Programme: **BSc – Computer Science**

Course Code: **COM 120**

Course Title: **SYSTEM HARDWARE**

Lecturer: **Francis Onyango (*BSc, Meng*)**

COURSE SUMMARY

This is an introductory course to computer systems hardware. The student will not only learn the underlying concepts, but will also perform some practical exercises to actualize the lessons learnt.

Course objectives

At the end of the course, the student should be able to

- explain the basic principles of processor structure
- identify various I/O devices and their diversity
- explain the principles underlying communications between computers at the physical and data link levels

EXAMINATIONS

- The University and School of Science, Technology and Engineering examinations regulations shall apply
- The University Common Rules and Regulations for undergraduate Examinations shall apply.
- Examinations shall be held at the end of the semester in which courses are taught
- Admission to Examination will depend on satisfactory attendance of the prescribed courses as per senate regulations.
- Continuous assessment tests, assignments account for 30% and, the final three-hour written examination will account for 70% of the final grade.

Course delivery

The mode of executing this course will be by:

- Classroom lectures
- Research projects and discussion groups
-

WEEK 1: Introduction to systems hardware

- The structure of the von Neumann computer
- Data bus

Week 2/3: Processor structure

- Modern processor technologies
- Data and Instructions
- Instruction cycles, systems interrupts
- Instruction Set Processor (ISP) level view of computer hardware,
- Assembly language level use.

Week 4/5: Memory systems

- RAM and disks
- Hierarchy of memories
- Modern memory technologies: DIMMs, SIMMs
- Modern memory technologies: SRAMs, DRAMs

Week 6: CAT 1

Week 7/8: I/O organization,

- I/O devices and their diversity,
- I/O interconnection to CPU and Memory.

Week 9/10: data communications

- Communication between computers at the physical level.
- Networks and computers.

Week 11: CAT 2

Week 12: Revision and Course Overview

Week 15/16: end of semester exams

Recommended references:

Gary B. Shelly & Misty E. Vermaat (2011); Discovering Computers: living in an digital world, complete; Course Technology, Cengage Learning, Boston USA

Hennessy, John L. & David A. Patterson (2012); with contributions by Andrea C. Arpaci-Dusseau . . . [et al.]. Computer Architecture: a quantitative approach, 5th Ed. Morgan Kaufmann is an imprint of Elsevier, 225 Wyman Street, Waltham, MA 02451, USA

William Stallings (2010), Computer Organization and Architecture, Pearson Education, Inc., Upper Saddle River, New Jersey, 07458.

1 WEEK 1: Introduction to systems hardware

- The structure of the von Neumann computer
- Data bus

1.1 Introduction to computer technology

Introduction: this lesson provides a background to the development of information technology over the years. It also discusses the contribution of various researchers at different times in this development.

Objectives:

at the end of this lesson you are expected to understand and describe the following IT concepts:

- the evolution of computers
- Characteristics of computers
- how to evaluate the performance of a computer

1.1.1 The evolution of computers

Man has always been in search of mechanical aids for computation. The development of the abacus around 3000 BC introduced the positional notation of number systems. In seventeenth-century France, Pascal and Leibnitz developed mechanical calculators that were later developed into desk calculators. In 1801, Jacquard used punched cards to instruct his looms in weaving various patterns on cloth.

Table 1-1: An Operator entering data onto punched cards (source: www.computerhistory.org)

In 1822, Charles Babbage, an Englishman, developed the difference engine, a mechanical device that carried out a sequence of computations specified by the settings of levers, gears, and cams. Data were entered manually as the computations progressed. Around 1820, Babbage proposed the analytical engine, which would use a set of punched cards for program input, another set of cards for data input, and a third set of cards for output of results. The mechanical technology was not sufficiently advanced and the analytical engine was never built; nevertheless, the analytical engine as designed probably was the first computer in the modern sense of the word.

Several unit-record machines to process data on punched cards were developed in the United States of America in 1880 by Herman Hollerith for census applications. In 1944, Mark I, the first automated computer, was introduced. It was an electromechanical device that used punched cards for input and output of data and paper tape for program storage. The desire for faster computations than those Mark I could provide resulted in the development of Electronic Numerical Integrator and Calculator (ENIAC), the first electronic computer built out of vacuum tubes and relays by a team led by Americans Eckert and Mauchly. ENIAC employed the stored-program concept in which a sequence of instructions (i.e., the program) is stored in the memory for use by the machine in processing data. ENIAC had a control board on which the programs were wired. A rewiring of the control board was necessary for each computation sequence. See figure 1-2 below.

Table 1-2: Part of the ENIAC computer. Note the many cables on the left (source: commons.wikimedia.org)

John von Neumann, a member of the Eckert–Mauchly team, developed Electronic Discrete Variable Automatic Computer (EDVAC), the first stored-program computer. At the same time, Wilkes developed Electronic Delay Storage Automatic Calculator (EDSAC), the first operational stored-program machine, which also introduced the concept of primary and secondary memory hierarchy. Von Neumann is credited for developing the stored-program concept, beginning with his 1945 first draft of EDVAC. The structure of EDVAC established the organization of the stored program computer (von Neumann machine), which contains five main components:

i. An input device through which data and instructions can be entered
ii. A Read/write, random access memory storage unit into which both program instructions, data and results can be entered and from which instructions and data can be fetched
iii. An arithmetic logic unit to perform basic arithmetic operations and process data
iv. A control unit to fetch, interpret, and execute the instructions from the storage and then ***sequentially*** coordinates operations to accomplish the programmed task
v. An output device to deliver the results to the user

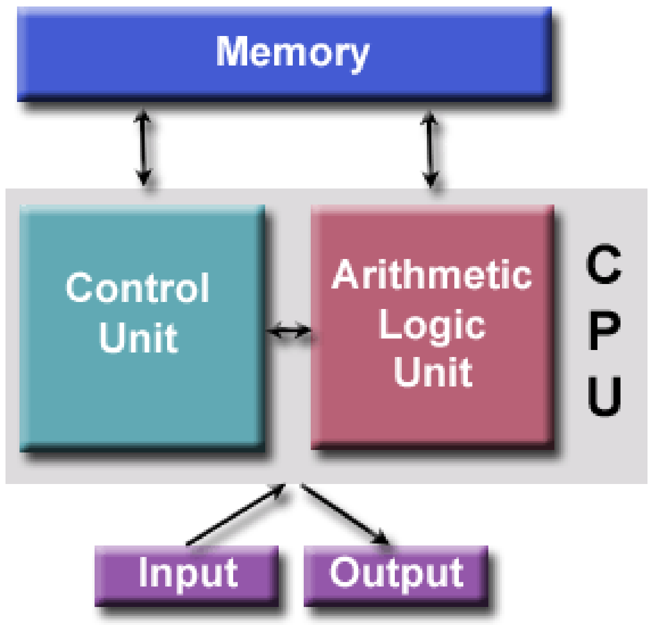


Table 1-3: The von Neumann Architecture

Figure 1-3 is a graphical representation of the Von Neumann computer. It was named after the Hungarian mathematician, John von Neumann, who first authored the general requirements for an electronic computer in his 1945 papers. Since then, virtually all computers have followed this basic design, differing from earlier computers which were programmed through "hard wiring".

All contemporary computers are von Neumann machines, although various alternative architectures have evolved. This blueprint is the basis for all mainstream computer systems today, and its inherent problems still prevail:

- Instructions and data must be continuously fed to the control and arithmetic units, so that the speed of the memory interface poses a limitation on compute performance. This is often called the von Neumann bottleneck. In the following sections and chapters we will show how architectural optimizations and programming techniques may mitigate the adverse effects of this constriction, but it should be clear that it remains a most severe limiting factor.

- The architecture is inherently sequential, processing a single instruction with (possibly) a single operand or a group of operands from memory. The term SISD (Single Instruction Single Data) has been coined for this concept.

Despite these drawbacks, no other architectural concept has found similarly widespread use in nearly 70 years of electronic digital computing.

1.1.2 Generations of Computer Technology

Commercial computer system development has followed development of hardware technology and is usually divided into four generations:

1. First generation (1945–1955) — vacuum tube technology.
2. Second generation (1955–1965) — transistor technology.
3. Third generation (1965–1980) — integrated circuit (IC) technology.
4. Fourth generation (1980–to-date) —Very Large Scale Integrated (VLSI) circuit technology.

I will not elaborate on the architectural details of the various machines developed during the three generations, except for the following brief evolution account.

First-generation computers

These machines (such as ENIAC, EDVAC, UNIVAC 1 and IBM 701), built out of vacuum tubes, were slow and bulky and accommodated a limited number of Input/output devices. Magnetic tape was the predominant I/O medium. Data access time was measured in milliseconds. UNIVAC I (see Fig.1-4) used 5,200 vacuum tubes, weighed 29,000 pounds (13 metric tons), consumed 125 kW, and could perform about 1,905 operations per second running on a 2.25 MHz clock. The Central Complex alone (i.e. the processor and memory unit) was 4.3 m by 2.4 m by 2.6 m high. The complete system occupied more than 35.5 m² of floor space.

Table 1-4: UNIVAC 1 (note the machine used to enter data using punched cards in the foreground and the magnetic tape drives in the background)

The Electronic Numerical Integrator Computer (ENIAC) was built at the University of Pennsylvania as a response to the USA wartime needs of the 1st World war. It was basically designed to assist in developing accurate range and trajectory tables for newly developed weapons. The technology used did no have a convenient programming provision.

The von Neumann Machine was developed to address the challenge of programming the ENIAC. There was a need to store a program and its data so that the computer could read them from memory during execution. In addition the programs could easily be modified by setting the values in a portion of memory. Figure 1-5 shows a picture of memory that was used in UNIVAC1.

Table 1-5: Mercury delay line memory of UNIVAC I (www.wikipedia.org)

Second-generation computers

These pioneer computer systems used random-access core memories, transistor technology, multifunctional units, and multiple processing units. Data access time was measured in microseconds. Assembler and high-level language were developed.

Third generation computers

The integrated-circuit technology used in third-generation machines such as the IBM 360, UNIVAC 1108; ILLIAC-IV, and CDC STAR-100 contributed to Nano-second data access and processing times. Multiprogramming, array, and pipeline processing concepts came into being. Computer systems were viewed as general-purpose data processors until the introduction in 1965 of DEC PDP-8, a minicomputer (Fig. 1-6). Minicomputers were regarded as dedicated application machines with limited processing capability compared to that of large-scale machines. Since then, several new minicomputers were introduced and this distinction between the mini and large-scale machines became blurred due to advances in hardware and software technology.

Table 1-6: A PDP-11, model 40, an early member of DECs 16-bit minicomputer family, on display at the Vienna Technical Museum (www.wikipedia)

The decline of the minis happened due to the lower cost of microprocessor-based hardware, the emergence of inexpensive and easily deployable local area network systems, the emergence of the 68020, 80286 and the 80386 microprocessors, and the desire of end-users to be less reliant on inflexible minicomputer manufacturers and IT departments or "data centers". The result was that minicomputers and computer terminals were replaced by networked workstations, file servers and PCs in some installations, beginning in the latter half of the 1980s.

Fourth generation computers

The development of microprocessors in the early 1970s allowed a significant contribution to the third class of computer systems: microcomputers. Microprocessors are essentially computers on an integrated-circuit (IC) chip that can be used as components to build a dedicated controller or processing system. Advances in IC technology leading to the current VLSI era have made microprocessors as powerful as minicomputers of the 1970s. VLSI-based systems are called fourth-generation systems since their performance is so much higher than that of third-generation systems. Figure 1-7 below shows one of the microcomputers in the 1970s.

Table 1-7: The Compaq Portable was the first 100% IBM-compatible PC, and the first portable one (www.wikipedia.org).

During the 1990s, the change from minicomputers to inexpensive PC networks was cemented by the development of several versions of UNIX and Unix-like operating systems that ran on the Intel x86 microprocessor architecture, including Solaris, Linux, FreeBSD, NetBSD and OpenBSD. Also, the Microsoft Windows series of operating systems, beginning with Windows NT, now included server versions that supported preemptive multitasking and other features required for servers.

Modern computer system architecture exploits the advances in hardware and software technologies to the fullest extent. Due to advances in IC technology that make the hardware much less expensive, the architectural trend is to interconnect several processors to form a high-throughput system. Some claim that we are now witnessing the development of fifth-generation systems. However, there is no consensus on the accepted definition of what a fifth-generation computer is.

Fifth-generation computers

Development efforts in the United States involve building supercomputers with very high computational capability, large memory capacity, and flexible multiple-processor architectures, employing extensive parallelism. The Japanese fifth-generation activities aimed toward building artificial intelligence-based machines with very high numeric and symbolic processing capabilities, large memories, and user-friendly natural interfaces. Some attribute fifth generation computers to biology-inspired (neural networks, DNA) computers and optical computer systems.

The current generation of computer systems exploits parallelism in algorithms and computations to provide high performance. The simplest example of parallel architecture is the Harvard architecture, which utilizes two buses operating simultaneously. Parallel processing architectures utilize a multiplicity of processors and memories operating concurrently.

Table 1-8: AMD Opteron Supercomputer

Table 1-9: Summary of Generations of Computer

	Technology	**Period**
1	Vacuum tube -	1946-1957
2	Transistor -	1958-1964
3	Small scale integration - Up to 100 devices on a chip	1965 on to 1971
4	Medium scale integration - 100-3,000 devices on a chip	
5	Large scale integration - 3,000 - 100,000 devices on a chip	1971-1977
6	Very large scale integration - 100,000 - 100M devices on a chip	1978 to date
7	Ultra large scale integration - Over 100 M devices on a chip	Future

Things to do:

Look up the characteristics that distinguish between the following computer systems: Supercomputers, Mainframe, large-scale computers, Minicomputers, Microcomputers, Desktops, Laptops, Tablet PCs. List them in a table.

1.1.3 Evaluating the Performance of a computer system

The performance of smaller computer system can be evaluated based on the number of instruction cycles that can be executed per second. However, there are several measures of performance have been used in the evaluation of large computer systems based on the levels precision of computations. The most common ones are: million instructions per second (MIPS), million operations per second (MOPS), million floating-point operations per second (MFLOPS or megaflops), billion floating-point operations per second (GFLOPS or gigaflops), and million logical inferences per second (MLIPS).

Machines capable of trillion floating-point operations per second (Teraflops) are now available. Table 1.2 below lists the common prefixes used for these measures. Power of 10 prefixes is typically used for power, frequency, voltage, and computer performance measurements. Power-of-2 prefixes are typically used for memory, file, and register sizes. The measure used depends on the type of operations one is interested in, for the particular application for which the machine is being evaluated. As such, these measures have to be based on the mix of operations representative of their occurrence and the application.

Table 1-10: Common Prefixes Used in Computer Systems Measurements

Power-of-10	**Power-of-2**	**Prefix**	**Symbol**
Thousand (10^3)	2^{10}= 1024	Kilo	K
Million (10^6)	2^{20}= 2048	Mega	M
Billion (10^9)	2^{30}	Giga	G
Trillion (10^{12})	2^{40}	Tera	T
Quadrillion (10^{15})	2^{50}	Peta	P
Quintillion (10^{18})	2^{60}	Exa	E
Sextillion (10^{21})	2^{70}	Zetta	Z
Septillion (10^{24})	2^{80}	Yotta	Y
Thousandth (10^{-3})	2^{-10}	Milli	M
Million (10^{-6})	2^{-20}	Micro	m
Billion (10^{-9})	2^{-30}	Nano	N
Trillion (10^{-12})	2^{-40}	Pico	P
Quadrillion (10^{-15})	2^{-50}	Femto	F
Quintillion (10^{-18})	2^{-60}	Atto	A
Sextillion (10^{-21})	2^{-70}	Zepto	Z

Septillion (10^{-24})	2^{-80}	Yocto	Y

The performance rating could be either the peak rate (i.e., the MIPS rating the CPU cannot exceed) or the more realistic average or sustained rate. In addition, a comparative rating that compares the average rate of the machine to that of other well-known machines is also used. In addition to the performance, other factors considered in evaluating architectures are: generality (how wide is the range of applications suited for this architecture), ease of use, and expandability or scalability.

One feature that is receiving considerable attention now is the openness of the architecture. The architecture is said to be open if the designers publish the architecture details such that others can easily integrate standard hardware and software systems to it. The other guiding factor in the selection of architecture is the cost. Several analytical techniques are used in estimating the performance. All these techniques are approximations, and as the complexity of the system increases, most of these techniques become unwieldy. A practical method for estimating the performance in such cases is using benchmarks.

1.1.3.1 Benchmarks

Benchmarks are standardized batteries of programs run on a machine to estimate its performance. The results of running a benchmark on a given machine can then be compared with those on a known or standard machine, using criteria such as CPU and memory utilization, throughput and device utilization, etc. Benchmarks are useful in evaluating hardware as well as software and single processor as well as multiprocessor systems. They are also useful in comparing the performance of a system before and after certain changes are made. As a high-level language host, computer architecture should execute efficiently those features of a programming language that are most frequently used in actual programs. This ability is often measured by benchmarks. Benchmarks are considered to be representative of classes of applications envisioned for the architecture.

