UNIT I

STRUCTURE OF COMPUTERS

1. Digital Computer

The digital computer is a digital system that performs various computational tasks. The word digital implies that the information in the computer is represented by variables that take a limited number of discrete values. These values are processed internally by components that can maintain a limited number of discrete states. The decimal digits 0, 1, 2...9, for example, provide 10 discrete values. The first electronic digital computers, developed in the late 1940s, were used primarily for numerical computations. In this case the discrete elements are the digits. From this application the term digital computer has emerged. In practice, digital computers function more reliably if only two states are used. Because of the physical restriction of components, and because human Logic tends to be binary (i.e., true-or-false, yes or- no statements digital components that are constrained to take discrete values are further constrained to take only two values and are said to be binary.

Digital computers use the binary number system, which has two digits: 0 and 1. A binary digit is called a bit. Information is represented in digital computers in groups of bits. By using various coding techniques, groups of bits can be made to represent not only binary numbers but also other discrete symbols, such as decimal digits or letters of the alphabet. By judicious use of binary arrangements and by using various coding techniques, the groups of bits are used to develop complete sets of instructions for performing various types of computations.

Computer technology has made incredible improvement in the past half century. In the early part of computer evolution, there were no stored-program computer, the computational power was less and on the top of it the size of the computer was a very huge one. Today, a personal computer has more computational power, more main memory, more disk storage, smaller in size and it is available in affordable cost. This rapid rate of improvement has come both from advances in the technology used to build computers and from innovation in computer design. In this course we will mainly deal with the innovation in computer design. The task that the computer designer handles is a complex one: Determine what attributes are important for a new machine, then design a machine to maximize performance while staying within cost constraints. This task has many aspects, including instruction set design, functional organization, logic design, and implementation. While looking for the task for computer design, both the terms computer organization and computer architecture come into picture.

It is difficult to give precise definition for the terms Computer Organization and Computer Architecture. But while describing computer system, we come across these terms, and in literature, computer scientists try to make a distinction between these two terms. Computer architecture refers to those parameters of a computer system that are visible to a programmer or those parameters that have a direct impact on the logical execution of a program. Examples of architectural attributes include the instruction set, the number of bits used to represent different data types, I/O mechanisms, and techniques for addressing memory. Computer organization refers to the operational units and their interconnections that realize the architectural specifications. Examples of organizational attributes include those hardware details transparent to the programmer, such as control signals, interfaces between the computer and peripherals, and the memory technology used.

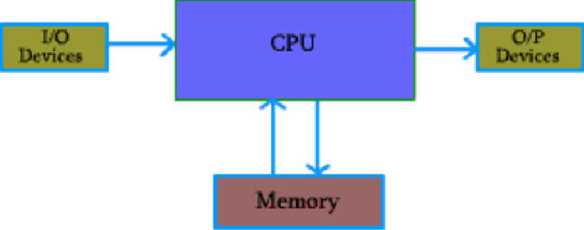
1. Computer Types

The different types of computers are

1. Personal computers: - This is the most common type found in homes, schools, Business offices etc., It is the most common type of desk top computers with processing and storage units along with various input and output devices.
2. Note book computers: - These are compact and portable versions of PC
3. Work stations: - These have high resolution input/output (I/O) graphics capability, but with same dimensions as that of desktop computer. These are used in engineering applications of interactive design work.
4. Enterprise systems: - These are used for business data processing in medium to large corporations that require much more computing power and storage capacity than work stations. Internet associated with servers have become a dominant worldwide source of all types of information.
5. Super computers: - These are used for large scale numerical calculations required in the applications like weather forecasting etc.
6. Basic Computer Model and Functional Units of a Computer

The model of a computer can be described by four basic units in high level abstraction. These basic units are:

Central Processor Unit Input Unit Output Unit Memory Unit



Central processor unit consists of two basic blocks:

* The program control unit has a set of registers and control circuit to generate control signals.
* The execution unit or data processing unit contains a set of registers for storing data and an Arithmetic and Logic Unit (ALU) for execution of arithmetic and logical operations.

Most of the computer operators are executed in ALU of the processor like addition, subtraction, division, multiplication, etc. the operands are brought into the ALU from memory and stored in high speed storage elements called register. Then according to the instructions the operation is performed in the required sequence. The control and the ALU are many times faster than other devices connected to a computer system. This enables a single processor to control a number of external devices such as key boards, displays, magnetic and optical disks, sensors and other mechanical controllers.

Control unit: It effectively is the nerve center that sends signals to other units and senses their states. The actual timing signals that govern the transfer of data between input unit, processor, memory and output unit are generated by the control unit.

Input Unit: With the help of input unit data from outside can be supplied to the computer. Program or data is read into main storage from input device or secondary storage under the control of CPU input instruction. Example of input devices: Keyboard, Mouse, Hard disk, Floppy disk, CD-ROM drive etc.

Output Unit: With the help of output emit computer results can be provided to the user or it can be stored in storage device permanently for future use. Output data from main storage go to output device under the control of CPU output instructions.

Example of output devices: Printer, Monitor, Plotter, Hard Disk, Floppy Disk etc.

Memory Unit:

Memory unit is used to store the data and program. CPU can work with the information stored in memory unit. This memory unit is termed as primary memory or main memory module. These are basically semiconductor memories. There are two types of semiconductor memories -

* Volatile Memory: RAM (Random Access Memory).
* Non-Volatile Memory: ROM (Read only Memory), PROM (Programmable ROM) EPROM (Erasable PROM), EEPROM (Electrically Erasable PROM)

Secondary Memory:

There is another kind of storage device, apart from primary or main memory, which is known as secondary memory. Secondary memories are non-volatile memory and it is used for permanent storage of data and program.

Example of secondary memories:

Hard Disk, Floppy Disk, Magnetic Tape These are magnetic devices,

CD-ROM is optical device

Thumb drive (or pen drive) is semiconductor memory.

1. Basic Operational Concepts

To perform a given task an appropriate program consisting of a list of instructions is stored in the memory. Individual instructions are brought from the memory into the processor, which executes the specified operations. Data to be stored are also stored in the memory.

Examples: - Add LOCA, RO

This instruction adds the operand at memory location LOCA, to operand in register RO & places the sum into register. This instruction requires the performance of several steps,

1. First the instruction is fetched from the memory into the processor.
2. The operand at LOCA is fetched and added to the contents of RO
3. Finally the resulting sum is stored in the register RO

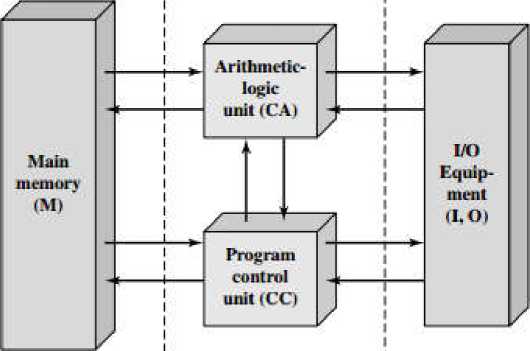
The preceding add instruction combines a memory access operation with an ALU Operations. In some other type of computers, these two types of operations are performed by separate instructions for performance reasons.

Load LOCA, R1 Add Rl, RO

Transfers between the memory and the processor are started by sending the address of the memory location to be accessed to the memory unit and issuing the appropriate control signals. The data are then transferred to or from the memory.

