UNIT 1

Functional Blocks of a Computer: Introduction, Block diagram of digital computer, Instruction codes, Computer Registers, Common bus system, Computer instructions, Instruction cycle and Instruction set, Register Transfer Language.

Data Representation: Fixed and floating point arithmetic- Addition, Subtraction, Multiplication, Division.

Control unit Design: Hardwired control unit, Control memory, Address sequencing, Micro-programmed control unit design, Hardwired Vs Micro-programmed design.

Register Transfer Language (RTL)

- Digital System: An interconnection of hardware modules that do a certain task on the information.
- Registers + Operations performed on the data stored in them = Digital Module
- Modules are interconnected with common data and control paths to form a digital computer system

Register Transfer Language cont.

- Microoperations: operations executed on data stored in one or more registers.
- For any function of the computer, a sequence of microoperations is used to describe it
- The result of the operation may be:
  - replace the previous binary information of a register or
  - transferred to another register

101101110011 —Shift Right Operation→ 010110111001

Register Transfer Language cont.

- The internal hardware organization of a digital computer is defined by specifying:
  - The set of registers it contains and their function
  - The sequence of microoperations performed on the binary information stored in the registers
  - The control that initiates the sequence of microoperations
- Registers + Microoperations Hardware + Control Functions = Digital Computer

Register Transfer Language cont.

- Register Transfer Language (RTL) : a symbolic notation to describe the microoperation transfers among registers

Next steps:

- Define symbols for various types of microoperations,
- Describe the hardware that implements these microoperations

Register Transfer

- Computer registers are designated by capital letters (sometimes followed by numerals) to denote the function of the register
    - R1: processor register
    - MAR: Memory Address Register (holds an address for a memory unit)
    - PC: Program Counter
    - IR: Instruction Register
    - SR: Status Register

Register Transfer cont.

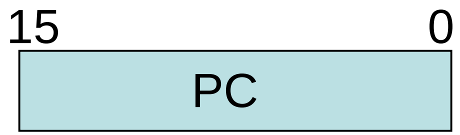
- The individual flip-flops in an n-bit register are numbered in sequence from 0 to n-1 (from the right position toward the left position)

Register R1 Showing individual bits

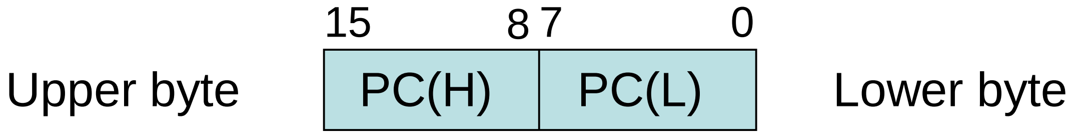
A block diagram of a register

Register Transfer cont.

Other ways of drawing the block diagram of a register:



Numbering of bits



Partitioned into two parts

Register Transfer cont.

- Information transfer from one register to another is described by a *replacement operator:* **R2 ← R1**
  - This statement denotes a transfer of the content of register R1 into register R2
- The transfer happens in one clock cycle
- The content of the R1 (source) does not change
- The content of the R2 (destination) will be lost and replaced by the new data transferred from R1
- We are assuming that the circuits are available from the outputs of the source register to the inputs of the destination register, and that the destination register has a parallel load capability

Register Transfer cont.

