BIL 362 Mikroişlemciler

M.Ali Akcayol
Gazi Üniversitesi
Bilgisayar Mühendisliği Bölümü

Konular

Veri Aktarım (Data Movement) Komutları

- MOV Komutu
- PUSH / POP Komutları
- LOAD Komutları
- String Data Transfer Komutları
- Diğer Data Transfer Komutları
- Segment Override Prefix
- Assembler Komutları
- Hafıza Organizasyonu

MOV

- MOV komutu register-register veya register-memory arasında 8086-80286 işlemciler için byte veya word 80386 ve üstü işlemciler için byte, word veya doubleword data transfer eder.
- **MOV CX,DX** komutu word (16-bit) boyutundaki DX içeriğini CX register'ına aktarır.
- 80386 ve üstü işlemcilerde doubleword (32-bit) boyutundaki veri aktarılabilir.
- **MOV ECX,EDX** komutu doubleword EDX içeriğini ECX register'ına aktarır.
- **MOV AL,22H** komutu byte boyutundaki 22H değerini AL register'ına aktarır.

MOV

- 80386 ve üstü işlemcilerde doubleword (32-bit) boyutundaki data aktarılabilir.
- **MOV EBX,12345678H** komutu doubleword 12345678H değerini EBX register'ına kopyalar.
- **MOV CX,LIST** komutu hafızada LIST adresindeki word boyutundaki içeriği CX register'ına aktarır.
- **MOV AX,[BX]** komutu data segment içerisinde BX offset adresindeki bir word datayı AX register'ına aktarır.

PUSH / POP

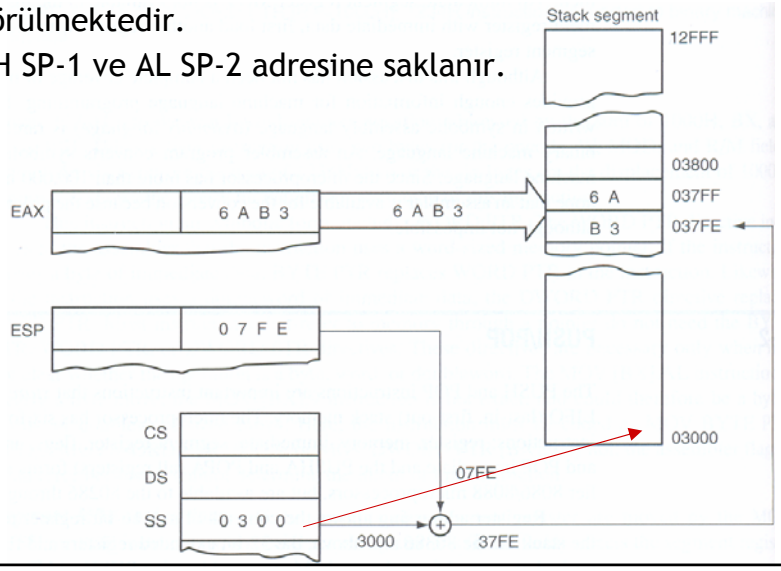
- PUSH ve POP komutları LIFO yapısındaki stack memory'den bilgi almak veya bilgi depolamak için kullanılırlar.
- Mikroişlemci 6 farklı PUSH ve POP yapısına sahiptir: **register, memory, immediate, segment register, flags ve all registers**.
- **Register** adresleme herhangi bir register içeriğinin stack'a saklanması veya stack'tan alınmasını sağlar.
- **Memory** adresleme bir memory alanının stack'a saklanması veya stack'tan alınmasını sağlar.
- **Immediate** adresleme sabit bir verinin stack'a saklanmasını sağlar.
- **Segment register** adresleme bir segment register'ın stack'a saklanması veya stack'tan alınmasını sağlar. CS PUSH edilebilir ancak CS'ye POP edilemez.
- **Flags** adreslemede flag'lar stack'a saklanabilir veya stack'tan alınabilir.
- **All register** adreslemede tüm register içerikleri PUSH veya POP edilebilir.

PUSH

- 8086-80286 mikroişlemcilerde PUSH komutu 16-bit data'yı stack'a saklar veya stack'tan alır. 80386 ve üstü işlemcilerde 2 veya 4 byte data stack'a saklanır.
- PUSHA komutu segment register'ları hariç diğer tüm register'ları stack'a saklar.
- PUSHA (push all) komutu register'ları AX,CX,DX,BX,SP,BP,SI ve DI sırasında stack'a saklar.
- PUSHF (push flags) komutu flag register'ların içeriğini stack'a saklar.
- PUSHAD ve POPAD komutları 32-bit register'ların tümünü stack'a push veya stack'tan pop eder.
- 16-bit push işlemi yapıldığında soldaki 8-bit SP-1 ve sağdaki 8-bit SP-2 adresine yazılır.

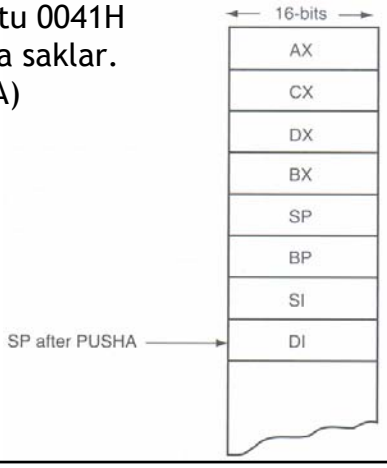
PUSH

- PUSH AX komutunun çalışmasından sonraki durum aşağıda görülmektedir.
- AH SP-1 ve AL SP-2 adresine saklanır.



