REGISTER TRANSFER LANGUAGE

REGISTER TRANSFER LANGUAGE

- The operations executed on the data stored in the registers are called **micro operations**.
- Classifications of micro operations
  - ✓ Register transfer micro operations
  - ✓ Arithmetic micro operations
  - ✓ Logic micro operations
  - ✓ Shift micro operations
- This is an elementary operation performed on the information stored in registers.
  Ex: shift, count, clear and load.
- The internal hardware organisation of a digital computer is best defined by specifying
  1) The set of registers it contains and their function.
  2) The sequence of micro operation performed on the binary information stored in the register.
  3) The control that initiates the sequence of micro operations.
- The term register transfer implies the availability of hardware logic circuits that can perform a stated micro operation and transfer the result to the same or the other register

Register transfer micro operations:

- Capital letters are used to designate registers
  Ex: MAR- memory address register (holds an address for memory unit)
  PC – Program Counter
  IR – Instruction Register
- Individual flip-flops are used in n-bit registers
- The symbolic representation of the register transfer is

  R2 ← R1
- The information is transferred from the register R1 (source) to register R2 (destination).
  Common ways of representing a register is shown in figure below:

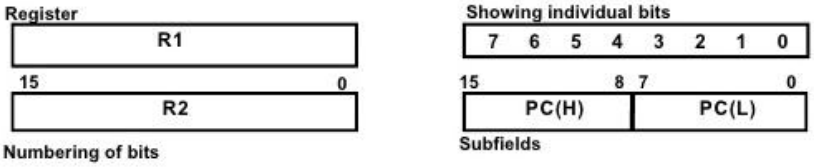


FIGURE: DESIGNATION OF REGISTERS

REGISTER TRANSFER LANGUAGE

Control function:

- Often actions need to only occur if a certain condition is true. This is similar to an "if" statement in a programming language. In digital systems, this is often done via a *control signal*, called a *control function*
  - If the signal is 1, the action takes place
- This is represented as: **P: R2 ← R1**

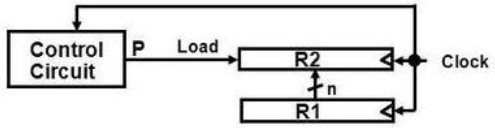
  Which means "if P = 1, then load the contents of register R1 into register R2", i.e., if (P = 1) then (R2 ← R1)

  Implementation of controlled transfer
- The same clock controls the circuits that generate the control function and the destination register
- Registers are assumed to use *positive-edge-triggered* flip-flops

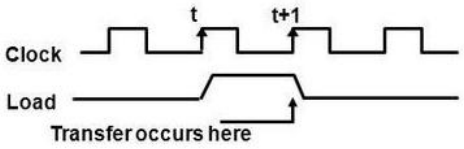
Implementation of controlled transfer

P: R2 ← R1

Block diagram



Timing diagram



- **The same clock controls the circuits that generate the control function and the destination register**
- **Registers are assumed to use *positive-edge-triggered* flip-flops**

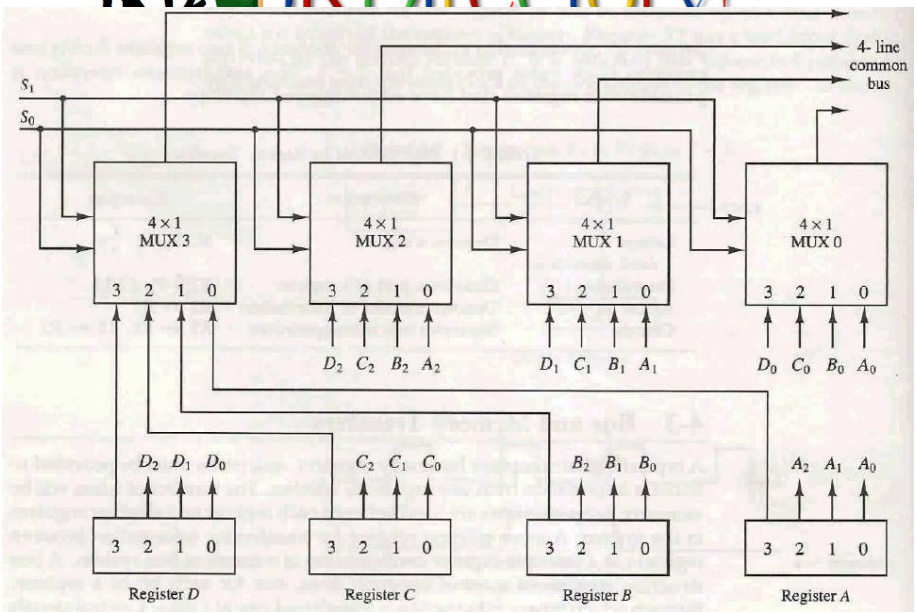
BASIC SYMBOLS FOR REGISTER TRANSFERS

Symbols	Description	Examples
Capital letters & 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