1. Von-Neumann Architecture

Central PrtKjEKsiilg Unit (CPU)



Structure of Lhc IAS Computer

The fig shows how memory & the processor can be connected. In addition to the ALU & the control circuitry, the processor contains a number of registers used for several different purposes.

The instruction register (IR): Holds the instructions that is currently being executed. Its output is available for the control circuits which generates the timing signals that control the various processing elements in one execution of instruction.

The program counter PC: This is another specialized register that keeps track of execution of a program. It contains the memory address of the next instruction to be fetched and executed.

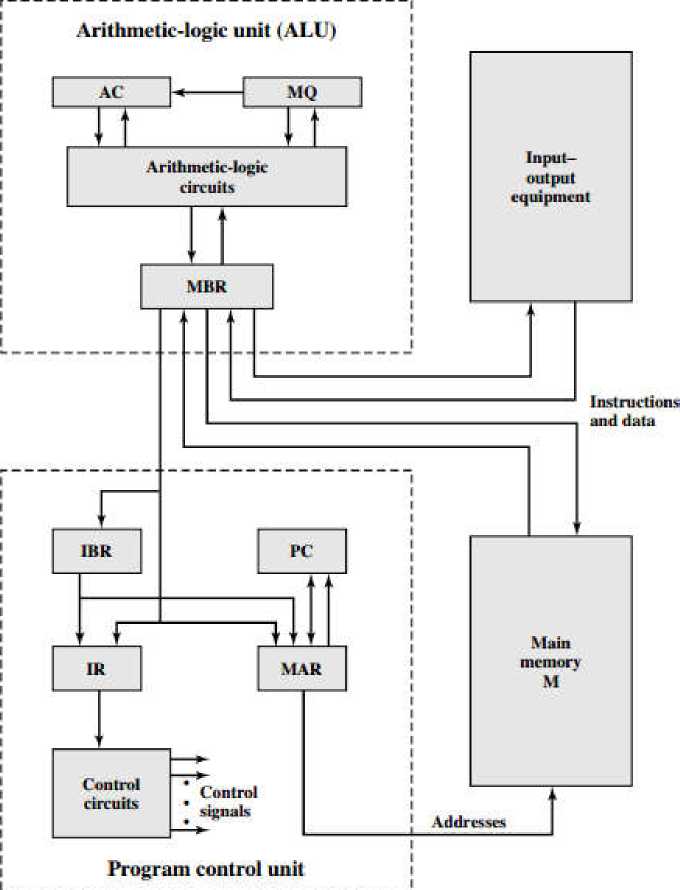
Besides IR and PC, there are n-general purpose registers RO through Rn-1.

The other two registers which facilitate communication with memory are: -

1. MAR - (Memory Address Register):- It holds the address of the location to be accessed.
2. MDR - (Memory Data Register):- It contains the data to be written into or read out of the address location.

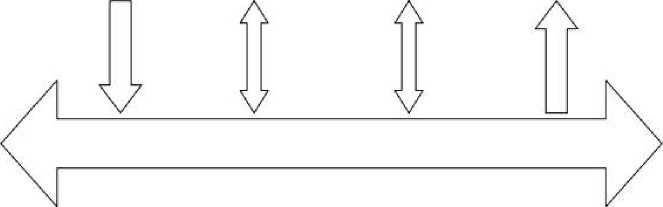
Operating steps are

1. Programs reside in the memory & usually get these through the I/P unit.
2. Execution of the program starts when the PC is set to point at the first instruction of the program.
3. Contents of PC are transferred to MAR and a Read Control Signal is sent to the memory.
4. After the time required to access the memory elapses, the address word is read out of the memory and loaded into the MDR.
5. Now contents of MDR are transferred to the IR & now the instruction is ready to be decoded and executed.
6. If the instruction involves an operation by the ALU, it is necessary to obtain the required operands.
7. An operand in the memory is fetched by sending its address to MAR & Initiating a read cycle.
8. When the operand has been read from the memory to the MDR, it is transferred from MDR to the ALU.
9. After one or two such repeated cycles, the ALU can perform the desired operation.
10. If the result of this operation is to be stored in the memory, the result is sent to MDR.
11. Address of location where the result is stored is sent to MAR & a write cycle is initiated.
12. The contents of PC are incremented so that PC points to the next instruction that is to be executed.



Expanded Structure of LAS Computer

1. Bus Structure

The simplest and most common way of interconnecting various parts of the computer. To achieve a reasonable speed of operation, a computer must be organized so that all its units can handle one full word of data at a given time. A group of lines that serve as a connecting port for several devices is called a bus. In addition to the lines that carry the data, the bus must have lines for address and control purpose. Simplest way to interconnect is to use the single bus as shown below.

|  |  |  |  |
| --- | --- | --- | --- |
| INPUT | MEMORY | PROCESSOR | OUTPUT |

Since the bus can be used for only one transfer at a time, only two units can actively use the bus at any given time. Bus control lines are used to arbitrate multiple requests for use of one bus. Single bus structure is Low cost

Very flexible for attaching peripheral devices

Multiple bus structure certainly increases, the performance but also increases the cost significantly.

**nr**

# nn

Addrvss fines

rrr

■ lias

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| ■=  CPU |  | Memory | • # \* | Memory |  | 1/0 | ■ \* i | 1/0 |
|  |  |  |  |  |  |  |  | / |

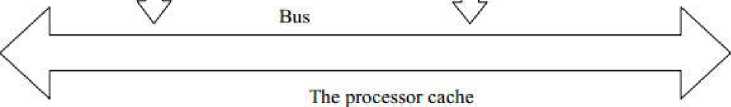
Lun[mL Lints

Lhiiu ILhl‘%

All the interconnected devices are not of same speed & time, leads to a bit of a problem. This is solved by using cache registers (i.e. buffer registers). These buffers are electronic registers of small capacity when compared to the main memory but of comparable speed. The instructions from the processor at once are loaded into these buffers and then the complete transfer of data at a fast rate will take place.

1. Performance

The most important measure of the performance of a computer is how quickly it can execute programs. The speed with which a computer executes program is affected by the design of its hardware. For best performance, it is necessary to design the compiles, the machine instruction set, and the hardware in a coordinated way. The total time required to execute the program is elapsed time is a measure of the performance of the entire computer system. It is affected by the speed of the processor, the disk and the printer. The time needed to execute an instruction is called the processor time. Just as the elapsed time for the execution of a program depends on all units in a computer system, the processor time depends on the hardware involved in the

execution of individual machine instructions. This hardware comprises the processor and the memory which are usually connected by the bus.

Main

Memory

## A

Cache

Memory

Processor

## A

Let us examine the flow of program instructions and data between the memory and the processor. At the start of execution, all program instructions and the required data are stored in the main memory. As the execution proceeds, instructions are fetched one by one over the bus into the processor, and a copy is placed in the cache later if the same instruction or data item is needed a second time, it is read directly from the cache. The processor and relatively small cache memory can be fabricated on a single

IC chip. The internal speed of performing the basic steps of instruction processing on chip is very high and is considerably faster than the speed at which the instruction and data can be fetched from the main memory. A program will be executed faster if the movement of instructions and data between the main memory and the processor is minimized, which is achieved by using the cache.

