Systems I: Computer Organization and Architecture

Lecture 9 - Register Transfer and Microoperations

Microoperations

- Digital systems are modular in nature, with modules containing registers, decoders, arithmetic elements, control logic, etc.
- These digital components are defined by the registers that they contain and the operations performed on their data. These operations are called microoperations.
- Microoperations are elementary operations performed on the information stored in one or more registers.

Hardware Organization

- The hardware organization of a digital computer is best defined by specifying:
  - The set of register that it contains and their function.
  - The sequence of microoperations performed on the binary information stored in the registers.
  - The control signals that initiates the sequence of microoperations.

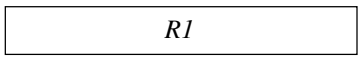
Register Transfer Language

- A register transfer language is a notation used to describe the microperation transfers between registers.
- It is a system for expressing in symbolic form the microoperation sequences among register that are used to implement machine-language instructions.

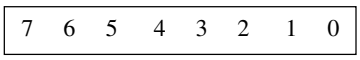
Registers and Register Transfer

- Registers are denoted by capital letters and are sometimes followed by numerals, e.g.,
  - MAR – Memory Address Register (holds addresses for the memory unit)
  - PC – Program Counter (holds the next instruction's address)
  - IR – Instruction Register (holds the instruction being executed)
  - R1 – Register 1 (a CPU register)
- We can indicate individual bits by placing them in parentheses, e.g., PC(8-15), R2(5), etc.

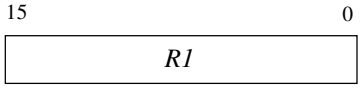
Block Diagrams of Registers



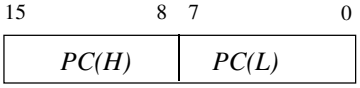
Register R



Showing Individual Bits



Numbering of Bits



Divided Into Two Parts

Register Transfer Language Instructions

- Register Transfer $R2 \leftarrow R1$
- Simultaneous Transfer

  $R2 \leftarrow R1, R1 \leftarrow R2$
- Conditional Transfer (***Control Function***)

  P: $R2 \leftarrow R1$

  or

  If (P = 1) Then $R2 \leftarrow R1$
- Conditional, Simultaneous Transfer

  T: $R2 \leftarrow R1, R1 \leftarrow R2$

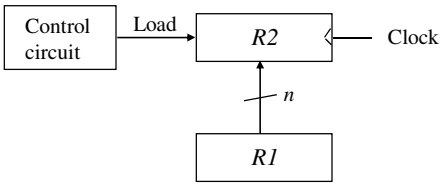
Basic Symbols For Register Transfer

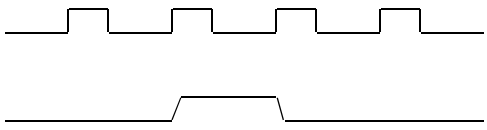
Symbol	Description	Examples
Letters (and numerals)	Denotes a register	MAR, R2
Parentheses ( )	Denotes a part of a register	R2(0-7), R2(L)
Arrow $\rightarrow$	Denotes Transfer of information	$R2 \leftarrow R1$
Comma ,	Separates 2 microoperations	$R2 \leftarrow R1, R1 \leftarrow R1$

Register Transfer and Hardware

- Every statement in Register Transfer Language implies the existence of hardware that implements the microoperation.
- The statement P: R2 ← R1 implies the existence of the necessary circuitry to implement the transfer as well as the mechanism to set and clear the control variable P.

Transfer from R1 to R2 when P = 1





The Bus

- A bus is a set of common wires that carries data between registers.
  - There is a separate wire for every bit in the registers.
  - There are also a set of control signals which determines which register is selected by the bus at a particular time.
- A bus can be constructed using multiplexer which enable a sets of registers to share a common bus for data transfer.

