5.2.5 Design of Accumulator

Some processors distinguish one register from others and it is known as accumulator register. It is a multipurpose register capable of performing not only the add microoperation but many other microoperations as well. The organization of a processor unit with an accumulator register is shown below.

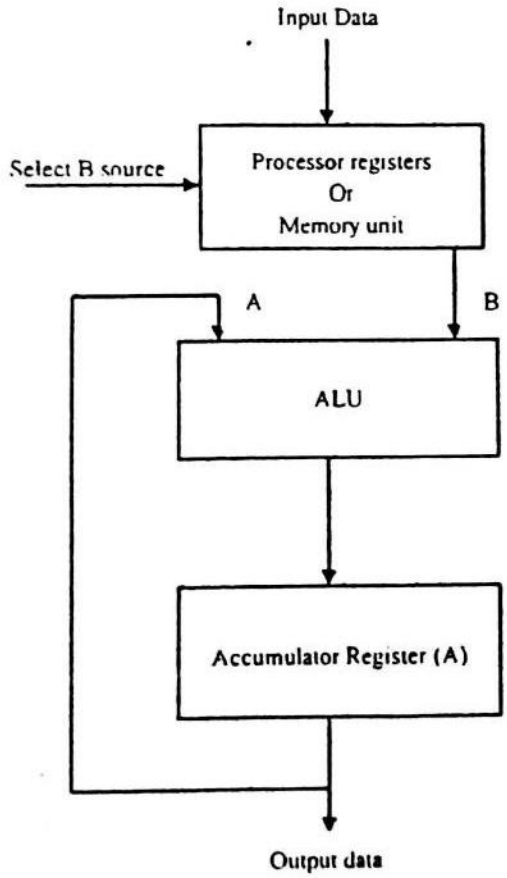


Fig 5.16: Processor with an accumulator register

The ALU associated with the register may be constructed as a combinational circuit. In this configuration, the accumulator register is essentially a bidirectional shift register with parallel load which is connected to the ALU. There is also a feedback connection from the output of accumulator register to one of the inputs in ALU. Because of this feedback connection, the accumulator register and its associated logic, when taken as one unit.

constitute a sequential circuit. The block diagram of the accumulator that forms as sequential circuit is shown below:

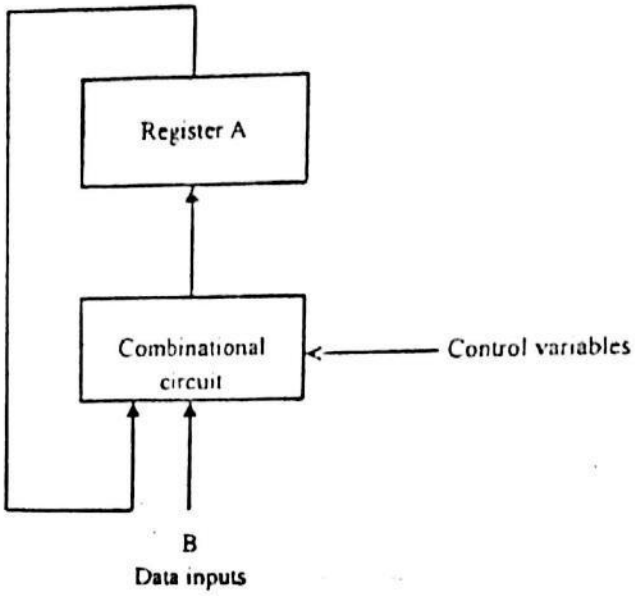


Fig 5.17: Block diagram of accumulator

The A register and associated combinational circuit constitute a sequential circuit. Here the combinational circuit replaces ALU, but it cannot be separated from the register. The accumulator register is denoted by A or AC (A register and associated combinational circuit).The external inputs to the accumulator are data inputs from B and the control variables (which will specify the microoperation to be executed).

Accumulator can also perform data processing operations .Total of nine operations are considered here for the design of accumulator circuit. These operations are described below.

Control variable	F with C_w=0	F with C_w=1
P1	A←A+B	Add
P2	A←0	Clear
P3	A←$\overline{A}$	Complement
P4	A←A ∧ B	AND
P5	A←AVB	OR
P6	A←A⊕B	Exclusive-OR
P7	A←shr A	Shift-right
P8	A←shl A	Shift-left
P9	A←A+1	Increment
Z bit	If(A=0)then(Z=1)	Check for zero

Table 5.9: List of microoperation for an accumulator

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. The nine control variables are considered as inputs to the sequential circuit. These variables are mutually exclusive. That means, only one variable must be enabled when a clock pulse occurs. The last entry in the table is a conditional control statement. It produces a binary 1 in the output variable Z when the content of register A is 0.

Design Procedure

Accumulator consists of n stages and n flip flops, numbered as A_1, A_2, A_3....An from right to left. It is convenient to partition the accumulator into n similar stages, with each stage consisting of one flip flop denoted by Ai, and one data input denoted by B_i, and the combinational logic associated with the flip flop. Each stage A_i is interconnected with neighbouring stage A_{i-1} on its right and A_{i+1} on its left. The register will be designed using JK type flip flops.

Each control variable P_i initiates a microoperation. Here nine microoperations are there by selecting control variables from P_1 to P_9. Accumulator is partitioned into n stages and each stage is again partitioned into those circuits that are needed for each microoperation. In the design procedure, we are designing various pieces separately and combine to form a one stage accumulator and then combine the stages to form a complete accumulator.

- **Add B to A (P_1)**

  Add microoperation is initiated when control variable P_1 is 1. To perform addition operation, accumulator can use a parallel adder composed of full adders. The full adder in each stage I will accept the input and (present state of A_i and data input B_i) and a previous carry bit C_i. Sum bit is transferred to flip flop A_i and output carries C_{i+1} is transferred to the next stage as input carry of that stage.

  The state table of a full adder, when considered as a sequential circuit is shown below.

Present State	Input		Next State	Flip-flop inputs		Output
A_i	B_i	C_i	A_i	JA_i	KA_i	C_{i+1}
0	0	0	0	0	X	0
0	0	1	1	1	X	0
0	1	0	1	1	X	0
0	1	1	0	0	X	1
1	0	0	1	X	0	0
1	0	1	0	X	1	1
1	1	0	0	X	1	1
1	1	1	1	X	0	1

Table 5.10: Excitation table for add microoperation

Columns of the table can be described as follows:

- Present state- the value of flip flop Ai before the clock pulse
- Next state- the value of flip flop Ai after the clock pulse (here, it will be determined by sum produced with inputs A_i, B_i and C_i)
- C_i - input carry
- C_{i+1} - output carry
- Flip flop inputs - shows the excitation input for the JK flip flop.

The excitation table of JK flip flop is shown below for reference.

Present state	Next State	J	K
0	0	0	X
0	1	1	X
1	0	X	1
1	1	X	0

Table 5.11: List of microoperation for an accumulator

According to these values the above flip flop inputs are set. The flip flop input functions and the Boolean functions for the output are simplified in the maps as shown in fig 5.19.

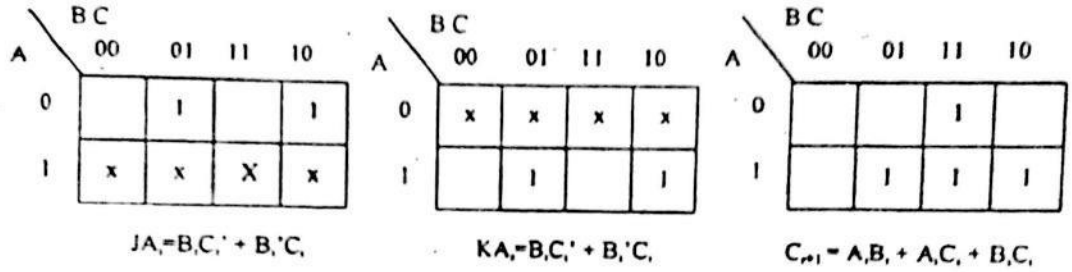


Fig 5.18: Map simplification for Add Microoperation

The J input of flip-flop Ai, designated by JA_i and the K input of flip flop A_i, designated by KA_i, do not include the control variable p1. These two equations should affect the flip flop only when P_1 is enabled. Therefore, they should be ANDed with control variable P_1. Then the equation becomes,

$$JA_i = B_iC_i'P_1 + B_i'C_iP_1$$

$$KA_i = B_iC_i'P_1 + B_i'C_iP_1$$

$$C_{i+1} = A_iB_i + A_iC_i + B_iC_i$$

- **Clear (P_2)**

Control variable P_2 clears all flip flops in register A. To cause this transition in a JK flip flop, we need only apply control variable P_2 to the K input of the flip flop. The J input will be assumed to be 0 if nothing is applied on it. The input functions can be written as:

$$JA_i = 0$$

$$KA_i = P_2$$

- **Complement(P_3)**

To cause this transition in a JK flip flop we need to apply P_3 to both J and K inputs.

$$JA_i = P_3$$

$$KA_i = P_3$$

- **AND (P_4)**

This microoperation is initiated with control variable P_4.This operation performs the logic AND operation between A_i and B_i and transfers the result to A_i. The excitation table for this operation is as shown below.

Present State	Input	Next State	Flip-flop Inputs	
A_i	B_i	A_i	JA_i	KA_i
0	0	0	0	X
0	1	0	0	X
1	0	0	X	1
1	1	1	X	0

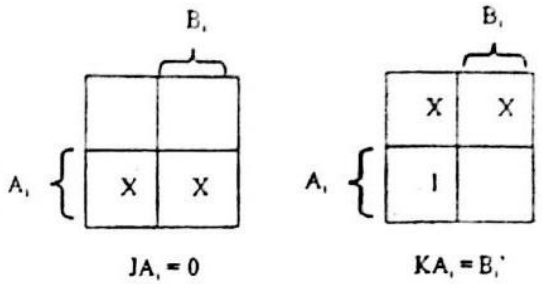


Fig 5.19: Excitation table for AND microoperations

The next state of A_i will be 1 only when the present state of A_i and data input B_i is 1s. The flip flop input functions can be simplified with the maps and the equations can be written as:

$$JA_i = 0$$

$$KA_i = B_i'$$

By including the control variable p4, the equation can be rewritten as:

$$JA_i = 0$$

$$KA_i = B_i'P_4$$

- **OR (P_5)**

Control variable P_5 initiates the logic OR operation between A_i and B_i. The result is transferred to A_i. The excitation table for this operation is as shown below.

Present State	Input	Next State	Flip-flop inputs	
A_i	B_i	A_i	JA_i	KA_i
0	0	0	0	X
0	1	1	1	X
1	0	1	X	0
1	1	1	X	0

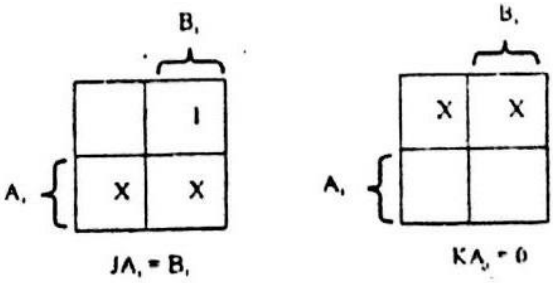


Fig 5.20: Excitation table for OR microoperations

The simplified equations in the maps dictate that the J input be enabled when $B_i=1$. When $B_i=0$, the present state and next state of Ai are the same. When $B_i=1$, the J input is enabled and the next state of A_i becomes 1. Input functions for the OR microoperation are:

$$JA_i = B_iP_5$$

$$KA_i = 0$$

- **Exclusive-OR (P_6)**

Control variable P_6 initiates the logic Exclusive-OR operation between A_i and B_i. The result is transferred to A_i. The excitation table and map simplification is as shown below.

Present State	Input	Next State	Flip-flop Inputs	
A_i	B_i	A_i	JA_i	KA_i
0	0	0	0	X
0	1	1	1	X
1	0	1	X	0
1	1	0	X	1

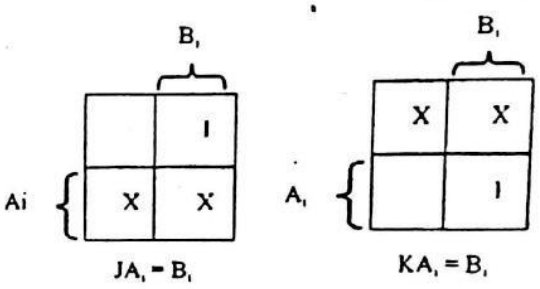


Fig 5.21: Excitation table for Exclusive-OR microoperations

The flip flop input functions are written as:

$$JA_i = B_iP_6$$

$$KA_i = B_iP_6$$

- **Shift-right (P_7)**

Control variable P_7 initiates the shift operation of A_i register one bit to the right. That is, the value of flip flop A_{i+1} is transferred to flip flop A_i. The flip flop input functions can be written as:

$$JA_i = A_{i+1}P_7$$

$$KA_i = A'_{i+1}P_7$$

- **Shift-left (P_8)**

Control variable P_8 initiates the shift operation of A_i register one bit to the left. That is, the value of flip flop A_{i-1} is transferred to flip flop A_i. The flip flop input functions can be written as:

$$JA_i = A_{i-1}P_8$$

$$KA_i = A'_{i-1}P_8$$

- **Increment (P_9)**

These operations increment the content of A register by one. The register behaves likes a synchronous binary counter with P_9 enabling the count. A 3 bit synchronous counter is shown in the following figure.

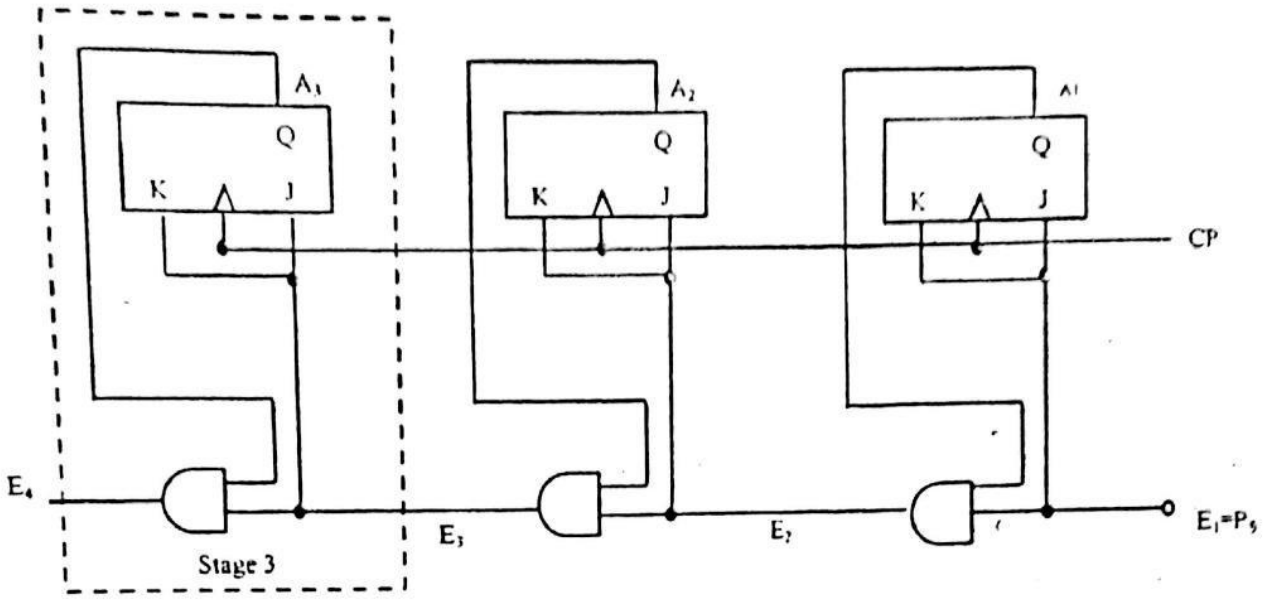


Fig 5.22: 3-bit synchronous binary counter

From the figure it can see that each stage is complemented when an input carry $E_i=1$. Each stage is generating an output carry E_{i+1} that is fed to the next stage on its left. The first stage is an exception, since it is complemented with the count-enable P_9. The Boolean function for a typical stage can be written as:

$$JA_i = E_i$$

$$KA_i = E_i$$

$$E_{i+1} = E_i A_i \qquad i=1,2,\ldots,n$$

$$E_1 = P_9$$

Input carry to the first stage of counter is E_1. It must be equal to the control variable p9 which enables the count. The input carry E_i is used to complement flip flop A_i. The input carry is ANDed with A_i to generate a carry for the next stage.

