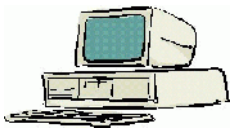
INTRODUCTION TO COMPUTER AND MICROCOMPUTERS



What is a Computer?

An electronic device that accepts input, stores large quantities of data, execute complex instructions which direct it to perform mathematical and logical operations and outputs the answers in a human readable form.

Computers are not very intelligent devices, but they handle instructions flawlessly and fast. They must follow explicit directions from both the user and computer programmer. Computers are really nothing more than a very powerful calculator with some great accessories. Applications like word processing and games are just a very complex math problem.

Computer Generations

From the 1950's, the computer age took off in full force. The years since then have been divided into periods or generations based on the technology used.

1. First Generation Computers (1945-1954): Vacuum Tubes

These machines were used in business for accounting and payroll applications. Valves were unreliable components generating a lot of heat. They had very limited memory capacity. Magnetic drums were developed to store information and tapes were also developed for secondary storage. They were initially programmed in machine language (binary). A major breakthrough was the development of assemblers and assembly language.

2. Second Generation (1955-1964): Transistors

The development of the transistor revolutionized the development of computers. Invented at Bell Labs in 1948, transistors were much smaller, more rugged, cheaper to make and far more reliable than valves. Core memory was introduced and disk storage was also used. The hardware became smaller and more reliable, a trend that stills continues. Another major feature of the second generation

was the use of high-level programming languages such as Fortran and Cobol. These revolutionized the development of software for computers.

3. Third Generation (1965-1974): Integrated Circuits (ICs)

IC's were again smaller, cheaper, faster and more reliable than transistors. Speeds went from the microsecond to the nanosecond (billionth) to the picosecond (trillionth) range. ICs were used for main memory despite the disadvantage of being volatile. Minicomputers were developed at this time. Terminals replaced punched cards for data entry and disk packs became popular for secondary storage. IBM introduced the idea of a compatible family of computers, 360 family easing the problem of upgrading to a more powerful machine. Operating systems were developed to manage and share the computing resources and time-sharing operating systems were developed. These greatly improved the efficiency of computers. Computers had by now pervaded most areas of business and administration. The number of transistors that be fabricated on a chip is referred to as the scale of integration (SI). Early chips had SSI (small SI) of tens to a few hundreds. Later chips were MSI (Medium SI): hundreds to a few thousands. Then came LSI chips (Large SI) in the thousands range.

4. Fourth Generation (1975-1984): VLSI (Very Large SI)

The term fourth generation is occasionally applied to VLSI-based computer architecture. VLSI has made it possible to fabricate an entire CPU, main memory, or similar devices with a single IC. This has resulted in new classes of machines such as inexpensive personal computers, and high- performance parallel processors that contain thousands of CPUs. VLSI allowed the equivalent of tens of thousand of transistors to be incorporated on a single chip. This led to the development of the microprocessor a processor on a chip. Intel produced the 4004 which was followed by the 8008,8080, 8088 and 8086 etc. Other companies developing microprocessors included Motorolla (6800, 68000), and Zilog. Personal computers were developed and IBM launched the IBM PC based on the 8088 and 8086 microprocessors. Mainframe computers have grown in power. Memory chips are in the megabyte range. VLSI chips had enough transistors to build 20 ENIACs. Secondary storage has also evolved at fantastic rates with storage devices holding gigabytes (1024Mb = 1 Gb) of data. On the software side, more powerful operating systems are available such as Unix. Applications software has become cheaper and easier to use.

5. Fifth Generation (1991-Present):

The race is now on building the next or "fifth" generation of computers, machine that exhibit artificial intelligence (AI). Thus new generations of computers will involve robotics and computer networks. Developments are still continuing. Computers are becoming faster, smaller and cheaper. Storage units are increasing in capacity. Distributed computing is becoming popular and parallel computers with large numbers of CPUs have been built.

Generation	Technology & Architecture	Software & Applications	Systems
First (1945-54)	Vacuum tubes, Relay memories, CPU driven by PC and accumulator; fixed point Arithmetic	Machine & Assembly language, Single user Basic I/O using programmed and Internet mode.	ENIAC TIFRAC IBM 701 Princeton IAS
Second (1955-64)	Discrete Transistors, Core Memories, Floating point, Arithmetic I/O, processors, Multiplexed memory access	HLL used with compilers, batch processing, Monitoring, Libraries	IBM7099 CDC 1604
Third (1965-74)	Integrated circuits, Microprogramming, Pipelining, Caching, Lookahead Processing	Multiprogramming, Time sharing OS, Multi-user applications	IBM 360/700 CDC 6000 TA-ASC PDP-8
Fourth (1975-84)	LSI/VLSI and Semiconductor memory, Microprocessors technology, Multiprocessors, vector super-computing, multi computer	Multiprocessor OS, languages, Compilers	VAX 9800, Cray X-MP, IBM 3600, Pentium Processor based systems (PCs), UltraSPARC etc.
Fifth (1991-present)	VLSI/VHSIC processors, scalable architecture	Massively parallel processing, Grand challenge Applications	Cray/MPP, TMC/CM-5, Intel paragon, Fujitsu VP500

Types of Computers

Computer now comes in a variety of shapes and sizes, which could be roughly classified according to their processing power into five sizes: *super large, large, medium, small,* and *tiny.*

Microcomputers are the type of computers that we are most likely to notice and use in our everyday life. In fact there are other types of computers that you may use directly or indirectly:

- ***Supercomputers-super large computers:*** super*computers* are high- capacity machines with hundreds of thousands of processors that can perform more than 1 trillion calculations per second. These are the most expensive but fastest computers available. "Supers," as they are called, have been used for tasks requiring the processing of enormous volumes of data, such as doing the U.S. census count, forecasting weather, designing aircraft, modeling molecules, breaking codes, and simulating explosion of nuclear bombs.
- ***Mainframe computers - large computers:*** The only type of computer available until the late 1960s, *mainframes* are water- or air-cooled computers that vary in size from small, to medium, to large, depending on their use. Small mainframes are often called *midsize computers;* they used to be called *minicomputers.* Mainframes are used by large organizations such as banks, airlines, insurance companies, and colleges-for processing millions of transactions. Often users access a mainframe using a *terminal,* which has a display screen and a keyboard and can input and output data but cannot by itself process data.
- ***Workstations - medium computer:*** Introduced in the early 1980s, *workstations,* are expensive, powerful computers usually used for complex scientific, mathematical, and engineering calculations and for computer-aided design and computer-aided manufacturing. Providing many capabilities comparable to midsize mainframes, workstations are used for such tasks as designing airplane fuselages, prescription drugs, and movie special effects. Workstations have caught the eye of the public mainly for their graphics capabilities, which are used to breathe three-dimensional life into movies such as *Jurassic Park* and *Titanic.* The capabilities of low-end workstations overlap those of high-end desktop microcomputers.
- ***Microcomputer - small computers:*** *Microcomputers,* also called *personal computers (PC),* can fit next to a desk or on a desktop, or can be carried around. They are either stand-alone machines or are connected to a computer network, such as a local area network. *A local area network (LAN)* connects, usually by special cable, a group of desktop PCs and other devices, such as printers, in an office or a building. Microcomputers are of several types:
  - ***Desktop PCs:*** are those in which the case or main housing sits on a

desk, with keyboard in front and monitor (screen) often on top.

- ***Tower PCs:*** are those Microcomputer in which the case sits as a "tower," often on the floor beside a desk, thus freeing up desk surface space.
- ***Laptop computers*** *(*also called *notebook computers):* are lightweight portable computers with built-in monitor, keyboard, hard-disk drive, battery, and AC adapter that can be plugged into an electrical outlet; they weigh anywhere from 1.8 to 9 pounds.
- ***Personal digital assistants (PDAs)*** *(*also called *handheld computers* or *palmtops)* combine personal organization tools-schedule planners, address books, to-do lists. Some are able to send e-mail and faxes. Some PDAs have touch-sensitive screens. Some also connect to desktop computers for sending or receiving information.
- **Microcontrollers-tiny computers:** Microcontrollers, also called embedded computers, are the tiny, specialized microprocessors installed in "smart" appliances and automobiles. These microcontrollers enable PDAs microwave ovens, for example, to store data about how long to cook your potatoes and at what temperature.

Basic Blocks of a Microcomputer

If we think of the computer as an information manipulation device the basic components of a microcomputer are:

Input Units -- *"How to tell it what to do"*

Devices allow us to enter information into the computer. A keyboard and mouse are the standard way to interact with the computer. Other devices include mice, scanners, microphones, joysticks and game pads used primarly for games.

Output Units -- *"How it shows you what it is doing"*

Devices are how the manipulated information is returned to us. They commonly include video monitors, printers, and speakers.

Memory -- "How the processor stores and uses immediate data"

When you use a program, the computer loads a portion of the program from the hard drive to the much faster memory (RAM). When you "save" your work or quit the program, the data gets written back to the hard drive.

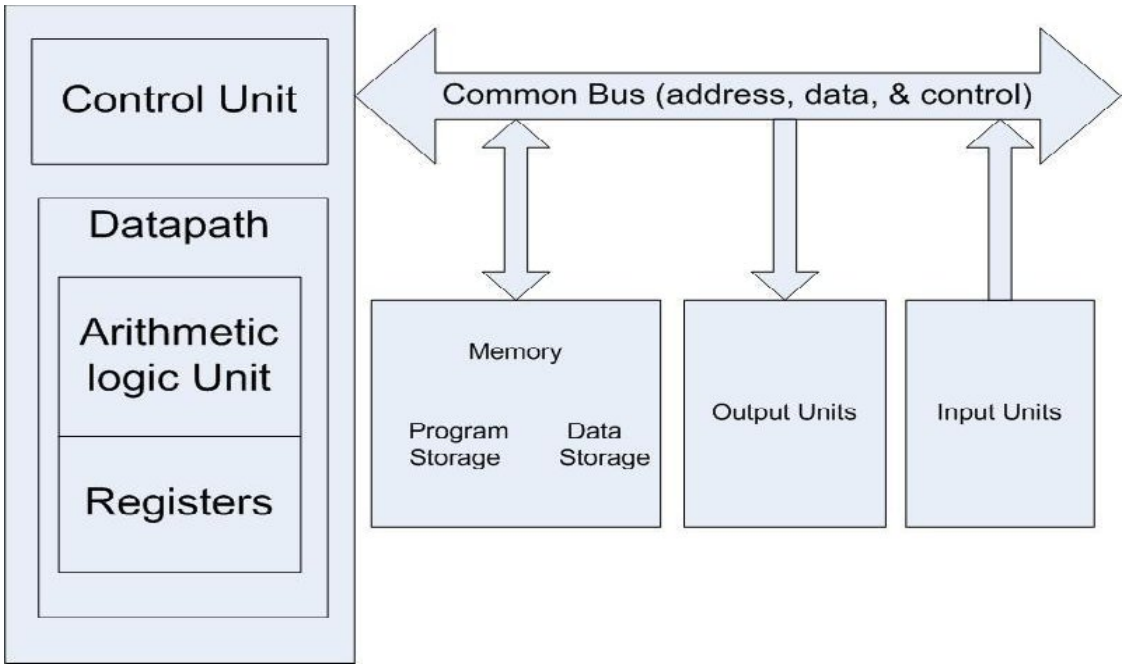


Fig. (1): Basic Block of a Microcomputer

Processor – Central Processing Unit

- **Datapath** - consists of register file and ALU
  - Register is a storage location. Used to hold data or a memory address during execution of an instruction
  - ALU receives data from main memory and/or register file, performs computations and writes result back to main memory or registers
- **Control unit** – decodes and monitors the execution of instructions and also acts as an arbiter while various systems compete for resources of CPU.

Computer Architecture

In computer engineering, **computer architecture** is the conceptual design and fundamental operational structure of a computer system. It is a blueprint and functional description of requirements (especially speeds and interconnections) and design implementations for the various parts of a computer — focusing largely on the way by which the central processing unit (CPU) performs internally and accesses addresses in memory.

Computer architecture comprises at least three main subcategories

- ***Instruction set architecture*, or ISA**, is the abstract image of a computing system that is seen by a machine language (or assembly language) programmer, including the instruction set, memory address modes, processor registers, and address and data formats.
- ***Microarchitecture***, also known as *Computer organization* is a lower level, more concrete, description of the system that involves how the constituent parts of the system are interconnected and how they interoperate in order to implement the ISA. The size of a computer's cache for instance, is an organizational issue that generally has nothing to do with the ISA.
- ***System Design*** which includes all of the other hardware components within a computing system such as:
  - system interconnects such as computer buses and switches
  - memory controllers and hierarchies
  - CPU off-load mechanisms such as direct memory access issues like multi-processing.

Once both ISA and microarchitecture has been specified, the actual device needs to be designed into hardware. This design process is often called *implementation*. Implementation is usually not considered architectural definition, but rather hardware design engineering.
Computer Organization deals with the advances in computer architecture right from the Von Neumann machines to the current day super scalar architectures.

Von Neumann Architecture

The earliest computing machines had fixed programs. Some very simple computers still use this design, either for simplicity or training purposes. For example, a desk calculator (in principle) is a fixed program computer. It can do basic mathematics, but it cannot be used as a word processor or to run video games. To change the program of such a machine, you have to re-wire, re-structure, or even re-design the machine. Indeed, the earliest computers were not so much "programmed"

as they were "designed". "Reprogramming", when it was possible at all, was a very manual process, starting with flow charts and paper notes, followed by detailed engineering designs, and then the often-arduous process of implementing the physical changes.

The idea of the stored-program computer changed all that. By creating an instruction set architecture and detailing the computation as a series of instructions (the program), the machine becomes much more flexible. By treating those instructions in the same way as data, a stored-program machine can easily change the program, and can do so under program control.