For example: Suppose a number of instructions are executed repeatedly over a short period of time as happens in a program loop. If these instructions are available in the cache, they can be fetched quickly during the period of repeated use. The same applies to the data that are used repeatedly.

1. Processor clock

Processor circuits are controlled by a timing signal called clock. The clock designer the regular time intervals called clock cycles. To execute a machine instruction the processor divides the action to be performed into a sequence of basic steps that each step can be completed in one clock cycle. The length P of one clock cycle is an important parameter that affects the processor performance. Processor used in today's personal computer and work station have a clock rates that range from a few hundred million to over a billion cycles per second.

1. Basic Performance Equation

We now focus our attention on the processor time component of the total elapsed time. Let 'V be the processor time required to execute a program that has been prepared in some high-level language. The compiler generates a machine language object program that corresponds to the source program. Assume that complete execution of the program requires the execution of N machine cycle language instructions. The number N is the actual number of instruction execution and is not necessarily equal to the number of machine cycle instructions in the object program. Some instruction may be executed more than once, which in the case for instructions inside a program loop others may not be executed all, depending on the input data used.

Suppose that the average number of basic steps needed to execute one machine cycle instruction is S, where each basic step is completed in one clock cycle. If clock rate is 'R' cycles per second, the program execution time is given by T = (N \* S)/R, this is often referred to as the basic performance equation. We must emphasize that N, S & R are not independent parameters changing one may affect another. Introducing a new feature in the design of a processor will lead to improved performance only if the overall result is to reduce the value of T.

1. Clock rate

These are two possibilities for increasing the clock rate 'R'. 1. Improving the IC technology makes logical circuit faster, which reduces the time of execution of basic steps. This allows the clock period P, to be reduced and the clock rate R to be increased. 2. Reducing the amount of processing done in one basic step also makes it possible to reduce the clock period, P. However if the actions that have to be performed by an instructions remain the same, the number of basic steps needed may increase. Increase in the value 'R' that are entirely caused by improvements in IC technology affects all aspects of the processor's operation equally with the exception of the time it takes to access the main memory. In the presence of cache the percentage of accesses to the main memory is small. Hence much of the performance gain excepted from the use of faster technology can be realized.

1. Performance measurements

It is very important to be able to access the performance of a computer, computer designers use performance estimates to evaluate the effectiveness of new features. The previous argument suggests that the performance of a computer is given by the execution time T, for the program of interest. In spite of the performance equation being so simple, the evaluation of'T is highly complex. Moreover the parameters like the clock speed and various architectural features are not reliable indicators of the expected performance. Hence measurement of computer performance using bench mark programs is done to make comparisons possible, standardized programs must be used. The performance measure is the time taken by the computer to execute a given bench mark. Initially some attempts were made to create artificial programs that could be used as bench mark programs. But synthetic programs do not properly predict the performance obtained when real application programs are run. A non-profit organization called SPEC- system performance Evaluation Corporation selects and publishes bench marks. The program selected range from game playing, compiler, and data base applications to numerically intensive programs in astrophysics and quantum chemistry. In each case, the program is compiled under test, and the running time on a real computer is measured. The same program is also compiled and run on one computer selected as reference.

The 'SPEC' rating is computed as follows.

Running time on the reference computer SPEC rating =

Running time on the computer under test

If the SPEC rating = 50, means that the computer under test is 50 times as fast as the ultra spare 10. This is repeated for all the programs in the SPEC suit, and the geometric mean of the result is computed. Let SPECi be the rating for program 'i' in the suite. The overall SPEC rating for the

computer is given by

SPEC rating

(n SPEC^ 21

where 'n' = number of programs in suite.

Since actual execution time is measured the SPEC rating is a measure of the combined effect of all factors affecting performance, including the compiler, the OS, the processor, the memory of comp being tested.

1. Multiprocessor & Multicomputer

* Large computers that contain a number of processor units are called multiprocessor system.
* These systems either execute a number of different application tasks in parallel or execute subtasks of a single large task in parallel.
* All processors usually have access to all memory locations in such system & hence they are called shared memory multiprocessor systems.
* The high performance of these systems comes with much increased complexity and cost.
* In contrast to multiprocessor systems, it is also possible to use an interconnected group of complete computers to achieve high total computational power. These computers normally have access to their own memory units when the tasks they are executing need to communicate data they do so by exchanging messages over a communication network. This properly distinguishes them from shared memory multiprocessors, leading to name message-passing multi computer.

1. Data Representation
2. Binary Number System

We have already mentioned that computer can handle with two type of signals, therefore, to represent any information in computer, we have to take help of these two signals. These two signals corresponds to two levels of electrical signals, and symbolically we represent them as 0 and 1. In our day to day activities for arithmetic, we use the Decimal Number System. The decimal number system is said to be of base, or radix 10, because it uses ten digits and the coefficients are multiplied by power of 10. A decimal number such as 5273 represents a quantity equal to 5 thousands plus 2 hundreds, plus 7 tens, plus 3 units. The thousands, hundreds, etc. are powers of 10 implied by the position of the coefficients. To be more precise, 5273 should be written as:

5xlG5 + 2\*1C3 +7X101 + 3x10\*

However, the convention is to write only the coefficient and from their position deduce the necessary power of 10. In decimal number system, we need 10 different symbols. But in computer we have provision to represent only two symbols. So directly we cannot use decimal number system in computer arithmetic.

For computer arithmetic we use binary number system. The binary number system uses two symbols to represent the number and these two symbols are 0 and 1. The binary number system is said to be of base 2 or radix 2, because it uses two digits and the coefficients are multiplied by power of 2. The binary number 110011 represents the quantity equal to:

1 **x2i** + 1 x **2\*** + 0 x ^ + 0 x 23 +■ 1 X21 + **\** x 2\* - 51 (in decimaJ)

We can use binary number system for computer arithmetic.

1. Representation of Unsigned Integers

Any integer can be stored in computer in binary form. As for example: The binary equivalent of integer 107 is 1101011, so 1101011 are stored to represent 107. What is the size of Integer that can be stored in a Computer? It depends on the word size of the Computer. If we are working with 8-bit computer, then we can use only 8 bits to represent the number. The eight bit computer means the storage organization for data is 8 bits. In case of 8-bit numbers, the minimum number that can be stored in computer is 00000000 (0) and maximum number is 11111111 (255) (if we are working with natural numbers).

So, the domain of number is restricted by the storage capacity of the computer. Also it is related to number system; above range is for natural numbers. In general, for n-bit number, the range for natural number is from 0 to 2n -1. Any arithmetic operation can be performed with the help of binary number system.

1. Signed Integer

We know that for n-bit number, the range for natural number is from 0 to 2n -1.

For n-bit, we have all together 2n different combination, and we use these different combination to represent 2n numbers, which ranges from 0 to 2n -1.

If we want to include the negative number, naturally, the range will decrease. Half of the combinations are used for positive number and other half is used for negative number.

For n-bit representation, the range is from -2n -1 to +2n -1.

For example, if we consider 8-bit number, then range for natural number is from 0 to 255; but for signed integer the range is from -127 to +127.

1. Representation of signed integer

We know that for n-bit number, the range for natural number is from 2n -1.

