Computer Organisation & Architecture.

(*) **Computer Architecture:**
→ concerned with the structure & behaviour of the comp. as viewed by a user
→ it includes the info, formats, instruction set and techniques for addressing memory.

(*) **Computer Organisation:** (implementation of architecture of comp)
→ concerned with the way the hardware components operate and the way they are connected together to form a comp. system.

(*) **Computer Design:**
→ concerned with the hardware design of the comp. Once the specifications are formulated, it is the task of designer to develop hardware for system.

MICRO-OPERATIONS:

→ The operations executed on data stored in registers.
Eg: shift, count, clear & load.

Types:

(1) Data transfer micro-operation:
a) Register transfer micro-operatin.
→ transfer data from one register to another
b) Memory transfer micro operation:
→ transfer data from register to memory or memory to register.

(2) Arithmetic microoperations
→ perform arithmetic operations on data stored in registers.

(3) Logic microoperation.
→ perform bit manipulation operations on non-numeric data stored in registers.

(4) Shift microoperations:
→ these shift the data in left or right direction stored in registers

REGISTER TRANSFER

→ transfer of info b/w registers with help of common bus.

→ The symbolic notation used to describe the microoperation transfers among registers is called a register transfer language.

→ A register transfer language is a system for expressing in symbolic form, the microoperation sequences among the registers of a digital module.

(*) Computer registers → designated by capital letters. (followed by numerals sometimes)

Eg: Register holding address for memory unit : MAR (memory address register)

Others → PC (program counter)
IR (instruction register)
R1 (processor register)

Info transfer from one register to another: $R_2 \leftarrow R_1$ (After transfer, content of this doesn't change)

(*) It designates a replacement of content of R_2 by the content of R_1.

(*) Normally, we want transfer to occur only under a predetermined control condition.

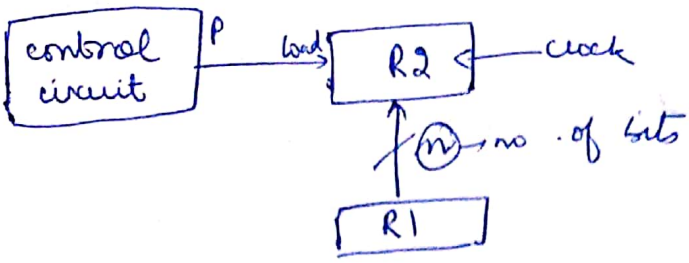
If $(P=1)$ then $(R_2 \leftarrow R_1)$

Where P = control signal generated in control section.

$P: R_2 \leftarrow R_1 \Rightarrow$ control func.

↓

Boolean variable (0 or 1)

