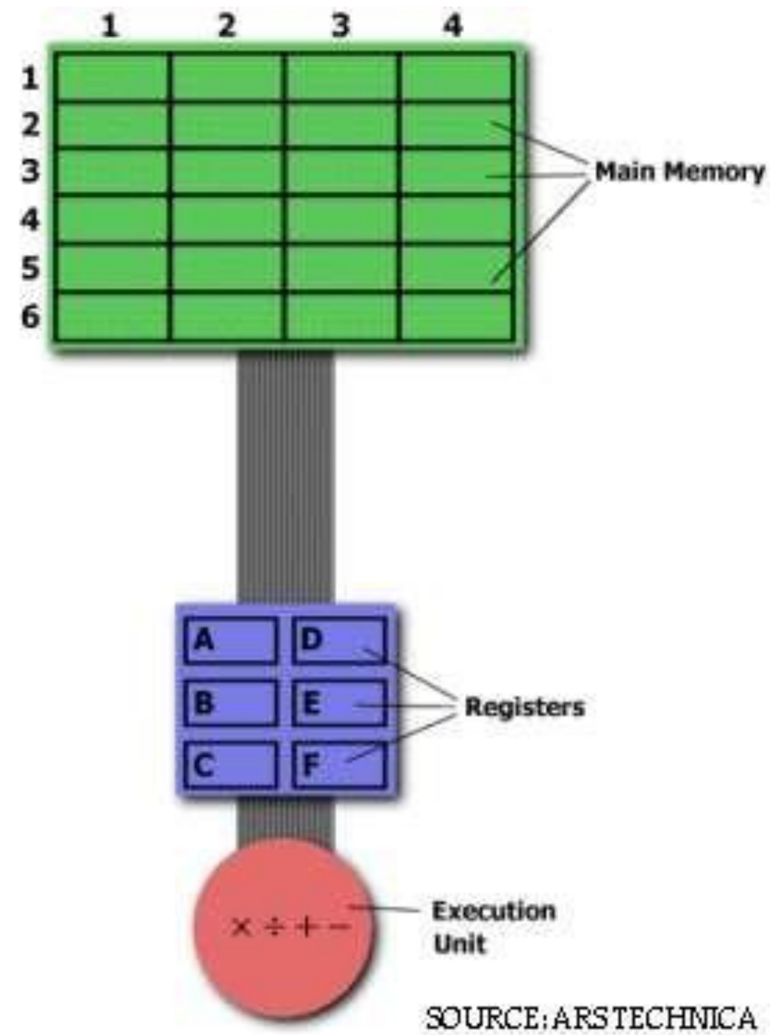
RISC Vs CISC, Harvard v/s Van Neumann

CISC Architecture

- The simplest way to examine the advantages and disadvantages of RISC architecture is by contrasting it with it's predecessor: CISC (Complex Instruction Set Computers) architecture.