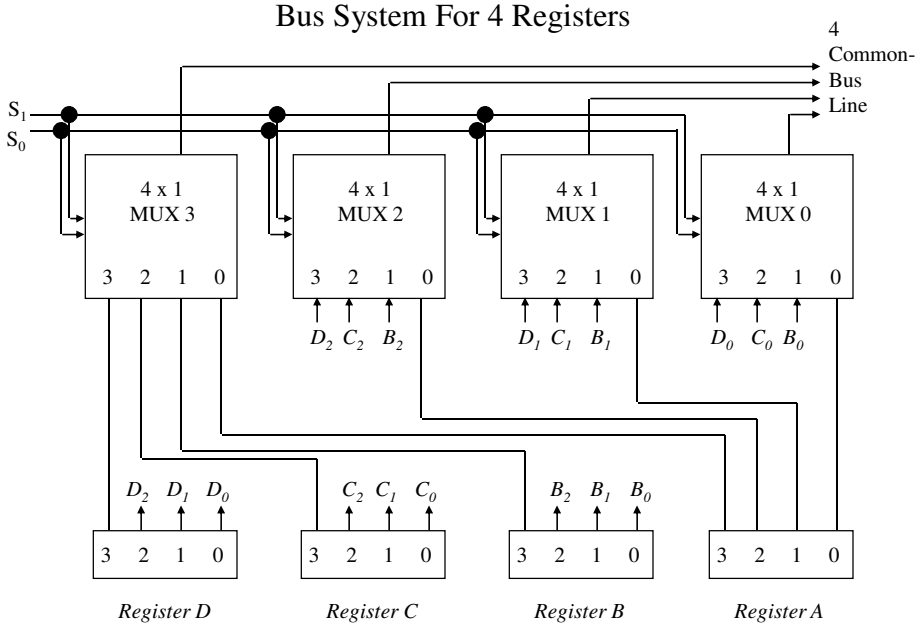
Data Transfer Using the Bus

- The select lines S_1 and S_0 indicate which of four register will have its contents transferred to the bus.
- In general, a bus system will multiplex *k* registers of *n* bit each to produce a *n*–line common bus.
- It will require *n* k x 1 multiplexers.
- The bus is connected to the inputs of all destination registers. and will activate the load control of the selected register when it is ready to transfer data. This can be written as:
- $R2 \leftarrow BUS,\ BUS \leftarrow R1$ **or** $R2 \leftarrow R1$

Function Table for the Bus

S_1	S_0	Register Selected
0	0	A
0	1	B
1	0	C
1	1	D

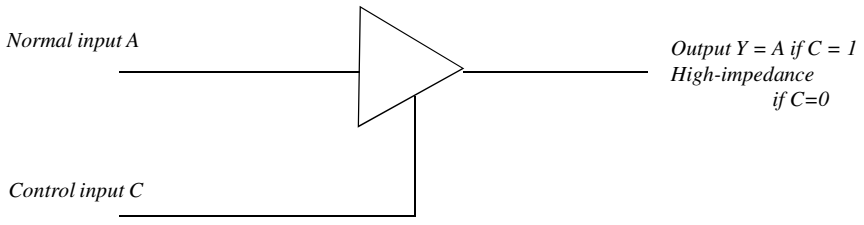
Bus System For 4 Registers



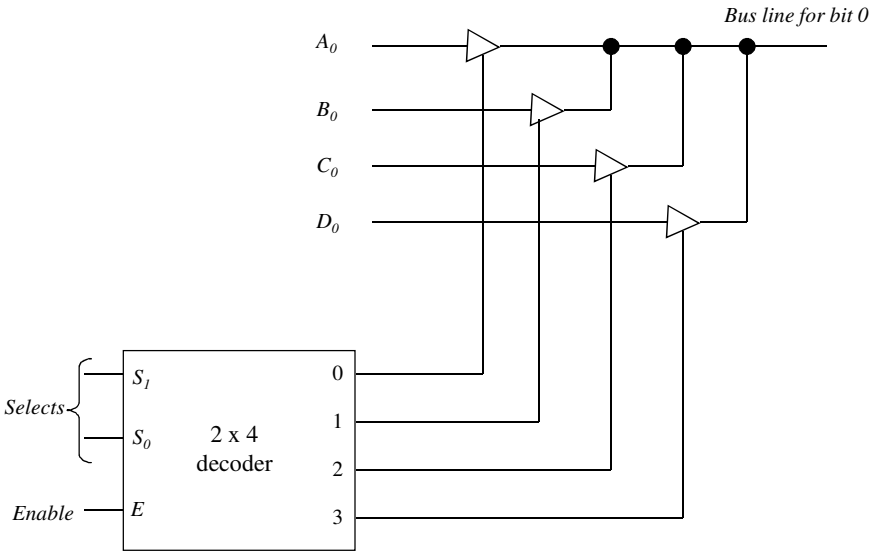
Three State-Bus Buffers

- A bus can be built using three-state buffers instead of multiplexers.
- A three-state gate has three states: 1, 0 and a high-impedance state, which behaves like an open circuit.
- It is possible to connect a large number of three-state gates in a common bus line without overloading it.