- Conditional transfer occurs only under a control condition

- Representation of a (conditional) transfer

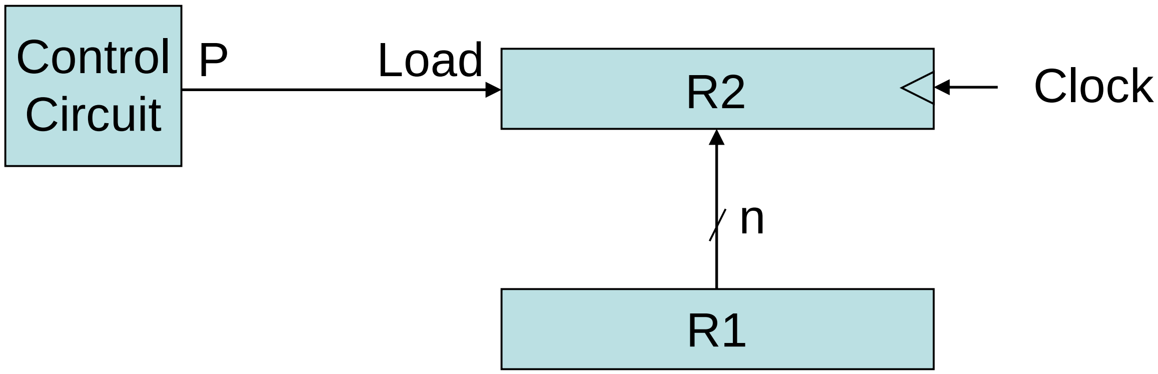
  P: $\quad R2 \leftarrow R1$

- A binary condition (P equals to 0 or 1) determines when the transfer occurs
- The content of R1 is transferred into R2 only if P is 1

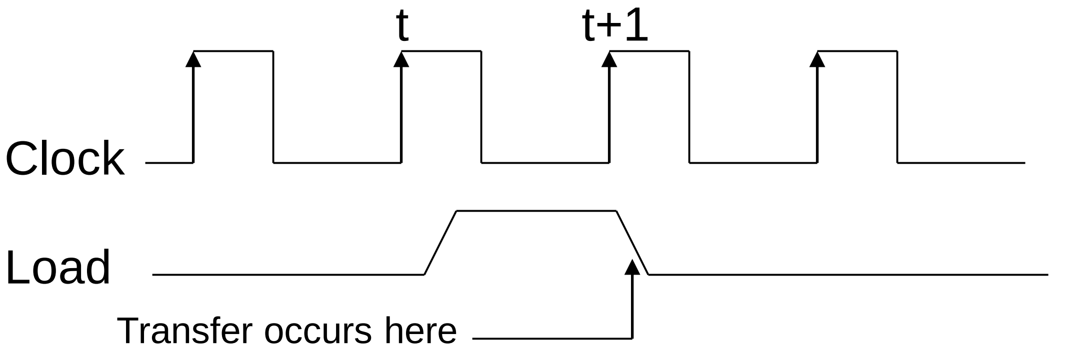
Register Transfer cont.

Hardware implementation of a controlled transfer: P: R2 $\leftarrow$ R1

Block diagram:



Timing diagram



Synchronized
with the clock

Register Transfer cont.

Basic Symbols for Register Transfers		
Symbol	**Description**	**Examples**
Letters & numerals	Denotes a register	MAR, R2
Parenthesis ( )	Denotes a part of a register	R2(0-7), R2(L)
Arrow $\leftarrow$	Denotes transfer of information	R2 $\leftarrow$ R1
Comma ,	Separates two microoperations	R2 $\leftarrow$ R1, R1 $\leftarrow$ R2

Arithmetic Microoperations

- The microoperations most often encountered in digital computers are classified into four categories:
  - Register transfer microoperations
  - Arithmetic microoperations (on numeric data stored in the registers)
  - Logic microoperations (bit manipulations on non-numeric data)
  - Shift microoperations

Arithmetic Microoperations cont.

- The basic arithmetic microoperations are: addition, subtraction, increment, decrement, and shift
- Addition Microoperation:

$$R3 \leftarrow R1+R2$$

- Subtraction Microoperation:

$$R3 \leftarrow R1-R2 \text{ or :}$$

1's complement

$$R3 \leftarrow R1+\overline{R2}+1$$

Arithmetic Microoperations cont.

