Dhaka University of Engineering & Technology, Gazipur

Department of Computer Science and Engineering (CSE)

Course Title: Microprocessor and Interfacing Sessional (CSE 3812)

Lab # 05

Logic, Shift and Rotate Instructions & Multiplication and Division Instructions in EMU8086.

Logic, Shift and Rotate Instructions

Logic Instructions

Logic instructions are: AND, OR, XOR and NOT

- **AND**

  The result of the operation is stored in the destination. Destination must be a register or memory location. Source may be a constant, register or memory location. Memory to memory operations are not allowed

  Effect on flags: SF, ZF and PF reflect the result. CF, OF = 0

  Example: Converting an ASCII digit to a number and conversion of lowercase letter to upper case letter.

  Syntax: AND destination, source

- **OR**

  The result of the operation is stored in the destination. Destination must be a register or memory location. Source may be a constant, register or memory location. Memory to memory operations are not allowed.

  Effect on flags: SF, ZF and PF reflect the result. CF, OF = 0

  Example: Testing a register for zero and to check the sign of the value.

  Syntax: OR destination, source

- **XOR**

  The result of the operation is stored in the destination. Destination must be a register or memory location. Source may be a constant, register or memory location. Memory to memory operations are not allowed.

  Effect on flags: SF, ZF and PF reflect the result. CF, OF = 0

  Example: Clearing the value of a register

  Syntax: XOR destination, source

- **NOT**

  Perform the one's complement operation on the destination. The result of the operation is stored in the destination. Destination must be a register or memory location.

  Effect on flags: There is no effect on the status flags.

  Example: Complement the bits of a register or memory location

  Syntax: NOT destination

Mask

One use of AND, OR and XOR is to selectively modify the bits in the destination. To do this we construct a source bit pattern known as mask. The mask bits are chosen so that the corresponding destination bits are modified in the desired way.

B AND 1 = B	B OR 0 = B	B XOR 0 = B
B AND 0 = 0	B OR 1 = 1	B XOR 1 = -B

The AND instruction can be used to clear specific destination bits while preserving the others.
The OR instruction can be used to set specific destination bits while preserving the others.
The XOR instruction can be used to complement specific destination bits while preserving the others.

The TEST Instruction

The TEST instruction performs an AND operation of the destination with the source but does not change the destination contents. The purpose of the TEST instruction is to set the status flags. The mask should contain 1's in the bit positions to be tested and 0's elsewhere. If the destination has 0's in all the tested positions, the result will be 0 and so ZF=1. Effect on flags: SF, ZF, PF reflect the result. CF, OF= 0

Syntax: TEST destination, source

Shift Instructions

The shift instructions shift the bits in the destination operand by one or more positions either to the left or right. For a shift instruction, the bits shifted out are lost. For intel's more advanced processors, a shift instruction also allows the use of an 8-bit constant

Syntax: opcode destination, 1; for a single shift

opcode destination, CL; for a shift of N positions where CL contains N.

In both cases, destination is an 8 or 16-bit register or memory location.

There are two types of shift instruction: Left shift and Right shift.

- Left Shift: The SHL Instruction, The SAL Instruction
- Right Shift: The SHR Instruction, The SAR Instruction

SHL Instruction

The SHL (shift left) instruction shifts the bits in the destination to the left. A 0 is shifted into the rightmost bit position and the MSB is shifted into CF.

Effect on flags: SF, PF, ZF reflect the result. CF= last bit shifted out. OF=1 if the result changes sign on the last shift.

Example: The SHL instruction on a binary number doubles the value.

Syntax: SHL destination, 1; for a single shift

SHL destination, CL; for a shift of N positions where CL contains N. The value of CL remains the same after the shift operation.

The SAL Instruction

The opcode SAL (shift arithmetic left) is often used in instances where numeric multiplication is intended. SAL instructions generate the same machine code as SHL instruction. Negative numbers can also be multiplied by powers of 2 by left shifts.

Example: If AX is FFFFh (-1), then shifting three times will yield AX= FFF8h (-8).

Overflow

When we treat left shifts as multiplication, overflow may occur. For a single left shift, CF and OF accurately indicate unsigned and signed overflow, respectively. But the overflow flags are not reliable indicators for a multiple left shift. This is because a multiple shift is really a series of single shifts, and CF, OF only reflect the result of the last shift.

