Lecture 5

Dr. Zbigniew Kalbarczyk

University of Illinois at Urbana- Champaign

- Basic conditional clauses
- Multiplication and division
- Shift instructions

If…then…else clauses

```
' Visual Basic Example
if ax > bx then
      'true instructions
else
      'false instructions
end if
```

```
      cmp ax, bx
      ja .true
            ;false instructions
       jmp .done
.true
            ;true instructions
.done
```


Switch…case clauses

```
/* C code example */
switch (choice) {

      case 'a':
            /* a instructions */
            break;

      case 'b':
            /* b instructions */
            break;

      default:
            /* default instructions */
}
```

```
      cmp   al, 'a'
      jne .b
            ; a instructions
      jmp .done

.b
      cmp al, 'b'
      jne .default
            ; b instructions
      jmp .done

.default
      ; default instructions
 .done
```

Do…while clause

```
do {
    // instructions
while (A=B);
```

```
.begin
    ; instructions
mov ax, [A]
cmp ax, [B]
je .begin
```


While…do clause

```
while (A=B) {
    // instructions
}
```

```
.begin
    mov ax, [A]
    cmp ax, [B]
    jne .done
    ; instructions
jmp .begin
```

Looping

- The LOOP instruction is a combination of the DEC and JNZ instructions
- LOOP decrements CX and if CX has not become zero jumps to the specified label
- If CX is zero, the instruction following the LOOP is executed

Loop example

```
ADDS
      mov   cx, 100           ;load count
      mov   si, BLOCK1
      mov   di, BLOCK2
.Again
      lodsw                   ;get Block1 data
                              ; AX = [SI]; SI = SI + 2
      add   AX, [ES:DI]       ;add Block2 data
      stosw                   ;store in Block2
                              ; [DI] = AX; DI = DI + 2
      loop  .Again            ;repeat 100 times

      ret
```

LOOPNE

- The LOOPNE instruction is useful for controlling loops that stop on some condition or when the loop exceeds some number of iterations
- Consider **String1** that contains a sequence of characters that end with the byte containing zero
- we want to convert those characters to upper case and copy them to **String2**

```
…..
String1  DB  'This string contains lower case characters', 0
String2  times 128 DB 0
…….. 
```


Example (LOOPNE)

```
   mov si, String1            ;si points to beginning of String1
   lea di, String2            ;di points to beginning of String2
   mov cx, 127                ;Max 127 chars to String2
.StrLoop:
   lodsb              ;get char from String1; AL=[DS:SI]; SI= SI + 1
   cmp al, 'a'                ;see if lower case
   jb .NotLower               ;chars are unsigned
   cmp al, 'z'
   ja .NotLower
   and al, 5Fh                ;convert lower -> upper case
                              ;bit 6 must be 0
.NotLower:
   stosb                      ;[ES:DI] = AL; DI = DI + 1
   cmp al, 0                  ;see if zero terminator
   loopne .StrLoop            ;quit if AL or CX = 0
```

Multiplication

- The product after a multiplication is always a double-width product
  - if we multiply two 16-bit numbers, they generate a 32-bit product
  - unsigned: $(2^{16} - 1) * (2^{16} - 1) = 2^{32} - 2 * 2^{16} + 1 < (2^{32} - 1)$
  - signed: $(-2^{15}) * (-2^{15}) = 2^{30} < (2^{31} - 1)$
  - overflow cannot occur
- Modification of Flags
  - Most flags are undefined after multiplication
  - **O** and **C** flags clear to 0 if the result fit into half-size register
  - if the most significant 16 bits of the product are 0, both flags **C** and **O** clear to 0

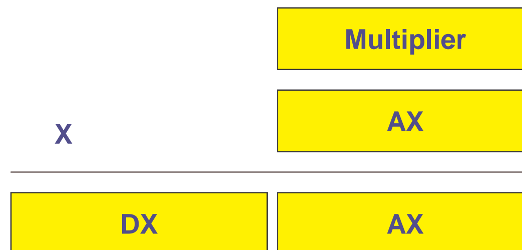
Multiplication

- Two different instructions for multiplication
  - MUL Unsigned integer multiplication
  - IMUL Singed integer multiplication
- MUL Multiplication is performed on bytes, words, or doubles
- Size of the multiplier determines the operation
- The multiplier can be any register or any memory location
- For MUL, the multiplicand is always AL, AX or EAX

```
mul     cx          ; AX * CX (unsigned result in DX:AX);
imul    word [si]   ; AX * [word content of memory location
                    ; addressed by SI] (signed product in  DX:AX)
```

Unsigned multiplication

- 16-bit multiplication
  Multiplicand AX
  Multiplier 16-bit register or 16-bit memory variable
  Product High word in DX, Low word in AX



Unsigned multiplication

- **8-bit multiplication**
  - Multiplicand AL
  - Multiplier 8-bit register or 8-bit memory variable
  - Product AX
- **32-bit multiplication**
  - Multiplicand EAX
  - Multiplier 32-bit register or 32-bit memory variable)
  - Product High word in EDX : Low word in EAX)

