Unit 1

Evolution of the Microprocessor

Structure

1.1 Introduction

1.2 Objectives

1.3 The Breakthrough in Microprocessors

1.4 What led to the development of microprocessors?

1.5 How a microprocessor works
1.1 Introduction

The Collegiate Webster dictionary describes microprocessor as a computer processor contained on an integrated-circuit chip. In the mid-seventies, a microprocessor was defined as a central processing unit (CPU) realized on a LSI (large-scale integration) chip, operating at a clock frequency of 1 to 5 MHz and constituting an 8-bit system. It was a single component having the ability to perform a wide variety of different functions. Because of their relatively low cost and small size, the microprocessors permitted the use of digital computers in many areas
where the use of the preceding mainframe—and even minicomputers—would not be practical and affordable.

Many non-technical people associate microprocessors with only PCs yet there are thousands of appliances that have a microprocessor embedded in them—telephone, dishwasher, microwave, clock radio, etc. In these items, the microprocessor acts primarily as a controller and may not be known to the user.

1.2 Objectives
At the end of this chapter you will be able to:

- Know what led to the development of microprocessors?
- Explain how a microprocessor works
- Give Architecture of a microprocessor
- List the Generation of microprocessors
- Know the Companies associated with microprocessors

1.3 The Breakthrough in Microprocessors

The switching units in computers that were used in the early 1940s were the mechanical relays. These were devices that opened and closed as they did the calculations. Such mechanical relays were used in Zuse’s machines of the 1930s.

Come the 1950s, and the vacuum tubes took over. The Atanasoff-Berry Computer (ABC) used vacuum tubes as its switching units rather than relays. The switch from mechanical relay to vacuum tubes was an important technological advance as vacuum tubes could perform calculations considerably faster and more efficient than relay machines. However, this technological advance was short-lived because the tubes could not be made smaller than they were being made and had to be placed close to each other because they generated heat.
Then came the transistor which was acknowledged as a revolutionary development. In “Fire in the Valley”, the authors describe the transistor as a device which was the result of a series of developments in the applications of physics. The transistor changed the computer from a giant electronic brain to a commodity like a TV set. This innovation was awarded to three scientists: John Bardeen, Walter Brattain, and William Shockley.

As a result of the technological breakthrough of transistors, the introduction of minicomputers of the 1960s and the personal computer revolution of the 1970s was made possible. However, researchers did not stop at transistors. They wanted a device that could perform more complex tasks—a device that could integrate a number of transistors into a more complex circuit. Hence, the terminology, integrated circuits or ICs. Because physically they were tiny chips of silicon, they came to be also referred to as chips. Initially, the demand for ICs was typically the military and aerospace industries which were great users of computers and who were the only industries that could afford computers.

Later, Marcian “Ted” Hoff, an engineer at Intel, developed a sophisticated chip. This chip could extract data from its memory and interpret the data as an instruction. The term that evolved to describe such a device was “microprocessor”. Therefore, the term “microprocessor” first came into use at Intel in 1972. A microprocessor was nothing more than an extension of the arithmetic and logic IC chips corporating more functions into one chip. Today, the term still refers to an LSI single-chip processor capable of carrying out many of the basic operations of a digital computer.

Infact, the microprocessors of the late eighties and early nineties are full-sclae 32-bit and 32-bit address systems, operating at clock cycles of 25 to 50 MHz.

1.4 What led to the development of microprocessors?

As stated above, microprocessors essentially evolved from mechanical relays to integrated circuits. It is important to illustrate here what aspects of the computing industry led to
the development of microprocessors.

(1) Digital computer technology
In the History of Computing class, we studied, throughout the semester, how the computer industry learned how to make large, complex digital computers capable of processing more data and also how to build and use smaller, less expensive computers. The digital computer technology had been growing steadily since the late 1940s.

(2) Semiconductors
Like the digital computer technology, semiconductors had also been growing steadily since the invention of the transistor in the late 1940s. The 1960s saw the integrated circuit develop from just a few transistors to many complicated tasks, all of the same chip.

(3) The calculator industry
It appears as if this industry grew overnight during the 1970s from the simplest of four-function calculators to very complex programmable scientific and financial machines.
From all this, one idea became obvious—if there was an inexpensive digital computer, there would be no need to keep designing different, specialized integrated circuits. The inexpensive digital computer could simply be reprogrammed to perform whatever was the latest brainstorm, and there would be the new product.

The development of microprocessors can be attributed to when, in the early 1970s, digital computers and integrated circuits reached the required levels of capability. However, the early microprocessor did not meet all the goals: it was too expensive for many applications, especially those in the consumer market, and it could not hold enough information to perform many of the tasks being handled by the minicomputers of that time.
1.5 How a microprocessor works

According to Krutz, a microprocessor executes a collection of machine instructions that tell the processor what to do. Based on the instructions, a microprocessor does three basic things:

- Using its ALU (Arithmetic/Logic Unit), a microprocessor can perform mathematical operations like addition, subtraction, multiplication and division. Modern microprocessors contain complete floating point processors that can perform extremely sophisticated operations on large floating point numbers.

- A microprocessor can move data from one memory location to another.

- A microprocessor can make decisions and jump to a new set of instructions based on those decisions.

There may be very sophisticated things that a microprocessor does, but those are its three basic activities. Put simply, it fetches instructions from memory, interprets (decodes) them, and then executes whatever functions the instructions direct. For example, if the microprocessor is capable of 256 different operations, there must be 256 different instruction words. When fetched, each instruction word is interpreted differently than any of the other 255. Each type of microprocessor has a unique instruction set.

1.6 Architecture of a microprocessor

This is about as simple as a microprocessor gets. It has the following characteristics:

- an address bus (that may be 8, 16 or 32 bits wide) that sends an address to memory;

- a data bus (that may be 8, 16 or 32 bits wide) that can send data to memory or receive data from memory;
a **RD** (Read) and **WR** (Write) line to tell the memory whether it wants to set or get the addressed location;

- a clock line that lets a clock pulse sequence the processor; and

- a reset line that resets the program counter to zero (or whatever) and restarts execution.

A typical microprocessor, therefore, consists of: logical components—enable it to function as a programmable logic processor; program counter, stack, and instruction register—provide for the management of a program; the ALU—provide for the manipulation of data; and a decoder & timing and control unit—specify and coordinate the operation of other components.

The connection of the microprocessors to other units—memory and I/O devices—is done with the Address, Data, and control buses.

### 1.7 Generation of microprocessors

Microprocessors were categorized into five generations: first, second, third, fourth, and fifth generations. Their characteristics are described below:

**First-generation**

The microprocessors that were introduced in 1971 to 1972 were referred to as the first generation systems. First-generation microprocessors processed their instructions serially—they fetched the instruction, decoded it, then executed it. When an instruction was completed, the microprocessor updated the instruction pointer and fetched the next instruction, performing this sequential drill for each instruction in turn.

**Second generation**
By the late 1970s, enough transistors were available on the IC to usher in the second generation of microprocessor sophistication: 16-bit arithmetic and pipelined instruction processing. Motorola’s MC68000 microprocessor, introduced in 1979, is an example. Another example is Intel’s 8080. This generation is defined by overlapped fetch, decode, and execute steps (Computer 1996). As the first instruction is processed in the execution unit, the second instruction is decoded and the third instruction is fetched.

The distinction between the first and second generation devices was primarily the use of newer semiconductor technology to fabricate the chips. This new technology resulted in a five-fold increase in instruction, execution, speed, and higher chip densities.

**Third generation**

The third generation, introduced in 1978, was represented by Intel’s 8086 and the Zilog Z8000, which were 16-bit processors with minicomputer-like performance. The third generation came about as IC transistor counts approached 250,000.

Motorola’s MC68020, for example, incorporated an on-chip cache for the first time and the depth of the pipeline increased to five or more stages. This generation of microprocessors was different from the previous ones in that all major workstation manufacturers began developing their own RISC-based microprocessor architectures (Computer, 1996).

**Fourth generation**

As the workstation companies converted from commercial microprocessors to in-house designs, microprocessors entered their fourth generation with designs surpassing a million transistors. Leading-edge microprocessors such as Intel’s 80960CA and Motorola’s 88100 could issue and retire more than one instruction per clock cycle.

**Fifth generation**
Microprocessors in their fifth generation, employed decoupled super scalar processing, and their design soon surpassed 10 million transistors. In this generation, PCs are a low-margin, high-volume-business dominated by a single microprocessor.

1.8 Companies associated with microprocessors

Overall, Intel Corporation dominated the microprocessor area even though other companies like Texas Instruments, Motorola, etc also introduced some microprocessors. Listed below are the microprocessors that each company created.

(A) Intel

As indicated previously, Intel Corporation dominated the microprocessor technology and is generally acknowledged as the company that introduced the microprocessor successfully into the market.

Its first microprocessor was the 4004, in 1971. The 4004 took the integrated circuit one step further by locating all the components of a computer (CPU, memory and input and output controls) on a minuscule chip. It evolved from a development effort for a calculator chip set. Previously, the IC had had to be manufactured to fit a special purpose, now only one microprocessor could be manufactured and then programmed to meet any number of demands. The 4004 microprocessor was the central component in a four-chip set, called the 4004 Family: 4001 – 2,048-bit ROM, a 4002 – 320-bit RAM, and a 4003 – 10-bit I/O shift register. The 4004 had 46 instructions, using only 2,300 transistors in a 16-pin DIP. It ran at a clock rate of 740kHz (eight clock cycles per CPU cycle of 10.8 microseconds)—the original goal was 1MHz, to allow it to compute BCD arithmetic as fast (per digit) as a 1960's era IBM 1620.

Following in 1972 was the 4040 which was an enhanced version of the 4004, with an additional 14 instructions, 8K program space, and interrupt abilities (including shadows of the first 8 registers). In the same year, the 8008 was introduced. It had a 14-bit PC. The 8008 was intended as a terminal controller and was quite similar to the 4040. The 8008 increased the 4004’s word
length from four to eight bits, and doubled the volume of information that could be processed.

In April 1974, 8080, the successor to 8008 was introduced. It was the first device with the speed and power to make the microprocessor an important tool for the designer. It quickly became accepted as the standard 8-bit machine. It was the first Intel microprocessor announced before it was actually available. It represented such an improvement over existing designs that the company wanted to give customers adequate lead time to design the part into new products. The use of 8080 in personal computers and small business computers was initiated in 1975 by MITS’s Altair microcomputer. A kit selling for $395 enabled many individuals to have computers in their own homes (Computer, 1996). Following closely, in 1976, was 8048, the first 8-bit single-chip microcomputer. It was also designed as a microcontroller rather than a microprocessor—low cost and small size was the main goal. For this reason, data was stored on-chip, while program code was external. The 8048 was eventually replaced by the very popular but bizarre 8051 and 8052 (available with on-chip program ROMs). While the 8048 used 1-byte instructions, the 8051 had a more flexible 2-byte instruction set, eight 8-bit registers plus an accumulator A. Data space was 128 bytes and could be accessed directly or indirectly by a register, plus another 128 above that in the 8052 which could only be accessed indirectly (usually for a stack).

In 1978, Intel introduced its high-performance, 16-bit MOS processor—the 8086. This microprocessor offered power, speed, and features far beyond the second-generation machines of the mid-70’s. It is said that the personal computer revolution did not really start until the 8088 processor was created. This chip became the most ubiquitous in the computer industry when IBM chose it for its first PC.

In 1982, the 80286 (also known as 286) was next and was the first Intel processor that could run all the software written for its predecessor, the 8088. Many novices were introduced to desktop computing with a “286 machine” and it became the dominant chip of its time. It contained 130,000 transistors.

In 1985, the first multi-tasking chip, the 386 (80386) was created. This multi-tasking ability
allowed Windows to do more than one function at a time. This 32-bit microprocessor was designed for applications requiring high CPU performance. In addition to providing access to the 32-bit world, the 80386 addressed 2 other important issues: it provided system-level support to systems designers, and it was object-code compatible with the entire family of 8086 microprocessors (Computer, 1996). The 80386 was made up of 6 functional units: (i) execution unit (ii) segment unit (iii) page unit (iv) decode unit (v) bus unit and (vi) prefetch unit. The 80386 had registers divided into such categories as general-purpose registers, debug registers, and test registers. It had 275,000 transistors.

The 486 (80486) generation of chips really advanced the point-and-click revolution. It was also the first chip to offer a built-in math coprocessor, which gave the central processor the ability to do complex math calculations. The 486 had more than a million transistors. In 1993, when Intel lost a bid to trademark the 586, to protect its brand from being copied by other companies, it coined the name Pentium for its next generation of chips and there began the Pentium series—Pentium Classic, Pentium II, III and currently, 4.

(B) Motorola

The MC68000 was the first 32-bit microprocessor introduced by Motorola in early 1980s. This was followed by higher levels of functionality on the microprocessor chip in the MC68000 series. For example, MC68020, introduced later, had 3 times as many transistors, was about three times as big, and was significantly faster. Motorola 68000 was one of the second generation systems that was developed in 1973. It was known for its graphics capabilities. The Motorola 88000 (originally named the 78000) is a 32-bit processor, one of the first load-store CPUs based on a Harvard Architecture (Noyce, 1981).

(C) Digital Equipment Corporation (DEC)

In March 1974, Digital Equipment Corporation (DEC) announced it would offer a series of microprocessor modules built around the Intel 8008.
(D) Texas Instruments (TI)

A precursor to these microprocessors was the 16-bit Texas Instruments 1900 microprocessor which was introduced in 1976. The Texas Instruments TMS370 is similar to the 8051, another of TI’s creations. The only difference between the two was the addition of a B accumulator and some 16-bit support.

1.9 Microprocessors Today

Technology has been changing at a rapid pace. Everyday a new product is made to make life a little easier. The computer plays a major role in the lives of most people. It allows a person to do practically anything. The Internet enables the user to gain more knowledge at a much faster pace compared to researching through books. The portion of the computer that allows it to do more work than a simple computer is the microprocessor.

Microprocessor has brought electronics into a new era and caused component manufacturers and end-users to rethink the role of the computer. What was once a giant machine attended by specialists in a room of its own is now a tiny device conveniently transparent to users of automobile, games, instruments, office equipment, and a large array of other products.

From their humble beginnings 25 years ago, microprocessors have proliferated into an astounding range of chips, powering devices ranging from telephones to supercomputers (PC Magazine, 1996). Today, microprocessors for personal computers get widespread attention—and have enabled Intel to become the world's largest semiconductor maker. In addition, embedded microprocessors are at the heart of a diverse range of devices that have become staples of affluent consumers worldwide.
The impact of the microprocessor, however, goes far deeper than new and improved products. It is altering the structure of our society by changing how we gather and use information, how we communicate with one another, and how and where we work. Computer users want fast memory in their PCs, but most do not want to pay a premium for it.

1.10 Where is the industry of microprocessors going?

Almost immediately after their introduction, microprocessors became the heart of the personal computer. Since then, the improvements have come at an amazing pace. The 4004 ran at 108 kHz—that's kilohertz, not megahertz—and processed only 4 bits of data at a time. Today's microprocessors and the computers that run on them are thousands of times faster. Effectively, they've come pretty close to fulfilling Moore's Law (named after Intel cofounder Gordon Moore), which states that the number of transistors on a chip will double every 18 months or so. Performance has increased at nearly the same rate.

Can the pace continue? Well, nothing can increase forever. But according to Gerry Parker, Intel's executive vice president in charge of manufacturing, “we are far from the end of the line in terms of microprocessor performance. In fact, we're constantly seeing new advances in technology, one example being new forms of lithography that let designers position electronic components closer and closer together on their chips. Processors are created now using a 0.35-micron process. But next year we'll see processors created at 0.25 microns, with 0.18 and 0.13 microns to be introduced in the years to come.”

However, it's not just improvements in lithography and density that can boost performance. Designers can create microprocessors with more layers of metal tying together the transistors and other circuit elements. The more layers, the more compact the design. But these ultracompact microprocessors are also harder to manufacture and validate. New chip designs take up less space, resulting in more chips per wafer. The original Pentium (60/66 MHz) was 294 square millimeters, then it was 164 square millimeters (75/90/100 MHz), and now it's 91 square millimeters (133- to 200-MHz versions).
When will all this end? Interestingly, it may not be the natural limits of technology that will eventually refute Moore's Law. Instead, it's more likely to be the cost of each successive generation. Every new level of advancement costs more as making microprocessor development is a hugely capital-intensive business. Currently, a fabrication plant with the capacity to create about 40,000 wafers a month costs some $2 billion. And the rapid pace of innovations means equipment can become obsolete in just a few years. Still, there are ways of cutting some costs, such as converting from today's 8-inch silicon wafers to larger, 300-mm (roughly 12-inch) wafers, which can produce 2.3 times as many chips per wafer as those now in use. Moving to 300-mm wafers will cost Intel about $500 million in initial capital. Still, nothing lasts forever. As Parker notes, “the PC industry is built on the assumption that we can get more and more out of the PC with each generation, keep costs in check, and continue adding more value. We will run out of money before we run out of technology. When we can't hold costs down anymore, then it will be a different business”

At the beginning of last year, the buzz was about PlayStation 2 and the Emotion Engine processor that would run it. Developed by Sony and Toshiba, experts predicted the high-tech processor would offer unprecedented gaming power and more importantly, could provide the processing power for the PlayStation 2 to challenge cheap PCs as the entry-level device of choice for home access to the Web. PlayStation2 is equipped with the 295MHz MIPS-based Emotion engine, Sony's own CPU designed with Toshiba Corp., a 147MHz graphics processor that renders 75 million pixels per second, a DVD player, an IEEE 1394 serial connection, and two USB ports. Sony will use DVD discs for game titles and gives consumers the option of using the product for gaming, DVD movie playing and eventually Web surfing.

Soon, instead of catching up on the news via radio or a newspaper on the way to work, commuters may soon be watching it on a handheld computer or cell phone. Early January this year, Toshiba America Electronic Components announced its TC35273XB chip. The chip has 12Mb of integrated memory and an encoder and decoder for MPEG-4, an audio-video compression standard. According to Toshiba, the integrated memory is what sets this chip apart from others. With integrated memory, the chip consumes less power, making it a good fit for
portable gadgets. This chip is designed to specifically address the issues of battery life which can be very short with portable devices. The chip will have a RISC processor at its core and running at a clock speed of 70MHz.

Toshiba anticipates that samples of this chip will be released to manufacturers in the second quarter, and mass production will follow in the third quarter. Shortly after this release, new handheld computers and cell phones using the chip and offering streaming media will be expected.
It is reported in CNET news, that in February this year, IBM started a program to use the Internet to speed custom-chip design, bolstering its unit that makes semiconductors for other companies.

IBM, one of the biggest makers of application-specific chips, would set up a system so that chip designs are placed in a secure environment on the Web, where a customer's design team and IBM engineers would collaborate on the blueprints and make changes in real time.

Designing custom chips, which are used to provide unique features that standard processors don't offer, requires time-consuming exchanges of details between the clients that provide a basic framework and the IBM employees who do the back-end work. Using the Internet will speed the process and make plans more accurate. IBM figures that since their customers ask for better turnaround time and better customer satisfaction, this would be one way to tackle this. As a pilot program, this service was to be offered to a set of particular, selected customers initially, and then would include customers who design the so-called system-on-a-chip devices that combine several functions on one chip.

A new microprocessor unveiled in February 2000 by Japan’s NEC, offers high-capacity performance while only consuming small amounts of power, making it ideal for use in mobile devices. This prototype could serve as the model for future mobile processors. The MP98 processor contains four microprocessors on the same chip that work together in such a way that they can be switched on and off depending on the job in hand. For example, a single processor can be used to handle easy jobs, such as data entry, through a keypad, while more can be brought online as the task demands, with all four working on tasks such as processing video. This gives designers of portable devices the best of both worlds—low power consumption and high capacity.

However, it should be noted that the idea of putting several processors together on a single chip is not new as both IBM and Sun Microsystems have developed similar devices. The
only difference is that MP98 is the first working example of a “fine grain” device that offers better performance. Commercial products based on this technology are likely to be seen around 2003.

In PC World, it was reported that, last September, a Japanese dentist received U.S. and Japanese patents for a method of planting a microchip into a false tooth. The one-chip microprocessor embedded in a plate denture can be detected using a radio transmitter-receiver, allowing its owner to be identified. This is useful in senior citizen’s home where all dentures are usually collected from their owners after meals, washed together and returned. In such a case, it is important to identify all the dentures to give back to their correct owners without any mistake.

In March this year, Advanced Micro Devices (AMD) launched its 1.3-GHz Athlon processor. Tests on this processor indicated that its speed surpassed Intel’s 1.5GHz Pentium 4. The Athlon processor has a 266-MHz front side bus that works with systems that use 266-MHz memory. The price starts from $2,988.

Intel’s Pentium 4, which was launched in late 2000, is designed to provide blazing speed—especially in handling multimedia content. Dubbed Intel NetBurst Microarchitecture, it is designed to speed up applications that send data in bursts, such as screaming media, MP3 playback, and video compression.

Even before the dust had settled on NetBurst, Intel released its much awaited 1.7 GHz Pentium 4 processor on Monday, April 23. The is said to be the company’s highest-performance microprocessor for desktops. Currently priced at $325 in 1,000 unit quantities. The vice president and general manager of Intel was quoted as saying, “the Pentium 4 processor is destined to become the center of the digital world. Whether encoding video and MP3 files, doing financial analysis, or experiencing the latest internet technologies—the Pentium 4 processor is designed to meet the needs of all users”.

Gordon Moore, co-founder of Intel, over thirty years ago, announced that the number of transistors that can be placed on a silicon would double every two years. Intel maintains that it has remained true since the release of its first processors, the 4004, in 1971.
The competition to determine who has produced the fastest and smallest processor between Intel and AMD continues. In fact, Intel Corp. predicts that PC chips will climb to more than 10GHz from today's 1GHz standard by the year 2011. However, researchers are paying increasing attention to software. That's because new generations of software, especially computing-intensive user interfaces, will call for processors with expanded capabilities and performance.

1.11 Summary

The microprocessor has become a formidable force in computing. From a humble beginning as a concept of reducing the price of a calculator to high powered, uniprocessor and multiprocessor machines in only two and a half decades is astounding pace. Like most classic inventions, its early years belong firmly to the start-ups and pre-pubescent companies. These didn't have the baggage of the established companies and grew quickly. However, the mid 1980s saw a changeover, mainly due to the spiralling cost of research into process technologies and the greater man-hours needed to implement hundreds of thousand transistors design. This was headed by Motorola, Intel, IBM and DEC. It is now acknowledged that the RISC concept is the superior architectural concept and all the aforementioned companies have leading designs using RISC.

The microprocessor was originally designed for a calculator, yet in recent years it has found its way into a multitude of designs. A seemingly exponential growth curve for applications has occurred. From cars to personal computers, televisions to telephones, the microprocessor proliferates, and the growth curve shows no signs of abating. This essay shows just part of the large history of the microprocessor and the path designs took. There are many other fields where the microprocessor has made a huge impact, not least in the low cost market, which deserve to be investigated further.
1.12 Keywords

- CPU
- LSI
- Semiconductors
- ALU
- TI

1.13 Exercise

1) What led to the development of microprocessors?
2) Explain how a microprocessor works.
3) Give Architecture of a microprocessor.
4) List the Generation of microprocessors.
5) List the Companies associated with microprocessors.

Unit 2

Advantages of Microprocessors

Structure

2.1 Introduction
2.2 Objectives
2.3 Microprocessor Advantages & Disadvantages
2.4 Microprocessor and it’s applications
2.5 Manufacturing of microprocessors
2.6 Performance of microprocessors
2.7 Other uses of microprocessors
2.8 Summary
2.9 Keywords
2.10 Exercise
2.1 Introduction

Advantages of Microprocessors is that these are general purpose electronic processing devices which can be programmed to execute a number of tasks. These are used in personal computers as well as a number of other embedded products. There are no disadvantages as such but when compared to fixed logic devices or certain ASICs (application specific integrated circuits), there is a need to program Microprocessors and write software/firmware when used in embedded applications.

2.2 Objectives

At the end of this chapter you will be able to

- List the Microprocessor Advantages & Disadvantages
- Know the Microprocessor and it’s applications
- Explain the Manufacturing of microprocessors
- List the Other uses of microprocessors

2.3 Microprocessor Advantages & Disadvantages

Microprocessor is defined as a silicon chip embedded with a Central Processing Unit or CPU. It is also referred to as a computer's logic chip, micro chip, and processor. Advantages of Microprocessors is that these are general purpose electronic processing devices which can be programmed to execute a number of tasks. These are used in personal computers as well as a number of other embedded products. There are no disadvantages as such but when compared to
fixed logic devices or certain ASICs (application specific intergrated circuits), there is a need to program Microprocessors and write software/firmware when used in embedded applications.

**Function**

The function of a Microprocessor is to conduct arithmetic and logic operations.

**Speed**

One advantage of a Microprocessor is its speed, which is measured in hertz. For instance, a Microprocessor with 3 gigahertz, shortly GHz, is capable of performing 3 billion tasks per second.

**Data Movement**

Another advantage of a Microprocessor is that it can quickly move data between the various memory locations.

**Complex Mathematics**

Microprocessors are used to perform complicated mathematical operations, like operating on the floating point numbers.

**Disadvantages**

Some of the disadvantages with the Microprocessor are that it might get over-heated, and the limitation it imposes on the size of data.

2.4 Microprocessor and it’s applications

The 4004 is the first microprocessor the word size is 4- bit, which was introduced in 1971 by Intel Corporation. The microprocessors introduced between 1971 and 1973 were the first-generation systems. It has P-type MOS technology, this is the low cost technology and slow speed.

In 1973, the second-generation microprocessors, Motorola 6800 and 6809, Intel 8085, and Zilog Z80 were evolved. This processor was fabricated using the N-type MOS technology. Which
offered faster speed and higher density than PMOS.

Intel 8085A was fabricated using HMOS technology. Thus, Intel offers a high-speed version of the 8085A called the 8085AH. The price of the 8085AH is higher than that of the 8085A.

After 1978, the third-generation microprocessors were introduced. The processors were Intel 8086/80186/80286 and the Motorola 68000/68010. These processors were 16 bits wide. Which were designed using the High-density MOS technology.

HMOS provides the following advantages over NMOS:

Speed-Power-Product (SPP) of HMOS is 4 times better than that of NMOS.
That is,

 NMOS = 4 Pico joules (PJ)
 HMOS = 1 Pico joules (PJ)

In 1980, the fourth-generation microprocessors evolved. Intel introduced 32-bit microprocessors 80386/80486, and the Motorola has introduced 68020/68030/68040 processors. These processors are fabricated using the low-power version of the HMOS technology called HCMOS, and they include an on-chip RAM called the cache memory to speed up program execution.
Intel 80960 and Motorola 88100 are RISC microprocessors. The trend in microprocessors is not toward introduction of 64-bit microprocessors.

Classification of Microprocessor:
The microprocessor is identified with the word size of data. For E.g. The ALU can perform a 4-bit data operation at a time these microprocessor is called as 4-bit microprocessor.

4-Bit Processors
INTEL 404
4040
8-Bit Processors
8008
8080
8085
MOTOROLA 6800 (M6800)

16-Bit Processors
8086
8088
Zilog Z800
80186
80286

32-Bit Processors
Intel 80386
80387
80486
PENTIUM
PENTIUM PRO

Advantages of Microprocessor:
1. Computational/Processing speed is high
2. Intelligence has been brought to systems
3. Automation of industrial process and office automation
4. Flexible.
5. Compact in size.
6. Maintenance is easier.

Applications of Microprocessors
Microprocessors are a mass storage device. They are the advanced form of computers. They are
also called as microcomputers.
The impact of microprocessor in different lures of fields is significant. The availability of low cost, low power and small weight, computing capability makes it useful in different applications. Now a days, a microprocessor based systems are used in instructions, automatic testing product, speed control of motors, traffic light control , light control of furnaces etc. Some of the important areas are mentioned below:

Instrumentation: it is very useful in the field of instrumentation. Frequency counters, function generators, frequency synthesizers, spectrum analyses and many other instruments are available, when microprocessors are used as controller. It is also used in medical instrumentation.

Control:
Microprocessor based controllers are available in home appliances, such as microwave oven, washing machine etc., microprocessors are being used in controlling various parameters like speed, pressure, temperature etc. These are used with the help of suitable transduction.

Communication:
Microprocessors are being used in a wide range of communication equipments. In telephone industry, these are used in digital telephone sets. Telephone exchanges and modem etc. The use of microprocessor in television, satellite communication have made teleconferencing possible. Railway reservation and air reservation system also uses this technology. LAN and WAN for communication of vertical information through computer network.

Office Automation and Publication:
Microprocessor based micro computer with software packages has changed the office environment. Microprocessors based systems are being used for word processing, spread sheet operations, storage etc. The microprocessor has revolutionize the publication technology.