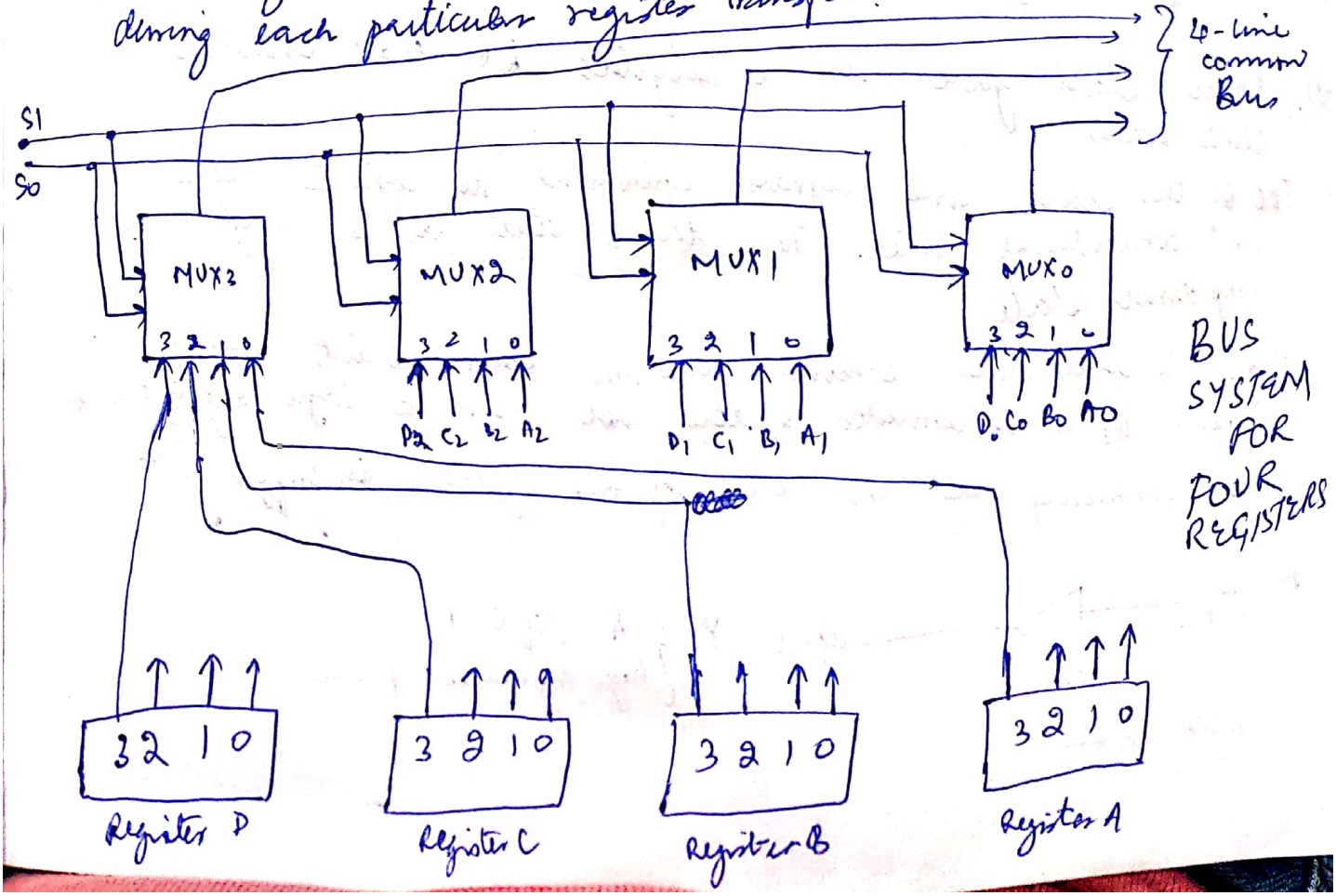


Block diagram

Symbol	Description	Examples
Letters (& numerals)	denotes register	MAR, R2
Parenthesis ( )	denotes part of register	R2(0-7), R2(L)
Arrow (←)	denotes transfer of info	R2 ← R1
comma (,)	separates 2 microoperations	R2 ← R1, R1 ← R2

#BUS TRANSFER:

- A typical digital comp has many registers & paths may be provided to transfer info from one register to another.
- A more efficient scheme for transfering info b/w 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, to carry data]. connecting several devices.
- Control signals determine which register is selected by the bus during each particular register transfer.



BUS SYSTEM FOR FOUR REGISTERS

Function Table of BUS:

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

In general, bus system with 'k' registers of 'n' bits produce 'n' line o/p

no. of MUX needed = n

size of each mux = k x 1

The transfer of info from a bus into one of many destination registers can be accomplished by connecting the bus lines to the input of all destination registers & activating the load control of particular destination register selected.

Symbolic: BUS ← C, R1 ← BUS.

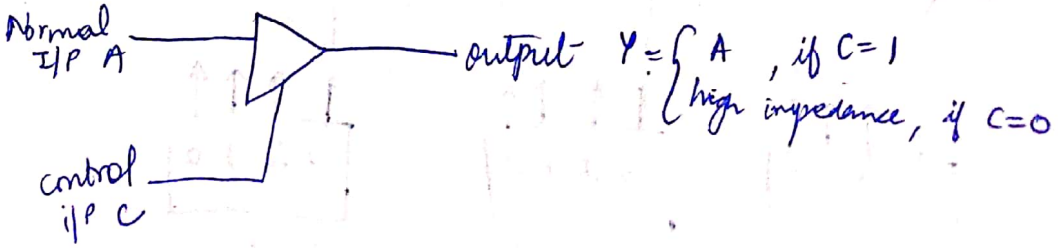
THREE-STATE BUS BUFFER:

→ A three state gate is a digital ckt that exhibits three states.

