикалық сызықтық жылжыту. Логикалық сызықтық жылжыту командасына келесілер жатады:

- *SHL операнд, санауыш_жылжыту* – логикалық солға жылжыту. Тағайындалған операнд мәні санауыш_жылжыту солға ауысатын бит санымен анықталады. Оңға (кіші бит позициясына) нөлдер енгізіледі;
- *SHR операнд, санауыш_жылжыту* – логикалық оңға жылжыту. Тағайындалған операнд мәні санауыш_жылжыту оңға ауысатын бит санымен анықталады. Солға (үлкен, белгі бит позициясына) нөлдер енгізіледі.

Арифметикалық сызықтық жылжыту командаларының логикалық жылжыту командасынан айырмашылығы, олар операндтың ерекше белгілі разрядымен жұмыс істейді.

Команда *SAL операнд,санауыш_жылжыту* – арифметикалық солға жылжыту. Тағайындалған операнд мәні санауыш_жылжыту солға ауысатын

бит санымен анықталады. Оңға (кіші бит позициясына) нөлдер енгізіледі. SAL командасы операнд таңбасын сақтамайды, бірақ операнд таңбасын өзгерткен кезде cf асатолу белгісін келесі битпен орнатады. SAL командасы SHL командасына ұқсас.

Команда *SAR операнд,санауыш_жылжыту* – арифметикалық оңға жылжыту. Тағайындалған операнд мәні санауыш_жылжыту оңға ауысатын бит санымен анықталады. Солға нөлдер енгізіледі. SAR командасы операнд таңбасын сақтайды, әрбір келесі бит ауысқаннан кейін оны қалпына келтіреді.

Сызықтық жылжыту командасының жұмыс істеу принципі 5 суретте көрсетілген.

10.3 Циклдық жылжыту

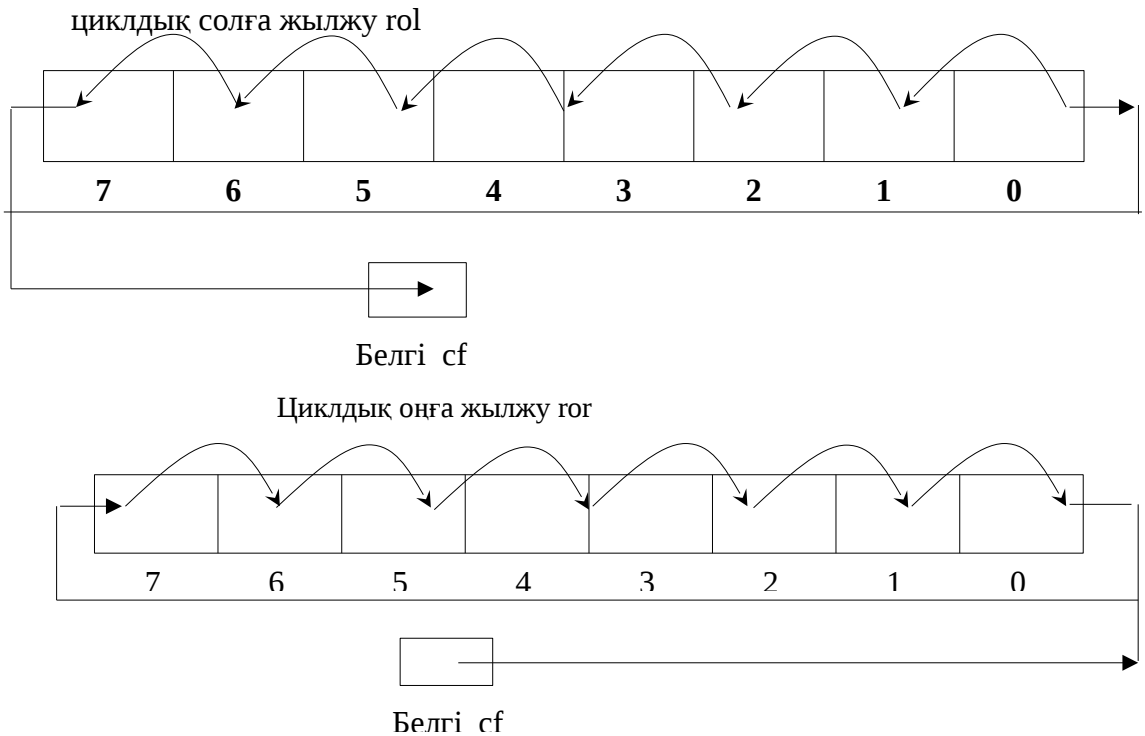
Циклдық жылжыту командаларына жылжыту битінің мәнін сақтайтын командалар жатады. Циклдық жылжытудың екі түрі бар:

- жай циклдық жылжыту командалары;
- cf тасымал белгісі арқылы циклдық жылжыту командасы.

Жай циклдық жылжыту командаларына жатады:

Команда ROL операнд, санауыш_жылжыту – циклдық солға жылжу. Тағайындалған операнд мәні санауыш_жылжыту солға ауысатын бит санымен анықталады. Солға жылжу биті оң жақ операндқа жазылады.

Команда ROR операнд, санауыш_жылжыту – циклдық оңға жылжу. Тағайындалған операнд мәні санауыш_жылжыту оңға ауысатын бит санымен анықталады. Оңға жылжу биті сол жақ операндқа жазылады.

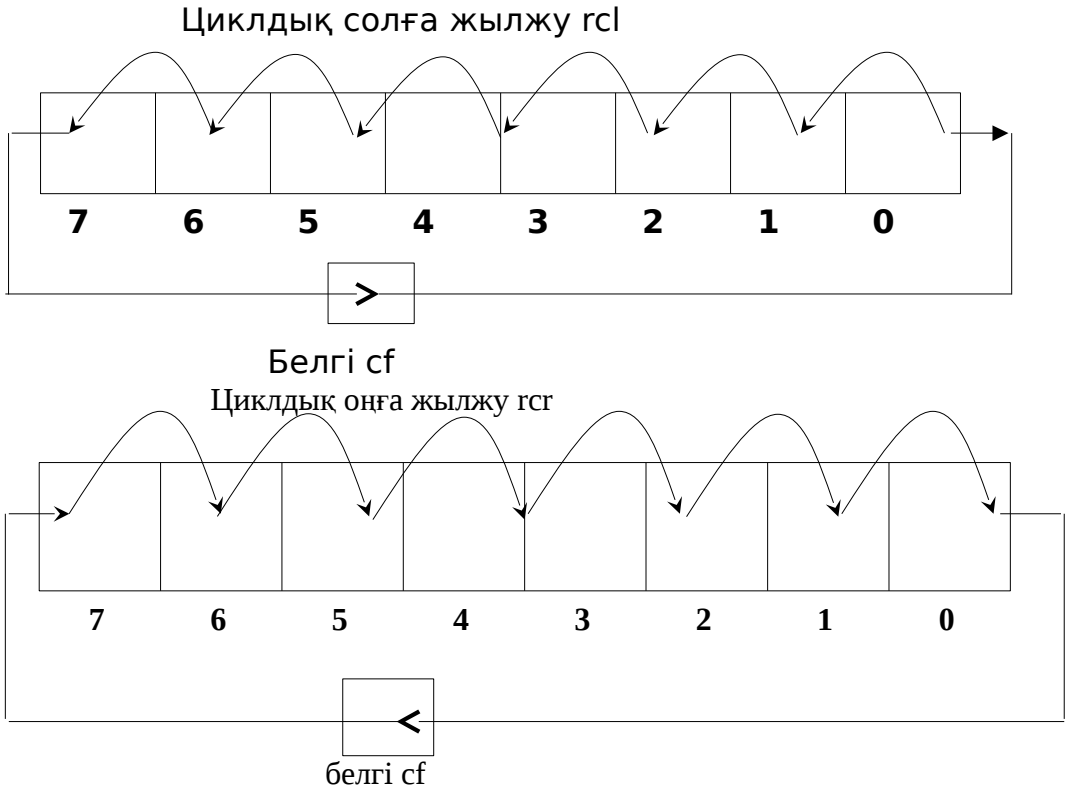


10.3 сурет - Жай циклдық жылжыту командасының жұмыс істеу сұлбасы

cf тасымал белгісі арқылы циклдық жылжыту командалары:

- команда *RCL* операнд, санауыш_жылжыту – тасымал белгісі арқылы циклдық солға жылжу. Тағайындалған операнд мәні санауыш_жылжыту солға ауысатын бит санымен анықталады. Жылжыған бит cf тасымал белгісінің кезекті мәніне айналады.
- команда RCR операнд,санауыш_жылжыту – тасымал белгісі арқылы циклдық оңға жылжу. Тағайындалған операнд мәні санауыш_жылжыту оңға

ауысатын бит санымен анықталады. Жылжыған бит cf тасымал белгісінің кезекті мәніне айналады.



10.4 сурет – cf тасымал белгісі арқылы циклдық жылжу командасы

11 Дәріс №11. Жолдарды өңдеу командалары

Дәрістің мақсаты: жолдарды өңдейтін программалық командаларды зерттеу.

Дәрістің мазмұны: қайталау префикстер түсінігі, жүктеу командалары, салыстыру, сканерлеу, іздеп-табу, жолдарды сақтау.

