University of Diyala
College of engineering
Department of computer Engineering
Second class

microprocessor Programming

Lecture 6

8086 instructions set

Presented by
Lecturer : Abdullah Thair Abdalsatir Al-Obaidi
Department of computer Engineering
University of Diyala

content

Lecture 6
Microprocessor programming

❖8086 Supports 6 Types Of Instructions.

1. Data Transfer Instructions
2. Arithmetic Instructions
3. Logical Instructions
4. String Manipulation Instructions
5. Process Control Instructions
6. Control Transfer Instructions

Lecturer : Abdullah Thair Abdalsatir
Department of computer Engineering

3. Logical Instructions

Mnemonics: AND, OR, XOR, TEST, SHR, SHL, RCR, ROL …

AND reg2/mem, reg1/mem AND reg2, reg1	(reg2) ← (reg2) & (reg1)
AND reg2, mem	(reg2) ← (reg2) & (mem)
AND mem, reg1	(mem) ← (mem) & (reg1)
AND A, data AND AL, data8	(AL) ← (AL) & data8
AND AX, data16	(AX) ← (AX) & data16
AND reg/mem, data AND reg, data	(reg) ← (reg) & data
AND mem, data	(mem) ← (mem) & data

Logical AND between all bits of two operands. Result is stored in operand1.

These rules apply:

1 AND 1 = 1
1 AND 0 = 0
0 AND 1 = 0
0 AND 0 = 0

Lecturer : Abdullah Thair Abdalsatir Al-Obaidi
Department of computer Engineering

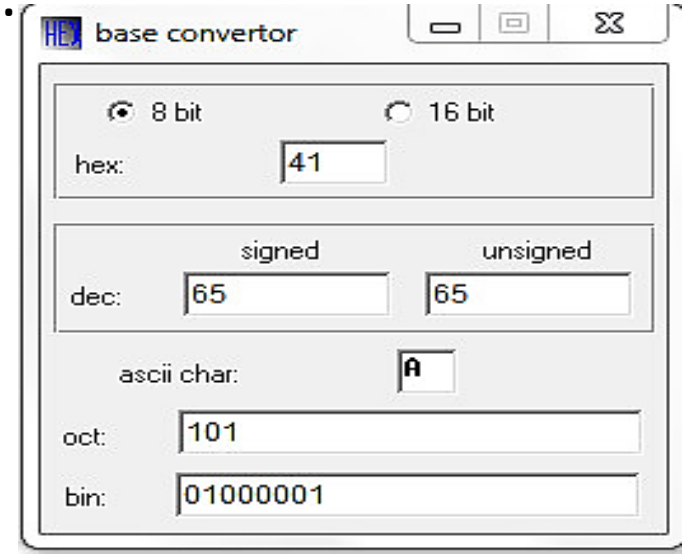
Mnemonics: AND, OR, XOR, TEST, SHR, SHL, RCR, ROL …

OR reg2/mem, reg1/mem OR reg2, reg1	(reg2) ← (reg2) | (reg1)
OR reg2, mem	(reg2) ← (reg2) | (mem)
OR mem, reg1	(mem) ← (mem) | (reg1)

OR reg/mem, data OR reg, data	(reg) ← (reg) | data
OR mem, data	(mem) ← (mem) | data

OR A, data OR AL, data8	(AL) ← (AL) | data8
OR AX, data16	(AX) ← (AX) | data16

Logical OR between all bits of two operands. Result is stored in first operand.



These rules apply:

1 OR 1 = 1
1 OR 0 = 1
0 OR 1 = 1
0 OR 0 = 0

Example:

```
MOV AL, 'A'       ; AL = 01000001b
OR AL, 00100000b  ; AL = 01100001b  ('a')
RET
```

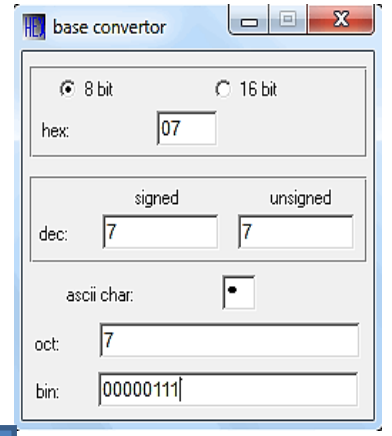
Mnemonics: AND, OR, XOR, TEST, SHR, SHL, RCR, ROL …