Summary

This lesson covered the early history of computer technology. This included the different generations of computers and the underlying technologies. It also discussed the performance indicators of computers.

2 Week 2/3: Processor structure

- Modern processor technologies
- Data and Instructions
- Instruction cycles, systems interrupts
- Instruction Set Processor (ISP) level view of computer hardware,
- Assembly language level use.
-

Lesson introduction:

The primary function of a digital computer is to process data input to it to produce results that can be better used in a specific application environment. This section introduces you to the functional units of a computer system

By the end of this lesson, you are expected to be able to:

1. To explain how a computer operates in terms of the fetch-execute cycles and the role of the data bus
2. Understand a computer in terms of the various parts of computer hardware: i.e. the processor and memory, peripheral devices

2.1 Introduction

2.1.1 Basic Computer Organization

Computer organization refers to the operational units and their interconnections that realize the architectural specifications. Examples of architectural attributes include the instruction set, the number of bits used to represent various data types (e.g., numbers, characters), I/O mechanisms, and techniques for addressing memory. Organizational attributes include those hardware details transparent to the programmer, such as control signals; interfaces between the computer and peripherals; and the memory technology used.

Historically, and still today, the distinction between architecture and organization has been an important one. Many computer manufacturers offer a family of computer models, all with the same architecture but with differences in organization. Consequently, the different models in the family have different price and performance characteristics. Furthermore, a particular architecture may span many years and encompass a number of different computer models, its organization changing with changing technology. A prominent example of both these phenomena is the IBM System/370 architecture. This architecture was first introduced in 1970 and included a number of models. The customer with modest requirements could buy a cheaper, slower model and, if demand increased, later upgrade to a more expensive, faster model without having to abandon software that had already been developed. Over the years, IBM has introduced many new models with improved technology to replace older models, offering the customer greater speed, lower cost, or both. These newer models retained the same architecture so that the customer's software investment was protected. Remarkably, the System/370 architecture, with a few enhancements, has survived to this day as the architecture of IBM's mainframe product line.

In a class of computers called microcomputers, the relationship between architecture and organization is very close. Changes in technology not only influence organization but also result in the introduction of more powerful and more complex architectures. Generally, there is less of a requirement for generation-to-generation compatibility for these smaller machines. Thus, there is more interplay between organizational and architectural design decisions.

2.1.2 Structure and function of a computer system

A computer is a complex system; contemporary computers contain millions of elementary electronic components. How, then, can one clearly describe them? The key is to recognize the hierarchical nature of most complex systems, including the computer [SIMO96]. A hierarchical system is a set of interrelated subsystems, each of the latter, in turn, hierarchical in structure until we reach some lowest level of elementary subsystem. The hierarchical nature of complex systems is essential to both their design and their description. The designer need only deal with a particular level of the system at a time. At each level, the system consists of a set of components and their interrelationships.

The behavior at each level depends only on a simplified, abstracted characterization of the system at the next lower level.

2.1.3 Computer architecture

By definition, computer architecture refers to the logical components that have influence how computer programs will execute. These are attributes of a system visible to a programmer or, put another way, those attributes that have a direct impact on the logical execution of a program. For example, it is an architectural design issue whether a computer will have a multiply instruction. It is an organizational issue whether that instruction will be implemented by a special multiply unit or by a mechanism that makes repeated use of the add unit of the system. The organizational decision may be based on the anticipated frequency of use of the multiply instruction, the relative speed of the two approaches, and the cost and physical size of a special multiply unit.

The processor, also called the central processing unit (CPU), interprets and carries out the basic instructions that operate a computer. The processor significantly impacts overall computing power and manages most of a computer's operations. On larger computers, such as mainframes and supercomputers, the various functions performed by the processor extend over many separate chips and often multiple circuit boards. On a personal computer, all functions of the processor usually are on a single chip. Some computer and chip manufacturers use the term microprocessor to refer to a personal computer processor chip.

Most processor chip manufacturers now offer multi-core processors. A processor core, or simply core, contains the circuitry necessary to execute instructions. The operating system views each processor core as a separate processor. A multi-core processor is a single chip with two or more separate processor cores. Two common multi-core processors are dual-core and quad-core. A dual-core processor is a chip that contains two separate processor cores. Similarly, a quad-core processor is a chip with four separate processor cores.

Each processor core on a multi-core processor generally runs at a slower clock speed than a single-core processor, but multi-core processors typically increase overall performance. For example, although a dual-core processor does not double the processing speed of a single-core processor, it can approach those speeds. The performance increase is especially noticeable when users are running multiple programs simultaneously such as antivirus software, spyware remover, e-mail program, instant messaging, media player, disc burning software, and photo editing software. Multi-core processors also are more energy efficient than separate multiple processors, requiring lower levels of power consumption and emitting less heat in the system unit.

Processors contain a control unit and an arithmetic logic unit (ALU). These two components work together to perform processing operations.

Modern computer systems are designed based on the von Neumann computer architecture which was based on three key concepts:

1. Data and instructions are stored in a single read-write memory
2. The contents of this memory are addressable by location, without regard to the type of data contained there
3. Execution occurs in a sequential fashion (unless explicitly modified) from one instruction to the next

For a computer system to perform its functions, it requires a systematic set of instructions, also known as a program. The process of instructing a computer is referred to as programming. There are two approaches to programming:

- **Hardwired programming** - constructing a configuration of hardware logic components to perform a particular set of arithmetic and logic operations on a set of data. It is also known as firmware. A computer user cannot alter hardwired programs because they are built into the electronic components by the manufacturers. Usually, these are standard components that perform specific standard/routine functions and do not need to be reprogrammed. Typical examples of devices containing firmware are embedded systems (such as traffic lights, consumer appliances, and digital watches), computers, computer peripherals, mobile phones, and digital cameras. The firmware contained in these devices provides the control program for the device

- **Software** - a sequence of codes or instructions, each of which supply the necessary control signals to a general-purpose configuration of control and logic functions (which may themselves be hardwired programs). Software can be prepackaged industry standard applications (e.g. MS Excel, MS Word, MS PowerPoint, etc.) or user-defined depending on specific needs. The operating system of a computer also provides a facility (an Interpreter) where a computer user can give commands for a computer to perform a specific function

This design also requires two other components:

4. Input/Output Components – these provide a means to accept data and instructions in some form, and convert to an internal form of signals report results. They also enable the user to access retrieve data or processed information
5. Main memory - distinguished from external storage/peripherals. It is a place to temporarily store both: instructions, i.e. data which is interpreted as codes for generating control signals, and data – the data upon which computations are performed

The Interactions among these Computer Components is made possible through a series of memory registers that perform different functions. The following are some of the examples:

1. Memory Address Register –specifies the address for next read or write instruction
2. Memory Buffer Register – contains data to be written into or receives data read from memory
3. I/O address register - specifies a particular I/O device
4. I/O buffer register - used for exchange of data between an I/O module and CPU (or memory)

A Memory module is a set of locations with sequentially numbered addresses. Each holds a binary number that can be either an instruction or data

2.1.4 How a Computer Functions

For every instruction, a processor repeats a set of four basic operations, which comprise a machine cycle (Figure 4-5): (1) fetching, (2) decoding, (3) executing, and, if necessary, (4) storing. Fetching is the process of obtaining a program instruction or data item from memory.

The term decoding refers to the process of translating the instruction into signals the computer can execute. Executing is the process of carrying out the commands. Storing, in this

The Steps in a Machine Cycle context, means writing the result to memory (not to a storage medium). In some computers, the processor fetches, decodes, executes, and stores only one instruction at a time. In these computers, the processor waits until an instruction completes all four stages of the machine cycle (fetch, decode, execute, and store) before beginning work on the next instruction.

A computer program is a set of instructions the computer requires to perform the tasks or actions according to user needs. The processing required for a single instruction is called an instruction cycle which involves fetching an instruction from storage and executing it in the following 2 simple steps as illustrated in the figure below:

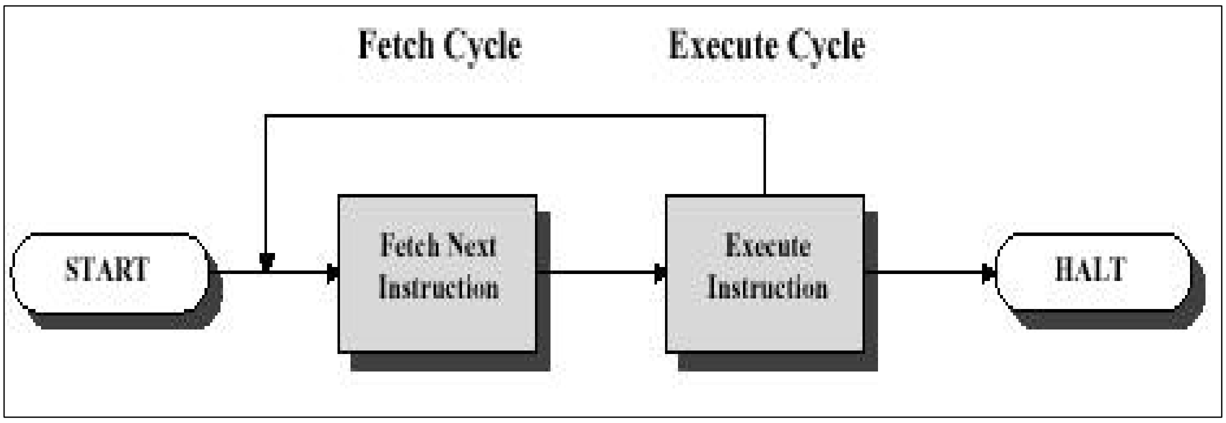


Table 2-1: Fetch - Execute Cycle

1. **The Fetch instruction** – the CPU reads an instruction from a location in memory
   a. The Program counter (PC) or register keeps track of which instruction executes next
   b. Normally, CPU increments PC after each fetch
   c. Fetched instruction is loaded into the Instruction Register (IR)

2. **The Execute instruction -** CPU executes the instruction
- May involve several operations
- May utilize previously changed state of CPU and (indirectly) other devices

The following are the General categories of instructions to be executed:

a. CPU-Memory: Data may be transferred from CPU to memory or vice-versa
b. CPU-IO: Data may be transferred between CPU and an I/O module
c. Data Processing: CPU may perform some arithmetic or logic operation on the data
d. Control: An instruction may specify that the sequence of execution be altered

There may be more complex instructions which include:

a. A combination of the categories listed above
b. An instruction to perform more than one reference to memory
c. A specific I/O operation instead of memory reference
d. A specific operation to be performed on a vector of numbers or a string of characters

An Expanded execution cycle would consist of the following:

a. Instruction Address Calculation (IAC) – it determines the address of the next instruction

b. Instruction Fetch (if)
c. Instruction Operation Decoding (iod) – it analyzes an operation to determine the operation type and the operands.
d. Operand Address Calculation (oac)
e. Operand Fetch (of)
f. Data Operation (do) - perform indicated op
g. Operand Store (os) - write result into memory or out to I/O

Interrupts are Mechanisms by which other modules may interrupt the normal processing of the CPU. The following are the classes of interrupts:

a. Program interrupts - as a result of program execution
b. Timer interrupts - generated by hardware timer
c. I/O interrupts - to signal completion of I/O or error
d. Hardware failure

The Fetch-Execute Instruction cycle with interrupts

Sometimes conditions require the processor to be interrupted from its normal processing activity. Some of the conditions are:

a) Power failure as detected by a sensor
b) Arithmetic conditions such as overflow and underflow
c) Illegal data or illegal instruction code
d) Errors in data transmission and storage
e) Software-generated interrupts (as intended by the user)
f) Normal completion of an asynchronous transfer

In each of these conditions, the processor must discontinue its processing activity, attend to the interrupting condition, and (if possible) resume the processing activity from where it had been when the interrupt occurred. In order for the processor to be able to resume normal processing after servicing the interrupt, it is essential to at least save the address of the instruction to be executed just before entering the interrupt service mode. In addition, contents of the accumulator and all other registers must be saved. Typically, when an interrupt is received, the processor completes the current instruction and jumps to an interrupt service routine. An interrupt service routine is a program preloaded into the machine memory that performs the following functions:

a) Disables further interrupts (temporarily)
b) Saves the processor status (all registers)
c) Enables further interrupts
d) Determines the cause of interrupt
e) Services the interrupt
f) Disables interrupts
g) Restores processor status
h) Enables further interrupts

i) Returns from interrupt

The processor disables further interrupts just long enough to save the status, since a proper return from interrupt service routine is not possible if the status is not completely saved. The processor status usually comprises the contents of all the registers, including the program counter and the program status word. ‘‘Servicing’’ the interrupt simply means taking care of the interrupt condition: in the case of an I/O interrupt, it corresponds to data transfer; in case of power failure, it is the saving o f registers and status for normal resumption of processing when the power is back; during an arithmetic condition, it is checking the previous operation or simply setting a flag to indicate the arithmetic error.

Once the interrupt service is complete, the processor status is restored. That is, all the registers are loaded with the values saved during step 2. Interrupt s are disabled during this restore period. This completes the interrupt service, and the processor returns to the norm al processing mode. Below is an illustration of how interrupts are handled:

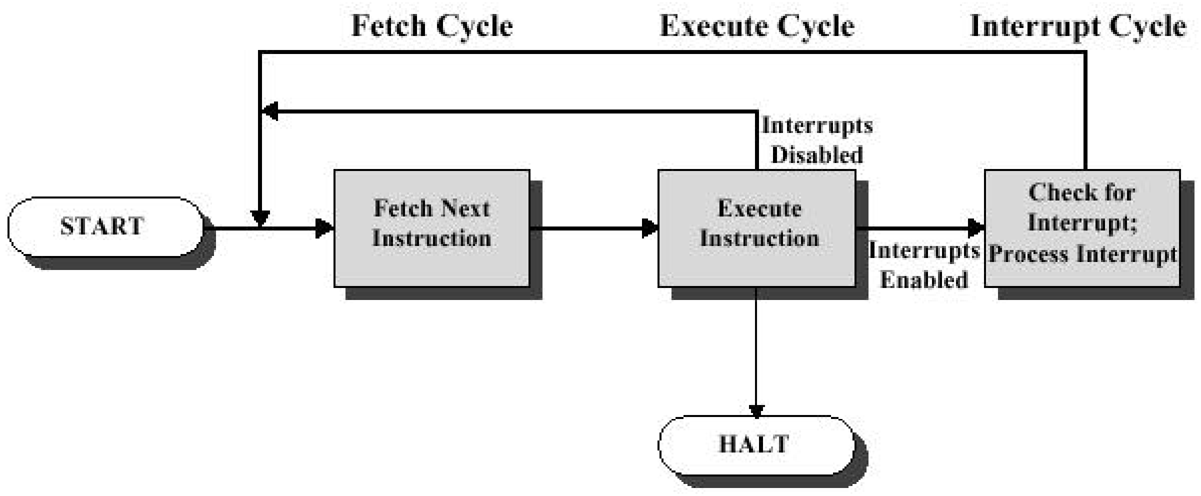


Table 2-2: Fetch - Execute cycle with Interrupts

When an interrupt signal is generated, the processor will do the following: Suspends execution of the current program and saves its context (such as PC and other registers). Then it will set the Program Counter (PC) to starting address of an interrupt handler routine

Multiple interrupts can be handled by disabling some or all interrupts. Disabled interrupts generally remain pending and are handled sequentially. They can also be handled by prioritizing interrupts, allowing a higher priority interrupt to interrupt one of lower priority

Physical Interrupts

a. Interrupts are represented as one or more lines in the system bus
b. One line: polling - when line goes high, CPU polls devices to determine which caused interrupt

c. Multiple lines: addressable interrupts - combination of lines indicates both interrupt and which device caused it. Ex. 386 based architectures use 4 bit interrupts, allowing IRQ's 0-15 (with an extra line to signal pending)

Activity:

Think of situations when a computer system needs to be interrupted. List down some three such circumstances.

2.1.5 Bus Interconnection

The CPU has to be able to send various data values, instructions, and information to all the devices and components inside your computer as well as the different peripherals and devices attached. If you look at the bottom of a motherboard you'll see a whole network of lines or electronic pathways that join the different components together. These electronic pathways are nothing more than tiny wires that carry information, data and different signals throughout the computer between the different components. This network of wires or electronic pathways is called the 'Bus'.

A computer's bus can be divided into two different types: Internal bus and External bus. The Internal Bus connects the different components inside the case: The CPU, system memory, and all other components on the motherboard. It's also referred to as the System Bus. The External Bus connects the different external devices, peripherals, expansion slots, I/O ports and drive connections to the rest of the computer. In other words, the External Bus allows various devices to be added to the computer. It allows for the expansion of the computer's capabilities. It is generally slower than the system bus. Another name for the External Bus, is the Expansion Bus.

2.1.5.1 Bus Structure

A system bus consists, typically, of from about 50 to hundreds of separate lines. Each line is assigned a particular meaning or function. Although there are many different bus designs, on any bus the lines can be classified into three functional groups (see Figure below): data, address, and control lines. In addition, there may be power distribution lines that supply power to the attached modules.

