CSE 331L - 1

Introduction to Assembly Language

Introduction

In this session you will be introduced to assembly language programming and to the emu8086 emolator software. emo8086 will be used as both an editor and as an assembler for all given assembly language programming.

Steps required to run an assembly program:

1. Write the necessary assembly source code.
2. Save the assembly source code.
3. Complete / Assemble source code to create machine code.
4. Emulate / Run machine code

Microcontrollers vs. Microprocessors

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

Features of 8086

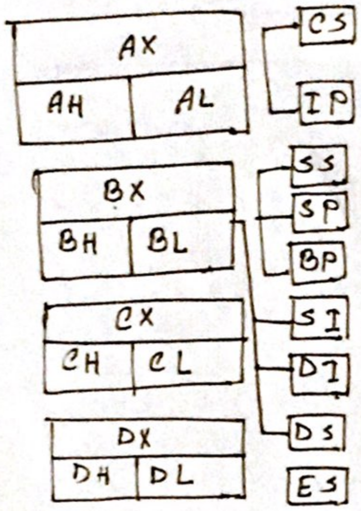
- 8086 is a 16 bit processor. It's ALU, internal registers work with 16-bit. binary word.
- 8086 has a 16 bit data bus. It can read or write data to a memory/ port either 16 bits or 8 bits at a time.
- 8086 has a 20 bit address bus which means, it can address up to 2^{20} = 1MB memory location.

Registrar - 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 registers.
- The 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.

Inside the CPU: Get to know the various registers

Central Processing Unit (or CPU)



General Purpose Registers (GPR)

- AX - The Accumulator register
- BX - The Base Address register
- CX - The count register.
- DX - The Data register
- SI - Source Index register
- DI - Base Index register
- BP - Base Pointer
- SP - Stack Pointer

Despite the name of a register, It's the programmer who determines the usage for each general-purpose register. The main purpose of a register is to keep a number. The size of the above register is 16bits.
4 general-purpose registers are made of two separates 8-bit registers.
Since registers are located inside CPU, they are much faster than memory. Accessing a memory location requires the use of a system bus, so it takes much Longer. Accessing data in a register usually takes no time.

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.
SS- points at the segment containing the stack.

Special Purpose Register

- IP - The Instruction Ponter.
- Flag Register - Determines the current state of the microprocessor.

Writing Your First Assembly Code

The following table shows the instruction name, the syntax of it's use, and it's description.

- REG: Any valid register
- Memory: Referring to a memory location in RAM
- Immediate: Using direct values.

Instruction	Operation	Description
MOV	REG, memory. memory, REG REG REG memory, Immediate REG, immediate	Copy Operand 2 to Operand 1 • Cannot set the value of CS and IP • Copy value of one segment register to another segment register. • copy an immediate value to segment register. Algorithm: operand 1 = operant 2
ADD	REG, memory memory, REG REG REG memory, immediate REG, immediate.	Add two number. Algorithm: operand 1 = operand 1 + operand 2.

CSE 331L - 2

Variables, I/O, Array

Creating Variable

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 letter.
- value: can be any numeric value in any supported numberi system or "?" symbol for variables that are not initialized.

Creating Constants

Constants are just like variables, but they exist only until the program is compiled. To define constants EQU directive is used.

```
name EQU <any expression>
```

For example:

```
K  EQU 5
MOV AX ,K
```

Creating Arrays

Arrays can be seen as chains of variables. A text string is an example of a byte array

Here are some array definition examples:

```
a DB 48h, 65h, 6ch, 6Fh, 00h
b DB 'Hello', 0
```

- You can access the value of any element in array using square brackets.

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

- You can also use any of the memory index registers BX, SI, DI, BP,

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

- If you need to declare a large array you 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.
for example:

```
c DB 5 DUP(9)
```
is an alternating way of declaring
```
c DB 9,9,9,9,9
```

Memory Access

To access memory, we can use these four registers: BX, SI, DI, BP.

[BX + SI]	[SI]	[BX + SI + d8]
[BX + DI]	[DI]	[BX + DI + d8]
[BP + SI]	d16 (variable offset only)	[BP + SI + d8]
[BP + DI]	[BX]	[BP + DI + d8]
[SI + d8]	[BX + SI + d16]	[SI + d16]
[DI + d8]	[BX + DI + d16]	[DI + d16]
[BP + d8]	[BP + SI + d16]	[BP + d16]
[BX + d8]	[BP + DI + d16]	[BX + d16]

- Displacement can be an immediate value or offset of a variable, or even both.
- Displacement can be inside or outside of the [ ] symbols, assembler generates the same machine code for both ways.
- Displacement is a signed value.

Instruction

Instructions	Operands	Description
INC	REG MEM	Increament Algorithm ~~oppe~~rand : operand+1 Example : MOV AL, 4 INC AL ; AL: 5 RET
DEC	REG MEM	Decreament Algorith operand = operan-1 Example ; MOV AL, 86 DEC AL; AL: 85 RET
LEA	REG, MEM	Load Effective Address Algorithm: REG: address of memory (offset) Example: MOV BX , 35h MOV DI , 12h LEA SI , [BX+DI]

Offset

Offset is an assembler directive in X 86 assembly language. It actually means 'address' and is a way of handling the over loading of 'mov' instructions

1- mov si , offset variables
2. mov , si , variables

$\boxed{\text{CSE 331L - 3}}$

Print & I/O

Print : Hello World in Assembly Language.

```
DATA SEGMENT
   MESSAGE DB "HELLO WORLD!!!$"
ENDS
CODE SEGMENT
    ASSUME  DS: DATA  CS: CODE
 START:
       MOV AX, DATA
       MOV DS, AX
       LEA  DX, MESSAGE
       MOV  AH,9
       INT  21H
       MOV  AH , 4CH
       INT  21H
  ENDS
  END START
```

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 it's Type. Character need One By so to store character or string we need DB only that don't mean DB can't hold number or numerical value.

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.

Next Line - ASSUME DS: DATA CS: CODE

In this Assembly Language Programming, there are Different Registers present for Different Purpose so we have to assume DATA is the name given to Data Segment Register and CODE is the name 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, still it is ~~the~~ compulsory to initialise DATA Segment to DS Register. MOV is a keyword to move the second element into the first element. But we cannot move DATA Directly to DS due to MOV commands restriction, hence we move DATA to AX and from AX to DS. AX is the first and the most important register in ALU.

Next Line- MOV AH, 4CH
INT 21 H

The above two line code is used to exit to dos or exit to operating system. Standard input and Standard output related Interrupt are found in INT 21 H which is also called DOS interrupt. It works with the value of AH register.

Next Line- CODE ENDS

CODE ENDS is the ~~the~~ END points 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 code Segment.