There are three different schemes to represent negative number:

* Signed-Magnitude form.
* l's complement form.
* 2's complement form.

Signed magnitude form:

In signed-magnitude form, one particular bit is used to indicate the sign of the number, whether it is a positive number or a negative number. Other bits are used to represent the magnitude of the number.

For an n-bit number, one bit is used to indicate the signed information and remaining (n-1) bits are used to represent the magnitude. Therefore, the range is from -2n -1 to +2n -1.

Generally, Most Significant Bit (MSB) is used to indicate the sign and it is termed as signed bit. 0 in signed bit indicates positive number and 1 in signed bit indicates negative number.

For example, 01011001 represents + 169 and 11011001 represents -169

1. The concept of complement

The concept of complements is used to represent signed number.

Consider a number system of base-r or radix-r. There are two types of complements,

* The radix complement or the r's complement.
* The diminished radix complement or the (r - l)'s complement.

1. Representation of Signed integer in l's complement form:

Consider the eight bit number 01011100, l's complements of this number is 10100011. If we perform the following addition:

If we add 1 to the number, the result is 100000000.

01011100 10100011 11111111

Since we are considering an eight bit number, so the 9th bit (MSB) of the result cannot be stored. Therefore, the final result is 00000000. Since the addition of two number is 0, so one can be treated as the negative of the other number. So, l's complement can be used to represent negative number.

Consider the eight bit number 01011100, 2's complements of this number is 10100100. If we perform the following addition:

01011100

10100011 **100000000**

Since we are considering an eight bit number, so the 9th bit (MSB) of the result cannot be stored. Therefore, the final result is 00000000. Since the addition of two number is 0, so one can be treated as the negative of the other number. So, 2's complement can be used to represent negative number.

|  |  |  |  |
| --- | --- | --- | --- |
| Decimal | 2's Complement | 1's complement | Signed Magnitude |
| +7 | 0111 | 0111 | 0111 |
| +6 | 0110 | 0110 | 0110 |
| +5 | 0101 | 0101 | 0101 |
| +4 | 0100 | 0100 | 0100 |
| +3 | 0011 | 0011 | 0011 |
| +2 | 0010 | 0010 | 0010 |
| +1 | 0001 | 0001 | 0001 |
| +0 | 0000 | 0000 | 0000 |
| -0 |  | 1111 | 1000 |
| -1 | 1111 | 1110 | 1001 |
| -2 | 1110 | 1101 | 1010 |
| -3 | 1101 | 1100 | 1011 |
| -4 | 1100 | 1011 | 1100 |
| -5 | 1011 | 1010 | 1101 |
| -6 | 1010 | 1001 | 1110 |
| -7 | 1001 | 1000 | 1111 |
| -a | 1000 | — | — |

1. Representation of Real Number

Binary representation of 41.6875 is 101001.1011

Therefore any real number can be converted to binary number system

There are two schemes to represent real number: Fixed-point representation and Floating-point representation.

1. Fixed-point representation

Binary representation of 41.6875 is 101001.1011 To store this number, we have to store two information,

-- the part before decimal point and -- the part after decimal point.

This is known as fixed-point representation where the position of decimal point is fixed and number of bits before and after decimal point are also predefined.

If we use 16 bits before decimal point and 8 bits after decimal point, in signed magnitude form, One bit is required for sign information, so the total size of the number is 25 bits (l(sign) + 16(before decimal point) + 8(after decimal point)).

1. Floating-point representation

In this representation, numbers are represented by a mantissa comprising the significant digits and an exponent part of Radix R. The format is:

*mantissa* \*

Numbers are often normalized, such that the decimal point is placed to the right of the first non­zero digit. For example, the decimal number, 5236 is equivalent to 5.236 \* 10"'

To store this number in floating point representation, we store 5236 in mantissa part and 3 in exponent part. IEEE has proposed two standard for representing floating-point number:

* Single precision
* Double precision

**Single Precision: Double Precision:**

l s l E ~g~l

M

S: sign bit: 0 denoted + and 1 denotes -

S: sign bit: 0 denoted + and 1 denotes -

E: 8-bit excess -27 exponent

E: 11-bit excess -1023 exponent

M: 23-bit mantissa

M: 52-bit mantissa

1. Error Detecting Codes

To detect and correct the errors, additional bits are added to the data bits at the time of transmission.

* The additional bits are called parity bits. They allow detection or correction of the errors.
* The data bits along with the parity bits form a code word.

1. Parity Checking of Error Detection

It is the simplest technique for detecting and correcting errors. The MSB of an 8-bits word is used as the parity bit and the remaining 7 bits are used as data or message bits. The parity of 8- bits transmitted word can be either even parity or odd parity.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| p | dG | dS | d4 | d3 | d2 | dl | dO |
| 1 |  |  |  |  |  |  |  |

MSB

LSB

bit

7 data bits

Even parity -- Even parity means the number of l's in the given word including the parity bit should be even (2,4,6,....).

Odd parity — Odd parity means the number of l's in the given word including the parity bit should be odd (1,3,5,....).

Use of Parity Bit

The parity bit can be set to 0 and 1 depending on the type of the parity required.

* For even parity, this bit is set to 1 or 0 such that the no. of "1 bits" in the entire word is even. Shown in fig. (a).
* For odd parity, this bit is set to 1 or 0 such that the no. of "1 bits" in the entire word is odd. Shown in fig. (b).

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| P | 4— Data bits —\* | P | | Data bits —\* |
| 0 | 1001011 |  | 1 | 1001011 |
|  |  | Fig-(a) |  |  |
| P | 4— Data bits —\* | P | | 4— Data bits —„ |
| 1 | 1001011 |  | 0 | 1 0 0 0 1 1 c |

Fig- {b)

1. How Does Error Detection Take Place?

Parity checking at the receiver can detect the presence of an error if the parity of the receiver signal is different from the expected parity. That means, if it is known that the parity of the transmitted signal is always going to be "even" and if the received signal has an odd parity, then the receiver can conclude that the received signal is not correct. If an error is detected, then the receiver will ignore the received byte and request for retransmission of the same byte to the transmitter.

|  |  |  |  |
| --- | --- | --- | --- |
|  | P | Data bits —» | |
| Transmitted  code | 0 | j | .001011 |
|  | P | | Error |
| Received code with one error | 0 | C | 00101 1 |

1. Register Transfer and Micro-operations
2. Register Transfer Language

A digi tal system is an interconnection of digital hardware modules that accomplish a specific infomation-prarasing task. The modules are constucted from such digital components as registers, decoders, arithmetic elements, and control logic. The various modules are interconnected with common data and control paths to form a digital computer system. Digital modules are best defined by the registers they contain and the operations that are performed on the data stored in them. The operations

executed on data stored in registers are called micro-operations.

The iteral hardware organition of a digital computer is best defined by specifying:

1. The set of registers it contains and their function.
2. The sequence of microoperations perfored on the biary inforation stored in the registers.
3. The control that initiates the sequence of microoperations.

The symbolc notation used to describe the microoperation transfers among registers is called a register transfer language. The term "register transfer" imples the availability of hardware logic circuits that can perform a stated microoperation and transfer the result of the operation to the same or another register.

