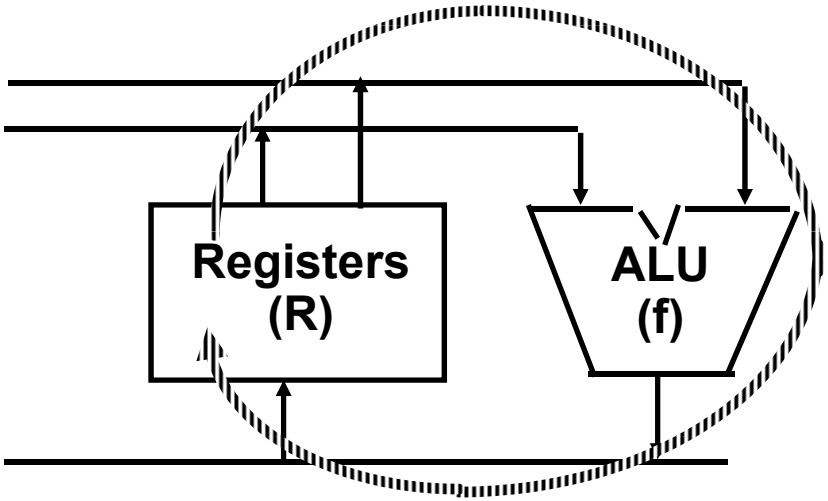
# REGISTER TRANSFER AND MICROOPERATIONS

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

# MICROOPERATION

**An elementary operation performed during one clock pulse, on the information stored in one or more registers**




1 clock cycle

$R \leftarrow f(R, R)$

f: shift, count, clear, load, add,...

# REGISTER TRANSFER LANGUAGE

**Definition of the (internal) organization of a computer**

- **Set of registers and their functions**
- **Microoperations Set**
  **Set of allowable microoperations provided by the organization of the computer**
- **Control signals that initiate the sequence of microoperations**

**For any function of the computer, a sequence of microoperations is used to describe it**

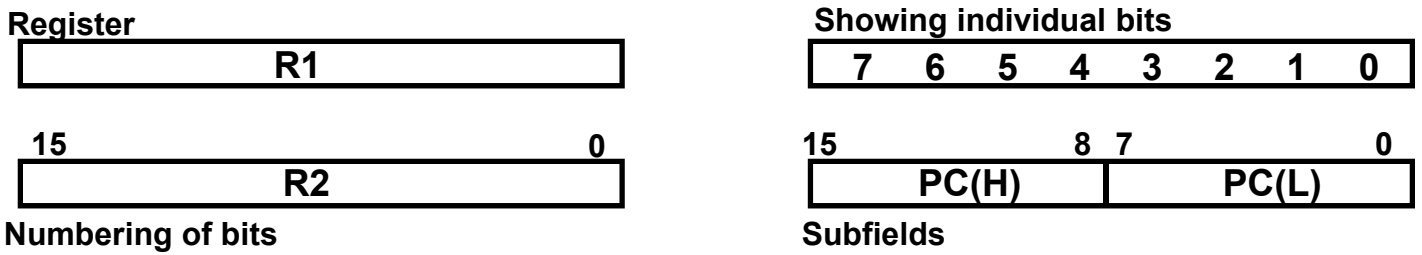
**----> *Register transfer language***

- **A symbolic language**
- **Used to describe the microoperation transfers among registers**

# REGISTER TRANSFER

**Designation of a register**
- a register
- portion of a register
- a bit of a register

**Common ways of drawing the block diagram of a register**




**Representation of a transfer(parallel)**

$R2 \leftarrow R1$

A simultaneous transfer of all bits from the source to the destination register, during one clock pulse

**Representation of a controlled(conditional) transfer**

P: $R2 \leftarrow R1$

A binary condition(p=1) which determines when the transfer is to occur

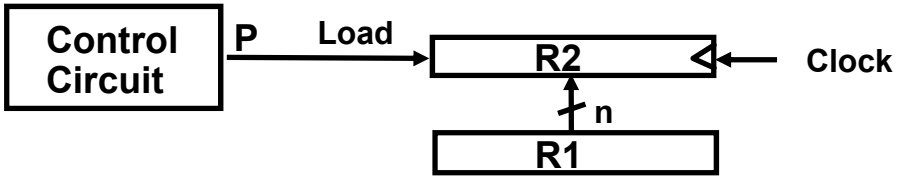
If (p=1) then ($R2 \leftarrow R1$)