- **Check for Zero (Z)**

Variable Z is an output from the accumulator. This variable can be used to indicate a zero content in the A register. All the flip flops in the accumulator is cleared Z variable will be set to 1. When a flip flop is cleared, its complement output Q' is equal to 1.

The following figure shows the first three stages of the accumulator that checks for zero content. Each stage generates a variable Z_{i+1} by ANDing the complement output of A_i to an input variable Z_i. In this way, a chain of AND gates through all stages will indicate if all flips are cleared.

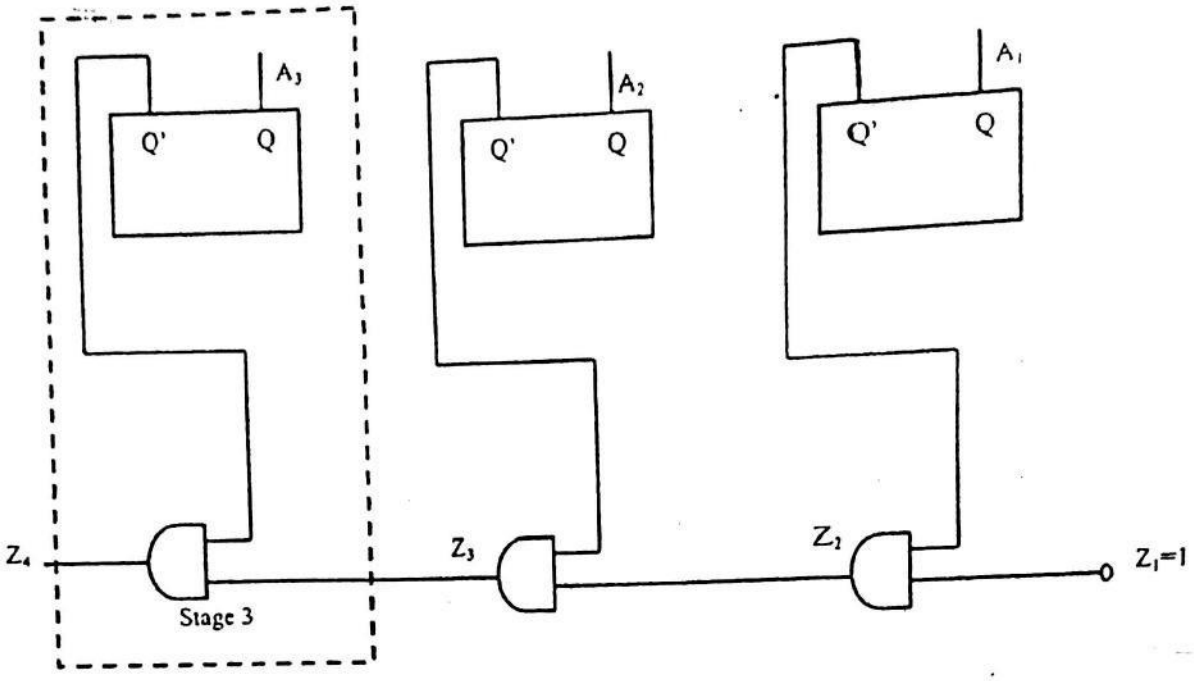


Fig 5.23: Chain of AND gates for checking the zero content of a register

The Boolean function for a typical stage can be expressed as:

$$Z_{i+1}=Z_iA_i' \qquad i=1,2,\ldots,n$$

$$Z_1=1$$

$$Z_{n+1}=Z$$

Variable Z becomes 1 if the output signal from the last stage Z_{n+1} is 1.

One stage of Accumulator

In the earlier sections we have derived the logic circuits for each individual microoperation that can be performed by the Accumulator. Now, we can combine them to all to form one stage of the Accumulator circuit. Combining all the input functions for the J and K inputs flip flop A_i produces a composite set of input Boolean functions for a typical stage.

$$JA_i=B_iC_i'P_1 + B_i'C_iP_1+P_3+B_iP_5+ B_iP_6+A_{i+1}P_7+A_{i-1}P_8+E_i$$

$$KA_i=B_iC_i'P_1 + B_i'C_iP_1+P_2+P_3+B_i'P_4+ B_iP_6+ A'_{i+1}P_7+ A'_{i-1}P_8+ E_i$$

Each stage in the accumulator must produce the carry for the next stage.

$$C_{i+1}= A_iB_i+ A_iC_i +B_iC_i$$

$$E_{i+1}= E_iA_i$$

$$Z_{i+1}=Z_iA_i'$$

The logic diagram for one typical stage of the Accumulator is shown below:

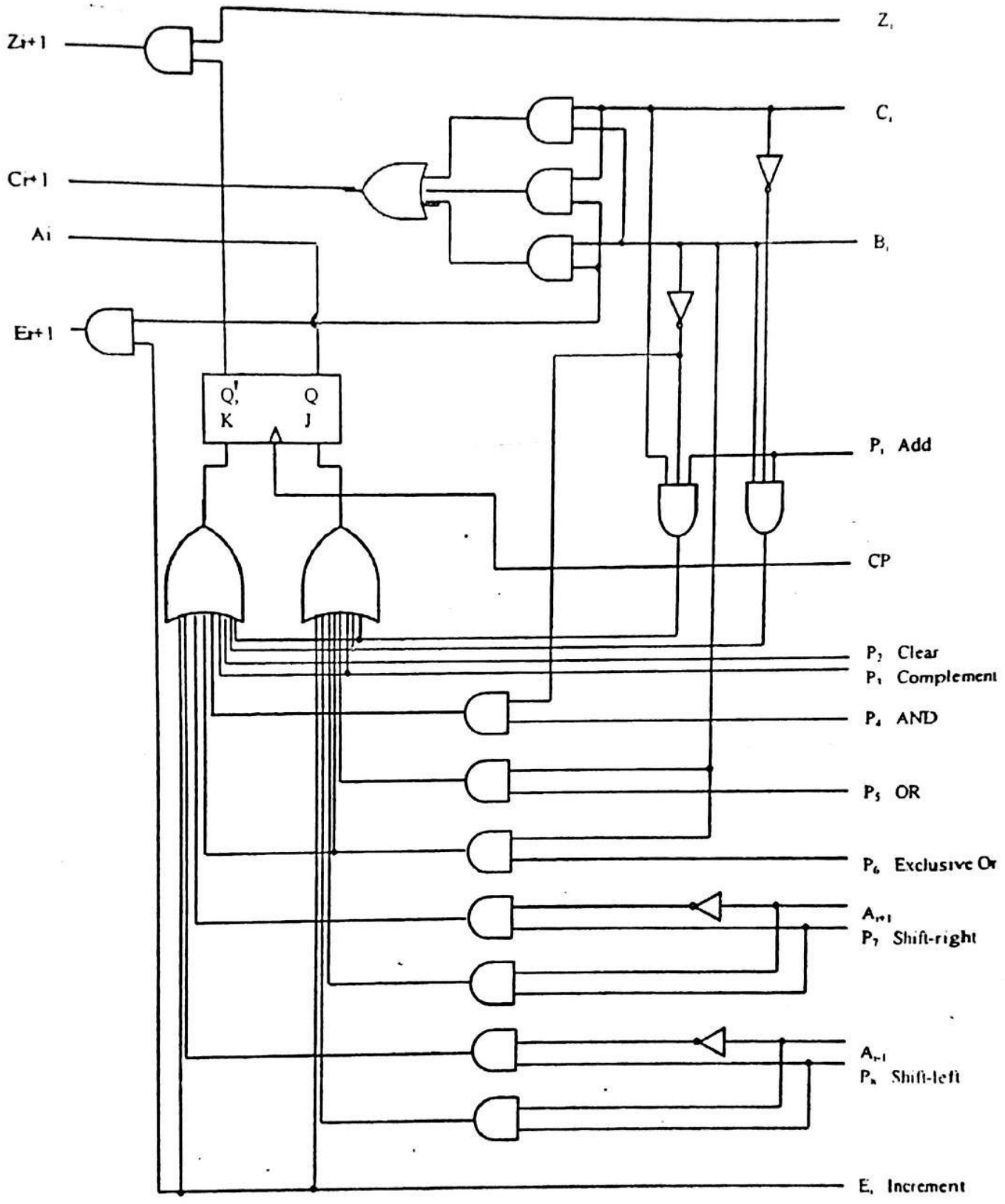


Fig 5.24: One typical stage of the accumulator

Each accumulator stage has eight control inputs from P_1 to P_8 that initiates one of eight possible microoperations. Control variable P_9 is applied only to the first stage to enable the increment operation through input E_i.

There are six other inputs in the circuit. Data input B_i, input carry C_i,A_{i-1} comes from the flip flop one position to the right,A_{i+1} comes from the flip flop one position to the left, E_i(carry input for the increment operation),and Z_i(used to form chain of zero detection).

There are four outputs to this circuit.

A_i	-	output of the flip flop
C_{i+1}	-	carry for the next stage
E_{i+1}	-	increment carry for the next stage
Z_{i+1}	-	carry for the next stage for zero detection.

Complete Accumulator

We have covered the design of one stage of Accumulator. For a complete accumulator there will be n stages like this. 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 following diagram shows the design.

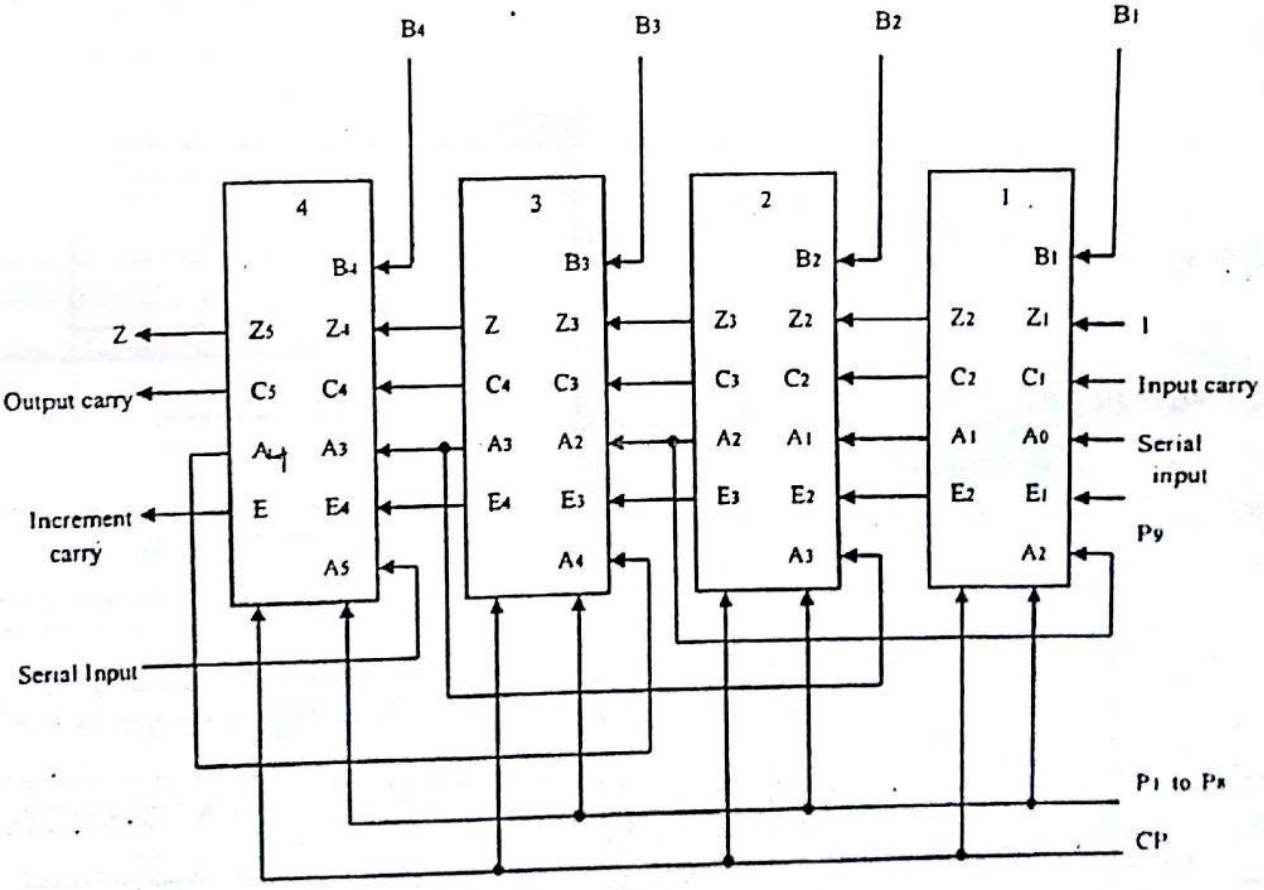


Fig 5.25: 4-bit accumulator constructed with four stages

The number on top of each block represents the bit position. All blocks receive 8 control variables p1 to p8 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 1. 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).

1. Processor organization:

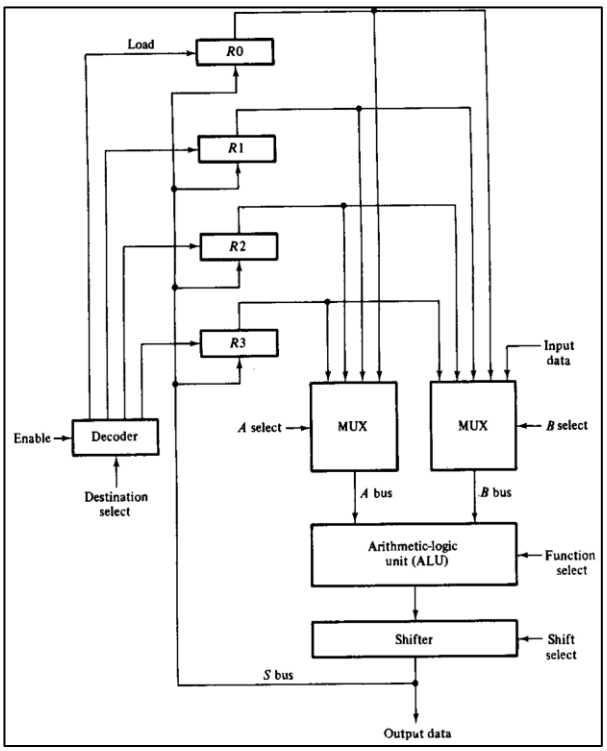
- The processor part of a computer CPU is sometimes referred to as the ***data path*** of the CPU
  - Processor forms the paths for data transfers between registers.
  - The various paths controlled by means of **gates** that **open** the required path and **close** all others.
  - The gating is achieved through **decoders** and **combinational circuit**.
- In a well-organized 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 following are few alternatives for organizing a general-purpose processor unit.

i. Bus Organization
ii. Scratchpad Memory
iii. Accumulator Register
   - All organizations employ a common ALU and shifter
   - The differences in organizations are mostly manifested in the organization of the registers and their common path to the ALU.

i.Bus Organization

- When a large number of registers are included in a processor unit, it is most efficient to connect them through common buses. [Very fast access time].
- The registers communicate with each other for
  - Direct data transfers
  - Performing micro-operations.



