INSTRUCTION SET

Ms. S. Angel Deborah
AP/CSE

Learning Objectives

1. To understand the different types on instruction.
2. To write Assembly Language Program

Overview

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. **Program Execution Transfer Instructions**

Instruction Set

1. Data Transfer Instructions

Instructions that are used to transfer data/ address in to registers, memory locations and I/O ports.

Generally involve two operands: Source operand and Destination operand of the same size.

Source: Register or a memory location or an immediate data
Destination : Register or a memory location.

The size should be a either a byte or a word.

A 8-bit data can only be moved to 8-bit register/ memory and a 16-bit data can be moved to 16-bit register/ memory.

Instruction Set

1. Data Transfer Instructions

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

MOV reg2/ mem, reg1/ mem	
MOV reg2, reg1 **MOV mem, reg1** **MOV reg2, mem**	**(reg2) ← (reg1)** **(mem) ← (reg1)** **(reg2) ← (mem)**
MOV reg/ mem, data	
MOV reg, data **MOV mem, data**	**(reg) ← data** **(mem) ← data**

XCHG reg2/ mem, reg1	
XCHG reg2, reg1 **XCHG mem, reg1**	**(reg2) ↔ (reg1)** **(mem) ↔ (reg1)**

Instruction Set

1. Data Transfer Instructions

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

PUSH reg16/ mem	
PUSH reg16	$(SP) \leftarrow (SP) - 2$ $MA_S = (SS) \times 16_{10} + SP$ $(MA_S ; MA_S + 1) \leftarrow (reg16)$
PUSH mem	$(SP) \leftarrow (SP) - 2$ $MA_S = (SS) \times 16_{10} + SP$ $(MA_S ; MA_S + 1) \leftarrow (mem)$
POP reg16/ mem	
POP reg16	$MA_S = (SS) \times 16_{10} + SP$ $(reg16) \leftarrow (MA_S ; MA_S + 1)$ $(SP) \leftarrow (SP) + 2$
POP mem	$MA_S = (SS) \times 16_{10} + SP$ $(mem) \leftarrow (MA_S ; MA_S + 1)$ $(SP) \leftarrow (SP) + 2$

Instruction Set

1. Data Transfer Instructions

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

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)**

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)**

Instruction Set

2. Arithmetic Instructions

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

ADD reg2/ mem, reg1/mem	
ADC reg2, reg1 **ADC reg2, mem** **ADC mem, reg1**	**(reg2) ← (reg1) + (reg2)** **(reg2) ← (reg2) + (mem)** **(mem) ← (mem)+(reg1)**
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**

Instruction Set

2. Arithmetic Instructions

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**

Instruction Set

2. Arithmetic Instructions

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

Instruction Set

2. Arithmetic Instructions

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**

Instruction Set

2. Arithmetic Instructions

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**

Instruction Set

2. Arithmetic Instructions

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

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

Instruction Set

2. Arithmetic Instructions

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**

Instruction Set

2. Arithmetic Instructions

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

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**

Instruction Set

2. Arithmetic Instructions

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

CMP reg2/mem, reg1/ mem	
CMP reg2, reg1	**Modify flags ← (reg2) – (reg1)** **If (reg2) > (reg1) then CF=0, ZF=0, SF=0** **If (reg2) < (reg1) then CF=1, ZF=0, SF=1** **If (reg2) = (reg1) then CF=0, ZF=1, SF=0**
CMP reg2, mem	**Modify flags ← (reg2) – (mem)** **If (reg2) > (mem) then CF=0, ZF=0, SF=0** **If (reg2) < (mem) then CF=1, ZF=0, SF=1** **If (reg2) = (mem) then CF=0, ZF=1, SF=0**
CMP mem, reg1	**Modify flags ← (mem) – (reg1)** **If (mem) > (reg1) then CF=0, ZF=0, SF=0** **If (mem) < (reg1) then CF=1, ZF=0, SF=1** **If (mem) = (reg1) then CF=0, ZF=1, SF=0**

