UNIT 3 ALU AND CONTROL UNIT ORGANISATION

Structure

3.0 Introduction
3.1 Objectives
3.2 ALU Organisation
  3.2.1 A simple ALU Organisation
  3.2.2 Floating point ALU
  3.2.3 Arithmetic Processors
3.3 Control Unit Organization
  3.3.1 Functional Requirements of a Control Unit
  3.3.2 Structure of Control Unit
  3.3.3 An Illustration of Control
  3.3.4 Hardwired Control Unit
3.4 Summary
3.5 Model Answers

3.0 INTRODUCTION

By now we have discussed about the instruction sets and register organisation followed by a discussion on the micro-operations and a simple arithmetic logic unit circuit. In all the previous Units, we have used a term control signals, without any definition. In this unit first we will discuss about the ALU organisation. We will also discuss about the floating point ALU and arithmetic co-processors which are commonly used for floating point computations. This discussion will be followed by the discussions on the control unit, a component which causes all the components of computer to behave effectively to achieve the basic objective i.e. program execution. The control unit causes all the things to happen in the computer.

We will discuss about the functions of a control unit, its structure followed by the hardwired type of control unit. We will introduce the micro-programmed based control unit in the next Unit. The details provided in Unit 3 and 4 about control units can be supplemented by the details given in the further readings of the block.

3.1 OBJECTIVES

At the end of this unit, you will be able to

- discuss the basic organisation of ALU
- discuss the requirements of a floating point ALU
- define the term arithmetic coprocessor
- define what is a control unit and its functions
- describe a simple control unit organisation
- defnine a hardwired control unit

3.2 ALU ORGANISATION

As discussed earlier, an ALU performs the simple arithmetic-logic and shift operations. The complexity of an ALU depends on the type of instruction set which has been realised for it. The simple ALUs can be constructed for fixed point numbers, on the other hand the floating point arithmetic implementation require more complex control logic and data processing capabilities, i.e. the hardware. Several micro-processor families utilizes only fixed point arithmetic capabilities in the ALUs and for floating point arithmetic or other complex functions they may utilise an auxiliary special purpose unit. This unit is called arithmetic co-processor. Let us discuss all these in greater details in this section.

3.2.1 A Simple ALU Organisation

An ALU consist of various circuits which are used for execution of data processing micro-operations. But how these ALU circuits are used in conjunction of other registers and control unit. The simplest organisation in this respect for fixed point ALU was suggested by John von Neumann in his IAS computer design. The structure of this simple organisation is given in figure 1.

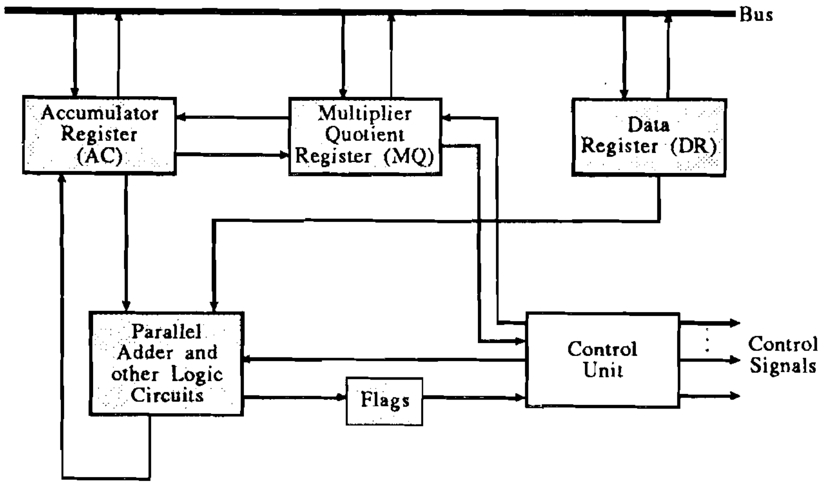

Bus
Accumulator Register (AC)
Multiplier Quotient Register (MQ)
Data Register (DR)
Parallel Adder and other Logic Circuits
Flags
Control Unit
Control Signals

Figure 1 : Structure of a fixed point Arithmetic logic unit

The organisation have three one word registers AC, MQ and DR which are used for data storage. Please note that the arithmetic, logic circuits have two inputs and only one output. In the present case the two inputs are AC and DR registers, while output is AC register. AC and MQ are generally organised as a single AC.MQ register. This register is capable of left or right shift operations. Some of the micro-operations which can be defined on this unit are :

Addition	:	$AC \leftarrow AC + DR$
Subtraction	:	$AC \leftarrow AC - DR$
AND	:	$AC \leftarrow AC \wedge DR$
OR	:	$AC \leftarrow AC \vee DR$
Exclusive OR	:	$AC \leftarrow AC \oplus DR$
NOT	:	$AC \leftarrow \overline{AC}$

In this ALU organisation the multiplication and division are implemented using shift-add/ subtract operations. The MQ (Multiplier–Quotient register) is a special register used for implementation of multiplication and division. We are not giving the details of how this register can be used for implementing multiplication and division algorithms. For more details on these algorithms please refer to further readings. The MQ register stores the multiplier if multiplication is to be performed or the quotient if division is to be performed. For multiplication or division operations DR register stores the multiplicand or divisor respectively. The result of multiplication or division on applying certain algorithm can finally be obtained in AC.MQ register combination. These operations can be represented as:

Multiplication	:	$AC.MQ \leftarrow DR \times MQ$
Division	:	$AC.MQ \leftarrow MQ \div DR$

DR is another important register which is used for storing second operand. In fact it acts as a buffer register which stores the data brought from the memory for an instruction. In machines where we have general purpose registers, for example motorola 68020, any of the register can be utilised as AC, MQ and DR.

