**University Of Diyala**
**College Of Engineering**
**Computer Engineering Department**

# COMPUTER ARCHITECTURE I

## PART 3: LOGIC MICRO-OPERATIONS

**Asst. Prof. Ahmed Salah Hameed**

**Second stage**

**2022-2023**

# CONTENTS

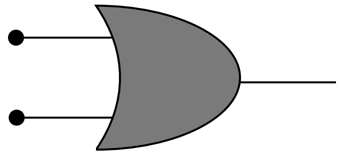
- **Logic Micro-operations**
- **Shift Micro-operations**
- **Arithmetic Logic Shift Unit**

# 4-5 LOGIC MICRO-OPERATIONS
## THE FOUR BASIC MICRO-OPERATIONS

### OR Microoperation

**Symbol: $\vee$, +**

**Gate:**



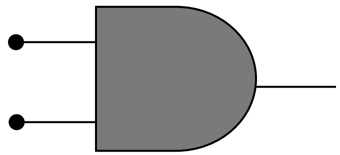
**Example: $100110_2 \vee 1010110_2 = 1110110_2$**

# 4-5 LOGIC MICRO-OPERATIONS
## THE FOUR BASIC MICRO-OPERATIONS

### AND Microoperation

**Symbol:** $\wedge$

**Gate:**



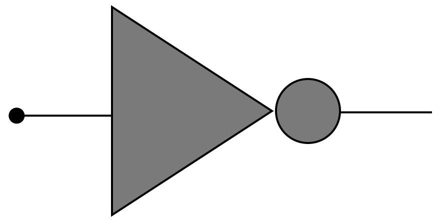
**Example:** $100110_2 \wedge 1010110_2 = 0000110_2$

# 4-5 LOGIC MICRO-OPERATIONS
## THE FOUR BASIC MICRO-OPERATIONS

### Complement (NOT) Microoperation

**Symbol:** $\overline{\phantom{A}}$

**Gate:**



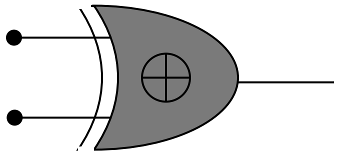
**Example:** $\overline{1010110_2} = 0101001_2$

# 4-5 LOGIC MICRO-OPERATIONS
## THE FOUR BASIC MICRO-OPERATIONS

### XOR (Exclusive-OR) Micro-operation

**Symbol: ⊕**

**Gate:**



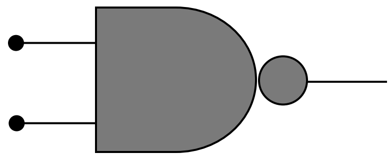
**Example: $100110_2 \oplus 1010110_2 = 1110000_2$**

# 4-5 LOGIC MICRO-OPERATIONS
# OTHER LOGIC MICRO-OPERATIONS

## NAND Micro-operation

**Symbols:** $\wedge$ **and** $\overline{\quad}$

**Gate:**



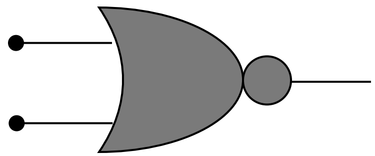
**Example:** $\overline{100110_2 \wedge 1010110_2} = 1111001_2$

# 4-5 LOGIC MICRO-OPERATIONS
# OTHER LOGIC MICRO-OPERATIONS

## NOR Micro-operation

**Symbols:** $\vee$ **and** $\overline{\quad}$

**Gate:**



**Example:** $\overline{100110_2 \vee 1010110_2} = 0001001_2$

# 4-5 LOGIC MICRO-OPERATIONS
# OTHER LOGIC MICRO-OPERATIONS

## Selective-set Operation

**Used to force selected bits of a register into logic-1 by using the OR operation**

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

In a processor register (refers to $0100_2$)

Loaded into a register from memory to perform the selective-set operation (refers to $1000_2$)

