“Basics of microcontroller”

Lecture 4.

Stack register, external and internal data bus, address bus

Lecturer: Onon Otgonbaatar, MD.

Ассемблер хэлний командуудын ангилал

- Удирдлага шилжүүлэх командууд: JMP, Jnnn, LOOP, CALL.
- Өгөгдөл шилжүүлэх командууд: MOV, XCHG, PUSH, POP.
- Логик командууд: AND, OR, XOR, TEST, NOT.
- Арифметик командууд: ADD, SUB, MUL, DIV болон бусад.
  INC регистр/ санах ойн утгыг 1-ээр нэмэгдүүлнэ.
  DEC регистр/ санах ойн утгыг 1-ээр хорогдуулна.

Удирдлага шилжүүлэх командууд

Удирдлага шилжүүлэхдээ
- Шийдвэр гарган шилжүүлэх
- Давталтаар шилжүүлэх гэсэн 2 арга хэрэглэнэ.

Программын удирдлагыг шилжүүлэх 4 төрлийн командууд байна.
1. Нөхцөлт бус үсрэлт: JMP
2. Нөхцөлт үсрэлт: CMP, JE, JNE, JS, JNS/JL, JB, JBE, JA, JAE, JC, JP, JZ, JO, JNC, JNP, JNZ
3. Давталт: LOOP, LOOPZ, LOOPNZ (CX register) +127..-128
4. Дэд программыг дуудах: CALL

Удирдлага шилжүүлэлтэд хэрэглэгддэг 3 янзын хаяг байна: SHORT, NEAR, FAR
SHORT гэсэн хаяг нь давталт, нөхцөлт үсрэлт, зарим нөхцөлт бус үсрэлт зэрэгт хэрэглэгдэнэ. -128 ба +127 байтуудын доторх JMP үйлдэл нь SHORT JUMP буюу богино үсрэлт болно. Ассемблер нь 00-оос FF хооронд 1 байт операндыг үүсгэдэг.
NEAR ба FAR гэсэн хаягууд нь CALL команд, short гэж тодорхойлдоггүй нөхцөлт бус үсрэлтийн командуудтай хэрэглэгддэг. Бүх төрлүүд нь IP регистрт нөлөөлөх ба far нь мөн CS регистрт нөлөөлнө. Харин FAR буюу холын үсрэлтийн үед ассемблер нь 2 байт операндыг үүсгэдэг.

- JMP нь нөхцөл шалгахгүйгээр бүх л тохиолдолд удирдлагыг шилжүүлдэг команд юм. Жишээ нь:

```
PAGE 60,132
TITLE EXJUMP (COM) Illustration of jmp for looping
.code
        org 100h

main    proc    near    ; машины код ба тайлбарыг командын ард бичив.
        mov     AX, 01; B8 00 01      AX-т нэг гэсэн анхны утга олгоно
        mov     BX, 01; BB 00 01      BX-т нэг гэсэн анхны утга олгоно
        mov     CX, 01; B9 00 01      CX-т нэг гэсэн анхны утга олгоно
A20:
        add     AX, 01;05 00 01       AX-дээр 1 –ийг нэмнэ
        add     BX, AX          ;03 D8 BX-дээр AX –ийг нэмнэ
```

```
        shl     CX, 2           ;D1 E1 CX-ийг 2 оор үржинэ
        jmp     A20             ;EB F7 A20 гэсэн label уруу үсэрнэ
main    endp
        end     main
```

Энэ программын үр дүнд:
AX: 1, 2, 3 гэх мэтээр утга нь 1-ээр нэмэгдэж явна
BX: 1, 3, 6, 10 гэх мэтээр утга нь нэмэгдэж явна
CX: 1, 2, 4, 8 гэх мэтээр утга нь 2-ээр үржигдэж явна

Энд хийгдэж байгаа давталтын эхлэл нь A20 гэсэн label, төгсгөл нь jmp A20 гэсэн команд байна. Энэ давталтаас гарах нөхцөл байхгүй тул төгсгөлгүй давталт болох нь харагдаж байна. A20 –ийн хаяг нь add AX, 01 командын эхний байтыг заана. Энэ давталт нь near гэсэн атрибутыг авна. A20 гэсэн хаяг нь JMP командаас -9 байтын хаягийн зөрүүтэй. JMP командын машины код нь EB бөгөөд энэ команд нь F7 гэсэн утгыг IP(командын заагч) дээр нэмнэ. Өөрөөр хэлбэл, JMP командын offset хаяг нь 0112 байсан бол үсрэлт хийсний дараа offset хагя нь 109 болно, энэ нь A20 –ийн offset хаяг болно.

