Introduction

0.1 The course

1. Basic structure of a digital computer, Number representation, Digital logic, Integer -unsigned, signed representation; Characters-ASCII coding, other coding schemes; Real numbers-fixed and floating point, IEEE754.

2. Basic functional blocks of a computer: CPU, memory, input-output subsystems, control unit.

3. Instruction set architecture of a CPU - registers, instruction execution cycle, RTL interpretation of instructions, addressing modes, instruction set.

4. Instruction set architecture CISC and RISC, Case study - instruction sets of some common CPUs.

5. Basic building blocks for the ALU, Adder, Subtractor, Shifter, Multiplication and division circuits, CPU Subblock, Datapath - ALU, registers, CPU buses;

6. Control unit design: Hardwired and microprogrammed design approaches

7. Memory system design: semiconductor memory technologies, memory organization. Cache; Cache memory hierarchy;

8. Peripheral devices and their characteristics: Input-output subsystems, I/O transfers - program controlled, interrupt driven and DMA, Secondary storage devices;

9. Privileged and non-privileged instructions, software interrupts and exceptions, programs and processes, role of interrupts in process state transitions.

10. Pipelining: Basic concepts of pipelining, throughput and speedup, pipeline hazards.

0.2 Books, References and Evaluation

- David A. Patterson and John L. Hennessy, Computer Organization and Design: The Hardware/Software Interface, Elsevier.
- Carl Hamachar, Zvonco Vranesic and Safwat Zaky, Computer Organization, McGraw Hill.
- John P. Hayes, Computer Architecture and Organization, McGraw Hill.
- William Stallings, Computer Organization and Architecture: Designing for Performance, Pearson Education.
- Vincent P. Heuring and Harry F. Jordan, Computer Systems Design and Architecture, Pearson Education.
- Computer System Architecture, M. Morris Mano, Prentice-Hall, 1993, Third Edition
- Structured Computer Organization, Andrew S. Tenenbaum, Pearson Prentice Hall, 2006, Fifth Edition
- Net, Wikipedia, OCW MIT
- http://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6 823-computer-system-architecture-fall-2005/lecture-notes/
- http://ocw.mit.edu/courses/electrical-engineering-and-computer-science/

Class test, attendance, Midsem, endsem evaluation:

- 15% Quizes, Assignments
- 15% first midsem, 15% II midsem
- 10% Semester Project
- 40% endsem
- 5% Attendance

0.3 Preliminaries

How do you define a computer? In fact, a computer is a system, built using electronics comprising digital system, having combinational and sequential logic circuits. It can be thought of as a big, *finite state machine*. A computer just does what the software tells it to do.

A Software is a series of instructions. With respect to the instructions, we would like to know - What instructions does a computer need? What kinds of instructions are there, and how do we represent instructions?

The next thing we would like to know is - what is computer architecture? An architecture is the attributes of a computer seen by the machine language programmer, for example, the instruction set of the computer, CPU registers and their sizes, range of memory addresses, and addressing modes, various flags in the CPU's flag register, like over-flow of result, result positive or negative, carry generated or not, etc.

Strictly speaking, a computer architecture specifies what the hardware looks like (its interface), so that we can write software to run on it. Exactly what instructions does it have, number of register storage locations, etc. A Computer architecture includes:

1. Instruction set
2. Instruction format
3. operation codes
4. addressing modes
5. all registers and memory locations that may be directly manipulated or tested by a machine language program, and
6. formats for data representation

The next curiosity is - why are the different types of computers, how do we classify those, and how do we tell computers what to do?

0.4 System Organization

The system organization comprises of: CPU, ALU, Control unit, memory, and the buses for connection between these components. The figure 1 shows the various functional blocks.

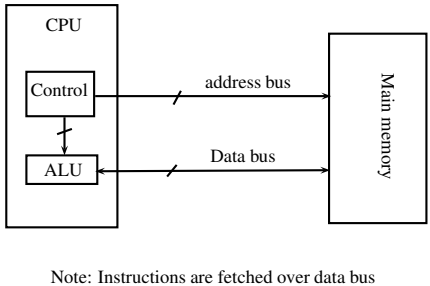

Figure 1: Functional block diagram of computer.

0.4.1 Functional Blocks

The figure 2 shows various functional blocks with input / output Sub-blocks.

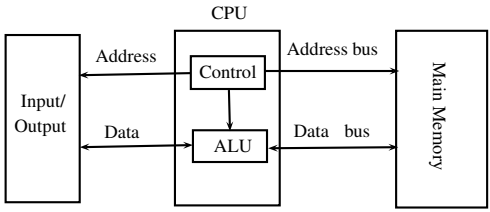

Figure 2: Functional block diagram of computer with IO.

Various connections between Sub-blocks are buses, with size of: 8, 16, 32, 64-bits. The older systems comprised the buses sizes of 8, 12, 24, 40-bits.

