x86 Assembly Programming Part 2

EECE416 Microcomputer

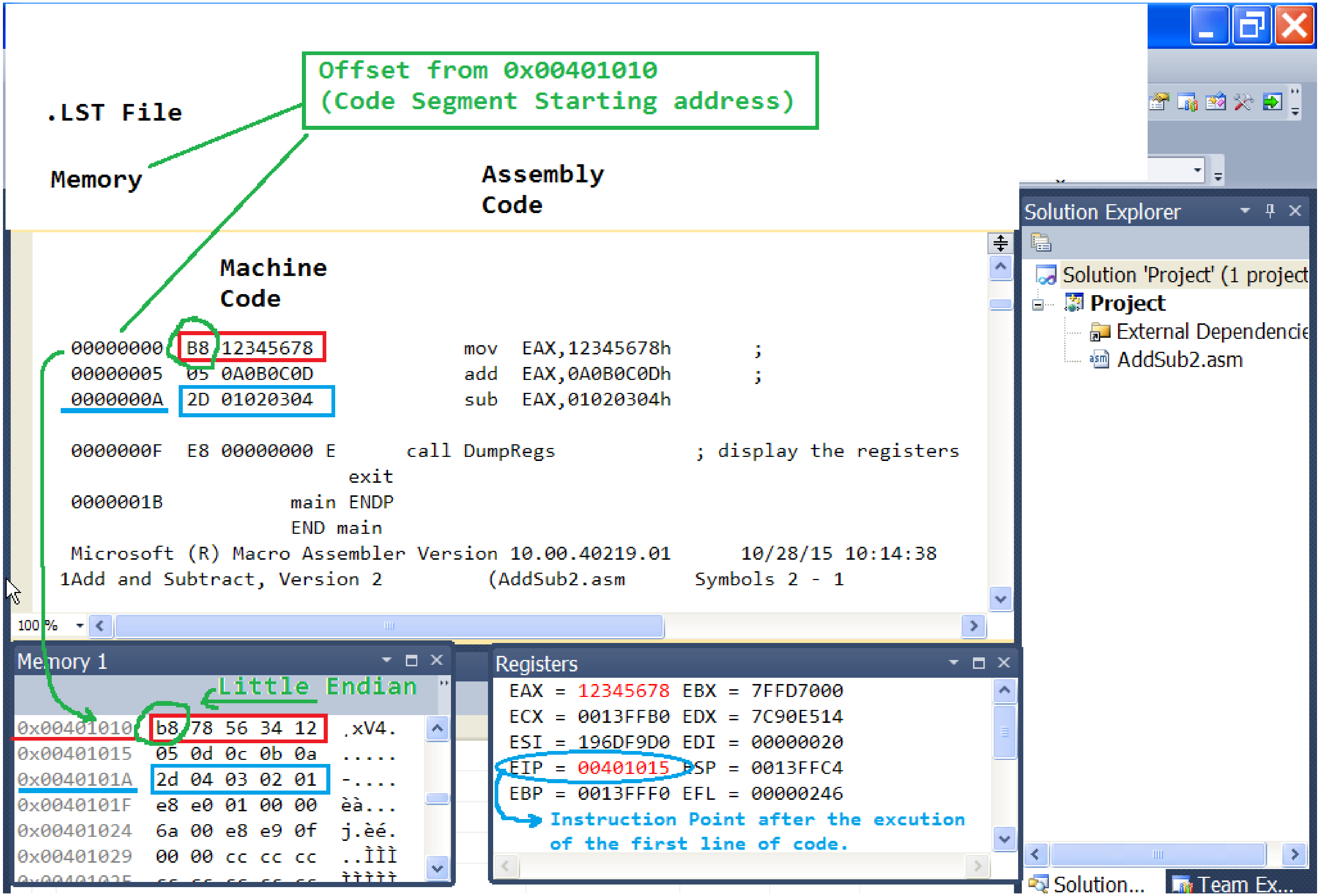
\

Resources:

Intel 80386 Programmers Reference Manual
Essentials of 80x86 Assembly Language
Introduction to 80x86 Assembly Language Programming

WWW.MWFTR.COM/uC.html

LST File and The Memory Contents of Code



Offset from 0x00401010
(Code Segment Starting address)

.LST File

Memory

Assembly Code

Machine Code

```
00000000  B8 12345678          mov   EAX,12345678h        ;
00000005  05 0A0B0C0D          add   EAX,0A0B0C0Dh        ;
0000000A  2D 01020304          sub   EAX,01020304h

0000000F  E8 00000000 E      call DumpRegs             ; display the registers
                          exit
0000001B            main ENDP
                    END main
Microsoft (R) Macro Assembler Version 10.00.40219.01      10/28/15 10:14:38
1Add and Subtract, Version 2          (AddSub2.asm      Symbols 2 - 1
```

Little Endian

```
0x00401010  b8 78 56 34 12  ¸xV4.
0x00401015  05 0d 0c 0b 0a  .....
0x0040101A  2d 04 03 02 01  -....
0x0040101F  e8 e0 01 00 00  èà...
0x00401024  6a 00 e8 e9 0f  j.èé.
0x00401029  00 00 cc cc cc  ..ÌÌÌ
0x0040102E  cc cc cc cc cc  ÌÌÌÌÌ
```

```
EAX = 12345678 EBX = 7FFD7000
ECX = 0013FFB0 EDX = 7C90E514
ESI = 196DF9D0 EDI = 00000020
EIP = 00401015 ESP = 0013FFC4
EBP = 0013FFF0 EFL = 00000246
```

Instruction Point after the excution of the first line of code.

Registers for x86

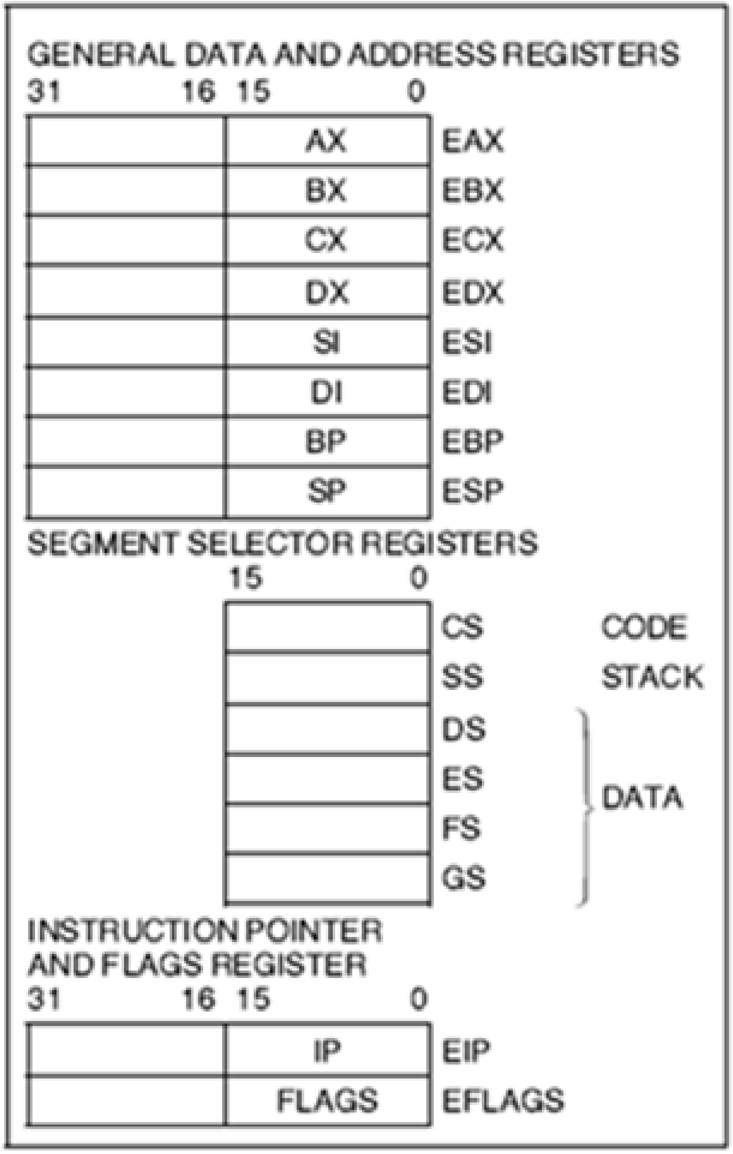
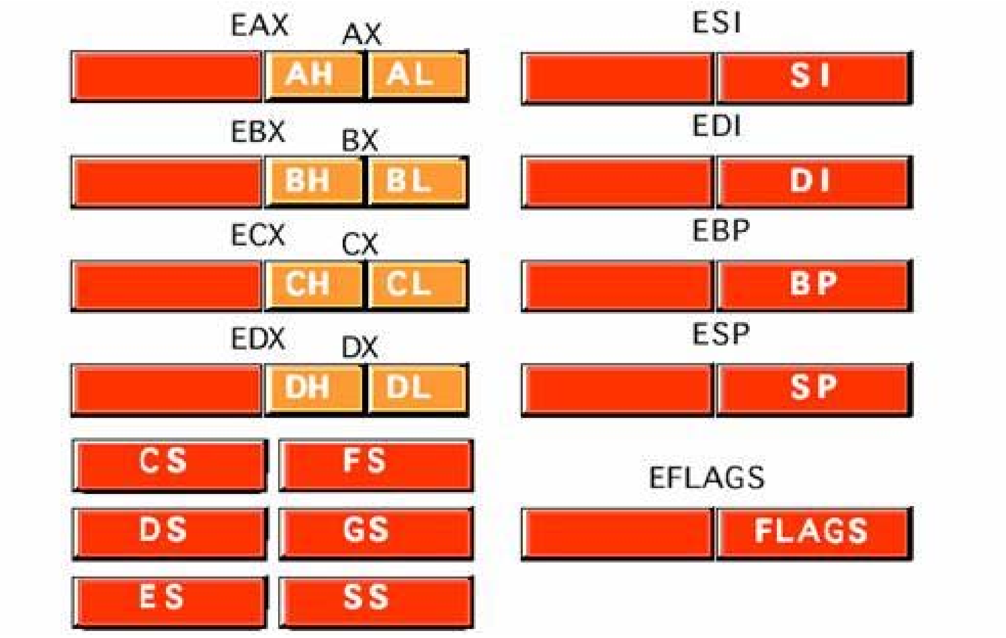
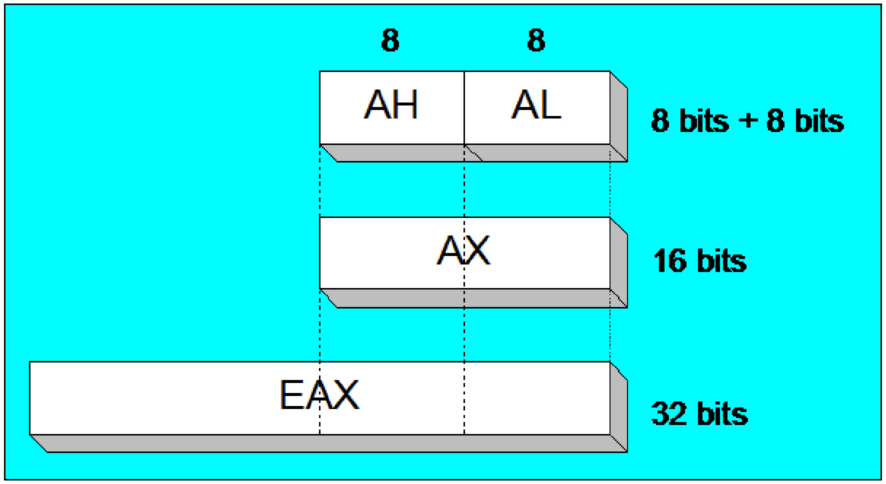


