Computer Organizations and Architecture

Flow Control Instructions – Part 1

Spring 24-25, CS 3205, Section D

Dr. Nazib Abdun Nasir

Assistant Professor, CS, AIUB

nazib.nasir@aiub.edu

April 28 & 30, 2025

Outline

- Flow Control
- Flow Control Instructions
- The CMP Instruction
- Conditional Jump
- The JNZ Instruction
- JNZ Example (High-Level)
- Assembly Language Coding Practice (JNZ)
- How the CPU Implements a Conditional Jump
- Implementing Control Structures
- Assembly Language Coding Practice (Loop)
- Singed v/s Unsigned Jumps
- Working with Characters
- The JMP Instruction
- JMP v/s JNZ

Flow Control Instructions

Flow Control

- Flow control in assembly language refers to the mechanisms that dictate the order in which instructions are executed.
- It allows programmers to implement decision-making and repetition in their programs.
- The primary tools for flow control in assembly are **loop** & **jump** instructions, which can be categorized into **unconditional** and **conditional** jumps.
  - → They transfer the execution control to another part of the program.

Flow Control Instructions

- Unconditional Jump (JMP):
  - → This instruction directs the program to jump to a specified address without any conditions.
  - → JMP label ; Jumps to the instruction at 'label'
- Conditional Jumps:
  - → These instructions alter the flow based on the evaluation of certain conditions, typically involving status flags set by previous operations.
  - → JE (Jump if Equal): Jumps if two values are equal.
  - → JNE (Jump if Not Equal): Jumps if two values are not equal.
  - → JG (Jump if Greater): Jumps if one value is greater than another.
  - → JL (Jump if Less): Jumps if one value is less than another.
  - → JNZ (Jump if Not Zero): Jump if the Zero Flag (ZF) is not set.

The CMP Instruction

- The jump condition is often provided by the CMP (compare) instruction.
- The CMP instruction is used to compare two operands by performing a subtraction operation, without modifying the operands themselves.
- Instead, the CMP instruction sets various status flags that can be used for conditional branching in the program flow.
  - → Result = Destination − Source
  - → The result of this subtraction is **NOT** stored; instead, it updates several flags in the processor's status register.

The CMP Instruction

→ Zero Flag (ZF): Set to 1 if the result is zero (i.e., the two operands are equal).

→ Carry Flag (CF): Set to 1 if there was a borrow in the subtraction (i.e., the destination operand is less than the source operand).

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

- MOV AX, 5
- MOV BX, 3
- CMP AX, BX

→ Since AX (5) is greater than BX (3), ZF will be set to 0 and CF will be set to 0.

Conditional Jump

- The CMP instruction is often used before a conditional jump.
  - → MOV AX, 10   ; Load 10 into AX
  - → CMP AX, 10   ; Compare AX with 10
  - → JE equal_label   ; Jump to equal_label if AX equals 10
- CMP can also control loops based on a counter.
  - → MOV CX, 0   ; Initialize counter in CX
  - → loop_start:
  - →   INC CX   ; Increment counter
  - →   CMP CX, 5   ; Compare counter with 5
  - →   JLE loop_start   ; Jump back if CX is less than or equal to 5

