Chapter 2
Register Transfer and Microoperations

2.1 Register and Register Transfer Language (RTL)

Register

→ Register is the storage device, inside CPU, of data on which microoperations are performed.

→ The operations executed on data stored in registers are called microoperations. A microoperation is an elementary operation performed on the information stored in one or more registers. The result of the operation may replace the previous binary information of a register or may be transferred to another register. Examples of microoperations are shift, count, clear and load.

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

→ The language, which is basically used to express the transfer of data among the registers, is called ***Register Transfer Language*** **(RTL).** It is the symbolic notation used to describe the microoperation transfers among registers. In such transfer, one of the source or destination should be register (not necessarily both).

Register Transfer

→ Computer registers are designated by capital letters.

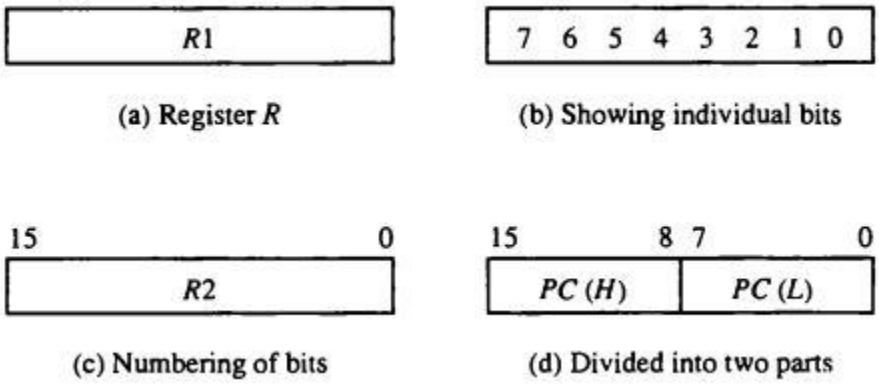
→ For example, the register that holds an address for the memory unit is usually called a memory address register and is designated by the name MAR. Other designations for registers are PC (for program counter), IR (for instruction register, and R1 (for processor register).

→ The most common way to represent a register is by a rectangular box with the name of the register inside. Fig (a).

→ The individual flip-flops in an n-bit register are numbered in sequence from 0 through n-1, starting from 0 in the rightmost position and increasing the numbers toward the left. For e.g. 8-bit register numbered: Fig (b).

→ The numbering of bits in a 8-bit register can be marked on top of the box. Fig (c).

→ A 16-bit register is partitioned into two parts in (d). Bits 0 through 7 are assigned the symbol L (for low byte) and bits 8 through 15 are assigned the symbol H (for high byte). The name of the 16 bit register is PC. The symbol PC(0—7) or PC(L) refers to the low order byte and PC(8—15) or PC(H) to the high order byte.



→ Information transfer from one register to another is designated in symbolic form by means of a replacement operator.

$$R2 \leftarrow R1$$

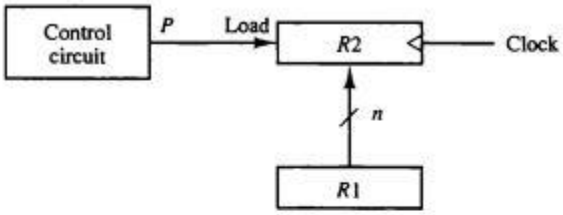
→ If there is predetermined control condition like

$$\text{If } (P=1) \text{ then } (R2 \leftarrow R1)$$

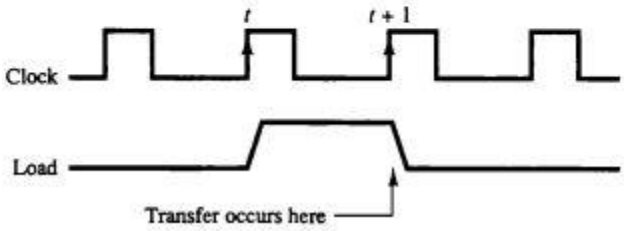
then we can write the statement as

P: R2 ← R1

where P is control signal usually a control function which is Boolean variable that is equal to 1 or 0.



