CSE 331/EEE 332 (Microprocessor Interfacing & Embedded System Lab)

Lab 03 : Conditional jumps/Unconditional jumps;
Procedures;
Instructions: MUL, DIV, CMP, SUB, AND, JZ, JMP
Lab Officer : Azmary Zannat Aurin

Topics to be covered in class today:

- Conditional Jumps/Unconditional Jumps
- Procedures
- Instructions: CMP, AND, SUB, JZ, JMP

Instruction	Operands	Description

MUL	REG, REG
	REG, memory
		Multiplication.
		8 bit multiplication
		
		If we multiply two 8 bit unsigned positive numbers, we will get an unsigned 16 bit result. For this operation, we have to put one operand in accumulator register. The output of the multiplication will be stored in ax.
		
		16 bit multiplication
		
		If we multiply two 16 bit unsigned positive numbers, we will get an unsigned 32 bit result. For this operation, we have to put one operand in accumulator register. The 32 bit result becomes available in the dx register and ax register. The lower 16 bit will be stored in ax register and the higher 16 bit will be stored in dx register.
		
		Example:
		
		MOV AL, 5
		
		MOV DL, 6
		
		MUL DL

<table>
<tr>
<td>DIV</td>
<td>REG, REG
REG, memory</td>
<td>Division.
8 bit division

If we divide a 16 bit unsigned positive number by an 8 bit unsigned positive number, the quotient of the division will be stored in al register and the remainder will be stored in ah register.
AL = Quotient
AH = Remainder
16 bit division

If we divide a 32 bit unsigned positive number by a 16 bit unsigned positive number, the quotient of the division will be stored in ax register and the remainder will be stored in dx register.
AX = Quotient
DX = Remainder
Example:

8 bit division

MOV AX, 01234H

MOV BL, 03FH
DIVBL;AX/BX

16 bit division

MOV DX, 01234H

MOV AX, 05678H
MOV BX, 09000H
DIV BX ; DX:AX / BX</td>
</tr>
<tr>
<td>CMP</td>
<td>REG, memory
memory, REG
REG, REG
memory, immediate
REG, immediate</td>
<td>Compare.

Algorithm:

operand1 - operand2
Result is not stored anywhere, flags are set (OF, SF,

ZF, AF, PF, CF) according to result.
Example:

MOV AL, 5

MOV BL, 5
CMP AL, BL ; AL = 5, ZF = 1 (so equal!)</td>
</tr>
</table>

SUB	REG, memory
	memory, REG REG, REG
	memory, immediate
	REG, immediate
		Subtract.
		
		Algorithm:
		
		operand1 = operand1 - operand2
		Example:
		
		MOV AL, 5
		
		SUB AL, 1 ;AL=4

AND	REG, memory memory, REG REG, REG memory, immediate REG, immediate	Logical AND between all bits of two operands. Result is stored in operand1. These rules apply: 1AND1=1 1AND0=0 0AND1=0 0AND0=0 Example: MOV AL, 'a' ; AL = 01100001b AND AL, 11011111b ; AL = 01000001b ('A')

Jump Instruction

Jump Instructions are used for changing the flow of execution of instructions in the processor. If we want jump to any instruction in between the code, then this can be achieved by these instructions. There are two types of Jump instructions:

Unconditional Jump Instructions
Conditional Jump Instructions

Instruction	Operands	Description

JZ	Label	Short Jump if Zero (equal). Set by CMP, SUB, ADD, TEST, AND, OR, XOR instructions.
		Algorithm:
		
		if ZF = 1 then jump
		
		Example:
		
		.MODEL SMALL
		
		.STACK 100H
		.DATA
		
		.CODE
		
		MAIN PROC
		
		MOV AL, 5
		
		CMP AL, 5
		JZ label1
		MOV DL, 1
		JMP exit
		label1:
		
		MOV DL, 0
		exit:
		
		ENDP MAIN
		END MAIN

JMP	Label	Unconditional Jump. Transfers control to another part of the program. 4-byte address may be entered in this form: 1234h:5678h, first value is a segment second value is an offset.
		