Instruction Set

2. Arithmetic Instructions

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

CMP reg/mem, data	
CMP reg, data	**Modify flags ← (reg) – (data)** **If (reg) > data then CF=0, ZF=0, SF=0** **If (reg) < data then CF=1, ZF=0, SF=1** **If (reg) = data then CF=0, ZF=1, SF=0**
CMP mem, data	**Modify flags ← (mem) – (mem)** **If (mem) > data then CF=0, ZF=0, SF=0** **If (mem) < data then CF=1, ZF=0, SF=1** **If (mem) = data then CF=0, ZF=1, SF=0**

Instruction Set

2. Arithmetic Instructions

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

CMP A, data	
CMP AL, data8	**Modify flags ← (AL) – data8** **If (AL) > data8 then CF=0, ZF=0, SF=0** **If (AL) < data8 then CF=1, ZF=0, SF=1** **If (AL) = data8 then CF=0, ZF=1, SF=0**
CMP AX, data16	**Modify flags ← (AX) – data16** **If (AX) > data16 then CF=0, ZF=0, SF=0** **If (mem) < data16 then CF=1, ZF=0, SF=1** **If (mem) = data16 then CF=0, ZF=1, SF=0**

Instruction Set

3. Logical Instructions

Mnemonics: **AND, OR, XOR, TEST, SHR, SHL, RCR, RCL …**

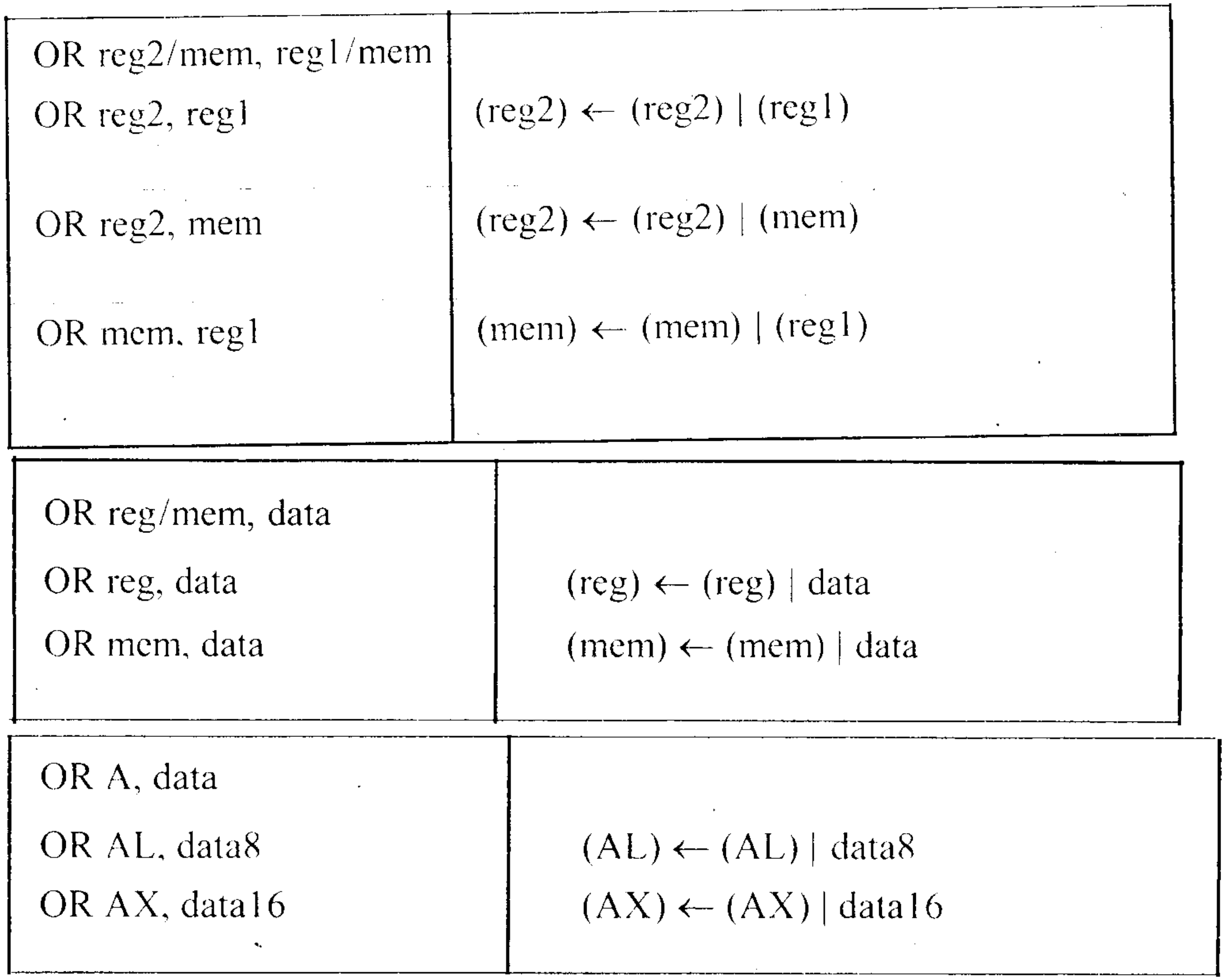
AND A, data AND AL, data8	(AL) ← (AL) & data8
AND AX, data16	(AX) ← (AX) & data16