Data lines (data bus) - move data between system modules. The Width is a key factor in determining overall system performance. The **data lines** provide a path for moving data among system modules. These lines, collectively, are called the data bus. The data bus may consist of 32, 64, 128, or even more separate lines, the number of lines being referred to as the width of the data bus. Because each line can carry only 1 bit at a time, the number of lines determines how many bits can be transferred at a time. The width of the data bus is a key factor in determining overall system performance. For example, if the data bus is 32 bits wide and each instruction is 64 bits long, then the processor must access the memory module twice during each instruction cycle.

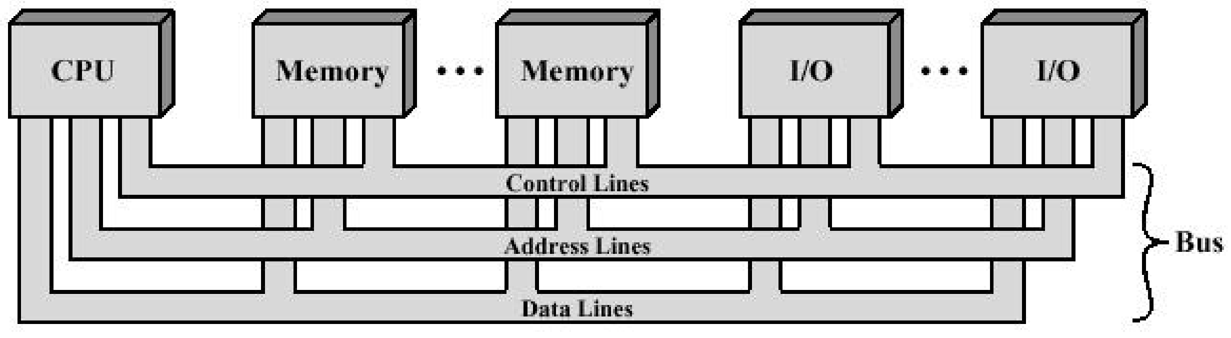


Table 2-3: a System Bus

Address lines - designate source or destination of data on the data bus. The Width determines the maximum possible memory capacity of the system (may be a multiple of width). Typically: high-order bits select a particular module while lower-order bits select a memory location or I/O port within the module.

The **address lines** are used to designate the source or destination of the data on the data bus. For example, if the processor wishes to read a word (8, 16, or 32 bits) of data from memory, it puts the address of the desired word on the address lines. Clearly, the width of the address bus determines the maximum possible memory capacity of the system. Furthermore, the address lines are generally also used to address I/O ports. Typically, the higher-order bits are used to select a particular module on the bus, and the lower-order bits select a memory location or I/O port within the module. For example, on an 8-bit address bus, address 01111111 and below might reference locations in a memory module (module 0) with 128 words of memory, and address 10000000 and above refer to devices attached to an I/O module (module 1).

Control lines – they control access to and use of the data and address lines. The **control lines** are used to control the access to and the use of the data and address lines. Because the data and address lines are shared by all components, there must be a means of controlling their use. Control signals transmit both command and timing information among system modules. Timing signals indicate the validity of data and address information. Command signals specify operations to be performed. Typical control lines include:

- Memory write: Causes data on the bus to be written into the addressed location
- Memory read: Causes data from the addressed location to be placed on the bus
- I/O write: Causes data on the bus to be output to the addressed I/O port
- I/O read: Causes data from the addressed I/O port to be placed on the bus
- Transfer ACK: Indicates that data have been accepted from or placed on the bus
- Bus request: Indicates that a module needs to gain control of the bus

- Bus grant: Indicates that a requesting module has been granted control of the bus
- Interrupt request: Indicates that an interrupt is pending
- Interrupt ACK: Acknowledges that the pending interrupt has been recognized
- Clock: Is used to synchronize operations
- Reset: Initializes all modules

2.1.5.2 How system bus operates

The operation of the bus is as follows. If one module wishes to send data to another, it must do two things: (1) obtain the use of the bus, and (2) transfer data via the bus. If one module wishes to request data from another module, it must (1) obtain the use of the bus, and (2) transfer a request to the other module over the appropriate control and address lines. It must then wait for that second module to send the data.

Physically, the system bus is actually a number of parallel electrical conductors. In the classic bus arrangement, these conductors are metal lines etched in a card or board (printed circuit board). The bus extends across all of the system components, each of which taps into some or all of the bus lines. If one system module wishes to send data to another, it must: Obtain use of the bus and then Transfer data via the bus. If one module wishes to request data from another, it must: Obtain use of the bus, Transfer a request to the other module over control and address lines and then Wait for second module to send data

Typical physical arrangement of a system bus

a. A number of parallel electrical conductors
b. Each system component (usually on one or more boards) taps into some or all of the bus lines (usually with a slotted connector)
c. System can be expanded by adding more boards
d. A bad component can be replaced by replacing the board where it resides

2.2 Introduction to the Central Processing Unit

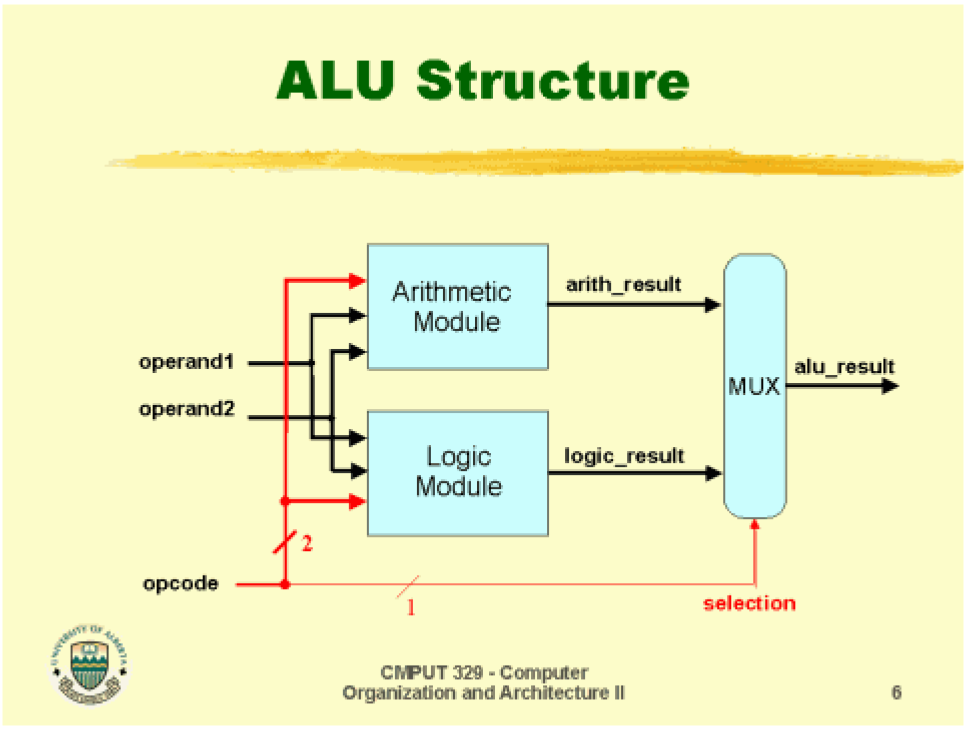
The four major hardware blocks of a general purpose computer system are:

a. its memory unit (MU)
b. Arithmetic and logic unit (ALU) -The ALU processes the data taken from the memory unit (or the ALU) and stores the processed data back in the memory unit (or the ALU).
c. Control unit (CU). Programs reside in the memory unit. The control unit coordinates the activities of the other three units. It retrieves instructions from programs resident in the MU, decodes these instructions, and directs the ALU to perform corresponding processing steps. It also oversees I/O operations.
d. Input/output unit (IOU) - these I/O devices input and output data into and out of the memory unit. In some systems, I/O devices send and receive data into and from the ALU rather than the MU.

In modern computer systems, the ALU and control unit have been merged in to what is now known as the Central Processing Unit (CPU). The CPU is equipped with some limited amount of fast memory for temporary storage of instructions and data that is currently being processed.

2.2.1 Arithmetic and Logic Unit

The ALU of a computer system is the hardware which performs all arithmetic and logical operations. For instance, if the instruction set implies that the ALU must perform addition of two numbers, it will compute the 2s complement of the number, and shift the contents of the accumulator either right or left by 1 bit. Additionally, the ALU must directly transfer either of its inputs to its output to support data transfer operations such as IR MB R and MA R IR.



We will assume that the control unit of the machine provides the appropriate control signals to enable the ALU to perform one of these operations. You can look at the ALU as comprising many subcomponents for each specific task that it is required to perform. Some of these tasks and their appropriate subcomponents are:

2.2.1.1 Addition and subtraction

These two tasks are performed by constructs of logic gates, such as half adders and full adders. While they may be termed 'adders', with the aid of they can also perform subtraction via use of inverters and 'two's complement' arithmetic.

The topic of logic gates is too expansive and detailed to be covered in full here. Many resources exist on the internet and elsewhere relating to this topic, however, so it is recommended that you read further into the areas outlined above to aid with your learning.

Boolean Addition

When adding two numbers, if the sum of the digits in a given position equals or exceeds the modulus, then a *carry* is propagated. For example, in Boolean addition, if two ones are added, the sum is obviously two (base 10), which exceeds the modulus of 2 for Boolean numbers ($\mathbf{B} = \mathbf{Z}_2 = \{0,1\}$, the integers modulo 2). Thus, we record a zero for the sum and propagate a carry valued at one into the next more significant digit, as shown in Figure 3.1.

• $5_{ten} + 6_{ten}$

```
  0000 0000 0000 0000 0000 0000 0000 0101  (5ten)
+ 0000 0000 0000 0000 0000 0000 0000 0110  (6ten)
= 0000 0000 0000 0000 0000 0000 0000 1011  (11ten)
```

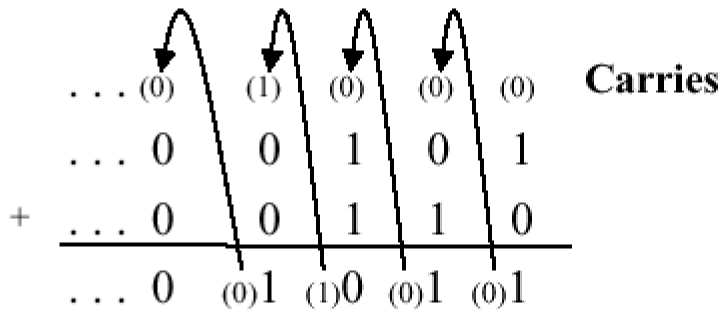


Figure 3.1. Example of Boolean addition with carry propagation, adapted from [Maf01].

Binary subtraction by 2s complement

When subtracting two numbers, two alternatives present themselves. First, one can formulate a subtraction algorithm, which is distinct from addition. Second, one can negate the subtrahend (i.e., in *a* - *b*, the subtrahend is *b*) then perform addition. Since we already know how to perform addition as well as twos complement negation, the second alternative is more practical. The Figure below illustrates both processes, using the decimal subtraction 12 - 5 = 7 as an example.

• $12_{ten} - 5_{ten}$

```
  0000 0000 0000 0000 0000 0000 0000 1100  (12ten)
- 0000 0000 0000 0000 0000 0000 0000 0101  ( 5ten)
= 0000 0000 0000 0000 0000 0000 0000 0111  ( 7ten)
```

• $12_{ten} - 5_{ten} = 12_{ten} + (- 5_{ten})$

```
  0000 0000 0000 0000 0000 0000 0000 1100  (12ten)
+ 1111 1111 1111 1111 1111 1111 1111 1011  ( -5ten)
= 0000 0000 0000 0000 0000 0000 0000 0111  ( 7ten)
```

Figure 3.2. Example of Boolean subtraction using (a) unsigned binary representation, and (b) addition with twos complement negation - adapted from [Maf01].

Just as we have a carry in addition, the subtraction of Boolean numbers uses a *borrow*. For example, in Figure 3.2a, in the first (least significant) digit position, the difference 0 - 1 in the one's place is realized by borrowing a one from the two's place (next more significant digit). The borrow is propagated upward (toward the most significant digit) until it is zeroed (i.e., until we encounter a difference of 1 - 0).

Example 2:

The decimal subtraction 29 - 7 = 22 is the same as adding (29) + (-7) = 22

1. Convert the number to be subtracted to its two's complement:

00000111	**(decimal 7)**
11111000	**(one's complement)**
+ <u>00000001</u>	**(add 1)**
11111001	**(two's complement)**

2. 11111001 now represents -7.

3. Add

29	**00011101**
+- <u>7</u>	**<u>11111001</u>**
22	**(1)00010110**

4. **Note that the final carry 1 is ignored.**

2.2.1.2 Multiplication and division

In most modern processors, the multiplication and division of integer values is handled by specific floating-point hardware within the CPU. Earlier processors used either additional chips known as maths co-processors, or used a completely different method to perform the task.

2.2.2 Logical tests

Further logic gates are used within the ALU to perform a number of different logical tests, including seeing if an operation produces a result of zero. Most of these logical tests are used to then change the values stored in the flag register, so that they may be checked later by seperate operations or instructions. Others produce a result which is then stored, and used later in further processing.

2.2.2.1 Logical Operations

Logical operations apply to fields of bits within a 32-bit word, such as bytes or bit fields (in C, as discussed in the next paragraph). These operations include shift-left and shift-right operations (`sll` and `srl`), as well as bitwise *and*, *or* (`and`, `andi`, `or`, `ori`). As we saw in Section 2, bitwise operations treat an operand as a vector of bits and operate on each bit position.

C bit fields are used, for example, in programming communications hardware, where manipulation of a bit stream is required. In Figure 3.5 is presented C code for an example communications routine, where a structure called `receiver` is formed from an 8-bit field called *receivedByte* and two one-bit fields called *ready* and *enable*. The C routine sets `receiver.ready` to 0 and `receiver.enable` to 1.

```
struct {
    unsigned int ready:          1;
    unsigned int enable:         1;
    unsigned int receivedByte:   8;
} receiver;
int data = receiver.receivedByte;
receiver.ready = 0;
receiver.enable = 1;
```

```
#$s0: data;  $s1: receiver

sll     $s0, $s1, 22
srl     $s0, $s0, 24
andi    $s1, $s1, 0xfffe
ori     $s1, $s1, 0x0002
```

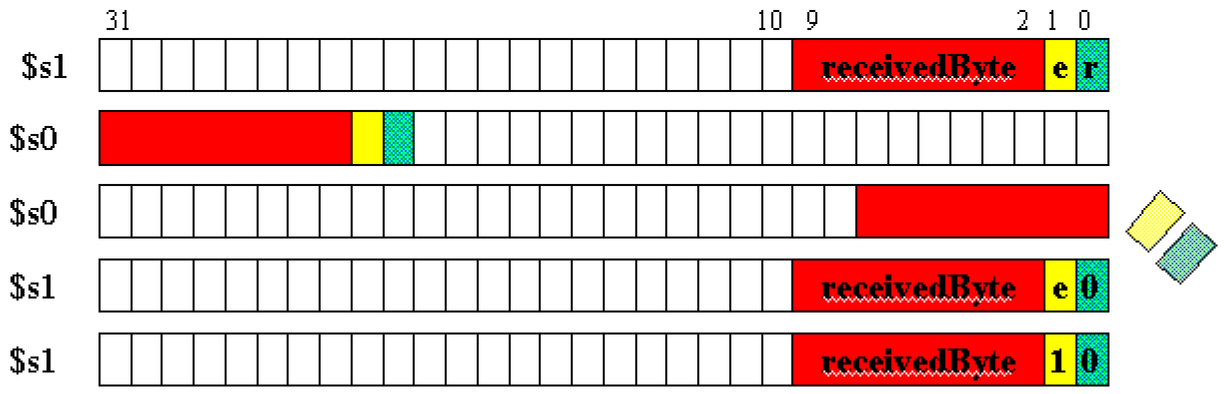


Figure 3.5. Example of C bit field use in MIPS, adapted from [Maf01].

Note how the MIPS code implements the functionality of the C code, where the state of the registers $s0 and $s1 is illustrated in the five lines of diagrammed register contents below the code. In particular, the initial register state is shown in the first two lines. The `sll` instruction loads the contents of `$s1` (the receiver) into `$s0` (the data register), and the result of this is shown on the second line of the register contents. Next, the `srl` instruction left-shifts `$s0` 24 bits, thereby discarding the *enable* and *ready* field information, leaving just the received byte. To signal the receiver that the data transfer is completed, the `andi` and `ori` instructions are used to set the enable and ready bits in `$s1`, which corresponds to the *receiver*. The data in `$s0` has already been received and put in a register, so there is no need for its further manipulation.

2.2.2.2 Comparison

Comparison operations compare values in order to determine such things as whether one number is greater than, less than or equal to another. These operations can be performed by subtraction of one of the numbers from the other, and as such can be handled by the aforementioned logic gates. However, it is not strictly necessary for the result of the calculation to be stored in this instance.. the amount by which the values differ is not required. Instead, the appropriate status flags in the flag register are set and checked to detemine the result of the operation.