11.1 Қайталау префикстері

Жолдарды өңдеу командалары байт блоктарымен немесе сөз жадының әрекеттерін орындайды. Блоктың немесе жолдың ұзындығы 64 кб жетеді.

Команда 1 циклде 1 элемент (байт немесе сөз) өңдей алады.

Қабылдағыш – жол қосымша ES регистрінде, ал таратқыш – жол DS деректер сегментінде орналасады деп болжайды микропроцессор. Қабылдағыш – жол DI регистрі арқылы адрестеледі, ал таратқыш - жол SI тізілімі арқылы. DF белгісі жолды өңдеу бағытын көрсетеді. Егер DF=0, жолды өңдеу солдан оңға жүреді, үлкейту адресі бағытында. Егер DF=1, жолды өңдеу оңнан солға жүреді , адрестеу азаяды.

DF белгі бағытын мына команда көмегімен орнатуға болады :

CLD (DF=0); STD (DF=1)

Микропроцессор қайталау префикстері тізбекті командаларды қайталап орындатуды жалғастырады. Қайталау саны CX (ECX) тізілімінен алынады.

REP префиксі – жолдың соңы табылғанға дейін қайталау керек екендігін білдіреді, яғни CX регистрінің мәні нөлге тең болғанға дейін қайталайды.

Ал қалған қайталау префикстері қайталау керек пе, жоқ па екендігін белгілер тізіліміндегі ZF нөлдік белгіге байланысты орындайды.

Сәйкесінше, олар жолдарды салыстыру және жолдардағы мәндерді іздеу командаларымен бірге қолданылады, олар ZF нөлдік белгіге әсерін тигізеді.

REPE префиксі (тең болғанға дейін қайтала), REPZ префиксі (нөл болмағанша қайтала), команданы ZF нөлдік белгі мәні 1-ге және CX тізілімінің мәні 0-ге тең емес болғанға дейін қайталайды. REPNE префиксі (тең болмағанша қайтала), REPNZ префиксі (нөл болмағанша қайтала), команданы ZF нөлдік белгі мәні 0-ге және CX регистрінің мәні 0-ге тең емес болғанға дейін қайталайды.

11.2 Жолдарды өңдеу командалары

MOVS *қабылдағыш-жол, таратқыш-жол.*

MOVSB/MOVSW/MOVSD.

Облыс үшін, қабылдағыш - жол, сегменті ES тізілімі болып табылады, және DI тізілімі салыстырмалы адресті болады. Облыс үшін, жолды жүктеу, DS тізілімі сегменттік тізілім болып табылады, ал SI тізілімі салыстырмалы адресті болады.

Топтық алып-орналастыру келесі 5 қадам бойынша орындалады:

1) DF бағыт белгісін нөлдейді оны кіші адрестерден үлкен адрестерге, әйтпесе керісінше ауысу болатындығына байланысты орналастырады.

2) SI тізіліміне таратқыш-жол адресінің ығысуын жүктейді.

3) DI тізіліміне қабылдағыш-жол адресінің ығысуын жүктейді.

4) CX тізіліміне элементтер санауышын жүктейді.

5) REP префиксі бар MOVS командасын орындайды.

Мысалы:

```
1) CLD; DF=0:солдан оңға бағытталуы      LEA  SI, SOURCE
LEA  DI, ES:DEST
MOV  CX, 100; цикл  100-ге
REP  MOVS DEST, SOURCE
2) CLD
LEA   SI, SOURCE
LEA   DI, ES:DEST
MOV CX, 100
REP  MOVSB ; топтық алып-орналастыру байты
```

11.3 Жолдарды салыстыру командасы

CMPS қабылдағыш_жол, таратқыш_жол.

CMPSB/CMPSW/CMPSD.

Кез келген өлшемді жолдарды салыстыру. REPE префиксі (операциялық жолдарды қайталау) әдетте жолдарды салыстыру командасының алдына қойылады.

Мысалы: REPE CMPSB DEST, SUORCE.

REP префиксін пайдаланған кезде салыстыру жол ұзындығының мәні CX тізілімінде болуы керек.

Командалар CMPS бір жад облысы адрестеу тізілімдерді DS:SI тізілімімен салыстырады, басқа облыс мазмұнымен ES:DI адрестеу тізілімдері.

Егер белгі DF=0, онда салыстыру оңнан солға болады, SI және DI тізіліміндегі адрестер әр салыстыру сайын көбейеді. Егер белгі DF=1, салыстыру солдан оңға болады, SI және DI тізіліміндегі адрестер азаяды.

Мысалы: ұқсас элементті жұп табылмағанша салыстыру.

```
    CLD
    MOV  CX, 100
    REPE CMPS DEST, SOURCE
    JNE  M1; сәйкес емес табылды => M1.
....
    M1:
```

Жолдарды сканерлеу командасы.

SCAS *қабылдағыш_жол.*

SCASB/SCASW/SCASD.

Жолдағы белгілі бір байт немесе сөзден кейін орындайды. SCASB командасына керекті мән AL тізіліміне жүктейді, ал SCASW үшін AX тізілімі. ES:DI тізілім жұп сканерленуге тиіс жадыдағы жолды көрсетеді. Команда деректері REPE және REPNE префикстерімен қолданылады. Егер белгі DF=0, онда операция жадыны солдан оңға сканерлейді және DI регистріндегі адресті улкейтеді. Егер DF=1, онда оңнан солға сканерлейді және DI тізілімінің адресін кішірейтеді.

Мысалы: элементті іздеу, басқа ’_’.

```
CLD
LEA  DI, ES: B_STRING
MOV AL, ’_’
MOV  CX, 100
REP   SCAS B_STRING
```

SCAS командасы әсіресе тексттік редакторларда пайдалы, программа жолды сканерлеуге тиіс жағдайда, пунктуациялық іздеу белгілерін орындайтын: нүкте, пробел мен үтір.

Жолдарды жүктеу командасы.

LODS *таратқыш_жол.*

LODSB/LODSW/LODSD.

Жадыдан бір байт AL тізіліміне немесе сөз AX тізіліміне жүктеледі.

Жад адресі DS:SI регистрлерімен анықталады. DS:SI жұптық регистр байт жадында адрестеледі (LODSBW үшін).

DF белгісінің мәніне байланысты SI тізіліміндегі мәннің үлкеюі және кішірейуі орындалады.

Мысалы: элементтердің кездейсоқ емесін іздеу, кездейсоқ емес табылса - AL тізілімінде элементті санау.

```
CLD
LEA  DI, ES; DEST
LEA  SI, SOURSE
MOV CX, 500
REPE CMPSB
JNE M1      ; кездейсоқ табылды
DEC SI      ; иә, SI тізілімін түзету
LODS        SOURCE    ; AL – де элементті санау
...
    M1:       ; кездейсоқ емес.
```

11.4 Жолдарды сақтау командасы

STOS командасының жазылу форматы *таратқыш_жол.*

STOSB/STOSW/STOSD.

Команда жадыда байт немесе сөзді сақтайды. Жадының адресі әрқашан ES:DI тізілімінде болады. REP префиксін пайдаланған кезде операция байт немесе сөз мәнін бірнеше рет қайталанады.

STOSB үшін – байт AL тізіліміне жүктеледі.

STOSW үшін – сөз AX тізіліміне жүктеледі.

Команданы берілген мәнмен жолдарды толтыруды орындаған ыңғайлы.

Мысалы: ұзындығы 200 сөз DEST жолын 1 нөлдік емес элементті іздеуді сканерлеу, егер табылса, онда келесі бес сөз нөлденеді.

```
CLD
LEA  DI,ES:DEST
MOV  CX, 200
MOV  AX, 0 ; ізделініп отырған мән 0
REPNE  SCASW
JNE  M1 ; нөлдік емес сөз табылды
SUB  DI, 2  ; иә, DI тізілім жөндеу
MOV  CX, 6
REP    STOS W_STRING  ; 6 сөзді 0-мен толтыру
...
M1:   жоқ
```

12 Дәріс №12. Ассемблердің директивалары

Дәрістің мақсаты: ассемблердің директиваларын және программада

қолданылуын оқып үйрену.

Дәрістің мақмұны: сегментті анықтайтын директивалар, деректер.

12.1 Сегментті анықтайтын директивалар

Ассемблер тіліндегі программа операторлардан тұрады. Оператор ретінде командалар, макрокомандалар және директиваларды қолдануға болады.

Директивалар ассемблерлеу процесін басқарады және машиналық кодты генерацияламайды.

Программа бөлек сегмент түрінде жасалады және оның ішінде мәліметтер, стек, шарт белгілер және қосымша сегменттері кіреді. Сегментті сипаттаған кезде басы SEGNENT нұсқауымен, ал соңы ENDS нұсқауымен белгіленеді. Сегмент келесі форматта сипатталады:

Сегмент_аты SEGMENT [теңестіру_түрі] [біріктіру_түрі] [класс] [сегмент_өлшемінің_түрі]

...

