

**INDIAN INSTITUTE OF ENGINEERING  
SCIENCE AND TECHNOLOGY, SHIBPUR**  
Howrah, West Bengal, India - 711103

**DEPARTMENT OF COMPUTER SCIENCE  
AND TECHNOLOGY**



Computer Architecture & Organization Laboratory

[CS2272]

**LAB REPORT**

*SUBMITTED BY*

**Abhiroop Mukherjee (510510109)**

*UNDER THE GUIDANCE OF*

**PROF. BIPLAB K. SIKDAR**

**PROF. JAYA SIL**

**(4th Semester, Academic Year: 2020-2021)**

## Contents

|   |   |    |
|---|---|----|
| 1 | Organization of $m \times d$ memory with Flip Flops   | 1  |
| 2 | Design of Carry-Look-Ahead Adder                      | 5  |
| 3 | Implementation of data movement instructions          | 8  |
| 4 | Swapping contents of registers using Microprogram     | 10 |
| 5 | Design of Sequence Counter for Processor Control Unit | 12 |

# 1 Organization of $m \times d$ memory with Flip Flops

## 1.1 Objective

To design memory module with the features of Read/Write, using RS Flip Flop, considering

- (a)  $m = 1, d = 1$
- (b)  $m = 2, d = 1$
- (c)  $m = 2, d = 2$ , using the design resulted out of (b)

## 1.2 Theoretical Basis

A  $2^m \times d$  Memory Module can be drawn as follows

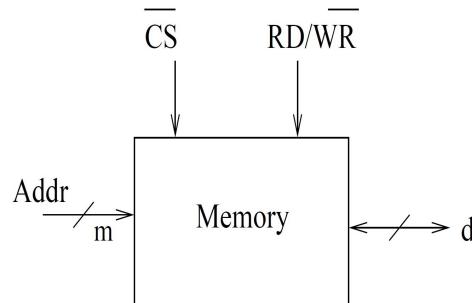


Figure 1: A  $m \times d$  Memory Module

Here CS is Chip Select and  $RD/\overline{WR}$  is Read or Write input.

Given an  $2^m \times d$  Memory Module, we can make  $2^{m+1} \times d$  and  $2^m \times 2d$  Memory Modules as follows

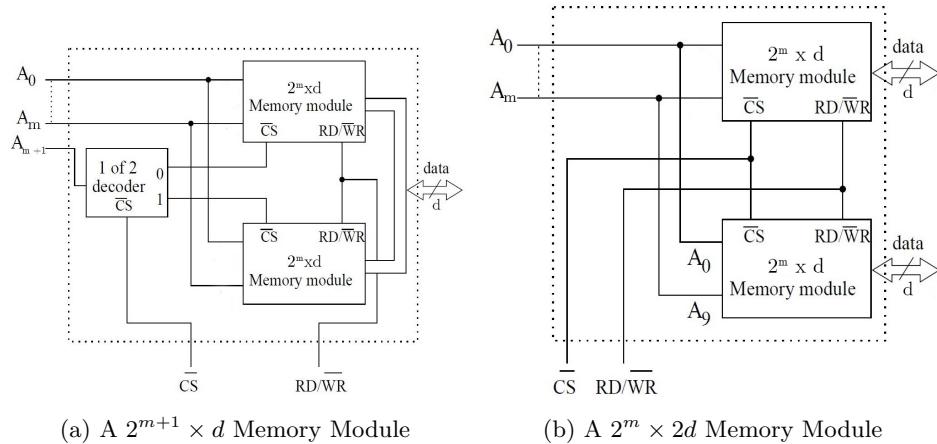


Figure 2: Expansion of Memory Modules

### 1.3 Task

To Realize Circuit of:

- (a) 1 bit 1 word memory
- (b) 1 bit 2 word memory
- (c) 2 bit 2 word memory

### 1.4 Major Components/Modules Used

1. RS Flip Flop
2. Tri-state logic
3. Triple-input NAND Gate
4. Inverter

### 1.5 Design

Following the above Ideas, we construct required memory modules as follows

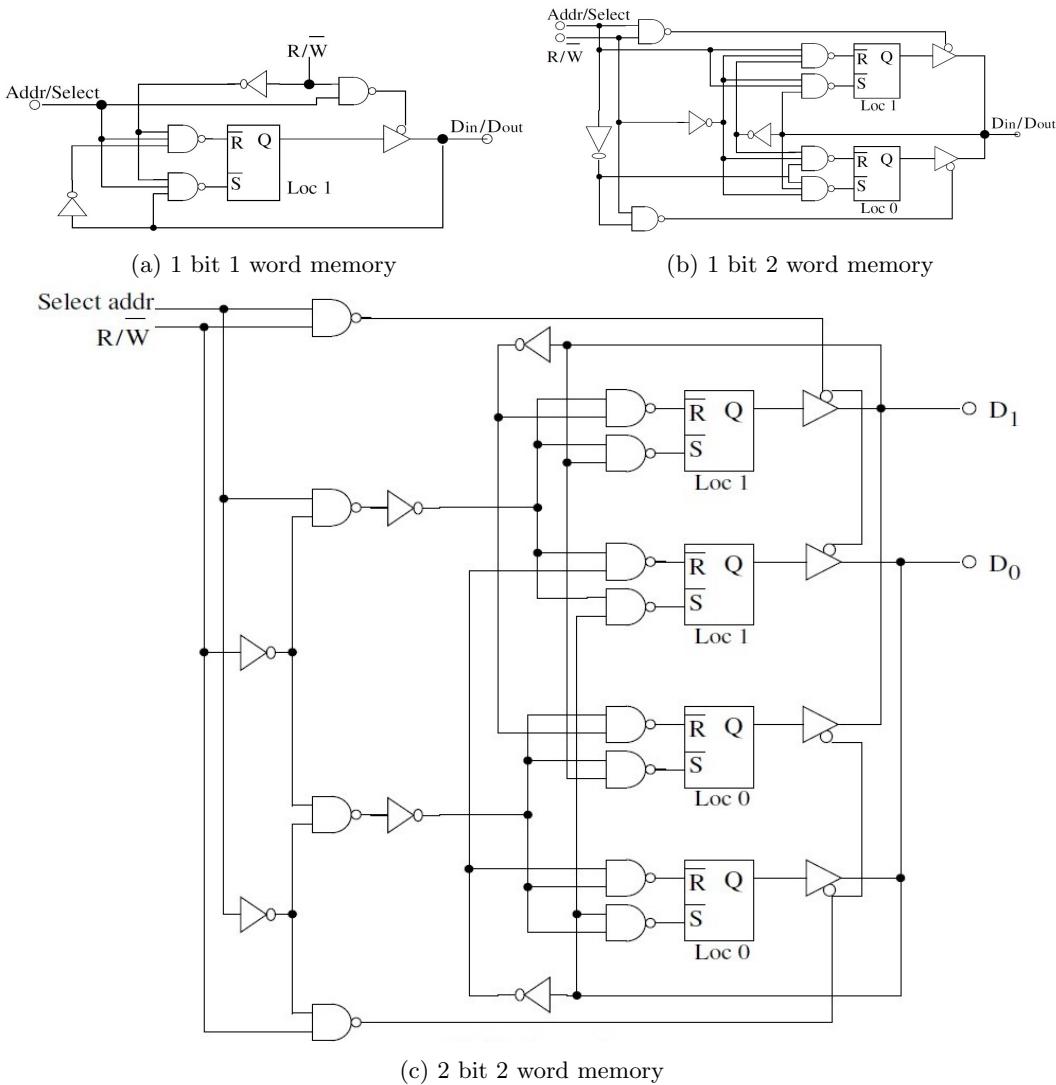


Figure 3: Design Of Memory

## 1.6 Implementation

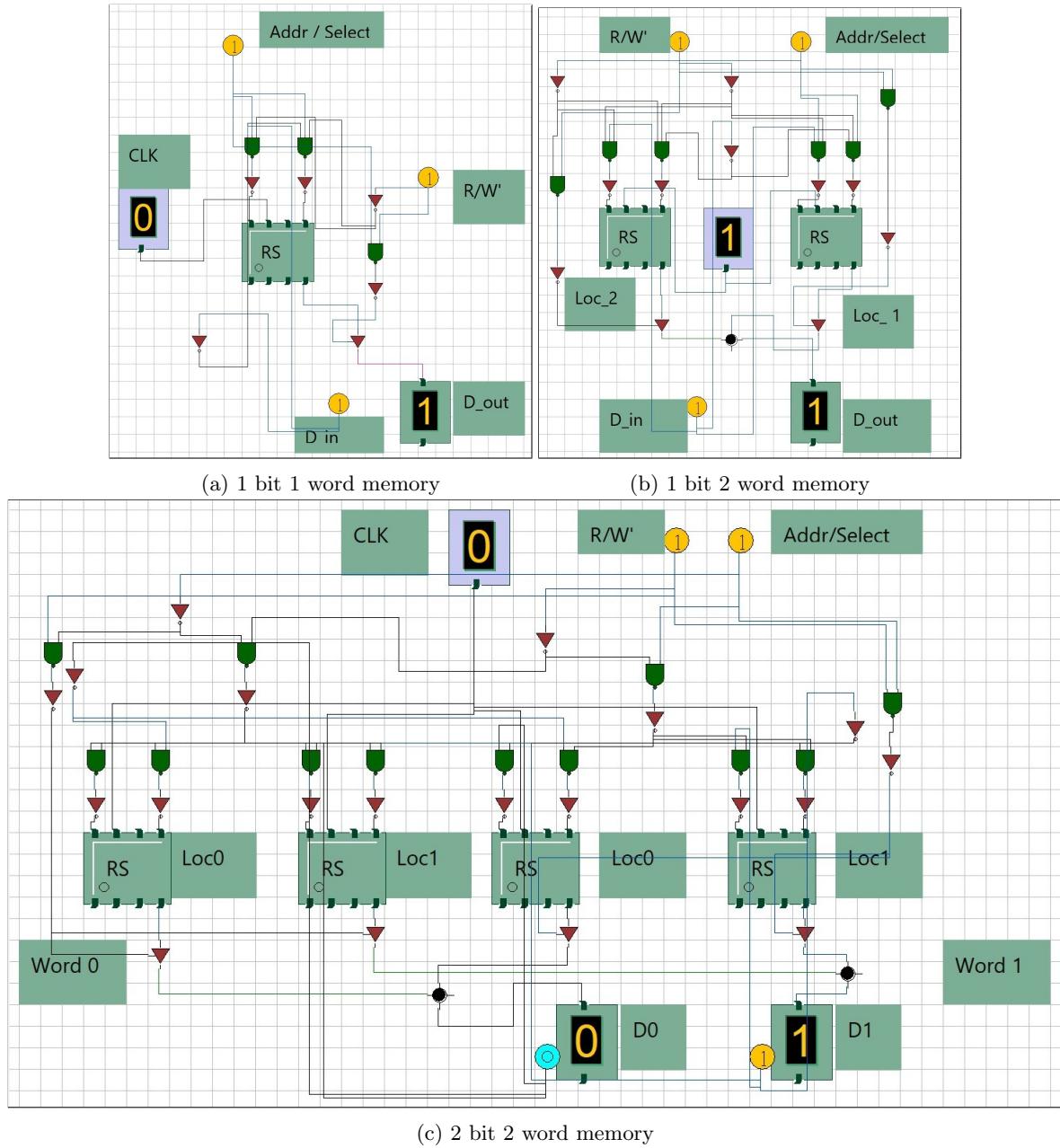


Figure 4: Implementation Of Memory

## 1.7 Verification/Simulation

| Sr. No | Select | R/ $\bar{W}$ | Data In | Data Out | Activity          |
|--------|--------|--------------|---------|----------|-------------------|
| 1      | 1      | 0            | 1       | -        | Write 1 in Loc-1  |
| 2      | 1      | 1            | -       | 1        | Read 1 from Loc-1 |
| 3      | 0      | x            | -       | -        | No operation      |
| 4      | 1      | 0            | 0       | -        | Write 0 in Loc-1  |
| 5      | 1      | 1            | 1       | 0        | Read 0 from Loc-1 |

Table 1: 1 bit 1 word memory

| Sr. No | Select | R/ $\bar{W}$ | Data In     | Data Out    | Activity          |
|--------|--------|--------------|-------------|-------------|-------------------|
| 1      | 1      | 0            | 1 ( $d_1$ ) | -           | Write 1 in Loc-1  |
| 2      | 0      | 0            | 1 ( $d_2$ ) | -           | Write 1 in Loc-0  |
| 3      | 1      | 1            | -           | 1 ( $d_1$ ) | Read 1 from Loc-1 |
| 4      | 0      | 1            | -           | 1 ( $d_2$ ) | Read 1 from Loc-0 |
| 5      | 1      | 0            | 0           | -           | Write 0 in Loc-1  |
| 6      | 0      | 0            | 1           | -           | Write 1 in Loc-0  |
| 7      | 1      | 1            | -           | 0           | Read 0 from Loc-1 |
| 8      | 0      | 1            | -           | 1           | Read 1 from Loc-0 |
