CSC 218 Foundation of Sequential Program

Lecture Note 5

(Stored Program Computers)

Stored program computers are also known as the Von Neumann architecture. They group bits into standard-sized sequences. In modern times, standard-sized sequences of bits are:

Bytes: A byte is 8-bits (256 possible values). Example: 00111010

Words: A word is only guaranteed to be more than a byte. Words are often 16-bits (2^{16} more = 65, 526 possible values). 32-bits ($2^{32} = 4 \times 10^9$) or 64-bits ($2^{64} = 10^{19}$).

64-bits CPU just means it is a CPU that uses 64-bit words

STORAGE DEVICES

Registers in Computer Architecture

If you are not familiar with logic gates concepts, you can learn it from here:

- ### Logic Gates used in Digital Computers

Binary information is represented in digital computers by physical quantities called **signals**. Electrical signals such as voltages exist throughout the computer in either one of the two recognizable states. The two states represent a binary variable that can be equal to 1 or 0.

For example, a particular digital computer may employ a signal of **3 volts** to represent binary 1 and **0.5 volt** to represent binary 0. Now the input terminals of digital circuits will accept binary signals of only 3 and 0.5 volts to represent binary input and output corresponding to 1 and 0, respectively.

So now we know, that at core level, computer communicates in the form of 0 and 1, which is nothing but **low** and **high** voltage signals.

But how are different operations performed on these signals? That is done using different logic **Gates**.

What are Gates?

Binary logic deals with binary variables and with operations that assume a logical meaning. It is used to describe, in algebraic or tabular form, the manipulation done by logic circuits called **gates**.

Gates are blocks of hardware that produce graphic symbol and its operation can be described by means of an algebraic expression. The input-output relationship of the binary variables for each gate can be represented in tabular form by a truth-table.

The most basic logic gates are **AND** and **inclusive OR** with multiple inputs and **NOT** with a single input.

Each gate with more than one input is sensitive to either logic 0 or logic 1 input at any one of its inputs, generating the output according to its function. For example, a multi-input AND gate is sensitive to logic 0 on any one of its inputs, irrespective of any values at other inputs.

The various logical gates are:

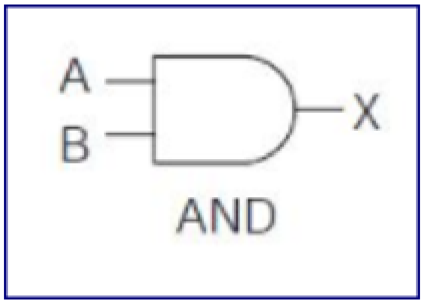
1. AND
2. OR
3. NOT
4. NAND
5. NOR
6. XOR
7. XNOR

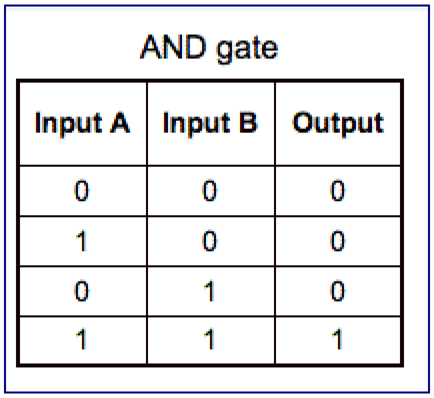
AND Gate

The AND gate produces the AND logic function, that is, the output is 1 if input A and input B are both equal to 1; otherwise the output is 0.

The algebraic symbol of the AND function is the same as the **multiplication** symbol of ordinary arithmetic.

We can either use a **dot** between the variables or concatenate the variables without an operation symbol between them. AND gates may have more than two inputs, and by definition, the output is 1 if and only if all inputs are 1.





