ECE 424 Design of Microprocessor-Based Systems

80x86 Instructions

Part 2

Haibo Wang
ECE Department
Southern Illinois University
Carbondale, IL 62901

Arithmetic Instructions

- ❑ ADD *Destination, Source*
  - — Destination + Source → Destination
  - — Destination and Source operands can not be memory locations at the same time
  - — It modifies flags AF CF OF PF SF ZF

- ❑ ADC *Destination, Source*
  - — Destination + Source + Carry Flag → Destination
  - — Destination and Source operands can not be memory locations at the same time
  - — It modifies flags AF CF OF PF SF ZF

- ❑ INC *Destination*
  - — Destination + 1 → Destination
  - — It modifies flags AF OF PF SF ZF (***Note CF will not be changed)***

- ❑ DEC *Destination*
  - — Destination - 1 → Destination
  - — It modifies flags AF OF PF SF ZF (***Note CF will not be changed)***

Arithmetic Instructions

- ❑ SUB *Destination, Source*
  - — Destination - Source → Destination
  - — Destination and Source operands can not be memory locations at the same time
  - — It modifies flags AF CF OF PF SF ZF

- ❑ SBB *Destination, Source*
  - — Destination - Source - Carry Flag → Destination
  - — Destination and Source operands can not be memory locations at the same time
  - — It modifies flags AF CF OF PF SF ZF

- ❑ CMP *Destination, Source*
  - — Destination – Source (the result is not stored anywhere)
  - — Destination and Source operands can not be memory locations at the same time
  - — It modifies flags AF CF OF PF SF ZF ***(if ZF is set, destination = source)***

Arithmetic Instructions

- MUL *Source*
  - Perform unsigned multiply operation
  - If source operand is a byte, ***AX = AL * Source***
  - If source operand is a word, ***(DX AX) = AX * Source***
  - Source operands can not be an immediate data
  - It modifies CF and OF (AF,PF,SF,ZF undefined)
- IMUL *Source*
  - Perform signed binary multiply operation
  - If source operand is a byte, ***AX = AL * Source***
  - If source operand is a word, ***(DX AX) = AX * Source***
  - Source operands can not be an immediate data
  - It modifies CF and OF (AF,PF,SF,ZF undefined)

➢ Examples:

```
MOV  AL, 20H              MOV  AL, 20H
MOV  CL, 80H              MOV  CL, 80H
MUL  CL                   IMUL  CL
```

Arithmetic Instructions

- DIV *Source*
  - — Perform unsigned division operation
  - — If source operand is a byte, ***AL = AX / Source; AH = Remainder of AX / Source***
  - — If source operand is a word, ***AX=(DX AX)/Source; DX=Remainder of (DX AX)/Source***
  - — Source operands can not be an immediate data

- IDIV *Source*
  - — Perform signed division operation
  - — If source operand is a byte, ***AL = AX / Source; AH = Remainder of AX / Source***
  - — If source operand is a word, ***AX=(DX AX)/Source; DX=Remainder of (DX AX)/Source***
  - — Source operands can not be an immediate data

➢ Examples:

```
MOV  AX, 5
MOV  BL, 2
DIV  BL
```

```
MOV  AL, -5
MOV  BL, 2
IDIV  BL
```

Arithmetic Instructions

- ❑ NEG *Destination*
  - — 0 – Destination → Destination (the result is represented in 2’s complement)
  - — Destination can be a register or a memory location
  - — It modifies flags AF CF OF PF SF ZF

- ❑ CBW
  - — Extends a signed 8-bit number in AL to a signed 16-bit data and stores it into AX
  - — It does not modify flags

- ❑ CWD
  - — Extends a signed 16-bit number in AX to a signed 32-bit data and stores it into DX and AX. DX contains the most significant word
  - — It does not modify flags

- ❖ Other arithmetic instructions:

DAA, DAS, AAA, AAS, AAM, AAD

Logical Instructions

- ❑ NOT *Destination*
  - — Inverts each bit of the destination operand
  - — Destination can be a register or a memory location
  - — It does not modify flags
- ❑ AND *Destination, Source*
  - — Performs logic AND operation for each bit of the destination and source; stores the result into destination
  - — Destination and source can not be both memory locations at the same time
  - — It modifies flags: CF OF PF SF ZF
- ❑ OR *Destination, Source*
  - — Performs logic OR operation for each bit of the destination and source; stores the result into destination
  - — Destination and source can not be both memory locations at the same time
  - — It modifies flags: CF OF PF SF ZF

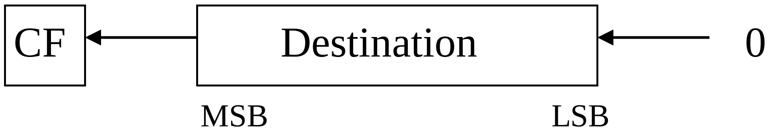
Logical Instructions

- ❑ XOR *Destination, Source*
  - — Performs logic XOR operation for each bit of the destination and source; stores the result into destination
  - — Destination and source can not be both memory locations at the same time
  - — It modifies flags: CF OF PF SF ZF

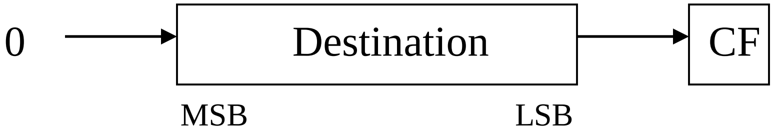
- ❑ TEST *Destination, Source*
  - — Performs logic AND operation for each bit of the destination and source
  - — Updates Flags depending on the result of AND operation
  - — Do not store the result of AND operation anywhere

Bit Manipulation Instructions

- SHL(SAL) *Destination, Count*
  - Left shift destination bits; the number of bits shifted is given by operand Count
  - During the shift operation, the MSB of the destination is shifted into CF and zero is shifted into the LSB of the destination
  - Operand Count can be either an immediate data or register CL
  - Destination can be a register or a memory location
  - It modifies flags: CF OF PF SF ZF

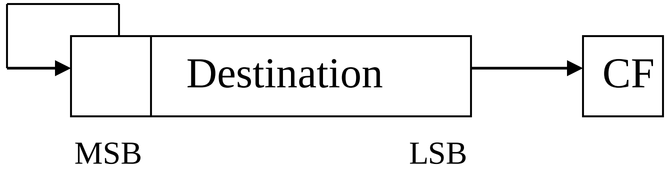


- SHR *Destination, Count*
  - Right shift destination bits; the number of bits shifted is given by operand Count
  - During the shift operation, the LSB of the destination is shifted into CF and zero is shifted into the MSB of the destination
  - Operand Count can be either an immediate data or register CL
  - Destination can be a register or a memory location
  - It modifies flags: CF OF PF SF ZF



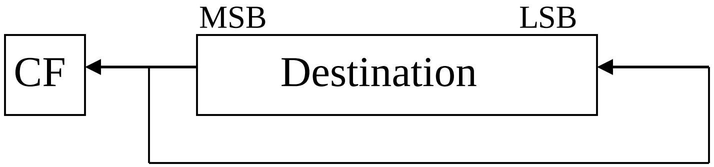
Bit Manipulation Instructions

- □ SAR *Destination, Count*
  - — Right shift destination bits; the number of bits shifted is given by operand Count
  - — The LSB of the destination is shifted into CF and the MSB of the destination remians the same
  - — Operand Count can be either an immediate data or register CL
  - — Destination can be a register or a memory location
  - — It modifies flags: CF PF SF ZF

