Register Transfer and Microoperations

(Reference: Computer System Architecture - M. Morris Mano, 3rd Ed.)

Before we start the process of **register** transfer and **microoperation** we need to understand why such kind of thing must be introduced?

So to answer this, we know the we have varied components in our computer system that are grouped together to execute the user instructions, so in order to make the varied components work in uniform way we must have a uniform pattern. Our all functional components are at basic made up of registers. **Registers are the basic unit of operation of various digital modules present in our digital systems**.

Registers store the data of the **digital modules**. Registers are also to be noted are the fastest way of accessing data in comparison to any other media like magnetic storage or optical storage. We can say by now that the **operations performed on the data stored in registers are termed as microoperations.** Examples are SHIFT, COUNT, CLEAR, LOAD.

The data that is being manipulated or operated in **registers** is now on the verge to be transferred or stored in some place. So **the transfer of data to the same register or any other register** (any other hardware logic) after being **operated is termed as register transfer.**

Briefly, we can say **register transfer language** (**RTL)** is a language to write the **microoperations** in sequential manner and the way processor will execute them.

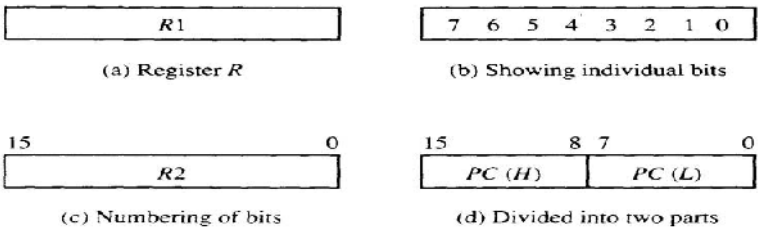
Microoperations are the basic operations that can be performed by a system on data stored in registers. Each microoperation describes a simple operation performed on data in one or more registers.

- Each assembly language instruction is made up of a sequence of microoperations
- Each microoperation defines a single register transfer or register-level operation
- A Register Transfer Language (**RTL**) is used to specify a microoperation

Register Transfer

- Designate computer registers by capital letters to denote its function
- The register that holds an address for the memory unit is called Memory Address Register (**MAR**)
- A register that contains data read from memory or data to be written to memory is called Memory Buffer Register (**MBR**)
- The program counter register is called **PC**
- **IR** is the instruction register and R1 is a processor register
- The individual flip-flops in an *n*-bit register are numbered in sequence from 0 to n-1
- Refer to the Figure below for the different representations of a register

Figure 4-1 Block diagram of register.



- Designate information transfer from one register to another by R2 ← R1
- This statement implies that the hardware is available
- The outputs of the source must have a path to the inputs of the destination
- The destination register has a parallel load capability
- A control function is a Boolean condition upon which a transfer depends. For example: if (P == 1) then R2 ← R1 The same conditional transfer can be written as: P: R2 ← R1
  The control function is P (P==0 or 1) and the microoperation is R2 ← R1. Every register transfer consists of a control function and a microoperation.
- Every statement written in register transfer notation implies the presence of the required hardware construction
- It is assumed that all transfers occur during a clock edge transition
- All microoperations written on a single line are to be executed at the same time

  T: R2 ← R1, R1 ← R2

Table 4-1 (Basic symbols for register transfer)

Symbol	Description	Example
Letters	Register name	MAR, R2, PC
Parentheses	Part of a register	PC(0-7), R2(L)
Arrow	Transfer	R2 ← R1, PC ← PC + 1
Comma	SIMULTANEOUS microoperations	R2 ← R1, PC ← PC + 1

Register Transfer Language (RTL) syntax

- Bits are numbered from the rightmost bit 0 (least significant) to leftmost bit n-1 (most significant)
- R1(0 - 3) denotes bits $R1_3$, $R1_2$, $R1_1$, and $R1_0$
- R1 ← M[AR] denotes that register R1 gets the data from memory “pointed to” by the contents of the address register.

Bus and Memory Transfers

BUS

If a computer has 16 registers, each holding **32 bits**, hence the number of wires that are needed to connect every register to every other are $\mathbf{16^2}$. For a large number of wide registers, the wires could end up taking most of the space in the circuit.