The **von Neumann architecture** is a computer design model that uses a processing unit and a single separate storage structure to hold both instructions and data as shown in Fig. (2). It is named after mathematician and early computer scientist John von Neumann. Such a computer implements a universal Turing machine, and the common "referential model" of specifying sequential architectures, in contrast with parallel architectures. The term "stored-program computer" is generally used to mean a computer of this design, although as modern computers are usually of this type, the term has fallen into disuse. All general-purpose computers are now based on the key concepts of the von Neumann architecture.

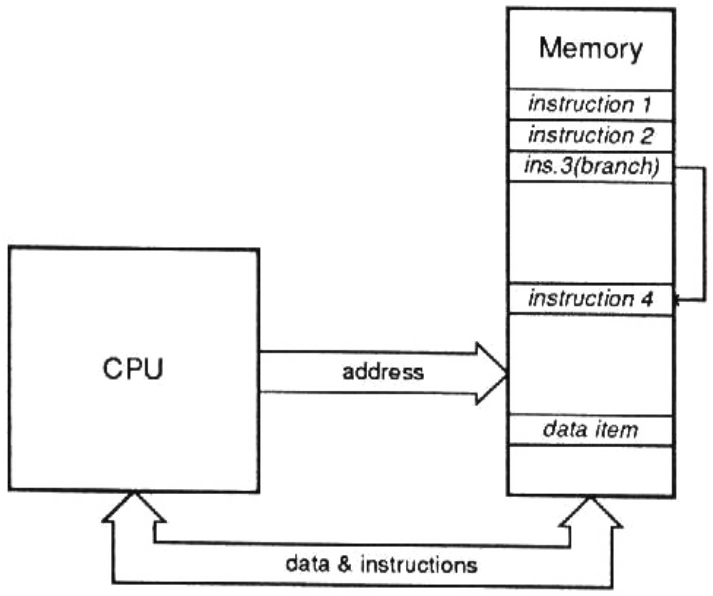


Fig.(2): The Von-Neumann Architecture

Though the von Neumann model is universal in general-purpose computing, it suffers from one obvious problem. All information (instructions and data) must flow back and forth between the processor and memory through a single channel, and this channel will have finite bandwidth. When this bandwidth is fully used the processor can go no faster. This performance limiting factor is called the *von Neumann bottleneck.*

Hardvard Architecture

A *Harvard Architecture* as shown in Fig. (3) has one memory for instructions and a second for data. The name comes from the Harvard Mark 1, an electromechanical computer which pre-dates the stored-program concept of von Neumann, as does the architecture in this form. It is still used for applications which run fixed programs, in areas such as digital signal processing, but not for general-purpose computing. The advantage is the increased bandwidth available due to having separate communication channels for instructions and data; the disadvantage is that the storage is allocated to code and data in a fixed ratio.

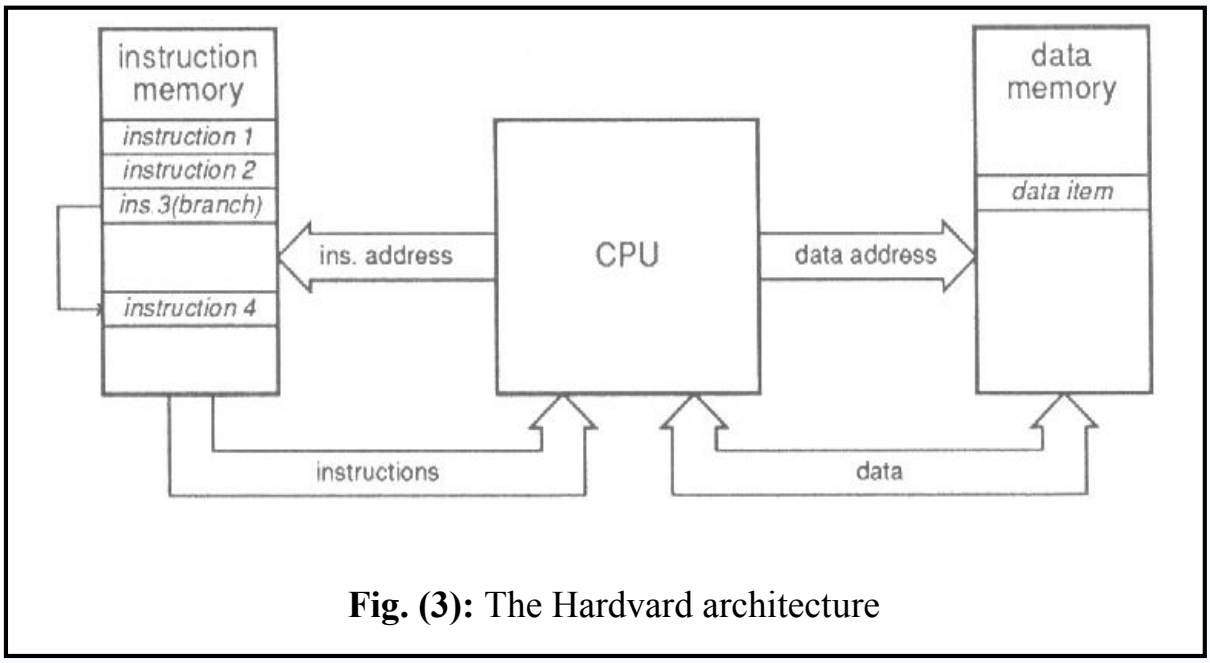


Fig. (3): The Hardvard architecture

In Harvard architecture, there is no need to make the two memories share characteristics. In particular, the word width, timing, implementation technology, and memory address structure can differ. Instruction memory is often wider than data memory. In some systems, instructions can be stored in read-only memory while data memory generally requires read-write memory. In some systems, there is much more instruction memory than data memory so instruction addresses are much wider than data addresses.

A pure Harvard architecture computer suffers from the disadvantage that mechanisms must be provided to separately load the program to be executed into instruction memory and any data to be operated upon into data memory. Additionally, modern Harvard architecture machines often use a read-only technology for the instruction memory and read/write technology for the data memory. This allows the computer to begin execution of a pre-loaded program as soon as power is applied. The data memory will at this time be in an unknown state, so it is not possible to provide any kind of pre-defined data values to the program.

The solution is to provide a hardware pathway and machine language instructions so that the contents of the instruction memory can be read as if they were data. Initial data values can then be copied from the instruction memory into the data memory when the program starts. If the data is not to be modified (for example, if it is a constant value, such as pi, or a text string), it can be accessed by the running program directly from instruction memory without taking up space in data memory (which is often at a premium).

For instance each port may be supplied from its own local cache memory (fig. (4)). The cache memories reduce the external bandwidth requirements sufficiently to allow them both to be connected to the same main memory, giving the bandwidth advantage of a Harvard architecture along with most of the flexibility of the simple von Neumann architecture. (The flexibility may be somewhat reduced because of cache consistency problems with self-modifying code). Note that this type of Harvard architecture is still a von Neumann machine.

The Modified Harvard architecture is very like the Harvard architecture but provides a pathway between the instruction memory and the CPU that allows words from the instruction memory to be treated as read-only data. This allows constant data, particularly text strings, to be accessed without first having to be copied into data memory, thus preserving more data memory for read/write variables. Special machine language instructions are provided to read data from the instruction memory.

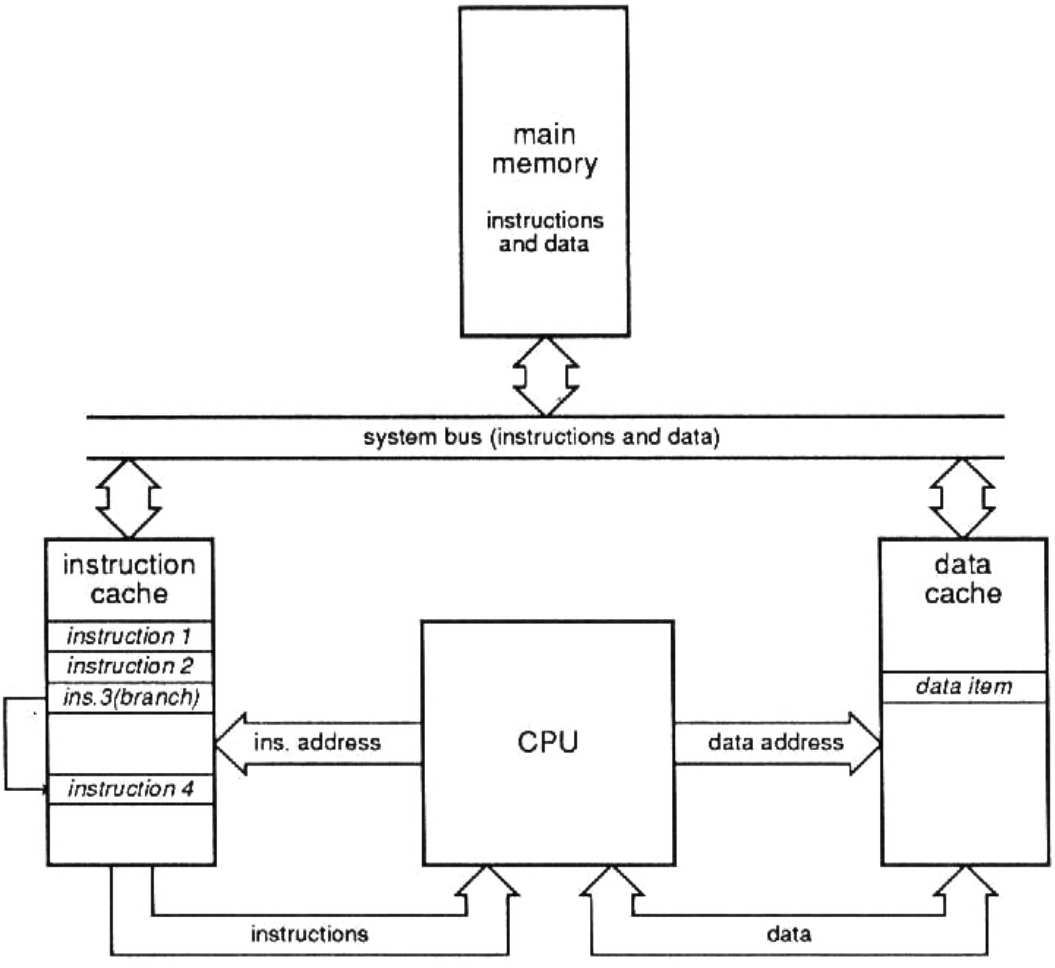
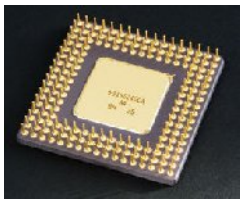
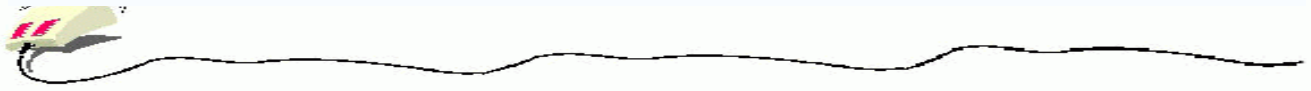


Fig. (4): A modified Harvard Architecture



2.1 Microprocessor

A microprocessor is a computer processor on a microchip. It's sometimes called a logic chip. It is the "engine" that goes into motion when you turn your computer on. A microprocessor is designed to perform arithmetic and logic operations that make use of small number-holding areas called registers. Typical microprocessor operations include adding, subtracting, comparing two numbers, and fetching numbers from one area to another. These operations are the result of a set of instructions that are part of the microprocessor design. When the computer is turned on, the microprocessor is designed to get the first instruction from the basic input/output system (BIOS) that comes with the computer as part of its memory. After that, either the BIOS, or the operating system that BIOS loads into computer memory, or an application program is "driving" the microprocessor, giving it instructions to perform.



2.2 Evaluation of the Microprocessors

The evolution of microprocessors has been known to follow Moore's Law when it comes to steadily increasing performance over the years. This law suggests that ***the complexity of an integrated circuit, with respect to minimum component cost, doubles every 18 months***. This dictum has generally proven true since the early 1970s. From their humble beginnings as the drivers for calculators, the continued increase in power has led to the dominance of microprocessors over every other form of computer; every system from the largest mainframes to the smallest handheld computers now uses a microprocessor at its core.

- Intel 4004 was a 4 bit up. Only 45 instructions P Channel Mosfet technology. 50 K instructions per second (ENIAC).
- Later 8008 as an 8 bit µ processor then 8080 and Motorolla 6800.
- 8080 was 10x faster than 8008 and TTL compatible.
- MITS Altair 8800 in 1974. The BASIC Interpreter was written by Bill Gates. Assembler program was written by Digital Research Corporation.
- In 1977, 8085 microprocessor. Internal clock generator, higher frequency at reduced cost and integration. There are 200 million 8085's around the world.

- In 1978, 8086+8088 microprocessors 16 bit. Addressed 1 Mbyte of memory. Small instruction cache (4-6 bytes) enabled prefetch of instructions.
- IBM decided to use 8088 in PC.
- In 1983, 80286 released, identical to 8086 except the addressing and higher clock speed.
- 32 bit microprocessor: In 1986 major overhaul on 80286 architecture, 80386 DX with 32bit data + 32 bit address (4 G bytes).
- In 1989, 80486 = 80386 + 80387 co processor + 8KB cache.
- In 1993, Pentium (80586). Includes 2 execution engines.
- Pentium Pro included 256K Level 2 cache mechanism as well as Level 1 cache. Also 3 execution engines which can execute at the same time and can conflict and still execute in parallel. The address bus was expended to 36.
- Pentium 2 included L2 cache on its circuit board (called slot)
- Later Pentium 3 and 4 released with several architectural and technological innovations.

2.3 Block diagram of a simple CPU

As mentioned before, the microprocessor is the CPU of the microcomputer. Therefore, the power of the microcomputer is determined by the capabilities of the microprocessor. Its clock frequency determines the speed of the microcomputer. The number of data and address pins on the microprocessor chip make up the microcomputer's word size and maximum memory size. The microcomputer's I/O and interfacing capabilities are determined by the control pins on the microprocessor chip.