Graphic Symbols For Three State-Buffer



Bus Line With Three-State Buffers



Memory Transfer

- There are two primary operations involving memory:
  - ***Read*** – transferring data ***from*** memory
  - ***Write*** – transferring data ***into*** memory
- To indicate in Register Transfer Language that we are moving data from a memory address to the data register, we write:

  Read: DR ← M[AR]
- To indicate in RTL that we are moving data from Register 1 to a memory location, we write:

  Write: M[AR] ← R1

Microoperations

- Microoperations are classified into four categories:
  - Register transfer microoperations (data moves from register to register)
  - Arithmetic microoperations (perform arithmetic on data in registers)
  - Logic microoperations (perform bit manipulation on data in registers)
  - Shift microoperations (perform shift on data in registers)

Arithmetic Microoperations

- Unlike register transfer microoperations, arithmetic microoperations change the information content.
- The basic arithmetic microoperations are:
  - addition
  - subtraction
  - increment
  - decrement
  - shift

Arithmetic Microoperations (continued)

- The RTL statement:

  $R3 \leftarrow R1 + R2$

  indicates an add microoperation. We can similarly specify the other arithmetic microoperations.
- Multiplication and division are not considered microoperations.
  - Multiplication is implemented by a sequence of adds and shifts.
  - Division is implemented by a sequence of substracts and shifts.

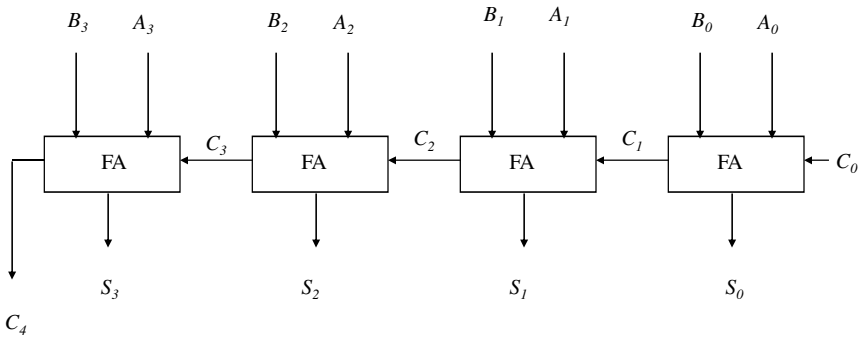
Arithmetic Microoperations

Symbolic Designation	**Description**
$R3 \leftarrow R1 + R2$	Contents of R1 plus R2 transferred to R3
$R3 \leftarrow R1 - R2$	Contents of R1 minus R2 transferred to R3
$R2 \leftarrow \overline{R2}$	Complement contents of R2 (1's comp.)
$R2 \leftarrow \overline{R2} + 1$	2's complment contens of R2 (negate)
$R3 \leftarrow R1+\overline{R2} + 1$	R1 plus 2's comp. of R2
$R1 \leftarrow R1 + 1$	Increment content of R1 by 1
$R1 \leftarrow R1 - 1$	Decrement content of R1 by 1

Binary Adder

- We implement a binary adder with registers to hold the data and a digital circuit to perform the addition (called a ***binary adder***).
- The binary adders is constructed using full adders connected in cascade so that the carry produced by one full adder becomes an input for the next.
- Adding two *n*-bit numbers requires *n* full adders.
- The *n* data bits for ***A*** and ***B*** might come from ***R1*** and ***R2*** respectively