PUSH

- PUSHA komutu 16-bit tüm register'ları stack'a saklar.
- Sıralama AX,CX,DX,BX,SP,BP,SI ve DI şeklindedir.
- PUSHAD komutu 32-bit tüm register'ları stack'a saklar.
- PUSH 'A' komutu 0041H değerini stack'a saklar.
  0041H = ASCII(A)



PUSH

- Aşağıda PUSH komutları görülmektedir.

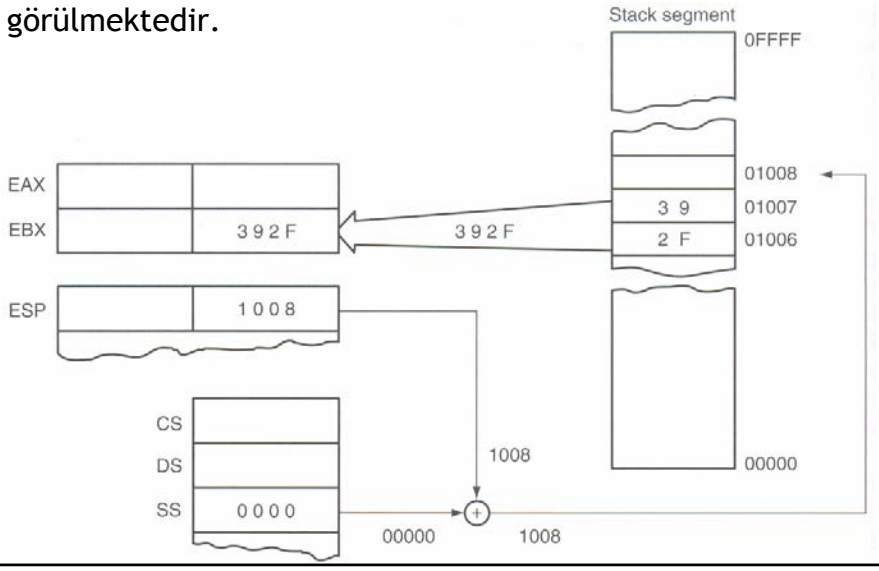
Symbolic	*Example*	*Note*
PUSH reg16	PUSH BX	16-bit register
PUSH reg32	PUSH EDX	32-bit register
PUSH mem16	PUSH WORD PTR[BX]	16-bit pointer
PUSH mem32	PUSH DWORD PTR[EBX]	32-bit pointer
PUSH seg	PUSH DS	Segment register
PUSH imm8	PUSH 'R'	8-bit immediate
PUSH imm16	PUSH 1000H	16-bit immediate
PUSHD imm32	PUSHD 20	32-bit immediate
PUSHA	PUSHA	Save all 16-bit registers
PUSHAD	PUSHAD	Save all 32-bit registers
PUSHF	PUSHF	Save flags
PUSHFD	PUSHFD	Save EFLAGS

POP

- POP komutu stack'tan veri alır ve 16-bit register'a, segment register'a veya 16-bit hafıza alanına saklar.
- 80386 ve üstü işlemcilerde 32-bit data aktarımı yapılabilir.
- POPF komutu stack'tan 16-bit alır ve flag'a aktarır.
- POPFD komutu stack'tan 32-bit alır ve extended flag'a aktarır.
- POPA komutu 16 byte datayı stack'tan alır sırasıyla DI,SI,BP,SP,BX,DX,CX ve AX register'larına aktarır.
- POPAD komutu 80386 ve üstü işlemcilerde 32-bit register'lara stack'tan değer alır.
- POP BX komutu stack'tan SP adresini BL'ye ve SP+1 adresini BH'ye aktarır.

POP

- Aşağıdaki şekilde POP BX komutu çalıştıktan sonraki durum görülmektedir.



POP

- Aşağıda POP komutları görülmektedir.

Symbolic	*Example*	*Note*
POP reg16	POP CX	16-bit register
POP reg32	POP EBP	32-bit register
POP mem16	POP WORD PTR[BX+1]	16-bit pointer
POP mem32	POP DATA3	32-bit memory address
POP seg	POP FS	Segment register
POPA	POPA	Pops all 16-bit registers
POPAD	POPAD	Pops all 32-bit registers
POPF	POPF	Pop flags
POPFD	POPFD	Pop EFLAGS

LOAD

- Herhangi bir hafıza alanını veya bir operand'ın offset adresini bir register'a yüklemek için kullanılır.
- LEA komutu 16-bit register'a operand'ın offset adresini yükler.
- LDS, LES 16-bit DS ve ES register'larına bir hafıza alanını yükler.
- 80386 ve üstü işlemcilerde LFS, LGS ve LSS komutları 32-bit offset adresinden FS, GS ve SS register'larına yükler.
- Aşağıda LOAD komutları görülmektedir.

Assembly Language	*Operation*
LEA AX,NUMB	Loads AX with the offset address of NUMB
LEA EAX,NUMB	Loads EAX with the offset address of NUMB
LDS DI,LIST	Loads DS and DI with the 32-bit contents of data segment memory location LIST
LDS EDI,LIST1	Loads the DS and EDI with the 48-bit contents of data segment memory location LIST1
LES BX,CAT	Loads ES and BX with the 32-bit contents of data segment memory location CAT
LFS DI,DATA1	Loads FS and DI with the 32-bit contents of data segment memory location DATA1
LGS SI,DATA5	Loads GS and SI with the 32-bit contents of data segment memory location DATA5
LSS SP,MEM	Loads SS and SP with the 32-bit contents of data segment memory location MEM

LOAD

LEA

- LEA komutu 16 veya 32-bit register'lara operand'la belirtilen datanın offset adresini yükler.
- LEA AX, NUMB komutu NUMB operand'ının adresini AX'e yükler.
- LEA BX,[DI] komutu [DI] operand'ının offset adresini BX'e yükler (DI içeriği). MOV BX,[DI] komutu [DI] operand'ının gösterdiği adresteki datayı BX'e yükler.