The logic inside the microprocessor chip can be divided into three main areas: the register section, the control unit, and the Arithmetic and Logic Unit (ALU). A microprocessor chip with the above three sections is shown in Fig. (2.1).

A typical CPU has three major components: (1) register set, (2) arithmetic logic unit (ALU), and (3) control unit (CU). The register set differs from one computer architecture to another. It is usually a combination of general-purpose and special-purpose registers. General-purpose registers are used for any purpose, hence the name general purpose. Special-purpose registers have specific functions within the CPU. For example, the program counter (PC) is a special-purpose register that is used to hold the address of the instruction to be executed next. Another example of special-purpose registers is the instruction register (IR), which is used to hold the instruction that is currently executed. The ALU provides the circuitry needed to perform the arithmetic, logical and shift operations demanded of the instruction set. In Chapter 4,

we have covered a number of arithmetic operations and circuits used to support computation in an ALU. The control unit is the entity responsible for fetching the instruction to be executed from the main memory and decoding and then executing it. Figure (2.1) shows the main components of the CPU and its interactions with the memory system and the input/output devices.

The CPU fetches instructions from memory, reads and writes data from and to memory, and transfers data from and to input/output devices. A typical and simple execution cycle can be summarized as follows:

1. The next instruction to be executed, whose address is obtained from the PC, is fetched from the memory and stored in the IR.
2. The instruction is decoded.
3. Operands are fetched from the memory and stored in CPU registers, if needed.
4. The instruction is executed.
5. Results are transferred from CPU registers to the memory, if needed.

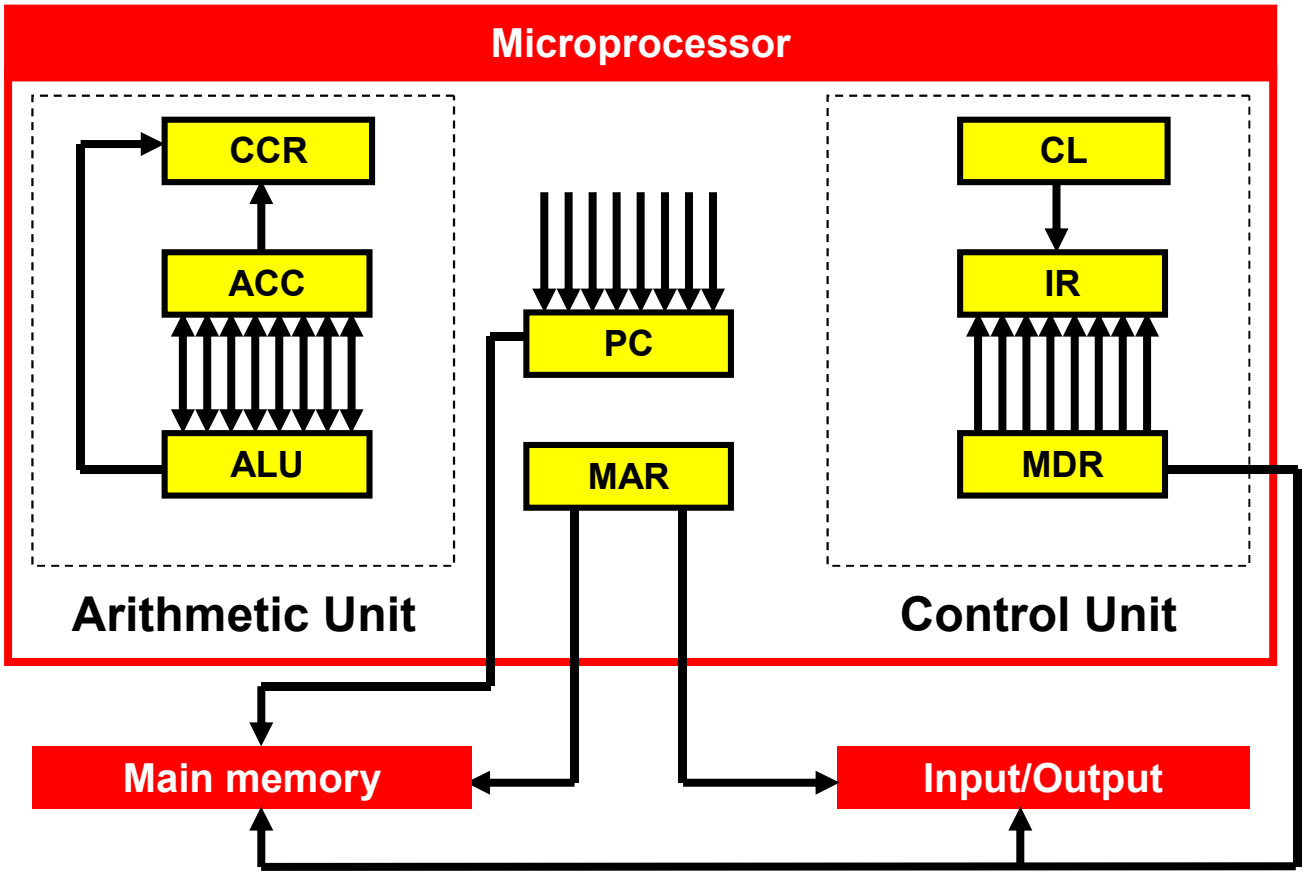


Fig. (2.1): Central processing unit main components and interactions with the memory and I/O.

The execution cycle is repeated as long as there are more instructions to execute. A check for pending interrupts is usually included in the cycle. Examples of interrupts include I/O device request, arithmetic overflow, or a page fault.
When an interrupt request is encountered, a transfer to an interrupt handling routine takes place. Interrupt handling routines are programs that are invoked to collect the state of the currently executing program, correct the cause of the interrupt, and restore the state of the program. The actions of the CPU during an execution cycle are defined by micro-orders issued by the control unit. These micro-orders are individual control signals sent over dedicated control lines. For example, let us assume that we want to execute an instruction that moves the contents of register X to register Y. Let us also assume that both registers are connected to the data bus, D. The control unit will issue a control signal to tell register X to place its contents on the data bus D. After some delay, another control signal will be sent to tell register Y to read from data bus D. The activation of the control signals is determined using either hardwired control or microprogramming. These concepts are explained later.

2.4 Register Set

The number, size, and types of registers vary from one microprocessor to another. However, the various registers in all microprocessors carry out similar operations. The register structures of microprocessors play a major role in designing the microprocessor architectures. Also, the register structures for a specific microprocessor determine how convenient and easy it is to program this microprocessor.

2.4.1. Memory Access Registers

Two registers are essential in memory write and read operations: the memory data register (MDR) and memory address register (MAR). The MDR and MAR are used exclusively by the CPU and are not directly accessible to programmers. In order to perform a write operation into a specified memory location, the MDR and MAR are used as follows:

1. The word to be stored into the memory location is frst loaded by the CPU intoMDR.
2. The address of the location into which the word is to be stored is loaded by the CPU into a MAR.
3. A write signal is issued by the CPU.

Similarly, to perform a memory read operation, the MDR and MAR are used as follows:

1. The address of the location from which the word is to be read is loaded into the MAR.
2. A read signal is issued by the CPU.
3. The required word will be loaded by the memory into the MDR ready for use by the CPU.

2.4.2. Instruction Fetching Registers

Two main registers are involved in fetching an instruction for execution: the program counter (PC) and the instruction register (IR). The PC is the register that contains the address of the next instruction to be fetched. The fetched instruction is loaded in the IR for execution. After a successful instruction fetch, the PC is updated to point to the next instruction to be executed. In the case of a branch operation, the PC is updated to point to the branch target instruction after the branch is resolved, that is, the target address is known.

2.4.3. Condition Registers

Condition registers, or flags, are used to maintain status information. Some architectures contain a special program status word (PSW) register. The PSW contains bits that are set by the CPU to indicate the current status of an executing program. These indicators are typically for arithmetic operations, interrupts, memory protection information, or processor status.

2.4.4. Special-Purpose Address Registers

1. Index Register:

The address of the operand is obtained by adding a constant to the content of a register, called the index register. The index register holds an address displacement. Index addressing is indicated in the instruction by including the name of the index register in parentheses and using the symbol X to indicate the constant to be added.

2. Segment Pointers:

In order to support segmentation, the address issued by the processor should consist of a segment number (base) and a displacement (or an offset) within the segment. A segment register holds the address of the base of the segment.

3. Stack Pointer:

A stack is a data organization mechanism in which the last data item stored is the first data item retrieved. Two specific operations can be performed on a stack. These are the Push and the Pop operations. A specific register, called the stack pointer (SP), is used to indicate the stack locationm that can be addressed. In the stack push operation, the SP value is used to indicate the location (called the top of the stack). After storing (pushing) this value, the SP is incremented (in some architectures, e.g. X86, the SP is decremented as the stack grows low in memory).

80386 Registers

The Intel basic programming model of the 386, 486, and the Pentium consists of three register groups. These are the general-purpose registers, the segment registers, and the instruction pointer (program counter) and the flag register. Fig. (2.2) shows the three sets of registers. The first set consists of general purpose registers A, B, C, D, SI (source index), DI (destination index), SP (stack pointer), and BP (base pointer). The second set of registers consists of CS (code segment), SS (stack segment), and four data segment registers DS, ES, FS, and GS. The third set of registers consists of the instruction pointer (program counter) and the flags (status) register. Among the status bits, the first five are identical to those bits introduced as early as in the 8085 8-bit microprocessor. The next 6 – 11 bits are identical to those introduced in the 8086. The flags in the bits 12 – 14 were introduced in the 80286 while the 16 – 17 bits were introduced in the 80386. The flag in bit 18 was introduced in the 80486.

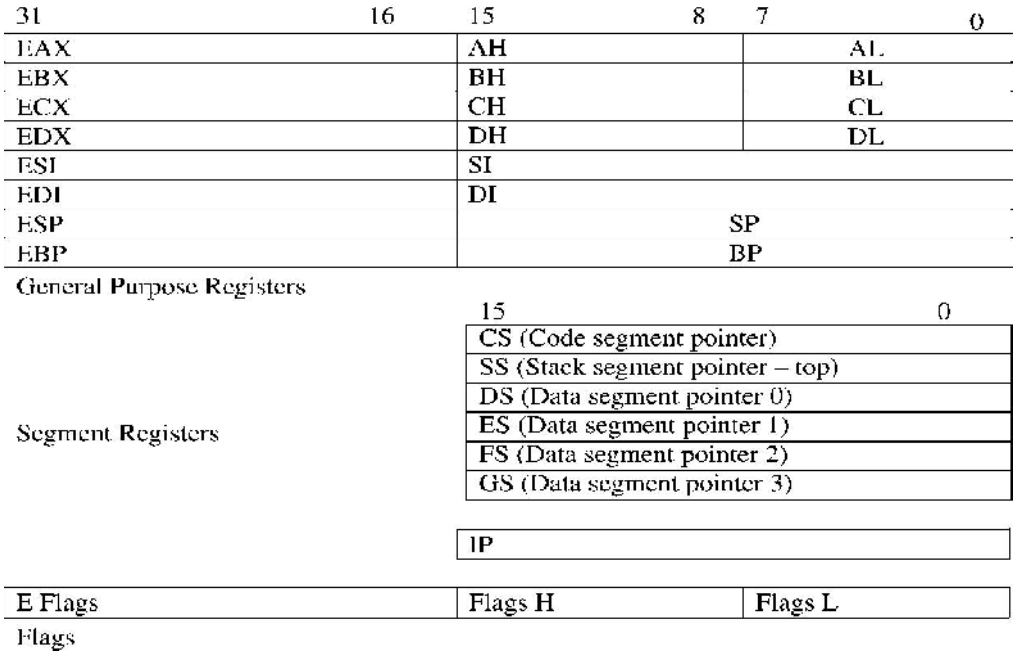


Fig. (2.2): The main register sets in 80X86 (80386 and above extended all 16 bit registers except segment registers).

2.5 Arithmetic Logic Unit

In computing, an arithmetic logic unit (ALU) is a digital circuit that performs arithmetic and logical operations. The ALU is a fundamental building block of the central processing unit of a computer, and even the simplest microprocessors contain one for purposes such as maintaining timers. The processors found inside modern CPUs and GPUs have inside them very powerful and very complex ALUs; a single component may contain a number of ALUs.

An ALU must process numbers using the same format as the rest of the digital circuit. For modern processors, that almost always is the two's complement binary number representation. Early computers used a wide variety of number systems, including one's complement, sign-magnitude format, and even true decimal systems, with ten tubes per digit.

Most of the computer's actions are performed by the ALU. The ALU gets data from processor registers. This data is processed and the results of this operation are stored into ALU output registers. Other mechanisms move data between these registers and memory. Most ALUs can perform the following operations:

- Integer arithmetic operations (addition, subtraction, and sometime multiplication and division, though this is more expensive)
- Bitwise logic operations (AND, NOT, OR, XOR)
- Bit-shifting operations (shifting or rotating a word by a specified number of bits to the left or right, with or without sign extension). Shifts can be interpreted as multiplications by 2 and divisions by 2.

2.6 Datapath

The CPU can be divided into a data section and a control section. The data section, which is also called the datapath, contains the registers and the ALU. The datapath is capable of performing certain operations on data items. The control section is basically the control unit, which issues control signals to the datapath. Internal to the CPU, data move from one register to another and between ALU and registers.

Internal data movements are performed via local buses, which may carry data, instructions, and addresses. Externally, data move from registers to memory and I/O devices, often by means of a system bus. Internal data movement among registers and

between the ALU and registers may be carried out using different organizations including one-bus, two-bus, or three-bus organizations. Dedicated datapaths may also be used between components that transfer data between themselves more frequently. For example, the contents of the PC are transferred to the MAR to fetch a new instruction at the beginning of each instruction cycle. Hence, a dedicated datapath from the PC to the MAR could be useful in speeding up this part of instruction execution.