- One's Complement Microoperation:

$$R2 \leftarrow \overline{R2}$$

- Two's Complement Microoperation:

$$R2 \leftarrow \overline{R2}+1$$

- Increment Microoperation:

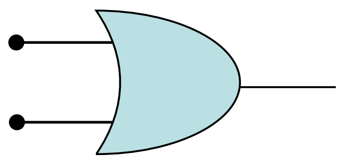
$$R2 \leftarrow R2+1$$

- Decrement Microoperation:

$$R2 \leftarrow R2-1$$

Logic Microoperations
The four basic microoperations

OR Microoperation

- Symbol: ∨, +
- Gate: 
- Example: $100110_2 \vee 1010110_2 = 1110110_2$

OR OR

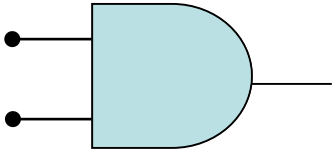
$$P+Q: R1 \leftarrow R2+R3,\ R4 \leftarrow R5 \vee R6$$

ADD

Logic Microoperations
The four basic microoperations
cont.

AND Microoperation

- Symbol: $\wedge$
- Gate:

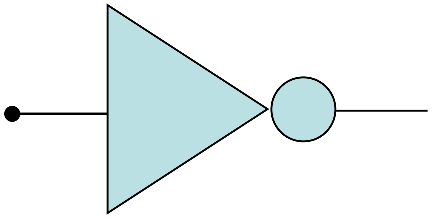


- Example: $100110_2 \wedge 1010110_2 = 0000110_2$

Logic Microoperations
The four basic microoperations
cont.

Complement (NOT) Microoperation

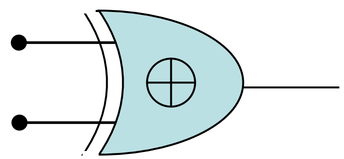
- Symbol: ‾
- Gate:



- Example: $\overline{1010110}_2 = 0101001_2$

Logic Microoperations
The four basic microoperations
cont.

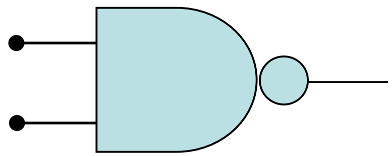
XOR (Exclusive-OR) Microoperation

- Symbol: ⊕
- Gate: 
- Example: $100110_2 \oplus 1010110_2 = 1110000_2$

Logic Microoperations
Other Logic Microoperations
cont.

NAND Microoperation

- Symbols: $\wedge$ and $\overline{}$

- Gate: 

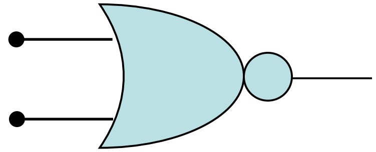
- Example: $100110_2 \overline{\wedge}\, 1010110_2 = 1111001_2$

Logic Microoperations

Other Logic Microoperations

cont.