Сегмент_аты ENDS

Теңестіру_түрі сегмент басының шекарасын табады және келесі тағайындаудың бірін алуы мүмкін:

BYTE - сегмент кез келген адрестен басталады

WORD - сегмент екіге бөлінетін (xxx0b) адрестен басталады

DWORD сегмент төртке бөлінетін (xx00b) адрестен басталады

PAPA - сегмент параграф адресінің шекарасынан басталады, 16-ға бөлінетін (xxx0h) үнсіздікпен қолданылады

PAGE - сегмент 256-байттық бет адресінің 256 бөлінетін (xx00h) шекарасынан басталады.

MEMPAGE — сегмент 4 Кбайтқа бөлінетін (x000h) адрестен басталады

Біріктіру_түр сегменті берілген сегмент қарастыру кезінде басқа сегменттермен біріктіруін анықтайды.

Келесі біріктіру түрлері қолданылады:

PRIVATE – сегмент берілген модульден тыс басқа атпен берілген басқа сегмент топтарымен бірікпейді, үнсіздікпен қолданылады.

PUBLIC – бір атпен берілген сегменттерді біріктіреді, біріктірген сегменттің ұзындығы біріктірілген сегменттердің қосылған ұзындығына тең.

COMMON – бір атпен берілген сегменттерді бір адреске біріктіреді. Бұл атпен берілген сегменттер бірін-бірі жабады және жадыны бірге қолданады, біріктірген сегменттің ұзындығы қосылған сегменттердің ең үлкен ұзындығына тең.

STACK – стек сегментін анықтау. Жөндеуішті бір атпен берілген сегменттерді қосуға мәжбүр етеді және бұл сегменттердің адресін SS регистріне сәйкес анықтайды. Сегменттің ұзындығы біріктірілетін сегменттердің қосындысына тең. Егер стек сегменті құрылса, STACK біріктіру түрі қолданылмаса, программист ss регистріне сегмент адресін жүктеу керек (осыған ұқсас, DS тізілімі үшін орындалатындай).

MEMORY – кодтар сегментінен кейін орналасқан деректер сегментін шақырады. Сегменттің өлшемі COMMON біріктіруіне ұқсас анықталады.

AT – ПАРАГРАФ – берілген операнд бекітілген адрестер бойынша таңбалар мен айнымалыларды анықтауды қамтамасыз етеді.

Мысалы, экрандық видеобуфер адресін анықтау үшін қолданылады

VIDEO - RAM SEGMENT AT 0B800H

Класс. Апострофқа алынған берілген операнд құрастыру кезінде топтау үшін қолданылады. «Класс» операнды ретінде «STACK», «CODE», «DATA» аталымдарын пайдалануға болады.

Сегмент өлшемінің типі. i80386 және одан жоғарғы микропроцессорлар сегменттері 16 немесе 32 разрядтық болады. Бұл негізінен біріншіден сегмент өлшеміне және оның ішіндегі физикалық адрестің қалыптастырылуы тәртібіне әсер етеді. Сегмент өлшемдерінің келесі типтері қолданылады:

USE16 – бұл сегмент 16 разрядтық адрестеуді қолданатындығын білдіреді. Физикалық адресті қалыптастыру кезінде 16 разрядтық ығысу қолданылады. Сәйкесінше, мұндай сегмент 64 Кбайт кодтар сегментінен немесе деректер сегментінен тұрады;

USE32 - сегмент 32 – разрядтық болады. Физикалық адресті қалыптастыру кезінде 32 разрядтық ығысу қолданылады. Сондықтан мұндай сегмент 4 Гбайт кодтар сегментінен немесе деректер сегментінен тұрады.

Сегментті анықтаудың қысқаша директивалары.

.CODE [аты] **–** код сегментінің басы немесе жалғасуы.

.DATA - инициализацияланған деректер сегментінің басы немесе жалғасы. Сондай-ақ , near типіндегі деректерді анықтау үшін қолданылады.

.CONST – модульдің тұрақты деректерінің (тұрақтыларының) сегментінің басы немесе жалғасы.

.DATA? - инициализацияланбаған деректер сегментінің басы немесе жалғасы. Сондай-ақ , near типіндегі деректерді анықтау үшін қолданылады.

.STACK [өлшемі] **–** модуль стек сегментінің басы немесе жалғасы. Параметр [өлшем] стек өлшемін береді.

.FARDATA[аты] – инициализацияланған far типтегі деректер сегментінің басы немесе жалғасы.

.FARDATA[аты] - инициализацияланбаған far типтегі деректер сегментінің басы немесе жалғасы.

12.2 Процедураны анықтау дерективасы

Программа өзіне бір немесе бірнеше процедураны қосады. Процедураны сипаттаған кезде басында PROC дерективасы қойылады, ал соңы ENDP дерективасымен аяқталады. Процедура келесі формат бойынша сипатталады:

Аты_рәсім ENDP

Қашықтық атрибуты ретінде FAR және NEAR операндтары қолданылады.

FAR ара қашықтық атрибутын қолданған кезде процедураға жүгінуді басқа программалық сегменттен жүзеге асырылады, ал NEAR атрибутын қолданғанда тек өзі сипатталатын сегменттен жүзеге асырылады.

RET командасы процедураны қайтаруды орындайы. Процедуралар қабаттасуы мүмкін. Қабаттасу деңгейі тек стек сегментінің өлшемімен шектеледі.

12.3 ASSUME директивасы

Нұсқау ASSUME ассемблерге қандай программалық сегменттер программаға жататындығын хабарлайды және сегменттер адрестерін шарт белгілер мен стек сегменттері үшін енгізеді. Қосымша сегмент және деректер сегменті үшін сегменттік адрестерді енгізуді программалаушы қамтамасыз етеді.

ASSUME нұсқауы шарт белгілер сегментіне келесі түрде сипатталады:

ASSUME SS:стек_аты, DS:деректер_аты, CS:шартбелгілер_аты,ES:қосымшат_аты

Егер программа қайсыбір сегментті қолданбаса, онда оны көрсетпей кетуге болады немесе NOTHING көрсету қажет.

Мысалы:

ASSUME SSDDEG,DS:DSEG,CS;CSEG әлде

ASSUME SSDDEG,DS:DSEG,CS;CSEG, ES:NOTHING

Содан да басқа NOTHING арқылы сегменттік тізілімнің тағайындауын қайтарып тастауға болады.

Программаны ақтайтын END директивасы.

Нұсқау END Ассемблерге қай жерде аяқтау керек екенін хабарлайды және келесі түрде сипатталады:

END[енгізу белгісі]

Енгізу белгісі қызметінде негізгі процедураның аты, әлде программаның басының белгісі қолданылады.

12.4 Мәліметтерді анықтайтын директивалар

Мәліметтерді анықтау үшін келесі нұсқаулар қолданылады:

- DB *–байт анықау;*
- DW*–сөзді анықтау;*
- DD *–қос сөздерді анықтау;*
- DQ *–8 байты немесе 4 сөзді анықтау;*
- DT*–10 байтты анықтау;*

Мәліметтерді анықтау нұсқаулардың форматының түрі:

[аты]Dn оқылуы

Осыған байланысты оқылуда мыналар қолданылады:

- тұрақты;
- кесте, массив немесе жол;
- символдық жол.

Тұрақты.

Мәліметтерді анықтау нұсқаулардың көмегімен айнымлыны тұрақты түрінде беруге болады:

A DB 10

Егер айнымалыны DD нұсқауы көмегімен жазылса немесе бастапқы мәнін көрсетпей, еске сақтау жадынан орын алуға болады.

A DD 10203040H, ал бізге нақты байт керек болса, онда оған келесі түрде жүгінуге болады:

```
MOV AL, byte ptr A          ;AL=40
MOV AL, byte ptr A+2        ;AL=20H
```

Нақты бір сөзге жүгіну үшін:

```
MOV AX, word ptr A          ;AX=3040H
MOV AX, word ptr A+2        ;Ax=1020H
```

DW нұсқауының көмегімен жадында қандай да бір белгімен процедураның, ығысу адресін сақтауға болады:

ADR_NEAR DD MAIN

DD нұсқауының көмегімен жадыда белгі мен процедураның толық адресін жазуға болады:

ADR_FAR DD MAIN

Кесте.

Кестені жазу кезінде, кестедегі элементтер үтір арқылы жазылады:

TAB DB 10,20,30,40,50

Немесе тек еске сақтау жадыдан кестеге орын алуға болады:

TAB DB 5 DUP(?)

Бұл жерде кестедегі р элемент 1 байт көлемінде орын алады.

Бұл жағдайда кесте 10 байт орын алады. Әр элементке 2 байт немесе сөз. Оператор DUP қайталау операциясын жүргізеді. Таблицаның элементтеріне жүгінгенде, оның өлшеміне қарамастан, кіші байт элементі жүреді.

Мысалы:

MAS DW 1122H, 3344H, 5566H,7788H

Мұндағы нөлдік сөздің адресі бойынша 22h мәнімен анықталатын санау нөлден басталады:

MAS- байттың адресі 22h мәні бойынша

MAS+1- байттың адресі 11h мәні бойынша

MAS+2- массив бірінші сөзінің адресі

MAS+4-массивтің екінші сөзінің адресі

Символдық жол.

DB нұсқауының көмегімен еске сақтау құрылғысында кез келген мәтінді сақтауға болады. Тырнақшаға алынған кез келген символдар,

символдық жол болып келеді. Еске сақтау құрылғысында әр символ ASCII-шарт белгіде жазылады.