2.6.1. One-Bus Organization

Using one bus, the CPU registers and the ALU use a single bus to move outgoing and incoming data. Since a bus can handle only a single data movement within one clock cycle, two-operand operations will need two cycles to fetch the operands for the ALU. Additional registers may also be needed to buffer data for the ALU.

This bus organization is the simplest and least expensive, but it limits the amount of data transfer that can be done in the same clock cycle, which will slow down the overall performance. Figure (2.3) shows a one-bus datapath consisting of a set of general-purpose registers, a memory address register (MAR), a memory data register (MDR), an instruction register (IR), a program counter (PC), and an ALU.

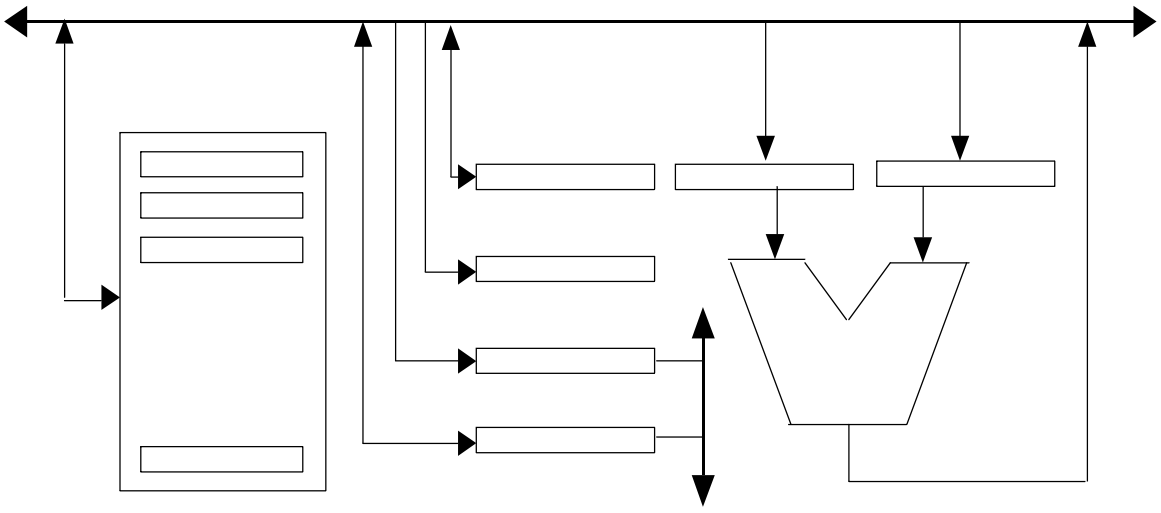
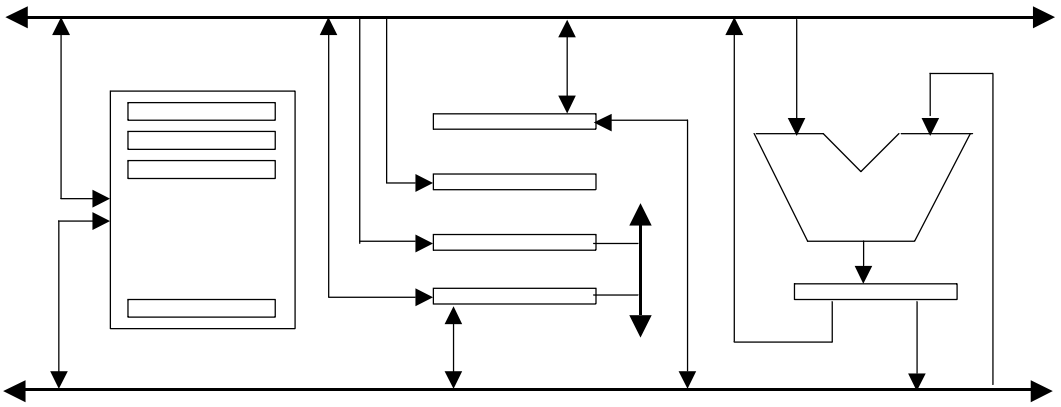


Fig. (2.3): One-Bus Datapath.

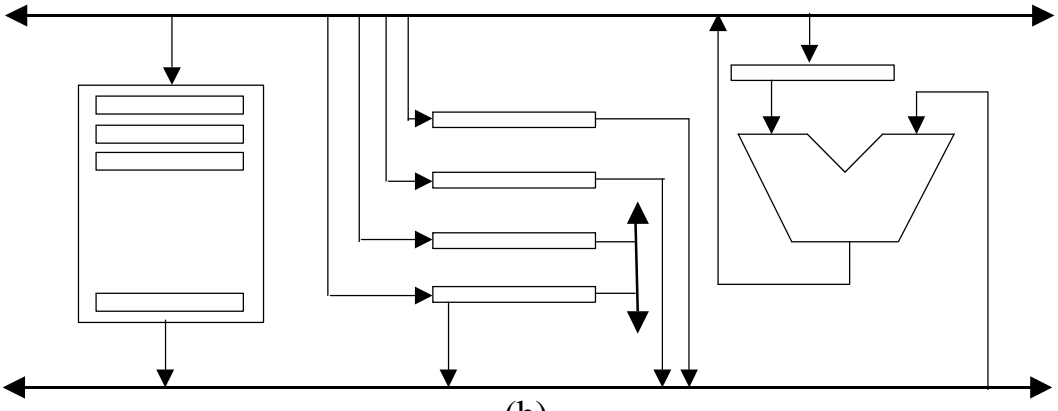
2.6.2. Two-Bus Organization

Using two buses is a faster solution than the one-bus organization. In this case, general-purpose registers are connected to both buses. Data can be transferred from

two different registers to the input point of the ALU at the same time. Therefore, a two-operand operation can fetch both operands in the same clock cycle. An additional buffer register may be needed to hold the output of the ALU when the two buses are busy carrying the two operands. Figure (2.4a) shows a two-bus organization. In some cases, one of the buses may be dedicated for moving data into registers (in-bus), while the other is dedicated for transferring data out of the registers (out-bus). In this case, the additional buffer register may be used, as one of the ALU inputs, to hold one of the operands. The ALU output can be connected directly to the in-bus, which will transfer the result into one of the registers. Figure (2.4b) shows a two-bus organization with in-bus and out-bus.



(a)



(b)

Fig. (2.4): Two-bus organizations. (a) An Example of Two-Bus Datapath. (b) Another Example of Two-Bus Datapath with in-bus and out-bus

2.6.3. Three-Bus Organization

In a three-bus organization, two buses may be used as source buses while the third is used as destination. The source buses move data out of registers (out-bus), and the destination bus may move data into a register (in-bus). Each of the two out-buses is connected to an ALU input point. The output of the ALU is connected directly to the in-bus. As can be expected, the more buses we have, the more data we can move within a single clock cycle. However, increasing the number of buses will also increase the complexity of the hardware. Figure (2.5) shows an example of a three-bus datapath.

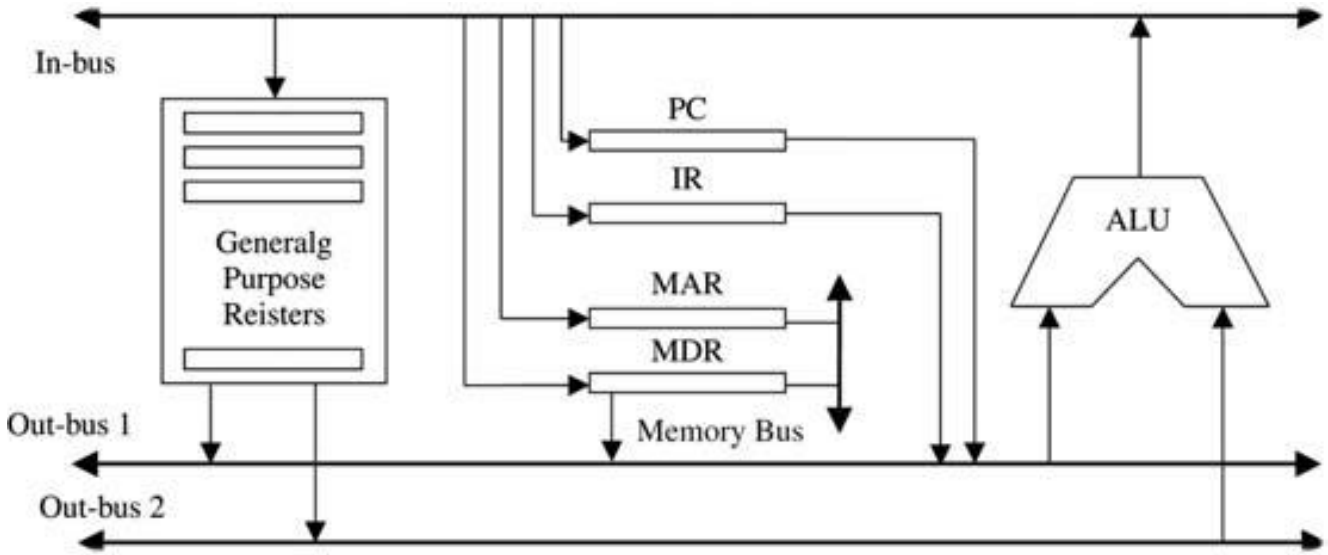


Fig. (2.5): Three-Bus Datapath.

3.1 CPU INSTRUCTION CYCLE

The sequence of operations performed by the CPU during its execution of instructions is presented in Fig. (3.1). As long as there are instructions to execute, the next instruction is fetched from main memory. The instruction is executed based on the operation specified in the opcode field of the instruction. At the completion of the instruction execution, a test is made to determine whether an interrupt has occurred. An interrupt handling routine needs to be invoked in case of an interrupt.

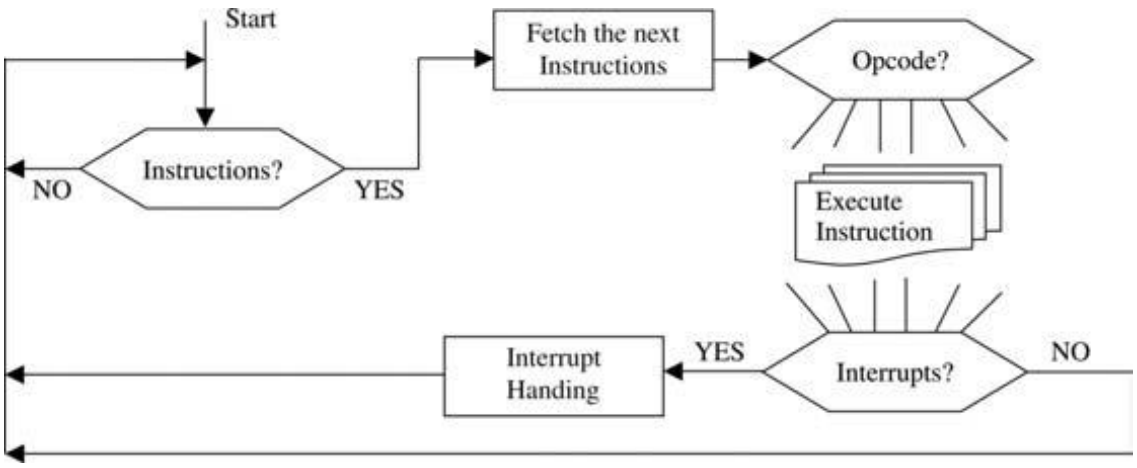


Fig. (3.1): CPU Functions.

The basic actions during fetching an instruction, executing an instruction, or handling an interrupt are defined by a sequence of micro-operations. A group of control signals must be enabled in a prescribed sequence to trigger the execution of a micro-operation. In this section, we show the micro-operations that implement instruction fetch, execution of simple arithmetic instructions, and interrupt handling.

3.1.1 Fetch Instructions

The sequence of events in fetching an instruction can be summarized as follows:

1. The contents of the PC are loaded into the MAR.
2. The value in the PC is incremented. (This operation can be done in parallel with a memory access).

3. As a result of a memory read operation, the instruction is loaded into the MDR.

4. The contents of the MDR are loaded into the IR.

Let us consider the one-bus datapath organization shown in Fig. (2.3). We will see that the fetch operation can be accomplished in three steps as shown in the table below, where $t0 < t1 < t2$. Note that multiple operations separated by ";" imply that they are accomplished in parallel.

Step	Micro-Operation
t_0	MAR ← (PC); A ← (PC)
t_1	MDR ← Mem [MAR]; PC ← (A)+4
t_2	IR ← (MDR)

using the three-bus datapath shown in Fig. (2.5), the following table shows the steps needed.

Step	Micro-Operation
t_0	MAR ← (PC); A ← (PC)+4
t_1	MDR ← Mem [MAR];
t_2	IR ← (MDR)

3.1.2 Execute Simple Arithmetic Operation

Add R_1, R_2, R_0 : This instruction adds the contents of source registers R_1 and R_2, and stores the results in destination register R_0. This addition can be executed as follows:

1. The registers R_0 , R_1 , R_2 are extracted from the IR.

2. The contents of R_1 and R_2 are passed to the ALU for addition.

3. The output of the ALU is transferred to R_0.

Using the one-bus datapath shown in Fig. (2.3), this addition will take three steps as shown in the following table, where $t_0 < t_1 < t_2$.

Step	Micro-Operation
t_0	$A \leftarrow (R_1)$
t_1	$B \leftarrow (R_2)$
t_2	$R_0 \leftarrow (A)+(B)$

Using the two-bus datapath shown in Fig. (2.4a), this addition will take two steps as shown in the following table, where $t_0 < t_1$.

Step	Micro-Operation
t_0	$A \leftarrow (R_1) + (R_2)$
t_1	$R_0 \leftarrow (A)$

Using the two-bus datapath with in-bus and out-bus shown in Fig. (2.4b), this addition will take two steps as shown in the following table, where $t0 < t1$.

Step	Micro-Operation
t_0	$A \leftarrow (R_1)$
t_1	$R_0 \leftarrow (A) + (R_2)$

Using the three-bus datapath shown in Fig. (2.5), this addition will take one steps as shown in the following table.