Remember, a *BUS* is a single shared set of wires connecting all registers.

BUS with Multiplexers

In Figure 4-3 we have a bus system for 4 registers using multiplexers. Only one register's contents can be on the bus for a given clock cycle.

For example, when $S_0 = S_1 = 0$ which register will be selected by the bus?
Ans.:

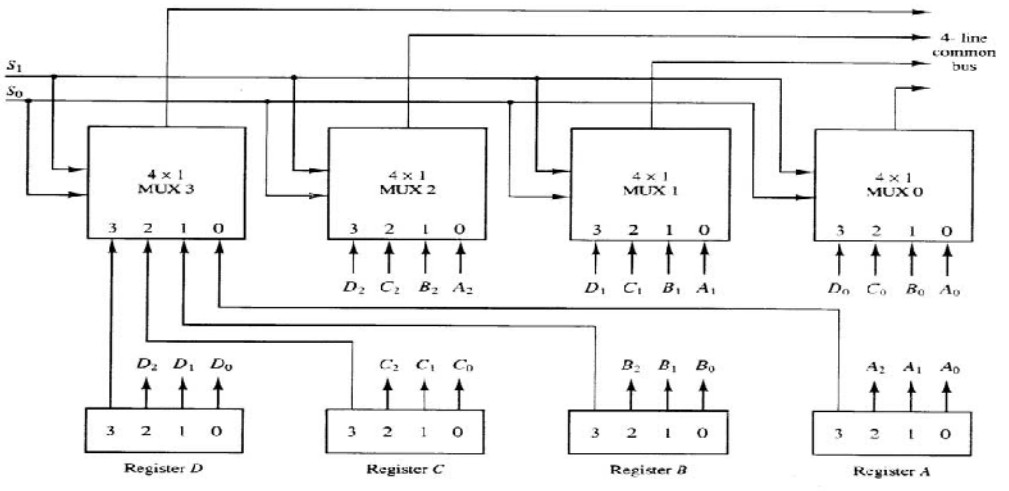
When $S_0 = S_1 = 0 \rightarrow$ Output of the 4 multiplexers:

$\text{MAX } 3 \rightarrow A_3$, $\text{MAX } 2 \rightarrow A_2$, $\text{MAX } 1 \rightarrow A_1$, $\text{MAX } 0 \rightarrow A_0$

Hence, register **A** will be selected and its content will be on the 4-line common bus.

- Rather than connecting wires between all registers, a common bus is used
- A bus structure consists of a set of common lines, one for each bit of a register
- Control signals determine which register is selected by the bus during each transfer
- Multiplexers can be used to construct a common bus
- Multiplexers select the source register whose binary information is then placed on the bus
- The select lines are connected to the selection inputs of the multiplexers and choose the bits of one register

Figure 4-3 Bus system for four registers.



- In general, a bus system will multiplex *k* registers of *n* bits each to produce an *n*- line common bus
- This requires *n* multiplexers – one for each bit
- The size of each multiplexer must be *k* x 1
- The number of select lines required is *log k*
- To transfer information from the bus to a register, the bus lines are connected to the inputs of all destination registers and the corresponding load control line must be activated

Memory Transfer

Remember, the internal bus connects only registers within the CPU, so how do we get data to and from memory?

The address register (AR) is used to select a memory address, and the data register (DR) is used to send and receive data. Both these registers are connected to the internal bus. DR is a bridge between the internal BUS and the memory data BUS.

Memory can also be connected directly to the internal BUS in theory.

Hence, accessing memory outside the CPU requires at least two clock cycles.
First we load AR with the desired memory address, and then transfer to or from DR.

Following operation might be used to show connections to memory unit:

M[AR] ← DR
DR ← M[AR]

In most typical computer systems, memory transfers take many clock cycles, known as *wait states*.

- 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.
- It is necessary to specify the address of M when writing memory transfer operations. This will be done by enclosing the address in square brackets followed by the letter M.

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.

The write operation transfers the content of a data register to a memory word M selected by the address. Assume that the input data are in register R1, and the address where the input data should be stored in memory is available in AR.