| 9      | 1      | 0            | 1           | -           | Write 1 in Loc-1  |
| 10     | 0      | 0            | 0           | -           | Write 0 in Loc-0  |
| 11     | 1      | 1            | -           | 1           | Read 1 from Loc-1 |
| 12     | 0      | 1            | -           | 0           | Read 0 from Loc-0 |

Table 2: 1 bit 2 word memory

| Sr. No | Select | R/ $\bar{W}$ | $D1_{in}$ | $D0_{in}$ | $D1_{out}$ | $D0_{out}$ | Verify             |
|--------|--------|--------------|-----------|-----------|------------|------------|--------------------|
| 1      | 1      | 0            | 1         | 0         | -          | -          | Write 10 in Loc-1  |
| 2      | 0      | 0            | 0         | 1         | -          | -          | Write 01 in Loc-0  |
| 3      | 1      | 1            | -         | -         | 1          | 0          | Read 10 from Loc-1 |
| 4      | 0      | 1            | -         | -         | 0          | 1          | Read 01 from Loc-0 |
| 5      | 1      | 0            | 0         | 0         | -          | -          | Write 00 in Loc-1  |
| 6      | 0      | 0            | 1         | 1         | -          | -          | Write 11 in Loc-0  |
| 7      | 1      | 1            | -         | -         | 0          | 0          | Read 00 from Loc-1 |
| 8      | 0      | 1            | -         | -         | 1          | 1          | Read 11 from Loc-0 |

Table 3: 2 bit 2 word memory

## 1.8 Conclusion

We can design  $2^{m_1} * d_1$  Memory Module where  $m_1 \geq m$  and  $d_1 \geq d$

## 2 Design of Carry-Look-Ahead Adder

### 2.1 Objective

To Design

1. 4-bit carry lookahead adder (CLA) using half adders/full adders.
2. A 16-bit CLA using 4-bit CLAs.

### 2.2 Theoretical Basis and Design

Basic Unit of Carry Look-Ahead Adder is as follows:

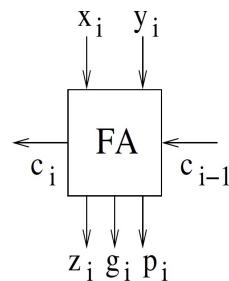


Figure 5: Basic Unit of CLA

where,

$$p_i = x_i \oplus y_i$$

$$g_i = x_i y_i$$

Using this building blocks, we can construct an n-bit CLA as follows

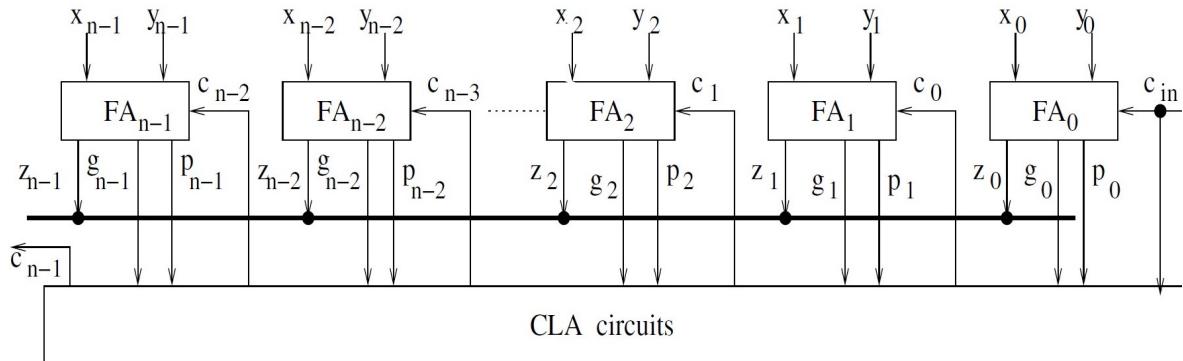


Figure 6: n-bit CLA

Using these we can form this for 4-bit CLA

$$c_0 = g_0 + p_0 c_{in}$$

$$c_1 = g_1 + p_1 c_0 = g_1 + p_1 g_0 + p_1 p_0 c_{in}$$

$$c_2 = g_2 + p_2 c_1 = g_2 + p_2 g_1 + p_2 p_1 g_0 + p_2 p_1 p_0 c_{in}$$

$$c_3 = g_3 + p_3 c_2 = g_3 + p_3 g_2 + p_3 p_2 g_1 + p_3 p_2 p_1 g_0 + p_3 p_2 p_1 p_0 c_{in}$$

Now from 4-bit CLA, we can construct 16-bit CLA as follows:

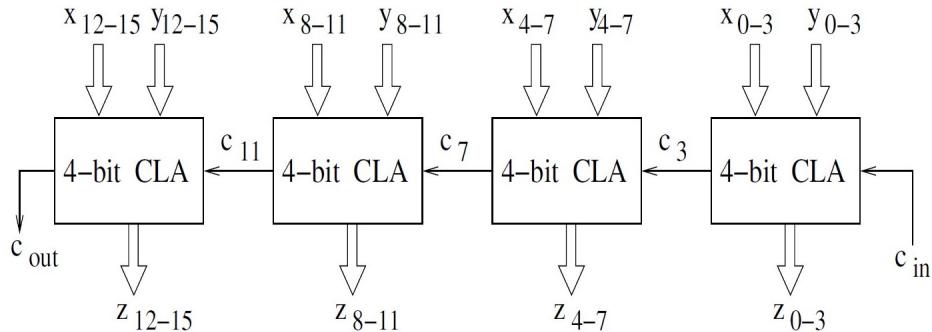


Figure 7: 16-bit CLA

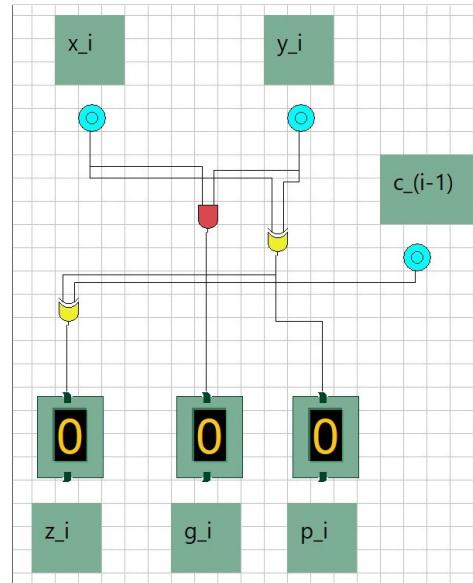
### 2.3 Task

1. Construct 4 modules as in Figure 5 (can use half adder).
2. Simplify the expressions for  $c_0$ ,  $c_1$ ,  $c_2$  and  $c_3$  assuming  $c_{in} = 0$ .
3. Design CLA circuits i.e, realize  $c_0$ ,  $c_1$ ,  $c_2$  and  $c_3$ .
4. Design the CLA as in Figure 6 and verify a part of its truth table.
5. Report on the design of a 16-bit CLA as in Figure 7.