A bus organization for ***4 processor registers*** is shown in Figure.

- Each register is connected to 2 MUX to form input buses A and B.
  - The A and B buses are applied to a common ALU.
  - **One input** of multiplexer A or B can receive data from the **outside.**
- The selection lines of each multiplexer select one register out of 4 for the particular bus.
- The function selected in the ALU determines the particular operation that is to be performed.
- The shift micro-operations are implemented separately in the shifter with the type of shift.
- The result of the microoperation goes through the **output bus S** into the **inputs of all registers**.
  - The output bus S provides the terminals for transferring data to an **external destination**
- 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.
- A processor unit may have more than four registers.
  - The construction of a bus-organized processor with more registers requires larger multiplexers and decoder.

#Consider an example to perform the microoperation: R1←R2+R3

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.

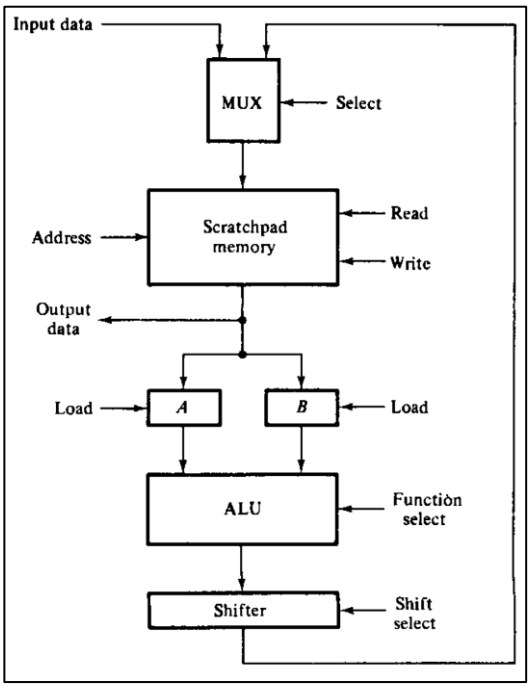
4. Shift selector: for direct transfer from the output of the ALU onto output bus S (no shift).

5. Decoder destination selector: to transfer the contents of bus S into R l.

- The five control selection variables must be generated simultaneously and must be available during one common clock pulse interval.
    - The binary information from the two source registers propagates through the combinational gates in the multiplexers, the ALU, and the shifter, to the output bus, and into the inputs of the destination register, all during **one clock pulse** interval.
    - Then, when the **next clock** pulse arrives, the binary information on the output bus is transferred into R1.
        - To achieve a fast response time, the ALU is constructed with carry look-ahead circuits and the shifter is implemented with combinational gates.

ii.Scratchpad Memory

- The registers in a processor unit can be enclosed within a small memory unit, called a ***scratchpad memory***.
    - It is a **cheaper alternative** to connect processor registers through a **bus system**.
- The difference between **scratchpad memory** and **bus systems** is the manner in which information is selected for transfer into the ALU.
    - In a **bus system**, the information transfer is **selected by the multiplexers** that form the buses.
    - In **scratchpad memory**, a single register in a group of registers organized as a small memory must be **selected by means of an address** to the memory unit.
- A scratchpad memory should be distinguished from the main memory of the computer.
    - Contrary to the **main memory** which stores instructions and data, **scratchpad memory** in a processor unit is an alternative to connect a number of processor registers through a **common transfer path**.
    - Consider, for example, a processor unit that employs **8 registers** of **16 bits each**. The registers can be enclosed within a **small memory** of 8 words of 16bits each, or an **8 x 16 RAM**. The eight memory words can be designated **R0**through **R7** constitute the registers for the processor.



A processor unit that with scratchpad memory is shown in Figure

- A **source register** is selected from memory and loaded into register A.
- A **second source** register is selected from memory and loaded into register B.
    - The **selection** is done by specifying the corresponding word address and activating the **memory-read input**.
- The information in A and B is manipulated in the **ALU** and **shifter**.
- The **result** of the operation is transferred to a **memory register** by specifying its **word address** and activating the **memory-write input control**.
- The **multiplexer** in the input of the memory can **select input** data from an external source.

➢ Assume that the memory has 8 words, so that an address must be specified with 3 bits.

To perform the operation: R1←R2+R3

- the control must provide binary selection variables to perform the following sequence of three micro-operations:

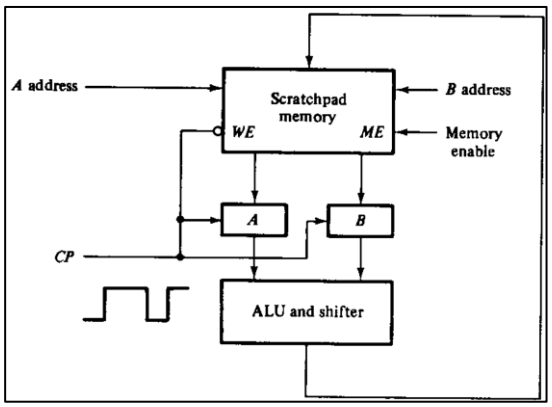
i.	T_1: A←M[010]	read R2 into register A
ii.	T_2: B←M[011]	read R3 into register B
iii.	T_3: M[001]← A + B	perform operation in ALU and transfer result to R1

- 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 (with no shift), apply an address 001 to the memory, select the output of the shifter for the MUX, and activate the memory write input.
    - The symbol M [xxx] designates a memory word (or register) specified by the address given in the binary number xxx.

- The reason for a sequence of 3 micro-operations, instead of just one as in a bus-organized processor, is due to the **limitation** of the memory unit.
    - Since the memory unit has only one set of address terminals but two source registers are to be accessed, two accesses to memory are needed to read the source information.
    - The third microoperation is needed to address the destination register.
    - If the destination register is the same as the second source register, the control could activate the read input to extract the second-source information, followed by a write signal to activate the destination transfer, without having to change the address value.

Some processors employ a **2-port memory** in order **to overcome the delay** caused when reading two source registers.

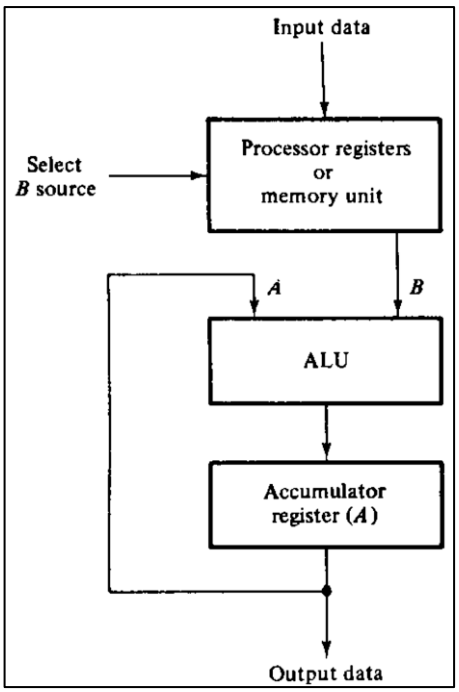
- A 2-port memory has **two separate address lines** to select two words of memory simultaneously.
    - Two source registers can be read at the same time.
    - If the **destination register is same as one of the source registers**, then the entire microoperation can be done **within one clock pulse period**.
    - The memory has **two sets of addresses**, one for port A and the other for port B.
    - Data from any word in memory are read into the register by **specifying an address**.
    - When enabled by the **Memory Enable**(ME) input, new data can be written into the word specified by the B address.
    - Thus the A and B addresses specify two source registers simultaneously, and the **B address always specifies the destination register**.



- The A and B registers, act as latches, **accept new information** as long as the clock pulse, **CP** is in the **1-state**.
    - When **CP goes to 0**, the latches are disabled, and they **hold the previous information.**
    - This **eliminates** any possible **race conditions** that could occur while new information is being written into memory.
    - The **clock input controls the memory read and write operations** through the **Write Enable** (WE) input. It also controls the transfers into the A and B latches.
- Thus, a microoperation can be done **within one clock pulse period**.

iii. Accumulator Register

- Some processor units separate one register from all others and call it an **Accumulator Register**, abbreviated **AC** or **A** register.
    - The name of this register is derived from the consecutive arithmetic addition process encountered in digital computers.
- The accumulator register in a processor unit is a multipurpose register capable of performing many microoperations other than addition.
    - The gates associated with an accumulator register provide all the functions found in an ALU.
- Figure shows the block diagram of a processor unit that employs an accumulator register.
    - The A register is distinguished from all other processor registers.
        - The **register itself can function as a shift register** to provide the shift microoperations.
    - Input B supplies information from other processor registers or directly from the main memory of the computer.
    - The result of an operation is transferred back to the A register and replaces its previous content.
        - The output from the A register may go to an external destination or into the input terminals of other processor registers or memory unit.



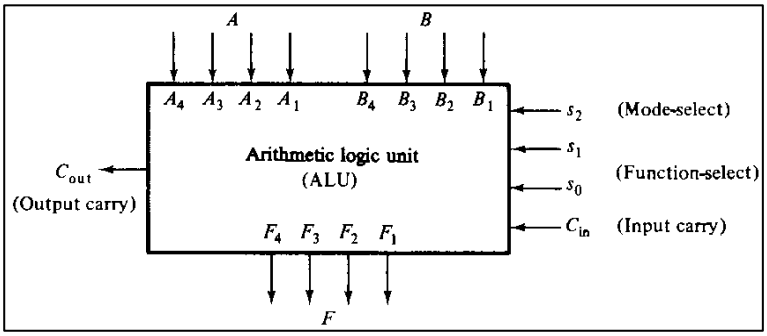
To form the **sum of two numbers** stored in processor registers, it is necessary to add them in the A register using the following sequence of microoperations:

T_1:	$A \leftarrow 0$	clear A
T_2:	$A \leftarrow A + R1$	transfer $R1$ to A
T_3:	$A \leftarrow A + R2$	add $R2$ to A

- Register A is first cleared. The first number in R1 is transferred into the A register by adding it to the present zero content of A. The second number in R2 is then added to the present value of A. The sum formed in A may be used for other computations or may be transferred to a required destination.

Arithmetic Logic Unit

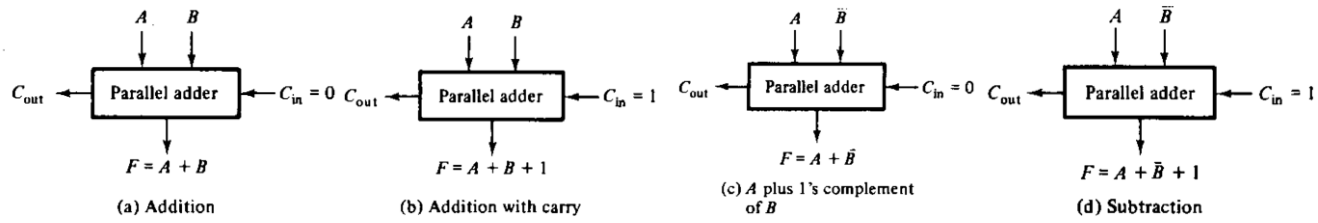
- An arithmetic logic unit (ALU)
  - is a **multi-operation**, combinational-logic digital function.
  - performs a set of basic **arithmetic** operations and **logic** operations.
- The ALU has a number of **selection lines** to select a particular operation in the unit.
  - The selection lines are **decoded** within the ALU so that **k selection variables** can specify up to 2^k **distinct operations**.
- Figure shows the block diagram of a 4-bit ALU.
  - The 4 data inputs from A are combined with the 4 inputs from B to generate an operation at the F outputs
  - The **mode-select input** s_2 distinguishes between arithmetic and logic operations.
  - The **two function-select inputs** and so specify the particular arithmetic or logic operation to be generated.

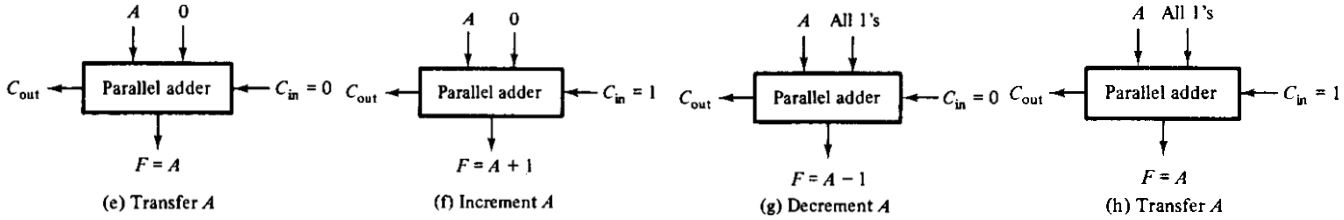


  - With **3 selection variables**, it is possible to specify **4 arithmetic operations** (with s_2 in one state) and **4 logic operations** (with $s_2$in the other state).
  - The **input and output carries** have meaning **only during an arithmetic operation**.
    - **The input carry** quite often used as **4thselection variable** that can double the number of arithmetic operations. In this way, it is possible to generate 4 more operations, for a total of **8 arithmetic operations**.
- The **design of a typical ALU** will be carried out in three stages.
  i. The **design of the arithmetic section** will be undertaken.
  ii. The **design of the logic section** will be considered.
  iii. The **arithmetic section will be modified** so that it can perform **both arithmetic and logic operations**.

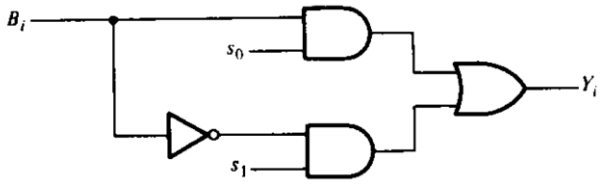
2. Design of Arithmetic unit

- 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 **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.