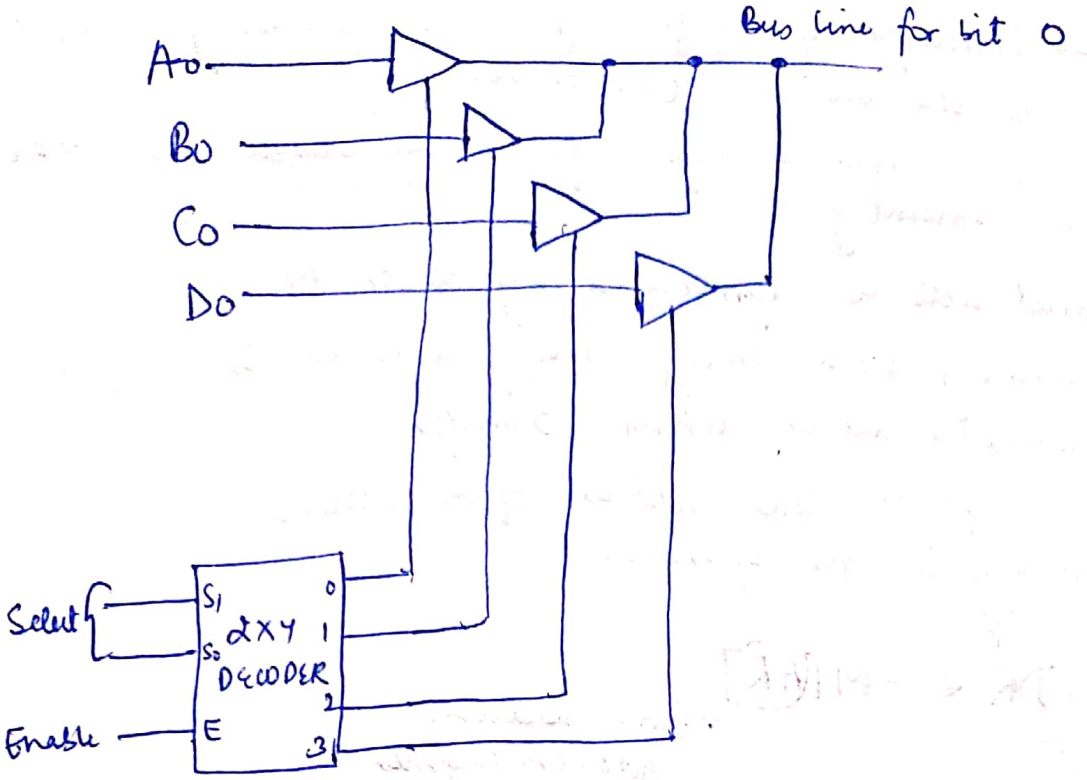
→ Two of the states are signals equivalent to logic 1 and 0 as in conventional gate. The third state is a high impedance state.

→ High impedance state behaves like an open circuit, which means o/p is disconnected & does not have a logic significance.

→ Most commonly used in design of bus system is buffer gate.



BUS USING TRI-STATE BUFFER



→ No more than 1 buffer may be in active state at any given time.

→ The connected buffers may be controlled so that only one of these tri-state buffer has access to the bus line while all other buffers are maintained in a high impedance state.

→ One way to ensure that not more than one control input is active at any given time is to use a decoder.

→ When E = 0, all 4 o/p are zero & bus line is in high impedance state because all 4 buffers are disabled.
When E = active, one of the four 3-state buffers will be active, depending on select inputs of decoder.

→ To construct common bus for 4 registers of 'n' bits each using tri-state buffer, we need 'n' circuits with 4 buffers in each.
Each group of 4 buffers receives one significant bit from the four registers
Only one decoder is necessary to select s/w 4 registers.

MEMORY TRANSFER:

Read operation → transfer of info from a memory word to the outside environment.

Write operation → transfer of new info to be stored into the memory

⊙ A memory word will be symbolised by letter M. The particular memory word among many available is selected by memory address during transfer.

⊙ It is necesary to specify the address of M when writing memory transfer operations.

Eg: Read : DR ← M[AR] (gives address) Address Register

Data register

↳ transfer of info into DR from memory word M selected by address in AR.

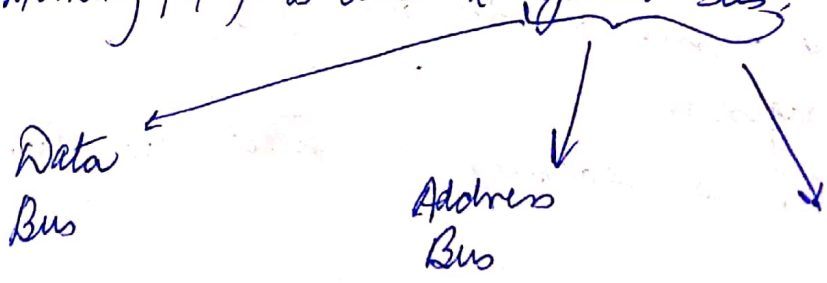
Write : M[AR] ← R1

↳ transfer of info from R1 to memory word M selected by the address in AR.

BUS ARCHITECTURE:

→ A group of wires connecting 2 or more devices & providing a path to perform communication is called a BUS.

