Conditional Structures

Computer Organization and Assembly Language

Computer Science Department

National University of Computer and Emerging Sciences Islamabad

Agenda for today

- Arithmetic Instructions
- Labels
- Conditional and Unconditional Flow of Program
- JMP instruction
- Loops

Arithmetic Instructions

- MUL *operand1*
  - operand1 is 8-bit (type BYTE): multiply with contents placed in register AL, result will be in AX.
  - operand1 is 16-bit (type WORD): multiply with contents placed in register AX, result will be in DX:AX.
  - operand1 is 32-bit (type DWORD): multiply with contents placed in register DX:AX, result will be in EDX:EAX.
  - Example

```
MOV AL, 200 ;
MOV BL, 4
MUL BL ;
```

MUL Instruction

Multiplicand	Multiplier	Product
al	reg/m8	ax
ax	reg/m16	dx:ax
eax	reg/m32	edx:eax

Arithmetic Instructions

- DIV *operand1*
  - when operand is a BYTE: AL = AX / operand then
    - **AL =Quotient**
    - **AH = remainder**
  - Example

```
MOV AX, 203;
MOV BL, 4;
DIV BL ; AL = 50 , AH = 3
```

DIV Instruction

- when operand is a word: AX = (DX AX) / operand then
  - **AX = Quotient**
  - **DX = remainder**

Dividend	Divisor	Quotient	Remainder
ax	reg/m8	al	ah
dx:ax	reg/m16	ax	dx
edx:eax	reg/m32	eax	edx

Divide overflow

- If a division operand produces quotient that is larger than storage then divide overflow condition occurs and the program terminates.
  - **MOV AX,1000h**
  - **MOV BL, 10h**
  - **DIV BL** ;AL cannot hold 100h
- Similarly when you divide by zero program terminates.

Labels in Assembly

- A label is an identifier that is followed by a colon.
- Names suffixed with colons (:) are symbolic labels.
- The labels do not create code.
- They are simply a way to tell the assembler that those locations have symbolic names.
- Example
  - Label1:
  - Above:
  - Start:

JMP Instruction

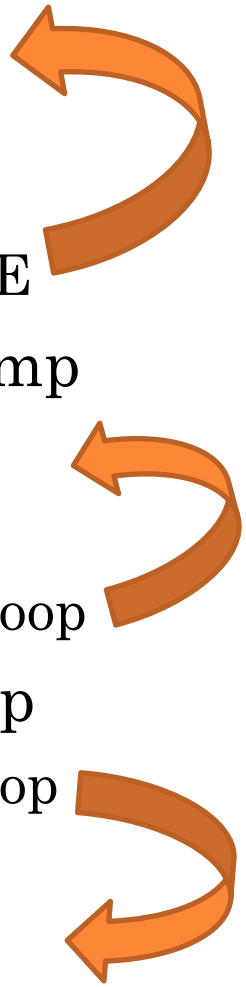
- Unconditional jump
- Jumps to the given label[Address]
- Syntax:
  - JMP [Label] ;PC will jump to the address of Label

Examples

- Jump
  - ABOVE:
    - mov ah,09
    - mov dh,32
  - JMP ABOVE
- Backward jump
  - StartLoop:

  ; statements
  - JMP StartLoop
- Forward jump
  - JMP EndLoop

  ; statements
  - EndLoop:



Conditional Jumps

- Transfers Control if condition is satisfied.
- Jumps based on unsigned data
- Jumps to label if equal
  - JE [label]
- Jump if not equal
  - JNE [label]

Jumps Based on Equality

Mnemonic	Description
JE	Jump if equal (*leftOp* = *rightOp*)
JNE	Jump if not equal (*leftOp* ≠ *rightOp*)
JCXZ	Jump if CX = 0
JECXZ	Jump if ECX = 0

CMP Instruction

- Compares value of two operands and set flags
- Syntax
  - CMP opr1,opr2
  - Opr1 = reg/mem
  - Opr2 = reg/mem/imme
- Example
  - CMP ax,bx
  - CMP ah,al

