Register Transfer and Micro operations

Register Transfer and Register Transfer language(RTL)

Register Transfer Language

A digital system is an interconnection of digital hardware modules that accomplishes a specific information-processing task. Digital modules are best defined by the registers they contain and the operations that are performed on the data stored in them. The operations executed on data stored is registers are called micro-operations. A micro-operation is an elementary operation performed on the information stored in one or more registers. The result of the operations may replace the previous binary information of a register or may be transferred to another register. Examples of micro-operations are shift, count, clear and load.

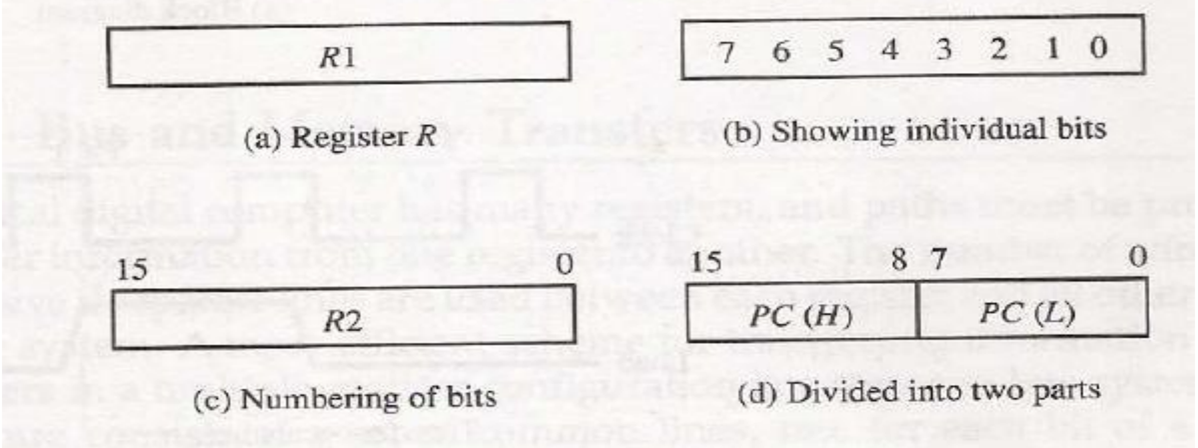
The internal hardware organization of a digital computer is best defined by specifying:

1) The set of registers it contains and their function.
2) The sequence of micro-operations performed on the binary information stored in the registers.
3) The control that initiates the sequence of micro-operations.

The process of specifying the micro-operations in the computer in descriptive explanations could be long and tedious. A more convenient way is to adopt a suitable symbolic notation to describe the micro-operations, transfer among registers is called register transfer language(RTL). A register transfer language is a system for expressing in symbolic form of micro-operations sequences among the register of a digital module.

Register Transfer

Figure 4-1 Block diagram of register.



Computer registers are those that hold the data, they are designated by capital letters to denote the function of the register, e.g. PC (Program Counter), IR (Instruction Register).

The individual flip-flops in an n-bit register are number in sequence from 0 through n-1, starting from 0 in the right most position and increasing the numbers towards left. Figure shows the representation of register block diagram. The name in 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 higher order byte.

Information transfer from one register to another is designated in symbolic form by means of replacement operations. The statement $R_2 \leftarrow R_1$ denotes a transfer of the content or register R_1 into register R_2. It designates a replacement of the content of R_2 by the content of R_1. By definition the content of the source register R_1 does not change after transfer.

We want transfer to occur only under a predefined control condition. This can be shown by means of an if-then statement

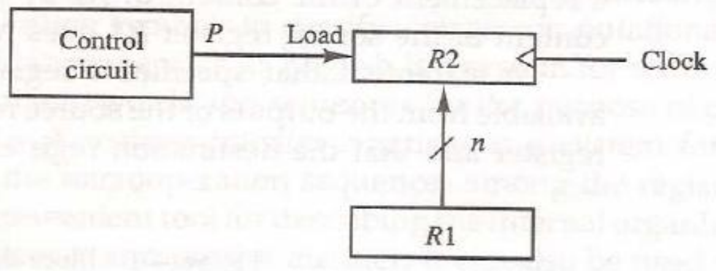
If(P=1), then($R_2 \leftarrow R_1$)