The write operation can be stated as follow:
Write: M[AR] ← R1

Microoperations

Microoperations are classified into the following four categories:

1. Register transfer i.e. the transfer of binary information from one register to another.
2. Arithmetic micro operation i.e. operation on numeric data.
3. Logic micro operation i.e. bit manipulation
4. Shift operation i.e. shift of bits on stored data.

Arithmetic Microoperations

- There are four categories of the most common microoperations:
  - Register transfer: transfer binary information from one register to another
  - Arithmetic: perform arithmetic operations on numeric data stored in registers
  - Logic: perform bit manipulation operations on non-numeric data stored in registers
  - Shift: perform shift operations on data stored in registers
- The basic arithmetic microoperations are addition, subtraction, increment, decrement, and shift
- Example of addition: R3 ← R1 +R2
- Subtraction is most often implemented through complementation and addition
- Example of subtraction: R3 ← R1 + $\overline{R2}$ + 1
  - Adding 1 to the 1's complement produces the 2's complement
  - Adding the contents of R1 to the 2's complement of R2 is equivalent to subtracting
- Multiply and divide are not included as microoperations
- A microoperation is one that can be executed by one clock pulse
- Multiply (divide) is implemented by a sequence of add and shift microoperations (or subtract and shift)
- To implement the add microoperation with hardware, we need the registers that hold the data and the digital component that performs the addition
- A full-adder adds two bits and a previous carry
- A *binary adder* is a digital circuit that generates the arithmetic sum of two binary numbers of any length
- A binary added is constructed with full-adder circuits connected in cascade
- An *n*-bit binary adder requires *n* full-adders

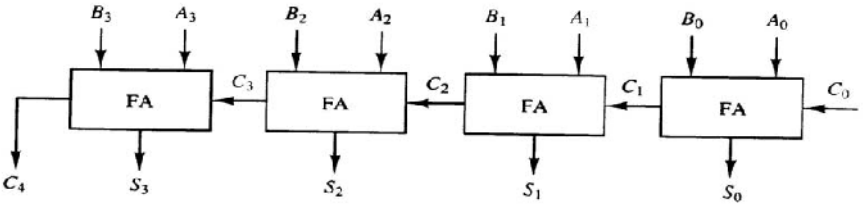


Figure 4-6 4-bit binary adder.

- The subtraction A-B can be carried out by the following steps
  - Take the 1's complement of B (invert each bit)
  - Get the 2's complement by adding 1
  - Add the result to A

- The addition and subtraction operations can be combined into one common circuit by including an XOR gate with each full-adder

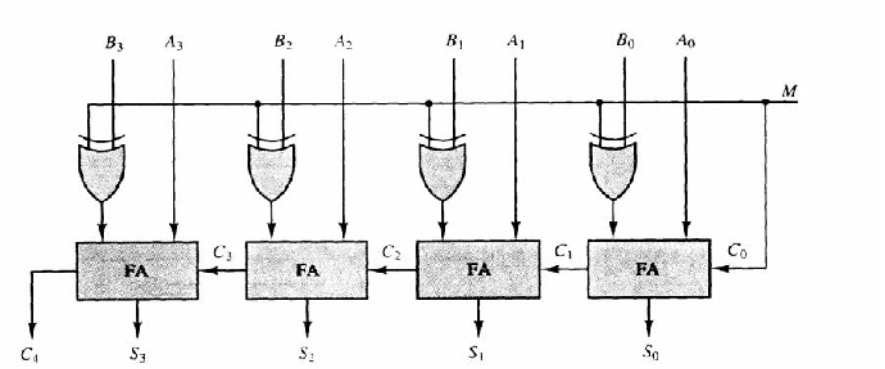


Figure 4-7 4-bit adder-subtractor.

Arithmetic Circuits

The arithmetic microoperations listed in the table below can be implemented in one arithmetic circuit.

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 the contents of $R2$ (1's complement)
$R2 \leftarrow \overline{R2} + 1$	2's complement the contents of $R2$ (negate)
$R3 \leftarrow R1 + \overline{R2} + 1$	$R1$ plus the 2's complement of $R2$ (subtraction)
$R1 \leftarrow R1 + 1$	Increment the contents of $R1$ by one
$R1 \leftarrow R1 - 1$	Decrement the contents of $R1$ by one

The above table may be simplified as follows

Example	Description
R3 ← R1 + R2	Addition
R3 ← R1 - R2 (R1 + R2' + 1)	Subtraction
R2 ← R2’	Complement (really a logic operation)
R2 ← -R2 (R2' + 1)	Negation
R1 ← R1 + 1	Increment
R1 ← R1 - 1	Decrement

Increment and decrement can be done with combinational incrementers and decrementers, counter registers, or by adding a 1.