LOAD

- Aşağıdaki örnekte önce SI register'ına DATA1 operand'ının offset adresi yüklenir.
- Ardından DI register'ına DATA2 operand'ının offset adresi yüklenir.
- Program çalıştıktan sonra DATA1 ve DATA2 hafıza alanlarının içeriği yer değiştirilir.

```
        .MODEL SMALL            ;select small model
        .DATA                   ;start data segment
DATA1   DW   2000H              ;define DATA1
DATA2   DW   3000H              ;define DATA2
        .CODE                   ;start code segment
        .STARTUP                ;start program
        LEA SI,DATA1            ;address DATA1 with SI
        MOV DI,OFFSET DATA2     ;address DATA2 with DI
        MOV BX,[SI]             ;exchange DATA1 with DATA2
        MOV CX,[DI]
        MOV [SI],CX
        MOV [DI],BX
        .EXIT
         END
```

LOAD

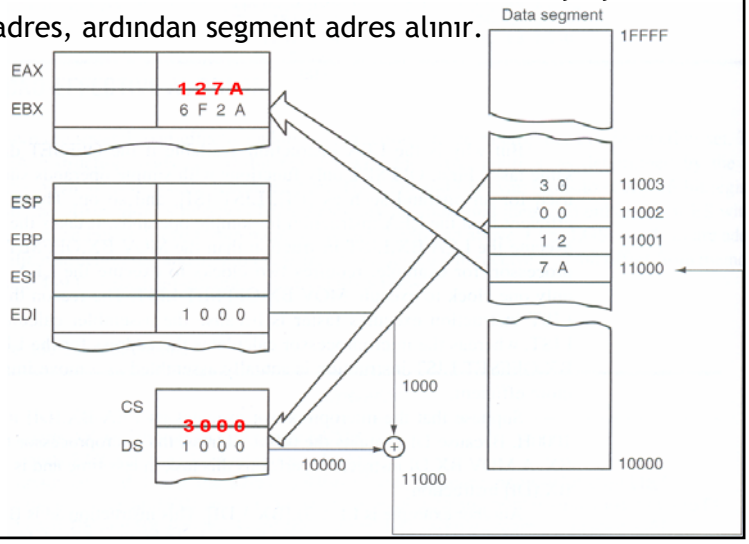
LDS, LES, LFS, LGS, LSS

- LDS, LES, LFS, LGS, LSS komutları 16 veya 32-bit register'lardan birisine offset adresini yükler, DS, ES, FS, GS, SS segment register'larından birisine segment adresini yükler.
- Bu komutlar segment ve offset adresini içeren 32-bit (16-bit segment+16-bit offset) veya 48-bit (32-bit offset+16-bit segment) hafıza alanına erişim yapar.
- LFS, LGS ve LSS komutları 80386 ve üstü işlemcilerde vardır.

LOAD

- Aşağıdaki örnekte LDS BX, [DI] komutunun çalışması görülmektedir.
- Data segment'te DI ile adreslenen 32-bit alan BX ve DS'ye yüklenir.
- Önce offset adres, ardından segment adres alınır.

BX = 127AH
DS = 3000H
olur.



LOAD

- En sık kullanılan LSS komutudur. 80386 ve üstü işlemcilerde vardır ve **.386** deyimi **.MODEL** deyiminden hemen sonra kullanılmalıdır.

```
                        .MODEL SMALL          ;select small model
                        .386                  ;select 80386
0000                    .DATA                 ;start data segment
0000 00000000     SADDR DD     ?              ;old stack address
0004 1000 [       SAREA DW     1000H DUP(?)   ;new stack area
        ????
           ]
2004 = 2004       STOP  EQU THIS WORD         ;define top of new stack
0000                    .CODE                 ;start code segment
                        .STARTUP              ;start program
0010 FA                 CLI                   ;disable interrupts
0011 8B C4              MOV AX,SP             ;save old SP
0013 A3 0000 R          MOV WORD PTR SADDR,AX
0016 8C D0              MOV AX,SS             ;save old SS
0018 A3 0002 R          MOV WORD PTR SADDR+2,AX

001B 8C D8              MOV AX,DS             ;load new SS
001D 8E D0              MOV SS,AX
001F B8 2004 R          MOV AX,OFFSET STOP    ;load new SP
0022 8B E0              MOV SP,AX
0024 FB                 STI                   ;enable interrupts

0025 8B C0              MOV AX,AX             ;do some dummy instructions
0027 8B C0              MOV AX,AX
0029 9F B2 26 0000 R    LSS SP,SADDR          ;get old stack
                        .EXIT                 ;exit to DOS
                        END                   ;end program listing
```

String Data Transfer

- LODS, STOS, MOVS, INS ve OUTS komutları kullanılır. String komutları byte,word veya doubleword data transfer eder.

Direction Flag

- Flag register'daki D biti string işlemleri için DI ve SI register'larının otomatik artırılması (D=0) veya otomatik azaltılması (D=1) için kullanılır.
- Direction biti sadece string işlemlerinde kullanılır.
- CLD komutu D=0 (clear) yapar, STD komutu D=1 (set) yapar.
- String komutu bir byte transfer ederse SI/DI 1 değişir, bir word için SI/DI 2 değişir, doubleword için SI/DI 4 değişir.
- Her komut DI/SI çiftinden birisini kullanır. STOSB komutu DI register'ını artırır veya azaltır (SI değişmez), LODSB SI register'ını artırır veya azaltır (DI değişmez).
- Tüm string komutları DI offset adresiyle **extra segment**'e, SI offset adresiyle **data segment**'e erişir.