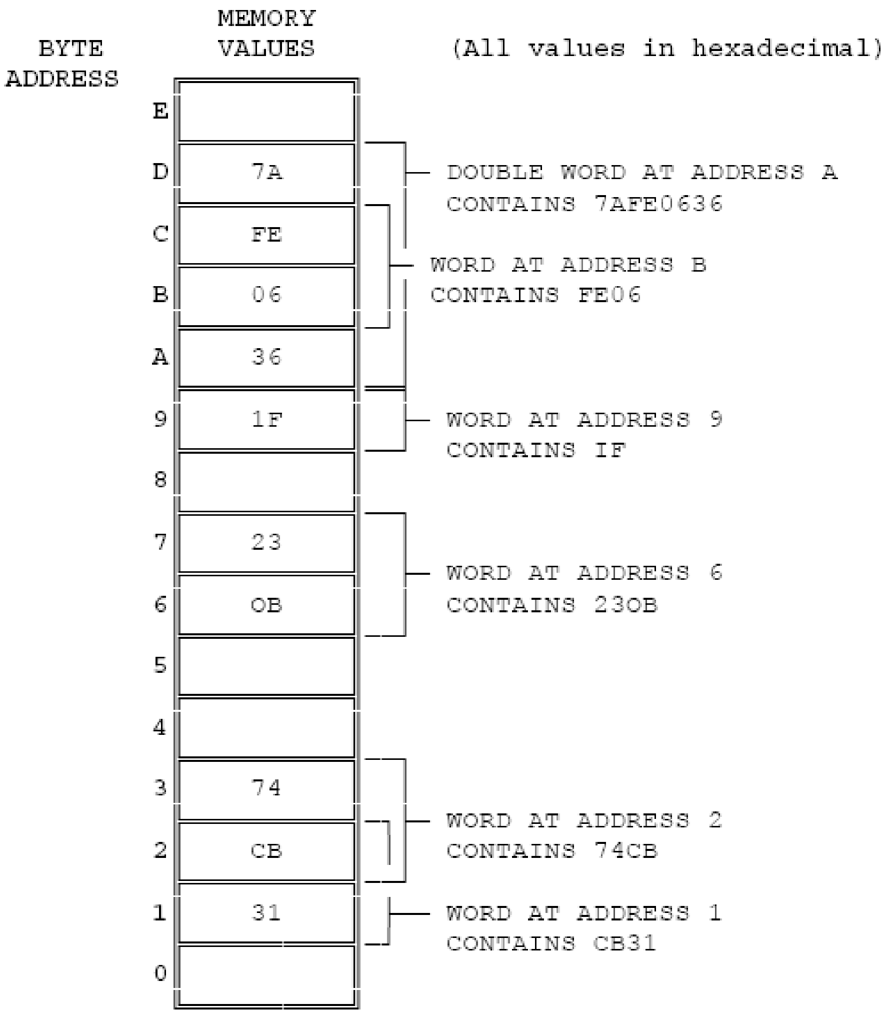
Figure 2-1. Intel386™ DX Base Architecture Registers





Basic Data Types

- Byte (BYTE), Words (WORD), Double Words (DWORD)
- Little-Endian
- Align by 2 (word) or 4 (Dword) for better performance – instead of odd address



(All values in hexadecimal)