Consumer: The use of microprocessor in toys, entertainment equipment and home applications is making them more entertaining and full of features. The use of microprocessors is more widespread and popular.
Now the Microprocessors are used in:
1. Calculators
2. Accounting system
3. Games machine
4. Complex Industrial Controllers
5. Traffic light Control
6. Data acquisition systems
7. Multi user, multi-function environments
8. Military applications
9. Communication systems.

### 2.5 Manufacturing of microprocessors

Economical manufacturing of microprocessors requires mass production. Microprocessors are constructed by depositing and removing thin layers of conducting, insulating, and semiconducting materials in hundreds of separate steps. Nearly every layer must be patterned accurately into the shape of transistors and other electronic elements. Usually this is done by photolithography, which projects the pattern of the electronic circuit onto a coating that changes when exposed to light. Because these patterns are smaller than the shortest wavelength of visible light, short wavelength ultraviolet radiation must be used. Microprocessor features are so small and precise that a single speck of dust can destroy the microprocessor. Microprocessors are made in filtered clean rooms where the air may be a million times cleaner than in a typical home (PC World, 2000)).

### 2.6 Performance of microprocessors

The number of transistors available has a huge effect on the performance of a processor. As seen earlier, a typical instruction in a processor like an 8088 took 15 clock cycles to execute. Because of the design of the multiplier, it took approximately 80 cycles just to do one 16-bit multiplication on the 8088. With more transistors, much more powerful multipliers capable of single-cycle speeds become possible.
More transistors also allow a technology called pipelining. In a pipelined architecture, instruction execution overlaps. So even though it might take 5 clock cycles to execute each instruction, there can be 5 instructions in various stages of execution simultaneously. That way it looks like one instruction completes every clock cycle.

Many modern processors have multiple instruction decoders, each with its own pipeline. This allows multiple instruction streams, which means more than one instruction can complete during each clock cycle. This technique can be quite complex to implement, so it takes lots of transistors.

The trend in processor design has been toward full 32-bit ALUs with fast floating point processors built in and pipelined execution with multiple instruction streams. There has also been a tendency toward special instructions (like the MMX instructions) that make certain operations particularly efficient. There has also been the addition of hardware virtual memory support and L1 caching on the processor chip. All of these trends push up the transistor count, leading to the multi-million transistor powerhouses available today. These processors can execute about one billion instructions per second.

With all the different types of Pentium microprocessors, what is the difference? Three basic characteristics stand out:

- Instruction set: The set of instructions that the microprocessor can execute.

- Bandwidth: The number of bits processed in a single instruction.

- Clock speed: Given in megahertz (MHz), the clock speed determines how many instructions per second the processor can execute.

In addition to bandwidth and clock speed, microprocessors are classified as being either RISC (reduced instruction set computer) or CISC (complex instruction set computer).
2.7 Other uses of microprocessors

There are many uses for microprocessors in the world today. Most appliances found around the house are operated by microprocessors. Most modern factories are fully automated—that means that most jobs are done by a computer. Automobiles, trains, subways, planes, and even taxi services require the use of many microprocessors. In short, there are microprocessors everywhere you go.

Another common place to find microprocessors is a car. This is especially applicable to sports cars. There are numerous uses for a microprocessor in cars. First of all, it controls the warning LED signs. Whenever there is a problem, low oil, for example, it has detectors that tell it that the oil is below a certain amount. It then reaches over and starts blinking the LED until the problem is fixed. Another use is in the suspension system. A processor, controls the amount of pressure applied to keep the car leveled. During turns, a processor, slows down the wheels on the inner side of the curb and speeds them up on the outside to keep the speed constant and make a smooth turn.

An interesting story appeared in the New York Times dated April 16 and goes to show that there’s no limit to what microprocessors can do and that researchers and scientists are not stopping at the current uses of microprocessors. The next time the milk is low in the refrigerator, the grocery store may deliver a new gallon before it is entirely gone. Masahiro S one, who lives in Raleigh, N.C., has won a patent for a refrigerator with an inventory processing system that keeps track of what is inside and what is about to run out and can ring up the grocery store to order more.

2.8 Summary

The impact of microprocessor in different lures of fields is significant. The availability of low cost, low power and small weight, computing capability makes it useful in different applications. Now a days, a microprocessor based systems are used in instructions, automatic testing product, speed control of motors, traffic light control, light control of furnaces etc.
2.9 Keywords
- ASICs
- Data Movement
- SPP
- ALU
- LAN
- WAN

2.10 Exercise
1) List the Microprocessor Advantages & Disadvantages.
2) Explain the Microprocessor and it’s applications.
3) Explain the Manufacturing of microprocessors.
4) List the Other uses of microprocessors.

Unit 3

Various MPU Families

Structure
3.1. Introduction
3.2. Objectives
3.3. The Microprocessor
3.4. Microcomputer - Microprocessor Unit
3.5. Microcomputer - Memory
3.6. Microcomputer - Microcomputer Systems
3.7. Applications - Computer Applications
3.8. How Microprocessors Work
3.9. Various MPU Families (SSI, LSI, VLSI, SLSI)
3.10. Summary
3.11. Keywords
3.12. Exercise
3.1 Introduction

Microprocessor, integrated circuit containing the arithmetic, logic, and control circuitry required to interpret and execute instructions from a computer program. When combined with other integrated circuits that provide storage for data and programs, often on a single semiconductor base to form a chip, the microprocessor becomes the heart of a small computer, or microcomputer. Microprocessors are classified by the semiconductor technology of their design (TTL, transistor-transistor logic; CMOS, complementary-metal-oxide semiconductor; or ECL, emitter-coupled logic), by the width of the data format (4-bit, 8-bit, 16-bit, 32-bit, or 64-bit) they process; and by their instruction set (CISC, complex-instruction-set computer, or RISC, reduced-instruction-set computer; see RISC processor). TTL technology is most commonly used, while CMOS is favored for portable computers and other battery-powered devices because of its low power consumption. ECL is used where the need for its greater speed offsets the fact that it consumes the most power. Four-bit devices, while inexpensive, are good only for simple control applications; in general, the wider the data format, the faster and more expensive the device. CISC processors, which have 70 to several hundred instructions, are easier to program than RISC processors, but are slower and more expensive.

Developed during the 1970s, the microprocessor became most visible as the central processor of the personal computer. Microprocessors also play supporting roles within larger computers as smart controllers for graphics displays, storage devices, and high-speed printers. However, the vast majority of microprocessors are used to control everything from consumer appliances to smart weapons. The microprocessor has made possible the inexpensive hand-held electronic calculator, the digital wristwatch, and the electronic game. Microprocessors are used to control consumer electronic devices, such as the programmable microwave oven and videocassette recorder; to regulate gasoline consumption and antilock brakes in automobiles; to monitor alarm systems; and to operate automatic tracking and targeting systems in aircraft, tanks, and missiles and to control radar arrays that track and identify aircraft, among other defense applications.
3.2 Objectives

At the end of this chapter you will be able to:

- Explain the Microprocessor
- Define RAM, ROM.
- Explain Microcomputer system
- Know How Microprocessors Work
- List Various MPU Families

3.3 The Microprocessor

The term microprocessor typically refers to the central processing unit (CPU) of a microcomputer, containing the arithmetic logic unit (ALU) and the control units. It is typically implemented on a single LSI chip. This separates the "brains" of the operation from the other units of the computer.

An example of microprocessor architecture.

The microprocessor contains the arithmetic logic unit (ALU) and the control unit for a microcomputer. It is connected to memory and I/O by buses which carry information between the units.
3.4 Microcomputer - Microprocessor Unit

The advent of the microprocessor followed on from advances in the field of large scale integration (LSI) technology first investigated in the late 1950s. The technology for manufacturing integrated circuits improved over the next twenty years and in 1971 the first microprocessor was produced on a single chip (wafer thin slice) of Silicon.

Today, many thousands of components can be built onto a single chip some 5 mm square and no more than 1 mm thick and the density of circuits is increasing all the time. A single chip can contain all the circuitry to perform the combined functions of the control unit and arithmetic/logic unit of the traditionally structured computer. Such a chip is called a microprocessor unit (MPU). It is not a complete computer as it lacks memory and input/output capability.

3.5 MICROCOMPUTER - Memory
Memory can also be contained on a chip which like the microprocessor chip is very small and cheap to manufacture when produced in large volumes. Memory may also be assembled on the same chip as the MPU but for the purpose of our description the memory is treated as a separate entity.

**Random Access Memory (RAM)**

There are various types of memory chips and a single microcomputer might utilize more than one type. The most common is random access memory. Like the memory of the traditional computer, information can be 'read' from a RAM chip and 'written' to it. When switched off, any information stored in the memory is lost. RAM behaves exactly as the Main memory.

![Fig.3.2](image)

**Read only memory (ROM)**

Another type of microcomputer memory is read only memory. Information is 'burnt' into the ROM chip at manufacturing time. It cannot be altered and fresh information cannot be 'written' into a RAM. The information can be 'read' and transferred for use elsewhere, for example to a RAM. When the power supply is switched off, the bit patterns in the memory are not lost as they are in the case of the central memory of a computer or a RAM chip. ROMs are used for applications in which it is known that the information never needs to be altered, for example the operating system software which controls the use of a complete microcomputer system or a monitor program for controlling a washing machine.
Programmable Read Only Memory (PROM)

A variation of the ROM chip is programmable read only memory. PROM can be programed to record information using a facility known as a prom-programr. However, once the chip has been programd the recorded information cannot be changed, i.e. the PROM becomes a ROM and the stored information can only be 'read'. PROM is also non-volatile storage.

Erasable Programmable Read Only Memory (EPROM)

A fourth type of memory is erasable programmable read only memory. As the name suggests, information can be erased and the chip programd anew to record different information using a special prom-programr facility. ERASURE is achieved by exposing the chip to ultraviolet light. When an EPROM is in use information can only be 'read' and the information remains on the chip until it is erased.

3.6 Microcomputer - Microcomputer Systems

The present generation of microcomputers is based on the use of 8-bit microprocessors, i.e. information is handled within the MPU in 8-bit registers, accumulators etc. There is also a rapidly evolving new generation built around 16-bit microprocessors. These more powerful microcomputers provide minicomputer capability at micro prices.

A microcomputer is often designed so that it can be configured with different amounts of RAM
(the equivalent of central memory). Typical ranges are 4K bytes to 64 K bytes and 32 K to 512 K bytes, expandable normally in add on steps of 4K and 16K or multiples of the same. The memory may be built up by adding boards containing RAM chips or by replacing an existing board with another board containing RAMs of a bigger capacity. The concept of add on and replacement boards is fundamental to the design of many microcomputers. It enables systems to be enhanced easily and faults to be remedied quickly are replacement of a complete unit in which a problem is known to lie. Maintenance by replacement is only feasible because of the comparatively low cost involved. Many of the replaced boards may be subsequently corrected for use again.

A microcomputer is usually equipped with a typewriter keyboard for input and can normally be linked to a CRT monitor device (cathode ray tube) for display of input and output. A terminal printer device may also for display of input and output. A terminal printer device may also be connected to provide printed copy of output on paper. Fully configured systems often include a floppy disk drive, and even a hard disk system (a Winchester drive) or a cartridge tape system which separately or collectively provide extra storage space for back-up purposes.

### 3.7 Applications - Computer Applications

The computer is used to assist man in business organizations, in research and in many other walks of life. In this chapter we shall examine some of these areas so as to give an indication of the very wide range of activities in which the computer is involved. Some may be surprising, if you consider the limited capabilities of the machine. However, versatility has been provided by man's ability to reduce what are often highly complicated problems to the simple level at which the computer can be used, and to design and implement ingenious computer system which can provide a myriad interplay of the basically simple tasks that the computer can handle.

You should remember that what has been achieved so far has been accomplished in a very short period of time. The first computer was developed about fifty years ago and ten years passed before the industry was established on a firm footing.
### 3.8 How Microprocessors Work

The computer you are using to read this page uses a **microprocessor** to do its work. The microprocessor is the heart of any normal computer, whether it is a **desktop machine**, a **server** or a **laptop**. The microprocessor you are using might be a Pentium, a K6, a PowerPC, a Spar or any of the many other brands and types of microprocessors, but they all do approximately the same thing in approximately the same way.

A microprocessor -- also known as a **CPU** or central processing unit -- is a complete computation engine that is fabricated on a single chip. The first microprocessor was the Intel 4004, introduced in 1971. The 4004 was not very powerful -- all it could do was add and subtract, and it could only do that 4 **bits** at a time. But it was amazing that everything was on one chip. Prior to the 4004, engineers built computers either from collections of chips or from discrete components (**transistors** wired one at a time). The 4004 powered one of the first portable electronic calculators.

**Microprocessor Progression: Intel**

![Intel 8080](image)

*Fig.3.4: The Intel 8080 was the first microprocessor in a home computer*

The first microprocessor to make it into a home computer was the Intel 8080, a complete 8-bit computer on one chip, introduced in 1974. The first microprocessor to make a real splash in the market was the Intel 8088, introduced in 1979 and incorporated into the IBM PC (which first appeared around 1982). If you are familiar with the PC market and its history, you know that the PC market moved from the 8088 to the 80286 to the 80386 to the 80486 to the Pentium to the Pentium II to the Pentium III to the Pentium 4. All of these microprocessors are made by Intel and all of them are improvements on the basic design of the 8088. The Pentium 4 can execute any piece of code that ran on the original 8088, but it does it about 5,000 times faster!
The following table helps you to understand the differences between the different processors that Intel has introduced over the years.

<table>
<thead>
<tr>
<th>Name</th>
<th>Date</th>
<th>Transistors</th>
<th>Microns</th>
<th>Clock speed</th>
<th>Data width</th>
<th>MIPS</th>
</tr>
</thead>
<tbody>
<tr>
<td>8080</td>
<td>1974</td>
<td>6,000</td>
<td>6</td>
<td>2 MHz</td>
<td>8 bits</td>
<td>0.64</td>
</tr>
<tr>
<td>8088</td>
<td>1979</td>
<td>29,000</td>
<td>3</td>
<td>5 MHz</td>
<td>16 bits</td>
<td>0.33</td>
</tr>
<tr>
<td>80286</td>
<td>1982</td>
<td>134,000</td>
<td>1.5</td>
<td>6 MHz</td>
<td>16 bits</td>
<td>1</td>
</tr>
<tr>
<td>80386</td>
<td>1985</td>
<td>275,000</td>
<td>1.5</td>
<td>16 MHz</td>
<td>32 bits</td>
<td>5</td>
</tr>
<tr>
<td>80486</td>
<td>1989</td>
<td>1,200,000</td>
<td>1</td>
<td>25 MHz</td>
<td>32 bits</td>
<td>20</td>
</tr>
<tr>
<td>Pentium</td>
<td>1993</td>
<td>3,100,000</td>
<td>0.8</td>
<td>60 MHz</td>
<td>32 bits</td>
<td>100</td>
</tr>
<tr>
<td>Pentium II</td>
<td>1997</td>
<td>7,500,000</td>
<td>0.35</td>
<td>233 MHz</td>
<td>32 bits</td>
<td>~300</td>
</tr>
<tr>
<td>Pentium III</td>
<td>1999</td>
<td>9,500,000</td>
<td>0.25</td>
<td>450 MHz</td>
<td>32 bits</td>
<td>~510</td>
</tr>
<tr>
<td>Pentium 4</td>
<td>2000</td>
<td>42,000,000</td>
<td>0.18</td>
<td>1.5 GHz</td>
<td>32 bits</td>
<td>~1,700</td>
</tr>
<tr>
<td>Microprocessor Logic</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

| Pentium 4         | 2004 | 125,000,000 | 0.09 | 3.6 GHz | 32 bits | 64-bit bus | ~7,000 |

Information about this table:

- **The date** is the year that the processor was first introduced. Many processors are re-introduced at higher clock speeds for many years after the original release date.
- **Transistors** is the number of transistors on the chip. You can see that the number of transistors on a single chip has risen steadily over the years.
- **Microns** is the width, in microns, of the smallest wire on the chip. For comparison, a human hair is 100 microns thick. As the feature size on the chip goes down, the number of transistors rises.
- **Clock speed** is the maximum rate that the chip can be clocked at. Clock speed will make more sense in the next section.
- **Data Width** is the width of the ALU. An 8-bit ALU can add/subtract/multiply/etc. two 8-bit numbers, while a 32-bit ALU can manipulate 32-bit numbers. An 8-bit ALU would have to execute four instructions to add two 32-bit numbers, while a 32-bit ALU can do it in one instruction. In many cases, the external data bus is the same width as the ALU, but not always. The 8088 had a 16-bit ALU and an 8-bit bus, while the modern Pentiums fetch data 64 bits at a time for their 32-bit ALUs.
- **MIPS** stands for "millions of instructions per second" and is a rough measure of the performance of a CPU. Modern CPUs can do so many different things that MIPS ratings lose a lot of their meaning, but you can get a general sense of the relative power of the CPUs from this column.
To understand how a microprocessor works, it is helpful to look inside and learn about the logic used to create one. In the process you can also learn about **assembly language** – the native language of a microprocessor – and many of the things that engineers can do to boost the speed of a processor.

A microprocessor executes a collection of machine instructions that tell the processor what to do. Based on the instructions, a microprocessor does three basic things:

- Using its ALU (Arithmetic/Logic Unit), a microprocessor can perform mathematical operations like addition, subtraction, multiplication and division. Modern microprocessors contain complete floating point processors that can perform extremely sophisticated operations on large floating point numbers.
- A microprocessor can move data from one memory location to another.
- A microprocessor can make decisions and jump to a new set of instructions based on those decisions.

There may be very sophisticated things that a microprocessor does, but those are its three basic activities. The following diagram shows an extremely simple microprocessor capable of doing those three things:
This is about as simple as a microprocessor gets. This microprocessor has:

- An **address bus** (that may be 8, 16 or 32 bits wide) that sends an address to memory
- A **data bus** (that may be 8, 16 or 32 bits wide) that can send data to memory or receive data from memory
- An **RD** (read) and **WR** (write) line to tell the memory whether it wants to set or get the addressed location
- A **clock line** that lets a clock pulse sequence the processor
- A **reset line** that resets the program counter to zero (or whatever) and restarts execution

Let's assume that both the address and data buses are 8 bits wide in this example.

Here are the components of this simple microprocessor:

- Registers A, B and C are simply latches made out of flip-flops. (See the section on "edge-triggered latches" in How Boolean Logic Works for details.)
- The address latch is just like registers A, B and C.
- The program counter is a latch with the extra ability to increment by 1 when told to do so, and also to reset to zero when told to do so.
- The ALU could be as simple as an 8-bit adder (see the section on adders in How Boolean Logic Works for details), or it might be able to add, subtract, multiply and divide 8-bit values. Let's assume the latter here.
- The test register is a special latch that can hold values from comparisons performed in the ALU. An ALU can normally compare two numbers and determine if they are equal, if one is greater than the other, etc. The test register can also normally hold a carry bit from the last stage of the adder. It stores these values in flip-flops and then the instruction decoder can use the values to make decisions.
- There are six boxes marked "3-State" in the diagram. These are tri-state buffers. A tri-state buffer can pass a 1, a 0 or it can essentially disconnect its output (imagine a switch that totally disconnects the output line from the wire that the output is heading toward). A tri-state buffer allows multiple outputs to connect to a wire, but only one of them to actually drive a 1 or a 0 onto the line.
- The instruction register and instruction decoder are responsible for controlling all of the other components.

Although they are not shown in this diagram, there would be control lines from the instruction decoder that would:

- Tell the A register to latch the value currently on the data bus
- Tell the B register to latch the value currently on the data bus
- Tell the C register to latch the value currently output by the ALU
- Tell the program counter register to latch the value currently on the data bus
- Tell the address register to latch the value currently on the data bus
- Tell the instruction register to latch the value currently on the data bus
- Tell the program counter to increment
- Tell the program counter to reset to zero
- Activate any of the six tri-state buffers (six separate lines)
- Tell the ALU what operation to perform
- Tell the test register to latch the ALU's test bits
• Activate the RD line
• Activate the WR line

Coming into the instruction decoder are the bits from the test register and the clock line, as well as the bits from the instruction register.

3.9 Various MPU Families (SSI, LSI, VLSI, SLSI)

A Digital Logic Gate is an electronic device that makes logical decisions based on the different combinations of digital signals present on its inputs. A digital logic gate may have more than one input but only has one digital output. Standard commercially available digital logic gates are available in two basic families or forms, TTL which stands for Transistor-Transistor Logic such as the 7400 series, and CMOS which stands for Complementary Metal-Oxide-Silicon which is the 4000 series of chips. This notation of TTL or CMOS refers to the logic technology used to manufacture the integrated circuit, (IC) or a "chip" as it is more commonly called.

![Digital Logic Gate](image)

**Fig.3.7**

**Digital Logic Gate**

Generally speaking, TTL IC's use NPN (or PNP) type Bipolar Junction Transistors while CMOS IC's use Field Effect Transistors or FET's for both their input and output circuitry. As well as TTL and CMOS technology, simple digital logic gates can also be made by connecting together diodes, transistors and resistors to produce RTL, Resistor-Transistor logic gates, DTL, Diode-Transistor logic gates or ECL, Emitter-Coupled logic gates but these are less common now compared to the popular CMOS family.

**Integrated Circuits** or IC's as they are more commonly called, can be grouped together into families according to the number of transistors or "gates" that they contain. For example, a
simple AND gate may contain only a few individual transistors, whereas a more complex microprocessor may contain many thousands of individual transistor gates. Integrated circuits are categorised according to the number of logic gates or the complexity of the circuits within a single chip with the general classification for the number of individual gates given as:

**Classification of Integrated Circuits**

- **Small Scale Integration or (SSI)** - Contain up to 10 transistors or a few gates within a single package such as AND, OR, NOT gates.

- **Medium Scale Integration or (MSI)** - between 10 and 100 transistors or tens of gates within a single package and perform digital operations such as adders, decoders, counters, flip-flops and multiplexers.

- **Large Scale Integration or (LSI)** - between 100 and 1,000 transistors or hundreds of gates and perform specific digital operations such as I/O chips, memory, arithmetic and logic units.

- **Very-Large Scale Integration or (VLSI)** - between 1,000 and 10,000 transistors or thousands of gates and perform computational operations such as processors, large memory arrays and programmable logic devices.

- **Super-Large Scale Integration or (SLSI)** - between 10,000 and 100,000 transistors within a single package and perform computational operations such as microprocessor chips, micro-controllers, basic PICs and calculators.

- **Ultra-Large Scale Integration or (ULSI)** - more than 1 million transistors - the big boys that are used in computers CPUs, GPUs, video processors, micro-controllers, FPGAs and complex PICs.

While the "ultra large scale" ULSI classification is less well used, another level of integration which represents the complexity of the Integrated Circuit is known as the **System-on-Chip** or (SOC) for short. Here the individual components such as the microprocessor, memory, peripherals, I/O logic etc, are all produced on a single piece of silicon and which represents a
whole electronic system within one single chip, literally putting the word "integrated" into integrated circuit. These chips are generally used in mobile phones, digital cameras, microcontrollers, PICs and robotic applications, and which can contain up to 100 million individual silicon-CMOS transistor gates within one single package.

3.10 Summary

The term microprocessor typically refers to the central processing unit (CPU) of a microcomputer, containing the arithmetic logic unit (ALU) and the control units. It is typically implemented on a single LSI chip. This separates the "brains" of the operation from the other units of the computer. There are various types of memory chips and a single microcomputer might utilize more than one type. The most common is random access memory. Like the memory of the traditional computer, information can be 'read' from a RAM chip and 'written' to it. When switched off, any information stored in the memory is lost. RAM behaves exactly as the Main memory.

3.11 Keywords

- TTL
- CMOS
- RISC
- ECL
- CISC
- CPU
- ALU
- LSI
- RAM
- ROM
- PROM
- SSI
- MSI
3.12 Exercise

1) Explain the Microprocessor
2) Define RAM, ROM.
3) Explain Micro computer system
4) How Microprocessors Work
5) List Various MPU Families

Unit 1

Introduction

Structure
1.1 Introduction
1.2 Objectives
1.3 Functional Components of a Microprocessor
1.4 Introduction to Microprocessor Operation
1.5 Summary
1.6 Keywords
1.7 Exercise
1.1 Introduction

Microprocessor is a Central Processing Unit (CPU) etched on a single chip. A single Integrated Circuit (IC) has all the functional components of a CPU namely Arithmetic Logic Unit (ALU), Control Unit and registers. The 8085 microprocessor is an 8-bit processor that includes on its chip most of the logic circuitry for performing computing tasks and for communicating with peripherals. The architecture of a microprocessor is to be learnt in terms of registers, memory addressing, addressing modes, instruction set, interfacing with memory and Input and Output (I/O) devices and interrupt handling. It is necessary to learn about the above mentioned concepts to write efficient assembly language programs, and to design microprocessor based systems. This unit gives you an overall idea about the microprocessors, the detailed discussion about 8085 architecture and interfacing of 8085 with Programmable Peripheral Interface (PPI) devices.

1.2 Objectives

At the end of this chapter you will be able to:

- Explain Functional Components of a Microprocessor
- Give Introduction to Microprocessor Operation

1.3 Functional Components of a Microprocessor

A digital computer is a programmable machine specially designed for making computation. Its main components are: CPU (Central Processing Unit), memory, input device and output device as shown in figure 1.1.
A microcomputer is a small digital computer. The CPU of a microcomputer is a microprocessor. Other components are same as those of any other digital computer. In figure 1.1, if we change the label CPU as Microprocessor, we get the organization of a microcomputer.

The physical devices and circuitry of a computer are called hardware. A physical device may be electronic, magnetic, mechanical or an optical device etc. A sequence of instructions to perform a particular task is called a program. A set of programs written for a particular computer is known as software for that computer. The input and output devices are known as peripherals. Sometimes the term peripheral also includes memory. Programs are subroutines stored in ROM (Read Only Memory)s, Programmable ROM (PROM)s, Erasable PROM(EPROM)s and/or EEPROMs are known as firmware. The commonly available firmwares are: monitors, microprograms, subroutines for input and output devices.

The Central Processing Unit (CPU) fetches instructions from the memory and performs specified tasks. It stores results in the memory or sends results to the output device according to the instructions given in the program. The CPU controls and communicates with memory and input/output devices. Under the control of the CPU, programs and data are stored in the memory and displayed on Cathode Ray Tube (CRT). The schematic diagram of a CPU is shown in figure...
The CPU of a large computer is implemented on one or more circuit boards. ICs are used as its components. Recent practice is to use microprocessors to perform different functions within the CPU of a large computer. The major sections of a CPU are Arithmetic and Logic Unit (ALU), Accumulator, General and Special purpose registers and Timing and Control Unit. The function of an ALU is to perform arithmetic operations such as addition and subtraction; and logical operations such as AND, OR and EXCLUSIVE-OR. Timing and control unit controls the entire operations of a computer. It acts as a brain. It also controls all other devices connected to the CPU. It generates timing signals necessary for input and output devices. The accumulator is a register, which contains one of the operands and stores results of most arithmetic and logical operations. General purpose registers are used for temporary storage of data and intermediate results while computer is making execution of a program. Special purpose registers are used by the microprocessor itself. Some of them are not accessible to programmers. Examples of special purpose registers are program counter, stack pointer, instruction register and status register.

The memory is a storage device. It stores program, data, results etc. The computer receives data and instructions through input devices. An input device converts instructions, input data and signals into proper binary form suitable for a digital computer. A key-board and simple switches

<table>
<thead>
<tr>
<th>ALU</th>
<th>ACCUMULATOR</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>GENERAL AND</td>
</tr>
<tr>
<td></td>
<td>SPECIAL PURPOSE</td>
</tr>
<tr>
<td></td>
<td>REGISTERS</td>
</tr>
<tr>
<td>TIMING AND CONTROL UNIT</td>
<td></td>
</tr>
</tbody>
</table>
are used as input devices. The user enters instructions and data through a keyboard or simple switches. Computers are also used to measure and control physical quantities like temperature, pressure, speed, position etc. For these purposes transducers are used to convert physical quantities into proportional electrical signals. A/D converters are used to convert analog electrical signals into digital signals, which are sent to the computer. Transducers and sensors, data acquisition system etc. are also included in input devices. A/D converter forms a part of data acquisition system.

The computer sends results to output devices. An output device may store, print, display or send electrical signal to control/actuate certain equipment. The examples of simple output devices are printers, CRT, LEDs, D/A converter, controllers, actuators etc. Sometimes input and output devices may be combined in a single unit, which acts as both an input as well as an output device. A keyboard and CRT are combined to form a video terminal, which is a common I/O device for human interaction with a computer.