String Data Transfer

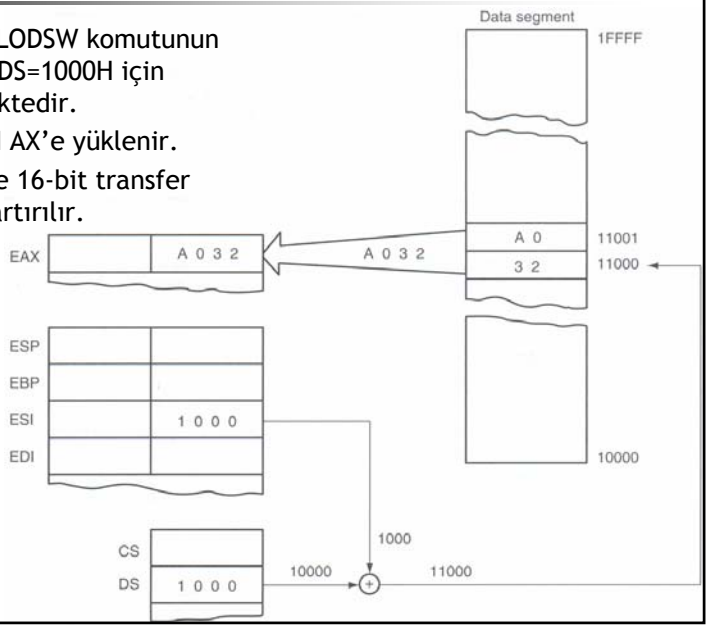
LODS

- LODS komutu AL, AX veya EAX (80386 ve üstü) register'larına data segment'te SI register'ıyla belirlenen offset adresinin içeriğini yükler.
- AL, AX veya EAX yüklendiğinde D=0 için SI artırılır D=1 için SI azaltılır.
- Byte boyutunda transfer için 1, word için 2 ve doubleword için 4 artırılır veya azaltılır.
- LODSB (loads a byte) komutu AL register'ına bir byte, LODSW (loads a word) AX register'ına bir word ve LODSD (loads a doubleword) EAX register'ına bir doubleword yükler.

Assembly Language	*Operation*
LODSB	AL = DS:[SI]; SI = SI ± 1
LODSW	AX = DS:[SI]; SI = SI ± 2
LODSD	EAX = DS:[SI]; SI = SI ± 4
LODS LIST	AL = DS:[SI]; SI = SI ± 1 (if LIST is a byte)
LODS DATA1	AX = DS:[SI]; SI = SI ± 2 (if DATA1 is a word)
LODS FROG	EAX = DS:[SI]; SI = SI ± 4 (if FROG is a doubleword)

String Data Transfer

- Aşağıdaki şekilde LODSW komutunun D=0, SI=1000H ve DS=1000H için çalışması görülmektedir.
- 11000H ve 11001H AX'e yüklenir.
- D=0 olduğundan ve 16-bit transfer edildiğinden SI 2 artırılır.
- SI=1002 olacaktır.



String Data Transfer

STOS

- STOS komutu AL, AX veya EAX (80386 ve üstü) register'larını extra segment'te DI register'ıyla belirlenen offset adresine kopyalar.
- AL, AX veya EAX aktarıldığında D=0 için DI artırılır D=1 için DI azaltılır.
- Byte boyutunda transfer için 1, word için 2 ve doubleword için 4 artırılır veya azaltılır.
- STOSB (stores a byte) komutu AL register'ını, STOSW (stores a word) AX register'ını ve STOSD (stores a doubleword) EAX register'ını extra segment'te DI ile belirtilen offset adresine kopyalar.

Assembly Language	*Operation*
STOSB	ES:[DI] = AL; DI = DI ± 1
STOSW	ES:[DI] = AX; DI = DI ± 2
STOSD	ES:[DI] = EAX; DI = DI ± 4
STOS LIST	ES:[DI] = AL; DI = DI ± 1 (if LIST is a byte)
STOS DATA3	ES:[DI] = AX; DI = DI ± 2 (if DATA3 is a word)
STOS DATA4	ES:[DI] = EAX; DI = DI ± 4 (if DATA4 is a doubleword)

String Data Transfer

STOS ve REP

- REP (repeat prefix) deyimi LODS hariç diğer string data transfer komutlarına eklenir.
- REP deyimi her string komutu çalıştığında CX sayacını 1 azaltır. CX = 0 olduğunda döngü biter ve sonraki komut çalışır.
- CX=100 iken REP STOSB komutu, DI register'ı her döngüde 1 azalacağı veya artacağı için AL register'ının içeriğini 100 adreslik bir hafıza bloğuna aktarır.
- STOSB (stores a byte) komutu AL register'ını, STOSW (stores a word) AX register'ını ve STOSD (stores a doubleword) EAX register'ını DI ile belirtilen extra segment alanına aktarır.
- STOSW komutu DI register'ını 2 değiştirir, STOSD komutu 4 değiştirir.