BYTE ADDRESS	MEMORY VALUES
E	
D	7A
C	FE
B	06
A	36
9	1F
8	
7	23
6	OB
5	
4	
3	74
2	CB
1	31
0	

DOUBLE WORD AT ADDRESS A CONTAINS 7AFE0636

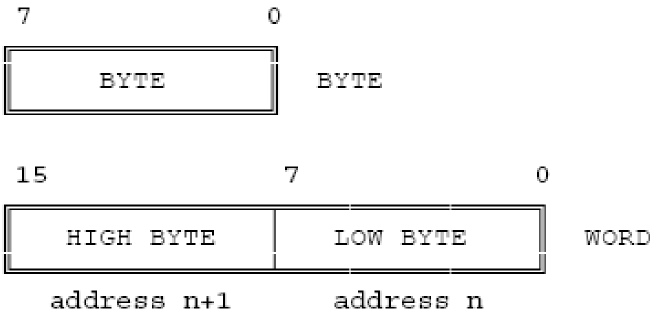
WORD AT ADDRESS B CONTAINS FE06

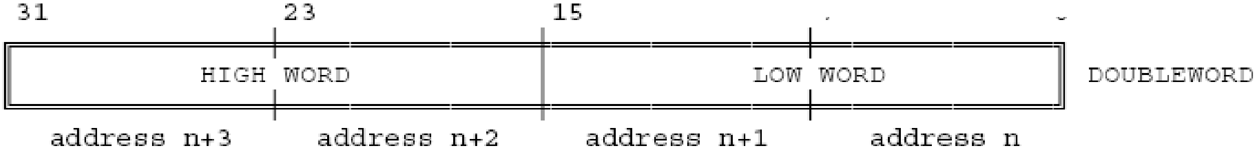
WORD AT ADDRESS 9 CONTAINS IF

WORD AT ADDRESS 6 CONTAINS 230B

WORD AT ADDRESS 2 CONTAINS 74CB

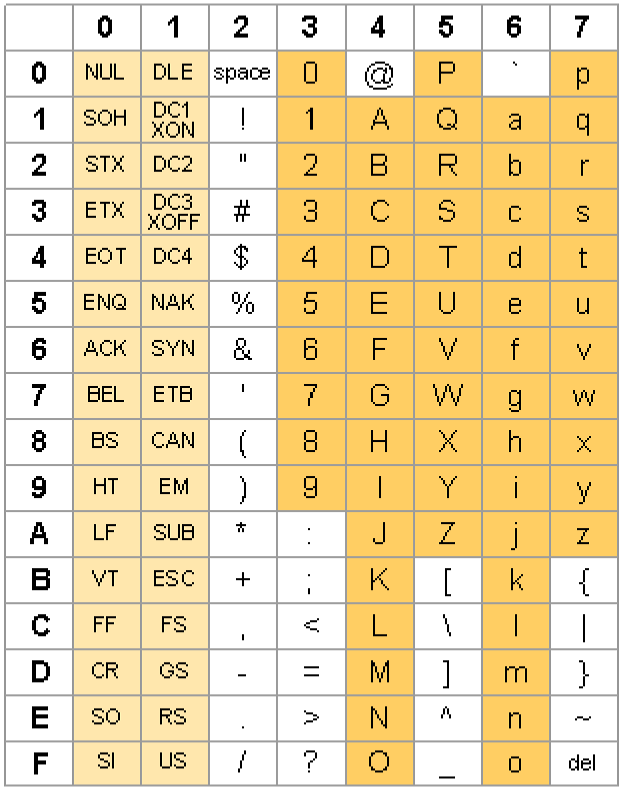
WORD AT ADDRESS 1 CONTAINS CB31





Data Declaration

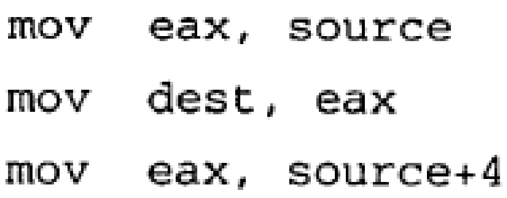
- Directives for Data Declaration and Reservation of Memory
  - BYTE: Reserves 1 byte in memory
    - Example:
      ```
      D1   BYTE      20
      D2   BYTE      00010100b
      String1  BYTE     “Joe” ;
                        [4A 6F   65]
      ```
  - WORD: 2 bytes are reserved
    - Example:
      ```
      num1      WORD        -10
      num2      WORD        FFFFH
      ```
  - DWORD: 4 bytes are reserved
    - Example :
      ```
      N1          DWORD      -10
      ```
  - QWORD: 8 bytes
    - 64 bit: RAX RBX RCX ,etc
    - 32 bit: **EDX:EAX** Concatenation for **CDQ** instruction



	0	1	2	3	4	5	6	7
0	NUL	DLE	space	0	@	P	`	p
1	SOH	DC1 XON	!	1	A	Q	a	q
2	STX	DC2	"	2	B	R	b	r
3	ETX	DC3 XOFF	#	3	C	S	c	s
4	EOT	DC4	$	4	D	T	d	t
5	ENQ	NAK	%	5	E	U	e	u
6	ACK	SYN	&	6	F	V	f	v
7	BEL	ETB	'	7	G	W	g	w
8	BS	CAN	(	8	H	X	h	x
9	HT	EM	)	9	I	Y	i	y
A	LF	SUB	*	:	J	Z	j	z
B	VT	ESC	+	;	K	[	k	{
C	FF	FS	,	<	L	\	l	|
D	CR	GS	-	=	M	]	m	}
E	SO	RS	.	>	N	^	n	~
F	SI	US	/	?	O	_	o	del

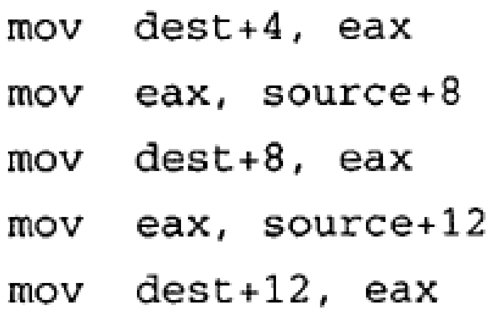
Instruction Format

- Opcode:
  - specifies the operation performed by the instruction.



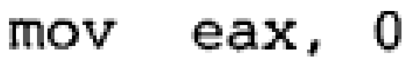
```
mov   eax, source
mov   dest, eax
mov   eax, source+4
```

- Register specifier
  - an instruction may specify one or two register operands.
- Addressing-mode specifier
  - when present, specifies whether an operand is a register or memory location.
- Displacement
  - when the addressing-mode specifier indicates that a displacement will be used to compute the address of an operand, the displacement is encoded in the instruction.



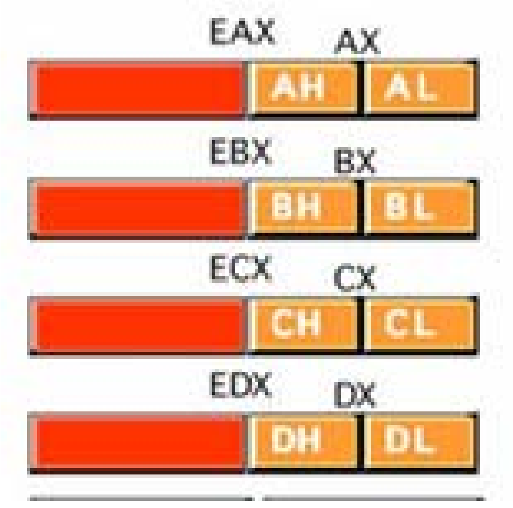
```
mov   dest+4, eax
mov   eax, source+8
mov   dest+8, eax
mov   eax, source+12
mov   dest+12, eax
```

- Immediate operand
  - when present, directly provides the value of an operand of the instruction. Immediate operands may be 8, 16, or 32 bits wide.



```
mov   eax, 0
```

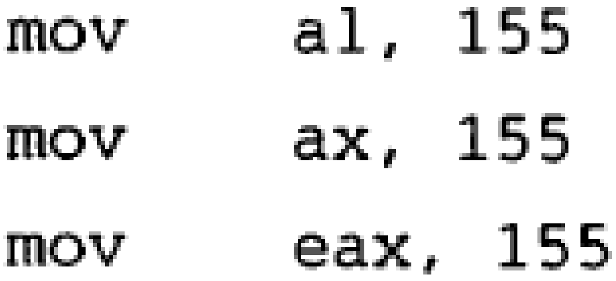
Register Size and Data



- Register/Data designation dependency:
- Before [EAX]= 01234567
- **Instruction**
  - `mov EAX, 1Fh`
  - After [EAX]= 0000001F
  - `mov eax, 1F00h`
  - After [EAX] = 00001F00
  - `mov AX, 1F00h`
  - After [EAX] = 01231F00
  - `mov AL, 1Fh`
  - After [EAX]=0123451F
  - `mov ah, 1Fh`
  - After [EAX] = 01231F67

Register Size and Data

- Assuming that the content of **eax** is [01FF01FF], what would be the content of **eax** after each instruction?

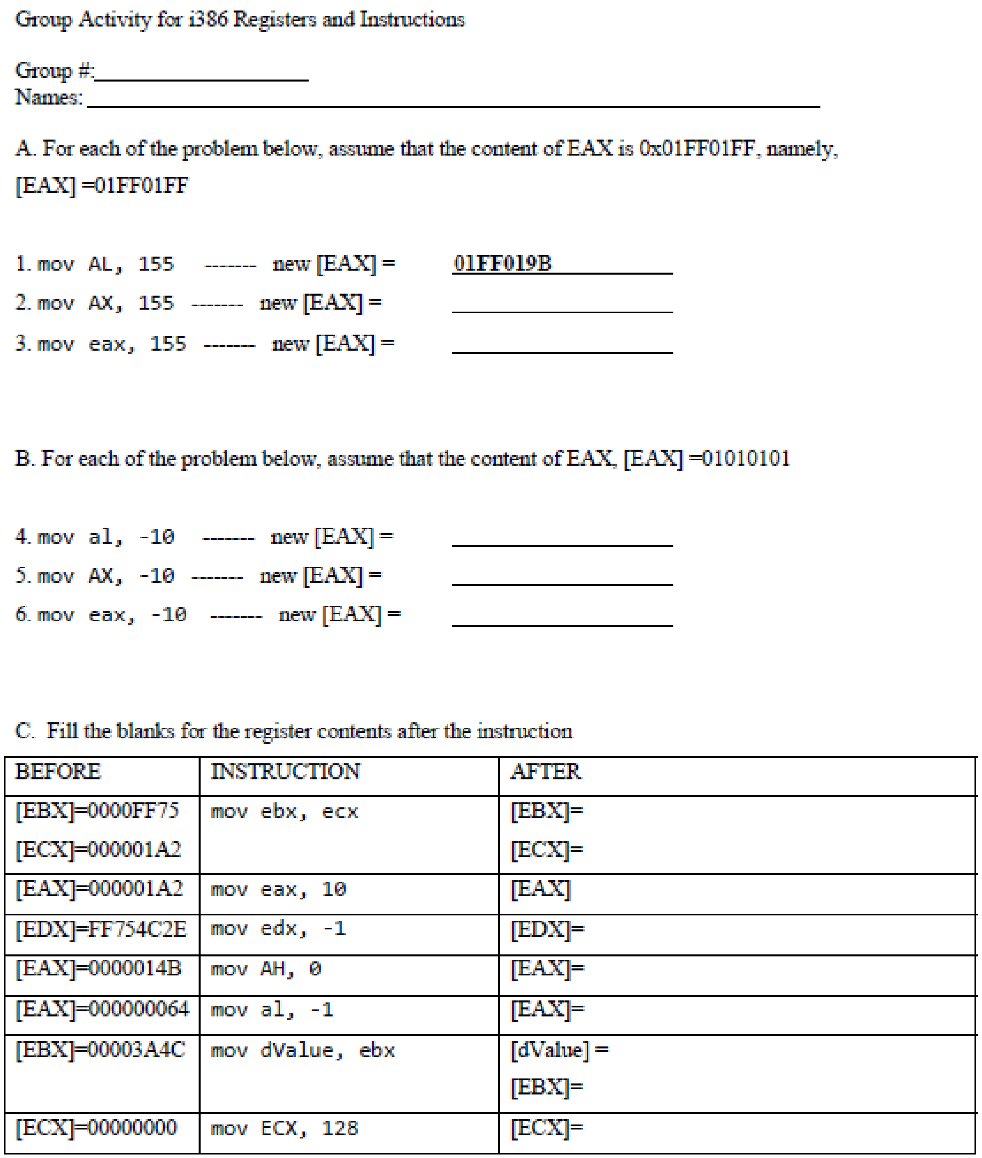


```
mov     al, 155        eax:[                    ]
mov     ax, 155        eax:[                    ]
mov     eax, 155       eax:[                    ]
```

