REGISTER TRANSFER AND MICROOPERATIONS

- Register Transfer Language
- Register Transfer
- Bus and Memory Transfers
- Arithmetic Microoperations
- Logic Microoperations
- Shift Microoperations
- Arithmetic Logic Shift Unit

REGISTER TRANSFER LANGUAGE

- Rather than specifying a digital system in words, a specific notation is used, , *register transfer language*
- For any function of the computer, the register transfer language can be used to describe the (sequence of) microoperations
- Register transfer language
  - A symbolic language
  - A convenient tool for describing the internal organization of digital computers
  - Can also be used to facilitate the design process of digital systems.

REGISTER TRANSFER

- A register transfer such as

  $R3 \leftarrow R5$

  Implies that the digital system has

  - the data lines from the source register (R5) to the destination register (R3)
  - Parallel load in the destination register (R3)
  - Control lines to perform the action

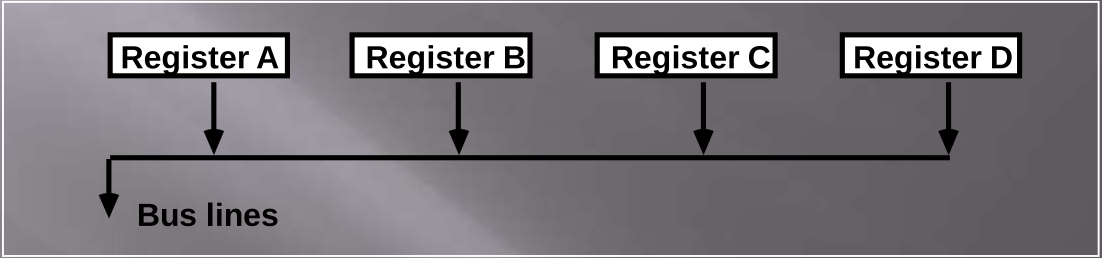
BASIC SYMBOLS FOR REGISTER TRANSFERS

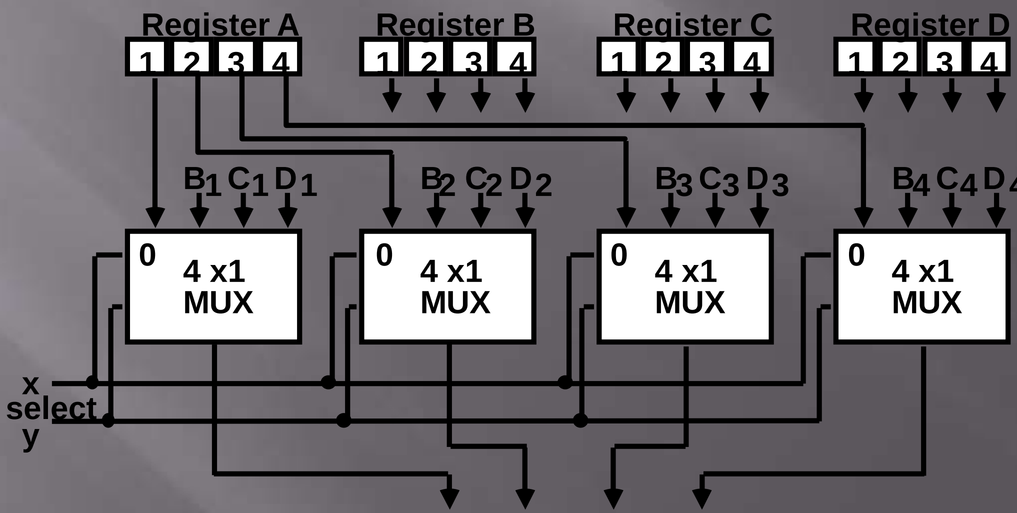
Symbols	Description	Examples
Capital letters & numerals	Denotes a register	MAR, R2
Parentheses ()	Denotes a part of a register	R2(0-7), R2(L)
Arrow $\leftarrow$	Denotes transfer of information	$R2 \leftarrow R1$
Colon :	Denotes termination of control function	P:
Comma ,	Separates two micro-operations	$A \leftarrow B$, $B \leftarrow A$

BUS AND BUS TRANSFER

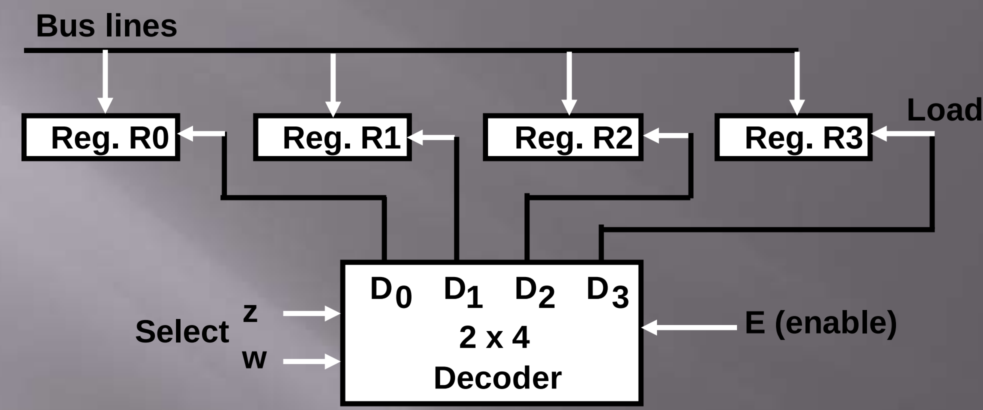
Bus is a path(of a group of wires) over which information is transferred, from any of several sources to any of several destinations.