# HARDWARE IMPLEMENTATION OF CONTROLLED TRANSFERS

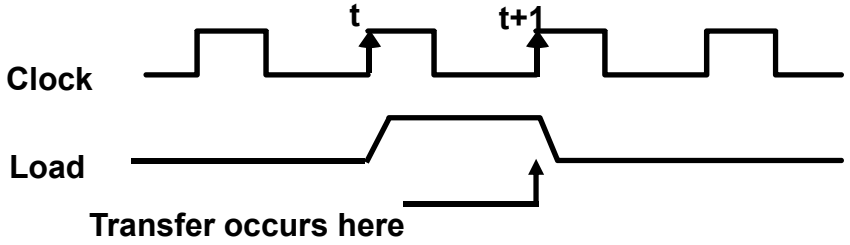
## Implementation of controlled transfer

P: R2 ← R1

## Block diagram




## Timing diagram



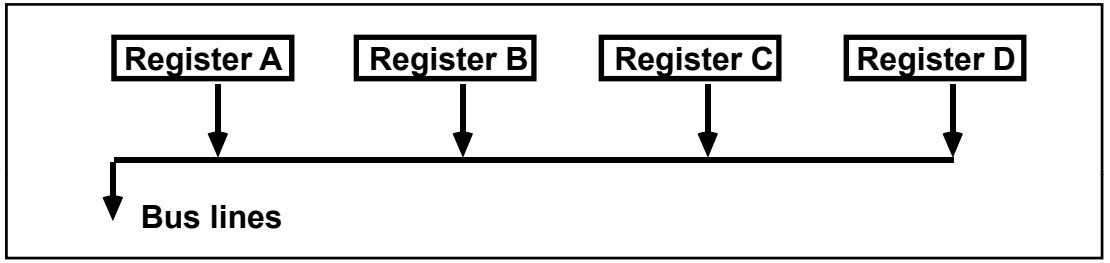

## Basic Symbols for Register Transfers

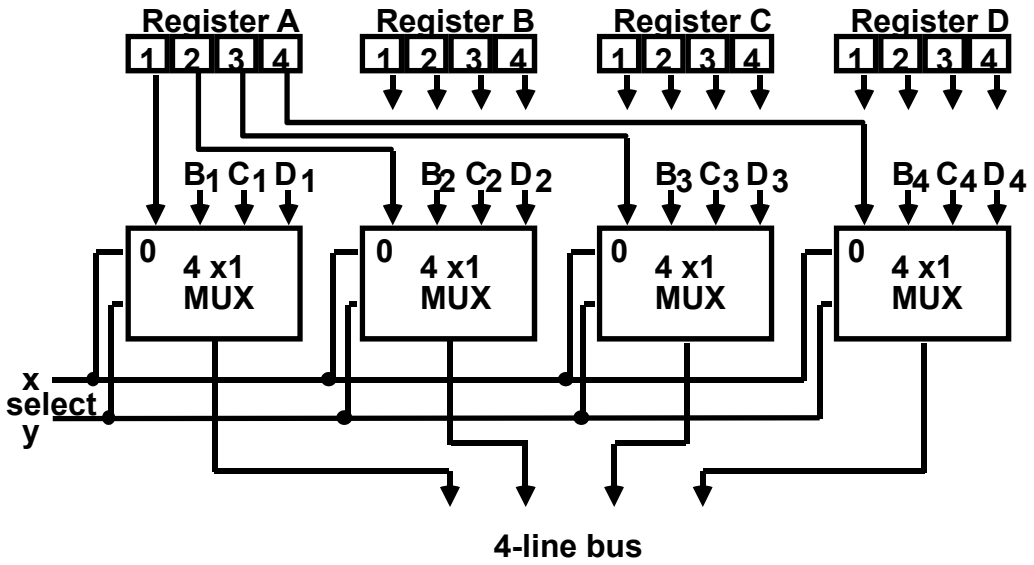
| Symbols | Description | Meaning |
|---|---|---|
| Capital letters and numerals | Denotes a register | MAR, R2 |
| Parentheses ( ) | Denotes a part of a register | R2(0-7), R2(L) |
| Arrow ← | Denotes transfer of information | R2 ← R1 |
| Colon : | Denotes termination of control function | P: |
| Comma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