- Further Example
- Before EAX: [01010101]

```
mov al -10 ;    EAX:[                    ]
mov ax, -10;    EAX: [                   ]
mov eax, -10;   EAX: [                  ]
```

Group Activity



Group Activity for i386 Registers and Instructions

Group #:________________
Names: ____

A. For each of the problem below, assume that the content of EAX is 0x01FF01FF, namely, [EAX] =01FF01FF

1. `mov AL, 155` ------- new [EAX] = **01FF019B**
2. `mov AX, 155` ------- new [EAX] = ________________
3. `mov eax, 155` ------- new [EAX] = ________________

B. For each of the problem below, assume that the content of EAX, [EAX] =01010101

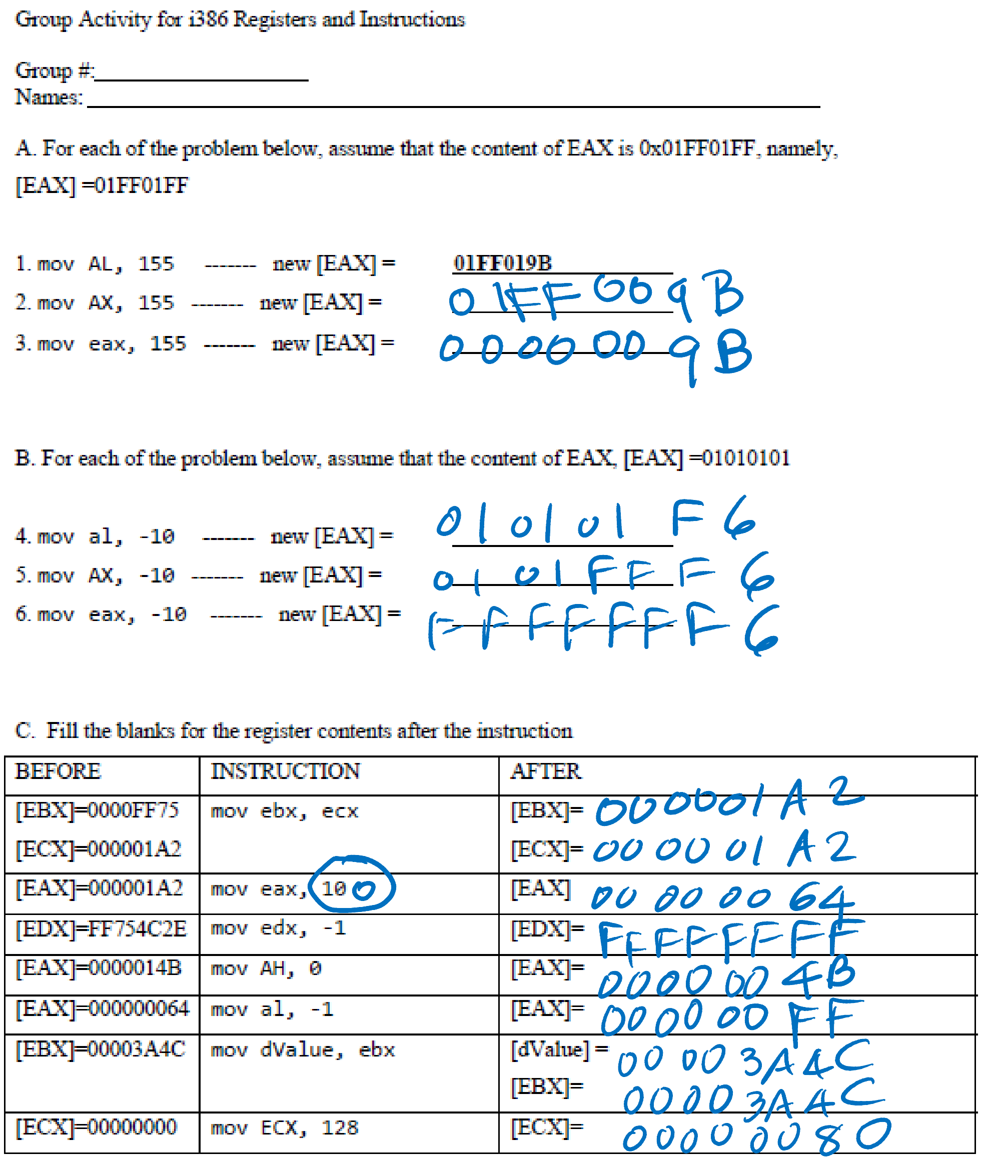
4. `mov al, -10` ------- new [EAX] = ________________
5. `mov AX, -10` ------- new [EAX] = ________________
6. `mov eax, -10` ------- new [EAX] = ________________

C. Fill the blanks for the register contents after the instruction

BEFORE	INSTRUCTION	AFTER
[EBX]=0000FF75 [ECX]=000001A2	`mov ebx, ecx`	[EBX]= [ECX]=
[EAX]=000001A2	`mov eax, 10`	[EAX]
[EDX]=FF754C2E	`mov edx, -1`	[EDX]=
[EAX]=0000014B	`mov AH, 0`	[EAX]=
[EAX]=000000064	`mov al, -1`	[EAX]=
[EBX]=00003A4C	`mov dValue, ebx`	[dValue] = [EBX]=
[ECX]=00000000	`mov ECX, 128`	[ECX]=

- Do first manually (Oct 14)
- Do then by Assembly coding (Today)

Group Activity



Group Activity for i386 Registers and Instructions

Group #:________________
Names: ____

A. For each of the problem below, assume that the content of EAX is 0x01FF01FF, namely, [EAX] =01FF01FF

1. `mov AL, 155` ------- new [EAX] = **01FF019B**
2. `mov AX, 155` ------- new [EAX] = 01FF009B
3. `mov eax, 155` ------- new [EAX] = 0000009B

B. For each of the problem below, assume that the content of EAX, [EAX] =01010101

4. `mov al, -10` ------- new [EAX] = 010101F6
5. `mov AX, -10` ------- new [EAX] = 0101FFF6
6. `mov eax, -10` ------- new [EAX] = FFFFFFF6

C. Fill the blanks for the register contents after the instruction

BEFORE	INSTRUCTION	AFTER
[EBX]=0000FF75 [ECX]=000001A2	`mov ebx, ecx`	[EBX]= 000001A2 [ECX]= 000001A2
[EAX]=000001A2	`mov eax, 100`	[EAX] 00000064
[EDX]=FF754C2E	`mov edx, -1`	[EDX]= FFFFFFFF
[EAX]=0000014B	`mov AH, 0`	[EAX]= 0000004B
[EAX]=000000064	`mov al, -1`	[EAX]= 000000FF
[EBX]=00003A4C	`mov dValue, ebx`	[dValue] = 00003A4C [EBX]= 00003A4C
[ECX]=00000000	`mov ECX, 128`	[ECX]= 00000080