With the advent of LSI and VLSI technology it became possible to build the entire CPU on a single chip IC. A CPU built into a single LSI/VLSI chip is called a microprocessor. A digital computer using microprocessor as its CPU is called a microcomputer. The term micro initiates its physical size; not it’s computing power. Today the computing power of a powerful microprocessor approaches that a CPU on earlier large computer. The main sections of a microprocessor are: ALU, timing and control unit, accumulator, general purpose and special purpose registers. In this subject we’ll study about two microprocessors namely Intel 8085 (8-bit) and Intel 8086 (16-bit).

1.4 Introduction to Microprocessor Operation

The microprocessor works by executing a program of instructions. Creating the program is similar in concept to programming in BASIC, C, or any other high-level computer language. Each type of microprocessor has its own instruction set which is the set of commands that it was designed to recognize and obey. Microprocessor instructions are very elemental and specific, and it usually takes more than one to accomplish what a single, high-level language instruction would. Many microprocessor instructions simply move data from one place to another within the
computer; others perform mathematical or logic operations. Still another group of instructions control program flow, such as jumping forward or backward in the program. Each instruction in the instruction set is assigned its own unique operation code, (which is typically 8 bits long and referred to as the op, code). The CPU uses this 8-bit number to identify the instruction. All microprocessors have at least one accumulator [Figure 1.3(a)], which is a data holding register in the CPU. The accumulator acts as a "staging area" for data. It is common for data coming to the CPU to go first to the accumulator where it can be operated on. Similarly, most data leaving the CPU exits from the accumulator. Mathematical operations usually store the result in the accumulator. Many of the instructions involve the accumulator in one way or another.

A machine language program is a list of instructions (in op-code form) for the microprocessor to follow. Before the program can be executed, it must first be loaded sequentially into memory. The op code for the first instruction is loaded at the first address location, the op-code for the second instruction is loaded next in line, and so on.

Figure 1.3(b) shows a section of memory with a short program loaded in. The program listing includes the address, op-code, mnemonic, and a brief explanation. (A mnemonic is an English abbreviation of an instruction. A program listing using only mnemonics is called assembly language.) The program in Figure 1.3(b) directs the CPU to get 1 byte of data from input port 01, add 1 to it, and send the result to output port 02. Before execution can start, the address of the first instruction must be loaded into the program counter. The program counter is a special address-storage register that the CPU uses to keep track of where it is in the program, much like a bookmark. The program counter always holds the address of the next instruction to be executed. Once the
Fig. 1.3: The CPU uses an accumulator and a program counter to execute a simple program. Microprocessor is activated, execution of the program is completely automatic. The execution process is a series of fetch-execute cycles, where by the microprocessor first fetches the instruction from memory and then executes it. The following are the specific steps the microprocessor would go through to execute the program of Figure 1.3(b):

1) The microprocessor fetches the first instruction from memory. It knows where to find the instruction because its address is in the program counter.

2) Once in the CPU, the op-code is decoded to see which instruction it is, then the proper hardware is activated to execute this instruction. In the example program of Figure 1.3(b), the 615t instruction (IN 01) is 2 bytes long. The first byte of the instruction is the op-code, telling the CPU to input data from a Port. The second byte of the instruction tells the CPU which port to read from. Execution of this instruction causes data from input port 01 to travel along the data bus to the accumulator. Also, the program counter advances to 02 (the address of the next instruction) Execution of the first instruction is now complete.

3) The next fetch-execute cycle starts, this time fetching the instruction from address 02. The new instruction (INR A) is "increment the accumulator" so the accumulator is sent...
to the ALU to be incremented (add 1) and the result put back in the accumulator. The program counter advances to 03, which is the address of the next instruction.

4) The next fetch-execute cycle starts, this time fetching the instruction from address 03. The instruction (OUT 02) is executed causing the accumulator data to be sent to output port 02.

5) The final instruction is fetched. It is a "halt," which causes the microprocessor to cease operating and go into a wait mode.

1.5 Summary
Microprocessor is a Central Processing Unit (CPU) etched on a single chip. A single Integrated Circuit (IC) has all the functional components of a CPU namely Arithmetic Logic Unit (ALU), Control Unit and registers. The 8085 microprocessor is an 8-bit processor that includes on its chip most of the logic circuitry for performing computing tasks and for communicating with peripherals. Microprocessor instructions are very elemental and specific, and it usually takes more than one to accomplish what a single, high-level language instruction would. Many microprocessor instructions simply move data from one place to another within the computer; others perform mathematical or logic operations.

1.6 Keywords
- CPU
- ALU
- IC
- PPI
- Microprocessor Operation

1.7 Exercise
1. What are the major components of a digital computer?
2. What are the functional components of a CPU?
3. What is a microprocessor?

Unit 2
Architecture of 8085

Structure
2.1 Introduction
2.2 Objectives
2.3 Stack and stack pointer.
2.4 Architecture
2.5 The 8085 Programming Model
2.6 Summary
2.7 Keywords
2.8 Exercise

2.1 Introduction

Intel 8085 is an 8-bit, N-channel Metal Oxide semiconductor (NMOS) microprocessor. It is a 40 pin IC package fabricated on a single Large Scale Integration (LSI) chip. The Intel 8085 uses a single +5V DC supply for its operation. Its clock speed is about 3MHz. The clock cycle is of 320 ns. The time for the clock cycle of the Intel 8085 is 200 ns. It has 80 basic instructions and 246
The 8085 is an enhanced version of its predecessor, the 8080A; its instruction set is upward compatible with that of the 8080A, meaning that 8085 instruction set includes all the 8080A instructions plus some additional ones. Programs written for 8080A will be executed by 8085, but the 8085 and 8080A are not pin compatible.

2.2 Objectives
At the end of this chapter you will be able to:
- Define Stack and stack pointer.
- Explain the Architecture of 8085
- Know the 8085 Programming Model

2.3 Stack and stack pointer.
Stack: A small portion of the RAM memory is declared as stack and it is used for temporary storage of the register contents, using instructions like PUSH and POP. The contents are stored and retrieved in LIFO (Last In First Out) form. Stack Pointer: It is a 16-bit memory pointing register, having the last address of the stack in RAM.

2.4 Architecture
The architecture of Intel 8085 consists of three main sections, arithmetic and logic unit, timing and control unit and several registers. The functional block diagram of 8085 is shown in figure 1.3. These important sections are described in the subsequent sections.

Arithmetic and Logic Unit (ALU)
The ALU performs the following arithmetic and logical operations.
1. Addition
2. Subtraction
3. Logical AND
4. Logical OR
5. Logical EXCLUSIVE OR
6. Complement (logical NOT)
7. Increment (add 1)
8. Decrement (subtract 1)
9. Left shift
10. Clear

The ALU is the unit that manipulates the data. ALU includes the accumulator, the temporary register, the arithmetic and logic circuits and flags. The ALU performs the actual numerical and logic operation such as ‘add’, ‘subtract’, ‘AND’, ‘OR’, etc. Uses data from memory and from Accumulator to perform arithmetic. Always stores result of operation in Accumulator.
**Control Unit**
Generates signals within uP to carry out the instruction, which has been decoded. In reality causes certain connections between blocks of the uP to be opened or closed, so that data goes where it is required, and so that ALU operations occur.

**Registers**
The 8085/8080A-programming model includes six registers, one accumulator, and one flag register, as shown in Figure. In addition, it has two 16-bit registers: the stack pointer and the program counter. They are described briefly as follows.

The 8085/8080A has six general-purpose registers to store 8-bit data; these are identified as B,C,D,E,H, and L as shown in the figure. They can be combined as register pairs - BC, DE, and HL - to perform some 16-bit operations. The programmer can use these registers to store or copy data into the registers by using data copy instructions.

**Accumulator**
The accumulator is an 8-bit register that is a part of arithmetic/logic unit (ALU). This register is used to store 8-bit data and to perform arithmetic and logical operations. The result of an operation is stored in the accumulator. The accumulator is also identified as register A.

**Flags**
The ALU includes five flip-flops, which are set or reset after an operation according to data conditions of the result in the accumulator and other registers. They are called Zero(Z), Carry (CY), Sign (S), Parity (P), and Auxiliary Carry (AC) flags; they are listed in the Table and their bit positions in the flag register are shown in the Figure below. The most commonly used flags are Zero, Carry, and Sign. The microprocessor uses these flags to test data conditions.

For example, after an addition of two numbers, if the sum in the accumulator id larger than eight bits, the flip-flop uses to indicate a carry -- called the Carry flag (CY) -- is set to one. When an
arithmetic operation results in zero, the flip-flop called the Zero(Z) flag is set to one. The first Figure shows an 8-bit register, called the flag register, adjacent to the accumulator. However, it is not used as a register; five bit positions out of eight are used to store the outputs of the five flip-flops. The flags are stored in the 8-bit register so that the programmer can examine these flags (data conditions) by accessing the register through an instruction.

These flags have critical importance in the decision-making process of the micro-processor. The conditions (set or reset) of the flags are tested through the software instructions. For example, the instruction JC (Jump on Carry) is implemented to change the sequence of a program when CY flag is set. The thorough understanding of flag is essential in writing assembly language programs.

**Program Counter (PC)**
This 16-bit register deals with sequencing the execution of instructions. This register is a memory pointer. Memory locations have 16-bit addresses, and that is why this is a 16-bit register.

The microprocessor uses this register to sequence the execution of the instructions. The function of the program counter is to point to the memory address from which the next byte is to be fetched. When a byte (machine code) is being fetched, the program counter is incremented by one to point to the next memory location.

**Stack Pointer (SP)**
The stack pointer is also a 16-bit register used as a memory pointer. It points to a memory location in R/W memory, called the stack. The beginning of the stack is defined by loading 16-bit address in the stack pointer. The stack concept is explained in the chapter "Stack and Subroutines."

**Instruction Register/Decoder**
Temporary store for the current instruction of a program. Latest instruction sent here from memory prior to execution. Decoder then takes instruction and ‘decodes’ or interprets the instruction. Decoded instruction then passed to next stage.
Memory Address Register
Holds address, received from PC, of next program instruction. Feeds the address bus with addresses of location of the program under execution.

Control Generator
Generates signals within uP to carry out the instruction which has been decoded. In reality causes certain connections between blocks of the uP to be opened or closed, so that data goes where it is required, and so that ALU operations occur.

Register Selector
This block controls the use of the register stack in the example. Just a logic circuit which switches between different registers in the set will receive instructions from Control Unit.

General Purpose Registers
uP requires extra registers for versatility. Can be used to store additional data during a program. More complex processors may have a variety of differently named registers.

Microprogramming
How does the uP knows what an instruction means, especially when it is only a binary number? The microprogram in a uP/uC is written by the chip designer and tells the uP/uC the meaning of each instruction uP/uC can then carry out operation.

2.5 The 8085 Programming Model
In the previous tutorial we described the 8085 microprocessor registers in reference to the internal data operations. The same information is repeated here briefly to provide the continuity and the context to the instruction set and to enable the readers who prefer to focus initially on the programming aspect of the microprocessor.
The 8085 programming model includes six registers, one accumulator, and one flag register, as shown in Figure. In addition, it has two 16-bit registers: the stack pointer and the program counter. They are described briefly as follows.

<table>
<thead>
<tr>
<th>ACCUMULATOR</th>
<th>FLAG REGISTER</th>
</tr>
</thead>
<tbody>
<tr>
<td>A (8)</td>
<td></td>
</tr>
<tr>
<td>B (8)</td>
<td>C (8)</td>
</tr>
<tr>
<td>D (8)</td>
<td>E (8)</td>
</tr>
<tr>
<td>H (8)</td>
<td>L (8)</td>
</tr>
<tr>
<td>Stack Pointer (SP) (16)</td>
<td></td>
</tr>
<tr>
<td>Program Counter (PC) (16)</td>
<td></td>
</tr>
</tbody>
</table>

**Fig.2.2**

**Registers**
The 8085 has six general-purpose registers to store 8-bit data; these are identified as B, C, D, E, H, and L as shown in the figure. They can be combined as register pairs - BC, DE, and HL - to perform some 16-bit operations. The programmer can use these registers to store or copy data into the registers by using data copy instructions.
**Accumulator**

The accumulator is an 8-bit register that is a part of arithmetic/logic unit (ALU). This register is used to store 8-bit data and to perform arithmetic and logical operations. The result of an operation is stored in the accumulator. The accumulator is also identified as register A.

**Flags**

The ALU includes five flip-flops, which are set or reset after an operation according to data conditions of the result in the accumulator and other registers. They are called Zero (Z), Carry (CY), Sign (S), Parity (P), and Auxiliary Carry (AC) flags; their bit positions in the flag register are shown in the Figure below. The most commonly used flags are Zero, Carry, and Sign. The microprocessor uses these flags to test data conditions.

```
<table>
<thead>
<tr>
<th>D7</th>
<th>D6</th>
<th>D5</th>
<th>D4</th>
<th>D3</th>
<th>D2</th>
<th>D1</th>
<th>D0</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td>Z</td>
<td>AC</td>
<td></td>
<td>P</td>
<td></td>
<td></td>
<td>CY</td>
</tr>
</tbody>
</table>
```

For example, after an addition of two numbers, if the sum in the accumulator is larger than eight bits, the flip-flop uses to indicate a carry -- called the Carry flag (CY) -- is set to one. When an arithmetic operation results in zero, the flip-flop called the Zero(Z) flag is set to one. The first Figure shows an 8-bit register, called the flag register, adjacent to the accumulator. However, it is not used as a register; five bit positions out of eight are used to store the outputs of the five flip-flops. The flags are stored in the 8-bit register so that the programmer can examine these flags (data conditions) by accessing the register through an instruction.

These flags have critical importance in the decision-making process of the micro-processor. The conditions (set or reset) of the flags are tested through the software instructions. For example, the instruction JC (Jump on Carry) is implemented to change the sequence of a program when CY flag is set. The thorough understanding of flag is essential in writing assembly language
Program Counter (PC)
This 16-bit register deals with sequencing the execution of instructions. This register is a memory pointer. Memory locations have 16-bit addresses, and that is why this is a 16-bit register.

The microprocessor uses this register to sequence the execution of the instructions. The function of the program counter is to point to the memory address from which the next byte is to be fetched. When a byte (machine code) is being fetched, the program counter is incremented by one to point to the next memory location.

Stack Pointer (SP)
The stack pointer is also a 16-bit register used as a memory pointer. It points to a memory location in R/W memory, called the stack. The beginning of the stack is defined by loading 16-bit address in the stack pointer.

This programming model will be used in subsequent tutorials to examine how these registers are affected after the execution of an instruction.

2.6 Summary
The 8085 is an 8-bit general purpose microprocessor that can address 64K Byte of memory. It has 40 pins and uses +5V for power. It can run at a maximum frequency of 3 MHz. Program, data and stack memories occupy the same memory space. The total addressable memory size is 64 KB. Program memory - program can be located anywhere in memory. Jump, branch and call instructions use 16-bit addresses, i.e. they can be used to jump/branch anywhere within 64 KB. All jump/branch instructions use absolute addressing.

2.7 Keywords
- NMOS
- RAM
- LIFO
- ALU
- Accumulator
- Flags
- PC
- SP

### 2.8 Exercise

1) Define stack and stack pointer.
2) Explain the Architecture of 8085.
3) Explain the 8085 Programming Model.

## Unit 3

**Block and Pin Diagram of 8085 and functions**

### Structure

3.1 Introduction
3.2 Objectives
3.3 Microprocessor pin description
3.4 Functional Description
3.6 Summary
3.7 Keywords
3.8 Exercise
3.1 Introduction
The Intel 8085A is a new generation, complete 8 bit parallel central processing unit (CPU). The 8085A uses a multiplexed data bus. The address is split between the 8bit address bus and the 8bit data bus. Figures are at the end of the document.

3.2 Objectives
At the end of this chapter you will be able to
- Explain Microprocessor pin description
- Know the Functional Description

3.3 Microprocessor pin description
Properties
1) Single + 5V Supply
2) 4 Vectored Interrupts (One is Non Maskable)
3) Serial In/Serial Out Port
4) Decimal, Binary, and Double Precision Arithmetic
5) Direct Addressing Capability to 64K bytes of memory

The Intel 8085A is a new generation, complete 8 bit parallel central processing unit (CPU). The 8085A uses a multiplexed data bus. The address is split between the 8bit address bus and the 8bit data bus.
The following describes the function of each pin:

A6 - A1s (Output 3 State)
Address Bus: The most significant 8 bits of the memory address or the 8 bits of the I/O address, 3 stated during Hold and Halt modes.

AD0 - 7 (Input/output 3state)
Multiplexed Address/Data Bus; Lower 8 bits of the memory address (or I/O addresses) appear on
the bus during the first clock cycle of a machine state. It then becomes the data bus during the second and third clock cycles. 3 stated during Hold and Halt modes.

**ALE (Output)**
Address Latch Enable: It occurs during the first clock cycle of a machine state and enables the address to get latched into the on chip latch of peripherals. The falling edge of ALE is set to guarantee setup and hold times for the address information. ALE can also be used to strobe the status information. ALE is never 3stated.

<table>
<thead>
<tr>
<th>S1</th>
<th>SO</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

**SO, S1 (Output)**
Data Bus Status. Encoded status of the bus cycle: S1 can be used as an advanced R/W status.

**RD (Output 3state)**
READ; indicates the selected memory or I/O device is to be read and that the Data Bus is available for the data transfer.

**WR (Output 3state)**
WRITE; indicates the data on the Data Bus is to be written into the selected memory or I/O location. Data is set up at the trailing edge of WR. 3stated during Hold and Halt modes.

**READY (Input)**
If Ready is high during a read or write cycle, it indicates that the memory or peripheral is ready to send or receive data. If Ready is low, the CPU will wait for Ready to go high before completing the read or write cycle.
**HOLD (Input)**

HOLD; indicates that another Master is requesting the use of the Address and Data Buses. The CPU, upon receiving the Hold request, will relinquish the use of buses as soon as the completion of the current machine cycle. Internal processing can continue. The processor can regain the buses only after the Hold is removed. When the Hold is acknowledged, the Address, Data, RD, WR, and IO/M lines are 3stated.

**HLDA (Output)**

HOLD ACKNOWLEDGE; indicates that the CPU has received the Hold request and that it will relinquish the buses in the next clock cycle. HLDA goes low after the Hold request is removed. The CPU takes the buses one half clock cycle after HLDA goes low.

**INTR (Input)**

INTERRUPT REQUEST; is used as a general purpose interrupt. It is sampled only during the next to the last clock cycle of the instruction. If it is active, the Program Counter (PC) will be inhibited from incrementing and an INTA will be issued. During this cycle a RESTART or CALL instruction can be inserted to jump to the interrupt service routine. The INTR is enabled and disabled by software. It is disabled by Reset and immediately after an interrupt is accepted.

**INTA (Output)**

INTERRUPT ACKNOWLEDGE; is used instead of (and has the same timing as) RD during the Instruction cycle after an INTR is accepted. It can be used to activate the 8259 Interrupt chip or some other interrupt port.

**RST 5.5**

**RST 6.5 - (Inputs)**

**RST 7.5**

RESTART INTERRUPTS; These three inputs have the same timing as INTR except they cause an internal RESTART to be automatically inserted.

**RST 7.5 Highest Priority**

**RST 6.5**
**RST 5.5 Lowest Priority**
The priority of these interrupts is ordered as shown above. These interrupts have a higher priority than the INTR.

**TRAP (Input)**
Trap interrupt is a non maskable restart interrupt. It is recognized at the same time as INTR. It is unaffected by any mask or Interrupt Enable. It has the highest priority of any interrupt.

**RESET IN (Input)**
Reset sets the Program Counter to zero and resets the Interrupt Enable and HLDA flip flops. None of the other flags or registers (except the instruction register) are affected The CPU is held in the reset condition as long as Reset is applied.

**RESET OUT (Output)**
Indicates CPU is being reset. It can be used as a system RESET. The signal is synchronized to the processor clock.

**X1, X2 (Input)**
Crystal or R/C network connections to set the internal clock generator X1 can also be an external clock input instead of a crystal. The input frequency is divided by 2 to give the internal operating frequency.

**CLK (Output)**
Clock Output for use as a system clock when a crystal or R/ C network is used as an input to the CPU. The period of CLK is twice the X1, X2 input period.

**IO/M (Output)**
IO/M indicates whether the Read/Write is to memory or I/O Tristated during Hold and Halt modes.
**SID (Input)**
Serial input data line: The data on this line is loaded into accumulator bit 7 whenever a RIM instruction is executed.

**SOD (output)**
Serial output data line. The output SOD is set or reset as specified by the SIM instruction.

**Vcc**
+5 volt supply.

**Vss**
Ground Reference.

### 3.4 Functional Description

The 8085A is a complete 8 bit parallel central processor. It requires a single +5 volt supply. Its basic clock speed is 3 MHz thus improving on the present 8080's performance with higher system speed. Also it is designed to fit into a minimum system of three IC's: The CPU, a RAM/IO, and a ROM or PROM/IO chip.

The 8085A uses a multiplexed Data Bus. The address is split between the higher 8 bit Address Bus and the lower 8 bit Address/Data Bus. During the first cycle the address is sent out. The lower 8 bits are latched into the peripherals by the Address Latch Enable (ALE). During the rest of the machine cycle the Data Bus is used for memory or I/O data.

The 8085A provides RD, WR, and IO/Memory signals for bus control. An Interrupt Acknowledge signal (INTA) is also provided. Hold, Ready, and all Interrupts are synchronized. The 8085A also provides serial input data (SID) and serial output data (SOD) lines for simple serial interface. In addition to these features, the 8085A has three maskable, restart interrupts and one non-maskable trap interrupt. The 8085A provides RD, WR and IO/M signals for Bus control.
Status Information

Status information is directly available from the 8085A. ALE serves as a status strobe. The status is partially encoded, and provides the user with advanced timing of the type of bus transfer being done. IO/M cycle status signal is provided directly also. Decoded S0, S1 Carry the following status information: HALT, WRITE, READ, FETCH.
S1 can be interpreted as R/W in all bus transfers. In the 8085A the 8 LSB of address are multiplexed with the data instead of status. The ALE line is used as a strobe to enter the lower half of the address into the memory or peripheral address latch. This also frees extra pins for expanded interrupt capability.

**Interrupt and Serial I/O**

The 8085A has 5 interrupt inputs: INTR, RST5.5, RST6.5, RST 7.5, and TRAP. INTR is identical in function to the 8080 INT. Each of the three RESTART inputs, 5.5, 6.5, 7.5, has a programmable mask. TRAP is also a RESTART interrupt except it is non maskable.

The three RESTART interrupts cause the internal execution of RST (saving the program counter in the stack and branching to the RESTART address) if the interrupts are enabled and if the interrupt mask is not set. The non-maskable TRAP causes the internal execution of a RST independent of the state of the interrupt enable or masks.

The interrupts are arranged in a fixed priority that determines which interrupt is to be recognized if more than one is pending as follows: TRAP highest priority, RST 7.5, RST 6.5, RST 5.5, INTR lowest priority This priority scheme does not take into account the priority of a routine that was started by a higher priority interrupt. RST 5.5 can interrupt a RST 7.5 routine if the interrupts were re-enabled before the end of the RST 7.5 routine. The TRAP interrupt is useful for catastrophic errors such as power failure or bus error. The TRAP input is recognized just as any other interrupt but has the highest priority. It is not affected by any flag or mask. The TRAP input is both edge and level sensitive.

**Basic System Timing**

The 8085A has a multiplexed Data Bus. ALE is used as a strobe to sample the lower 8 bits of address on the Data Bus. Figure 2 shows an instruction fetch, memory read and I/O write cycle (OUT). Note that during the I/O write and read cycle that the I/O port address is copied on both the upper and lower half of the address. As in the 8080, the READY line is used to extend the
read and write pulse lengths so that the 8085A can be used with slow memory. Hold causes the CPU to relinquish the bus when it is through with it by floating the Address and Data Buses.

**System Interface**

8085A family includes memory components, which are directly compatible to the 8085A CPU. For example, a system consisting of the three chips, 8085A, 8156, and 8355 will have the following features:

1) 2K Bytes ROM
2) 256 Bytes RAM
3) 1 Timer/Counter
4) 4 8bit I/O Ports
5) 1 6bit I/O Port
6) 4 Interrupt Levels
7) Serial In/Serial Out Ports

In addition to standard I/O, the memory mapped I/O offers an efficient I/O addressing technique. With this technique, an area of memory address space is assigned for I/O address, thereby, using the memory address for I/O manipulation. The 8085A CPU can also interface with the standard memory that does not have the multiplexed address/data bus.

### 3.6 Summary

Status information is directly available from the 8085A. ALE serves as a status strobe. The 8085A is a complete 8 bit parallel central processor. It requires a single +5 volt supply. Its basic clock speed is 3 MHz thus improving on the present 8080's performance with higher system speed. Also it is designed to fit into a minimum system of three IC's: The CPU, a RAM/ IO, and a ROM or PROM/IO chip.

### 3.7 Keywords

- ALE
- CPU
• RAM
• ROM
• PROM

3.8 Exercise

1) Explain 8085 microprocessor pin description with diagram.
2) Discuss on 8085 Functional Description with a block diagram.

Unit 4

BUS Details

Structure

4.1 Introduction
4.2 Objectives
4.3 BUS Details
4.4 Address Bus
4.5 Data Bus
4.6 Control Bus
4.7 Power supply and clock frequency
4.8 Basic components of microprocessor
4.9 Summary
4.10 Keywords
4.11 Exercise

4.1 Introduction

Address bus - 16 line bus accessing 216 memory locations (64 KB) of memory. Data bus - 8 line
bus accessing one (8-bit) byte of data in one operation. Data bus width is the traditional measure of processor bit designations, as opposed to address bus width, resulting in the 8-bit microprocessor designation. Control buses - Carries the essential signals for various operations. Typical system uses a number of busses, collection of wires, which transmit binary numbers, one bit per wire. A typical microprocessor communicates with memory and other devices (input and output) using three busses: Address Bus, Data Bus and Control Bus.

### 4.2 Objectives

At the end of this chapter you will be able to:

- Explain Address Bus
- Define Data Bus
- Know Control Bus
- Know the Basic components of microprocessor

### 4.3 BUS Details

MPU is defined as a device or a group of devices that can communicate with peripherals provide timing signals, direct data flow and perform computing tasks as specified by the instructions in memory. This unit will have the necessary address bus, the data bus, the control signals, and would have require only a power of supply and a crystal to be functional.

The 8085A is an 8-bit general purpose microprocessor capable of addressing 64KB of memory. The device has 40 pins, requires +5V single power supply. All the signals of 8085 can be classified into 6 groups.

They are:

- Address Bus
- Data Bus
- Control and Status Signals
- Power supply and frequency Signals
- Serial I/O Ports
The lower order address bus 8085 is multiplexed with the data bus. The bus need to be demultiplexed. Appropriate control signal need to be generated to interface memory and I/O with the 8085.

### 4.4 Address Bus

One wire for each bit, therefore 16 bits = 16 wires. Binary number carried alerts memory to open the designated box. Data (binary) can then be put in or taken out. The Address Bus consists of 16 wires, therefore 16 bits. Its width is 16 bits. A 16 bit binary number allows 216 different numbers, or 32000 different numbers, i.e. 0000000000000000 up to 1111111111111111. Because memory consists of boxes, each with a unique address, the size of the address bus determines the size of memory, which can be used. To communicate with memory the microprocessor sends an address on the address bus, e.g. 0000000000000111 (3 in decimal), to the memory. The memory the selects box number 3 for reading or writing data. Address bus is unidirectional, i.e. numbers only sent from microprocessor to memory, not the other way.

### 4.5 Data Bus

It carries data, in binary form, between μP and other external units, such as memory. Typical size is 8 or 16 bits. Size determined by size of boxes in memory and μP size helps determine performance of μP. The Data Bus typically consists of 8 wires. Therefore, 28 combinations of binary digits. Data bus is used to transmit data, i.e. information, results of arithmetic, etc., between memory and the microprocessor. Bus is bi-directional. Size of the data bus determines what arithmetic can be done. If only 8 bits wide then largest number is 11111111 (255 in decimal). Therefore, larger numbers have to be broken down into chunks of 255. This slows microprocessor. Data Bus also carries instructions from memory to the microprocessor. Size of the bus therefore limits the number of possible instructions to 256, each specified by a separate number.

### 4.6 Control Bus

Control Bus are various lines which have specific functions for coordinating and controlling uP operations. Eg: Read/NotWrite line, single binary digit. Control whether memory is being written to (data stored in mem) or read from (data taken out of mem) 1 = Read, 0 = Write. May also
include clock line(s) for timing/synchronising, interrupts, reset etc. Typically μP has 10 control lines. It cannot function correctly without these vital control signals.

