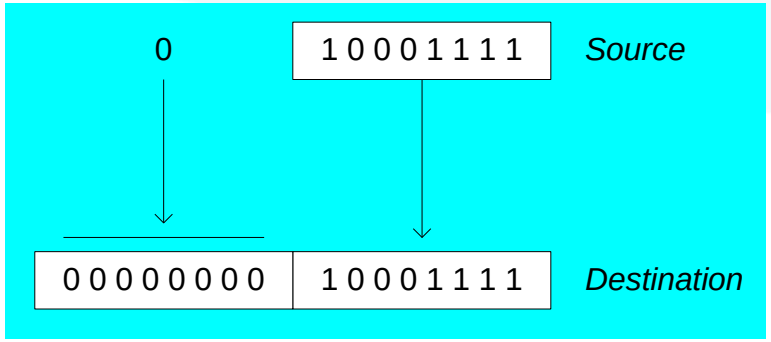
جَامعة المَنارة
MANARA UNIVERSITY

كلية الهندسة المعلوماتية
بنيان حواسيب 2
الفصل الأول 2024-2025
المحاضرة الرابعة +الخامسة

د كندة أبو قاسم

Zero Extension

MOVZX instruction fills (extends) the upper half of the destination with zeros.



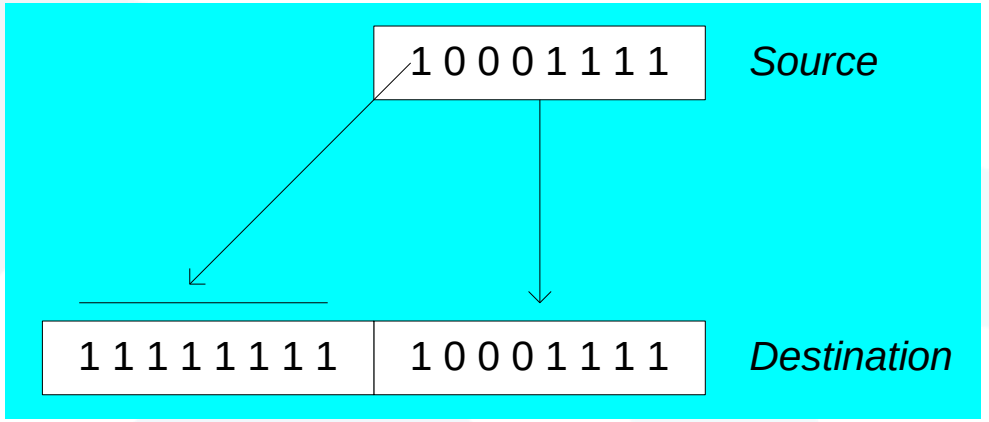

```
mov bl,10001111b
movzx ax,bl                    ; zero-extension
```

The destination must be a register.

ملاحظة يجب أن يكون الهدف مسجل

Sign Extension

The MOVSX instruction fills the upper half of the destination with a copy of the source operand's sign bit.




```
mov bl,10001111b
movsx ax,bl                    ; sign extension
```

XCHG Instruction

XCHG exchanges the values of two operands. At least one operand must be a register. No immediate operands are permitted.

```
.data
var1 WORD 1000h
var2 WORD 2000h
.code
xchg ax,bx                  ; exchange 16-bit regs
xchg ah,al                  ; exchange 8-bit regs
xchg var1,bx                ; exchange mem, reg
xchg eax,ebx                ; exchange 32-bit regs
```

جامعة المنارة
MANARA UNIVERSITY

تعليمة XCHG

تسمح بتبديل محتوى سجل بسجل آخر ولا تسمح بتبديل قيمة فورية

ملاحظة هامة: لا تسمح بتبديل محتوى ذاكرة بمحتوى موق ذاكر آخر

مثال

```
xchg var1,var2    ;
error: two memory operands
```

تعليمات نقل البيانات
1. Data Transfer Instructions

Mnemonics: MOV, XCHG, PUSH, POP, IN, OUT …

PUSH reg16/ mem	
PUSH reg16	(SP) ← (SP) – 2 MA_S = (SS) x 16_{10} + SP (MA_S ; MA_S + 1) ← (reg16)
PUSH mem	(SP) ← (SP) – 2 MA_S = (SS) x 16_{10} + SP (MA_S ; MA_S + 1) ← (mem)

POP reg16/ mem	
POP reg16	MA_S = (SS) x 16_{10} + SP (reg16) ← (MA_S ; MA_S + 1) (SP) ← (SP) + 2
POP mem	MA_S = (SS) x 16_{10} + SP (mem) ← (MA_S ; MA_S + 1) (SP) ← (SP) + 2