Bit slice ALUs

It was feasible to manufacture smaller such as 4 or 8 bits fixed point ALUs on a single IC chip. If these chips are designed as expandable types then using these 4 or 8 bit ALU chips we can make 16, 32, 64 bit array like circuits. There are called bit- slice ALUs. The basic advantage of such ALUs is that these ALUs can be constructed for a desired word size. More details on bit- slice ALUs can be obtained from further readings.

Check Your Progress 1

State true or false

1. A multiplication operation can be implemented as a logical operation.

   True ☐ False ☐

2. The multiplier-quotient register stores the remainder for a division operation

   True ☐ False ☐

3. A word is processed sequentially on a bit slice ALU

   True ☐ False ☐

1.2.2 Floating Point ALU

A floating point ALU can implement floating point operations. But before discussing such a unit, let us first discuss briefly about the floating point operations to get an idea of the requirements of such a unit.

Floating Point Arithmetic

Let us give you a brief introduction to floating point arithmetic and floating point number representation.

A binary floating point number is represented in a normalised form, that is, the number is of the form ± 0. (significand starting with non-zero bit) × $2^{\pm(\text{Exponent Value})}$. Figure 2 shows a format of a 32 bit floating point number.

0	1 8	9 31
Sign	Biased Exponent = 8 bits	Significand = 23 bits

Sign bit is for the significand.

Figure 2 : Floating Point Number Representation

The characteristics of a typical floating point representation of 32 bit in the above figure are:

— Left-most bit is the sign bit of the number

— Mantissa or significand should be in normalised form

— The base of the number is 2

— A value of 128 is added to the *exponent.* (why ?) This is called a bias.

A normal exponent of 8 bit normally can represent exponent values as 0 to 255. However, as we are adding 128 in the biased exponent, thus, the actual exponent values represented will be - 128 to 127.

Now, let us define the range which a normalised mantissa can represent. As for a normalised mantissa the left most bit can not be zero, therefore, it has to be 1. Thus, it is not necessary to store this first bit and it is assumed implicitly for the number. Therefore, a 23 bit mantissa can represent 23 + 1 = 24 bit significand.

Minimum value of the significand:

  The implicit first bit as 1 followed by 23 zero's

    0.1000 0000 0000 0000 0000 0000

    Decimal equivalent = 1×2^{-1} = 0.5

Maximum value of the significand:

  The implicit first bit 1 followed by 23 one's

    0.1111 1111 1111 1111 1111 1111

    Decimal equivalent:

```
    Binary: 0.1111  1111   1111   1111   1111   1111
          + 0.0000  0000   0000   0000   0000   0001   = 2^-24
            1.0000  0000   0000   0000   0000   0000   = 1
```

So decimal equivalent of mantissa = $(1-2^{-24})$

Therefore, in normalised mantissa and biased exponent form, the format of figure 2 can represent binary floating point number in the range:

Lowest negative number : Maximum significand and maximum exponent

$= -(1-2^{-24}) \times 2^{127}$

Highest negative number : Minimum significand and Minimum exponent

$= -0.5 \times 2^{-128}$

Lowest positive number : 0.5×10^{-128}

Highest positive number : $(1-2^{-24}) \times 10^{127}$

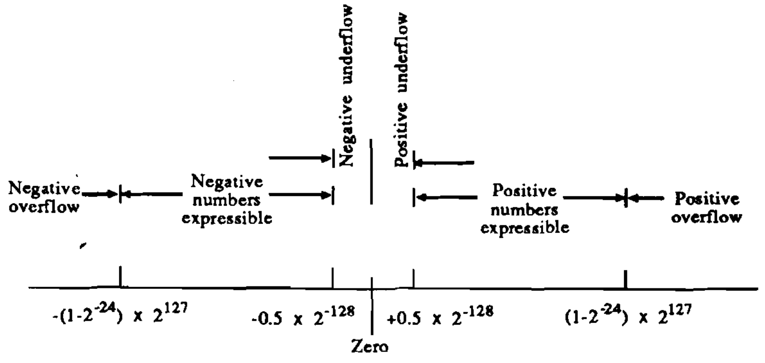


Figure 3 : Binary floating point number range for 32 bit format

In floating point numbers, the basic trade off is between the range of the numbers and accuracy or precision of numbers. If in 32 bit format we increase the exponent bits, the range can be increased, however the accuracy of numbers will go down as significand will become smaller. Let us give an example which will clarify the term precision. Suppose we have one bit binary significand then we can represent only 0.10 and 0.11 in a normalised form. The values such as 0.101, 0.1011 and so on can not be represented as a complete numbers. Either they have to be approximated or truncated and will be represented as either 0.10 or 0.11. Thus, it will create an error. The higher the number of bits in significand better will be precision.

In floating point numbers for increasing both precision and range more number of bits are needed. This can be achieved by using double precision numbers. A double precision format is normally of 64 bits.

Institute of Electrical and Electronics Engineers (IEEE) a society which has created lot of standards regarding various aspects of computer have created IEEE standard 754 for floating point representation and arithmetic. The basic objective of developing this standard was to facilitate the portability of programs from one to another computer. This standard has resulted in development of some standard numerical capabilities in various micro-processors. This representation is shown in figure 4.

