

Computer Architecture Concepts for System Programmers

System programming is about developing programs to run directly on top of hardware or developing programs to run on top of operating system by using its system calls.

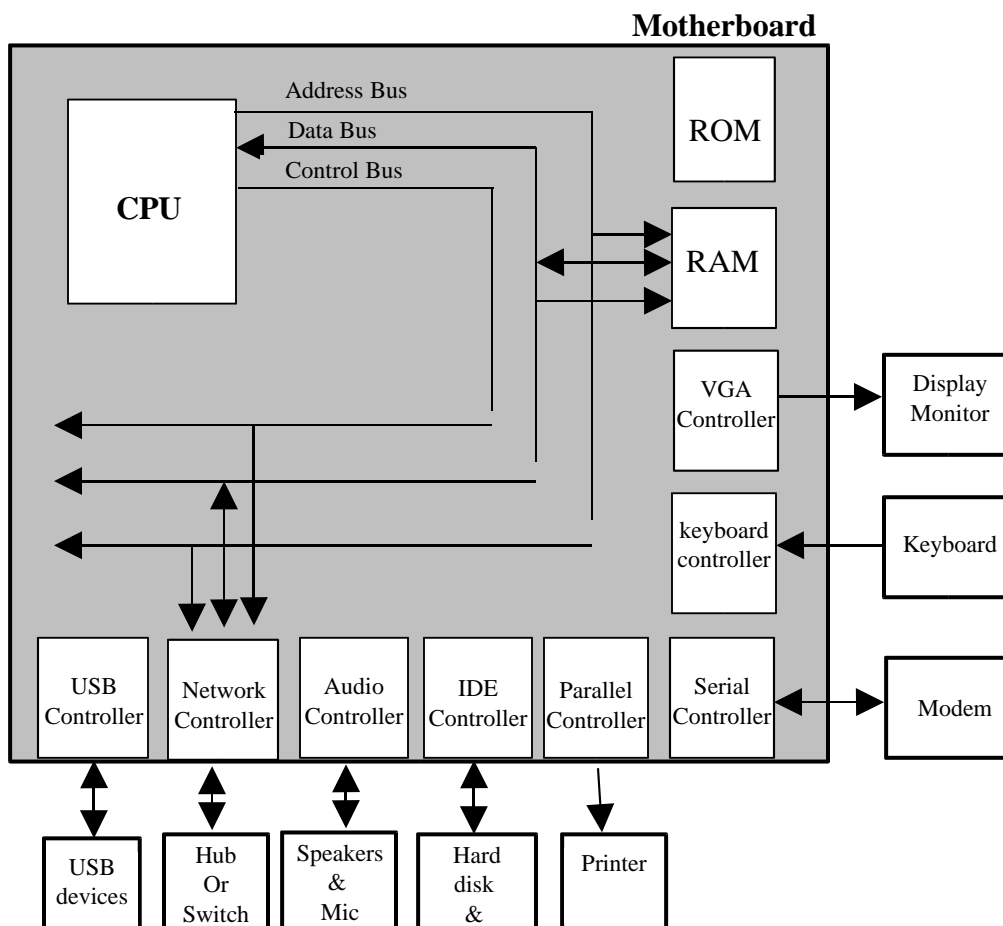
If you already written programs for any microprocessor kit such as 8085 kit or 8086 kit during your college days, then you have written programs to run directly on top of hardware.

So system programmers, who wish to program hardware directly, should have basic knowledge of computer hardware. Understanding how a computer works is fairly simple for engineering students who studied Microprocessors in their curriculum. But I guess any student could understand the following.

Inside a Computer

Any computer, how small or how large it may be, contains the following four important components.

- CPU or Microprocessor
- Memory (Non-volatile memory, Volatile memory)
- IO Controllers
- IO devices or Peripheral devices



In the above figure, motherboard is a printed circuit board (PCB) on which CPU, ROM, RAM and various I/O controller chips are mounted. The CPU is connected with these chips by using address bus lines, data bus lines and control bus lines. In the figure, the bus connections are shown only for two chips for clarity. But note that all chips are connected to these bus lines in the same way. IO devices lie outside the motherboard but connected to the corresponding I/O controllers through the cables.