Мысалы: STR DB ‘символдық жол’

Анықтағышты анықтайтын директивалар.

Анықтағышты анықтайтын директиваларға келесі нұсқаулар кіреді.

EQU

Бұл директивалар деректерге аталымдар меншіктейді және жадының орынды алмайды. Келесі пішім қоданылады:

аты EQU *өрнек*

аты=сандық өрнек

EQU директивасының «=» айырмашылығы мынада: «=» директивасының көмегімен сандық өрнекті ауыстыруға болады. Ал EQU директивасының көмегімен тұрақтыға, тізілімге, адрес комбинациларына аталымды меншіктеуге, синононимді анықтауға болады.

12.5 Сыртқы сілтеме директивалары

Сыртқы сілтеме директиваларға келесі директивалар кіреді :PUBLIC, EXTERN және INCLUDE.

PUBLIC директивасы берілген индефикатор басқа программалық модульдерден қалай алынғанын көрсетеді. Идентификатор есебінде айнымалы, белгі, тұрақтыны қолдануға болады. Директива келесі пішім түрінде болады:

PUBLIC идентификатор[,…]

Мысалы:

```
PUBLIC A
DSEG SEGMENT
A DW 1020H
DSEG ENDS
```

EXTERN директивасы жариялайды, берілген программалық модульде аттар қолданылады. Директива EXTERN пішімі:

EXTERN аты:түрі[,…]

Аты қызыметінде айнымалылар , белгілер,тұрақтылар қолданылады.

Егерде аты айнымалы болып келсе және деректер сегментінде сипатталса, онда түрі мынадай мән қабылдай алады: WORD,DWORD,BYTE. Егерде аты белгі болып келсе, онда түрі мән қабылдайды: FAR немесе NEAR. Егерде аты тұрақты болып келсе және директивасының көмегімен сипатталса, онда түрі ABS мәнін алады.

INCLUDE директивасы аудармалау кезінде программаға директивада көрсетілмеген файлдың мәтінін енгізеді. Директиваның пішімі: INCLUDE файл_аты

Листингті басқару директивалары.

Оларға PAGE,TITLE және SUBTTL директивалары жатады.

PAGE директивасы листинг бетінің өлшемін береді. PAGE директивасы мынадай пішім түрінде жазылады: PAGE[жол_саны][,баған_саны]

Жолдың саны 10-255 дейінгі аралықта өзгертіледі.

Бағана жолдағы символдар санын көрсетеді және 60-132 дейінгі аралықта өзгереді. Аталмаған жағдайда беттердің өлшемі тағайындалады, бір жолға 80 символды 66 жолға орнатады.

PAGE директивасы операндасыз беттерді жылдам ауыстырады.

TITLE директивасы әр беттің жоғарғы жағында программаның бастамасын жазады. TITLE директивасының пішімі келесі түрде болады:

TITLE мәтін

Мәтіннің максималды ұзындығы – 60 символ.

SUBTTL директивасы бастамадан кейінгі келесі жолда ішкі бастаманы жазады. Оның форматы: SUBTTL мәтін.

13 Дәріс №13. Программаларды ұйымдастыру. Макроанықтағыштар

Дәрістің мақсаты: ассемблер тілінің макроқұралын оқып үйрену.

Дәрістің мазмұны: негізгі түсініктер, макроанықтамалар және макрокомандалар.

13.1 Ассемблердің макроқұралдары

Ассемблер тілінің макроқұралдары IBM PC Ассемблер тілінің ең мықты құралдарының бірі болып табылады. Оның командалар жинағын бір команда ретінде қолдана алатыны қабілеті үшін оны макроассемблер деп те атайды. Осылайша кейбір процедураларды макрос түрінде қорытып, сақтай аламыз.

Макроқұрылғыларды қолдану бізге мынандай мүмкіндіктер береді:

- программаны макрокомандалар қолдану арқылы түсініктірек жасай аламыз;
- берілген мәтінді жеңілдетіп, қысқарта аламыз;
- кодтаудың мүмкін болатын қателіктер санын дамыған макрокомандаларды қолдану арқылы азайта аламыз;
- кіру параметрлер арқылы программаны динамикалық түрде өзгерте аламыз;
- шартты ассемблерлеу және листингті басқару директиваларын қолдану;
- өзінің макроанықтамалар кітапханасын жасау;
- айтарлықтай программаны орнындауды процедураны шақыруға үстеме шығынның болмауы арқылы тездете аламыз, дегенмен программаның ұзындығы өседі.
- дайын макроанықтамалар кітапханасымен қолдана алу және оларды өзіміздің программаға қоя аламыз. Ол бізге ассемблердегі программалауды тездетеді.

Макростарды қолданудың өзіндік кемшіліктері бар:

- макростар бастапқы файлдар ретінде сақталады және программаға қосылған кезде ассемблерлеуді қажет етеді және олар көп болса ассемблерлеу уақыты өседі.

- макростар жұмыс нәтижесін оңай жасыра алады.

Макростармен С тілі функцияны сипаттау деңгейінде препроцессорлік #define директивасы арқылы кең жұмыс істей алады және С++ тілі #inline функциясы деңгейінде. Бірақ бұл тілдер макростарға айтарлықтай жеткілікті шектеулер қояды, әрине ол ассемблерде жоқ нәрсе. Ассемблер IBM PC тілі өзіне үш құрастырушы қосады:

1) Макроанықтамалар (макрос) - бұл белгілі бір іс-әрекетті немесе алгоритмі бар командалар жинағы. Жалпы кез келген ассемблерлік процедурадан макрос жасай аламыз. Сегменттер анықталғанға дейін макроанықтамалар программаның басында болуы керек. Макростардың келесідей құрылымы бар:

макростың аты MACRO[ФормальдыПараметрлер]

; макрос денесі

Endm

2)Макрокоманда - макроанықтамаларға қысқаша сілтеме (макростарды шақыру):

Макрос аты[НақтыПараметрлер]

3)Макрокеңейту (макроподстановка, макрокірістіру) - макрокоманда макросы орнына формальдік параметрлерді нақтылармен ауыстырамыз, егер олар бар болса.

Макроанықтамалар жәй және қабаттасқан болуы мүмкін, демек өздерінде басқа макроанықтамалар бар. Макроанықтамалардың қабаттасу деңгейі әртүрлі болуы мүмкін. Бір макростан біз басқа да макростарды шақыра аламыз. Макростарда командалар әртүрлі бола алады.

13.2 Көпмодульді программа ұйымдастыру

Қиын программалар комплексін жасау кезінде процедураның бір бөлігін бөлек бастапқы модульге бөліп шығару ыңғайлы, және олар дербес жіберіледі.

Жіберілген соң бұл процедураның объектік модулі объектік кітапхана құрамына кіре алады, құрастыру кезеңінде негізгі программаға қосылады. Мұндай программа кешенінің құрылымы келесідей түрде көрініс алады:

```
;MAIN.ASM - Басты процедурамен файл
text  segment public ‘code’
      extern     subpr: proc                ;supr- сыртқы сілтеме
mymain proc
…
Call               subpr                    ; тікелей жақын шақыру
…
Mymain endp
```

```
Text ends
        End          mymain                ;mymain- басты процедураға кіру нүктесі
;MYSUB.ASM-
text  segment public ‘code’
PUBLIC SUBPR                               ;subproc- процедура «жалпы қолдану»,
Subpr  proc near                           ;жақын процедура
…
Ret
Supr endp
Text  ends
        End                                ;кіру нүктесінсіз
```

Екі программалық сегменттінде бастапқы модульдерінің бір ғана аттары ба Text, ол екеуінің бірыңғай болып қосылуын кешен арқылы қамтамасыз етеді. PUBLIC қосылу түрі конкантенация арқылы қосылу түрін анықтайды, демек екінші моуль бріншінің соңына қосылады (және де сипаттауыш STACK жұмыс істейді, сегменттік стектерде қолданылады, сол уақытта қосылу түрі COMMON кешеннен бірыңғай сегменттерді бір-біріне салуды қажет етеді).

ETERN директивасы басты модульде анықталады, символдық мағынасы сыртқы сілтеме болып табылады және процедура атын береді. Сыртқы сілтеме кешен кезеңінде рұқсат етіледі.

Басқа программалық модульдерден шақырылатын процедуралар PUBLIC директивасы көмегімен «жалпы қолданыстағы» процедура ретінде анықталуы тиіс.

Берілген мысал программалық модульдер қосылған соң бір команда сегментін құрайтындығын болжайды, демек соммалық өлшемі 64 байттан аспайды. Сол себепті ішкі программалық процедура жақын болып жарияланады, ал басты процедурада тікелей жақын программа қолданылған.

Ішкі программалық модуль аяқтау директивасы арқылы аяқталады (END) егер модуль басты процедураны қамтыса және сол процедураға кіру нүктесін көрсеткен жағдайда.