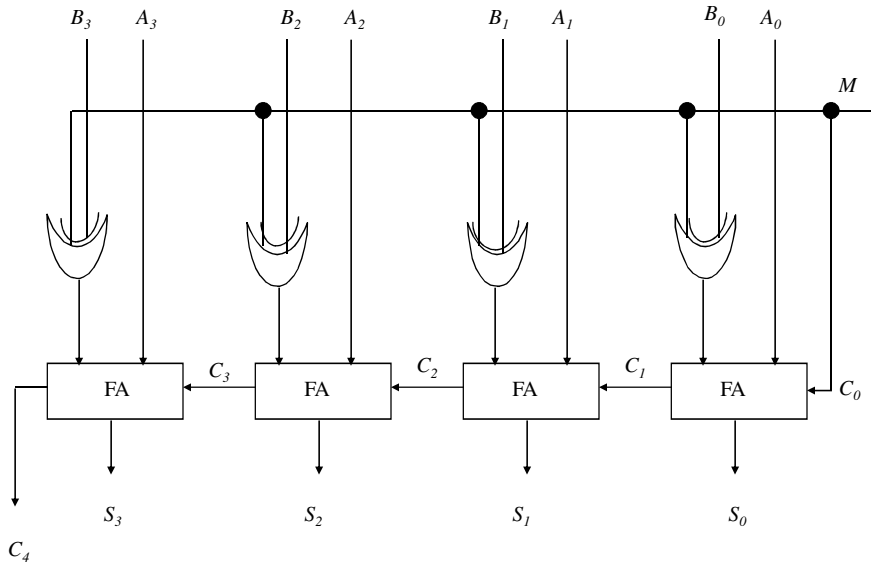
4-Bit Binary Adder



Adder-Subtracter

- Subtracting ***A – B*** is most easily done by adding ***B'*** to ***A*** and then adding 1.
- This makes it convenient to combine both addition and subtraction into one circuit, called an adder-subtracter.
- M is the mode indicator
  - M = 0 indicates addition (B is left alone and C_0 is 0)
  - M = 1 indicates subtraction (B is complement and C_0 is 1).

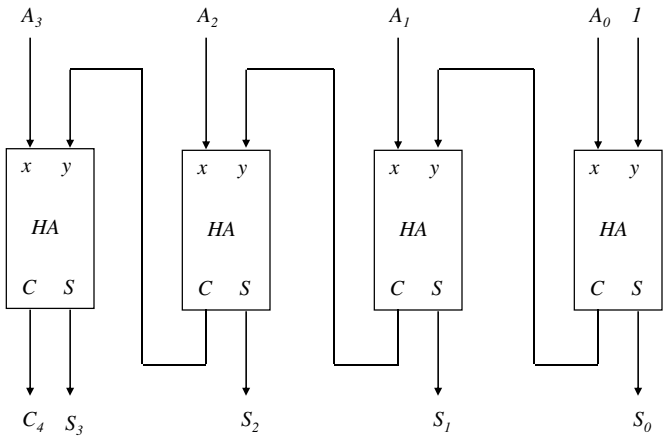
4-Bit Adder-Subtractor



Binary Incrementer

- The binary incrementer adds 1 to the contents of a register, e.g., a register storing 0101 would have 0110 in it after being incremented.
- There are times when we want incrementing done independent of a register. We can accomplish this with a series of cascading half-adders.