	Decimal	Hex
IP:	274	112
JMP offset:	-9	F7 (2-тын комплимент)
Шилжсэн хаяг:	265	109

- LOOP команд нь мөн давталтыг үүсгэх ба CX регистрт анхны утга олгон хэрэглэнэ. Давталт бүрт CX-ийн утга нэгээр хорогдох ба CX-ийн утга 0 болоогүй бол үсрэнэ, 0 болсон бол доорх команд уруугаа шилжинэ. Жишээ нь:

```
PAGE 60,132
TITLE EXLOOP (COM) Illustration of loop
.code

        org 100h
main    proc    near    ; машины код ба тайлбарыг командын ард бичив.
        mov     AX, 01; AX-т нэг гэсэн анхны утга олгоно
        mov     BX, 01;BX-т нэг гэсэн анхны утга олгоно
        mov     CX, 10;CX-т 10 гэсэн давталтын утга олгоно
        mov     DX, 01;DX-т нэг гэсэн анхны утга олгоно
A20:
        inc     AX              ;AX-дээр 1 –ийг нэмнэ
        add     BX, AX          ;BX-дээр AX –ийг нэмнэ
        shl     DX, 1           ;DX-ийг хоёроор үржинэ
        loop    A20             ;CX-ийг нэгээр хорогдуулаад,
                                ;0 биш бол A20 гэсэн label уруу үсэрнэ
        ret                     ;Программыг зогсооно
main    endp
        end     main
```

Энэ программ нь нэг ижил үйлдлүүдийг 10 удаа давтах ба үүний дараа программыг зогсооно. LOOP –ийн хувьд үсрэлтийн хязгаар нь –128-аас +127-гийн хооронд байна. Энэ программын төгсгөлд AX=000BH, BX=0042H, CX=0, DX=0400H гэсэн утгуудтай байна.

LOOP командын өөр 2 хувилбар нь LOOPE (эсвэл LOOPZ) ба LOOPNE (эсвэл LOOPNZ) юм.
LOOPE нь хэрэв CX нь 0 биш, Zero флаг нь 1 үед үсрэнэ. LOOPNE нь CX нь 0 биш, Zero флаг нь 0 үед үсрэнэ.

- Нөхцөлт үсрэлт

LOOP командыг нөхцөлт үсрэлтээр орлуулбал:

```
LOOP A20     гэдэг нь      DEC CX
                           JNZ   A20 гэсэнтэй ижил юм.
```

CX регистрийг нэгээр хорогдуулаад хэрэв 0 биш бол A20 уруу үсэрнэ гэсэн үг.

Signed ба Unsigned өгөгдөлүүдийн ялгаа

Нөхцөлт үсрэлт нь signed ба unsigned өгөгдлүүдэд ялгаатай хэрэглэгддэг.
Unsigned өгөгдөл нь бүх битүүдийг өгөгдлийн бит гэж ойлгоно.
Signed өгөгдөл нь хамгийн зүүн талын нэг битийг тэмдэг гэж авна, үлдсэн битүүдийг өгөгдлийн бит гэж ойлгоно. Эерэг буюу сөрөг өгөгдлүүдийг тэр битээр нь ялгана. Жишээ нь:
AX регистрт 1100 0110, BX регистрт 0001 0110 гэсэн утгууд байвал CMP AX, BX гэсэн үйлдэл нь AX ба BX регистрүүдийг харьцуулна. Unsigned өгөгдлийн хувьд AX регистрийн утга илүү их, харин signed өгөгдлийн хувьд BX регистрийн утга илүү их болно.

Signed byte
0000 0000–аас 0111 1111 хүртэл 128 ширхэг эерэг тоонууд байх ба харин 1000 0000-ээс 1111 1111 хүртэл 127 ширхэг сөрөг тоонууд байна.
Unsigned byte
0000 0000-оос 1111 1111 хүртэл 512 ширхэг тоо байна.

- Нөхцөлт үсрэлтүүд гэдэг нь өмнөх үйлдлийнхээ үр дүн дээр үүсэх төлөвийн битүүдийг шалган, нөхцөл нь биелэгдсэн үед заасан хаяг уруу үсэрнэ. Эдгээрийг дараах байдлаар 2 ангилж болно.