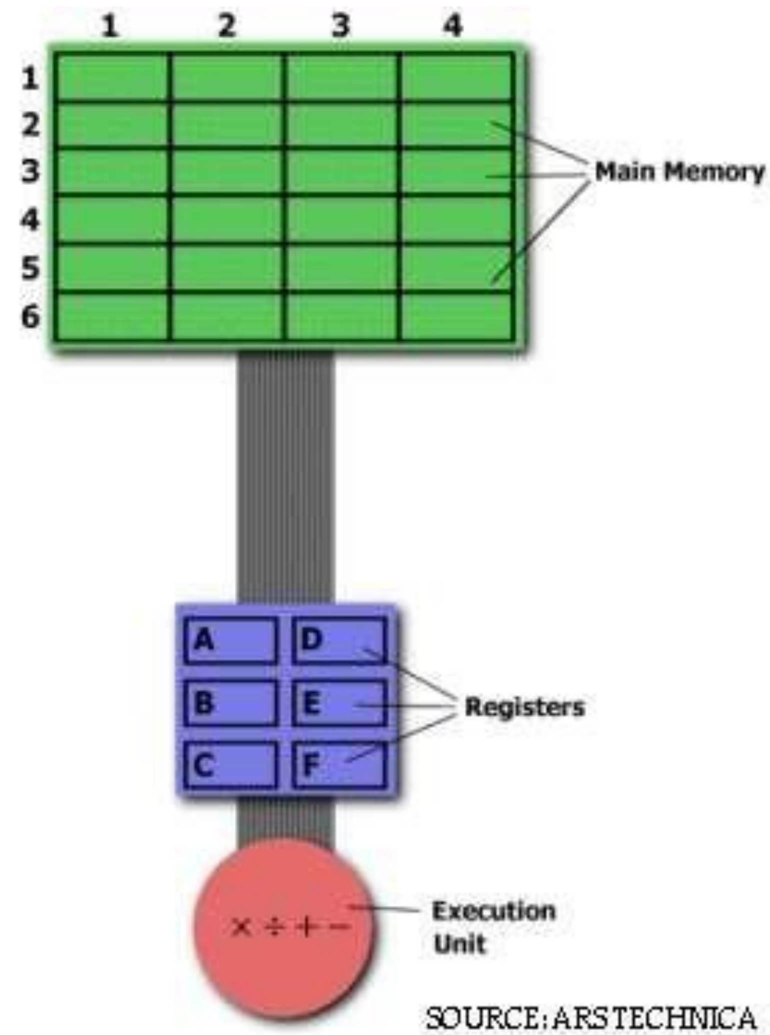
Multiplying Two numbers in Memory

- On the right is a diagram representing the storage scheme for a generic computer. The main memory is divided into locations numbered from (row) 1: (column) 1 to (row) 6: (column) 4.



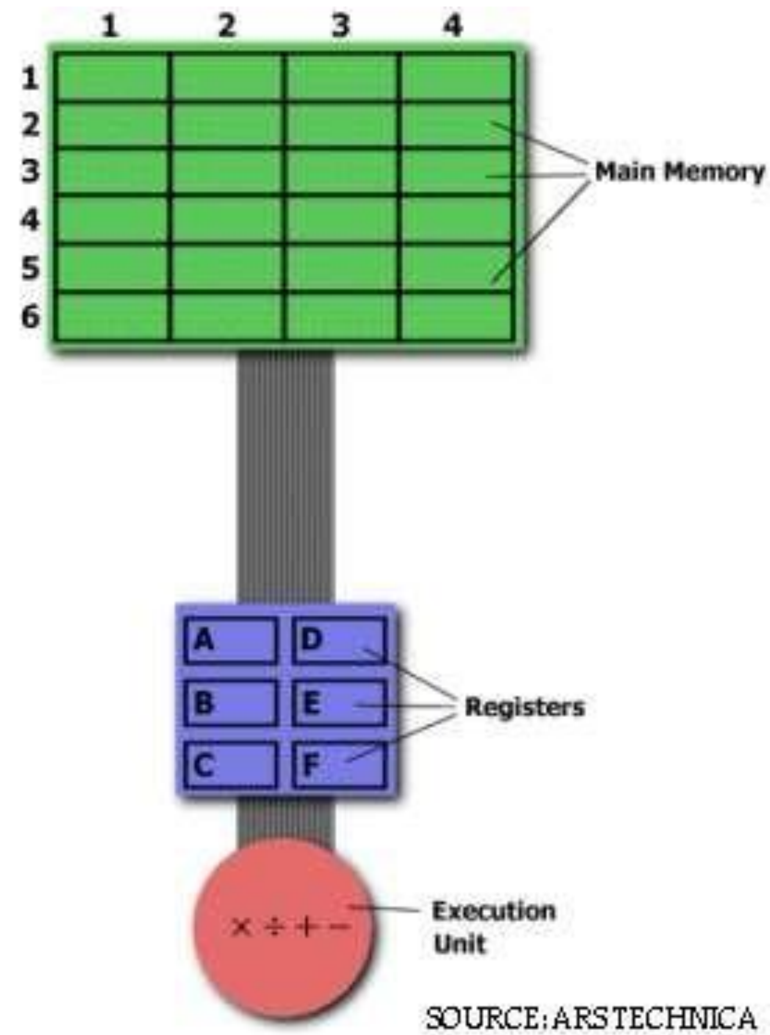
Multiplying Two numbers in Memory

- The execution unit is responsible for carrying out all computations. However, the execution unit can only operate on data that has been loaded into one of the six registers (A, B, C, D, E, or F).



Multiplying Two numbers in Memory

- Let's say we want to find the product of two numbers - one stored in location 2:3 and another stored in location 5:2 - and then store the product back in the location 2:3.