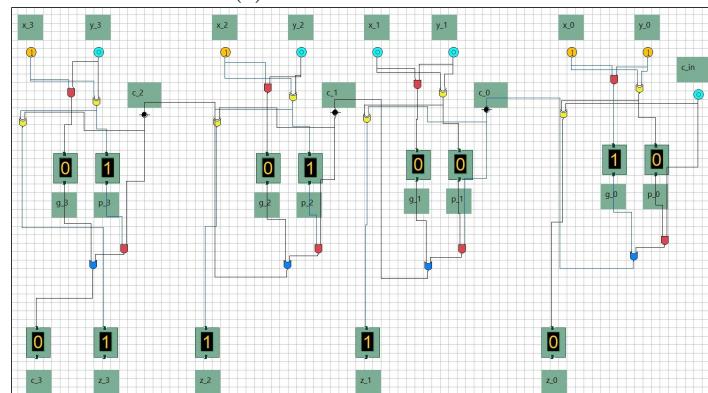
### 2.4 Major Components/Modules Used

1. AND Gate
2. OR Gate
3. Half Adder
4. Full Adder

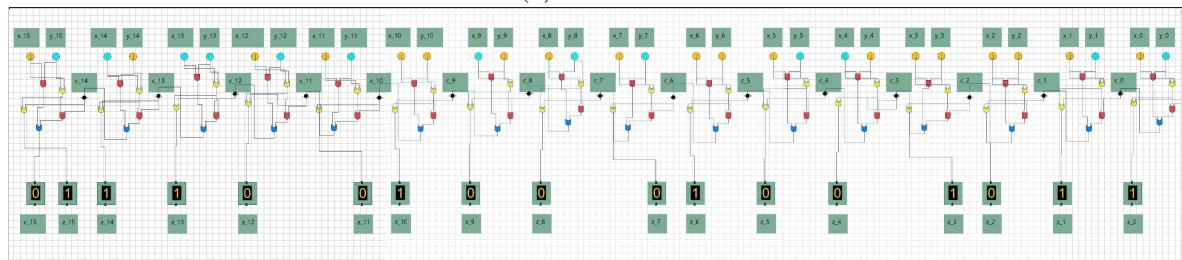
## 2.5 Implementation



(a) Block of CLA Adder



(b) 4 bit CLA



(c) 16 bit CLA

Figure 8: Carry Look Ahead Adder

## 2.6 Verification

- We are successfully able to add two 4-bit numbers using circuit in Figure 8b.
- We are successfully able to add two 16-bit numbers using circuit in Figure 8c

## 2.7 Conclusion

We are able to add two 4-bit number in constant delay time and are able to add two 16-bit numbers in some delay, which is not proportional to number of bits.

### 3 Implementation of data movement instructions

#### 3.1 Objective

To realize data movement instructions in a CPU having four 2-bit registers.

#### 3.2 Theoretical Basis

Data movement instructions move data from one place, called the source operand, to another place, called the destination operand.

Data is transferred directly from a source register in a register file connected to a data bus from a destination register in the register file, through a data bus bypass mechanism.

The figure shown below shows how to implement such data movement instructions when the source register and the destination register belong to the same set of registers.

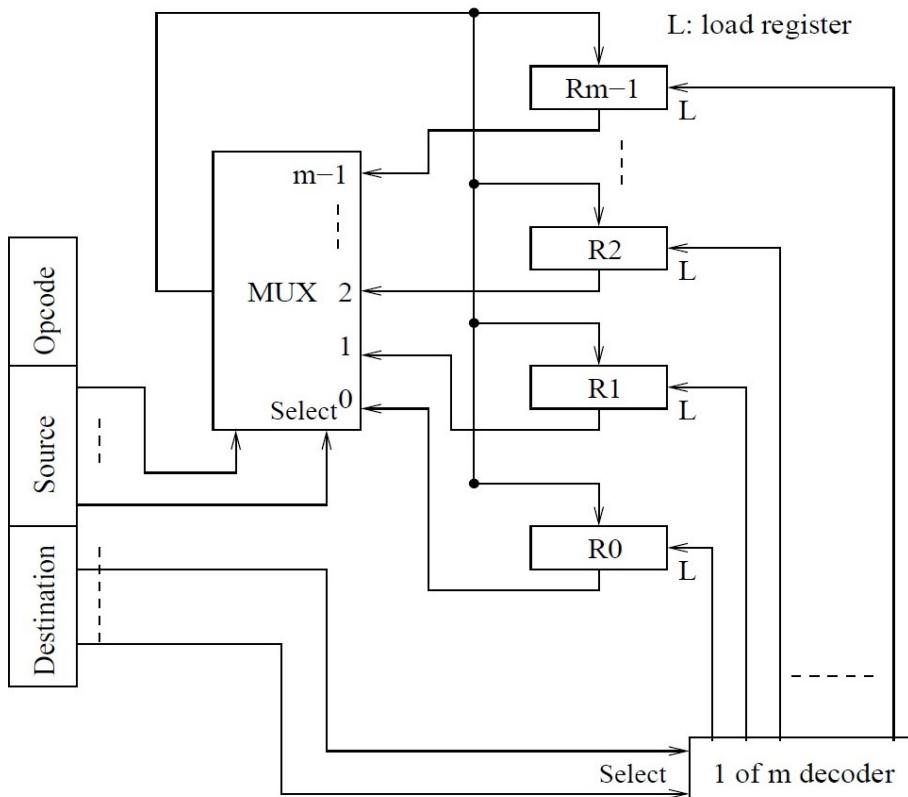


Figure 9: Data Transfer between same set of Registers

#### 3.3 Task

Given four 2-bit registers  $R_0, R_1, R_2$ , and  $R_3$ . We need to realize the implementation of data movement instruction like

*MOV {dest reg addr} {source reg addr}*

Here we set source and destination address manually

#### 3.4 Major Components/Modules Used

1. Four 4-bit Registers, but only using 2 MSB to essentially simulate 2-bit Register
2. Two 4 to 1 Multiplexer
3. One 2 of 4 Decoder

