| | |
|---|---|
| **Module: 2** | **REGISTER TRANSFER LOGIC:** Inter Register Transfer – Arithmetic, Logic and Shift Micro Operations. **PROCESSOR LOGIC DESIGN:** Processor Organisation - Arithmetic Logic Unit- Design of Arithmetic Unit, Design of Logic circuit, Design of Arithmetic Logic Unit – Status Register- Design of Shifter –Processor Unit –Design of Accumulator. |

## REGISTER TRANSFER LOGIC:

Digital system is a collection of digital hardware modules. A digital system is a sequential logic system constructed with flip flops and gates. The sequential circuit can be specified by means of a state table. Specifying a large digital system with a state table would be very difficult, since the number of states would be very large.

To overcome this difficulty, digital systems are designed using a modular approach, where each modular subsystem performs some functional task. The modules are constructed from such digital functions as registers, counters, decoders, multiplexers, arithmetic elements and control logic. Various modules are interconnected with data and control path. The interconnection of digital functions cannot be described by means of combinational or sequential logic techniques.

The information flow and the processing task among the data stored in the registers can be described by means of **register transfer logic**. The registers are selected as primitive components of the system. Register transfer logic uses a set of expressions and statements which compare the statements used in programming language. It provides the necessary tool for specifying the interconnection between various digital functions.

### Components of Register Transfer Logic

1. ***The set of registers in the system and their functions:***A register also encompasses all type of registers including shift registers, counters and memory units.
2. ***The binary-coded information stored in the registers:*** The binary information stored in registers may be binary numbers, binary coded decimal numbers, alphanumeric characters, control information or any other binary coded information.
3. ***The operations performed on the information stored in the registers:*** The operations performed on data stored in registers are called **micro operations**. Examples are shift, count, add, clear and load
4. ***The control functions that initiate the sequence of operations:*** The control functions that initiate the sequence of operations consists of timing signals that sequence the operations one at a time.

**Register transfer language (Computer hardware description language)**

Symbolic notation used for registers, for specifying operations on the contents of registers and specifying control functions . A statement in a register transfer language consists of control function and a list of microoperations

**Micro-Operation:** Operations performed in data stored in registers. Elementary operation that can be performed parallel during one clock pulse period. The result of operation may replace the previous binary information of a register or may be transfered to another register. Example: Shift, count, clear, add & load

A micro-operation requires one clock pulse for the execution if the operation done in parallel. In serial computer a microoperation requires a number of clock pulses equal to the word time in the system.

**Types of Binary Informations**

Micro operations performed is based on the type of data kept in registers. Type of binary information in the register can be classified into three categories:

- Numerical data such as binary numbers or binary-coded decimal numbers.
- Non-numerical data such as alphanumeric characters or other binary-coded symbols.
- Instruction codes, addresses and other control information used to specify the data processing requirements in the system

**Types of Micro-Operations in digital system**

- ***Interregister transfer micro-operation:*** Do not change the information content when the binary information moves from one register to another
- ***Arithmetic operation:*** Perform arithmetic on numbers stored in registers.
- ***Logic microoperation:*** Perform operations such as AND and OR on individual pairs of bits stored in registers.
- ***Shift microoperation:*** Specify operations for shift registers.

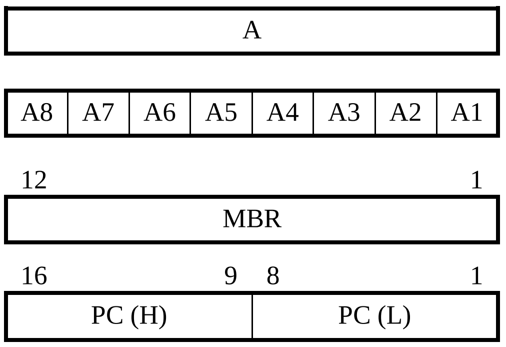
## INTER REGISTER TRANSFER

Computer registers are designated by capital letters (sometimes followed by numerals) to denote the function of the register. [Example: R1 - Processor Register, MAR - Memory Address Register (holds an address for a memory unit), PC - Program Counter, IR - Instruction Register, SR: Status Register].The cells or flipflops of n-bit register are numbered in sequence from1 to n (from 0 to n-1) starting either from left or from right

The register can be represented in 4 ways:

- Rectangular box with name of the register inside,
- The individual cells is assigned a letter with a subscript number,

- The numbering of cells from right to left can be marked on top of the box as the 12 bit register Memory Buffer Register (MBR).
- 16 bit register is partitioned into 2 parts , bits 1 to 8 are assigned the letter L(for low) and bits 9 to 16 are assigned the letter H(for high)



a) Register A

b) Showing individual Cells

c) Numbering of Cells

d) Portions of Register

Registers can be specified in a register transfer language with a declaration statement. For example: Registers in the above figure can be defined with declaration statement such as

DECLARE REGISTER A(8), MBR(12), PC(16)

DECLARE SUBREGISTER PC(L) = PC(1-8), PC(H) = PC(9-16).

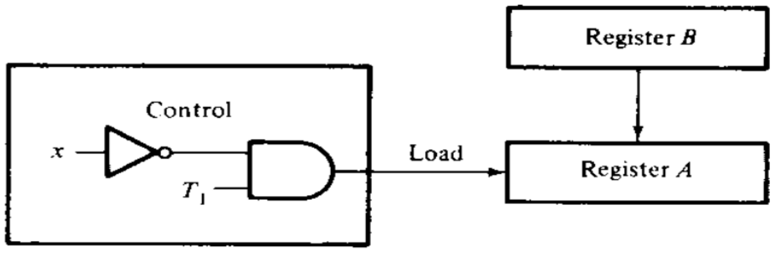
Information transfer from one register to another is described by a **replacement operator**: A ← B. This statement denotes a transfer of the content of register B into register A and this transfer happens in one clock cycle. After the operation, the content of the B (source) does not change. The content of the A (destination) will be lost and replaced by the new data transferred from B.

***Conditional transfer occurs only under a control condition:*** The condition that determines when the transfer is to occurs called a **control function.** A control function is a Boolean function that can be equal to 1 or 0. The control function is included with the statement as follows

$$x'T_1: A \leftarrow B$$

The control function is terminated with a colon. It symbolizes the requirement that the transfer operation be executed by the hardware only when the Boolean function $x'T_1 = 1$. ie; when variable $x = 0$ and timing variable $T_1 = 1$.

Hardware implementation of a controlled transfer: $x'T_1: A \leftarrow B$ is as follows

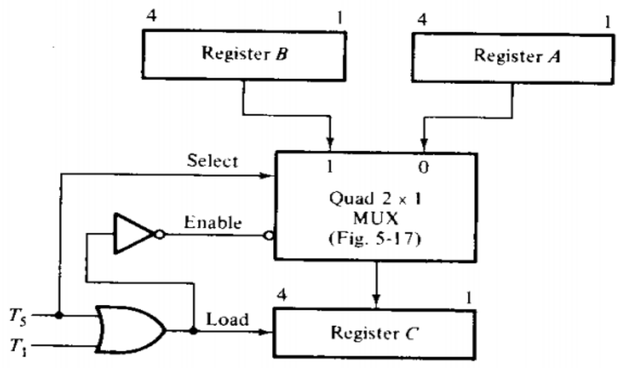


The outputs of register B are connected to the input of register A and the number of lines in this connection is equal to the number of bits in the registers. Register A must have a load control input so that it can be enabled when the control function is 1. It is assumed that register A has an additional input that accepts continuous synchronized clock pulses. The control function is generated by means of an inverter and an AND gate. It is also assumed that the control unit that generates the timing variable $T_1$ i synchronized with the same clock pulses that are applied to register A. The control function stays on during one clock pulse period (when the timing variable is equal to 1) and the transfer occurs during the next transaction of a clock pulse.

Destination register receives information from two sources but not at the same time. Consider,

T1 : C ← A

T5 : C ← B



The first line states that the contents of register A are to be transferred to register C when timimg variable $T_1$ occurs. The second statement uses the same destination register C as the first but with a different source register and a different timing variable. The connections of two source registers to the same destinationr register cannot be done direcly but requires a multiplexer circuit to select between the two possibe paths. The block diagram of the circuit that implements the two statement is shown in the figure. For registers with four bits each, we need a quadruple 2 to 1 line multiplexer inorder to select either A or B. When $T_5$ =1, register B is selected but when $T_1$=1, register A is selected (beacsue $T_5$ must be 0 when $T_1$ is 1). The multiplexer and the load input of register C are enabled everytime $T_1$ and $T_5$ occurs. This causes a transfer of information from the selected ouce register to destination regeister.

The basic symbols of Register Transfer Logic are

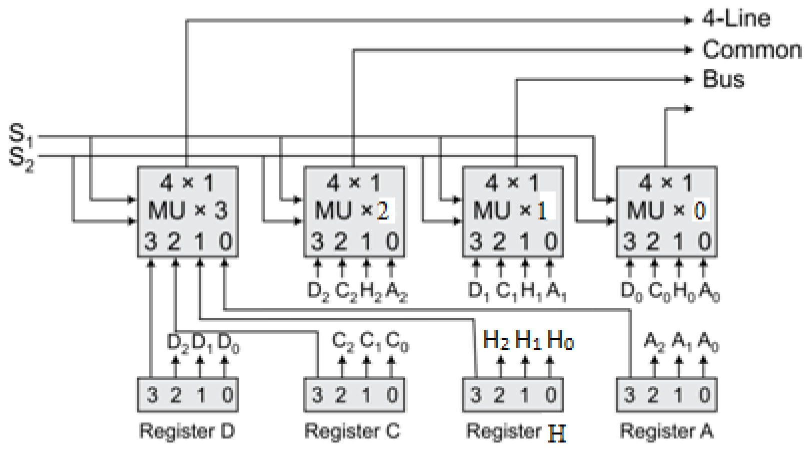
| Symbol | Description | Examples |
|---|---|---|
| Letter (and Numerals) | Denotes a Register | A, MDR, R2 |
| Subscript | Denotes a bit of a Register | $A_2$, $B_6$ |
| Parenthesis ( ) | Denotes a portion of Register | PC(H), MBR (OP) |
| Arrow ← | Denotes transfer of information | A ← B |
| Colon : | Terminates a control function | $X'T_0$: |
| Comma | Seperates two micro-operations | A ← B, B ← A |
| Square Brackets [ ] | Specifies an address for memory transfer | MBR ← M [ MAR ] |

## Bus transfer

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.