1. Register Transfer

Computer registers are designated by capital letters (someties followed by numerals) to denote the function of the register. For example, the register that holds an address for the memory unit is usually called a memory address register and is designated by the name MR. Other designations for registers are PC (for program counter), IR (for istruction register, and R1 (for processor register). The idividual flip-flops in an n-bit register are numbered i sequence from 0 through n - 1, starting from 0 i the rightmost position and icreasing the numbers toward the left. Figure shows the representation of registers in block diagram for. The most common way to represent a register is by a rectangular box with the name of the register inside.

Block diagram of register.

|  |  |  |  |
| --- | --- | --- | --- |
| R\ |  | 7 6 5 4 3 2 1 0 | |
| (a) Register/?  15 0 | | (b) Showing individual bits 15 8 7 0 | |
| R2 |  | PC(W) | PC(L) |

(c) Numbering of bits (d) Divided into two parts

A statement that specifies a register transfer imples that circuits are available from the outputs of the source register to the inputs of the destination register and that the destation register has a parallel load capability. Every statement witten in a register transfer notation implies a hardware construction for implementing the transfer.

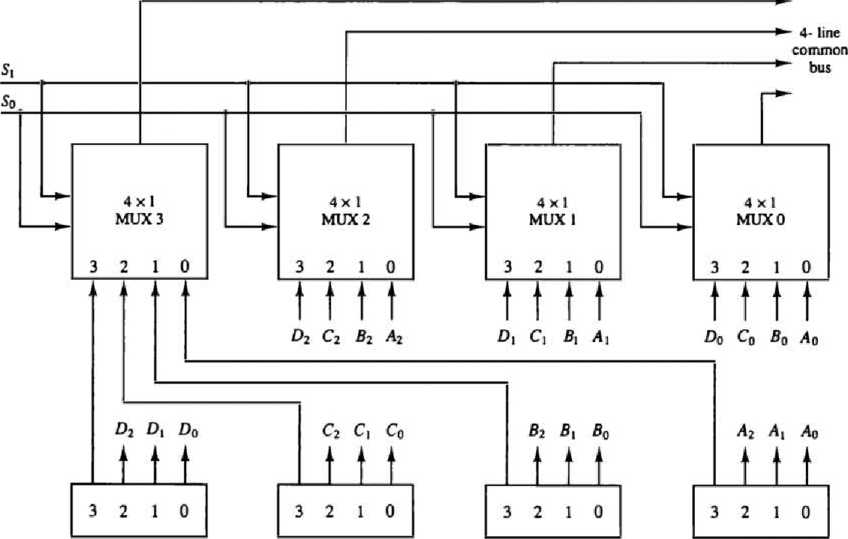
The basic symbols o f the register transfer notation are listed in Table. Registers are denoted by capital letters, and numerals may follow the letters. Parentheses are used to denote a part of a register by specifyig the range of bits or by giving a symbol name to a portion of a register. The

arrow denotes a transfer of information and the direction of transfer. A comma is used to separate two or more operations that are executed at the same time.

|  |  |  |
| --- | --- | --- |
| Symbol | Description | Examples |
| Letters  (and numerals) | Denotes a register | MAR, R2 |
| Parentheses ( ) | Denotes a part of a register | f?2(0-7), R2(L) |
| Arrow +— | Denotes transfer of information | R 2 <- R1 |
| Comma , | Separates two microoperations | R2 <- RL R1 <- R2 |

Basic Symbols for Register Transfers

1. Bus and Memory Transfers

A typical digital computer has many registers, and paths must be provided to transfer information from one register to another. The number of wires will be excessive if separate lines are used between each register and all other registers in the system. A more efficient scheme for transferring information between registers in a multiple-register configuration is a common bus system. A bus structure consists of a set of common lines, one for each bit of a register, through which binary information is transferred one at a time. Control signals determine which register is selected by the bus during each particular register transfer. One way of constructing a common bus system is with multiplexers. The multiplexers select the source register whose binary information is then placed on the bus. The construction of a bus system for four registers is shown in Figure.

Register D

Register C

Register B

Register A

Each register has four bits, numbered 0 through 3. The bus consists of four 4x1 multiplexers each having four data inputs, 0 through 3, and two selection inputs. SI and SO. In order not to complicate the diagram with 16 lines crossing each other, we use labels to show the connections from the outputs of the registers to the inputs of the multiplexers. For example, output 1 of register A is connected to input 0 of MUX 1 because this input is labeled Al. The diagram shows that the bits in the same significant position in each register are connected to the data inputs of one multiplexer to form one line of the bus. Thus MUX 0 multiplexes the four 0 bits of the registers, MUX 1 multiplexes the four 1 bits of the registers, and similarly for the other two bits.

The two selection lines Si and S 0 are connected to the selection inputs of all four multiplexers. The selection lines choose the four bits of one register and transfer them into the four-line common bus. When S1S0 = 00, the 0 data inputs of all four multiplexers are selected and applied to the outputs that form the bus. This causes the bus lines to receive the content of register A since the outputs of this register are connected to the 0 data inputs of the multiplexers. Similarly, register B is selected if S1S0 = 01, and so on. Table shows the register that is selected by the bus for each of the four possible binary value of the selection lines.

In general, a bus system will multiplex k registers of n bits each to produce an n - line common bus. The number of multiplexers needed to construct the bus is equal to n, the number of bits in each register. The size of each multiplexer must be k x 1 since it multiplexes k data lines. For example, a common bus for eight registers of 16 bits each requires 16 multiplexers, one for each line in the bus. Each multiplexer must have eight data input lines and three selection lines to multiplex one significant bit in the eight registers.

|  |  |  |
| --- | --- | --- |
| s, | So | Register selected |
| 0 | 0 | A |
| 0 | 1 | B |
| 1 | 0 | C |
| 1 | 1 | D |

The transfer of information from a bus into one of many destination registers can be accomplished by connecting the bus lines to the inputs of all destination registers and activating the load control of the particular destination register selected. The symbolic statement for a bus transfer may mention the bus or its presence may be implied in the statement. When the bus is includes in the statement, the register transfer is symbolized as follows:

BUS C, R1 <r BUS

The content of register C is placed on the bus, and the content of the bus is loaded into register R1 by activating its load control input. If the bus is known to exist in the system, it may be convenient just to show the direct transfer.

Rl<-C

From this statement the designer knows which control signals must be activated to produce the transfer through the bus.

1.13.1 Memory Transfer

The transfer of information from a memory word to the outside environment is called a read operation. The transfer of new information to be stored into the memory is called a write operation. A memory word will be symbolized by the letter M. The particular memory word among the many available is selected by the memory address during the transfer. It is necessary to specify the address of M when writing memory transfer operations. This will be done by enclosing the address in square brackets following the letter M. Consider a memory unit that receives the address from a register, called the address register, symbolized by AR. The data are transferred to another register, called the data register, symbolized by DR. The read operation can be stated as follows:

Read: DR <- M[AR]

This causes a transfer of information into DR from the memory word M selected by the address in AR. The write operation transfers the content of a data register to a memory word M selected by the address. Assume that the input data are in register R1 and the address is in AR. The write operation can be stated symbolically as follows:

Write: M[AR] <- R1

This causes a transfer of information rom R1 into the memory word M selected by the address in AR.