2.2.2.3 Bit shifting

Shifting operations move bits left or right within a word, with different operations filling the gaps created in different ways. This is accomplished via the use of a shift register, which uses pulses from the clock within the control unit to trigger a chain reaction of movement across the bits that make up the word. Again, this is a quite complicated logical procedure, and further reading may aid your understanding.

2.2.3 Types of Control Units

As mentioned earlier, the function of the control unit is to generate the control signals in the appropriate sequence to bring about the instruction cycle that corresponds to each instruction in the program. In a simple computer system, an instruction cycle consists of three phases. Each phase in the instruction cycle is composed of a sequence of micro-operations. A micro-operation is one of the following:

1. A simple register transfer operation, to transfer contents of one register to another register
2. A complex register transfer involving ALU, such as the transfer of the complement of the contents of a register, the sum of the contents of two registers, etc. to the destination register
3. A memory read or write operation

Thus, a machine instruction is composed of a sequence of micro-operations (i.e., a register transfer sequence). We will use the terms register transfer and micro-operation interchange ably.

There are two popular implementation methods for control units (CU):

1. Hardwired Control Unit (HCU): The output (i.e., control signals) of the CU is generated by the logic circuitry built of gates and flip-flops.
2. Micro-programmed Control Unit (MCU): The sequence of micro-operations corresponding to each machine instruction, are stored in a read-only memory called control ROM (CROM). The sequence of micro-operations is called the microprogram, and the microprogram consists of micro-instructions.

A micro instruction corresponds to one or more micro-operations, depending on the CROM storage format. The control signals are generated by decoding the micro-instructions.

2.2.4 Processor Storage

Flip-flops are electronic logic gates that are used in storing binary information. An array of flip-flops put together form a register. A register is used either to store data temporarily or to manipulate data stored in it using the logic circuitry around it. The memory subsystem of a digital computer is functionally a set of such registers where data and programs are stored. The instructions from the programs stored in memory are retrieved by the control unit of the machine (digital computer system) and are decoded to perform the appropriate operation on the data stored either in memory or in a set of registers in the processing unit.

For optimum operation of the machine, it is required that programs and data be accessible by control and processing units as quickly as possible. The main memory (primary memory) allows such a fast access. This fast-access requirement adds a considerable amount of hardware to the main memory and thus makes it expensive.

To reduce memory cost, data and programs not immediately needed by the machine are normally stored in a less-expensive secondary memory subsystem (ASC does not have a secondary memory). They are brought into the main memory when the processing unit needs them. The larger the main memory, the more information it can store and hence the faster the processing, since most of the information required is immediately available. But because main-memory hardware is expensive, a speed-cost tradeoff is needed to decide on the amounts of main and secondary storage needed.

Activity: Consider the kind of computer you are currently using. Do you see the von Neumann design inherent in its design? Discuss with a colleague and list in a table this comparison between the von Neumann design and your computer.

A processor includes both user-visible registers and control/status registers. The former may be referenced, implicitly or explicitly, in machine instructions. User-visible registers may be general purpose or have a special use, such as fixed-point or floating-point numbers, addresses, indexes, and segment pointers. Control and status registers are used to control the operation of the processor. One obvious example is the program counter. Another important example is a program status word (PSW) that contains a variety of status and condition bits. These include bits to reflect the result of the most recent arithmetic operation, interrupt enable bits, and an indicator of whether the processor is executing in supervisor or user mode.

To understand the organization of the processor, let us consider the requirements placed on the processor, the things that it must do:

- Fetch instruction: The processor reads an instruction from memory (register, cache, main memory).
- Interpret instruction: The instruction is decoded to determine what action is required.
- Fetch data: The execution of an instruction may require reading data from memory or an I/O module.

- Process data: The execution of an instruction may require performing some arithmetic or logical operation on data.
- Write data: The results of an execution may require writing data to memory or an I/O module.

To do these things, it should be clear that the processor needs to store some data temporarily. It must remember the location of the last instruction so that it can know where to get the next instruction. It needs to store instructions and data temporarily while an instruction is being executed. In other words, the processor needs a small internal memory.

Figure 12.1 is a simplified view of a processor, indicating its connection to the rest of the system via the system bus. A similar interface would be needed for any of the interconnection structures described in Chapter 3.The reader will recall that the major components of the processor are an arithmetic and logic unit (ALU) and a control unit (CU).The ALU does the actual computation or processing of data. The control unit controls the movement of data and instructions into and out of the processor and controls the operation of the ALU. In addition, the figure shows a minimal internal memory, consisting of a set of storage locations, called registers.

Figure 12.2 is a slightly more detailed view of the processor. The data transfer and logic control paths are indicated, including an element labeled internal processor bus. This element is needed to transfer data between the various registers and the ALU because the ALU in fact operates only on data in the internal processor memory. The figure also shows typical basic elements of the ALU. Note the similarity between the internal structure of the computer as a whole and the internal structure of the processor. In both cases, there is a small collection of major elements (computer: processor, I/O, memory; processor: control unit, ALU, registers) connected by data paths.

2.2.5 Register organization

A computer system employs a memory hierarchy. At higher levels of the hierarchy, memory is faster, smaller, and more expensive (per bit).Within the processor, there is a set of registers that function as a level of memory above main memory and cache in the hierarchy. The registers in the processor perform two roles:

• User-visible registers: Enable the machine- or assembly language programmer to minimize main memory references by optimizing use of registers.

• Control and status registers: Used by the control unit to control the operation of the processor and by privileged, operating system programs to control the execution of programs.

There is not a clean separation of registers into these two categories. For example, on some machines the program counter is user visible (e.g., x86), but on many it is not. For purposes of the following discussion, however, we will use these categories.

2.2.5.1 User-Visible Registers

A user-visible register is one that may be referenced by means of the machine language that the processor executes. We can characterize these in the following categories:

- General purpose
- Data
- Address
- Condition codes

General-purpose registers can be assigned to a variety of functions by the programmer. Sometimes their use within the instruction set is orthogonal to the operation. That is, any general-purpose register can contain the operand for any opcode. This provides true general-purpose register use. Often, however, there are restrictions. For example, there may be dedicated registers for floating-point and stack operations. In some cases, general-purpose registers can be used for addressing functions (e.g., register indirect, displacement). In other cases, there is a partial or clean separation between data registers and address registers.

Data registers may be used only to hold data and cannot be employed in the calculation of an operand address.

Address registers may themselves be somewhat general purpose, or they may be devoted to a particular addressing mode. Examples include the following:

- Segment pointers: In a machine with segmented addressing (see Section 8.3), a segment register holds the address of the base of the segment. There may be multiple registers: for example, one for the operating system and one for the current process.
- Index registers: These are used for indexed addressing and may be auto-indexed.
- Stack pointer: If there is user-visible stack addressing, then typically there is a dedicated register that points to the top of the stack. This allows implicit addressing; that is, push, pop, and other stack instructions need not contain an explicit stack operand.

There are several design issues to be addressed here. An important issue is whether to use completely general-purpose registers or to specialize their use. We have already touched on this issue in the preceding chapter because it affects instruction set design. With the use of specialized registers, it can generally be implicit in the opcode which type of register a certain operand specifier refers to. The operand specifier must only identify one of a set of specialized registers rather than one out of all the registers, thus saving bits. On the other hand, this specialization limits the programmer's flexibility.

Another design issue is the number of registers, either general purpose or data plus address, to be provided. Again, this affects instruction set design because more registers require more operand specifier bits. Somewhere between 8 and 32 registers appears optimum. Fewer registers result in more memory references; more registers do not noticeably reduce memory references. However, a new approach, which finds advantage in the use of hundreds of registers, is exhibited in some RISC systems.

Finally, there is the issue of register length. Registers that must hold addresses obviously must be at least long enough to hold the largest address. Data registers should be able to hold values of most data types. Some machines allow two contiguous registers to be used as one for holding double-length values.

A final category of registers, which is at least partially visible to the user, holds condition codes (also referred to as flags). **Condition codes** are bits set by the processor hardware as the result of operations. For example, an arithmetic operation may produce a positive, negative, zero, or overflow result. In addition to the result itself being stored in a register or memory, a condition code is also set. The code may subsequently be tested as part of a conditional branch operation.

Condition code bits are collected into one or more registers. Usually, they form part of a control register. Generally, machine instructions allow these bits to be read by implicit reference, but the programmer cannot alter them. Many processors, including those based on the IA-64 architecture and the MIPS processors, do not use condition codes at all. Rather, conditional branch instructions specify a comparison to be made and act on the result of the comparison, without storing a condition code. Table 12.1, lists key advantages and disadvantages of condition codes.

Table 12.1 Condition Codes

Advantages	**Disadvantages**
1. Because condition codes are set by normal arithmetic and data movement instructions, they should reduce the number of COMPARE and TEST instructions needed.	**1.** Condition codes add complexity, both to the hardware and software. Condition code bits are often modified in different ways by different instructions, making life more difficult for both the micro-programmer and compiler writer.
2. Conditional instructions, such as BRANCH are simplified relative to composite instructions, such as TEST AND BRANCH.	**2.** Condition codes are irregular; they are typically not part of the main data path, so they require extra hardware connections.
3. Condition codes facilitate multiway branches. For example, a TEST instruction can be followed by two branches, one on less than or equal to zero and one on greater than zero.	**3.** Often condition code machines must add special non-condition-code instructions for special situations anyway, such as bit checking, loop control, and atomic semaphore operations.
	4. In a pipelined implementation, condition codes require special synchronization to avoid conflicts.

In some machines, a subroutine call will result in the automatic saving of all user-visible registers, to be restored on return. The processor performs the saving and restoring as part of the execution of call and return instructions. This allows each subroutine to use the user-visible registers independently. On other machines, it is the responsibility of the programmer to save the contents of the relevant uservisible registers prior to a subroutine call, by including instructions for this purpose in the program.

2.2.5.2 Control and Status Registers

There are a variety of processor registers that are employed to control the operation of the processor. Most of these, on most machines, are not visible to the user. Some of them may be visible to machine instructions executed in a control or operating system mode. Of course, different machines will have different register organizations and use different terminology. We list here a reasonably complete list of register types, with a brief description. Four registers are essential to instruction execution:

- Program counter (PC): Contains the address of an instruction to be fetched
- Instruction register (IR): Contains the instruction most recently fetched
- Memory address register (MAR): Contains the address of a location in memory
- Memory buffer register (MBR): Contains a word of data to be written to memory or the word most recently read

Not all processors have internal registers designated as MAR and MBR, but some equivalent buffering mechanism is needed whereby the bits to be transferred to the system bus are staged and the bits to be read from the data bus are temporarily stored.

Typically, the processor updates the PC after each instruction fetch so that the PC always points to the next instruction to be executed. A branch or skip instruction will also modify the contents of the PC. The fetched instruction is loaded into an IR, where the opcode and operand specifiers are analyzed. Data are exchanged with memory using the MAR and MBR. In a bus-organized system, the MAR connects directly to the address bus, and the MBR connects directly to the data bus. Uservisible registers, in turn, exchange data with the MBR.

The four registers just mentioned are used for the movement of data between the processor and memory. Within the processor, data must be presented to the ALU for processing. The ALU may have direct access to the MBR and user-visible registers. Alternatively, there may be additional buffering registers at the boundary to the ALU; these registers serve as input and output registers for the ALU and exchange data with the MBR and user-visible registers.

Many processor designs include a register or set of registers, often known as the program status word (PSW), that contain status information. The PSW typically contains condition codes plus other status information. Common fields or flags include the following:

- Sign: Contains the sign bit of the result of the last arithmetic operation.
- Zero: Set when the result is 0.
- Carry: Set if an operation resulted in a carry (addition) into or borrow (subtraction) out of a high-order bit. Used for multiword arithmetic operations.
- Equal: Set if a logical compare result is equality.
- Overflow: Used to indicate arithmetic overflow.
- Interrupt Enable/Disable: Used to enable or disable interrupts.
- Supervisor: Indicates whether the processor is executing in supervisor or user mode. Certain privileged instructions can be executed only in supervisor mode, and certain areas of memory can be accessed only in supervisor mode.

A number of other registers related to status and control might be found in a particular processor design. There may be a pointer to a block of memory containing additional status information (e.g., process control blocks). In machines using vectored interrupts, an interrupt vector register may be provided. If a stack is used to implement certain functions (e.g., subroutine call), then a system stack pointer is needed. A page table pointer is used with a virtual memory system. Finally, registers may be used in the control of I/O operations.

A number of factors go into the design of the control and status register organization. One key issue is operating system support. Certain types of control information are of specific utility to the operating system. If the processor designer has a functional understanding of the operating system to be used, then the register organization can to some extent be tailored to the operating system.

Another key design decision is the allocation of control information between registers and memory. It is common to dedicate the first (lowest) few hundred or thousand words of memory for control purposes. The designer must decide how much control information should be in registers and how much in memory. The usual trade-off of cost versus speed arises.

2.2.6 Instruction pipelining

As computer systems evolve, greater performance can be achieved by taking advantage of improvements in technology, such as faster circuitry. In addition, organizational enhancements to the processor can improve performance. We have already seen some examples of this, such as the use of multiple registers rather than a single accumulator, and the use of a cache memory. Another organizational approach, which is quite common, is instruction pipelining.

2.2.6.1 Pipelining Strategy

Instruction pipelining is similar to the use of an assembly line in a manufacturing plant. An assembly line takes advantage of the fact that a product goes through various stages of production. By laying the production process out in an assembly line, products at various stages can be worked on simultaneously. This process is also referred to as pipelining, because, as in a pipeline, new inputs are accepted at one end before previously accepted inputs appear as outputs at the other end.

To apply this concept to instruction execution, we must recognize that, in fact, an instruction has a number of stages. Figures 12.5, for example, breaks the instruction cycle up into 10 tasks, which occur in sequence. Clearly, there should be some opportunity for pipelining.

Figure 12.9: Two-stage instruction pipelining

As a simple approach, consider subdividing instruction processing into two stages: fetch instruction and execute instruction. There are times during the execution of an instruction when main memory is not being accessed. This time could be used to fetch the next instruction in parallel with the execution of the current one. Figure 12.9a depicts this approach. The pipeline has two independent stages. The first stage fetches an instruction and buffers it. When the second stage is free, the first stage passes it the buffered instruction. While the second stage is executing the instruction, the first stage takes advantage of any unused memory cycles to fetch and buffer the next instruction. This is called instruction prefetch or fetch overlap. Note that this approach, which involves instruction buffering, requires more registers. In general, pipelining requires registers to store data between stages.

It should be clear that this process will speed up instruction execution. If the fetch and execute stages were of equal duration, the instruction cycle time would be halved. However, if we look more closely at this pipeline (Figure 12.9b), we will see that this doubling of execution rate is unlikely for two reasons:

1. The execution time will generally be longer than the fetch time. Execution will involve reading and storing operands and the performance of some operation. Thus, the fetch stage may have to wait for some time before it can empty its buffer.
2. A conditional branch instruction makes the address of the next instruction to be fetched unknown. Thus, the fetch stage must wait until it receives the next instruction address from the execute stage. The execute stage may then have to wait while the next instruction is fetched.

Guessing can reduce the time loss from the second reason. A simple rule is the following: When a conditional branch instruction is passed on from the fetch to the execute stage, the fetch stage fetches the next instruction in memory after the branch instruction. Then, if the branch is not taken, no time is lost. If the branch is taken, the fetched instruction must be discarded and a new instruction fetched.

3 Week 4/5: Memory systems

- RAM and disks
- Hierarchy of memories
- Modern memory technologies: DIMMs, SIMMs
- Modern memory technologies: SRAMs, DRAMs
- Advanced DRAM

3.1 Computer Memory Basics

Technically, the term memory is any form of electronic storage. However, most often it is used to identify fast, temporary forms of storage. If your computer's CPU had to constantly access the hard drive to retrieve every piece of data it needs, it would operate very slowly. When the information is kept in memory, the CPU can access it much more quickly. Most forms of memory are intended to store data temporarily.