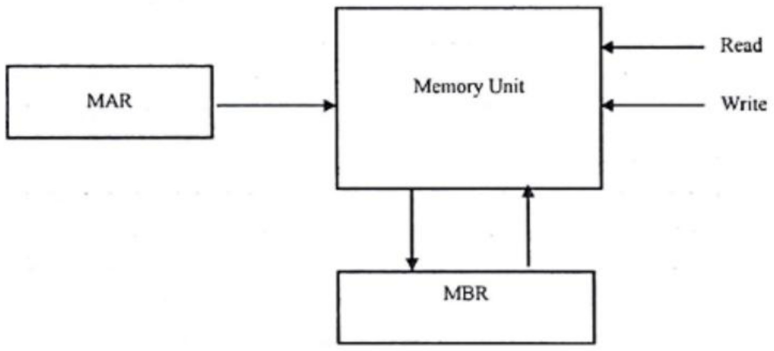
The diagram shows that the bits in the same significant position in each register are connected to the data inputs of one multiplexer to form one line of the bus. Thus, MUX 0 multiplexes the four 0 bits of the registers, MUX 1 multiplexes the four 1 bits of the registers, and similarly for the other two bits.

## 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 write operation. A memory word will be symbolized by the letter M.

The read operation is a transfer from the selected memory register M into MBR (memory buffer register).

Read: MBR←M



Write operation is the transfer from MBR to the selected memory register M.

Write: M ← MBR

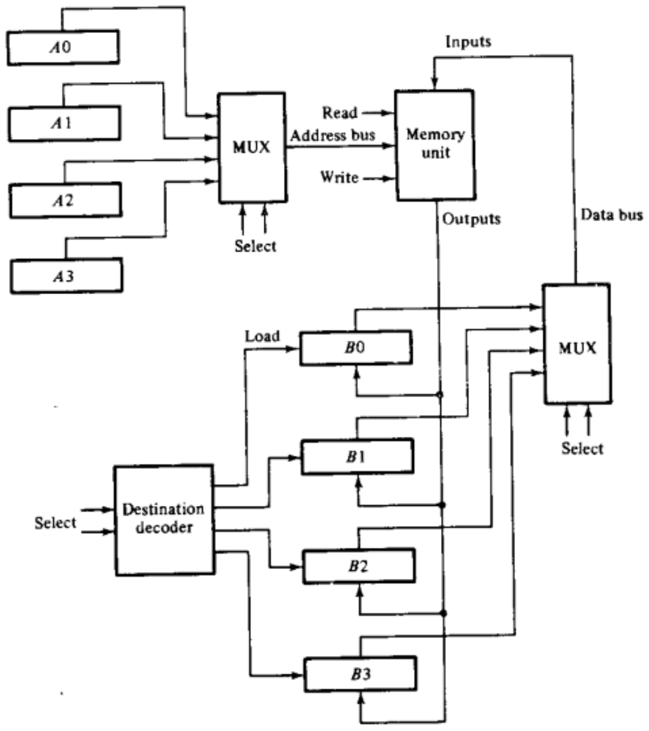
It is necessary to specify the address of M when writing memory transfer operations. This will be done by enclosing the addressing square brackets following the letter M.

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 memory 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 memory write operation transfers the content of a register R1 to a memory word M selected by the address in address AR. The notation is:

Write: M [AR] ← R1



The block diagram shows the memory unit that communicates with multiple registers. The address to the memory unit comes from an address bus. Four registers are connected to the bus and any one may supply an address. The output of the memory can go to any one of four registers which are selected by a decoder. The data input to the memory come from the data bus which selects one of four registers. A memory word is specified in such a system by the symbol M followed by the register enclosed in a square bracket. The contents of the register within the square bracket specifies the address for M.

## ARITHMETIC, LOGIC AND SHIFT MICRO OPERATION

### Arithmetic Micro-Operation

The basic arithmetic micro-operations are:

- Addition,
- Subtraction,
- Increment,
- Decrement
- Arithmetic shift.

The increment and decrement micro-operations are implemented with a combinational circuit or with a binary up-down counter as these micro-operations use plus-one and minus-one operation respectively.

The arithmetic add microoperations are defined by the statement

$$F \leftarrow A + B.$$

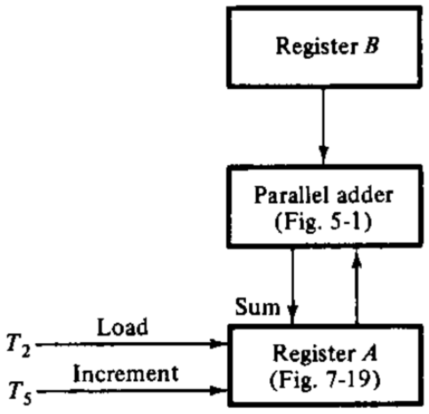
It states that the contents of register A are to be added to the contents of register B and the sum is transfered to register F To implement this statement require 3 registers A, B and F and a digital functon that performs the addition operation such as parallel addder.

| Symbolic designation | Description |
|---|---|
| $F \leftarrow A + B$ | Contents of *A* plus *B* transferred to *F* |
| $F \leftarrow A - B$ | Contents of *A* minus *B* transferred to *F* |
| $B \leftarrow \overline{B}$ | Complement register *B* (1's complement) |
| $B \leftarrow \overline{B} + 1$ | Form the 2's complement of the contents of register *B* |
| $F \leftarrow A + \overline{B} + 1$ | *A* plus the 2's complement of *B* transferred to *F* |
| $A \leftarrow A + 1$ | Increment the contents of *A* by 1 (count up) |
| $A \leftarrow A - 1$ | Decrement the contents of *A* by 1 (count down) |

There must be a direct relationship between the statements written in a register transfer language and the registers and digital functions which are required for the implementation

Consider the statements

$$T2 : A \leftarrow A + B$$
$$T5 : A \leftarrow A + 1$$



Timing variable T2 initiates an operation to add the contents of register B to the present contents of A with a parallel adder. Timing variable T5 increments register A with a counter. The transfer of the sum from parallel adder into register A can be activated with a load input in the register. Register be a counter with parallel load capability.

The parallel adder receives input information from registers A and B. The sum bits from the parallel adder are applied to the inputs of A and timing variable T2 loads the sum into register A. Timing variable T5 increments there by enabling increment input register.

Two basic arithmetic operations (multiplication and divide) are not included in the basic set of micro-operations. They are implemented by means of a combinational circuit. In general, the multiplication micro-operation is implemented with a sequence of add and shift micro-operations. Division is implemented with a sequence of subtract and shift micro-operations.

**Logic Micro-Operations**

Logic micro-operations 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 micro-operation with the contents of two registers A and B is symbolized by the statement

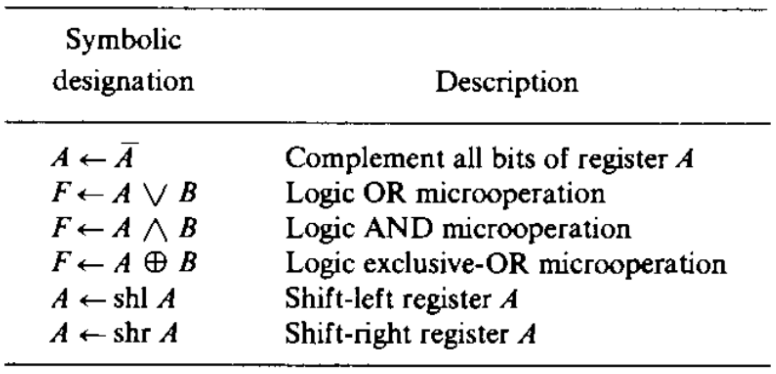
$$F \leftarrow A \oplus B$$

It specifies a logic micro-operation that consider each pair of bits in the registers as a binary variable. Let the content of register A is 1010 and the content of register B is 1100. The exclusive-OR micro-operation stated above symbolizes the following logic computation:

| | |
|---|---|
| 1010 | Content of A |
| $\underline{1100}$ | Content of B |
| 0110 | Content of $F \leftarrow A \oplus B$ |

The content of F, after the execution of the micro-operation, is equal to the bit-by-bit exclusive-OR operation on pairs of bits in B and values of A. The logic micro-operations are seldom used in scientific computations, but they are very useful for bit manipulation of binary data and for making logical decisions.

Logic and Shift Micro instructions are



| Symbolic designation | Description |
|---|---|
| $A \leftarrow \overline{A}$ | Complement all bits of register $A$ |
| $F \leftarrow A \vee B$ | Logic OR microoperation |
| $F \leftarrow A \wedge B$ | Logic AND microoperation |
| $F \leftarrow A \oplus B$ | Logic exclusive-OR microoperation |
| $A \leftarrow \text{shl } A$ | Shift-left register $A$ |
| $A \leftarrow \text{shr } A$ | Shift-right register $A$ |

The + symbol has different meaning. When + occurs in a microoperation , it denotes arithmetic plus.. When it occurs in a control or Boolean function, it denotes a logical OR operation.

*Example:* $T1 + T2 : A \leftarrow A + B, C \leftarrow D \vee F$

The + between T1 and T2 is an OR operation between 2 timing variables of a control function and the + between A & B specifies an add microoperation

**Shift Micro-Operations**

Shift micro-operations shift the contents of a register either left or right. These micro-operations are generally used for serial transfer of data. They are also used along with arithmetic, logic, and other data-processing operations.

No conventional symbol for shift operation. Here adopt symbols shl or shr [shl - shift left shr - shift right]

| | | |
|---|---|---|
| Example: | $A \leftarrow$ shl A | 1-bit shift to the left of register A |
| | $B \leftarrow$ shr B | 1-bit shift to the right of register B |

When 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.

Example:

$A \leftarrow$ shl, $A_1 \leftarrow A_n$

Circular shift that transfers the leftmost bit from $A_n$ into the rightmost flip flop $A_{1.}$

$A \leftarrow$ shr, $A_n \leftarrow E$

Shift right operation with leftmost flip flop $A_n$ receiving the value of the 1-bit register E.

