

---

# Lecture 4: Comparing and Branching

Dragan Mirkovic  
Department of Computer Science  
University of Houston

---

D. Mirkovic, COSC 2410: Computer Organization and Programming, Fall 2003

## Announcements

---

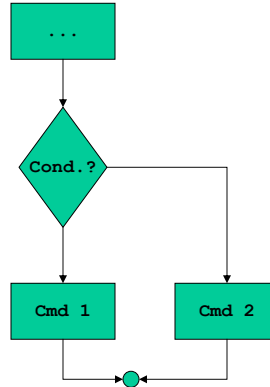
- Quiz scores are available on the course web page.
- Quiz #2 02/17/2004 (check the quiz schedule on the web)
  - Integer arithmetic and basic operations
    - Ch. 2 and 4.
- Today we will talk about the instructions for controlling the program flow
  - Comparing and Branching (Ch. 5 in text)

---

D. Mirkovic, COSC 2410: Computer Organization and Programming, Fall 2003

# Introduction

- All computer languages have instructions for controlling the program flow
- One can divide these into
  - Conditional and unconditional jumps
  - Loops
  - Subroutine calls
- Loops and subroutine calls could be implemented using jumps
- We will start with conditional jump instructions first.
  - Two step process
    - Comparison
    - jump
- Typical **if-then-else** construct:



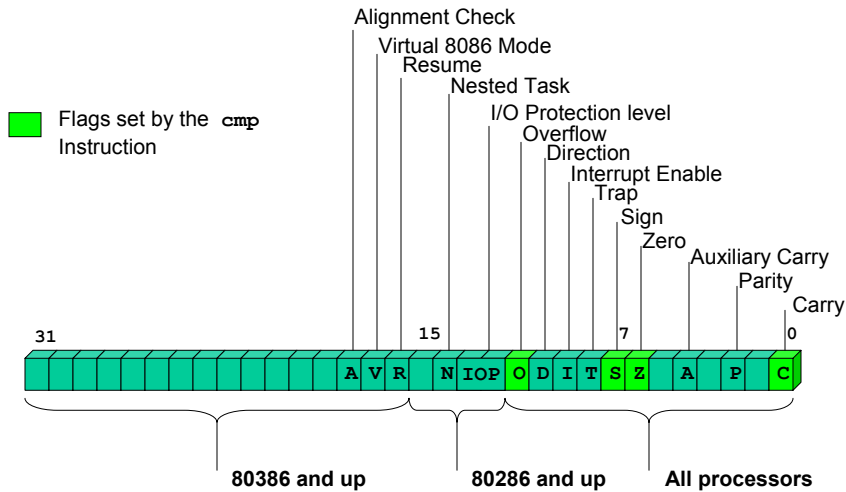
D. Mirkovic, COSC 2410: Computer Organization and Programming, Fall 2003

# Conditional Jumps

- All conditional jumps except two (jcxz and jcxnz) use the processor flags for their criteria.
- All of the conditional jump instructions come in pairs
  - `j<condition>`                      `j<not condition>`
- There are 30 conditional-jump instructions.
- Several of them have two or three names which assemble to exactly the same machine instruction.
  - Example: `je` (Jump if Equal) and `jz` (Jump if Zero)
- You may choose whichever mnemonic seems more appropriate for a given situation
- Use of conditional jumps is a two step process:
  1. Test the condition – results in some of the flags being set
  2. Jump if the condition is true or continue if it is false

D. Mirkovic, COSC 2410: Computer Organization and Programming, Fall 2003

# FLAGS Register



D. Mirkovic, COSC 2410: Computer Organization and Programming, Fall 2003

# Flags Description

- **Carry:** Set if an operation generates a carry to or a borrow from a destination operand
- **Parity:** Set if the low-order bits of the result of an operation contain an even number of set bits
- **Auxiliary Carry:** Carry or borrow from the low-order 4 bits of an operand
- **Zero:** Set if the result of an operation is 0.
- **Sign:** Equal to the high-order bit of the result of an operation.
- **Trap:** If set, the processor generates a single-step interrupt after each instruction.
- **Interrupt enable:** Enables or disables interrupts.
- **Direction:** If set the string operations process down from high addresses to low addresses.
- **Overflow:** Set if the result of an operation is too large or too small to fit the destination operand.