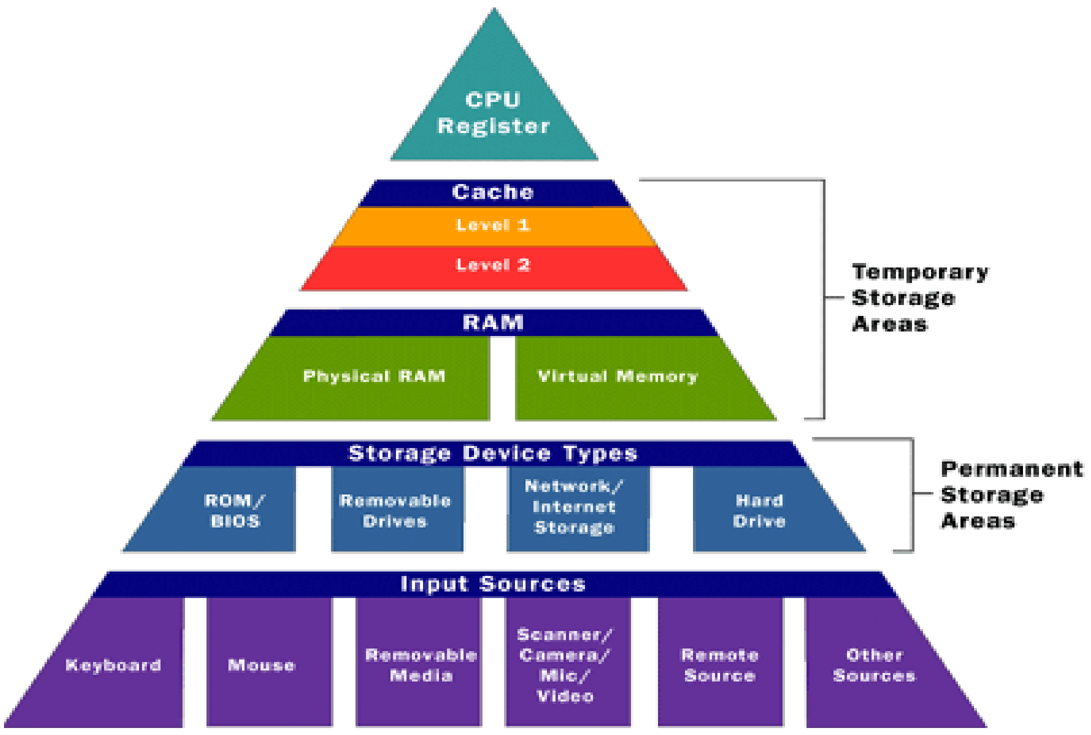


Figure 3:1 Computer Memory Hierarchy

As you can see in the diagram above, the CPU accesses memory according to a distinct hierarchy. Whether it comes from permanent storage (e.g. hard drive) or input (e.g. keyboard), most data goes in **random access memory** (RAM) first. The CPU then stores pieces of data it will need to access, often in a **cache**, and maintains certain special instructions in the **register**. We'll talk about cache and registers later.

All of the components in your computer, such as the CPU, the hard drive and the operating system, work together as a team, and memory is one of the most essential parts of this team. From the moment you turn your computer on until the time you shut it down, your CPU is constantly using memory.

Consider the following scenario when you turn on your computer system:

1. You turn the computer on.
2. The computer loads data from **read-only memory** (ROM) and performs a **power-on self-test** (POST) to make sure all the major components are functioning properly. As part of this test, the **memory controller** checks all of the memory addresses with a quick **read/write** operation to ensure that there are no errors in the memory chips. Read/write means that data is written to a bit and then read from that bit.
3. The computer loads the **basic input/output system** (BIOS) from ROM. The BIOS provides the most basic information about storage devices, boot sequence, security, **Plug and Play** (auto device recognition) capability and a few other items.
4. The computer loads the **operating system** (OS) from the hard drive into the system's RAM. Generally, the critical parts of the operating system are maintained in RAM as long as the computer is on. This allows the CPU to have immediate access to the operating system, which enhances the performance and functionality of the overall system.
5. When you open an **application**, it is loaded into RAM. To conserve RAM usage, many applications load only the essential parts of the program initially and then load other pieces as needed.
6. After an application is loaded, any **files** that are opened for use in that application are loaded into RAM.
7. When you **save** a file and **close** the application, the file is written to the specified storage device, and then it and the application are purged from RAM.

In the list above, every time something is loaded or opened, it is placed into RAM. This simply means that it has been put in the computer's **temporary storage area** so that the CPU can access that information more easily. The CPU requests the data it needs from RAM, processes it and writes new data back to RAM in a **continuous cycle**. In most computers, this shuffling of data between the CPU and RAM happens millions of times every second. When an application is closed, it and any accompanying files are usually **purged** (deleted) from RAM to make room for new data. If the changed files are not saved to a permanent storage device before being purged, they are lost. Below is an example of different types of modern memory modules

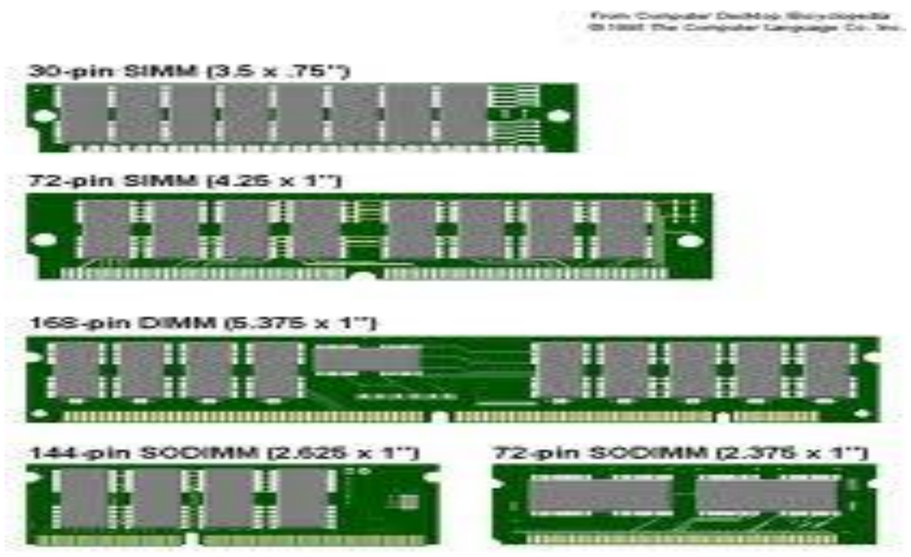

From Computer Desktop Encyclopedia
©1998 The Computer Language Co. Inc.
30-pin SIMM (3.5 x .75")
72-pin SIMM (4.25 x 1")
168-pin DIMM (5.375 x 1")
144-pin SODIMM (2.625 x 1")
72-pin SODIMM (2.375 x 1")

Figure 3:2 Modern memory modules

One recurrent question about desktop computers that comes up all the time is, "Why does a computer need different types of memory systems?"

3.2 Types of Memory

3.2.1 Attributes and Characteristics of memory

The complex subject of computer memory is made more manageable if we classify memory systems according to their key characteristics. The most important of these are listed in Table 3.1.

Table 3-1: Attributes and Characteristics of Memory Storage

Attribute	**Characteristics**	**Attribute**	**Characteristics**
Performance	Access time Cycle time Transfer rate	**Physical Type**	Semiconductor Magnetic Optical Magneto-optical
Physical Characteristics	Volatile/non-volatile Erasable/non-erasable	**Location**	Internal (e.g. processor registers, main memory, cache) External (e.g. optical disks, magnetic disks, tapes)
Capacity	Number of words Number of bytes	**Unit of Transfer**	Word Block
Access Method	Sequential Direct Random Associative	**Organization**	Memory modules

3.2.1.1 Memory location

The term **location** in Table 3.1 refers to whether memory is internal and external to the computer. **Internal memory** is often equated with main memory. But there are other forms of internal memory. The processor requires its own local memory, in the form of registers. Further, as we shall see, the control unit portion of the processor may also require its own internal memory. We will defer discussion of these latter two types of internal memory to later chapters. Cache is another form of internal memory. **External memory** consists of peripheral storage devices, such as disk and tape, which are accessible to the processor via I/O controllers

3.2.1.2 Memory capacity

An obvious characteristic of memory is its **capacity**. For internal memory, this is typically expressed in terms of bytes (1 byte 8 bits) or words. Common word lengths are 8, 16, and 32 bits. External memory capacity is typically expressed in terms of bytes.

3.2.1.3 Unit of transfer

A related concept is the **unit of transfer**. For internal memory, the unit of transfer is equal to the number of electrical lines into and out of the memory module. This may be equal to the word length, but is often larger, such as 64, 128, or 256 bits. To clarify this point, consider three related concepts for internal memory:

- Word: The "natural" unit of organization of memory. The size of the word is typically equal to the number of bits used to represent an integer and to the instruction length. Unfortunately, there are many exceptions. For example, the CRAY C90 (an older model CRAY supercomputer) has a 64-bit word length but uses a 46-bit integer representation. The Intel x86 architecture has a wide variety of instruction lengths, expressed as multiples of bytes, and a word size of 32 bits.
- Addressable units: In some systems, the addressable unit is the word. However, many systems allow addressing at the byte level. In any case, the relationship between the length in bits A of an address and the number N of addressable units is 2A N.
- Unit of transfer: For main memory, this is the number of bits read out of or written into memory at a time. The unit of transfer need not equal a word or an addressable unit. For external memory, data are often transferred in much larger units than a word, and these are referred to as blocks.

3.2.1.4 Method of access

Another distinction among memory types is the method of accessing units of data. These include the following:

- Sequential access: Memory is organized into units of data, called records. Access must be made in a specific linear sequence. Stored addressing information is used to separate records and assist in the retrieval process. A shared read–write mechanism is used, and this must be moved from its current location to the desired location, passing and rejecting each intermediate record. Thus, the time to access an arbitrary record is highly variable. Tape units, discussed in Chapter 6, are sequential access.

- Direct access: As with sequential access, direct access involves a shared read–write mechanism. However, individual blocks or records have a unique address based on physical location. Access is accomplished by direct access to reach a general vicinity plus sequential searching, counting, or waiting to reach the final location. Again, access time is variable. Disk units, discussed in Chapter 6, are direct access.
- Random access: Each addressable location in memory has a unique, physically wired-in addressing mechanism. The time to access a given location is independent of the sequence of prior accesses and is constant. Thus, any location can be selected at random and directly addressed and accessed. Main memory and some cache systems are random access.
- Associative: This is a random access type of memory that enables one to make a comparison of desired bit locations within a word for a specified match, and to do this for all words simultaneously. Thus, a word is retrieved based on a portion of its contents rather than its address. As with ordinary random-access memory, each location has its own addressing mechanism, and retrieval time is constant independent of location or prior access patterns. Cache memories may employ associative access.

3.2.1.5 Memory performance

From a user's point of view, the two most important characteristics of memory are capacity and performance. Three performance parameters are used:

- Access time (latency): For random-access memory, this is the time it takes to perform a read or write operation, that is, the time from the instant that an address is presented to the memory to the instant that data have been stored or made available for use. For non-random-access memory, access time is the time it takes to position the read–write mechanism at the desired location.
- Memory cycle time: This concept is primarily applied to random-access memory and consists of the access time plus any additional time required before a second access can commence. This additional time may be required for transients to die out on signal lines or to regenerate data if they are read destructively.

Note that memory cycle time is concerned with the system bus, not the processor.

Transfer rate: This is the rate at which data can be transferred into or out of a memory unit. For random-access memory, it is equal to 1/(cycle time). For non-random-access memory, the following relationship holds:

$$T_N = T_A + \frac{n}{R}$$

Where:

- TN Average time to read or write N bits
- TA Average access time
- n Number of bits
- R Transfer rate, in bits per second (bps)

3.2.1.6 Physical type

A variety of physical types of memory have been employed. The most common today are semiconductor memory, magnetic surface memory, used for disk and tape, and optical and magneto-optical.

Several physical characteristics of data storage are important. In a volatile memory, information decays naturally or is lost when electrical power is switched off.

In a **nonvolatile memory**, information once recorded remains without deterioration until deliberately changed; no electrical power is needed to retain information. Magnetic-surface memories are nonvolatile. Semiconductor memory may be either volatile or nonvolatile. Non-erasable memory cannot be altered, except by destroying the storage unit. Semiconductor memory of this type is known as read-only memory (ROM). Of necessity, a practical non-erasable memory must also be nonvolatile.

For random-access memory, the organization is a key design issue. By organization is meant the physical arrangement of bits to form words.

Using the mechanism for storage and retrieval of data, a memory system can be classified as one of the following four types:

1. Random-access memory (RAM)
    a. Read = write memory (RWM)
    b. Read-only memory (ROM)
2. Content-addressable memory (CAM) or associative memory (AM)
3. Sequential-access memory (SAM)
4. Direct-access memory (DAM)

Primary memory is of the RAM type. CAMs are used in special applications in which rapid data search and retrieval are needed. SAM and DAM are used as secondary memory devices.

3.2.2 Random-Access Memory

In a RAM, any addressable location in the memory can be accessed in a random manner. That is, the process of reading from and writing into a location in a RAM is the same and consumes an equal amount of time no matter where the location is physically in the memory. The two types of RAM available are read/write and ROMs.

- **Read /Write Memory (RWM).** The most common type of main memory is the RWM. In an RWM, each memory register or memory location has an ‘‘address’’ associated with it. Data are input into (written into) and output from (read from) a memory location by accessing the location using its ‘‘address.’’ The memory address register (MAR) stores such an address. With n bits in the MAR, 2^n locations can be addressed, and they are numbered from 0 through 2^{n-1}.

  A memory word is defined as the most often accessed unit of data. The typical word sizes used in memory organizations of commercially available machines are 6, 16, 32, 36, and 64 bits. In addition to addressing a memory word, it is possible to address a portion of it (e.g., half-word, quarter -word) or a multiple of it (e.g., double-word, quad word), depending on the memory organization. In a ‘‘byte-addressable,’’ memory, for example, an address is associated with each byte (usually 8 bits per byte) in the memory, and a memory word consists of one or more bytes.

- **Read-Only Memory (ROM):** The literature routinely uses the acronym RAM to mean RWM. We will follow this popular practice and use RWM only when the context requires us to be more specific. We have included Memory Address Register (MAR) and Memory Buffer Register (MBR) as components of the memory system in this mode l. In practice, these registers may not be located in the memory subsystem, but other registers in the system may serve the functions of these registers.

  ROM is also a RAM, except that data can only be read from it. Data are usually written into a ROM either by the memory manufacturer or by the user in an off-line mode; i.e., by special devices that can write (burn) the data pattern into the ROM. A ROM is also used as main memory and contains data and programs that are not usually altered in real time during the system operation. In general, we assume that the data on output lines are available as long as the memory enable signal is on and it is latched into an external buffer register. A buffer is provided as part of the memory system, in some technologies.

3.2.3 Content-Addressable Memory

In this type of memory, the concept of address is not usually present: rather, the memory logic searches for the locations containing a specific pattern, and hence the descriptor ‘‘content addressable’ ’ or ‘‘associative’’ is used. In the typical operation f this memory, the data to be searched for are first provided to the memory. The memory hardware then searches for a match and either identifies the location or locations containing that data or returns with a ‘‘no match’’ if none of the locations contain the data.

3.2.4 Memory Hierarchy

The primary memory of a computer system is always built out of RAM devices, thereby allowing the processing unit to access data and instructions in the memory as quickly as possible. It is necessary that the program or data be in the primary memory when the processing unit needs them. This would call for a large primary memory when programs and data blocks are large, thereby increasing the memory cost. In practice, it is not really necessary to store the complete program or data in the primary memory as long as the portion of the program or data needed by the processing unit is in the primary memory.

A secondary memory built out of direct- or serial-access devices is then used to store programs and data not immediately needed by the processing unit. Since random-access devices are more expensive than secondary memory devices, a cost-effective memory system results when the primary memory capacity is minimized. But this organization introduces an overhead into the memory operation. This is because mechanisms to bring the required portion of the programs and data into primary memory as needed will have to be devised. These mechanisms form what is called a virtual memory scheme.

In a virtual memory scheme, the user assumes that the total memory capacity (primary plus secondary) is available for programming. The operating system manages the moving in and out of portions (segments or pages) of program and data into and out of the primary memory.

Even with current technologies, the primary memory hardware is slow compared with the processing unit hardware. To reduce this speed gap, a small but faster memory is usually introduced between the main memory and the processing unit. This memory block is called cache memory and is usually 10–100 times faster than the primary memory. A virtual memory mechanism similar to that between primary and secondary memories is then needed to manage operations between main memory and cache. The set of instructions and data that are immediately needed by the processing unit are brought from the primary memory into cache and retained there. A parallel fetch operation is possible in that while the cache unit is being filled from the main memory, the processing unit can fetch from the cache, thus narrowing the memory-to-processor speed gap.

Note that the registers in the processing unit are temporary storage devices. They are the fastest components of the computer system memory.

Thus, in a general-purpose computer system there is a memory hierarchy in which the highest speed memory is closest to the processing unit and is most expensive. The least-expensive and slowest memory devices are farthest from the processing unit. The next Figure illustrates the memory hierarchy.