## PROCESSOR ORGANIZATION

The processor part of a computer CPU is sometimes referred to as the data path of the CPU because the processor forms the paths for the data transfers between the registers in the unit. The various paths are said to be controlled by means of gates that open the required path and close all others. A processor unit can be designed to fulfill the requirements of a set of data paths for a specific application.

In a processor unit, the data paths are formed by means of buses and other common lines. The control gates that formulate the given path are essentially multiplexers and decoders whose selection lines specify the required path. The processing of information is done by one common digital function whose data path can be specified with a set of common selection variables.

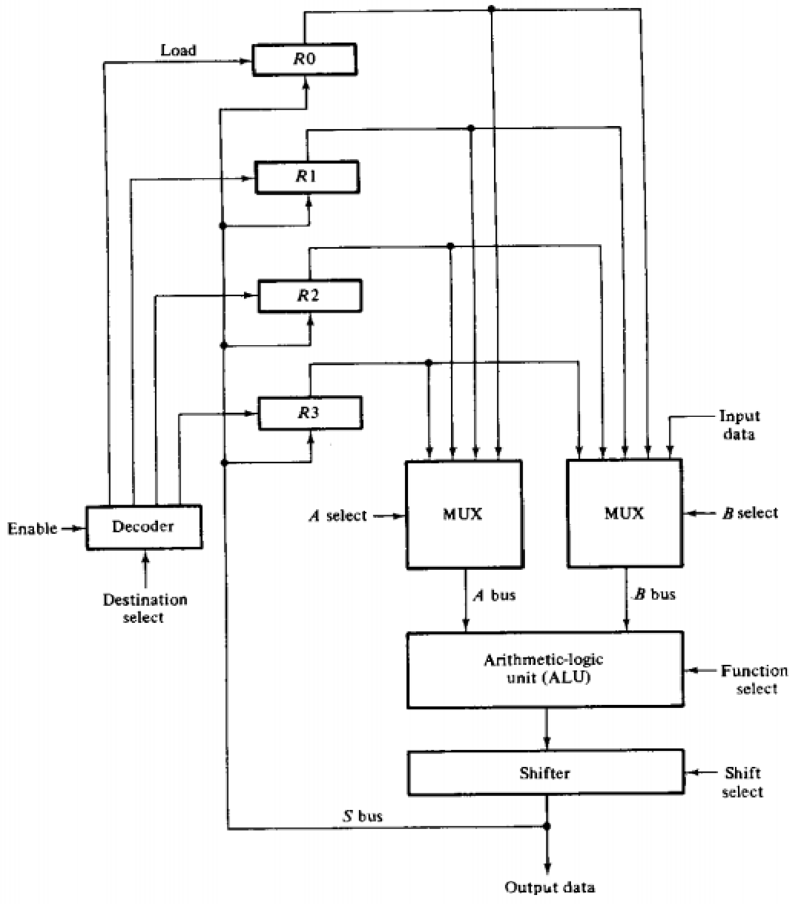
### Bus Organization

A bus organization for four processor registers is shown in Figure. Each register is connected to two multiplexers (MUX) to form input buses A and B. The selection lines of each multiplexer select one register for the particular bus. The A and B buses are applied to a common arithmetic logic unit.The function selected in the ALU determines the particular operation that is to be performed.

The shift micro-operations are implemented in the shifter .The result of the micro-operation goes through the output bus S into the inputs of all registers. The destination register that receives the information from the output bus is selected by a decoder.

When enabled, this decoder activates one of the register load inputs to provide a transfer path between the data on the S bus and the inputs of the selected destination register. The output bus S provides the terminals for transferring data to an external destination. One input of multiplexer A or B can receive data from the outside

The control unit that supervises the processor bus system directs the information flow through the ALU by selecting the various components in the unit.



For example, to perform the microoperation:

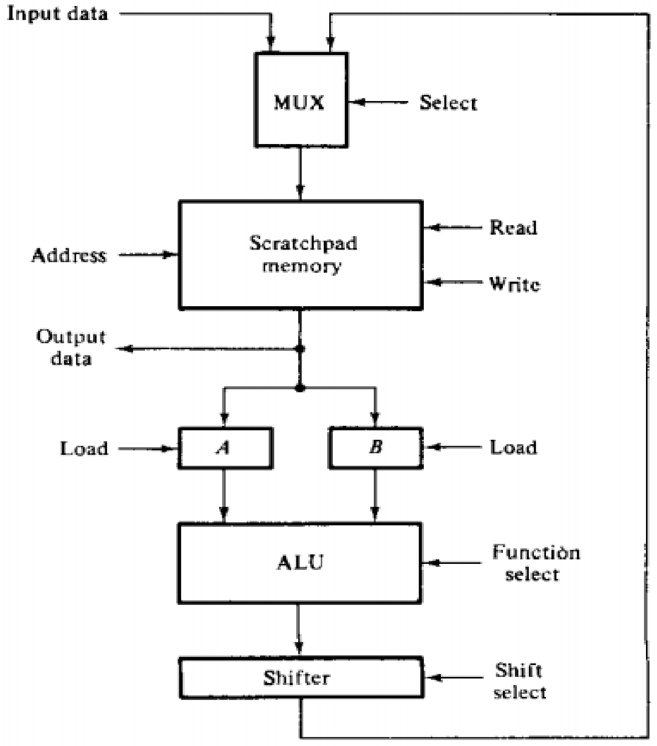
$$R1 \leftarrow R2 + R3$$

The control must provide binary selection variables to the following selector inputs:

1. ***MUX A selector:*** to place the contents of R2 onto bus A.
2. ***MUX B selector***: to place the contents of R3 onto bus B.
3. ***ALU function selector:*** to provide the arithmetic operation A + B.
4. ***Shift selector:*** for direct transfer from the output of the ALU onto output bus S (no shift).
5. ***Decoder destination selector***: to transfer the contents of bus S into R 1.

**Scratchpad memory**

The register in a processor unit can be enclosed in a small memory unit. When included in a processor unit,a small memory is sometime called a scratchpad memory.The use of a small memory is a cheaper alternative to collecting processor registers through a bus system.The difference between the two system is the manner in which information is selected for transfer into the ALU. In a bus system, the information transfer is selected by the multiplexer that form the buses.



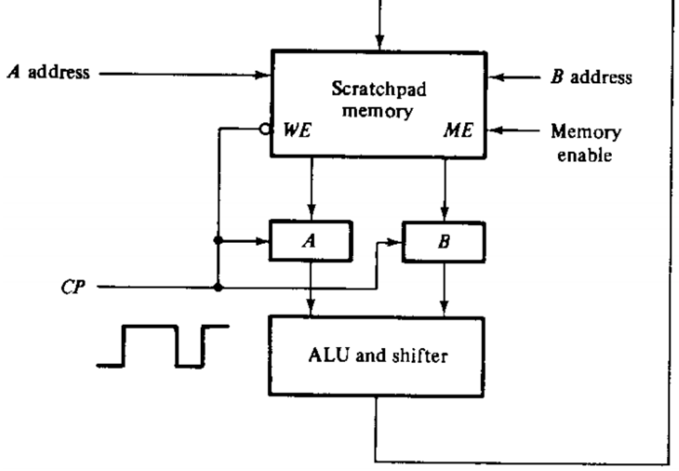
Processor unit that uses scratchpad memory is shown in figure. Resource register is selected from memory and loaded into register A. A Second source register is selected from memory and loaded into register B. The information in A and B is manipulated in the ALU and shifter. Result of the operation is transferred to a memory register specifying its word address and activating the memory-write input control.

Assume that the memory has eight words, so that an address must be specified with three bits. To perform the operation

$R1 \leftarrow R2 + R3$

The control must provide binary selection variable to perform the following sequence of micro-operations

| | |
|---|---|
| $T_1$: $A \leftarrow M[010]$ | read R2 to register A |
| $T_2$: $B \leftarrow M[011]$ | read R3 to register B |
| $T_3$: $M[001] \leftarrow A + B$ | perform operation in ALU and transfer result to R1 |



Control function $T_1$ must supply an address of 010 to the memory and activate the read and load A inputs. control function $T_2$ must supply an address 011 to the memory and activate the read and load B inputs. Control function $T_3$ must supply the function code to the ALU and shifter to perform an add operation, apply an address 001 to the memory, select the output of the shifter for the MUX and activate the memory write input.

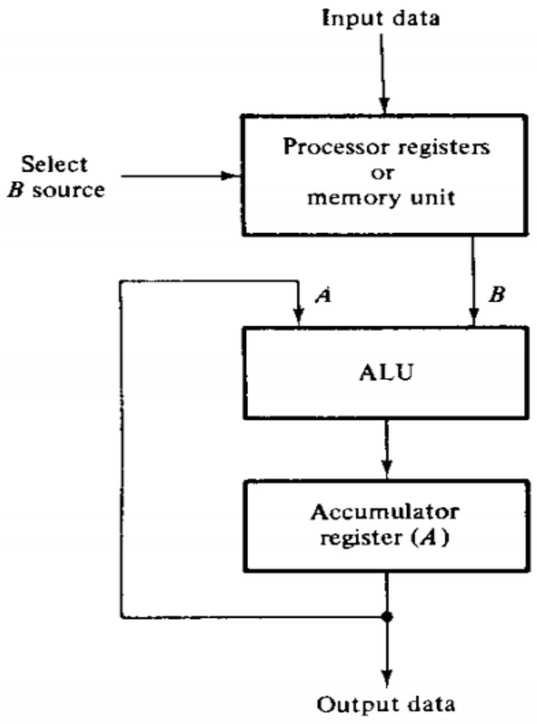
Some processor employ a 2 port memory in order to overcome the delay caused when reading two source registers. A 2-port has two separate address lines to select two words of memory simultaneously. The organization of a processor unit with a 2-port scratchpad memory is shown in figure.

**Accumulator Register**

An accumulator is a register for short-term, intermediate storage of arithmetic and logic data in a computer's CPU (central processing unit).The most elementary use for an accumulator is adding a sequence of numbers. The numerical value in the accumulator increases as each number is added, exactly as it happens in a simple desktop calculator (but much faster, of course). Once the sum has been determined, it is written to the main memory or to another register.