The Control Bus carries control signals partly unidirectional, partly bi-directional. Control signals are things like read or write. This tells memory that we are reading from a location, specified on the address bus, or writing to a location specified. Various other signals to control and coordinate the operation of the system. Modern day microprocessors, like 80386, 80486 have much larger buses. Typically 16 or 32 bit buses, which allow larger number of instructions, more memory location, and faster arithmetic.

In the microprocessor the three buses are external to the chip (except for the internal data bus). In case of external buses, the chip connects to the buses via buffers, which are simply an electronic connection between external bus and the internal data bus.

4.7 Power supply and clock frequency

- Vcc: +5V power supply.
- Vss: Ground reference.
- X1, X2: A crystal is connected at these two pins. The frequency is internally divided by two; therefore, to operate a system at 3MHz, the crystal should have a frequency of 6MHz.
- CLK (Out): Clock output – this signal can be used as the system clock for other devices.

4.8 Basic components of microprocessor

Basic units of a microprocessor

The basic units or blocks of a microprocessor are ALU, an array of registers and control unit.

Software and Hardware

The Software is a set of instructions or commands needed for performing a specific task by a programmable device or a computing machine.

The Hardware refers to the components or devices used to form computing machine in which the software can be run and tested. Without software the Hardware is an idle machine.

Assembly language
The language in which the mnemonics (short hand form of instructions) are used to write a program is called assembly language. The manufacturers of microprocessor give the mnemonics.

**Machine language and assembly language programs**
The software developed using 1's and 0's are called machine language, programs. The software developed using mnemonics are called assembly language programs.

**Drawback in machine language and assembly language, programs**
The machine language and assembly language programs are machine dependent. The programs developed using these languages for a particular machine cannot be directly run on another machine.

**Bit, byte, bus and word.**
A digit of the binary number or code is called bit. Also, the bit is the fundamental storage unit of computer memory. The 8-bit (8-digit) binary number or code is called byte and 16-bit binary number or code is called word. (Some microprocessor manufactures refer the basic data size operated by the processor as word).

Bus is a group of conducting lines that carries data, address and control signals.

**Why data bus is bi-directional?**
The microprocessor has to fetch (read) the data from memory or input device for processing and after processing, it has to store (write) the data to memory or output device. Hence the data bus is bi-directional.

**Why address bus is unidirectional?**
The address is an identification number used by the microprocessor to identify or access a memory location or I / O device. It is an output signal from the processor. Hence the address bus is unidirectional.
4.9 Summary

A Bus is a set of data/control lines which are basically just a group of wires. e.g if 8085 wants to send 0xFF (255) to an Output device, it would make all the lines of the data bus high, bcoz 0xFF is 11111111 in binary, that is to say all the wires of the bus are 1. (1 means at +5V). The address bus is unidirectional because it is the CPU that tells the external hardware what address to use, not the other way around. The width of the address bus on the 8085 was decided based on a compromise between functionality, cost, and complexity. Intel decide to use 16 bits, as that was in keeping with common design at that time, and also because the 8085 was actually an enhanced version of the 8080.

4.10 Keywords

- KB
- Address Bus
- Data Bus
- Control Bus
- Bit
- Byte

4.11 Exercise

1) Explain Address Bus.
2) Define Data Bus, Control Bus.
3) Why address bus is unidirectional?
4) Why data bus is bi-directional?

Unit 1

Instruction formats & Addressing Modes
1.1 Introduction

In its basic operation the central processing unit alternates between fetching instructions and executing instructions. The instructions cause the processor to manipulate the contents of specific programming model register- and of memory and I/O locations. Certain types of instructions. Namely load/store, arithmetic logic, and input/output, must identify specific registers and/or locations for the processor in order that it may locate operands or store results property. Test branch instructions must also identify the branch target locations in the program. This chapter examines the various classes of instructions which processors must be able to execute.

1.2 Objectives

At the end of this chapter you will be able to:

- List the Types of Instructions
1.3 Types of Instructions

The computer can do certain basic types of operations: load/store, arithmetic logic, test/branch, and input/output. Although the instructions which direct these operations in the central processing unit vary widely from processor to processor, certain common characteristics are found in all processor instruction sets.

1.3.1 Load/Store Instructions

Load/store operations are those which move data between a register in the processor and a memory location (or another register). They are often collectively referred to as data movement instructions. The two terms load and store are often confused and should be carefully distinguished from each other. Figure 1.1 illustrates the difference.
Fig: 1.1: Load and store operations: (a) load, (b) store.

One of the universal characteristics of instruction execution is that the source does not get changed, only the destination. Thus, in a load operation the processor copies data into a register, and in a store operation the processor copies data from a register into a memory location.

Examples of load/store instructions in the MC6809 are LDA, STA, LDB, STB, LDD, STD, and so forth. Examples of load/store instructions in the MC68000 are MOVE (for either direction of data movement), MOVEM (for moving the content of several registers into or out of memory with a single instruction), and EXG (for exchanging register contents). These instructions and others like them must include information identifying the source and destination.

1.3.2 Arithmetic/Logic Instructions

Arithmetic/logic operations provide the primary data-processing capabilities of the computer. The minimum arithmetic operations required are those of addition and subtraction. All processor, implement the basic logic operations of AND, OR, and exclusive-OR on a bit-by-bit basis between two operands.

With operations requiring two operands, smaller microprocessors usually require that one of the operands must be in a destination register initially, while the other may be in another register or in memory. Their instructions may identify at most a single operand in memory, as shown in Figure 1.2a. They support such operations as adding or ANDing to processor registers as illustrated but not to memory locations.
Some examples of two-operand arithmetic/logic instructions in the MC6809 are ADDB, SUBA, SUBD, ANDA, and EORA (exclusive-OR). Each of these requires one operand to be in an Accumulator. Some two-operand arithmetic/logic instructions in the MC68000 instruction set are ADD, AND, OR, SUB, and EOR. All of the two-operand instructions in the MC68000 must specify the source and the destination. With a few exceptions, these may both be registers or memory locations or one of each.

Among the single-operand arithmetic/logic operations are increment and decrement operations, the 1's and 2's complement, and various shift and rotate operations. These operate directly upon the specified operand to modify it. Thus, the operand location is both the source and the
destination for the operation.

Examples of one-operand instructions in the MC6809 include COMA (complement A), ASLB (arithmetic shift left B), INC (increment a memory location whose address is identified in the instruction), and NEGA (2’s complement or negate A). The MC68000 includes the instructions CLR (clear), NEG (change sign of), NOT (complement), and EXT (extend the sign bit to double the number of bits in an operand).

1.3.3 Test/Branch Instructions

Test/branch operations provide the decision-making capabilities so important in computer programs. The test must often be performed on the result of some prior arithmetic/logic operation. It determines whether the result was zero or nonzero, positive or negative, or some other binary choice involving bits in the condition code register. During the execution of a test/branch instruction the processor examines the appropriate condition code bits to determine the result of the test. Depending upon the result, the processor may or may not branch to a remote location for the next instruction.

Examples of test/branch instructions are BCC (branch if carry is cleared), BEQ (branch if equal to 0), BMI (branch if minus), and BRA (branch always).

1.4 Addressing Modes

Most of the instructions must refer to the address or content of a specific memory location. These so-called memory reference instructions must somehow identify the address of the location as a part of the instruction encoding. The manner in which this target address or effective address is identified within the instruction is called the addressing mode. This section describes the more common addressing modes used in microprocessors.

1.4.1 Direct Addressing

When the instruction explicitly states the location of an operand or a destination (either in memory or in a processor register), the addressing mode is known as direct addressing. The effective address itself is included in the subsequent words of the instruction (post-words). Two sub-classifications within the direct addressing mode are often recognized. When the
location is in memory the mode may be referred to as absolute addressing. When the location is a processor register it may be referred to as register direct addressing. Figure 1.3 illustrates the two modes. In part a the instruction specifies the address of the operand; in part b the instruction specifies the register containing the operand.

\[ \text{(LDA 100BH, ADDA 100CH, STA 100DH)} \]

Direct addressing is used when the memory address or the selected register is to be fixed in the program.

### 1.4.2 Immediate Addressing

In many cases an instruction requires a constant quantity, a bit pattern which will never change no matter when or how often the instruction is executed. This mode of including a bit pattern as a part of an instruction is called the immediate addressing mode. The op-word for the instruction includes a group of bits which identifies this mode of addressing, and the post-words include the bit pattern itself.

Since the instruction is located in program memory the constant itself is also in program memory. The immediate addressing mode the instruction does not state explicitly the location of the operand; rather, it explicitly states the operand itself. The example in Figure 1.4 illustrates this mode. Note that the operand becomes an integral part of the instruction.
Immediate addressing is used when a particular constant value is to be fixed within the program itself. The value is found in memory "immediately" after the instruction code word and may never change at any time.

1.4.3 Indirect Addressing

In the indirect addressing mode the instruction tells the processor neither the address of the operand nor the operand itself. Instead, it tells the processor where to go to find the address of the operand. The instruction may explicitly state either the address of a location in memory or the name of a processor register, but the binary number which is found there is not the operand. Instead, it is the effective address, the address of a location in memory to which the processor must go to find the operand. The result is that the processor must take one extra step in order to locate the operand.

The op-word for the instruction includes a group of bits which identifies this mode of addressing, and the (indirect) address is specified in one or more additional post-words. If the instruction names a processor register as the source of the effective address, then the register identification number may fit into the op-word itself.

Indirect addressing is used when a program must operate upon different data values under different circumstances. Figure 1.6 compares and contrasts the first three addressing modes. Note that in the immediate mode the instruction includes the operand, in the direct mode it includes the address of the operand,
1.4.4 Multi-Component Addressing Modes

Each of the following related addressing modes requires that the processor assemble two or more components together during the execution of the program in order to create the effective address. In each case the effective address itself is that of a memory location. However, at least one of the components is found in a processor register.

Instructions which use indexed addressing specify two registers, often by coding within the op-word itself and known as "indexed addressing". During program execution the processor temporarily adds the contents of these registers to generate the effective address. One of the registers is an address register and it is said to hold the base address. The other is commonly a
data register – the displacement or index register.

Based addressing is a similar mode wherein the instruction specifies an address register and a fixed constant (an offset or displacement). The register designation often fits within the op-word and the offset usually requires post-words. In this mode the content of the register is the base and the constant is the displacement. During execution the processor adds the constant and the value in the register to generate the effective address. This addressing mode is also known as "relative based indexed" mode.

The relative addressing mode permits the writing of "position-independent code," programs which will be properly executed by the processor regardless of where they are located in memory.

The entire program (together with any necessary data) may be picked up from one region of memory and moved to another with no adverse effect. In order to be location-independent a program may not refer to any specific location by address. All references to memory must be through the use of relative addressing. Examples will be shown in a later chapter.
1.4.5 Implied or Inherent Addressing

Certain instructions allow no choice of register or location but always cause the processor to refer to the same registers. One example is the multiply instruction in the MC6809 which always assumes that the operands are in accumulators A and B. Examples in the MC6809 are ABX (add B to X), CLRA (clear A), and TSTA (test A).

1.5 The MC6809 Addressing Modes

This section describes in detail the addressing modes used in the Motorola MC6809 microprocessor, their formats within program memory, and how they are specified in mnemonic form. The terminology is that used in the manufacturer's literature.

The MC6809 includes five addressing modes: extended, immediate, direct, relative, and
indexed. In addition, the indexed mode has a large variety of options. Because of the large number of instructions and addressing modes, the instructions vary in length from one to five bytes. The basic addressing modes are illustrated in Fig. 1.10

**Fig. 1.10 MC6809 addressing modes: (a) extended, (b) immediate, (c) direct, (d) relative (branches only), (e) indexed.**

**Auto Increment/Decrement Options in the MC6809**

While repeating a program loop the processor must often change a pointer to an adjacent address in memory on each pass through the loop. It may be necessary to increment or decrement the value in the pointer, depending upon the direction in which the array is being scanned.

Two examples of auto increment/decrement options are shown in Figure 5.18. Part a is the
instruction LDA ,X+, load A indexed with X post-inc by one. Figure 5.18b is the instruction STY ,U. When it is executed the processor will first decrement the index register U by 2. Following that, it will store the upper half of the two-byte register Y into the new location indicated by register U and the lower half into the next subsequent location.

Fig. 1.18 MC6809 indexed addressing, auto increment/decrement options: (a) post-increment, (b) pre-decrement.

Pre-decrementing by 2 opens up two new slots in memory where the double-byte value in Y can be stored in accordance with the instruction. As a consequence of the increment/decrement option, not only does the instruction accomplish its primary task of manipulating the targeted operands, it also modifies the content of the selected index register.

1.6 Summary

Each instruction of a computer specifies an operation on certain data. They are various ways of specifying address of the data to be operated on. These different ways of specifying data are called the addressing modes. The instructions cause the processor to manipulate the contents of specific programming model register- and of memory and I/O locations.
1.7 Keywords

- Instructions
- AND Logical operation
- BEQ
- BCC
- BMI

1.8 Exercise

1) List the Types of Instructions.
2) Explain Addressing Modes.
3) Explain The MC6809 Addressing Modes.

Unit 2

Instruction Set

Structure

2.1 Introduction
2.2 Objectives
2.3 Instruction Set Classification
2.4 Instruction Naming Conventions:
2.5 Summary
2.6 Keywords
2.7 Exercise
2.1 Introduction

An Instruction is a command given to the computer to perform a specified operation on given data. The instruction set of a microprocessor is the collection of the instructions that the microprocessor is designed to execute. The instructions described here are of Intel 8085. These instructions are of Intel Corporation. They cannot be used by other microprocessor manufactures. The programmer can write a program in assembly language using these instructions.

2.2 Objectives

At the end of this chapter you will be able to:

- Explain Instruction Set Classification
- Know about Instruction Naming Conventions

2.3 Instruction Set Classification

An instruction is a binary pattern designed inside a microprocessor to perform a specific function. The entire group of instructions, called the instruction set, determines what functions the microprocessor can perform. These instructions can be classified into the following five functional categories: data transfer (copy) operations, arithmetic operations, logical operations, branching operations, and machine-control operations.

2.3.1 Data Transfer (Copy) Operations

This group of instructions copy data from a location called a source to another location called a destination, without modifying the contents of the source. In technical manuals, the term data transfer is used for this copying function. However, the term transfer is misleading; it creates the impression that the contents of the source are destroyed when, in fact, the contents are retained without any modification. The various types of data transfer (copy) are listed below together with examples of each type:

<table>
<thead>
<tr>
<th>Types</th>
<th>Examples</th>
</tr>
</thead>
</table>

1. Between Registers.
   1. Copy the contents of the register B into register D.

2. Specific data byte to a register or a memory location.
   2. Load register B with the data byte 32H.

3. Between a memory location and a register.
   3. From a memory location 2000H to register B.

4. Between an I/O device and the accumulator.
   4. From an input keyboard to the accumulator.

### 2.3.2 Arithmetic Operations

These instructions perform arithmetic operations such as addition, subtraction, increment, and decrement.

**Addition** - Any 8-bit number, or the contents of a register or the contents of a memory location can be added to the contents of the accumulator and the sum is stored in the accumulator. No two other 8-bit registers can be added directly (e.g., the contents of register B cannot be added directly to the contents of the register C). The instruction DAD is an exception; it adds 16-bit data directly in register pairs.

**Subtraction** - Any 8-bit number, or the contents of a register, or the contents of a memory location can be subtracted from the contents of the accumulator and the results stored in the accumulator. The subtraction is performed in 2’s compliment, and the results if negative, are expressed in 2’s complement. No two other registers can be subtracted directly.

**Increment/Decrement** - The 8-bit contents of a register or a memory location can be incremented or decrement by 1. Similarly, the 16-bit contents of a register pair (such as BC) can be incremented or decrement by 1. These increment and decrement operations differ
from addition and subtraction in an important way; i.e., they can be performed in any one of
the registers or in a memory location.

2.3.3 Logical Operations
These instructions perform various logical operations with the contents of the accumulator.

**AND, OR Exclusive-OR** - Any 8-bit number, or the contents of a register, or of a memory
location can be logically ANDed, Ored, or Exclusive-ORed with the contents of the accumulator.
The results are stored in the accumulator.

**Rotate** - Each bit in the accumulator can be shifted either left or right to the next position.

**Compare** - Any 8-bit number, or the contents of a register, or a memory location can be
compared for equality, greater than, or less than, with the contents of the accumulator.

**Complement** - The contents of the accumulator can be complemented. All 0s are replaced by 1s
and all 1s are replaced by 0s.

2.3.4 Branching Operations
This group of instructions alters the sequence of program execution either conditionally or
unconditionally.

**Jump** - Conditional jumps are an important aspect of the decision-making process in the
programming. These instructions test for a certain conditions (e.g., Zero or Carry flag) and alter
the program sequence when the condition is met. In addition, the instruction set includes an
instruction called *unconditional jump*.

**Call, Return, and Restart** - These instructions change the sequence of a program either by
calling a subroutine or returning from a subroutine. The conditional Call and Return instructions
also can test condition flags.

### 2.3.5 Machine Control Operations

These instructions control machine functions such as Halt, Interrupt, or do nothing. The microprocessor operations related to data manipulation can be summarized in four functions:

1. copying data
2. performing arithmetic operations
3. performing logical operations
4. testing for a given condition and alerting the program sequence

Some important aspects of the instruction set are noted below:

1. In data transfer, the contents of the source are not destroyed; only the contents of the destination are changed. The data copy instructions do not affect the flags.
2. Arithmetic and Logical operations are performed with the contents of the accumulator, and the results are stored in the accumulator (with some expectations). The flags are affected according to the results.
3. Any register including the memory can be used for increment and decrement.
4. A program sequence can be changed either conditionally or by testing for a given data condition.

### 2.4 Instruction Naming Conventions:

The mnemonics assigned to the instructions are designed to indicate the function of the instruction. The instructions fall into the following functional categories:

**Data Transfer Croup:**
The data transfer instructions move data between registers or between memory and registers.

MOV       Move
MVI       Move Immediate
<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LDA</td>
<td>Load Accumulator Directly from Memory</td>
</tr>
<tr>
<td>STA</td>
<td>Store Accumulator Directly in Memory</td>
</tr>
<tr>
<td>LHLD</td>
<td>Load H &amp; L Registers Directly from Memory</td>
</tr>
<tr>
<td>SHLD</td>
<td>Store H &amp; L Registers Directly in Memory</td>
</tr>
</tbody>
</table>

An 'X' in the name of a data transfer instruction implies that it deals with a register pair (16-bits);

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LXI</td>
<td>Load Register Pair with Immediate data</td>
</tr>
<tr>
<td>LDAX</td>
<td>Load Accumulator from Address in Register Pair</td>
</tr>
<tr>
<td>STAX</td>
<td>Store Accumulator in Address in Register Pair</td>
</tr>
<tr>
<td>XCHG</td>
<td>Exchange H &amp; L with D &amp; E</td>
</tr>
<tr>
<td>XTHL</td>
<td>Exchange Top of Stack with H &amp; L</td>
</tr>
</tbody>
</table>

**Arithmetic Group:**
The arithmetic instructions add, subtract, increment, or decrement data in registers or memory.

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADD</td>
<td>Add to Accumulator</td>
</tr>
<tr>
<td>ADI</td>
<td>Add Immediate Data to Accumulator</td>
</tr>
<tr>
<td>ADC</td>
<td>Add to Accumulator Using Carry Flag</td>
</tr>
<tr>
<td>ACI</td>
<td>Add Immediate data to Accumulator Using Carry</td>
</tr>
<tr>
<td>SUB</td>
<td>Subtract from Accumulator</td>
</tr>
<tr>
<td>SUI</td>
<td>Subtract Immediate Data from Accumulator</td>
</tr>
<tr>
<td>SBB</td>
<td>Subtract from Accumulator Using Borrow (Carry) Flag</td>
</tr>
<tr>
<td>SBI</td>
<td>Subtract Immediate from Accumulator Using Borrow (Carry) Flag</td>
</tr>
<tr>
<td>INR</td>
<td>Increment Specified Byte by One</td>
</tr>
<tr>
<td>DCR</td>
<td>Decrement Specified Byte by One</td>
</tr>
<tr>
<td>INX</td>
<td>Increment Register Pair by One</td>
</tr>
<tr>
<td>DCX</td>
<td>Decrement Register Pair by One</td>
</tr>
<tr>
<td>DAD</td>
<td>Double Register Add; Add Content of Register Pair to H &amp; L Register Pair</td>
</tr>
</tbody>
</table>
**Logical Group:**
This group performs logical (Boolean) operations on data in registers and memory and on condition flags.

The logical AND, OR, and Exclusive OR instructions enable you to set specific bits in the accumulator ON or OFF.

- **ANA** Logical AND with Accumulator
- **ANI** Logical AND with Accumulator Using Immediate Data
- **ORA** Logical OR with Accumulator
- **OR** Logical OR with Accumulator Using Immediate Data
- **XRA** Exclusive Logical OR with Accumulator
- **XRI** Exclusive OR Using Immediate Data

The Compare instructions compare the content of an 8-bit value with the contents of the accumulator;

- **CMP** Compare
- **CPI** Compare Using Immediate Data

The rotate instructions shift the contents of the accumulator one bit position to the left or right:

- **RLC** Rotate Accumulator Left
- **RRC** Rotate Accumulator Right
- **RAL** Rotate Left Through Carry
- **RAR** Rotate Right Through Carry

Complement and carry flag instructions:

- **CMA** Complement Accumulator
- **CMC** Complement Carry Flag
STC      Set Carry Flag

**Branch Group:**
The branching instructions alter normal sequential program flow, either unconditionally or conditionally. The unconditional branching instructions are as follows:

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>JMP</td>
<td>Jump</td>
</tr>
<tr>
<td>CALL</td>
<td>Call</td>
</tr>
<tr>
<td>RET</td>
<td>Return</td>
</tr>
</tbody>
</table>

Conditional branching instructions examine the status of one of four condition flags to determine whether the specified branch is to be executed. The conditions that may be specified are as follows:

<table>
<thead>
<tr>
<th>Condition</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NZ</td>
<td>Not Zero (Z = 0)</td>
</tr>
<tr>
<td>Z</td>
<td>Zero (Z = 1)</td>
</tr>
<tr>
<td>NC</td>
<td>No Carry (C = 0)</td>
</tr>
<tr>
<td>C</td>
<td>Carry (C = 1)</td>
</tr>
<tr>
<td>PO</td>
<td>Parity Odd (P = 0)</td>
</tr>
<tr>
<td>PE</td>
<td>Parity Even (P = 1)</td>
</tr>
<tr>
<td>P</td>
<td>Plus (S = 0)</td>
</tr>
<tr>
<td>M</td>
<td>Minus (S = 1)</td>
</tr>
</tbody>
</table>

Thus, the conditional branching instructions are specified as follows:

<table>
<thead>
<tr>
<th>Jumps</th>
<th>Calls</th>
<th>Returns</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>CC</td>
<td>RC</td>
</tr>
<tr>
<td>INC</td>
<td>CNC</td>
<td>RNC</td>
</tr>
<tr>
<td>JZ</td>
<td>CZ</td>
<td>RZ</td>
</tr>
<tr>
<td>JNZ</td>
<td>CNZ</td>
<td>RNZ</td>
</tr>
<tr>
<td>JP</td>
<td>CP</td>
<td>RP</td>
</tr>
</tbody>
</table>
Two other instructions can affect a branch by replacing the contents or the program counter:

- **PCHL**: Move H & L to Program Counter
- **RST**: Special Restart Instruction Used with Interrupts

**Stack I/O, and Machine Control Instructions:**

The following instructions affect the Stack and/or Stack Pointer:

- **PUSH**: Push Two bytes of Data onto the Stack
- **POP**: Pop Two Bytes of Data off the Stack
- **XTHL**: Exchange Top of Stack with H & L
- **SPHL**: Move content of H & L to Stack Pointer

**The I/O instructions are as follows:**

- **IN**: Initiate Input Operation
- **OUT**: Initiate Output Operation

**The Machine Control instructions are as follows:**

- **EI**: Enable Interrupt System
- **DI**: Disable Interrupt System
- **HLT**: Halt
- **NOP**: No Operation

### 2.5 Summary
The instruction set of a microprocessor is the collection of the instructions that the microprocessor is designed to execute. The instructions described here are of Intel 8085. These instructions are of Intel Corporation. They cannot be used by other microprocessor manufactures. The programmer can write a program in assembly language using these instructions. These instructions have been classified into the following groups:

1. Data Transfer Group
2. Arithmetic Group
3. Logical Group
4. Branch Control Group
5. I/O and Machine Control Group

Data Transfer Group: Instructions, which are used to transfer data from one register to another register, from memory to register or register to memory, come under this group. Examples are: MOV, MVI, LXI, LDA, STA etc. When an instruction of data transfer group is executed, data is transferred from the source to the destination without altering the contents of the source. For example, when MOV A, B is executed the content of the register B is copied into the register A, and the content of register B remains unaltered. Similarly, when LDA 2500 is executed the content of the memory location 2500 is loaded into the accumulator. But the content of the memory location 2500 remains unaltered.

Arithmetic Group: The instructions of this group perform arithmetic operations such as addition, subtraction; increment or decrement of the content of a register or memory. Examples are: ADD, SUB, INR, DAD etc.

Logical Group: The Instructions under this group perform logical operation such as AND, OR, compare, rotate etc. Examples are: ANA, XRA, ORA, CMP, and RAL etc.

Branch Control Group: This group includes the instructions for conditional and unconditional jump, subroutine call and return, and restart. Examples are: JMP, JC, JZ, CALL, CZ, RST etc.

I/O and Machine Control Group: This group includes the instructions for input/output ports, stack and machine control. Examples are: IN, OUT, PUSH, POP, and HLT etc.

2.6 Keywords
• Logical Operations
• Branching Operations
• Machine Control Operations
• Data Transfer Group
• Arithmetic Group

2.7 Exercise
  1) Explain Instruction Set Classification.
  2) Explain Arithmetic Group.
  3) Explain Logical Group.

Unit 3
Timing Diagrams and Status Signals

Structure
3.1 Introduction
3.2 Objectives
3.3 Processor Cycle
3.4 Timing Diagram of OPCODE FETCH
3.5 Summary
3.6 Keywords
3.7 Exercise
3.1 Introduction

Timing diagram is the display of initiation of read/write and transfer of data operations under the control of 3-status signals IO / M, S₁, and S₀. As the heartbeat is required for the survival of the human being, the CLK is required for the proper operation of different sections of the microprocessors. All actions in the microprocessor is controlled by either leading or trailing edge of the clock. If I ask a man to bring 6-bags of wheat, each weighing 100 kg, he may take 6-times to perform this task in going and bringing it. A stronger man might perform the same task in 3-times only. Thus, it depends on the strength of the man to finish the job quickly or slowly. Here, we can assume both weaker and strong men as machine. The weaker man has taken 6-machine cycle (6-times going and coming with one bag each time) to execute the job where as the stronger man has taken only 3-machine cycle for the same job. Similarly, a machine may execute one instruction in as many as 3-machine cycles while the other machine can take only one machine cycle to execute the same instruction. Thus, the machine that has taken only one machine cycle is efficient than the one taking 3-machine cycle. Each machine cycle is composed of many clock cycle. Since, the data and instructions, both are stored in the memory, the µP performs fetch operation to read the instruction or data and then execute the instruction. The µP in doing so may take several cycles to perform fetch and execute operation. The 3-status signals : IO / M, S₁, and S₀ are generated at the beginning of each machine cycle. The unique combination of these 3-status signals identify read or write operation and remain valid for the duration of the cycle. Table-3.1(a) shows details of the unique combination of these status signals to identify different machine cycles. Thus, time taken by any µP to execute one instruction is calculated in terms of the clock period.

The execution of instruction always requires read and writes operations to transfer data to or from the µP and memory or I/O devices. Each read/ write operation constitutes one machine cycle (MC₁) as indicated in Fig. 3.1 (a). Each machine cycle consists of many clock periods/ cycles, called T-states. The heartbeat of the microprocessor is the clock period. Each and every operation inside the microprocessor is under the control of the clock cycle. The clock signal
determines the time taken by the microprocessor to execute any instruction. The clock cycle shown in Fig. 3.1 (a) has two edges (leading and trailing or lagging). State is defined as the time interval between 2-trailing or leading edges of the clock. Machine cycle is the time required to transfer data to or from memory or I/O devices.