# BUS AND MEMORY TRANSFER

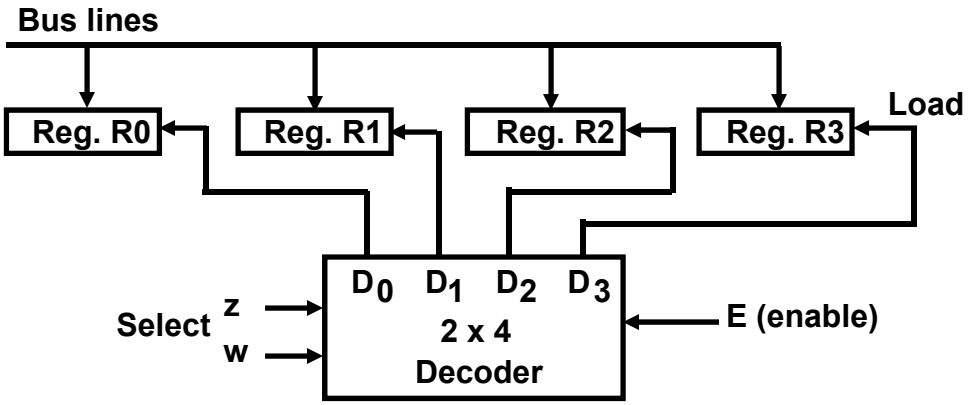
**Bus is a path(of a group of wires) over which information is transferred, from any of several sources to any of several destinations.**