REGISTER TRANSFER LANGUAGE

BUS AND MEMORY TRANSFERS:

- Since transferring info from one register to another in separate lines becomes complex, a more efficient scheme called common bus system is used.
- A bus consists of a common set of lines, one for each bit of a register through which the binary info is transferred at a time.
- Control signals determine which register is selected by the bus during each particular register transfer.
- One way of constructing the common bus system is using multiplexers
- The bus consists of 4 registers and four 4 by 1 mux each having data inputs 0 through 3 and selection lines S_1 and S_0
  Ex: o/p of register A is connected to input 0 of mux1 because this i/p is labelled A1.
- When S_1S_0=00, the four bits of one register are selected and transfer it to four line common bus.
  Ex: S_1S_0=00, the data i/p's of all mux are selected and applied to bus which causes the bus line to receive content of reg A
- Generally a bus system multiplexes k registers with n bits to produce an n line common bus using n multiplexers
- The size of the mux should be k by 1 since it multiplexes k data lines
  Ex: for a common bus of eight registers of 16 registers should have 16 mux with 8 by 1 size with 3 selection lines



- The transfer of information from a bus into one of many destination registers can be accomplished by connecting the bus lines to the i/p's of the destination register and activating the load control of the particular destination register selected
- The content of reg C is placed on bus and the content of bus is loaded to R1 by activating its load control i/p.

REGISTER TRANSFER LANGUAGE

THREE STATE BUFFER:

- A three state gate is a digital circuit that exhibits threes states
  - ✓ Two of them being conventional logics 0 and 1
  - ✓ Third is high impedance state which behaves as an open circuit i.e., output disconnected
- They may perform any conventional logics such as AND or NAND
- Where as in three state buffer gate
  - ✓ Control i/p determine the o/p state
  - ✓ When control i/p is 1, the o/p is enabled and behaves as normal i/p
  - ✓ When control i/p is 0, the o/p acts as high impedance state
  - ✓ Because of this high impedance state a large number of 3 state gates o/ps can be connected to form a common bus line without loading effect

MEMORY TRANSFER:

- The transfer of information from a memory word to outside environment is called read operation.

  Read: DR ← M[AR]

  Transfer of information into DR from the memory word M selected by address in AR.
- The transfer of new information to be stored into the memory is called write memory.

  Write: M[AR] ←R1
- The particular memory among the many available is selected by the memory during the transfer

ARTHIMETIC MICRO OPERATIONS:

- The basic arithmetic micro operations are addition, subtraction, increment, decrement and shift
- The arithmetic micro operation is defined by the statement

  R3 ←R1+R2

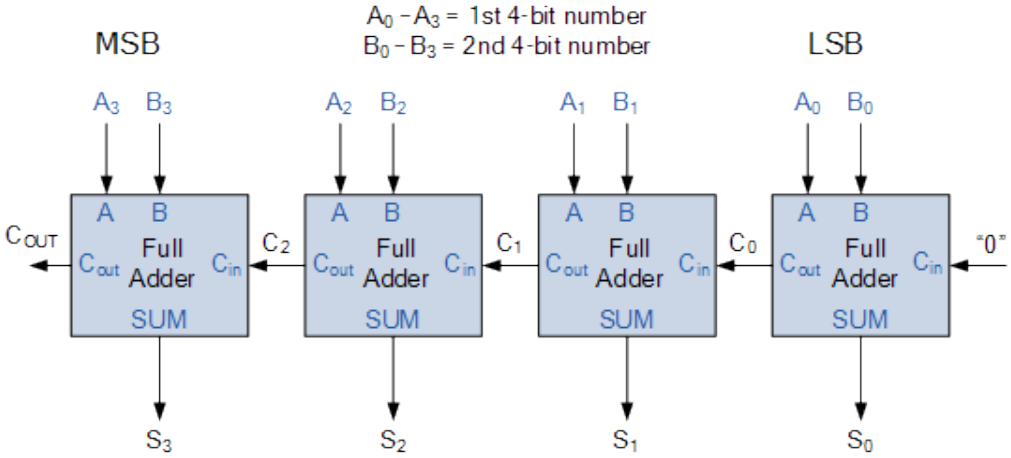
  It states that the contents of register 1 and register 2 are added and the sum is transfer to r3
- To implement this statement with hardware we need 3 registers and a digital component that performs addition operation
- To implement subtraction we use complements and addition
- The table shows different arithmetic micro operations