CPU / Microprocessor

CPU or microprocessor is the most important component of a computer. It is responsible for fetching (reading) instructions from memory and executing them. These instructions are present in the memory in the form of opcodes.

CPU consists of a set of registers. These registers can be classified into General Purpose registers and Function Specific Registers. Following are the function specific registers:

- Instruction Pointer (also called Program Counter)
- Program Status Register (also called Flags register)
- Stack Pointer Register
- Frame Pointer Register

The moment power is given to a CPU, it will go through a reset process. During the reset process all the registers of the CPU will get initialized to default values. So the Instruction Pointer (also called Program Counter) register is also initialized to certain default address. After reset process, CPU fetches instruction (opcode) from this default address of memory into the CPU's decoding logic. Next it decodes the instruction and executes it. The instruction pointer register is incremented to the next instruction's address. CPU again fetches instruction from this address and executes. This goes on and on forever as long as CPU has got the power.

If we compare computer with human being, CPU role is like a brain's role. But if you consider how it works, CPU is like heart. CPU's fetch and execution cycle is as rhythmic as heart beat. As long as human being is alive its heart beats. Same way as long as CPU is powered, it tries to fetch and execute instructions. Like heart it goes on and on forever. When powered-on CPU simply executes the program present in the memory. So it is this program present in the memory that decides what computer is going to do.

Memory

Memory is of two types. Non-volatile memory typically called ROM/PROM/Flash does not lose its contents when power is removed. So once we write something in this memory, it will retain it, even if power is switched off. Typically non-volatile memory of a computer contains a fixed program in it.

The other type of memory is volatile memory typically called RAM or DRAM. Whatever we write into RAM, it retains it as long as power is present. The moment power is off, the contents

of RAM will be lost. So when we power on a computer, the contents of RAM at that time will be unknown or garbage values.

So it is a must for any computer to have a non-volatile memory and this memory must have valid program inside it. So when CPU is powered on, it fetches instructions from this non-volatile memory and executes them.

We call the program present in the non-volatile memory with different names. The program present in non-volatile memory of our personal computer (PC) is called a BIOS program. The program present in 8085 or 8086 microprocessor kit is called a 'Monitor' program. The program present in embedded development boards is called 'Boot loader' or 'Boot monitor'. In general, let us call this program as 'Boot Program'.

So note that every computer must have a non-volatile memory and in that memory, boot program must be present. So when computer is powered on, this boot program will get executed. The primary purpose of this boot program is to load some other large program into the volatile (RAM) memory and jumps to that RAM address. Loading some other program and jumping to it is called booting. That's why we call this program as boot program. We will discuss this booting process in detail later.

Note that memory always means ROM or RAM memory as we discussed above. These ROM and RAM are semiconductor chips, similar to CPU and present on the motherboard or Printed Circuit Board (PCB). We should not call hard disk, floppy disk and CD content as memory. These should be called as storage devices or storage memory.

The memory chips hold a set of locations inside them. Each location will have a unique address. The size of each location is one byte. So if a memory chip size is 1 MB. Then it contains one million memory locations, and each location is of size 1 byte. CPU is capable of reading or writing into any of these locations present in memory chips.

IO Controllers and IO Devices

It is very important to distinguish between IO controllers and IO devices. IO controllers are the semiconductor chips present on the motherboard along with CPU chip and memory chips. Whereas IO devices are peripheral devices attached to IO controllers and typically present outside the motherboard.

The primary purpose of any IO controller is to interface the IO device to the CPU. CPU cannot talk to IO device directly. CPU can talk to IO controller and IO controller in turn talks to IO device. In this way IO controller chips are acting as bridges between CPU and IO devices. Following are the list of IO controllers and the corresponding IO devices:

- Keyboard controller : Keyboard
- Serial controller : Modem
- Parallel controller : Printer
- Graphics or VGA controller : Display Monitor
- IDE controller : Hard disk or CD ROM drive
- Floppy controller : Floppy driver
- Network controller : LAN Switch or Hub
- Audio controller : Speaker or microphone