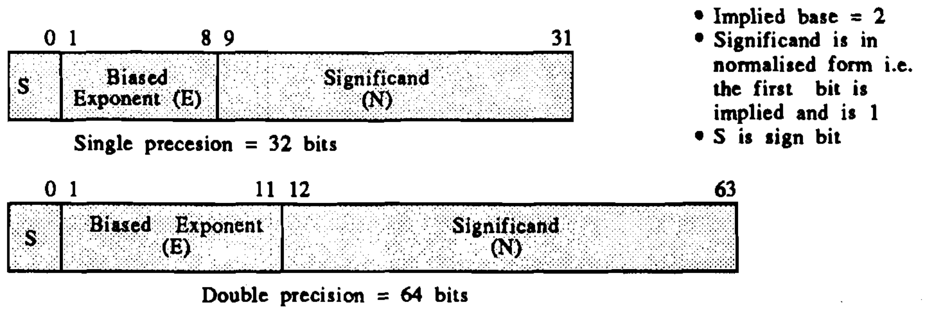

0 1 8 9 31
S
Biased Exponent (E)
Significand (N)
Single precesion = 32 bits
• Implied base = 2
• Significand is in normalised form i.e. the first bit is implied and is 1
• S is sign bit
0 1 11 12 63
S
Biased Exponent (E)
Significand (N)
Double precision = 64 bits

Figure 4 : IEEE Standard 754 format

Figure 5 gives the floating point numbers specified by the IEEE standard 754.

Single Precision Numbers (32 bits)

Exponent (E)	*Significand (N)*	*Value / Comments*
255	Not equal to 0	Do not represent a number

255	0	– or + ∞ depending on sign bit
0 < E < 255	Any	± (1.N) 2^{E-127} For example, if S is zero that is positive number; N = 101 (rest 20 zeros) and E = 207, then the number is = $+(1.101)2^{207-127}$ = $+1.101\text{x}2^{80}$
0	Not equal to 0	± $(0.N)2^{-126}$
0	0	± 0 depending on the sign bit.

Double precision	**Numbers (64 bits)**	
Exponent (E)	*Significand (N)*	*Value / Comments*
2047	Not equal to 0	Do not represent a number
2047	0	– or + ∞ depending on the sign bit
0 < E < 2047	Any	± (1.N) 2^{E-1023}
0	Not equal to 0	± $(0.N)2^{-1022}$
0	0	± 0 depending on the sign bit.

Figure 5 : Values of floating point numbers as per IEEE standard 754

Please note that IEEE standard 754 specifies plus zero and minus zero and plus infinity and minus infinity. Floating point arithmetic is more sticky than fixed point arithmetic. For floating point addition and subtraction we have to:

- check whether a typical operand is zero
- align the significant such that both the significands have same exponent
- add or subtract the significand only and finally
- the significand is normalised again

These operations can be represented as :

$$x + y = (N_x \times 2^{Ex-Ey} + N_y) \times 2^{Ey}$$

and $$x - y = (N_x \times 2^{Ex-Ey} - N_y) \times 2^{Ey}$$

Here, the assumption is that exponent of x (E_x) is greater than exponent of y (E_y). N_x and N_y represent mantissa of x and y respectively.

While for multiplication and division operations the significand need to be multiplied or divided respectively however, the exponent is to be added or to be subtracted respectively. In case we are using bias of 128 or any other bias for exponent then on addition of exponents since both the exponents have bias, the bias gets doubled. Therefore, we must subtract the bias from the exponent on addition of exponents. However, bias is to be added if we are subtracting the exponents. The division and multiplication operating can be represented as:

$$x \times y = (N_x \times N_y) \times 2^{Ex+Ey}$$

$$x + y = (N_x + N_y) \times 2^{Ex-Ey}$$

For more details on floating point arithmetic you can refer to the further readings.

Floating Point ALU

A floating point ALU is implemented using two loosely coupled fixed point arithmetic circuits. Figure 6 shows a simple structure of such a unit.

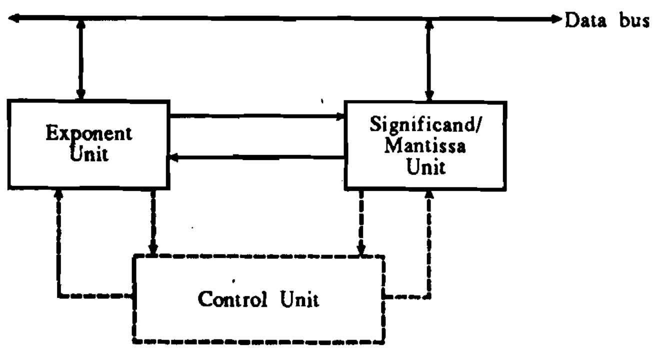


Figure 6 : A Floating point arithmetic unit

The two units can be termed as exponent unit and mantissa unit. The mantissa unit has to perform all the four arithmetic operations on the mantissa. Therefore, a general purpose fixed- point mantissa unit may be used for this purpose. However, for exponent unit we only need circuits to add, subtract and compare the exponents. Thus, a simple circuit containing these functions will be sufficient. The comparison can be performed by a comparator or by simple subtraction operation.

The implementation details of floating point arithmetic on floating point ALUs can be seen from the further readings.