مثال :

بفرض AX =3FC4
والمسجلات SS = 0050 , SP = 1234
PUSH AX;
الحل : push 21ABH
نحسب العنوان الفزيائي لذاكرة STACK
SS:SP
00500
1234 +
01734

دفع القيمة 21AB في ذاكرة المسجل تحت العنوان
01734

1- دفع مسجل الأعلام في ذاكرة STACK

- PUSHFD and POPFD
- **pushfw** and **popfw** for 16-bit flags (FLAGS)

```
pushfd (push 32-bit flags)
popfd  (pop 32-bit flags)
```

2- تعليمة دفع 8 مسجلات 16 bit في Stack

PUSHA and POPA
AX, CX, DX, BX, SP, BP, SI, DI

3- تعليمة دفع 8 مسجلات 32 bit في ذاكرة Stack

PUSHAD pushes the 32-bit general-purpose registers on the stack

order: EAX, ECX, EDX, EBX, ESP, EBP, ESI, EDI

POPAD pops the same registers off the stack in reverse order

مجموعة التعليمات Instruction Set

Mnemonics: MOV, XCHG, PUSH, POP, IN, OUT …

تعليمات نقل البيانات من وحدة دخل الى مسجل

IN A, [DX]	
IN AL, [DX]	$PORT_{addr}$ = (DX) (AL) ← (PORT)
IN AX, [DX]	$PORT_{addr}$ = (DX) (AX) ← (PORT)
IN A, addr8	
IN AL, addr8	(AL) ← (addr8)
IN AX, addr8	(AX) ← (addr8)

تعليمات نقل البيانات من مسجل الى وحدة خرج

OUT [DX], A	
OUT [DX], AL	$PORT_{addr}$ = (DX) (PORT) ← (AL)
OUT [DX], AX	$PORT_{addr}$ = (DX) (PORT) ← (AX)
OUT addr8, A	
OUT addr8, AL	(addr8) ← (AL)
OUT addr8, AX	(addr8) ← (AX)

التعليمات الحسابية
Arithmetic Instructions

Mnemonics: ADD, ADC, SUB, SBB, INC, DEC, MUL, DIV, CMP

تعليمة الجمع ADD

ADD reg/mem, data
ADD reg, data
ADD mem, data
	(reg) ← (reg)+ data
	(mem) ← (mem)+data

ADD A, data
 	
ADD AL, data8
ADD AX, data16
	
	(AL) ← (AL) + data8
	(AX) ← (AX) +data16

Flags Affected by Arithmetic
Zero
Sign
Carry
Overflow

مثال

```
.data
var1 DWORD 10000h
var2 DWORD 20000h
.code                          ; ---EAX---
   mov eax,var1                ; 00010000h
   add eax,var2                ; 00030000h
   add ax,0FFFFh               ; 0003FFFFh
   add eax,1                   ; 00040000h
   sub ax,1                    ; 0004FFFFh
```

Arithmetic Instructions التعليمات الحسابية

Mnemonics: ADD, ADC, SUB, SBB, INC, DEC, MUL, DIV, CMP…

ADC reg2/ mem, reg1/mem	
ADC reg2, reg1 ADC reg2, mem ADC mem, reg1	(reg2) ← (reg1) + (reg2)+CF (reg2) ← (reg2) + (mem)+CF (mem) ← (mem)+(reg1)+CF
ADC reg/mem, data	
ADC reg, data ADC mem, data	(reg) ← (reg)+ data+CF (mem) ← (mem)+data+CF
ADDC A, data	
ADD AL, data8 ADD AX, data16	(AL) ← (AL) + data8+CF (AX) ← (AX) +data16+CF

Mnemonics: ADD, ADC, SUB, SBB, INC, DEC, MUL, DIV, CMP…

SUB reg2/ mem, reg1/mem	

SUB reg2, reg1
SUB reg2, mem
SUB mem, reg1
	(reg2) ← (reg1) - (reg2)
	(reg2) ← (reg2) - (mem)
	(mem) ← (mem) - (reg1)

SUB reg/mem, data	
SUB reg, data SUB mem, data	(reg) ← (reg) - data (mem) ← (mem) - data
SUB A, data	
SUB AL, data8 SUB AX, data16	(AL) ← (AL) - data8 (AX) ← (AX) - data16