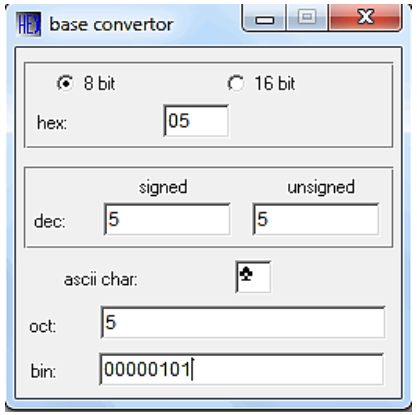
XOR reg2/mem, reg1/mem	
XOR reg2, reg1	(reg2) ← (reg2) ^ (reg1)
XOR reg2, mem	(reg2) ← (reg2) ^ (mem)
XOR mem, reg1	(mem) ← (mem) ^ (reg1)
XOR reg/mem, data	
XOR reg, data	(reg) ← (reg) ^ data
XOR mem, data	(mem) ← (mem) ^ data
XOR A, data	
XOR AL, data8	(AL) ← (AL) ^ data8
XOR AX, data16	(AX) ← (AX) ^ data16

Logical XOR (Exclusive OR) between all bits of two operands. Result is stored in first operand.

These rules apply:

1 XOR 1 = 0
1 XOR 0 = 1
0 XOR 1 = 1
0 XOR 0 = 0





Example:

```
MOV AL, 00000111b
XOR AL, 00000010b    ; AL = 00000101b
RET
```

Mnemonics: AND, OR, XOR, TEST, SHR, SHL, RCR, ROL …

TEST reg2/mem, reg1/mem TEST reg2, reg1 TEST reg2, mem TEST mem, reg1	 Modify flags ← (reg2) & (reg1) Modify flags ← (reg2) & (mem) Modify flags ← (mem) & (reg1)
TEST reg/mem, data TEST reg, data TEST mem, data	 Modify flags ← (reg) & data Modify flags ← (mem) & data
TEST A, data TEST AL, data8 TEST AX, data16	 Modify flags ← (AL) & data8 Modify flags ← (AX) & data16

Logical AND between all bits of two operands for flags only These flags are effected: ZF, SF, PF. Result is not stored anywhere.

These rules apply:

1 AND 1 = 1
1 AND 0 = 0
0 AND 1 = 0
0 AND 0 = 0

Example:

```
MOV AL, 00000101b
TEST AL, 1        ; ZF = 0.
TEST AL, 10b      ; ZF = 1.
RET
```

Mnemonics: AND, OR, XOR, TEST, SHR, SHL, RCR, ROL …

Shift operand1 Right. The number of shifts is set by operand2.

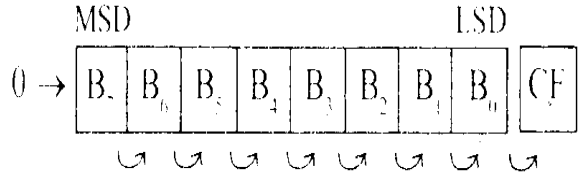
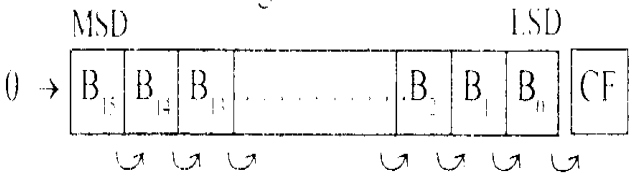
Algorithm:

Shift all bits right, the bit that goes off is set to CF. Zero bit is inserted to the left-most position.

Example:

```
MOV AL, 00000111b
SHR AL, 1        ; AL = 00000011b,  CF=1.
RET
```

OF=0 if first operand keeps original sign.

SHR reg/mem SHR reg i) SHR reg, 1 ii) SHR reg, CL SHR mem i) SHR mem, 1 ii) SHR mem, CL	$CF \leftarrow B_{LSD} ; B_n \leftarrow B_{n+1} ; B_{MSD} \leftarrow 0$ reg 8 / mem 8  reg 16 / mem 16 

Lecturer : Abdullah Thair Abdalsatir Al-Obaidi
Department of computer Engineering

AND, OR, XOR, TEST, SHR, SHL, RCR, ROL …

Shift operand1 Left. The number of shifts is set by operand2.