Where P is a control signal generated in control section. It is sometimes convenient to separate the control variables from the register transfer operation by specifying a control function. A control function is a Boolean variables that is equal to 1 or 0. The control function is included in the statement as follows: $P: R2 \leftarrow R1$

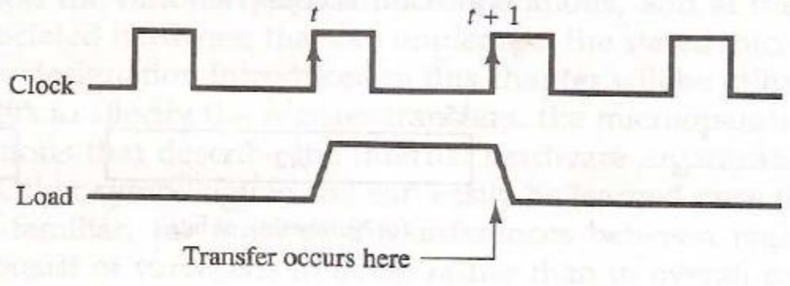
It symbolizes the requirement that the transfer operation be executed by hardware only if P=1.

Figure shows the block diagram that depicts the transfer from R1 to R2.

Figure 4-2 Transfer from $R1$ to $R2$ when $P = 1$.



(a) Block diagram



(b) Timing diagram

The n outputs of register R_1, are connected to the n inputs of register R_2, register R_2 has a load input that is activated by control variable P. It is assumed that the control variable is synchronized with same clock as the one applied to the register. P is active in the control section by the rising edge of a clock pulse at time t. The next positive transition of the clock at time(t+1) find the load input active and data inputs of R_2 are then loaded into the register in parallel. P may go back 0 at time t+1, otherwise the transfer will occur with every clock pulse transition while P remains active.

The statement $T: R_2 \leftarrow R_1 , R_4 \leftarrow R_3$

Denotes an operation that exchanges the content of two register during one common clock pulse provided that T=1.

Bus and memory transfer

In a digital computer there may be many register, so that there will be many data paths if separate lines are used between each register, so a more efficient way for transferring information between register in multiple register configuration in a common bus system. A bus structure consists of a set of common lines, one for each bit of register through which binary information is transferred one at a time.

The transfer of information from a bus into one of many destination register can be accomplished by connection the bus lines to the inputs of all destination registers and activation the load control of the particular destination register selected. When the bus is included in the statement the register transfer may be symbolized as

BUS$\leftarrow$C, $R_1\leftarrow$BUS

The content of register C is placed on the bus and the content of the bus is loaded into register R1, by activating load control input. If bus is known to exist in the system, it may be convenient just to show the direct transfer

$R_1\leftarrow C$

Memory transfer

The transfer of information from memory word to the outside environment is called read operation. The transfer of new information to be stored into the memory is called write operation. A memory word will be symbolized by letter M. 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 following letter M.

Example : read DR$\leftarrow$M[R]

Consider a memory unit that receives the address from a register called the address register(AR). The data are transferred to another register called data register(DR). This causes a transfer of information into DR from the memory word M selected by the address AR.

Example : $R_3 \leftarrow R_1 + \overline{R_2} + 1$

Assume the input data are in register R_1, $\overline{R_2}$ is the symbol for the 1's complement of R_2. Adding the content of R_1 to 2's complement of R_2 is equivalent to R_1-R_2.

Arithmetic micro-operations

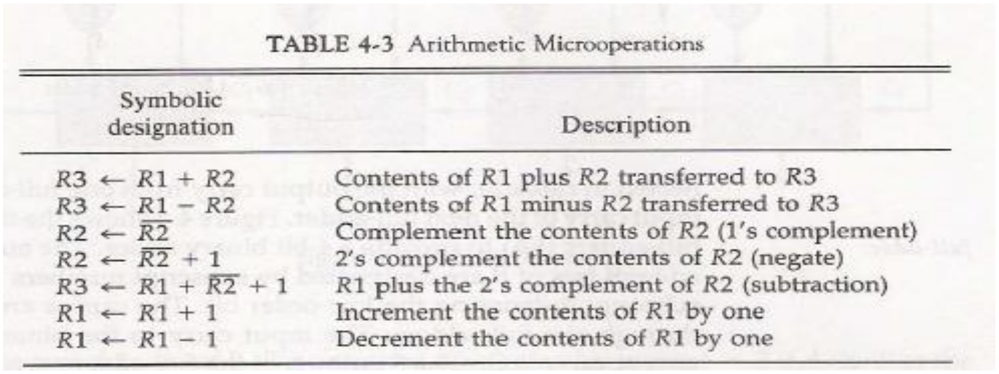


TABLE 4-3 Arithmetic Microoperations

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 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