The accumulator register in a processor unit is a multipurpose register capable of performing not only the add micro-operation, but many other operations as well.

The block diagram shows the processor unit that employs an accumulator units.



To form the sum of two numbers stored in processor registers, it is nessesary to add them in the A register using the following sequence of micro-operations:

| | |
|---|---|
| $T_1$: A ← 0 | Clear A |
| $T_2$: A ← A + R1 | Transfer R1 to A |
| $T_3$: A ← A + R2 | Add R2 to A |

The sum / result formed in A may be used for other computation or may be transfered to a required destination.

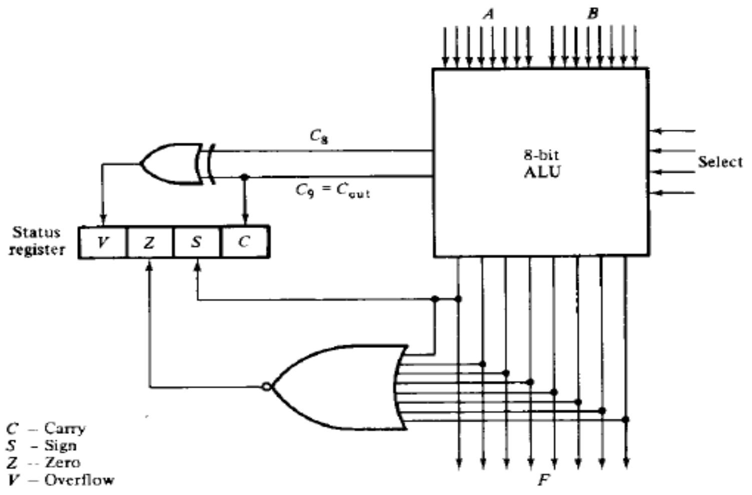
**Status Registers**

The relative magnitude of two numbers may be determined by subtracting one number from the other andthen checking certain bit conditions in the resultant difference. This status bit conditions (often calledcondition-code bits or flag bits) are stored in a status register.

Status register is a 4 bit register. The four bits are C (carry), Z (zero),S (sign) and V (overflow).These bits are set or cleared as a result of an operation performed in the ALU.

- Bit C is set if the output carry of an ALU is 1.
- Bit S is set to 1 if the highest order bit of the result in the output of the ALU is 1.
- Bit Z is set to 1 if the output of the ALU contains all O's.
- Bit V is set if the exclusive —OR of carries C8 and C9 is 1, and cleared otherwise. This is the condition for overflow when the numbers are in signed 2's complement representation. For an 8 bit ALU, V is set if the result is greater than 127 or less than -128.

After an ALU operation, status bits can be checked to determine the relationship that exist between the values of A and B.



If bit V is set after the addition two signed numbers, it indicates an overflow condition. If Z is set after anexclusive OR operation, it indicates that A=B. A single bit in A can be checked to determine if it is 0 or 1by masking all bits except the bit in question and then checking the Z status bit.

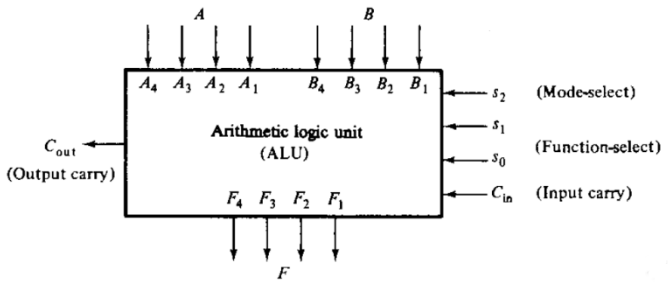
Relative magnitudes of A and B can be checked by compare operation. If A-B is performed for twounsigned binary numbers, relative magnitudes of A and B can be determined from the values transferred tothe C and Z bits. If Z=1,we knows that A=B, since A-B=0. If Z=0, then we know that A is not equal to B.

Similarly C=1 if A>=B and C=0 if A<B. The following table lists the various conditions

| Relation | Condition of status bits | Boolean function |
|---|---|---|
| $A > B$ | $C = 1$ and $Z = 0$ | $CZ'$ |
| $A \geqslant B$ | $C = 1$ | $C$ |
| $A < B$ | $C = 0$ | $C'$ |
| $A \leqslant B$ | $C = 0$ or $Z = 1$ | $C' + Z$ |
| $A = B$ | $Z = 1$ | $Z$ |
| $A \neq B$ | $Z = 0$ | $Z'$ |

## ARITHMETIC LOGIC UNIT

An arithmetic logic unit (ALU) is a multi operation, combinational-logic digital function. It can perform a set of basic arithmetic operations and a set of logic operations. The ALU has a number of selection lines to select a particular operation in the unit. The selection lines are decoded within the ALU so that k selection variables can specify up to $2^k$ distinct operations.The figure shows the block diagram of 4 bit ALU.

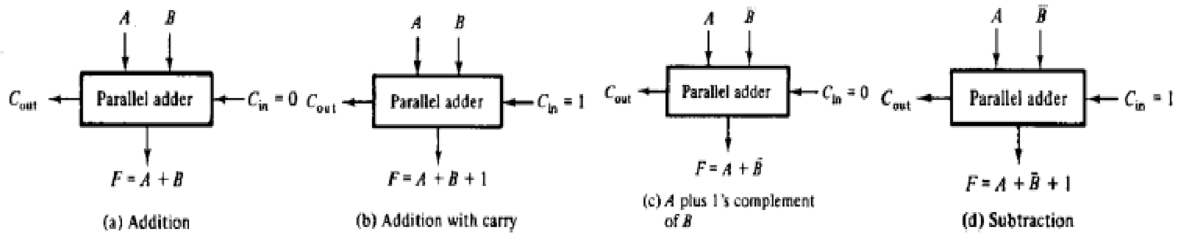


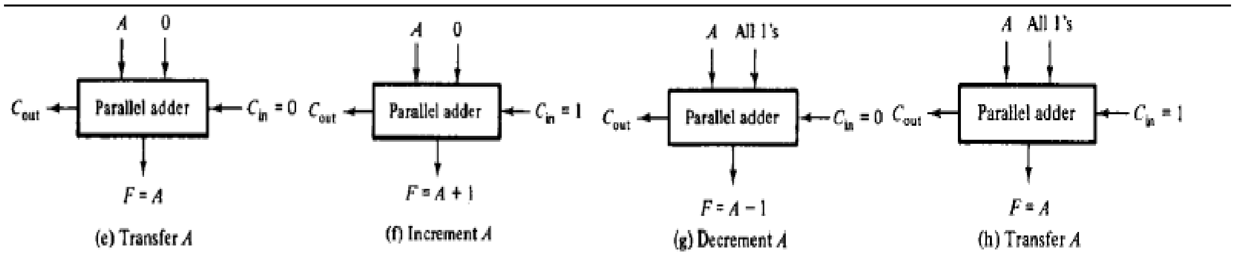
The design of a typical ALU will be carried out in three stages. First, the design of the arithmetic section will be undertaken. Second, the design of the logic section will be considered. Finally, the arithmetic section will be modified so that it can perform both arithmetic and logic operations.

### Design of Arithmetic Circuit

The basic component of the arithmetic section of an ALU is a parallel adder. A parallel adder is constructed with a number of full-adder circuits connected in cascade. By controlling the data inputs to the parallel adder, it is possible to obtain different types of arithmetic operations.

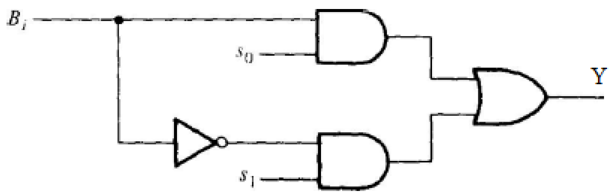
The figure demonstrates the arithmetic operations obtained when one set of inputs to a parallel adder is controlled externally. The number of bits in the parallel adder may be of any value. The input carry $C_{in}$ goes to the full-adder circuit in the least significant bit position. The output carry $C_{out}$ comes from the full-adder circuit in the most significant bit position.





The circuit that controls input B to provide the functions is called a **true/complement, one/zero** element. This circuit is illustrated in the following figure.

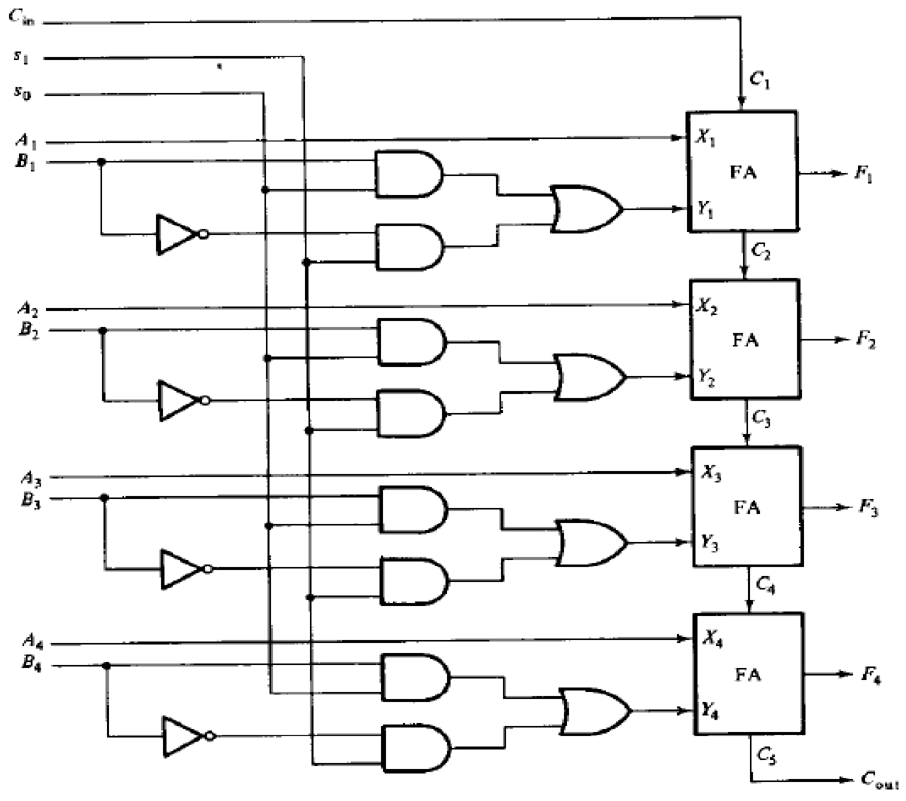
The **2 selection lines** s1 and s0 control the input of each B terminal.



| $S_0$ | $S_1$ | Y |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | $B'_1$ |
| 1 | 0 | $B_1$ |
| 1 | 1 | 1 |

The input A is applied directly to the 4-bit parallel adder and the input B is modified. The resultant arithmetic circuit is shown in below figure.