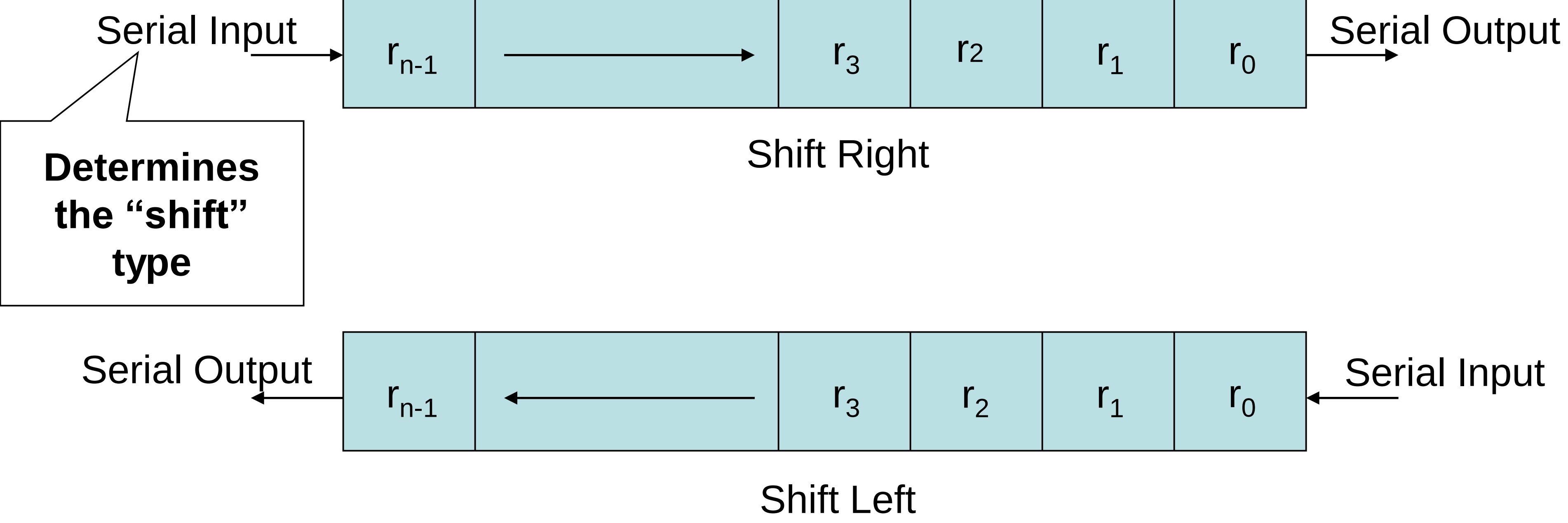
NOR Microoperation

- Symbols: $\vee$ and $\overline{}$
- Gate: 
- Example: $\overline{100110_2 \vee 1010110_2} = 0001001_2$

Shift Microoperations

- Used for serial transfer of data
- Also used in conjunction with arithmetic, logic, and other data-processing operations
- The contents of the register can be shifted to the left or to the right
- As being shifted, the first flip-flop receives its binary information from the serial input
- Three types of shift: Logical, Circular, and Arithmetic

Shift Microoperations cont.



**Note that the bit ri is the bit at position (i) of the register

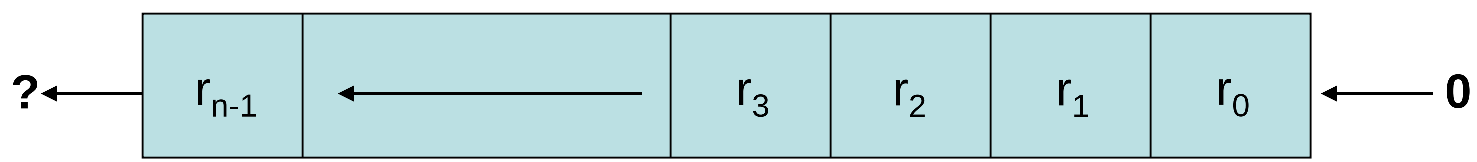
Shift Microoperations: Logical Shifts

- Transfers 0 through the serial input
- Logical Shift Right: $R1 \leftarrow \text{shr } R1$

The same

- Logical Shift Left: $R2 \leftarrow \text{shl } R2$

The same



Logical Shift Left

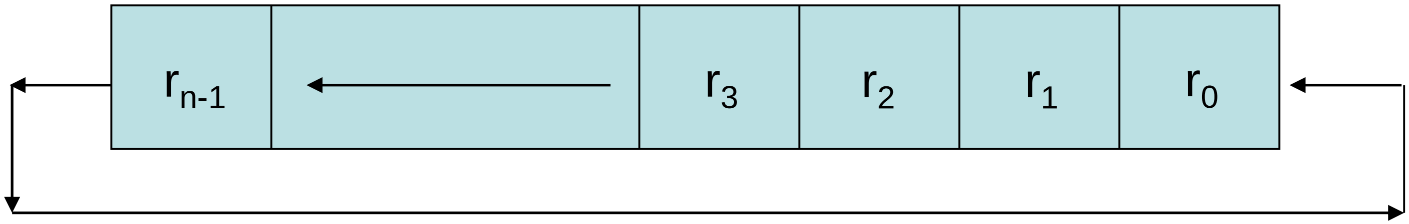
Shift Microoperations:
Circular Shifts (Rotate Operation)

- Circulates the bits of the register around the two ends without loss of information
- Circular Shift Right: R1 ← cir R1