String Data Transfer

STOS ve REP

- Aşağıdaki örnek Buffer adlı hafıza bloğunu Count değeri kadar temizler.
- Program C++ ortamında inline assembler ile yazılmış ve ClearBuffer fonksiyonu çağırılarak çalıştırılmaktadır.
- REP STOSW komutu Buffer adlı hafıza bloğunu AX=0 yazarak temizler.

```
void ClearBuffer ( int Count, short* Buffer )
{
   _asm{
         push edi              ;save registers
         push es
         push ds
         mov  ax,0
         mov  ecx, Count
         mov  edi, Buffer
         pop  es               ;load ES with DS
         rep  stosw            ;clear Buffer
         pop  es               ;restore registers
         pop  edi
   }
}
```

her tekrarda ECX 1 azalır, EDI 2 (D=0)artar.

String Data Transfer

Operand operatörleri

- Herhangi bir operand operatörlerle birlikte kullanılabilir.

Operator	Example	Comment
+	MOV AL,6+3	Copies 9 into AL
–	MOV AL,6–3	Copies 3 into AL
*	MOV AL,4*3	Copies 12 into AL
/	MOV AX,12/5	Copies 2 into AX (remainder is lost)
MOD	MOV AX,12 MOD 7	Copies 5 into AX (quotient is lost)
AND	MOV AX,12 AND 4	Copies 4 into AX (1100 AND 0100 = 0100)
OR	MOV EAX,12 OR 1	Copies 13 into EAX (1100 OR 0001 = 1101)
NOT	MOV AL,NOT 1	Copies 254 into AL (NOT 0000 0001 = 1111 1110 or 254)

String Data Transfer

MOVS

- MOVS komutu iki hafıza alanı arasında string data aktarır.
- 8086-Pentium 4 mikroişlemcilerde memory-to-memory işlemine izin veren tek komuttur.
- MOVS komutu data segment'te SI ile adreslenen byte, word veya doubleword data'yı DI ile adreslenen extra segment'e kopyalar.
- SI ve DI için artma/azalma direction bayrağıyla belirlenir.

Assembly Language	Operation
MOVSB	ES:[DI] = DS:[SI]; DI = DI ± 1; SI = SI ± 1 (byte transferred)
MOVSW	ES:[DI] = DS:[SI]; DI = DI ± 2; SI = SI ± 2 (word transferred)
MOVSD	ES:[DI] = DS:[SI]; DI = DI ± 4; SI = SI ± 4 (doubleword transferred)
MOVS BYTE1, BYTE2	ES:[DI] = DS:[SI]; DI = DI ± 1; SI = SI ± 1 (byte transferred if BYTE1 and BYTE2 are bytes)
MOVS WORD1,WORD2	ES:[DI] = DS:[SI]; DI = DI ± 2; SI = SI ± 2 (word transferred if WORD1 and WORD2 are words)
MOVS TED,FRED	ES:[DI] = DS:[SI]; DI = DI ± 4; SI = SI ± 4 (doubleword transferred if TED and FRED are doublewords)

String Data Transfer

Blok Transfer

- REP MOVSD komutu BlockA'yı BlockB'ye kopyalar.

```
void TransferBlocks (int BlockSize, int* BlockA, int* BlockB)
{
      _asm{
            push es                    ;save registers
            push edi
            push esi
            push ds                    ;copy DS into ES
            pop  es
            mov  esi, BlockA           ;address BlockA
            mov  edi, BlockB           ;address BlockB
            mov  ecx, BlockSize        ;load count
            rep  movsd                 ;move data
            pop  esi                   ;restore registers
            pop  edi
            pop  es
      }
}
```

String Data Transfer

INS

- INS komutu bir I/O cihazından byte, word veya doubleword data'yı extra segment içerisinde DI ile adreslenen alana aktarır. (8086/8088'de yoktur.)
- I/O adresi DX register'ından alınır.
- Bir harici I/O cihazından hafızaya blok data aktarımında etkindir.
- INSB byte, INSW word ve INSD doubleword data transfer eder.

Assembly Language	*Operation*
INSB	ES:[DI] = [DX]; DI = DI ± 1 (byte transferred)
INSW	ES:[DI] = [DX]; DI = DI ± 2 (word transferred)
INSD	ES:[DI] = [DX]; DI = DI ± 4 (doubleword transferred)
INS LIST	ES:[DI] = [DX]; DI = DI ± 1 (if LIST is a byte)
INS DATA4	ES:[DI] = [DX]; DI = DI ± 2 (if DATA4 is a word)
INS DATA5	ES:[DI] = [DX]; DI = DI ± 4 (if DATA5 is a doubleword)

String Data Transfer

INS - örnek

- Aşağıdaki örnekte 03ACH adresine sahip I/O cihazından 50 byte data'yı extra segment'te LISTS dizisine aktarılır.

```
MOV DI,OFFSET LISTS     ;address array
MOV DX,3ACH             ;address I/O
CLD                     ;auto-increment
MOV CX,50               ;load counter
REP INSB                ;input data
```

String Data Transfer

OUTS

- OUTS komutu data segment içerisinden SI ile adreslenen alandan byte, word veya doubleword data'yı bir I/O cihazına aktarır. (8086/8088'de yoktur.)
- I/O adres DX register'ındadır.
- Bir blok datanın harici I/O cihazına aktarımında etkindir.
- OUTSB byte, OUTW word ve OUTSD doubleword data transfer eder.