1. Arithmetic Micro-operations

The basic arithmetic microoperations are addition, subtraction, increment, decrement and shift. The basic arithmetic microoperations are listed in Table. Subtraction is most ofted implemented through complemetation and addition. Instead of using minus operator, we can specifythe subtraction by the following statement:

R3 <- **Rl + R2 + 1**

Symbolic

designation Description

Contents of Rl plus R2 transferred to R3 Contents of R1 minus R2 transferred to R3 Complement the contents of R2 (l's complement) 2's complement the contents of R2 (negate)

**R3**

**R3**

**R2**

**R2**

**R3**

**Rl**

**Rl**

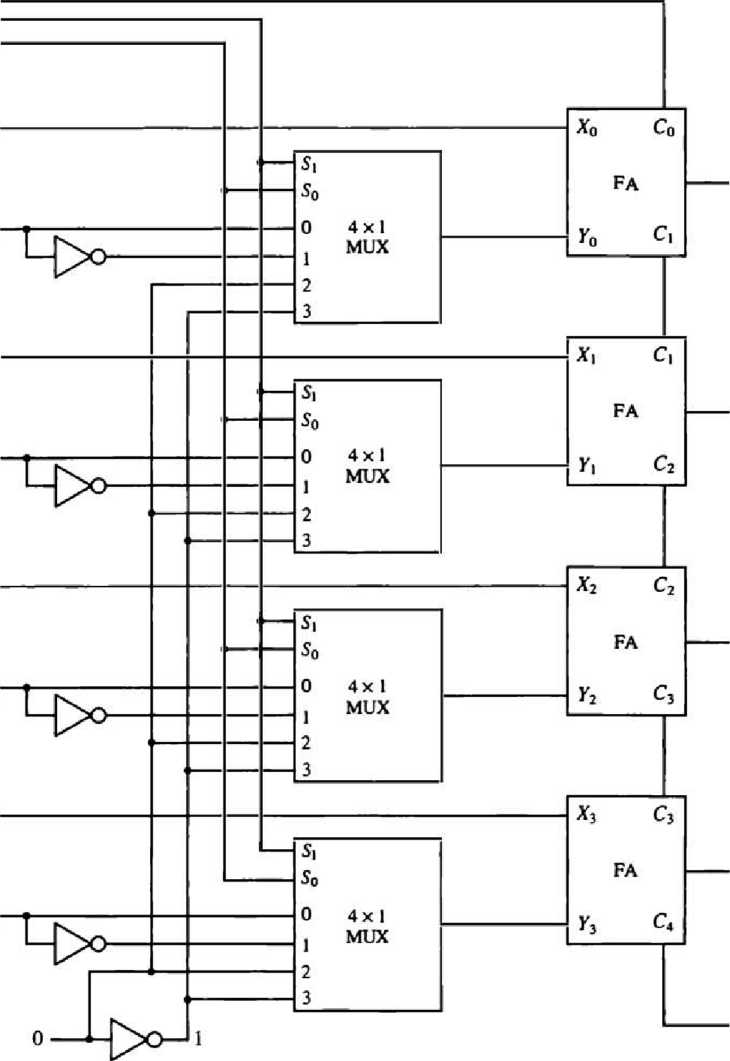
**Rl + R2 JM- R2 R2**

**R2 + 1 Rl + R2 + 1 Rl + 1 Rl - 1**

Rl plus the 2's complement of R2 (subtraction) Increment the contents of R1 by one Decrement the contents of Rl by one

The increment and decrement microoperations are symbolied by plusone and mius-one operations, respectively. These microoperations are iplemented with a combinational circuit or with a binary up-down counter.

The arithmetic microoperations listed in Table can be implemented in one composite arithmetic circuit. The basic component of an arithmetic circuit is the parallel adder. By controlling the data inputs to the adder, it is possible to obtain different types of arithmetic operations. The diagram of a 4-bit arithmetic circuit is shown in Figure. It has four full-adder circuits that constitute the 4-bit adder and four multiplexers for choosing different operations. There are two

4-bit inputs A and B and a 4-bit output D. The four inputs from A go directly to the X inputs of the binary adder. Each of the four inputs from B are connected to the data inputs of the multiplexers. The multiplexers data inputs also receive the complement of B.

CM

Si

So

Ao

Bo

Ai

«i

*A*2

*b2*

^3

A>

Di

*d2*

£>3

The other two data inputs are connected to logic-0 and logic-1. Logic-0 is a fixed voltage value (0 volts for TTL integrated circuits) and the logic-1 signal can be generated through an inverter whose input is 0. The four multiplexers are controlled by two selection inputs. SI and SO . The input cany On goes to the cany input of the FA in the least significant position. The othercarries are connected from one stage to the next. The output of the binary adder is calculated from the following arithmetic sum:

D = A + Y + Qn

where A is the 4-bit binary number at the X inputs and Y is the 4-bit binary number at the Y inputs of the binary adder. Cin is the input carry, which can be equal to 0 or 1. Note that the symbol + in the equation above denotes an arithmetic plus. By controlling the value of Y with the two selection inputs SI and SO and making Qn equal to 0 or 1, it is possible to generate the eight arithmetic microoperations listed in Table.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | Select |  | Input  Y | Output  D = A + Y + Cm | | Microoperation |
| S, | So | cin |
| 0 | 0 | 0 | B | D = A | + B | Add |
| 0 | 0 | 1 | B | D = A | + B + 1 | Add with carry |
| 0 | 1 | 0 | B | D = A | + B | Subtract with borrow |
| 0 | 1 | 1 | B | D = A | + B + 1 | Subtract |
| 1 | 0 | 0 | 0 | D = A |  | Transfer A |
| 1 | 0 | 1 | 0 | D = A | + 1 | Increment A |
| 1 | 1 | 0 | 1 | D = A | - 1 | Decrement A |
| 1 | 1 | 1 | 1 | D = A |  | Transfer A |

When S1S0 = 00, the value of B is applied to the Y inputs of the adder. If Qn = 0, the output D = A + B. If Cin = 1, output D = A + B + 1. Both cases perform the add microoperation with or without adding the input carry.

When S1S0 = 01, the complement of B is applied to the Y inputs of the adder. If Cin = 1, then D = A + B +1. This produces A plus the 2's complement of B, which is equivalent to a subtraction of A - B. When Cin= 0, then D = A + B . This is equivalent to a subtract with borrow, that is, A - B -1.

When S1S0 = 10, the inputs from B are neglected, and instead, all 0's are inserted into the Y inputs. The output becomes D = A + 0 + Cin. This gives D = A when Cin= 0 and D = A + 1 when Cin = 1. In the first case we have a direct transfer from input A to output D. In the second case, the value of A is incremented by 1.

When S1S0 = 11, all l's are inserted into the Y inputs of the adder to produce the decrement operation D = A - 1 when On = 0. This is because a number with all l's is equal to the 2's complement of 1 (the 2's complement of binary 0001 is 1111). Adding a number A to the 2's complement of 1 produces F = A + 2's complement of 1 = A -1. When Cin= 1, then D = A -1 + 1 = A, which causes a direct transfer from input A to output D. Note that the microoperation D = A is generated twice, so there are only seven distinct microoperations in the arithmetic circuit.

