Bus and Memory Transfers

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

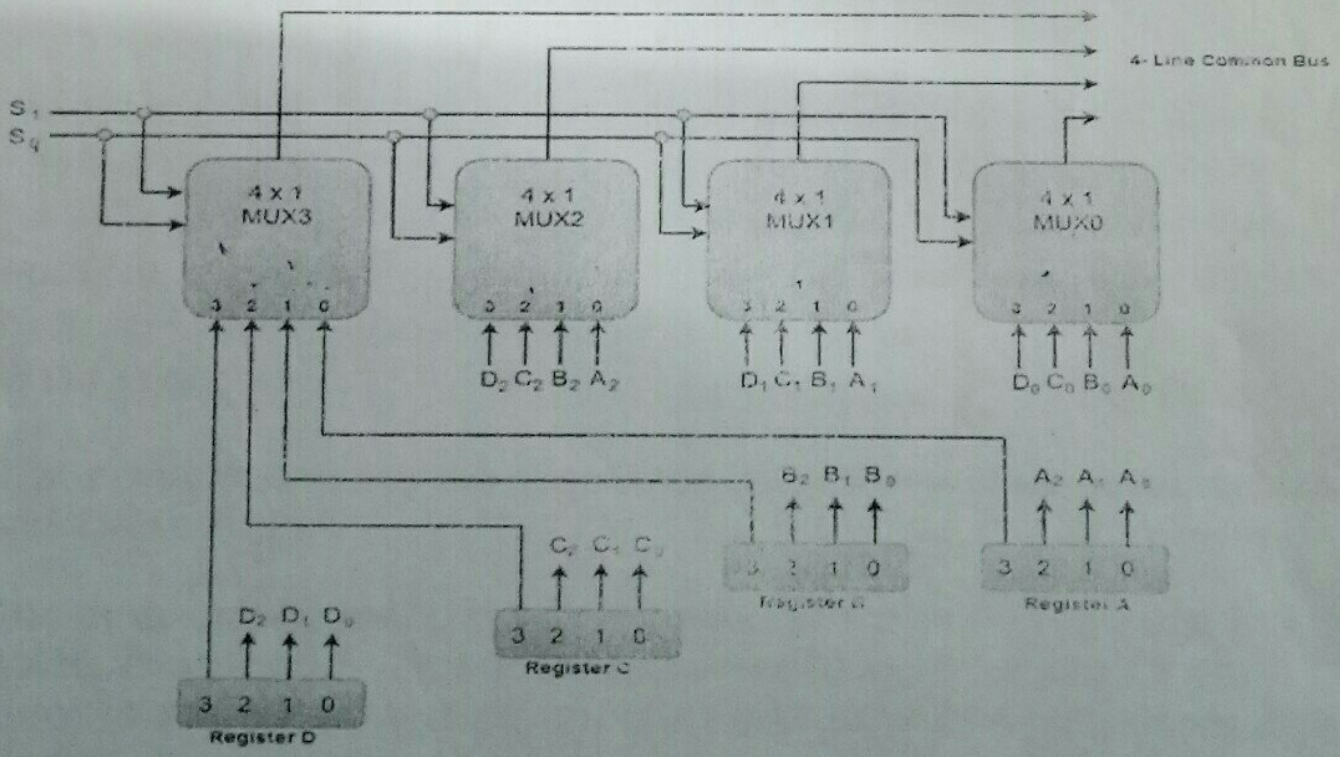
A bus structure, on the other hand, is more efficient for transferring information between registers in a multi-register configuration system.

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

The following block diagram shows a Bus system for four registers. It is constructed with the help of four 4 * 1 Multiplexers each having four data inputs (0 through 3) and two selection inputs (S1 and S2).

We have used labels to make it more convenient for you to understand the input-output configuration of a Bus system for four registers. For instance, output 1 of register A is connected to input 0 of MUX1.

Bus System for 4 Registers:



The two selection lines S1 and S2 are connected to the selection inputs of all four multiplexers. The selection lines choose the four bits of one register and transfer them into the four-line common bus.

When both of the select lines are at low logic, i.e. S1S0 = 00, the 0 data inputs of all four multiplexers are selected and applied to the outputs that forms the bus. This, in turn, causes the bus lines to receive the content of register A since the outputs of this register are connected to the 0 data inputs of the multiplexers.