Logical micro-operations

Logic micro-operations specify binary operations for strings of data bits stored in registers. These operations consider each bit of register separately and treat them as binary variables. For example, the exclusive-OR micro operations with the content of two register R_1 and R_2 is symbolized by the statement $P{:}R_1 \leftarrow R_1 \oplus R_2$

It specifies a logic micro-operations to be executed on the individual bits of the register provided that the control variable P=1. Example consider R_1 contains 1010 & R_2 contains 1100. The exclusive –OR micro operation stated above symbolized the logical operations of logical XOR of individual bits.

1	0	1	0
1	1	0	0
0	1	1	0

The content of R_1 after the execution of the micro-operations is equal the bit-by –bit exclusive OR on pairs of bits in R_2 and previous value of R_1.

Special symbols

Special symbols will be adopted for the logic micro-operations of OR, AND and complement, to distinguish them from the corresponding symbols used to express Boolean functions. Example ‘V’ will be used to denote an OR , ’Λ’ between R_5 & R_6 designated ‘OR’ micro-operations.

Shift micro-operations

Shift micro-operations are used for serial transfer of data. They are used in conjunction with arithmetic, logical and other data processing operations. The content of a register can be shifted to the left or the right. The information transferred through the serial inputs determines the types of shift. There are three types of shift

1. Logical shift: a logical shift is one that transfers 0 through the serial input. We will adopt the symbol shl and shr for logical shift-left and shift right micro-operations.
   $R_1 \leftarrow \text{shl } R_1$, $R_2 \leftarrow \text{shr } R_2$

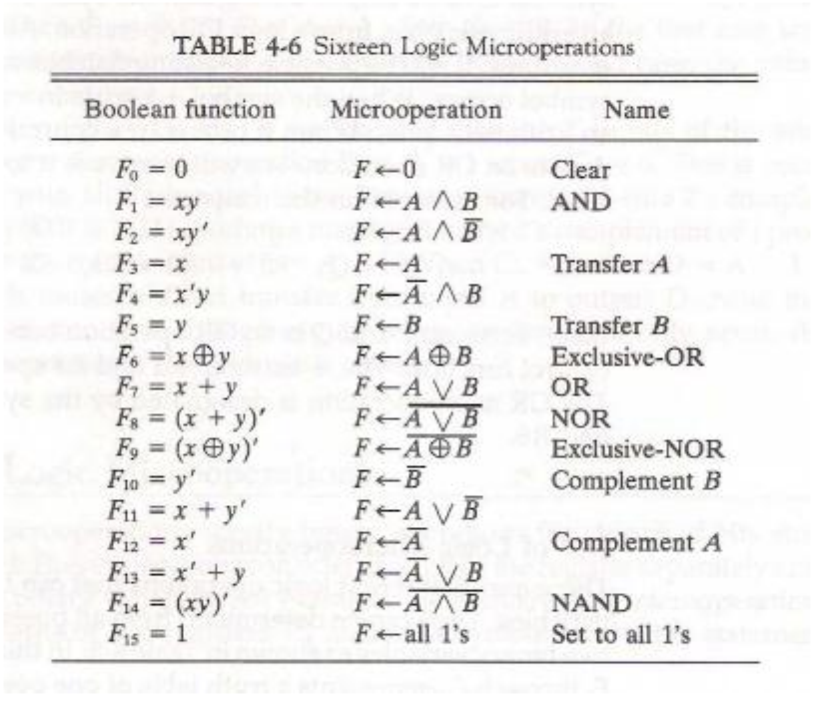


TABLE 4-6 Sixteen Logic Microoperations

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

1	0	1	1
0	1	0	1

Above are two micro-operations that specify 1 bit to the left of the content of register R_1 and 1bit shift to the right of the content of register R_2. The register symbol must be of the same on the both side of the arrow.