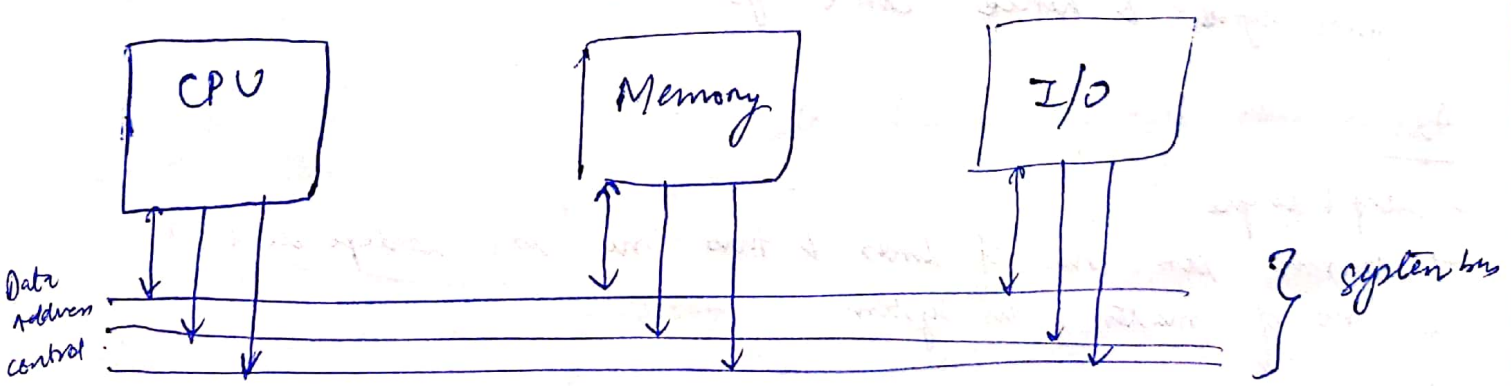
→ A bus that connects major components/modules (CPU, Memory, I/O) is called a system bus.



1. Data Bus:
→ consists of 8, 16, 32 or more parallel lines. The data bus lines are bidirectional.
→ This means the CPU can read data on these lines from memory or from a port as well as data out on these lines to a memory location or to a port.
→ it is a bidirectional bus used to carry data b/w system units.

2. Address Bus:
→ unidirectional bus.
→ consists of 16, 20, 24 or more parallel lines
→ The CPU sends out the address of the memory location or I/O port that is to be written or read from by using this address bus.
→ its function is to select the particular device or memory location.

3. Control Bus:
→ control lines regulate activity on the bus.
→ CPU sends signals on the control bus to enable the outputs of the addressed memory device or port device.
→ it is also unidirectional bus. Its function is to control the units based on the request.



→ no. of lines in data bus = width of data bus.
= 16, 32, 64 ...
→ Each line carries 1 bit at a time, so the no. of lines in data bus determine how many bits can be transferred parallelly.
→ Data bus width determines overall system performance.

Address Bus width determines memory capacity

7

BUS ARBITRATION

A mechanism which decides the selection of current master to access bus.

Three types of mechanisms:

1. Daisy chaining
2. Parallel arbitration
3. Independent Requesting.

1) Daisy chaining

→ cheap & simple method.

→ All masters make use of same line for bus request. The bus grant signal serially propagates through each master until it encounters the first one that is requesting access to the bus. This master blocks the propagation of bus grant signal, activates the busy line & gains control of the bus. Therefore, any other requesting module will not receive the grant signal & hence can't get bus access.

Adv:

→ cheap & single

→ requires less no of lines & this no. is independent of no. of masters in system.

Disadv

→ propagation time delay of bus grant signal makes its slow & limit the no. of masters.

→ priority of master is fixed by physical location of masters.

→ Failure of any one master causes whole system to fail.

(2) Parallel Arbitration:

→ It consists of priority encoder & a decoder.

→ In this mechanism, each bus arbiter has a bus request output line & a bus acknowledges input line.

→ Each arbiter enables the request line when its device (processor) is requesting access to system bus.

→ Each arbiter bus request output line is connected to the i/p of priority encoder.

→ The output of encoder generates a 2-bit code which represents highest priority unit among those requesting.

→ This 2-bit code is given to i/p of 2×4 decoder which enables the proper acknowledge line to grant bus access to highest priority unit.

→ A device is utilized in the system only after it receives the acknowledge signal.

Adv:

→ priority can be changed by altering priority sequence stored in controller.

→ If one module fails, entire system does not fail.