(a) Block diagram



(b) Timing diagram

→ The n outputs of register R1 are connected to the n inputs of register R2. The letter n will be used to indicate any number of bits for the register.

→ It is assumed that all transfers occur during a clock edge transition. Even though the control condition such as P becomes active just after time t, the actual transfer does not occur until the register is triggered by the next positive transition of the clock at time t + 1.

→ A comma is used to separate two or more operations that are executed at the same time. The statement

T: R2 ← R1, R1 ← R2

denotes an operation that exchanges the contents of two registers during one common clock pulse provided that T = 1.

Symbol	Description	Examples
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*
Comma ,	Separates two microoperations	*R2* ← *R1*, *R1* ← *R2*

Fig: Basic Symbols for Register Transfers

→ For example, RTL of fetch cycle can be written as:

T1: MAR←PC
T2: MBR←[MAR]
T3: IR←MBR
T4: unspecified; PC←PC+1

The notation (T1, T2, T3, T4) represents successive time units. All three units are of equal duration. A time unit is defined by regularly spaced clock pulses. The operations performed within this single unit of time are called microoperations. A single time unit can contain one or more microoperations. Since each microoperation can specifies the transfer of data into or out of a register, such type is called **RTL**.

2.2 Bus and Memory Transfer

Bus

→ A typical digital computer has many registers, and paths must be provided to transfer information from one register to another. The number of wires will be excessive if separate lines are used between each register and all other registers in the system.

→ A more efficient scheme for transferring information between registers in a multiple register configuration is a common bus system.

→ A bus structure consists of a set of common lines, one for each bit of a register, through which binary information is transferred one at a time. Control signals determine which register is selected by the bus during each particular register transfer.