Table 3.1(a) Machine cycle status and control signals

<table>
<thead>
<tr>
<th>Machine cycle</th>
<th>Status</th>
<th>Controls</th>
</tr>
</thead>
<tbody>
<tr>
<td>Opcode Fetch (OF)</td>
<td>0 1 1</td>
<td>0 1 1</td>
</tr>
<tr>
<td>Memory Read</td>
<td>0 1 0</td>
<td>0 1 1</td>
</tr>
<tr>
<td>Memory Write</td>
<td>0 0 1</td>
<td>1 0 1</td>
</tr>
<tr>
<td>I/O Read (I/OR)</td>
<td>1 1 0</td>
<td>0 1 1</td>
</tr>
<tr>
<td>I/O Write (I/OW)</td>
<td>1 0 1</td>
<td>1 0 1</td>
</tr>
<tr>
<td>Acknowledge of INTR (INTA)</td>
<td>1 1 1</td>
<td>1 1 0</td>
</tr>
<tr>
<td>BUS Idle (BI) : DAD</td>
<td>0 1 0</td>
<td>1 1 1</td>
</tr>
<tr>
<td>ACK of RST, TRAP</td>
<td>1 1 1</td>
<td>1 1 1</td>
</tr>
<tr>
<td>HALT</td>
<td>Z 0 0</td>
<td>Z Z 1</td>
</tr>
<tr>
<td>HOLD</td>
<td>Z X X</td>
<td>Z Z 1</td>
</tr>
</tbody>
</table>

Unspecified, and High impedance state

Fig. 3.1 (a) Machine cycle showing clock periods
3.2 Objectives

At the end of this chapter you will be able to:

- Explain Processor Cycle
- Give Timing Diagram of OPCODE FETCH

3.3 Processor Cycle

The function of the microprocessor is divided into fetch and execute cycle of any instruction of a program. The program is nothing but number of instructions stored in the memory in sequence. In the normal process of operation, the microprocessor fetches (receives or reads) and executes one instruction at a time in the sequence until it executes the halt (HLT) instruction. Thus, an instruction cycle is defined as the time required to fetch and execute an instruction. For executing any program, basically 2-steps are followed sequentially with the help of clocks

- Fetch, and
- Execute.

The time taken by the µP in performing the fetch and execute operations are called fetch and execute cycle. Thus, sum of the fetch and execute cycle is called the instruction cycle as indicated in Fig. 3.2 (a).

Instruction Cycle (IC) = Fetch cycle (FC) + Execute Cycle (EC)
Fig. 3.2 (a) Processor cycle

Instruction cycle (I = FC + EC)

Fetch cycle (FC)

Execute cycle
These cycles have been illustrated in Figs. 3.2(a) and (b). Each read or writes operation constitutes a machine cycle. The instructions of 8085 require 1–5 machine cycles containing 3–6 states (clocks). The 1st machine cycle of any instruction is always an Op. Code fetch cycle in which the processor decides the nature of instruction. It is of at least 4-states. It may go up to 6-states.

It is well known that an instruction cycle consists of many machine cycles. Each machine cycle consists of many clock periods or cycles, called T-states. The 1st machine cycle (M₁) of every instruction cycle is the **opcode fetch** cycle. In the opcode fetch cycle, the processor comes to know the nature of the instruction to be executed. The processor during (M₁ cycle) puts the program counter contents on the address bus and reads the opcode of the instruction through **read process**. The T₁, T₂, and T₃ clock cycles are used for the basic memory read operation and the T₄ clock and beyond are used for its interpretation of the opcode. Based on these interpretations, the μP comes to know the type of additional information/data needed for the execution of the instruction and accordingly proceeds further for 1 or 2-machine cycle of memory read and writes.

The Op. code fetch cycle is of fixed duration (normally 4-states), whereas the instruction cycle is of variable duration depending on the length of the instruction. As an example, STA instruction, requires opcode fetch cycle, lower-order address fetch cycle and higher order fetch cycle and then the execute cycle. Thus opcode fetch cycle is of one machine cycle in this example. A particular microprocessor requires a definite time to performing a specific task. This time is called **machine cycle**. Thus, one machine cycle is required each time the μP access I/O port or memory. A fetch opcode cycle is always 1-machine cycle, whereas, execute cycle may be of one or more machine cycle depending
upon the length of the instruction.

**Instruction Fetch (FC)** An instruction of 1 or 2 or 3-bytes is extracted from the memory locations during the fetch and stored in the µP’s instruction register.

**Instruction Execute (EC)** The instruction is decoded and translated into specific activities during the execution phase. Thus, in an instruction cycle, instruction fetch, and instruction execute cycles are related as depicted in Fig. 3.2 (a). Every instruction cycle consists of 1, 2, 3, 4 or 5-machine cycles as indicated in Fig. 3.2 (c). One machine cycle is required each time the µP access memory or I/O port. The fetch cycle, in general could be 4 to 6-states whereas the execute cycle could be 3 to 6-states. The 1st machine cycle of any instruction is always the fetch cycle that provides identification of the instruction to be executed.

The fetch portion of an instruction cycle requires one machine cycle for each byte of instruction to be fetched. Since instruction is of 1 to 3 bytes long, the instruction fetch is one to 3-machine cycles in duration. The 1st machine cycle in an instruction cycle is always an opcode fetch. The 8-bits obtained during an opcode fetch are always interpreted as the Opcode of an instruction. The machine cycle including wait states is shown in Fig. 3.2 (c).

![Machine cycle including wait states](image)

**Fig. 3.2 (c)** Machine cycle including wait states
Note: Some instructions do not require any machine cycle other than that necessary to fetch the instruction. Other instructions, however, require additional machine cycles to write or read data to or from memory or I/O devices.

A typical fetch cycle is explained in Fig. 3.2 (d). In Fig. 3.2 (d) only two clock cycles have been shown as the requirement to read the instruction. Since the access time of the memory may vary and it may require more than 2-clock cycles, the microprocessor has to wait for more than 2-clocks duration before it receives the opcode instruction. Hence, most of the microprocessors have the provisions of introducing wait cycle within the fetch cycle to cope up with the slow memories or I/O devices.

![Fig. 3.2 (d) Fetch cycle](image)

**Opcode Fetch**

A microprocessor either reads or writes to the memory or I/O devices. The time taken to read or write for any instruction must be known in terms of the µP clock. The 1st step in communicating between the microprocessor and memory is reading from the memory. This read-ing process is called opcode fetch. The process of opcode fetch operation requires minimum 4-clock cycles $T_1$, $T_2$, $T_3$, and $T_4$ and is the 1st machine cycle ($M_1$) of every instruction.

In order to differentiate between the data byte pertaining to an opcode or an address, the machine cycle takes help of the status signal $\text{IO / M }, S_1$, and $S_0$. The $\text{IO / M} = 0$ indicates memory operation and $S_1 = S_0 = 1$ indicates Opcode fetch operation.

The opcode fetch machine cycle $M_1$ consists of 4-states ($T_1$, $T_2$, $T_3$, and $T_4$). The 1st 3-states are used for fetching (transferring) the byte from the memory and the 4th-state is used to decode it.
Thus, thorough understanding about the communication between memory and microprocessor can be achieved only after knowing the processes involved in reading or writing into the memory by the microprocessor and time taken w.r.t. its clock period. This can be explained by examples.

The process of implementation of each instruction follows the fetch and execute cycles. In other words, first the instruction is fetched from memory and then executed. Figs. 3.2 (e) and (f) depict these 2-steps for implementation of the instruction ADI 05H. Let us assume that the accumulator contains the result of previous operation i.e., 03H and instruction is held at memory locations 2030H and 2031H.

Fig. 3.2 (e) Instruction fetch : reads 1st byte (Opcode) in instruction register (IR)

The fetch part of the instruction is the same for every instruction. The control unit puts the contents of the program counter (PC) 2030H on the address bus. The 1st byte (opcode C6H in this example) is passed to the instruction register. In the execute cycle of the instruction, the control unit examines the opcode and as per interpretation further memory read or write operations are performed depending upon whether additional information/data are required or not. In this case, the data 05H from the memory is transferred through the data bus to the ALU. At the same time the control unit sends the contents of the accumulator (03H) to the ALU and performs the addition operation. The result of the addition operation 08H is passed to the accumulator overriding the previous contents 03H. On the completion of one instruction, the program counter is automatically incremented to point to the next memory location to execute the subsequent instruction.
Fig. 3.2 (f) Instruction execute: reads 2nd byte from memory and adds to accumulator.

Note: The slope of the edges of the clock pulses has been shown to be much exaggerated to indicate the existence of rise and fall time.

3.4 Timing Diagram of OPCODE FETCH

The process of opcode fetch operation requires minimum 4-clock cycles $T_1$, $T_2$, $T_3$, and $T_4$ and is the 1st machine cycle ($M_1$) of every instruction.

Example

Fetch a byte 41H stored at memory location 2105H.

For fetching a byte, the microprocessor must find out the memory location where it is stored. Then provide condition (control) for data flow from memory to the microprocessor. The process of data flow and timing diagram of fetch operation are shown in Figs. 3.3 (a), (b), and (c). The µP fetches opcode of the instruction from the memory as per the sequence below

- A low IO / M means microprocessor wants to communicate with memory.
• The µP sends a high on status signal $S_1$ and $S_0$ indicating fetch operation.

• The µP sends 16-bit address. AD bus has address in 1st clock of the 1st machine cycle, $T_1$.
• $AD_7$ to $AD_0$ address is latched in the external latch when ALE = 1.
• AD bus now can carry data.
• In $T_2$, the $RD$ control signal becomes low to enable the memory for read operation.

• The memory places opcode on the AD bus

• The data is placed in the data register (DR) and then it is transferred to IR.

---

**Fig. 3.3 (a) Opcode fetch**
During $T_3$ the $RD$ signal becomes high and memory is

- disabled.

5. During $T_4$ the opcode is sent for decoding and decoded in $T_4$.
6. The execution is also completed in $T_4$ if the instruction is single byte.

7. More machine cycles are essential for 2- or 3-byte instructions. The 1st machine cycle $M_1$ is meant for fetching the opcode. The machine cycles $M_2$ and $M_3$ are required either to read/ write data or address from the memory or I/O devices.

**Example**

Opcode fetch MOV B,C.

$T_1$: The 1st clock of 1st machine cycle ($M_1$) makes ALE high indicating address latch enabled which loads low-order address 00H on $AD_7$ $AD_0$ and high-order address 10H simultaneously on $A_{15}$ $A_8$. The address 00H is latched in $T_1$.

$T_2$: During $T_2$ clock, the microprocessor issues $RD$ control signal to enable the memory and memory places 41H from 1000H location on the data bus.

![Data flow from memory to microprocessor](image)

**Fig. 3.3 (b)** Data flow from memory to microprocessor
T₃: During T₃, the 41H is placed in the instruction register and RD = 1 (high) disables signal. It means the memory is disabled in T₃ clock cycle. The opcode cycle is completed by end of T₃ clock cycle.

T₄: The opcode is decoded in T₄ clock and the action as per 41H is taken accordingly. In other word, the content of C-register is copied in B-register. Execution time for opcode 41H is

Clock frequency of 8085 = 3.125 MHz

Time (T) for one clock = 1/3.125 MHz = 325.5 ns = 0.32 µS

Execution time for opcode fetch = 4T = 4*0.32 µS = 1.28 µS

Explain the execution of MVI B,05H stored at locations indicated below

<table>
<thead>
<tr>
<th>Mnemonics</th>
<th>Machine code</th>
<th>Memory Locations</th>
</tr>
</thead>
<tbody>
<tr>
<td>MVI B, 05H</td>
<td>06H</td>
<td>1000H</td>
</tr>
<tr>
<td></td>
<td>05H</td>
<td>1001H</td>
</tr>
</tbody>
</table>

![Fig. 3.3 (c) Opcode fetch (MOV B,C) Diagram](image-url)
The MVI B,05H instruction requires 2-machine cycles (M₁ and M₂). M₁ requires 4-states and M₂ requires 3-states, total of 7-states as shown in Fig. 3.3 (d). Status IO / M, S₁ and S₀ specifies the 1st machine cycle as the op-code fetch.

In T₁-state, the high order address {10H} is placed on the bus and low-order address {00H} on the bus AD₀ and ALE = 1. In T₂-state, the RD line goes low, AD₇

the data 06H from memory location 1000H are placed on the data bus. The fetch cycle becomes complete in T₃-state. The instruction is decoded in the T₄-state. During T₄-state, the contents of the bus are unknown. With the change in the status signal, IO / M = 0, S₁ = 1 and S₀ = 0, the 2nd machine cycle is identified as the memory read. The address is 1001H and the data byte [05H] is fetched via the data bus. Both M₁ and M₂ perform memory read operation, but the M₁ is called op-code fetch i.e., the 1st machine cycle of each instruction is identified as the opcode fetch cycle. Execution time for MBI B,05H i.e., memory read machine cycle and instruction cycle is
<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Instruction Byte</th>
<th>Machine Cycle</th>
<th>T-states</th>
</tr>
</thead>
<tbody>
<tr>
<td>MVI B,05H</td>
<td>Opcode</td>
<td>Opcode Fetch</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Immediate Data</td>
<td>Read Immediate Data</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>3</td>
</tr>
</tbody>
</table>

Clock frequency of 8085 = 3.125 MHz

Time (T) for one clock = 1/3.125 MHz = 0.32 µS

Time for Memory Read = 3T = 3*0.320 µS = 0.96 µS

Total Execution time for Instruction = 7T = 7*0.320 µS = 2.24 µS

**Read Cycle**

The high order address (A₁₅ A₈) and low order address (AD₇ AD₀) are asserted on

1st low going transition of the clock pulse. The timing diagram for IO/M read are shown in Fig. 3.3 (e) and (f). The A₁₅ A₈ remains valid in T₁, T₂, and T₃ i.e. duration of the bus cycle, but AD₇ AD₀ remains valid only in T₁. Since it has to remain valid for the whole bus cycle, it must be saved for its use in the T₂ and T₃.
Fig. 3.3 (e) Memory read timing diagram

ALE is asserted at the beginning of $T_1$ of each bus cycle and is negated towards the end of $T_1$. ALE is active during $T_1$ only and is used as the clock pulse to latch the address ($AD_7$ $AD_0$) during $T_1$. The RD is asserted near the beginning of $T_2$. It ends at the end of $T_3$. As soon as the $RD$ becomes active, it forces the memory or I/O port to assert data. $RD$ becomes inactive towards the end of $T_3$, causing the port or memory to terminate the data.

Fig. 3.4 (f) I/O Read timing diagram

Write Cycle
Immediately after the termination of the low order address, at the beginning of the $T_2$, data

is asserted on the address/data bus by the processor. WR control is activated near the start of $T_2$ and becomes inactive at the end of $T_3$. The processor maintains valid data until after WR is terminated. This ensures that the memory or port has valid data while WR is active.

It is clear from Figs. 3.3 (g) and (h) that for READ bus cycle, the data appears on the bus as a result of activating RD and for the WR bus cycle, the time the valid data is on the bus overlaps
the time that the WR is active.

**Fig. 3.3 (g) Memory write timing diagram.**

**Fig. 3.3 (h) I/O write timing diagram**

STA

The STA instruction stands for storing the contents of the accumulator to a memory location whose address is immediately available after the instruction (STA). The 8085 have 16-address lines, it can address $2^{16} = 64$ K. Since the STA instruction is meant to store the contents of the accumulator to the memory location, it is a 3-byte instruction. 1st byte is the opcode, the 2nd and 3rd bytes are the address of the memory locations. The storing of the STA instruction in the memory locations is as
Opcode       1st byte
Low address  2nd byte
High address 3rd byte

Three machine cycles are required to fetch this instruction: opcode Fetch transfers the
opcode from the memory to the instruction register. The 2-byte address is then transferred, 1-byte
at a time, from the memory to the temporary register. This requires two Memory read machine
cycles. When the entire instruction is in the microprocessor, it is executed. The execution process
transfers data from the microprocessor to the memory. The contents of the accumulator are
transferred to memory, whose address was previously transferred to the microprocessor by the
preceding 2-Memory Read machine cycles. The address of the memory location to be written is
generated as

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Instruction Byte</th>
<th>Machine Cycle</th>
<th>T-states</th>
</tr>
</thead>
<tbody>
<tr>
<td>STA</td>
<td>Opcode LOW Address</td>
<td>Opcode Fetch Memory Read Memory Read</td>
<td>4 3 3 13</td>
</tr>
<tr>
<td>STA</td>
<td>HIGH Address</td>
<td>Memory Write</td>
<td>3</td>
</tr>
</tbody>
</table>

The high order address byte in the temporary register is transferred to the address latch and
the low order address byte is transferred to the address/data latch. This data transfer is affected
by a Memory Write machine cycle. Thus 3-byte STA instruction has four machine cycles in its
instruction cycle.

The timing and control section of the microprocessor automatically generates the proper
machine cycles required for an instruction cycle from the information provided by the opcode. The
The timing diagram of the instruction STA is shown in Fig. 3.3 (i). The status of IO / M , S₁ and S₀ for 4-machine cycles are obtained from Table 3.1. The condition of IO / M , S₁ and S₀ would be 0, 1 and 1 respectively in MC₁. The status of ALE is high at the beginning of 1st state of each machine cycle so that AD₀ work as the address bus. RD remains high during AD₇ 1st state of each machine cycle, since during 1st state of each machine cycle AD₀ work as address bus. It remains high during 4th state of the 1st machine cycle also as the 4th state is used to decode the op code for generating the required control signals.

The opcode fetch of STA instruction has 4-states (clock cycles). Three states have been used to read the opcode from the main memory and the 4th to decode it and set up the subsequent machine cycle.
The action of memory read or write cycles containing 3-states i.e., T₁, T₂, and T₃ are explained as

T₁ : During this period the address and control signals for the memory access are set up.

T₂ : The µP checks up the READY and HOLD control lines. If READY = 0, indicating a slow memory device, the µP enters in the wait state until READY = 1, indicating DMA request, then only the µP floats the data transfer lines and enters into wait until HOLD = 0.

T₃ : In memory read cycles the µP transfers a byte from the data bus to an internal register and in memory write cycle the µP transfers a byte from an internal register to the data bus.

Thus STA instruction requires 4-machine cycles containing 13-states (clock cycles). With a typical clock of 3 MHz (= 330 ns), the STA instruction requires 13*330 ns = 4.29 ms for its execution.

3.5 Summary
Timing Diagram is a graphical representation. It represents the execution time taken by each instruction in a graphical format. The execution time is represented in T-states.

3.6 Keywords
- Processor Cycle
- Timing Diagram
- status signals
- T-states
• opcode fetch
• Read process
• Machine cycle

3.7 Exercise

5. Calculate the execution time required for executing the instructions with the system frequency of 3 MHz.
   MOV A,B
   MOV C,D
   MOV A,M
   MVI A,05H
   MVI B,05H

2. With relevant diagram, explain the role of timing and control unit in the operation of microprocessors.

3. Define (a) Instruction cycle, (b) Machine cycle, (c) Clock cycle.

4. Draw and explain the timing diagram for the memory read instruction.

5. Draw and explain the timing diagram for the I/O write instruction.

Unit 4

Simple Programs

Structure
4.1 Introduction
4.2 Objectives
4.3 Microprocessor Programs
4.4 Summary
4.1 Introduction

The 8085 is a binary compatible follow up on the 8080, using the same basic instruction set as the 8080. Only a few minor instructions were new to the 8085 above the 8080 set. In this chapter we go through the microprocessor programs.

4.2 Objectives

At the end of this chapter you will be able to:

- Write the programs to perform operations like addition, subtraction and many more.

4.3 Microprocessor Programs

Program 1: To perform addition of two 8 bit numbers using 8085.

ALGORITHM:

1) Start the program by loading the first data into Accumulator.
2) Move the data to a register (B register).
3) Get the second data and load into Accumulator.
4) Add the two register contents.
5) Check for carry.
6) Store the value of sum and carry in memory location.
7) Terminate the program.

PROGRAM:

```
MVI    C, 00   Initialize C register to 00
LDA    4150   Load the value to Accumulator.
MOV    B, A   Move the content of Accumulator to B register.
LDA    4151   Load the value to Accumulator.
ADD    B      Add the value of register B to A
JNC    LOOP   Jump on no carry.
INR    C      Increment value of register C
LOOP:  STA    4152 Store the value of Accumulator (SUM).
        MOV    A, C Move content of register C to Acc.
        STA    4153 Store the value of Accumulator (CARRY)
        HLT    Halt the program.
```

OBSERVATION:

Input: 80 (4150)
80 (4251)

Output: 00 (4152)
01 (4153)

RESULT:
Thus the program to add two 8-bit numbers was executed.

**Program2: To perform the subtraction of two 8 bit numbers using 8085.**

**ALGORITHM:**

1. Start the program by loading the first data into Accumulator.
2. Move the data to a register (B register).
3. Get the second data and load into Accumulator.
4. Subtract the two register contents.
5. Check for carry.
6. If carry is present take 2’s complement of Accumulator.
7. Store the value of borrow in memory location.
8. Store the difference value (present in Accumulator) to a memory location and terminate the program.

**PROGRAM:**

```
MVI C, 00            Initialize C to 00
LDA 4150            Load the value to Acc.
MOV B, A            Move the content of Acc to B register.
LDA 4151            Load the value to Acc.
SUB B               Subtract the two register contents.
JNC LOOP            Jump on no carry.
CMA                 Complement Accumulator contents.
INR A               Increment value in Accumulator.
INR C               Increment value in register C
```
LOOP: STA 4152 Store the value of A-reg to memory address.

MOV A, C Move contents of register C to Accumulator.

STA 4153 Store the value of Accumulator memory address.

HLT Terminate the program.

OBSERVATION:

Input: 06 (4150)
        02 (4251)

Output: 04 (4152)
         01 (4153)

RESULT:

Thus the program to subtract two 8-bit numbers was executed.

Program3: To perform the multiplication of two 8 bit numbers using 8085.

ALGORITHM:

1) Start the program by loading HL register pair with address of memory location.
2) Move the data to a register (B register).
3) Get the second data and load into Accumulator.
4) Add the two register contents.
5) Check for carry.
6) Increment the value of carry.
7) Check whether repeated addition is over and store the value of product and carry in memory location.
8) Terminate the program.

PROGRAM:

MVI D, 00 Initialize register D to 00
MVI A, 00 Initialize Accumulator content to 00
LXI H, 4150
MOV B, M
Get the first number in B - reg INX H
MOV C, M Get the second number in C- reg.

LOOP: ADD B Add content of A - reg to register B.
JNC NEXT Jump on no carry to NEXT.
INR D Increment content of register D

NEXT: DCR C Decrement content of register C.
JNZ LOOP Jump on no zero to address
STA 4152 Store the result in Memory
MOV A, D
STA 4153 Store the MSB of result in Memory
HLT Terminate the program.
OBSERVATION:

Input: FF (4150)
       FF (4151)

Output: 01 (4152)
        FE (4153)

RESULT:

Thus the program to multiply two 8-bit numbers was executed.

Program 4: To perform the division of two 8 bit numbers using 8085.

ALGORITHM:

1) Start the program by loading HL register pair with address of memory location.
2) Move the data to a register (B register).
3) Get the second data and load into Accumulator.
4) Compare the two numbers to check for carry.
5) Subtract the two numbers.
6) Increment the value of carry.
7) Check whether repeated subtraction is over and store the value of product and carry in memory location.
8) Terminate the program.

PROGRAM:

LXI     H, 4150
MOV     B, M          Get the dividend in B -
MVI C, 00 Clear C - reg for quotient
INX H
MOV A, M Get the divisor in A - reg.

NEXT: CMP B Compare A - reg with register B.

JC LOOP Jump on carry to LOOP
SUB B Subtract A - reg from B-reg.

INR C Increment content of register C.

JMP NEXT Jump to NEXT

LOOP: STA 4152 Store the remainder in Memory

MOV A, C

STA 4153 Store the quotient in memory

HLT Terminate the program.

OBSERVATION:

Input: FF (4150)
       FF (4251)

Output: 01 (4152) ---- Remainder
         FE (4153) ---- Quotient
RESULT:

Thus the program to divide two 8-bit numbers was executed.

**Program 5: To find the largest number in an array of data using 8085 instruction set.**

**ALGORITHM:**

1) Load the address of the first element of the array in HL pair
2) Move the count to B - reg.
3) Increment the pointer
4) Get the first data in A - reg.
5) Decrement the count.
6) Increment the pointer
7) Compare the content of memory addressed by HL pair with that of A - reg.
8) If Carry = 0, go to step 10 or if Carry = 1 go to step 9
9) Move the content of memory addressed by HL to A - reg.
10) Decrement the count
11) Check for Zero of the count. If ZF = 0, go to step 6, or if ZF = 1 go to next step.
12) Store the largest data in memory.
13) Terminate the program.

**PROGRAM:**
LXI    H,4200        Set pointer for array
MOV    B,M        Load the Count
INX    H
MOV    A,M        Set 1st element as largest data
DCR    B        Decrement the count
LOOP:  INX    H
        CMP    M        If A-reg > M go to AHEAD
        JNC    AHEAD
        MOV    A,M        Set the new value as largest
AHEAD:  DCR    B
        JNZ    LOOP        Repeat comparisons till count = 0
        STA    4300        Store the largest value at 4300
        HLT

OBSERVATION:

Input:      05 (4200) ----- Array Size
            0A (4201)
            F1 (4202)
            1F (4203)
            26 (4204)
            FE (4205)
Output:     FE (4300)
RESULT:

Thus the program to find the largest number in an array of data was executed

Program 6: To find the smallest number in an array of data using 8085 instruction set.

ALGORITHM:

1) Load the address of the first element of the array in HL pair
2) Move the count to B - reg.
3) Increment the pointer
4) Get the first data in A - reg.
5) Decrement the count.
6) Increment the pointer
7) Compare the content of memory addressed by HL pair with that of A - reg.
8) If carry = 1, go to step 10 or if Carry = 0 go to step 9
9) Move the content of memory addressed by HL to A - reg. 10) Decrement the count
11) Check for Zero of the count. If ZF = 0, go to step 6, or if ZF = 1 go to next step. 12) Store the smallest data in memory.
13) Terminate the program.

PROGRAM:

LXI H,4200  Set pointer for array
MOV B,M   Load the Count
INX H
MOV A,M   Set 1st element as largest data
DCR B Decrement the count

LOOP:
    INX H
    CMP M If A-reg < M go to AHEAD

AHEAD
    JC AHEAD
    MOV A,M Set the new value as smallest

AHEAD:
    DCR B
    JNZ LOOP Repeat comparisons till count = 0
    STA 4300 Store the largest value at 4300
    HLT

OBSERVATION:

Input: 05 (4200) ---- Array Size
       0A (4201)
       F1 (4202)
       1F (4203)
       26 (4204)
       FE (4205)

Output: 0A (4300)

RESULT:

Thus the program to find the smallest number in an array of data was executed

Program 7: To write a program to arrange an array of data in ascending order

ALGORITHM:
1. Initialize HL pair as memory pointer
2. Get the count at 4200 into C - register
3. Copy it in D - register (for bubble sort (N-1) times required)
4. Get the first value in A - register
5. Compare it with the value at next location.
6. If they are out of order, exchange the contents of A - register and Memory
7. Decrement D - register content by 1
8. Repeat steps 5 and 7 till the value in D- register become zero
9. Decrement C - register content by 1
10. Repeat steps 3 to 9 till the value in C - register becomes zero

PROGRAM:

LXI H,4200
MOV C,M
DCR C

REPEAT:
MOV D,C
LXI H,4201

LOOP:
MOV A,M
INX H
CMP M
JC SKIP
MOV B,M
MOV M,A
DCX H
MOV M,B
INX H

SKIP:
DCR D
JNZ LOOP
DCR C
JNZ REPEAT
HLT

OBSERVATION:

Input:
4200 05 (Array Size)
4201 05
4202 04
4203 03
4204 02
4205 01

Output:
4200 05 (Array Size)
4201 01
4202 02
4203 03
4204 04
4205 05

RESULT:

Thus the given array of data was arranged in ascending order.

Program 8: To write a program to arrange an array of data in descending order

ALGORITHM:

1. Initialize HL pair as memory pointer
2. Get the count at 4200 into C - register
3. Copy it in D - register (for bubble sort (N-1) times required)
4. Get the first value in A - register
5. Compare it with the value at next location.
6. If they are out of order, exchange the contents of A - register and Memory
7. Decrement D - register content by 1
8. Repeat steps 5 and 7 till the value in D- register become zero
9. Decrement C - register content by 1
10. Repeat steps 3 to 9 till the value in C - register becomes zero

PROGRAM:

```
LXI    H,4200
MOV    C,M
DCR    C
REPEAT: MOV    D,C
         LXI    H,4201
LOOP:   MOV    A,M
         INX    H
         CMP    M
         JNC    SKIP
         MOV    B,M
         MOV    M,A
         DCX    H
         MOV    M,B
         INX    H
SKIP:   DCR    D
         JNZ    LOOP
         DCR    C
         JNZ    REPEAT
         HLT
```
OBSERVATION:

Input: 
4200  05 (Array Size) 
4201  01 
4202  02 
4203  03 
4204  04 
4205  05 

Output: 
4200  05 (Array Size) 
4201  05 
4202  04 
4203  03 
4204  02 
4205  01 

RESULT:

Thus the given array of data was arranged in descending order.