(3) Independent Priority:

→ In this scheme, each master has a separate pair of bus request & bus grant lines & each pair has a priority assigned to it.

→ The built in priority encoder within the controller selects the highest priority & asserts the corresponding bus grant signal.

→

ARITHMETIC MICROOPERATIONS:

elementary operation performed with data stored in registers.

→ The basic arithmetic microoperations are: addition, subtraction, increment, decrement & shift.

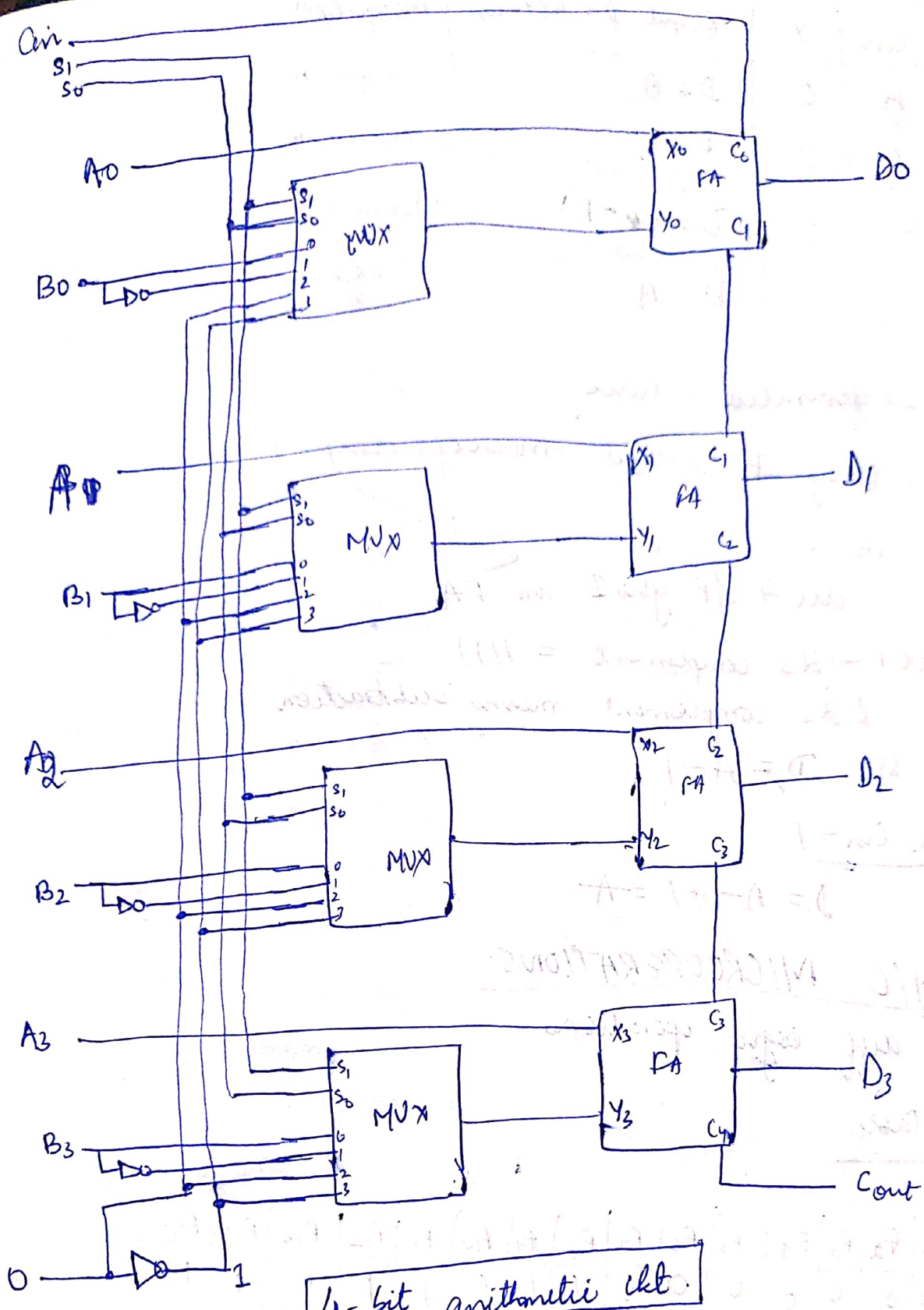
Symbolic Designation	Description
$R_3 \leftarrow R1 + R2$	contents of R1 plus R2 transferred to R3
$R3 \leftarrow R1 - R2$	contents of R1 minus R2 transfd to R3
$R2 \leftarrow \overline{R2}$	complement the contents of R2 (1's complement)
$R2 \leftarrow \overline{R2} + 1$	2's complement of contents of R2 (negate)
$R3 \leftarrow R1 + \overline{R2} + 1$	R1 plus 2's complement of R2 (subtraction)
$R1 \leftarrow R1 + 1$	increment by 1
$R1 \leftarrow R1 - 1$	decrement by 1

implemented with updown counter.