Signed multiplication

- Not available in 8088 and 8086
- For 2's Complement signed integers only
- Four forms
  - imul reg/mem – assumes AL, AX or EAX
  - imul reg, reg/mem
  - imul reg, immediate data
  - imul reg, reg/mem, immediate data

Signed multiplication

- Examples
  - imul bl ; BL * AL à AX
  - imul bx ; BX * AX à DX:AX
  - imul cl, [bl] ; CL * [DS:BL] à CL ; overflows can occur
  - imul cx, dx, 12h ; 12h * DX à CX

Binary Multiplication

- Long multiplication is done through shifts and additions

```
        0 1 1 0 0 0 1 0    (98)
      x 0 0 1 0 0 1 0 1    (37)
      -----------------
        0 1 1 0 0 0 1 0
    0 1 1 0 0 0 1 0 - -
0 1 1 0 0 0 1 0 - - - - -  (3626)
```

- This works if both numbers are positive
- To multiply negative numbers, the CPU will store the sign bits of the numbers, make both numbers positive, compute the result, then negate the result if necessary

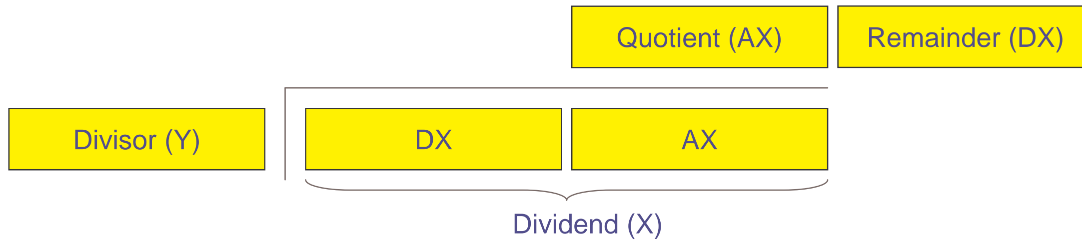
Division

- Two different instructions for division
  - DIV Unsigned division
  - IDIV Signed division (2's complement)
- Division is performed on bytes, words, or double words
- The size of the divisor determines the operation
- The dividend is always a double-width dividend that is divided by the operand (divisor)
- The divisor can be any register or any memory location

Division

32-bit/16-bit division - the use of the AX (and DX) registers is implied

Dividend	high word in DX, low word in AX
Divisor	16-bit register or 16-bit memory variable
Quotient	AX
Remainder	DX



Division

- **16-bit/8-bit**
  - Dividend AX
  - Divisor 8-bit register or 8-bit memory variable
  - Quotient AL
  - Remainder AH

- **64-bit/32-bit**
  - Dividend high double word in EDX, low double word in EAX
  - Divisor 32-bit register or 32-bit memory variable)
  - Quotient EAX
  - Remainder EDX

Division

- Division of two equally sized words
  - *Unsigned numbers:* move zero into high order-word
  - *Signed numbers:* use signed extension to fill high-word with ones or zeros
  - **CBW** (convert byte to word)

    AX = xxxx xxxx snnn nnnn (before)

    AX = ssss ssss snnn nnnn (after)
  - **CWD** (convert word to double)

    DX:AX = xxxx xxxx xxxx xxxx snnn nnnn nnnn nnnn (before)

    DX:AX = ssss ssss ssss ssss snnn nnnn nnnn nnnn (after)
  - **CWDE** (convert double to double-word extended)

Division

- Flag settings
  - none of the flag bits change predictably for a division
- A division can result in two types of errors
  - divide by zero
  - divide overflow (a small number divides into a large number), e.g., 3000 / 2
    - AX = 3000
    - Devisor is 2 => 8 bit division is performed
    - Quotient will be written to AL => but 1500 does not fit into AL
    - consequently we have divide overflow
    - in both cases microprocessor generates interrupt (interrupts are covered later in this course)

Example

Division of the byte contents of memory NUMB
by the contents of NUMB1

Unsigned

```
MOV AL, [NUMB]   ;get NUMB
MOV AH, 0        ;zero extend
DIV byte [NUMB1]
MOV [ANSQ], AL   ;save quotient
MOV [ANSR], AH   ;save remainder
```

Signed

```
MOV AL, [NUMB]   ;get NUMB
CBW              ;signed-extend
IDIV byte [NUMB1]
MOV [ANSQ], AL   ;save quotient
MOV [ANSR], AH   ;save remainder
```


Division

- What do we do with remainder after division?
  - use the remainder to round the result
  - drop the remainder to truncate the result
  - if the division is unsigned, rounding requires that double the remainder is compared with the divisor to decide whether to round up the quotient
  - Example, sequence of instructions that divide AX by BL and round the result

```
        DIV     BL
        ADD     AH, AH  ;double remainder
        CMP     AH, BL  ;test for rounding
        JB      .NEXT
        INC     AL
.NEXT
```

Shift left

- Syntax: SHL reg, count (immediate count)
  SHL reg, CL (count stored in CL)
  - moves each bit of the operand one bit position to the left the number of times specified by the count operand
  - zeros fill vacated positions at the L.O. bit
  - H.O. bit shifts into the carry flag
- a quick way to multiply by two
- useful in *packing data,* e.g., consider two nibbles in AL and AH that we want to combine

  ```
  shl ah, 4
  or al,ah
  ```
- the 8086 and 8088 allow an immediate shift of 1 only