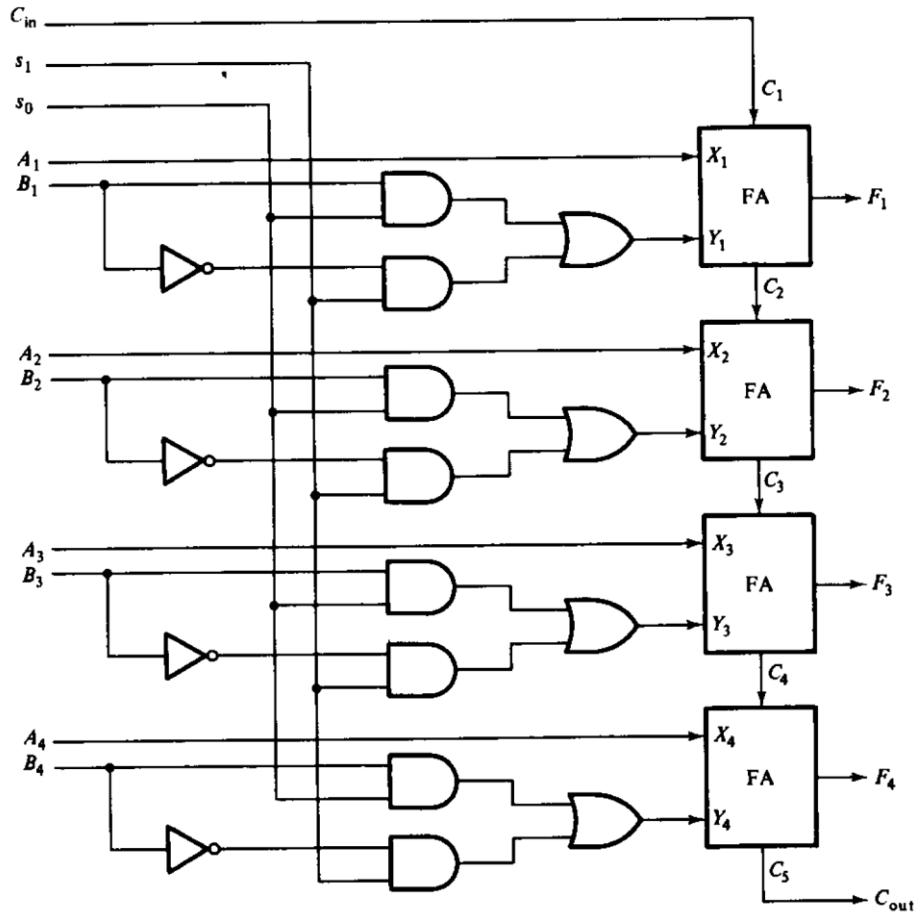


- The circuit that controls input B to provide the functions is called a **true/complement, one/zero** element.
- This circuit is illustrated in the following figure.
    - The **2 selection lines** s_1 and s_0 control the input of each B terminal.



s_1	s_0	Y_i
0	0	0
0	1	B_i
1	0	B_i'
1	1	1

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



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	$\overline{B}$	$F = A + \overline{B}$	Add 1's complement of B to A
1	0	1	$\overline{B}$	$F = A + \overline{B} + 1$	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 s_1, s_0, and C_{in} .
  - Variables s_1 and s_0 **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 variables** s_1 and s_0.
  - **Adding** the value of **Y** in each case to the value of **A** plus the C_{in} 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**.
  - The **combinational circuit** will be **different** if the circuit generates **different arithmetic operations**.

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	$F = A + B$	$(A + B) \geq 2^n$	Overflow occurs if $C_{out} = 1$
0	1	1	$F = A + B + 1$	$(A + B) \geq (2^n - 1)$	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	$F = A - B$	$A \geq B$	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

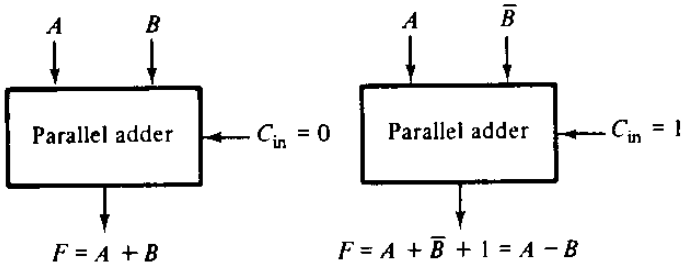
Design of other Arithmetic Circuits

- 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**.

Qu: Design an adder/subtractor circuit with one selection variable s and two inputs A and B.

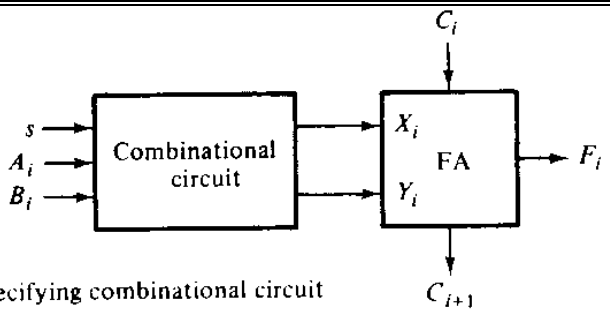
When s = 0, the circuit performs A+B

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

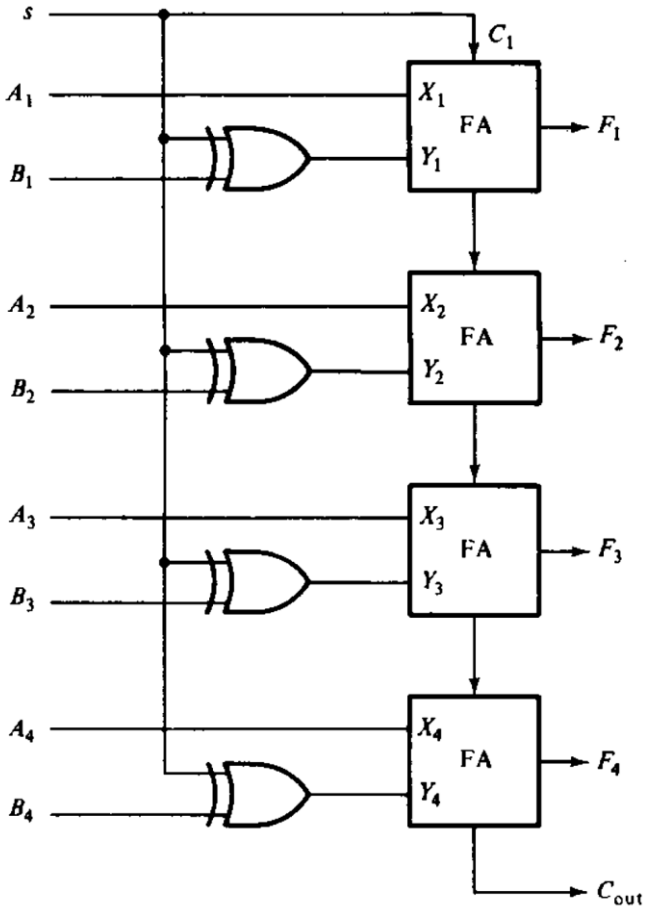


Figure 9-10 4-bit adder/subtractor circuit

3. Design of Logic unit,

The logic microoperations **manipulate the bits** of the operands **separately** and treat**each bit as a binary variable**.

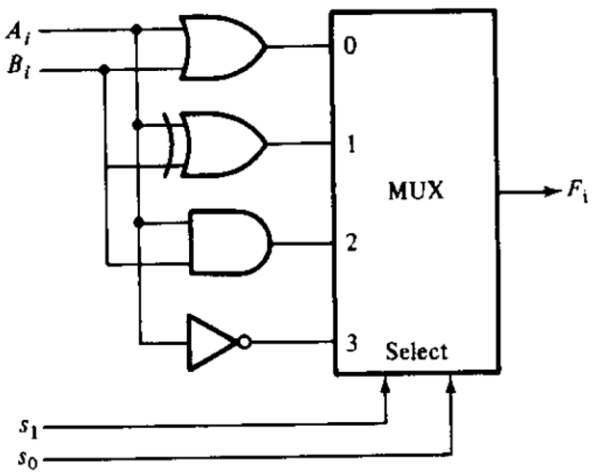
Boolean functions	Operator symbol	Name	Comments
$F_0 = 0$		Null	Binary constant 0
$F_1 = xy$	$x \cdot y$	AND	x and y
$F_2 = xy'$	x/y	Inhibition	x but not y
$F_3 = x$		Transfer	x
$F_4 = x'y$	y/x	Inhibition	y but not x
$F_5 = y$		Transfer	y
$F_6 = xy' + x'y$	$x \oplus y$	Exclusive-OR	x or y but not both
$F_7 = x + y$	$x + y$	OR	x or y
$F_8 = (x + y)'$	$x \downarrow y$	NOR	Not-OR
$F_9 = xy + x'y'$	$x \odot y$	Equivalence*	x equals y
$F_{10} = y'$	y'	Complement	Not y
$F_{11} = x + y'$	$x \subset y$	Implication	If y then x
$F_{12} = x'$	x'	Complement	Not x
$F_{13} = x' + y$	$x \supset y$	Implication	If x then y
$F_{14} = (xy)'$	$x \uparrow y$	NAND	Not-AND
$F_{15} = 1$		Identity	Binary constant 1

*_Equivalence_ is also known as _equality_, _coincidence_, and _exclusive-NOR_.

- Table listed **16 logic operations** that can be performed with **two binary variables**.
- 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.

The simplest and most straightforward way to design a logic circuit is shown in the following Figure.

- The diagram shows one typical stage designated by subscript i.
- The circuit must be **repeated n times** for an **n-bit logic circuit**.
- The four gates generate the 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.



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	$F_i = A_i B_i$	AND
1	1	$F_i = A_i'$	NOT

- 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 following Figure.

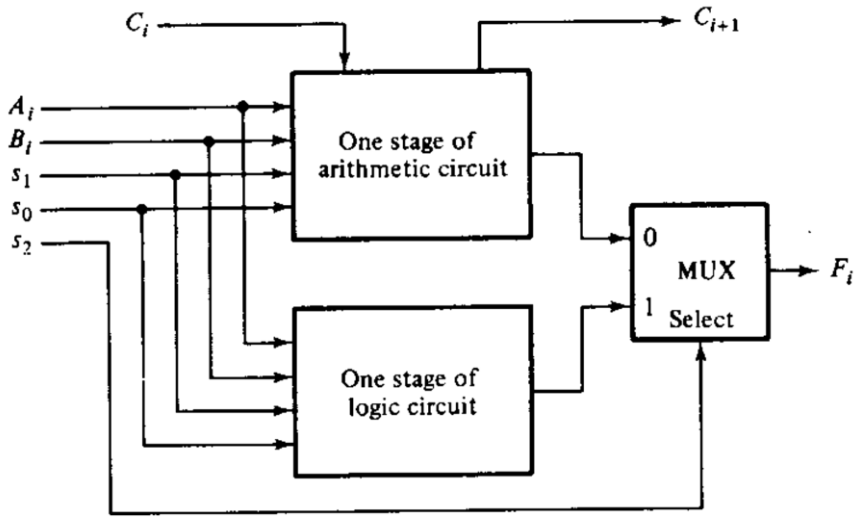


Figure 9-12 Combining logic and arithmetic circuits

- The outputs of the logic and arithmetic circuits in each stage go through a multiplexer with selection variable s_2.
  - $s_2 = 0$, the **arithmetic output** is selected, and 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:

$$F_i = X_i \oplus Y_i \oplus C_i$$

- The input carry C_i in each stage can be made to be equal to 0 when a selection variable is equal to 1.
  - The result would be:

$$F_i = X_i \oplus Y_i$$

  - This expression is valid because of the property of the exclusive-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 $\mathbf{Y_i}$ can be selected by means of the **two selection variables** to be equal to either **0, B_i, B_i', or 1.**
  - The value of $\mathbf{X_i}$ is always equal to input $\mathbf{A_i}$. Table given below shows the four logic operations obtained when $s_2=0$

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

  - This selection variable forces $\mathbf{C_i}$ **to be equal to 0** whiles_1 **and** s_0 **choose a particular value for** $\mathbf{Y_i}$.

- The 4 logic operations obtained by this configuration are **transfer**, **exclusive-OR**, **equivalence**, and **complement**. The third entry is the equivalence operation because:

$$A_i \oplus B_i' = A_i B_i + A_i' B_i' = A_i \odot B_i$$

- 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.

4. Design of Arithmetic logic unit

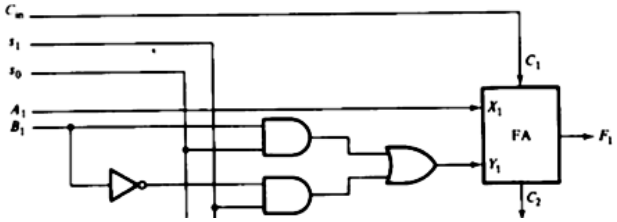
A basic ALU with eight arithmetic operations and four logic operations can be designed with the details already have.

- We already have
  - Three selection variables s_2, s_1, and $s_0$to select eight different operations
  - The input carry C_{in} is used to select four additional arithmetic operations.
  - With $s_2 = 0$, selection variables $s_1$and s_0 together with C_{in} will select the eight arithmetic operations
  - With $s_2 = 1$, 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: A_i, B_i, C_i,s_2, s_1 and s_0.
  - There are **two outputs** in each stage: output Fi and the carry out C_{i+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.

➢ The steps involved in the **design of an ALU** are as follows:

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 **s_2= 1**, the input carry C_i in each stage must be 0. **C_i=0.**
- With **s_1,s_0 = 00**, each stage as it stands generates the function **$F_i = A_i$**. [Since **X_i=A_i** and **Y_i=0;** refer diagram]

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

  - To change the output to an **OR** operation, we must change the input to each full-adder circuit from **A_i to A_i+B_i**. This can be accomplished by **OR**ing **B_i** and **A_i** when **$s_2s_1s_0$ = 100**.

  - The other selection variables that give an undesirable output occur when**$s_2s_1s_0$= 110**. The unit as it stands generates an output **$F_i = A_i \odot B_i$**, but we want togenerate the AND operation **$F_i = A_i.B_i$**.

    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.

The function so obtained is then used for Xi when $s_2s_1s_0$= **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 $s_2s_1s_0$= **110**, the output will generate the **AND** operation.

- The **final ALU** is shown in Figure.
  - 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:

$$X_i = A_i + s_2s_1's_0'B_i + s_2s_1s_0'B_i'$$
$$Y_i = s_0B_i + s_1B_i'$$
$$Z_i = s_2'C_i$$

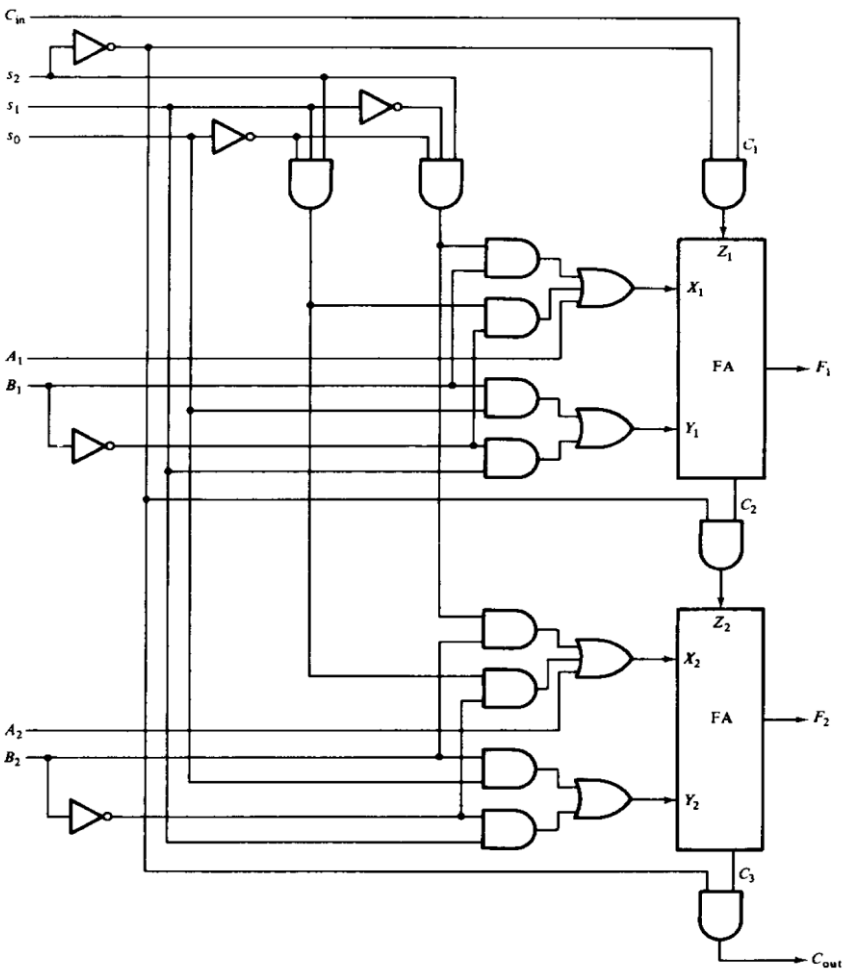
  - Whens_2=**0**, these functions will reduce to

$$X_i = A_i$$
$$Y_i = s_0B_i + s_1B_i'$$
$$Z_i = C_i$$

    which are the functions for the **Arithmetic Circui**t
  - When s_2=**1**, these functions will reduce to

$$X_i = A_i + s_1's_0'B_i + s_1s_0'B_i'$$
$$Y_i = s_0B_i + s_1B_i'$$
$$C_i = 0$$

    which are the functions of **Logic Circuit**.