AND gate

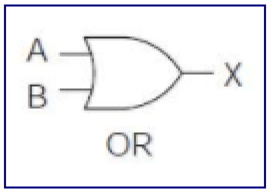
Input A	Input B	Output
0	0	0
1	0	0
0	1	0
1	1	1

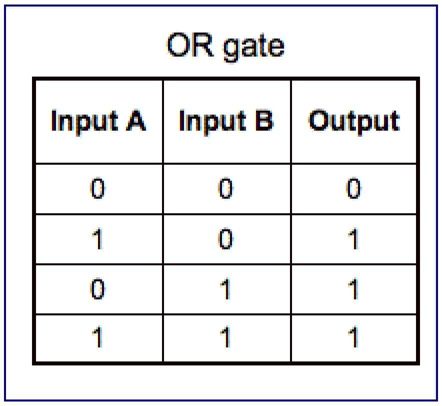
OR Gate

The OR gate produces the inclusive-OR function; that is, the output is 1 if input A or input B or both inputs are 1; otherwise, the output is 0.

The algebraic symbol of the OR function is +, similar to arithmetic **addition**.

OR gates may have more than two inputs, and by definition, the output is 1 if any input is 1.





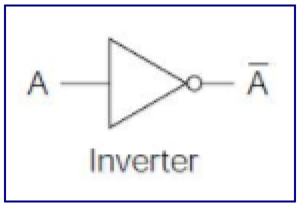
OR gate

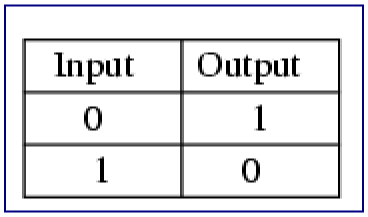
Input A	Input B	Output
0	0	0
1	0	1
0	1	1
1	1	1

Inverter (NOT) Gate

The inverter circuit inverts the logic sense of a binary signal. It produces the NOT, or complement, function.

The algebraic symbol used for the logic complement is either a prime or a bar over the variable symbol.



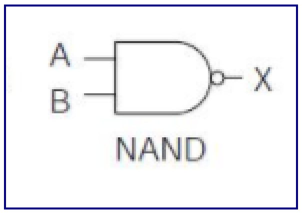


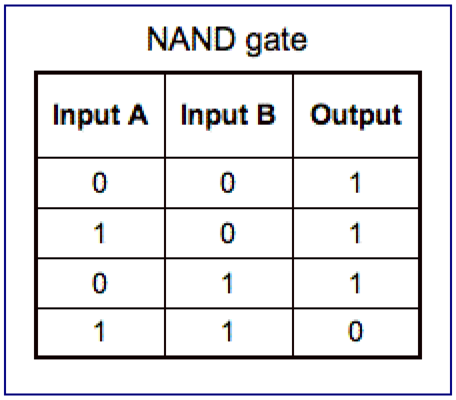
Input	Output
0	1
1	0

NAND Gate

The NAND function is the complement of the AND function, as indicated by the graphic symbol, which consists of an AND graphic symbol followed by a small circle.

The designation NAND is derived from the abbreviation of NOT-AND.



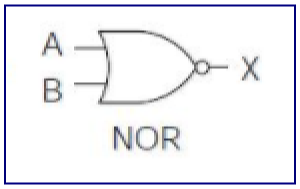


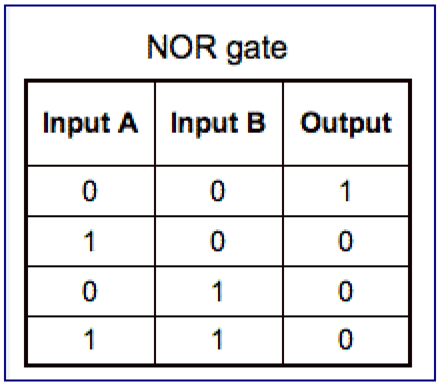
NAND gate

Input A	Input B	Output
0	0	1
1	0	1
0	1	1
1	1	0

NOR Gate

The NOR gate is the complement of the OR gate and uses an OR graphic symbol followed by a small circle.