4-Bit Binary Incrementer



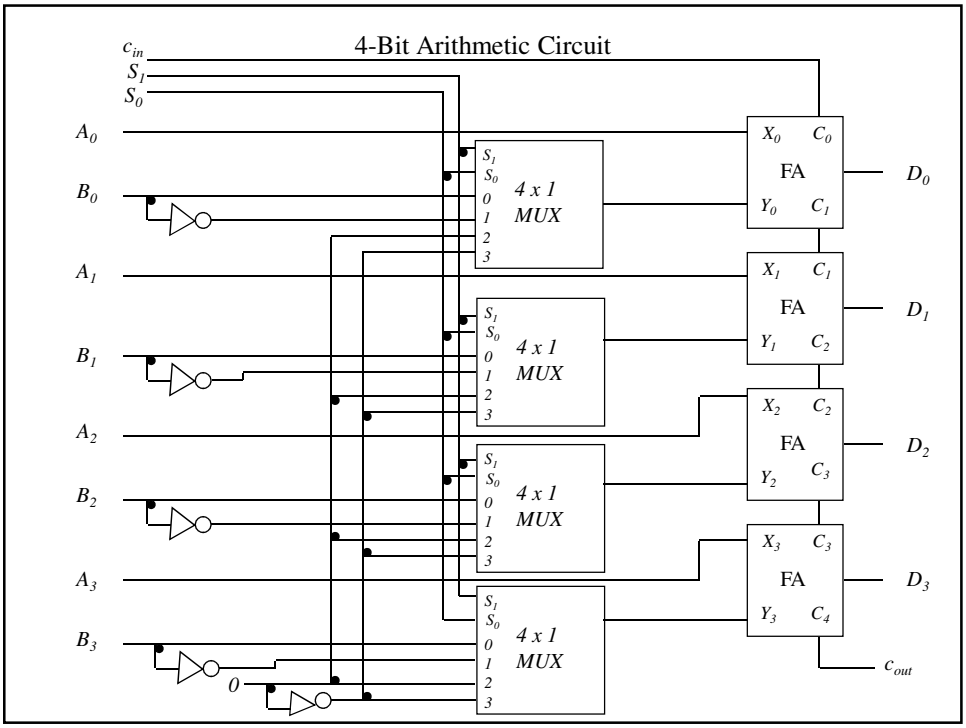
Arithmetic Circuit

- We can implement 7 arithmetic micro-operations (add, add with carry, subtract, subtract with borrow, increment, decrement and transfer) with one circuit.
- We provide a series of cascading full adders with A_i and the output of a 4x1 multiplexer.
  - The multiplexers' inputs are two selects, B_i, B_i', logical 0 and logical 1.
  - Which of these four values we provide (together with the carry) determines which microoperation is performed.

Arithmetic Circuit Function Table

Select **Input** **Output**

S_1	S_0	C_{in}	Y	$D = A + Y + C_{in}$	Microoperation
0	0	0	B	D = A + B	Add
0	0	1	B	D = A + B + 1	Add with Carry
0	1	0	$\overline{B}$	D = A + $\overline{B}$	Subtract with Borrow
0	1	1	$\overline{B}$	D = A + $\overline{B}$ + 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



The Microoperations of the Arithmetic Circuit

- When $S_1S_0 = 00$, the MUX provides B. The result is Add (for $C_{in} = 0$) or Add With Carry (for $C_{in} = 1$).
- When $S_1S_0 = 01$, the MUX provides B'. The result is Subtract with Borrow (for $C_{in} = 0$) or Subtract (for $C_{in} = 1$).
- When $S_1S_0 = 10$, the MUX provides 0. The result is Transfer (for $C_{in} = 0$) or Increment (for $C_{in} = 1$).
- When $S_1S_0 = 11$, the MUX provides 1. The result is Decrement (for $C_{in} = 0$) or Transfer (for $C_{in} = 1$).

Logic Microoperations

- Logic microoperations are binary operations performed on corresponding bits of two bit strings.
- Example: P: R1 ← R1 ⊕ R2

  1010 Content of R1

  1100 Content of R2

  0110 Content of R1 after P = 1
- Special Symbols used for logic operations:

  ∧ - AND ∨ - OR ⊕ - XOR

  This avoids confusing AND with multiplication, OR with addition, etc.

Truth Tables for 16 2-Variable Function

x	y	F_0	F_1	F_2	F_3	F_4	F_5	F_6	F_7
0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	1	1	1	1
1	0	0	0	1	1	0	0	1	1
1	1	0	1	0	1	0	1	0	1

Truth Tables for 16 2-Variable Function (continued)

x	y	F_8	F_9	F_{10}	F_{11}	F_{12}	F_{13}	F_{14}	F_{15}
0	0	1	1	1	1	1	1	1	1
0	1	0	0	0	0	1	1	1	1
1	0	0	0	1	1	0	0	1	1
1	1	0	1	0	1	0	1	0	1

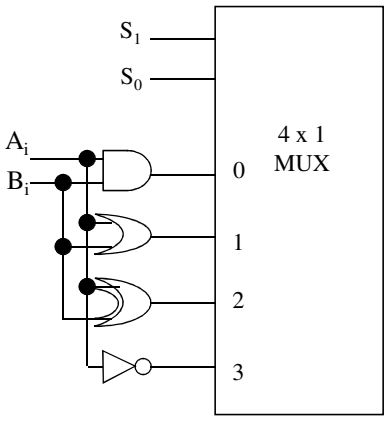
Sixteen Logic Microoperations

Boolean Function	Microoperation	Name
$F_0 = 0$	$F \leftarrow 0$	Clear
$F_1 = xy$	$F \leftarrow A \wedge B$	AND
$F_2 = xy'$	$F \leftarrow A \wedge \overline{B}$	
$F_3 = x$	$F \leftarrow A$	Transfer A
$F_4 = x'y$	$F \leftarrow \overline{A} \wedge B$	
$F_5 = y$	$F \leftarrow B$	Transfer B
$F_6 = x \oplus y$	$F \leftarrow A \oplus B$	Exclusive-OR
$F_7 = x + y$	$F \leftarrow A \vee B$	OR