The same

- Circular Shift Left: R2 ← cil R2

The same



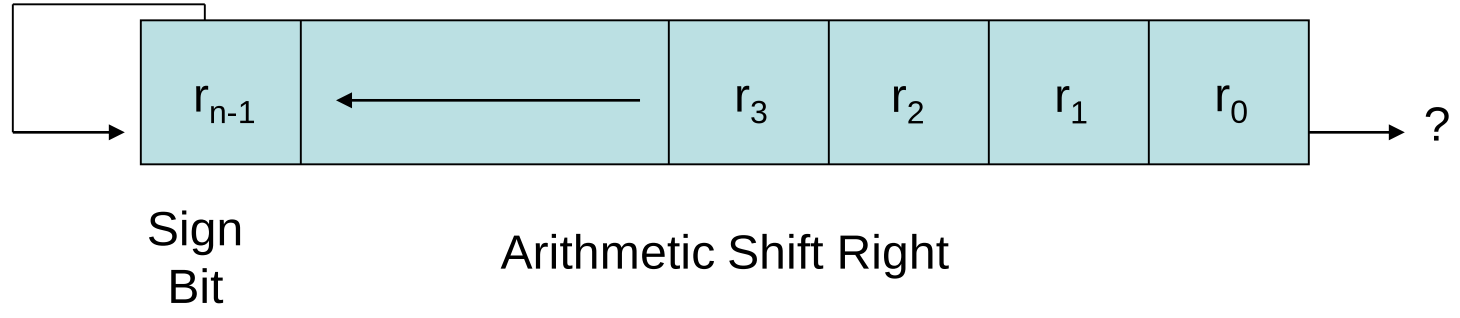
Circular Shift Left

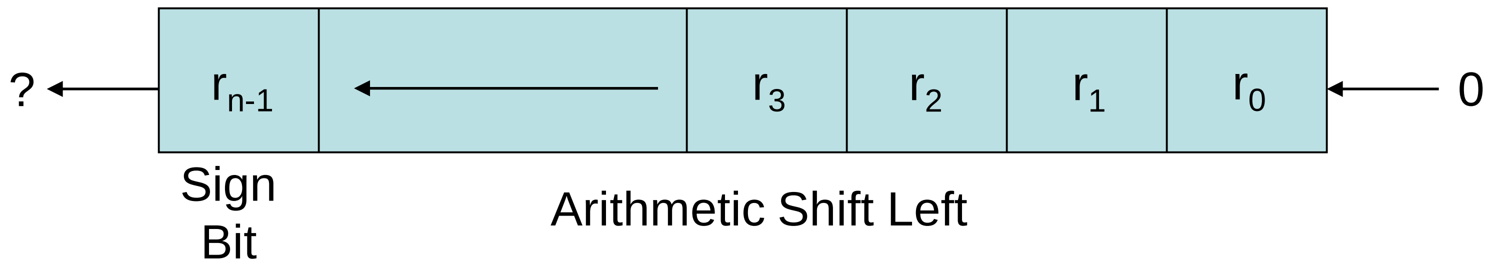
Shift Microoperations
Arithmetic Shifts

- Shifts a signed binary number to the left or right
- An arithmetic shift-left multiplies a signed binary number by 2:   ashl (00100): 01000
- An arithmetic shift-right divides the number by 2

  ashr (00100) : 00010
- An overflow may occur in arithmetic shift-left, and occurs when the sign bit is changed (sign reversal)

Shift Microoperations Arithmetic Shifts cont.





Shift Microoperations cont.

- Example: Assume R1=11001110, then:
  - Arithmetic shift right once : R1 = 11100111
  - Arithmetic shift right twice : R1 = 11110011
  - Arithmetic shift left once : R1 = 10011100
  - Arithmetic shift left twice : R1 = 00111000
  - Logical shift right once : R1 = 01100111
  - Logical shift left once : R1 = 10011100
  - Circular shift right once : R1 = 01100111
  - Circular shift left once : R1 = 10011101