1.2.3 Arithmetic Processors

The very first question in this regard is : "What is an arithmetic processor?" and, "What is the need for arithmetic processors". A typical CPU needs most of the control and data processing hardware for implementing non-arithmetic functions. As the hardware costs are directly related to chip area, a floating point circuit being complex in nature are costly to implement. They are normally not included in the instruction set of a CPU. In such systems, floating point operations are implemented by using software routines. This implementation of floating point arithmetic is definitely slower than the hardware implementation. Now, the question is whether a processor can be constructed only for arithmetic operations? A processor if devoted exclusively to arithmetic functions; can be used to implement a full range of arithmetic functions in the hardware at a relatively low cost. This can be done in a single IC. Thus, a special purpose arithmetic processor, for performing only the arithmetic operations, can be constructed. Although this processor physically is separate, yet will be utilised by the CPU to execute a class of arithmetic instructions. Please note in the absence of arithmetic processors, these instructions may be executed using the slower software routines by the CPU itself. Thus, this auxiliary processor enhances the speed of execution of programs having lot of complex arithmetic computations. In addition, it also helps in reducing program complexity, as it provides more instructions to a machine. Some of the instructions which can be assigned to arithmetic processors can be: add, subtract, multiply and divide fixed and floating point numbers of various lengths; exponentiation; logarithms and trignometric functions.

How can this arithmetic processor be connected to the CPU? Two mechanisms are used for connecting arithmetic processor to the CPU.

If an arithmetic processor is treated as one of the I/O or peripheral unit then it is termed as *peripheral processor.* The CPU sends data and instructions to the peripheral processor which performs the required operations on the data and communicates the results back to the CPU. A peripheral processor have several registers to communicate with the CPU. These registers may be addressed by the CPU as Input/Output register addresses. The CPU and peripheral processors are normally quite independent and communicate with each other by exchange of information using data transfer instructions. This data transfer instructions must be specific instructions in the CPU. This type of connection is called loosely coupled.

On the other hand if the arithmetic processor has register and instruction set which can be considered extension of the CPU registers and instruction set then it is called tightly coupled processor. Here the CPU reserves a special subset of code for arithmetic processor. In such a system the instructions meant for arithmetic processor are fetched by CPU and decoded jointly by CPU and the arithmetic processor, and finally executed by arithmetic processor. Thus, these processors can be considered logical extension of the CPU. Such attached arithmetic processors are termed as *Co-processors.* Let us discuss them in more details.

Peripheral Processor

An example of one such arithmetic processor is the AMD 9511/12 one chip floating point processor. The advantage of this processor is that they can be utilised with any CPU, while the disadvantages are that they need explicitly programmed and slow communication links with the CPU. These processors can be utilised as given in the following figure:

Performer	*Instructions executed*	*Processing details*
1) CPU	Data-transfer instructions	These instructions help in sending a set input operands and commands, e.g. arithmetic operations, to the peripheral processor.
2) Peripheral	Decode & Execute	Results are generated and placed in

	Processor	the command received from CPU using the operands.	registers directly accessible to the CPU
3)	CPU	Checks status by polling a status register or by receiving interrupt from the peripheral processor.	It determines whether the peripheral processor has completed the task.
4)	CPU	Data transfer instruction is executed	CPU obtains the results from the peripheral processor by executing this data transfer instruction.

Figure 7 : Communication between the CPU and Peripheral Processor

In certain implementations CPU has to wait for peripheral processor to finish, therefore, remains idle for that time.

Coprocessors

Coprocessors, unlike peripheral processors, are tailor made for a particular family of CPUs. Normally, each CPU is designed to have a coprocessor interface. The control signal circuits of the CPU is designed for the interface beforehand. Special instructions are earmarked for execution by the coprocessors. These coprocessor instructions can appear in any assembly or machine language program similar to any other instruction. The CPU hardware takes care of the instruction execution by the coprocessors. The coprocessor instructions can be executed even if a coprocessor is not present, by already stored software routines at pre-determined memory locations. If a coprocessor is not attached, then the CPU issues a software (coprocessor) trap which executes a desired software location routine for the instruction. Thus, without changing the source or object code we can execute the coprocessor instructions by the CPU even if the coprocessor is not present. Figure 8 shows a general structure, along with some of the control lines between the CPU and the coprocessor.

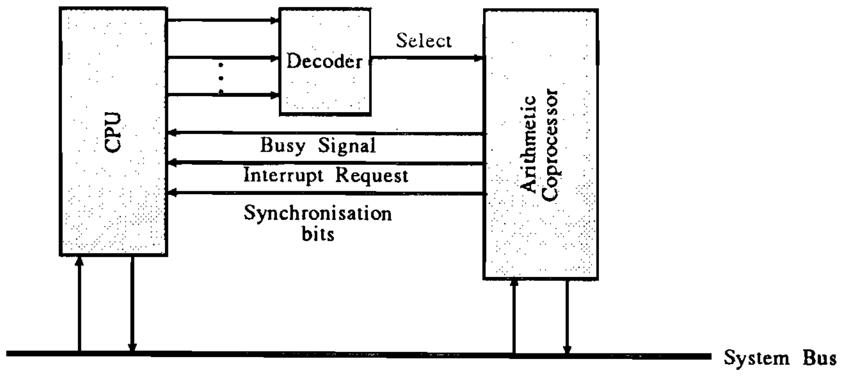


Figure 8 : General Structure of CPU-Coprocessor

As both the processors are directly linked, therefore, they can be synchronised easily. The control lines between them are few. The data transfer between the processors can take place through the system bus. The CPU may act as the master of coprocessor. The registers of coprocessor can be written into or read by the CPU directly as it do for the main memory. Sometimes it is useful to allow the coprocessor to control the bus as in such cases it can control data transfer from memory or can initiate data transfer to the CPU.