D. Mirkovic, COSC 2410: Computer Organization and Programming, Fall 2003

## Jumping based on the Processor Flags

Instruction	Jumps if
JC/JB/JNAE	CF is set (below, not above and equal)
JNC/JNB/JAE	CF is clear (not below, above or equal)
JBE/JNA	Either CF or ZF is set (below or equal, not above)
JNBE/JA	CF or ZF are clear (not below or equal, above)
JZ/JE	ZF is set (equal)
JNZ/JNE	ZF is clear (not equal)
JL/JNGE	SF <> OF (less, not greater than or equal)
JNL/JGE	SF = OF (not less, greater than or equal)
JO	OF is set
JNO	OF is clear
JS	SF is set
JNS	SF is clear

D. Mirkovic, COSC 2410: Computer Organization and Programming, Fall 2003

## Decision-making in MASM

- In IBM PC assembly language decision making is a two step process
  1. Two numbers are compared using the compare instruction `cmp`
    - Compare instruction sets the flags in the flags register
  2. A conditional jump instruction is executed which may change the next instruction in the program
- `cmp` instruction format

Limitation: at most one memory operand per instruction

D. Mirkovic, COSC 2410: Computer Organization and Programming, Fall 2003

# CMP Instruction

---

- The instruction:

```
cmp    op1, op2
```

1. Performs the subtraction  $op1 - op2$ ,
2. Sets the flags according to the result,
3. Discards the result.

- The flags set are:

**Overflow Flag (OF):** Set if the result doesn't fit the destination.

**Sign Flag (SF):** Set to the sign of the result.

**Zero Flag (ZF):** Set if the result is zero.

**Carry Flag (CF):** Set if the result produces a carry.

---

D. Mirkovic, COSC 2410: Computer Organization and Programming, Fall 2003

# Conditional jumps

---

- For each of the flags above there are two conditional jumps:

jo and jno for OF

js and jns for SF

jz and jnz for ZF

jc and jnc for CF

- These are too complicated in usual situations in which we would like to compare  $op1$  and  $op2$  using  $<=>$ .

---

D. Mirkovic, COSC 2410: Computer Organization and Programming, Fall 2003

## Signed numbers jumps

---

- For signed numbers we would like to have instructions corresponding to the usual if/then/else constructs with the simple mnemonics for logical operators
- Example: after `cmp op1, op2`

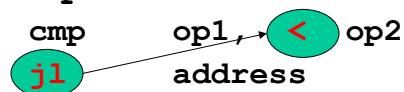
<code>je &lt;address&gt;</code>	Jump if equal	<code>op1 = op2</code>
<code>jne &lt;address&gt;</code>	Jump if not equal	<code>op1 &lt;&gt; op2</code>
<code>jg &lt;address&gt;</code>	Jump if greater	<code>op1 &gt; op2</code>
<code>jge &lt;address&gt;</code>	Jump if greater or equal	<code>op1 &gt;= op2</code>
<code>j1 &lt;address&gt;</code>	Jump if less	<code>op1 &lt; op2</code>
<code>jle &lt;address&gt;</code>	Jump if less or equal	<code>op1 &lt;= op2</code>

D. Mirkovic, COSC 2410: Computer Organization and Programming, Fall 2003

## Examples

---

- You can think of the relation in conditional jump as sitting between the operands of the `cmp` instruction



- This is much simpler than evaluating the flags
- The address in the jump instructions is the new value of the instruction pointer IP
- Also it is much easier to use labels than the actual addresses

D. Mirkovic, COSC 2410: Computer Organization and Programming, Fall 2003

# Examples...

---

- If/then/else construct

- Example:

```
if (ax < bx) then X=-1 else X = 1
```

- Can be implemented as:

```
      cmp     ax, bx
      j1     axLess; go to axLess if ax < bx
      mov    X, 1 ; X = 1 if ax >= bx
      jmp    Both ;
axLess: mov    X, -1; X = -1 if ax < bx
Both:
```