The JNZ Instruction

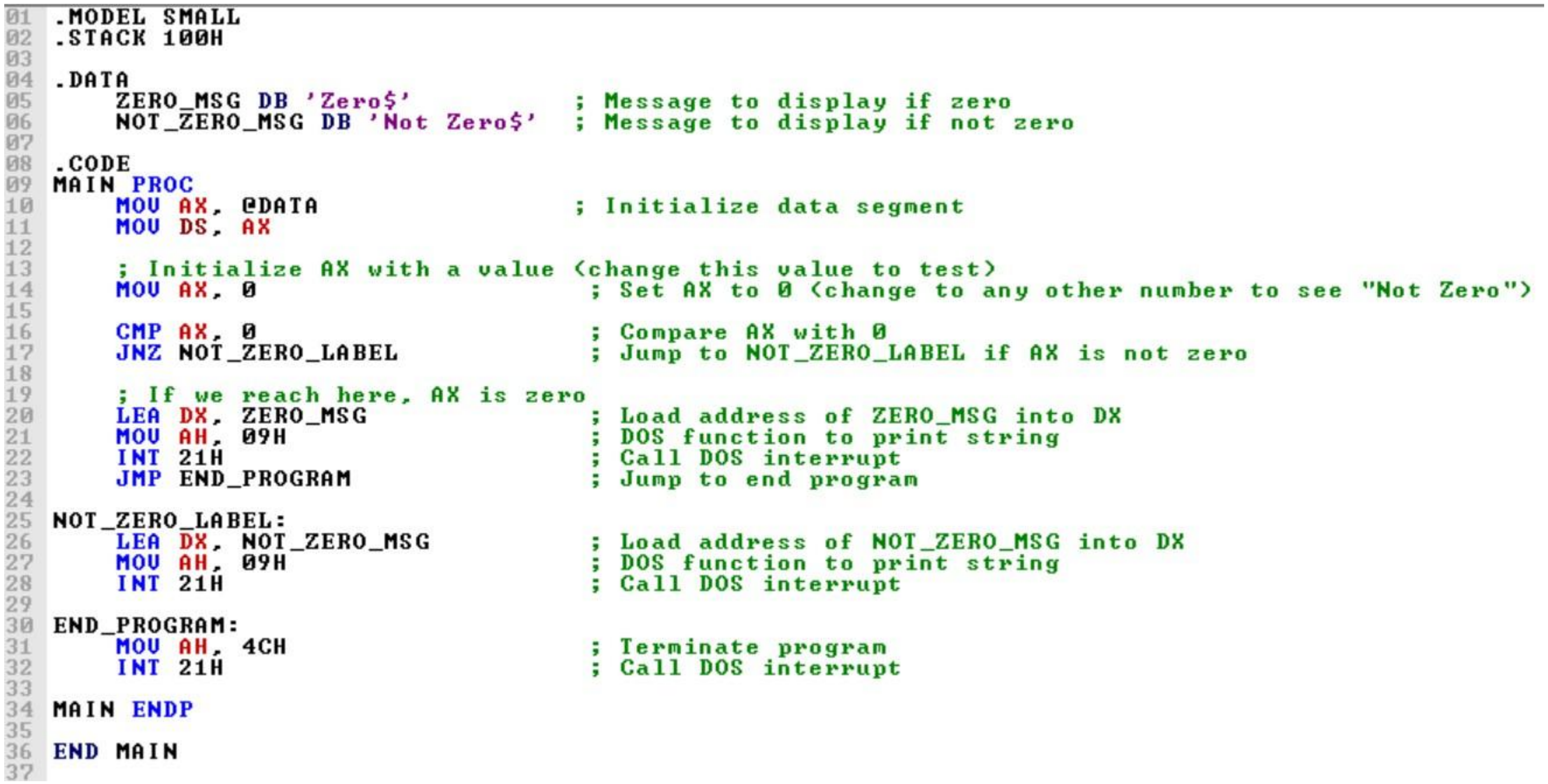
- JNZ, which stands for **Jump if Not Zero**, is a conditional jump instruction that directs the flow of execution to a specified label or address if the Zero Flag (ZF) is not set.
  - → It means that the result of the previous operation was not zero.
  - → If the condition for the jump is true, the next instruction to be executed is at destination_label.
  - → If the condition is false (preceding instruction is non-zero), the instruction following the jump is done next.
  - → Labels are needed to refer another instruction and labels end with colon (:).
  - → Labels are placed on a line by themselves to make it stand out.

JNZ Example (High-Level)

```
MOV AX, 1              ; Load 1 into AX register
CMP AX, 0              ; Compare AX with 0
JNZ NOT_ZERO           ; Jump to 'NOT_ZERO' if ZF is not set (AX is not zero)
PRINT 'ZERO.'          ; This line executes if AX is zero
JMP EXIT               ; Jump to exit
NOT_ZERO:
PRINT 'NOT ZERO’       ; This line executes if AX is not zero
EXIT:
RET                    ; Return control to the OS
```

Write the Complete Assembly Language Code that Illustrates the Previous Example

JNZ Example – Full Assembly Code



```
.MODEL SMALL
.STACK 100H

.DATA
    ZERO_MSG DB 'Zero$'          ; Message to display if zero
    NOT_ZERO_MSG DB 'Not Zero$'  ; Message to display if not zero

.CODE
MAIN PROC
    MOV AX, @DATA                ; Initialize data segment
    MOV DS, AX

    ; Initialize AX with a value (change this value to test)
    MOV AX, 0                     ; Set AX to 0 (change to any other number to see "Not Zero")

    CMP AX, 0                     ; Compare AX with 0
    JNZ NOT_ZERO_LABEL            ; Jump to NOT_ZERO_LABEL if AX is not zero

    ; If we reach here, AX is zero
    LEA DX, ZERO_MSG              ; Load address of ZERO_MSG into DX
    MOV AH, 09H                   ; DOS function to print string
    INT 21H                       ; Call DOS interrupt
    JMP END_PROGRAM               ; Jump to end program

NOT_ZERO_LABEL:
    LEA DX, NOT_ZERO_MSG          ; Load address of NOT_ZERO_MSG into DX
    MOV AH, 09H                   ; DOS function to print string
    INT 21H                       ; Call DOS interrupt

END_PROGRAM:
    MOV AH, 4CH                   ; Terminate program
    INT 21H                       ; Call DOS interrupt

MAIN ENDP

END MAIN
```