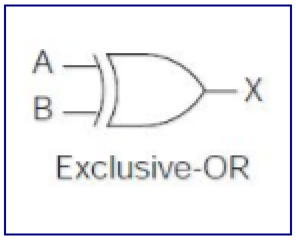
NOR gate

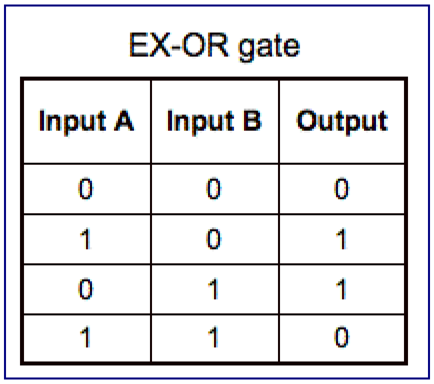
Input A	Input B	Output
0	0	1
1	0	0
0	1	0
1	1	0

Exclusive-OR Gate

The exclusive-OR gate has a graphic symbol similar to the OR gate except for the additional curved line on the input side.

The output of the gate is 1 if any input is 1 but excludes the combination when both inputs are 1. It is similar to an odd function; that is, its output is 1 if an odd number of inputs are 1.





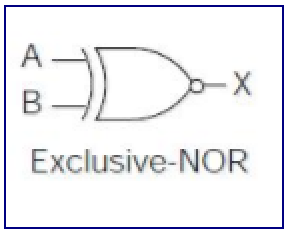
EX-OR gate

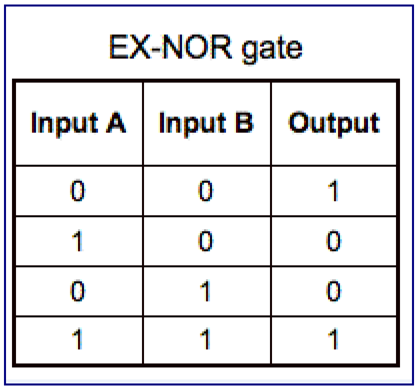
Input A	Input B	Output
0	0	0
1	0	1
0	1	1
1	1	0

Exclusive-NOR Gate

The exclusive-NOR is the complement of the exclusive-OR, as indicated by the small circle in the graphic symbol.

The output of this gate is 1 only if both the inputs are equal to 1 or both inputs are equal to 0.





EX-NOR gate

Input A	Input B	Output
0	0	1
1	0	0
0	1	0
1	1	1

Registers

Register is a very fast computer memory, used to store data/instruction in-execution.

A **Register** is a group of flip-flops with each flip-flop capable of storing **one bit** of information. An *n-bit* register has a group of *n flip-flops* and is capable of storing binary information of *n-bits*.

A register consists of a group of flip-flops and gates. The flip-flops hold the binary information and gates control when and how new information is transferred into a register. Various types of registers are available commercially. The simplest register is one that consists of only flip-flops with no external gates. These days registers are also implemented as a register file.

Registers are typically a finite number of fixed-sized sequences of bits, called registers. You can put bits in, peek at them, and modify them.

Calculators typically have 2-3 registers for recalling numbers and maintaining state.

There are a couple of downsides to registers. They're expensive to build, which is why there is a finite number of them. They're also difficult to keep track of.

Loading the Registers

The transfer of new information into a register is referred to as loading the register. If all the bits of register are loaded simultaneously with a common clock pulse than the loading is said to be done in parallel.

Register Transfer Language

The symbolic notation used to describe the micro-operation transfers amongst registers is called **Register transfer language**.

The term **register transfer** means the availability of **hardware logic circuits** that can perform a stated micro-operation and transfer the result of the operation to the same or another register.

The word **language** is borrowed from programmers who apply this term to programming languages. This programming language is a procedure for writing symbols to specify a given computational process.