→ For example,

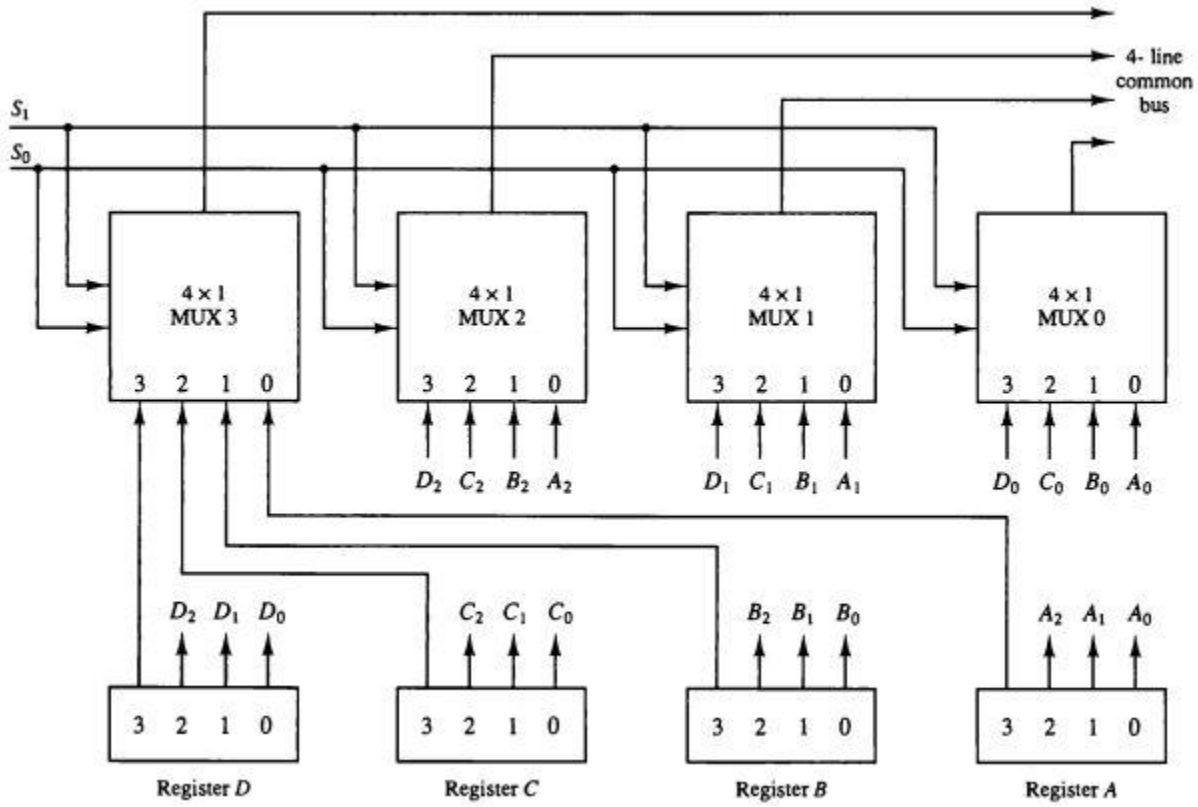


Fig: Bus system for four registers.

S_1	S_0	Register selected
0	0	*A*
0	1	*B*
1	0	*C*
1	1	*D*

Fig: Function table for Bus

→ In general, a bus system will multiplex k registers of n bits each to produce an n line common bus. The number of multiplexers needed to construct the bus is equal to n, the number of bits in each register. The size of each multiplexer must be k X 1 since it multiplexes k data lines.

→ For example, a common bus for eight registers of 16 bits each requires 16 multiplexers, one for each line in the bus. Each multiplexer must have eight data input lines and three selection lines to multiplex one significant bit in the eight registers.

→ The transfer of information from a bus into one of many destination registers can be accomplished by connecting the bus lines to the inputs of all destination registers and activating the load control of the particular destination register selected.

Memory Transfer

→ The transfer of information from a memory word to the outside environment is called a read operation. The transfer of new information to be stored into the memory is called a write operation.

→ A memory word will be symbolized by the letter M. The particular memory word among the many available is selected by the memory address during the transfer.

→ Consider a memory unit that receives the address from a register, called the address register, symbolized by AR. The data are transferred to another register, called the data register, symbolized by DR. The read operation can be stated as follows:

Read: DR ← M[AR]

` This causes a transfer of information into DR from the memory word M selected by the address in AR.

2.3 Microoperations

→ The operations on the data in registers are called microoperations.

→ Alternatively we can say that an elementary operation performed during one clock pulse on the information stored in one or more registers is called micro-operation.

→ The result of the operation may replace the previous binary information of the resister or may be transferred to another resister.

→ Register Transfer Language (RTL) can be used to describe the (sequence of) micro-operations.

→ The microoperations most often encountered in digital computers are classified into 4 categories:

i) Register transfer microoperations
ii) Arithmetic microoperations
iii) Logic microoperations
iv) Shift microoperations

Register transfer microoperations

Registers are designated by capital letters, sometimes followed by numbers (e.g., A, R13, IR). Often the names indicate function:

MAR	memory address register
PC	program counter
IR	instruction register

Information transfer from one register to another is described in symbolic form by replacement operator. The statement "R2 ←R1"

denotes a transfer of the content of the R1 into resister R2.

Control Function

Often actions need to only occur if a certain condition is true. In digital systems, this is often done via a control signal, called a control function.

Example: P: R2 ← R1 i.e. if (P = 1) then (R2← R1)

Which means "if P = 1, then load the contents of register R1 into register R2".

If two or more operations are to occur simultaneously, they are separated with commas.