Program 9: To convert two BCD numbers in memory to the equivalent HEX number using 8085 instruction set

ALGORITHM:

1) Initialize memory pointer to 4150 H 
2) Get the Most Significant Digit (MSD) 
3) Multiply the MSD by ten using repeated addition 
4) Add the Least Significant Digit (LSD) to the result obtained in previous step 
5) Store the HEX data in Memory
PROGRAM:

LXI H,4150
MOV A,M Initialize memory pointer
ADD A MSD X 2
MOV B,A Store MSD X 2
ADD A MSD X 4
ADD A MSD X 8
ADD B MSD X 10
INX H Point to LSD
ADD M Add to form HEX
INX H
MOV M,A Store the result
HLT

OBSERVATION:

Input: 4150 : 02 (MSD)  
        4151 : 09 (LSD)

Output: 4152 : 1D H

RESULT: Thus the program to convert BCD data to HEX data was executed.

Program 10: To convert given Hexa decimal number into its equivalent BCD number using 8085 instruction set

ALGORITHM:

1) Initialize memory pointer to 4150 H
2) Get the Hexa decimal number in C - register
3) Perform repeated addition for C number of times
4) Adjust for BCD in each step
5) Store the BCD data in Memory

PROGRAM:

LXI H,4150  Initialize memory pointer
MVI D,00  Clear D-reg for Most significant Byte
XRA A  Clear Accumulator
MOV C,M  Get HEX data
LOOP2: ADI 01  Count the number one by one
        DAA  Adjust for BCD count
        JNC LOOP1
        INR D
LOOP1: DCR C
        JNZ LOOP2
        STA 4151  Store the Least Significant Byte
        MOV A,D
        STA 4152  Store the Most Significant Byte
        HLT

OBSERVATION:

Input: 4150 : FF

Output: 4151 : 55 (LSB)
         4152 : 02 (MSB)

RESULT: Thus the program to convert HEX data to BCD data was
Program 11: To convert given Hexadecimal number into its equivalent ASCII number using 8085 instruction set.

ALGORITHM:

1. Load the given data in A-register and move to B-register
2. Mask the upper nibble of the Hexadecimal number in A-register
3. Call subroutine to get ASCII of lower nibble
4. Store it in memory
5. Move B-register to A-register and mask the lower nibble
6. Rotate the upper nibble to lower nibble position
7. Call subroutine to get ASCII of upper nibble
8. Store it in memory
9. Terminate the program.

PROGRAM:

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Address</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LDA</td>
<td>4200</td>
<td>Get Hex Data</td>
</tr>
<tr>
<td>MOV</td>
<td>B,A</td>
<td></td>
</tr>
<tr>
<td>ANI</td>
<td>0F</td>
<td>Mask Upper Nibble</td>
</tr>
<tr>
<td>CALL</td>
<td>SUB1</td>
<td>Get ASCII code for upper nibble</td>
</tr>
<tr>
<td>STA</td>
<td>4201</td>
<td></td>
</tr>
<tr>
<td>MOV</td>
<td>A,B</td>
<td></td>
</tr>
<tr>
<td>ANI</td>
<td>F0</td>
<td>Mask Lower Nibble</td>
</tr>
<tr>
<td>RLC</td>
<td></td>
<td></td>
</tr>
<tr>
<td>RLC</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Program 12: To convert given ASCII Character into its equivalent Hexa Decimal number using 8085 instruction set.

ALGORITHM:

1. Load the given data in A-register
2. Subtract 30 H from A-register
3. Compare the content of A - register with 0A H
5. Subtract 07 H from A - register
6. Store the result
7. Terminate the program

PROGRAM:

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>LDA</td>
<td>4500</td>
</tr>
<tr>
<td>SUI</td>
<td>30</td>
</tr>
<tr>
<td>CPI</td>
<td>0A</td>
</tr>
<tr>
<td>JC</td>
<td>SKIP</td>
</tr>
<tr>
<td>SUI</td>
<td>07</td>
</tr>
<tr>
<td>SKIP:</td>
<td>STA</td>
</tr>
<tr>
<td></td>
<td>HLT</td>
</tr>
</tbody>
</table>

OBSERVATION:

Input: 4500 31
Output: 4501 0B

RESULT:

Thus the given ASCII character was converted into its equivalent Hexa Value.

Program 13: To find the square of the number from 0 to 9 using a Table of Square.

ALGORITHM:

1. Initialize HL pair to point Look up table
2. Get the data .
3. Check whether the given input is less than 9.
4. If yes go to next step else halt the program
5. Add the desired address with the accumulator content
6. Store the result

PROGRAM:

LXI H,4125         Initialsie Look up table address
LDA 4150          Get the data
CPI 0A             Check input > 9
JC AFTER           if yes error
MVI A,FF           Error Indication
STA 4151
HLT

AFTER:
MOV C,A           Add the desired Address
MVI B,00
DAD B
MOV A,M
STA 4151         Store the result
HLT               Terminate the program

LOOKUP TABLE:

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>4125</td>
<td>01</td>
</tr>
<tr>
<td>4126</td>
<td>04</td>
</tr>
<tr>
<td>4127</td>
<td>09</td>
</tr>
<tr>
<td>4128</td>
<td>16</td>
</tr>
<tr>
<td>4129</td>
<td>25</td>
</tr>
<tr>
<td>4130</td>
<td>36</td>
</tr>
<tr>
<td>4131</td>
<td>49</td>
</tr>
<tr>
<td>4132</td>
<td>64</td>
</tr>
<tr>
<td>4133</td>
<td>81</td>
</tr>
</tbody>
</table>
OBSERVATION:

Input: 4150: 05

Output: 4151 25 (Square)

Input: 4150: 11

Output: 4151: FF (Error Indication)

RESULT: Thus the program to find the square of the number from 0 to 9 using a Look up table was executed.

4.4 Summary
The 8085 is a binary compatible follow up on the 8080, using the same basic instruction set as the 8080. Only a few minor instructions were new to the 8085 above the 8080 set. In this chapter we studied the microprocessor programs.

4.5 Keywords

- Binary
- Microprocessor

4.6 Exercise
1) Write an assembly program to add two numbers
2) Write an assembly program to find greatest between two numbers
3) Write an assembly program to multiply a number by 8

Unit 1

Memory mapping
Structure

1.1 Introduction
1.2 Objectives
1.3 Memory Mapping
1.4 Moving Data to and from Memory.
1.5 Summary
1.6 Keywords
1.7 Exercise

1.1 Introduction
The 8085 family can address 64K bytes of memory which is used for both code and data space. Memory is accessed via 20 pins. 8 address high pins and 8 pins that are used both for the 8 address low signals during the address setup phase and for the 8 data signals during the data transfer phase. 4 pins are used for control. A READY input line allows memory or I/O access to slow down the data transfer. This allows slow memory or I/O hardware to be easily interfaced. And a HOLD line allows peripheral hardware to take over the memory bus allowing DMA transfers to be implemented.

1.2 Objectives
At the end of this chapter you will be able to:

- Explain Memory Mapping
- Know how to Move Data to and from Memory.

1.3 Memory Mapping
In order to help us understand the role of the address bus and the concept of memory mapping, we take another look at the memory chips themselves. Figure 1 schematically shows a hypothetical memory chip, which has storage for 16 words (each 8 bits in length) of information. Actual memory chips typically store many hundreds of kilobytes and have a correspondingly larger number of memory access pins. You should observe however, that in our hypothetical chip of Figure 1.1
Fig. 1.1- Schematic of a Hypothetical 16-Word Memory Chip

The memory chip is composed of two functional sections - the Boolean decoding logic section and the actual storage section. The decoding section is responsible for controlling the transfer of data to/from the computer data bus from/to a storage address in the chip. This access control is based upon the status of the memory address pins (MA3..MA0), the Read/Write pin and the Chip Enable (or Chip Select) pin.

The Chip Enable (or Chip Select) pin is the access device for the chip. In order to use the chip, the Chip Enable pin must be set either high or low (depending on the manufacturer's design). The Read/Write pin is used on RAM chips to define whether data should flow into or out of the chip. Depending upon the high or low state of this pin, data will be written to or read from the appropriate address location in the chip. As an example, if the microprocessor in our hypothetical system is to force the memory chip to place its fourth row (0011) of storage (ie: 00000111) onto the data bus, then it would have to set:

\[
\begin{align*}
    \text{MA3} &= 0 \\
    \text{MA2} &= 0 \\
    \text{MA1} &= 1 \\
    \text{MA0} &= 1 \\
    \text{Read/Write} &= 1 \\
    \text{Chip Enable} &= 1
\end{align*}
\]
thus accessing the third row in the memory chip.

The microprocessor controls the memory chip address pins and chip enable (select) pin, by setting appropriate lines on the address bus high or low. In other words, a selection of address lines from the microprocessor is connected to pins on the memory chips. These lines are selectively set or reset by the microprocessor in order to make the memory chips respond in the desired manner. The Read/Write line of the memory chip is tied to a corresponding driver line on the microprocessor.

The dilemma that immediately arises is that if there are many identical memory chips, connected to the address bus of a microprocessor system in an identical manner, then all the chips will respond simultaneously to each request from the microprocessor. This is clearly ridiculous, since it would imply that no matter how many memory chips we have, we would only effectively have the storage capacity of one chip. Since it would be equally ridiculous to have scores of specially designed memory chips, the problem is overcome through the use of "memory mapping" techniques. Each memory chip in a microprocessor system must have a unique connection to the microprocessor address bus, otherwise a conflict occurs. This unique addressing is achieved through address bus decoding logic, as shown in Figure 6.6. In a simple system, this logic can be implemented through Boolean logic gates.

Let us now assume that two of the memory chips in the system shown in Figure 1.2, both have a structure identical to the hypothetical one shown in Figure 1.2. The following Boolean logic could be used to decode the address lines for chip 1:

\[
\begin{align*}
MA0 &= A0 \\
MA1 &= A1 \\
MA2 &= A2 \\
MA3 &= A3 \\
\text{Chip Enable} &= X1
\end{align*}
\]

where:

\[
X1 = A4 + A5 + A6 + A7 + A8 + A9 + A10 + A11 + A12 + A13 + A14 + A15
\]

Memory chip number 2 in the system could have the following Boolean logic gates for address bus decoding:
MA0 = A0
MA1 = A1
MA2 = A2
MA3 = A3
Chip Enable = A4 . X2

where:

\[ X2 = A5 + A6 + A7 + A8 + A9 + A10 + A11 + A12 + A13 + A14 + A15 \]

This arrangement implies that memory chip 1 can only be accessed (enabled) when all microprocessor address lines A4 through to A15 are low. Otherwise chip 1 remains locked. Memory chip 2 can only be accessed when address line A4 is high and lines A5 to A15 are low. Otherwise chip 2 remains locked. Table 1 shows the effect that this has from the microprocessor's point of view. A number of such chips could be selectively "mapped" so that to the microprocessor, all address locations from 0000 0000 0000 0000 to 1111 1111 1111 1111 can be accessed, with no two chips responding to the same address.

<table>
<thead>
<tr>
<th>Address Bus Value</th>
<th>Memory Chip Accessed</th>
<th>Memory Chip Internal Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000 0000 0000 0000</td>
<td>1</td>
<td>0000</td>
</tr>
<tr>
<td>0000 0000 0000 0001</td>
<td>1</td>
<td>0001</td>
</tr>
<tr>
<td>0000 0000 0000 0010</td>
<td>1</td>
<td>0010</td>
</tr>
<tr>
<td>0000 0000 0000 0011</td>
<td>1</td>
<td>0011</td>
</tr>
<tr>
<td>.</td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>0000 0000 0000 1111</td>
<td>1</td>
<td>1111</td>
</tr>
<tr>
<td>0000 0000 0001 0000</td>
<td>2</td>
<td>0000</td>
</tr>
<tr>
<td>0000 0000 0001 0001</td>
<td>2</td>
<td>0001</td>
</tr>
<tr>
<td>0000 0000 0001 0010</td>
<td>2</td>
<td>0010</td>
</tr>
<tr>
<td>0000 0000 0001 0011</td>
<td>2</td>
<td>0011</td>
</tr>
<tr>
<td>.</td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>0000 0000 0001 1111</td>
<td>2</td>
<td>1111</td>
</tr>
</tbody>
</table>

*Table 1 - Memory Mapping of Identical Memory Chips in Figure 6.6 to Unique Locations with respect to Microprocessor*
Memory mapping is a vitally important concept in computing because most devices are interfaced to the microprocessor via a range of memory addresses. For example, the parallel to serial conversion chip (UART) in a computer system has a number of internal registers. One of the registers in the chip contains data that tells the device how to perform its function and another register is used for incoming data and yet another for outgoing data. All these registers are mapped, just as if they were normal system memory. The microprocessor communicates with them as though they are normal memory locations, when in reality they are often part of another special purpose chip or system.

The same memory mapping technique can be used to interface disk-drive controllers and graphics controller cards to a microprocessor within a computer system. A range of memory locations or registers in these devices are mapped into system memory as if they are normal memory chips. The microprocessor moves data to and from them as though they are normal memory locations. The devices that are mapped onto the system use the data supplied by the microprocessor to do their respective tasks. For example, the graphics controller card uses the data provided by the microprocessor to create an image on a screen. This is shown in Figure 2.

Fig. 2 - Using Memory Mapping Techniques to Create a Common Shared Area of Memory (or Registers) to Transfer Data To and From Non-Memory Devices (eg: Graphics Controller Card)
Most computer systems are composed of many memory chips and each and every memory chip must ultimately have a unique address. Most computer users would be familiar with the concept of memory in the form of a card that plugs into the address/data bus structure of the computer. Each card may hold a number of memory chips. The manufacturers of the card generate the decoding logic that enables the entire card to be treated as one large "chip" (because each of the memory chips is actually decoded on the card itself). Users then complete the memory mapping function by setting dip-switches on the card to make the entire card map into an appropriate region of the computer's system memory. The dip-switches enable one card to be easily mapped into a number of different memory locations so that they can be used on a range of computers.

Newer computer systems resolve the problem of memory mapping for cards by using very sophisticated address and data bus structures. Each device that plugs into the address and data bus structure has to be equipped with a special interface that is capable of responding to commands issued by the CPU. The memory mapping and configuration process can then occur automatically. Although this may appear, to end users, to be a very useful solution to the memory mapping problem, it creates considerable problems and costs for those that have to generate cards for such systems.

1.4 Moving Data to and from Memory.

So far we have moved immediate numbers into registers, and values from register to register. Now we will see how to retrieve data from memory. In this section we will also see what a source program looks like. To move data into a memory location, we must first decide what address to use. Each of the 65,536 memory locations has a unique address, a number from $0000 to $FFFF. It takes a 16-bit number count that high, and it can be done on the 8085. We have already seen that the 8085 has seven 8-bit registers. Six of the registers can be "doubled-up" to give you three 16-bit registers. The B and the C register give you the BC register, the D and E registers give you the DE register, and the H and L registers give you, surprise, the HL register.

The 16-bit registers are not separate from the 8-bit registers. If you store an 8-bit number in L, and then a 16-bit number in HL, the 8-bit number is lost. To add to the confusion, most assemblers allow you to use only the first letter of a 16-bit register. Then the only way to tell the difference between an 8 or 16-bit register is from context, i.e. from the instruction that precedes
it. Here we will use both letters for a 16-bit register.

The HL register is the only register that can be used to specify a memory address. It is called a pointer register because it points to a memory location. The H register contains the high byte of the address, and the L register the low byte (that's why it's not called the FG register.)

High and low bytes may be unfamiliar to you. Consider the decimal number 47. From convention we know that 4 is the number of "tens", and the 7 is the number of "units", high and then low. For the hex number $47, 4 is the number of "sixteen", and 7 is the number of "units". In the same way for the number $4768, $47 is the number of "two-hundred-fifty-sixes", and $68 is the number of "units". We split it this way because the first two digits can fit in the high byte (register) and the second two can fit in the low register. However, as we shall see later, sometimes the low byte comes first, with the high byte following, but the HL register is always high-low.

The instruction that loads an address into the HL register is the LXI instruction. The LXI instruction loads a 16-bit immediate value into the register pair designated in the first half of the operand, the destination. The second half of the operand is the 16-bit immediate value. The format is LXI rp, $nnnn, where rp is any of the three register pairs, and $nnnn is a 2 byte (up to four hex digit) number.

Once HL points to a memory address, we can treat that memory location just like it was a register. We can use:

```
MVI M,$AB ;memory location pointed to by HL will equal $AB
MOV A,M ;A register will have value of memory location
```

The LXI instruction requires three bytes of memory. The first specifies the instruction and the destination register (all in one byte), and the last two contain the immediate 16-bit value. It is interesting (and confusing) to note that addresses in memory are always stored low byte first.

The instruction LXI HL,$1234 assembles into the three consecutive bytes $21, $34, $12. $21 is the LXI H, instruction. The $34 and $12 is the address with the low byte first. You don't have to
worry about the order when you are programming because the assembler puts all the addresses in the proper order.

You do have to worry about it when debugging your program. Sometimes you have to examine the machine language code to find the problem, and this high-low business can get confusing. To repeat, addresses in the registers are always stored high-low. Addresses in memory are stored low-high.

### 1.5 Summary

The processor has seven 8-bit registers named A, B, C, D, E, H, and L, where A is the 8-bit accumulator and the other six can be used as independent byte-registers or as three 16-bit register pairs, BC, DE, and HL, depending on the particular instruction. Some instructions use HL as a (limited) 16-bit accumulator. As in the 8080, the contents of the memory address pointed to by HL could be accessed as pseudo register M. It also has a 16-bit stack pointer to memory and a 16-bit program counter. HL pair is called the primary data pointers

### 1.6 Keywords

- READY
- HOLD
- DMA
- CPU
1.7 Exercise

1) Explain Memory Mapping.
2) How to Move Data to and from Memory.

Unit 2
Interrupts

Structure
2.1 Introduction
2.2 Objectives
2.3 Interrupts
2.4 8085 Interrupt Types
2.5 Maskable Interrupts
2.6 Non-maskable Interrupts
2.7 Summary
2.8 Keywords
2.9 Exercise

2.1 Introduction
The intel 8085 has five interrupt inputs namely TRAP, RST7.5, RST6.5, RST5.5 AND INTR. The INTR has the lowest priority. when interrupts are to be used they are enabled by software using teh off interrupts of intel 8085. The instruction EI sets the interrupt enable flip flop to
enable the interrupts ..the use of the instruction EI enables all the interrupt..the instruction DI
disable the interrupt....the DI instruction sets resets teh interrupt enable flip flop and disable all
teh interrupt except non-maskable interrupt TRAP..the system Reset also resets the interrupt
enable flip flop.

2.2 Objectives
At the end of this chapter you will able to:

- Explain Interrupts
- List the 8085 Interrupt Types
- Explain Maskable and Non-maskable Interrupts

2.3 Interrupts

Definition: The meaning of ‘interrupts’ is to break the sequence of operation. While the cpu is
executing a program, on ‘interrupt’ breaks the normal sequence of execution of instructions,
diverts its execution to some other program called Interrupt Service Routine (ISR). After
executing ISR, the control is transferred back again to the main program. Interrupt processing is
an alternative to polling.

Need for Interrupt: Interrupts are particularly useful when interfacing I/O devices, that provide
or require data at relatively low data transfer rate.

Types of Interrupts: There are two types of Interrupts in 8086. They are:

(i) **Hardware Interrupts** and

(ii) **Software Interrupts**

(i) **Hardware Interrupts** (External Interrupts). The Intel microprocessors support hardware
interrupts through:

- Two pins that allow interrupt requests, INTR and NMI
• One pin that acknowledges, INTA, the interrupt requested on INTR.

**INTR and NMI**

- **INTR** is a maskable hardware interrupt. The interrupt can be enabled/disabled using STI/CLI instructions or using more complicated method of updating the FLAGS register with the help of the POPF instruction.
- When an interrupt occurs, the processor stores FLAGS register into stack, disables further interrupts, fetches from the bus one byte representing interrupt type, and jumps to interrupt processing routine address of which is stored in location 4 * <interrupt type>. Interrupt processing routine should return with the IRET instruction.
- **NMI** is a non-maskable interrupt. Interrupt is processed in the same way as the INTR interrupt. Interrupt type of the NMI is 2, i.e. the address of the NMI processing routine is stored in location 0008h. This interrupt has higher priority than the maskable interrupt.
- – Ex: NMI, INTR.

(ii) **Software Interrupts** (Internal Interrupts and Instructions). Software interrupts can be caused by:

- INT instruction - breakpoint interrupt. This is a type 3 interrupt.
- INT <interrupt number> instruction - any one interrupt from available 256 interrupts.
- INTO instruction - interrupt on overflow
- Single-step interrupt - generated if the TF flag is set. This is a type 1 interrupt. When the CPU processes this interrupt it clears TF flag before calling the interrupt processing routine.
- Processor exceptions: Divide Error (Type 0), Unused Opcode (type 6) and Escape opcode (type 7).
- Software interrupt processing is the same as for the hardware interrupts.
- - Ex: INT n (Software Instructions)
- **Control is provided** through:
  - IF and TF flag bits
  - IRET and IRETD
2.4 8085 Interrupt Types

The 8085 has facilities for servicing interrupts similar to the 8080. The functional items required are an Interrupt Request (INTR) pin, an Interrupt Acknowledge (INTA) pin, an Interrupt Enable (INTE) pin, eight interrupt vectors in low RAM, and the Restart instruction. These perform in the same way as the 8080 interrupt system. Here is a brief review:

1. A program is running normally in the system. The 8214 Priority Interrupt Controller or similar circuit has its compare mask set to some priority level. The Interrupt Enable bit has been set on by some previous routine, enabling interrupts.

2. A device wishes to interrupt the system. It raises its own line which connects directly to the 8214. The 8214 compares this request with the current status of the system. If the new request is higher in priority than the existing (if any), the interrupt will be allowed. If not, the interrupt will be latched for later use, but no further action is taken.

3. The Interrupts Enabled line exiting the 8085 is high, indicating that interrupts are permitted. The 8214 raises the Interrupt line, which causes the MP to finish the current instruction, and then enter an interrupt service cycle. The MP generates the Interrupts Acknowledge line at the beginning of this cycle to permit the 8214 to proceed.

4. Upon receipt of the INTA line, the 8214 along with an 8212 octal latch or similar circuit, generates a Restart instruction which it jams onto the data bus at T3 of the interrupt service cycle. The MP receives this, and removes from it the three-bit modulo-8 vector, which it then multiplies by 8 to find the vector in low RAM. This vector contains one or more instructions which can service the device causing the interrupt.

5. The execution of the Restart instruction causes the address of the next normal instruction to be executed, obtained from PC, to be placed onto the stack. The next machine cycle will be the M1 of the instruction located in the vector in low RAM. This instruction can now guide the MP to the routine to service the interrupt.
6. At the end of the interrupt service routine, a Return (RTN) instruction will cause the popping of the address off the stack which was of the next instruction to be serviced if the interrupt had not occurred.

The system now finds itself back where it came from.

There are three possible variations to the above scenario. First, unlike the 8080, the 8085 will permit the interrupt as described above as long as no other interrupts are pending which are of greater importance. These, of course, are the 5.5, 6.5, 7.5, and Trap. If any of these are pending, they will be serviced first.

Secondly, while the 8214 was the original device to service interrupts on the 8080 system, the 8085 can work with the 8259A Programmable Interrupt Controller as well. This is a more complex device, programmable as to how it handles interrupts, and stackable to two levels, providing as many as 64 levels of interrupt for the ‘85. The 8259A, moreover, generates Call instructions as well as Restarts. This means that a Call may be jammed onto the data bus during T3 of the interrupt cycle, instead of Restart. While the Restart provides a vector to eight different places in low RAM, depending upon the modulo-8 bits it contains, the Call contains a full two-byte-wide address, which can effectively vector the MP to any-place within the 64K RAM address space. This obviously provides a vastly extended ability to handle interrupts more efficiently.

The third item to be aware of is that the Interrupt Enable flip-flop of the 8080 is now observable as the IE bit #3 of the byte obtained by executing the RIM instruction. It hitherto has not been available, and its status must be remembered by the programmer. Now the bit may be checked with the RIM instruction, to aid in programming.

2.5 MASKABLE INTERRUPTS

Three maskable interrupts are provided in the 8085, each with their own pins. They are named RST 5.5, RST 6.5, and RST 7.5, respectively. To see where these names come from, study this
chart:

NAME: ADDRESS:
RST 0 00H
RST 1 08H
RST 2 10H
RST 3 18H
RST 4 20H
TRAP 24H
RST 5 28H
REST 5.5 2CH
RST 6 30H
RST 6.5 34H
RST 7 38H
RST 7.5 3CH

Note in the chart that the items in light face are those with which we are already familiar. They are the normal vectors for the Restart instructions 0 through 7, as created by the 8214. They are 8 bytes apart, which is ample room for such jumps as are needed to obtain the interrupt servicing routines.

Now look at the bold face items. These items have vector areas which are between the original vectors in RAM. 12he 5.5, for instance, is half way between the RST 5 and the RST 6 vectors, hence the "5.5". If all the vectors were in use, those located above address 20H would each have only four bytes in which to locate and jump to the interrupt service routine. This should be enough room, however, if used wisely. Note also that the Trap interrupt is located at the 4.5 point in the vectors.

The 5.5, 6.5, and 7.5 vectors have several items in common. First, they each have their own pin directly into the 8085. These pins will accept asynchronous interrupt requests without the need for any sort of external priority interrupt device. Secondly, these interrupts are individually
maskable. This is accomplished via the Set Interrupt Mask instruction. This instruction allows bits to be set or cleared which will permit or deny an interrupt on one of these lines to force the ‘85 into an interrupt service cycle. When an input is received on one of these lines and its respective mask bit is enabled (set to 0), the processor will finish the current machine cycle, then enter a interrupt service cycle in which an automatic jam inside the MP will vector it to 2CH, 34H, or 3CH for 5.5, 6.5, or 7.5 respectively. Those locations will presumably have been previously set to contain directions to the interrupt servicing routines.

The RST 5.5 and RST 6.5 interrupts are "level sensitive" This means that the device wishing to interrupt will apply a steady high level to the appropriate pin and hold it there until the 8085 gets around to responding. When the ‘85 recognizes the applied high level, it will permit the interrupt to be serviced in the next machine cycle. The mask bits set by the SIM instruction will directly determine what the RIM instruction sees with respect to the 5.5 and 6.5 interrupt pending bits. If the mask bits are set high (to a 1), these interrupts are masked off. This means that a following RIM will not see them as pending. If the mask bits are set to 0 (enabled), a RIM will see the true condition in bits 4 and 5 of the mask byte.

The RST 7.5 interrupt is "edge sensitive". This means that a pulse applied to this pin, requesting an interrupt, can come and go before the processor gets around to servicing it. This is possible because, unlike the 5.5 and 6.5, the 7.5 has a flip-flop just inside its pin which instantly registers the fact that an interrupt request, albeit short, was applied to the device. This flip-flop provides a bit which is read in RIM instruction as bit 6. This bit will indicate an interrupt pending if a quick pulse is applied to pin 7.5, even though bit 2 of the SIM instruction, the 7.5 mask bit, is turned on (disabled). Bit 2 of SIM byte, therefore, acts differently as a mask bit than does bits 0 and 1 for 5.5 and 6.5. Whereas bits 0 and 1 will mask off all indication of action on pins 5.5 and 6.5, bit 2 will allow the indication of a 7.5 interrupt pending, but will prevent the actual servicing of the 7.5 vector unless the mask is enabled for it. In this way, even though the mask set by the SIM prevents the MP from servicing a 7.5 interrupt, the fact that such an interrupt did occur, captured by the flip-flop, is indicated to whatever routine next executes a RIM instruction. While the normal interrupt and 5.5 and 6.5 interrupts’ enable bits are reset when these are serviced, the 7.5 interrupt flip-flop must be turned off individually. This may be accomplished by
actually responding to the interrupt, just like the other interrupts above; by having the 8085 receiving a /RESET IN, which would also reset the whole system; or by executing a SIM instruction in which bit 4 of the SIM byte is set on. This bit 4 is the "Reset RST 7.5" bit, and will reset the flip-flop if it is on when a SIM is executed.

2.6 NON-MASKABLE INTERRUPT

The Trap instruction is a non-maskable interrupt provision for the 8085. There is no mask bit related to it, and no control bits of any kind. It is used for interrupts of a catastrophic nature, such as the impending doom of a power failure. It is essentially an edge-sensitive input, since its pin connects directly inside the ‘85 to a flip-flop to capture the fact that a request was made. However, the inside circuitry around the flip-flop requires that although the flip-flop is set, the asserted level be continually applied thereafter until the processor enters the service cycle. This is shown in a diagram in the documentation. The Trap, therefore, is called both edge-sensitive and level sensitive as well. The order of priority for all of the interrupts of the 8085, from least important to most important, are the Restart 0 through Restart 7, RST 5.5, RST 6.5, RST 7.5, and finally the Trap. Remember that through the use of the 8214, the RST 0 through 7 interrupts are also prioritized, with 0 as the least important and 7 as the most important. Collectively, the 8085 has a complete set of interrupt capabilities that should serve every need.