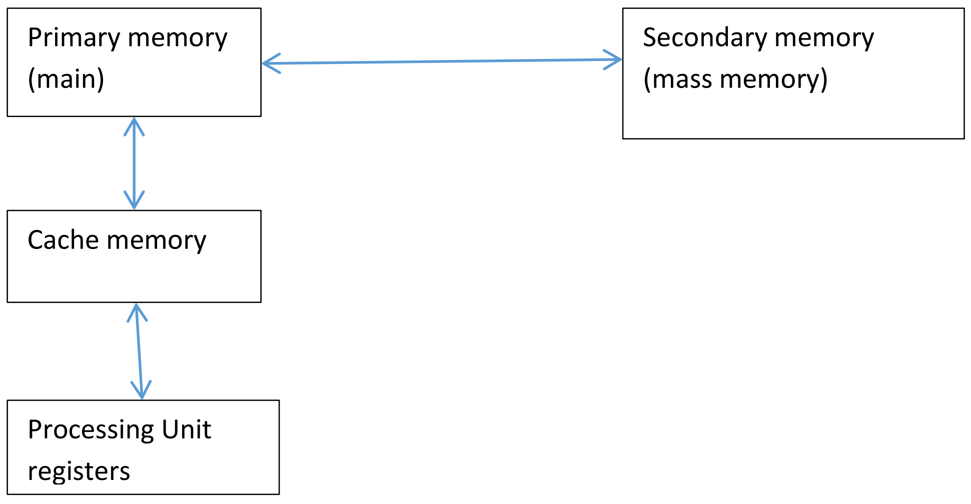


Table 3-2: Memory Hierarchy

Based on different levels of memory access with respect to the processing unit, a typical computer has:

- Level 1 and level 2 caches

- Normal system RAM
- Virtual memory
- A hard disk

Fast, powerful CPUs need quick and easy access to large amounts of data in order to maximize their performance. If the CPU cannot get to the data it needs, it literally stops and waits for it. Modern CPUs running at speeds of over **3 gigahertz** can consume massive amounts of data -- potentially billions of bytes per second. The problem that computer designers face is that the memory that can keep up with a 3-gigahertz CPU is extremely **expensive** -- much more expensive than anyone can afford in large quantities.

Computer designers have solved the cost problem by "**tiering**" memory -- using expensive memory in small quantities and then backing it up with larger quantities of less expensive memory. The cheapest form of read/write memory in wide use today is the **hard disk**. Hard disks provide large quantities of inexpensive, permanent storage. You can buy hard disk space for pennies per megabyte, but it can take a good bit of time (approaching a second) to read a megabyte off a hard disk. Because storage space on a hard disk is so cheap and plentiful, it forms the final stage of a CPUs memory hierarchy, called virtual memory.

The next level of the hierarchy is **RAM**.

The **bit size** of a CPU tells you how many bytes of information it can access from RAM at the same time. For example, a 16-bit CPU can process 2 bytes at a time (1 byte = 8 bits, so 16 bits = 2 bytes), and a 64-bit CPU can process 8 bytes at a time.

Megahertz (MHz) is a measure of a CPU's processing speed, or **clock cycle**, in millions per second. So, a 32-bit 800-MHz Pentium III can potentially process 4 bytes simultaneously, 800 million times per second (possibly more based on pipelining)! The goal of the memory system is to meet those requirements.

A computer's system RAM alone is not fast enough to match the speed of the CPU. That is why you need a **cache** (discussed later). However, the faster RAM is, the better. Most chips today operate with a cycle rate of 50 to 70 nanoseconds. The read/write speed is typically a function of the type of RAM used, such as DRAM, SDRAM, RAMBUS. We will talk about these various types of memory later.

3.2.5 System RAM

System RAM speed is controlled by **bus width** and **bus speed**. Bus width refers to the number of bits that can be sent to the CPU simultaneously, and bus speed refers to the number of times a group of bits can be sent each second. A **bus cycle** occurs every time data travels from memory to the CPU. For example, a 100-MHz 32-bit bus is theoretically capable of sending 4 bytes (32 bits divided by 8 = 4 bytes) of data to the CPU 100 million times per second, while a 66-MHz 16-bit bus can send 2 bytes of data 66 million times per second. If you do the math, you'll find that simply changing the bus width from 16 bits to 32 bits and the speed from 66

MHz to 100 MHz in our example allows for three times as much data (400 million bytes versus 132 million bytes) to pass through to the CPU every second.

In reality, RAM doesn't usually operate at optimum speed. **Latency** changes the equation radically. Latency refers to the number of clock cycles needed to read a bit of information. For example, RAM rated at 100 MHz is capable of sending a bit in 0.00000001 seconds, but may take 0.00000005 seconds to start the read process for the first bit. To compensate for latency, CPUs uses a special technique called **burst mode**.

Burst mode depends on the expectation that data requested by the CPU will be stored in **sequential memory cells**. The memory controller anticipates that whatever the CPU is working on will continue to come from this same series of memory addresses, so it reads several consecutive bits of data together. This means that only the first bit is subject to the full effect of latency; reading successive bits takes significantly less time. The **rated burst mode** of memory is normally expressed as four numbers separated by dashes. The first number tells you the number of clock cycles needed to begin a read operation; the second, third and fourth numbers tell you how many cycles are needed to read each consecutive bit in the row, also known as the **wordline**. For example: 5-1-1-1 tells you that it takes five cycles to read the first bit and one cycle for each bit after that. Obviously, the lower these numbers are, the better the performance of the memory.

Burst mode is often used in conjunction with **pipelining**, another means of minimizing the effects of latency. Pipelining organizes data retrieval into a sort of assembly-line process. The memory controller simultaneously reads one or more words from memory, sends the current word or words to the CPU and writes one or more words to memory cells. Used together, burst mode and pipelining can dramatically reduce the lag caused by latency.

So why wouldn't you buy the fastest, widest memory you can get? The speed and width of the memory's bus should match the system's bus. You can use memory designed to work at 100 MHz in a 66-MHz system, but it will run at the 66-MHz speed of the bus so there is no advantage, and 32-bit memory won't fit on a 16-bit bus.

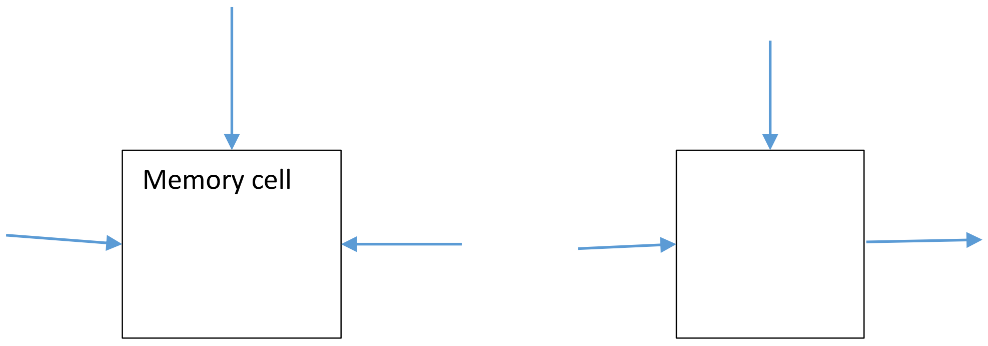
Even with a wide and fast bus, it still takes longer for data to get from the memory card to the CPU than it takes for the CPU to actually process the data. That's where caches come in.

3.2.5.1 Organization

The basic element of a semiconductor memory is the memory cell. Although a variety of electronic technologies are used, all semiconductor memory cells share certain properties:

- They exhibit two stable (or semi-stable) states, which can be used to represent binary 1 and 0.
- They are capable of being written into (at least once), to set the state.
- They are capable of being read to sense the state.

Figure 5.1 depicts the operation of a memory cell. Most commonly, the cell has three functional terminals capable of carrying an electrical signal. The select terminal, as the name suggests, selects a memory cell for a read or write operation.



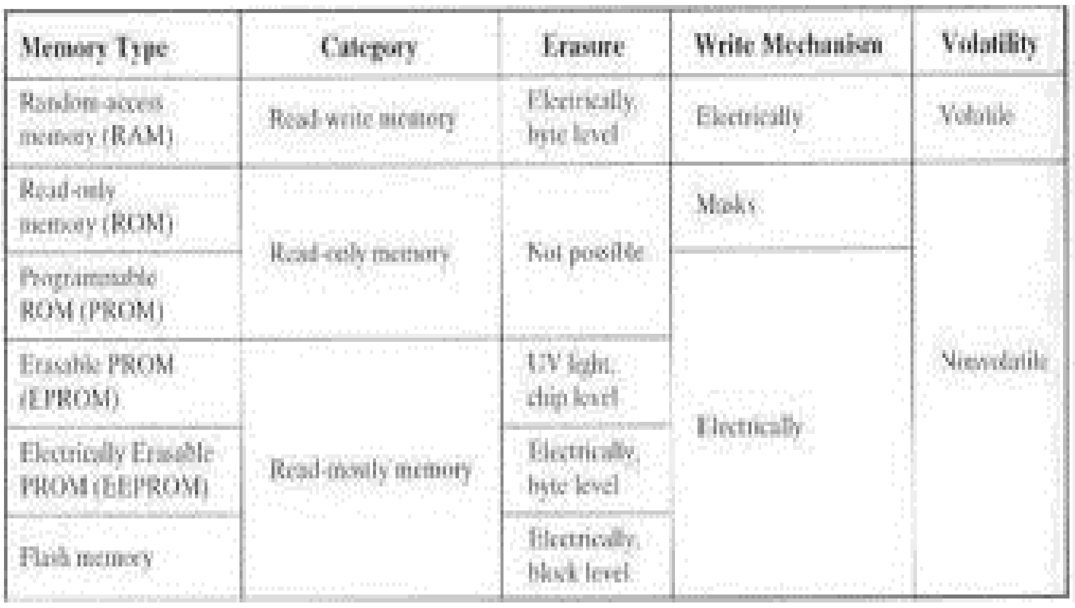
The control terminal indicates read or write. For writing, the other terminal provides an electrical signal that sets the state of the cell to 1 or 0. For reading, that terminal is used for output of the cell's state. The details of the internal organization, functioning, and timing of the memory cell depend on the specific integrated circuit technology used and are beyond the scope of this book, except for a brief summary.

For our purposes, we will take it as given that individual cells can be selected for reading and writing operations.

3.2.5.2 DRAM and SRAM

All of the memory types that we will explore in this chapter are random access. That is, individual words of memory are directly accessed through wired-in addressing logic. Table 5.1 lists the major types of semiconductor memory. The most common is referred to as random-access memory (RAM). This is, of course, a misuse of the term, because all of the types listed in the table are random access. One distinguishing characteristic of RAM is that it is possible both to read data from the memory and to write new data into the memory easily and rapidly. Both the reading and writing are accomplished through the use of electrical signals.

Table 3-3: Types of Memory



Memory Type	Category	Erasure	Write Mechanism	Volatility
Random-access memory (RAM)	Read-write memory	Electrically, byte level	Electrically	Volatile
Read-only memory (ROM)	Read-only memory	Not possible	Masks	Nonvolatile
Programmable ROM (PROM)			Electrically	
Erasable PROM (EPROM)	Read-mostly memory	UV light, chip level		
Electrically Erasable PROM (EEPROM)		Electrically, byte level		
Flash memory		Electrically, block level		

3.2.6 Advanced DRAM

One of the most critical system bottlenecks when using high-performance processors is the interface to main internal memory. This interface is the most important pathway in the entire computer system. The basic building block of main memory remains the DRAM chip, as it has for decades; until recently, there had been no significant changes in DRAM architecture since the early 1970s. The traditional DRAM chip is constrained both by its internal architecture and by its interface to the processor's memory bus.

We have seen that one attack on the performance problem of DRAM main memory has been to insert one or more levels of high-speed SRAM cache between the DRAM main memory and the processor. But SRAM is much costlier than DRAM, and expanding cache size beyond a certain point yields diminishing returns.

Table 5.3 Performance Comparison of Some DRAM Alternatives

	Clock Frequency (MHz)	**Transfer Rate (GB/s)**	**Access Time (ns)**	**Pin Count**
SDRAM	166	1.3	18	168
DDR	200	3.2	12.5	184
RDRAM	4.8	12	162	600

In recent years, a number of enhancements to the basic DRAM architecture have been explored, and some of these are now on the market. The schemes that currently dominate the market are SDRAM, DDR-DRAM, and RDRAM. Table 5.3 provides a performance comparison. CDRAM has also received considerable attention.

We examine each of these approaches in this section.

3.2.6.1 Synchronous DRAM

One of the most widely used forms of DRAM is the synchronous DRAM (SDRAM). Unlike the traditional DRAM, which is asynchronous, the SDRAM exchanges data with the processor synchronized to an external clock signal and running at the full speed of the processor/memory bus without imposing wait states.

In a typical DRAM, the processor presents addresses and control levels to the memory, indicating that a set of data at a particular location in memory should be either read from or written into the DRAM. After a delay, the access time, the DRAM either writes or reads the data. During the access-time delay, the DRAM performs various internal functions, such as activating the high capacitance of the row and column lines, sensing the data, and routing the data out through the output buffers. The processor must simply wait through this delay, slowing system performance.

With synchronous access, the DRAM moves data in and out under control of the system clock. The processor or other master issues the instruction and address information, which is latched by the DRAM. The DRAM then responds after a set number of clock cycles. Meanwhile, the master can safely do other tasks while the SDRAM is processing the request.

There is now an enhanced version of SDRAM, known as double data rate SDRAM (DDR-SDRAM) that overcomes the once-per-cycle limitation. DDRSDRAM can send data to the processor twice per clock cycle.

3.2.6.2 Rambus DRAM

RDRAM, developed by Rambus, has been adopted by Intel for its Pentium and Itanium processors. It has become the main competitor to SDRAM. RDRAM chips are vertical packages, with all pins on one side. The chip exchanges data with the processor over 28 wires no more than 12 centimeters long.The bus can address up to 320 RDRAM chips and is rated at 1.6 GBps.

The special RDRAM bus delivers address and control information using an asynchronous block-oriented protocol. After an initial 480 ns access time, this produces the 1.6 GBps data rate. What makes this speed possible is the bus itself, which defines impedances, clocking, and signals very precisely. Rather than being controlled by the explicit RAS, CAS, R/W, and CE signals used in conventional DRAMs, an RDRAM gets a memory request over the high-speed bus. This request contains the desired address, the type of operation, and the number of bytes in the operation.

3.2.7 Cache and Registers

Caches are designed to alleviate this bottleneck by making the data used most often by the CPU instantly available. This is accomplished by building a small amount of memory, known as **primary** or **level 1** cache, right into the CPU. Level 1 cache is very small, normally ranging between 2 kilobytes (KB) and 64 KB.

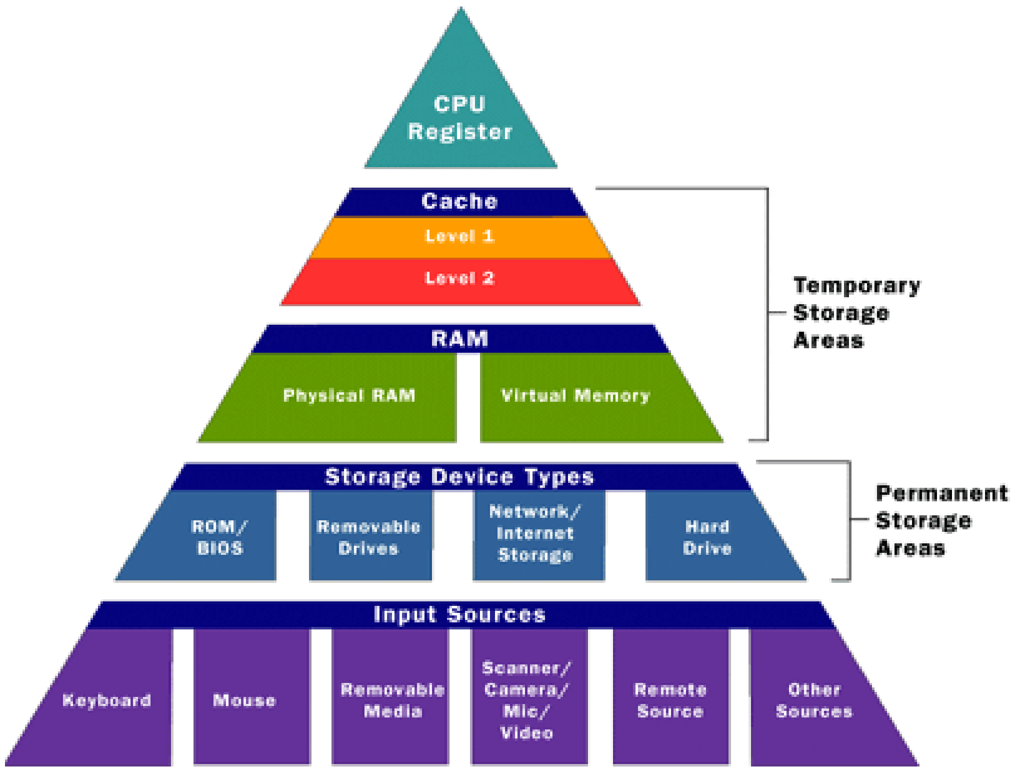


Table 3-4: Computer memory hierarchy

The **secondary** or **level 2** cache typically resides on a memory card located near the CPU. The level 2 cache has a direct connection to the CPU. A dedicated integrated circuit on the motherboard, the **L2 controller**, regulates the use of the level 2 cache by the CPU. Depending on the CPU, the size of the level 2 cache ranges from 256 KB to 2 megabytes (MB). In most systems, data needed by the CPU is accessed from the cache approximately 95 percent of the time, greatly reducing the overhead needed when the CPU has to wait for data from the main memory.