From a register to bus: BUS ← R

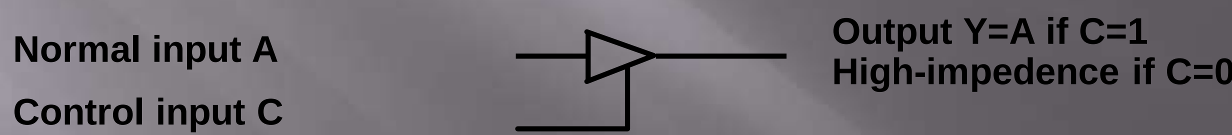




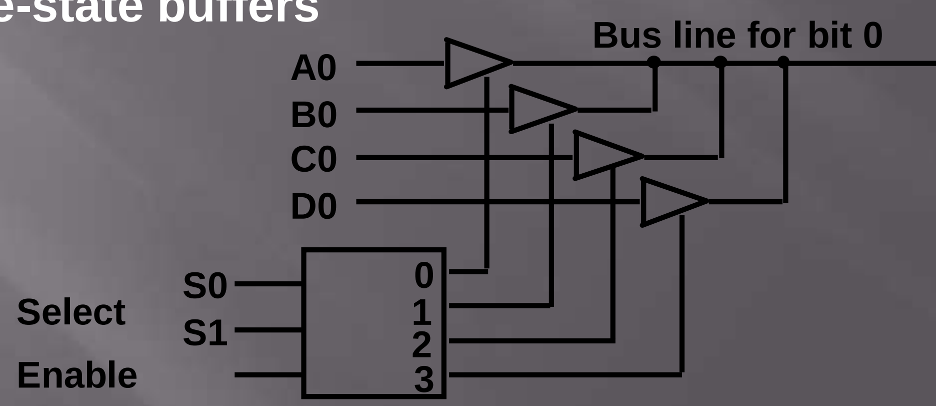
TRANSFER FROM BUS TO A DESTINATION REGISTER



Three-State Bus Buffers



Bus line with three-state buffers



BUS TRANSFER IN RTL

- Depending on whether the bus is to be mentioned explicitly or not, register transfer can be indicated as either

R2 ← R1

or

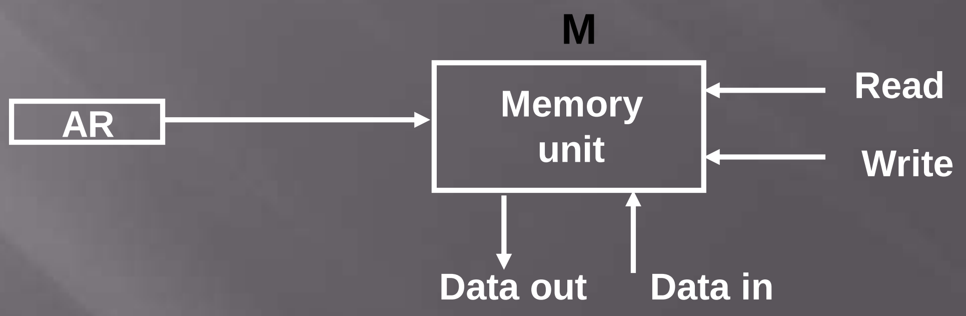
BUS ← R1, R2 ← BUS

- In the former case the bus is implicit, but in the latter, it is explicitly indicated

MEMORY TRANSFER

- Collectively, the memory is viewed at the register level as a device, M.
- Since it contains multiple locations, we must specify which address in memory we will be using
- This is done by indexing memory references

- Memory is usually accessed in computer systems by putting the desired address in a special register, the *Memory Address Register* (*MAR*, or *AR*)
- When memory is accessed, the contents of the MAR get sent to the memory unit's address lines



MICROOPERATIONS

- Arithmetic microoperations

- Logic microoperations

- Shift microoperations

ARITHMETIC MICROOPERATIONS

- The basic arithmetic microoperations are
  - Addition
  - Subtraction
  - Increment
  - Decrement

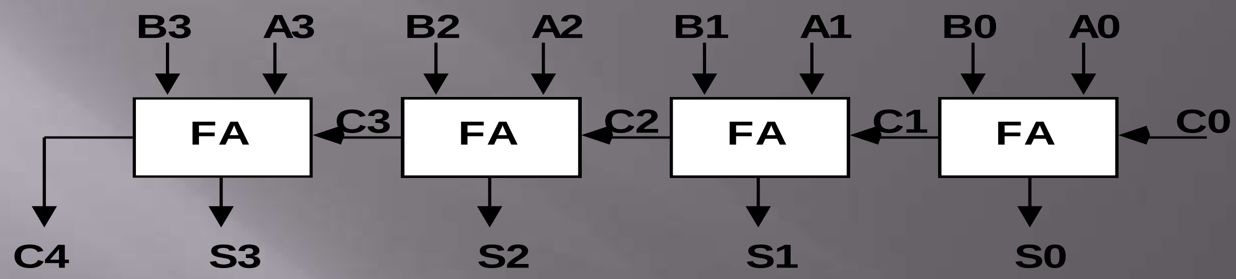
- The additional arithmetic microoperations are
  - Add with carry
  - Subtract with borrow
  - Transfer/Load
  - etc. …