Assembly Language	*Operation*
OUTSB	[DX] = DS:[SI]; SI = SI ± 1 (byte transferred)
OUTSW	[DX] = DS:[SI]; SI = SI ± 2 (word transferred)
OUTSD	[DX] = DS:[SI]; SI = SI ± 4 (doubleword transferred)
OUTS DATA7	[DX] = DS:[SI]; SI = SI ± 1 (if DATA7 is a byte)
OUTS DATA8	[DX] = DS:[SI]; SI = SI ± 2 (if DATA8 is a word)
OUTS DATA9	[DX] = DS:[SI]; SI = SI ± 4 (if DATA9 is a doubleword)

String Data Transfer

OUTS - örnek

- Aşağıdaki örnekte data segment'te ARRAY adresindeki 100 byte data 03ACH adresine sahip I/O cihazına aktarılır.

```
MOV SI,OFFSET ARRAY     ;address array
MOV DX,3ACH             ;address I/O
CLD                     ;auto-increment
MOV CX,100              ;load counter
REP OUTSB               ;output data
```

Diğer Data Transfer Komutları

XCHG

- XCHG komutu iki register veya bir register ile bir hafıza alanı arasında data'yı karşılıklı değiştirir.
- XCHG komutu segment register'larını değiştiremez veya memory-to-memory değiştiremez.
- XCHG AL,[DI] ile XCHG [DI],AL aynı işlemi yapar.

Assembly Language	*Operation*
XCHG AL,CL	Exchanges the contents of AL with CL
XCHG CX,BP	Exchanges the contents of CX with BP
XCHG EDX,ESI	Exchanges the contents of EDX with ESI
XCHG AL,DATA2	Exchanges the contents of AL with data segment memory location DATA2

Diğer Data Transfer Komutları

IN ve OUT

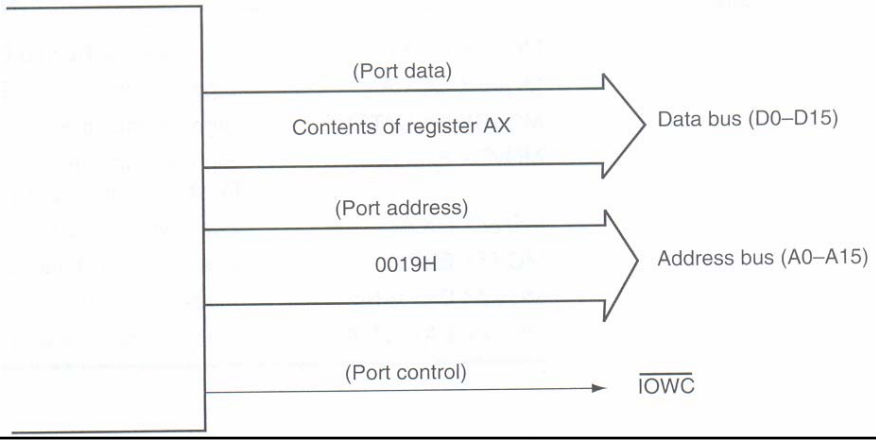
- IN komutu harici I/O cihazından AL,AX ve EAX register'larına data aktarır.
- OUT komutu AL,AX ve EAX'ten harici I/O cihazına data aktarır.
- 16-bit I/O port adres **Address Bus** ile ilgili cihaza iletilir.
- Fixed port adreslemede port adresi opcode'dan hemen sonra gelir. Variable port adreslemede DX register'ı port adresini saklar.

Assembly Language	*Operation*
IN AL,p8	8 bits are input to AL from I/O port p8
IN AX,p8	16 bits are input to AX from I/O port p8
IN EAX,p8	32 bits are input to EAX from I/O port p8
IN AL,DX	8 bits are input to AL from I/O port DX
IN AX,DX	16 bits are input to AX from I/O port DX
IN EAX,DX	32 bits are input to EAX from I/O port DX
OUT p8,AL	8 bits are output to I/O port p8 from AL
OUT p8,AX	16 bits are output to I/O port p8 from AX
OUT p8,EAX	32 bits are output to I/O port p8 from EAX
OUT DX,AL	8 bits are output to I/O port DX from AL
OUT DX,AX	16 bits are output to I/O port DX from AX
OUT DX,EAX	32 bits are output to I/O port DX from EAX

Diğer Data Transfer Komutları

IN ve OUT - örnek

- Aşağıdaki örnekte OUT 19H,AX komutunun çalışması görülmektedir.
- AX içeriği I/O port 19H adresine aktarılır. Adres bus 0019H (16-bit) değerine, data bus AX içeriğine ve control bus $\overline{\text{IOCW}}$ (I/O write control) kontrol işaretine (lojik 0 ilgili I/O cihazını etkin yapar (enable)) sahiptir.



Segment Override Prefix

- Komutlar için default (varsayılan) register değiştirilerek farklı segment'teki data'ya erişmeyi sağlar.
- MOV AX,[DI] komutu default olarak data segment'e erişir. MOV AX,ES:[DI] komutu extra segment'e erişmeyi sağlar.
- Aşağıda bazı komutlar görülmektedir.

Assembly Language	*Segment Accessed*	*Default Segment*
MOV AX,DS:[BP]	Data	Stack
MOV AX,ES:[BP]	Extra	Stack
MOV AX,SS:[DI]	Stack	Data
MOV AX,CS:LIST	Code	Data
MOV ES:[SI],AX	Extra	Data
LODS ES:DATA1	Extra	Data
MOV EAX,FS:DATA2	FS	Data
MOV GS:[ECX],BL	GS	Data

Assembler Komutları

- Assembler komutları (directives), operand veya programın bir kısmında assembler tarafından nasıl işlem yapılacağını belirler.

