



Department of Computer Science and Engineering

Course Code: CSE341	Credits: 1.5
Course Name: Microprocessors	Semester: Fall 21

Lab 04

Flow control instructions and Branching Structures

I. Topic Overview:

For assembly language-programs to carry out useful tasks, there must be a way to make decisions. In this lab, students will familiarize themselves with how decisions can be made with the jump instruction. The jump instructions transfer control to another part of the program. This transfer can be unconditional or can depend on a particular combination of status flag settings. After introducing the jump instructions, we'll use them to implement high-level language decision structures.

II. Lesson Fit:

There is prerequisite to this lab. Students must have a basic idea on the following concepts:

- Flag register
- Some basic operations such as MOV, ADD, SUB, MUL and DIV
- Basic I/O operations
- Character encoding using ASCII

III. Learning Outcome:

After this lecture, the students will be able to:

- Control flow of the program
- Write conditional statements in assembly

IV. Anticipated Challenges and Possible Solutions

- a. Students may find it difficult to visualize how the program control flow changes when using jump

Solutions:

- i. Step by step simulation
- b. Directly coding in assembly may come off as challenging.

Solutions:

- i. Writing the pseudocode first then then converting it to assembly may help

V. Acceptance and Evaluation

Students will show their progress as they complete each problem. They will be marked according to their class performance. There may be students who might not be able to finish all the tasks, they will submit them later and give a viva to get their performance mark. A deduction of 30% marks is applicable for late submission. The marks distribution is as follows:

Code: 50%

Viva: 50%

VI. Activity Detail

- a. **Hour: 1**

Discussion: Jump instruction

Conditional Jumps:

```
if (x>5) {  
    // code  
}
```

In the line if (x>5), a comparison is done between the content of x and 5. The decision whether to execute the enclosed code or not depends on the result of the comparison. In assembly the comparison is done by a piece of code called CMP. The syntax of CMP is CMP destination, source. The comparison is done by destination - source. The result is not saved anywhere but affects the flags. The result of subtraction can be 0, AX > BX or AX < BX. Below is a piece of code.

```
MOV AX, first_number

MOV BX, second_number

CMP AX, BX
```

The comparison is done in the third line. Now there could be 5 possibilities $AX == BX$, $AX > BX$, $AX < BX$, $AX \geq BX$, $AX \leq BX$. We will take the decision based on one of these options. This decision is performed by "jump" instruction denote as "j".

Condition	Jump Instruction	Explanation
$AX == BX$	JE	jump if destination and source are equal
$AX > BX$	JG	jump if destination $>$ source
$AX < BX$	JL	jump if destination $<$ source
$AX \geq BX$	JGE	jump if destination \geq source
$AX \leq BX$	JLE	jump if destination \leq source

But jump where???

Jump in that line which we want to execute when one of the above conditions is satisfied. How will we get the line number? We do not have to worry about the line numbers because we will name the line(s) ourselves.

<pre>int x = 5; if (x > 4) { //code }</pre>	<pre>mov ax, 5 cmp ax, 4 JG My_Line My_Line: ;code</pre>
-----------------------------------------------------	-----------------------------------------------------------

Note the declaration of the line name ends with a colon (:).

Please go through page number 96 of the book to know about more conditional jumps.

Unconditional jump

For skipping a portion of code, we use/ need unconditional jump. There is no comparison, it is just a jump from one line to the one we want. The instruction is called JMP and its syntax is `JMP destination line_name`.

```
mov ax,5
```

```
mov bx,8
```

```

jmp my_line
mov ah,4
mov dl,6

my_line
mov dl,7

```

In the above code, when the 3rd line is executing the program jumps to the line named my_line without executing the codes in between, This is how codes are skipped.

Branching Structures

In high-level languages, branching structures enable a program to take different paths, depending on conditions. In this section, we'll look at three structures.

IF-THEN :

The IF-THEN structure may be expressed in pseudo code as follows:

```

If CONDITION is TRUE
Then
    excute true branch STATEMENTS
End-If

```

The condition is an expression that is true or false. If it is true, the true-branch statements are executed. If it is false, nothing is done, and the program goes on to whatever follows.

Example: Replace the number in AX by its absolute value

Solution: A pseudocode algorithm is:

```

IF AX < 0
THEN replace AX by -AX
END_IF

```

It can be coded as follows:

```

; if AX < 0
    CMP AX, 0    ; AX < 0 ?
    JNL END_IF  ; IF No, then exit
; then
    NEG AX       ; IF yes, then change sign
END_IF:

```

The condition $AX < 0$ is expressed by `CMP AX,0`. If AX is not less than 0, there is nothing to do, so we use a `JNL` (jump if not less) to jump around the `NEG AX`. If condition $AX < 0$ is true, the program goes on to execute `NEG AX`.

Problem Task: Task 01 (Page 8)

b. **Hour: 2**

Discussion: Branching Structures (Cont.)

IF-THEN-ELSE:

```

If CONDITION is TRUE
Then
    execute true branch STATEMENTS
Else
    execute false branch STATEMENTS

```

The `jmp` instruction comes in use when if - else condition is employed. Let us see the use of conditional and unconditional jumps together in a program where we will find the greater of 2 numbers.

JAVA

```

System.out.println("enter the first number");
int x = k.nextInt();

System.out.println("enter the second number");
int y = k.nextInt();

if (x>y){

    System.out.println(x+" is greater");

}

else{

    System.out.println(y+" is greater");
}

```

ASSEMBLY:

```
data segment
; add your data here!
pkey db "press any key...$"
a db "Enter first number$"
b db "Enter second
number$" c db " is
larger$"
ends
stack segment
dw 128 dup(0)
ends
code
segment
start:
; set segment registers:
mov ax, data
mov ds,
ax mov
es, ax
; add your code here
lea dx, a ;Print
a mov ah,9
int 21h

mov
ah,1
int
21h
mov bl,al ;move input to bl

lea dx, b ;Print
b mov ah,9
int 21h

mov
ah,1
int
21h

mov dl,bl
mov ah,2
int 21h
```

```
lea dx,
c mov
ah,9 int
21h jmp

line1:
mov dl,cl
mov ah,2
int 21h

lea dx, c
mov ah,9
int 21h

e:
lea dx, pkey
mov ah, 9
int 21h ;output string
ds:dx

;wait for any key....
mov ah, 1
int 21h

mov ax, 4c00h ;exit
operating
system.
int 21h

ends
end start ; set entry point
stop the assembler.
```

Discussion on CASE

A CASE is a multiway branch structure that tests a register, variable, or expression for particular values or a range of values. The general form is as follows:

CASE expression

Values _ 1: statements_1

Values _ 2: statements_2

..

..

Values_n: statements_n

END_CASE

In this structure, expression is tested; if its value is a member of the set values_1, then statements_1 are executed. We assume that sets values_1, ... , values_n are disjoint

c. Hour: 3

Discussion:

Check progress while the students carry on with the rest of the tasks.

Problem Task: Task 10-14 (Page 9)

VII. Home Tasks: All the unfinished lab tasks.