MAIN.ASM және MYSUB.ASM файлдары бастапқы мәтін компоненттер кешеніне бөлек жіберу керек, ал сосын бірыңғай жүктеу модульіне біріктіру:

```
MASM MAIN,MAIN,MAIN;
MASM MYSUB,MYSUB,MYSUB;
LINK MAIN+MYSUB,PROG.
```

Жүктеу нәтижесінде MAIN.OBJ,MAIN.LST ,MYSUB.OBJ ,MYSUB.LST файылдары аламыз. Компоновщик MAIN.OBJ және MYSUB.OBJ файлдарын қосып жүктеу (орындау модулі) PROG.EXE жасайды.

Көп модульдік программалық комплексті процедураны жасаудың басқа нұсқасы- MYSUB.OBJ объектік модульді кітапханаға сыйдыру және оны дәйекті ол жақтан шығару. Кітапхананы орынды қолдану қажет, көптеген модульдік объектер болған жағдайда, әртүрлі программалық комплекстерде.

Ол жағдайда жүктеу аяқталған соң MYSUB.ASM файлы объектікфайл MYSUB.OBJ қолданушының кітапханасына жазылады. Объектік файлдарды программалармен жаңа құрылған кітапханаға жазу келесі түрде орындалады:

LIB MYOBJ.LIB + MYSUB.OBJ,MYOBJ.LST.

LIB командасы – кітапханашы аты, MYSUB.OBJ – кітапханаға жазылатын объектік файлдың аты, ал MYOBJ.LST. – құрастырылатын файлдың аты. Аттың алдынддағы «+» белгісі кітапханаға қосу дегенді білдіреді(«-» белгісі өщшіру). MYOBJ.LST. файлына кітапхананың мазмұны жазылады.

Барлық кеңейтуді, файл католігіне басқа, жіберуге болады. Себебі мысалда келтірілген әдепкі бойынша болжайды.

Егер объект кітапхана бар болса және біз оған объектік модуль қосқымыз келсе, онда кітапханашыны шақыру жолына кітапханашының атын жазу керек:

LIB MYOBJ+NEW1+NEW2+NEW2,MYOBJ.LST,MYOBJ.

Бұл жерде бірінші параметр бастапқы кітапхананың атын анықтайды, ал соңғы- құрастырылатын кітапхана атын, жаңа модульдер арқылы кеңейтілген. Бұл ат бұрыңғы кітапхана атымен ұқсас болуы мүмкін және де ұқсамауы да мүмкін. Объектік кітапхананың бар болуы компоновщикті шақыру жолында команданың төртінші параметрі ретінде кітапхананың атын көрсету тиіс:

LINK/CO MAIN.OBJ,PROG.EXE,PROG.MAP,MYOBJ.LIB.

Көрсетілген мысалда MAIN.OBJ –объектік файл негізгі программамен, PROG.MAP подпрограмманы талап етеді. Алдында көрсетілген мысалдарда, барлық кеңейтулерді жіберуге болады, солай олар әдеп бойынша болжанады және тағайындалады. Орындалатын модуль картасы әншейінде керек емес, ал жүктеу модулінің аты әдеп бойынша объектімен ұқсайды. Сондықтан келтірілген команданы қысқартуға болады:

LINK MAIN,PROG,MYOBJ.LIB

Бұл команданы орындау нәтижесінде компоновщик орындаушы модуль PROG.EXE жасайды.

Егер басты программаның соммалық өлшемі және программа 64 кбайттан асса, жүктеу модулі көп сегментті бола алады. Мұндай программалық кешеннің құрылымы келесідей болады:

```
;MAIN.ASM-
text1  segment     ‘code’
       extern        subpr:proc
text1  ends
text   segment     ‘code’
mymain  proc
…
Call   ptr subpr
…
Mymain ends
Text ends
```

```
        End mymain
;MYSUB.ASM
Text1  segment  ‘code’
         Public    subpr
Subpr  proc       far
…
Subpr   endp
Text1  ends
         End
```

Text және Text1 сегменттерінің аттары әртүрлі болғандықтан, олар қосылмайды. Бастапқы модульде, программаны шақыруы бар, ол EXTERN директивасымен сипатталады. Бірақ бұл сипаттама сегментте болуы қажет. Сол себепті MAIN.ASM модулі Text1 бос сегментіне қосылған, ол тек шақырылатын процедураның бір ғана жолын қамтиды *EXTERNSUBPR&PROC.* Одан кейін басты программамен Text1 командасының негізгі сегменті жалғасады. Онда тікелей алыс шақыру қолданылады.

Процедура-программа, өз кезегінде , FAR атрибутымен жарияланады. Жіберу және компоновка көп модульдік бір сегментті программа сияқты орындалады.

13.3 Макрокомандалар

Ассемблер тілінде жазылған командаларда мәтіннің қайталанатын жерлері болады, бірдей құрылыммен. Мәтіннің ондай жерлерін макроанықтамалар ретінде енгізуге болады және нақты аргумменттер тізіміне, ол қажет ететін мәтіннің генерациясын қажет етеді, оны макрокеңейту деп атайды. Нақты аргументті варияциялай отырып, макрокеңейтудің өзгермес құрылымын сақтауға болады және бөлек элементтерін өзгертуге болады.

Макроанықтамалар аргументтер жолында макроанықтамалардың аттарының жолымен және MACRO директивасымен басталу керек . ENDM директивасымен аяқталуы тиіс.

Әр нақты жағдайларда нөмірлерін және олардың тәртібі макроанықтамалар ретінде ерекшеленді:

```
Psh macro a,b,c
     Push   a
     Push  a
     Push  b
     Push  c
     Endm
```

Программаның бастапқы жолында пайда болу

```
     Psh                AX,BX,CX
```

Келесі мәтіннің генерациясына әкеледі:

```
     Push               AX
```

```
    Push        BX
    Push              CX
```
Егер бастапқы мәтінде жол болса
```
     psh        DX,ES,BP
```
онда келісетін макрокеңейту мынадай түрде болады:
```
    push              DX
    push              ES
    push              BP
```
Нақты аргументтер ретінде ассемблердің кез келген белгісі бола алады, берілген командаға рұқсат етілсе. Негізінде макрошақырулар:

Psh men [BX],ES:[17]h

Келесі макрокеңейтуге әкеліп соқтырады:
```
Push  men
Push [BX]
Push ES:[17]
```
Егер қандайда бір макроанықтамалар салыну керек болса, онда белгілер LOCAL операторымен жариялануы керек. Бұл жағдайда ассемблер, макрокеңейтуді генерациялай отырып, белгілерді кеңейтудің өзіндік жолын табады.
```
Delay macro
        Local   point
        Mov     cx,20000
Point: loop     point
        Endm
```
Макроанықтау мәтінін тікелей программа мәтініне қоса аламыз, ол жағдайда, макрокомандалар кейбір стандарттық процедураларды кең түрде сипаттайды.

Макрокітапхана ол макроанықтама мәтіні бар файл. Файлға ол тура мәтін программасына жазылған күйде жазылады. Төменде макрокітапхананың мәтін файлы MYMACRO.MAC атымен берілген, онда екі макрокоманда бар:
```
; outprog   макрокомандасы программаны анықтайды
Outprog macro
          Mov       AX,4C00h
          Int       21h
endm
;delay макрокомандасы кішігірім программалық ұстап қалу
Delay macro
        Local       point
         Mov        CX,2000
Point:loop      point
          Endm
```

Функциялық макроанықтаулар процедураға ұқсас болып келеді. Екеуіне бірлет сипаттау жиелікті, кейін арнайы жолмен шақырылады. Осы жерде олардың ұқсастықтары аяқталады, және айырмашылықтары басталады. Олардың айырмашылықтарының мақсатты орнатуына байланысты артықшылықтары мен кемшіліктері ретінде қарастыра аламыз:

- процедураға қарағанда, өзгермейтін мәтін, макроанықтамалар макрогенерация процесінде нақты параметрлер жиынтығына байланысты өзгере алады. Бұл кезде түзетулер команда операндтары ретінде ұшырайды, және командалардың өздері де. Бұл қатынаста процедуралар нысаны аса икемді;
- әр шақыру кезінде макрокоманда мәтінді макрокеңейту ретінде программаға енгізеді. Процедураны шақыру кезінде микропроцессор басқаруды процедураның басына жіберуді орындайды, ол жадының кейбір бөліктерінде бір ғана дана ретінде болады. Бұл жағайда код шағын болып шығады, бірақ жіберулер әсерінен тез орындалу төмендейді.

13.4 Макродирективалар

Ассемблер макроқұралдары көмегімен макроанықтамаларға кіретін жолдарды ішінара өзгертуді ғана орындамай, оған қоса жолдар жиынтығын және олардың жалғасу тәртібін өзі модификациялайды. Оны макродиректива арқылы жасауға болады. Оларды екі топқа бөлуге болады:

- WHILE, REPT, IRP және IRPC қайталау директивалары. Бұл топ директивалары макрос құру үшін қолданылады, олар бірінен соң бірі жол орналасқан тізбекті жолдардан тұрады. Бұл жолдардың кей жерлері модификациялануы мүмкін;
- EXITM және GOTO директивалары макрокеңейтуді генерациялау процесін басқару үшін қолданылады. Олар сәйкесінше макроанықтағыш жолдар жиынынан тұратын макрокеңейту процесін қалыптастыруды басқару үшін қолданылады. Бұл директивалардың көмегімен макрокеңейтулердегі кейбір жолдарды алып тастауға және генерация процесін тоқтатуға болады. EXITM және GOTO директивалары әдетте компиляцияның шартты директиваларымен бірге қолданылады.