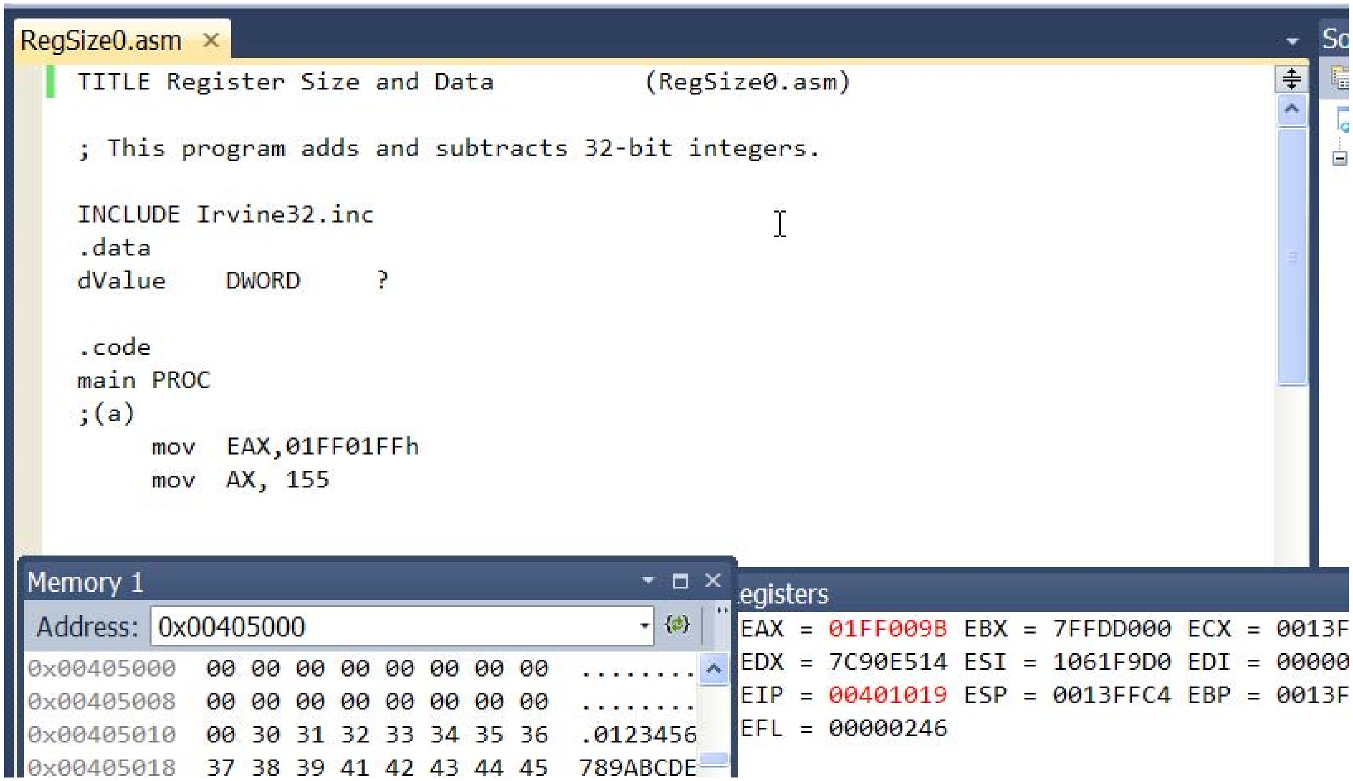
- Do first manually
- Do then by Assembly coding

Class activity ---- ASM code for verification

Write a code and debug to validate the manual execution of the problems below.

Capture the register or memory screen and paste for each of the problems.

- #1. mov AL, 155 ; with [EAX]=01FF01FF



```
TITLE Register Size and Data          (RegSize0.asm)

; This program adds and subtracts 32-bit integers.

INCLUDE Irvine32.inc
.data
dValue    DWORD     ?

.code
main PROC
;(a)
     mov  EAX,01FF01FFh
     mov  AX, 155
```

```
0x00405000  00 00 00 00 00 00 00 00  ........
0x00405008  00 00 00 00 00 00 00 00  ........
0x00405010  00 30 31 32 33 34 35 36  .0123456
0x00405018  37 38 39 41 42 43 44 45  789ABCDE
```

```
EAX = 01FF009B EBX = 7FFDD000 ECX = 0013F
EDX = 7C90E514 ESI = 1061F9D0 EDI = 00000
EIP = 00401019 ESP = 0013FFC4 EBP = 0013F
EFL = 00000246
```

386 Instruction Set

- 9 Operation Categories
  - Data Transfer
  - Arithmetic
  - Shift/Rotate
  - String Manipulation
  - Bit Manipulation
  - Control Transfer
  - High Level Language Support
  - Operating System Support
  - Processor Control
- Number of operands: 0, 1, 2, or 3

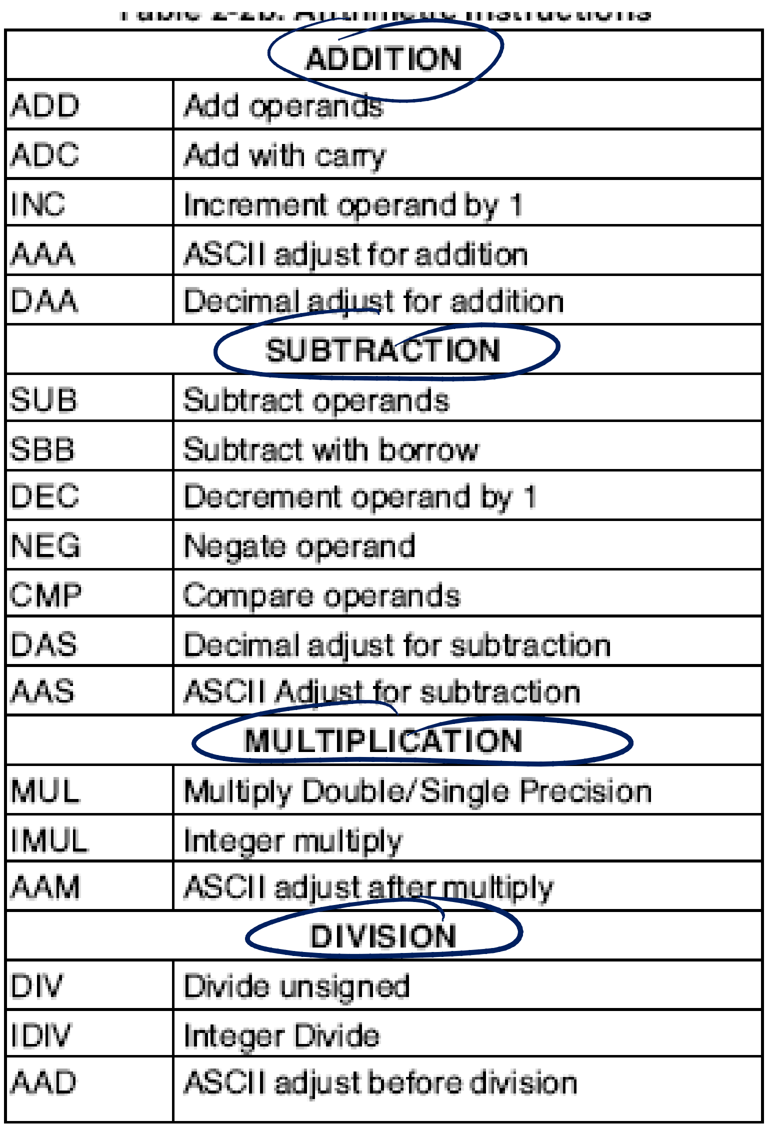


Table 2-2b. Arithmetic Instructions

ADDITION	
ADD	Add operands
ADC	Add with carry
INC	Increment operand by 1
AAA	ASCII adjust for addition
DAA	Decimal adjust for addition
SUBTRACTION	
SUB	Subtract operands
SBB	Subtract with borrow
DEC	Decrement operand by 1
NEG	Negate operand
CMP	Compare operands
DAS	Decimal adjust for subtraction
AAS	ASCII Adjust for subtraction
MULTIPLICATION	
MUL	Multiply Double/Single Precision
IMUL	Integer multiply
AAM	ASCII adjust after multiply
DIVISION	
DIV	Divide unsigned
IDIV	Integer Divide
AAD	ASCII adjust before division

Data movement Instructions

- MOV (Move)
  - transfers a byte, word, or doubleword from the source operand to the destination operand: R→ M, M → R, R→ R, I→R, I→ M
  - The MOV instruction cannot move M→M
  - M→M via MOVS (string)

mem

Reg.

"Immediate"

⇒ a number

- MOVZX (Move with Zero-Extended)
- MOVSX (Move with Sign-Extended)
- XCHG (Exchange)
  - swaps the contents of two operands.
  - swap two byte operands, two word operands, or two doubleword operands.
  - The operands for the XCHG instruction may be two register operands, or a register operand with a memory operand.

R, R

R, M

MOVZX (Before: EAX= [1111FFFF])

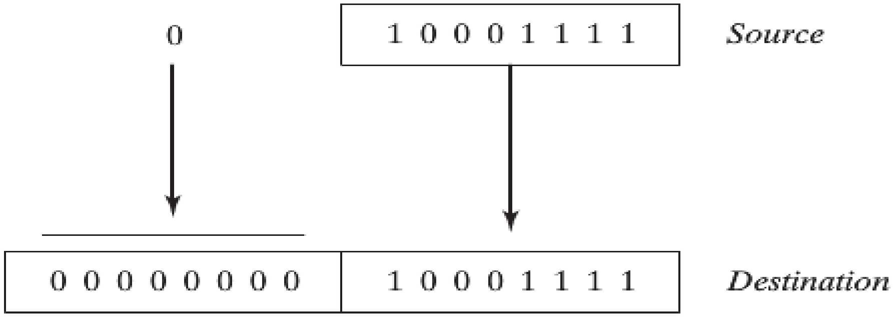
- MOVZX

mov   AL, 8Fh

movzx AX, AL

- After [EAX] = 1111008F

Using MOVZX to copy a byte into a 16-bit destination.



