Name: Sakiba Sabrina
ID: 1631165042

Assignment

Lecture 2 (1)

1. Steps required to run an assembly program:

- Write the necessary assembly source code
- Save the assembly source code.
- Compile / Assemble source code to create machine code.
- Emulate/Run the machine code.

2. Microcontrollers vs Microprocessors

- A microprocessor is a CPU on a single chip
- If a microprocessor, it's associated support circuitry, memory, and peripheral I/O components are implemented on a single chip, it is a microcontroller.

3. Features of 8086

• 8086 is a 16bit processor. It's ALU, internal registers work with 16bit binary word

• 8086 has a 16bit data bus. It can read or write data to a memory/port either 16 bits or 8bits at a time.

• 8086 has a 20bit address bus which means, it can address up to $2^{20} = 1MB$ memory location.

4. Registers - Register - Resistor

• Both ALU & FPU have a very small amount of super-fast, private memory placed right next to them for their exclusive use. These are called registers.

• Both ALU & FPU store intermediate and final results from their calculations in these registers.

• Processed data goes back to the data cache and then to the main memory from these registers.

5. General Purpose Registers (GPR)

The 8086 CPU has 8 general-purpose registers; each register has it's own name:

• AX - The Accumulator register (divided into AH / AL)

• BX - The Base Address register (divided into BH / BL)

• CX - The Count Register (divided into CH/CL)

- DX - The Data register (Divided int - DH/DL)
- SI - Source Index register
- DI - Destination Index register.
- BP - Base pointer
- SP - Stack Pointer

6. Segment Registers

CS - points at the segment containing the current program.

DS - generally points at the segment where variables are defined.

ES - extra segment register, it's up to a coder to define it's usage.

SS - points at the segment containing the stack.

7. Special Purpose Registers

• IP - The Instruction Pointer. Points to the next location of instruction in the memory

• Flag Register - Determines the current state of the microprocessor. Modified automatically by the CPU after some mathematical operations, determines certain types of results and determines how to transfer control of a program.

8. First Assembly code -

REG: Any valid register

Memory: Referring to a memory location in RAM

Immediate: Using direct values.

MOV (REG, memory, immediate (REG)

Algorithm: operand1 = operand2

ADD(REG, memory / REG, memory / immediate / REG)

Algorithm: operand_1 = operand_1 + operand_2.

Lecture (2)

Syntax for a variable declaration:

name DB value
name DW value

DB - stands for Define Byte
DW - stands for Define Word

- name - can be any letter or digit combination, though it should start with a letter.

- value - can be any numeric value in any supported numbering system, or "?" symbol for variables that are not

initialized.

Creating Constants:

name EQU <any expression>

for example: k EQU 5

MOV AX, k

Creating Arrays:

Text string is an example of a byte array, each character is presented as an ASCII code value (0-255).

a DB 48h, 65h, 6Ch, 6Ch, 6Fh, 00h

b DB 'Hello', 0

- Can Access the value of any element in array using square brackets, example:

MOV AL, a[3]

• Can also use any of the memory index registers BX, SI, DI, BP. Example:

```
MOV SI, 3
MOV AL, a[SI]
```

• To declare a large array can use DUP operator.

The syntax for DUP:

number DUP (value(s))

number - number of duplicates to make

(any constant value)

value - expression that DUP will duplicate

Example:

```
c DB 5 DUP (9)
```

is an alternative way of declaring:

```
c DB 9, 9, 9, 9, 9
```

d DB 5 DUP (1, 2)

is an alternative way of declaring:

d DB 1, 2, 1, 2, 1, 2, 1, 2, 1, 2

Instructions:

INC (REG MEM)

operand = operand + 1

Example: MOV AL, 4

DEC (REG MEM) INC AL ; AL = 5
RET

operand = operand - 1

Example: MOV AL, 86
DEC AL ; AL = 85
RET

LEA (REG, MEM) REG = address of memory (offset)

Example: MOV BX, 35h
MOV DI, 12h
LEA SI, [BX+DI]

Declaring Array:

Array Name db Size DUP(?)

Value initialize:

```
arr1 db 50 dup(5, 10, 12)
```

Index Values:

```
mov bx, offset arr1
mov [bx], 5 ; inc bx
mov [bx+1], 10
mov [bx+2], 9
```

OFFSET:

"Offset" is an assembler directive in x86 assembly language. It actually means "address" and is a way of handling the overloading of the "mova" instruction.

1. mov si, offset variable
2. mov si, variable

The first line loads SI with the address of variable, the second line loads SI with the value stored at the address of variable.

As a matter of style, when I wrote x86 assembler I would write it this way -

1. mov si, offset variable
2. mov si, [variable]

LEA is an instruction that will load "offset variable" while adjusting the address between 16 and 32 bits as necessary.

"LEA loads the lower 16 bits of the address into the register, and "LEA" loads with the 32-bit register with the address zero extended to 32 bits.

Lecture (3)

In this Assembly Language Programming, A single program is divided into four Segments which are:-

1. Data Segment
2. Code Segment
3. Stack Segment, and
4. Extra Segment

First Line - DATA SEGMENT

DATA SEGMENT is the starting point of the Data segment in a program and DATA is the name given to this segment and SEGMENT is the keyword for defining segments where we can declare our variables.

Next Line - MESSAGE DB "HELLO WORLD!!!"

MESSAGE is the variable name given to a Data Type (Size) that is DB. DB stands for

Define Byte and is of one byte (8 bits).

In Assembly language programs, variables are defined by Data size not its Type. character need One Byte. so to store character or string we need DB only that don't mean DB can't hold number or numerical value. The string is given in double quotes. $ is used as NULL in c programming, so that compiler can understand where to STOP.

Next Line - DATA ENDS

DATA ENDS is the End point of the Data segment in a Program.

Next Line - CODE SEGMENT

CODE & SEGMENT is the starting point of the code segment in a Program and CODE is the name given to this segment and SEGMENT is the keyword for defining

Segments, where we can write the coding of the program.

Next Line - ~~Assume~~ ASSUME DS: DATA CS: CODE

In this Assembly Language Programming, there are Different present for Different Purpose so we have to assume DATA is the name given to ~~DS~~ Data Segment register and CODE is the name given to code segment register.

Next Line - START:

START is the label used to show the starting point of the code which is written in the Code Segment: is used to define a label as in C programming.

Next Line - MOV AX, DATA

MOV DS, AX

After Assuming DATA and CODE segment MOV is a keyword to move the second element into the first element. But we can not move DATA Directly to DS due to MOV command restriction, hence we move DATA to AX and then from AX to DS. AX is the first and most important register in the ALU unit.

Next Line - LEA DX, MESSAGE

MOV AH, 9

INT 21H

The above three-line code is used to print the string inside the MESSAGE variable.

To do input and output we use Interrupts. Standard Input and Standard Output related Interrupts are found in INT 21H which is also called as DOS interrupt.

Next Line - MOV AH, 4CH
INT 21H

The above two-line code is used to exit to dos or exit to operating system. If the value is 4ch, that means Return to Operating System or DOS.

Next Line - CODE ENDS

CODE ENDS is the End point of the CODE Segment in a Program.

Last Line - END START

END START is the end of the label used to show the ending point of the code which is written in the Code Segment.

Execution of program explanation - Hello World

First save the program with HelloWorld.asm filename. No space is allowed in the

name of the Program File and an extension as .asm (dot asm because it's an Assembly language program). The written Program has to be compiled and Run by clicking on the RUN button on the top.