How the CPU Implements a Conditional Jump

- The CPU looks at the FLAGS register (it reflects the result of last thing that processor did).
- If the conditions for the jump (combination of status flag settings) are true, the CPU adjusts the IP to point to the destination label.
- The instruction at this label will be done next.
- If the jump condition is false, then IP is not altered and naturally the next instruction is performed.
- The structure of the machine code of a conditional jump requires that destination label must precede the jump instruction by no more than 126 bytes or follow it by no more than 127 bytes.

Implementing Control Structures

- Control structures such as if-else statements and loops can be implemented using these jump instructions.
- **If-Else Structure:** To implement an if statement, we typically use a comparison followed by a conditional jump.

```
→ CMP AX, 0          ; Compare AX with 0
→ JGE POSITIVE       ; Jump to 'POSITIVE' if AX >= 0
→ MOV BX, -1         ; Else, set BX to -1
→ JMP END_IF         ; Jump to END_IF
```

Implementing Control Structures

- **Loops Structure:** Loops are created by combining jump instructions with conditions. A common structure is a *while* loop.
  - → MOV CX, 10 ; Initialize counter
  - → LOOP_START:
  - → CMP CX, 0 ; Check if counter is zero
  - → JE LOOP_END ; Exit loop if zero
  - → ; Perform loop body actions here
  - → DEC CX ; Decrement counter
  - → JMP LOOP_START ; Repeat loop
  - → LOOP_END:

Singed v/s Unsigned Jumps

- Signed Jumps
  - → Signed jumps are used when the values being compared are interpreted as signed integers. This means that the most significant bit (MSB) indicates the sign of the number (0 for positive, 1 for negative).
  - → Overflow Flag (OF), Sign Flag (SF), Zero Flag (ZF); JG, JL, JGE, JLE.
- Unsigned Jumps
  - → Unsigned jumps, on the other hand, are used when interpreting values as unsigned integers, where all bits contribute to the magnitude of the number.
  - → Carry Flag (CF), Zero Flag (ZF); JA, JB, JAE, JBE.
- Differences
  - → Signed jumps treat numbers as having both positive and negative values, which affects how comparisons are made.
  - → Unsigned jumps treat all numbers as non-negative, which changes the logic for determining relationships between values.

Working with Characters

- To deal with ASCII character set, either Signed or Unsigned jumps may be used.
  - → i.e. sign bit of a byte in a character is always zero.

- However, while comparing extended ASCII characters (80H to FFH), UNSIGNED jumps should be used.

The JMP Instruction

- JMP instruction causes an unconditional transfer of control.
  - → JMP destination
- Destination is usually a label in the same segment as the JMP itself.
- To get around the range of restriction of a conditional jump, JMP can be used.

```
→ START:
→     MOV AX, 0       ; Initialize AX to 0
→     MOV BX, 0       ; Initialize BX to 0
→     MOV CX, 1       ; Initialize CX to 1
```

```
→ LOOP_START:
→     ADD AX, 1       ; Increment AX by 1
→     ADD BX, AX      ; Add the value of AX to BX
→     JMP LOOP_START
                 ; Jump back to LOOP_START
```

The JMP Instruction

- JMP can be used to implement branches and loops.
- However, it is difficult to code an algorithm with JMP without guidelines.
- The high-level languages (conditional IF-ELSE and While loops) can also be simulated in assembly language.

JMP v/s JNZ

Instruction	Type	Condition Checked	Action Taken
JMP	Unconditional Jump	None	Always jumps to the specified label.
JNZ	Conditional Jump	ZF = 0	Jumps only if the Zero Flag is not set.

Write the Complete Assembly Language Code that Prints A Counter 1 to 5, Using Loop.

References

- Chapter 6 of the Text-Book
- Flow Control Instructions – Provided Material – 1
- Online Website Research