(*) Multiplication is implemented by sequence of add & shift microoperation.

(*) Division is implemented by sequence of subtract & shift microoperations.

ARITHMETIC CIRCUIT



4-bit arithmetic ckt.

S1	S0	Cin	I/P Y	O/P D = A+Y+Cin	M=0
0	0	0	B	D = A+B	Add
0	0	1	$\bar{B}$	D = A+B+1	Add with carry
0	1	0	$\bar{B}$	D = A+$\bar{B}$	subtract with borrow
0	1	1	$\bar{B}$	D = A+$\bar{B}$+1	subtract

S_1	S_0	C_{in}	Y	Output $D = A+Y+C_{in}$	Micro-Op^n
1	0	0	0	$D = A$	Transfer A
1	0	1	0	$D = A+1$	Increment A
1	1	0	1	$D = A-1$	Decrement A
1	1	1	1	$D = A$	Transfer A

$D = A$ → generated twice

∴ Only 7 distinct microoperations.

When $C_{in} = 0$

all 4 i/p goes 1 in FA

1 → 0001 → 2's complement = 1111

& 2's complement means subtraction.

So, $D = A - 1$

When $C_{in} = 1$

$D = A - 1 + 1 = A$

LOGIC MICROOPERATIONS

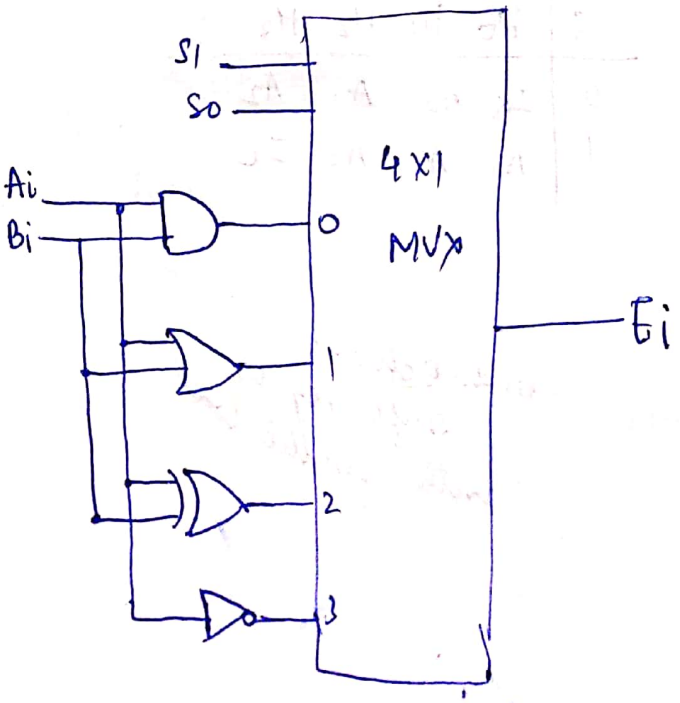
16 diff logic operations.

Truth Table

S_1 S_0	F_0	F_1	F_2	F_3	F_4	F_5	F_6	F_7	F_8	F_9	F_{10}	F_{11}	F_{12}	F_{13}	F_{14}	F_{15}
0 0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
0 1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
1 0	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
1 1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1

Boolean func	Microoperation	Name
$F_0 = 0$	$F \leftarrow 0$	Clear
$F_1 = xy$	$F \leftarrow A \wedge B$	AND
$F_2 = xy'$	$F \leftarrow A \wedge \bar{B}$	
$F_3 = x$	$F \leftarrow A$	Transfer A
$F_4 = x'y$	$F \leftarrow \bar{A} \wedge B$	
$F_5 = y$	$F \leftarrow B$	Transfer B
$F_6 = x \oplus y$	$F \leftarrow A \oplus B$	EX-OR
$F_7 = x + y$	$F \leftarrow A \vee B$	OR
$F_8 = (x+y)'$	$F \leftarrow \overline{A \vee B}$	NOR
$F_9 = (x \oplus y)'$	$F \leftarrow \overline{A \oplus B}$	EX-NOR
$F_{10} = y'$	$F \leftarrow \bar{B}$	Complement B
$F_{11} = x + y'$	$F \leftarrow A \vee \bar{B}$	
$F_{12} = x'$	$F \leftarrow \bar{A}$	Complement A
$F_{13} = x' + y$	$F \leftarrow \bar{A} \vee B$	
$F_{14} = (xy)'$	$F \leftarrow \overline{A \wedge B}$	NAND
$F_{15} = 1$	$F \leftarrow$ all 1's	set to all 1's.