Similarly, when S1S0 = 01, register B is selected, and the bus lines will receive the content provided by register B.

The following function table shows the register that is selected by the bus for each of the four possible binary values of the Selection lines.

S1	S0	Register Selected
0	0	A
0	1	B
1	0	C
1	1	D

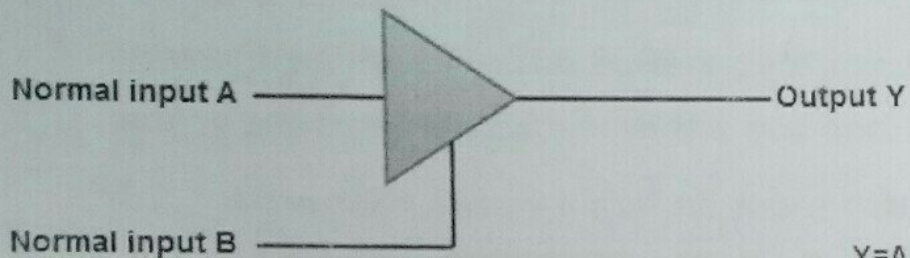
*Note: The number of multiplexers needed to construct the bus is equal to the number of bits in each register. The size of each multiplexer must be 'k * 1' since it multiplexes 'k' data lines. For instance, a common bus for eight registers of 16 bits each requires 16 multiplexers, one for each line in the bus. Each multiplexer must have eight data input lines and three selection lines to multiplex one significant bit in the eight registers.*

A bus system can also be constructed using **three-state gates** instead of multiplexers.

The **three state gates** can be considered as a digital circuit that has three gates, two of which are signals equivalent to logic 1 and 0 as in a conventional gate. However, the third gate exhibits a high-impedance state.

The most commonly used three state gates in case of the bus system is a **buffer gate.**

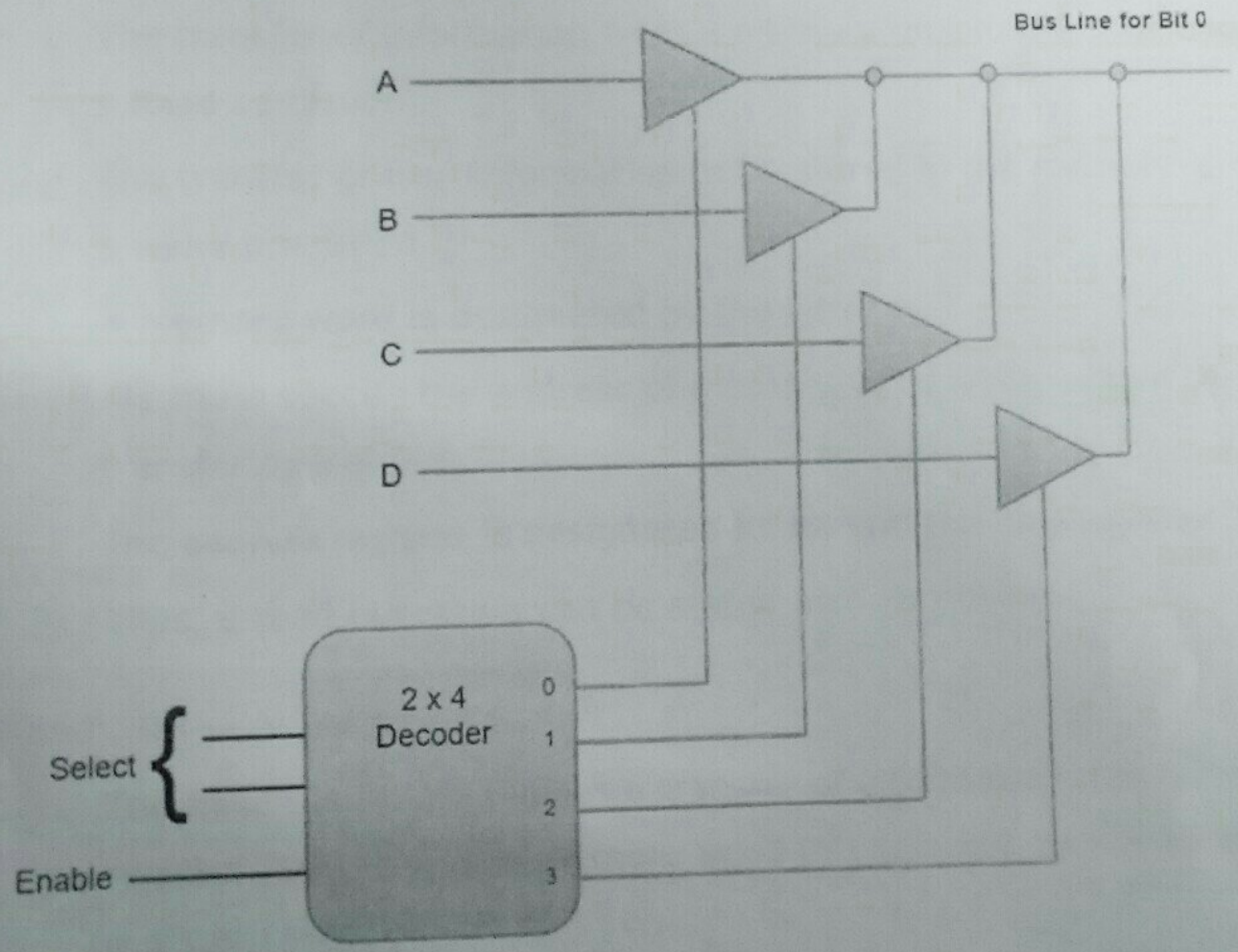
The graphical symbol of a three-state buffer gate can be represented as:



Y=A, if C=1
High-impedance if C=0

The following diagram demonstrates the construction of a bus system with three-state buffers.

Bus line with three state buffer:



- The outputs generated by the four buffers are connected to form a single bus line.
- Only one buffer can be in active state at a given point of time.
- The control inputs to the buffers determine which of the four normal inputs will communicate with the bus line.
- A 2 * 4 decoder ensures that no more than one control input is active at any given point of time.

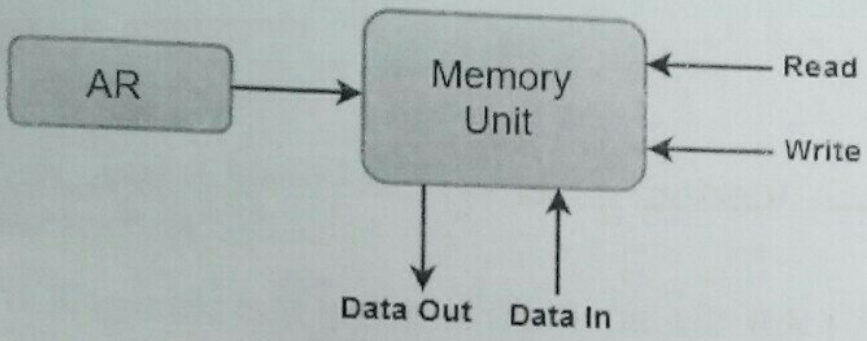
Memory Transfer

Most of the standard notations used for specifying operations on memory transfer are stated below.

- The transfer of information from a memory unit to the user end is called a **Read** operation.
- The transfer of new information to be stored in the memory is called a **Write** operation.
- A memory word is designated by the letter **M**.
- We must specify the address of memory word while writing the memory transfer operations.
- The **address register** is designated by **AR** and the **data register** by **DR**.
- Thus, a read operation can be stated as:

1. Read: DR ← M [AR]
   - The **Read** statement causes a transfer of information into the data register (DR) from the memory word (M) selected by the address register (AR).
   - And the corresponding write operation can be stated as:

1. Write: M [AR] ← R1
    - The Write statement causes a transfer of information from register R1 into the memory word (M) selected by address register (AR).



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.

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.

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. (AC)
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 → 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 → 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 → 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.

$R3 \leftarrow R1 + (R2)' + 1$	R1 + the 2's compliment of R2 (subtraction).
$R1 \leftarrow R1 + 1$	Increment the contents of R1 by 1.
$R1 \leftarrow R1 - 1$	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:

010 → R1

100 → R2

110 → R1 after P=1

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 ← shr 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.