- The <u>12 operations generated in the ALU</u> are summarized in Table.
  - The particular function is selected through s_2,s_1, s_0, and C_{in}.
  - The arithmetic operations are identical to the ones listed for the arithmetic circuit.
  - The value of C_{in} 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.

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	$F = \overline{A}$	Complement A

5. Design of Status register

It is sometimes convenient to supplement the ALU with a **status register** where the status-bit conditions like **sign of the result**, a **zero indication**, and an **overflow condition** are stored for further analysis. **Status-bit conditions** are sometimes called **condition-code** bits or **flag bits**.

- Figure shows the block diagram of an 8-bit ALU with a 4-bit status register.
  - The **4 status bits** are symbolized by **C, S, Z**, and **V**.
  - The bits are **set** or **cleared** as a result of an operation performed in the ALU.

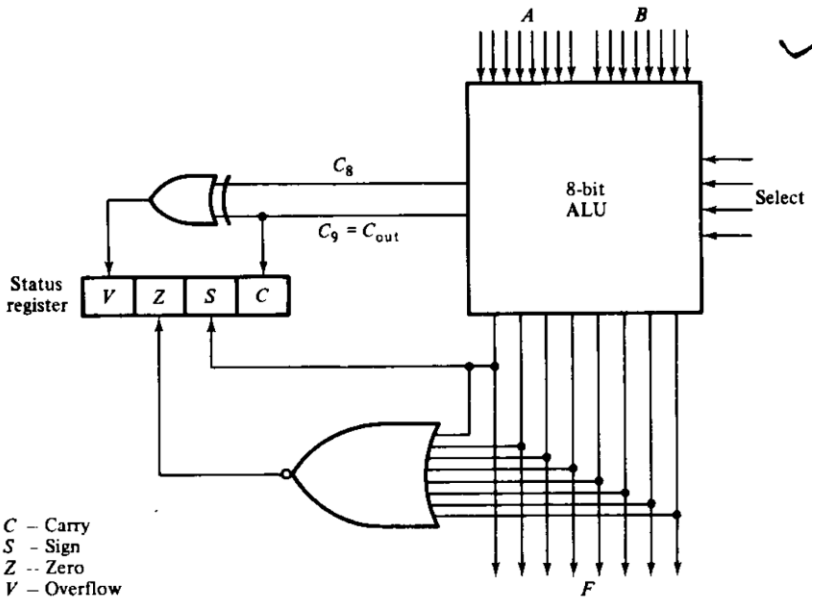


Figure 9-14 Setting bits in a status register

l. Bit **C is set** if the **output carry** of the ALU **is 1**. It is **cleared** if the **output carry is 0**.

2. **Bit S is set** if the **highest-order bit** of the result in the output of the ALU(the **sign bit) is 1**(**Negative number**). It is **cleared if** the **highest-order bit is 0** (**Positive Number**).

3. Bit **Z is set** if the output of the ALU contains **all0's**, and **cleared otherwise**. **Z = 1**if the **result is zero**, and **Z = 0** if the result is **nonzero**.

4. **Bit V is set** if the **exclusive-OR** of carries **C8 and C9 is 1** (one of them is exclusively 1), and **cleared otherwise**.[This is the condition for **overflow** when the numbers are insign-2's-complement representation.]

- The **status bits can be checked** after an **ALU operation** to determine certain relationships that exist between the values of A and B.
  - If bit **V is set after the addition** of two signed numbers, it indicates an **overflow** condition.
  - If **Z is set after an exclusive-OR** operation, it indicates that **A = B**.
  - **Exclusive-OR** of **two equal operands** gives an **all-0's** result which **sets the Z bit**.
  - A **single bit in A** can be **checked to determine if it is 0 or 1** by **masking all bits** except the bit in question and then **checking the Z status bit**.
  - The **compare operation** is a subtraction of B from A, except that the result of the operation is not transferred into a destination register, but the **status bits are affected**, which gives the **relative magnitudes** of A and B

TABLE 9-5 Status bits after the subtraction of unsigned numbers ($A - B$)

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

The table lists the six possible relationships that can exist between A and B and the corresponding values of Z, S and V in each case for the operation A-B. Boolean functions listed in the table express the status-bit conditions in algebraic form.

TABLE 9-6 Status bits after the subtraction of sign-2's complement numbers ($A - B$)

Relation	Condition of status bits	Boolean function
$A > B$	$Z = 0$ and ($S = 0, V = 0$ or $S = 1, V = 1$)	$Z'(S \odot V)$
$A \geq B$	$S = 0, V = 0$ or $S = 1, V = 1$	$S \odot V$
$A < B$	$S = 1, V = 0$ or $S = 0, V = 1$	$S \oplus V$
$A \leq B$	$S = 1, V = 0$ or $S = 0, V = 1$ or $Z = 1$	$(S \oplus V) + Z$
$A = B$	$Z = 1$	Z
$A \neq B$	$Z = 0$	Z'

6. Design of Shifter

- The **shift unit** attached to a processor transfers the **output of the ALU onto the output bus**
- The **shifter** provides the **shift microoperations** commonly not available in an ALU.

The shifter may

- Transfer the information directly **without a shift**,
- Shift the information to the **right or left**.
- **No transfer** from the **ALU to the output bus**, sometimes

An obvious circuit for a shifter is a **bidirectional shift-register** with **parallel load**.

- The **information** from the ALU can be **transferred** to the register **in parallel** and then **shifted to the right or left.**

A combinational-logic shifter can be constructed with multiplexers as shown in Figure.

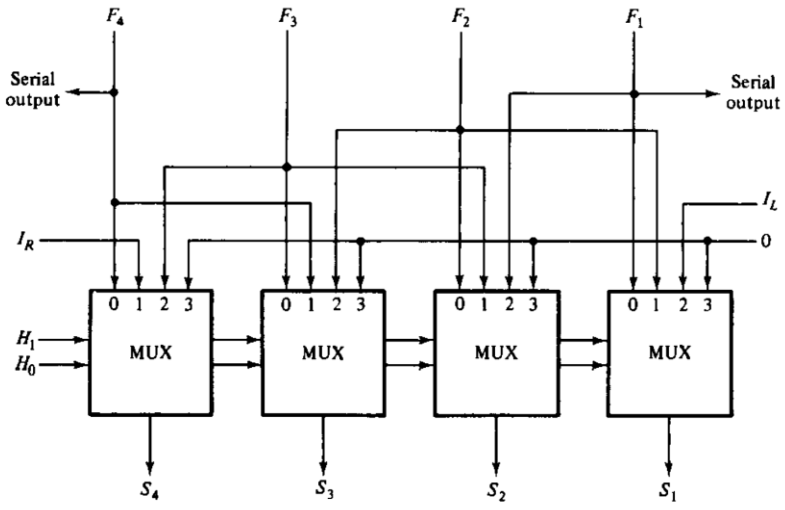


Figure 9-15 4-bit combinational-logic shifter

TABLE 9-7 Function table for shifter

H_1	H_0	Operation	Function
0	0	$S \leftarrow F$	Transfer F to S (no shift)
0	1	$S \leftarrow$ shr F	Shift-right F into S
1	0	$S \leftarrow$ shl F	Shift-left F into S
1	1	$S \leftarrow 0$	Transfer 0's into S

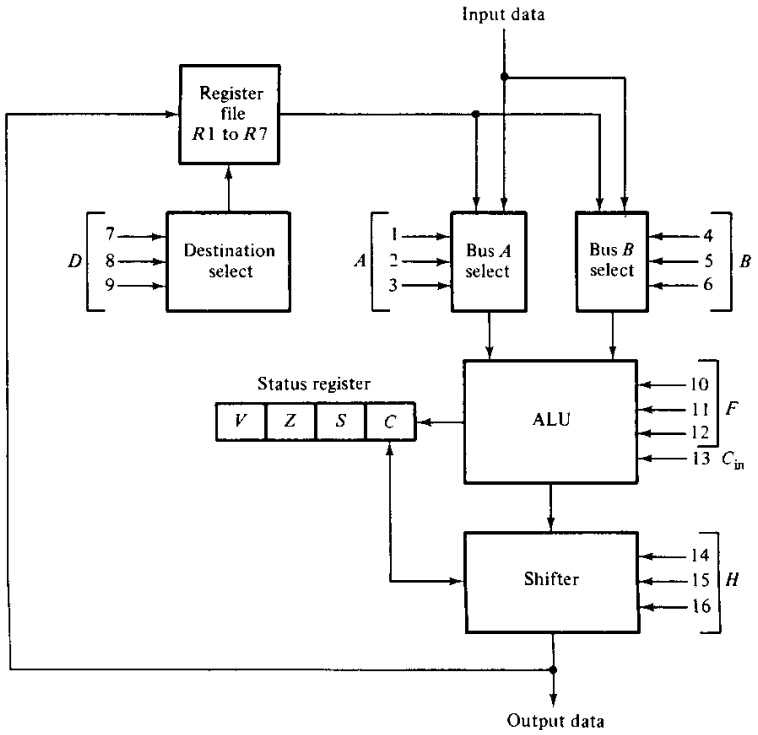
- The diagram shows only 4 stages of the shifter. The shifter, must consist of **n stages in a system with n parallel lines**
- Inputs **IR and IL serve as serial inputs** for the last and first stages during a **shift-right** or **shift-left**, respectively.

7. Design of Processor unit

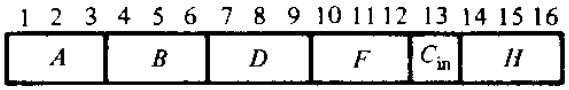
- The **selection variables** in a processor unit **control the micro-operations** executed within the processor during any given clock pulse.
- The **selection variables** control the **Buses**, the **ALU**, the **Shifter**, and the **Destination Register**.

How the control variables select the microoperations in a processor unit?

A block diagram of a processor unit is shown in Figure



- It consists of 7 registers **R1 through R7** and a **Status Register**.
- The **outputs of the 7 Registers** go through **2 Multiplexers** to **select the inputs to the ALU**.
  - 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.



(b) Control word

- 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, 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**.

TABLE 9-8 Functions of control variables for the processor of Fig. 9-16

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	*R*1	*R*1	*R*1	$A + B$	$A + B + 1$	Shift-right, $I_R = 0$
0 1 0	*R*2	*R*2	*R*2	$A - B - 1$	$A - B$	Shift-left, $I_L = 0$
0 1 1	*R*3	*R*3	*R*3	$A - 1$	$A, C \leftarrow 1$	0's to output bus
1 0 0	*R*4	*R*4	*R*4	$A \vee B$	—	—
1 0 1	*R*5	*R*5	*R*5	$A \oplus B$	—	Circulate-right with *C*
1 1 0	*R*6	*R*6	*R*6	$A \wedge B$	—	Circulate-left with *C*
1 1 1	*R*7	*R*7	*R*7	$\bar{A}$	—	—

TABLE 9-9 Examples of microoperations for processor

Microoperation	Control word						Function
	A	*B*	*D*	*F*	C_{in}	*H*	
$R1 \leftarrow R1 - R2$	001	010	001	010	1	000	Subtract *R*2 from *R*1
$R3 - R4$	011	100	000	010	1	000	Compare *R*3 and *R*4
$R5 \leftarrow R4$	100	000	101	000	0	000	Transfer *R*4 to *R*5
$R6 \leftarrow$ Input	000	000	110	000	0	000	Input data to *R*6
Output $\leftarrow R7$	111	000	000	000	0	000	Output data from *R*7
$R1 \leftarrow R1, C \leftarrow 0$	001	000	001	000	0	000	Clear carry bit *C*
$R3 \leftarrow$ shl $R3$	011	011	011	100	0	010	Shift-left *R*3 with $I_L = 0$
$R1 \leftarrow$ crc $R1$	001	001	001	100	0	101	Circulate-right *R*1 with carry
$R2 \leftarrow 0$	000	000	010	000	0	011	Clear *R*2

- When **D = 000, no destination register** is selected.
- The **3 bits in the F field**, together **with the input carry Cin**, provide the **12 operations of the ALU**.

- 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**.

- Many microoperations can be generated in the processor unit by different variety of control words.
  - The control word for each microoperation is derived from the function table of the processor.
- A processor unit with a complete set of microoperations is a general-purpose device that can be adapted for many applications.
  - The register-transfer method is a convenient tool for specifying the operations.
- The sequence of control words for the system is stored in a control memory.
  - The output of the control memory is applied to the selection variables of the processor.
  - By reading consecutive control words from memory, it is possible to sequence the microoperations in the processor.

8. Design of accumulator.

- Some processor units distinguish one register from all others and call it an **Accumulator Register**.
- The **accumulator** register is essentially a **bidirectional shift register with parallel load which is connected to an ALU**.
- Because of the feedback connection from the output of the register to one of the inputs in the ALU, the accumulator register and its associated logic, when taken as one unit, constitute a **sequential circuit**. And an accumulator register can be designed by sequential-circuit techniques instead of using a combinational-circuit ALU.
- The external inputs to the accumulator are the data inputs from B and the control variables that determine the microoperations for the register.
- An accumulator is a **multi-function register** that, by itself, can be made to **perform all of the microoperations** in a processor unit.

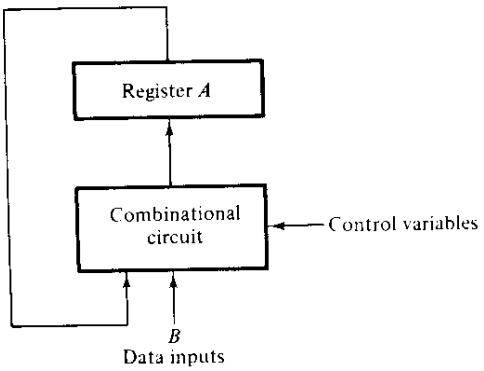


Figure 9-17 Block diagram of accumulator

To demonstrate the logic design of a multipurpose operational register such as an accumulator, we will design the circuit with **9 microoperations**.

TABLE 9-10 List of microoperations for an accumulator

Control variable	Microoperation	Name
p_1	$A \leftarrow A + B$	Add
p_2	$A \leftarrow 0$	Clear
p_3	$A \leftarrow \overline{A}$	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	$A \leftarrow \text{shr } A$	Shift-right
p_8	$A \leftarrow \text{shl } A$	Shift-left
p_9	$A \leftarrow A + 1$	Increment
	If $(A = 0)$ then $(Z = 1)$	Check for zero

- Control variables through are generated by **control logic circuits** and should be considered as **control functions** that initiate the corresponding **register-transfer operations**.
- **Register A** is a **source** register in all the listed microoperations.
  - In essence, this represents the present state of the sequential circuit.
- The **B register** is used as a **second source** register for microoperations that need two operands.
  - The B register is assumed to be connected to the accumulator and supplies the inputs to the sequential circuit.
- The **destination register** for all microoperations is always **register A**.
- The new information transferred to A constitutes the next state of the sequential circuit.