- Unsigned ба signed өгөгдлүүд дээр хийгдэх үсрэлт

Команд	Тайлбар	Шалгах флагууд
JE / JZ	тэнцүү буюу үр дүн 0	ZF
JNE / JNZ	тэнцүү биш буюу үр дүн 0 биш	ZF
JS	сөрөг үр дүн гарвал	SF
JNS	эерэг үр дүн гарвал	SF
JC	carry үүсвэл	CF
JNC	carry үүсээгүй бол	CF
JO	overflow үүсвэл	OF
JNO	overflow үүсээгүй бол	OF
JP / JPE	(parity) сондгой бол	PF
JNP / JP	(parity) тэгш бол	PF
J**CX**Z	CX регистрийн утга 0 эсэх	ZF

- Unsigned өгөгдөл дээр хийгдэх үсрэлт

Команд	Тайлбар	Шалгах флагууд
JA / JNBA	илүү буюу тэнцүү/бага биш	CF, ZF
JAE / JNB	илүү/тэнцүү буюу бага биш	CF
JB / JNEA	бага буюу илүү/тэнцүү биш	CF
JBE / JNA	бага/тэнцүү буюу илүү биш	CF, AF

- Signed өгөгдөл дээр хийгдэх үсрэлт

Команд	Тайлбар	Шалгах флагууд
JG / JNLE	илүү буюу тэнцүү/бага биш	ZF, SF, OF
JGE / JNL	илүү/тэнцүү буюу бага биш	SF, OF
JL / JNGE	бага буюу илүү/тэнцүү биш	SF, OF
JLE / JNG	бага/тэнцүү буюу илүү биш	ZF, SF, OF

- CALL функц ба процедурууд

Энэ команд нь функц ба процедуруудыг дуудаж ажиллуулахад хэрэглэгдэнэ. CALL командаар дуудагдсан функц ба процедуруудаас буцахдаа RET командыг ашиглана. Уг команд нь SP регистрийн утгыг 2-оор хорогдуулаад, IP регистр дахь биелэгдэх дараагийн командын хаягийг стек санах ойд хийнэ. Дараа нь IP регистрт дуудаж байгаа программынхаа шилжлэг хаягийг ачаална. Дуудагдах процедур нь нэг сегмент дотор болон гадна байгаагаас хамааран 4 төрлийн CALL команд байна. Нэг сегментээс гаднах дуудалт нь SP регистрийн утгыг мөн 2-оор хорогдуулан, CS регистрийн утгыг стек санах ойд хийнэ. Дараа нь CP регистрт дуудагдах сегментийн хаягийг ачаална.

Код сегмент нь FAR гэж дуудагддаг нэг л процедурыг агуулна.
FAR операнд нь системд энэ хаяг бол программыг эхлүүлэх хэсэг гэдгийг хэлнэ. Код сегментэд PROC, ENDP гэсэн бичиглэлээр ялгаатай хэдэн ч процедурууд байж болно.

```
.code
;---------- Үндсэн процедур ----------
Begin       proc    far
            …
            …
            call    B10
            call    C10
            ret
Begin       endp
;---------- B10 нэртэй процедур ----------
B10         proc    near
            …
            ret
B10         endp
;---------- C10 нэртэй процедур ----------
C10         proc    near
            …
            ret
C10         endp
            end     Begin
```

B10 ба C10 зэргийг зарлах Proc үйлдэл нь near гэсэн операндыг агуулсан нь энэ процедурууд өмнө зарлагдсан код сегментэд байгааг илэрхийлнэ. Энэ near операндыг бичихгүй орхиж болно, тэгвэл автоматаар near гэж авна.

Begin процедур нь удирдлага шилжүүлэх команд болох 2 ширхэг CALL командыг агуулж байна. Энэ командуудыг биелүүлэхдээ программын удирдлагыг B10, C10 гэсэн дэд процедурууд уруу шилжүүлэх ба RET гэсэн команд дээр очоод эргэж CALL командлын дараагийн команд уруу орон үргэлжилнэ.

RET команд нь үргэлж анх дуудсан функц үрүү буцаах ба Begin процедур B10 ба C10 дэд процедуруудыг дуудсан учир RET үйлдлээр программын удирдлага Begin процедур урууга буцна. Программыг биелүүлж эхлэхийн өмнө DOS нь Begin процедурыг ажиллуулсан ба Begin –ий RET үйлдэл нь DOS уруу удирдлагыг буцаана.