Step	Micro-Operation
t_0	$R_0 \leftarrow (R_1) + (R_2)$

Add X, R_0 : This instruction adds the contents of memory location X to register R_0 and stores the result in R_0. This addition can be executed as follows:

1. The memory location X is extracted from IR and loaded into MAR.

2. As a result of memory read operation, the contents of X are loaded into MDR.

3. The contents of MDR are added to the contents of R_0.

Using the one-bus datapath shown in Fig. (2.3), this addition will take five steps as shown in the following table, where $t_0 < t_1 < t_2 < t_3 < t_4$.

Step	Micro-Operation
t_0	MAR ← X
t_1	MDR ← Mem[MAR]
t_2	A ← (R_0)
t_3	B ← (MDR)
t_4	R_0 ← (A)+(B)

Using the two-bus datapath shown in Fig. (2.4a), this addition will take four steps as shown in the following table, where $t_0 < t_1 < t_2 < t_3$.

Step	Micro-Operation
t_0	MAR ← X
t_1	MDR ← Mem[MAR]
t_2	A ← (R_0) + (MDR)
t_3	R_0 ← (A)

Using the two-bus datapath with in-bus and out-bus shown in Fig. (2.4b), this addition will take four steps as shown in the following table, where $t_0 < t_1 < t_2 < t_3$.

Step	Micro-Operation
t_0	MAR ← X
t_1	MDR ← Mem[MAR]
t_2	A ← (R_0)
t_3	R_0 ← (A) + (MDR)

Using the three-bus datapath shown in Fig. (2.5), this addition will take three steps as shown below, where $t_0 < t_1 < t_2$.

Step	Micro-Operation
t_0	MAR ← X
t_1	MDR ← Mem[MAR]
t_2	R_0 ← (R_0) + (MDR)

3.1.3 Interrupt Handling

After the execution of an instruction, a test is performed to check for pending interrupts. If there is an interrupt request waiting, the following steps take place:

1. The contents of PC are loaded into MDR (to be saved).

2. The MAR is loaded with the address at which the PC contents are to be saved.

3. The PC is loaded with the address of the first instruction of the interrupt handling routine.

4. The contents of MDR (old value of the PC) are stored in memory.

The following table shows the sequence of events, where $t_1 < t_2 < t_3$.

Step	Micro-Operation
t_1	MDR ← PC
t_2	MAR ← Address1 (where to save PC); PC ← Address 2 (interrupt handling routine)
t_3	Mem[MAR] ← (MDR)

3.2 Control Unit

Even though the datapath is capable of performing all the operations of the microprocessor, it cannot, however, do it on its own. In order for the datapath to execute the operations automatically, the control unit is required. The control unit, also known as the controller, controls the operations of the datapath, and therefore, the operations of the entire microprocessor.

The control unit is the main component that directs the system operations by sending control signals to the datapath. These signals control the flow of data within the CPU and between the CPU and external units such as memory and I/O. Control buses generally carry signals between the control unit and other computer components in a clock-driven manner. The system clock produces a continuous sequence of pulses in a specified duration and frequency. A sequence of steps (t_0, t_1, t_2, … where $t_0 < t_1 < t_2 < \ldots$) are used to execute a certain instruction. The op-code field of a fetched instruction is decoded to provide the control signal generator with information about the instruction to be executed. Step information generated by a logic circuit module is used with other inputs to generate control signals. The signal generator can be specified simply by a set of Boolean equations for its output in terms of its inputs. Fig. (3.2) shows a block diagram that describes how timing is used in generating control signals.

There are mainly two different types of control units:

1. Microprogrammed.

2. Hardwired.

In microprogrammed control, the control signals associated with operations are stored in special memory units inaccessible by the programmer as control words. A control word is a microinstruction that specifies one or more micro-operations. A sequence of microinstructions is called a microprogram, which is stored in a ROM or RAM called a control memory CM.

In hardwired control, fixed logic circuits that correspond directly to the Boolean expressions are used to generate the control signals. Clearly hardwired control is faster than microprogrammed control.

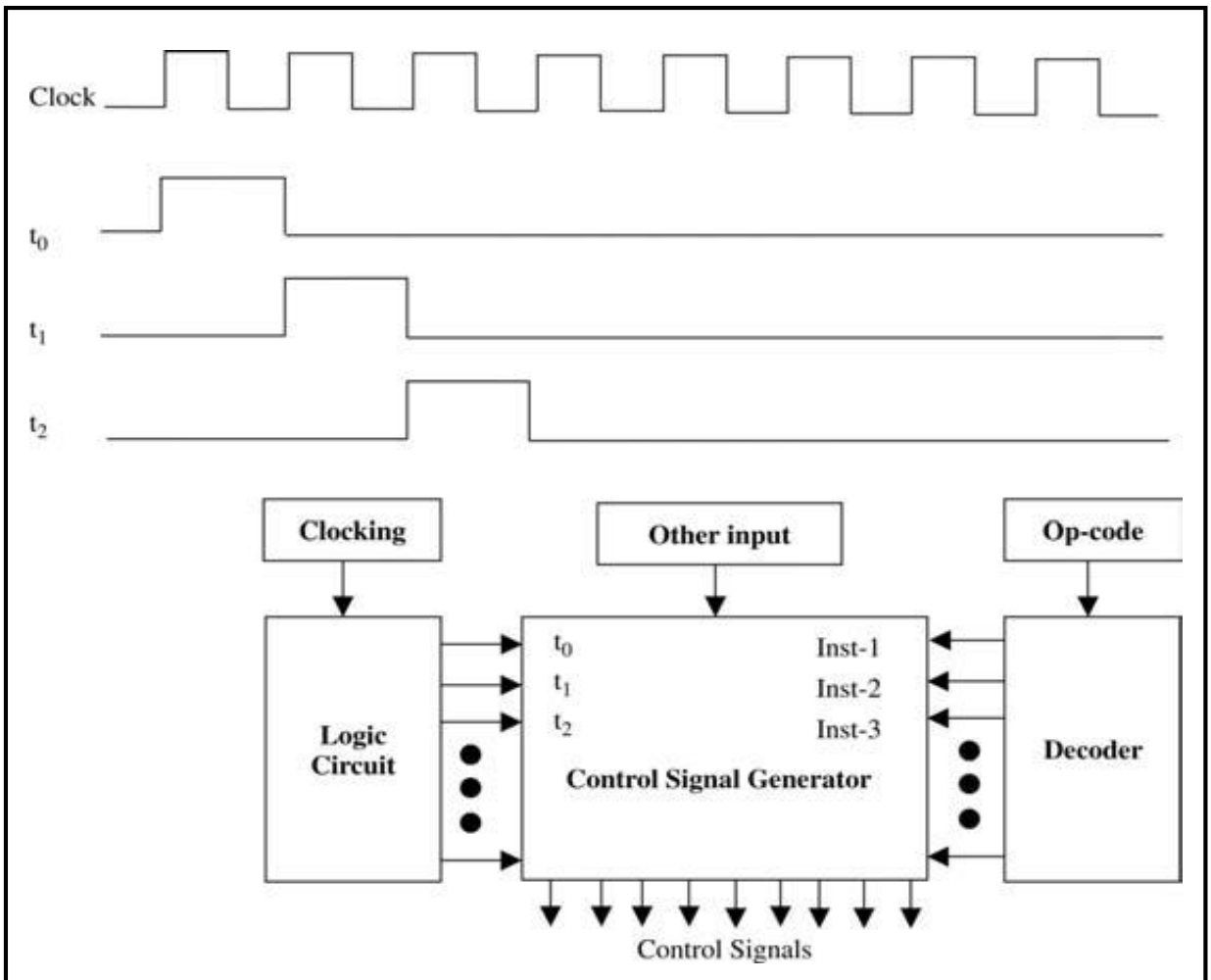


Fig. (3.2): Timing of control signals.

However, hardwired control could be very expensive and complicated for complex systems. Hardwired control is more economical for small control units. It should also be noted that microprogrammed control could adapt easily to changes in the system design. We can easily add new instructions without changing hardware. Hardwired control will require a redesign of the entire systems in the case of any change.

Example 1:

Let us revisit the add operation in which we add the contents of source registers R_1, R_2, and store the results in destination register R_0.We have shown earlier that this operation can be done in one step using the three-bus datapath shown in Fig. (2.5).
Let us try to examine the control sequence needed to accomplish this addition at step t_0. Suppose that the op-code field of the current instruction was decoded to Inst-x type. First we need to select the source registers and the destination register, then we select Add as the ALU function to be performed. The following table shows the needed step and the control sequence.

Step	Instruction Type	Micro-operation	Control
t_0	Ins-x	$R_0 \leftarrow (R_1) + (R_2)$	Select R_1 as source 1 on out-bus1 (R_1 out-bus1)
			Select R_2 as source 2 on out-bus2 (R_2 out-bus2)
			Select R_0 as destination on in-bus (R_0 in-bus)
			Select the ALU function Add (Add)

Fig. (3.3) shows the signals generated to execute Inst-x during time period t_0. The AND gate ensures that these signals will be issued when the op-code is decoded into Inst-x and during time period t_0. The signals (R_1out-bus 1), (R_2out-bus2), (R_0in-bus), and (Add) will select R1 as a source on out-bus1, R_2 as a source on out-bus2, R_0 as destination on in-bus, and select the ALUs add function, respectively.

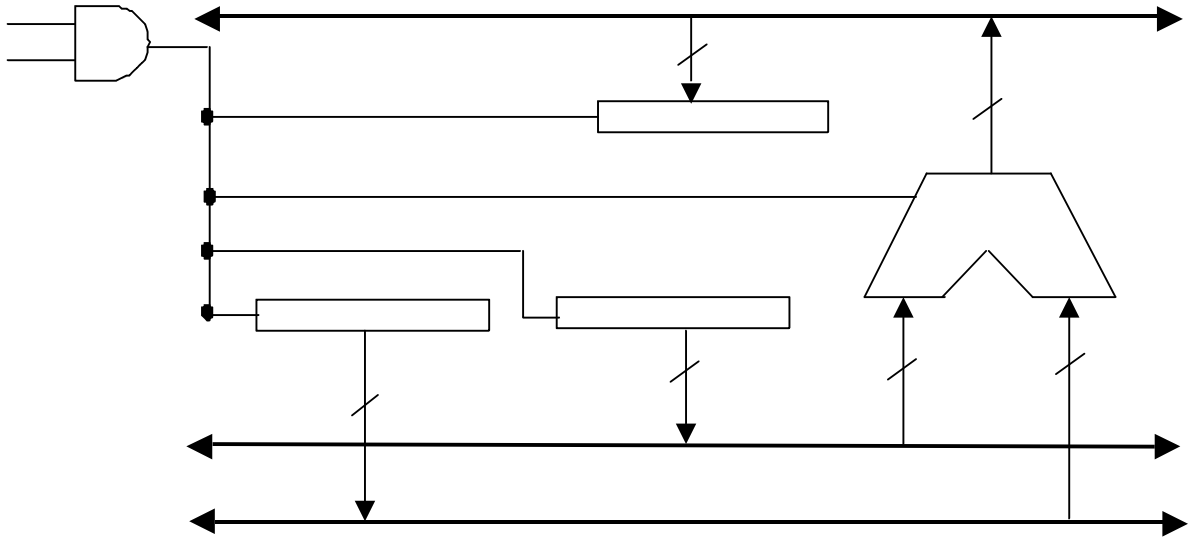


Fig. (3.3): Signals generated to execute Inst-x on three-bus datapath during time period t_0 .

Example 2 :

Let us repeat the operation in the previous example using the one-bus datapath shown in Fig. (2.3). We have shown earlier that this operation can be carried out in three steps using the one-bus datapath. Suppose that the op-code field of the current instruction was decoded to Inst-x type. The following table shows the needed steps and the control sequence.

Step	Instruction Type	Micro-operation	Control
t_0	Inst-x	$A \leftarrow (R_1)$	Select R_1 as source (R_1 out) Select A as destination (A in)
t_1	Inst-x	$A \leftarrow (R_2)$	Select R_2 as source (R_2 out) Select B as destination (B in)
t_2	Inst-x	$R_0 \leftarrow (R_1) + (R_2)$	Select the ALU function Add (Add). Select Ro as destination (R_0 in)

Fig. (3.4) shows the signals generated to execute Inst-x during time periods t_0 , t_1, and t_2. The AND gates ensure that the appropriate signals will be issued when the op-code is decoded into Inst-x and during the appropriate time period. During t_0, the signals (R_1out) and (A in) will be issued to move the contents of R_1 into A. Similarly during t_1, the signals (R_2 out) and (B in) will be issued to move the contents of R_2 into B. Finally, the signals (R_0in) and (Add) will be issued during t_2 to add the contents of A and B and move the results into R_0.

