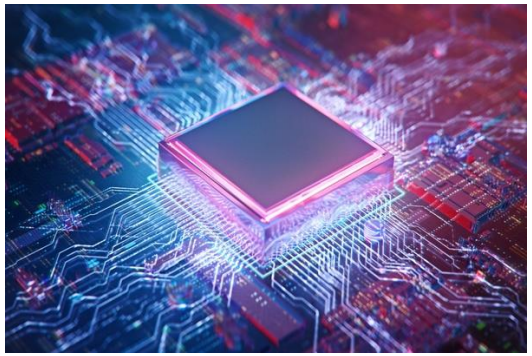

EL2003
Computer Organization and Assembly Language

Dr. Muhammad Usman Abbasi
Assistant Professor | Head of Electrical Engineering Department
National University of Computer and Emerging Sciences - FAST

1. Introduction to Assembly Language

- 1.1. Basic Computer Architecture
- 1.2. Registers
- 1.3. Instruction Groups
- 1.4. Intel iapx88 Architecture
- 1.5. History
- 1.6. Register Architecture
- 1.7. Our First Program
- 1.8. Segmented Memory Model

1.2. Registers

- The basic purpose of a computer is to perform operations.
- Operations need operands.
- Operands are the data on which we want to perform a certain operation.

- Consider the addition operation.
- It involves adding two numbers (8-bit, 16-bit, 32-bit, 64-bit etc.)
- We can have precisely one address on the address bus and consequently precisely one element on the data bus.
- At the very same instant the second operand cannot be brought inside the processor.
- As soon as the second is selected, the first operand is no longer there.
- For this reason there are temporary storage places inside the processor called *registers*.

More on registers…

- Registers are like a scratch pad ram inside the processor and their operation is very much like normal memory cells.
- They have precise locations and remember what is placed inside them.
- Names: r0, r1, r2, … or A, B, C, D, …, etc.
- X stands for an index register.

- Accumulator
- Pointer, Index, or Base Register
- Flags Register or Program Status Word
- Program Counter or Instruction Pointer

1.2. Registers

- Accumulator
  - There is a central register in every processor called the accumulator.
  - Traditionally all mathematical and logical operations are performed on the accumulator.
  - The word size of a processor is defined by the width of its accumulator.
  - A 32-bit processor has an accumulator of 32 bits.

1.2. Registers

- **Pointer, Index, or Base Register**
  - It does not hold data but holds the address of data.
  - Index register is used in such a situation to hold the address of the current array location.
  - Now the value in the index register cannot be treated as data, but it is the address of data.

  - In newer architectures the distinction between accumulator and index registers has become vague.
  - They have general registers which are more versatile and can do both functions.

1.2. Registers

- **Flags Register or Program Status Word**
  - This is a special register in every architecture called the flags register or the program status word.
  - Like the accumulator it is an 8, 16, or 32-bit register but unlike the accumulator, it is meaningless as a unit, rather the individual bits carry different meanings.
  - The bits of the flags register work independently and individually, and combined its value is meaningless.

  - For example: Carry flag
    If a 16bit number is added to a 16-bit accumulator, and the result is 17-bit the 17th bit is placed in the carry bit of the flags register.

1.2. Registers

- **Program Counter or Instruction Pointer**
  - Everything must translate into a binary number for our dumb processor to understand it, be it an operand or an operation itself.
  - Every instruction themselves must be translated into numbers.
    - For example: To add numbers we understand the word “add”. This “add” needs to be translated into a number so that **uP/uC** understands it.
  - All the objects, inheritance and encapsulation constructs in higher-level languages translate down to just a number in assembly language in the end.
  - A program is defined to be “an ordered set of instructions.”

1.2. Registers

- A program is defined to be “an ordered set of instructions.”
- “The program counter holds the address of the next instruction to be executed.”

1.3. Instruction Groups

- Usual opcodes in every processor exist for moving data, arithmetic and logical manipulations etc.
- Some manufacturers name the mnemonics for data movement instructions as “move,” some call it “load” and “store” and still other names are present.

Data Movement Instructions

- These instructions are used to move data from one place to another.
- These places can be registers, memory, or even inside peripheral devices.
- Some examples are:
  - mov ax, bx
  - lad 1234

Arithmetic and Logic Instructions

- Arithmetic instructions like addition, subtraction, multiplication, division and Logical instructions like logical and, logical or, logical xor, or complement are part of this group. Some examples are:
  - and ax, 1234
  - add bx, 0534
  - add bx, [1200]

- The bracketed form is a complex variation meaning to add the data placed at address 1200.

Program Control Instructions

- cmp ax, 0
- jne 1234

Special Instructions

- cli
- sti