The CISC Approach

- The primary goal of CISC architecture is to complete a task in as few lines of assembly as possible.
- This is achieved by building processor hardware that is capable of understanding and executing a series of operations.

The CISC Approach

- For this particular task, a CISC processor would come prepared with a specific instruction (we'll call it "MULT").
- When executed, this instruction loads the two values into separate registers, multiplies the operands in the execution unit, and then stores the product in the appropriate register.
- Thus, the entire task of multiplying two numbers can be completed with one instruction:

MULT 2:3, 5:2

The CISC Approach

- MULT is what is known as a "complex instruction."
- It operates directly on the computer's memory banks and does not require the programmer to explicitly call any loading or storing functions.
- It closely resembles a command in a higher level language.
- For instance, if we let "a" represent the value of 2:3 and "b" represent the value of 5:2, then this command is identical to the C statement "a = a * b."

The CISC Approach

- One of the primary advantages of this system is that the compiler has to do very little work to translate a high-level language statement into assembly.
- Because the length of the code is relatively short, very little RAM is required to store instructions.
- The emphasis is put on building complex instructions directly into the hardware.

The RISC Approach

- RISC processors only use simple instructions that can be executed within one clock cycle.
- Thus, the "MULT" command described above could be divided into three separate commands: "LOAD," which moves data from the memory bank to a register, "PROD," which finds the product of two operands located within the registers, and "STORE," which moves data from a register to the memory banks.

The RISC Approach

- In order to perform the exact series of steps described in the CISC approach, a programmer would need to code four lines of assembly:

  LOAD A, 2:3
  LOAD B, 5:2
  PROD A, B
  STORE 2:3,
  A

The RISC Approach

- At first, this may seem like a much less efficient way of completing the operation.
- Because there are more lines of code, more RAM is needed to store the assembly level instructions.
- The compiler must also perform more work to convert a high-level language statement into code of this form.

The RISC Approach

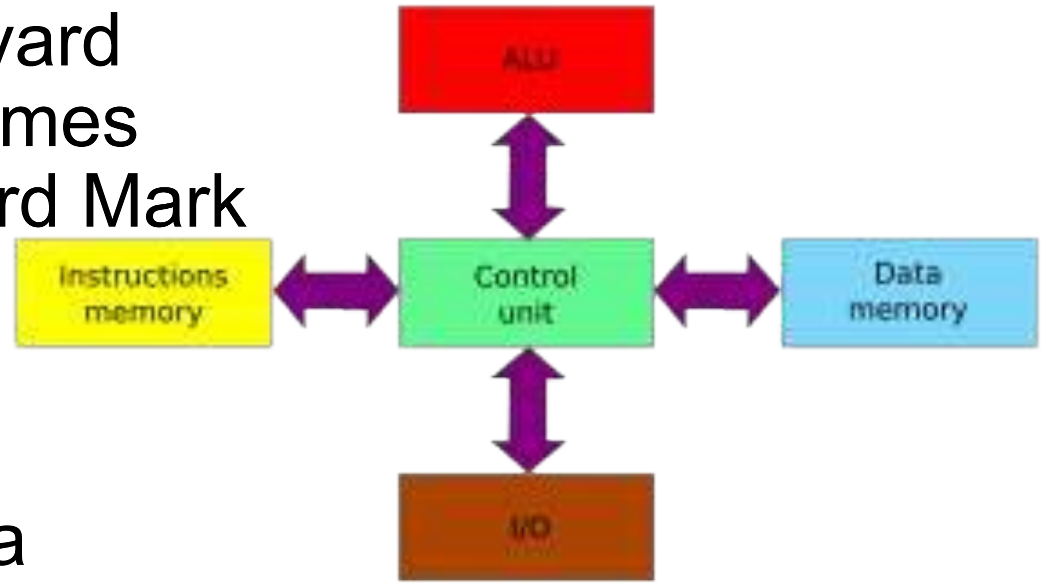
- However, the RISC strategy also brings some very important advantages.
- Because each instruction requires only one clock cycle to execute, the entire program will execute in approximately the same amount of time as the multi-cycle "MULT" command.
- These RISC "reduced instructions" require less transistors of hardware space than the complex instructions, leaving more room for general purpose registers.
- Because all of the instructions execute in a uniform amount of time (i.e. one clock), pipelining is possible.