AND reg/mem, data AND reg, data	(reg) ← (reg) & data
AND mem, data	(mem) ← (mem) & data

Instruction Set

3. Logical Instructions

Mnemonics: **AND, OR, XOR, TEST, SHR, SHL, RCR, RCL …**



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

Instruction Set

3. Logical Instructions

Mnemonics: **AND, OR, XOR, TEST, SHR, SHL, RCR, RCL …**

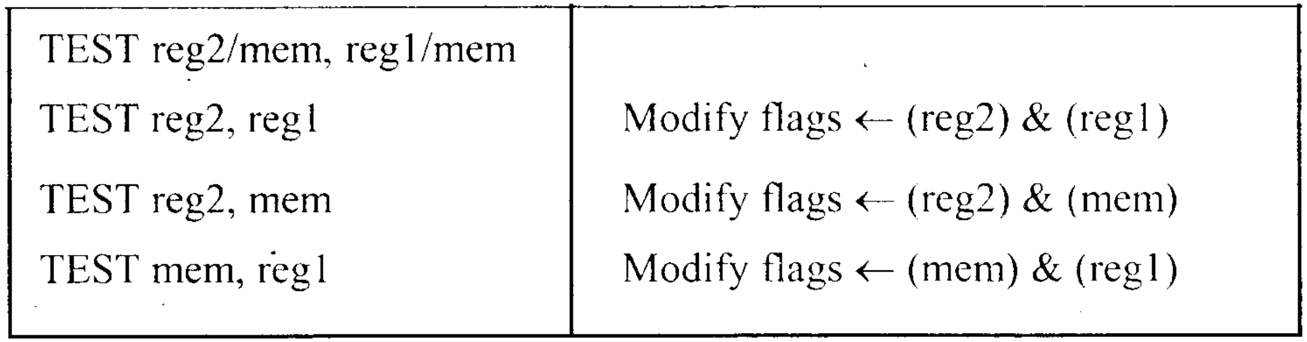
XOR reg2/mem, reg1/mem	
XOR reg2, reg1	(reg2) ← (reg2) ^ (reg1)
XOR reg2, mem	(reg2) ← (reg2) ^ (mem)
XOR mem, reg1	(mem) ← (mem) ^ (reg1)

XOR reg/mem, data	
XOR reg, data	(reg) ← (reg) ^ data
XOR mem, data	(mem) ← (mem) ^ data