Directive	*Function*
.286	Selects the 80286 instruction set
.286P	Selects the 80286 protected mode instruction set
.386	Selects the 80386 instruction set
.386P	Selects the 80386 protected mode instruction set
.486	Selects the 80486 instruction set
.486P	Selects the 80498 protected mode instruction set
.586	Selects the Pentium instruction set
.586P	Selects the Pentium protected mode instruction set
.686	Selects the Pentium Pro–Pentium 4 instruction set
.686P	Selects the Pentium Pro–Pentium 4 protected mode instruction set
.287	Selects the 80287 math coprocessor
.387	Selects the 80387 math coprocessor
.CODE	Indicates the start of the code segment (models only)
.DATA	Indicates the start of the data segment (models only)
.EXIT	Exits to DOS (models only)
.MODEL	Selects the programming model
.STACK	Selects the start of the stack segment (models only)
.STARTUP	Indicates the starting instruction in a program (models only)
ALIGN n	Align to boundary n (n = 2 for words, n = 4 for doublewords)
ASSUME	Informs the assembler to name each segment (full segments only)
BYTE	Indicates byte-sized, as in BYTE PTR

Assembler Komutları

DB	Defines byte(s) (8 bits)
DD	Defines doubleword(s) (32 bits)
DQ	Defines quadwords(s) (64 bits)
DT	Defines ten byte(s) (80 bits)
DUP	Generates duplicates
DW	Defines word(s) (16 bits)
DWORD	Indicates doubleword-sized, as in DWORD PTR
END	Ends a program file
ENDM	Ends a MACRO sequence
ENDP	Ends a procedure
ENDS	Ends a segment or data structure
EQU	Equates data or a label to a label
FAR	Defines a far pointer as in FAR PTR
MACRO	Designates the start of a MACRO sequence
NEAR	Defines a near pointer as in NEAR PTR
OFFSET	Specifies an offset address
ORG	Sets the origin within a segment
OWORD	Indicates octalwords, as in OWORD PTR
PROC	Starts a procedure
PTR	Designates a pointer
SEGMENT	Starts a segment for full segments
STACK	Starts a stack segment for full segments
STRUC	Defines the start of a data structure
USES	Automatically pushes and pops registers
USE16	Uses 16-bit instruction mode
USE32	Uses 32-bit instruction mode
WORD	Indicates word-sized, as in WORD PTR

Assembler Komutları

- DB (define byte), DW (define word) ve DD (define doubleword) hafıza tanımlama ve data aktarmak için kullanılır.
- SEGMENT deyimiyle istenildiği kadar yeni sembolik isme sahip segment tanımlanabilir.
- DB, DW ve DD komutlarıyla tanımlanan alanlar ? ile sonraki kullanımlar için reserve edilebilir.
- DUP (duplicates) komutu bir dizi oluşturur. 10 DUP (?) komutu herhangi bir değer atamadan 10 hafıza alanı ayırır.
- DATA1 DB 10 DUP (2) komutu 10 byte hafıza alanını ayırır ve her birisinin içerisine 02H atar.

Assembler Komutları

Örnek

```
0000                    LIST_SEG      SEGMENT

0000 01 02 03           DATA1   DB    1,2,3             ;define bytes
0003 45                         DB    45H               ;hexadecimal
0004 41                         DB    'A'               ;ASCII
0005 F0                         DB    11110000B         ;binary
0006 000C 000D          DATA2   DW    12,13             ;define words
000A 0200                       DW    LIST1             ;symbolic
000C 2345                       DW    2345H             ;hexadecimal
000E 00000300           DATA3   DD    300H              ;define doubleword
0012 4007DF3B                   DD    2.123             ;real
0016 544269E1                   DD    3.34E+12          ;real
001A 00                 LISTA   DB    ?                 ;reserve 1 byte
001B 000A[              LISTB   DB 10 DUP(?)            ;reserve 10 bytes
         ??
         ]
0025 00                         ALIGN  2                ;set word boundary
0026 0100[              LISTC   DW    100H DUP(0)       ;reserve 100H words
         0000
           ]
0226 0016[              LISTD   DD    22 DUP(?)         ;reserve 22 doublewords
         ????????
              ]
027E 0064[              SIXES   DB    100 DUP(6)        ;reserve 100 bytes
         06
           ]
02E2                    LIST_SEG      ENDS
```

Assembler Komutları

ASSUME, EQU ve ORG

- **ASSUME** deyimi code, data, extra ve stack segment için seçilen isimleri belirtmek için kullanılır. **EQU** (equate) deyimi bir sayıyı, ASCII karakteri veya etiketi başka bir etikete eşitler.
- **ORG** (origin) deyimi data segment içindeki offset adresini değiştirmek için kullanılır. **THIS** deyimi THIS BYTE, THIS WORD veya THIS DWORD şeklinde kullanılır ve ilgili etikete belirtilen data aktarılabilir.

```
                        ;Using the THIS and ORG directives
                        ;
0000                    DATA_SEG      SEGMENT

0300                            ORG   300H

= 0300                  DATA1   EQU   THIS BYTE
0300                    DATA2   DW    ?
0302                    DATA_SEG      ENDS

0000                    CODE_SEG      SEGMENT 'CODE'
                                ASSUME CS:CODE_SEG, DS:DATA_SEG
0000 8A 1E 0300 R               MOV   BL,DATA1
0004 A1 0300 R                  MOV   AX,DATA2
0007 8A 3E 0301 R               MOV   BH,DATA1+1

000B                    CODE_SEG      ENDS
```

