Chapter 4:

Register Transfer and Microoperations

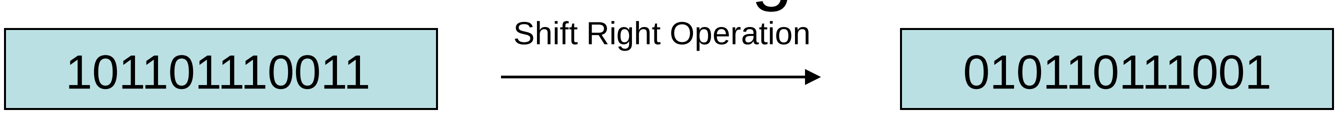
(4.1, 4.2, 4.4, 4.5)

contents

- **Register Transfer Language**
- **Register Transfer**
- **Arithmetic Microoperations**
- **Logic Microoperations**

4-1 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



4-1 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

4-2 Register Transfer (our first microoperation)

- 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

4-2 Register Transfer cont.

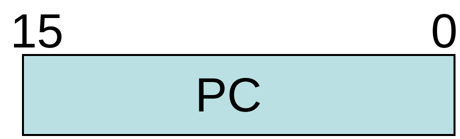
- The individual flip-flops in an n-bit register are numbered in sequence from 0 to n-1 (from the right position toward the left position)

Register R1 Showing individual bits

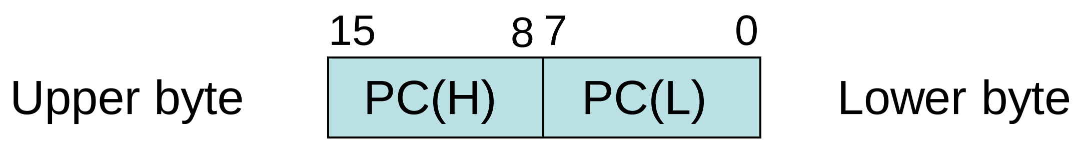
A block diagram of a register

4-2 Register Transfer cont.

Other ways of drawing the block diagram of a register:



Numbering of bits



Partitioned into two parts

4-2 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

4-2 Register Transfer cont.

- Conditional transfer occurs only under a control condition

- Representation of a (conditional) transfer

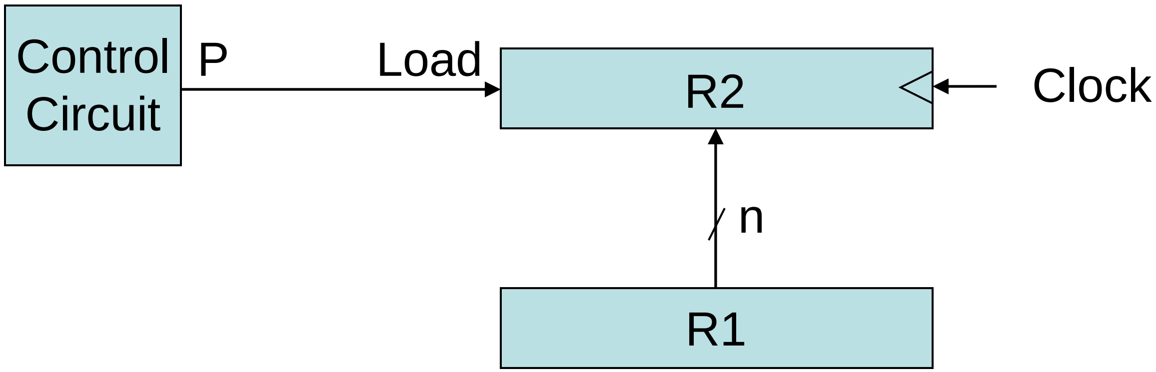
  P: $R2 \leftarrow R1$

- 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

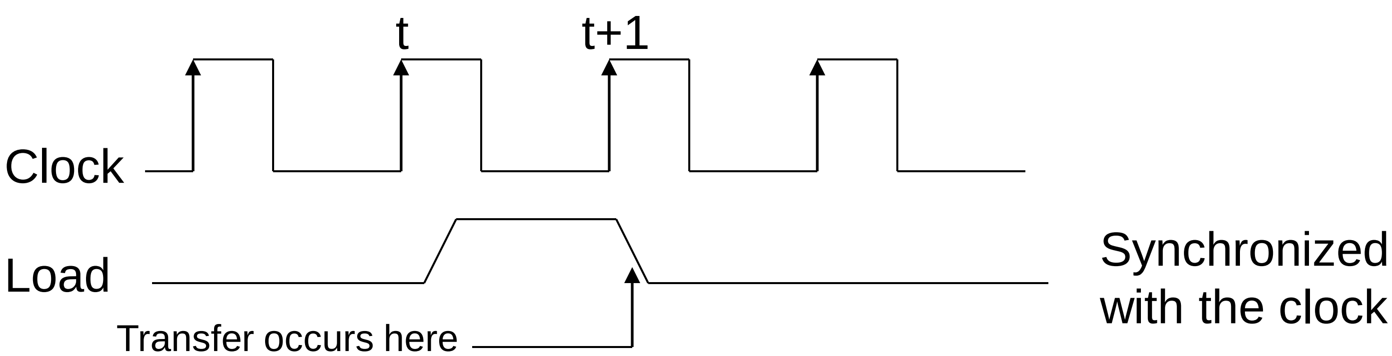
4-2 Register Transfer cont.