NOTE:**movzx** can extend to 32-bit destination too.

movzx EAX, AL   ; [EAX]=0000008F

movzx EAX, AX   ; [EAX]=0000008F

MOVSX (Before: EAX= [1111FFFF])

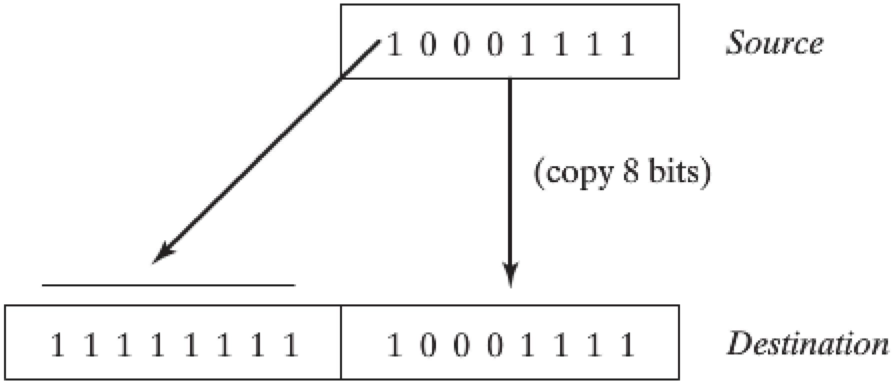
- MOVSX

mov AL, 8Fh

movsx, AX, AL

- After[EAX]= 1111FFF8

Using MOVSX to copy a byte into a 16-bit destination.



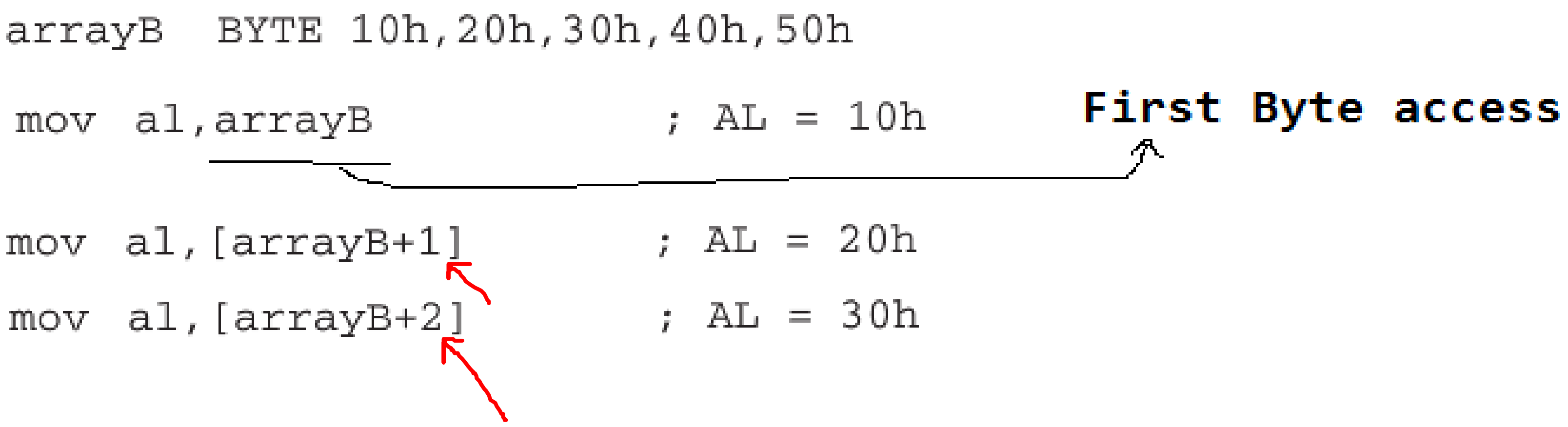
NOTE:**movsx** can extend to 32-bit destination too.

movsx EAX, AL ; [EAX]=FFFFFF8F

movsx EAX, AX ; [EAX]=FFFFFF8F

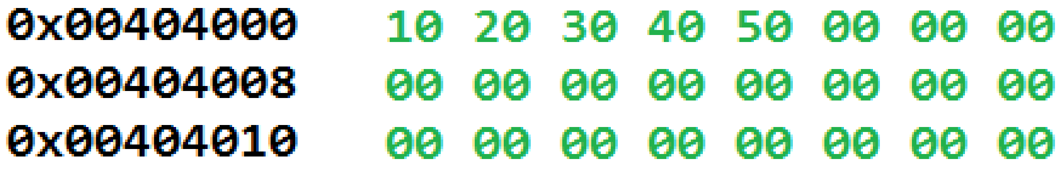
Direct-Offset Operands

- Add displacement to the name of a variable
- Accessing memory locations that may not have explicit labels
- BYTE Case [AL]



```
arrayB  BYTE 10h,20h,30h,40h,50h

 mov  al,arrayB             ; AL = 10h

mov  al,[arrayB+1]          ; AL = 20h

mov  al,[arrayB+2]          ; AL = 30h
```

First Byte access

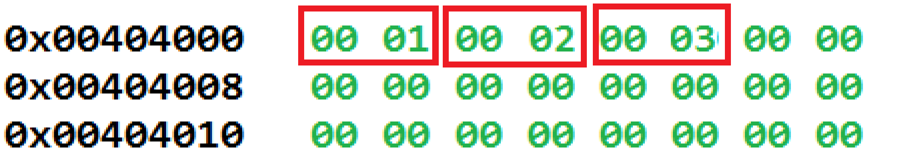


```
0x00404000   10 20 30 40 50 00 00 00
0x00404008   00 00 00 00 00 00 00 00
0x00404010   00 00 00 00 00 00 00 00
```

Direct-Offset Operands

- WORD case [AX]

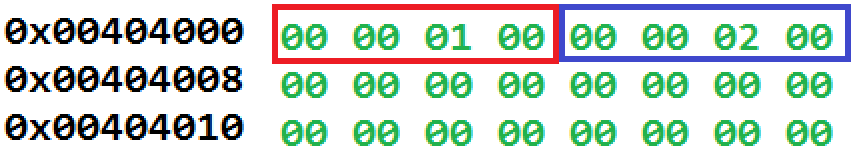
```
.data
arrayW WORD 100h,200h,300h
```



```
0x00404000   00 01 00 02 00 03 00 00
0x00404008   00 00 00 00 00 00 00 00
0x00404010   00 00 00 00 00 00 00 00
```

```
.code
mov  ax,arrayW                    ; AX = 100h
mov  ax,[arrayW+2]                ; AX = 200h
```

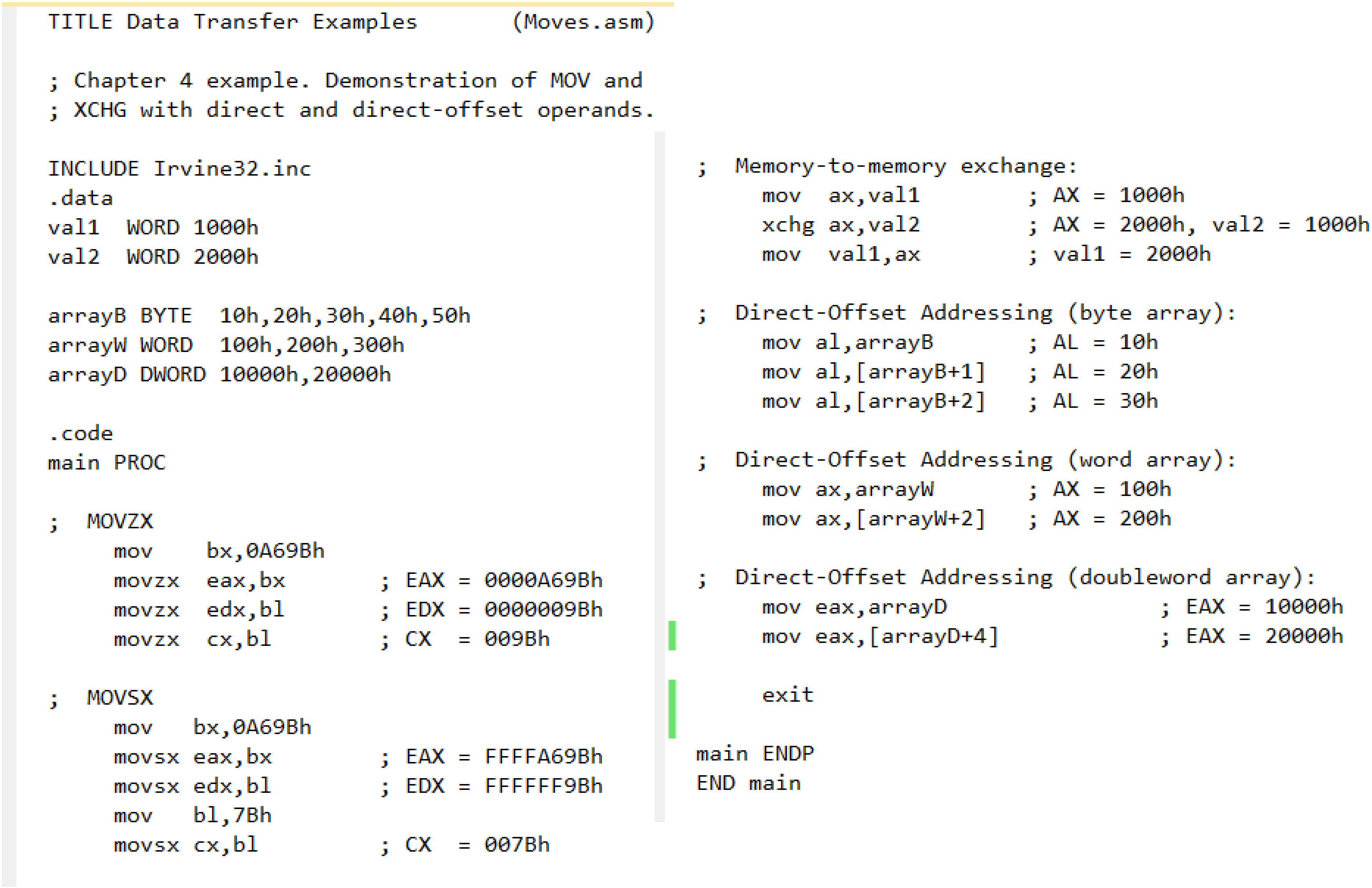
- DWORD case [EAX]



```
0x00404000   00 00 01 00 00 00 02 00
0x00404008   00 00 00 00 00 00 00 00
0x00404010   00 00 00 00 00 00 00 00
```

```
.data
arrayD DWORD 10000h,20000h
.code
mov  eax,arrayD                   ; EAX = 10000h
mov  eax,[arrayD+4]               ; EAX = 20000h
```

