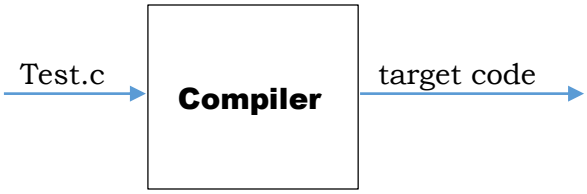
Introduction to compilers

22/07 & 23/07/13

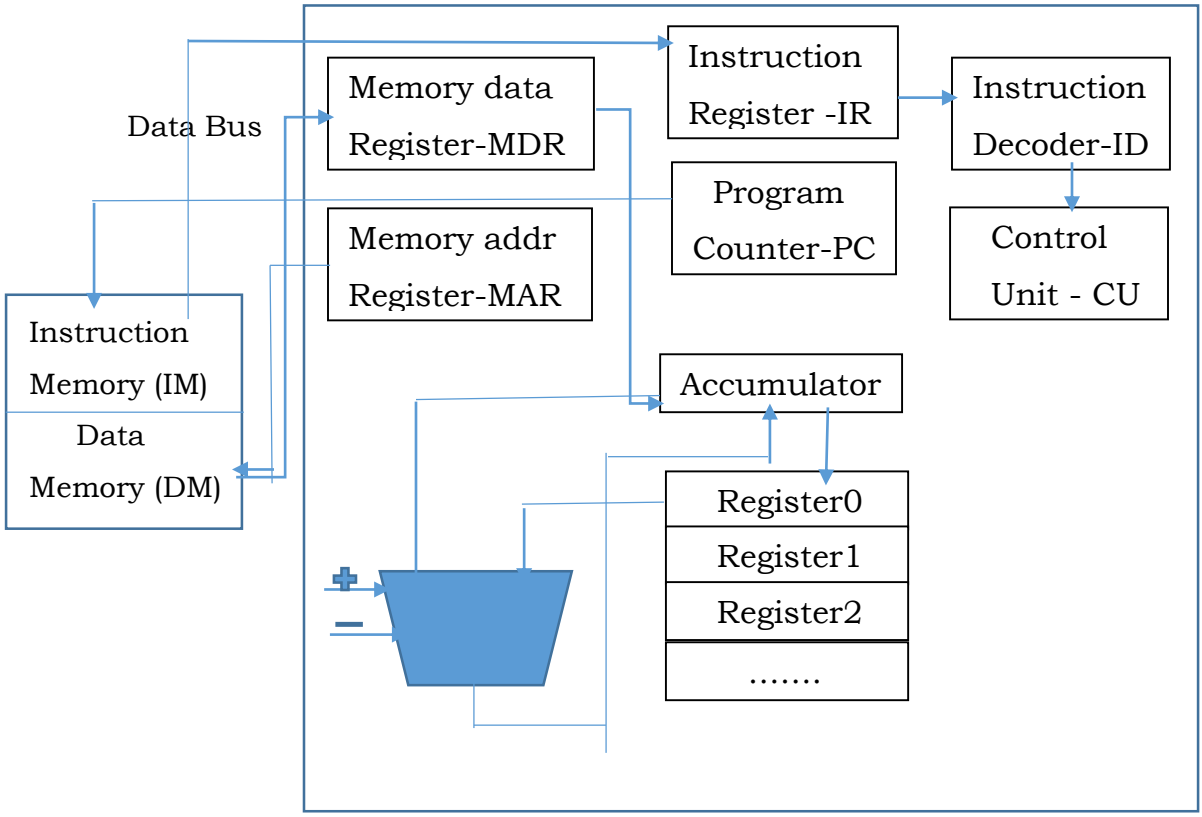
Compiler basically performs two tasks.

- Transforms the high level source code into assembly level target code
- Detects the errors in the source code



We already know that every computer has its own processor. Below is given, the internal architecture of the processor.

Processor



Processor consists of a set of registers. There are 2 types of registers

- General purpose registers
- Special purpose registers

Here are, two of the few special purpose registers. These are used to access the main memory.

- Memory Data register
- Memory address registers

Main memory also two parts. They are

- Instruction Memory - stores the body of the code i.e set of instructions
- Data Memory - stores the variable(s) values/data

Accumulator is also a special purpose register. It is used for temporary storage. Whenever we read something from the memory, that content/value will be stored in the accumulator. Even the result of any operation Ex:- addition, is also stored in the accumulator before writing back into the memory.

The program counter stores the address of the next instruction to be executed. The instruction decoder and Control unit together are used to determine the **control signals** and the next instruction.

First, the instruction at the address provided by the PC will be stored in the IR and then the ID decodes the instruction and the CU generates the control signals ex:- read/write for the data bus. The operand (if any) in the data memory will be read (or) written through the address stored in the address bus provided by the MAR. The data in the MDR will be copied into the accumulator and if another operand comes then the content of the accumulator will be moved into the register 0. The ALU has, accumulator and a register as its two inputs and after performing the arithmetic/logic operation the result will again be stored in the accumulator and then to the MDR and finally back into DM.

Now, consider the operation [X] <- [P] + [Q] where [P] denotes the value at the address P. Here are the sequence of instructions performed by processor upon getting this instruction.

1) Fetch [P]
2) Transfer the content of accumulator to R0
3) Fetch [Q]
4) Perform the add operation
5) Pass the result to the accumulator
6) Pass the write address to MAR and the write data to MDR from accumulator
7) CU performs the write signal

Now, here are the corresponding assembly level instructions for the above steps

LDA 1000 -> fetches the content of memory location 1000 into accumulator
MOV R0 -> moves the content of accumulator into Register R0
LDA 2000 -> fetches the content of memory location 2000 into accumulator
ADD R0 -> Adds the contents of accumulator and R0 and stores in accum
STR 2050 -> Stores the content of accumulator in the memory location 2050

Each of the above assembly level instruction has a corresponding machine level interpretation. We write in assembly level just for readability purposes.
For example:

LDA - 1001001 -> 73
MOV - 1010010-> 82
STA - 0001111 -> 15

Hence the machine level code looks like

73 1000
82
71 2000
.....

The machine level code is machine/processor dependent.
While designing the processor we need to simultaneously define the instructions that are to be supported by the processor. Each processor has its own finite set of Instruction set.

Each compiler has 2 parts

- General purpose
- Machine dependent

Compiler v/s Interpreter:-

The interpreter translates and executes the source code line by line where as the compiler first translates the entire code and then executes it. The compiler works faster than the interpreter since the interpreter translates and executes line by line which takes more time but the interpreter can give better error diagnostics since it works line by line.