In case coprocessor can control the system bus then it is allowed to decode and identify the instructions at the same time CPU is doing. The coprocessor then can execute the instructions meant for it directly. This type of approach is followed in 8087 arithmetic coprocessor of 8086. While in some CPUs, only the CPU can decode the coprocessor instructions. This is the case for the 68881 floating point coprocessor of Motorola 68000 series. A CPU can employ more than one different coprocessors if desired.

Check Your Progress 2

1. Find out the range of a number for the following floating point representation:

Base	$\rightarrow$	2
Sign	$\rightarrow$	1 bit
Exponent	$\rightarrow$	4 bits. Bias of 8 is used

Significand → 3 bits

Assume the normalised mantissa representation

....

....

....

2. State true or false

   (a) A double precision number is used when accuracy requirements are higher.

   True ☐ False ☐

   (b) A zero can not be represented in IEEE standard 754 format.

   True ☐ False ☐

   (c) On multiplication of floating point numbers, the value of bias need to be subtracted after adding the two exponents.

   True ☐ False ☐

   (d) The exponent unit of floating point ALU must perform all the four arithmetic operations.

   True ☐ False ☐

3. What is an arithmetic processor? Compare the co-processor with peripheral processor.

....

....

....

3.3 CONTROL UNIT ORGANISATION

After discussing about the ALU, let us now discuss about a very important component of the CPU i.e. the control unit.

The basic responsibilities of the control unit is to control:

- data exchange of CPU with the memory or I/O modules
- internal operations in the CPU such as:
  - — moving data between registers (register transfer operations)
  - — making ALU to perform a particular operation on the data.
  - — regulating other internal operations.

But how does a control unit controls the above operations? What are the functional requirements of the control unit? What is its structure? Let us explore answer to these questions in the subsequent sub-sections.

3.3.1 Functional Requirements of a Control Unit

Let us first try to define the functions which a control unit must perform in order to get the things to happen. But in order to define the functions of a control unit, one must know what resources and means it have at its disposal. A control unit must know about the:

(a) basic components of the CPU
(b) micro-operation this CPU performs

CPU of a computer consist of the following basic functional components:

- The Arithmetic - Logic Unit (ALU): which performs the basic arithmetic and logical operations.
- Registers: which are used for information storage within the CPU.
- Internal data paths: These paths are useful for moving the data between two registers or between a register and ALU.
- External data paths: The role of these data paths are normally to link the CPU registers with the memory or I/O modules. This role is normally fulfilled by the system bus.
- The control unit: which causes all the operations to happen in the CPU.

The micro-operations performed by the CPU can be classified as:

- Micro-operations for register to register data transfer
- Micro-operations for register to external interface (i.e. in most cases system bus data transfer)
- Micro-operations for external interface to register data transfer
- Micro-operations for performing arithmetic and logic operations. These micro-operations involve use of registers for input and output.

The basic responsibility of the control unit lies in the fact that the control unit must be able to guide the various components of CPU to perform a specific sequence of micro- operations to achieve the execution of an instruction.

What are the functions which a control unit performs to make an instruction execution feasible? The instruction execution is achieved by executing micro-operations in a specific sequence. For different instructions this sequence may be different. Thus, the control unit must perform two basic functions:

- cause the execution of a micro-operation.
- enable the CPU to execute a proper sequence of micro-operation which is determined by the instruction to be executed.

But how does these two tasks achieved? The control unit generates control signals which in turn are responsible for achieving the above two tasks. But, how are these control signals generated? We will answer this question in the later sections. But first let us discuss a simple structure of a control unit.

3.3.2 Structure of Control Unit

A control unit have a set of input values on the basis of which it produces an output control signal which in turn perform micro-operations. These output signals controls the execution of a program. A general model of a control unit is shown in Figure 9.

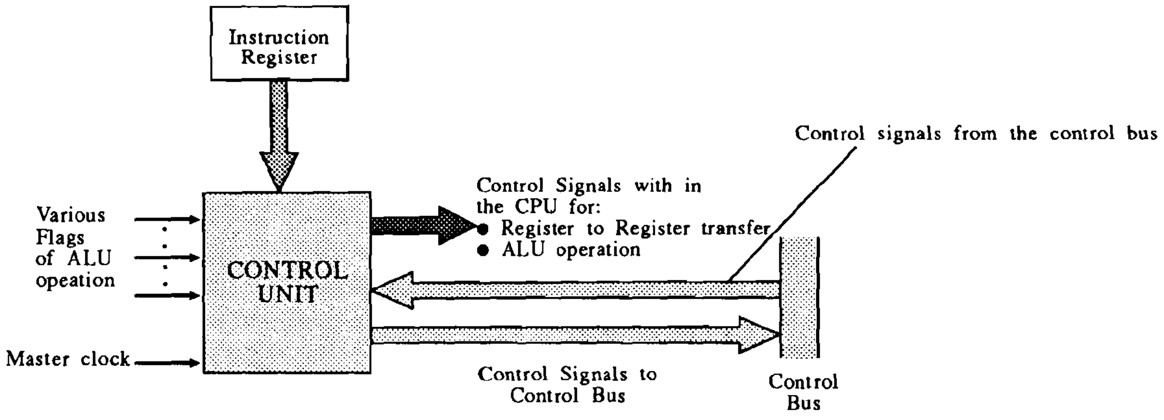

Instruction Register
Control signals from the control bus
Control Signals with in the CPU for:
● Register to Register transfer
● ALU operation
Various Flags of ALU opeation
CONTROL UNIT
Master clock
Control Signals to Control Bus
Control Bus

Figure 9 : A general model of Control Unit

In the model given above the control unit is a black box which has certain inputs and outputs.