- It uses unconditional jump instruction `jmp`
- 

D. Mirkovic, COSC 2410: Computer Organization and Programming, Fall 2003

# Notes

---

- Branching in assembly language can be quite complicated
- Do not forget to jump out of the true condition!
- It is useful if you can draw simple diagrams before writing the actual code
- Example:

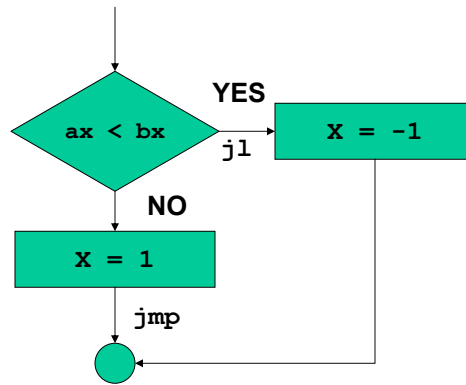
```
if (ax < bx) then X=-1 else X = 1
```

false

---

D. Mirkovic, COSC 2410: Computer Organization and Programming, Fall 2003

# Flow Chart



D. Mirkovic, COSC 2410: Computer Organization and Programming, Fall 2003

# More Examples

- If/then construct

- Example:

if (X > XMAX) then X = 0 **false**

- Can be implemented as:

```
mov    ax,X
cmp    ax, XMAX
jng    Lab;    go to Lab if not X > XMAX
mov    X, 0;    X = -1 if ax < bx
```

Lab: ; end if

- Sometimes it is simpler to use false jump condition for construction
  - Luckily there are mnemonics for all possible situations

D. Mirkovic, COSC 2410: Computer Organization and Programming, Fall 2003



# Code Optimization

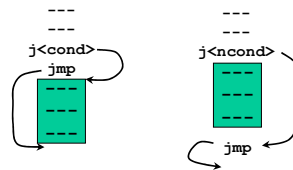
- Our original code can be simplified
- We start with \_\_\_\_\_
- So, finally :

```

mov ax, A
cmp ax, B
jnl Lab1
cmp B, 14
jge Lab2
Lab1:
mov ax, B
Lab2:
mov X, ax ; X = A or B

```

1. The second `mov` is unnecessary: `A` is already in `ax`
2. Both if branches end in instruction `mov X, ax` which can be 'factored' out
3. Two jumps can be replaced by a single jump of the form: `jge Lab2`



D. Mirkovic, COSC 2410: Computer Organization and Programming, Fall 2003

# Exercises 5.1

5.1-5. Write an assembly program that converts a number in the word variable `D`,  $0 \leq D \leq 15$  to an ASCII character in the byte variable `HEXD` which is its hex value. For instance, if `D = 3` then `HEXD = '3'` and if `D = 14` then `HEXD = 'E'`.

## Solution:

Remember

```

'0' - '9' = 30h 39h,
'A' = 41h, and 'a' = 61h

```

Logic:

```

if (D>9) then
    HEXD = D - 10 + 41h
//or HEXD = D - 10 + 61h
else
    HEXD = D + 30h
end

```

```

INCLUDE PCMAC.INC
.MODEL SMALL
.586
.STACK 100h
.DATA
D DW 10
Hstr DB 'Hex = '
HEXD DB ?
CrLf DB 13,10, '$'
.CODE
EXTERN PutDec : NEAR
Hx PROC
    _Begin
    mov ax, D
    cmp ax, 9
    jg Letter
    add al, '0'
    jmp Store
Letter:
    add al, 'A' - 10;
Store:
    mov HEXD, al
    _PutStr Hstr
    _Exit 0
Hx ENDP
END Hx

```

D. Mirkovic, COSC 2410: Computer Organization and Programming, Fall 2003

## Unsigned conditional jumps

---

- Sometimes we have to use unsigned forms of conditional jumps
  - Example: address calculations
- The integer arithmetic (`sub`) is the same
- The jump conditions are not the same
- Different flags need to be tested