BINARY ADDER:

- To implement the add operation with hardware we need the registers to hold the data and the digital component that performs the arithmetic addition

REGISTER TRANSFER LANGUAGE

- The digital circuit that generates the arithmetic sum of two binary numbers of any length is caller binary adder, which can be constructed using full adder circuits in cascade
- The n bit binary adder needs n full adders



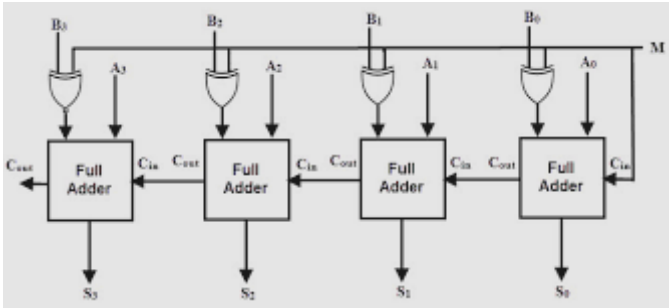
BINARY ADDER-SUBTRACTER:

- The addition and subtraction operations can be combined into one common circuit by including and XOR operation with each full adder
- When M=0 circuit acts an adder, we have

  B XOR 0=B. The full adder receives the value of B, the input carry is 0, and the circuit performs the A+B

- When M=1 circuit acts as subtractor

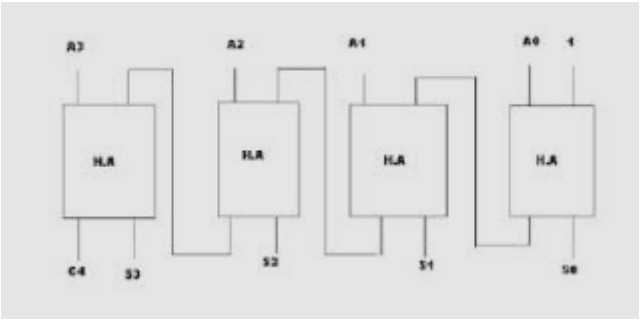
  B XOR 1= B'. The full adder performs A+B'+1



BINARY INCREMENTER:

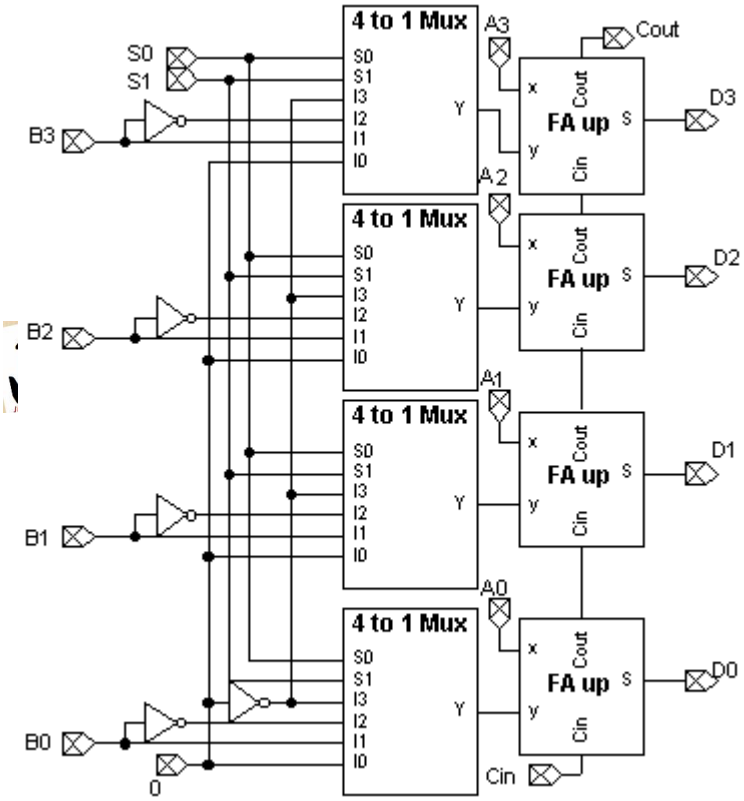
- The increment micro operation adds 1 to a number in a register

REGISTER TRANSFER LANGUAGE



ARITHMETIC CIRCUIT:

- The basic component of an arithmetic circuit is the parallel adder. By controlling the data i/ps to the adder, it is possible to obtain different types of arithmetic operations.



