**Computer Organization and Assembly Language**

|  |  |
| --- | --- |
| **Lab 04** | |
| **Topic** | * JMP instructions * Conditional/Unconditional JMP instructions |

PART 1 (A)

**JUMP INSTRUCTIONS**

**Two main types of jump instructions**

1. Unconditional jump:

To start executing instructions unconditionally

jmp skip ; unconditional JMP instruction

MOV AX, 2;

ADD AX, 1;

skip: ; skip is a label

mov ax,7

mov ax,0x4c00

int 21h

VAR1: DB 5

VAR2: DW 77

1. Conditional jump:

To start executing instructions based on some condition.

Some conditional jumps are as follows:

|  |  |
| --- | --- |
| JC | Jump if carry flag set |
| JNC | Jump if not carry |
| JZ | Jump if zero flag set |
| JE | Jump if zero flag set |
| JNZ | Jump if zero flag not set |
| JNE | Jump if zero flag not set |
| JS | Jump if sign flag is set |
| JNS | Jump if sign flag not set |
| JP | Jump if parity flag is set |
| JNP | Jump if not parity |
| JO | Jump if overflow |
| JNO | Jump if not overflow |

1. Conditional jumps after *signed operand comparison*

|  |  |
| --- | --- |
| JG | Jump if greater |
| JNG | Jump if not greater |
| JGE | Jump if greater or equal |
| JNGE | Jump if not greater or equal |
| JL | Jump if less |
| JNL | jump if not less |
| JLE | Jump if less or equal |
| JNLE | jump if not less or equal |

1. Conditional jumps after *unsigned operand comparison*

|  |  |
| --- | --- |
| JA | Jump if above |
| JNA | Jump if not above |
| JAE | Jump if above or equal |
| JNAE | Jump if not above or equal |
| JB | Jump if below |
| JNB | Jump if not below |
| JBE | Jump if below or equal |
| JNBE | jump if not below or equal |

* **Compare Instruction**

cmp operand1,operand2

It subtracts operand2 from operand1 and updates the flags only ***without updating the value of the operands.***

**Example: Conditional jmp instruction Example**

In this example code compares two numbers and stores the greater number in result variable.

[org 0x100]

mov al,[num1]

mov bl,[num2]

cmp al,bl

JB myless ; if first operand is less than second operand

;**(JA AND JB IS USED FOR UNSIGNED NUMBERS)**

;if jump is not taken then it means al is greater

;so the code continues to execute in sequence

mov [result],al ; Means AL is greater

jmp exit ;unconditional jump to skip other instructions

myless:

mov [result],bl ; Means BL is greater

exit:

mov ax,0x4c00

int 21h

num1: db 0x20

num2: db 0x30

result: db 0

**Example: Signed number comparison**

In this example it compares two numbers and stores **1** in ax if *al* is greater than *bl* else **0**.

[org 0x100]

MOV AL, 5

MOV BL, 0xFF

CMP AL, BL

JG L1 ; signed statement; jump if greater …

MOV AX, 0

JMP Exit

L1:

MOV AX, 1

Exit:

mov ax,0x4c00

int 21h

**Example: Unsigned number comparison**

[org 0x100]

MOV AL, 5

MOV BL, 0xFF

CMP AL, BL

JA L1 ; unsigned statement; jump if above …

MOV AX, 0

JMP Exit

L1:

MOV AX, 1

Exit:

mov ax,0x4c00

int 21h

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | |  |  | **Considering 8-bits** |
| **Unsigned** | | **5** | | **<** | **255(0xFF)** |
| **Signed** | | **5** | | **>** | **-1(0xFF)** |

**Note:**

PART 1 (B)

* ArrayName: db 1,2,3,4,5,6,7,8,9,10; *This will reserve 10 byte sized values in consecutive memory locations.*
* ArrayName: dw 1,2,3,4,5,6,7,8,9,10; *This will reserve 10 word sized values in consecutive memory locations.*
* var: times 10 db 7; Initializing array var with total 10 elements with value 7*.*

**Sample Code 1**: (Declaration and Access from arrays….)

[org 0x100]

MOV AL, [ARRAY1]; THIS WILL LOAD FIRST ELEMENT FROM ARRAY1

MOV BX, [ARRAY2]; THIS WILL LOAD FIRST ELEMENT FROM ARRAY2

mov ax,0x4c00

int 21h

ARRAY1 DB 1,2,3,4,5;

ARRAY2 DW 0xA, 0xB, 0xC, 0xD, 0xE;

**Sample Code 2**: (Access data from an array….)

**TO ACCESS NEXT ELEMENTS WITHIN AN ARRAY ADD OFFSETS TO ARRAY NAME DEPENDING UPON ARRAY SIZE TYPE.**

[org 0x100]

MOV AL, [ARRAY1+1]; THIS WILL LOAD SECOND ELEMENT FROM ARRAY1

MOV BX, [ARRAY2+2]; THIS WILL LOAD SECOND ELEMENT FROM ARRAY2

mov ax,0x4c00

int 21h

ARRAY1 DB 1,2,3,4,5;

ARRAY2 DW 0XA, 0XB, 0XC, 0XD, 0XE;

**Sample Code 3: Access data from an array (Indexed register Indirect mode)**

[org 0x100]

mov SI, VEC1 ;Loads the effective address of array VEC1 in SI

mov DI, VEC2 ;Loads the effective address of array VEC2 in DI

L1:

MOV AL, [SI]

MOV [DI], AL

INC SI; ;adds ‘1’ to the destination operand.

INC DI;

DEC byte [COUNT]; Subtracts ‘1’ from the destination operand.

JNZ L1; Repeats executing instructions from label L1 until last arithmetic operation

;produces 0.

mov ax,0x4c00

int 21h

VEC1: DB 1, 2, 5, 6,8

VEC2: DB 0,0,0,0,0

COUNT: DB 5

**Use of Loop Instruction:**

It performs the following tasks ***in sequence*** automatically.

* Decrement CX by 1 i.e., CX=CX-1
* Check if CX≠0, Repeat executing instructions from specified label

For example, if you use instruction **loop L1**, it would do following operations.

* DEC CX;
* CMP CX, 0; *It sets the Zero FLAG if both values are same.*
* JNZ L1; *This jump is taken if the last arithmetic operation did not produce a zero.*

Dec cx

Cmp cx,0 Loop L1

JNZ L1

**Sample Code 4: (Same example as above using Loop instruction)**

[org 0x100]

mov SI, VEC1 ;Loads the effective address of array VEC1 in SI

mov DI, VEC2 ;Loads the effective address of array VEC2 in DI

MOV CX, 5

L1:

MOV AL, [SI]

MOV [DI], AL

INC SI; ;adds ‘1’ to the destination operand.

INC DI;

Loop l1

mov ax,0x4c00

int 21h

VEC1: DB 1, 2, 5, 6,8

VEC2: DB 0,0,0,0,0