Sixteen Logic Microoperations (continued)

Boolean Function	Microoperation	Name
$F_8 = (x + y)'$	$F \leftarrow \overline{A \vee B}$	NOR
$F_9 = (x \oplus y)'$	$F \leftarrow \overline{A \oplus B}$	Exclusive-NOR
$F_{10} = y'$	$F \leftarrow \overline{B}$	Complement B
$F_{11} = x + y'$	$F \leftarrow A \vee \overline{B}$	
$F_{12} = x'$	$F \leftarrow \overline{A}$	Complement A
$F_{13} = x' + y$	$F \leftarrow \overline{A} \vee B$	
$F_{14} = (xy)'$	$F \leftarrow \overline{A \wedge B}$	NAND
$F_{15} = 1$	$F \leftarrow$ all 1's	Set to all 1's

One Stage of Logic Circuit



S_1	S_0	Output	Operation
0	0	E = A ^ B	AND
0	1	E = A ∨ B	OR
1	0	E = A ⊕ B	XOR
1	1	E = A'	Complement

Logic Applications

- Logic Operations allow us to manipulate individual bits in ways that we could not do otherwise.
- These applications include:
  - selective set
  - selective complement
  - select clear
  - mask
  - insert
  - clear

Selective-Set

- Selective-set sets to 1 the bits in register A where there is a corresponding 1 in register B:

1010	Content of A before
1100	Content of B (logic operand)
1110	Content of A after

- This is done using the logical-OR operation.

Selective Complement

- Selective-complement complements the bits in register A where there is a corresponding 1 in register B:

1010	Content of A before
1100	Content of B (logic operand)
0110	Content of A after

- This is done using the exclusive-OR operation.

Selective Clear

- Selective-clear clears to 0 the bits in register A where there is a corresponding 1 in register B:

1010	Content of A before
1100	Content of B (logic operand)
0010	Content of A after

- This is done using the logical-AND operation and ***B'***.

Mask

- Mask clears to 0 the bits in register A where there is a corresponding 0 in register B:

  1010 Content of A before
  1100 Content of B (logic operand)
  1000 Content of A after

- This is done using the logical-AND operation and ***B***.

Insert

- Insert inserts a new value into a set of bits in register A.
- First we mask out the upper four bits (in our 8-bit value):

  0110 1010 Content of A before
  0000 1111 Content of B (logic operand)
  0000 1010 Content of A after

- In the second step, we insert the new values:

  0000 1010 Content of A before
  1001 0000 Content of B (logic operand)
  1001 1010 Content of A after

- The masking is done using an AND and the insertion is done with an OR.

Clear

- Clear compares A and B and produces all 0s if the numbers are equal. the bits in register A where there is a corresponding 0 in register B:

  1010 Content of A before

  <u>1010</u> Content of B (logic operand)

  0000 $A \leftarrow A \oplus B$

  If A& B are both 1 or both 0, this produces 0.This is done using the logical-AND operation and ***B***.

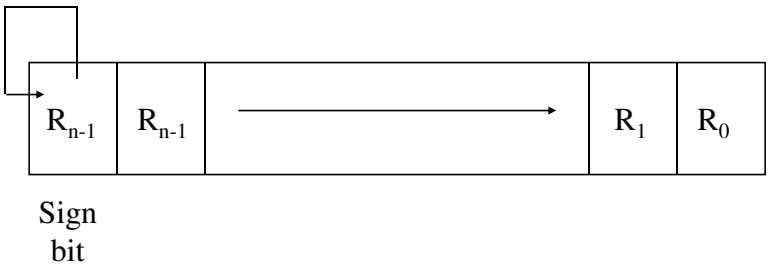
Shift Microoperations

- Shift microoperations are used for serial transfer of data and are used in conjunction with arithmetic and logic operations.
- The register contents can be shifted to the left or to the right.
- There are three types of shift operations:
  - Logical shifts transfers 0 through the serial input, with all the bits involved in the shifting.
  - Arithmetic shifts multiplies (or divides) a signed number by 2.
  - Circular shifts circulates the bits of the register around the two ends with no loss of information.