A **4-bit arithmetic circuit** that performs **8 arithmetic operations** is shown in following Figure.



The function table for the arithmetic circuit is given below.

| Function select | | | Y equals | Output equals | Function |
|---|---|---|---|---|---|
| $s_1$ | $s_0$ | $C_{in}$ | | | |
| 0 | 0 | 0 | 0 | $F = A$ | Transfer $A$ |
| 0 | 0 | 1 | 0 | $F = A + 1$ | Increment $A$ |
| 0 | 1 | 0 | $B$ | $F = A + B$ | Add $B$ to $A$ |
| 0 | 1 | 1 | $B$ | $F = A + B + 1$ | Add $B$ to $A$ plus 1 |
| 1 | 0 | 0 | $\bar{B}$ | $F = A + \bar{B}$ | Add 1's complement of $B$ to $A$ |
| 1 | 0 | 1 | $\bar{B}$ | $F = A + \bar{B} + 1$ | Add 2's complement of $B$ to $A$ |
| 1 | 1 | 0 | All 1's | $F = A - 1$ | Decrement $A$ |
| 1 | 1 | 1 | All 1's | $F = A$ | Transfer $A$ |

The **4 full-adder (FA)** circuits constitute the parallel adder.

- ➢ The carry into the **first stage** is the **input carry**.
- ➢ The carry out of the **fourth stage** is the **output carry**.
- ➢ **All other carries** are **connected internally** from one stage to the next.

The **selection variables** are **s1, s0,** and **Cin** .

- ➢ Variables **s1**and**s0 control** all of the **B inputs** to the full-adder circuits.

The **A inputs go directly** to the other inputs of the full adders.

The arithmetic operations implemented in the arithmetic circuit are listed in Table.

- ➢ The values of the **Y inputs** to the full-adder circuits are a **function of selection variabless1** and **s0**.
- ➢ **Adding** the value of **Y** in each case to the value of **A** plus the **Cin** value gives the arithmetic operation in each entry.
- ➢ The arithmetic circuit of above Figure needs a combinational circuit in each stage specified by the Boolean functions:

$$X_i = A_i$$
$$Y_i = B_i s_0 + B_i' s_1 \qquad i = 1, 2, \ldots, n$$

where **n** is the **number of bits** in the **arithmetic circuit**.

**Effect of Output Carry in the arithmetic circuit**

| Function select | | | Arithmetic function | $C_{out} = 1$ if | Comments |
|---|---|---|---|---|---|
| $s_1$ | $s_0$ | $C_{in}$ | | | |
| 0 | 0 | 0 | $F = A$ | | $C_{out}$ is always 0 |
| 0 | 0 | 1 | $F = A + 1$ | $A = 2^n - 1$ | $C_{out} = 1$ and $F = 0$ if $A = 2^n - 1$ |
| 0 | 1 | 0 | $F = A + B$ | $(A + B) > 2^n$ | Overflow occurs if $C_{out} = 1$ |
| 0 | 1 | 1 | $F = A + B + 1$ | $(A + B) > (2^n - 1)$ | Overflow occurs if $C_{out} = 1$ |
| 1 | 0 | 0 | $F = A - B - 1$ | $A > B$ | If $C_{out} = 0$, then $A < B$ and $F =$ 1's complement of $(B - A)$ |
| 1 | 0 | 1 | $F = A - B$ | $A > B$ | If $C_{out} = 0$, then $A < B$ and $F =$ 2's complement of $(B - A)$ |
| 1 | 1 | 0 | $F = A - 1$ | $A \neq 0$ | $C_{out} = 1$, except when $A = 0$ |
| 1 | 1 | 1 | $F = A$ | | $C_{out}$ is always 1 |

**Design of other Arithmetic Circuits**

The design of any arithmetic circuit can be done by following the procedure outlined in the previous example.

- Assuming that **all operations in the set can be generated through a parallel adder**

**Steps in design**

i. Start by obtaining a **function diagram**.

ii. Obtain a **function table** from the function diagram that relates the inputs of the full-adder circuit to the external inputs.

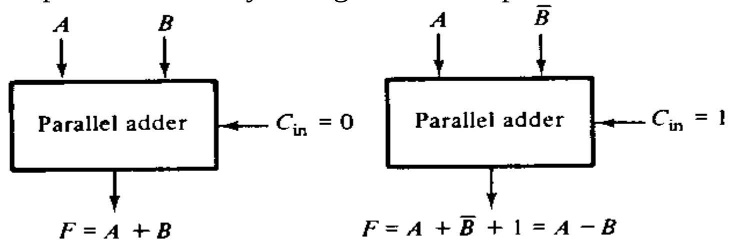
iii. Obtain the **combinational gates** from the function table that must be added to each full-adder stage.

This procedure is demonstrated in the following **example**.

**Qn) Design an adder/subtractor circuit with one selection variable s and two inputs A and B.**

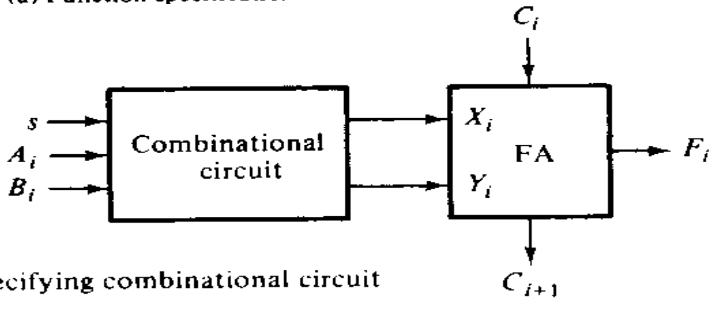
When s = 0, the circuit performs A+B

When s = 1, the circuit performs A-B by taking the 2's complement of B.



(a) Function specification

| $s$ | $X_i$ | $Y_i$ | $C_{in}$ |
|---|---|---|---|
| 0 | $A_i$ | $B_i$ | 0 |
| 1 | $A_i$ | $B_i'$ | 1 |



(b) Specifying combinational circuit

| $s$ | $A_i$ | $B_i$ | $X_i$ | $Y_i$ |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 1 |
| 0 | 1 | 0 | 1 | 0 |
| 0 | 1 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 | 1 |
| 1 | 0 | 1 | 0 | 0 |
| 1 | 1 | 0 | 1 | 1 |
| 1 | 1 | 1 | 1 | 0 |

$X_i = A_i$

$Y_i = B_i \oplus s$

$C_{in} = s$

(c) Truth table and simplified equations

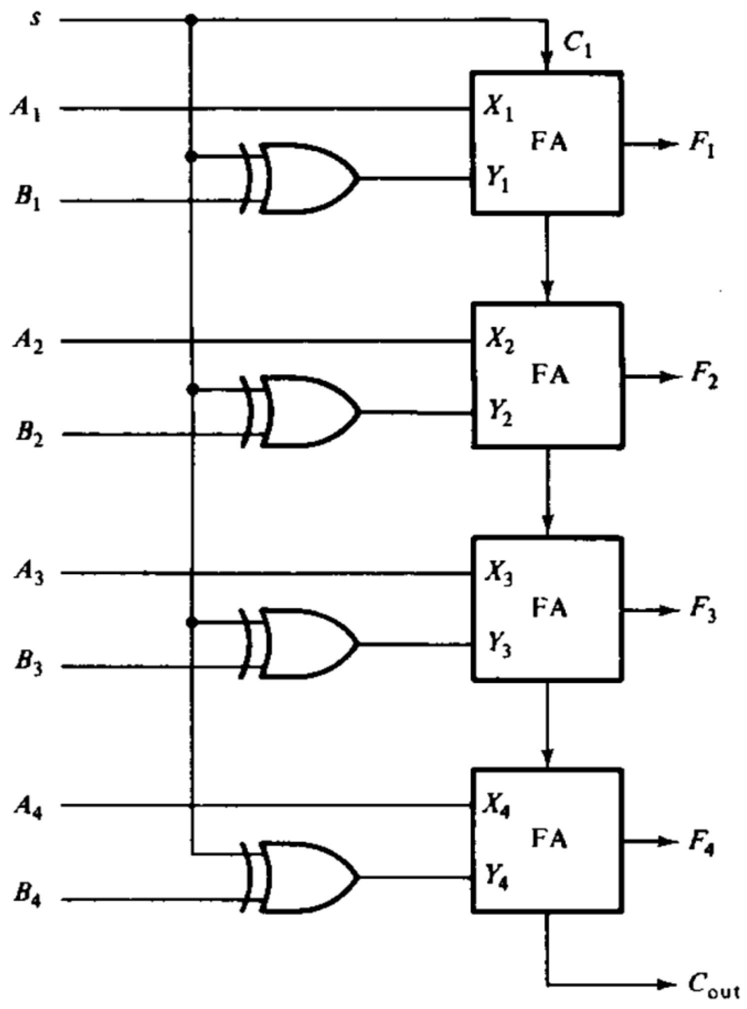


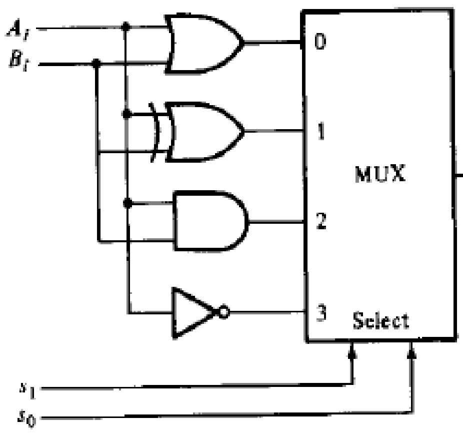
**Figure 9-10** 4-bit adder/subtractor circuit

**Design of Logic Circuit**

The logic microoperations manipulate the bits of the operands separately and treat each bit as a binary variable. The 16 logic operations can be generated in one circuit and selected by means of four selection lines. Since all logic operations can be obtained by means of AND, OR, and NOT (complement) operations, it may be more convenient to employ a logic circuit with just these operations.