Assembler Komutları

PROC ve ENDP

- **PROC** ve **ENDP** deyimleri bir prosedürün başlangıcını ve bitişini gösterir.
- PROC ve ENDP deyimleri başlangıç ve bitişi için etikete gerek duyarlar.
- PROC deyimi NEAR veya FAR şeklinde iki ifade kullanır.
- NEAR (lokal) procedürün aynı code segment'e yerleştirilmesi, FAR (global) hafızada herhangi bir yere yerleştirilmesini sağlar.
- Aşağıdaki örnekte BX, CX, DX toplanarak sonuç AX register'ına aktarılmaktadır.

```
                        ;A procedure that adds BX, CX, and DX with the
                        ;sum stored in AX
                        ;
0000                    ADDEM PROC    FAR           ;start of procedure

0000 03 D9                    ADD     BX,CX
0002 03 DA                    ADD     BX,DX
0004 8B C3                    MOV     AX,BX
0006 CB                       RET

0007                    ADDEM ENDP                  ;end of procedure
```

Hafıza Organizasyonu

- Assembler iki farklı hafıza organizasyonu kullanır. Birisi model kullanımı diğeri tüm segment tanımlarının (full-segment definitions) yapılmasıdır.
- Model kullanımı kolaydır ve basit programlarda kullanılmalıdır.
- Full-segment definitions assembly dilinnde daha iyi kontrol sağlar ve kompleks programlarda kullanılmalıdır.

Model

- MASM assembler **tiny**, **small**, **medium**, **compact**, **large**, **huge**, **flat** modelleri kullanır.
- **Tiny** model tüm programın 64 KB tek segment'e sığdırılmasını sağlar.
- **Small** model bir data segment ve bir code segment kullanılmasını sağlar ve toplam boyut 128KB olur.
- **Medium** model birden çok code segment ve bir data segment kullanılmasını sağlar.
- **Compact** model, bir code segment ve birden çok data segment kullanılmasını sağlar.
- **Large** model, birden çok code segment ve birden çok data segment kullanılmasını sağlar. 64K dan büyük dizi kullanılamaz.
- **Huge** model, birden çok code segment ve birden çok data segment kullanılmasını sağlar. 64K dan büyük dizi kullanılabilir.
- **Flat** model, 4GB'a kadar bir segment kullanılmasını sağlar. Tüm data ve code 32-bit bir segmentte bulunabilir.

Hafıza Organizasyonu

Model

- Aşağıdaki örnekte 100-byte LISTA isimli hafıza bloğunu LISTB alanına kopyalar.

```
                        .MODEL SMALL          ;select small model
                        .STACK 100H           ;define stack
                        .DATA                 ;start data segment

0000 0064[          LISTA   DB    100 DUP(?)
        ??
          ]
0064 0064[          LISTB   DB    100 DUP(?)
        ??
          ]

                        .CODE                 ;start code segment

0000 B9 ---- ?      HERE:   MOV   AX,@DATA     ;load ES and DS
0003 8E C0                  MOV   ES,AX
0005 8E D8                  MOV   DS,AX
0007 FC                     CLD                ;move data
0008 BE 0000 R              MOV   SI,OFFSET LISTA
000B BF 0064 R              MOV   DI,OFFSET LISTB
000E B9 0064                MOV   CX,100
0011 F3/A4                  REP   MOVSB

0013                        .EXIT 0            ;exit to DOS
                            END HERE
```

Hafıza Organizasyonu

Full-Segment Tanımı

- Model yaklaşımına göre daha yapısaldır. **Örnek sonraki sayfadadır.**
- STACK_SEG için ayrılan alan SEGMENT ve ENDS arasıdır.
- DW 100H DUP(?) komutu stack segment için 100H adet word alan oluşturur.
- STACK kelimesi SS ve SP register'larını otomatik olarak yükler.
- DATA_SEG için LISTA ve LISTB her elemanı byte olan 100 elemanlı iki dizi tanımlanmıştır.
- CODE_SEG ile ENDS arası code segment'i göstermektedir.
- 'DATA' ve 'CODE' kelimeleri MS CodeView ile debug işlemi için kullanılır.
- ASSUME deyimi SS için STACK_SEG, DS için DATA_SEG ve CS için CODE_SEG isminin kullanıldığını gösterir.

```
0000                    STACK_SEG       SEGMENT         'STACK'
0000 0064[                              DW      100H DUP(?)
          ????
                ]
0200                    STACK_SEG       ENDS

0000                    DATA_SEG        SEGMENT         'DATA'
0000 0064[              LISTA   DB      100 DUP(?)
          ??
               ]
0064 0064[              LISTB   DB      100 DUP(?)
          ??
               ]
00CB                    DATA_SEG        ENDS

0000                    CODE_SEG        SEGMENT         'CODE'
                                ASSUME CS:CODE_SEG,DS:DATA_SEG
                                ASSUME SS:STACK_SEG
0000                    MAIN    PROC    FAR
0000 B8 ---- R                  MOV     AX,DATA_SEG             ;load DS and ES
0003 8E C0                      MOV     ES,AX
0005 8E D8                      MOV     DS,AX
0007 FC                         CLD                             ;save data
0008 BE 0000 R                  MOV     SI,OFFSET LISTA
000B BF 0064 R                  MOV     DI,OFFSET LISTB
000E B9 0064                    MOV     CX,100
0011 F3/A4                      REP     MOVSB
0013 B4 4C                      MOV     AH,4CH                  ;exit to DOS
0015 CD 21                      INT     21H
0017                    MAIN    ENDP
0017                    CODE_SEG        ENDS
                                END     MAIN
```

Ödev

- Her birisi byte olan 10 elemanlı DIZI1 adlı dizinin elemanlarını hepsi byte olan 100 elemanlı DIZI2 adlı diziye 10 kez ardarda kopyalayan bir program yazınız. Programda gerekli gördüğünüz yerlere açıklama yazınız. Program courier new font ile ve 12 boyutunda yazılacaktır. Programda sadece etiket, komut ve açıklama kısımlarının yazılması yeterlidir.