Hardware implementation of a controlled transfer: P: R2 ← R1

Block diagram:



Timing diagram



4-2 Register Transfer $^{\text{cont.}}$

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

4-4 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

4-4 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

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

4-4 Arithmetic Microoperations cont.

- One's Complement Microoperation:

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

- Two's Complement Microoperation:

$$R2 \leftarrow R2+1$$

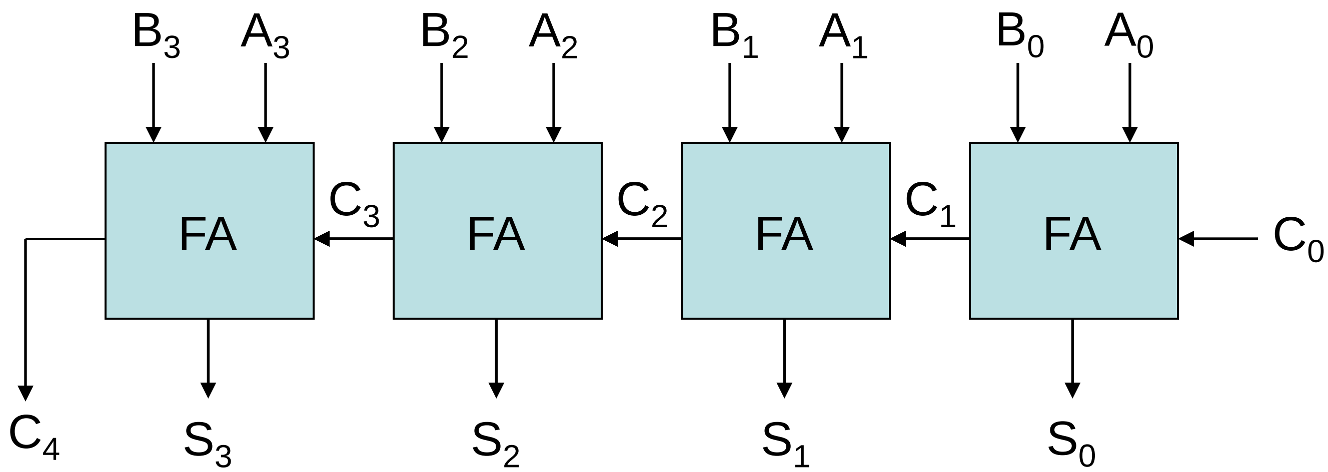
- Increment Microoperation:

$$R2 \leftarrow R2+1$$

- Decrement Microoperation:

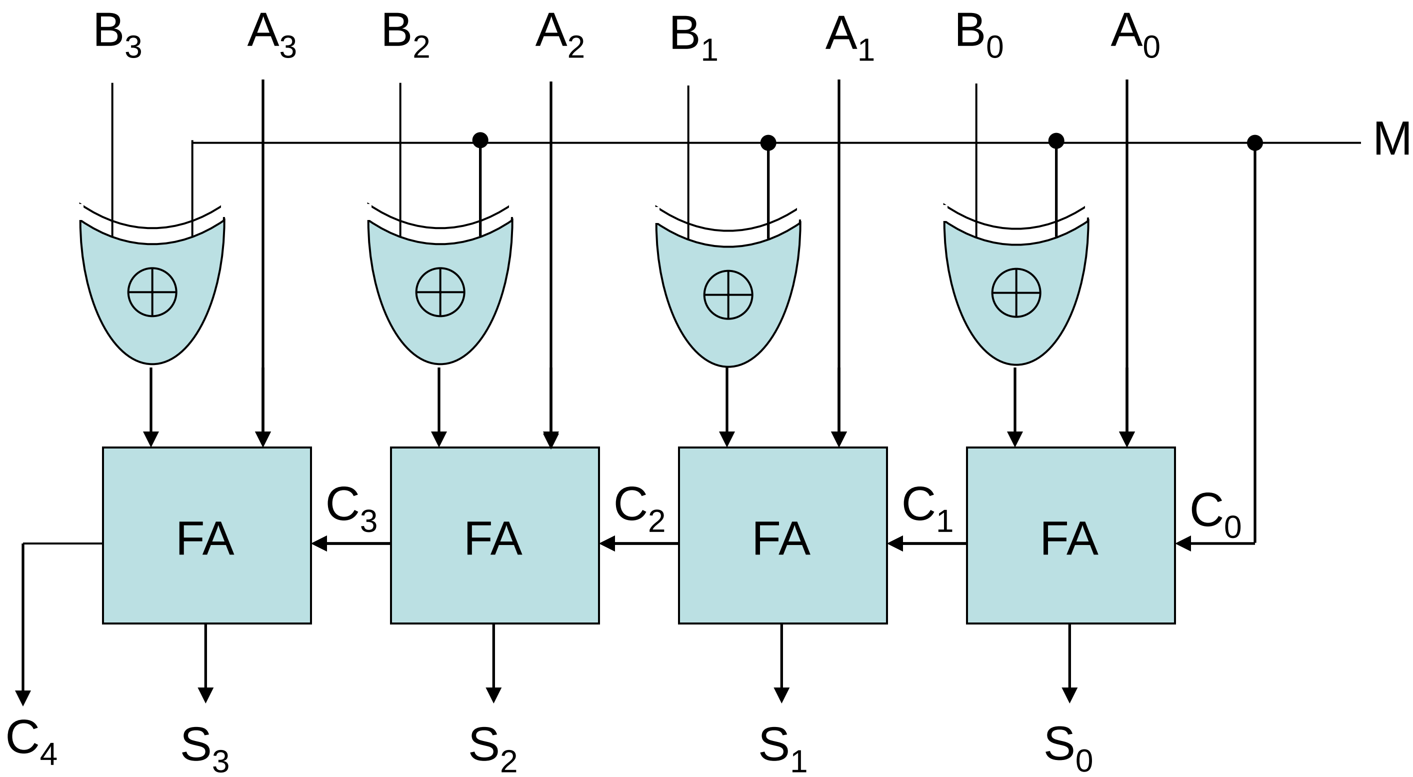
$$R2 \leftarrow R2-1$$

4-4 Arithmetic Microoperations
Binary Adder



4-bit binary adder
(connection of FAs)