Example

```
; Pack the lower nibbles from AH and AL into a single byte in AL
; for example AH = 0110 1101 and AL = 1010 0111

and al, 0Fh                ; clears upper nibble in AL
                           ; AL = 0000 0111
shl ah, 4                  ; Moves lower nibble in AH up
                           ; AH = 1101 0000
or  al, ah                 ; combines nibbles
                           ; AL = 1101 0111
```

Shift right

- Syntax: SHR reg, count (immediate count)
  SHR reg, CL (count stored in CL)
  - moves each bit of the operand one bit position to the right the number of times specified by the count operand
  - zeros fill vacated positions at the H.O. bit
  - L.O. bit shifts into the carry flag
- a quick way to divide by two
- useful in *unpacking data,* e.g., suppose you want to extract the two nibbles in the AL register, leaving the H.O. nibble in AH and the L.O. nibble in AL

```
mov  ah, al
shr  ah, 4
and  al, 0Fh
```

- the 8086 and 8088 allow an immediate shift of 1 only

SHL 0

Shift arithmetic right

- Syntax: SAR reg, count (immediate count)
  SAR reg, CL (count stored in CL)
  - moves each bit of the operand one bit position to the right the number of times specified by the count operand
  - H.O. bit replicates
  - L.O. bit shifts into the carry flag
- main purpose is to perform a *signed division* by some power of two

```
mov  ax, -15
sar  ax, 1                ; result is -8
```

- In 80286 and later you can use SAR to sign extend one register into another

```
mov  ah, al
sar  ah, 7
```

if AL contains 1111 0001 then AX will be 1111 1111 1111 0001 after execution

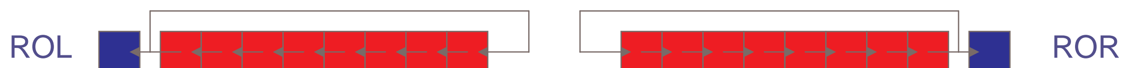
SAR

Rotate through carry L/R

- Syntax: RCL reg, count (immediate count)
  RCL reg, CL (count stored in CL)
  - rotates bits to the left, through the carry flag
  - bit in the carry flag is written back into bit zero on the right
- Syntax: RCR reg, count (immediate count)
  RCR reg (count stored in CL)
  - rotates bits to the right, through the carry flag
  - bit in the carry flag is written back into the H.O. bit on the left

Rotate left/right

- Syntax: ROL reg, count (immediate count)
  ROL reg, CL (count stored in CL)
  - rotates bits to the left
  - H.O. bit goes to carry flag and L.O. bit
- Syntax: ROR reg, count (immediate count)
  ROR reg (count stored in CL)
  - rotates bits to the right
  - L.O. bit goes to carry flag and H.O. bit



Examples

```
mov ax,3              ; Initial register values     AX = 0000 0000 0000 0011
mov bx,5              ;                             BX = 0000 0000 0000 0101
or ax,9               ; ax <- ax | 0000 1001        AX = 0000 0000 0000 1011
and ax,10101010b      ; ax <- ax & 1010 1010        AX = 0000 0000 0000 1010
xor ax,0FFh           ; ax <- ax ^ 1111 1111        AX = 0000 0000 1111 0101
neg ax                ; ax <- (-ax)                 AX = 1111 1111 0000 1011
not ax                ; ax <- (~ax)                 AX = 0000 0000 1111 0100
or ax,1               ; ax <- ax | 0000 0001        AX = 0000 0000 1111 0101
shl ax,1              ; logical shift left by 1 bit AX = 0000 0001 1110 1010
shr ax,1              ; logical shift right by 1 bit AX = 0000 0000 1111 0101
ror ax,1              ; rotate right (LSB=MSB)      AX = 1000 0000 0111 1010
rol ax,1              ; rotate left (MSB=LSB)       AX = 0000 0000 1111 0101
mov cl,3              ; Use CL to shift 3 bits      CL = 0000 0011
shr ax,cl             ; Divide AX by 8              AX = 0000 0000 0001 1110
mov cl,3              ; Use CL to shift 3 bits      CL = 0000 0011
shl bx,cl             ; Multiply BX by 8            BX = 0000 0000 0010 1000
```


Multiplication Using Shift Operations

```
; multiply AX by the decimal 10 (1010b)
; (multiply by 2 and 8, then add results)
    shl ax, 1       ; AX x 2
    mov bx, ax      ; save 2AX
    shl ax, 2       ; 2 x (original AX) x 4 = 8 x (orig AX)
    add ax, bx      ; 2AX + 8AX = 10AX

; multiply AX by decimal 18 (10010b)
; (multiply by 2 and 16, then add results)
    shl ax, 1       ; AX x 2
    mov bx, ax      ; save 2AX
    shl ax, 3       ; 2AX x 2 x 2 x 2 = 16AX
    add ax, bx      ; 2AX + 16AX = 18AX
```