### 3.5 Design

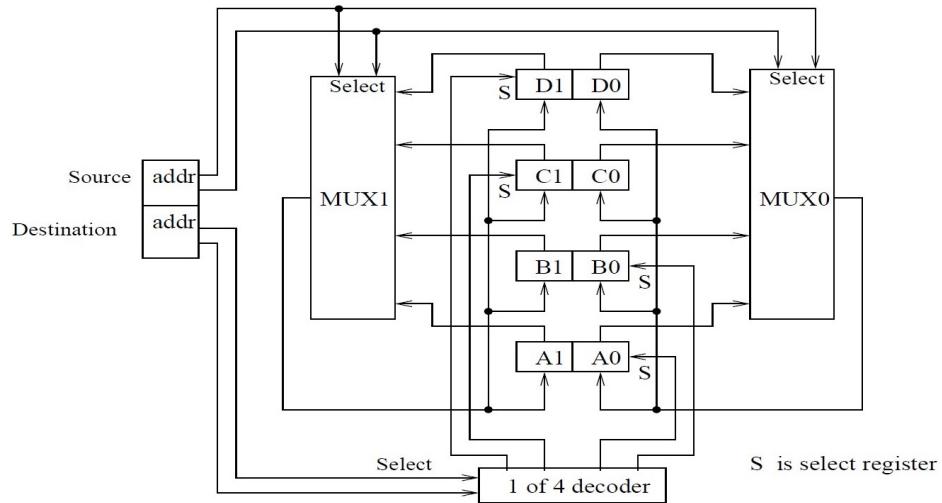


Figure 10: Design Idea to implement Data Transfer between four Registers

### 3.6 Implementation

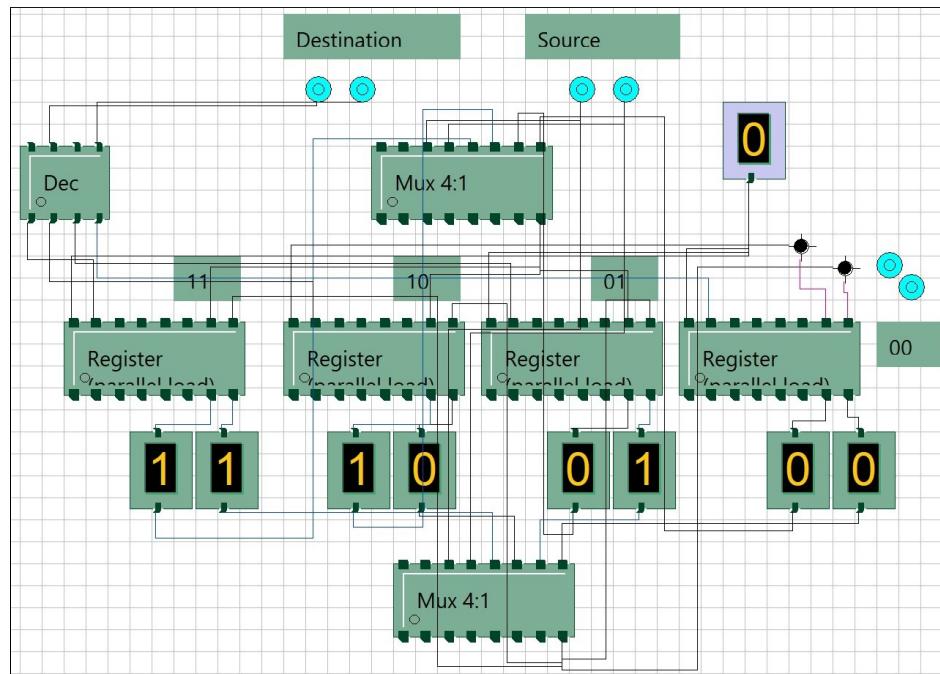


Figure 11: Implementation of Data Transfer between four Registers

### 3.7 Verification

The circuit developed for the implementation of Data Transfer between Registers works as expected. i.e. it moves 2-bit data from one register to destination register.

### 3.8 Conclusion

A circuit for the implementation of data movement between registers has been successfully designed and verified using simulation. Hence the assigned task is complete.

## 4 Swapping contents of registers using Microprogram

### 4.1 Objective

To become familiar with

- (a) Register level data transfer through common bus
- (b) Microprogrammed realization of the register level data transfer

### 4.2 Theoretical Basis

The  $\mu$ -programming is a method of designing a control unit in which control signal selection and sequencing information is stored in a ROM/RAM. The control signals to be activated at time  $t$  are specified by a  $\mu$ -instruction. A sequence of  $\mu$ -instructions related to a task is the  $\mu$ -program. In the present design, the  $\mu$ -program is stored in memory/RAM.

### 4.3 Task and Design

1. Setting registers and control switches (manually) to realize swapping of data:
  - (a) Realize registers A, B and T with D latches/registers as in Figure 12a.
  - (b) Set the switches (control signals) C1, C2, and C3 to realize data swap.
  - (c) Identify additional control signals required to realize swapping through common bus.
2. Automated swapping of data:
  - (a) Realize bi-directional bus (Figure 12b).
  - (b) Identify the control signals required to realize the following sequence of  $\mu$ -instructions
    - i. Copy  $T \leftarrow A$
    - ii. No operation
    - iii. Copy  $A \leftarrow B$
    - iv. No operation
    - v. Copy  $B \leftarrow T$
3. Setting up control memory:
  - (a) Write  $\mu$ -instructions in memory. Generate memory address using counter. Supply clock to generate next addresses (Figure 12c).
  - (b) Rin  $\mu$ -program.