Summary of Typical Arithmetic Micro-Operations

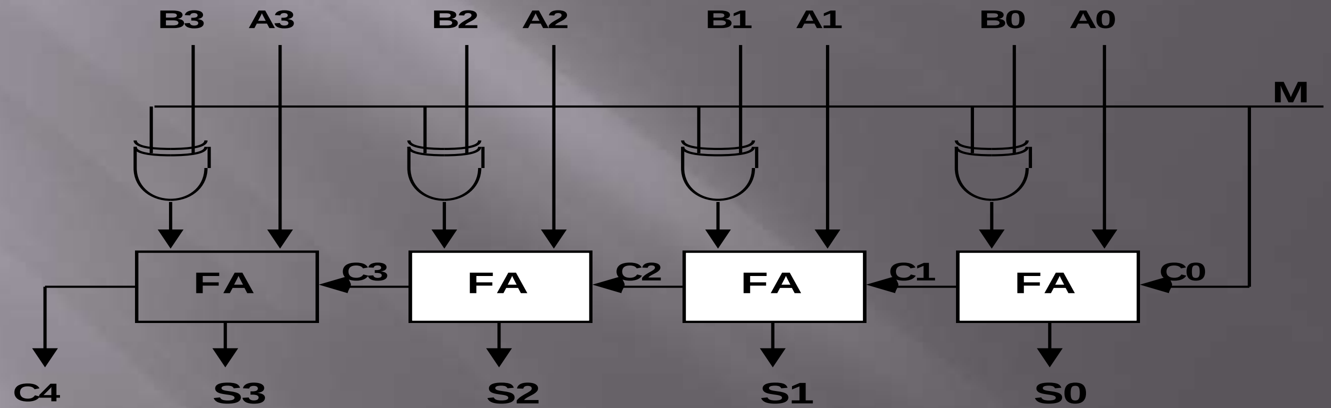
R3 ← R1 + R2	Contents of R1 plus R2 transferred to R3
R3 ← R1 - R2	Contents of R1 minus R2 transferred to R3
R2 ← R2’	Complement the contents of R2
R2 ← R2’+ 1	2's complement the contents of R2 (negate)
R3 ← R1 + R2’+ 1	subtraction
R1 ← R1 + 1	Increment
R1 ← R1 - 1	Decrement

BINARY ADDER / SUBTRACTOR / INCREMENTER

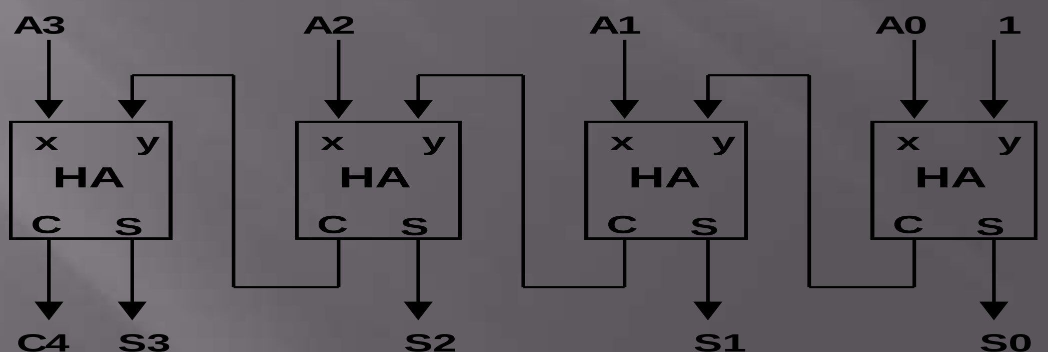
Binary Adder



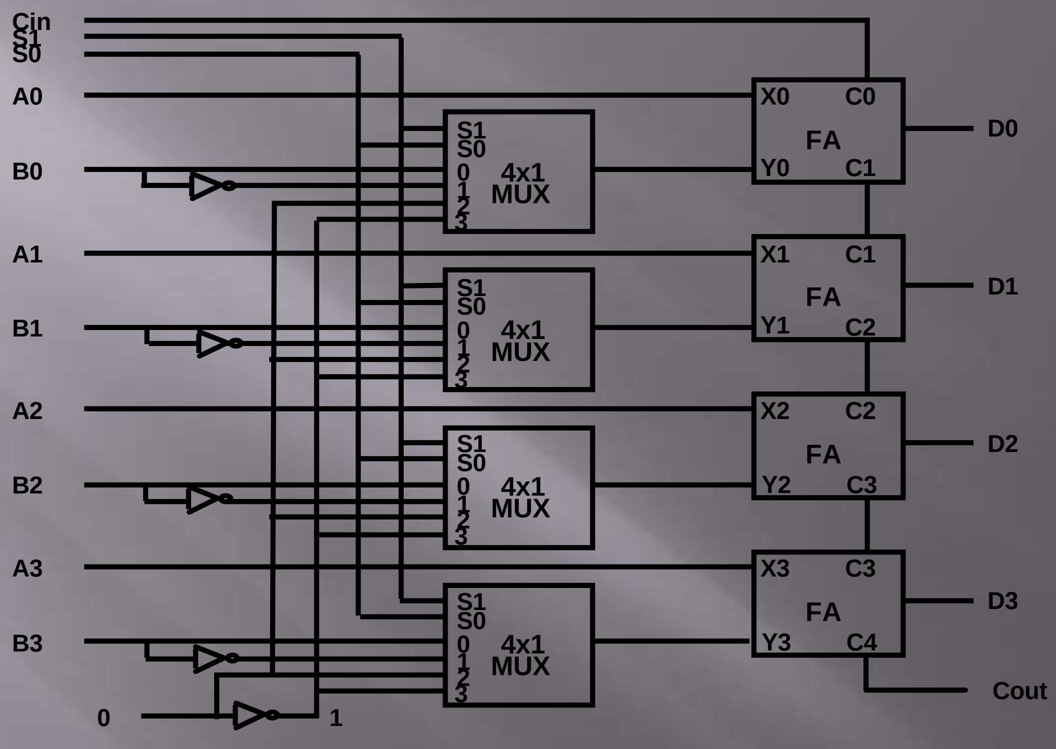
Binary Adder-Subtractor



Binary Incrementer



ARITHMETIC CIRCUIT



S1	S0	Cin	Y	Output	Microoperation
0	0	0	B	D = A + B	Add
0	0	1	B	D = A + B + 1	Add with carry
0	1	0	B’	D = A + B’	Subtract with borrow
0	1	1	B’	D = A + B’+ 1	Subtract
1	0	0	0	D = A	Transfer A
1	0	1	0	D = A + 1	Increment A
1	1	0	1	D = A - 1	Decrement A
1	1	1	1	D = A	Transfer A

LOGIC MICROOPERATIONS

- Specify binary operations on the strings of bits in registers
  - Logic microoperations are bit-wise operations, i.e., they work on the individual bits of data
  - useful for bit manipulations on binary data
  - useful for making logical decisions based on the bit value
- There are, in principle, 16 different logic functions that can be defined over two binary input variables