Example: If BL contains 80h, CL contains 2 and we execute SHL BL, CL; then CF = OF = 0 even though both signed and unsigned overflow occur.

Example

Write some code to multiply the value of AX by 8. Assume that overflow will not occur.

Solution: To multiply by 8, we need to do three left shifts.

```
MOV    CL, 3; number of shifts to do
SAL    AX, CL; multiply by 8
```

The SHR Instruction

The instruction SHR (shift right) performs right shifts on the destination operand. A 0 is shifted into the MSB position, and the rightmost bit is shifted into CF.
Effect on flags: SF, PF, ZF reflect the result. CF = last bit shifted out. OF = 1 if result changes sign on last shift
Example: The SHR instruction on a binary number halves the value if it is an even number. For odd numbers, a right shift halves it and rounds down to the nearest integer.
Syntax: SHR destination, 1; for a single shift
SHR destination, CL; for a shift of N positions where CL contains N.

The SAR Instruction

The SAR instruction (shift arithmetic right) operates like SHR. The MSB retains its original value.
Effect on flags: SF, PF, ZF reflect the result. CF = last bit shifted out. OF =1 if result changes sign on last shift.
Syntax: SAR destination,1; for a single shift
SAR destination, CL; for a shift of N positions where CL contains N.

Rotate Instructions

The rotate instructions rotate the bits in the destination operand by one or more positions either to the left or right. For a rotate instruction, bits shifted out from one end of the operand are put back into the other end. For intel's more advanced processors, a rotate instruction also allows the use of an 8-bit constant.

Syntax: opcode destination, 1; for a single rotate
opcode destination, CL; for a rotate of N positions where CL contains N.
In both cases, destination is an 8 or 16-bit register or memory location.

Rotate instructions are of two kinds: Left Rotate and right rotate.

- Left Rotate: The ROL Instruction, The RCL Instruction
- Right Rotate: The ROR Instruction, The RCR Instruction

The ROL Instruction

The instruction ROL (rotate left) shifts bits to the left. The MSB is shifted into the rightmost bit. The CF also gets the bit shifted out of the MSB. Destination bits forming a circle, with the least significant bit following the MSB in the circle. In ROL, CF reflects the bit that is rotated out. This can be used to inspect the bits in a byte or word without changing the contents.

Syntax: ROL destination, 1; for a single rotate
ROL destination, CL; for a rotate of N positions where CL contains N

The ROR Instruction

The instruction ROR (rotate right) shifts bits to the right. The rightmost bit is shifted into the MSB and also into the CF. In ROR, CF reflects the bit that is rotated out. This can be used to inspect the bits in a byte or word without changing the contents.

Syntax: ROR destination, 1; for a single rotate
ROR destination, CL; for a rotate of N positions where CL contains N

Example

Use ROL to count the number of 1 bit in BX, without changing BX. Put the answer in AX.

Solution:

```
XOR     AX, AX; AX counts
bits MOV        CX, 16; loop
counter TOP:
```

```
ROL     BX,1; CF=bit rotated out
```

```
JNC    NEXT; 0 bit
INC    AX; 1 bit, increment total
NEXT:
LOOP   TOP; loop until done
```

The RCL Instruction

The instruction RCL (Rotate through Carry Left) shifts the bits of the destination to the left. The MSB is shifted into the CF, and the previous value of CF is shifted into the rightmost bit. RCL works like ROL, except that CF is part of the circle of bits being rotated.

Effect on the flags: SF, PF, ZF reflect the result. CF = last bit shifted out. OF = 1 if result changes sign in the last rotation.

Syntax: RCL destination, 1; for a single rotate

RCL destination, CL; for a rotate of N positions where CL contains N

The RCR Instruction

The instruction RCR (Rotate through Carry Right) shifts the bits of the destination to the right. The LSB is shifted into the CF, and the previous value of CF is shifted into the leftmost bit. RCR works like ROR, except that CF is part of the circle of bits being rotated.

Effect on the flags: SF, PF, ZF reflect the result. CF = last bit shifted out.

Syntax: RCR destination, 1; for a single rotate

RCR destination, CL; for a rotate of N positions where CL contains N, OF = 1 if result changes sign in the last rotation.