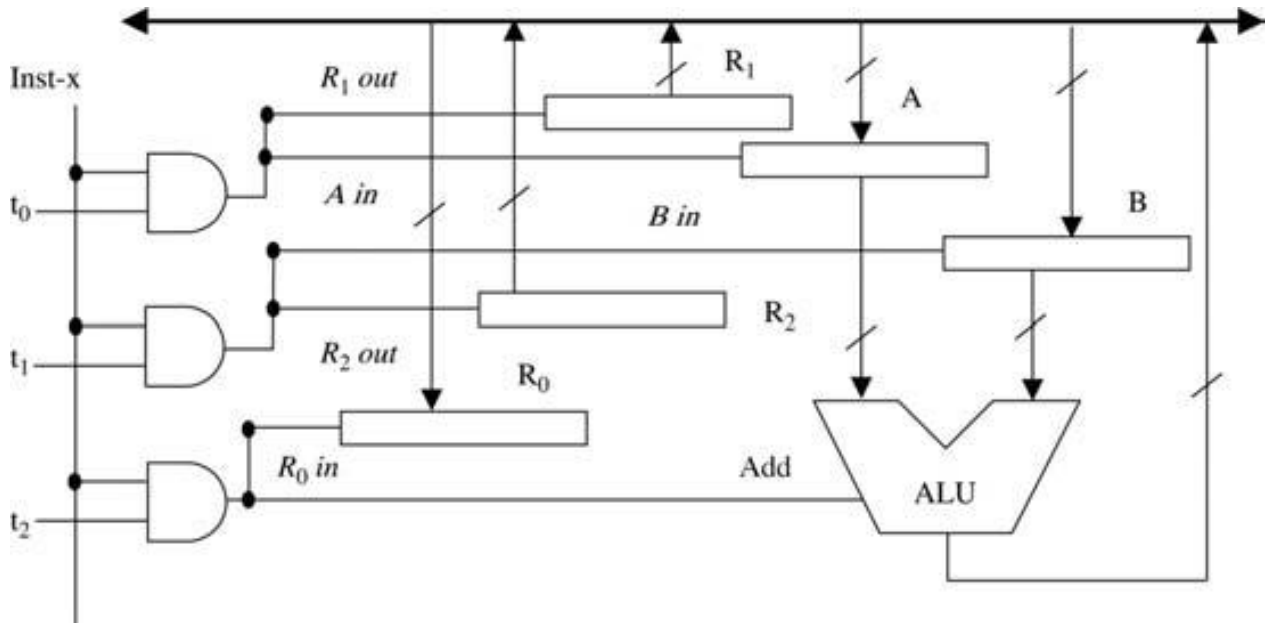


Fig. (3.4): Signals generated to execute Inst-x on one-bus datapath during time period t_0, t_1, t_2.

3.1 Introduction

The 80386 microprocessor is a full 32-bit version of the earlier 8086/80286 16-bit microprocessors, and represents a major advancement in the architecture, a switch from a 16-bit architecture to a 32-bit architecture. Along with this larger word size are many improvements and additional features. The 80386 microprocessor features multitasking, memory management, virtual memory (with or without paging), software protection, and a large memory system. All software written for the early 8086/8088 and the 80286 are upward-compatible to the 80386 microprocessor. The amount of memory addressable by the 80386 is increased from the 1M bytes found in the 8086/8088 and the 16M bytes found in the 80286, to 4G bytes in the 80386.

Before the 80386 or any other microprocessor can be used in a system, the function of each pin must be understood. Figure (3-1a) illustrates the pin-out of the 80386DX microprocessor. The 80386DX is packaged in a 132-pin PGA (pin grid array). Two versions of the 80386 are commonly available: the 80386DX, which is illustrated and described in this section; the other is the 80386SX, which is a reduced bus version of the 80386. A new version of the 80386-the 80386EX-incorporates the AT bus system, dynamic RAM controller, programmable chip selection logic, 26 address pins, 16 data pins, and 24 I/O pins. Figure (3-1b) illustrates the 80386SX embedded PC.

The 80386DX addresses 4G bytes of memory through its 32-bit data bus and 32-bit address. The 80386SX, more like the 80286, addresses 16M bytes of memory with its 24-bit address bus via its 16-bit data bus. The 80386SX was developed after the 80386DX for applications that didn't require the full 32-bit bus version. The 80386SX is found in many personal computers that use the same basic motherboard design as the 80286. At the time that the 80386SX was popular, most applications, including Windows, required fewer than 16M bytes of memory, so the 80386SX is a popular and a less costly version of the 80386 microprocessor.

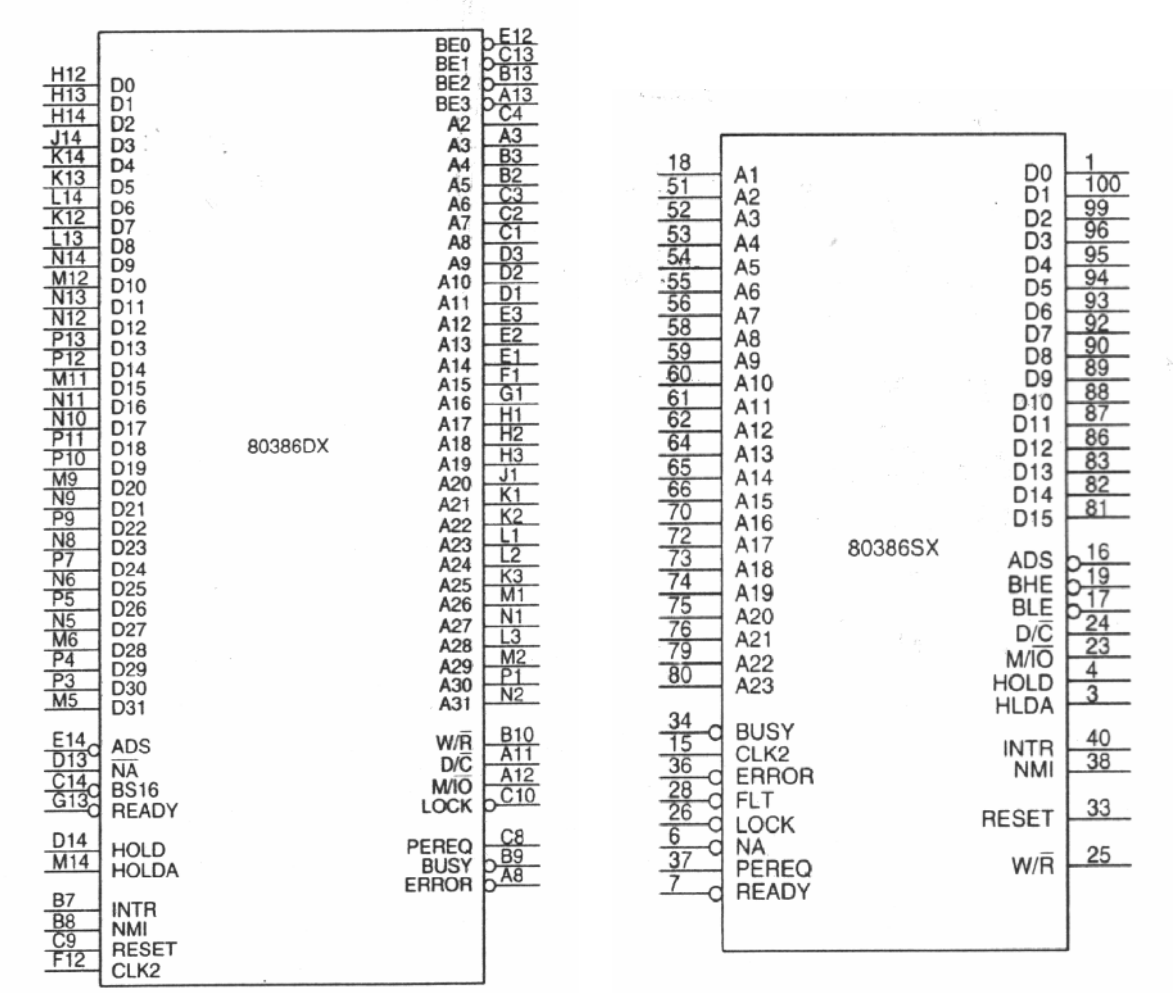


Fig. (3.1): The pin-outs of the 80386DX and 80386SX microprocessors.

A31-A2: Address bus connections address any of the 1Gx32 memory locations found in the 80386 memory system. Note that A0 and A1 are encoded in the bus enable ($\overline{BE3} - \overline{BE0}$) to select any or all of the four bytes in a 32-bit wide memory location. Also note that because the 80386SX contains a 16-bit data bus in place of the 32-bit data bus found on the 80386DX, A1 is present on the 80386SX, and the bank selection signals are replaced with $\overline{BHE}$ and $\overline{BLE}$. The $\overline{BHE}$ signal enables the upper data bus half; the $\overline{BLE}$ signal enables the lower data bus half.

D31-D0: Data bus connections transfer data between the microprocessor and its memory and I/O system. Note that the 80386SX contains D15-DO.

$\overline{BE3}$ **–** $\overline{BE0}$***: Bank enable signals*** select the access of a byte, word, or double word of data. These signals are generated internally by the microprocessor from address bits A1 and A0. On the 80386SX, these pins are replaced by $\overline{BHE}$, $\overline{BLE}$, and A1.

M / $\overline{IO}$: Memory/IO selects a memory device when a logic 1 or an I/O device when a logic 0. During the I/O operation, the address bus contains a 16-bit I/O address on address connections A15-A2.

W / $\overline{R}$: Write/read indicates that the current bus cycle is a write when a logic 1 or a read when a logic 0.

$\overline{ADS}$***: The address data strobe*** becomes active whenever the 80386 has issued a valid memory or I/O address. This signal is combined with the ***W / $\overline{R}$*** signal to generate the separate read and write signals present in the earlier 8086-80286 microprocessor-based systems.

RESET: Reset initializes the 80386, causing it to begin executing software at memory location FFFFFFF0H. The 80386 is reset to the real mode, and the leftmost 12 address connections remain logic 1s (FFFH) until a far jump or far call is executed. This allows compatibility with earlier microprocessors.

CLK2: Clock times 2 is driven by a clock signal that is twice the operating frequency of the 80386. For example, to operate the 80386 at 16 MHz, we apply a 32 MHz clock to this pin.

$\overline{READY}$***: Ready*** controls the number of wait states inserted into the timing to lengthen memory accesses. .

$\overline{LOCK}$***: Lock*** becomes a logic 0 whenever an instruction is prefixed with the LOCK:prefix. This is used most often during DMA accesses.

D*/$\overline{C}$: *Data/control indicates that the data bus contains data for or from memory or I/O when a logic 1. If ***D*/$\overline{C}$** is a logic 0, the microprocessor is halted or executes an interrupt acknowledge.

$\overline{BS16}$: *Bus size 16* selects either a 32-bit data bus (**$\overline{BS16}$** = 1) or a 16-bit data bus (**$\overline{BS16}$**=0). In most cases, if an 80386DX is operated on a 16-bit data bus, we use the 80386SX that has a 16-bit data bus.

$\overline{NA}$: ***Next address*** causes the 80386 to output the address of the next instruction or data in the current bus cycle. This pin is often used for pipelining the address.

HOLD: Hold requests a DMA action.

HLDA: Hold acknowledge indicates that the 80386 is currently in a hold condition.

$\overline{PEREQ}$: *The coprocessor request* asks the 80386 to relinquish control and is a direct connection to the 80387 arithmetic coprocessor.

$\overline{BUSY}$: *Busy* is an input used by the WAIT or FWAIT instruction that waits for the coprocessor to become not busy. This is also a direct connection to the 80387 from the 80386.

$\overline{ERROR}$: *Error* indicates to the microprocessor that an error is detected by the coprocessor.

INTR: An ***interrupt*** request is used by external circuitry to request an interrupt.

NMI: A ***non-maskable interrupt*** requests a non-maskable interrupt as it did on the earlier versions of the microprocessor.

3.2 Internal Architecture of the 80386DX Microprocessor

The internal architecture of the 8086 family of microprocessors has changed a lot as part of the evolutionary process from the original 8086 to the 80386. All members of the 8086 family employ what is called *parallel processing*. That is, they are implemented with simultaneously operating multiple processing units. Each unit

has a dedicated function and they operate at the same time. The more the parallel processing, the higher the performance of the microprocessor.

The 8086 microprocessor contains just two processing units: the bus interface unit and execution unit. In the 80286 microprocessor, the internal architecture was further partitioned into four independent processing elements: the bus unit, the instruction unit, the execution unit, and the address unit. This additional parallel processing provided an important contribution to the higher level of performance achieved with the 80286, architecture.

The 80386DX's internal architecture: is illustrated in Fig. (3.2). Here we see that to enhance the performance, more parallel processing elements are provided. Notice that now there are six functional units: the ***execution unit***, the ***segment unit***, the ***page unit***, the ***bus unit***, the ***prefetch unit***, and the ***decode unit***. Let us now look more closely at each of the processing units of the 80386DX.

The bus unit is the 80386DX's interface to the outside world. By interface, we mean the path by which it connects to external devices. The bus interface provides a 32-bit data bus, a 32-bit address bus, and the signals needed to control transfers over the bus. In fact, 8-bit, l6-bit, and 32-bit data transfers are supported. These buses are demultiplexed like those of the 80286. That is, the 80386DX has separate pins for its address and data bus lines. This demultiplexing of address and data results in higher performance and easier hardware design. Expanding the data bus width to 32 bits further improves the performance of the 80386DX's hardware architecture as compared to that of either the 8086 or 80286.

The bus unit is responsible for performing all external bus operations. This processing unit contains the latches and drivers for the address bus, transceivers for the data bus and control logic for signaling whether a memory, input/output or interrupt-acknowledg bus cycle is being performed. Looking at Fig. (3.2), we find that for data accesses, the address of the storage location that is to be accessed is input

from the paging unit, and for code accesses the address is provided by the prefetch unit.

The prefetch unit implements a mechanism known as an instruction stream queue. This queue permits the 80386DX to prefetch up to 16 bytes of instruction code. Whenever the queue is not full-that is, it has room for at least 4 more bytes and, at the same time, the execution unit is not asking it to read or write operands from memory-the prefetch unit supplies addresses to the bus interface unit and signals it to look ahead in the program by fetching the next sequential instructions. Prefected instructions are held in the FIFO queue for use by the instruction decoder. Whenever bytes are loaded at the input end of the queue, they are automatically shifted up through the FIFO to the empty locations near the output. With its 32-bit data bus. the 80386DX fetches 4 bytes of instruction code in a single memory cycle. Through this prefetch mechanism, the fetch time for most instructions is hidden. If the queue in the prefetch unit is full and the execution unit is not requesting access to operands in memory the bus interface unit does not need to perform any bus cycle.

