Register Transfer and Micro Operations

Register Transfer and Register Transfer Language

The term **register transfer** means performing a stated micro-operation using registers and transfer the result of the operation to the same or another register.

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.

Describing the Register Transfers in words do not give sense. So, Microoperations are expressed in terms of a Register Transfer Language (RTL). The symbolic notation used to describe the micro-operation transfers amongst registers is called Register transfer language.
In RTL, Information transferred from one register to another (Register Transfer) is designated in symbolic form by means of replacement operator.

Example: R2 ← R1

It denotes the transfer of the data from register R1 into R2.

Micro-Operations

The operations executed on data stored in registers are called micro-operations. A micro-operation is an elementary operation performed on the information stored in one or more registers.
The micro-operations in digital computers are of 4 types:

1. Register transfer micro-operations transfer binary information from one register to another.
   Example: R2 ← R1
2. **Arithmetic Micro-operations**
   In general, the Arithmetic Micro-operations deals with the operations performed on numeric data stored in the registers. The basic Arithmetic Micro-operations are classified in the following categories:
   1. Addition
   2. Subtraction
   3. Increment
   4. Decrement
   5. Shift

   Some additional Arithmetic Micro-operations are classified as:

   1. Add with carry
   2. Subtract with borrow
   3. Transfer/Load, etc.

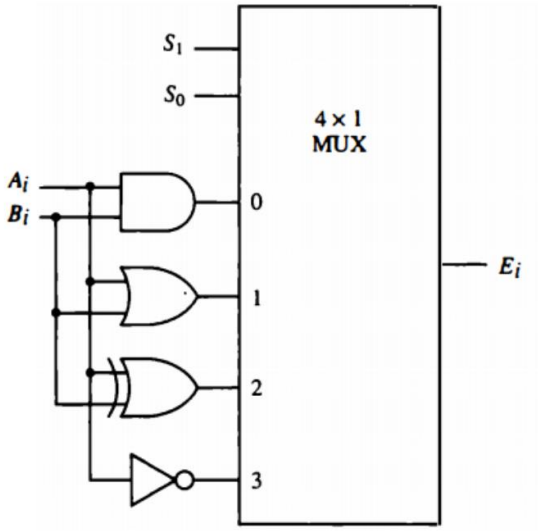
The following table shows the symbolic representation of various Arithmetic Micro-operations.

Symbolic Representation	**Description**
R3 ← R1 + R2	The contents of R1 plus R2 are transferred to R3.
R3 ← R1 - R2	The contents of R1 minus R2 are transferred to R3.
R2 ← R2'	Complement the contents of R2 (1's complement)
R2 ← R2' + 1	2's complement the contents of R2 (negate)
R3 ← R1 + R2' + 1	R1 plus the 2's complement of R2 (subtraction)
R1 ← R1 + 1	Increment the contents of R1 by one
R1 ← R1 - 1	Decrement the contents of R1 by one

3. Logical Micro-operations

They perform bit manipulation operation on non-numeric data stored in registers. There are 16 different logic functions that can be defined over two binary input variables. However, most of the systems only implement four of these. They are

Figure One stage of logic circuit.



(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

a. AND
b. OR
c. XOR
d. COMPLEMENT

Applications of Logic Microoperations

Logic microoperations can be used to manipulate individual bits or a portion of a word in a register.

- Consider the data in a register A. In another register, B, is bit data that will be used to modify the contents of A

  Selective-set operation:
  Sets to 1 the bits in A where there are corresponding 1's in B .

  1010 A before
  1100 B (logic operand)
  1110 A after $A \leftarrow A \vee B$

  Selective-complement:
  Complements bits in A where there are corresponding 1's in B.
  1010 A before
  1100 B (logic operand)
  0110 A after $A \leftarrow A \oplus B$

  Selective-clear operation:
  Clears to 0 the bits in A only where there are corresponding 1's in B.
  1010 A before
  1100 B (logic operand)
  0010 A after clear operation. $A \leftarrow A \wedge B'$

  Mask operation:
  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 $A \leftarrow A \wedge B$

  Insert operation:
  Inserts a new value into a group of bits.
  Step 1: mask the bits to be replaced
  Step 2: OR them with the bits to be inserted.

EX: INSERT 1001 IN THE LEFT HALF OF THE WORD "A"

Step 1:
0110 1010 A before
0000 1111 B (mask) (AND operation)
0000 1010 A after masking [A ←]

Step 2:
0000 1010 A before
1001 0000 B (insert) (OR operation)
1001 1010 A after insertion[A ← A ∨ B]
*A complete insert operation goes through micro operation: A ← (A ∧ B) ∨ C

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

4. **Shift microoperations**
   These are used for serial transfer of data. That means we can shift the contents of the register to the left or right. In the shift left operation the serial input transfers a bit to the right most position and in shift right operation the serial input transfers a bit to the left most position.

   a. **Logical Shift**
      It transfers 0 through the serial input. The symbol **"shl"** is used for logical shift left and **"shr"** is used for logical shift right.

   b. **Circular Shift**
      This circulates or rotates the bits of register around the two ends without any loss of data or contents. In this, the serial output of the shift register is connected to its serial input. **"cil"** and **"cir"** is used for circular shift left and right respectively.

   c. **Arithmetic Shift**
      This shifts a signed binary number to left or right. An **arithmetic shift left** multiplies a signed binary number by 2 and **shift right** divides the number by 2. Arithmetic shift micro-operation leaves the sign bit unchanged because the signed number remains same when it is multiplied or divided by 2.
      *In Arithmetic left shift, overflow is said to be occurred if the sign bit and MSB of magnitude are different.