WHILE және REPT директивалары

WHILE және REPT директивалары кейбір жолдар тізбегін анықталған бірнеше рет қайталауды орындау үшін қолданылады. Бұл директивалардың синтаксисі келесі түрде беріледі:

WHILE тұрақты_өрнек
Тізбекті_жолдар
ENDM
REPT тұрақты_өрнек
Тізбекті_жолдар
ENDM

Келесі екі өрнек IRP және IRPC директивалары, бұл процесті ыңғайлы етеді яғни әрбір интеграция сайын кейбір тізбек жолдардағы элементтерді модификациялауға рұқсат береді.

IRP директивасы

IRP директивасының синтаксисі:

*IRP формальді*_аргумен, *<жолдар_символы_1...,жолдар_символы_N>*

тізбеті_жолдар

ENDM

Бұл директиваның қызметі, тізбегі_жолдарды N рет қайталайды яғни IRP директивасының екінші операнында берілген үшбұрыш символдарына қанша жолдар_символы жазылса, сонша рет қайталайды. Ал тізбекті жолдарды қайталау формальді аргументпен орындалатын екінші операндтағы жолдар символымен ауыстырылады. Сонымен бірінші генерация кезінде жолдар символы ауыстырылады. Егер жолдар символы 2 бар болса , онда екінші генерация орындалады яғни тізбекті жолдардағы формальді аргумент жолдар символы ауысады, осылайша жолдар символы N дейін қайталана береді.

Мысалы, программадағы келесі конструкцияны анықтау нәтижесін қарайық:

```
Irp ini<1,2,3,4,5>
    db.  Ini
    endm
```

Макрогенератормен келесі макрокеңейту генерацияланады:

```
db  1
db  2
db  3
db 4
db  5
```

IRPC директивасы

IRPC директивасының синтаксисі:

```
IRPC  формальді_аргумент,  жолдар_символы
                          тізбекті _жолдар
ENDM
```

Берілген директива *IRP* ұқсас, бірақ ол әрбір итерация кезінде формальді аргумент кезектегі жолдар символында орналасқан символмен ауыстырылғанымен ерекшеленеді. Сәйкесінше тізбекті жолдардағы қайталау жолдар символындағы символдың санымен анықталады.

Мысалы:

```
Irpc  rg,
Push rg&x
Endm
```

Генерация процесі кезінде макрогенератормен келесі макрокеңейту генерацияланады:

Push ax
Push Bx
Push Cx
Push dx

Шартты компилция директивалары.

EXITM директивасы операндтары болмайды, бұл директиваның қызметі макроанықтағышта кездескен жерінен бастап макрокеңейту процесін тезарада тоқтату болып табылады.

GOTO таңба_аты директивасы макроанықтағыштағы генерация процесін басқа орынға аустырады.

Шартқа байланысты компиляция директивалары.

Берілген директивалар программалық код фрагменттерін таңдау арқылы аударуды ұйымдастыру үшін қолданылады. Мұндай таңдау компиляциясы макроанықтағыштағы барлық жолдар макрокеңейтуге кірмейтіндігін білдіреді, оларды шартқа байланысты орындайды. Ал нақты қандай шарттар болатындығы шартты директива типімен анықталады.

Барлығы мұндай шартты компиляция директиваларының 10 типі кездеседі:

- IfF және IFE директивалары – логикалық өрнекті есептеу нәтижесіне байланысты шартты аударғыш;
- IFDEF және IFNDEF директивасы- символдық атын анықтау фактіне байланысты шартты аударғыш;
- IFB және IFNB директивасы – макрокоманданы шақыру кезінде нақты аргументтің анықтау фактісіне байланысты шартты аударғыш;

-IFIDN,IFIDNI,IFDIF және IFDIFI директивалары – жолдар символын салыстыру нәтижесіне байланысты шартты аударғыш.

Шартқа байланысты компиляция директиваларының жалпы синтаксисі келесі құрамдағы синтаксистік конструкцияда берілген:

IFxxx логикалық_өрнек_ немесе_ аргумент:
Программа_ 1_фрагменті
ELSE
Программа_ 2_фрагменті
ENDIF

14 Дәріс №14. MS DOS үшін программалау негізі

Дәріс мақсаты: MS DOS үшін программалау негізідерін оқып үйрену.

Лекция мазмұны: DOS функциялары, BIOS функциялары, пернетақтадан мәліметтерді енгізетін жүйелік құрал.

14.1 DOS функциясының негіздері

Қолданбалы программалардың жүйелік құралдарына жүгіну. Ассемблерде жазылған программа басқа тілде жазылған программалар сияқты

операцилық жүйе көмегімен орындалады. Операциялық жүйе жадыда программаға орын бөледі, оны жүктейді, оның басқаруын жібереді және программаның енгізу–шығару құрылғысымен өзара келісімді жұмыс істеуін, файлдық жүйелермен және басқа программалармен қамтамасыз етеді. DOS және BIOS функцияларына жүгіну программалық үзілу арқылы жасалады.

IBM PC типіндегі машиналардың үзу жүйесінің басқа жүйеден айырмашылығы болмайды. Жедел жадының 0000h-тан 03FFh-қа дейін бастапқы төрт байттық аймақтарының адресі үзу векторларына бөлінеді – онда үзулерді өңдеу программаларының адресі сақталады. Әрбір вектордың екі үлкен байтына ҮӨП сегменттік адресі жазылады, ал екі кіші байтқа – сегменттегі ҮӨП кіріс нүктесінің салыстырмалы адресі жазылады. Векторлар, оларға сәйкес үзулер сиқты нөмірлерден тұрады, оларды типтер деп атайды, мысалы, 0 нөміріндегі вектор 0 адрестен басталады, 1 типтегі вектор - 4 адрестен басталады , 2 типтегі вектор-8 адрестен басталады т.б. с.с. Сонымен N нөміріндегі вектор, N*4-ден м*4+3-ке дейін жады байттарында орын алады. Барлығы вектор үшін жады аймағында 256 вектор орын бөлінген.

Қолданбалы программадан жүйелік функцияға қатынау біркелкі орындалады. AH регистіріне функция нөмірі орналастырылады, ал басқа регистрлерге керекті жүйелік программаны орындауға арналған деректер орналастырылады. Бұдан кейін сандық аргументі бар INT командасы орындалады, мұнда үзу типі орналасады, мысалы INT 21h.

DOS функцияларының көбі және BIOS функцияларының көбі CF тасымал белгісінде аяқтау кодын қайтарады. Егер функция дұрыс орындалса, онда CF=0, ал қате болса CF=1. Соңғы жағдайда регистрлердің бірінде қате коды қайтарылады. Сонымен, жүйелік құралдарға қатынаудың қарапайым процедурасы келесідей түрде болады:

```
Mov AH,funk ; funk-
;регистрлерді толтыру
;берілген функцияны орындау үшін параметрлер
…
Int 21h ; MS DOS өту
Jc error ;жол брден орындады
          ; DOS қайтқасын
;
Error cmp ax,1;  аяқтау кодының анализі
        Je    err1
        Cmp ax2
Je      err2
…
BIOS функциясы да осылай шақырылады.
```

14.2 Пернетақтадан деректерді енгізу жүйелік құралдары

Операциялық жүйе пернетақтадан деректерді енгізудің бірнеше әдістерін ұсынады:

- пернетақтаға, файлға қатынағандай қатынау, DOS-тың INT 21h үзуінің 3Fh функциясы;
- DOS INT 21h-тағы 1..Сh-қа дейінгі функциялар тобын қолдану, пернетақтадан әртүрлі режимде бір символ дұрыс енгізуді орындайды;
- бір символдап енгізуді DOS арқылы емес , BIOS драйверлері арқылы орындау, INT 16h үзуінің көмегімен орындалады.

Файлдық жүйе құралдарымен пернетақтадан енгізу, файлдан оқумен бірегей орындалады.

Әдетте алдын ала анықталған, стандартты енгізу құралына бекітілген 0 дискриптор қолданылады. Енгізілетін символдар саны CX регистрінде көрсетіледі, бірақ енгізу қанша символ енгізілгенге байланысты емес <Enter> басқышы басылған кезде ғана тоқтатылады. Сондықтан символдарды енгізген кезде алдын ала ұзындығын көрсетпесе де болады. Кез келген жағдайда AX регистріне нақты енгізілген символдардың саны қайтарылады, бұл кезде екі байт<Enter> басқышының да саны есептелінеді.

Программаға пернетақтадан деректерді енгізудің екінші әдісі DOS-тың 1...Сh дейінгі функцияларының көмегімен орындалады. Енгізу үшін INT 21h үзуінің жеті функциясын қолдануға болады:

- 01h- символды бейнесімен шығару;
- 06h- тікелей енгізу-консоль арқылы шығару;
- 07h- сиволды бейнесіз енгізу және Ctrl/C тексерілмейді;
- 08h- сиволды бейнесіз енгізу және Ctrl/C тексеріледі;
- 0Ah-буфер арқылы бейнесі бар жол енгізу;
- 0Bh-стандартты құрылғы күйін тексеру;
- 0Ch- кіріс буферін тазарту және енгізу.