# 4-5 LOGIC MICRO-OPERATIONS
# OTHER LOGIC MICRO-OPERATIONS

**Selective-complement (toggling) Operation**

**Used to force selected bits of a register to be complemented by using the XOR operation**

**Example:** $0001_2 \oplus 1000_2 = 1001_2$

In a processor register

Loaded into a register from memory to perform the selective-complement operation

# 4-5 LOGIC MICRO-OPERATIONS
# OTHER LOGIC MICRO-OPERATIONS

**Insert Operation**

**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 MICRO-OPERATIONS
# OTHER LOGIC MICRO-OPERATIONS

**Set (Preset) Micro-operation**

Force all bits into 1's by ORing them with a value in which all its bits are being assigned to logic-1

**Example:** $\mathbf{100110_2 \vee 111111_2 = 111111_2}$

**Clear (Reset) Micro-operation**

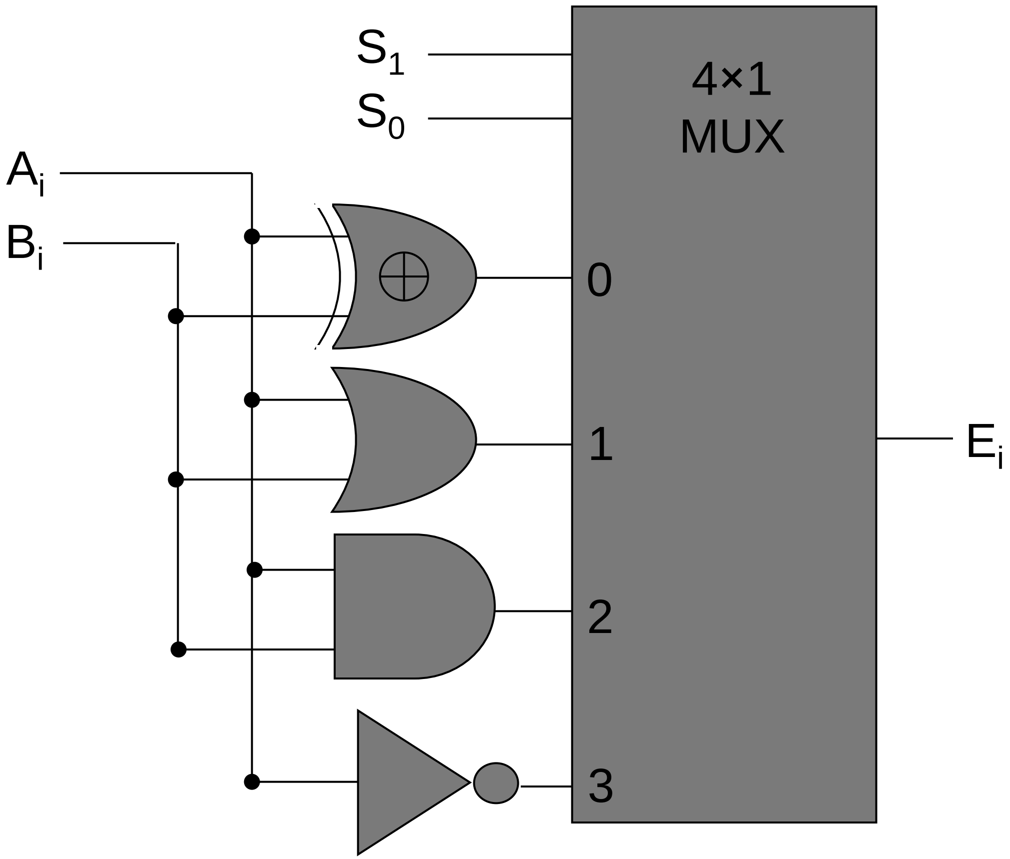
Force all bits into 0's by ANDing them with a value in which all its bits are being assigned to logic-0