Introduction to HDL and VHDL

Hardware Description Language
Hardware description language (HDL) is a specialized computer language used to program electronic and digital logic circuits. The structure, operation and design of the circuits are programmable using HDL. HDL includes a textual description consisting of operators, expressions, statements, inputs and outputs.

Instead of generating a computer executable file, the HDL compilers provide a gate map. The gate map obtained is then downloaded to the programming device to check the operations of the desired circuit. The language helps to describe any digital circuit in the form of structural, behavioral and gate level and it is found to be an excellent programming language for Programmable Logic Devices like FPGAs (Field Programmable Gate Array) and CPLDs (Complex Programmable Logic Device). The three common HDLs are Verilog, VHDL, and SystemC. Of these, SystemC is the newest. The HDLs will allow fast design and better verification. In most of the industries, Verilog and VHDL are common. Verilog, one of the main Hardware Description Language standardized as IEEE 1364 is used for designing all types of circuits. It consists of modules and the language allows Behavioural, Dataflow and Structural Description. VHDL (Very High-Speed Integrated Circuit Hardware Description Language) is standardized by IEEE1164. The design is composed of entities consisting of multiple architectures. SystemC is a language that consist a set of C++ classes and macros. It allows electronic system level and transaction modeling.

Computer Organization | Instruction Formats (Zero, One, Two and Three Address Instruction)

A computer performs a task based on the instruction provided. Instruction in computers comprises groups called fields. These fields contain different information as for computers everything is in 0 and 1 so each field has different significance based on which a CPU decides what to perform. The most common fields are:

- Operation field specifies the operation to be performed like addition.
- Address field which contains the location of the operand, i.e., register or memory location.
- Mode field which specifies how operand is to be founded.

Instruction is of variable length depending upon the number of addresses it contains. Generally, CPU organization is of three types based on the number of address fields:

1. Single Accumulator organization
2. General register organization
3. Stack organization

In the first organization, the operation is done involving a special register called the accumulator. In second on multiple registers are used for the computation purpose. In the third organization the work on stack basis operation due to which it does not contain any address field. Only a single organization doesn't need to be applied, a blend of various organizations is mostly what we see generally.

Based on the number of address, instructions are classified as:

Note that we will use X = (A+B)*(C+D) expression to showcase the procedure.

1. **Zero Address Instructions**
   A stack-based computer does not use the address field in the instruction. To evaluate an expression first it is converted to reverse Polish Notation i.e. Postfix Notation.

   Expression: X = (A+B)*(C+D)

   Postfixed: X = AB+CD+*
   TOP means top of stack
   M[X] is any memory location

```
PUSH    A    TOP = A

PUSH    B    TOP = B

ADD          TOP = A+B

PUSH    C    TOP = C

PUSH    D    TOP = D

ADD          TOP = C+D

MUL          TOP = (C+D)*(A+B)

POP     X    M[X] = TOP
```

2. **One Address Instructions –**
   This uses an implied ACCUMULATOR register for data manipulation. One operand is in the accumulator and the other is in the register or memory location. Implied means that the CPU already knows that one operand is in the accumulator so there is no need to specify it.

   Expression: X = (A+B)*(C+D)
   AC is accumulator
   M[] is any memory location
   M[T] is temporary location

```
LOAD    A    AC = M[A]

ADD     B    AC = AC + M[B]

STORE   T    M[T] = AC

LOAD    C    AC = M[C]

ADD     D    AC = AC + M[D]

MUL     T    AC = AC * M[T]

STORE   X    M[X] = AC
```

3. **Two Address Instructions –**
   This is common in commercial computers. Here two addresses can be specified in the instruction. Unlike earlier in one address instruction, the result was stored in the accumulator, here the result can be stored at different locations rather than just accumulators, but require more number of bit to represent address.

   Here destination address can also contain operand.
   Expression: X = (A+B)*(C+D)
   R1, R2 are registers

   M[] is any memory location

```
MOV     R1, A       R1 = M[A]

ADD     R1, B       R1 = R1 + M[B]

MOV     R2, C       R2 = C

ADD     R2, D       R2 = R2 + D

MUL     R1, R2      R1 = R1 * R2

MOV     X, R1       M[X] = R1
```

4. **Three Address Instructions –**
   This has three address field to specify a register or a memory location. Program created are much short in size but number of bits per instruction increase. These instructions make creation of program much easier but it does not mean that program will run much faster because now instruction only contain more information but each micro operation (changing content of register, loading address in address bus etc.) will be performed in one cycle only.

   Expression: X = (A+B)*(C+D)
   R1, R2 are registers
   M[] is any memory location

```
ADD     R1, A, B      R1 = M[A] + M[B]

ADD     R2, C, D      R2 = M[C] + M[D]

MUL     X, R1, R2     M[X] = R1 * R2
```