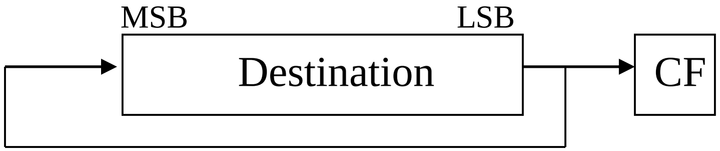


Bit Manipulation Instructions

- ❑ ROL *Destination, Count*
  - — Left shift destination bits; the number of bits shifted is given by operand Count
  - — The MSB of the destination is shifted into CF, it also goes to the LSB of the destination
  - — Operand Count can be either an immediate data or register CL
  - — Destination can be a register or a memory location
  - — It modifies flags: CF OF

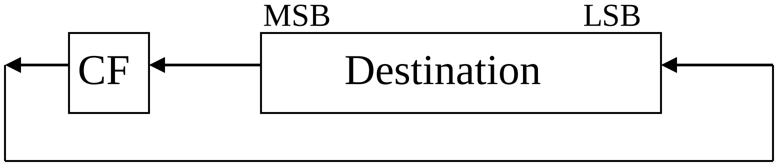


- ❑ ROR *Destination, Count*
  - — Right shift destination bits; the number of bits shifted is given by operand Count
  - — The LSB of the destination is shifted into CF, it also goes to the MSB of the destination
  - — Operand Count can be either an immediate data or register CL
  - — Destination can be a register or a memory location
  - — It modifies flags: CF OF

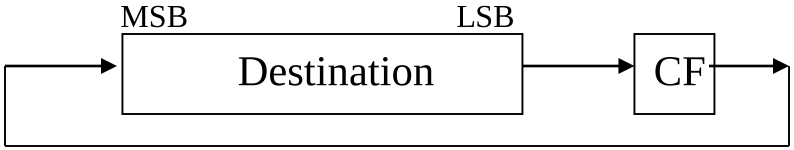


Bit Manipulation Instructions

- ❑ RCL *Destination, Count*
  - — Left shift destination bits; the number of bits shifted is given by operand Count
  - — The MSB of the destination is shifted into CF; the old CF value goes to the LSB of the destination
  - — Operand Count can be either an immediate data or register CL
  - — Destination can be a register or a memory location
  - — It modifies flags: CF OF PF SF ZF



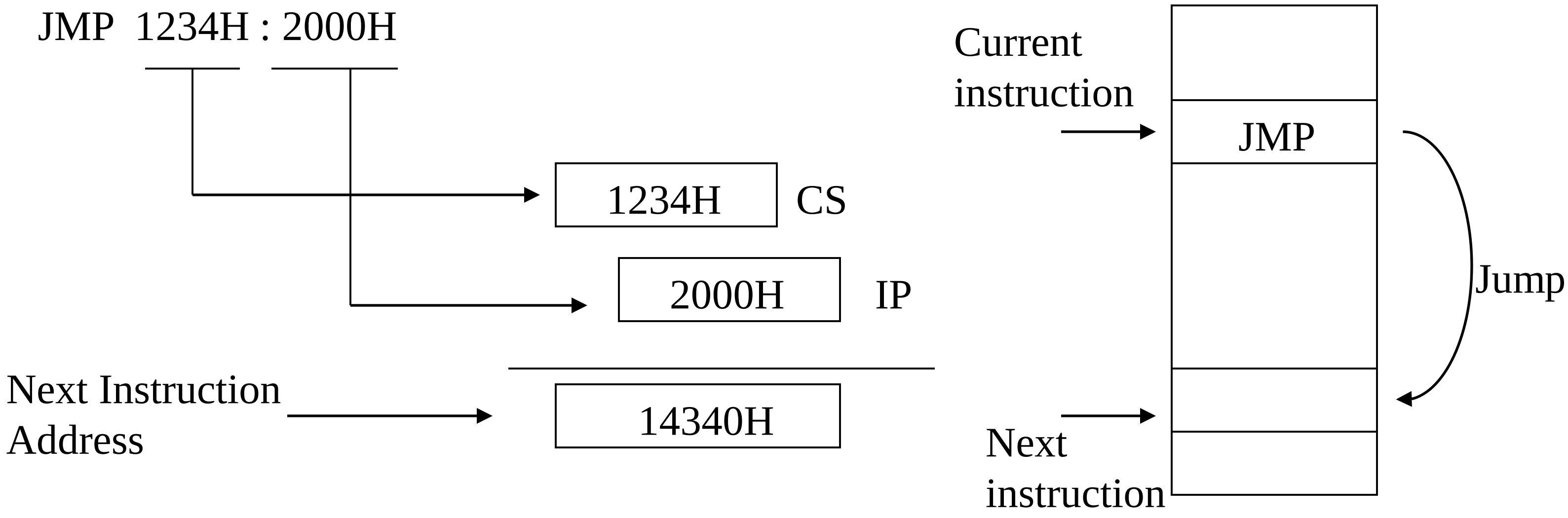
- ❑ RCR *Destination, Count*
  - — Right shift destination bits; the number of bits shifted is given by operand Count
  - — The LSB of the destination is shifted into CF, the old CF value goes to the MSB of the destination
  - — Operand Count can be either an immediate data or register CL
  - — Destination can be a register or a memory location
  - — It modifies flags: CF OF PF SF ZF