Example: P: R3 ← R5, MAR ← IR

Arithmetic microoperations

The basic arithmetic microoperations are

– Addition

- Subtraction
- Increment
- Decrement

The additional arithmetic microoperations are

- Add with carry
- Subtract with borrow
- Transfer/Load

Summary of typical arithmetic microoperations

Symbolic Designation	**Description**
R3 ← R1 + R2	Contents of R1 plus R2 transferred to R3
R3 ← R1 + R2' + 1	Contents of R1 minus R2 transferred to R3
R1 ← R1 + 1	Increment the contents of R1 by one
R1 ← R1 - 1	Decrement the contents of R1 by one
R3 ← R1 + R2 + 1	Add with carry
R3 ← R1 + R2'	Subtract with borrow
R1 ← R1' +1	2's complement the contents of R1 (negate)

Logic microoperations

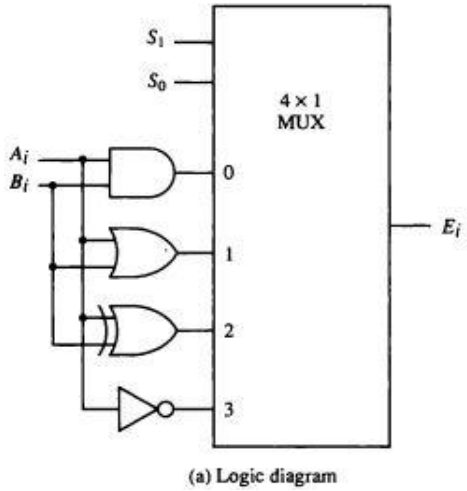
Logic microoperations are bit-wise operations, i.e., they work on the individual bits of data. Useful for bit manipulations on binary data and for making logical decisions based on the bit value. There are, in principle, 16 different logic functions that can be defined over two binary input variables. However, most systems only implement four of these

- AND (/\), OR (\/), XOR (⊕), Complement/NOT

The others can be created from combination of these four functions.

Microoperation	**Name**
F ← R1 /\ R2	AND
F ← R1 \/ R2	OR
F ← R1 ⊕ R2	XOR
F ← R1'	Complement (NOT)

Hardware Implementation



(a) Logic diagram

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

(b) Function table

Shift microoperations

The operation that changes the adjacent bit position of the binary values stored in the register is known as shift microoperation. They are used for serial transfer of data. The shift microoperations are classified into 3 types:

i) **Logical shift**: A logical shift transfer 0 through the serial input. It can be defined in RTL by:

R←shl R shift-left register R

R←shr R shift-right register R

ii) **Circular shift**: A circular shift rotates the bit from one end of the register to another end of the register. It can be defined in RTL by:

R←cil R circular shift-left register R

R←cir R circular shift-right register R

iii) **Arithmetic shift**: It shifts signed-binary number left or right. For shift left the content of the register is multiplied by 2 whereas For shift right the content of the register is divided by 2. The arithmetic shift must leave the sign bit unchanged. It can be defined in RTL by:

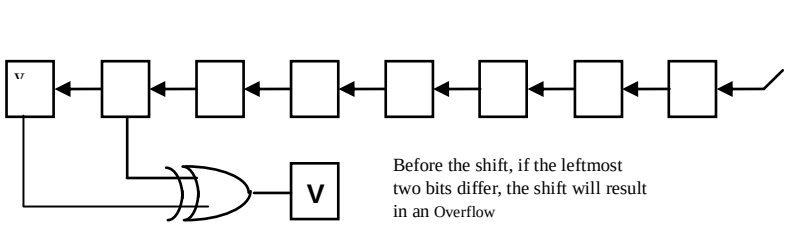
R←ashl R arithmetic shift-left register R

R←ashr R arithmetic shift-right register R

Overflow case during arithmetic shift-left:

If a bit in R_{n-1} changes in value after the shift, sign reversal occurs in the result. This happens if the multiplication by 2 causes an overflow.

Thus, left arithmetic shift operation must be checked for the overflow: an overflow occurs after an arithmetic shift-left if before shift $R_{n-1} \neq R_{n-2}$.



An overflow flip-flop V can be used to detect an arithmetic shift-left overflow.

$$V = R_{n-1} \oplus R_{n-2}$$

If V = 0, there is no overflow but if V = 1, overflow is detected.