**Example:** $\mathbf{100110_2 \wedge 000000_2 = 000000_2}$

# 4-5 LOGIC MICRO-OPERATIONS HARDWARE IMPLEMENTATION

- **The hardware implementation of logic micro-operations requires that logic gates be inserted for each bit or pair of bits in the registers to perform the required logic function**
- **Most computers use only four (AND, OR, XOR, and NOT) from which all others can be derived.**

# 4-5 LOGIC MICRO-OPERATIONS HARDWARE IMPLEMENTATION CONT.




| $S_1$ | $S_0$ | Output | Operation |
|---|---|---|---|
| 0 | 0 | $E = A \oplus B$ | XOR |
| 0 | 1 | $E = A \vee B$ | OR |
| 1 | 0 | $E = A \wedge B$ | AND |
| 1 | 1 | $E = \overline{A}$ | Complement |

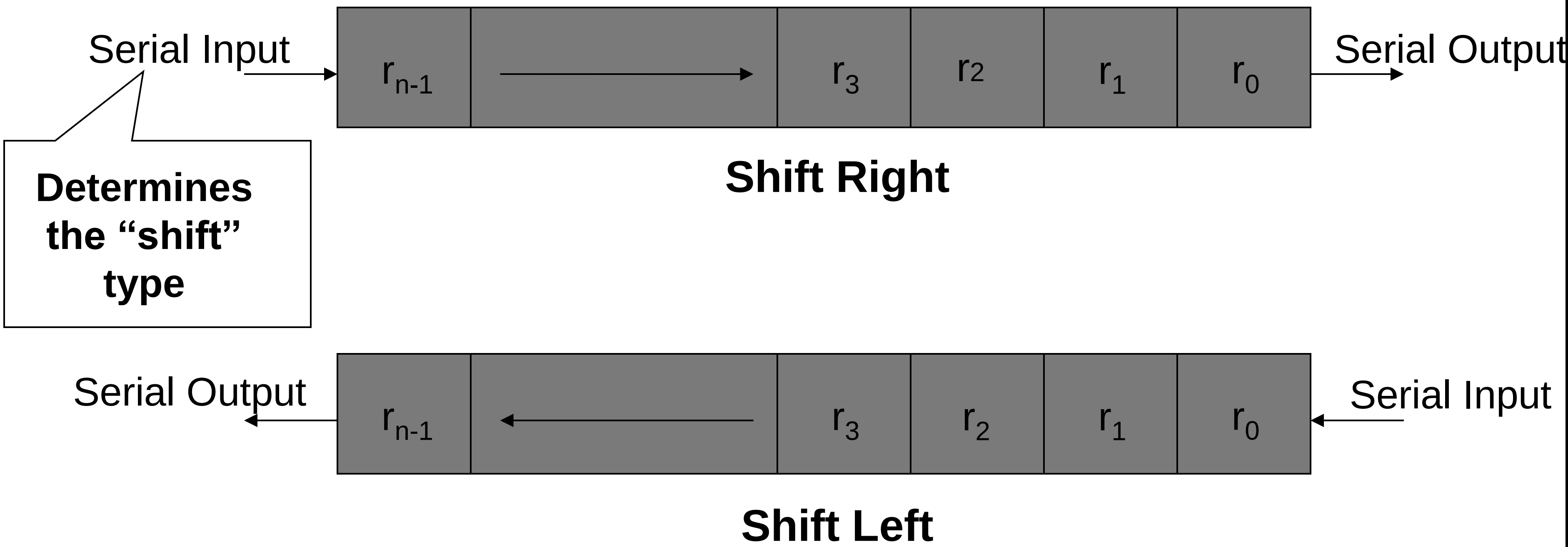
This is for one bit i

Figure B

# 4-6 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

# 4-6 SHIFT MICROOPERATIONS CONT.




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

# 4-6 SHIFT MICRO-OPERATIONS: LOGICAL SHIFTS

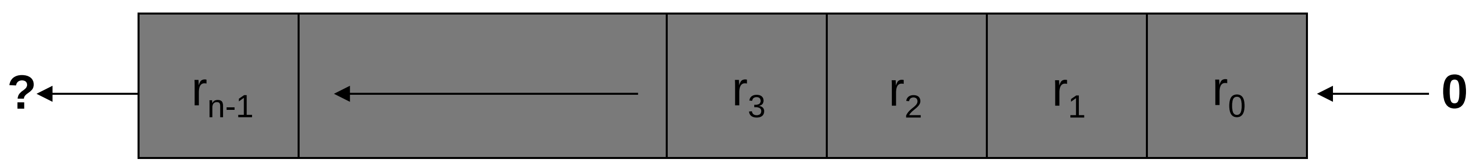
**Transfers 0 through the serial input**

**Logical Shift Right: R1←shr R1**

The same

**Logical Shift Left: R2←shl R2**

The same




**Logical Shift Left**

# 4-6 SHIFT MICRO-OPERATIONS: CIRCULAR SHIFTS (ROTATE OPERATION)

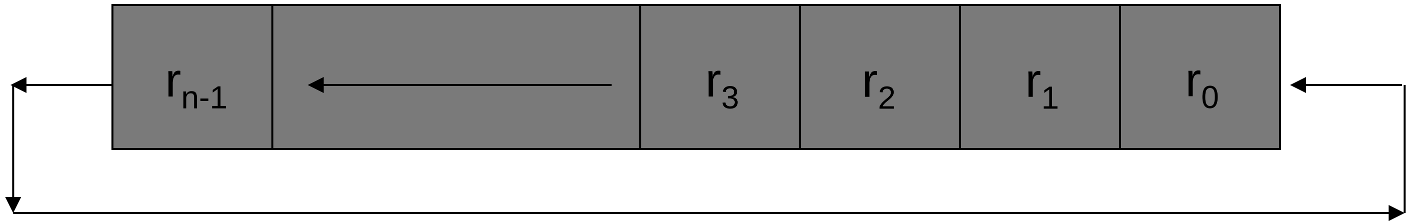
**Circulates the bits of the register around the two ends without loss of information**