- USB controller : USB devices

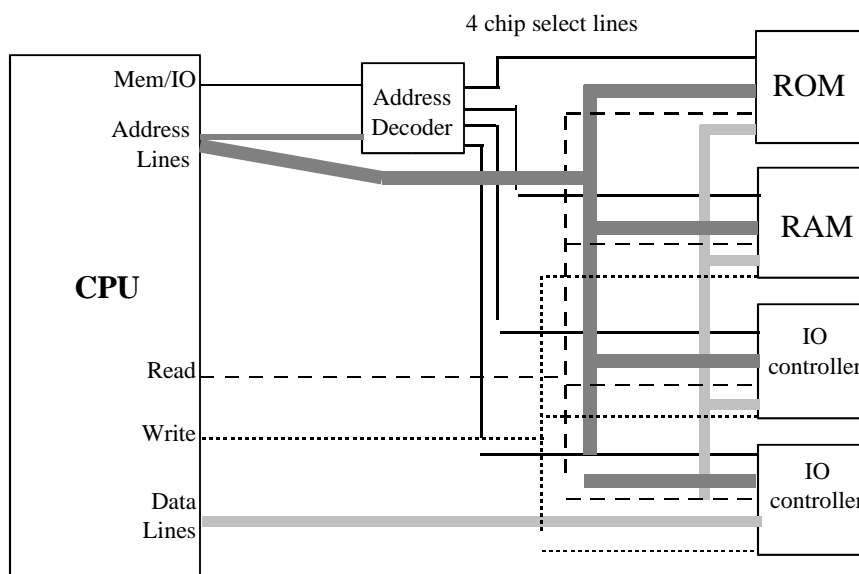
Similar to memory chips, IO controller chips also contain set of locations inside them. These locations are called IO registers or IO ports or simply IO locations. The number of IO registers present inside an IO controller varies from controller to controller. Simple IO controllers will have less than ten IO registers. Medium complex controllers will have 10 to 100 registers. But complex IO controllers may have more than 100 registers.

But note that memory chips typically will have millions of locations in them. Purpose of memory locations is to store program instructions and program data. But IO controller chips will have only few registers. But the purpose of these IO registers is to exchange data between CPU and IO devices, or to read the status of IO device, or to control the IO device.

The IO controller chips are connected to the CPU in the same way as memory chips. CPU can read or write to locations present in the IO controllers in the same manner as they do read or write into memory locations.

Interfacing CPU with Memory and IO controller chips and Bus cycles

The following figure shows how CPU is interfaced to the memory and IO controller chips.



The above figure is very important one to understand. This shows how CPU interacts with memory chips and IO controller chips. CPU interacts with these chips to do the following. Each interaction is called a bus cycle.

- Fetching an instruction (opcode) from memory (Instruction Fetch bus cycle)
- Reading a memory location of some address (Memory Read bus cycle)
- Writing to a memory location of some address (Memory Write bus cycle)
- Reading an IO register present in an IO controller (IO Read bus cycle)
- Writing to an IO register present in an IO controller (IO Write bus cycle)

CPU uses the following lines to perform the above interactions:

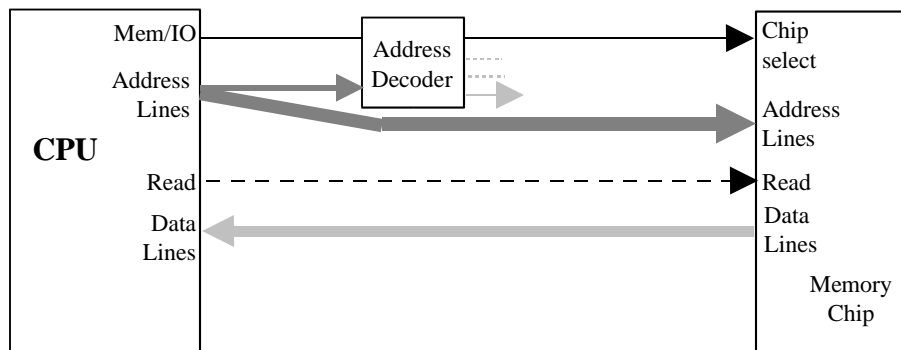
- Address lines
- Memory / IO select line
- Data lines
- Read line
- Write line