Ингэж процедуруудыг ашиглах нь программыг хоорондоо хамааралтай логикоор зохион байгуулж, ойлгомжтой бичихэд хэрэгтэй юм.

STACK СЕГМЕНТ

Push үйлдэл хийн стек ойд 1 байт хаяг буюу утга хийдэг. Pop үйлдэл нь яг өмнө нь push хийсэн утгыг буцааж авдаг. Стек санах ой нь LIFO (last input first output) зарчимаар буюу сүүлд орсон өгөгдөл нь эхэлж гардаг санах ой юм. Push, pop үйлдлүүд нь 2-уулаа SP (стекийн заагч) регистрийн агуулж буй offset хаягийг өөрчилнө. Стек ой нь push хийж байгаа утгуудийг багтаахаар хангалттай зайтай байх хэрэгтэй. 32 word хэмжээтэйгээр стек ойг зарлах нь голдуу хүрэлцдэг.

PUSH, PUSHF, CALL, INT, INTO гэсэн командууд нь буцах хаяг, флаг регистрийн утга зэргийг стекд хадгалдаг.
POP, POPF, RET, IRET гэсэн командууд нь буцах хаяг, флаг регистрийн утга зэргийг стекээс гаргаж авна.

Exe программын эхэнд систем регистрүүдэд доорх утгуудыг хийнэ.
DS ба ES: санах ойд программ эхлэхийн өмнө PSP гэсэн 256 байт (100h) хэсгийн хаяг
CS: Эхний биелэгдэх командын хаяг
IP: 0 утга
SS: Стек сегментийн хаяг
SP: Стекийн оройн offset хаяг
Жишээ нь:
Стекийг 32 words буюу 64 байтын хэмжээтэй тодорхойлбол:
DW 32 DUP(?)
гэж зарлах ба SP нь 64 (40h) гэсэн утгыг авна.
Push команд нь SP-ийн утгыг 2-оор хорогдуулан, утгаа стекийн оройд хийнэ. Жишээ нь: push ax
Call команд нь SP-ийн утгыг 1-ээр хорогдуулан, дуудаж байгаа дэд процедурынхаа offset хаягийг стекийн оройд хийнэ. Жишээ нь: call b10
Ret команд нь стекийн оройгоос хаягийг авч IP регистрт хийгээд, SP-ийн утгыг 2-оор нэмэгдүүлнэ.

АЧААЛАХ ҮЙЛДЛҮҮД

Бид өмнө шууд утгыг регистрт, санах ойгоос өгөгдлийг регистрт, регистрийн утгыг санах ойд, регистрээс регистрт утга ачаалах гэх мэт үйлдлүүдийг үзсэн. Эдгээрт MOV командыг ашиглаж байсан бөгөөд өгөгдөл нь 1 буюу 2 байт урттай байсан. Харин санах ойгоос санах ойд утга шилжүүлэлт хийдэггүй.

Одоо 2 байтаас урт мэдээллийг зөөх ба мөн санах ойгоос санах ойд утга шилжүүлэхийг үзье. Энэ зорилгод бид LEA гэсэн командыг санах ой дахь хувьсагчийн хаягийг авахад ашиглана.

```
Page    65, 132
Title   Exmove (exe) Extended move operations
.model small
.stack  100h
.data
        name1 db    ‘ABCDEFGHI’
        name2 db    ‘JKLMNOPQR’
        name3 db    ‘STUVWXYZ*’
.code
BEGIN       proc    far
            mov     ax,@data
            mov     es,ax
```

```
                mov     ds,ax

                call    b10move         ; дэд процедурыг дуудна.
                call    c10move         ; дэд процедурыг дуудна.
                Ret                     ; процедурыг дуусгана.
BEGIN           endp

; нөхцөлт үсрэлтийг ашиглан, санах ойгоос санах ойд хуулах
b10move         proc
                lea     si, name1       ; name1-ийн эхлэх хаягийг SI регистрт
                lea     di, name2       ; name2-ийн эхлэх хаягийг DI регистрт
                                        ;                                   олгоно.
                mov     cx, 09
b20:
                mov     al, [si]        ; name1-ийн утгыг AL регистрт авна.
                mov     [di], al        ; name2-ийн үүрэнд AL регистрийн
                                        ;       утгыг хийнэ.
                inc     si              ; хаягуудыг
                inc     di              ;       нэмэгдүүлнэ.
                dec     cx              ; тоолуурын утгыг хорогдуулна.
                jnz     b20             ; ажиллагаа дууссан уу?
                ret                     ; тийм бол процедураас гарна.
b10move         endp

; давталтыг ашиглан, санах ойгоос санах ойд хуулах
c10move         proc
                lea     si, name2
                lea     di, name3
c20:
                mov     al, [si]
                mov     [di], al
                inc     si
                inc     di
                loop    c20             ; cx регистрийн утгыг хорогдуулан шалгана.
                ret                     ; 0 болсон бол процедураас гарна.
c10move         endp
                end     BEGIN
```