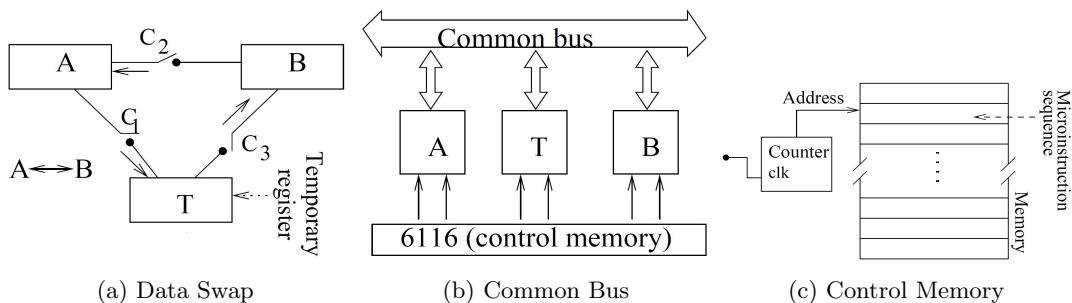


Figure 12: Data Swap between A and B

#### 4.4 Major Components/Modules Used

1. Registers
2. Multiplexers and Decoders
3. D Flip Flops
4. Memory Module

#### 4.5 Implementation

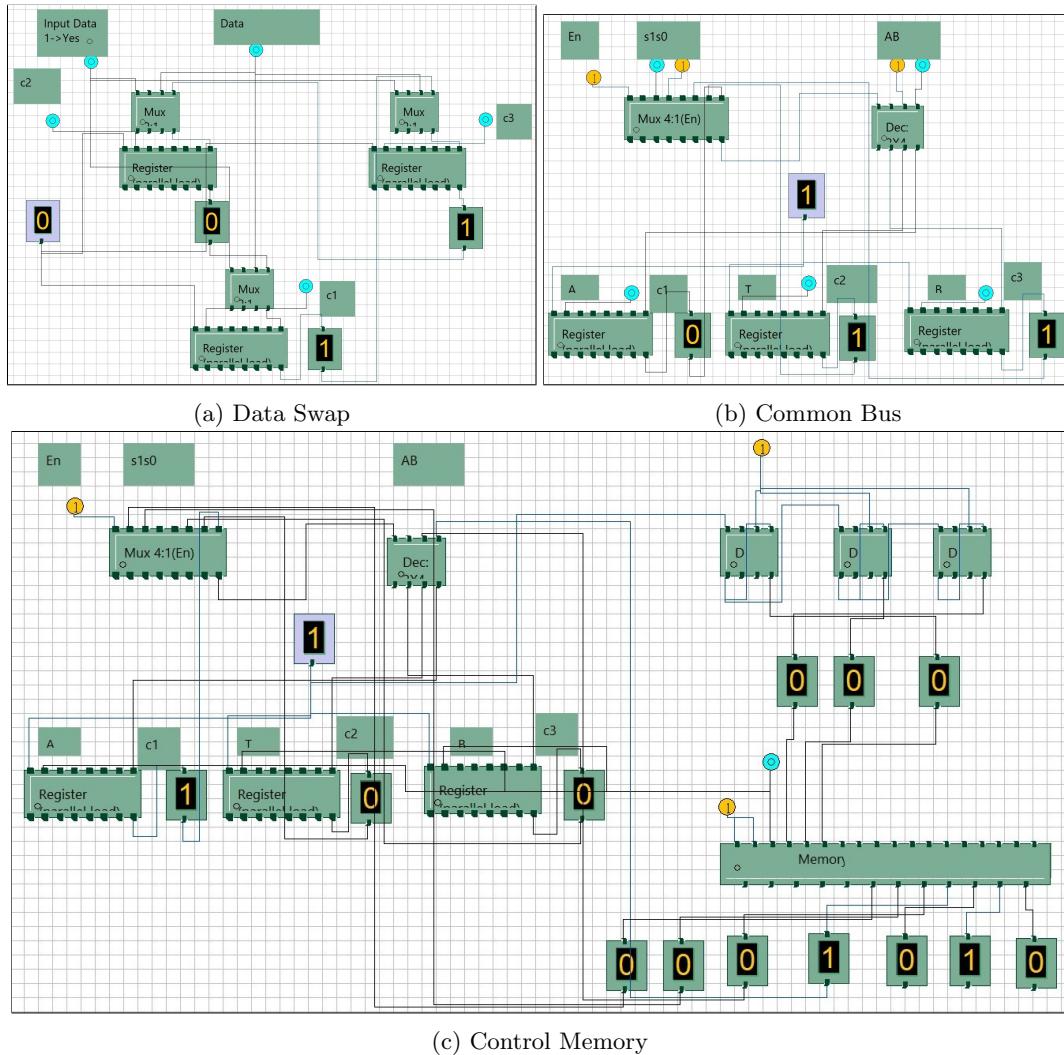


Figure 13: Implementation of Data Swap between A and B

#### 4.6 Verification

1. Successfully swapped content of A and B manually, using T as a temporary register.
2. Successfully implemented common bus and implemented data swap on it.
3. Successfully used Memory to automate the swapping procedure.

#### 4.7 Conclusion

We hence realized register level data transfer using common bus and control memory.

## 5 Design of Sequence Counter for Processor Control Unit

### 5.1 Objective

To build a mod-8 sequence counter for generation of control signals of a processor ALU.

### 5.2 Theoretical Basis

The instruction cycle of a CPU is shown in Figure 14. Assuming that each step is performed in an appropriately chosen clock period. A control unit for this CPU can be built around a modulo-8 sequence counter.