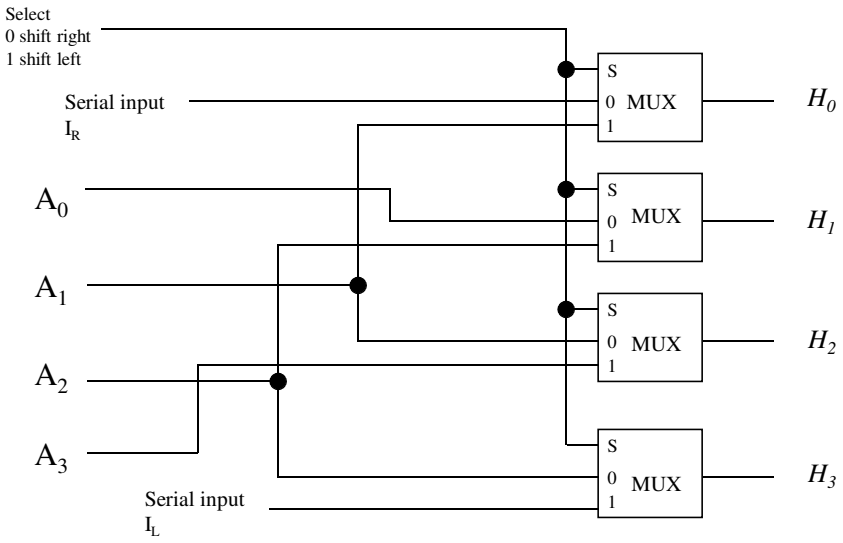
Shift Microoperations

Symbolic Designation	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 shift-right register R
R ← ashl R	Arithmetic Shift-left register R
R ← ashr R	Arithmetic Shift-right register R

Arithmetic Shift Right



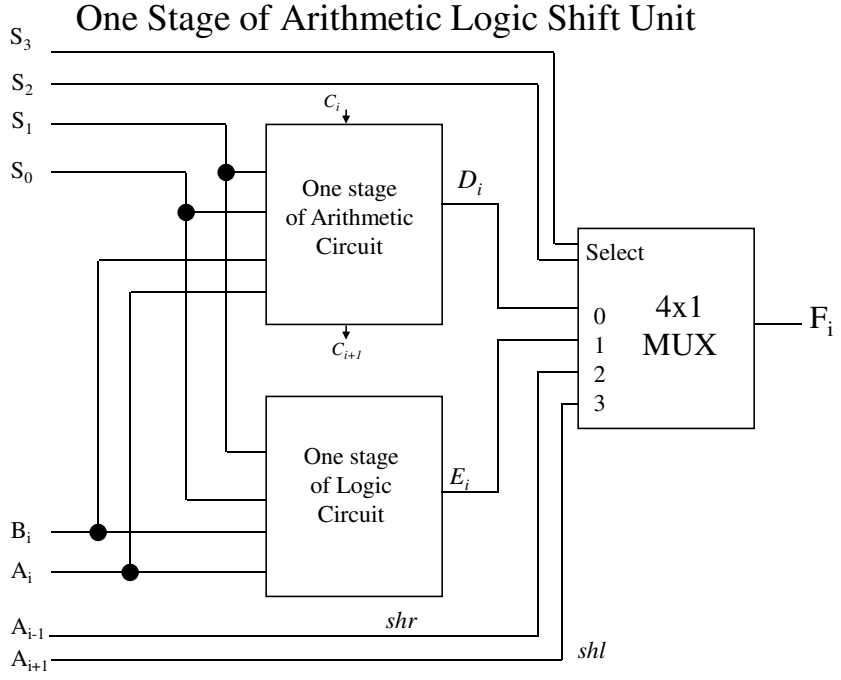
4-Bit Combinational Circuit Shifter



Arithmetic Logic Shift Unit

- Instead of having individual registers performing the various microoperations, computers use an Arithmetic Logic Unit which combine these functions.

One Stage of Arithmetic Logic Shift Unit



Function Table for 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

Function Table for Arithmetic Logic Shift Unit

Operation Select

S_3	S_2	S_1	S_0	C_{in}	Operation	Function
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