One of the standard model of computing is is "Von Neumann Model", comprising of Arithmetic and Boolean logic, memory: R/W, Execution Control (branches and Jumps). The bottlenecks of Von Neumann architecture is excessively dependent on addresses, where every memory access must start by sending memory addresses, and other disadvantage are sequential execution and centralized control.

0.4.2 Single-bus Architecture Functional blocks

The single bus system has low cost and flexibility for attaching IO devices. However, it allows only one transfer at a time. One of the difficulty of single bus system is speed mismatch between various IO devices con-

nected to the same bus. It is resolved using buffer registers for devices for storing the data when transmitter is delivering at a speed faster than it can be received by the device at the receiving end. Figure 3 shows the structure of a single bus system.

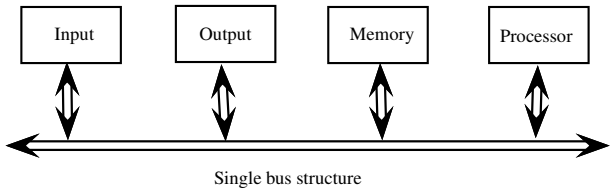

Figure 3: Functional block diagram of computer with IO, connected to a single bus.

0.4.3 The memory

The memory, which holds the programs and data is called primary memory of computer. The memory can be thought of as an array of storage elements of uniform size, say 8-bits or 16-bits or more, usually multiple of 8-bits. The figure 4 shows a memory total size of n-locations, each of one-byte. The `ADD A,B` and `STORE A,2000` are instruction in *Assembly language*. The First instruction indicates that value in register B shall be added into the register A, while the next indicates that value in register A shall be stores at memory location 2000. The `ADDA,B` is of small length, hence occupies a single location (one-byte), while the *store* instruction occupies two memory location, due to address field 2000.

The *pneumonics* like, ADD, STORE, represent operation to be performed by the CPU as dictated by the instruction, hence this peneumanic is called *opcode* field of the instruction. The remaining part, like A, B and $A, 2000$ are called address / *operand* field.

0.5 Generations of Computers

The present computers are descendants of many generations of their predecessors. The technology was the dominant factor in computer design and evolutions. The start generation is called *0th generation*. The main feature of this generation was mechanical / electromechanical devices to build the computers. Following are some of the landmark system of 0th generation computers.

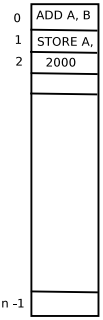

Figure 4: RAM memory of n locations.

1. Liebniz's calculator (1685), Joseph Jacquard loom (1805)
2. Charles Babbage's difference and analytical engines (1833, 1837, 1853)
3. Herman Hollerith's census tabulator (1890)
4. Howard Aiken's (Harvard) Mark-I (1944)
5. John Von Neumann: stored program concept (i.e., *program* and *data* in same memory)(1940s)

The next generation is called 1st generation, was due to arrival of vacuum tubes, which replaced the mechanical and electromechanical parts. The landmark computers of this generation were:

1. ENIAC (1946), UNIVAC (1951)
2. IAS machine (1952)
3. IBM 701 (1953), IBM 709 (1958)

The 2nd generation was due to transistors, which replaced the vacuum tubes. Since transistors were compact and consumed far less power than vacuum tubes, these computers were compact, and consumed far less power. The popular system of 2nd generation were:

1. DEC PDP-1 / 4 / 7 / 9 / 15
2. DEC PDP-5 / 8 / 12
3. DEC PDP-6 / 10

4. IBM 7090 / 7094
5. IBM 1401, IBM 1620, CDC1604, CDC 6600

The arrival of third generation was due to integrated circuits. The ICs consumed far less power as well as the space compared to transistors, as well, their failure rate and temperature characteristics were superior than the transistors. Following were the major system of this generation.

1. IBM 360/370
2. DEC PDP-8/I, DEC PDP-11/40, DEC VAX 11/780

The 4th generation was due to arrival of next technological advance - the very large scale integration (VLSI) technology. The following are the major landmark systems of this generation:

1. Intel 4004, Intel 8008, Intel 8080 / 8085, Zilog Z80 / Z8 / Z8000
2. Intel 8086 / 8088, Intel 80186 / 80286 / 80386 / 80486
3. Intel IA-32: Pentium, PII, PIII, p4, Celeron, Xeon...
4. Motorola 6800, / 68010 / 68020 / 68030/68040/68060 ... DEC PDP-11/03, DEC MicroVAX
5. SPARC-1 / SPARC-2 / SuperSPARC, HyperSPARC, UltraSPARC, IBM RISCSystem-6000, Power series
6. DEC Alpha,

The 5th generation has property of Homogeneous and parallel processors.

0.6 System Software

Unlike *application software*, which is not not hardware dependent, and designed for specific applications, like - editors and word processors, the *system software* is hardware dependent hence requires the knowledge of system for writing it. The system software has following uses:

1. Receiving and interpreting user commands
2. Managing the storage, file I/O
3. Running standard programs, like spreadsheet, word-processors,..