CISC	RISC
1.Emphasis on hardware	1.Emphasis on software
2.Includes multi-clock complex instructions	2.Single-clock, reduced instruction only
3.Memory-to-memory: "LOAD" and "STORE" incorporated in instructions	3.Register to register: "LOAD" and "STORE" are independent instructions
4.Small code sizes, high cycles per second	4.Low cycles per second, large code sizes
5.Transistors used for storing complex instructions	5.Spends more transistors on memory registers

http://cs.stanford.edu/people/eroberts/courses/soco/projects/risc/developments/index.html

Harvard Architecture

- The name Harvard Architecture comes from the Harvard Mark I relay-based computer.
- The Harvard architecture is a computer architecture with physically separate storage and signal pathways for instructions and data.



Harvard Architecture

- In other words, it physically separate signals and storage for code/program and data memory.
- It is possible to access program memory and data memory simultaneously.
- Typically, code (or program) memory is read-only and data memory is read- write.
- Therefore, it is impossible for program contents to be modified by the program itself.

Von Neumann Architecture

- The von Neumann Architecture is named after the mathematician and early computer scientist John von Neumann.
- Von Neumann machines have shared signals and memory for code and data.
- Thus, the program can be easily modified by itself since it is stored in read-write memory.

VAN-NEUMANN ARCHITECTURE	**HARVARD ARCHITECTURE**
Used in conventional processors found in PCs and Servers, and embedded systems with only control functions.	Used in DSPs and other processors found in latest embedded systems and Mobile communication systems, audio, speech, image processing systems
The data and program are stored memories in the same memory	The data and program are separate
The code is executed serially and takes more clock cycles	The code is executed in parallel
There is no exclusive Multiplier	It has MAC (Multiply Accumulate)
Absence of Barrel Shifter	Barrel Shifter help in shifting and rotating operations of the data
The programs can be optimized in lesser size	The program tend to grow big size