- The **9 control variables** are also considered **as inputs** to the sequential circuit.
  - These variables are mutually exclusive and only one variable must be enabled when a clock pulse occurs.
  - The last entry in Table is a conditional control statement. It produces a binary 1 in an output variable Z when the content of register A is 0, i.e., when all flip-flops in the register are cleared.

One Stage of Accumulator

- A typical accumulator stage consists of all the circuits that were designed for the individual microoperations.
- Control variables p_1 through p_9 are **Mutually Exclusive**. (i.e. only one at a time).

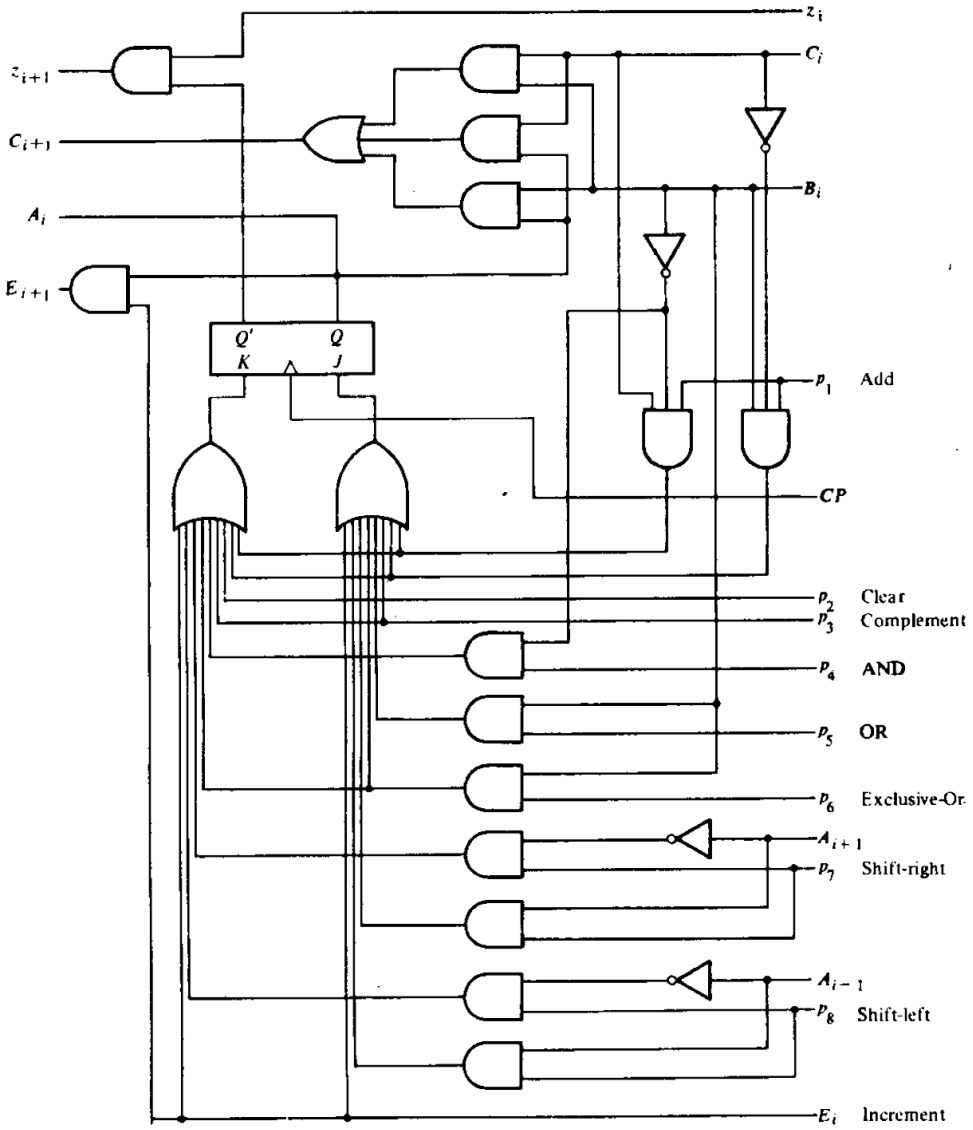


Figure 9-22 One typical stage of the accumulator

- The accumulator is to be partitioned into **n stages**, and each stage is to be partitioned into those circuits that are needed for each microoperation.
  - Once the various pieces are designed separately, it will be possible to combine them to obtain one typical stage of the accumulator and then to combine the stages into a **complete accumulator**.

Complete Accumulator

- An accumulator with **n bits** requires **n stages** connected in **cascade**, with each stage having the circuit shown in above Figure.
  - All control variables, except p_9, must be applied to each stage.
  - The other inputs and outputs in each stage must be connected in cascade to form a complete accumulator.
- The interconnection among stages to form a complete accumulator is illustrated in the 4-bit accumulator shown below.

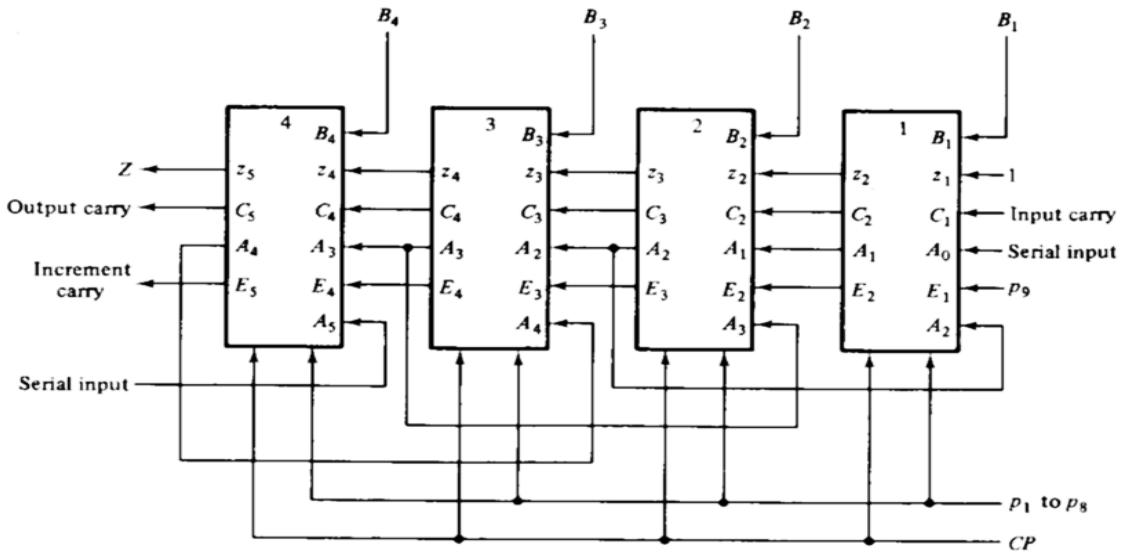


Figure 9-23 4-bit accumulator constructed with four stages

- The number on top of each block represents the bit position in the accumulator.
  - All blocks receive **8 control variables**, through p1 to p8, and the **clock pulses** from CP.
  - The other **6 inputs** and **4 outputs** in each block are identical to those of a typical stage.
- The circuit has **four B inputs**.
- The **zero-detect chain** is obtained by connecting the **z variables in cascade**.
  - The last stage in this chain produces the zero-detect variable Z.
- The carries for the arithmetic addition are connected in cascade as in full-adder circuits.
- The serial input for the shift-left operation goes to input A_0, which corresponds to input A_{i-1} in the first stage.
- The serial input for the shift-right operation goes to input A_5, which corresponds to A_{i+1} in the fourth and last stage.
- The increment operation is enabled with control variable in the first stage.
- The other blocks receive the increment carry from the previous stage.
- The 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 variables can be reduced from 9 to 4 if a decoder is inserted in the IC.
    - In such a case, the IC pin count can be reduced to 22 and the accumulator can be extended to 16 microoperations without adding external pins.

Module 2

1.What do you meant by logic micro operations?

2. Design a 4bit combination logic shifter.

3.a) **Design a 4 bit arithmetic unit with two selection variables s0 and s1 and two n-bit data inputs A&B and input carry Cin**

s1	s2	Cin=0	Cin=1	
0	0	F=A	F=A+1	
0	1	F=A+B	F=A+B+1	
1	0	F=A+B'	F=A+B'+1	
1	1	F=A-1	F=A	

b) **Explain the design of an accumulator.**

4 a) **Design an adder/subtractor circuit with one selection variable s and two inputsA and B. When s=0, the circuit performs A+B and when s=1 it performs A-B, by taking 2's complement of B.**

b) **Explain the design of status register.**

5. **What are conditional control statements? Represent the following conditional control statement by two register transfer statements with control functions.**

$$If\,(P=1)\ then\ (R1 \leftarrow R2)\ \ else\ if\ (Q=1)\ then\ (R1 \leftarrow R3)$$

6.Illustrate the basic arithmetic microoperations in a 4 bit ALU with the help of a parallel adder.

7.Write the Register Transfer Logic format for a conditional control statement. Give an example and explain the same.

b) Mention the advantages of using a scratch pad memory. Draw the diagram of a processor that employs a scratch pad memory and explain the same.

8.**Design a 4-bit combinational logic shifter with 2 control signals H1 and H0 that performs the following operations (bit values given in parenthesis are the values of control variables H1 and H0 respectively):- No shift (00), Shift-right (01), Shiftleft (10), Transfer 0's to S (11).**

9. **Draw and explain the block diagram for a 4-bit complete accumulator**

10.Which are the different methods of processor organization?

11 Explain the design of a 4bit Arithmetic unit with two selection variables, which performs the basic arithmetic functions.

12.. a) Explain the design of status register

b) Give the design of a 4 bit shifter.

13.Write short notes on Arithmetic , logic and shift microoperations with examples

b) **Show the block diagram that executes the following conditional control statements**

C' T2 : F ← A

C T2 : F ← B where C is the conditional variable and A, B , F are registers

14.Give a simple design for generating status bits for a 8-bit ALU.

15.**Draw a labelled block diagram of a processor unit with seven registers R1 to R7,a status register,ALU with 3-selection variables and Cin, and shifter with 3 selection variables.**

16.**Discuss shift and conditional control micro operations**

17.An 8-bit register A has one input x. The register operation is represented symbolically as P: A7 ← x, Ai ← Ai+1 i = 0,1,2,3 ... 6. What is the function of the register?

18.Draw the block diagram for the hardware that implements the following statement x + yz: AR ← AR + BR where AR and BR are two n-bit registers and x, y, and z are control variables. Include the logic gates for the control function. (The symbol + designates an OR operation in a control or Boolean function and an arithmetic plus in a micro operation.)

19. **The 8-bit registers AR, BR, CR and DR initially have the following values: AR = 11110010, BR = 11111111, CR = 10111001, DR = 11101010. Determine the 8-bit values in each register after the complete execution of the following sequence of micro-operations.**

AR = AR + BR Add BR to AR storing answer in AR

CR = CR ^ DR , BR = BR + 1 AND DR to CR, INC BR

AR = AR – CR **Subtract CR from AR**

20. Explain about arithmetic microoperations.

21. Explain design of accumulator

22. Explain shift microoperations with examples.

23. Design an adder/subtractor circuit with one selection variable s and two inputs A and B .When s=0 the circuit performs A+B and s=1 the circuit performs A-B by taking 2's complement of B.

Computer Organization
Processor Logic Design

Overview

1. Register Trasfer Logic
2. Inter Register Transfer
3. Arithmetic Microoperation
4. Logic Micooperation
5. Shift Micooperation
6. Conditional Control Statements

Register Transfer Logic

- Digital system
  - Sequential logic system constructed with flip flops and gates Specified by means of state table
- Difficult to specify a large digital system with state table because of the huge number of states.
- Modular approach
  - System is partitioned into modular subsytems with a specific functional task
  - Modules constructed from such digital functions are registers, counters, decorders, multiplexers, arithmetic elements and control logic
  - Modules are interconnected with common data and control paths to form a digital computer system .
- Typical digital system module - processor unit of a digital computer
- To decribe Digital system module high level mathemetical notations are used - Register Trasfer Logic

Register Transfer Logic(cont.)

- Register Trasfer Logic
  - Regiters are the primitive component
  - Information flow and processing tasks among the data stored in registers are described in precise and concise mannner
  - Uses set of expressions & statements which resemble the statements in programming language

Register Transfer Logic(cont.)

- Components that form the basis of register transferlogic
  1. The set of registers in the system and their functions
  2. The binary coded information stored in the registers
  3. The operations performed on the information stored in the registers - Microoperations
  4. The control functions that initiate the sequence of operations

Register Transfer Logic(cont.)

- Registers
  - Encompasses all type of registers such as shift registers, counters and memory units
  - Counter :- Funtion is to increment by 1 the information stored within it
  - Memory unit :- Collection of storage registers where information can be stored
  - Flip flop :- stand alone flip flop is a 1 bit register
  - Flip flops and associated gates of any sequential circuit are called a register

Register Transfer Logic(cont.)

- The binary coded information stored in the registers
  - Binary information stored in registers may be binary numbers, binary coded decimal numbers, alphanumeric characters, control information or any other binary coded information
  - The operations performed on the data stored in registers depend upon the type of data
  - Numbers are manipulated with arithmetic operations
  - Control information is manipulated with logic operations such as setting and clearing specified bits in the register

Register Transfer Logic(cont.)

- Microoperation
  - 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
  - Eg. of microoperation - Shift, count, clear, add & load
  - A counter with parallel load - perform microoperations increment & load
  - Bidirectional shift register - Peform shift left & shift right microoperation
  - Binry parallel adder - Used for implementing add micooperation on the content of two registers that hold binary numbers
  - A microoperation requires one clock pulse for the execution if the operation done in parallel.

.

Register Transfer Logic(cont.)

Types of micooperations in digital system

- Interregister transfer microoperation
  - 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

Register Transfer Logic(cont.)

Categories of binary information found in registers of digital computers

- Numerical data such as binary numbers or binary coded decimal numbers used in arithmetic computations
- Nonnumerical data such as alphanumeric characters or any other binary coded symbols used for special applications
- Instruction codes, addresses and any other control information used to specify the data processing requirements in the system

Inter Register Transfer

- Registers in a digital system are designated by capital letters(sometimes followed by numerals) to denote function of register
- Eg :- Register that holds address of memory - Memory address register designated by MAR

  other designations are A, B, R1, R2 and IR
- The cells or flipflops of n-bit register are numbered in sequence from 1 to n (from 0 to n-1) starting either from left or from right

Inter Register Transfer(cont.)

- 4 ways to represent register
  - 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
  - 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)

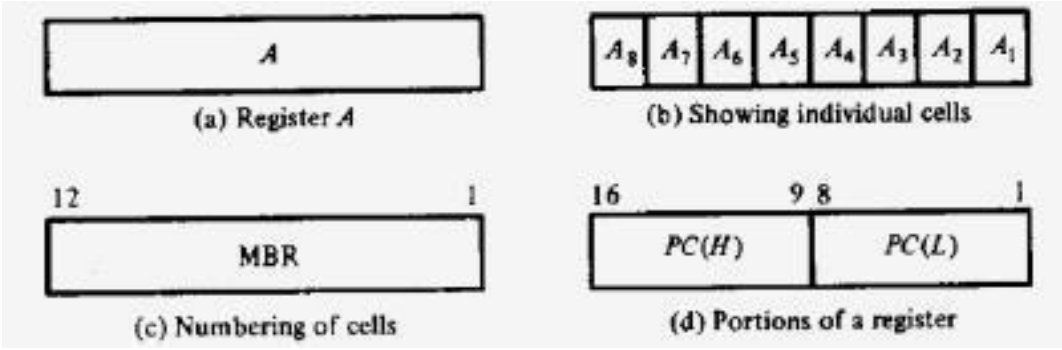


Figure:Block diagram of registers

Inter Register Transfer(cont.)