2.7 Summary

The processor has 5 interrupts. They are presented below in the order of their priority (from lowest to highest):

**INTR** is maskable 8080A compatible interrupt. When the interrupt occurs the processor fetches from the bus one instruction, usually one of these instructions:

- One of the 8 RST instructions (RST0 - RST7). The processor saves current program counter into stack and branches to memory location N * 8 (where N is a 3-bit number from 0 to 7 supplied with the RST instruction).
CALL instruction (3 byte instruction). The processor calls the subroutine, address of which is specified in the second and third bytes of the instruction.

RST5.5 is a maskable interrupt. When this interrupt is received the processor saves the contents of the PC register into stack and branches to 002Ch (hexadecimal) address.

RST6.5 is a maskable interrupt. When this interrupt is received the processor saves the contents of the PC register into stack and branches to 0034h (hexadecimal) address.

RST7.5 is a maskable interrupt. When this interrupt is received the processor saves the contents of the PC register into stack and branches to 003Ch (hexadecimal) address.

Trap is a non-maskable interrupt. When this interrupt is received the processor saves the contents of the PC register into stack and branches to 0024h (hexadecimal) address.

All maskable interrupts can be enabled or disabled using EI and DI instructions. RST 5.5, RST6.5 and RST7.5 interrupts can be enabled or disabled individually using SIM instruction.

2.8 Keywords

- Hardware Interrupts
- Software Interrupts
- INTR
- INTE
- RAM

2.9 Exercise

1) Explain Interrupts.
2) List the 8085 Interrupt Types.
3) Explain Maskable and Non-maskable Interrupts.

Unit 3
Peripheral mapped I/O

Structure

3.1 Introduction
3.2 Objectives
3.3 Input and Output (I/O) Devices
3.4 Advantages and disadvantages of I/O.
3.5 Distinguish between the memories mapped I/O peripheral I/O?
3.6 Summary
3.7 Keywords
3.8 Exercise

3.1 Introduction
The lower 8 of the 16 address bus memory along with I/O and READ or WRITE control signals allow 512 I/O addresses to be referenced using the IN and OUT instructions. Two lines SID and SOD can be used to generate serial IN and OUT signals under direct software control (so called "bit banging" or "software UART" as the software has to be timed to turn each bit of the serial stream on and off). You may also interface your I/O devices to the address bus by memory mapping them into addresses on the multiplexed address and data bus.

3.2 Objectives
At the end of this chapter you will be able to:

- Explain I/O Devices.
- Know the Advantages and disadvantages of I/O.
- Distinguish between the memories mapped I/O peripheral I/O?

### 3.3 Input and Output (I/O) Devices

Input/output devices are the means through which the MPU communications with “the outside world”. The MPU accepts binary data as input from devices such as keyboards and A/D converters and sends data to output devices such as LEDs or printers.

There are two different methods by which an MPU can identify I/O devices:

- 8 bit address
- 16 bit address

#### 3.3.1 I/Os with 8-bit Addresses (Peripheral-Mapped I/O)

The MPU uses eight address lines to identify an input or an output device. The MPU can identify 256 input and 256 output devices with addresses ranging from $00_{16}$ to $FF_{16}$.

The steps in communicating with I/O device can be summarized as follows:

1. The MPU places an 8 bit address on the address bus, which is decodes by the external decode logic.
2. The MPU sends a control signal (I/O Read or I/O Write) to enable the I/O device.
3. Data are transferred on the data bus.

#### 3.3.2 I/Os with 16 bit Addresses (Memory-Mapped I/O)

The MPU uses 16 address lines to identify an I/O device; an I/O is connected as if it is a memory register. In memory-mapped I/O, the MPU uses the same control signals (Memory Read or Memory Write) and instructions as there of memory and follows the same steps as when it is accessing a memory register.
3.3.3 How does the system work?

In figure above let us assume that the memory address of the first R/W memory location is 8000H and the address of the fan output port is 43H.

The example of the two byte instruction that tells the processor to turn on the fan with the address 43H.

<table>
<thead>
<tr>
<th>Memory</th>
<th>Hex Code</th>
<th>Code explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>8000</td>
<td>D3</td>
<td>Out put (turn on)</td>
</tr>
<tr>
<td>8001</td>
<td>43</td>
<td>The fan</td>
</tr>
</tbody>
</table>

To execute this one instruction, the MPU performs the following operations.

1. MPU must read the binary code of the out instruction stored in location 8000H. To read the code D3, the MPU places the address 8000H on the address bus, enables the memory chip by asserting the Memory Read (MEMRD) signal, and fetches the code D3H using the data bus. This is called the fetch operation.

2. The code D3H in the instruction decoder and it is interpreted as a two-byte instruction. The MPU places the next address 8001H on the address bus, asserts the MEMRD signal, and fetches the address 43H. Now it has read the complete instruction, and is ready to execute the instruction.

3. The MPU places the address of the fan output port (43H) on the address bus, sends a byte to turn on the fan using the data bus, and asserts the IOWR signal to enable the output port.

3.4 Advantages and disadvantages of I/O.

There are three basic forms of input and output that a typical computer system will use: I/O-mapped I/O, memory-mapped input/output, and direct memory access (DMA). I/O-mapped input/output uses special instructions to transfer data between the computer system and the outside world; memory-mapped I/O uses special memory locations in the normal address space of the CPU to communicate with real-world devices; DMA is a special form of memory-mapped I/O where the peripheral device reads and writes memory without going through the CPU. Each
I/O mechanism has its own set of advantages and disadvantages, we will discuss these in this section.

The first thing to learn about the input/output subsystem is that I/O in a typical computer system is radically different than I/O in a typical high level programming language. In a real computer system you will rarely find machine instructions that behave like writeln, printf, or even the x86 Get and Put instructions. In fact, most input/output instructions behave exactly like the x86's mov instruction. To send data to an output device, the CPU simply moves that data to a special memory location (in the I/O address space if I/O-mapped input/output [see "The I/O Subsystem"] or to an address in the memory address space if using memory-mapped I/O). To read data from an input device, the CPU simply moves data from the address (I/O or memory) of that device into the CPU. Other than there are usually more wait states associated with a typical peripheral device than actual memory, the input or output operation looks very similar to a memory read or write operation (see "Memory Access and the System Clock").

An I/O port is a device that looks like a memory cell to the computer but contains connections to the outside world. An I/O port typically uses a latch rather than a flip-flop to implement the memory cell. When the CPU writes to the address associated with the latch, the latch device captures the data and makes it available on a set of wires external to the CPU and memory system:

![Diagram of an I/O port](image)

Fig.3.1

Note that I/O ports can be read-only, write-only, or read/write. The port in the figure above, for example, is a write-only port. Since the outputs on the latch do not loop back to the CPU's data bus, the CPU cannot read the data the latch contains. Both the address decode and write control lines must be active for the latch to operate; when reading from the latch's address the decode
line is active, but the write control line is not.

The figure below shows how to create a read/write input/output port. The data written to the output port loops back to a transparent latch. Whenever the CPU reads the decoded address the read and decode lines are active and this activates the lower latch. This places the data previously written to the output port on the CPU's data bus, allowing the CPU to read that data. A read-only (input) port is simply the lower half of this figure; the system ignores any data written to an input port.

A perfect example of an output port is a parallel printer port. The CPU typically writes an ASCII character to a byte-wide output port that connects to the DB-25F connect on the back of the computer's case. A cable transmits this data to the printer where an input port (to the printer) receives the data. A processor inside the printer typically converts this ASCII character to a sequence of dots it prints on the paper.

Generally, a given peripheral device will use more than a single I/O port. A typical PC parallel printer interface, for example, uses three ports: a read/write port, an input port, and an output port. The read/write port is the data port (it is read/write to allow the CPU to read the last ASCII character it wrote to the printer port). The input port returns control signals from the printer; these signals indicate whether the printer is ready to accept another character, is off-line, is out of paper, etc. The output port transmits control information to the printer such as whether data is available to print.
To the programmer, the difference between I/O-mapped and memory-mapped input/output operations is the instruction to use. For memory-mapped I/O, any instruction that accesses memory can access a memory-mapped I/O port. On the x86, the mov, add, sub, cmp, and, or, and not instructions can read memory; the mov and not instructions can write data to memory. I/O-mapped input/output uses special instructions to access I/O ports. For example, the x86 CPUs use the get and put instructions, the Intel 80x86 family uses the in and out instructions. The 80x86 in and out instructions work just like the mov instruction except they place their address on the I/O address bus rather than the memory address bus (See "The I/O Subsystem"). Memory-mapped I/O subsystems and I/O-mapped subsystems both require the CPU to move data between the peripheral device and main memory. For example, to input a sequence of ten bytes from an input port and store these bytes into memory the CPU must read each value and store it into memory. For very high-speed I/O devices the CPU may be too slow when processing this data a byte at a time. Such devices generally contain an interface to the CPU's bus so it directly read and write memory. This is known as direct memory access since the peripheral device accesses memory directly, without using the CPU as an intermediary. This often allows the I/O operation to proceed in parallel with other CPU operations, thereby increasing the overall speed of the system. Note, however, that the CPU and DMA device cannot both use the address and data busses at the same time. Therefore, concurrent processing only occurs if the CPU has a cache and is executing code and accessing data found in the cache (so the bus is free). Nevertheless, even if the CPU must halt and wait for the DMA operation to complete, the I/O is still much faster since many of the bus operations during I/O or memory-mapped input/output consist of instruction fetches or I/O port accesses which are not present during DMA operations.

### 3.5 Distinguish between the memories mapped I/O peripheral I/O?

<table>
<thead>
<tr>
<th>Memory Mapped I/O</th>
<th>Peripheral Mapped I/O</th>
</tr>
</thead>
<tbody>
<tr>
<td>16-bit device address</td>
<td>8-bit device address</td>
</tr>
<tr>
<td>Data transfer between any general-purpose register and I/O port.</td>
<td>Data is transfer only between accumulator and I/O port</td>
</tr>
<tr>
<td>The memory map (64K) is shared between I/O device and system memory.</td>
<td>The I/O map is independent of the memory map; 256 input device and 256 output device can be connected</td>
</tr>
<tr>
<td>---------------------------------------------------------------</td>
<td>--------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>More hardware is required to decode 16-bit address</td>
<td>Less hardware is required to decode 8-bit address</td>
</tr>
<tr>
<td>Arithmetic or logic operation can be directly performed with I/O data</td>
<td>Arithmetic or logical operation cannot be directly performed with I/O data</td>
</tr>
</tbody>
</table>

### 3.6 Summary

Microprocessors normally use two methods to connect external devices: memory mapped and port mapped I/O. To understand how to emulate microprocessors (for gaming or other purposes) it is important to understand this subtle difference. As far as the peripheral is concerned, both methods are really identical. A device connected to a microprocessor must decode its address from various numbers of address lines and read/write its data from various numbers of data lines. The difference between the two schemes occurs within the microprocessor. Intel has, for the most part, used the port mapped scheme for their microprocessors and Motorola has used the memory mapped scheme. It is certainly possible for a hardware engineer to design a system with memory mapped I/O which uses a CPU supporting port mapped I/O; below you can see why that would not necessarily be a good idea.

### 3.7 Keywords

- I/O
- MPU
- LED
- Peripheral-Mapped I/O
- Memory-Mapped I/O
- DMA
3.8 Exercise

1) Explain I/O Devices.

2) Give the Advantages and disadvantages of I/O.

3) Distinguish between the memories mapped I/O peripheral I/O?

Unit 1
Introduction to 8086

Structure

1.1 Introduction
1.2 Objectives
1.3 8086 Microprocessor
1.4 What is a Microprocessor?
1.5 Why Choose the 8086?
1.6 Summary
1.7 Keywords
1.8 Exercise
1.1 Introduction

The 8086 was the first 16-bit Microprocessor to be introduced by Intel Corporation. It is designed to be upwardly compatible with the older 8080/8085 series of 8-bit microprocessors. The upward compatibility allows programs written for the 8080/8085 to be easily converted to run on the 8086.

The word 16-bit means that its arithmetic logical unit, internal registers, and most of its instructions are designed to work with 16-bit binary words. The 8086 has a 16-bit data bus, so it can read data form or write data to memory and ports either 16-bits or 8-bits at a time. The 8086 has a 20-bit address bus, so it can address any one of 2^20 or 1,048,576 memory locations. Each of the 1,048,576 memory addresses of the 8086 represents a byte-wide location. Words will be stored in two consecutive memory locations. If the first byte of a word is at an even address, the 8086 can read the entire word in one operation. If the first byte of the word is at an odd address, the 8086 will read the first byte of the word in one operation, and the second byte in another operation.

1.2 Objectives

At the end of this chapter you will be able to:

- Give introduction to 8086 Microprocessor
- Know What is a Microprocessor?
- Explain Why Choose the 8086?

1.3 8086 Microprocessor
Computers

Fig. 1.1 shows a block diagram of a simple computer. The major parts are the central processing unit or CPU, memory and the input and output circuitry or I/O and three sets of parallel lines called Buses connecting these parts together. The three buses are called address bus, data bus and control bus.

![Block Diagram of a Simple Computer or a Microcomputer](image)

Memory

The memory section usually consists of RAM and ROM. It may also have magnetic floppy disks, magnetic hard disks or laser option disks. Memory has two purposes. The first purpose is to store the binary codes for the sequence of instructions you want the computer to carryout. The second purpose of the memory is to store the binary-coded data with which the computer is going to be working.

Input/Output

The input/output or I/O section allows the computer to take in data from the outside world or send data to the outside world. Peripherals such as keyboards, video display terminals, printers and modems are connected to the I/O section. These allow the user and the computer to communicate with each other. The actual physical devices used to interface the computer buses to external systems are often called ports.
Address Bus
The address bus consists of 16, 20, 24 or more parallel signal lines. On these lines the CPU sends out the address of the memory location that is to be written to or read from. The number of memory locations that the CPU can address is determined by the number of address lines. If the CPU has \( N_{\text{address}} \) address lines then it can directly address \( 2^N \) memory locations. For example, a CPU with 16 address lines can address 216 or 65,536 memory locations.

Data Bus
The data bus consists of 8, 16, 32 or more parallel signal lines. The data bus lines are bi-directional.

Control Bus
The control bus consists of 4 - 10 parallel signal lines. The CPU sends out signals on the control bus to enable the outputs of addressed memory devices or port devices. Typical control bus signals are memory/read, memory write, I/O read, and I/O write. To read a byte of data from a memory location, for example, the CPU sends out the address of the desired byte on the address bus and then sends out a memory read signal on the control bus. The memory read signal enables the addressed memory device to output the byte of data on the data bus where it is read by the CPU.

Hardware, Software, and Firmware
When working around computers you hear the terms hardware, software and firmware. Hardware is the name given to the physical devices and circuitry of the computer. Software refers to the programs written for the computer. Firmware is the term given to the programs stored in ROMs or in other devices which keep their stored information when the power is turned off.

1.4 What is a Microprocessor?
The entire CPU with timing and control functions on a single chip is known as Microprocessor. Therefore a Microprocessor or MPU is an integrated circuit that contains many processing capabilities of a large computer.
**Microprocessor Evolution**

A common way of categorizing is by the number of bits that their ALU can work with at a time. A Microprocessor with a 4-bit ALU will be referred to as a 4-bit Microprocessor, regardless of the number of address lines or the number of data bus lines that it has. The first microprocessor was the Intel 4004 produced in 1971. This 4004 was a 4-bit device intended to be used with some other devices in making a calculator. Some logic designers, however, saw that this device could be used to replace PC boards full of combinational and sequential logic devices. Also, the ability to change the function of a system by just changing the programming, rather than redesigning the hardware, is very appealing. It was these factors that pushed the evolution of microprocessors. In 1972 Intel came out with the 8008 which was capable of working with 8-bit words. In 1974 Intel announced the 8080 which had a much larger instruction set than 8008. The 8080 is referred to as a second-generation microprocessor. Soon after Intel produced 8080, Motorola came out with MC 6800, another 8-bit general purpose CPU. Some of the other competitors were the MOS technology 6502 and the Zilog Z80. The 16-bit microprocessors entered the marketplace in the late 1970s and early 1980s. Then came the 32-bit processors. Most Widely, Microprocessors are divided into two groups based on their origin. These groups may be tabled as the 6’s group and that of the 8’s. A family tree of the 6’s group and that of the 8’s group is shown in figure 2.
Fig. 1.2 Genealogy for 6’s group and 8’s group of microprocessors

We observe that as we progress upward on the family tree the trend is towards greater complexity. Complexity is noted in the figure, in terms of the bit size of the internal registers. The 6’s group traces its origin back to the original 6800 Microprocessor designed by Motorola. The 8’s group traces its origin back to Intel’s 8080 Microprocessor. Each branch in Fig. 2 is labeled near the top with the manufacturer responsible for its development.

1.5 Why Choose the 8086?

- **Firstly**, the answer might be -- well I'm already familiar with the 8085 and Z80 8bit microprocessors and I need to learn how to use a 16bit system quickly!
- **Secondly**, there is a wealth of support chips designed to interlace with this processor and many of them cost less than £5.00
• **Thirdly**, the PC market which uses Intel based devices takes up some 60% of the total microprocessor market! The other main processor used by industry is the Motorola 68000 family of microprocessors.

• **Fourthly**, and perhaps most importantly, there is an excellent all round educational package available - namely the Flite 86 from Flite Electronics International Limited.

The major problem facing educators and students is finding a complete (student centered) 16bit microprocessor learning package. When upgrading to a new system it is necessary to consider not just the hardware and software cost but staff expertise and in service training. Ideally, educational establishments require packages that are versatile. Often it is necessary to justify your choice by indicating what range of courses can make use of the chosen system. If, for example the chosen system can be used over a wide range of microprocessors, then the capital expenditure can be more easily justified.

The Flite 86 system is a complete learning package. The 8086 Controller Board is designed to simplify the teaching of the 8086 CPU and some of its commonly used peripherals. It can be linked to most PC's with a simple serial line, so that code may be assembled and debugged in a supportive software environment before being down-loaded into the RAM on the board. The board itself may then be linked to other peripherals.

Once downloaded the code may be executed and examined in a system which is accessible to the user. Data may be manipulated on the board and the effects viewed on the PC. The software which handles this two way transfer is supplied with the board, in the form of a monitor program resident on the board in EPROM, and a disk containing the 'host' software for the PC.

Apart from its use, linked to a PC, the board may also be used independently, under the control of the user, either for fault finding on 8086 systems, or for control projects. For fault finding exercises a number of test routines are supplied with the monitor EPROM which enables many faults to be investigated using simple 'scope techniques'. The accessibility of components on the board means that the faults may be easily applied.

In control applications, the board is ideal for projects from the simple 'flashing LED' variety to sophisticated, real time systems such as floppy disc controllers. The control program can be
blown into the board's EPROM's either in place of, or in addition to, the monitor program already present. The board then becomes a powerful stand alone control system. The development and testing of the software is helped enormously by using the system linked to a PC initially, and downloading development code into the RAM.

Most importantly it comes with a comprehensively written (and easy to read) Technical Reference Manual and a set of well documented and technically accurate experiment books taking one through from simple ROM/RAM testing to multiple interrupt routines, using Flite's experiment board.

1.6 Summary

The 8086 is a true 16bit machine which means that the ALU (Arithmetic Logic Unit) is designed to work with 16bit numbers and the data bus is also 16bits wide, because the 8088 is hybrid in that the ALU is 16bits but the data bus is 8bits. In almost every other respect the two processors are identical. It is possible to address 1Mbyte of memory with the 20bit address bus. Both devices have time multiplexed address and data buses and some of the control pins have more than one function depending upon whether the device is operated in the min or max mode. The min mode is designed for small single processor systems whilst in the max mode the device is designed to work in medium or large systems using more than one processor.

1.7 Keywords

- Microprocessor
- Buses
- RAM
- ROM
- CPU
- Control Bus
- Firmware
1.8 Exercise

1) Give introduction to 8086 Microprocessor.
2) What is a Microprocessor?
3) Why Choose the 8086?

Unit 2

Architecture of 8086

Structure

2.1 Introduction
2.2 Objectives
2.3 Architecture of 8086
2.4 Bus Interface Unit (BIU)
2.5 General Purpose Registers
2.6 Stack Pointer Register
2.7 Other pointer and Index Registers
2.8 Summary
2.9 Keywords
2.10 Exercise

2.1 Introduction

8086 has 16-bit ALU; this means 16-bit numbers are directly processed by 8086. It has 16-bit data bus, so it can read data or write data to memory or I/O ports either 16 bits or 8 bits at a time. It has 20 address lines, so it can address up to 220 i.e. 1048576 = 1Mbytes of memory (words
i.e. 16 bit numbers are stored in consecutive memory locations). Due to the 1Mbytes memory size multiprogramming is made feasible as well as several multiprogramming features have been incorporated in 8086 design.

2.2 Objectives
At the end of this chapter you will be able to:

- Explain the Architecture of 8086
- Define BIU
- Explain Stack Pointer Register

2.3 Architecture of 8086
The term architecture, as used in microprocessor circuits, describes the functional components that make up the MPU and the interaction between them. These include the temporary storage devices known as registers, which are used to hold data, instructions, and status information. There are also devices to perform arithmetic and logical operations. Control devices are used to control the flow of information through the MPU.
As shown by the block diagram in fig.2.1, the 8086 MPU is divided into two independent functional parts known as the execution unit (EU) and the bus interface unit (BIU). **8 MEMORY INTERFACE**

Execution unit (EU) The EU is where the actual processing of data takes place inside the 8086 MPU. It is here that the arithmetic and logic unit (ALU) is located, along with the registers used to manipulate data and store immediate results. The EU accepts instructions and data that have been fetched by the BIU and then processes the information. Data processed by the EU can be transmitted to the memory or peripheral devices through the BIU. EU has no direct connection with the outside world and relies solely on the BIU to feed it with instructions and data as indicated in fig.2.2
2.4 Bus Interface Unit (BIU)

The BIU is made up of the address generation and bus-control unit, the instruction queue, and the instruction pointer. It has the task of making sure that the bus is used to its fullest capacity in order to speed-up operations. This function is carried in two ways. First, by fetching the instructions before they are needed by the execution unit and storing them in the instruction queue, the 8086 MPU is able to increase computing speed. Second, by taking care of all bus-control functions, the EU is free to concentrate on processing data and carrying out the instructions. The instruction pointer contains the location or address of the next instruction to be executed. Inside the EU the EU is made up of two parts known as the ALU and the general registers. It is here that instructions are received, decoded, and executed from the instruction queue portion of BIU. The instructions are taken from the top of the instruction queue on the first-in, first-out, or FIFO, basis. ALU the ALU is the calculator part of the execution unit. It consists of electronic circuitry that performs arithmetic operations or logical operations on the binary represented electrical signals. The control system for the execution unit can also be thought of as part of ALU. It provides a path for the flow of instructions into the ALU, the general registers, and the flag register. Flag Register A flag is a flip-flop which indicates some condition produced by the execution of an instruction or controls certain operations of the EU. The Flag Register is a special register associated with the ALU. A 16-bit flag register in the EU contains nine active flags. Fig.2.3 shows the location of the nine flags in the flag register.
Six flags are status flags - AF, CF, OF, SF, PF and ZF. The remaining three flags are control flags - DF, IF, and TF. Table 1 presents a flag summary and highlights key concerns. Each flag is next discussed in detail.

### Table 1: Flag summary

<table>
<thead>
<tr>
<th>Status Flags</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AF (auxiliary flag)</td>
<td>Indicates if the instruction generated a carry out the 4 LSBS.</td>
</tr>
<tr>
<td>CF (carry flag)</td>
<td>Indicates if the instruction generated a carry out the MSB.</td>
</tr>
<tr>
<td>MSB. OF (overflow flag)</td>
<td>Indicates if the instruction generated a signed result that is out of range.</td>
</tr>
<tr>
<td>SF (sign flag)</td>
<td>Indicates if the instruction generated a negative result.</td>
</tr>
<tr>
<td>PF (parity flag)</td>
<td>Indicates if the instruction generated a result having an even number of 1s.</td>
</tr>
<tr>
<td>ZF (zero flag)</td>
<td>Indicates if the instruction generated a zero result.</td>
</tr>
<tr>
<td>DF (direction flag)</td>
<td>Controls the direction of the string manipulation instructions.</td>
</tr>
<tr>
<td>IF (interrupt-enable flag)</td>
<td>Enables or disables external interrupts.</td>
</tr>
<tr>
<td>TF (trap flag)</td>
<td>Puts the processor into a single-step mode for program debugging.</td>
</tr>
</tbody>
</table>

AF (auxiliary flag). If this flag is set, there has been a carry out or borrow of the 4 least significant bits. This flag is used during decimal arithmetic instructions.

CF (carry flag). If this flag is set, there has been a carry out or overflow of the most significant bit. It is used by instructions that add and subtract multi byte numbers.

OF (overflow flag). If this flag is set, an arithmetic overflow has occurred; that is, a significant
digit has been lost because the size of the result exceeded the capacity of its destination location.
SF (sign flag). Since negative binary numbers are represented in the 8086/8088 in standard 2s complement notation. SF indicates the sign of the result (0 = positive, 1 = negative).
PF (party flag). If this flag is set, the result has even parity, an even number of 1s. This flag can be used to check for transmission errors.
ZF (zero flag). If this flag is set, the result of the operation is 0.
DF(direction flag). Setting DF causes string instructions to auto-decrement (count down); that is, to process strings from the high address to the low address, or from right to left. Clearing DF causes string instructions to auto-increment (count up), or process strings from left to right.
IF (interrupt-enable flag) Setting IF allows the MPU to recognize external (maskable) interrupt requests. Clearing IF disables these interrupts. IF has no effect on either nonmaskable external or internally generated interrupts. 11
TF (trap flag) . Setting TF puts the processor into single-step mode for debugging. In this mode the MPU automatically generates an internal interrupt after each instruction, allowing a program to be inspected as it executes instruction by instruction.

2.5 General Purpose Registers
EU has eight general purpose registers labeled AH, AL, BH, BL, CH, CL, DH and DL. These registers are a set of data registers, which are used to hold intermediate results. The H represents the high-order or most-significant byte and the L represents the low-order or least-significant byte. Each of these registers may be used separately as 8-bit storage areas or combined to form one 16-bit (one word) storage area. The acceptable register pairs are AH and AL, BH and BL, CH and CL and DH and DL. The AH-AL pair is referred to as the AX register, the BH-BL pair is referred to as the BX register, the CH-CL pair is referred to as the CX register, and the BH-BL pair is referred to as the DX register. The AL register is also called as the Accumulator. For 16-bit operations, AX is called the accumulator. The 8086 register set is very similar to those of earlier generation 8080 and 8085 microprocessors. Many programs written for the 8080 and 8085 could easily be translated to run on the 8086.

2.6 Stack Pointer Register
A Stack is a section of memory set aside to store addresses and data while a subprogram is being executed. An entire 64 K bytes segment is set aside as stack in 8086 MPU. The upper 16 bits of
the starting address for this segment is kept in the stack segment register. The Stack Pointer (SP) register contain the 16-bit offset from the start of the segment to the memory location where a word was most recently stored on the Stack. The memory location where a word was most recently stored is called the top of Stack. Fig.2.4 shows the details.

![Diagram](image)

**Fig.2.4: Addition of SS and SP to produce physical address of top of stack (a) diagram b) computation**

The physical address for a stack reader for a stack write is produced by adding the contents of the stack pointer register to the segment base address in SS. To do this the contents of the Stack segment register are shifted four bit positions left and the contents of SP are added to the shifted result. In the figure 5000 H in SS is shifted left four bit positions to give 50000H. When FFE0H in the SP is added to this, the resultant physical address for the top of the stack will be 5FFEOH. The physical address can be represented either as a single number 5FFEOH, or it can be represented in SS:SP form as 5000:FFEOH.

### 2.7 Other pointer and Index Registers

In addition to the Stack Pointer register, SP, the EU contains a 16-bit base pointer (BP) register. It also contains a 16-bit Source index (SI) register and a 16-bit destination index (DI) register. These three registers can be used for temporary storage of data just as the general purpose registers. However, their main use is to hold the 16-bit offset of a data word in one of the segments. That is, the pointer and index registers are usually used to point to or index to an
address in memory. When used in this manner, these registers are address registers that designate a specific location in the memory that may be frequently used by the program. The addresses contained in these registers can be combined with information from the BIU to physically locate the data in the memory.