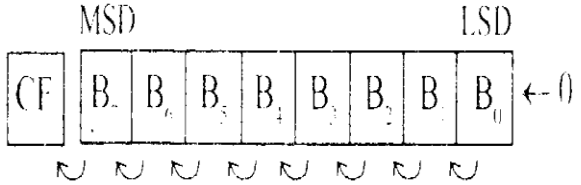
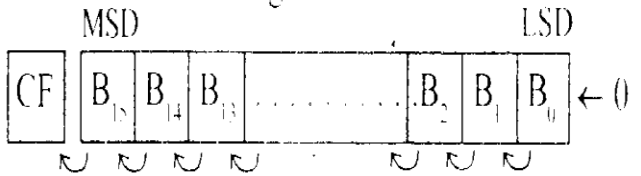
Algorithm:

Shift all bits left, the bit that goes off is set to CF. Zero bit is inserted to the right-most position.

Example :

```
MOV AL, 11100000b
SHL AL, 1    ; AL = 11000000b,  CF=1.
RET
```

OF=0 if first operand keeps original sign. Back to Top

SHL reg/mem or SAL reg/mem SHL reg or SAL reg i) SHL reg, 1 or SAL reg, 1 ii) SHL reg, CL or SAL reg, CL SHL mem or SAL mem i) SHL mem, 1 or SAL mem, 1 ii) SHL mem, CL or SAL mem, CL	$CF \leftarrow B_{MSD} ; B_{n-1} \leftarrow B_n ; B_{LSD} \leftarrow 0$ reg 8 / mem 8  reg 16 / mem 16 

Lecturer : Abdullah Thair Abdalsatir Al-Obaidi
Department of computer Engineering

Mnemonics: AND, OR, XOR, TEST, SHR, SHL, RCR, ROL …

Rotate operand1 right through Carry Flag. The number of rotates is set by operand2.

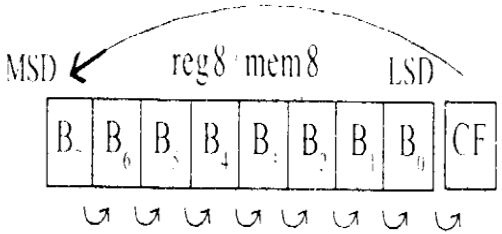
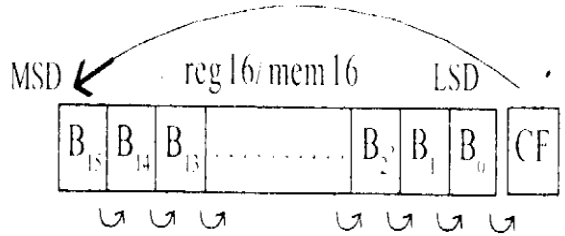
Algorithm :

shift all bits right, the bit that goes off is set to CF and previous value of CF is inserted to the left-most position.

Example:

```
STC             ; set carry (CF=1).
MOV AL, 1Ch     ; AL = 00011100b
RCR AL, 1       ; AL = 10001110b,  CF=0.
RET
```

OF=0 if first operand keeps original sign. **Back to Top**

RCR reg/mem	$B_n \leftarrow B_{n-1}$; $B_{MSD} \leftarrow CF$; $CF \leftarrow B_{LSD}$
RCR reg i) RCR reg, 1 ii) RCR reg, CL	
RCR mem i) RCR mem, 1 ii) RCR mem, CL	

Lecturer : Abdullah Thair Abdalsatir Al-Obaidi
Department of computer Engineering

Mnemonics: AND, OR, XOR, TEST, SHR, SHL, RCR, ROL ...

Rotate operand1 left. The number of rotates is set by operand2.

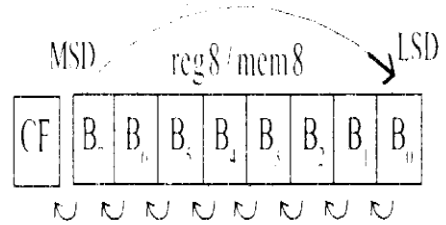
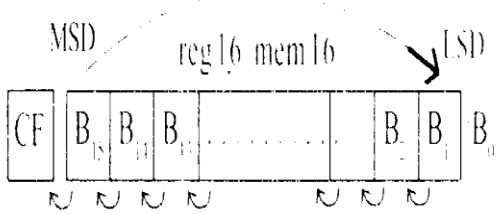
Algorithm:

shift all bits left, the bit that goes off is set to CF and the same bit is inserted to the right-most position.