- Registers can be specified in a register transfer language with a declaration statement
- Eg :- Registers in the above figure can be defined with declaration statement such as

  DECLARE REGISTER A(8), MBR(12), PC(16)
- Information transfer from one register to another is designated in symbolic form by means of replacement operator

$$A \leftarrow B$$

Transfer of the contents of rgister B to register A.
Content of source register do not change

Inter Register Transfer(cont.)

- Sometimes transfer occures under a predefined condition called control function which is a boolean function equal to 1 or 0 Eg:-

$$x^1 T_1 : A \leftarrow B$$

- Control function terminated with a colon

Transfer occurs when $x^1 T_1 = 1$ ie, x=0 and $T_1 = 1$

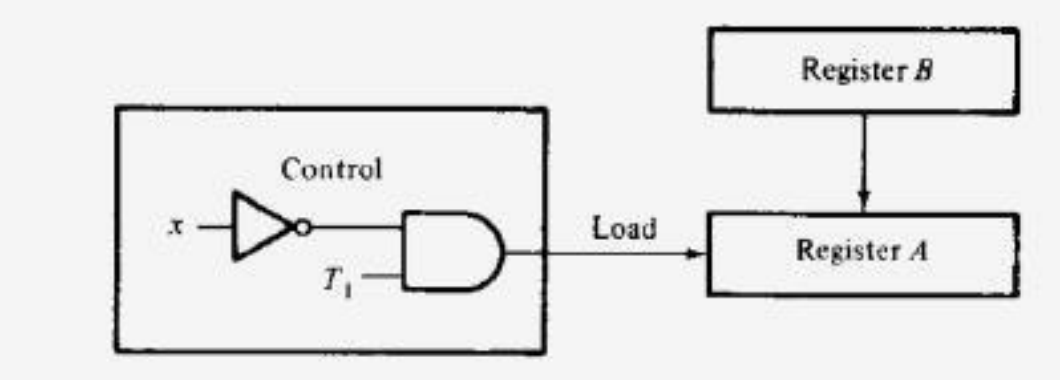


Figure:hardware implementation of the statement $x^t T_1 : A \leftarrow B$

Inter Register Transfer(cont.)

Table:Basic symbols for register trnsfer logic

Symbol	Description	Examples
Letters (and numerals)	Denotes a register	A, MBR, $R2$
Subscript	Denotes a bit of a register	A_2, B_6
Parentheses ( )	Denotes a portion of a register	$PC(H)$, $MBR(OP)$
Arrow $\leftarrow$	Denotes transfer of information	$A \leftarrow B$
Colon :	Terminates a control function	$x'T_0$:
Comma ,	Separates two microoperations	$A \leftarrow B$, $B \leftarrow A$
Square brackets [ ]	Specifies an address for memory transfer	$MBR \leftarrow M[MAR]$

Inter Register Transfer(cont.)

- Destination register receives information from two sources but not at the same time

$$T_1 : C \leftarrow A$$
$$T_5 : C \leftarrow B$$

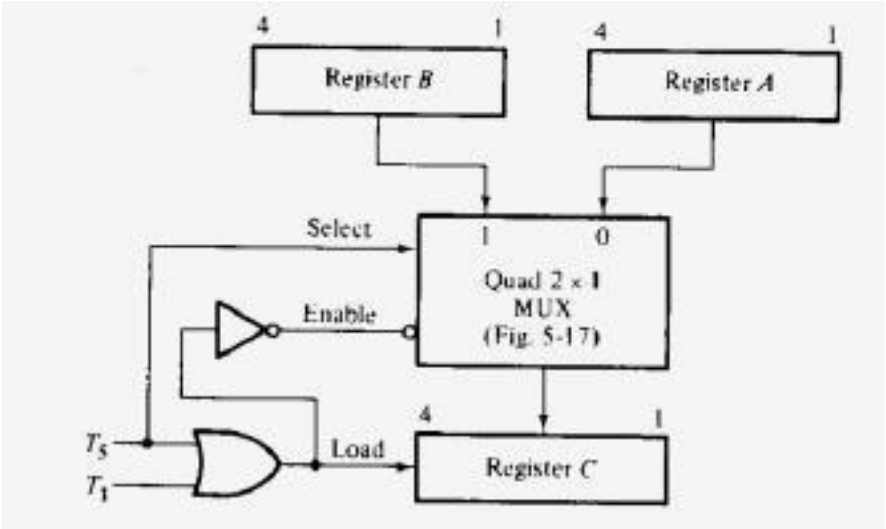


Figure:Use of multiplexer to transfer information from two sources into a single destination

Inter Register Transfer(cont.)

Bus transfer

- Paths must be provided to transfer information from one register to another in digital system

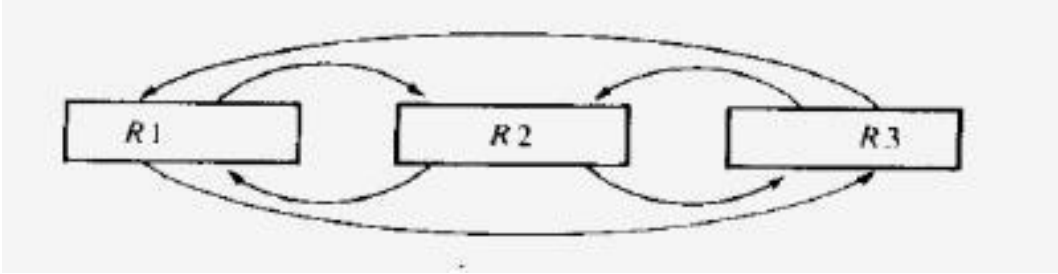


Figure:Transfer among threeregisters

- Eg :- Data tranfer among 3 registers - 6 data paths - each register requires a multiplexer to select between 2 sources

  If a register contains n flip flops - 6n lines - 3 multiplexers
- Number of registers increases number of interconnection lines and multiplexer increases

Inter Register Transfer(cont.)

- If restrict transfer one at a time then the number of paths among registers can be reduced

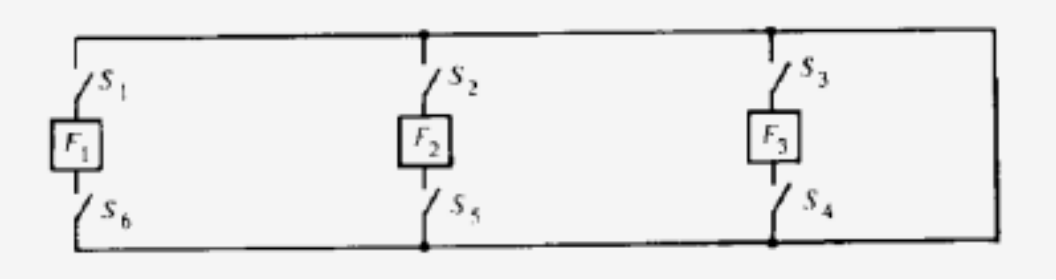


Figure:Transfer through one common line

- Each flipflop is connected to a common line through an electronic circuit that acts as a switch
- This scheme can be extended to registers with n flip flops and it requires n common lines

Inter Register Transfer(cont.)

- Bus :- Group of wires through which binary information is transferred
- For parallel transfer number of lines in the bus is equal to the number of bits in the registers
- A common bus system can be constructed with multiplexers and decoder
- Multiplexer selects one of the source register for the bus
- Decoder selects one destination register to transfer the information from the bus
- The 4-bits in the same significant position in the registers go through 4-to-1 line multiplexer to form one line of the bus
- 2 multiplexers are shown in figure - one for high order significant bit and other for low order significant bits
- For registers with n bits n multiplexers are needed to produce n line bus

Inter Register Transfer(cont.)

- n lines in the bus are connected to the n inputs of all registers
- The transfer of information from the bus into one destination register is accomplished by activting the load control of that register
- The particular load control activated is selected by outputs of the decoder when enabled.
- If decoder is not enabled no information will be transferred even though the multiplexers place the contents of a source register onto the bus
- Eg :- The statement $C \leftarrow A$ The control function that enables this transfer must select register A for the bus and register C for the destination
  The multiplexer and decoder selects inputs must be select
  source = 00(MUXs select register A)
  select destination = 10 (Decoder selects register C)
  Decoder enable =0 (Decoder is enabled)

Inter Register Transfer(cont.)

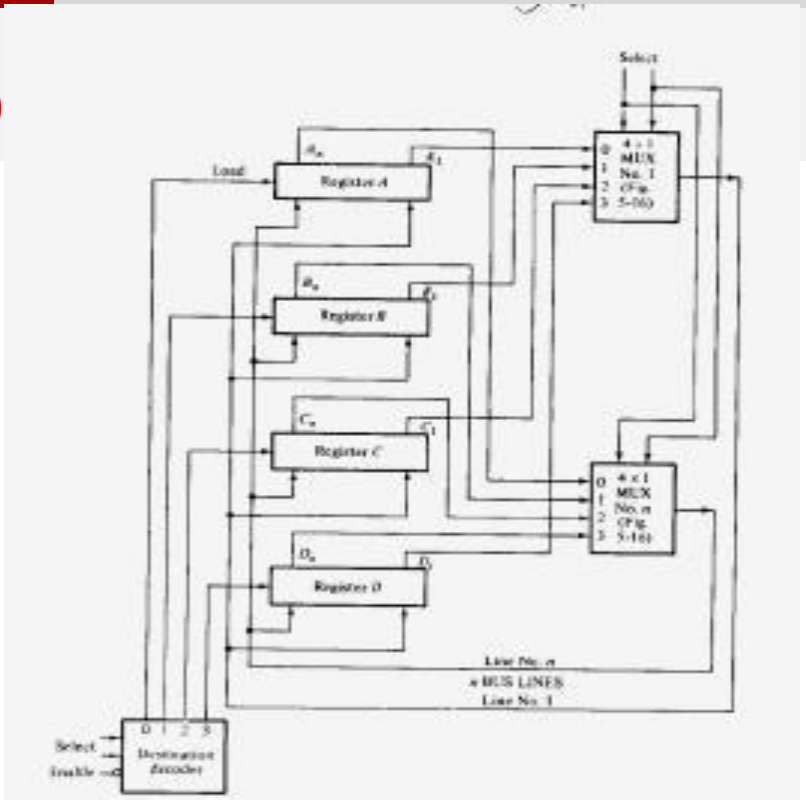


Figure:Bus system four registers

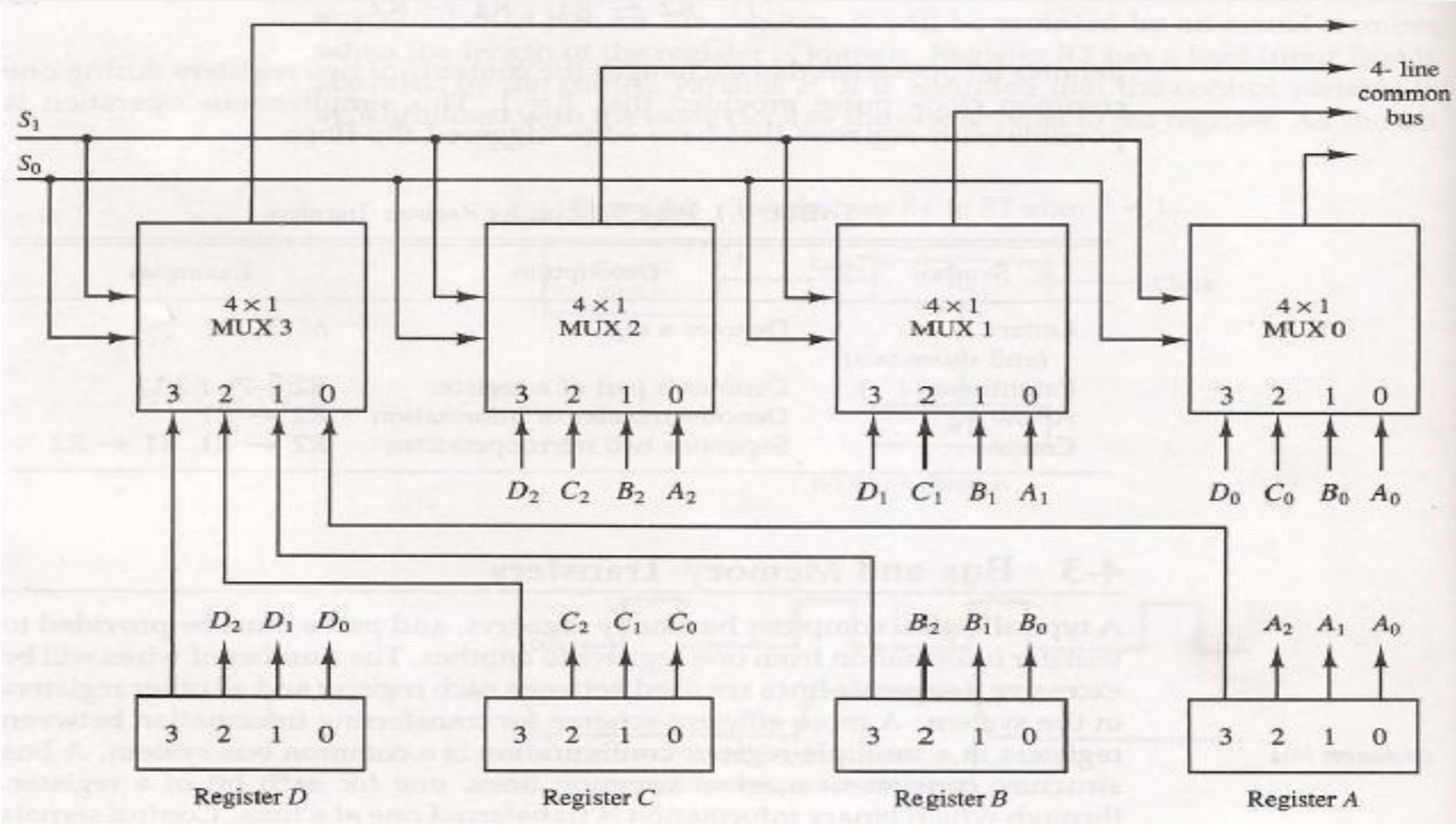
- on the next clock pulse the content of A, being on the bus, are loaded into register C

Inter Register Transfer(cont.)

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 below Figure.

Inter Register Transfer(cont.)



Inter Register Transfer(cont.)

- The bus consists of four 4 x 1 multiplexers each having four data inputs, 0 through 3, and two selection inputs, S_1 and S_0.
- For example, output 1 of register A is connected to input 0 of MUX 1 because this input is labelled A_1.
- 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 S1 and S0 are connected to the selection inputs of all four multiplexers.

Inter Register Transfer(cont.)

- The selection lines choose the four bits of one register and transfer them into the four-line common bus.
- When $S_1S_0 = 00$, 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 $S_1S_0 = 01$, and so on.
- Table 4-2 shows the register that is selected by the bus for each of the four possible binary value of the selection lines.

Inter Register Transfer(cont.)

S_1	S_0	Register selected
0	0	A
0	1	B
1	0	C
1	1	D

- In general a bus system has
  - ✓ multiplex “k” Registers

Inter Register Transfer(cont.)