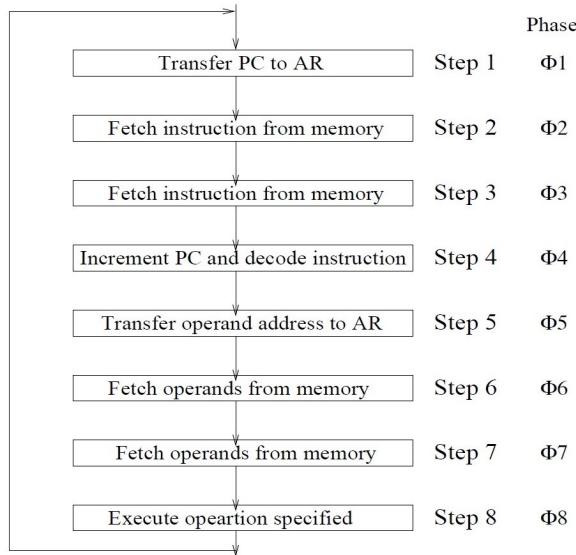


Figure 14: Instruction Cycle of CPU

A modulo-8 counter counts input clock pulses from 000 to 111 and then sets to 000. The modulo-8 sequence counter can be designed with a modulo-8 counter and a 3-to-8 decoder. The decoder decodes the 3-bit binary code into a 1-bit binary code.

### 5.3 Task and Design

1. Design Modulo-8 sequence counter as in figure 16.
2. Design a combinational logic that takes the phase signals ( $\phi_1, \phi_2, \phi_3, \phi_4, \phi_5, \phi_6, \phi_7, \phi_8$ ) as the input generates a sequence of signals to control the primitive operations of the ALU.

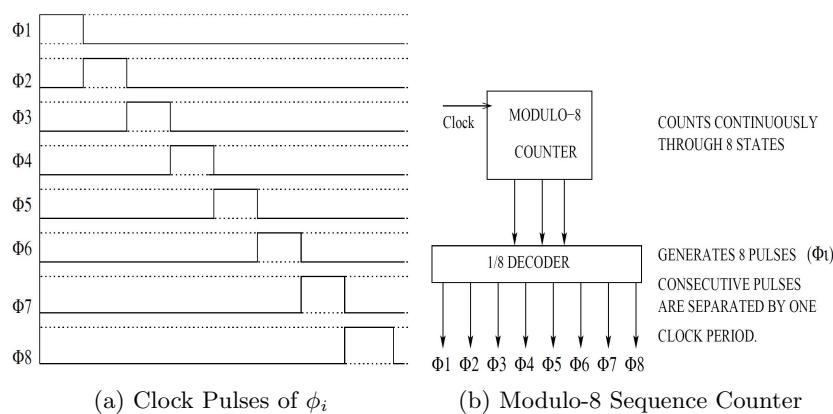
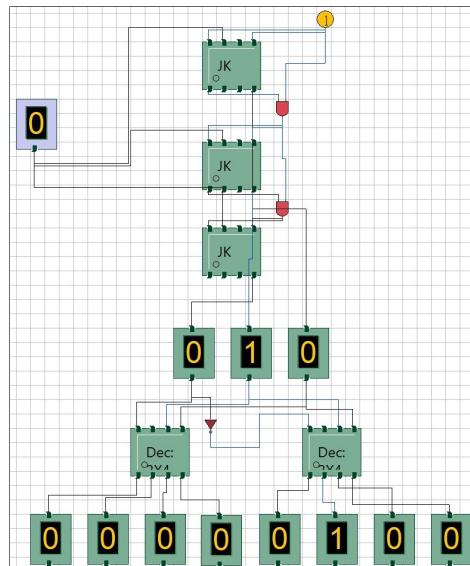


Figure 15: Design Of Modulo-8 Sequence Counter

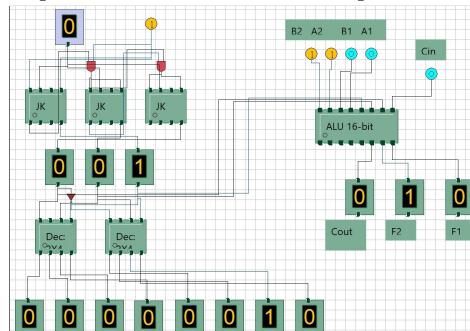
## 5.4 Major Components/Modules Used

1. A modulo-8 counter. The counter was constructed using 3 JK flip flops and 2 AND gate as there was no counter available in the simulator.
2. One 3-to-8 decoder. The decoder was constructed using two 2-to-4 decoders as there was no 3-to-8 decoder available in the simulator.
3. ALU Chip

## 5.5 Implementation



(a) Implementation of Modulo-8 Sequence Counter



(b) ALU running from Modulo-8 Sequence Counter

Figure 16: Implementation of Sequence Counter to Drive ALU

## 5.6 Verification

- The modulo-8 counter designed in the simulator counts from 000 to 111 properly when simulated i.e. when the clock is turned on. It then sets it back to 000. The 3-to-8 decoder decodes the 3-bit binary code (000-111) and produces a single control signal at a time.
- The Control signal then properly drives the ALU

## 5.7 Conclusion

A modulo-8 sequence counter with ALU is successfully designed and verified using the simulator. Hence, the given task is completed.