**Circular Shift Right: R1←cir R1**

The same

**Circular Shift Left: R2←cil R2**

The same




**Circular Shift Left**

# 4-6 SHIFT MICRO-OPERATIONS ARITHMETIC SHIFTS

**Shifts a signed binary number to the left or right**

**An arithmetic shift-left multiplies a signed binary number by 2:**

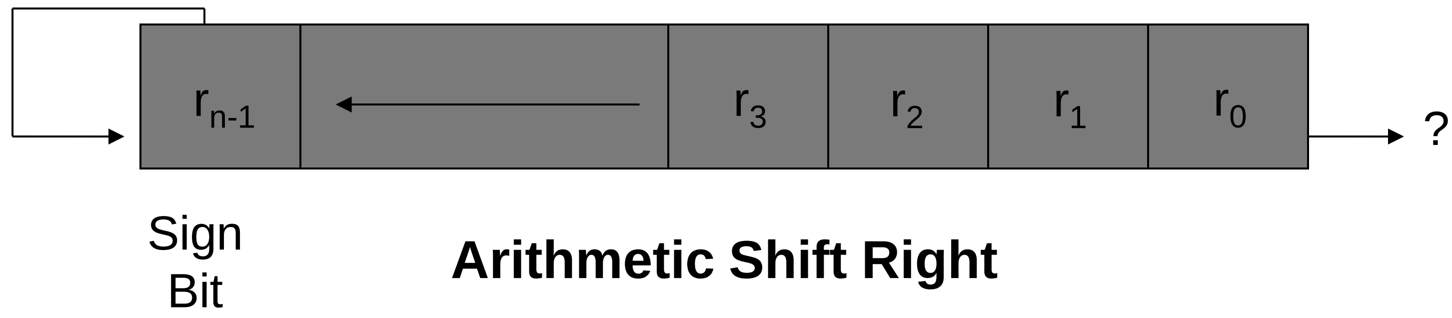
**ashl (00100): 01000**

**An arithmetic shift-right divides the number by 2**

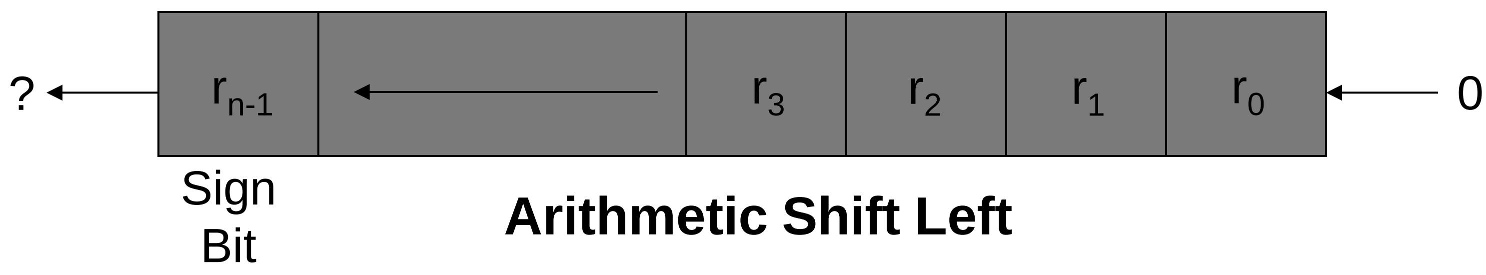
**ashr (00100) : 00010**

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