A	B	F_0	F_1	F_2	…	F_{13}	F_{14}	F_{15}
0	0	0	0	0	…	1	1	1
0	1	0	0	0	…	1	1	1
1	0	0	0	1	…	0	1	1
1	1	0	1	0	…	1	0	1

- However, most systems only implement four of these
  - AND ($\wedge$), OR ($\vee$), XOR ($\oplus$), Complement/NOT
- The others can be created from combination of these

LIST OF LOGIC MICROOPERATIONS

- **List of Logic Microoperations**
  - **16 different logic operations with 2 binary vars.**
  - **n binary vars $\rightarrow 2^{2^n}$ functions**

- **Truth tables for 16 functions of 2 variables and the corresponding 16 logic micro-operations**

x: 0 0 1 1 y: 0 1 0 1	*Boolean Function*	*Micro-Operations*	*Name*
0 0 0 0	$F0 = 0$	$F \leftarrow 0$	Clear
0 0 0 1	$F1 = xy$	$F \leftarrow A \wedge B$	AND
0 0 1 0	$F2 = xy'$	$F \leftarrow A \wedge B'$	
0 0 1 1	$F3 = x$	$F \leftarrow A$	Transfer A
0 1 0 0	$F4 = x'y$	$F \leftarrow A' \wedge B$	
0 1 0 1	$F5 = y$	$F \leftarrow B$	Transfer B
0 1 1 0	$F6 = x \oplus y$	$F \leftarrow A \oplus B$	Exclusive-OR
0 1 1 1	$F7 = x + y$	$F \leftarrow A \vee B$	OR
1 0 0 0	$F8 = (x + y)'$	$F \leftarrow (A \vee B)'$	NOR
1 0 0 1	$F9 = (x \oplus y)'$	$F \leftarrow (A \oplus B)'$	Exclusive-NOR
1 0 1 0	$F10 = y'$	$F \leftarrow B'$	Complement B
1 0 1 1	$F11 = x + y'$	$F \leftarrow A \vee B$	
1 1 0 0	$F12 = x'$	$F \leftarrow A'$	Complement A
1 1 0 1	$F13 = x' + y$	$F \leftarrow A' \vee B$	
1 1 1 0	$F14 = (xy)'$	$F \leftarrow (A \wedge B)'$	NAND
1 1 1 1	$F15 = 1$	$F \leftarrow$ all 1's	Set to all 1's

APPLICATIONS OF LOGIC MICROOPERATIONS

- Logic microoperations can be used to manipulate individual bits or a portions of a word in a register

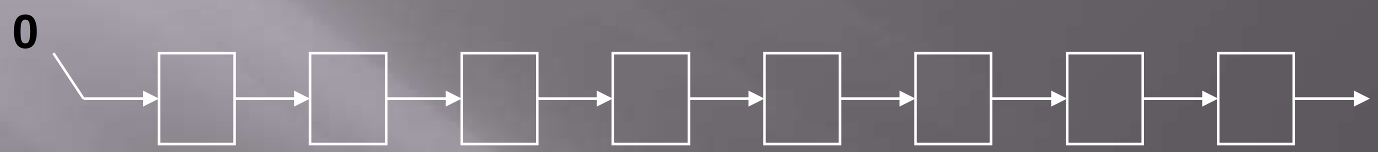
- Consider the data in a register A. In another register, B, is bit data that will be used to modify the contents of A

  - Selective-set $A \leftarrow A + B$
  - Selective-complement $A \leftarrow A \oplus B$
  - Selective-clear $A \leftarrow A \bullet B'$
  - Mask (Delete) $A \leftarrow A \bullet B$
  - Clear $A \leftarrow A \oplus B$
  - Insert $A \leftarrow (A \bullet B) + C$
  - Compare $A \leftarrow A \oplus B$
  - . . .

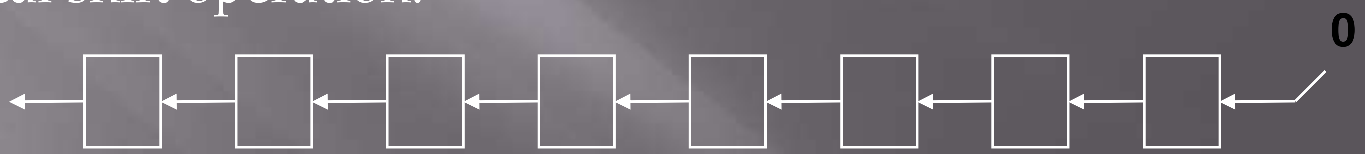
LOGICAL SHIFT

- In a logical shift the serial input to the shift is a 0.

- A right logical shift operation:



- A left logical shift operation:

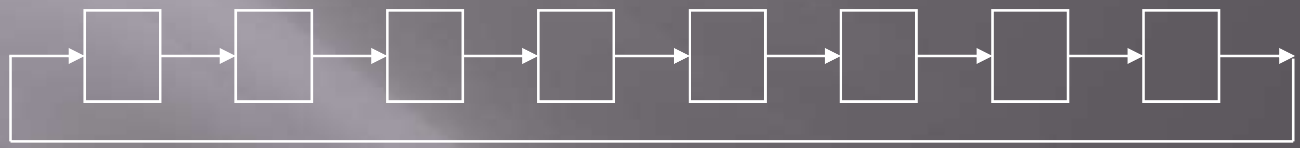


- In a Register Transfer Language, the following notation is used
  - *shl* for a logical shift left
  - *shr* for a logical shift right
  - Examples:
    - R2 ← *shr* R2
    - R3 ← *shl* R3

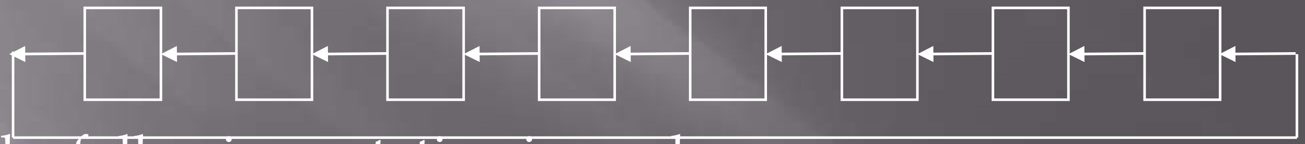
CIRCULAR SHIFT

- In a circular shift the serial input is the bit that is shifted out of the other end of the register.