**From a register to bus: BUS <- R**

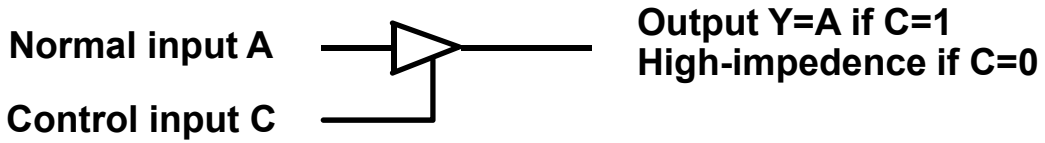


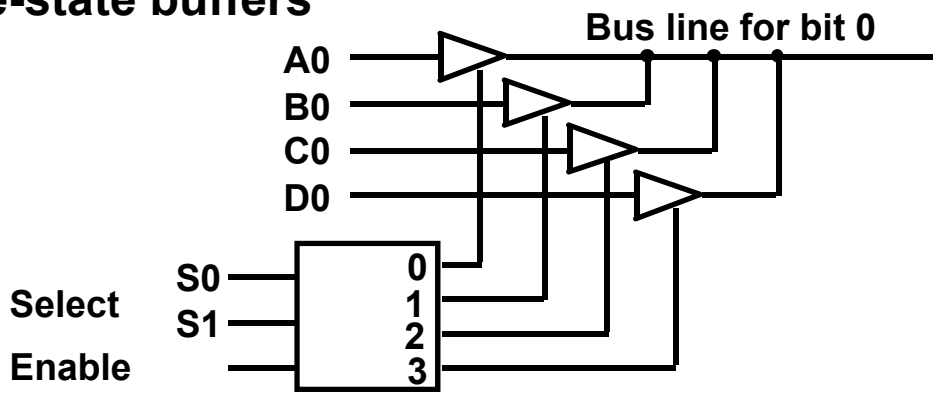
# TRANSFER FROM BUS TO A DESTINATION REGISTER



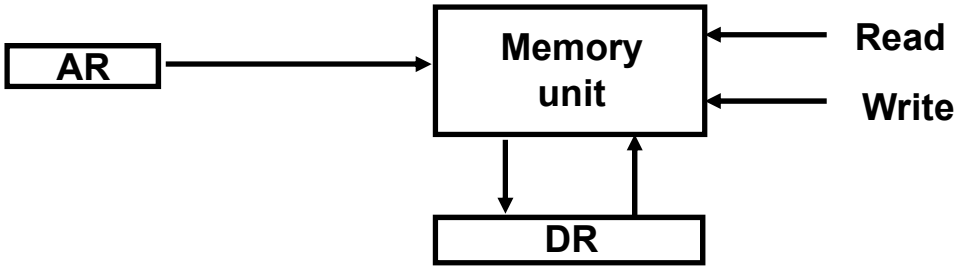

## Three-State Bus Buffers




## Bus line with three-state buffers



# MEMORY TRANSFERS




**Memory *read* micro-op: DR ← M or ( DR ← M[AR] )**

**Memory *write* micro-op: M ← DR or ( M[AR] ← DR )**

# ARITHMETIC MICROOPERATIONS

**Four types of microoperations**

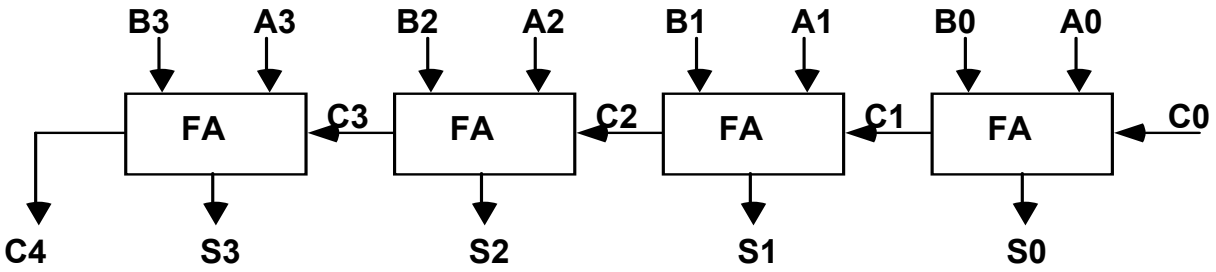
- **Register transfer microoperations**
- **Arithmetic microoperations**
- **Logic microoperations**
- **Shift microoperations**

*** Summary of Arithmetic Micro-Operations**

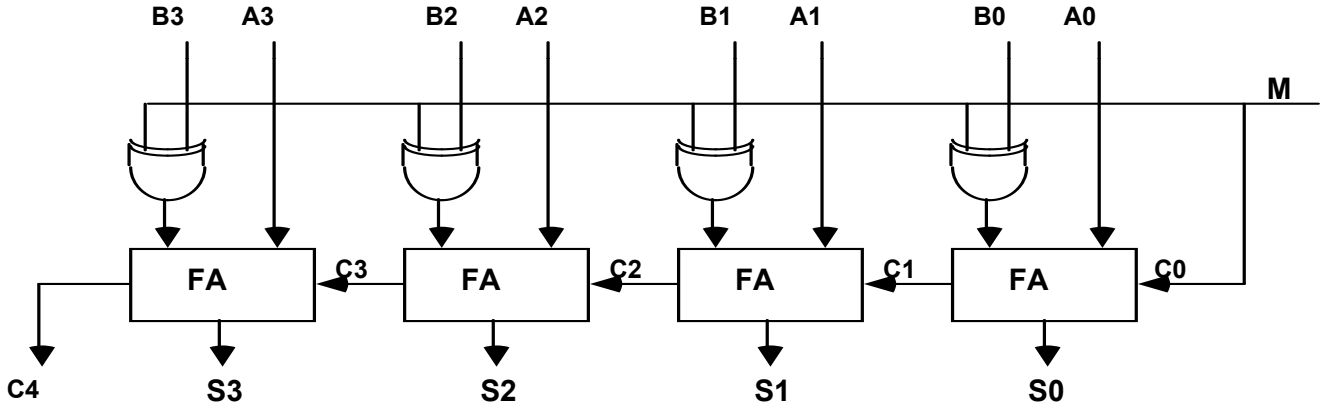
| | |
|---|---|
| R3 ← R1 + R2 | Contents of R1 plus R2 transferred to R3 |
| R3 ← R1 - R2 | Contents of R1 minus R2 transferred to R3 |
| R2 ← R2' | Complement the contents of R2 |
| R2 ← R2'+ 1 | 2's complement the contents of R2 (negate) |
| R3 ← R1 + R2'+ 1 | subtraction |
| R1 ← R1 + 1 | Increment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