# 4-6 SHIFT MICRO-OPERATIONS ARITHMETIC SHIFTS CONT.




**Arithmetic Shift Right**




**Arithmetic Shift Left**

# 4-6 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

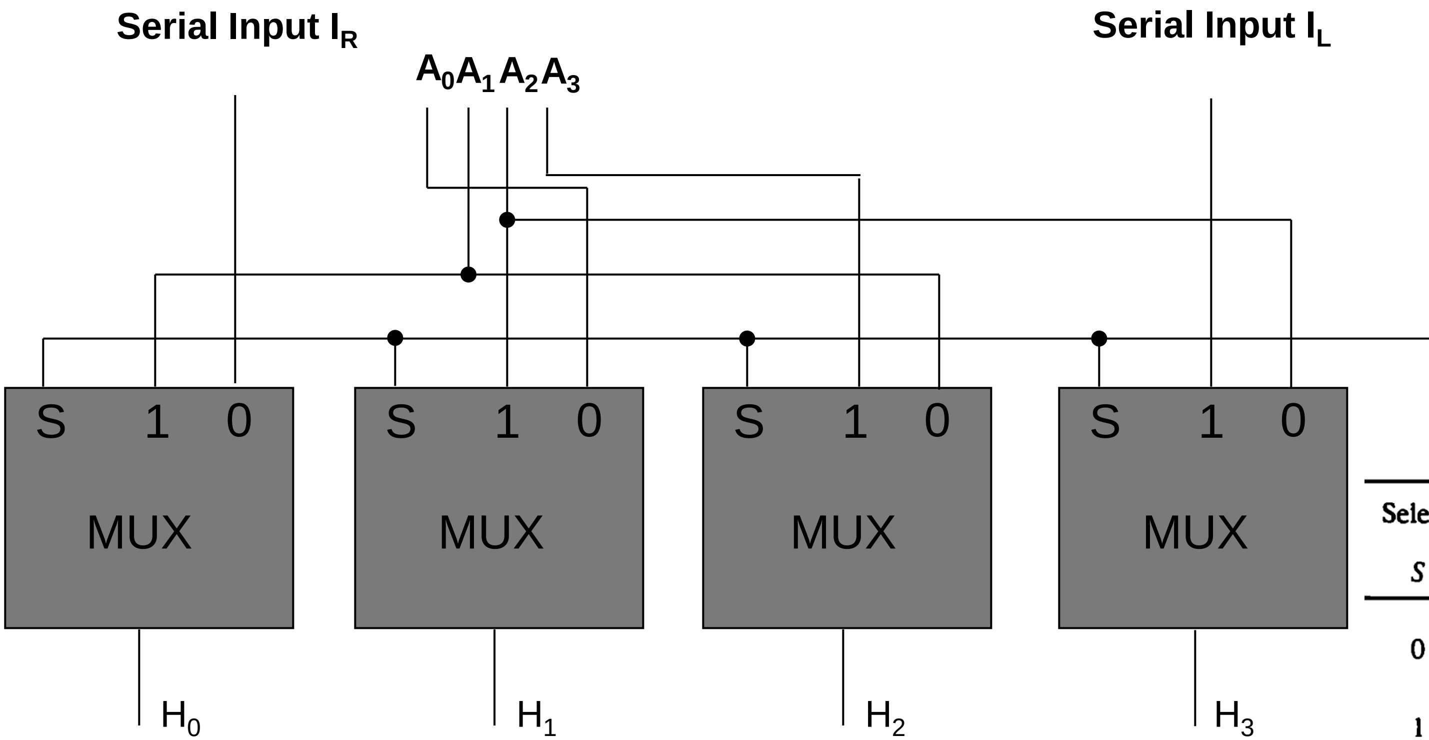
# 4-6 SHIFT MICRO-OPERATIONS HARDWARE IMPLEMENTATION

**A possible choice for a shift unit would be a bidirectional shift register with parallel load (refer to Fig 2-9). Has drawbacks:**

- Needs two pulses (the clock and the shift signal pulse)
- Not efficient in a processor unit where multiple number of registers share a common bus