Volatility

Memory can be split into two main categories: volatile and nonvolatile. Volatile memory loses any data as soon as the system is turned off; it requires constant power to remain viable. Most types of RAM fall into this category. Nonvolatile memory does not lose its data when the system or device is turned off. A number of types of memory fall into this category. The most familiar is ROM, but Flash memory storage devices such as CompactFlash or SmartMedia cards are also forms of nonvolatile memory.

Some inexpensive systems dispense with the level-2 cache altogether. Many high performance CPUs now have the level 2 cache actually built into the CPU chip itself. Therefore, the size of the level 2 cache and whether it is **onboard** (on the CPU) is a major determining factor in the performance of a CPU.

A particular type of RAM, **static random access memory** (SRAM), is used primarily for cache. SRAM uses multiple transistors, typically four to six, for each memory cell. It has an external gate array known as a **bistable-multivibrator** that switches, or flip-flops, between two states. This means that it does not have to be continually refreshed like DRAM. Each cell will maintain its data as long as it has power. Without the need for constant refreshing, SRAM can operate extremely quickly. But the complexity of each cell make it prohibitively expensive for use as standard RAM.

The SRAM in the cache can be **asynchronous** or **synchronous**. Synchronous SRAM is designed to exactly match the speed of the CPU, while asynchronous is not. That little bit of timing makes a difference in performance. Matching the CPU's clock speed is a good thing, so always look for synchronized SRAM.

The final step in memory is the **registers**. These are memory cells built right into the CPU that contain specific data needed by the CPU, particularly the **arithmetic and logic unit** (ALU). An integral part of the CPU itself, they are controlled directly by the compiler that sends information for the CPU to process.

3.2.8 Memory System Parameters

The most important characteristics of any memory system are:

- **Capacity** – The capacity of the storage system is the maximum number of units (bits, bytes, or words) of data it can store. The capacity of a RAM, for instance, is the product of the number of memory words and the word size. A 2K34 memory, for example, can store 2K (K=1024=210) words each containing 4 bits, or a total of 23102434 bits.
- **Data-Access Time** - The access time is the time taken by the memory module to access the data after an address is provided to the module. The data appear in the MBR at the end of this time in a RAM. The access time in a non-RAM is a function of the location of the data on the medium with reference to the position of read/write transducers.
- **The Data-Transfer Rate** - The data-transfer rate is the number of bps at which the data can be read out of the memory. This rate is the product of the reciprocal of access time and the number of bits in the unit of data (data word) being read. This parameter is of more significance in non-RAM systems than in RAMs.
- **The Cycle Time (frequency at which memory can be accessed**) - The cycle time is a measure of how often the memory can be accessed. The cycle time is equal to the access time in nondestructive readout memories in which the data can be read without being destroyed. In some storage systems, data are destroyed during a read operation (destructive read-out). A rewrite operation is necessary to restore the data. The cycle

time in such devices is defined as the time it takes to read and restore data, since a new read operation cannot be performed until the rewrite has been completed.

- **Cost** - The cost is the product of capacity and the price of memory device per bit. RAMs are usually more costly than other memory devices.

Some of the other parameters of interest are fault tolerance, radiation hardness, weight, and data compression and integrity depending on the application the memory is used.

Self-Review Exercises

1. Question One
    a. Give any four differences between primary memory and secondary memory (4Mks)
    b. State any three types of secondary storage media. (3 marks)
    c. State three functions of the central processing unit (CPU) (3mks)
    d. Convert the number FC1 to its binary equivalent (3 marks)

Further reading:

Rabaey, Jan M.; Chadrakasan, Anatha; & Nikolic, Barivoje (2003); Digital Integrated Circuits: a design perspective 5th Ed.; Pearson Education Inc

4 Week 7/8: Input/output organization

- Introduction to Input Output system
- I/O devices and their diversity,
- I/O interconnection to CPU and Memory.
- Direct memory access (DMA)

4.1 Introduction

In addition to the processor and a set of memory modules, the third key element of a computer system is a set of I/O modules. Each module interfaces to the system bus or central switch and controls one or more peripheral devices. An I/O module is not simply a set of mechanical connectors that wire a device into the system bus. Rather, the I/O module contains logic for performing a communication function between the peripheral and the bus.

One may wonder why it is not possible to connect peripherals directly to the system bus. The reasons are as follows:

- There are a wide variety of peripherals with various methods of operation. It would be impractical to incorporate the necessary logic within the processor to control a range of devices.
- The data transfer rate of peripherals is often much slower than that of the memory or processor. Thus, it is impractical to use the high-speed system bus to communicate directly with a peripheral.
- On the other hand, the data transfer rate of some peripherals is faster than that of the memory or processor. Again, the mismatch would lead to inefficiencies if not managed properly.
- Peripherals often use different data formats and word lengths than the computer to which they are attached.

Thus, for the foregoing reasons, an I/O module is required. This module has two major functions

- Interface to the processor and memory via the system bus or central switch
- Interface to one or more peripheral devices by tailored data links

This section starts with a brief discussion of external devices, followed by an overview of the structure and function of an I/O module. Then we look at the various ways in which the I/O function can be performed in cooperation with the processor and memory: the internal I/O interface. Finally, we examine the external I/O interface, between the I/O module and the outside world.

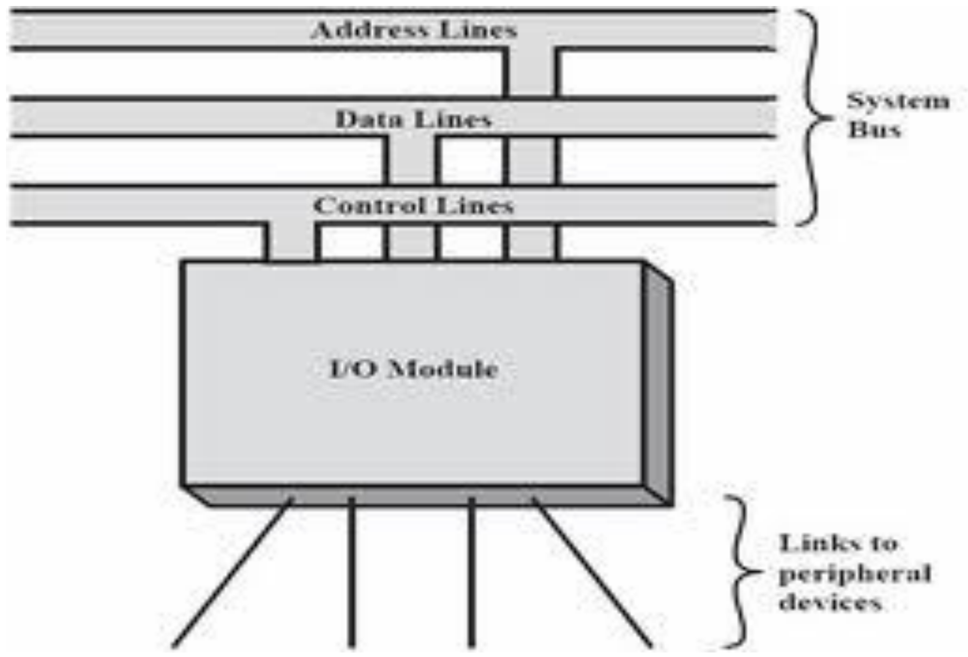


Figure 4:1 Generic Input/output Module

4.2 External devices

I/O operations are accomplished through a wide assortment of external devices that provide a means of exchanging data between the external environment and the computer. An external device attaches to the computer by a link to an I/O module (Figure 4.1). The link is used to exchange control, status, and data between the I/O module and the external device. An external device connected to an I/O module is often referred to as a peripheral device or, simply, a peripheral.

External devices can be broadly classified into three categories:

- **Human readable**: Suitable for communicating with the computer user. Examples of human-readable devices are video display terminals (VDTs) and printers.
- **Machine readable**: Suitable for communicating with equipment. Examples of machine-readable devices are magnetic disk and tape systems, and sensors and actuators, such as are used in a robotics application. NB: disk and tape systems can be viewed as I/O devices or memory devices. From a functional point of view, these devices are part of the memory hierarchy and from a structural point of view, they devices are controlled by I/O modules and are hence to be considered in this chapter.
- **Communication**: Suitable for communicating with remote devices. They allow a computer to exchange data with a remote device, which may be a human-readable device, such as a terminal, a machine-readable device, or even another computer

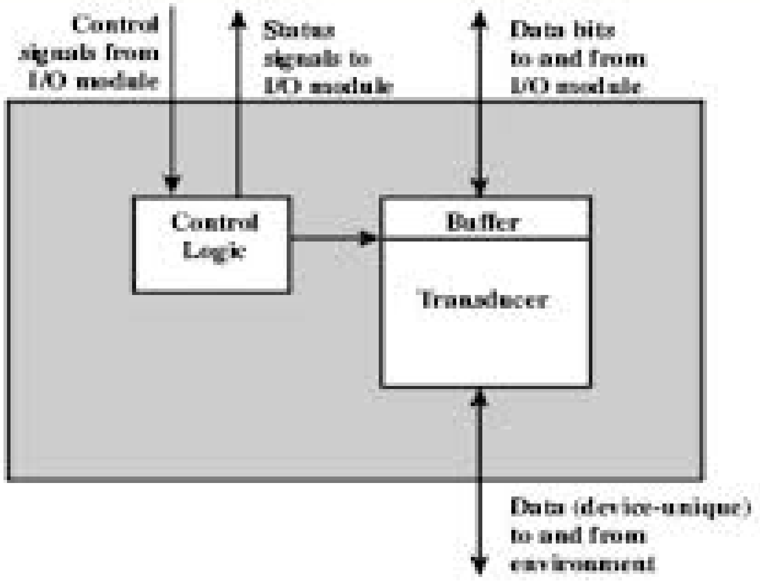

Control signals from I/O module
Status signals to I/O module
Data bits to and from I/O module
Control Logic
Buffer
Transducer
Data (device-unique) to and from environment

Figure 4:2 Block Diagram of an External Device

Generally, the nature of an external device is indicated in Figure 4.2. The interface to the I/O module is in the form of control, data, and status signals. Control signals determine the function that the device will perform, such as send data to the I/O module (INPUT or READ), accept data from the I/O module (OUTPUT or WRITE), report status, or perform some control function particular to the device (e.g., position a disk head). Data are in the form of a set of bits to be sent to or received from the I/O module. Status signals indicate the state of the device. Examples are READY/NOT-READY to show whether the device is ready for data transfer.

Control logic associated with the device controls the device's operation in response to direction from the I/O module. The transducer converts data from electrical to other forms of energy during output and from other forms to electrical during input. Typically, a buffer is associated with the transducer to temporarily hold data being transferred between the I/O module and the external environment; a buffer size of 8 to 16 bits is common.

The interface between the I/O module and the external device will be examined later.

4.3 Input/Output bus Module

The most common bus structures are single bus and multi-bus. In a single-bus structure, all data and address flow is through one bus. A multi-bus structure typically consists of several buses, each dedicated to certain transfers; for example, one bus could be a data bus and the other could be an address bus. Multi-bus structures provide the advantage of tailoring each bus to the set of transfers it is dedicated to and permits parallel operations. Single-bus structures have the advantage of uniformity of design.

Other characteristics to be considered in evaluating bus structures are the amount of hardware required and the data transfer rates possible.

4.3.1 Module Function

The major functions or requirements for an I/O module fall into the following categories:

- Control and timing
- Processor communication
- Device communication
- Data buffering
- Error detection

During any period of time, the processor may communicate with one or more external devices in unpredictable patterns, depending on the program's need for I/O. The internal resources, such as main memory and the system bus, must be shared among a number of activities, including data I/O. Thus, the I/O function includes a control and timing requirement, to coordinate the flow of traffic between internal resources and external devices. For example, the control of the transfer of data from an external device to the processor might involve the following sequence of steps:

1. The processor interrogates the I/O module to check the status of the attached device.
2. The I/O module returns the device status.
3. If the device is operational and ready to transmit, the processor requests the transfer of data, by means of a command to the I/O module.
4. The I/O module obtains a unit of data (e.g., 8 or 16 bits) from the external device.
5. The data are transferred from the I/O module to the processor.

If the system employs a bus, then each of the interactions between the processor and the I/O module involves one or more bus arbitrations. The preceding simplified scenario also illustrates that the I/O module must communicate with the processor and with the external device. Processor communication involves the following:

- Command decoding: The I/O module accepts commands from the processor, typically sent as signals on the control bus. For example, an I/O module for a disk drive might accept the following commands: READ SECTOR, WRITE SECTOR, SEEK track number, and SCAN record ID. The latter two commands each include a parameter that is sent on the data bus.
- Data: Data are exchanged between the processor and the I/O module over the data bus.
- Status reporting: Because peripherals are so slow, it is important to know the status of the I/O module. For example, if an I/O module is asked to send data to the processor (read), it may not be ready to do so because it is still working on the previous I/O command. This fact can be reported with a status signal. Common status signals are BUSY and READY. There may also be signals to report various error conditions.
- Address recognition: Just as each word of memory has an address, so does each I/O device. Thus, an I/O module must recognize one unique address for each peripheral it controls.

On the other side, the I/O module must be able to perform device communication. This communication involves commands, status information, and data (Figure 4.2).

An essential task of an I/O module is data buffering. Whereas the transfer rate into and out of main memory or the processor is quite high, the rate is orders of magnitude lower for many peripheral devices and covers a wide range. Data coming from main memory are sent to an I/O module in a rapid burst. The data are buffered in the I/O module and then sent to the peripheral device at its data rate. In the opposite direction, data are buffered so as not to tie up the memory in a slow transfer operation. Thus, the I/O module must be able to operate at both device and memory speeds. Similarly, if the I/O device operates at a rate higher than the memory access rate, then the I/O module performs the needed buffering operation.

Finally, an I/O module is often responsible for error detection and for subsequently reporting errors to the processor. One class of errors includes mechanical and electrical malfunctions reported by the device (e.g., paper jam, bad disk track). Another class consists of unintentional changes to the bit pattern as it is transmitted from device to I/O module. Some form of error-detecting code is often used to detect transmission errors. A simple example is the use of a parity bit on each character of data. For example, the IRA character code occupies 7 bits of a byte. The eighth bit is set so that the total number of 1s in the byte is even (even parity) or odd (odd parity).When a byte is received, the I/O module checks the parity to determine whether an error has occurred.

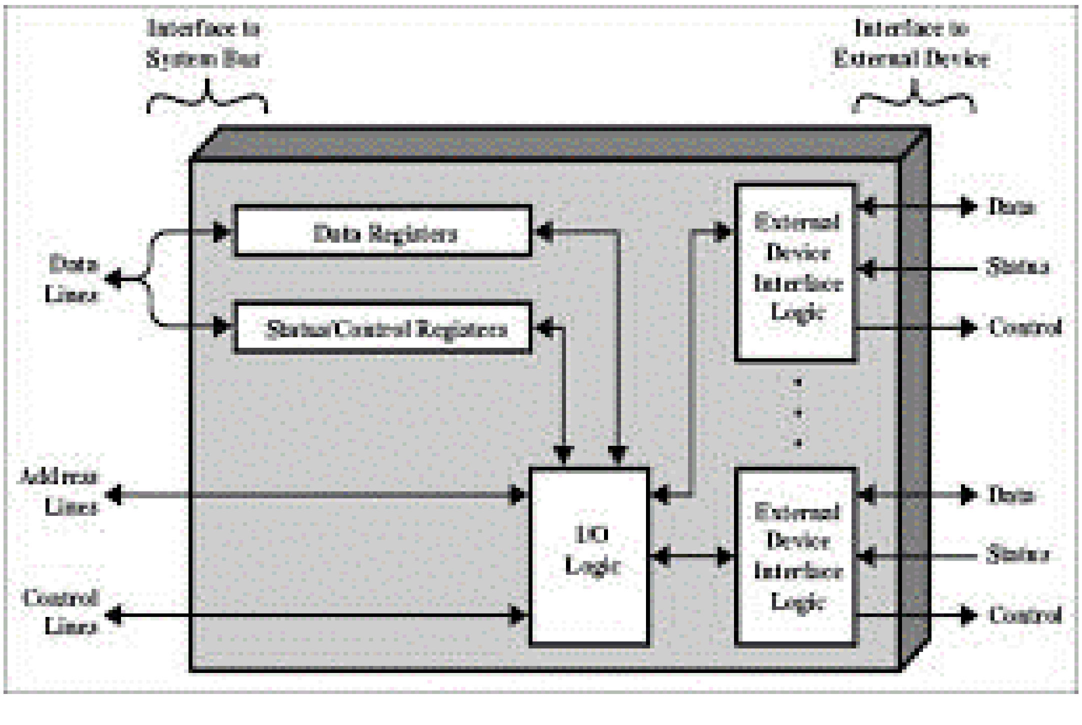


Figure 4:3: Block Diagram of an I/O Module

4.3.2 I/O Module Structure

I/O modules vary considerably in complexity and the number of external devices that they control. Figure 4.3 provides a general block diagram of an I/O module. The module connects to the rest of the computer through a set of signal lines (e.g., system bus lines). Data transferred to and from the module are buffered in one or more data registers. There may also be one or more status registers that provide current status information. A status register may also function as a control register, to accept detailed control information from the processor. The logic within the module interacts with the processor via a set of control lines. The processor uses the control lines to issue commands to the I/O module.