# BINARY ADDER

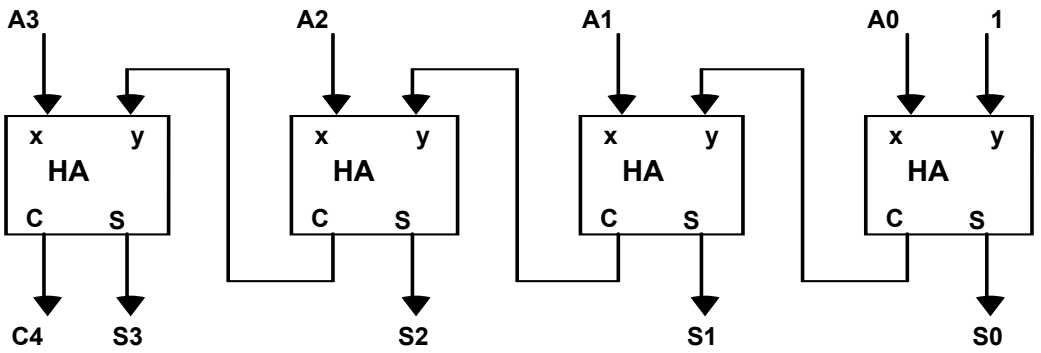
## Binary Adder



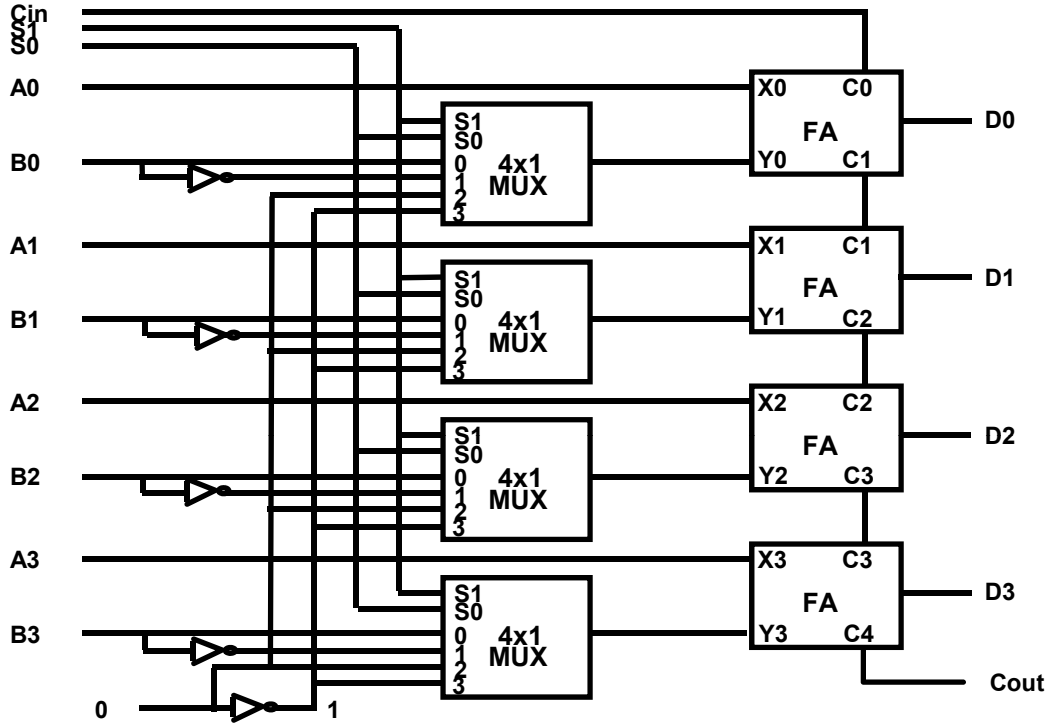

## Binary Adder-Subtractor




## 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.**

**- useful for bit manipulations on binary data**

**AND: Mask out certain group of bits**
**OR : Merge binary or character data**

**- useful for making logical decisions based on the bit value**

**Applications**

**Manipulating individual bits or a field(portion) of a word in a register**

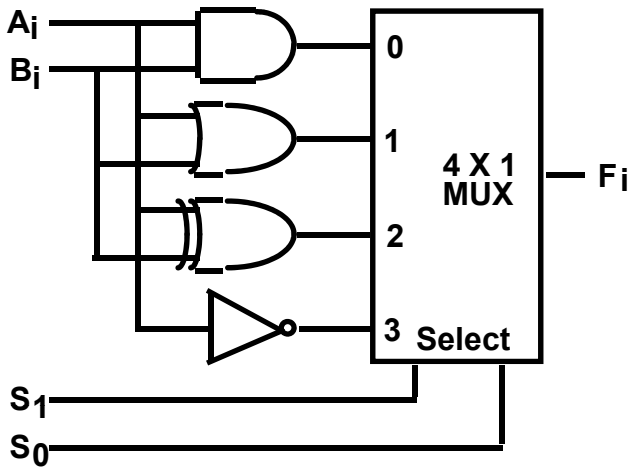
| | |
|---|---|
| **- Selective-set** | $A + B$ |
| **- Selective-complement** | $A \oplus B$ |
| **- Selective-clear** | $A \cdot \overline{B}$ |
| **- Mask (Delete)** | $A \cdot B$ |
| **- Insert** | $(A \cdot B) + C$ |
| **- Compare** | $A \oplus B$ |
| **- Packing** | $(A \cdot B) + C$ |
| **- Unpacking** | $A \cdot B$ |