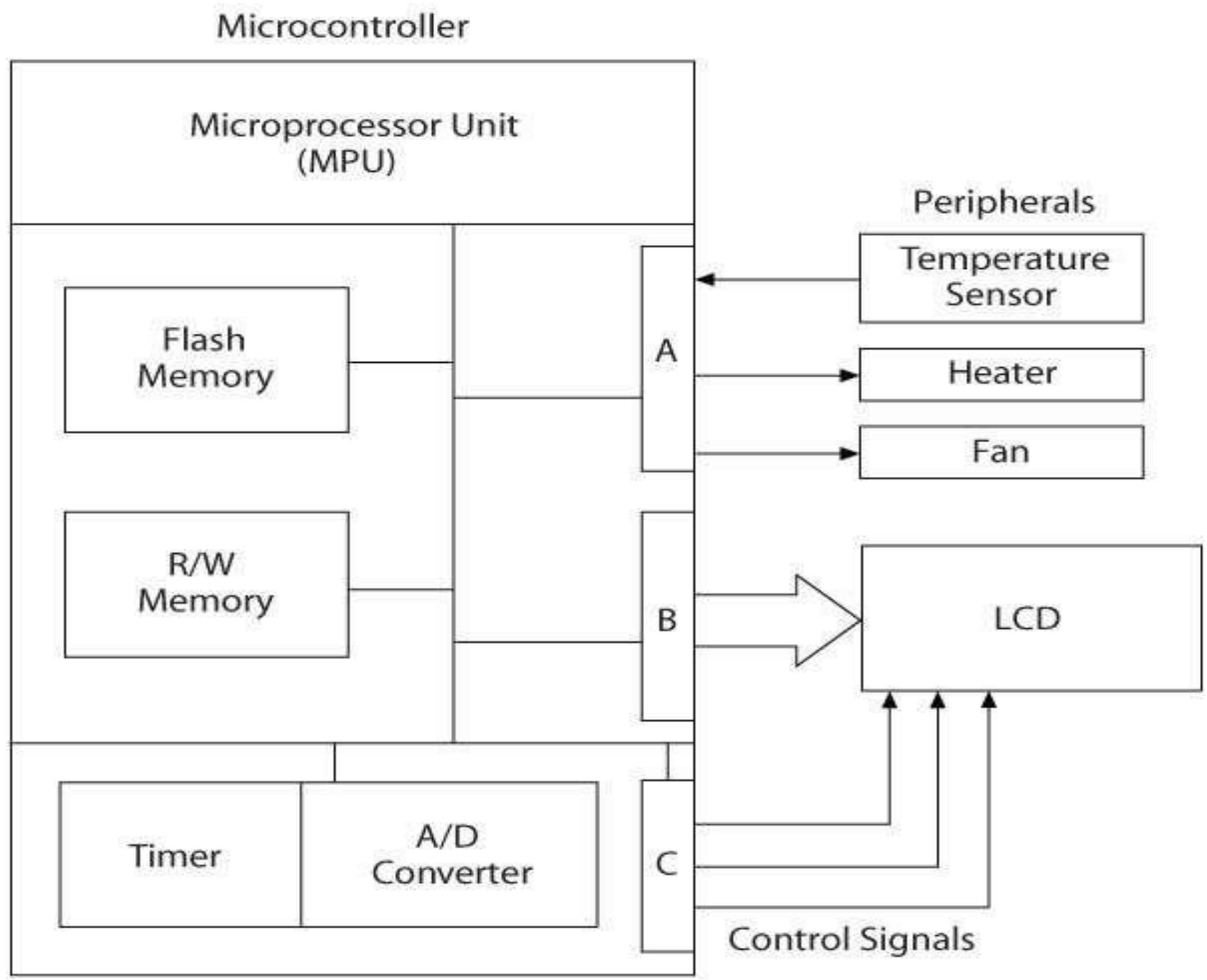
Microcontrollers

- Embedded Systems
  - Operations managed behind the scenes by a microcontroller
- Microcontroller (MCU)
  - Integrated electronic computing device that includes three major components on a single chip
    - Microprocessor (MPU)
    - Memory
    - I/O (Input/Output) ports

Microcontrollers

- Support Devices
  - Timers
  - A/D converter
  - Serial I/O
- Common communication lines
  - System Bus

MCU-Based System



Software

- Machine Language
  - Binary Instructions
  - Difficult to decipher and write
    - Error-prone
  - All programs converted into machine language for execution

Instruction	Hex	Mnemonic	Description	Processor
10000000	80	ADD B	Add reg B to Acc	Intel 8085
00101000	28	ADD A, R0	Add Reg R0 to Acc	Intel 8051
00011011	1B	ABA	Add Acc A and B	Motorola 6811

Software

- Assembly Language
  - Machine instructions represented in mnemonics
  - One-to-one correspondence
  - Efficient execution and use of memory
  - Machine-specific

Evolution of Programming

- Machine Language
  - Binary format

```
11100101100111110001000000010000
11100101100111110000000000001000
11100000100000010101000000000000
11100101100011110101000000001000
```

  - Hexadecimal format

```
E59F1010
E59F0008
E0815000
E58F5008
```

Evolution of Programming

- Assembly Language
  - Mnemonic codes

```
E59F1010  LDR  R1, num1
E59F0008  LDR  R0, num2
E0815000  ADD  R5, R1, R0
E58F5008  STR  R5, sum
```