The *inputs* to the control unit are:

- **The Master Clock Signal:** This signal causes micro-operations to be performed. In a single clock cycle either a single or a set of simultaneous micro-operations can be performed. The time taken in performing a single micro-operation is also termed as processor cycle time in some machines. However, some micro-operations, such as memory read, may require more than one clock cycle if t_{mem} / t_{cpu} is greater than one.

- **The Instruction Register:** The operation code (opcode) which normally include addressing mode bits of the instruction, help in determining the various cycles to be performed and hence determines the related micro-operations which are needed to be performed.

- **Flags:** Flags are used by the control unit for determining the status of the CPU. The outcomes of a previous operation on ALU can also be detected using flags. For example, a zero flag will help the control unit while executing an instruction ISZ (skip the next instruction if zero flag is set). In case the zero flag is set then control unit will issue control signals which will cause Program Counter (PC) to be incremented by 1. In effect skipping the instruction, which CPU was supposed to execute next.

- **Control Signals from Control Bus:** Some of the control signals are provided to the control unit through the control bus. These signals are issued from outside the CPU. Some of these signals are interrupt signals and acknowledgment signals.

On the basis of the input signals the control unit activates certain output control signals which in turn are responsible for the execution of an instruction. These output control signals are:

- **Control signals which are required within the CPU :** These control signals cause two types of micro-operations, viz. for data transfer from one register to another; and for performing an ALU operation using input and output registers.

- **Control signals to control bus :** The basic purpose of these control signals are to bring or to transfer data from CPU register to memory or I/O modules. These control signals are issued on the control bus to activate a data path.

Now, let us discuss about the requirements from such a unit. A prime requirement for control unit is that it must know all the instructions to be executed and also the nature of the results along with the indication of possible errors. All this is achieved with the help of flags, op-codes, clock and some control signals to itself.

A control unit contain a clock portion, whose job is to provide clock pulses. This clock signal of control unit is used for measuring the timing of the micro-operations. In general, the timing signals from control unit are kept sufficiently long keeping in mind the propogational delays of signals within the CPU along various data paths. As within the same instruction cycle, different control signals are generated at different times for performing different micro-operations, therefore, a counter can be utilised with the clock to keep the count. However, at the end of each instruction cycle the counter should be reset to the initial condition. Thus, the clock to the control unit must provide counted timing signals. Examples, of the functionality of control units along with timing diagrams are given in the further readings. We will not discuss about the timing diagrams in this block.

How are these control signals applied to achieve the particular operation? The control signals are applied directly as the binary inputs to the logic gates of the logic circuits. Do you remember the Enable input defined in Unit 2 of Block-1 or the select inputs of multiplexers. All these inputs are the control signals which are applied to select a circuit (in case of enable) or a path (in case of MUX) or any other operation in the logical circuits.

As one of the responsibilities of the control unit is to keep track on the instruction cycle. Therefore, the control unit can determine when which micro-operation is to be performed. Let us discuss this with the help of an example in the following subsection.

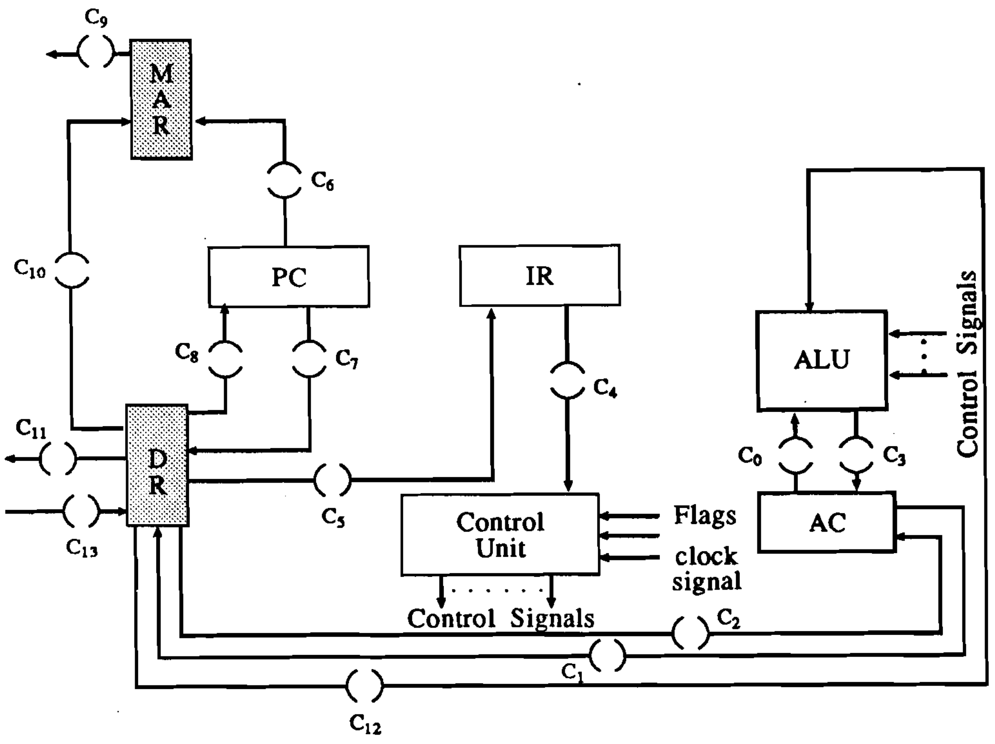
3.3.3 An Illustration of Control

Suppose we have an accumulator machines with the following registers (in the usual functions they do)