- A right circular shift operation:



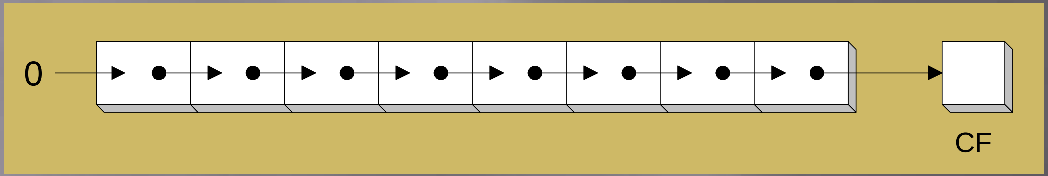
- A left circular shift operation:



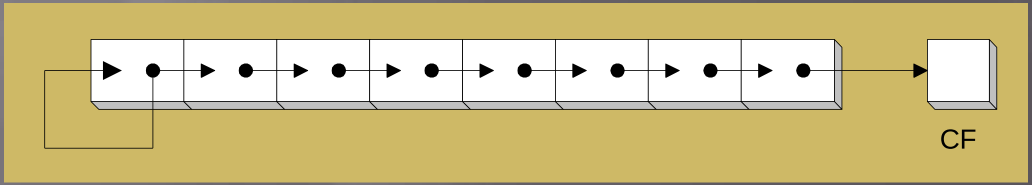
- In a RTL, the following notation is used
  - *cil* for a circular shift left
  - *cir* for a circular shift right
  - Examples:
    - R2 ← *cir* R2
    - R3 ← *cil* R3

Logical versus Arithmetic Shift

- A logical shift fills the newly created bit position with zero:

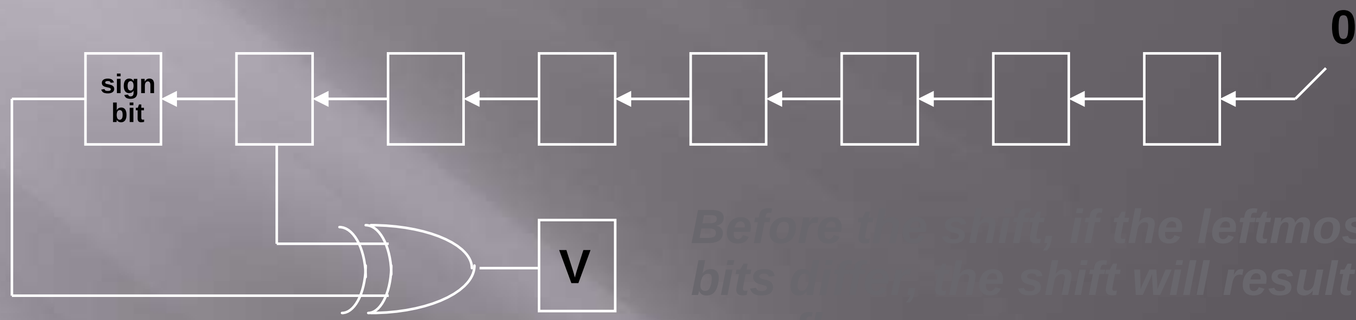


- An arithmetic shift fills the newly created bit position with a copy of the number's sign bit:



ARITHMETIC SHIFT

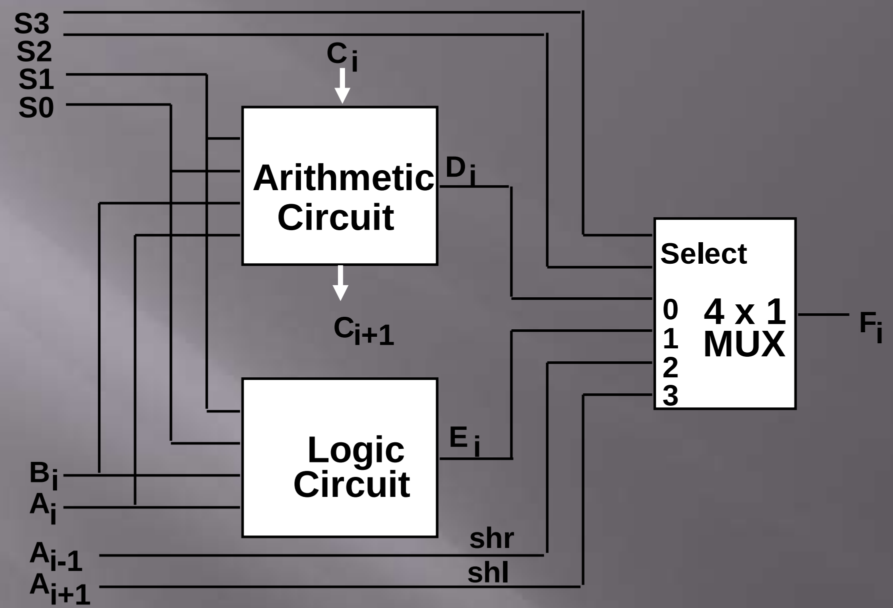
- An left arithmetic shift operation must be checked for the overflow



Before the shift, if the leftmost two bits differ, the shift will result in an overflow

- **In a RTL, the following notation is used**
  - ***ashl*** **for an arithmetic shift left**
  - ***ashr*** **for an arithmetic shift right**
  - **Examples:**
    - » **R2 ← *ashr* R2**
    - » **R3 ← *ashl* R3**

ARITHMETIC LOGIC SHIFT UNIT



S3	S2	S1	S0	Cin	Operation	Function
0	0	0	0	0	F = A	Transfer A
0	0	0	0	1	F = A + 1	Increment A
0	0	0	1	0	F = A + B	Addition
0	0	0	1	1	F = A + B + 1	Add with carry
0	0	1	0	0	F = A + B’	Subtract with borrow
0	0	1	0	1	F = A + B’+ 1	Subtraction
0	0	1	1	0	F = A - 1	Decrement A
0	0	1	1	1	F = A	TransferA
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 = 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

CONTROL UNIT

A control unit is a major component of the computer it controls the flow of data between the CPU , memory and peripherals.