2. Circular shift: the circular shift (also known as rotate operations) circulates the bits of the register around the two ends without loss of information. This is accomplished by connecting the serial output of the shift register to its serial inputs. Use the symbol as cil or cir
3. Arithmetic shift: an arithmetic shift is a micro-operation that shifts a signed binary number to the left or right. An arithmetic shift left multiples a signed binary number by 2 and arithmetic shift right divides the number by 2.

The arithmetic shift must leave the sign bit unchanged because the sign of the number remains the same when it is multiples or divided by 2. The left most bit in register holds the sign bit and the remaining bits hold the numbers. Figure shows a typical register of n bits. Bit R_{n-1} in the left most position holds the sign bit. R_{n-2} is the most significant bit of the number and R_0 is the least significant bit. The arithmetic shift-right leaves the sign bit unchanged and shift the number(including the sign bit) to the right. Thus R_{n-1} remains the same R_{n-2} receives the bit from R_{n-1} and so on for other bits in the register. The bit R_0 is lost.

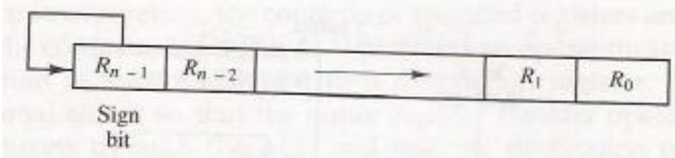


Figure 4-11 Arithmetic shift right.

Introduction to HDL and VHDL

Hardware Description Language(HDL) is a specialized computer language used to program the structure, design and operation of electronic circuits and most commonly digital logic circuits.

A hardware description language enables a precise, formal description of an electronic circuits that allows for the automated analysis, simulation and simulated testing of an electronic circuit. It also allows for the compilation of an HDL program into a lower level specification of physical electronic components.

What are the advantages of using HDLs?

- Can express large, complex design.
- Flexible modelling capabilities.

- Description can include the very abstract to the very structural level
- Productivity
    - Logic synthesis: easily develop, use the available logics.
    - Design changes are fast and easily done.
    - Optimization of design is easier.
    - Exploration of alternative design can be done quickly.
- Reusability:
    - Packages, libraries, design all can be reused.
    - Vendors and technology independence
    - E.g. CMOS, ECL, gates same code.
- Documentation
    - Textual documentation is part of the code, not a separate document.

Example:

```
// This module creates an address/accumulator

Module adder (

Input clk, // input clock

Input reset_n, // async active low

Input first_select, // on first data item use this

Input rd_fifo, //enable for ff

Input [7:0] data, //data in

Output reg[11:0] acc_out //data out of accumulator

);

Always @ (posedge clk, nedge reset_n)

If(!reset_n) acc_out<=12'h000;

Else

If(rd_fifo==1 'b')

If(first_selected==1'b') acc_out<=data

Else                    acc_out<=acc_out+data;

End module
```

What can't an HDL do?

- We cannot make architectural tradeoffs, but it can help.
- Does not relieve in understanding digital design.
- If you can't draw the structure you are looking for stop coding.

VDHL(Very High Speed Integrated Circuit (VHSIC) HDL

- More difficult to learn
- Widely used for FPGAs

Verilog

- Simpler to learn.
- Look like C.

VHDLs history

- VHDL is the product of US government request for a new means of describing digital hardware.
- VHSIC program was an initiative of the defense department to push the state of the art in VLSI technology and HDL was proposed as a versatile hardware description language.

Reason for Using VHDL

- International IEEE standard specification language.
- Enables hardware modelling from the gate to system level.
- Provides a mechanism for digital design and design documentation.
- Formal model to communicate.
- Modelling: documentation.
- Testing and validation using simulation.
- Performance prediction.
- Automatic synthesis.
- Concurrency, more than one thing going at a time.
- VHDL allows the designers to work at various levels of abstraction.
- Behavioural, RTL, Boolean equation, gates.
- Allows for various design methodologies, top→down, bottom →up.
- Flexible in describing hardware.
- Provides technology independence.
- Describes a wide variety of digital hardware.