**The Bus Interface Unit**

The BIU sends out addresses, fetches instructions from memory, reads data from ports and memory. In other words, the BIU handles all transfers of data and addresses on the buses for the execution unit. The BIU can be thought of as three functional blocks; Bus control, Instruction queue, and Address control.

**Bus control**

The bus-control unit performs the bus operations for the MPU. It fetches and transmits instructions, data, and control signals between MPU and the other devices of the system.

**Instruction Queue**

The instruction queue is used as a temporary memory storage area for data instructions that are to be executed by the MPU. The BIU, through the bus-control unit, perfectness instructions and stores them in the instruction queue. This allows the execution unit to perform its calculations at maximum efficiency. Because the BIU and EU essentially operate independently, the BIU concentrates on loading instructions into the instruction queue. This usually takes more time to do than the calculations performed by the execution unit. In effect, the BIU and the EU work in parallel. The instruction queue is a first-in, first-out (FIFO) memory. This means that the first instruction loaded into the instruction queue by the bus control unit will be the first instruction to be used on the ALU.

**Address control**

The address-control unit is used to generate the 20-bit memory address that gives the physical or actual location of the data or instruction in memory. This unit consists of the instruction pointer, the segment registers, and the address generator as shown in fig 2.5
Instruction Pointer

The Instruction Pointer (IP) is a 16-bit register that is used to point to, or tell the MPU, the instruction to execute next. Therefore, the instruction pointer is used to control the sequence in which the program is executed. Each time the execution unit accepts an instruction, the instruction pointer, is incremented to point to the next instruction in the program.

Segment Registers There are four segment registers. They are the code segment (CS), the data segment (DS), the stack segment (SS), and the extra segment (ES). These registers are used to define a logical memory space or memory segment that is set aside for a particular function. The CS register points to the current code segment. Instructions are fetched from this segment. The DS register points to the current data segment. Program variables and data are held in this area. The SS register points to the current stack segment, stack operations are performed on locations in the SS segment. The ES register points to the current extra segment, which is also used for data storage. Each of the segment registers can be up to 64 kilo bytes long. Each segment is made up of an uninterrupted section of memory locations. Each segment can be addressed separately.
using the base address that is contained in its segment register. The base address is the starting address for that segment.

**Address Generator**

The address-generator unit is used with the segment registers to generate the 20-bit physical address required to identify all the possible memory addresses. The 20 address lines give a maximum physical memory size of 20 address locations, or 1,048,576 bytes of memory. But all the registers in the MPU are only 16 bits wide. The physical address is obtained by shifting the segment base value four bit positions (one hexadecimal position) and adding the offset or logical address of the segment.

**2.8 Summary**

The microprocessors function as the CPU in the stored program model of the digital computer. Its job is to generate all system timing signals and synchronize the transfer of data between memory, I/O, and itself. It accomplishes this task via the three-bus system architecture previously discussed.

The microprocessor also has a S/W function. It must recognize, decode, and execute program instructions fetched from the memory unit. This requires an Arithmetic-Logic Unit (ALU) within the CPU to perform arithmetic and logical (AND, OR, NOT, compare, etc) functions.

The 8086 CPU is organized as two separate processors, called the Bus Interface Unit (BIU) and the Execution Unit (EU). The BIU provides H/W functions, including generation of the memory and I/O addresses for the transfer of data between the outside world -outside the CPU, that is-and the EU.

The EU receives program instruction codes and data from the BIU, executes these instructions, and stores the results in the general registers. By passing the data back to the BIU, data can also be stored in a memory location or written to an output device. Note that the EU has no connection to the system buses. It receives and outputs all its data thru the BIU.
2.9 Keywords

- BIU
- ALU
- MPU
- EU
- SF
- PF
- MSB
- DF
- IF
- TF

2.10 Exercise

1) Explain the Architecture of 8086.
2) Define BIU.
3) Explain Stack Pointer Register.

Unit 3

Block and Pin Diagram of 8086 and it’s functions

Structure

3.1 Introduction
3.2 Objectives
3.3 Pin description of 8086
3.4 Fetch and Execute
3.5 Programming Model
3.1 **Introduction**

The 8086 had eight 16-bit **registers** including the **stack pointer**, but excluding the instruction pointer, flag register and segment registers. Four of them, AX, BX, CX, DX, could also be accessed as twice as many 8-bit registers while the other four, BP, SI, DI, SP, were 16-bit only.

3.2 **Objectives**

At the end of this chapter you will be able to:

- Draw the block diagram and Pin description of 8086
- Explain Fetch and Execute
- Know Programming Model
- Define Segment Memory, Memory Map
- Know the 8086 function calls

3.3 **Pin description of 8086**

The Microprocessor 8086 is a 16-bit CPU available in different clock rates and packaged in a 40 pin CERDIP or plastic package. The 8086 operates in single processor or multiprocessor configuration to achieve high performance. The pins serve a particular function in minimum
mode (single processor mode) and other function in maximum mode configuration (multiprocessor mode).

The 8086 signals can be categorised in three groups.

- The first are the signal having common functions in minimum as well as maximum mode.
- The second are the signals which have special functions for minimum mode.
- The third are the signals having special functions for maximum mode.

The following signal descriptions are common for both modes.

- **AD15-AD0**: These are the time multiplexed memory I/O address and data lines.
  - Address remains on the lines during T1 state, while the data is available on the data bus during T2, T3, Tw and T4. These lines are active high and float to a tristate during interrupt acknowledge and local bus hold acknowledge cycles.

- **A19/S6,A18/S5,A17/S4,A16/S3**: These are the time multiplexed address and status lines.
  - During T1 these are the most significant address lines for memory operations.
  - During I/O operations, these lines are low.
  - During memory or I/O operations, status information is available on those lines for T2, T3, Tw and T4.
  - The status of the interrupt enable flag bit is updated at the beginning of each clock cycle.
  - The S4 and S3 combine to indicate which segment register is presently being used for memory accesses as in below fig.
  - These lines float to tri-state off during the local bus hold acknowledge. The status line S6 is always low.
  - The address bit are separated from the status bit using latches controlled by the ALE signal.

<table>
<thead>
<tr>
<th>S4</th>
<th>S3</th>
<th>Indication</th>
</tr>
</thead>
</table>

BHE/S7 : The bus high enable is used to indicate the transfer of data over the higher order (D15-D8) data bus as shown in table. It goes low for the data transfer over D15-D8 and is used to derive chip selects of odd address memory bank or peripherals. BHE is low during T1 for read, write and interrupt acknowledge cycles, whenever a byte is to be transferred on higher byte of data bus. The status information is available during T2, T3 and T4. The signal is active low and tristated during hold. It is low during T1 for the first pulse of the interrupt acknowledge cycle.

RD – Read : This signal on low indicates the peripheral that the processor is performing memory or I/O read operation. RD is active low and shows the state for T2, T3, Tw of any read cycle. The signal remains tristated during the hold acknowledge.

READY : This is the acknowledgement from the slow device or memory that they have completed the data transfer. The signal made available by the devices is synchronized by the 8284A clock generator to provide ready input to the 8086. the signal is active high.

INTR-Interrupt Request : This is a triggered input. This is sampled during the last clock cycles of each instruction to determine the availability of the request. If any interrupt request is pending, the processor enters the interrupt acknowledge cycle. This
can be internally masked by resulting the interrupt enable flag. This signal is active high and internally synchronized.

- **TEST**: This input is examined by a ‘WAIT’ instruction. If the TEST pin goes low, execution will continue, else the processor remains in an idle state. The input is synchronized internally during each clock cycle on leading edge of clock.

- **CLK- Clock Input**: The clock input provides the basic timing for processor operation and bus control activity. It’s an asymmetric square wave with 33% duty cycle.

Fig 8 shows the 8086 pin diagram. Vcc is on pin 40 and ground on pins 1 and 20. 8086 requires +5v supply. Clock input labeled CLK is on pin 19. An 8086 requires a clock signal from some external, crystal-controlled clock generator to synchronize internal operations in the processor. Different versions of the 8086 have maximum clock frequencies ranging from MHz to 10 MHz.
Fig. 3.1: Pin diagram

Pins 2 through 16 and pins 35 through 39 are used for the address bus. Pins 35 through 38 are used by multiplexing to provide information or status about the MPU. The status signals are labeled $S_3$, $S_4$, $S_5$ and $S_6$ as shown. The data bus lines $AD_0$ through $AD_{15}$ are used at the start of the machine cycle to send out addresses, and later in the machine cycle they are used to send or receive data. The 8086 sends out a signal called address latch enable or ALE on pin 25 to let external circuitry know that an address is on the data bus. The upper 4 bits of an address are sent on the lines labeled $A_{16}/S_3$ through $A_{19}/S_6$.

Some of the control bus lines on a microprocessor usually have mnemonics such as RD, WR and M/IO. Pin 32 of the 8086 is labeled RD. A tri-state active-low output signal on pin 32 indicates that the 8086 is reading data from memory or from a port. Pin 29 has a label WR next to it. However, pin 29 also has a label LOCK next to it, because this pin has two functions. The
function of this pin and the functions of the pins between 24 and 31 depend on the mode in which the 8086 is operating. The operating mode of the 8086 is determined by the logic level applied to the MN / MX input on pin 33.

If pin 33 is asserted high, then the 8086 will function in minimum mode, and pins 24 through 31 will have functions shown in parentheses next to the pins in fig. 8. If the MN / MX pin is asserted low, then the 8086 is in maximum mode. In this mode pins 24 through 31 will have the functions described by the mnemonics next to the pins in fig. 8. A tri-state active-low output signal on pin 29 indicates that MPU has put valid and stable data on the data bus. Pin 28 will function as M / IO. The 8086 will assert this signal high if it is reading from or writing to a memory location, and it will assert a signal low if it is reading from or writing to a port. In the maximum mode the control bus signals \( S_0, S_1, S_2 \) are sent out in encoded form on pins 26, 27 and 28. An external bus controller device decodes these signals to produce the control bus signals required for a system, which has two or more microprocessors sharing the same buses. If pin 21, the RESET input is made high, the 8086 will, no matter what it is doing, reset its DS, SS, ES, IP and flag registers to all 0's. It will set its CS register to FF. When the RESET signal is removed from pin 21, the 8086 will then fetch its next instruction from physical address \( \text{FFFF0H} \). This address is produced in the 8086 Bus Interface unit (BIU) by shifting the \text{FFFFH} in the CS register 4 bits left by adding the \text{0000H} in the instruction pointer to it. The first instruction that has to be executed after a reset is put at this address \( \text{FFF0H} \).

8086 has two interrupt inputs, non-maskable interrupt (NMI) input on pin 17 and the interrupt (INTR) input on pin 18. An active-high on any one of these pins will cause the 8086 to stop execution of its current program and go execute a specified procedure. At the end of the procedure it can return to executing the interrupted program. The NMI cannot be ignored, or masked, by the MPU. The INTR (interrupt request) is maskable and can be made to be ignored by the MPU through software control.

A tri-state active-low output signal on pin 26 DEN (data enable) determines whether the data buffer is enabled or disabled. A tri-state output signal on pin 27 DT / R (data transmit receive) is used to control the direction of data flow. A logic level 1 indicates data bits are being transmitted from the MPU. A logic level 0 indicates that data bits are being received into the MPU. All microprocessors use an oscillator to generate a master frequency clock to synchronize or time operations. For the 8086 microprocessor the oscillator frequency, or clock frequency is
typically 5 MHz. The period of one clock cycle is then equal to. 
\[ T = \frac{1}{F} = \frac{1}{5 \times 10^6} \text{ Hz} \]

\[ = 0.2 \times 10^{-6} \text{ sec.} \]

\[ = 200 \text{ n sec} \]

The 8086 operates in time periods called bus cycles. Each bus cycle requires 4 clock cycles to complete. Therefore, the bus cycle is completed very 800 ns. A typical bus cycle is shown in fig 3.2.

One cycle of this is referred to as a state. A state is measured from the 50 percent point on the falling edge of one clock pulse to 50 percent point on the falling edge of the next clock pulse - T1 in the figure is a state. Each basic bus operation such as reading a byte from memory or writing a word to a port requires some number of states. The group of states required for a basic bus operation is called a machine cycle. The total time it takes the 8086 to fetch and execute an instruction is called an instruction cycle. An instruction cycle consists of one or more machine cycles. To summarize, an instruction cycle is made up of machine cycles, and a machine cycle is made up of states. Two major bus cycles are the read bus cycle and the write bus cycle. The read bus cycle is activated when the microprocessor is reading information from the memory or an I/O device. During the read bus cycle, there are normally four clock cycles T1, T2, T3, and T4. However, if the device outputting data to the MPU needs more time to send the data, a wait state (Tw) is initiated by placing extra clock cycles (Tw's) between cycles T3 and T4.

Fig 3.2 : Timing bus cycle
### 3.4 Fetch and Execute

Although the 8086/88 still functions as a stored program computer, organization of the CPU into a separate BIU and EU allows the fetch and execute cycles to overlap. To see this, consider what happens when the 8086 or 8088 is first started.

1. The BIU outputs the contents of the instruction pointer register (IP) onto the address bus, causing the selected byte or word to be read into the BIU.

2. Register IP is incremented by 1 to prepare for the next instruction fetch.

3. Once inside the BIU, the instruction is passed to the queue. This is a first-in, first-out storage register sometimes likened to a "pipeline".

4. Assuming that the queue is initially empty, the EU immediately draws this instruction from the queue and begins execution.

5. While the EU is executing this instruction, the BIU proceeds to fetch a new instruction. Depending on the execution time of the first instruction, the BIU may fill the queue with several new instructions before the EU is ready to draw its next instruction.

---

**Fig. 3.3**

<table>
<thead>
<tr>
<th>BIU</th>
<th>EU</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fetch</td>
<td>Wait</td>
</tr>
<tr>
<td>Fetch</td>
<td>Execute</td>
</tr>
<tr>
<td>Fetch</td>
<td>Execute</td>
</tr>
<tr>
<td>Fetch</td>
<td>Execute</td>
</tr>
<tr>
<td>Fetch</td>
<td>Execute</td>
</tr>
<tr>
<td>Fetch</td>
<td>Execute</td>
</tr>
<tr>
<td>Fetch</td>
<td>Execute</td>
</tr>
<tr>
<td>Fetch</td>
<td>Execute</td>
</tr>
<tr>
<td>Fetch</td>
<td>Execute</td>
</tr>
<tr>
<td>Fetch</td>
<td>Execute</td>
</tr>
<tr>
<td>Fetch</td>
<td>Execute</td>
</tr>
<tr>
<td>Fetch</td>
<td>Execute</td>
</tr>
</tbody>
</table>

*This instruction requires a request for data not in the queue.

1Jump instruction occurs.

2These bytes are discarded.

**Instruction Pipeline**

Fig.3.3
The BIU is programmed to fetch a new instruction whenever the queue has room for one (with the 8088) or two (with the 8086) additional bytes. The advantage of this pipelined architecture is that the EU can execute instructions almost continually instead of having to wait for the BIU to fetch a new instruction.

There are three conditions that will cause the EU to enter a "wait" mode. The first occurs when an instruction requires access to a memory location not in the queue. The BIU must suspend fetching instructions and output the address of this memory location. After waiting for the memory access, the EU can resume executing instruction codes from the queue (and the BIU can resume filling the queue).

The second condition occurs when the instruction to be executed is a "jump" instruction. In this case control is to be transferred to a new (nonsequential) address. The queue, however, assumes that instructions will always be executed in sequence and thus will be holding the "wrong" instruction codes. The EU must wait while the instruction at the jump address is fetched. Note that any bytes presently in the queue must be discarded (they are overwritten).

One other condition can cause the BIU to suspend fetching instructions. This occurs during execution of instructions that are slow to execute. For example, the instruction AAM (ASCII Adjust for Multiplication) requires 83 clock cycles to complete. At four cycles per instruction fetch, the queue will be completely filled during the execution of this single instruction. The BIU will thus have to wait for the EU to pull over one or two bytes from the queue before resuming the fetch cycle.

A subtle advantage to the pipelined architecture should be mentioned. Because the next several instructions are usually in the queue, the BIU can access memory at a somewhat "leisurely" pace. This means that slow-mem parts can be used without affecting overall system performance.

3.5 Programming Model

As a programmer of the 8086 or 8088 you must become familiar with the various registers in the EU and BIU.
The data group consists of the accumulator and the BX, CX, and DX registers. Note that each can be accessed as a byte or a word. Thus BX refers to the 16-bit base register but BH refers only to the higher 8 bits of this register. The data registers are normally used for storing temporary results that will be acted on by subsequent instructions.

The pointer and index group are all 16-bit registers (you cannot access the low or high bytes alone). These registers are used as memory pointers. Sometimes a pointer reg will be interpreted as pointing to a memory byte and at other times a memory word. As you will see, the 8086/88 always stores words with the high-order byte in the high-order word address.

Register IP could be considered in the previous group, but this register has only one function - to point to the next instruction to be fetched to the BIU. Register IP is physically part of the BIU and not under direct control of the programmer as are the other pointer registers.

Six of the flags are status indicators, reflecting properties of the result of the last arithmetic or logical instructions. The 8086/88 has several instructions that can be used to transfer program control to a new memory location based on the state of the flags.

Three of the flags can be set or reset directly by the programmer and are used to control the operation of the processor. These are TF, IF, and DF.
The final group of registers is called the segment group. These registers are used by the BIU to determine the memory address output by the CPU when it is reading or writing from the memory unit. To fully understand these registers, we must first study the way the 8086/88 divides its memory into segments.

### 3.6 Segment Memory

Even though the 8086 is considered a 16-bit processor, (it has a 16-bit data bus width) its memory is still thought of in bytes. At first this might seem a disadvantage:

Why saddle a 16-bit microprocessor with an 8-bit memory?

Actually, there are a couple of good reasons. First, it allows the processor to work on bytes as well as words. This is especially important with I/O devices such as printers, terminals, and modems, all of which are designed to transfer ASCII-encoded (7- or 8-bit) data.

Second, many of the 8086's (and 8088's) operation codes are single bytes. Other instructions may require anywhere from two to seven bytes. By being able to access individual bytes, these odd-length instructions can be handled.

We have already seen that the 8086/88 has a 20-bit address bus, allowing it to output $2^{10}$, or 1,048,576, different memory addresses. As you can see, 524,288 words can also be visualized.

As mentioned, the 8086 reads 16 bits from memory by simultaneously reading an odd-addressed byte and an even-addressed byte. For this reason the 8086 organizes its memory into an even-addressed bank and an odd-addressed bank.

With regard to this, you might wonder if all words must begin at an even address. Well, the answer is yes. However, there is a penalty to be paid. The CPU must perform two memory read cycles: one to fetch the low-order byte and a second to fetch the high-order byte. This slows down the processor but is transparent to the programmer.

The last few paragraphs apply only to the 8086. The 8088 with its 8-bit data bus interfaces to the 1 MB of memory as a single bank. When it is necessary to access a word (whether on an even- or
an odd-addressed boundary) two memory read (or write) cycles are performed. In effect, the 8088 pays a performance penalty with every word access. Fortunately for the programmer, except for the slightly slower performance of the 8088, there is no difference between the two processors.

3.7 Memory Map

Still another view of the 8086/88 memory space could be as 16 64K-byte blocks beginning at hex address 000000h and ending at address 0FFFFFFh. This division into 64K-byte blocks is an arbitrary but convenient choice. This is because the most significant hex digit increments by 1 with each additional block. That is, address 20000h is 65.536 bytes higher in memory than address 10000h. Be sure to note that five hex digits are required to represent a memory address.

![Memory Map Diagram](image)

The diagram is called a memory map. This is because, like a road map, it is a guide showing how the system memory is allocated. This type of information is vital to the programmer, who must know exactly where his or her programs can be safely loaded.

Note that some memory locations are marked reserved and others dedicated. The dedicated locations are used for processing specific system interrupts and the reset function. Intel has also reserved several locations for future H/W and S/W products. If you make use of these memory locations, you risk incompatibility with these future products.
3.8 8086 function calls

Control flow for 8086 function calls involves two aspects.

- The CALL instruction is similar to jal in MIPS, but the return address is placed on the stack instead of in a register.
- RET pops the return address on the stack and jumps to it.

The flow of data in 8086 function calls faces similar issues as MIPS. Arguments and return values can be passed either in registers or on the stack. Functions are expected to preserve the original values of any registers they modify—in other words, all registers are callee-saved.

The 8086 also relies upon a stack for local storage. The stack can be manipulated explicitly, via the esp register. The CPU also includes special PUSH and POP instructions, which can manage the stack pointer automatically.

3.9 Summary

8086 is a 16bit processor. It’s ALU, internal registers works with 16bit binary word. 8086 has a 16bit data bus. It can read or write data to a memory/port either 16bits or 8 bit at a time. 8086 has a 20bit address bus which means, it can address up to $2^{20} = 1$MB memory location. Frequency range of 8086 is 6-10 MHz. Additional registers called segment registers generate memory address when combined with other in the microprocessor.

3.10 Keywords

- READY
- Clock Input
- MPU
- bus cycle
- BIU
- Memory Map

3.11 Exercise

1) Draw the block diagram and Pin description of 8086.
2) Explain Fetch and Execute.
3) Explain Programming Model.
4) Define Segment Memory, Memory Map.
5) Explain the 8086 function calls.

## Unit 4
### Bus Details

**Structure**

4.1 Introduction
4.2 Objectives

4.3 Segment Registers

4.4 Logical and physical address

4.5 Advantages of segmented memory

4.6 Address Bus
4.7 Data Bus
4.8 Control Bus

4.9 Bus Request and Bus Grant Timings in Minimum Mode System of 8086

4.10 Summary
4.11 Keywords
4.12 Exercise

### 4.1 Introduction

A Bus is a group of common wires in which signals travel. The three types of buses used are the
Address Bus, the Data Bus and the control Bus.

4.2 Objectives

At the end of this chapter you will be able to:

- Define Segment Registers
- Explain Logical and physical address
- List the Advantages of segmented memory
- Define Address Bus, Data Bus, Control Bus

4.3 Segment Registers

Within the 1 MB of memory space the 8086/88 defines four 64K-byte memory blocks called the code segment, stack segment, data segment, and extra segment. Each of these blocks of memory is used differently by the processor.

The code segment holds the program instruction codes. The data segment stores data for the program. The extra segment is an extra data segment (often used for shared data). The stack segment is used to store interrupt and subroutine return addresses.

You should realize that the concept of the segmented memory is a unique one. Older-generation microprocessors such as the 8-bit 8086 or Z-80 could access only one 64K-byte segment. This mean that the programs instruction, data and subroutine stack all had to share the same memory. This limited the amount of memory available for the program itself and led to disaster if the stack should happen to overwrite the data or program areas.

The four segment registers (CS, DS, ES, and SS) are used to "point" at location 0 (the base address) of each segment. This is a little "tricky" because the segment registers are only 16 bits wide, but the memory address is 20 bits wide. The BIU takes care of this problem by appending four 0's to the low-order bits of the segment register. In effect, this multiplies the segment register contents by 16.
The point to note is that the beginning segment address is not arbitrary—it must begin at an address divisible by 16. Another way of saying this is that the low-order hex digit must be 0.

Also note that the four segments need not be defined separately. Indeed, it is allowable for all four segments to completely overlap (CS = DS = ES = SS).

Memory locations not defined to be within one of the current segments cannot be accessed by the 8086/88 without first redefining one of the segment registers to include that location. Thus at any given instant a maximum of 256 K (64K * 4) bytes of memory can be utilized. As we will see, the contents of the segment registers can only be specified via S/W. As you might imagine, instructions to load these registers should be among the first given in any 8086/88 program.
4.4 Logical and physical address

Addresses within a segment can range from address 00000h to address 0FFFFh. This corresponds to the 64K-byte length of the segment. An address within a segment is called an offset or logical address. A logical address gives the displacement from the address base of the segment to the desired location within it, as opposed to its "real" address, which maps directly anywhere into the 1 MB memory space. This "real" address is called the physical address.

What is the difference between the physical and the logical address?

The physical address is 20 bits long and corresponds to the actual binary code output by the BIU on the address bus lines. The logical address is an offset from location 0 of a given segment.

When two segments overlap it is certainly possible for two different logical addresses to map to the same physical address. This can have disastrous results when the data begins to overwrite the subroutine stack area, or vice versa. For this reason you must be very careful when segments are allowed to overlap.

You should also be careful when writing addresses on paper to do so clearly. To specify the logical address XXXX in the stack segment, use the convention SS:XXXX, which is equal to
4.5 Advantages of segmented memory

Segmented memory can seem confusing at first. What you must remember is that the program op-codes will be fetched from the code segment, while program data variables will be stored in the data and extra segments. Stack operations use registers BP or SP and the stack segment. As we begin writing programs the consequences of these definitions will become clearer.

An immediate advantage of having separate data and code segments is that one program can work on several different sets of data. This is done by reloading register DS to point to the new data. Perhaps the greatest advantage of segmented memory is that programs that reference logical addresses only can be loaded and run anywhere in memory. This is because the logical addresses always range from 0000h to 0FFFFh, independent of the code segment base. Such programs are said to be relocatable, meaning that they will run at any location in memory. The requirements for writing relocatable programs are that no references be made to physical addresses, and no changes to the segment registers are allowed.

4.6 Address Bus

An address is a unique location in memory. It is like a mailbox in the post office, where each mailbox has its own unique number to identify its location. An address bus consists of 16, 20, 24 or more parallel signal lines. On these lines the CPU sends out the addresses of the memory location that is to be written to or read from. The total number of memory locations is determined by the number or address lines. In the 8086 the address is determined by a 20-bit number. This gives us 220 possible address locations, or 1,048,576 bytes of memory. An address bus is made up of 20 wires, or conductors, labeled A0 through A19, with A0 as the LSB and A19 as the MSB. It is used to locate or find information in memory. It is also used to define a location in memory where information is to be stored. The address bus is sometimes used to identify which I/O port is used for input/output operations.

4.7 Data Bus

A data bus is used to move information (data and instruction) from the MPU to memory and other devices. This is referred to as a write operation. The data bus is also used to receive
information into the MPU. This is called as a read operation. Because the data bus receives and transmits information, it is known as a bi-directional bus. However, it cannot receive and transmit data at the same time. The Intel 8086 has a 16-bit data bus labeled D0 to D15, where D0 is the LSB and D15 is the MSB. The 8086 microprocessor multiplexes the address and data buses. Multiplexing is the process of using the same wires or pins to do different things at different times. When acting as a data bus, the signal lines carry read/write information for memory or input/output information for I/O devices. When acting as an address bus, the same signal lines are used to locate information.

4.8 **Control Bus**

The CPU sends out signals on the control bus to enable the outputs of addressed memory devices or port devices. The control line determines the sequence of operations to be performed. The control bus consists of 4 to 10 parallel signal lines. Typical control bus signals are memory read, memory write, I/O read, and I/O write. To read a byte of data from a memory location, for example, the CPU sends out the address of the desired byte on the address bus and then sends out a memory read signal on the control bus. The memory read signal enables the addressed memory device to output the byte of data on to the data bus where it is required by the CPU.

4.9 **Bus Request and Bus Grant Timings in Minimum Mode System of 8086**

- **Hold Response sequence**: The HOLD pin is checked at leading edge of each clock pulse. If it is received active by the processor before T4 of the previous cycle or during T1 state...
of the current cycle, the CPU activates HLDA in the next clock cycle and for succeeding bus cycles, the bus will be given to another requesting master.

- The control of the bus is not regained by the processor until the requesting master does not drop the HOLD pin low. When the request is dropped by the requesting master, the HLDA is dropped by the processor at the trailing edge of the next clock.

4.10 Summary

All internal registers as well as internal and external data buses were 16 bits wide, firmly establishing the "16-bit microprocessor" identity of the 8086. A 20-bit external address bus gave a 1 MB physical address space ($2^{20} = 1,048,576$). This address space was addressed by means of internal 'segmentation'. The data bus was multiplexed with the address bus in order to fit a standard 40-pin dual in-line package. 16-bit I/O addresses meant 64 KB of separate I/O space ($2^{16} = 65,536$). The maximum linear address space was limited to 64 KB, simply because internal registers were only 16 bits wide. Programming over 64 KB boundaries involved adjusting segment registers (see below) and was therefore fairly awkward. Some of the control pins, which carry essential signals for all external operations, had more than one function depending upon whether the device was operated in min or max mode. The former was intended for small single processor systems whilst the latter was for medium or large systems, using more than one processor.

4.11 Keywords

- Segment Registers
- Address Bus
- Control Bus
- Data Bus
- MB
- CS
- DS
- ES
- SS
4.12 Exercise

1) Define Segment Registers.
2) Explain Logical and physical address.
3) List the Advantages of segmented memory.
4) Define Address Bus, Data Bus, Control Bus.