# LIST OF LOGIC MICROOPERATIONS

- List of Logic Micro-Operations
  - 16 different logic operations with 2 binary vars.
  - n binary vars -> $2^{2^n}$ functions
- Truth tables for 16 functions of 2 variables and the corresponding 16 logic micro-operations

| x<br>y | 0 0 1 1<br>0 1 0 1 | *Boolean Function* | *Micro-Operations* | *Name* |
|---|---|---|---|---|
| | 0 0 0 0 | F0 = 0 | F <– 0 | Clear |
| | 0 0 0 1 | F1 = xy | F <– A ∧ B | AND |
| | 0 0 1 0 | F2 = xy' | F <– A ∧ B’ | |
| | 0 0 1 1 | F3 = x | F <– A | Transfer A |
| | 0 1 0 0 | F4 = x'y | F <– A’∧ B | |
| | 0 1 0 1 | F5 = y | F <– B | Transfer B |
| | 0 1 1 0 | F6 = x ⊕ y | F <– A ⊕ B | Exclusive-OR |
| | 0 1 1 1 | F7 = x + y | F <– A ∨ B | OR |
| | 1 0 0 0 | F8 = (x + y)' | F <– (A ∨ B)’ | NOR |
| | 1 0 0 1 | F9 = (x ⊕ y)' | F <– (A ⊕ B)’ | Exclusive-NOR |
| | 1 0 1 0 | F10 = y' | F <– B’ | Complement B |
| | 1 0 1 1 | F11 = x + y' | F <– A ∨ B | |
| | 1 1 0 0 | F12 = x' | F <– A’ | Complement A |
| | 1 1 0 1 | F13 = x' + y | F <– A’∨ B | |
| | 1 1 1 0 | F14 = (xy)' | F <– (A ∧ B)’ | NAND |
| | 1 1 1 1 | F15 = 1 | F <– all 1's | Set to all 1's |

# HARDWARE IMPLEMENTATION OF LOGIC MICROOPERATIONS




**Function table**

| S1 | S0 | Output | μ-operation |
|---|---|---|---|
| 0 | 0 | $F = A \wedge B$ | AND |
| 0 | 1 | $F = A \vee B$ | OR |
| 1 | 0 | $F = A \oplus B$ | XOR |
| 1 | 1 | F = A' | Complement |

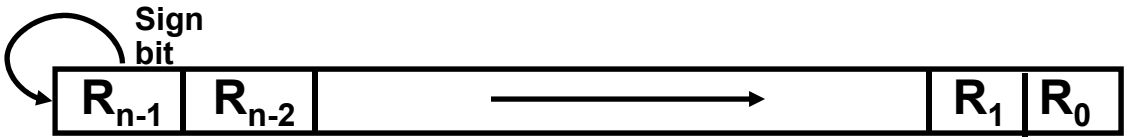
# SHIFT MICROOPERATIONS

**Shifts**

- *Logical shift* : shift in a 0 into the extreme flip-flop
- *Circular shift* : circulates the bits of the register around the two ends
- *Arithmetic shift* : shifts a signed number (shift with sign extension)
  Left shift -> multiplied by 2
  Right shift -> divided by 2