Following are some commonly used registers:

1. **Accumulator**: This is the most common register, used to store data taken out from the memory.
2. **General Purpose Registers**: This is used to store data intermediate results during program execution. It can be accessed via assembly programming.
3. **Special Purpose Registers**: Users do not access these registers. These registers are for Computer system,
    - **MAR:** Memory Address Register are those registers that holds the address for memory unit.
    - **MBR:** Memory Buffer Register stores instruction and data received from the memory and sent from the memory.
    - **PC:** Program Counter points to the next instruction to be executed.
    - **IR:** Instruction Register holds the instruction to be executed.

Register Transfer

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

R2 ← R1

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

Normally we want the transfer to occur only in predetermined control condition. This can be shown by following **if-then** statement: if (P=1) then (R2 ← R1)

Here **P** is a control signal generated in the control section.

Control Function

A control function is a Boolean variable that is equal to 1 or 0. The control function is shown as:

P: R2 ← R1

The control condition is terminated with a colon. It shows that transfer operation can be executed only if P=1.

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.

Example: Shift, count, clear and load.

Types of Micro-Operations

The micro-operations in digital computers are of 4 types:

1. Register transfer micro-operations transfer binary information from one register to another.
2. Arithmetic micro-operations perform arithmetic operations on numeric data stored in registers.
3. Logic micro-operations perform bit manipulation operation on non-numeric data stored in registers.
4. Shift micro-operations perform shift micro-operations performed on data.

Arithmetic Micro-Operations

Some of the basic micro-operations are addition, subtraction, increment and decrement.

Add Micro-Operation

It is defined by the following statement:

$R3 \rightarrow R1 + R2$

The above statement instructs the data or contents of register R1 to be added to data or content of register R2 and the sum should be transferred to register R3.

Subtract Micro-Operation

Let us again take an example:

$R3 \rightarrow R1 + R2' + 1$

In subtract micro-operation, instead of using minus operator we take **1's compliment** and add 1 to the register which gets subtracted, i.e **R1 - R2** is equivalent to **R3 → R1 + R2' + 1**

Increment/Decrement Micro-Operation

Increment and decrement micro-operations are generally performed by adding and subtracting 1 to and from the register respectively.

$R1 \rightarrow R1 + 1$

$R1 \rightarrow R1 - 1$

Symbolic Designation	**Description**
R3 ← R1 + R2	Contents of R1+R2 transferred to R3.
R3 ← R1 - R2	Contents of R1-R2 transferred to R3.
R2 ← (R2)'	Compliment the contents of R2.
R2 ← (R2)' + 1	2's compliment the contents of R2.
R3 ← R1 + (R2)' + 1	R1 + the 2's compliment of R2 (subtraction).

R1 ← R1 + 1	Increment the contents of R1 by 1.
R1 ← R1 - 1	Decrement the contents of R1 by 1.

Logic Micro-Operations

These are binary micro-operations performed on the bits stored in the registers. These operations consider each bit separately and treat them as binary variables.

Let us consider the X-OR micro-operation with the contents of two registers R1 and R2.

P: R1 ← R1 X-OR R2

In the above statement we have also included a Control Function.

Assume that each register has 3 bits. Let the content of R1 be **010** and R2 be **100**. The X-OR micro-operation will be:

$$\begin{array}{l} 010 \rightarrow R1 \\ \underline{100} \rightarrow R2 \\ \underline{110} \rightarrow R1 \text{ after } P=1 \end{array}$$

Shift Micro-Operations

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.

There are three types of shifts as follows:

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.