---

D. Mirkovic, COSC 2410: Computer Organization and Programming, Fall 2003

## Unsigned numbers jumps

---

- For unsigned numbers after `cmp op1, op2`

<code>ja &lt;address&gt;</code>	Jump if above	<code>op1 &gt; op2</code>
<code>jae &lt;address&gt;</code>	Jump if above or equal	<code>op1 ≥ op2</code>
<code>jb &lt;address&gt;</code>	Jump if below	<code>op1 &lt; op2</code>
<code>jbe &lt;address&gt;</code>	Jump if below or equal	<code>op1 ≤ op2</code>

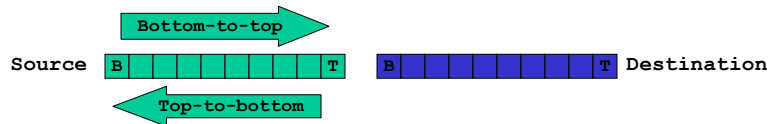
- Always use unsigned jumps when comparing addresses
- Example:
  - 7000h > 8000h as a signed condition (8000h is negative), but
  - 8000h > 7000h in the unsigned case.

---

D. Mirkovic, COSC 2410: Computer Organization and Programming, Fall 2003

# Example

- Move a block of bytes starting at an address in  $ax$  to a block starting at an address in  $bx$ , with the size of the block in  $cx$ .
- There are two choices of how to scan the source and the destination:
  - Top-to-bottom or bottom-to-top
- If the blocks don't overlap either way will work



D. Mirkovic, COSC 2410: Computer Organization and Programming, Fall 2003

# Example...

- If the blocks overlap caution is needed not to overwrite the source before it is copied

	Correct	Wrong!
Source below Destination		
Source above Destination		

- Correct solution requires a comparison of the addresses in  $ax$  and  $bx$  and a jump to a different move block

D. Mirkovic, COSC 2410: Computer Organization and Programming, Fall 2003

# Solution

---

- Bad solution:

```
        cmp     ax, bx
        jl     MoveTopDown
MoveBottUp:
```

- because `jl` uses a signed comparison

- Correct solution:

```
        cmp     ax, bx
        jb     MoveTopDown
MoveBottUp:
```

- Similarly, for comparisons of ASCII character values one should use unsigned jumps.

- Example:

```
cmp al, '\ '
jb  CtrlChar ; Correct (jl is wrong)
```

---

D. Mirkovic, COSC 2410: Computer Organization and Programming, Fall 2003

# Exercises 5.2

---

- In many cases it doesn't really matter which kind of jumps you use when comparing characters.

- Example:

- Range test for digits:  $'0' \leq \text{digit} \leq '9'$

**Correct and good programming**      **Correct, but bad programming**

```
        cmp     al, '0'
        jb     notDigit
        cmp     al, '9'
        jbe     isDigit
        cmp     al, '0'
        jl     notDigit
        cmp     al, '9'
        jle     isDigit
```

- For positive values the signed and the unsigned jumps have the same effect.
- `jl` filters out the negative values (`jb` doesn't!)

---

D. Mirkovic, COSC 2410: Computer Organization and Programming, Fall 2003

# Arithmetic and the Flags

---

- There are other instructions that we can use in order to set the flags
- Most of the arithmetic instructions may change the flags depending on the result of the operation
  - `add`, `sub`, `inc`, `dec`, and `neg` set the flags based on the result of the operation
  - `mul` and `imul` set the OF and CF to 1 if there are any significant bits of the product in `dx`.
  - `div` and `idiv` leave flags in an unpredictable state
- Check the Reference Manual for details
  - Example: `add` and `sub` change O, S, Z, A, C flags.
- `mov` NEVER alters the flags!

---

D. Mirkovic, COSC 2410: Computer Organization and Programming, Fall 2003

## Example

---

- Exercises 5.3.
  - lower case characters are between 'a' and 'z'

```
sub    al, 'a'
```

```
cmp    al, 0
```

```
jl     notLc
```
  - The flags after `add`, `sub`, `inc`, `dec`, and `neg` are set as if each of these instructions were followed by `cmp al, 0`, hence

```
sub    al, 'a'
```

```
jl     notLc
```
  - Remember that `sub` will change the content of `al` which is sometimes convenient and sometimes not!