- Accumulator (AC)
- Instruction Register (IR)

- The Program Counter (PC)
- Memory Address Register (MAR)
- Data Register (DR)

Figure 10 is a simple representation of such a machine with the requirements of control signals.



Data path	Control Signal
AC → ALU	C_0
AC → DR	C_1
DR → AC	C_2
ALU → AC	C_3
IR → CU	C_4
DR → IR	C_5
PC → MAR	C_6
PC → DR	C_7
DR → PC	C_8
MAR → BUS	C_9
DR → MAR	C_{10}
DR → BUS	C_{11}
DR → ALU	C_{12}
BUS → DR	C_{13}

Figure 10 : A typical CPU with some Control signals

The register transfers for this machine are:

Register	Input from	Output to	Comments
AC	ALU, DR	DR, ALU	AC receives and sends data to DR or to ALU.
IR	DR	CU	Instruction register receives instruction fetched in DR. The op-code of the instruction is used by the control unit for generating control signals for instruction execution.
PC	DR	MAR, DR	Program counter is loaded by the address supplied by DR, however, it can send next instruction address to MAR; or can send the current address for storage, through DR in the case of a subroutine call.
MAR	DR, PC	System Bus	MAR stores the address of the memory unit or the I/O module which is in turn passed on to the system bus. It can be loaded by PC or DR
DR	System Bus, AC	System Bus, ALU, AC, IR, MAR, PC	DR can receive data on system bus from memory unit or I/O module or data for storage from AC. However, it can output data to system bus for storage or to ALU as second operand or to AC as first operand or MAR in case of address, or to the program counter when it contains the address of a ' called subroutine.

Thus, the various sets of data transfer for the machine in figure 10 are: (Source → Destination)

AC → ALU ;	AC → DR ;	DR → AC ;	ALU → AC;
IR → CU ;	DR → IR ;		
PC → MAR ;	PC → DR ;	DR → PC ;	
MAR → (BUS) ;	(PC → MAR) ;	DR → MAR ;	
DR → (BUS) ;	DR → ALU ;	(DR → AC) ;	(DR → IR);
(DR → MAR) ;	(DR→ PC) ;	BUS → DR ;	(AC → DR).

Each of the above transfer require a control signal, however, the data transfer entries given in brackets are occurring again. Thus, we require 14 control signals as shown in figure 10. A connection line in this figure does not indicate a control signal but it indicates data path which exist between the two components. The direction of arrow indicates the direction of data transfer. All the keys on the data transfer paths are triggered by a control signal which is marked there. Please note that for simplicity of diagram we have not shown how these control signals comes from control unit. However, you must keep in mind that all these fourteen control signals are only a subset of control signals generated by the control unit. These control signals can go to their separate destinations. The control signals which are normally needed for different destinations can be categorised as :

- Control signals activating a data path: These control signals opens a gate temporarily. This allows flow of data on the path controlled by that gate. Some of these control signals are shown in the figure given above.
- Control signals activating a function/operation of ALU: These signals open various logic devices and gates inside the ALU. These signals are shown as a group of signals in the figure 10.
- Control signals activating the system bus: These control signals may activate a control line on system bus. Examples of such a signal can be memory or I/O read or write.

Now, let us see what control signals shown in the figure get activated during an instruction cycle for a simple add instruction, where indirect addressing has been used.

Fetch Cycle

Timing	*Micro-operation*	*Comment*	*Control signals needed*
t_1	MAR ← PC	Memory address register is assigned the content of Program Counter.	C_6
t_2	DR ← (BUS)	Read the contents. Additional control signal (not shown here) is needed to active memory read.	C_{13} and control signal for memory read. MAR address input is applied on the system bus.
	PC ← PC + 1	Increment program counter	Control signal for incrementing PC.
t_3	IR ← DR	Transfer the fetched instruction into instruction register.	C_5

Indirect Cycle

t_1	MAR ← IR (ADDRESS)	Assign address from the instruction register to MAR. This content can be acquired from DR as it stores same contents as of IR at present.	C_{10}
t_2	DR ← (BUS)	—	C_{13} and control signal for memory read. MAR address input is applied on the system bus.
t_3	IR (ADDRESS) ←	DR (ADDRESS)	C_5

Execute Cycle

t_1	MAR ← IR (ADDRESS)	C_{10}
t_2	DR ← (BUS)	C_{13} and control signal for memory read and address is applied on the BUS.
t_3	AC ← AC + DR	Control signals for performing this operation along with C_0, C_3 and C_{12}

Interrupt Cycle

t_1	DR	← PC	Store the content of PC	C_7
t_2	MAR	← ADDRESS OF LOCATION OF STORING RETURN ADDRESS	in the memory at an address specified by machine.	Control signals for preforming these opeations
	PC	← Address of the interrupt service program's first instruction		
t_3	(BUS)	← DR	Save the DR contents	C_{11} and control signal which enables memory write

But do we have an internal CPU organisation as shown in Figure 10? Such an organisation will have a large range of data paths hence will be very complex if more registers are there in the CPU. A simple yet effective solution in such a case will be to use internal data bus within CPU. This type of organisation is used in microprocessors, such as INTEL 8085. The organisation of the machine which we have shown in Figure 10, if developed with internal data bus will be very much simplified. This is shown in Figure 11.