		Algorithm:
		always jump
		Example:
		.MODEL SMALL
		.STACK 100H
		.DATA
		.CODE
		MAIN PROC
		MOV AL, 5
		JMP exit ; jump over 2 lines!
		MOV AL, 0
		exit:
		ENDP MAIN
		END MAIN

PUSH		Store 16 bit data into two locations of SSM (stack) pointed by SS:SP The data source may be: • 16 bit register (except IP, CS) • Two consecutive memory locations ; assume ax = 4567H PUSH AX PUSH DS PUSH WORD PTR DS:[BX]
POP		Retrieve 16 bit from two locations of stack pointed by SS:SP The data destination may be: • 16 bit register • Two consecutive memory locations POP AX POP DS POP WORD PTR DS:[BX]

Difference between CMP and SUB

CMP: Comparison of two numbers, is carried out in the form of a subtraction to determine which of the operands has a greater value. After a CMP instruction, PSW or flag resister get updated. For example, if the operands have equal values, then ZF will be set to 1.

The CMP instruction **does not modify** the **destination field**

SUB: SUB instruction subtracts the source value from the destination. The logic of the SUB instruction is:

destination = destination - source

The SUB instruction **modifies** the **destination field**

Labels

- Labels mark places in a program which other instructions and directives reference
- Labels in the code segment always end with a colon
- Labels in the data segment never end with a colon
- Labels can be from 1 to 31 characters long and may consist of letters, digits, and the special characters ? . @ _ $ %
- If a period is used, it must be the first character
- Labels must not begin with a digit
- The assembler is case insensitive

Legal and Illegal Labels

- Examples of legal names
    - COUNTER1
    - @character
    - SUM_OF_DIGITS
    - $1000 o DONE?
    - .TEST
- Examples of illegal names
    - TWO WORDS contains a blank
    - 2abc begins with a digit
    - A45.28 . not first character
    - YOU&ME contains an illegal character

Example :

```
Start: mov mov jmp mov
                    ax,@data
```

```
    ds, ax
Exit
cx, 10
```

Procedures

Procedure is a part of code that can be called from your program in order to make some specific task. Procedures make program more structural and easier to understand. Generally procedure returns to the same point from where it was called.

The syntax for procedure declaration:

```
name PROC

        ; code

        RET

name ENDP
```

name - is the procedure name, the same name should be in the top and the bottom, this is used to check correct closing of procedures.

Probably, you already know that RET instruction is used to return to operating system. The same instruction is used to return from procedure (actually operating system sees your program as a special procedure).

PROC and ENDP are compiler directives, so they are not assembled into any real machine code.

Compiler just remembers the address of procedure.

CALL instruction is used to call a procedure.

Example:

```
.MODEL SMALL

.STACK 100H

.DATA

.CODE

M2 PROC

        MULBL;AX=AL*BL.

        RET
```

```
M2 ENDP
MAIN PROC
        MOV    AL, 1
        MOV    BL, 2
        CALL  m2 ; 1*2 = 2
        CALL  m2 ; 2*2 = 4
        CALL  m2 ; 4*2 = 8
        CALL  m2 ; 8*2 = 16
ENDP MAIN
END MAIN
```

To work with parameters like other languages you can use PUSH and POP instructions.

Example:

```
.MODEL SMALL
.STACK 100H
.DATA
.CODE
ADD_TWO PROC
        POP AX
        POP DX
        POP CX
        PUSH AX
        ADD DX, CX
        RET
ENDP ADD_TWO
```

```
MAIN PROC
        PUSH 2
        PUSH 3
        CALL ADD_TWO
ENDP MAIN
END MAIN
```

Task 1

Write a program that will count the number of characters in a string.

Task 2

Write a program that will concatenate (join) two strings. Make sure the input strings are not destroyed and the final answer must be inside a third array. Input from user not required. Create two strings in your program.

Example:

String 1: “Hello World, “

String 2: “this is Assembly Language Programming”