* Hardware Implementation



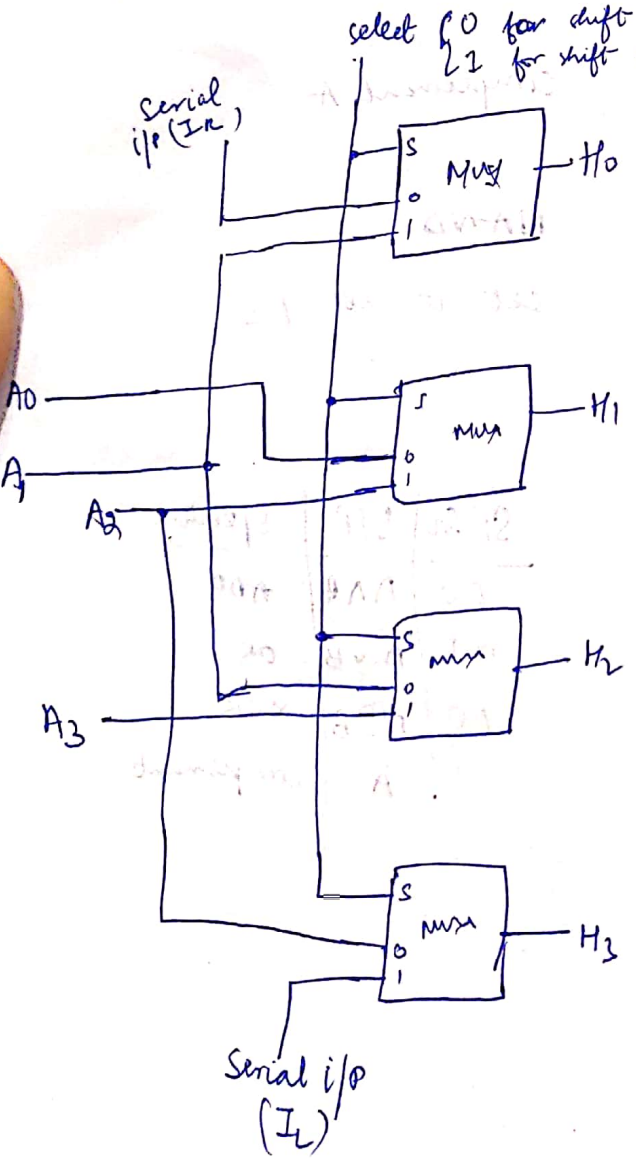
S1	S0	O/P	Operation
0	0	$A \wedge B$	AND
0	1	$A \vee B$	OR
1	0	$A \oplus B$	XOR
1	1	$\bar{A}$	complement

SHIFT MICROOPERATIONS :

→ used for serial transfer of data

Types of shifts → logical (0 - left shift to left/right)
→ circular (around 2 ends)
→ arithmetic (shifts signed binary no. to left/right)
→ left (multiplies by 2)
→ right (divides by 2)

Symbol designation	Description
R ← shl R	shift - left register R
R ← shr R	shift - right register R
R ← cil R	circular shift left register R
R ← cir R	circular shift right register C
R ← ashl R	arithmetic shift-left R
R ← ashr R	arithmetic - shift-right R