For three operations, we need two selection variables. But two selection lines can select among four logic operations, so we choose also the exclusive-OR (XOR) function for the logic circuit to be designed in this and the next section.

The simplest and most straight forward way to design a logic circuit is shown in figure given below. The diagram shows one typical stage designated by subscript i. The circuit must be repeated n times for an n-bit logic circuit.



(a) Logic diagram

| $s_1$ | $s_0$ | Output | Operation |
|---|---|---|---|
| 0 | 0 | $F_i = A_i + B_i$ | OR |
| 0 | 1 | $F_i = A_i \oplus B_i$ | XOR |
| 1 | 0 | $F_i = A_iB_i$ | AND |
| 1 | 1 | $F_i = A_i'$ | NOT |

(b) Function table

The four gate generate four logic operations OR, XOR, AND, and NOT. The two selection variables in the multiplexer select one of the gates for the output. The function table lists the output logic generated as a function of the two selection variables.

The logic circuit can be combined with the arithmetic circuit to produce one arithmetic logic unit. Selection variables $S_1$ and $S_0$ can be made common to both sections provided we use a third selection variable, $S_2$, to differentiate between the two. This configuration is illustrated in the below figure.

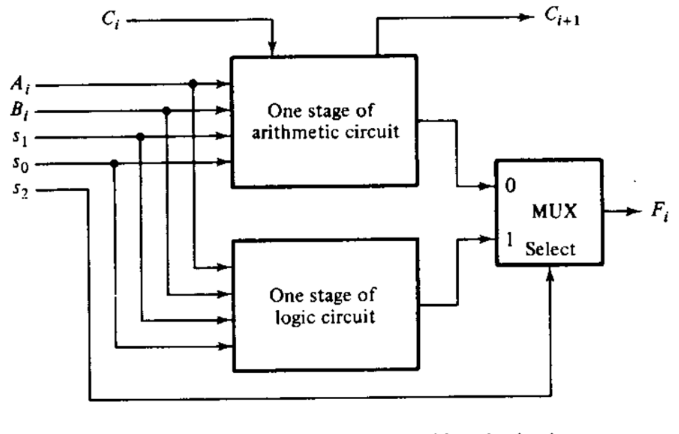


**Figure 9-12** Combining logic and arithmetic circuits

The outputs of the logic and arithmetic circuits in each stage go through a multiplexer with selection variable $S_2$.

When $S_2 = 0$, the arithmetic output is selected,

when $S_2 = 1$, the logic output is selected.

Although the two circuits can be combined in this manner, this is not the best way to design an ALU.A more efficient ALU can be obtained if we investigate the possibility of generating logic operations in an already available arithmetic circuit. This can be done by inhibiting all input carries into the full-adder circuits of the parallel adder. Consider the Boolean function that generates the output sum in a full-adder circuit:

$$F=X \oplus Y \oplus C_{in}$$

The input carry $C_{in}$ in each stage can be made to be equal to 0 when a selection variable$S_2$is equal to 1. The result would be:

$$F=X \oplus Y$$

This expression is valid because of the property of the X-OR operation:

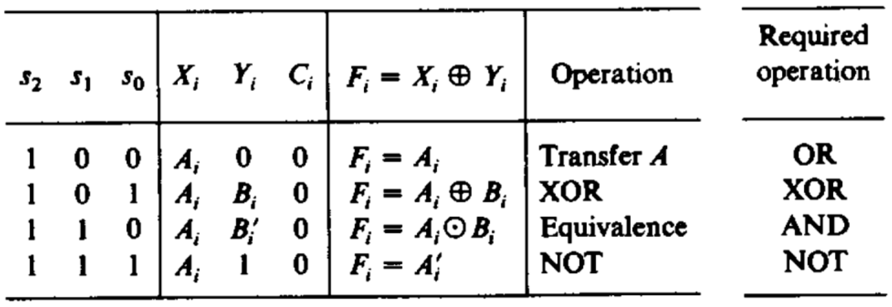
$$X\oplus 0 = X$$

Thus, with the input carry to each stage equal to 0, the full-adder circuits generate the exclusive-OR operation.

Now refer the figure of arithmetic unit circuit.

- The value of **Yi** can be selected by means of the **two selection variables** to be equal to either **0, Bi, Bi', or l.**
- The value of **Xi** is always equal to input **Ai**. Table given below shows the four logic operations obtained when **s2=0**
- This selection variable forces Ci to be equal to 0 whiles1 and s0 choose a particular value for Yi.

**TABLE 9-3 Logic operations in one stage of arithmetic circuit**



| $s_2$ | $s_1$ | $s_0$ | $X_i$ | $Y_i$ | $C_i$ | $F_i = X_i \oplus Y_i$ | Operation | Required operation |
|---|---|---|---|---|---|---|---|---|
| 1 | 0 | 0 | $A_i$ | 0 | 0 | $F_i = A_i$ | Transfer $A$ | OR |
| 1 | 0 | 1 | $A_i$ | $B_i$ | 0 | $F_i = A_i \oplus B_i$ | XOR | XOR |
| 1 | 1 | 0 | $A_i$ | $B_i'$ | 0 | $F_i = A_i \odot B_i$ | Equivalence | AND |
| 1 | 1 | 1 | $A_i$ | 1 | 0 | $F_i = A_i'$ | NOT | NOT |

- The 4 logic operations obtained by this configuration are transfer, exclusive-OR, equivalence, and complement.
  The third entry is the equivalence operation because:
  $$A_i \oplus B_i' = A_iB_i + A_i'B_i' = A_i \odot B_i$$
  The last entry in the table is the **NOT** or **complement operation** because:
  $$A_i \oplus 1 = A_i'$$
  The table has one more column which lists the four logic operations we want to include in the ALU.
- Two of these operations, **XOR** and **NOT**, are **already available**.
- It is possible to modify the arithmetic circuit further so that it will generate the logic functions **OR** and **AND instead of** the **transfer** and **equivalence** functions.

**Design of Arithmetic Logic Unit**

A basic ALU with eight arithmetic operations and four logic operations can be designed with the details already have.
We already have

- Three selection variables s2, s1, and s0to select eight different operations
- The input carry Cin is used to select four additional arithmetic operations.
- With s2 = 0, selection variables s1and s0 together with Cin will select the eight arithmetic operations
- With s2 = l, selection variables s1 and s0 will select the four logic operations OR, XOR, AND, and NOT.

The design of an ALU is a combinational-logic problem.

- We can design one stage of the ALU and then duplicate it for the number of stages required.
- There are **six inputs** to each stage: Ai, Bi, Ci,s2, s1 and s0.
- There are **two outputs** in each stage: output Fi and the carry out Ci+1

One can formulate a truth table with **64 entries** and simplify the **two output functions**.

Here we choose to employ an alternate procedure that uses the availability of a parallel adder.

***Design steps of ALU***

1. Design the arithmetic section independent of the logic section.
2. Determine the logic operations obtained from the arithmetic circuit in step 1, assuming that the input carries to all stages are 0.
3. Modify the arithmetic circuit to obtain the required logic operations.

   - ➢ The solution to the first design step is shown in Arithmetic unit design.
   - ➢ The solution to the second design step is presented in Logic unit design.
   - ➢ The solution of the third step is carried out below.

From **Table**, we see that

- ➢ When **s2= l**, the input carry Ci in each stage must be 0. **Ci=0.**
- ➢ With **s1,s0 = 00**, each stage as it stands generates the function **Fi = Ai**. [Since **Xi=Ai** and **Yi=0;** refer diagram]

To change the output to an **OR** operation, we must change the input to each full-adder circuit from **Ai to Ai+Bi**. This can be accomplished by **OR**ing **Bi** and **Ai** when **s2s1s0 = 100**.

The other selection variables that give an undesirable output occur when **s2s1s0= 110**. The unit as it stands generates an output $F_i = A_i \odot B_i,$ but we want togenerate the AND operation **Fi = Ai.Bi**.

Let us investigate the possibility of **OR**ing each input **Ai** with some Boolean function **Ki,** to change Xi. Since we can't change Yi.

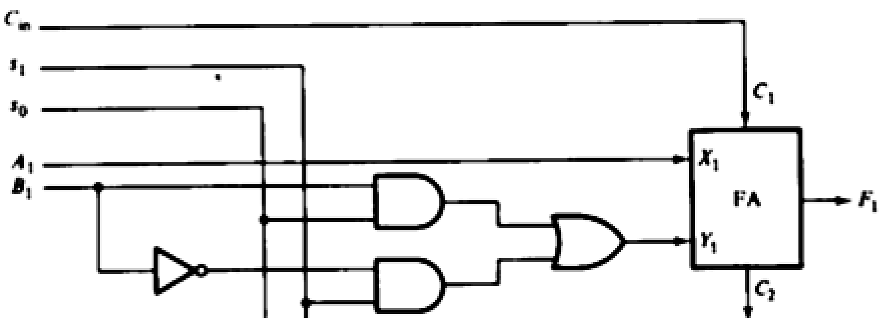


TABLE 9-3 Logic operations in one stage of arithmetic circuit

| $s_2$ | $s_1$ | $s_0$ | $X_i$ | $Y_i$ | $C_i$ | $F_i = X_i \oplus Y_i$ | Operation | Required operation |
|---|---|---|---|---|---|---|---|---|
| 1 | 0 | 0 | $A_i$ | 0 | 0 | $F_i = A_i$ | Transfer $A$ | OR |
| 1 | 0 | 1 | $A_i$ | $B_i$ | 0 | $F_i = A_i \oplus B_i$ | XOR | XOR |
| 1 | 1 | 0 | $A_i$ | $B_i'$ | 0 | $F_i = A_i \odot B_i$ | Equivalence | AND |
| 1 | 1 | 1 | $A_i$ | 1 | 0 | $F_i = A_i'$ | NOT | NOT |

The function so obtained is then used for Xi when **s2s1s0= 110**:

$$F_i = X_i \oplus Y_i = (A_i + K_i) \oplus B_i' = A_iB_i + K_iB_i + A_i'K_i'B_i'$$

Careful inspection of the result reveals that if the variable Ki = B,' , we obtain an output:

$$F_i = A_iB_i + B_i'B_i + A_iB_iB_i' = A_iB_i$$

Two terms are equal to 0 because **Bi.Bi' = 0**. The result obtained is the **AND** operation as required.
The conclusion is that, if **Ai** is **OR**ed with **Bi'** when **s2s1s0= 110**, the output will generate the **AND** operation.