- Two major types of Control Unit
  - Hardwired Control :
    - The control logic is implemented with gates, F/Fs, decoders, and other digital circuits
    - + Fast operation, - Wiring change(if the design has to be modified)
  - Microprogrammed Control :
    - The control information is stored in a control memory, and the control memory is programmed to initiate the required sequence of microoperations
    - + Any required change can be done by updating the microprogram in control memory,

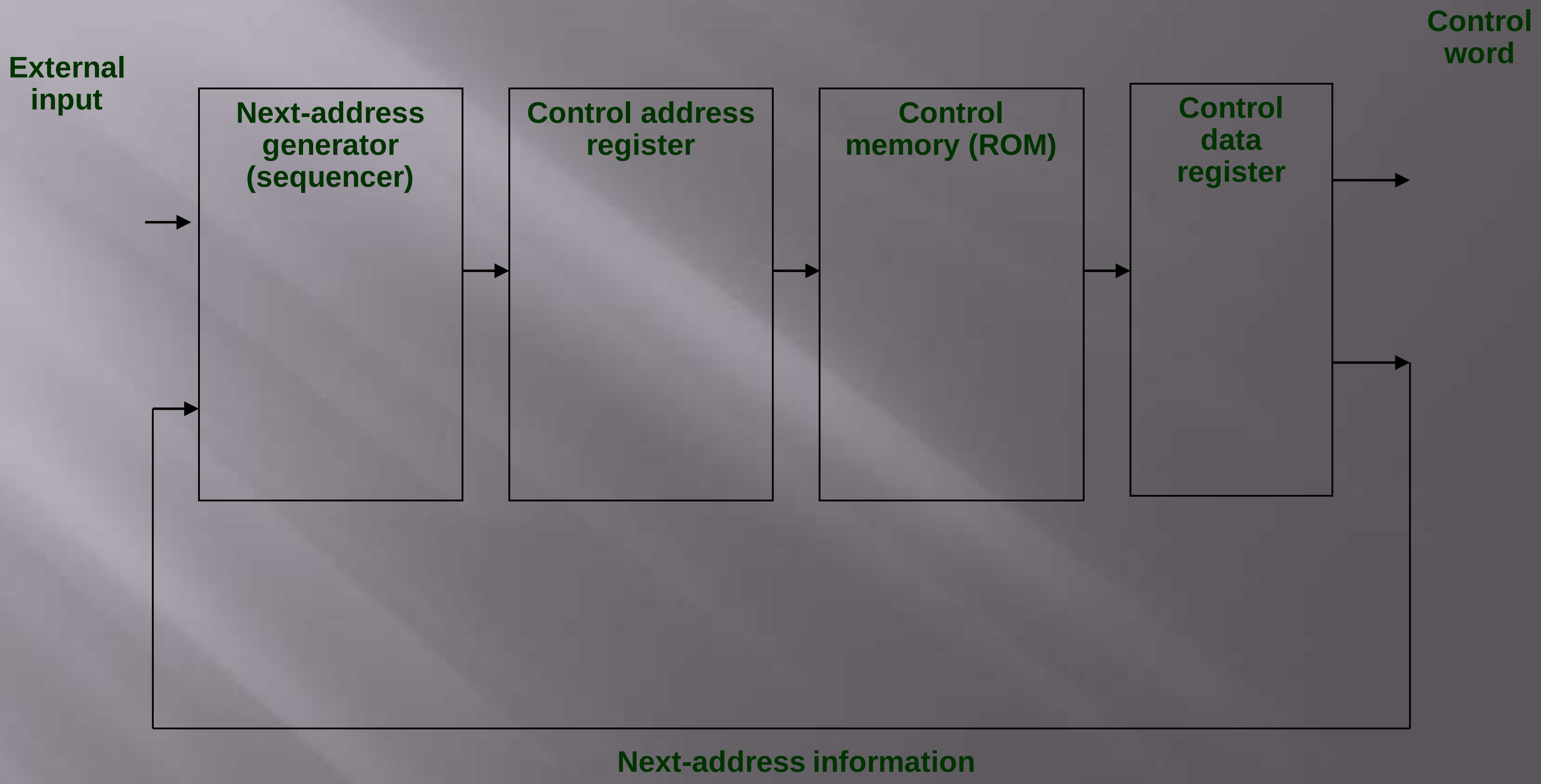
MICROPROGRAMMED CONTROL UNIT(MCU)

- Control Word
  - The control variables at any given time can be represented by a string of 1's and 0's

Control Memory

- A memory is part of a control unit : *Microprogram*
- Computer Memory (*employs a microprogrammed control unit*)
  - Main Memory : for storing user program (*Machine instruction/data*)
  - Control Memory : for storing microprogram (*Microinstruction*)