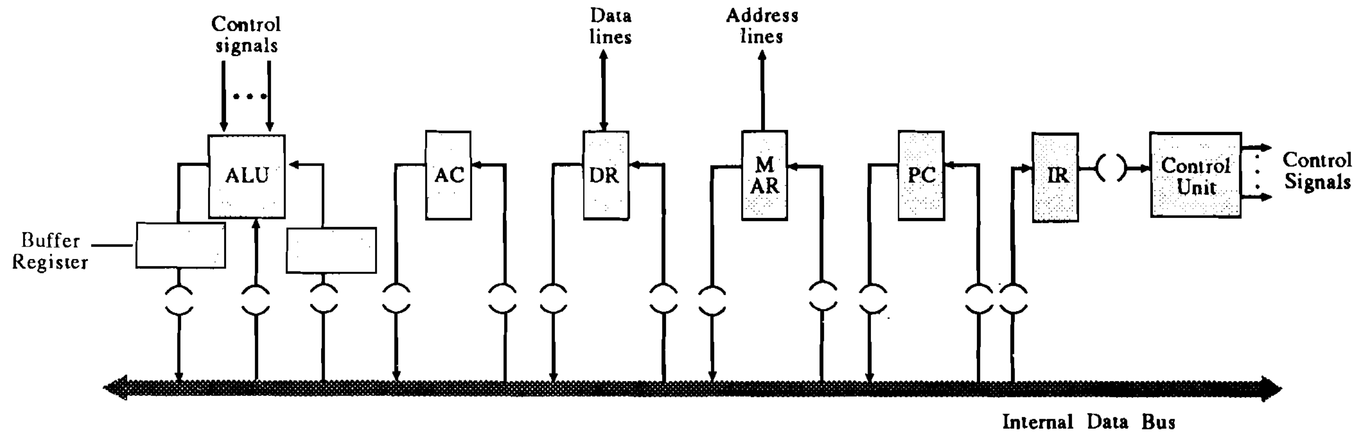


Figure 11 : A bus based CPU with some control signals

In the above case, for each register one input and one output line is controlled by a gate and is connected to data bus (except for IR where we have only input). The required data transfer can be initiated by activating two gates (one output and one input). We have provided two temporary storage with the ALU, otherwise the output of ALU will go back to its input, as both input and output gates of ALU are open for processing, which is undesirable.

The advantages of using internal bus arrangements are:

- Simple data path interconnections which means easy layout for control
- Saving of CPU space as inter-register connection space is minimised. This is very useful in case of microprocessors.

The next question about the control unit is : How can we implement a control unit such that it generates the necessary control signals? The control units are implemented using two general approaches. These are called:

Hardwired Control Unit, and

Micro-programmed Control Unit

We will give only a very brief account of hardwired control units in this unit. The micro-programmed control unit is discussed in details in the next unit of this block.

3.3.4 Hardwired Control Unit

A hardwired control unit is implemented as a logic circuits in the hardware. The inputs to control unit are: the instruction register, flags, timing signals and control bus signals. On the basis of these inputs the output signal sequences are generated. Thus, output control signals are functions of inputs. Thus, we can derive a logical function for each control signal. This, however, will be very complicated if we have a large control unit. The implementation of all the combinational circuits may be very difficult. Therefore, a new approach microprogramming was used. This approach will be discussed in the next unit.

Check Your Progress 3

1. What are the inputs to control unit?

   ....

   ....

2. How does a control unit control the instruction cycle?

   ....

   ....

3. What is the importance of an internal data bus?

   ....

   ....

4. What is a hardwired control unit?

   ....

   ....

3.4 SUMMARY

In this unit, we have discussed about two main components of the CPU, the ALU and the control unit. We have explained the concepts of the basic ALU structure, floating point ALUs and coprocessors. Coprocessors, in today's computers, are used widely and help in implementing graphical and other computation intensive applications. As far as control unit is concerned, we have discussed about a simple structure of a control unit along with an example. More details on these aspects with examples can be seen from the further readings. In this unit we have also introduced the concept of a hardwired control unit. A microprogrammed control unit which is more commonly used is the topic of the next unit.

3.5 MODEL ANSWERS

Check Your Progress 1

1. False
2. False
3. False

Check Your Progress 2

1. The maximum significand = 0.1 111
   Decimal equivalent = $(1-2^{-4})$
   Minimum significand = 0.1 000

	=	0.5
Exponent is	=	-8 to $+8$
Minimum negative number	=	$-(1-2^{-4}) \times 2^{8}$
Maximum negative number	=	-0.5×2^{-8}
Minimum positive number	=	0.5×2^{-8}
Maximum positive number	=	$(1-2^{-4}) \times 2^{8}$

2. (a) True (b) False (c) True (d) False
3. No model answer. Refer to section 3 .2.3

Check Your Progress 3

1. Refer to section 3.3.2
2. Refer to section 3.3.3
3. Refer to section 3.3.3
4. Refer to section 3.3.4