Энэ программд өгөгдлийн сегмент нь 3 ширхэг 9 байтийн name1, name2, name3 гэсэн талбаруудыг агуулна. Программын зорилго бол name1-ийг name2 уруу, name2-ыг name3 уруу ачаалах юм. Эдгээр талбарууд нь 9 байтын өргөн учир энгийн MOV командаас илүү зүйл хэрэгтэй болно.

Программд BEGIN процедур нь сегмент регистрүүдийн анхны утгыг олгосны дараа b10move, c10move гэсэн дэд процедуруудыг дуудсан. b10move нь name1-ийн утгуудыг name2 уруу, c10move нь name2-ын утгуудыг name3 уруу ачаалах юм.

Тухайн агшинд нэг л байтыг хуулах тул name1-ийн хамгийн зүүн талын 1 дэх байтаас эхлэн, 2 дахь, 3 дахь байт гэх мэтээр доорх байдлаар хуулна.

```
Name1 A     B     C     D     E     F     G     H     I
            |     |     |     |     |     |     |     |     |
Name2 J     K     L     M     N     O     P     Q     R
```

9 удаа хуулах тул CX регистрт 9 гэсэн анхны утга олгоод, мөн SI ба DI индекс регистрүүдийг хэрэглэсэн.

lea si, name1 энэ команд нь si регистрт name1-ийн offset (шилжлэг) хаягийг ачаална.
lea di, name2 энэ команд нь di регистрт name1-ийн offset (шилжлэг) хаягийг ачаална.
mov al, [si] нь al регистрт si шилжлэг хаяган доторх name1-ийн утгыг авч хийнэ.
mov [di], al нь al регистр доторх утгыг name2-ийн di шилжлэг хаяганд хийнэ.
Inc si, inc di зэрэг нь si, di-ийн утгыг нэгээр нэмэгдүүлнэ.
Dec cx, jnz b20 ба loop c20 гэсэн хэсгүүд нь cx регистрийн утга 0 эсэхийг шалгана. 0 бол буцаах команд уруу, 0 биш бол дахин давталтыг эхэлнэ.
Si, di зэргийг нэгээр нмэхэд дараагийн хаяг нь name1+1, name2+1 болно. Иймм маягаар name1+8, name2+8 болтол үргэлжилнэ.

Санал болгох нь:
Энэ программыг компьютерт бичээд, боловсруулаад (assemble), хөрвүүлээд (link), DEBUG программыг ашиглан алхамаар ажиллуулж үз. Регистрүүдийн өөрчлөлт, командын заагч, стек зэргийг ажигла. Turbo Assembler-ийн Debug программ уруу доорх байдлаар орж, name2 ба name3-ын өөрчлөлтийг хар.
td Exmove.exe гэж ороод ажиллуулна. ALT/V товчлууруудаар View цэсэнд орж Dump үйлдлийг сонгоход, санах ойн өгөгдлийн сегментийн утгууд харагдана.

Textbook

1. Peter Abel, “IBM PC Assembler language and programming”, USA, 1987
2. Jim Mischel, "Macro magic with Turbo Assember ”, USA, 1993
3. William C.Runnion, “Structured programming in Assembly Language for the IBM PC”, Boston, 1988
4. Thomas A.Wadlow, “Memory resident programming on the IBM PC”, USA, 1987
5. Robert S.Lai, “Writing MS-DOS device drivers”, USA, 1987
6. E. Majigsuren, “IBM assembly language”, MGL, 2003
7. Muhammad Ali Mazidi, Janice Gillispie Mazidi, Rolin D. McKinlay, “The 8051 microcontroller and Embedded Systems Using Assembly and C”,USA, 2007