1. Logic Microoperations

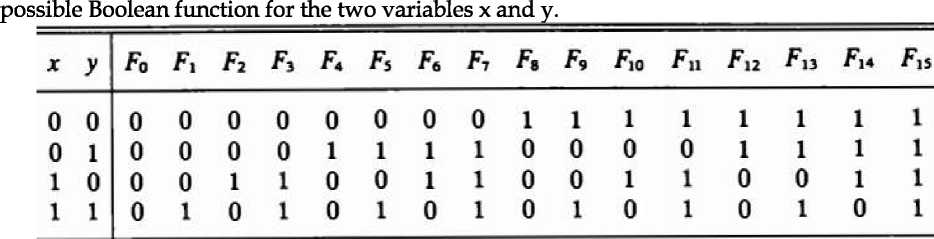
Logic microoperations specify binary operations for strings of bits stored in registers. These operations consider each bit of the register separately and treat them as binary variables. For example, the exclusive-OR microoperation with the contents of two registers R1 and R2 is symbolized by the statement P: R1 «— R1 ® R2

It specifies a logic microoperation to be executed on the individual bits of the registers provided that the control variable P = 1. The content of Rl, after the execution of the microoperation, is equal to the bit-by-bit exclusive-OR operation on pairs of bits in R2 and previous values of Rl. The logic microoperations are seldom used in scientific computations, but they are very useful for bit manipulation of binary data and for making logical decisions.

Special symbols will be adopted for the logic microoperations OR, AND, and complement, to distinguish them from the corresponding symbols used to express Boolean functions. The symbol V will be used to denote an OR microoperation and the symbol A to denote an AND microoperation. The complement microoperation is the same as the Ts complement and uses a bar on top of the symbol that denotes the register name. By using different symbols, it will be possible to differentiate between a logic microoperation and a control (or Boolean) function.

1.15.1 List of Logic Microoperations

There are 16 different logic operations that can be performed with two binary variables. They can be determined from all possible truth tables obtained with two binary variables as shown in Table. In this table, each of the 16 columns FO through F15 represents a truth table of one



|  |  |  |
| --- | --- | --- |
| Boolean function | Microoperation | Name |
| F0 = 0 | F<—0 | Clear |
| H H II II | F<—A A B F<—A/\B | AND |
| F3 = r F\* = x'y | F\*—A F<-A A B | Transfer A |
| Fs = y | F<—B | Transfer B |
| F6 = x ®y | F<—A ©B | Exclusive-OR |
| F7 = x + y | F\*—A \J B | OR |
| F,= {x + y)' | F\*—A V B | NOR |
| F,= (\*©y)' | F\*—A ©£ | Exclusive-NOR |
| Fio = y'  Fu = x + y' | F«-fl F\*—A V B | Complement B |
| Fu =x'  F13 = x' + y | F \*—A F <—A V B | Complement A |
| F, 4 = (xy)' | F<-A A B | NAND |
| Fu = 1 | F<—all l’s | Set to all l’s |

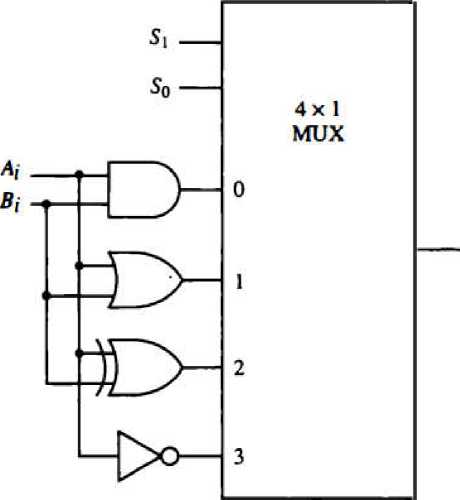
Note that the functions are determined from the 16 binary combinations that can be assigned to F. The 16 Boolean functions of two variables x and y are expressed in algebraic form in the first column of Table. The 16 logic microoperations are derived from these functions by replacing variable x by the binary content of register A and variable y by the binary content of register B. It is important to realize that the Boolean functions listed in the first column of Table represent a relationship between two binary variables x and y. The logic microoperations listed in the second column represent a relationship between the binary content of two registers A and B . Each bit of the register is treated as a binary variable and the microoperation is performed on the string of bits stored in the registers.

1. Hardware Implementation

The hardware implementation of logic microoperations requires that logic gates be inserted for each bit or pair of bits in the registers to perform the required logic function. Although there are 16 logic microoperations, most computers use only four — AND, OR, XOR (exclusive-OR), and complement from which all others can be derived. Figure shows one stage of a circuit that generates the four basic logic microoperations. It consists of four gates and a multiplexer. Each of the four logic operations is generated through a gate that performs the required logic. The outputs of the gates are applied to the data inputs of the multiplexer. The two selection inputs SI and SO choose one of the data inputs of the multiplexer and direct its value to the output. The diagram shows one typical stage with subscript i. For a logic circuit with n bits, the diagram must be repeated n times for i = 0, 1, 2, . . . , n - 1. The selection variables are applied to all stages. The function table in Figure lists the logic microoperations obtained for each combination of the selection variables.

|  |  |  |  |
| --- | --- | --- | --- |
| Si | So | Output | Operation |
| 0 | 0 | E = A\*B | AND |
| 0 | l | E = AvB | OR |
| 1 | 0 | E = A(B B | XOR |
| 1 | 1 | E = A | Complement |

(b) Function table

(a) Logic diagram

Logic microoperations are very useful for manipulating individual bits or a portion of a word stored in a register. They can be used to change bit values, delete a group of bits, or insert new bit values into a register.

1. Shift Microoperations

Shift microoperations are used for serial transfer of data. They are also used in conjunction with arithmetic, logic, and other data-processing operations. The contents of a register can be shifted to the left or the right. At the same time that the bits are shifted, the first flip-flop receives its binary information from the serial input. During a shift-left operation the serial input transfers a bit into the rightmost position. During a shift-right operation the serial input transfers a bit into the leftmost position. The information transferred through the serial input determines the type of shift. There are three types of shifts: logical, circular, and arithmetic.

A logical shift is one that transfers 0 through the serial input. We will adopt the symbols shl and shr for logical shift-left and shift-right microoperations. For example:

R1 <r shl R1 R2 <- shr R2

are two microoperations that specify a 1-bit shift to the left of the content of register R1 and a 1- bit shift to the right of the content of register R2. The register symbol must be the same on both sides of the arrow. The bit transferred to the end position through the serial input is assumed to be 0 during a logical shift.