Memory chips and IO controller chips will have the following pins:

- Chip select
- Address lines
- Data lines
- Read line
- Write line

Memory Read Bus Cycle

When CPU wants to read any memory location it performs the following actions:

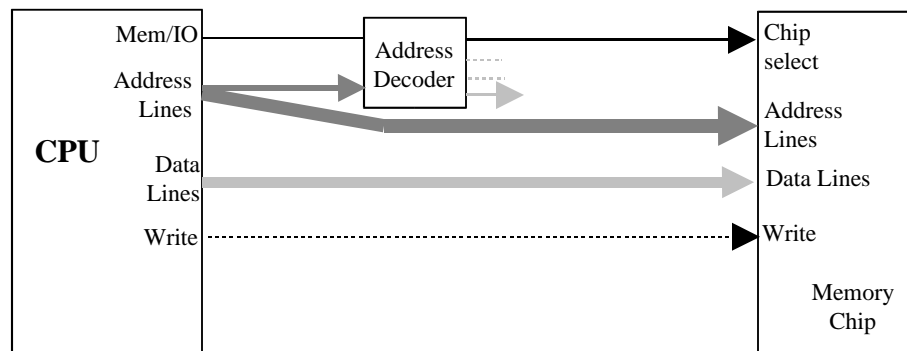


- CPU places the address of the memory location on the address lines. Some most significant address lines will go the address decoder and remaining address lines will reach address pins of all the chips
- While placing the address, CPU also sets Memory/IO line to memory state (This and above action will cause address decoder to activate only one of the 'chip select' line. The memory chip connected to this 'chip select' line will become active. This active memory chip will read the address on its address pins (came from CPU) and selects the corresponding memory location)
- CPU activates the 'read' line (Even though this read line reaches all the chips, only activated memory chip will respond to it)
- Active memory chip will place the data of the selected memory location on the data lines
- CPU will read the data present (came from active memory chip) on the data lines into its internal register

In this way CPU selects a memory location by placing its address on the address bus, next it activates read line so that selected chip would keep data on the data bus. Finally CPU reads this data from data bus. Note the direction of data transfer. It is from memory to CPU.

Memory Write Bus Cycle

When CPU wants to write to any memory location it performs the following actions:



- CPU places the address of the memory location on the address lines. Some most significant address lines will go the address decoder and remaining address lines will reach address pins of all the chips
- While placing the address, CPU also sets Memory/IO line to memory state (This and above action will cause address decoder to activate only one of the 'chip select' line. The memory chip connected to this 'chip select' line will become active. This active memory chip will read the address on its address pins (came from CPU) and selects the corresponding memory location)
- CPU places the value it wants to write on to the data lines. Similar to address lines these data lines will reach to all the chips
- CPU activates the 'write' line (Even though this write line reaches all the chips, only activated memory chip will respond to it)
- Active memory chip will read the data present on the data lines (placed by CPU) and writes to the selected internal memory location.

So to write to a memory location, CPU first places the address of that location on the address bus. This makes the chip and location to get selected. Next CPU keeps the data on the data lines and after that it activates the write line. When write line active, the selected memory chip reads the data from data bus and writes to the selected memory location. Note the direction of data transfer. It is from CPU to memory.

IO Read and Write Bus Cycles

When CPU wants to read or write to IO register present in the IO controller, the actions CPU performs are similar to the actions in memory read/write operations. Only difference is that, the Memory/IO line is set to IO state instead of Memory state. This causes one of the chip select

connected to IO controller will get selected. Now that selected IO controller chip will become active and responds to the CPU.

Instruction Fetch Bus Cycle

The Instruction fetch bus cycle is exactly similar to memory read bus cycle. Only difference lies in the cause or purpose to perform these two cycles. Instruction fetch cycles are automatically performed by the CPU, the moment power is applied. After fetching the instruction by fetch cycle, the CPU will execute that instruction. Execution of some instruction will be done completely inside the CPU. These instructions will not require any external bus cycles. But while executing some instructions the CPU will perform external bus cycles like memory read, memory write, IO read and IO write.