---

D. Mirkovic, COSC 2410: Computer Organization and Programming, Fall 2003

# Counting Loops

---

- Loops can be coded by using compare and branch instructions
- 80x86 instruction set has additional instructions that simplify the construction of some standard loop types
- In some cases, however, the `cmp/jmp` loop is faster!
- The **counting loop**:
  - Execute the *body* of the loop  $N > 0$  times

```
        mov    cx, N;
theLoop:
        <body>
        dec    cx;
        jnz   theLoop; or jg
```

---

D. Mirkovic, COSC 2410: Computer Organization and Programming, Fall 2003

# Loop Examples

---

- The `jnz` test is unsafe but fast
  - If  $N \leq 0$ , the loop doesn't terminate.
  - While  $N < 0$  is meaningless,  $N = 0$  makes sense in some cases and `jcxz` can be used to prevent the infinite loop for  $N = 0$ .
- Example: Factorial computation  $N! = N(N-1)\dots 1$ . ( $0! = 1$  by definition).

```
        mov    cx, N; Counter and multiplier
        mov    ax, 1; Initialize ax
        jcxz   Done ; 0! = 1

theLoop:
        mul    cx ;
        dec    cx;
        jnz   theLoop;

Done:
```

---

D. Mirkovic, COSC 2410: Computer Organization and Programming, Fall 2003

# Reading Single Characters

---

- Keyboard input is a good example of layering of hardware and BIOS and DOS software.
- The hardware sends a keyboard **scan code** when the key is pressed and released.
- The BIOS processes the key presses and releases
  - It takes into account the shift keys
  - It uses the interrupt 9 which converts each 'real' keystroke into a two-byte pair
    - ASCII value (or 0 if there isn't any) + scan code
  - These are saved in a 16 entry circular queue (two bytes per entry) starting at 40h:1eh

---

D. Mirkovic, COSC 2410: Computer Organization and Programming, Fall 2003

# Keyboard Basics

---

- The keyboard buffer uses the following variables:

```
40:1a - HeadPtr DW ?           ; Head
40:1c - TailPtr DW ?           ; Tail
40:1e - Buffer   DW 16 dup (?) ; Buffer
```

– Example

```
>debug
-d 40:1a
0040:0010                22 00 22 00 61 1E                ".a.
0040:0020 0D 1C 64 20 34 05 30 0B-31 02 61 1E 08 0E 08 0E  ...d 4.0.1.a....
0040:0030 08 0E 08 0E 08 0E 34 05-30 0B 3A 27 31 02 00 80  ....4.0.:1...
```

---

D. Mirkovic, COSC 2410: Computer Organization and Programming, Fall 2003

# Hardware/Software Interface

---

- The actions of the BIOS and hardware happen independently of the program
- The program reads the characters from the keyboard by using the BIOS or the DOS call
- DOS is layered on top of the BIOS
- The BIOS call returns the ASCII value and the scan code of the first character in the buffer and deletes it from the buffer
- DOS add functionality by allowing an echo of the keystroke to the screen

---

D. Mirkovic, COSC 2410: Computer Organization and Programming, Fall 2003

# BIOS Keyboard Interface

---

- MS-DOS BIOS keyboard services are implemented through the `int 16h` instruction (`ah = 0 - 12h`)
- Example

Function # (AH)	Input	Output	Description
0	-	<code>a1</code> - ASCII character <code>ah</code> - scan code	Read character. Reads next available character from the system's type ahead buffer. Wait for a keystroke if the buffer is empty.
1	-	<code>ZF</code> - Set if no key. <code>ZF</code> - Clear if key available. <code>a1</code> - ASCII code <code>ah</code> - scan code	Checks to see if a character is available in the type ahead buffer.

- See the example 5.5-1 in the text

---

D. Mirkovic, COSC 2410: Computer Organization and Programming, Fall 2003

# Homework

---

- Read Ch. 5 in Jones
- Problems:  
pp. 109, 1(a,d), 2a, 3.  
pp. 114, 1.