Example:

```
MOV AL, 1Ch       ; AL = 00011100b
ROL AL, 1         ; AL = 00111000b,  CF=0.
RET
```

OF=0 if first operand keeps original sign. Back to Top

ROL reg/mem	$B_{n+1} \leftarrow B_n ; CF \leftarrow B_{MSD} ; B_{LSD} \leftarrow B_{MSD}$
ROL reg	
i) ROL reg, 1	
ii) ROL reg, CL	
ROL mem	
i) ROL mem, 1	
ii) ROL mem, CL	

4. String Manipulation Instructions

- String : Sequence of bytes or words , bytes= 8 bit , word =16 bit,
- 8086 instruction set includes instruction for string movement, comparison, scan, load and store.
- REP instruction prefix : used to repeat execution of string instructions
- String instructions end with S or SB or SW.
  S represents string, SB string byte and SW string word.
- Offset or effective address of the source operand is stored in SI register ,and that of the destination operand is stored in DI register.
- Depending on the status of DF, SI and DI registers are automatically updated.
- DF = 0 ⇒ SI and DI are incremented by 1 for byte and 2 for word.
- DF = 1 ⇒ SI and DI are decremented by 1 for byte and 2 for word.

4. String Manipulation Instructions [Instruction Set]

Mnemonics: REP, MOVS, CMPS, SCAS, LODS, STOS

Repeat following MOVSB, MOVSW, LODSB, LODSW, STOSB, STOSW instructions CX times .

Algorithm:

check_cx :

if CX <> 0 then

do following chain instruction

CX = CX - 1

go back to check_cx

else

exit from REP cycle

REP	
REPZ/ REPE **(Repeat CMPS or SCAS until ZF = 0)**	**While CX ≠ 0 and ZF = 1, repeat execution of string instruction and (CX) ← (CX) – 1**
REPNZ/ REPNE **(Repeat CMPS or SCAS until ZF = 1)**	**While CX = 0 and ZF = 0, repeat execution of string instruction and (CX) ← (CX) - 1**

Lecturer : Abdullah Thair Abdalsatir Al-Obaidi
Department of computer Engineering

Mnemonics: REP, MOVS, CMPS, SCAS, LODS, STOS

MOVS

MOVSB	**MA = (DS) x 16_{10} + (SI)** **MA_E = (ES) x 16_{10} + (DI)** **(MA_E) ← (MA)** **If DF = 0, then (DI) ← (DI) + 1; (SI) ← (SI) + 1** **If DF = 1, then (DI) ← (DI) - 1; (SI) ← (SI) - 1**
MOVSW	**MA = (DS) x 16_{10} + (SI)** **MA_E = (ES) x 16_{10} + (DI)** **(MA_E ; MA_E + 1) ← (MA; MA + 1)** **If DF = 0, then (DI) ← (DI) + 2; (SI) ← (SI) + 2** **If DF = 1, then (DI) ← (DI) - 2; (SI) ← (SI) - 2**

Lecturer : Abdullah Thair Abdalsatir Al-Obaidi
Department of computer Engineering

Mnemonics: REP, MOVS, CMPS, SCAS, LODS, STOS

Compare two string byte or string word

CMPS	
CMPSB	**MA = (DS) x 16_{10} + (SI)** **MA_E = (ES) x 16_{10} + (DI)** **Modify flags ← (MA) -(MA_E)** **If (MA) > (MA_E), then CF = 0; ZF = 0; SF = 0** **If (MA) < (MA_E), then CF = 1; ZF = 0; SF = 1**
CMPSW	**If (MA) = (MA_E), then CF = 0; ZF = 1; SF = 0** **<u>For byte operation</u>** **If DF = 0, then (DI) ← (DI) + 1; (SI) ← (SI) + 1** **If DF = 1, then (DI) ← (DI) - 1; (SI) ← (SI) - 1** **<u>For word operation</u>** **If DF = 0, then (DI) ← (DI) + 2; (SI) ← (SI) + 2** **If DF = 1, then (DI) ← (DI) - 2; (SI) ← (SI) - 2**

Mnemonics: REP, MOVS, CMPS, SCAS, LODS, STOS

Scan (compare) a string byte or word with accumulator.
Copy byte at DS:[SI] to ES:[DI]. Update SI and DI.