Multiplication and Division Instructions

Signed & Unsigned Multiplication

In binary multiplication, signed and unsigned numbers must be treated differently. For example, we want to multiply the eight-bit numbers 10000000 and 11111111.

Interpreted as unsigned numbers, they represent 128 and 255 respectively. The product is 32640 = 0111111110000000b.

Interpreted as signed numbers, they represent -128 and -1 respectively and the product is

128 = 0000000010000000b.
Because signed and unsigned multiplication lead to different results there are two multiplication instructions: MUL and IMUL. For multiplication of positive numbers MUL and IMUL give the same result.
These instructions multiply bytes or words.
For byte multiplication, one number is contained in the source and the other is assumed to be in AL. The 16-bit product will be in AX. The source may be a byte register or memory byte, but not a constant.
For word multiplication, one number is contained in the source and the other is assumed to be in AX. The most significant 16-bits of the double word product will be in DX, and the least significant 16 bits will be in AX (DX:AX). The source may be a 16-bit register or memory word, but not a constant.

MUL Instructions

MUL (multiply) is used for unsigned multiplication.

Syntax: MUL source

Effect on status flags: SF, ZF, PF and AF undefined. CF/OF is 0 if the upper half of the result is zero otherwise 1.

IMUL Instructions

IMUL (integer multiply) is used for signed multiplication.

Syntax: IMUL source

Effect on status flags: SF, ZF, PF and AF undefined. CF/OF is 0 if the upper half of the result is the sign extension of the lower half otherwise 1.

Example

Suppose AX contains 1 and BX contains FFFFh

Instruction	Decimal Product	Hex Product	DX	AX	CF/OF
MUL BX	65535	0000FFFF	0000	FFFF	0
IMUL BX	-1	FFFFFFFF	FFFF	FFFF	0

Simple Application of MUL and IMUL

Translate the high-level language assignment statement, A= 5 x A - 12 x B into assembly code. Let A and B be word variables and suppose there is no overflow. Use IMUL for multiplication.

```
MOV    AX,5
IMUL A

MOV     A, AX
MOV
AX,12 IMUL
B
SUB    A, AX
```

Signed & Unsigned Division

Signed and unsigned division lead to different results. There are two division instructions: DIV and IDIV. These instructions divide 8 (or 16) bits into 16 (or 32) bits.
The quotient and remainder have the same size as the divisor.
In byte form, the divisor is an 8-bit register or memory byte. The 16-bit dividend is assumed to be in AX. After division, the 8-bit quotient is in AL and the 8-bit remainder is in AH. The divisor may not be a constant.

In word form, a divisor is a 16-bit register or memory word. The 32-bit dividend is assumed to be in DX:AX, after division, the 16-bit quotient is in AX and the 16-bit remainder is in DX. The divisor may not be a constant.
The effect of DIV and IDIV on the flags is that all status flags are undefined.
It is possible that the quotient will be too big to fit in the specified destination (AL or AX). This can happen if the divisor is much smaller than the dividend. If this happens, the program terminates and the system displays the message "Divide Overflow".

DIV Instruction

DIV (divide) is used for unsigned division.

Syntax: DIV divisor

IDIV Instruction

IDIV (integer divide) is used for signed division. For signed division; the remainder has the same sign as the dividend.

Syntax: IDIV divisor

Example

- Suppose DX contains 0000h, AX contains 0005h, and BX contains 0002h.

Instruction	Decimal Quotient	Decimal Remainder	AX	DX
DIV BX	2	1	0002	0001
IDIV BX	2	1	0002	0001

Sign Extension of the Dividend

Word Division

The dividend is in DX:AX even if the actual dividend will fit in AX. In this case DX should be prepared as follows: For DIV, DX should be cleared. For IDIV, DX should be made the sign extension of AX. The instruction CWD (convert word to double word) will do the extension.
Example: Divide -1250 by 7

Solution:

```
MOV     AX, -1250
```

CWD

```
MOV     BX,7
IDIV    BX
```

Byte Division

The dividend is in AX. If the actual dividend is a byte then AH should be prepared as follows: For DIV, AH should be cleared. For IDIV, AH should the sign extension of AL. The instruction CBW (convert byte to word) will do the extension.

Example: Divide the signed value of the byte variable: XBYTE by -7.

Solution:

```
MOV     AL, XBYTE
CBW
AH MOV      BL,
-7 IDIV BL
```