Multiply and divide are not often implemented as microoperations due to the amount of time they require. They are usually implemented as a multi-clock-cycle routine of shifts and adds.

The basic component of the arithmetic circuit is the parallel adder. By controlling the data inputs to the adder, it is possible to obtain different types of arithmetic operations.

Logic Microoperations

Table 4-6 below shows all sixteen different logic microoperations for 16 functions of 2 variables.

Logic operations are performed independently on all bits of a word.

```
  10100010
^ 01001010
  ----------
  00000010
```

As a result, the logic circuit is simply a group of gates working independently on each bit. To implement a single circuit with many logic functions, simply use a multiplexer.

Logic operations are used heavily in systems programming (device drivers, etc.), cryptography, etc.

- Selective set
- Selective complement
- Selective clear

Logic operations specify binary operations for strings of bits stored in registers and treat each bit separately

- Example: the XOR of R1 and R2 is symbolized by

  P: $R1 \leftarrow R1 \oplus R2$

- Example: R1 = 1010 and R2 = 1100

  1010 Content of R1
  1100 Content of R2
  0110 Content of R1 after P = 1

- Symbols used for logical microoperations:

  OR: $\vee$
  AND: $\wedge$
  XOR: $\oplus$

- The + sign has two different meanings: logical OR and summation
- When + is in a microoperation, then summation
- When + is in a control function, then OR
- Example:

  P + Q: $R1 \leftarrow R2 + R3$, $R4 \leftarrow R5 \vee R6$

- There are 16 different logic operations that can be performed with two binary Variables

Table 4-5 below shows the truth for the 16 functions of two variables

TABLE 4-5: The truth for the 16 functions of two variables

x	y	F_0	F_1	F_2	F_3	F_4	F_5	F_6	F_7	F_8	F_9	F_{10}	F_{11}	F_{12}	F_{13}	F_{14}	F_{15}
0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1
0	1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
1	0	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1

Table 4-6 below shows all sixteen different logic microoperations.

TABLE 4-6 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
$F_8 = (x + y)'$	$F \leftarrow \overline{A \vee B}$	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

- The hardware implementation of logic microoperations requires that logic gates be inserted for each bit or pair of bits in the registers
- All 16 microoperations can be derived from using four logic gates (AND, OR, XOR and Complement).

Shift Microoperations

- Shift microoperations are used for serial transfer of data
- They are also used in conjunction with arithmetic, logic, and other data-processing 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 ← shl R1
- The *circular shift* (aka 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

TABLE 4-7 Shift Microoperations

Symbolic designation	Description
$R \leftarrow \text{shl } R$	Shift-left register R
$R \leftarrow \text{shr } R$	Shift-right register R
$R \leftarrow \text{cil } R$	Circular shift-left register R
$R \leftarrow \text{cir } R$	Circular shift-right register R
$R \leftarrow \text{ashl } R$	Arithmetic shift-left R
$R \leftarrow \text{ashr } R$	Arithmetic shift-right R

- 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 R_{n-1} changes in value after the shift
- This happens if the multiplication causes an overflow
- An overflow flip-flop V_s can be used to detect the overflow

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

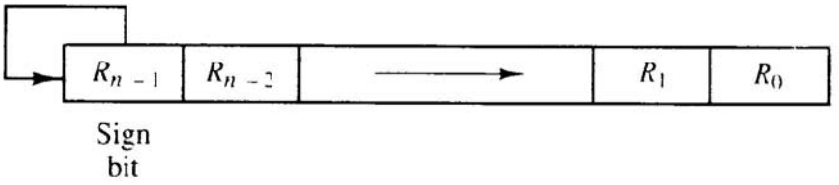


Figure 4-11 Arithmetic shift right.

- 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
- To perform a microoperation, 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