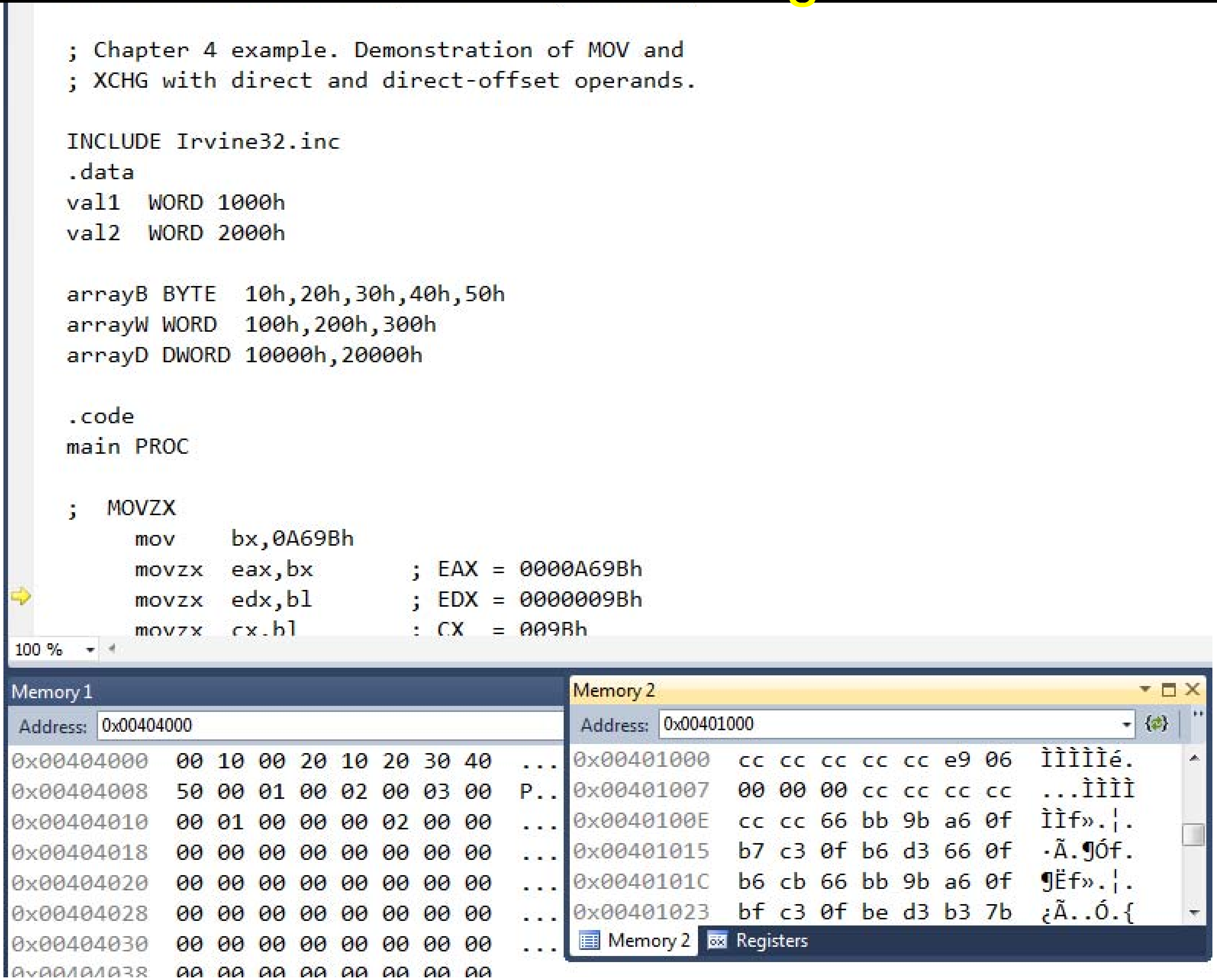
Example Code /ch04/moves.asm



```
TITLE Data Transfer Examples        (Moves.asm)

; Chapter 4 example. Demonstration of MOV and
; XCHG with direct and direct-offset operands.

INCLUDE Irvine32.inc
.data
val1  WORD 1000h
val2  WORD 2000h

arrayB BYTE  10h,20h,30h,40h,50h
arrayW WORD  100h,200h,300h
arrayD DWORD 10000h,20000h

.code
main PROC

;  MOVZX
     mov     bx,0A69Bh
     movzx   eax,bx         ; EAX = 0000A69Bh
     movzx   edx,bl         ; EDX = 0000009Bh
     movzx   cx,bl          ; CX  = 009Bh

;  MOVSX
     mov    bx,0A69Bh
     movsx eax,bx           ; EAX = FFFFA69Bh
     movsx edx,bl           ; EDX = FFFFFF9Bh
     mov    bl,7Bh
     movsx cx,bl            ; CX  = 007Bh
```

```
;  Memory-to-memory exchange:
     mov   ax,val1          ; AX = 1000h
     xchg ax,val2           ; AX = 2000h, val2 = 1000h
     mov   val1,ax          ; val1 = 2000h

;  Direct-Offset Addressing (byte array):
     mov al,arrayB          ; AL = 10h
     mov al,[arrayB+1]      ; AL = 20h
     mov al,[arrayB+2]      ; AL = 30h

;  Direct-Offset Addressing (word array):
     mov ax,arrayW          ; AX = 100h
     mov ax,[arrayW+2]      ; AX = 200h

;  Direct-Offset Addressing (doubleword array):
     mov eax,arrayD                   ; EAX = 10000h
     mov eax,[arrayD+4]               ; EAX = 20000h

     exit

main ENDP
END main
```

Data and Code Segment



```
; Chapter 4 example. Demonstration of MOV and
; XCHG with direct and direct-offset operands.

INCLUDE Irvine32.inc
.data
val1  WORD 1000h
val2  WORD 2000h

arrayB BYTE  10h,20h,30h,40h,50h
arrayW WORD  100h,200h,300h
arrayD DWORD 10000h,20000h

.code
main PROC

;  MOVZX
     mov    bx,0A69Bh
     movzx  eax,bx        ; EAX = 0000A69Bh
     movzx  edx,bl        ; EDX = 0000009Bh
     movzx  cx,bl         ; CX  = 009Bh
```

Memory 1

Address: 0x00404000

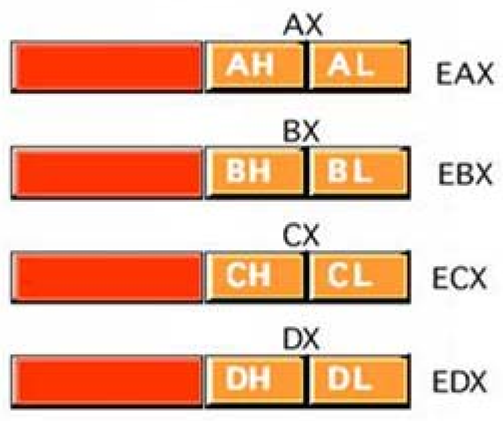
```
0x00404000  00 10 00 20 10 20 30 40  ...
0x00404008  50 00 01 00 02 00 03 00  P..
0x00404010  00 01 00 00 00 02 00 00  ...
0x00404018  00 00 00 00 00 00 00 00  ...
0x00404020  00 00 00 00 00 00 00 00  ...
0x00404028  00 00 00 00 00 00 00 00  ...
0x00404030  00 00 00 00 00 00 00 00  ...
0x00404038  00 00 00 00 00 00 00 00
```