Microprogrammed Control Organization



INPUT-OUTPUT ORGANIZATION

- INPU T-OUTPUT ORGANIZATION

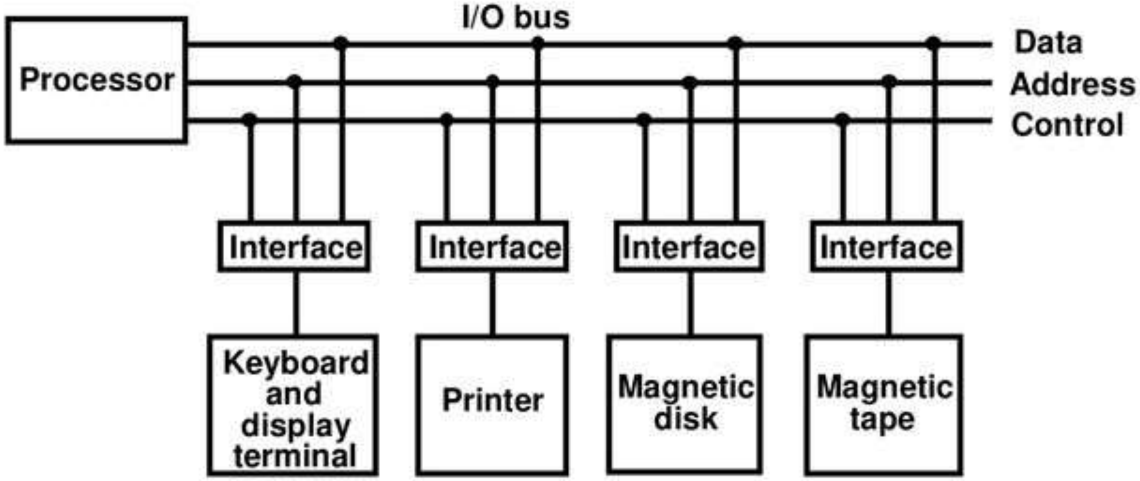
- Provides a method for transferring information between internal storage (such as memory and CPU registers) and external I/O devices
- Resolves the *differences* between the computer and peripheral devices
  - Peripherals - Electromechanical Devices
  - CPU or Memory - Electronic Device

I/O BUS AND INTERFACE MODULES

- Each peripheral has an interface module associated with it

  Interface
- - Decodes the device address (device code)
- - Decodes the commands (operation)
- - Provides signals for the peripheral controller
- - Synchronizes the data flow and supervises
- the transfer rate between peripheral and CPU or Memory

I/O BUS AND INTERFACE MODULES



Each peripheral has an interface module associated with it

Interface

- Decodes the device address (device code)
- Decodes the commands (operation)
- Provides signals for the peripheral controller
- Synchronizes the data flow and supervises the transfer rate between peripheral and CPU or Memory

Typical I/O instruction

Op. code	Device address	Function code

(Command)

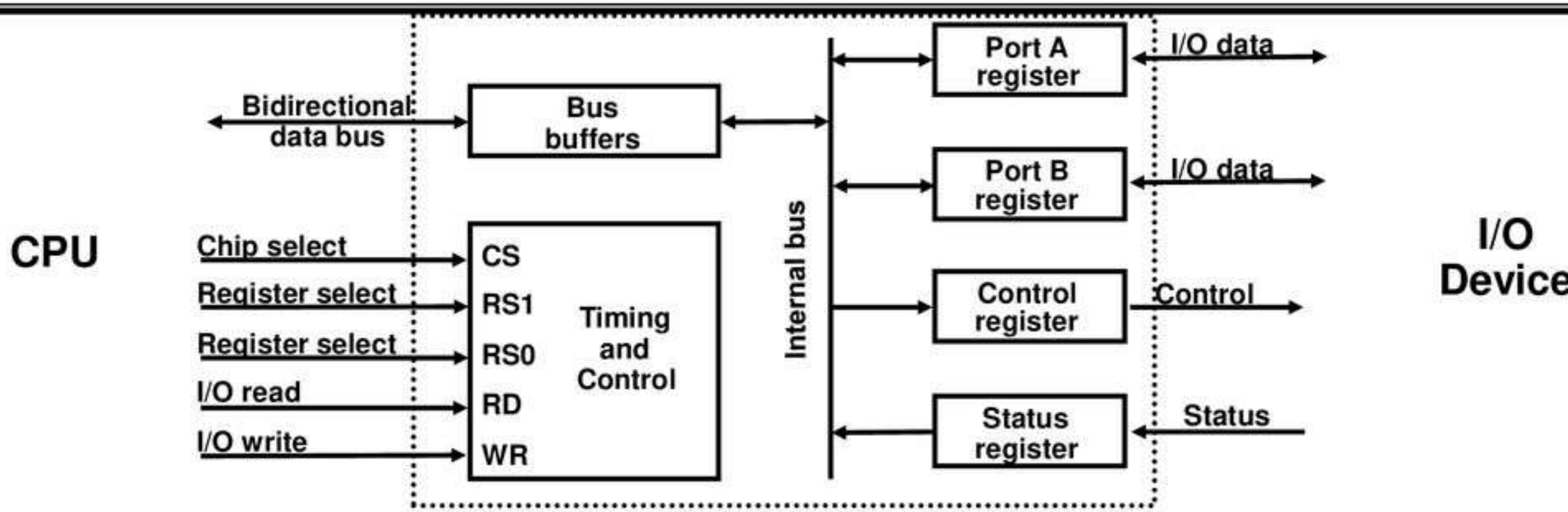
I/O BUS AND MEMORY BUS

- Functions of Buses

  - *MEMORY BUS* is for information transfers between CPU and the MM

  - * *I/O BUS* is for information transfers between CPU and I/O devices through their I/O interface

I/O INTERFACE



CS	RS1	RS0	Register selected
0	x	x	None - data bus in high-impedence
1	0	0	Port A register
1	0	1	Port B register
1	1	0	Control register
1	1	1	Status register

Programmable Interface

- Information in each port can be assigned a meaning depending on the mode of operation of the I/O device
  → Port A = Data; Port B = Command; Port C = Status

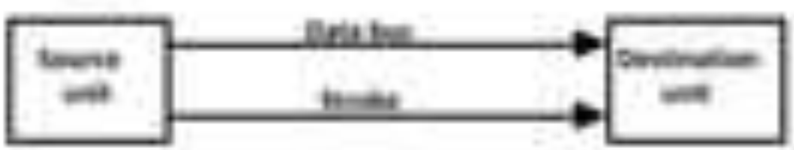
- CPU initializes(loads) each port by transferring a byte to the Control Register
  → Allows CPU can define the mode of operation of each port
  → *Programmable Port*: By changing the bits in the control register, it is possible to change the interface characteristics