R1 ← she R1

R1 ← she R1

The register symbol must be same on both sides of arrows.

b) Circular Shift

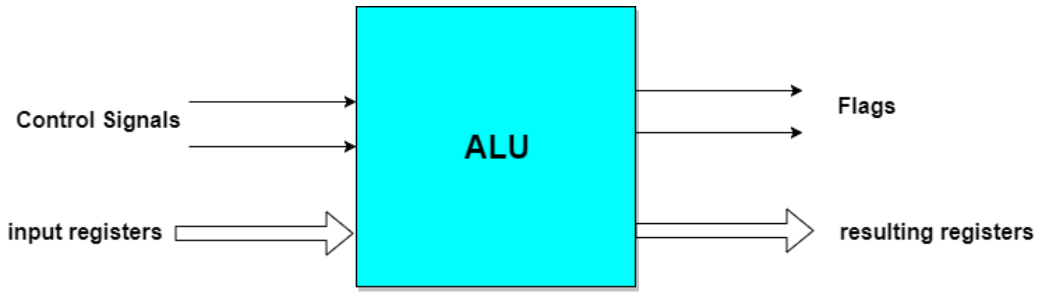
This circulates or rotates the bits of register around the two ends without any loss of data or contents. In this, the serial output of the shift register is connected to its serial input. **"cil"** and **"cir"** is used for circular shift left and right respectively.

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 left** 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.

Arithmetic Logical Unit

Instead of having individual registers performing the micro-operations, computer system provides a number of registers connected to a common unit called as Arithmetic Logical Unit (ALU). ALU is the main and one of the most important unit inisde CPU of computer. All the logical and mathematical operations of computer are performed here. The contents of specific register is placed in the in the input of ALU. ALU performs the given operation and then transfer it to the destination register.



Register Transfer Language (RTL)

In symbolic notation, it is used to describe the micro-operations transfer among registers. It is a kind of intermediate representation (IR) that is very close to assembly language, such as that which is used in a compiler. The term "Register Transfer" can perform micro-operations and transfer the result of operation to the same or other register.

Micro-operations:

The operation executed on the data store in registers is called micro-operations. They are detailed low-level instructions used in some designs to implement complex machine instructions.

Register Transfer:

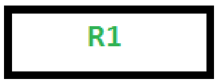
The information transformed from one register to another register is represented in symbolic form by replacement operator is called Register Transfer.

Replacement Operator:

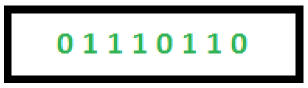
In the statement, R2 <- R1, <- acts as a replacement operator. This statement defines the transfer of content of register R1 into register R2.

There are various methods of RTL –

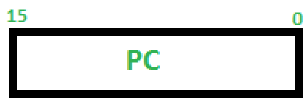
1. General way of representing a register is by the name of the register inclosed in a rectangular box as shown in (a).
2. Register is numbered in a sequence of 0 to (n-1) as shown in (b).
3. The numbering of bits in a register can be marked on the top of the box as shown in (c).
4. A 16-bit register PC is divided into 2 parts- Bits (0 to 7) are assigned with lower byte of 16-bit address and bits (8 to 15) are assigned with higher bytes of 16-bit address as shown in (d).



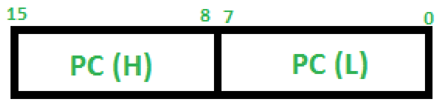
(a)



(b)