Memory 2

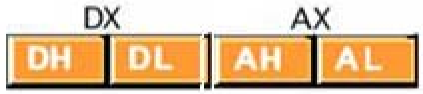
Address: 0x00401000

```
0x00401000  cc cc cc cc cc e9 06  ÌÌÌÌÌé.
0x00401007  00 00 00 cc cc cc cc  ...ÌÌÌÌ
0x0040100E  cc cc 66 bb 9b a6 0f  ÌÌf».¦.
0x00401015  b7 c3 0f b6 d3 66 0f  ·Ã.¶Óf.
0x0040101C  b6 cb 66 bb 9b a6 0f  ¶Ëf».¦.
0x00401023  bf c3 0f be d3 b3 7b  ¿Ã..Ó.{
```

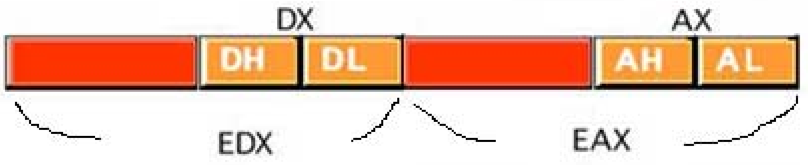
Data type Conversion Instructions



- CBW (Convert Byte to Word)
  - **extends the sign of the byte in register AL throughout AX.**
- CWDE (Convert Word to Doubleword Extended)
  - **extends the sign of the word in register AX throughout EAX.**
- CWD (Convert Word to Doubleword)
  - **extends the sign of the word in register AX throughout register DX**
  - can be used to produce a doubleword dividend from a word before a word division

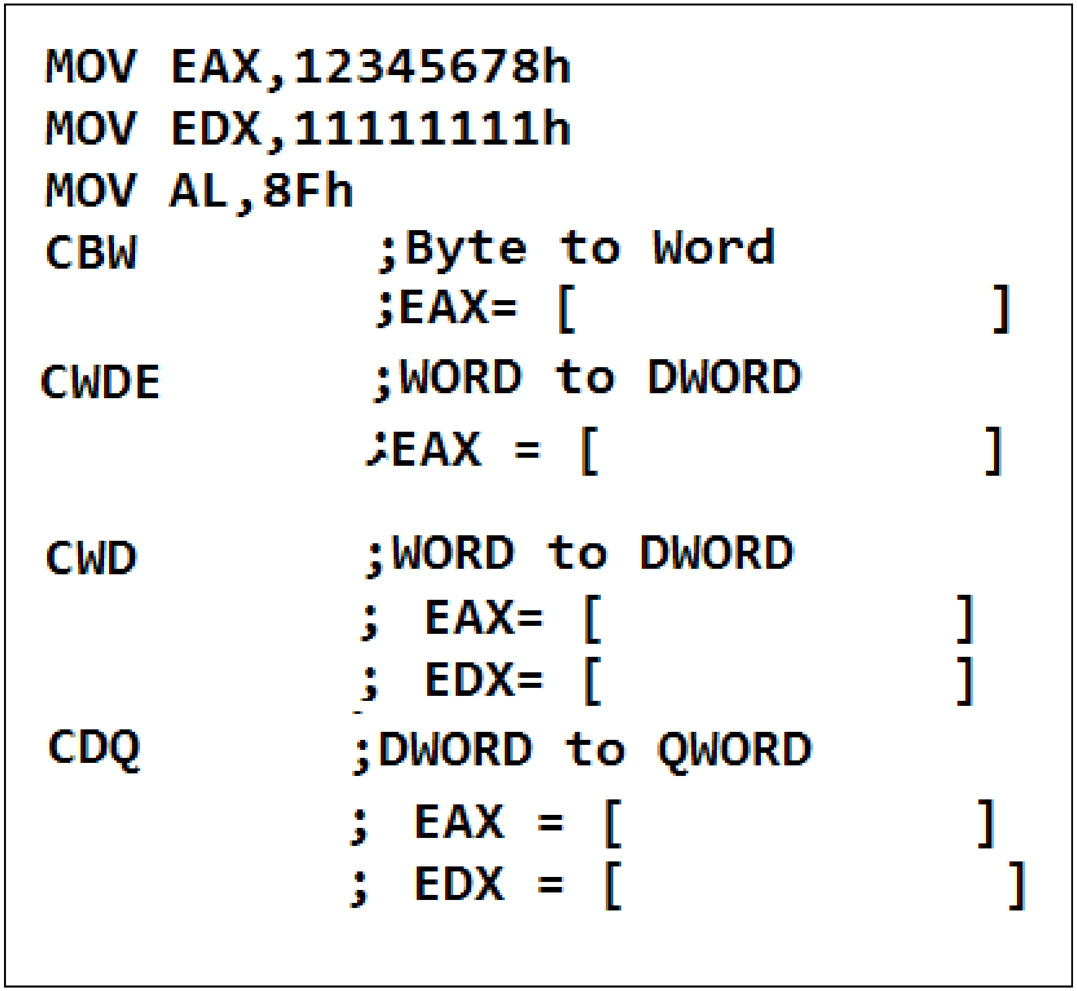


- CDQ (Convert Doubleword to Quad-Word)
  - **extends the sign of the doubleword in EAX throughout EDX.**
  - can be used to produce a quad-word dividend from a doubleword before doubleword division.

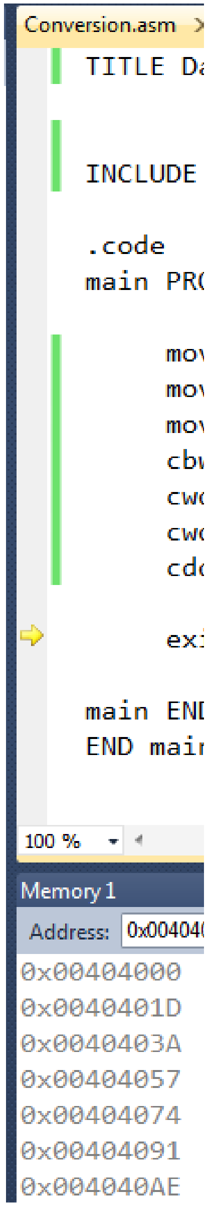


Data type Conversion Instructions – Practice

- CBW (Convert Byte to Word)
  - **extends the sign of the byte in register AL throughout AX.**
- CWDE (Convert Word to Doubleword Extended)
  - **extends the sign of the word in register AX throughout EAX**.
- CWD (Convert Word to Doubleword)
  - **extends the sign of the word in register AX throughout register DX**
  - can be used to produce a doubleword dividend from a word before a word division
- CDQ (Convert Doubleword to Quad-Word)
  - **extends the sign of the doubleword in EAX throughout EDX.**
  - can be used to produce a quad-word dividend from a doubleword before doubleword division.



```
MOV EAX,12345678h
MOV EDX,11111111h
MOV AL,8Fh
CBW           ;Byte to Word
              ;EAX= [              ]
CWDE          ;WORD to DWORD
              ;EAX = [             ]

CWD           ;WORD to DWORD
              ; EAX= [             ]
              ; EDX= [             ]
CDQ           ;DWORD to QWORD
              ; EAX = [            ]
              ; EDX = [             ]
```



```
Conversion.asm
TITLE D

INCLUDE

.code
main PR

      mo
      mo
      mo
      cb
      cw
      cw
      cd

      ex

main EN
END mai
```

```
0x00404000
0x0040401D
0x0040403A
0x00404057
0x00404074
0x00404091
0x004040AE
```

Group #:________________

Names: ____

1. **Execute each line manually (this will prepare your for Quiz #2)**

2. Then write a code which contains all 8 lines shown below, and debug (by F10 key) to find out the contents of the registers as we execute each line at a time.

```
mov   EAX, 12345678h
mov   EBX, 0FFFFFFFFh
mov   ECX, 11223344h
mov   EDX, 0AABBCCDDh
mov   BX, 0A69Bh           ;EBX= [                  ]
movzx EDX, BL              ;EDX= [                  ]
movzx EAX,BX               ;EAX= [                  ]
movzx CX,BL                ;ECX= [                  ]
mov   BX, 0A69Bh           ;EBX = [                 ]
movsx EAX,BX               ;EAX= [                  ]
movsx EDX,BL               ;EDX = [                 ]
mov   BL,70                ;EBX= [                  ]
movsx CX, BL               ; ECX = [                ]
mov   EAX, 12345678h
mov   EDX, 11111111h
mov   AL, 8Fh              ;EAX =  [                ]
CBW                        ;EAX= [                  ]
CWDE                       ;EAX  = [                ]
CWD                        ;EAX = [        ]; EDX = [           ]
CDQ                        ;EAX = [        ] ;EDX = [           ]
```

ode

AL throughout

d Extended)

AX throughout

)

AX throughout

dividend from a

Word)

EAX throughout

ividend from a

n.