- The 4 bit arithmetic circuit consists of four full adders and four multiplexers choosing different operations.
- The 4 i/ps of A directly go to the X i/ps of full adder
- The each of the four B i/ps as well as the complements of B are connected to the 0 and 1 data i/ps of the multiplexers.
- The other two data i/ps (i.e., 2 and 3) of the mux are connected to logic 0 and 1 respectively.
- So from the selection lines S1 and S0, the 4 muxs are controlled.
- The i/p carry Cin goes to the carry i/p of the FA in least significant position. The other carries connected from one stage to the next.
    - The o/p of the binary adder is calculated from **D=A + Y + Cin**

REGISTER TRANSFER LANGUAGE

The 8 possible arithmetic operations are as shown below:

Select			In	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 w/carry
0	1	0	B	$D = A + \overline{B}$	Subtract w/borrow
0	1	1	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

LOGIC MICRO OPERATIONS:

- Logic micro operations specify binary operations for strings of bits stored in registers. For example P: R1← R1 XOR R2

  1 0 1 0 content of R1
  1 1 0 0 content of R2
  0 1 1 1 content of R1 after P=1

LIST OF LOGIC MICROOPERATIONS

- **List of Logic Microoperations**
  - 16 different logic operations with 2 binary variables.
  - n binary variables $\rightarrow 2^{2^n}$ functions

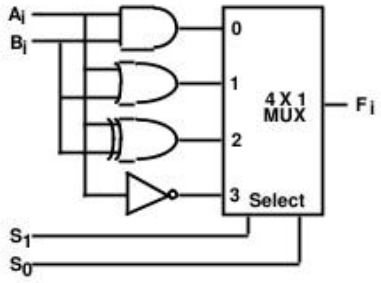
- **Truth tables for 16 functions of 2 variables and the corresponding 16 logic micro-operations**

x y	0 0 1 1 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

REGISTER TRANSFER LANGUAGE

SOME APPLICATIONS:

HARDWARE IMPLEMENTATION OF LOGIC MICROOPERATIONS



Function table

S_1	S_0	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

• Logic microoperations can be used to change bit values, delete a group of bits, or insert new bit values into a register.

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

1010 A before

<u>1100</u> B (logic operand)

1110 A after $A \leftarrow A \vee B$

• The selective-complement operation complements bits in A where there are corresponding 1's in B

1010 A before

<u>1100</u> B (logic operand)

0110 A after $A \leftarrow A \oplus B$

• The selective-clear operation clears to 0 the bits in A only where there are corresponding 1's in B

1010 A before

<u>1100</u> B (logic operand)

0010 A after $A \leftarrow A \wedge B$

• The mask operation is similar to the selective-clear operation, except that the bits of A are cleared only where there are corresponding 0's in B

1010 A before

1100 B (logic operand)

1000 A after A ← A ∧ B

• The insert operation inserts a new value into a group of bits

• This is done by first masking the bits to be replaced and then Oring them with the bits to be inserted

0110 1010 A before
0000 1111 B (mask)
0000 1010 A after masking

0000 1010 A before
1001 0000 B (insert)
1001 1010 A after insertion

• The clear operation compares the bits in A and B and produces an all 0's result if the two number are equal

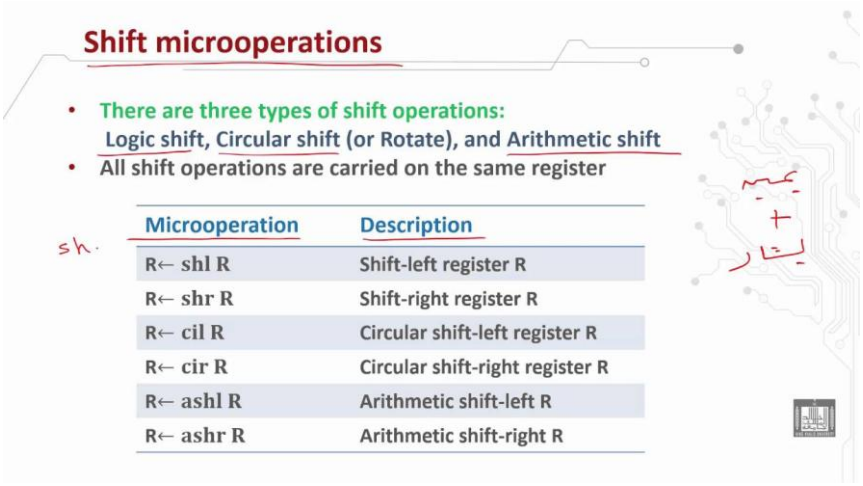
1010 A
1010 B
0000 A ← A ⊕ B

Shift Microoperations

Shift microoperations are used for serial transfer of data

They are also used in conjunction with arithmetic, logic, and other dataprocessing operations