Program Transfer Instructions

- ❑ JMP *Target*
  - — Unconditional jump
  - — It moves microprocessor to execute another part of the program
  - — Target can be represented by a label, immediate data, registers, or memory locations
  - — It does not affect flags

➢ The execution of JMP instruction



Program Transfer Instructions

- Intrasegment transfer ***v.s.*** Intersegment transfer

  — Intrasegment transfer: the microprocessor jumps to an address within the same segment
  — Intersegment transfer: the microprocessor jumps to an address in a difference segment
  — Use assembler directive ***near*** and ***far*** to indicate the types of JMP instructions

  — For intrasegment transfer, we can provide only new IP value in JMP instructions.
  For Example: JMP 1000H
  — For intersegment transfer, we need provide both new CS and IP values in JMP instructions
  For Example: JMP 2000H : 1000H

- Direct Jump ***v.s.*** Indirect Jump

  — Direct Jump: the target address is directly given in the instruction
  — Indirect Jump: the target address is contained in a register or memory location

- Short Jump

  — If the target address is within +127 or –128 bytes of the current instruction address, the jump is called a short jump
  — For short jumps, instead of specifying the target address, we can specify the relative offset (the distance between the current address and the target address) in JMP instructions.

Program Transfer Instructions

- Conditional Jumps
  - JZ: *Label_1*
    — If ZF =1, jump to the target address labeled by *Label_1;* otherwise, do not jump
  - JNZ: *Label_1*
    — If ZF =0, jump to the target address labeled by *Label_1;* otherwise, do not jump
- Other Conditional Jumps

JNC	JAE	JNB	JC	JB	JNAE	JNG
JNE	JE	JNS	JS	JNO	JO	JNP
JPO	JP	JPE	JA	JBNE	JBE	JNA
JGE	JNL	JL	JNGE	JG	JNLE	JLE

  - JCXZ: *Label_1*
    — If CX =0, jump to the target address labeled by *Label_1;* otherwise, do not jump

Program Transfer Instructions

- LOOP *Short_Label*
  - It is limited for short jump
  - Execution Flow:

```
CX = CX –1
If  CX != 0  Then
      JMP Short_Label
End IF
```

- LOOPE/LOOPZ *Short_Label*

```
CX = CX –1
If  CX != 0 & ZF=1 Then
      JMP Short_Label
End IF
```

- LOOPNE/LOOPNZ *Short_Label*

```
CX = CX –1
If  CX != 0 & ZF=0 Then
      JMP Short_Label
End IF
```

Processor Control Instructions

CLC	*Clear carry flag*
STC	*Set carry flag*
CMC	*Complement carry flag*
CLD	*Clear direction flag*
STD	*Set direction flag*
CLI	*Clear interrupt-enable flag*
STI	*Set interrupt-enable flag*
HLT	*Halt microprocessor operation*
NOP	*No operation*
LOCK	*Lock Bus During Next Instruction*