**Arithmetic shifts for signed binary numbers**

- Arithmetic shift-right

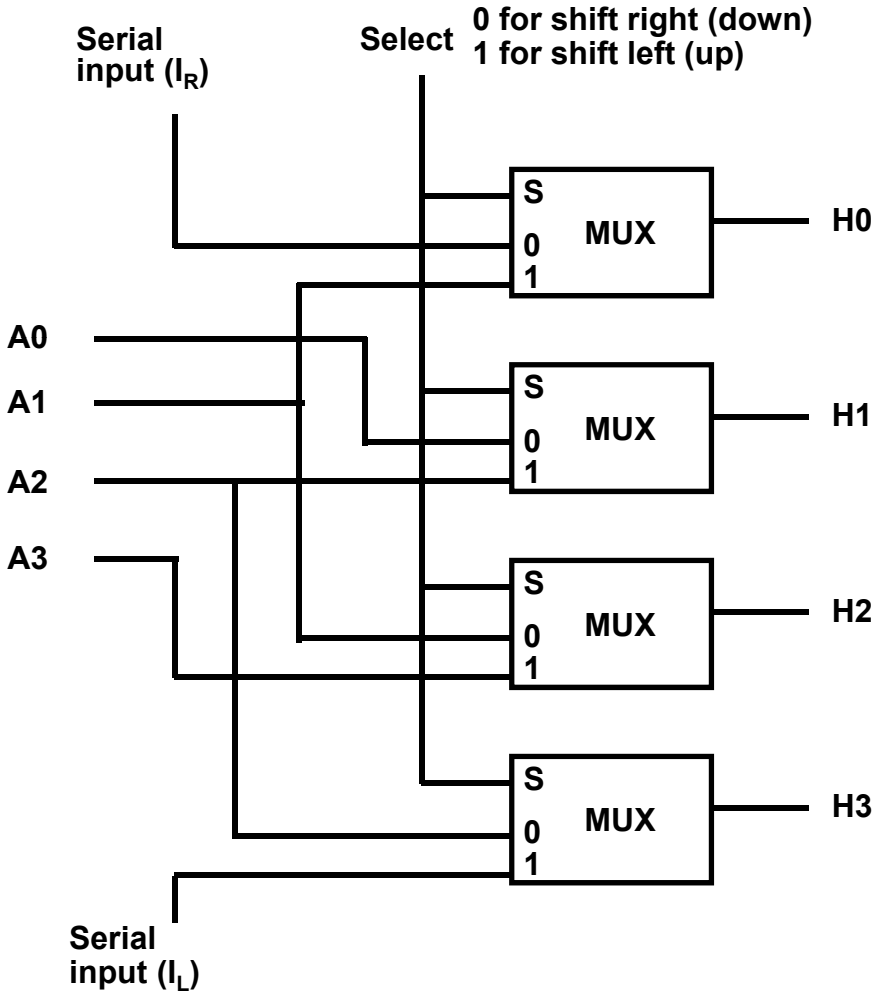



- Arithmetic shift-left Overflow V may occur

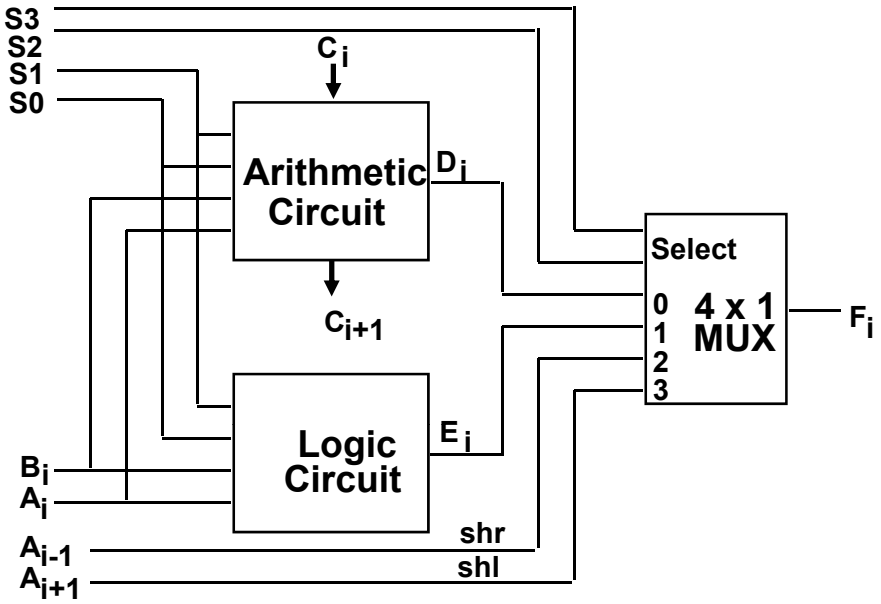
**Shift Micro-Operations**

| *Symbol* | *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 right-shift register R |
| R ← ashl R | Arithmetic shift-left register R |
| R ← ashr R | Arithmetic shift-right register R |

# HARDWARE IMPLEMENTATION OF SHIFT MICROOPERATIONS



# 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 |