There are three types of shifts: logical, circular, and arithmetic



A **logical shift** is one that transfers 0 through the serial input. The symbols shl and shr are for logical shift-left and shift-right by one position

$$R1 \leftarrow shl\ R1$$

The **circular shift** (rotate) circulates the bits of the register around the two ends without loss of information. The symbols cil and cir are for circular shift left and right 14. The **arithmetic shift** shifts a signed binary number to the left or right. To the left is multiplying by 2, to the right is dividing by 2.

Arithmetic shifts must leave the sign bit unchanged. A sign reversal occurs if the bit in Rn-1 changes in value after the shift. This happens if the multiplication causes an overflow. An overflow flip-flop Vs can be used to detect the overflow

$$Vs = Rn\text{-}1 \oplus Rn\text{-}2$$

A bi-directional shift unit with parallel load could be used to implement this two clock pulses are necessary with this configuration: one to load the value and another to shift. In a processor unit with many registers it is more efficient to implement the shift operation with a combinational circuit.

The content of a register to be shifted is first placed onto a common bus and the output is connected to the combinational shifter, the shifted number is then loaded back into the register. This can be constructed with multiplexers.

Arithmetic Logic Shift Unit

• The arithmetic logic unit (ALU) is a common operational unit connected to a number of storage registers

REGISTER TRANSFER LANGUAGE

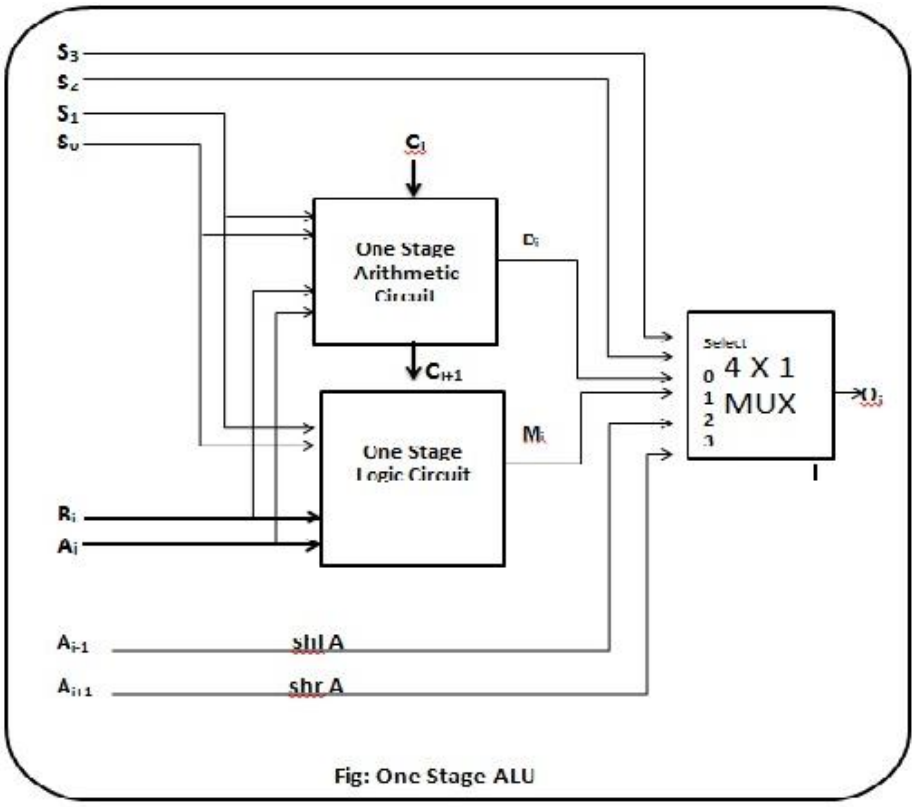


Fig: One Stage ALU

• To perform a micro operation, the contents of specified registers are placed in the inputs of the ALU

• The ALU performs an operation and the result is then transferred to a destination register

• The ALU is a combinational circuit so that the entire register transfer operation from the source registers through the ALU and into the destination register can be performed during one clock pulse period.

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	×	×	×	$F =$ shr A	Shift right A into F
1	1	×	×	×	$F =$ shl A	Shift left A into F