4-4 Arithmetic Microoperations
Binary Adder-Subtractor

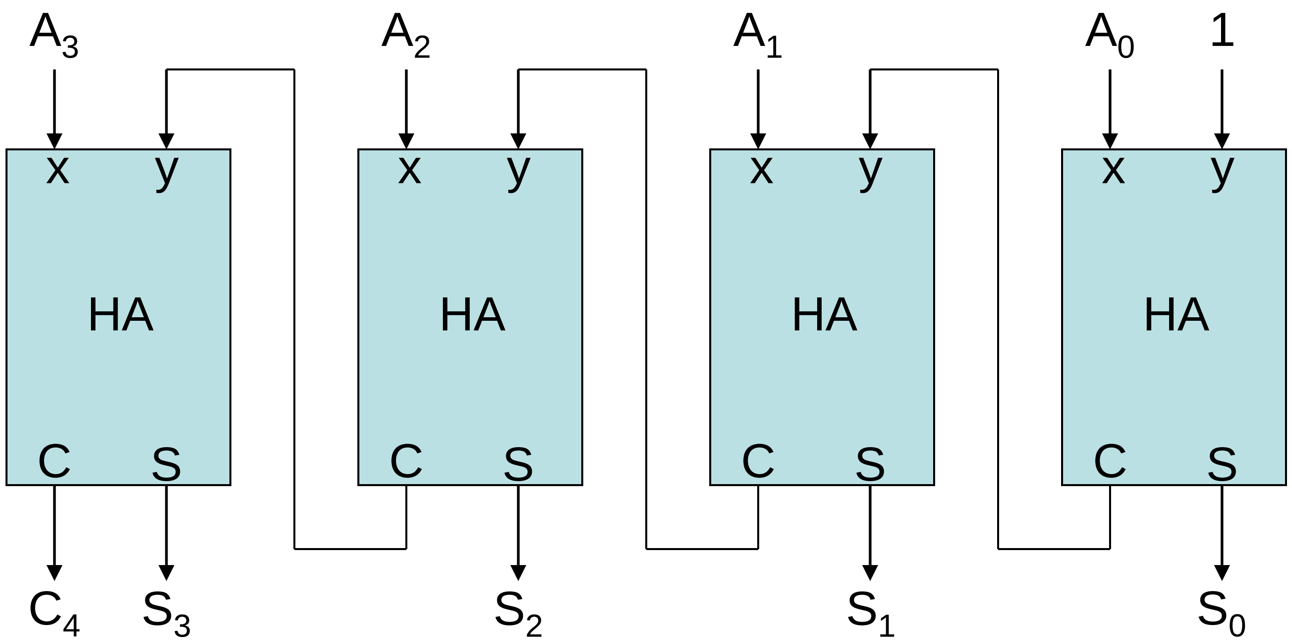


4-bit adder-subtractor

4-4 Arithmetic Microoperations Binary Adder-Subtractor

- If M=0, It works as an adder and if M=1, then
- For unsigned numbers, this gives A – B if A≥B

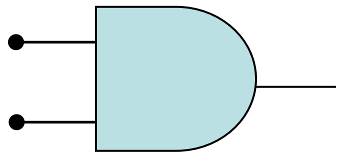
4-4 Arithmetic Microoperations
Binary Incrementer



4-bit Binary Incrementer

4-5 Logic Microoperations The four basic microoperations cont.

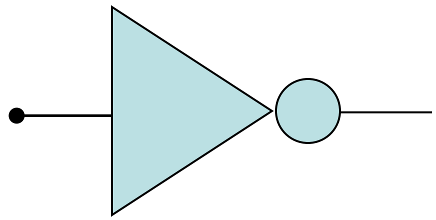
AND Microoperation

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

4-5 Logic Microoperations The four basic microoperations cont.

Complement (NOT) Microoperation

- Symbol: $\overline{\quad}$
- Gate:

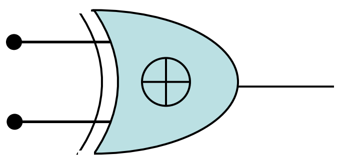


- Example: $\overline{1010110}_2 = 0101001_2$

4-5 Logic Microoperations The four basic microoperations cont.

XOR (Exclusive-OR) Microoperation

- Symbol: ⊕
- Gate:



- Example: $100110_2 \oplus 1010110_2 = 1110000_2$

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 $A \leftarrow A \oplus B$

4-5 Logic Microoperations Other Logic Microoperations

Selective-set Operation

- Selective-set $A \leftarrow A + B$

- The selective set operation sets to 1 the bits in register A where there are corresponding 1's in register B.

- Example: $0100_2 \vee 1000_2 = 1100_2$

A-register B-register

4-5 Logic Microoperations
Other Logic Microoperations cont.

Selective-complement (toggling) Operation

Selective-complement $A \leftarrow A \oplus B$

- Used to force selected bits of a register to be complemented by using the XOR operation
- (It complements bits in A where there are corresponding 1's in B. It does not effect bit positions that have 0's in B)
- Example: $0001_2 \oplus 1000_2 = 1001_2$

- Selective-clear $A \leftarrow A \cdot B'$
- The selective clear operation clears to 0 in A where there are corresponding 1's in B.

4-5 Logic Microoperations Other Logic Microoperations cont.

Insert Operation

Insert $A \leftarrow (A \cdot B) + C$

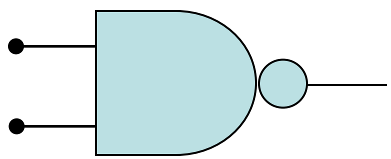
- Step1: mask the desired bits
- Step2: OR them with the desired value

- Example: suppose R1 = 0110 1010, and we desire to replace the leftmost 4 bits (0110) with 1001 then:
  - Step1: 0110 1010 $\wedge$ 0000 1111
  - Step2: 0000 1010 $\vee$ 1001 0000
- → R1 = 1001 1010

4-5 Logic Microoperations
Other Logic Microoperations
cont.

NAND Microoperation

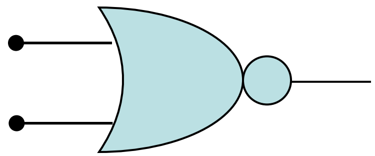
- Symbols: $\wedge$ and $\overline{}$
- Gate:



- Example: $100110_2 \wedge 1010110_2 = 1111001_2$

4-5 Logic Microoperations
Other Logic Microoperations
cont.

NOR Microoperation

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

4-5 Logic Microoperations Other Logic Microoperations
cont.

Set (Preset) Microoperation

- Force all bits into 1's by ORing them with a value in which all its bits are being assigned to logic-1
- Example: $100110_2 \vee 111111_2 = 111111_2$

Clear (Reset) Microoperation

- Force all bits into 0's by ANDing them with a value in which all its bits are being assigned to logic-0
- Example: $100110_2 \wedge 000000_2 = 000000_2$