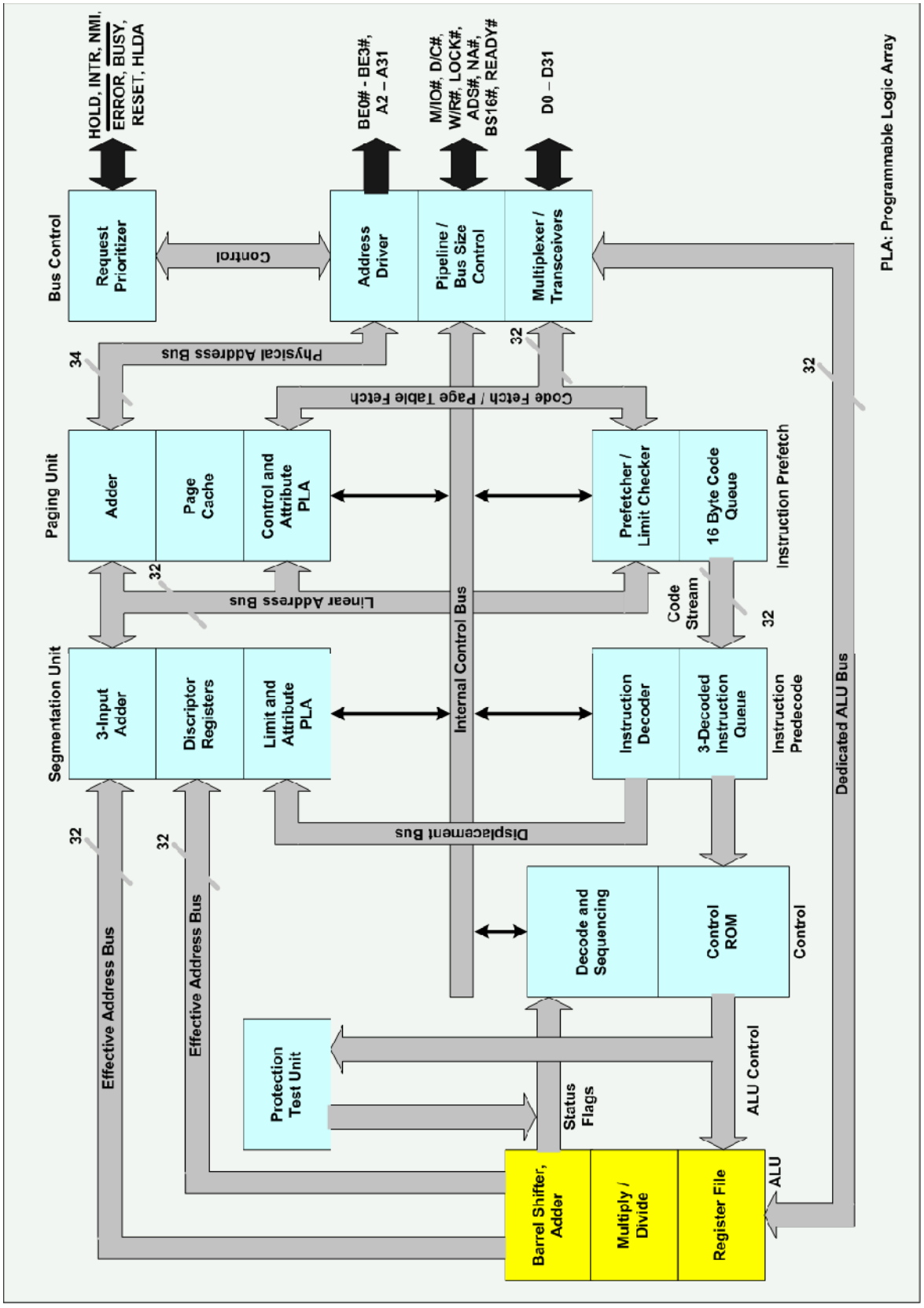
The execution unit includes the arithmetic/logic unit (ALU), the 80386DX's registers, special multiply, divide, and shift hardware, and a control ROM. By registers, we mean the 80386DX's general-purpose registers, such as EAX, EBX, and ECX. The control ROM contains the microcode sequences that define the operation performed by each of the 80386DX's machine code instructions. The execution unit reads decoded instructions from the instruction queue and performs the operations that they specify. It is the ALU that performs the arithmetic, logic, and shift operations required by an instruction. If necessary, during the execution of an instruction, it requests the segment and page units to generate operand addresses and the bus interface unit to perform read or write bus cycles to access data in memory or I/O devices. The extra hardware that is provided to perform multiply, divide, shift, and rotate operations improves the performance of instructions that employ these

functions.

The segment and page units provide the memory-management and protection services for the 80386DX. They off-load the responsibility for address generation, address translation, and segment checking from the bus interface unit, thereby further boosting the performance of the MPU. The segment unit implements the segmentation model of the 80386DX's memory management. That is, it contains dedicated hardware for performing high-speed address calculations, logical-to-linear address translation, and protection check. For instance, when in the real mode, the execution unit requests the segment unit to obtain the address of the next instruction to be fetched by adding an appended version of the current contents of the code segment (CS) register with the value in the instruction pointer (IP) register to obtain the 20-bit physical address that is to be output on the address bus. This address is passed on to the bus unit.

For protected mode, the segment unit performs the logical-to-linear address translation and various protection checks needed when performing bus cycles. It contains the segment registers and the 6-wordx64-bit cache that is used to hold the current descriptors within the 80386DX.

The page unit implements the protected mode paging model of the 80386DX's memory management. It contains the translation lookaside buffer that stores recently used page directory and page table entries. When paging is enabled, the linear address produced by the segment unit is used as the input of the page unit. Here the linear address is translated into the physical address of the memory or I/O location to be accessed. This physical memory or I/O address is output to the bus interface unit.


HOLD, INTR, NMI, ERROR, BUSY, RESET, HLDA
BE0# - BE3#, A2 – A31
M/IO#, D/C#, W/R#, LOCK#, ADS#, NA#, BS16#, READY#
D0 – D31
Bus Control
Request Prioritizer
Control
Address Driver
Pipeline / Bus Size Control
Multiplexer / Transceivers
PLA: Programmable Logic Array
34
Physical Address Bus
32
Code Fetch / Page Table Fetch
32
Paging Unit
Adder
Page Cache
Control and Attribute PLA
Prefetcher / Limit Checker
16 Byte Code Queue
Instruction Prefetch
32
Linear Address Bus
Code Stream
32
Segmentation Unit
3-Input Adder
Discriptor Registers
Limit and Attribute PLA
Internal Control Bus
Instruction Decoder
3-Decoded Instruction Queue
Instruction Predecode
32
Displacement Bus
32
Effective Address Bus
Effective Address Bus
Decode and Sequencing
Control ROM
Control
Protection Test Unit
Status Flags
ALU Control
Barrel Shifter, Adder
Multiply / Divide
Register File
ALU
Dedicated ALU Bus

SOFTWARE MODEL OF THE 80386DX MICROPROCESSOR

The purpose of a *software model* is to aid the programmer in understanding the operation of the microcomputer system from a software point of view. To be able to program a microprocessor, one does not necessarily need to know all its hardware architecture features. For instance, we do not need to know the function of the signals at its various pins, their electrical connections, or their electrical switching characteristics. The function interconnection, and operation of the internal circuits of the microprocessor also need not normally be considered. What is important to the programmer is to know the various registers within the device and to understand their purpose, functions, operating capabilities, and limitations.

Furthermore, it is essential to know how external memory is organized and how it is accessed to obtain instructions and data.

4.1 Memory Organization and Segmentation

The physical memory of an 80386 system is organized as a sequence of 8-bit bytes. Each byte is assigned a unique address that ranges from zero to a maximum of 232-1 (4 gigabytes). 80386 programs, however, are independent of the physical address space. This means that programs can be written without knowledge of how much physical memory is available and without knowledge of exactly where in physical memory the instructions and data are located.

The model of memory organization seen by applications programmers is determined by systems-software designers. The architecture of the 80386 gives designers the freedom to choose a model for each task. The model of memory organization can range between the following extremes:

- A "flat" address space consisting of a single array of up to 4 gigabytes.
- A segmented address space consisting of a collection of up to 16,383 linear address spaces of up to 4 gigabytes each.

With the flat memory model (see Figure 4-1), memory appears to a program as a single, continuous address space, called a linear address space. Code (a program's instructions), data, and the procedure stack are all contained in this address space. The linear address space is byte addressable, with addresses running contiguously

from 0 to 232 - 1. An address for any byte in the linear address space is called a linear address.

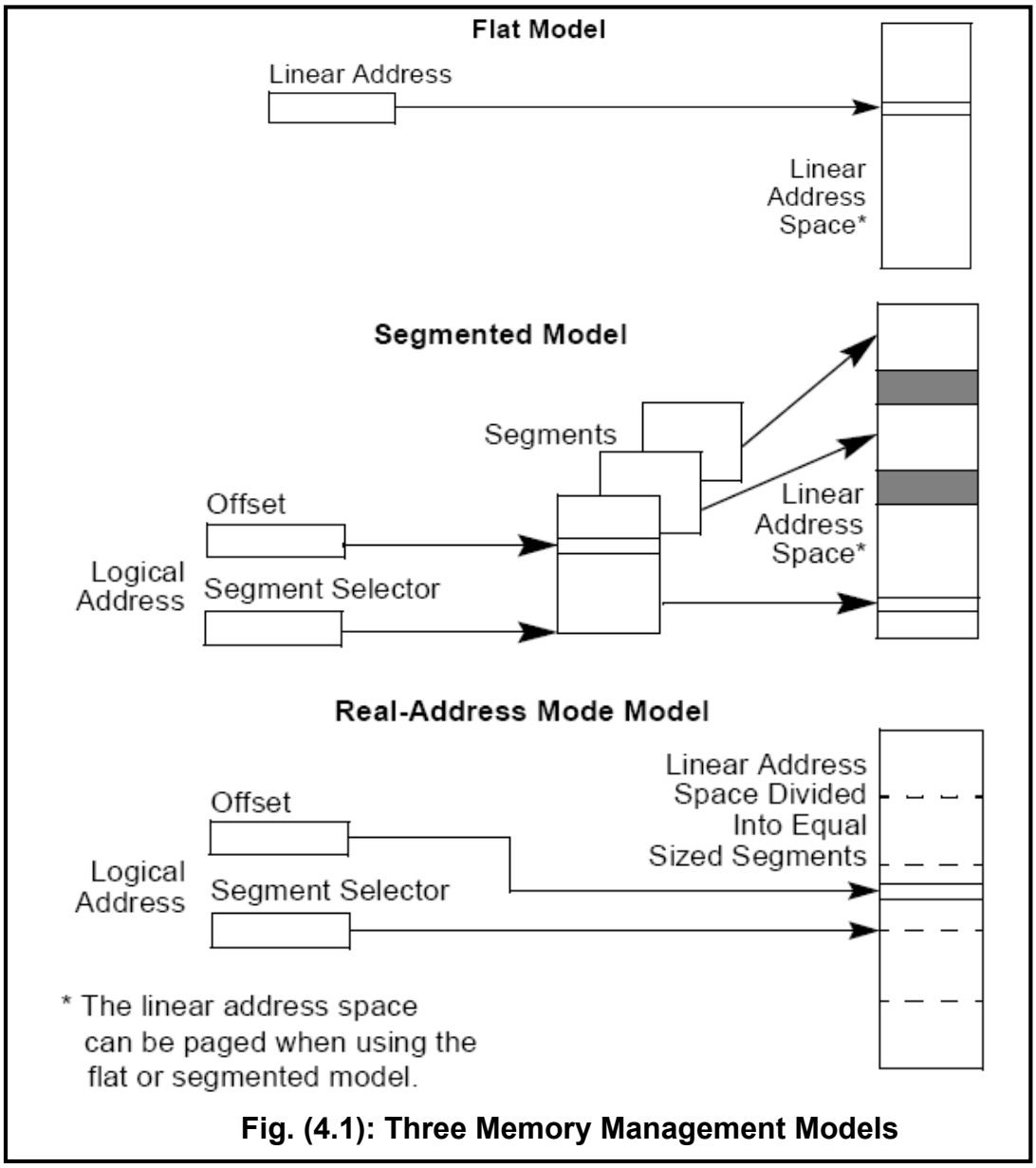


Fig. (4.1): Three Memory Management Models

`

With the **segmented memory model**, memory appears to a program as a group of independent address spaces called segments. When using this model, code, data, and stacks are typically contained in separate segments. To address a byte in a segment, a program must issue a logical address, which consists of a segment selector and an offset. (A logical address is often referred to as a far pointer.) The segment selector identifies the segment to be accessed and the offset identifies a byte in the address space of the segment. The programs running on an Intel Architecture

processor can address up to 16,383 segments of different sizes and types, and each segment can be as large as 232 bytes.
The **real-address mode model** uses the memory model for the Intel 8086 processor, the first Intel Architecture processor. It was provided in all the subsequent Intel Architecture processors for compatibility with existing programs written to run on the Intel 8086 processor. The real address mode uses a specific implementation of segmented memory in which the linear address space for the program and the operating system/executive consists of an array of segments of up to 64K bytes in size each. The maximum size of the linear address space in real-address mode is 220 bytes.

4.2. REGISTERS

The processor provides 16 registers for use in general system and application programing. These registers can be grouped as follows:

1. General-purpose data registers. These eight registers are available for storing operands and pointers.
2. Segment registers. These registers hold up to six segment selectors.
3. Status and control registers. These registers report and allow modification of the state of the processor and of the program being executed.

4.2.1. General-Purpose Data Registers

The 32-bit general-purpose data registers EAX, EBX, ECX, EDX, ESI, EDI, EBP, and ESP are provided for holding the following items:

- Operands for logical and arithmetic operations
- Operands for address calculations
- Memory pointers.

Although all of these registers are available for general storage of operands, results, and pointers, caution should be used when referencing the ESP register. The ESP register holds the stack pointer and as a general rule should not be used for any other purpose. Many instructions assign specific registers to hold operands. For example, string instructions use the contents of the ECX, ESI, and EDI registers as operands. When using a segmented memory model, some instructions assume that pointers in certain registers are relative to specific segments. For instance, some instructions assume that a pointer in the EBX register points to a memory location in the DS segment.
The following is a summary of these special uses:
• EAX—Accumulator for operands and results data.
• EBX—Pointer to data in the DS segment.