- ✓ each register of "n" bits
- ✓ to produce "n-line bus"
- ✓ no. of multiplexers required = n
- ✓ size of each multiplexer = k x 1

- When the bus is includes in the statement, the register transfer is symbolized as follows:
- BUS← C, R1← 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.
- R1← C

Inter Register Transfer(cont.)

Memory transfer

- Read operation :- Transfer of information from memory register to outside enviornment
- Write operation :- Transfer of new information into memory register selected
- Memory register is selected by an address Memory register is symbolised by the letter M

Inter Register Transfer(cont.)

- Only one address register is connected to the address terminals of memory
  - This register specifies the address
  - The letter M stands by itself in a statement designate a memory register selected by the address presently in MAR
  - A single memory buffer register MBR used to transfer data into and out of memory
  - 2 memory transfer operations - Read & Write
  - Read operation :- Transfer of information from selected memory register M specified by the address in MAR into MBR

    $$R : MBR \leftarrow M$$

    R is the control function that initiate the read operation
  - Write operation :- Trasfer of information from MBR into the register M selected by the address presently in MAR

    $$W : M \leftarrow MBR$$

    W is the control function that intiate write operation

Inter Register Transfer(cont.)

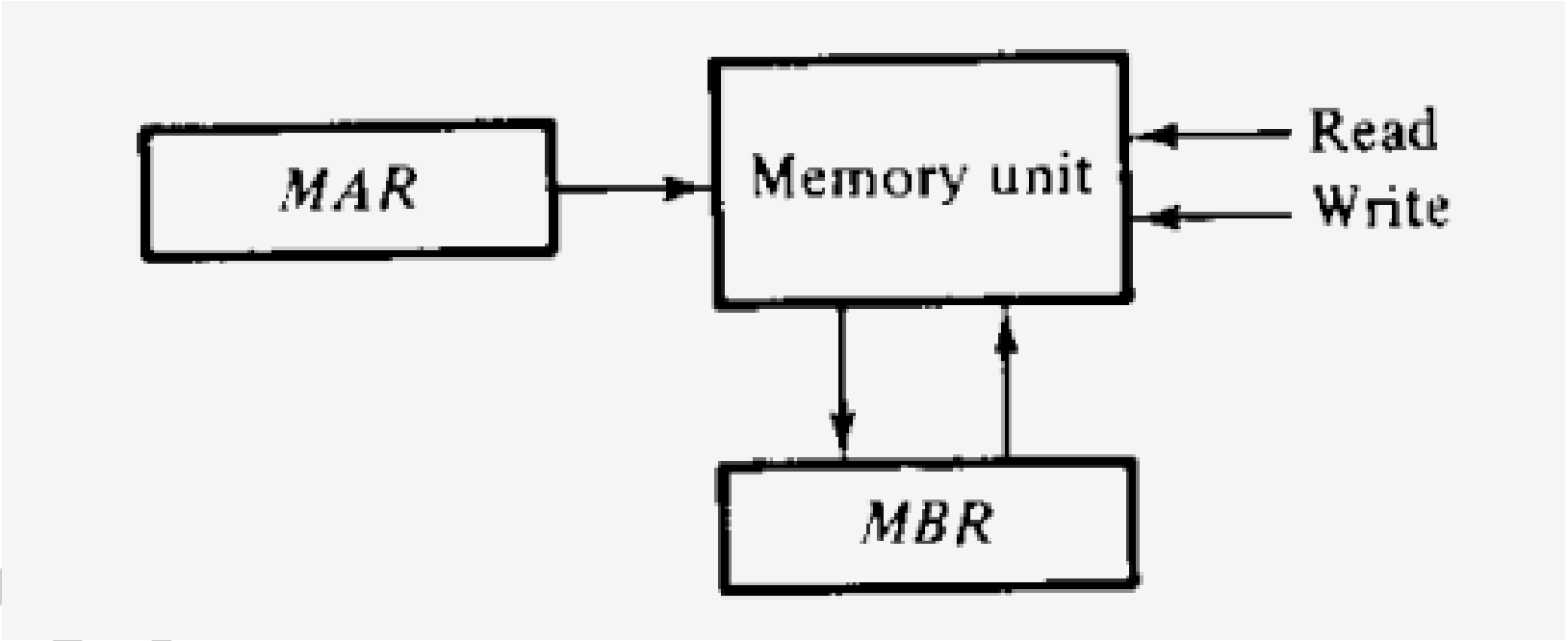


Figure:Memory unit that communicate with two external registers

Inter Register Transfer(cont.)

- The address lines form a common bus system to allow many registers to specify an address
  - The register specifies the address will be enclosed within square bracket after the symbol M
  - The address to the memory unit comes from an address bus
  - 4 registers are connected to the bus and any one may supply an address
  - The output of the memory can go to any one of 4 registers selected by the decoder
  - Data input to the memory comes from the data bus which selects one of the four registers
  - Write operation :- The transfer of information from register B2 to a memory word selected by the register A1 is

$$\text{W: } M[A1] \leftarrow B2$$

  - Read operation :- R : $B0 \leftarrow M[A3]$

Inter Register Tran

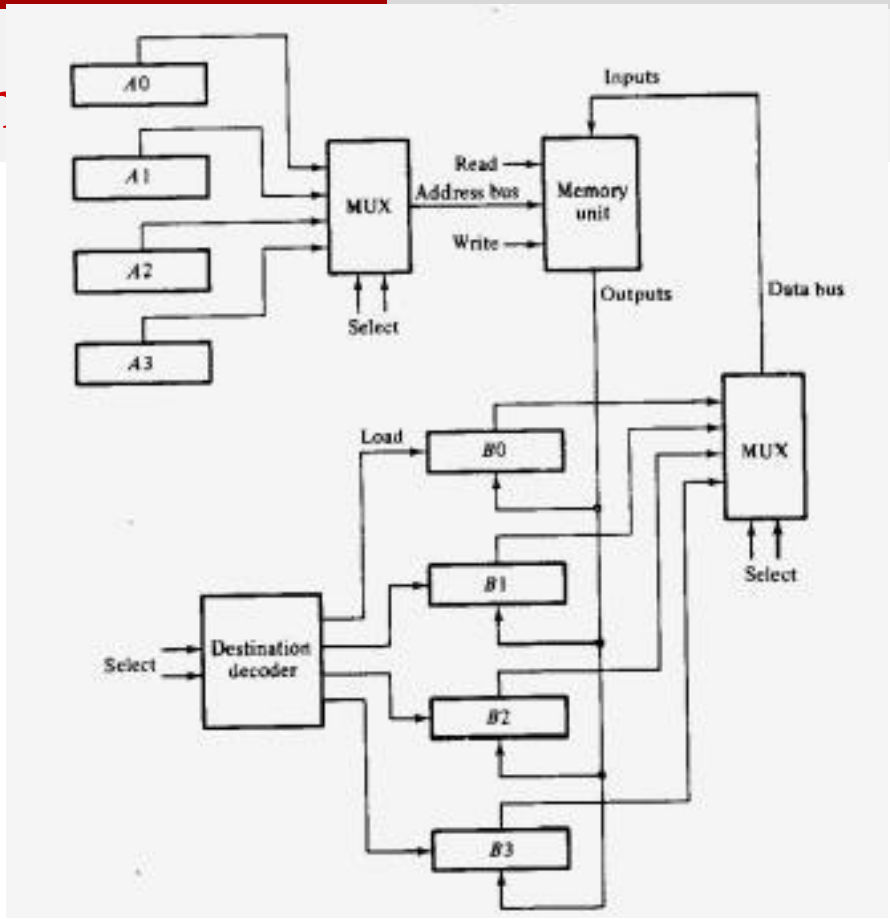


Figure:Memory unit that communicate with multiple registers

Arithmetic Microoperation

- Basic arithmetic microoperations - add, subtract, complement & shift
- 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

Arithmetic Microoperation(cont.)

Table: Arithmetic microoperation

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)

- Increment micropoperation symbolised by plus-one & implemented with an up counter
- Decrement micropoperation symbolised by minus-one & implemented with an down counter

Arithmetic Microoperation(cont.)

- 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

$$T_2 : A \leftarrow A + B$$
$$T_5 : A \leftarrow A + 1$$

Timing variable T_2 initiates an operation to add the contents of register B to the present contents of A with a paralleladder.

Timing variable T_5 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.

Arithmetic Microoperation(cont.)

- 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 T_2 loads the sum into register A. Timing variable T_5 increments there by enabling increment input register
- Multiplication - Implemented with sequence of add &shift microoperations
- Division - Implemented with sequence of subtract &shift microoperations

Arithmetic M

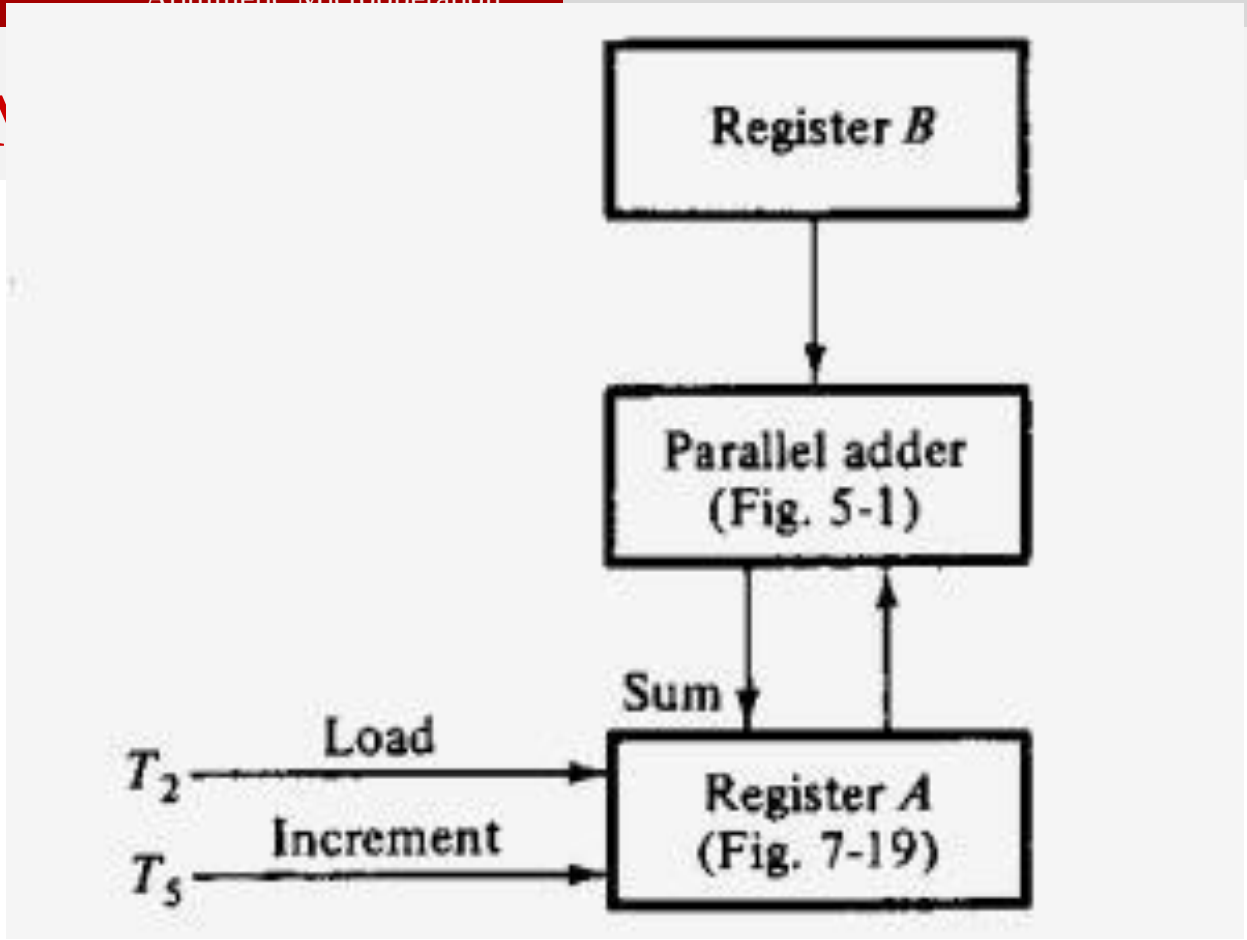


Figure:Implementation for addd and incrment microoperation

Logic Micro operations

- Specify binary operations for a string of bits stored in registers
- Consider each bit in the registers separately and treat as a binary variable
- Exclusive OR operation is symbolised by the statement

$$F \leftarrow A \oplus B$$

- The + symbol has different meaning

$$T_1 + T_2 : A \leftarrow A + B,\ C \leftarrow D \vee F$$

The + between T_1 and T_2 is an OR opeartion between 2 timing variables of a control function

The + between A & B specifies an add microoperation

Logic Micooperation(cont.)

Table:Logic & shift microoperation

Symbolic designation	Description
$A \leftarrow \bar{A}$	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

Shift Micooperation

- Transfers binary information between registers in serial computers Used in parallel computers for arithmetic, logic and control operations
- Registers are shifted to the left or to the right
- No conventioal symbol for shift operation Here
- adopt symbols shl or shr
  shl - shift left
  shr - shift right
- Eg :-

$$A \leftarrow shl\ A \text{ - 1-bit shift to the left of register A}$$

$$B \leftarrow shr\ B \text{ - 1-bit shift to the right of register B}$$

- While the bits are shifted extreme flip flops receive information from the serial input

Shift Micooperation(cont.)

- Information transferred to extreme flip flop is not specified by shl or shr symbols
- Shift operation is accompanied with other microopeartion that specifies the value of the serial input for the bit transfer into the extreme fip flop
- Eg:-

  $A \leftarrow shl$, $A_1 \leftarrow A_n$ - Circular shift that tranfers the leftmost bit from A_n into the rightmost flipflop A_1

  $A \leftarrow shr$, $A_n \leftarrow E$ - Shift right opration with the leftmost flip flop A_n receiving the value of the 1-bit register E

Conditional Control Statements

- Specify a control condition by a conditional statement rather than a boolean control function
- Symbolised by if-then-else statement

  P : If(condition) then [microoperation(s)] else [microoperation(s)]

  mean that if control condition stated within the parentheses after the word if is true, then microoperation enclosed within the parentheses after the word then is executed otherwise the microoperation listed within after the word else is executed
- In any case the control function P must occur for anything to be done
- If else part of the statement is missing then nothing is executed if the condition is not true

Conditional Control Statements(cont.)

- Eg:-

$$T_2 : \text{If (C=0) then } (F \leftarrow 1) \text{ else } (F \leftarrow 0)$$

- F is asssumed to be 1-bit register(flip flop) that can be set or cleared. If register C is a 1-bit register the statement is equivalent to the following statements

$$C' T_2 : F \leftarrow 1$$

$$C T_2 : F \leftarrow 0$$

- Same timing variable can occur in two separate control function. The valu of C is 0 or 1. So only one microoperation will be executed during T_2 depending on the value of C.
- If C has more than 1 bit the condition C = 0 means that all bits of C must be 0.
- Register C has 4 bits C_1, C_2, C_3 and C_4 the condition C=0 can be expressed with boolean function

Conditional Control Statements(cont.)

$$x = C_1^1 C_2^1 C_3^1 C_4^1 = (C_1 + C_2 + C_3 + C_4)^1$$

variable x can be generated with a NOR gate.

- Conditional control statements now equivalent to 2 statements

$$xT_2 : F \leftarrow 1$$
$$x^J T_2 : F \leftarrow 0$$

variable x = 1 if C = 0 but is equal to 0 if $C \neq 0$