Following are the examples of instructions that will get executed internally without using any external bus cycles:

```
MOV  AX, BX    ; move bx into ax register
INC  AX        ; Increment register AX
ADD  AX, CX    ; Add CX to AX
OR   AX, BX    ; OR BX with AX
```

Following are the examples of instructions that perform external bus cycles while executing:

```
MOV  AX, [BX] ; Read memory pointed by BX, into AX (memory read cycle)
MOV  [AX], BX ; Write BX into memory pointed by AX (memory write cycle)
IN   AX 378H ; Read IO port at address 378H into AX (IO read cycle)
OUT  378H, AX ; Write AX to IO port at address 378H (IO write cycle)
```

Executing a Program on a computer

Assume that this computer does not loaded with any operating system. First of all this computer do not have any hard disk. This computer has got only 'Monitor' program in the ROM. This monitor program is similar to monitor program present the 8085 or 8086 kit. When computer is powered on, this monitor program runs and provides user interface commands to read or write into memory locations. Monitor program also supports 'Go' command, which loads the Program Counter register with given address.

To execute a program on computer we have to write a program first. Next convert that program into opcodes as understood by the CPU. Next load those opcodes into the computer's memory (RAM) by using the monitor commands. Finally load the Program Counter register of CPU with the starting address of this program by using 'Go' command of the monitor.

Simple Assembly Program to add 10 numbers

Let us write a small Pentium assembly language program to add 10 numbers and to store the result in another memory location. This program assumes that starting address of program will be 100000H i.e 1 MB. And 10 numbers are stored in the memory starting from 100100H and the final sum stored in the memory location 100200H.

```
MOV  EAX, 0      ; Load EAX register with zero
MOV  EBX, 100100H ; Load EBX with address of first number
MOV  ECX, 10     ; Load ECX with count of numbers
```

```

LOOP:
MOV   EDX, [EBX]    ; Load EDX with number pointed by EBX
ADD   EAX, EDX      ; Add contents of EDX to EAX
ADD   EBX, 4        ; Increment EBX by 4 to point next number
DEC   ECX           ; Decrement ECX count by one
JNZ   LOOP         ; ECX is not reached to zero jump to loop
MOV   EBX, 100200H  ; Load EBX with Sum location address
MOV   [EBX], EAX    ; Load Sum present EAX to Sum location
INT   3

```

Simple Assembly Program to read scan code from the Keyboard

The above program is accessing only memory locations. It is not accessing any I/O devices. So let us write one more program, which reads a scan code from the keyboard device. The keyboard device is connected to the keyboard controller. The keyboard controller has got two IO registers. One register is called DATA register and other is STATUS register.

Whenever a key is pressed on the keyboard, It will send a scan code to the keyboard controller. The keyboard controller, when scan code is received, it loads the scan code into the DATA register and sets 0th bit in the STATUS register. So our program will read the STATUS register into a CPU register and verifies the 0th bit. If 0th bit is zero, the program continues to read the STATUS register till bit zero is set. Once bit zero is set, the program reads the scan code from the DATA register of keyboard controller.

```

LOOP:
IN    AL, 61H      ; Read STATUS register of keyboard controller
AND   AL, 01H     ; AND with DATA READY Bit
JNZ   LOOP        ; If not ready jump to loop
IN    AL, 63H     ; Read the scan code from the DATA register

```

Running an Operating System

Microprocessor kits and Embedded development boards will have debug monitor program in its ROM. This monitor program allows us to load opcodes into the memory and execute them. Where as PC motherboard will have a BIOS program inside the ROM. The BIOS program loads the operating system present in the boot device such as hard disk or CD-ROM into RAM and jumps to it. In this way PC starts running the operating system such as Linux or Windows.

Review Questions

- What are the four important components present in side a computer?
- What are the two types of memory?
- What is the difference between storage memory and RAM/ROM memory?
- What is the difference between IO controllers and IO devices and give examples.
- What are the registers present in side a CPU?
- What is the purpose of registers present inside an IO controller chip?
- List all the bus cycles performed by a CPU?
- What is the difference between memory read cycle and instruction fetch cycle?
- What are the two registers present inside a keyboard controller?
- What is boot program?
- What BIOS program present in the ROM of motherboard will do?