• ECX—Counter for string and loop operations.
• EDX—I/O pointer.
• ESI—Pointer to data in the segment pointed to by the DS register; source pointer for string operations.
• EDI—Pointer to data (or destination) in the segment pointed to by the ES register; destination pointer for string operations.
• ESP—Stack pointer (in the SS segment).
• EBP—Pointer to data on the stack (in the SS segment).

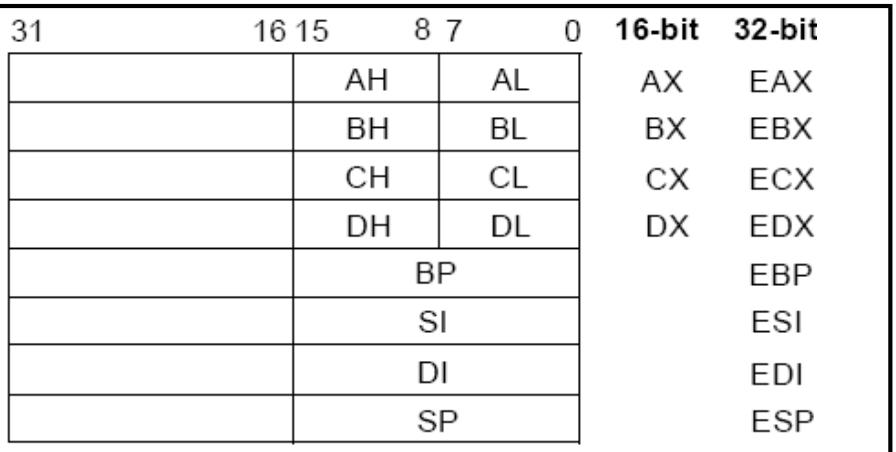


Fig. (4.2): Alternate General-Purpose Register Names

4.2.2. Segment Registers

The segment registers (CS, DS, SS, ES, FS, and GS) hold 16-bit segment selectors. A segment selector is a special pointer that identifies a segment in memory. When using the segmented memory model, each segment register is ordinarily loaded with a different segment selector so that each segment register points to a different segment within the linear-address space (as shown in Figure 4-3).

Code segment (CS) is a 16-bit register containing address of 64 KB segment with processor instructions. The processor uses CS segment for all accesses to instructions referenced by instruction pointer (IP) register. CS register cannot be changed directly. The CS register is automatically updated during far jump, far call and far return instructions.

Stack segment (SS) is a 16-bit register containing address of 64KB segment with program stack. By default, the processor assumes that all data referenced by the stack pointer (SP) and base pointer (BP) registers is located in the stack segment. SS register can be changed directly using POP instruction.

Data segment (DS) is a 16-bit register containing address of 64KB segment with program data. By default, the processor assumes that all data referenced by general registers (AX, BX, CX, DX) and index register (SI, DI) is located in the data segment. DS register can be changed directly using POP and LDS instructions.

Extra segment (ES) is a 16-bit register containing address of 64KB segment, usually with program data. By default, the processor assumes that the DI register references the ES segment in string manipulation instructions. ES register can be changed directly using POP and LES instructions.

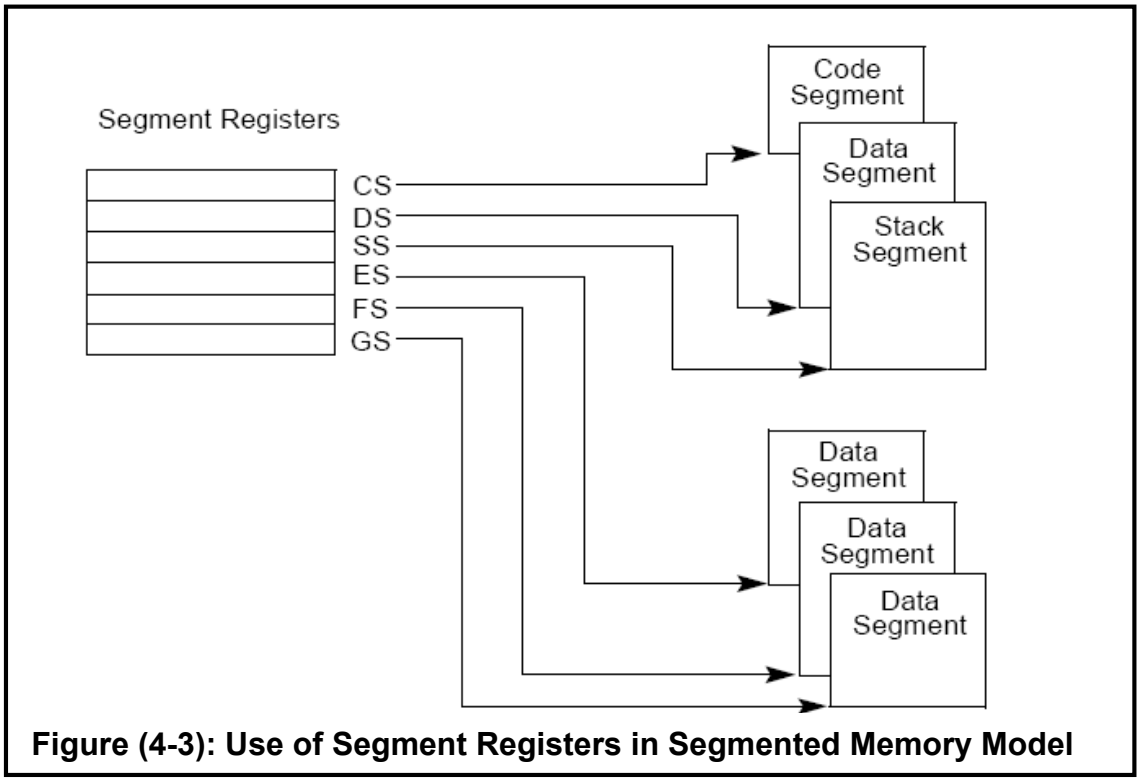


Figure (4-3): Use of Segment Registers in Segmented Memory Model

4.2.3. EFLAGS Register

The 32-bit EFLAGS register contains a group of status flags, a control flag, and a group of system flags. Figure (4-4) defines the flags within this register. Some of the flags in the EFLAGS register can be modified directly, using special-purpose instructions (described in the following sections). There are no instructions that allow the whole register to be examined or modified directly.

When a call is made to an interrupt or exception handler procedure, the processor automatically saves the state of the EFLAGS registers on the procedure stack. When

an interrupt or exception is handled with a task switch, the state of the EFLAGS register is saved in the TSS for the task being suspended.

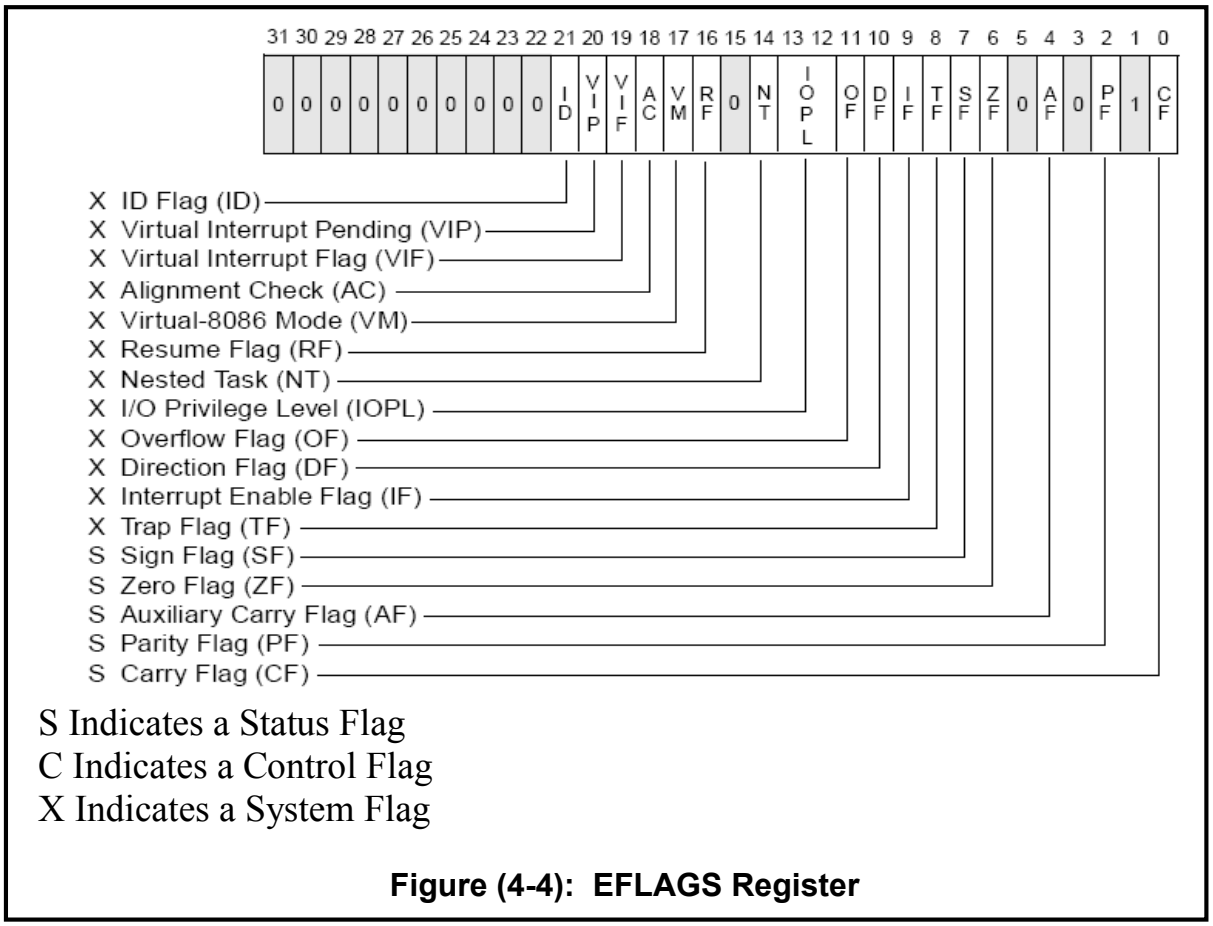


Figure (4-4): EFLAGS Register

STATUS FLAGS

The status flags (bits 0, 2, 4, 6, 7, and 11) of the EFLAGS register indicate the results of arithmetic instructions, such as the ADD, SUB, MUL, and DIV instructions. The functions of the status flags are as follows:

CF (bit 0) Carry flag: Set if an arithmetic operation generates a carry or a borrow out of the most-significant bit of the result; cleared otherwise. This flag indicates an overflow condition for unsigned-integer arithmetic. It is also used in multiple-precision arithmetic.

PF (bit 2) Parity flag: Set if the least-significant byte of the result contains an even number of 1 bits; cleared otherwise.

AF (bit 4) Adjust flag: Set if an arithmetic operation generates a carry or a borrow out of bit 3 of the result; cleared otherwise. This flag is used in binarycoded decimal (BCD) arithmetic.
ZF (bit 6) Zero flag: Set if the result is zero; cleared otherwise.
SF (bit 7) Sign flag: Set equal to the most-significant bit of the result, which is the sign bit of a signed integer. (0 indicates a positive value and 1 indicates a negative value.)
OF (bit 11) Overflow flag: Set if the integer result is too large a positive number or too small a negative number (excluding the sign-bit) to fit in the destination operand; cleared otherwise. This flag indicates an overflow condition for signed-integer (two's complement) arithmetic.

Control Flags

DF (bit 10) Direction flag: Controls the string instructions (MOVS, CMPS, SCAS, LODS, and STOS). Setting the DF flag causes the string instructions to auto decrement (that is, to process strings from high addresses to low addresses). Clearing the DF flag causes the string instructions to auto-increment (process strings from low addresses to high addresses). The STD and CLD instructions set and clear the DF flag, respectively.

TF (bit 8) Trap flag: Set to enable single-step mode for debugging; clear to disable single-step mode.

IF (bit 9) Interrupt enable flag: Controls the response of the processor to maskable interrupt requests. Set to respond to maskable interrupts; cleared to inhibit maskable interrupts.

System Flags and IOPL field

The system flags and IOPL field in the EFLAGS register control operating-system or executive operations. They should not be modified by application programs. The functions of the status flags are as follows:
IOPL (bits 12 and 13) I/O privilege level field: Indicates the I/O privilege level of the currently running program or task. The current privilege level (CPL) of the currently running program or task must be less than or equal to the I/O privilege level to access the I/O address space. This field can only be modified by the POPF and IRET instructions when operating at a CPL of 0.
NT (bit 14) Nested task flag: Controls the chaining of interrupted and called tasks. Set when the current task is linked to the previously executed task; cleared when the current task is not linked to another task.

RF (bit 16) Resume flag. Controls the processor's response to debug exceptions.
VM (bit 17) Virtual-8086 mode flag. Set to enable virtual-8086 mode; clear to return to protected mode.
AC (bit 18) Alignment check flag: Set this flag and the AM bit in the CR0 register to enable alignment checking of memory references; clear the AC flag and/or the AM bit to disable alignment checking.
VIF (bit 19) Virtual interrupt flag: Virtual image of the IF flag. Used in conjunction with the VIP flag. (To use this flag and the VIP flag the virtual mode extensions are enabled by setting the VME flag in control register CR4.)
VIP (bit 20) Virtual interrupt pending flag: Set to indicate to that an interrupt is pending; clear when no interrupts are pending. (Software sets and clears this flag. The processor only reads it.) Used in conjunction with the VIF flag.
ID (bit 21) Identification flag: The ability of a program to set or clear this flag indicates support for the CPUID instruction.