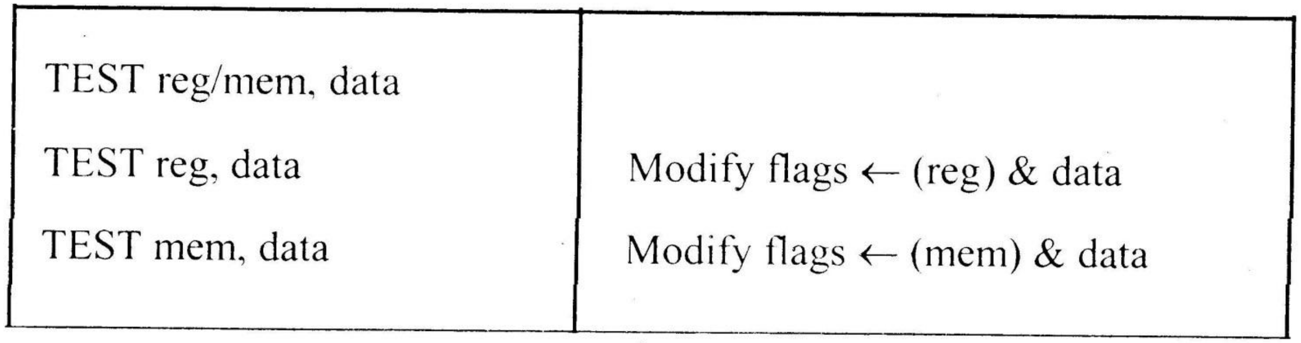
XOR A, data	
XOR AL, data8	(AL) ← (AL) ^ data8
XOR AX, data16	(AX) ← (AX) ^ data16

3. Logical Instructions

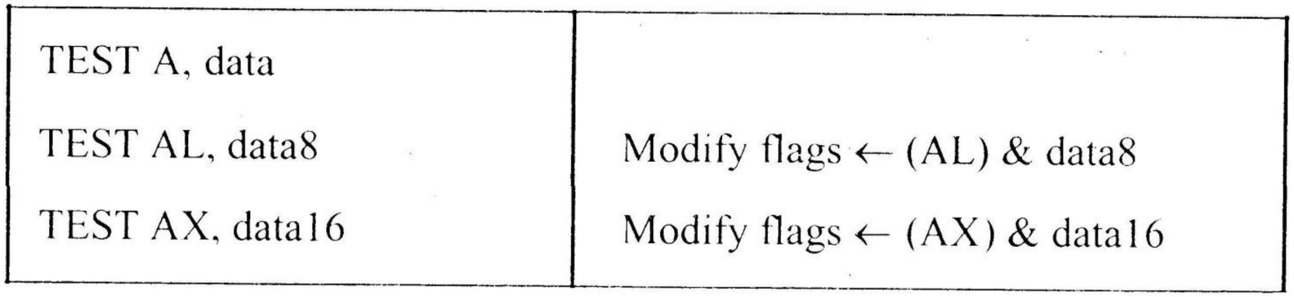
Mnemonics: **AND, OR, XOR, TEST, SHR, SHL, RCR, RCL …**



TEST reg2/mem, reg1/mem	
TEST reg2, reg1	Modify flags ← (reg2) & (reg1)
TEST reg2, mem	Modify flags ← (reg2) & (mem)
TEST mem, reg1	Modify flags ← (mem) & (reg1)



TEST reg/mem, data	
TEST reg, data	Modify flags ← (reg) & data
TEST mem, data	Modify flags ← (mem) & data



TEST A, data	
TEST AL, data8	Modify flags ← (AL) & data8
TEST AX, data16	Modify flags ← (AX) & data16