The circular shift (also known as a rotate operation) circulates the bits of the register around the two ends without loss of information. This is accomplished by connecting the serial output of the shift register to its serial input. We will use the symbols cil and cir for the circular shift left and right, respectively. The symbolic notation for the shift microoperations is shown in Table.

|  |  |
| --- | --- |
| Symbolic designation | Description |
| R «—shl R | Shift-left register R |
| R«— shr R | Shift-right register R |
| R <-ril R | Circular shift-left register R |
| R <—cir R | Circular shift-right register R |
| R «— ashl R | Arithmetic shift-left R |
| R <— ashr R | Arithmetic shift-right R |

An arithmetic shift is a microoperation that shifts a signed binary number to the left or right. An arithmetic shift-left multiplies a signed binary number by 2. An arithmetic shift-right divides the number by 2. Arithmetic shifts must leave the sign bit unchanged because the sign of the number remains the same when it is multiplied or divided by 2. The leftmost bit in a register holds the sign bit, and the remaining bits hold the number. The sign bit is 0 for positive and 1 for negative. Negative numbers are in 2's complement form. Figure shows a typical register of n bits. Bit Rn-i in the leftmost position holds the sign bit. Rn-2 is the most significant bit of the number and Ro is the least significant bit. The arithmetic shift-right leaves the sign bit unchanged and shifts the number (including the sign bit) to the right. Thus Rn-i remains the same, Rn-2 receives the bit from Rn-i, and so on for the other bits in the register. The bit in Ro is lost. The arithmetic shift-left inserts a 0 into Ro, and shifts all other bits to the left. The initial bit of Rn-i is lost and replaced by the bit from Rn-2 ■ A sign reversal occurs if the bit in Rn-i changes in value after the shift. This happens if the multiplication by 2 causes an overflow. An overflow occurs after an arithmetic shift left if initially, before the shift, Rn-i is not equal to Rn-2.

Sign

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  | | | | |
|  | Rn | - 1 | Rn -2 | ► | \*1 |  |

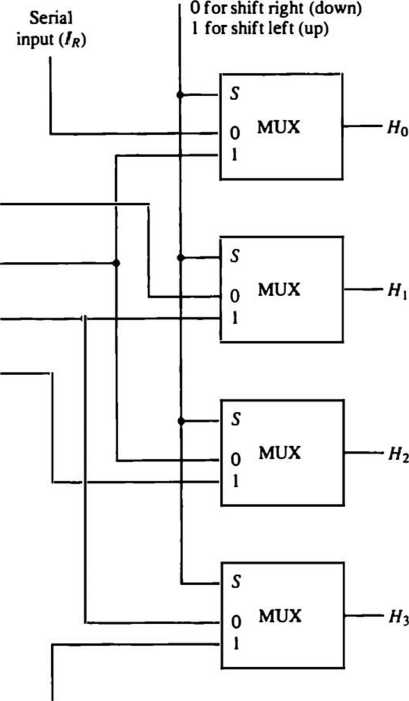
bit

1.16.1 Hardware Implementation

A possible choice for a shift unit would be a bidirectional shift register with parallel load. Information can be transferred to the register in parallel and then shifted to the right or left. In this type of configuration, a clock pulse is needed for loading the data into the register, and another pulse is needed to initiate the shift. In a processor unit with many registers it is more efficient to implement the shift operation with a combinational circuit. In this way the content of a register that has to be shifted is first placed onto a common bus whose output is connected to the combinational shifter, and the shifted number is then loaded back into the register. This requires only one clock pulse for loading the shifted value into the register.

Function table

|  |  |  |  |
| --- | --- | --- | --- |
| Select |  | Output |  |
| S | H0 | Hi H2 | H'i |
| 0 | Ir | i40 A | | a2 |
| 1 | A, | \*2 \*3 | 1l |

A combinational circuit shifter can be constructed with multiplexers as shown in Figure. The 4- bit shifter has four data inputs, AO through A3, and four data outputs, HO through H3. There are two serial inputs, one for shift left (IL) and the other for shift right (IR)- When the selection input S = 0, the input data are shifted right (down in the diagram). When S = 1, the input data

Select

A,

A2

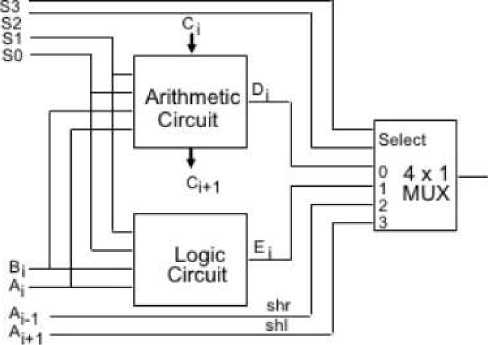
\*3

Serial input (11)

are shifted left (up in the diagram). The function table in Figure shows which input goes to each output after the shift. A shifter with n data inputs and outputs requires n multiplexers. The two serial inputs can be controlled by another multiplexer to provide the three possible types of shifts.

1. Arithmetic Logic Shift Unit

Instead of having individual registers performing the microoperations directly, computer systems employ a number of storage registers connected to a common operational unit called an arithmetic logic unit, abbreviated ALU. To perforin a microoperation, the contents of specified registers are placed in the inputs of the common ALU. The ALU performs an operation and the result of the operation is then transferred to a destination register. The ALU is a combinational circuil so that the entire register transfer operation from the source registers through the ALU and into the destination register can be performed during one clock pulse period. The shift microoperations are often performed in a separate unit, but sometimes the shift unit is made part of the overall ALU. The arithmetic, logic, and shift circuits introduced in previous sections can be combined into one ALU with common selection variables. One stage of an arithmetic logic shift unit is shown in Figure. The subscript i denotes a typical stage. Inputs Ai and Bi are applied to both the arithmetic and logic units. A particular microoperation is selected with inputs SI and SO.



A 4 x 1 multiplexer at the output chooses between an arithmetic output in Ei, and a logic output in Hi. The data in the multiplexer are selected with inputs S3 and S2. The other two data inputs to the multiplexer receive inputs An for the shift-right operation and Ai+i for the shift-left operation. Note that the diagram shows just one typical stage. The circuit must be repeated n times for an n-bit ALU. The output carry Q+i of a given arithmetic stage must be connected to the input carry Ci of the next stage in sequence. The input carry to the first stage is the input carry Cin, which provides a selection variable for the arithmetic operations. The circuit whose one stage is specified in Figure provides eight arithmetic operation, four logic operations, and two shift operations. Each operation is selected with the five variables S3, S2, SI, SO and Cin. The input carry is used for selecting an arithmetic operation only. Table lists the 14 operations of the ALU. The first eight are arithmetic operations and are selected with S3S2 = 00. The next four are logic operations and are selected with S3S2 = 01. The input carry has no effect during the logic operations and is marked with don't-care x's. The last two operations are shift operations and are selected with S3S2 = 10 and 11. The other three selection inputs have no effect on the shift.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| ^3 | Operation  S2 Si | | select  So | C | Operation | Function |
| 0 | 0 | 0 | 0 | 0 | F = A | Transfer A |
| 0 | 0 | 0 | 0 | 1 | F = A + 1 | Increment A |
| 0 | 0 | 0 | 1 | 0 | F = A + B | Addition |
| 0 | 0 | 0 | 1 | 1 | F=A+B+1 | Add with carry |
| 0 | 0 | 1 | 0 | 0 | F = A + B | Subtract with borrow |
| 0 | 0 | 1 | 0 | 1 | F=A+B+1 | Subtraction |
| 0 | 0 | 1 | 1 | 0 | F = A - 1 | Decrement A |
| 0 | 0 | 1 | 1 | 1 | F — A | Transfer A |
| 0 | 1 | 0 | 0 | X | F = A AB | AND |
| 0 | 1 | 0 | 1 | X | F = A\y B | OR |
| 0 | 1 | 1 | 0 | X | F = AQ>B | XOR |
| 0 | 1 | 1 | 1 | X | F = A | Complement A |
| 1 | 0 | X | X | x | F = shr A | Shift right A into F |
| 1 | 1 | X | X | X | F= shl A | Shift left A into F |