01h,06h,07h және08h функцилары программадағы әрбір шақыру сайын бір символды енгізеді; символдар тобын енгізу үшін функция циклі қолдану керек. Бұл функциялар символды бейнелейді немесе бейнелемейтіндігімен ерекшеленеді, сондай-ақ <Ctrl>/C басқыштарының тексерілуіне не тексерілмеуіне байланысты болады. 01h және 0Ah функциялары енгізілген символдарды экранға шығарады; 07h және 08h функциялары енгізілген символдарды экранға шығарады; 07h және 08h функцилары бұны жасамайды, бұл командалар жасырын деректерді енгізуге мүмкіндік береді. Бұл функциялардың екіншісі маңызды ерекшелігі: ол <Ctrl>/C басқыштрының тексерілуіне не тексерілмеуіне байланысты. 01h және 08h функцияларының орындалуы кезінде DOS әрбір енгізілген символды тексереді, кіріс ағымында <Ctrl>/C 03h кодын көрген кезде, программа жұмысын апатты түрде тоқтады. 06h және 07h функциялары бұл кодты <Ctrl>/C программаға оған ешқандай әрекет жасамай жібереді. Бұл әдіс қолданбалы программалармен программаны аяқтамастан бұрын қандай да бір программалық әрекет жасау керек болса

қолданылады. Мұнай <Ctrl>/C тексерілетін программаларды DOS құралдарымен орындау мүмкін емес.

0Aһ функциясы пайдаланушы буферіне пернетақтадан енгізілген жолды орналастырады; жол < Enter > басқышымен аяқталуы керек. Жол ұзындығы 254 символға дейін; жол < Enter > басқыштарымен ақталуы керек. Жол ұзындығы 254 символға дейін болуы мүмкін. Енгізілген символдар экранда бейнеленеді; <Ctrl>/C басқыштарының басылуы программа жұмысын апатты аяқтауға мәжбүр етеді.

0Bһ функцисы енгізілетін күтіліп тұрған символдардың буферінде болуын тексеруді орындайды. Егер олар бар болса, онда енгізу функцияларының бірімен оны шығаруды орындайды, егер ол болмаса, онда программа орындалуын әрі қарай жалғастырады.

0Bһ функциясы <Ctrl>/C тексереді. 0Cһ функциясы алдын ала буферді тазалап барып енгізуді орындайды.

Барлық функцилар, 0Cһ басқа, барлық бұрын жиналған символдармен қоса шығарады. 0Cһ функциясы ғана бірінші тазалап барып сосын пернетақтадан енгізуді күтеді. Нәтижесінде бұрын енгізілген басқыштардың кодтарды жоғалтады. Сонымен енгізу режимі 0Cһ функциясының ішінде қолданылған функцияға байланысты болады.

01h,07h,08h және 0Aһ функциялары синхронды болып табылады яғни буферде символ болмаған жағдайда оның енгізілуін күтеді, 06h функциясы буфер күйін анықтау үшін немесе онда код болған жағдайда, ол кодты шығарып оны өңдеу үшін, ал код болмаған жағайда программаның орындалуын жалғастыру үшін қолданылады.

01h, 06h, 07h, және 08h функциялары программаға кеңейтілген ASCII кодтарын енгізуде мүмкіндік береді. Ол үшін енгізілген ASCII код нөлге тең екендігін көрген кезде функцияны қайтадан орындау керек. Бұл қолданбалы программаларды функциональді басқыштар арқылы және <Alt>/ сан, <Alt>/ әріп және т.б қосылыстарын бірге пайдалана отырып басқаруға мүмкіндік береді.

BIOS деңгейінде пернетақтамен жұмыс жасау. Яғни енгізу буферіне түсетін екі байттық кодтарды оқу және пернетақта сөзінің белгісін анализдеу үшін қолданылады.

Енгізу үшін келесі INT 16h үзуінің функциялары қолданылады:

- 00h-буферден екі байттық кодты оқу;
- 01h-пернетақта күйін және буферден шығармай екі байттық кодты оқу;
- 02 H-пернетақта белгісін оқу.
- 00h функциясы бір әрекеттен басылған бір басқыштың немесе бірнеше басқыштардың екі байттық кодын, соның ішінде олардың кодын, және кеңейтілген ASCII кодын ала алатын мүмкіндіктері қаралған. 00h функциясы синхронды болып табылады: орындалу кезінде программа жұмысын белгілі бір басқыш басылғанға дейін күтеді.

01h функциясы синхронды үзулер жолына жатады: пернетақта күйін анықтап болғаннан кейін, ол басқаруды программаға береді. Буфер күйі ZF белгісіне қайтарылады: егер буферде программаға енгізілетін символдар болса, онда ZF=0,егер буфер болса, онда ZF=1. Буферде символ коды орналасқан болса, оны анализдеуге болады, өйткені ол функциямен AX регистіріне қатарылады. Бірақ есте сақтау керек, 01h функциясы екі байттық кодты AX регистріне көшіреді, бұл кезде буфер тазартылмайды. Бұл функция арқылы символды алып кеткеннен кейін 00h функцисымен буферді тазалаған дұрыс.

02h функциясы- пернетақта белгілерін оқу- программаға белгілер сөзінің құрамын береді. Бұл функция скен-код деңгейінде жұмыс жасайтын программалармен <Shift>,<Cops Lock> және т.б басқыштарының күйін анықтау үшін қолданылады.

15 Дәріс №15. Ассемблерде Windows қосымшаларын жасау

Дәрістің мақсаты: Windows қосымшаларын жасаудың ерекшеліктерін оқып үйрену.

Дәріс мазмұны: API функцисын пайдалану, терезелік және консольдік қосымшалар туралы түсінік, стек арқылы параметрлерді жіберу.

15.1 API функциясының негіздері

MS DOS және Windows операциялық жүйелері программалау идеолгиясының екі түрін қабылдайды. Windows қосымшалардың екі түрін қабылдайды: терезелік қосымша – API функциясының арнайы жинағында құралады және терезелік емес, және де консольдік деп аталатын, текстік режимде істейтін программа болып табылады. Консольдік режим арнайы функцилармен қамтамасыз етеді.

Windows. Программа құрылымының үш түрін бөліп айтуға болады: классикалық, диалогтік және консольді.

Windows-та программалау API функциясын пайдаланумен негізделген. Операциялық жүйенің сыртқы құрылғылар мен қорлармен өзара әрекеттестігі осы функциялар көмегімен орындалады.

Windows программасының басты эллементі терезе болып табылады. Әр терезе үшін өзінің процедурасы анықталады. Терезе басқару элементін ұстай алады: басқыштар, тізімдер, редакциялау терезесі және т.б. Бұл элементтер негізінде терезелер болып табылады, бірақ ерекше құрамымен ерекшеленеді. Бұл элементтермен болатын оқиғалар процедура терезесіне хабарламаның келуіне әкеліп соқтырады.

Windows операциялық жүйесі жадының сызықтық моделін қолданады. Басқаша айтқанда, бүкіл жадыны бір сегмент ретінде қарастыра аламыз. Бұл кез келген ұяшықтың адресін 32 биттік регистр арқылы анықталатынын айтады. Мысалы EBX регистрі.

Windows операциялық жүйесі көп міндетті болып табылады. Әр міндеттің өзінің адрестік кеңістігі және кезегі бар.

API функциясын қалай шақырамыз, содан бастайық. Кез келген API функциясын таңдайық, мысаы MessageBox:

Int message Box(HWND hwnd6LPCTSTR 1pText, LPCTSTR 1pCaption, UINT u Type).

Берілген функция экранға хабарламамен және басқышпен терезе шығарады. Параметр мағынасы: hWend –терезе дескрипторы, ол жерде хабарлама терезесі шығып тұрады, 1pText-терезеде шығып тұратын мәтін, 1pCaption- терезенің тақырыпшасындағы мәтін, u Type- терезе түрі, атап айтқанда басқыштар санын анықтайды.

Енді параметр типтері туралы. Олардың бәрі 32 биттік бүтін сандар: HWND -32 битті бүтін, LPCTSTR-32 битті жолды көрсететін нұсқауыш, UINT 32 биттік бүтін.

API функциясының бөлігі «A» суффиксін алды. Функциялардың екі прототипі бар: «A» суффиксімен – ANSIді қолдайды, ал «W» суффиксімен Unicode. Сондықтан функция атына суффикс «A» қосылады, одан басқа MASM ды қолданғанда атының соңына @16 қосу керек. Осылайша шақырылған функция былай көрініс табады: CALL MessageBox@16. Ал параметрлерді не істейміз? Оларды ақырындап стекке енгіземіз. Бұл ережені есте сақтаңыз: СОЛ ЖАҚТАН ОҢ ЖАҚҚА-ТӨМЕННЕН ЖОҒАРЫ. Сөйтіп дескриптор HW адресі бойынша орналассын, ал жолдар мына адрес бойынша STR1және STR2, ал хабарлама терезенің орны константа. Ең қарапайым түрі 0 белгісін алады және MB_OK деп аталады. Келесі енгіземіз:

```
MB_OK equ 0
STR1 DB”неверный код”,0
HW    DWORD?
…
PUSH    MB_OK
PUSH  OFFSET STR1
PUSH  OFFSET STR2
PUSH   HW
CALL ,MessageBox@16
```

Кез келген функцияның нәтижесі бүтін сан. Ол EAX регистірінде қайтарылады.