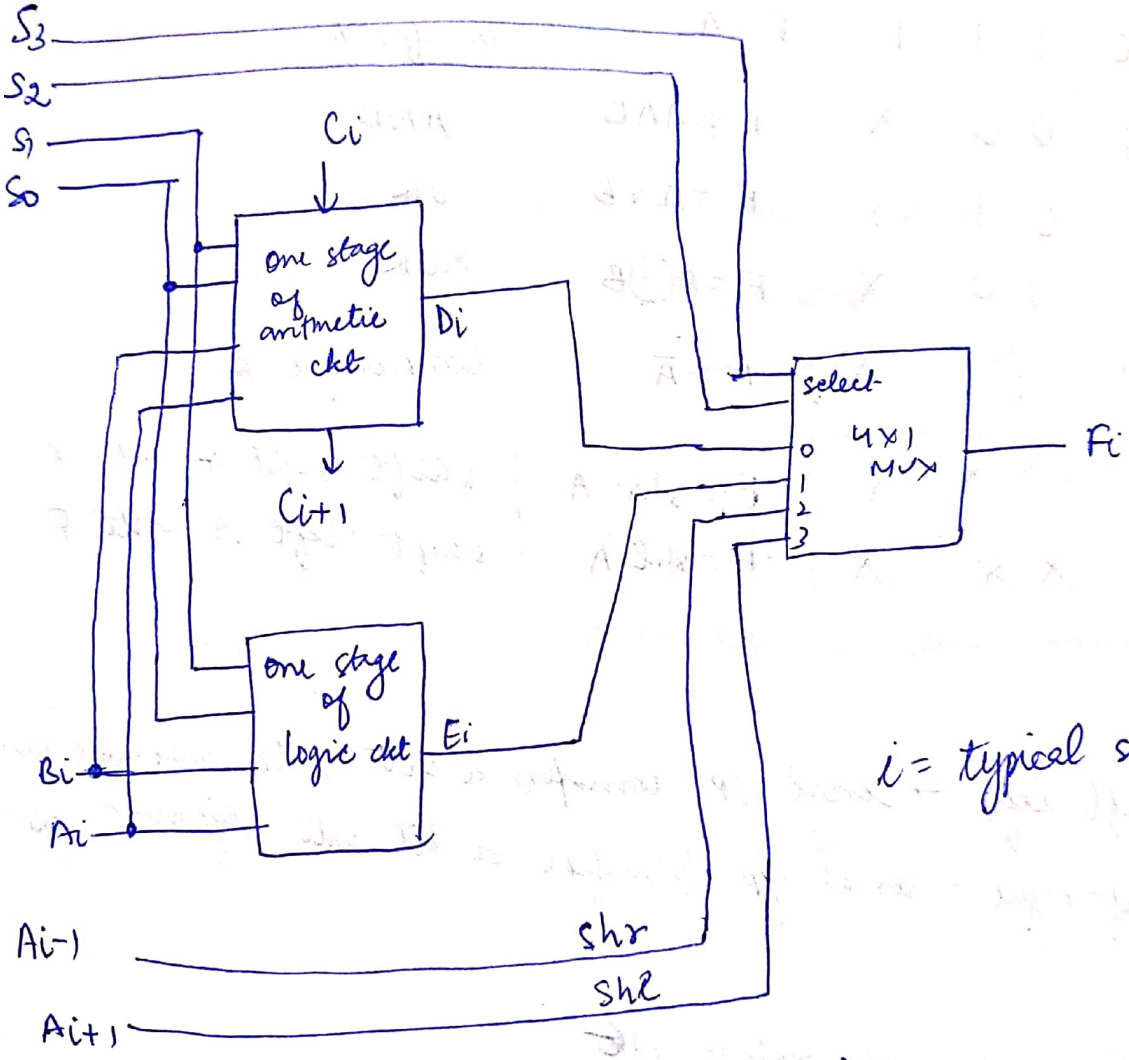
S	o/p H0	H1	H2	H3
0	IR	A0	A1	A2
1	A1	A2	A3	IL

bidirectional shift register with parallel load.

ARITHMETIC LOGIC SHIFT UNIT :

→ Instead of having individual registers performing micro operations directly, computer systems employ a no. of storage registers connected to common operational unit called an ALU.

→ ALU is a combinational ckt so that entire register transfer operation from source register through the ALU & into the destination register can be performed during one clock pulse period.



i = typical stage

One stage of Arithmetic Logic Shift Unit

S_3	S_2	S_1	S_0	C_{in}	Operation	Function
0	0	0	0	0	$F = A$	Transfer A
0	0	0	0	1	$F = A + 1$	Increment A
0	0	0	1	0	$F = A + B$	Addition
0	0	0	1	1	$F = A + B + 1$	Add with carry
0	0	1	0	0	$F = A + \bar{B}$	Subtraction with borrow
0	0	1	0	1	$F = A + \bar{B} + 1$	Subtraction
0	0	1	1	0	$F = A - 1$	decrement A
0	0	1	1	1	$F = A$	Transfer A
0	1	0	0	X	$F = A \wedge B$	AND
0	1	0	1	X	$F = A \vee B$	OR
0	1	1	0	X	$F = A \oplus B$	XOR
0	1	1	1	X	$F = \bar{A}$	Complement A
1	0	X	X	X	$F = shr\ A$	shift right A into F
1	1	X	X	X	$F = shl\ A$	shift left A into F.

Shift-left → serial i/p transfers a bit into rightmost pos^n

shift right → serial i/p transfers a bit into leftmost pos^n.

[illegible]