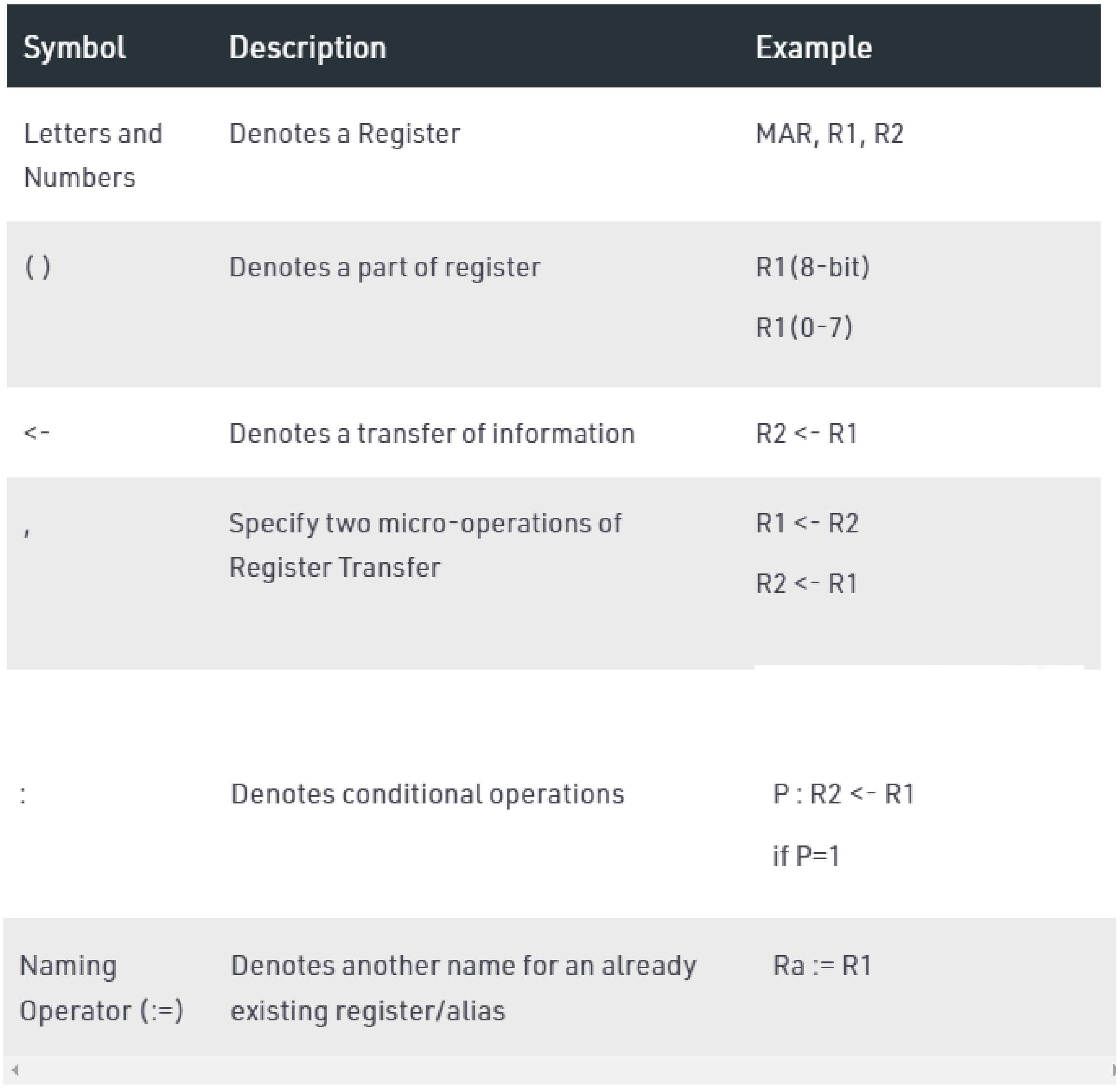


(c)



(d)

Basic symbols of RTL:



Symbol	Description	Example
Letters and Numbers	Denotes a Register	MAR, R1, R2

( )	Denotes a part of register	R1(8-bit)
		R1(0-7)

<-	Denotes a transfer of information	R2 <- R1

,	Specify two micro-operations of Register Transfer	R1 <- R2
		R2 <- R1

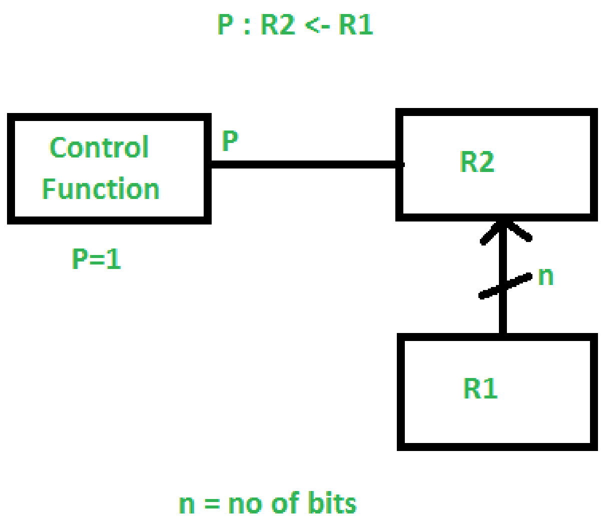
:	Denotes conditional operations	P : R2 <- R1 if P=1
Naming Operator (:=)	Denotes another name for an already existing register/alias	Ra := R1

Register Transfer Operations:
The operation performed on the data stored in the registers is referred to as register transfer operations.