**It is more efficient to implement the shift operation with a combinational circuit**

# 4-6 SHIFT MICRO-OPERATIONS HARDWARE IMPLEMENTATION CONT.




Select

0 for shift right

1 for shift left

Function table

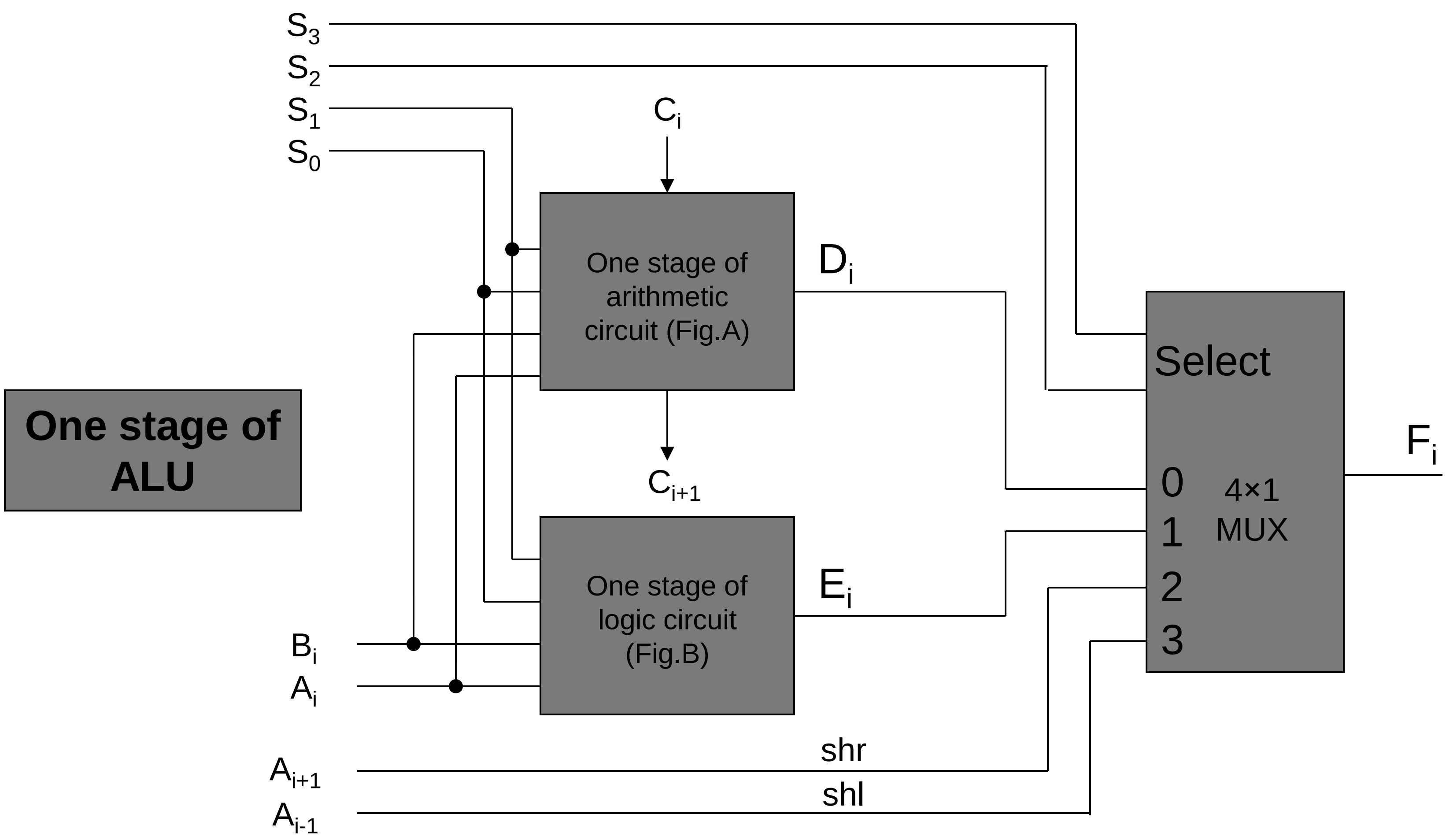
| Select | Output | | | |
|---|---|---|---|---|
| $S$ | $H_0$ | $H_1$ | $H_2$ | $H_3$ |
| 0 | $I_R$ | $A_0$ | $A_1$ | $A_2$ |
| 1 | $A_1$ | $A_2$ | $A_3$ | $I_L$ |

**4-bit Combinational Circuit Shifter**

# 4-7 ARITHMETIC LOGIC SHIFT UNIT

**Instead of having individual registers performing the Microoperations directly, computer systems employ a number of storage registers connected to a common operational unit called an Arithmetic Logic Unit (ALU)**

# 4-7 ARITHMETIC LOGIC SHIFT UNIT
## CONT.



# 4-7 FUNCTION TABLE ARITHMETIC LOGIC SHIFT UNIT

| Operation select | | | | | | |
|---|---|---|---|---|---|---|
| $S_3$ | $S_2$ | $S_1$ | $S_0$ | $C_{in}$ | 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 | $F = A + \overline{B}$ | Subtract with borrow |
| 0 | 0 | 1 | 0 | 1 | $F = A + \overline{B} + 1$ | Subtraction |
| 0 | 0 | 1 | 1 | 0 | $F = A - 1$ | Decrement $A$ |
| 0 | 0 | 1 | 1 | 1 | $F = A$ | Transfer $A$ |
| 0 | 1 | 0 | 0 | × | $F = A \wedge B$ | AND |
| 0 | 1 | 0 | 1 | × | $F = A \vee B$ | OR |
| 0 | 1 | 1 | 0 | × | $F = A \oplus B$ | XOR |
| 0 | 1 | 1 | 1 | × | $F = \overline{A}$ | Complement $A$ |
| 1 | 0 | × | × | × | $F = \text{shr } A$ | Shift right $A$ into $F$ |
| 1 | 1 | × | × | × | $F = \text{shl } A$ | Shift left $A$ into $F$ |