Some of the control lines may be used by the I/O module (e.g., for arbitration and status signals). The module must also be able to recognize and generate addresses associated with the devices it controls. Each I/O module has a unique address or, if it controls more than one external device, a unique set of addresses. Finally, the I/O module contains logic specific to the interface with each device that it controls.

4.4 Programmed I/O

Three techniques are possible for I/O operations. With programmed I/O, data are exchanged between the processor and the I/O module. The processor executes a program that gives it direct control of the I/O operation, including sensing device status, sending a read or write command, and transferring the data. When the processor issues a command to the I/O module, it must wait until the I/O operation is complete. If the processor is faster than the I/O module, this is wasteful of processor time. With interrupt-driven I/O, the processor issues an I/O command, continues to execute other instructions, and is interrupted by the I/O module when the latter has completed its work. With both programmed and interrupt I/O, the processor is responsible for extracting data from main memory for output and storing data in main memory for input. The alternative is known as direct memory access (DMA).

Table 4-1: I/O Techniques

	No Interrupts	Use of Interrupts
I/O-to-memory transfer through processor	Programmed I/O	Interrupt-driven I/O
Direct I/O-to-memory transfer)		Direct memory access (DMA

In this mode, the I/O module and main memory exchange data directly, without processor involvement. Table 4.1 indicates the relationship among these three techniques. In this section, we explore programmed I/O. Interrupt I/O and DMA are explored in the following two sections, respectively.

4.4.1 Overview of Programmed I/O

When the processor is executing a program and encounters an instruction relating to I/O, it executes that instruction by issuing a command to the appropriate I/O module. With programmed I/O, the I/O module will perform the requested action and then set the appropriate bits in the I/O status register (Figure 4.3). The I/O module takes no further action to alert the processor. In particular, it does not interrupt the processor. Thus, it is the responsibility of the processor periodically to check the status of the I/O module until it finds that the operation is complete.

To explain the programmed I/O technique, we view it first from the point of view of the I/O commands issued by the processor to the I/O module, and then from the point of view of the I/O instructions executed by the processor.

4.4.2 I/O Commands

To execute an I/O-related instruction, the processor issues an address, specifying the particular I/O module and external device, and an I/O command. There are four types of I/O commands that an I/O module may receive when it is addressed by a processor:

- Control: Used to activate a peripheral and tell it what to do. For example, a magnetic-tape unit may be instructed to rewind or to move forward one record. These commands are tailored to the particular type of peripheral device.
- Test: Used to test various status conditions associated with an I/O module and its peripherals. The processor will want to know that the peripheral of interest is powered on and available for use. It will also want to know if the most recent I/O operation is completed and if any errors occurred.
- Read: Causes the I/O module to obtain an item of data from the peripheral and place it in an internal buffer (depicted as a data register in Figure 7.3).The processor can then obtain the data item by requesting that the I/O module place it on the data bus.
- Write: Causes the I/O module to take an item of data (byte or word) from the data bus and subsequently transmit that data item to the peripheral.

4.5 Interrupt-driven I/O

The problem with programmed I/O is that the processor has to wait a long time for the I/O module of concern to be ready for either reception or transmission of data. The processor, while waiting, must repeatedly interrogate the status of the I/O module. As a result, the level of the performance of the entire system is severely degraded.

An alternative is for the processor to issue an I/O command to a module and then go on to do some other useful work. The I/O module will then interrupt the processor to request service when it is ready to exchange data with the processor. The processor then executes the data transfer, as before, and then resumes its former processing.

Let us consider how this works, first from the point of view of the I/O module.

For input, the I/O module receives a READ command from the processor. The I/O module then proceeds to read data in from an associated peripheral. Once the data are in the module's data register, the module signals an interrupt to the processor over a control line. The module then waits until its data are requested by the processor. When the request is made, the module places its data on the data bus and is then ready for another I/O operation.

From the processor’s point of view, the action for input is as follows. The processor issues a READ command. It then goes off and does something else (e.g., the processor may be working on several different programs at the same time). At the end of each instruction cycle, the processor checks for interrupts (Figure 3.9).

When the interrupt from the I/O module occurs, the processor saves the context (e.g., program counter and processor registers) of the current program and processes the interrupt. In this case, the processor reads the word of data from the I/O module and stores it in memory. It then restores the context of the program it was working on (or some other program) and resumes execution.

Figure 7.4b shows the use of interrupt I/O for reading in a block of data. Compare this with Figure 7.4a. Interrupt I/O is more efficient than programmed I/O because it eliminates needless waiting. However, interrupt I/O still consumes a lot of processor time, because every word of data that goes from memory to I/O module or from I/O module to memory must pass through the processor.

4.5.1 Interrupt Processing

Let us consider the role of the processor in interrupt-driven I/O in more detail. The occurrence of an interrupt triggers a number of events, both in the processor hardware and in software. Figure 7.6 shows a typical sequence. When an I/O device completes an I/O operation, the following sequence of hardware events occurs:

1. The device issues an interrupt signal to the processor.

2. The processor finishes execution of the current instruction before responding to the interrupt, as indicated in Figure 3.9.

3. The processor tests for an interrupt, determines that there is one, and sends an acknowledgment signal to the device that issued the interrupt. The acknowledgment allows the device to remove its interrupt signal.

4. The processor now needs to prepare to transfer control to the interrupt routine. To begin, it needs to save information needed to resume the current program at the point of interrupt. The minimum information required is (a) the status of the processor, which is contained in a register called the program status word (PSW), and (b) the location of the next instruction to be executed, which is contained in the program counter. These can be pushed onto the system control stack.2

5. The processor now loads the program counter with the entry location of the interrupt-handling program that will respond to this interrupt. Depending on the computer architecture and operating system design, there may be a single program; one program for each type of interrupt; or one program for each device and each type of interrupt. If there is more than one interrupt-handling routine, the processor must determine which one to invoke. This information may have been included in the original interrupt signal, or the processor may have to issue a request to the device that issued the interrupt to get a response that contains the needed information.

Once the program counter has been loaded, the processor proceeds to the next instruction cycle, which begins with an instruction fetch. Because the instruction fetch is determined by the contents of the program counter, the result is that control is transferred to the interrupt-handler program. The execution of this program results in the following operations:

6. At this point, the program counter and PSW relating to the interrupted program have been saved on the system stack. However, there is other information that is considered part of the "state" of the executing program. In particular, the contents of the processor registers need to be saved, because these registers may be used by the interrupt handler. So, all of these values, plus any other state information, need to be saved. Typically, the interrupt handler will begin by saving the contents of all registers on the stack. Figure 7.7a shows a simple example. In this case, a user program is interrupted after the instruction at location N. The contents of all of the registers plus the address of the next instruction (N 1) are pushed onto the stack. The stack pointer is updated to point to the new top of stack, and the program counter is updated to point to the beginning of the interrupt service routine.

7. The interrupt handler next processes the interrupt. This includes an examination of status information relating to the I/O operation or other event that caused an interrupt. It may also involve sending additional commands or acknowledgments to the I/O device.

8. When interrupt processing is complete, the saved register values are retrieved from the stack and restored to the registers (e.g., see Figure 7.7b).

9. The final act is to restore the PSW and program counter values from the stack. As a result, the next instruction to be executed will be from the previously interrupted program.

Note that it is important to save all the state information about the interrupted program for later resumption. This is because the interrupt is not a routine called from the program. Rather, the interrupt can occur at any time and therefore at any point in the execution of a user program. Its occurrence is unpredictable. Indeed, as we will see in the next chapter, the two programs may not have anything in common and may belong to two different users.

4.6 Direct memory access (DMA)

Drawbacks of Programmed and Interrupt-Driven I/O

Interrupt-driven I/O, though more efficient than simple programmed I/O, still requires the active intervention of the processor to transfer data between memory and an I/O module, and any data transfer must traverse a path through the processor.

Thus, both these forms of I/O suffer from two inherent drawbacks:

i. The I/O transfer rate is limited by the speed with which the processor can test and service a device.
ii. The processor is tied up in managing an I/O transfer; a number of instructions must be executed for each I/O transfer (e.g., Figure 7.5).

There is somewhat of a trade-off between these two drawbacks. Consider the transfer of a block of data. Using simple programmed I/O, the processor is dedicated to the task of I/O and can move data at a rather high rate, at the cost of doing nothing else. Interrupt I/O frees up the processor to some extent at the expense of the I/O transfer rate. Nevertheless, both methods have an adverse impact on both processor activity and I/O transfer rate.

When large volumes of data are to be moved, a more efficient technique is required: direct memory access (DMA).

4.6.1 DMA Function

DMA involves an additional module on the system bus. The DMA module (Figure 7.11) is capable of mimicking the processor and, indeed, of taking over control of the system from the processor. It needs to do this to transfer data to and from memory over the system bus. For this purpose, the DMA module must use the bus only when the processor does not need it, or it must force the processor to suspend operation temporarily. The latter technique is more common and is referred to as cycle stealing, because the DMA module in effect steals a bus cycle.

When the processor wishes to read or write a block of data, it issues a command to the DMA module, by sending to the DMA module the following information:

- Whether a read or write is requested, using the read or write control line between the processor and the DMA module
- The address of the I/O device involved, communicated on the data lines
- The starting location in memory to read from or write to, communicated on the data lines and stored by the DMA module in its address register
- The number of words to be read or written, again communicated via the data lines and stored in the data count register

The processor then continues with other work. It has delegated this I/O operation to the DMA module. The DMA module transfers the entire block of data, one word at a time, directly to or from memory, without going through the processor. When the transfer is complete, the DMA module sends an interrupt signal to the processor. Thus, the processor is involved only at the beginning and end of the transfer (Figure 7.4c).

Figure 7.12 shows where in the instruction cycle the processor may be suspended.

In each case, the processor is suspended just before it needs to use the bus. The DMA module then transfers one word and returns control to the processor. Note that this is not an interrupt; the processor does not save a context and do something else. Rather, the processor pauses for one bus cycle. The overall effect is to cause the processor to execute more slowly. Nevertheless, for a multiple-word I/O transfer, DMA is far more efficient than interrupt-driven or programmed I/O.

The DMA mechanism can be configured in a variety of ways. Some possibilities are shown in Figure 7.13. In the first example, all modules share the same system bus. The DMA module, acting as a surrogate processor, uses programmed I/O to exchange data between memory and an I/O module through the DMA

4.7 Input/ Output

A simple computer system is assumed to have one input device and one output device. Both these functions could very well be performed by a terminal with a keyboard for input and a display or printer for the output.

The ALU and the control unit together form the central processing unit (CPU), which we will also refer to as the processor. In the programmed I/O scheme, during the execution of the Read a Word (RWD) instruction, the CPU commands the input device to send a data word and then waits. The input device gathers the data from the input medium and when the data is ready in its data buffer, informs the CPU that the data are ready. The CPU then gates the data into the ACC over the DIL s. During the Write a Word (WWD), the CPU gates the Accumulator (ACC) content onto Data Output Lines (DOLs), commands the output device to accept the data, and waits. When the data are gated into its data buffer, the output device informs the CPU of the data acceptance. The CPU then proceeds to execute the next instruction in the sequence.

A variety of I/O devices are used to communicate with computers. They can be broadly classified into the following categories:

i. Online devices such as terminals that communicate with the processor in an interactive mode
ii. Off-line devices such as printers that communicate with the processor in a non-interactive mode
iii. Devices that help in real-time data acquisition and transmission (analog-to-digital and digital-to-analog converters)
iv. Storage devices such as tapes and disks that can also be classified as I/O devices

We will provide very brief descriptions of selected devices in this section. This area of computer systems technology also changes very rapidly and newer and more versatile devices are announced on a daily basis. As such, the most up-to-date information on their characteristics is only available from vendor’s literature and magazines.

Terminals: A computer terminal is a remote electronic or electromechanical hardware device that is used for entering data into, and displaying data from, a computer or a computing system. A typical terminal consist s of a monitor, a keyboard, and a mouse. A wide variety of terminals are now available. The most common cathode ray tube (CRT)-based monitors are being rapidly replaced by flat panel displays. These displays use liquid crystal display (LCD), thin film transistor (TFT) and Plasma technologies. Displays can be either character mapped or bit mapped. Character-mapped monitors typically treat the display as a matrix of characters (bytes), while the bit-mapped monitors treat the display as an array of picture elements (pixels) that can be on or off, with each pixel depicting 1 bit of information. Display technology is experiencing rapid change with newer capabilities added to display devices almost daily.

Sophisticated alphanumeric and graphic displays are now commonly available. Touch-screen capability as an input mechanism is now common. This allows selection from a menu displayed on the screen, just by touching an item on the menu.

A variety of keyboards are now available. A typical keyboard used with personal computers consists of 102 keys. Various ergonomic keyboards are now appearing. The mouse allows pointing to any area on the screen by its movement on the mouse pad. The buttons are used to perform various operations based on the information at the selected spot on the screen. In addition to these three components of a typical terminal, various devices such as light pen, joystick, micro phones (for direct audio input), speakers (for audio output), and cameras (for video input) are commonly used I/O devices.

Mouse: The mouse is the most common input device today. It typically has two buttons and a scroll wheel. The scroll wheel allows the scrolling of the image on the screen and the buttons allow select ion of a particular position on the screen, as the mouse is moved on the mouse pad. The left button selects the cursor position and the right button typically provides the menu of possible operations at the selected cursor position. Earlier mouse used the position of a ball underneath the mouse to track its motion mechanically. It is now common to see optical mouse devices.

Apple Computer's wireless Mighty Mouse's tracking engine is based on laser technology that delivers 20 times the performance of standard optical tracking, giving more accuracy and responsiveness on more surfaces. It offers 3608 scrolling capability, perfectly positioned to roll smoothly under just one finger. Touch-sensitive technology employed under its seamless top shell detects where we are clicking. The force-sensing buttons on either side of Mighty Mouse let us squeeze the mouse to activate a whole host of customizable features instantly. It is available in wired and wireless versions.

Printers: Printers have come a long way from t he noisy daisy-w heel and dot-matrix printers of the 1980s. Today we can print any document with crisp, realistic colors and s harp text in essentially any font we can imagine. Whether we want to print photos, family projects, or documents on the go, there is a specially designed printer for the job. The most common type o f printer found in homes today is the inkjet printer. This printer works by spraying ionized ink onto the paper with magnetized plates directing the ink to the desired shape. Inkjet printers are capable of producing high-quality text and images in black and white or color, approaching the quality that is produced by more costly laser printers. Many inkjet printers today are capable of printing photo-quality images.

Laser printers provide the highest quality text and images. They operate by using a laser beam to produce an electrically charged image on a drum, which is then rolled through a reservoir of toner. The toner is picked up by the electrically charged portions of the drum, and transferred to the paper through a combination of heat and pressure. While full-color laser printers are available, they tend to be much more expensive than black and white versions and require a great deal of printer memory to produce high-resolution images. A portable printer is a compact mobile inkjet printer that can fit in a briefcase, weigh very little, and run on battery power.

Infrared-compatible (wireless) printers, available in both inkjet and LaserJet models, allow us printing from a handheld device, laptop computer, or digital camera. The w ireless short-range radio technology that allows this to happen is called Bluetooth. All-in-one devices (inkjet or laser-based) that combine the functions of printer, scanner, copier, and fax into one machine are now available.

As an example, the HP Color LaserJet 1600 family is a basic color laser printer designed for light printing needs. It has a 264 MHz processor and 16 MB of memory.

Scanners: Scanners are used to transfer the content of a hard copy document to the machine memory. The image can be retained as is and transmitted as needed. It can also be processed (reduced, enlarge, rotate, etc.). If the image is to be treated as a word-process-able document, optical character recognition is first performed after scanning it. Scanners of various capabilities and price ranges are now available. The MicrotekScanMaker i900 is a flatbed scanner that can handle legal-size originals and features a dual-scanning bed that produces the best film scans. It is connected to the processor via USB or FireWire. It employs Digital ICE Photo Print technology to correct dust and scratches and ColorRescue for color-balance corrections to film and reflective scans. In addition to the typical flatbed glass plate, i900has glassless film scanner with the glass scan surface underneath. By eliminating the glass, the way a standalone film scanner does, the i900 can capture more tonal information from film—from light shades to deep shadows. The scanner’s optical resolution is 3200 by 6400 dots per inch (dpi).

Tapes and Disks: Storage devices such as magnetic tapes (re el-to-re el, cassette, and streaming cartridge) and disks (hard and floppy, magnetic and optical) are also used as I/O devices. The description provided in this section is intentionally very brief. The reader should refer to vendors’ literature for current information on these and other devices.

5 Week 9/10: data communications

- Communication between computers at the physical level.
- Networks and computers.

Week 11: CAT 2

Week 12: Revision and Course Overview

Week 15/16: end of semester exams