2.4 Arithmetic Logic Shift Unit

Arithmetic logic shift unit is a digital circuit that performs arithmetic calculations, logical manipulation and shift operation. It is often abbreviated as ALU. The above figure shows the one stage of arithmetic logic shift unit.

The block diagram of ALU includes one stage of arithmetic circuit, one stage of logic circuit and one 4*1 multiplexer. The subscript i designates a typical stage.

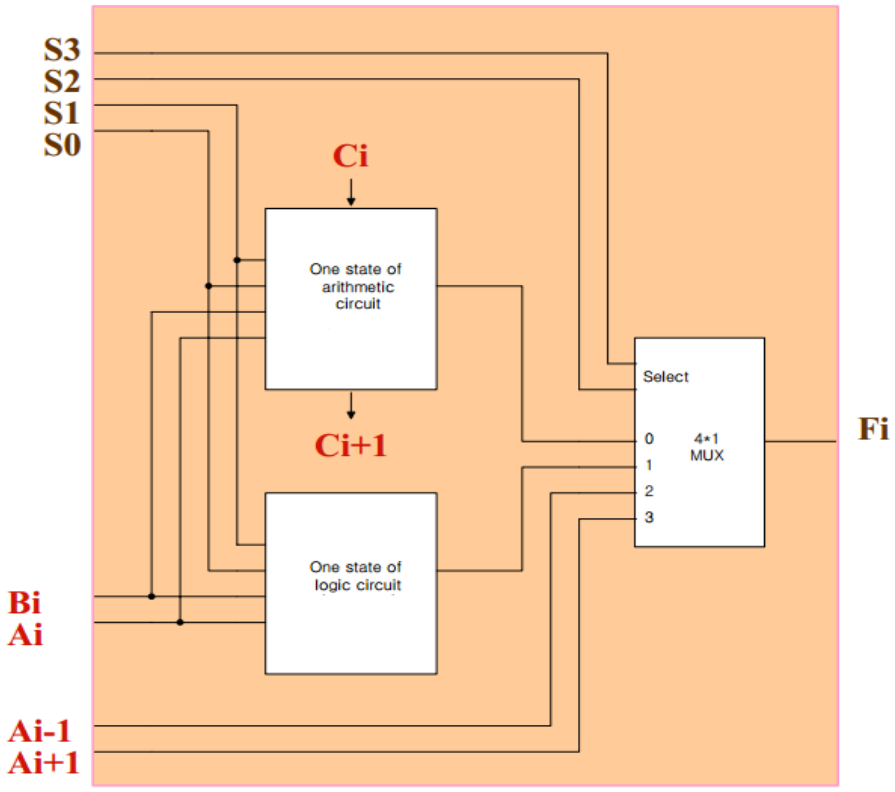


Fig: one stage of arithmetic logic shift unit

Inputs A_i and B_i are applied to both the arithmetic and logic units. A particular microoperation is selected with inputs S_1 and S_0. A 4*1 MUX selects the final output. The two inputs of the MUX are received from the output of the arithmetic circuit and logic circuit. The other two is A_{i-1} for the shift-right operation and A_{i+1} for the shift left operation. The circuit is repeated n times for n-bit ALU. The output carry C_{i+1} is connected to the input carry C_{in}. In every stage the circuit specifies 8 arithmetic operations, 4 logical operations and 2 shift operations, where each operation is selected by the five variables S_3, S_2, S_1, S_0 and C_{in}.

The operations of ALU can be summarized in table below:

Operation select					Operation	Function
S_3	S_2	S_1	S_0	C_{in}		
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	Transfer A
0	1	0	0	╳	F=A∧B	AND
0	1	0	1	╳	F=A∨B	OR
0	1	1	0	╳	F=A⊕B	XOR
0	1	1	1	╳	F=A’	Complement A
1	0	╳	╳	╳	F=shr A	Shift right A into F
1	1	╳	╳	╳	F= shl A	Shift left A into F