SCAS	
SCASB	**MA_E = (ES) x 16_{10} + (DI)** **Modify flags ← (AL) -(MA_E)** **If (AL) > (MA_E), then CF = 0; ZF = 0; SF = 0** **If (AL) < (MA_E), then CF = 1; ZF = 0; SF = 1** **If (AL) = (MA_E), then CF = 0; ZF = 1; SF = 0** **If DF = 0, then (DI) ← (DI) + 1** **If DF = 1, then (DI) ← (DI) −1**
SCASW	**MA_E = (ES) x 16_{10} + (DI)** **Modify flags ← (AL) -(MA_E)** **If (AX) > (MA_E ; MA_E + 1), then CF = 0; ZF = 0; SF = 0** **If (AX) < (MA_E ; MA_E + 1), then CF = 1; ZF = 0; SF = 1** **If (AX) = (MA_E ; MA_E + 1), then CF = 0; ZF = 1; SF = 0** **If DF = 0, then (DI) ← (DI) + 2** **If DF = 1, then (DI) ← (DI) −2**

Mnemonics: REP, MOVS, CMPS, SCAS, LODS, STOS

Load string byte in to AL or string word in to AX

LODS	
LODSB	**MA = (DS) x 16_{10} + (SI)** **(AL) ← (MA)** **If DF = 0, then (SI) ← (SI) + 1** **If DF = 1, then (SI) ← (SI) − 1**
LODSW	**MA = (DS) x 16_{10} + (SI)** **(AX) ← (MA ; MA + 1)** **If DF = 0, then (SI) ← (SI) + 2** **If DF = 1, then (SI) ← (SI) − 2**

Mnemonics: REP, MOVS, CMPS, SCAS, LODS, STOS

Store byte from AL or word from AX in to string

STOS	
STOSB	$MA_E = (ES) \times 16_{10} + (DI)$ $(MA_E) \leftarrow (AL)$ If DF = 0, then (DI) ← (DI) + 1 If DF = 1, then (DI) ← (DI) − 1
STOSW	$MA_E = (ES) \times 16_{10} + (DI)$ $(MA_E ; MA_E + 1) \leftarrow (AX)$ If DF = 0, then (DI) ← (DI) + 2 If DF = 1, then (DI) ← (DI) − 2

5. Processor Control Instructions [Instruction Set]

Mnemonics	Explanation
STC	Set CF ← 1
CLC	Clear CF ← 0
CMC	Complement carry CF ← CF/
STD	Set direction flag DF ← 1
CLD	Clear direction flag DF ← 0
STI	Set interrupt enable flag IF ← 1
CLI	Clear interrupt enable flag IF ← 0
NOP	No operation
HLT	Halt after interrupt is set
WAIT	Wait for TEST pin active
ESC opcode mem/ reg	Used to pass instruction to a coprocessor which shares the address and data bus with the 8086
LOCK	Lock bus during next instruction

6. Control Transfer Instructions [Instruction Set]

- Transfer the control to a specific destination or target instruction
- Do not affect flags

❑ 8086 Unconditional transfers

Mnemonics	Explanation
CALL reg/ mem/ disp16	Call subroutine
RET	Return from subroutine
JMP reg/ mem/ disp8/ disp16	Unconditional jump

- 8086 signed conditional branch instructions
- 8086 unsigned conditional branch instructions

➡ Checks flags

➡ If conditions are true, the program control is transferred to the new memory location in the same segment by modifying the content of IP

- 8086 signed conditionals branch instructions

Name	Alternate name
JE disp8 Jump if equal	JZ disp8 Jump if result is 0
JNE disp8 Jump if not equal	JNZ disp8 Jump if not zero
JG disp8 Jump if greater	JNLE disp8 Jump if not less or equal
JGE disp8 Jump if greater than or equal	JNL disp8 Jump if not less
JL disp8 Jump if less than	JNGE disp8 Jump if not greater than or equal
JLE disp8 Jump if less than or equal	JNG disp8 Jump if not greater

- 8086 unsigned conditional branch instructions

Name	Alternate name
JE disp8 Jump if equal	JZ disp8 Jump if result is 0
JNE disp8 Jump if not equal	JNZ disp8 Jump if not zero
JA disp8 Jump if above	JNBE disp8 Jump if not below or equal
JAE disp8 Jump if above or equal	JNB disp8 Jump if not below
JB disp8 Jump if below	JNAE disp8 Jump if not above or equal
JBE disp8 Jump if below or equal	JNA disp8 Jump if not above

- 8086 conditional branch instructions affecting individual flags

Mnemonics	Explanation
JC disp8	Jump if CF = 1
JNC disp8	Jump if CF = 0
JP disp8	Jump if PF = 1
JNP disp8	Jump if PF = 0
JO disp8	Jump if OF = 1
JNO disp8	Jump if OF = 0
JS disp8	Jump if SF = 1
JNS disp8	Jump if SF = 0
JZ disp8	Jump if result is zero, i.e, Z = 1
JNZ disp8	Jump if result is not zero, i.e, Z = 1

Thank you so much

Any questions ?

Lecturer : Abdullah Thair Abdalsatir
Department of computer Engineering
University of Diyala