The final ALU is shown in figure below Only the first two stages are drawn, but the diagram can be easily extended to more stages. The inputs to each full-adder circuit are specified by the Boolean functions:

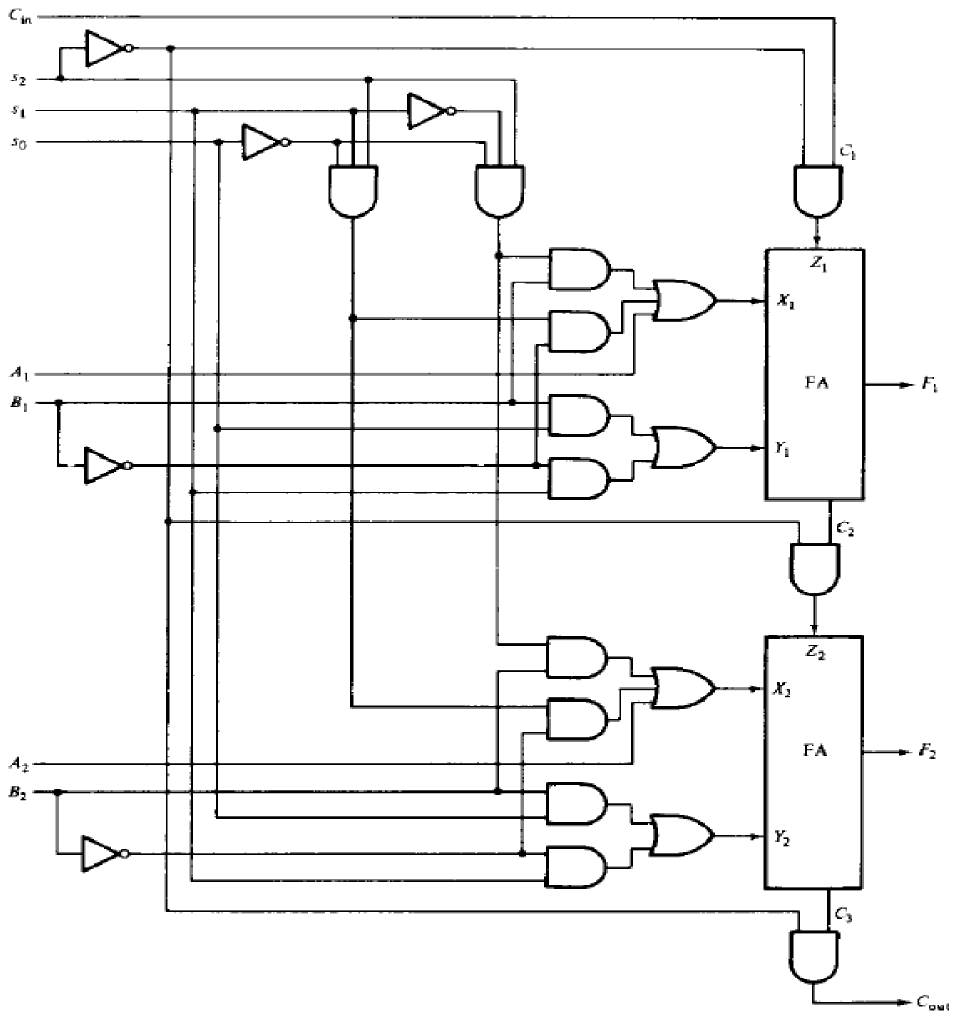
$X_i = A_i + S_2 S_1' S_0' B_i + S_2 S_1 S_0' B_i$
$Y_i = S_0 B_i + S_1 B_i'$
$Z_i = S_2' C_i$

When $S_2 = 0$, the three functions reduce to:

$X_i = A_i$
$Y_i = S_0 B_i + S_1 B_i'$
$Z_i = C_i$

Which are the function for the arithmetic circuit. The logical operations are generated when $S_2 = 1$. For $S_2\ S_1\ S_0 = 1\ 0\ 1$ or $1\ 1\ 1$, the function reduce to:

$$X_i = A_i \quad Y_i = S_0 B_i + S_1 B_i' \qquad Z_i = 0$$



The function table for the Arithmetic and Logic Unit is shown below. The 12 operations generated in the ALU are summarized in Table.

| Selection | | | | Output | Function |
|---|---|---|---|---|---|
| $s_2$ | $s_1$ | $s_0$ | $C_{in}$ | | |
| 0 | 0 | 0 | 0 | $F = A$ | Transfer $A$ |
| 0 | 0 | 0 | 1 | $F = A + 1$ | Increment $A$ |
| 0 | 0 | 1 | 0 | $F = A + B$ | Addition |
| 0 | 0 | 1 | 1 | $F = A + B + 1$ | Add with carry |
| 0 | 1 | 0 | 0 | $F = A - B - 1$ | Subtract with borrow |
| 0 | 1 | 0 | 1 | $F = A - B$ | Subtraction |
| 0 | 1 | 1 | 0 | $F = A - 1$ | Decrement $A$ |
| 0 | 1 | 1 | 1 | $F = A$ | Transfer $A$ |
| 1 | 0 | 0 | $X$ | $F = A \vee B$ | OR |
| 1 | 0 | 1 | $X$ | $F = A \oplus B$ | XOR |
| 1 | 1 | 0 | $X$ | $F = A \wedge B$ | AND |
| 1 | 1 | 1 | $X$ | $F = \bar{A}$ | Complement $A$ |

The particular function is selected through s2,sl, s0, and Cin.

The arithmetic operations are identical to the ones listed for the arithmetic circuit.

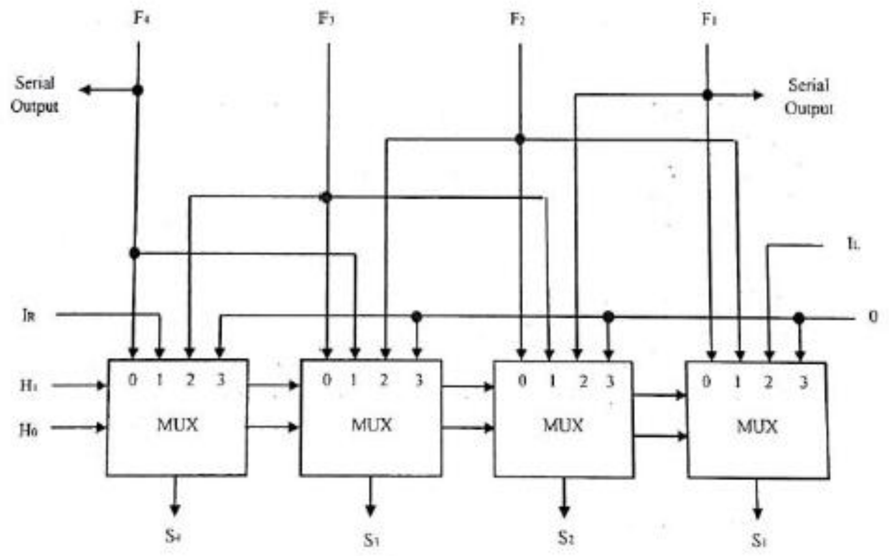
The value of Cin for the four logic functions has no effect on the operation of the unit and those entries are marked with don't-care X 's.

**Design of Combinational Logic Shifter**

The shift unit attached to the processor transfers the output of the ALU onto the output bus. Shifter may function in four different ways.

1. The shifter may transfer the information directly without a shift.
2. The shifter may shift the information to the right.
3. The shifter may shift the information to the left.
4. In some cases no transfer is made from ALU to the output bus.

A shifter is a bi-directional shift-register with parallel load. The information from ALU can be transferred to the register in parallel and then shifted to the right or left. In this configuration, a clock pulse is needed for the transfer to the shift register, and another pulse is needed for the shift. Another clock pulse may also in need of when information is passed from shift register to destination register.



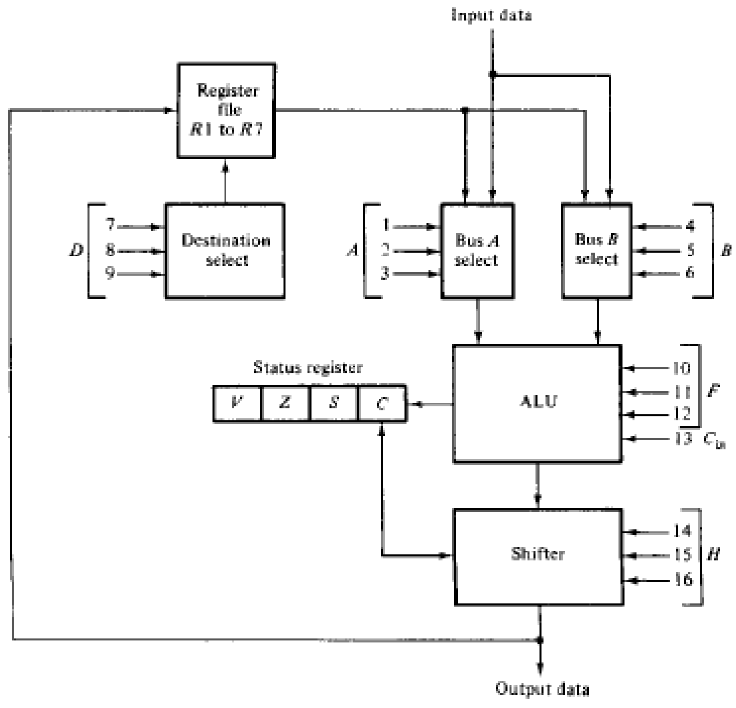
The number of clock pulses may reduce if the shifter is implemented with a combinational circuit. A combinational—logic shifter can be constructed with multiplexers. The above figure will show the same.