Аналогтық түрде ассембледе Си құрылымдары оңай ашалады. Мысалға жүйелік хабарламаны анықтайтын құрылымды қарастырайық:

```
Typedef struct tagMSG{//msq
HWND hwnd;UINT message; WPARAM wParam; LPARAM  lParam;
DWORD time; POINT pt;}MSG.
```

Ассемблерде бұл құрылым мынадай көрініс табады:

```
MSGSTRUCT STRUC
MSHWND DD?
MSMESSAGE DD?
```

MSWPARAM DD?
MSLPARAM DD?
MSTIME DD?
MSPT DD?
MSGSTRUCT ENDS

Енді бүкіл программаның құрылымына жолығамыз. Windowsқа арналған классикалық құрылымдағы программаны қарастырайық. Ондай программада басты терезе болады, соған байланысты басты терезенің процедурасы да болады. Жалпы программа кодында келесі бөлімдерді бөліп айта аламыз:

а) терезе класының тіркелуі.

б) басты терезені құру.

в) хабарлама кезегін өңдеу циклі.

г) басты терезе процедурасы.

Әрине программада басқа да бөлімдер бола алады, бірақ берілген бөлімдер программаның негізгі қаңқасын құрайды. Бұл бөліктер тәртіп бойынша шешіп алайық.

Терезенің кластарының тіркелуі.

Терезенің кластарының тіркелуі Registre Class A функциясының көмегімен жасалады, бірыңғай параметрі WNDCLASS нұсқауышы болып табылады, ол терезе туралы ақпаратты қамтиды.

Тіркелген кластар негізінде Create WindowsExA функциясының көмегімен терезенің даналарын жасауға болады.

Хабарлама кезегін өңдеу циклі.

Си тілінде бұл цикл былай жазылады:

```
While(GetMessage(&msq,NULL,0,0))
{
//пернетақтаны қолдануға рұсат беру,
//трансляция жолымен виртуалды пернетақталар туралы хабарлау,
// хабарламаларға алфавитті-сандар пернетақталары жайлы
Translate Message(&msq);
//Windows басқаруын қайтару
DispatchMessage(&msg);
}
```

GetMesage() функциясы кезекті хабарламаны «ұстайды» берілген қосымшаның ішіндегі қатар хабарламаның ішінен және оның құрылымын MSG енгізеді.

TranslateMessage функциясына келетін болсақ, оның құзіреті WM_KEYDOWN және WM_KEYUP хабарларына қатысты, олар WM_CHAR және WM_DEDCHAR ға таратылады, және WM_SYSKEYDOWN және WM_SYSKEYUP - ға, олар WM_SYSDEADCHAR айналады. Таратудың мәні ауысуда, алфавиті-сандық басқышты басқанда терезеге бірінші WM_KEYDOWN хабарлама келеді, сосын WM_KEYUP, одан кейін WM_CHAR.

Күту циклінен шығу тек GetMessage функциясы 0-ді қайтарса ғана орыналады. Ол тек шығу туралы хабарлама келсе ғана орындалады. Осылайша, күту циклі екі түрлі мән береді: анықталған бейне бойынша терезеге арналған хабарлама жасалады және программадан шығу хабарламасы күтіледі.

Басты терезе процедурасы.

Си тіліндегі басты терезе функциясы:

LRESULT CALLBACK WindowFunc (HWND hwnd, UNIT message, WPARAM wParam,

PARAM lParam)

Функциямен қайтарылатын мән типі бізге қажет болмайды.

Параметрлерді қарастырайық: hwnd-терезе индификаторы. Message-хабар идентификаторы. WPARAM және lParam –хабар мағынасын дәлелдейтін параметрлер. Барлық төрт параметрдің DWORD типі бар.

Енді ассемблер тілінде осы функциясының «қаңқасын» қарастырайық.

```
PROC
PUSH EBP
MOV EBP,ESP
PUSH EBX
PUSH ESI
PUSH EDI
PUSH DWORD PTR[EBP+14H]    ;LPARAM(lParam)
PUSH DWORD PTR[EBP+10H]    ;WPARAM (wParam)
PUSH DWORD PTR[EBP+0H]     ;MES (message)
PUSH DWORD PTR[EBP+08H]    ;HWND(hwnd)
POP EDI
POP ES
POP EBX
POP EBP
RET 16
WNDPROC ENDP
```

Үзіндіні қарастырайық. RET 16 - стекінің төрт параметрден босатылуынан шығу (16=4*4).

Параметрлерге қатынас құру EBP регистрі арқылы жүзеге асырылады:

```
DWORD PTR[EBP+14H]      ;LAPARAM(lParam)
DWORD PTR[EBP+10H]      ;WPARAM(Wparam)
DWORD PTR[EBP+0CH]      ;MES(message) – хабар шарттаңбасы
DWORD PTR[EBP+08H]      ;HWND(hwnd) - терезе дескрипторы
```

DefwindowProc функциясы терезе функциясында өңделмейтін хабарлар үшін шақырылады.

15.2 Ассемблердің негізгі модулінің құрылымы

Ассемблерде модуль құрылымы Windows қосымшасының программалауында EXE форматта жүреді. Файл келесі түрде бола алады:

```
.386  ;    .386p немсе үлкендеу
.MODEL Flat,STDCALL
Include win32.inc                ;Windows-кітапхана
…….
Includelib import3.lib           ; Windows –кітапхана
.DATA
            ;мәліметтер
.CONST
             ;константа
……
.CODE
Енгізу нүктесі:
        ;код
.....
End енгізу нүктесі
```

Берілген құрылымда .MODEL Flat, STDCALL- бұл директива жазық модельмен қамтамасыз етеді, ол Win32 платформасы үшін қолданылады. Win32 функцияны шақыру түрін ғана қолданады STDCALL, ол паскаль мен Си қосындысынан шыққан параметрлер тәртіп бойынша оңнан солға қарай беріледі, C/C++ тілінде сиқты, жұмыстың соңында процедура стекті өзі тазарту керек, Паскаль тілінде сияқты.

Windows-5s аз ғана өзгешеліктерді мысалда қарастырайық- ассемблерде программалау. Қарапайым терезелік қосымшаларды құрастыру, ол экранға «мен Windows программалауды үйренемі» деген мәтінді экранға шығару керек.

```
.386
.model flat,stdcall
Include win32.inc                ; Windows- макрокітапхана
Includelib import32.lib  ;Windows- макрокітапхана
.data
Mesbox_text      db ‘мен Windows программалауды үйренемі’,0
Mesbox_title   db ‘терезе тақырыпшасы Messagebox’,0
.code
        Start:
;Стекке параметрлерді жүктеу(керсінше)
Push 0  ;терезе стилі (UNIT u Type)-үнсіз келісім бойынша
Push offset mesbox_title   ;терезе тақырыпшасының аты(LPCTSTR lpCaptio)
Push offset mesbox_text  ;шығарылатын текст адресі(LPCTSTR lpText)
```

```
Push 0; HWND h Wnd=0-терезенің иесі жоқ
; MessageBox функциясын  шақыру
Call  MessageBox
Push 0; барлық ағымдағы кодтар үшін (UNIT uExitCode)
Call ExitProcess
Ends
End start
```

Енді терезе қосымшасының EXE –файлын жасап оны іске қосамыз:

```
EXE32win.batMESSBOX.ASM MESSBOX.exe
```

Әдебиеттер тізімі

1 Пильщиков В.Н. Assembler. Программирование на языке ассемблера IBM PC. -М.: Диалог-МИФИ, 2005. - 288 с.

2 Галисеев Г.В. Ассемблер для Win 32. Самоучитель. - К.:Диалектика, 2007.- 368 с.

3 Рудольф Марек. Ассемблер на примерах. - М.: Наука и техника, 2005. - 240 с.

4 Калашников О.А. Ассемблер - это просто. Учимся программировать - Санкт-Петербург: БХВ-Петербург, 2011. - 336 с.

5 Джесси Рассел. Ассемблер. - М.: Книга по Требованию, 2012. - 98 с.

6 Михаэль Хофманн . Микроконтроллеры для начинающих (+ CD-ROM) - Санкт-Петербург: БХВ-Петербург, 2010. - 304 с.

7 Иванов В.Б. Программирование микроконтроллеров для начинающих. Визуальное проектирование, язык С, ассемблер (+ CD-ROM). - Санкт-Петербург: Корона-Век, МК-Пресс, 2010. - 176 с.

8 Андрей Жуков, Андрей Авдюхин. Самоучитель Ассемблер. - Санкт – Петербург:БХВ-Петербург, 2002. - 448 с.

9 Голубь Н.Г. Исскуство программирования на Ассемблере. – М.: ООО «ДиаСофтЮП», 2005. – 832 с.

10 Пирогов В.Ю. Ассемблер для Windows. - Санкт – Петербург: БХВ-Петербург, 2005.