Instruction Set

3. Logical Instructions

Mnemonics: **AND, OR, XOR, TEST, SHR, SHL, RCR, RCL …**

SHR reg/mem

SHR reg

i) SHR reg, 1

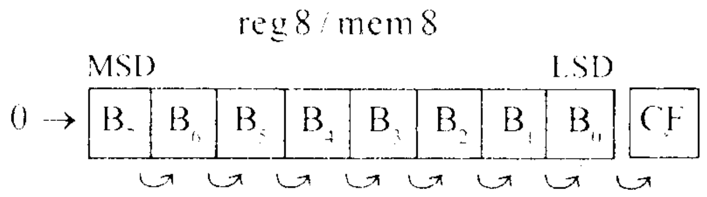
ii) SHR reg, CL

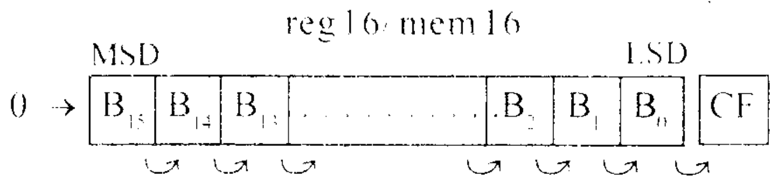
SHR mem

i) SHR mem, 1

ii) SHR mem, CL

$$CF \leftarrow B_{LSD} \; : \; B_n \leftarrow B_{n+1} \; ; \; B_{MSD} \leftarrow 0$$





Instruction Set

3. Logical Instructions

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

SHL reg/mem or SAL reg/mem

SHL reg or SAL reg

i) SHL reg, 1 or SAL reg, 1

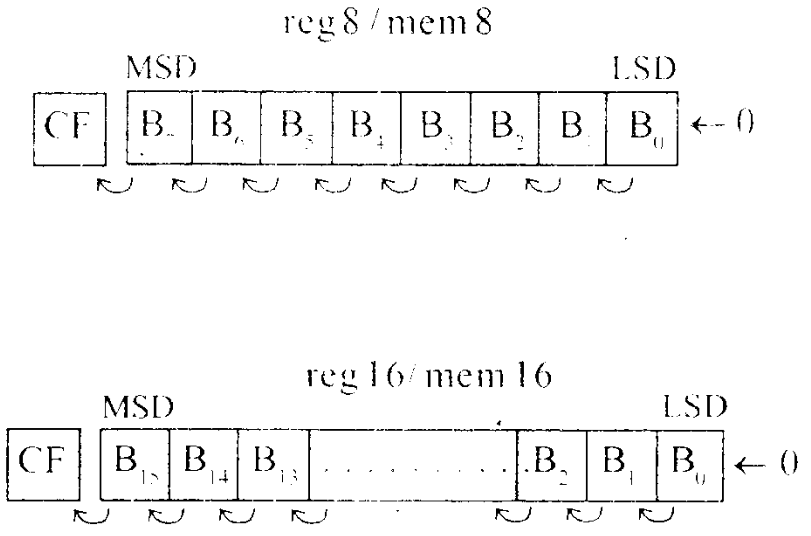
ii) SHL reg, CL or SAL reg, CL

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
MSD
LSD
CF
B_7 B_6 B_5 B_4 B_3 B_2 B_1 B_0
← 0
reg 16 / mem 16
MSD
LSD
CF
B_{15} B_{14} B_{13} . . . . B_2 B_1 B_0
← 0