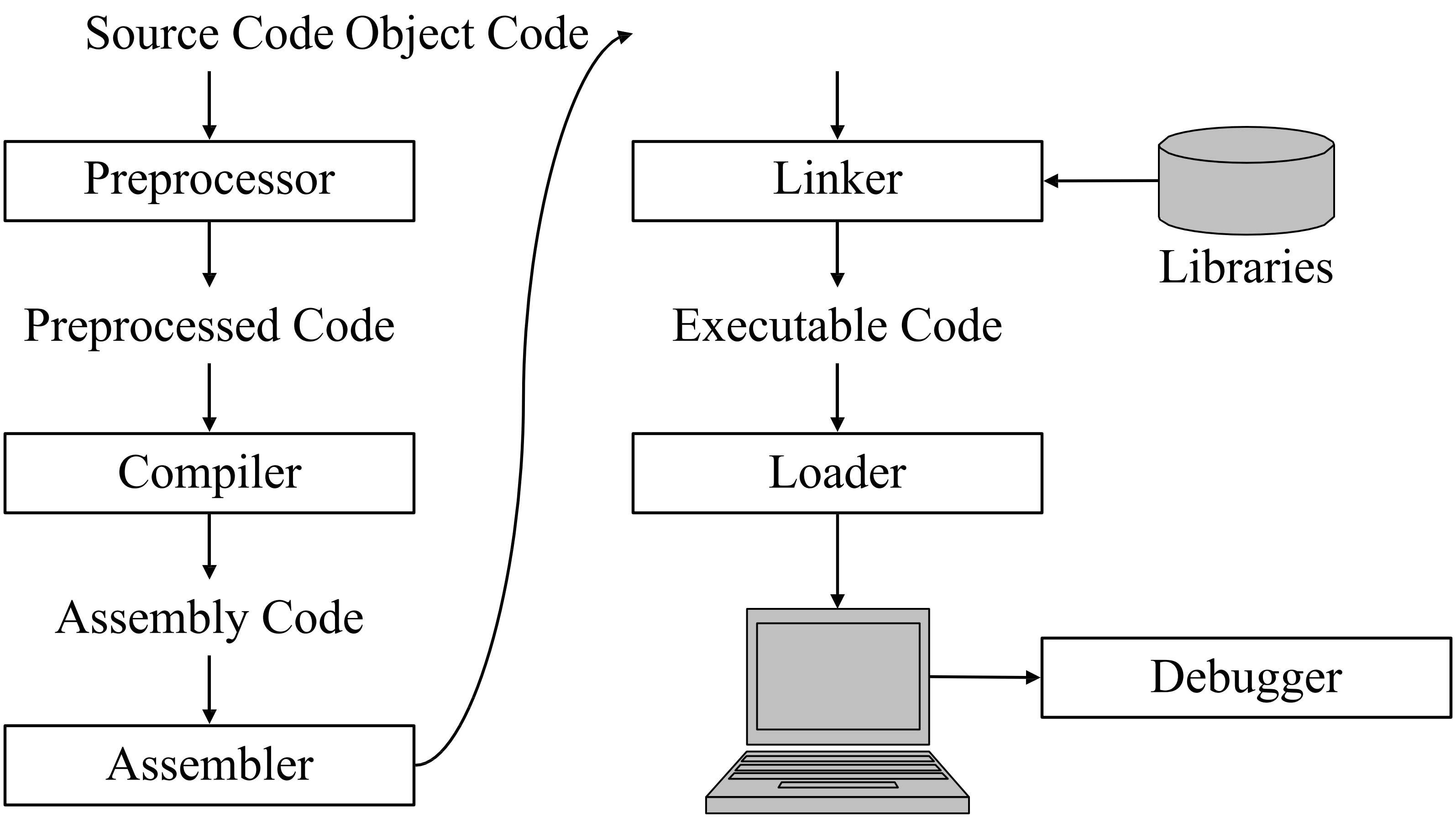
- High-Level Language
  - C language

```
sum = num1 + num2;
```

Approaches

	Machine Independent	Machine Dependent
Human Readable	High-Level Languages (C, C++, Java, Pascal)	Assembly Languages
Human Unreadable	Pseudo Code (Bytecode, P-code)	Machine Languages (x86, MIPS, ARM)

From Source to Executable



Compiler, Assembler, Linker, Cross Compiler...

Compiler

- A software program that converts source code that written in high level programming language into low level language.
- A ***Native-compiler*** runs on a computer platform and produces code for that same computer platform.
- A ***Cross-compiler*** runs on one computer platform and produces code for *another computer platform*

Assembler

- Convert assembly into object file

Linker

- A linker or link editor is a program that takes one or more objects generated by compilers and assembles them into a single executable program or a library that can later be linked to in itself.
- Link the object files and libraries to form an executable

Cross-Compiler

- A cross compiler is a compiler capable of creating executable code for a platform other than the one on which the compiler is running.
- For example, a compiler that runs on a Windows 7 PC but generates code that runs on Android smartphone is a cross compiler.

Debugger

- A **debugger** or **debugging tool** is a computer program that is used to test and debug other programs.
- The code to be examined might alternatively be running on an *instruction set simulator* .
- When the program crashes, the debugger shows the actual position(Segment) in the original code if it is a source-level debugger.
- If it is a low-level debugger or a machine-language debugger it shows that line in the program.

Emulator

- An emulator is a piece of Hardware/Software that enables one computer system to run programs that are written for another computer system.
- For example emulator 8086, 8086 microprocessor programs.
- An emulator is used on the target processor (the processor for which the program is being written).

Emulator Example

Android Virtual Machine(AVM)