JUMP Conditions

JA/JNBE	(CF and ZF) = 0	Above / Not Below or Equal
JAE/JNB	CF = 0	Above or Equal / Not Below
JB/JNAE/JC	CF = 1	Below / Not Above or Equal / Carry
JBE/JNA	(CF or ZF) = 1	Below or Equal / Not Above
JE/JZ	ZF = 1	Equal / Zero
JMP	none	Unconditionally
JNC	CF = 0	No Carry
JNE/JNZ	ZF = 0	Not Equal / Not Zero
JNO	OF = 0	No Overflow
JNP/JPO	PF = 0	No Parity / Parity Odd
JNS	SF = 0	No Sign / Positive
JO	OF = 1	Overflow
JP/JPE	PF = 1	Parity / Parity Even
JS	SF = 1	Sign
JG/JNLE	ZF = 0 and SF = OF	Greater / Not Less nor Equal
JGE/JNL	SF = OF	Grater or Equal / Not Less
JL / JNGE	SF <> OF	Less / Not Greater nor Equal
JLE/JNG	(ZF = 1) or (SF <> OF)	Less or equal / not greater
JCXZ	Register CX = 0	CX is equal to zero

- Example:
  - …
  - Mov al,5
  - Cmp al,5
  - Je label1
  - Jmp exit
  - Label1:
    - Mov ax,bx
    - …
  - Exit: …

Jumps Based on Flags

Mnemonic	Description	Flags
JZ	Jump if zero	ZF = 1
JNZ	Jump if not zero	ZF = 0
JC	Jump if carry	CF = 1
JNC	Jump if not carry	CF = 0
JO	Jump if overflow	OF = 1
JNO	Jump if not overflow	OF = 0
JS	Jump if signed	SF = 1
JNS	Jump if not signed	SF = 0

Jumps Based on Conditions

Instruction	Description	Flags
Ja	If op1>op2	Cf=0 and zf=0
Jnbe	if op1 not <=op2	Cf=0 and zf=0
Jae	If op1>=op2	Cf=0
Jnb	If op1 not<op2	Cf=0
Jb	If op1<op2	Cf=1
Jnae	If op1not >=op2	Cf=1
Jbe	If op1<=op2	Cf=1 and zf=1

YOUR TURN . . .

Implement the following pseudocode in assembly language. All values are unsigned:

```
if( bx <= ax
  && cx > dx )
    {
   ax = 5;
   dx = 6;
    }
```

(There are multiple correct solutions to this problem.)

Block-Structured IF Statements

- Assembly language programmers can easily translate logical statements written in C++/Java into assembly language. For example:

```
if( op1 == op2 )
  X = 1;
else
  X = 2;
```

```
    mov ax,op1
    cmp ax,op2
    jne L1
    mov X,1
    jmp L2
L1:
    mov X,2
L2:
```

YOUR TURN . . .

Implement the following pseudo code in assembly language. All values are 16-bit signed integers:

```
if( var1 == var2 )
     var3 = 10;
       else
         {
       var3 = 6;
       var4 = 7;
          }
```

(There are multiple correct solutions to this problem.)

Another Example

```
if (al > bl) AND (bl > cl)
  X = 1;
```

```
    cmp al,bl                ; first expression...
    jbe next                 ; quit if false
    cmp bl,cl                ; second expression...
    jbe next                 ; quit if false
    mov X,1                  ; both are true
next:
```

And Another

```
mov ax,var1
cmp ax,var2
je L1
mov var3,6
mov var4,7
jmp L2
 L1: mov var3,10
 L2:
```

```
if( var1 == var2 )
     var3 = 10;
       else
         {
        var3 = 6;
        var4 = 7;
         }
```

Loop Instruction

- CX used as Counter
- Automatically Decremented after every iteration
- Label is used to jump
- Ends when counter reaches zero
- Example:
  - MOV cx,10 ;loop runs 10 times
  - Loop1:
    - …
  - Loop Loop1

WHILE LOOPS

A WHILE loop is really an IF statement followed by the body of the loop, followed by an unconditional jump to the top of the loop. Consider the following example:

```
while( ax < bx)
    ax = ax + 1;
```

This is a possible implementation:

```
top:cmp ax,bx                ; check loop condition
    jae next                 ; false? exit loop
    inc ax                   ; body of loop
    jmp top                  ; repeat the loop
next:
```

Another Example

Implement the following loop

```
while( bx <= val1)
{
    bx = bx + 5;
    val1 = val1 - 1
}
```

```
top: cmp bx,val1              ; check loop condition
     ja  next                 ; false? exit loop
     add bx,5                 ; body of loop
     dec val1
     jmp top                  ; repeat the loop
next:
```