4. Controlling IO devices
5. Translation of programs
6. Linking, loading, etc.

The figure 5 demonstrates the sharing of processor by user program and OS routine. It also shows how to use the resources in an efficient way. The time duration $t_0 - t_1$: is for application program loading from device to memory, $t_1 - t_2$: is time when the application program runs, $t_2 - t_3$: is required for loading data file from device into the primary memory. during the time $t_3 - t_4$: the application run and produces results by performing computations on the data, and finally, $t_4 - t_5$: time slot prints the results. At time instance t_5 another program starts.

In the above example, there are four resources in the y-axis, printer, disk, operating system routine, and program. When, none of the disk, program, and printer are in operation, then OS-routine is running.This true. Since, when no work is being carried out by the computer, then the CPU is executing OS routine.

With above we can appreciate that how may steps are needed to run an application program. Following are more observations:

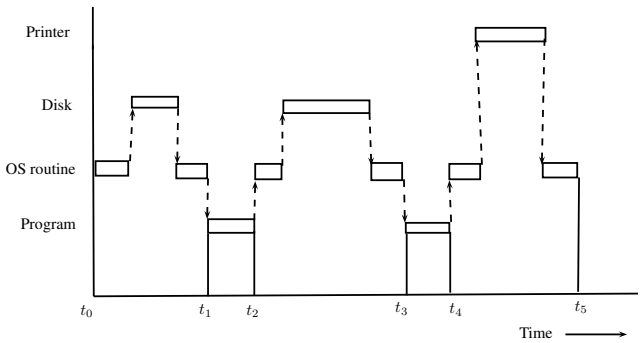

Figure 5: User program and OS sharing CPU, in time-division multiplexing.

A CPU will first fetch the instruction from memory, then it *decodes* it to find out what operation is to be performed and what are the operands. Also, due to decoding, CPU will come to know, whether the data is given along with the instruction, or they are to be fetched from an address specified along with the instruction. The fetching, decoding, and execution activities carried out for an instruction, are together called *instruction cycle*.

- During the time duration $t_4 - t_5$: CPU and disk are free. Similarly, during $t_0 - t_1$, and $t_2 - t_3$ CPU and printer free. Hence, if it is possible to use some of the resources, like CPU, disk, memory, etc., during the time they are free, higher throughput is possible.

- If OS concurrently execute several programs, better utilization of resources possible. This pattern is called *multiprogramming*, *multitasking*, and *time-sharing*.

0.7 Performance

The performance is indicated by the fact, how quickly can a CPU execute programs. This is function of hardware and Machine language instructions. Does a compiler effect it? The answer is yes. A compiler may generate very inefficient machine code, and good quality compilers generate codes which executes much faster. The total time for execution in our exercise=$t_5 - t_0$ (elapsed time): it is effected by: speed of processor, disk, printer. The processor time dependents on hardware (i.e., the net speed of CPU + Mem + Cache).

Lets us assume that a processor's clock has clock time-period P, then rate or frequency of execution can be not more than $R = \frac{1}{P}$, provided that it completes a smallest operation in a clock cycle. Thus, to increase the speed of program execution, we must increase R. Let N is total number of instructions in a program P. Assume that S is average basic steps needed per instructions. Thus, the total execution time T for program P in seconds is,

$$T = \frac{N \times S}{R} \tag{1}$$

To decrease execution time T we need to reduce the total number of instructions N, this will bring down the average basic steps needed per instruction, i.e., S, and this will increase R.

0.8 Advanced Processors

To understand the significance of advanced processors, consider the following example:

- Instruction: *ADD R1, R2, R3*: $(R_3 \leftarrow R_1 + R_2)$

- Instruction cycle time = Fetch time + decode time + execute time

$$t_i = t_f + t_d + t_e$$

- Next Instruction can be read, while previous addition takes place.
- This results to overlapping execution (called pipelining). Ideally $|S| = 1$.
- Higher degree of execution possible by multiple instruction-pipelines (called super-scalar architecture)

Exercises

1. Discuss the advantages and disadvantages of storing the programs and data in the same memory. Under what circumstances is it desirable to have these in separate memories?
2. What are the functions performed by each of the following: CPU, Memory, OS, Compiler.
3. What is the difference between types of operations performed by first generation computers and present generation computers?
4. What is difference between *Computer Organization* and *Computer Architecture*. What is open Architecture? (Hint: Open Source)
5. Suggest two examples in each case: 1. Too large data but computation is small, 2. Too small data but computation is large.
6. Write a small C program with nesting of loops so that code size is small but should take more time. Run this program with following command in linux and justify the answer you get as well as explain the answer. `$ time executable-prog-name`
7. Does human brain do execution of instruction, like computers do? Justify.
8. Does human brain perform parallel computation or has potential for this? Suggest your logic.
9. Can you suggest some other models of computation than the Von Neumann?
10. Is the evolution process computational ? Justify you answer.
11. Find out by a command of linux by which we can say that linux is a multiprogramming OS.

12. Find out the linux commands for: size of RAM, memory area, OS area, etc.

13. How can you work editing a file and at the same time run another program in linux?

14. Find out the hardware specifications of your laptop/notebook.