تعليمة الطرح مسجل من مسجل آخر
أو طرح محتوى ذاكرة من مسجل

Mnemonics: ADD, ADC, SUB, SBB, INC, DEC, MUL, DIV, CMP…

SBB reg2/ mem, reg1/mem	
SBB reg2, reg1 SBB reg2, mem SBB mem, reg1	(reg2) ← (reg1) - (reg2) - CF (reg2) ← (reg2) - (mem)- CF (mem) ← (mem) - (reg1) –CF
SBB reg/mem, data	
SBB reg, data SBB mem, data	(reg) ← (reg) – data - CF (mem) ← (mem) - data - CF
SBB A, data	
SBB AL, data8 SBB AX, data16	(AL) ← (AL) - data8 - CF (AX) ← (AX) - data16 - CF

Mnemonics: ADD, ADC, SUB, SBB, INC, DEC, MUL, DIV, CMP…

INC reg/ mem	
INC reg8	(reg8) ← (reg8) + 1
INC reg16	(reg16) ← (reg16) + 1
INC mem	(mem) ← (mem) + 1

DEC reg/ mem	
DEC reg8	(reg8) ← (reg8) - 1
DEC reg16	(reg16) ← (reg16) - 1
DEC mem	(mem) ← (mem) - 1

- Add 1, subtract 1 from destination operand
  - operand may be register or memory
- INC *destination*
  - Logic: *destination* ← *destination* + 1
- DEC *destination*
  - Logic: *destination* ← *destination* – 1

```
.data
myByte BYTE 0FFh, 0
.code
    mov al,myByte              ; AL = FFh
    mov ah,[myByte+1]          ; AH = 00h
    dec ah                     ; AH = FFh
    inc al                     ; AL = 00h
    dec ah                     ; Ah = FE
```

```
.data
myWord  WORD 1000h
myDword DWORD 10000000h
.code
    inc myWord                 ; 1001h
    dec myWord                 ; 1000h
    inc myDword                ; 10000001h

    mov ax,00FFh
    inc ax                     ; AX = 0100h
    mov ax,00FFh
    inc al                     ; AX = 0000h
```

NEG (negate) Instruction

جامعة المنارة

مثال اكتب برنامج بلغة Assembly لحل المعادلة

```
Rval = -Xval + (Yval – Zval)
```

حيث أن المتغييرات التالية : 32 bit

```
Rval ,Xval ,Yval , Zval
```

```
.data
valB BYTE -1
valW WORD +32767
.code
    mov al,valB                 ; AL = -1
    neg al                      ; AL = +1
    neg valW                    ; valW = -32767
```

Example

mov al,-128 ; AL = 10000000b

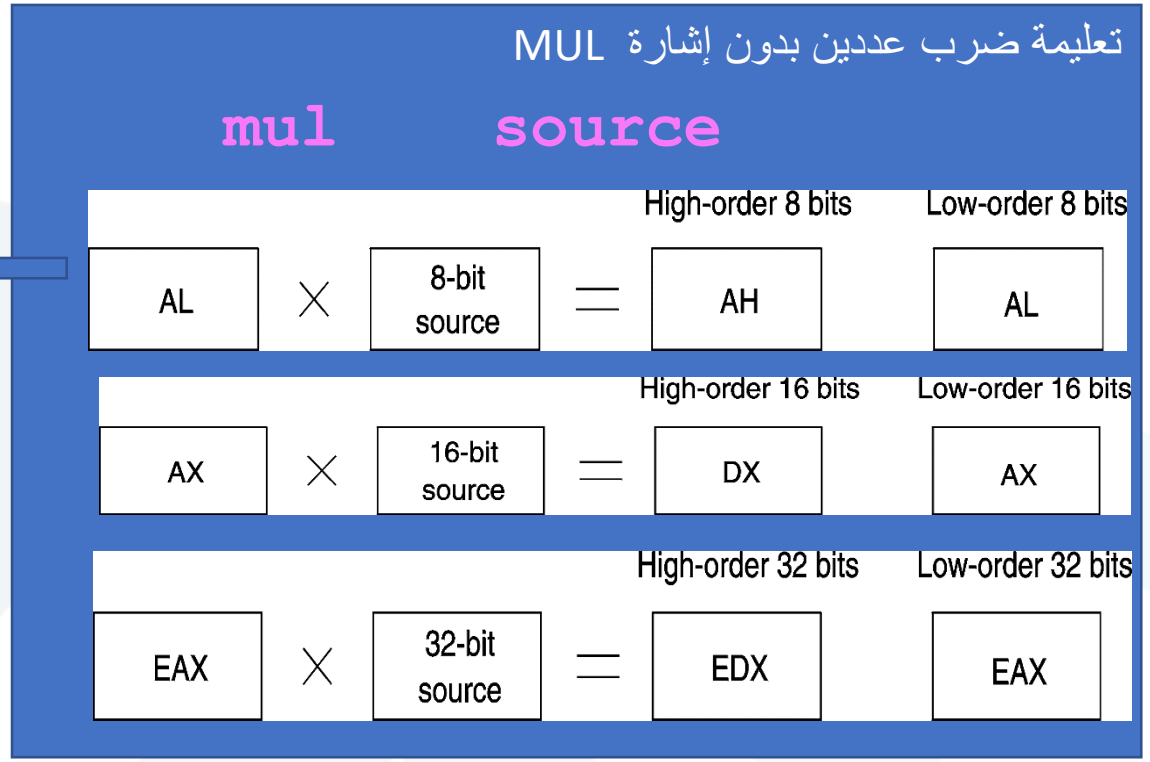
neg al ; AL = 10000000b, OF = 1

```
Rval DWORD ?
Xval DWORD 26
Yval DWORD 30
Zval DWORD 40
.code
    mov eax,Xval
    neg eax                     ; EAX = -26
    mov ebx,Yval
    sub ebx,Zval                ; EBX = -10
    add eax,ebx
    mov Rval,eax                ; -36
```