Instruction Set

3. Logical Instructions

Mnemonics: **AND, OR, XOR, TEST, SHR, SHL, RCR, RCL …**

RCR reg/mem

RCR reg

i) RCR reg, 1

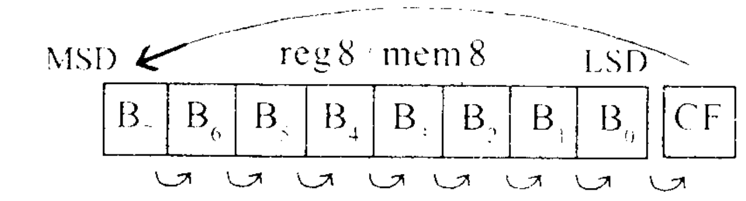
ii) RCR reg, CL

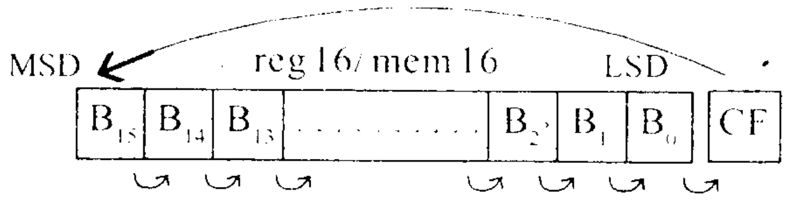
RCR mem

i) RCR mem, 1

ii) RCR mem, CL

$B_n \leftarrow B_{n-1}$; $B_{MSD} \leftarrow CF$; $CF \leftarrow B_{LSD}$

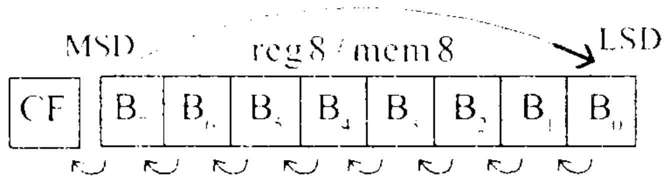
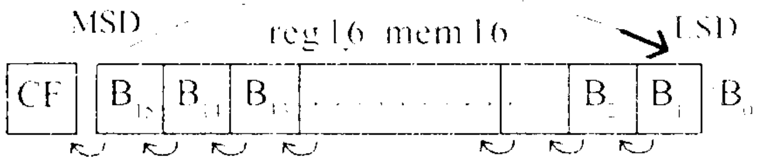




Instruction Set

3. Logical Instructions

Mnemonics: **AND, OR, XOR, TEST, SHR, SHL, RCR, RCL …**

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	 MSD reg8 / mem8 LSD CF B_7 B_6 B_5 B_4 B_3 B_2 B_1 B_0
ROL mem i) ROL mem, 1 ii) ROL mem, CL	 MSD reg16 / mem16 LSD CF B_{15} B_{14} B_{13} B_2 B_1 B_0

Instruction Set

4. String Manipulation Instructions

- **String : Sequence of bytes or words**
- **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.**

Instruction Set

4. String Manipulation Instructions

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

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**

Instruction Set

4. String Manipulation Instructions

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

Instruction Set

4. String Manipulation Instructions

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** **If (MA) = (MA_E), then CF = 0; ZF = 1; SF = 0**
CMPSW	**For byte operation** **If DF = 0, then (DI) ← (DI) + 1; (SI) ← (SI) + 1** **If DF = 1, then (DI) ← (DI) - 1; (SI) ← (SI) - 1** **For word operation** **If DF = 0, then (DI) ← (DI) + 2; (SI) ← (SI) + 2** **If DF = 1, then (DI) ← (DI) - 2; (SI) ← (SI) - 2**

Instruction Set

4. String Manipulation Instructions

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

Scan (compare) a string byte or word with accumulator

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

Instruction Set

4. String Manipulation Instructions

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

Instruction Set

4. String Manipulation Instructions

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

Instruction Set

5. Processor Control Instructions

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

Instruction Set

6. Program Execution Transfer Instructions

- **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

Instruction Set

6. Program Execution Transfer Instructions

- **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**

Instruction Set

6. Program Execution Transfer Instructions

❑ **8086 signed 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
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

6. Program Execution Transfer Instructions

- **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

Check your Understanding

1. What is the role of STI?
2. List out the instructions for performing logical operation?

Summary

1. Data Transfer Instructions
2. Arithmetic Instructions
3. Logical Instructions
4. String manipulation Instructions
5. Process Control Instructions
6. Program Execution Transfer Instructions

Reference

Doughlas V Hall, “Microprocessors and Interfacing, Programming and Hardware”, TMH, 2012.

Thank You