There are different types of register transfer operations:

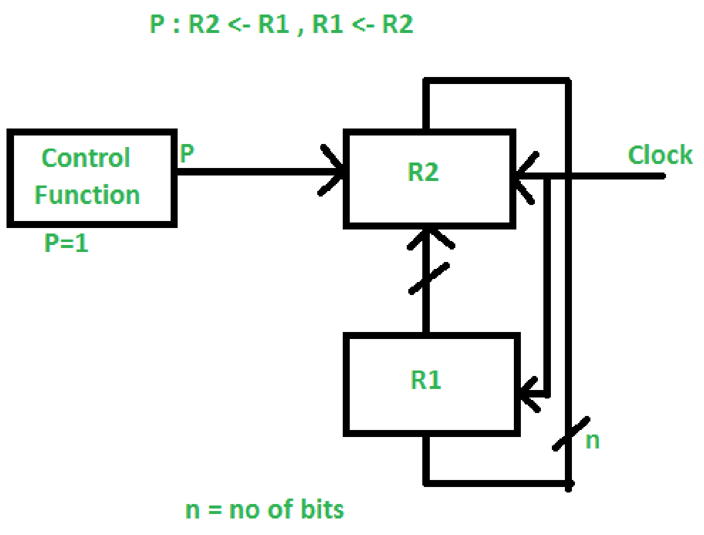
1. Simple Transfer – R2 <- R1
The content of R1 are copied into R2 without affecting the content of R1. It is an unconditional type of transfer operation.

2. Conditional Transfer –



It indicates that if P=1, then the content of R1 is transferred to R2. It is a unidirectional operation.

3. Simultaneous Operations –
If 2 or more operations are to occur simultaneously then they are separated with comma **(,)**.



If the control function P=1, then load the content of R1 into R2 and at the same clock load the content of R2 into R1.

RAM (Random Access Memory)

RAM is essentially a physical array that has address lines, data lines, and control lines.
Data is fed into RAM using electrical lines. Data will remain in RAM until overwritten.
If you want to place a happy face character at address 100, you set the address lines to 100, the data lines to 10001110 (which is the Unicode representation of a happy face), and give the control lines a kick.

RAM could be implemented in several different ways. It could even be created with a cathode ray tube. The core method is synonymous with RAM, however. It involves a magnetic core, and the data remains magnetized after the magnet is removed. Bits are read by toggling the state (toggling the magnetic poles) and seeing if it was easier to toggle than expected (similar to unlocking an already-unlocked door), and then toggling back after. No one really uses magnetic cores anymore.

Capacitive memory (also known as dynamic RAM or DRAM) is still used today. It involves an insulator and two conductive plates, one of which is more negatively-charged than the other. The electrons will remain in their state even when the poles are removed.
There is a problem, however. Insulators are not perfect – electrons will eventually make their way through the insulator. In order to alleviate this, we have to refresh the charge fairly often (every second, for instance).

Switches are typically used only for registers and cache. They produce more heat, but are much faster.

RAM vs. Registers

There are some key differences between RAM and registers:

- There is lots of RAM available, but there are a finite number of registers available (usually not very many).
- You can compute addresses with RAM, but registers have fixed names that cannot be computed (i.e. you can compute memory address 0x00000008 = 0x00000004 + 0x0000004, but you can’t compute $2).
- You can create large, rich data structures in RAM. Registers provide small, fixed, fast storage mechanisms.