Mnemonics: ADD, ADC, SUB, SBB, INC, DEC, MUL, DIV, CMP…

MUL reg/ mem	
MUL reg	For byte : (AX) ← (AL) x (reg8) For word : (DX)(AX) ← (AX) x (reg16)
MUL mem	For byte : (AX) ← (AL) x (mem8) For word : (DX)(AX) ← (AX) x (mem16)
IMUL reg/ mem	
IMUL reg	For byte : (AX) ← (AL) x (reg8) For word : (DX)(AX) ← (AX) x (reg16)
IMUL mem	For byte : (AX) ← (AX) x (mem8) For word : (DX)(AX) ← (AX) x (mem16)

تعليمة ضرب عددين بدون إشارة MUL



تعليمة ضرب عددين بإشارة IMUL

Mnemonics: ADD, ADC, SUB, SBB, INC, DEC, MUL, DIV, CMP…

DIV reg/ mem	
DIV reg	**For 16-bit :- 8-bit :** (AL) ← (AX) :- (reg8) Quotient (AH) ← (AX) MOD(reg8) Remainder **For 32-bit :- 16-bit :** (AX) ← (DX)(AX) :- (reg16) Quotient (DX) ← (DX)(AX) MOD(reg16) Remainder
DIV mem	**For 16-bit :- 8-bit :** (AL) ← (AX) :- (mem8) Quotient (AH) ← (AX) MOD(mem8) Remainder **For 32-bit :- 16-bit :** (AX) ← (DX)(AX) :- (mem16) Quotient (DX) ← (DX)(AX) MOD(mem16) Remainder

- The DIV (unsigned divide) instruction performs 8-bit, 16-bit, and 32-bit division on unsigned integers
- A single operand is supplied (register or memory operand), which is assumed to be the divisor
- Instruction formats:

```
DIV r/m8
DIV r/m16
DIV r/m32
```

Default Operands:

Dividend	Divisor	Quotient	Remainder
AX	*r/m8*	AL	AH
DX:AX	*r/m16*	AX	DX
EDX:EAX	*r/m32*	EAX	EDX

DIV Examples

Mnemonics: ADD, ADC, SUB, SBB, INC, DEC, MUL, DIV, CMP…

تعليمة القسمة اذا كان العدد بإشارة IDIV

Divide 8003h by 100h, using 16-bit operands:

```
mov dx,0        ; clear dividend, high
mov ax,8003h    ; dividend, low
mov cx,100h     ; divisor
div cx          ; AX = 0080h, DX = 3
```

Same division, using 32-bit operands

```
mov edx,0       ; clear dividend, high
mov eax,8003h   ; dividend, low
mov ecx,100h    ; divisor
div ecx         ; EAX = 00000080h, DX = 3
```

What will be the hexadecimal values of DX and AX after the following instructions execute? Or, if divide overflow occurs, you can indicate that as your answer:

IDIV reg/ mem	
IDIV reg	**For 16-bit :- 8-bit :** (AL) ← (AX) :- (reg8) Quotient (AH) ← (AX) MOD(reg8) Remainder **For 32-bit :- 16-bit :** (AX) ← (DX)(AX) :- (reg16) Quotient (DX) ← (DX)(AX) MOD(reg16) Remainder
IDIV mem	**For 16-bit :- 8-bit :** (AL) ← (AX) :- (mem8) Quotient (AH) ← (AX) MOD(mem8) Remainder **For 32-bit :- 16-bit :** (AX) ← (DX)(AX) :- (mem16) Quotient (DX) ← (DX)(AX) MOD(mem16) Remainder