Shifter operation can be selected by two variables $H_1$ $H_0$

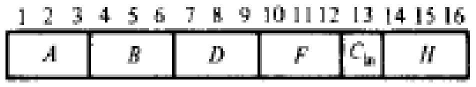
- If $H_1$ $H_0$ = 0 0 No shift is executed and the signal from F go directly to S lines
- If $H_1$ $H_0$ = 0 1 Shift Right is executed
- If $H_1$ $H_0$ = 1 0 Shift Left is executed
- If $H_1$ $H_0$ = 1 1 No operations

## PROCESSOR UNIT

A block diagram of a processor unit is shown in figure. It consists of seven registers R1 through R7 and a status register. The outputs of the seven registers go through two multiplexers to select the inputs to the ALU.



(a) Block diagram



(b) Control word

**Figure 9-16** Processor unit with control variables

Input data from an external source are also selected by the same multiplexers. The output of the ALU goes through a shifter and then to a set of external output terminals. The output from the shifter can be transferred to any one of the registers or to an external destination.

There are **16 selection variables** in the unit, and their function is specified by **a Control Word**. The 16-bit control word, when applied to the selection variables in the processor, specifies a given microoperation. The Control Word is partitioned into 6 fields, with each field designated by a letter name. All fields, except Cin, have a code of three bits.

The functions of all selection variables are specified in table below.

| Binary code | Function of selection variables | | | | | |
|---|---|---|---|---|---|---|
| | $A$ | $B$ | $D$ | $F$ with $C_{in} = 0$ | $F$ with $C_{in} = 1$ | $H$ |
| 0 0 0 | Input data | Input data | None | $A, C \leftarrow 0$ | $A + 1$ | No shift |
| 0 0 1 | $R1$ | $R1$ | $R1$ | $A + B$ | $A + B + 1$ | Shift-right, $I_R = 0$ |
| 0 1 0 | $R2$ | $R2$ | $R2$ | $A - B - 1$ | $A - B$ | Shift-left, $I_L = 0$ |
| 0 1 1 | $R3$ | $R3$ | $R3$ | $A - 1$ | $A, C \leftarrow 1$ | 0's to output bus |
| 1 0 0 | $R4$ | $R4$ | $R4$ | $A \vee B$ | — | — |
| 1 0 1 | $R5$ | $R5$ | $R5$ | $A \oplus B$ | — | Circulate-right with $C$ |
| 1 1 0 | $R6$ | $R6$ | $R6$ | $A \wedge B$ | — | Circulate-left with $C$ |
| 1 1 1 | $R7$ | $R7$ | $R7$ | $\bar{A}$ | — | — |

TABLE 9-9 Examples of microoperations for processor

| | Control word | | | | | | |
|---|---|---|---|---|---|---|---|
| Microoperation | $A$ | $B$ | $D$ | $F$ | $C_{in}$ | $H$ | Function |
| $R1 \leftarrow R1 - R2$ | 001 | 010 | 001 | 010 | 1 | 000 | Subtract $R2$ from $R1$ |
| $R3 - R4$ | 011 | 100 | 000 | 010 | 1 | 000 | Compare $R3$ and $R4$ |
| $R5 \leftarrow R4$ | 100 | 000 | 101 | 000 | 0 | 000 | Transfer $R4$ to $R5$ |
| $R6 \leftarrow$ Input | 000 | 000 | 110 | 000 | 0 | 000 | Input data to $R6$ |
| Output $\leftarrow R7$ | 111 | 000 | 000 | 000 | 0 | 000 | Output data from $R7$ |
| $R1 \leftarrow R1, C \leftarrow 0$ | 001 | 000 | 001 | 000 | 0 | 000 | Clear carry bit $C$ |
| $R3 \leftarrow$ shl $R3$ | 011 | 011 | 011 | 100 | 0 | 010 | Shift-left $R3$ with $I_L = 0$ |
| $R1 \leftarrow$ crc $R1$ | 001 | 001 | 001 | 100 | 0 | 101 | Circulate-right $R1$ with carry |
| $R2 \leftarrow 0$ | 000 | 000 | 010 | 000 | 0 | 011 | Clear $R2$ |

The **3 bits of A** select a **Source Register** for the input to **left side of the ALU**.
- The **B field** is the same, but it selects the source information for the **right input of the ALU**.
- The **D field** selects a **Destination Register**.
- The **F field**, together with the bit in Cin, **selects a Function** for the ALU.
- The**H field** selects the **type of Shift** in the shifter unit.

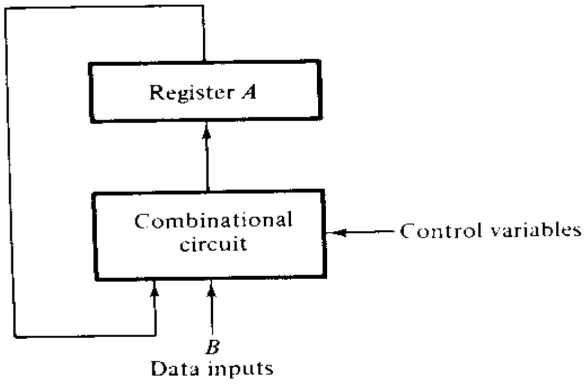
The 3-bit binary code listed in the table specifies the code for each of the five fields A, B, D, input data, F, and H. The register selected by A, B, and D is the one whose decimal number is equivalent to the binary number in the code. When the A or B field is 000, the corresponding multiplexer selects the input data. When D = 000, no destination register is selected.

The three bits in the F field, together with the input carry $C_{in}$, provide the 12 operations of the ALU as specified in above table. Note that there are two possibilities for F = A. In one case the carry bit C is cleared, and in the other case it is set to 1.

A **control word of 16 bits** is needed to **specify a microoperation for the processor unit**.
- The most efficient way to is to store them in a memory unit which functions as a **control memory**.
- The **sequence of control words** is then **read** from the control memory, **one word at a time**, to initiate the desired **sequence of microoperations**.
- This type of **control organization** is called **Microprogramming**.

## DESIGN OF ACCUMULATOR

Some processor units distinguish one register from all others and call it an accumulator register. The block diagram of an accumulator that forms a sequential circuit is shown in figure below.



The A register and the associated combinational circuit constitutes a sequential circuit. The combinational circuit replaces the ALU but cannot be separated from the register, since it is only the combinational-circuit part of a sequential circuit. The A register is referred to as the accumulator register and is sometimes denoted by the symbol AC. Here, accumulator refers to both the A register and its associated combinational circuit.

The external inputs to the accumulator are the data inputs from B and the control variables that determine the micro operations for the register. The next state of register A is a function of its present state and of the external inputs.

Accumulator can also perform data processing operations. Total of nine operations is considered here for the design of accumulator circuit.

| Control variable | Microoperation | Name |
|---|---|---|
| $p_1$ | $A \leftarrow A + B$ | Add |
| $p_2$ | $A \leftarrow 0$ | Clear |
| $p_3$ | $A \leftarrow \bar{A}$ | Complement |
| $p_4$ | $A \leftarrow A \wedge B$ | AND |
| $p_5$ | $A \leftarrow A \vee B$ | OR |
| $p_6$ | $A \leftarrow A \oplus B$ | Exclusive-OR |
| $p_7$ | $A \leftarrow$ shr $A$ | Shift-right |
| $p_8$ | $A \leftarrow$ shl $A$ | Shift-left |
| $p_9$ | $A \leftarrow A + 1$ | Increment |
| | If $(A = 0)$ then $(Z = 1)$ | Check for zero |

In all listed microoperations A is the source register. B register is used as the second source register. The destination register is also accumulator register itself. For a **complete accumulator** there will be n stages.

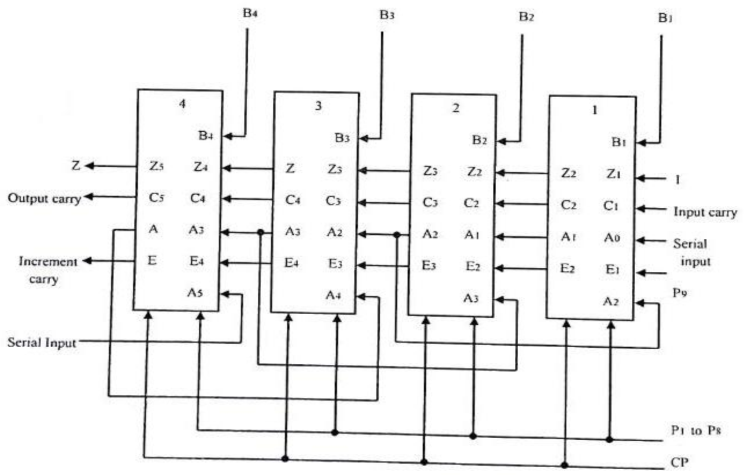


Fig: 4 bit accumulator constructed with 4 bits

The inputs and outputs of each stage can be connected in cascade to form a complete accumulator. Here we are discussing the design of a 4 bit accumulator. The number on top of each block represents the bit position.

All blocks receive 8 control variables $P_1$ to $P_8$ and the clock pulses from CP. The other six inputs and four outputs are same as with the typical stage. The zero detect chain is obtained by connecting the z variables in cascade, with the first block receiving a binary constant I . The last stage produces the zero detect variable Z.

Total number of terminals in the 4 bit accumulator is 25, including terminals for the A outputs. Incorporating two more terminals for power supply, the circuit can be enclosed within one IC package having 27 or 28 pins.

The number of terminals for the control variable can be reduced from 9 to 4 if a decoder is inserted in the IC. In such cases, IC pin count is also reduced to 22 and the accumulator can be extended to 16 microoperations without adding external pins (That is, with 4 bits we can identify 16 operations).