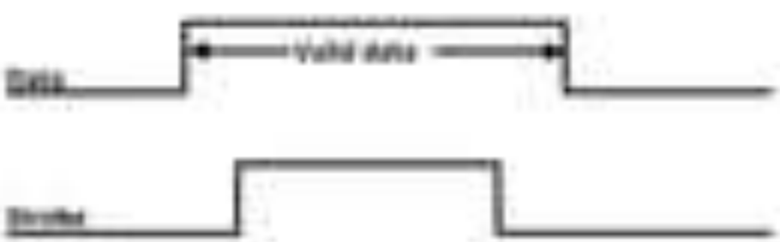
Strobe Control

- Employs a single control line to time each transfer
- The strobe may be activated by either the source or the destination unit

Source-Initiated Strobe for Data Transfer

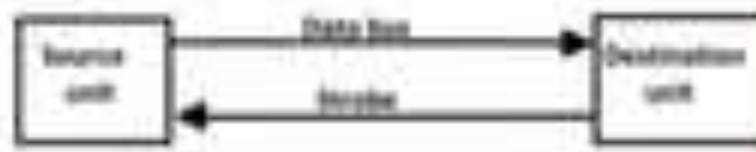
Block Diagram



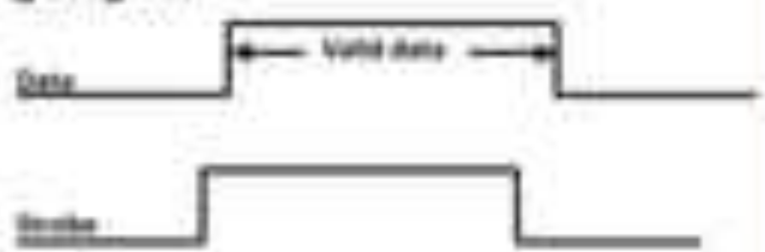


Destination-Initiated Strobe for Data Transfer

Block Diagram



Timing Diagram



HANDSHAKING

- Strobe Methods

Source-Initiated

The source unit that initiates the transfer has no way of knowing whether the destination unit has actually received data

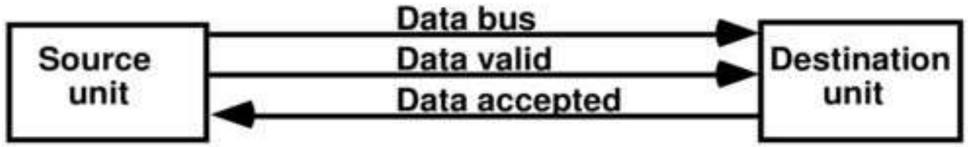
- Destination-Initiated

The destination unit that initiates the transfer no way of knowing whether the source has actually placed the data on the bus

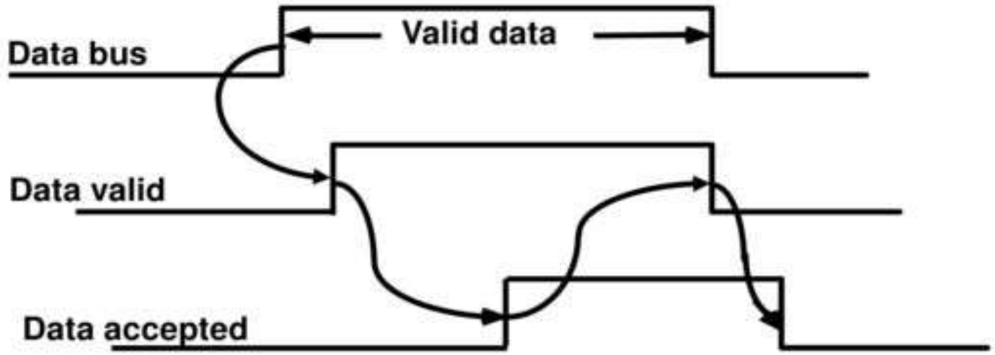
To solve this problem, the *HANDSHAKE* method introduces a second control signal to provide a *Reply* to the unit that initiates the transfer

SOURCE-INITIATED TRANSFER USING HANDSHAKE

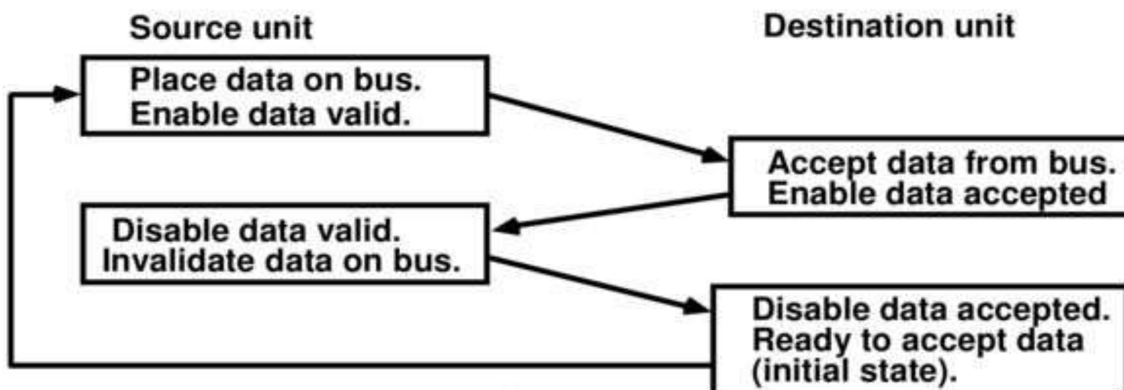
Block Diagram



Timing Diagram



Sequence of Events



- * Allows arbitrary delays from one state to the next
- * Permits each unit to respond at its own data transfer rate
- * The rate of transfer is determined by the slower unit

Destination initiated transfer