CSE 404
Report on 4-bit SAP Computer Simulation
Group: A1-G4

Makhdum Ahsan Rupak

Student ID: 1405009

Ali Haisam Muhammad Rafid

Student ID: 1405013

Gazi Abdur Rakib

Student ID: 1405014

Md. Toufikuzzaman

Student ID: 1405015

Sadik Ahammed Siddique

Student ID: 1405016

Department of Computer Science and Engineering

Bangladesh University of Engineering and Technology

July 31, 2018

Contents

1 Introduction

In this assignment, we constructed a 4-bit SAP(simple as possible) computer simulation. Proteus Design Suite was used to create the simulation. The 4-bit PC has few registers (Accumulator, B , Temp, MAR, MDR, IR and OUT), an ALU as processing unit, program counter, stack, 256 byte memory and a control unit which is microprogrammed. Details of each part is discussed in this report. The 4-bit PC has 25 instructions. A compiler was also designed to convert assembly code into machine code for this PC. This report contains the technical details of implementation of the 4-bit PC.

2 Instruction Set

Our 4-bit PC instruction set has 25 instructions. This instructions are listed below.

Instruction	**Description**
LDA address	ACC←Memory[Address]
STA address	Memory[Address]←ACC
MOV B, ACC	B ←ACC
MOV ACC, Immediate	ACC ←Immediate
ADC B	ACC ←ACC + B + carry
ADC address	ACC ←ACC + Memory[address] +carry
SBB address	ACC ←ACC - Memory[address] -carry
OUT	OUT ←ACC
PUSH	Pushes the content of the Accumulator to the Stack
POP	Pops off top element of stack to Accumulator
AND B	ACC ←ACC.B

Instruction	**Description**
OR address	ACC ←ACC|Memory[address]
ORI Immediate	ACC ←ACC|Immediate
NOT	ACC ←!ACC
XCHG	ACC $\Longleftrightarrow$ B [Exchanges the content of Accumulator and B]
CMP B	Accumulator will be unchanged. Set flags according to (ACC-B)
TEST B	Accumulator will be unchanged. Set flags according to (ACC.B)
ROL	ACC ←ACC<<1 ACC[LSB] ←ACC[MSB]
RCL	ACC←ACC<<1 ACC[LSB] ←carry carry ←ACC[MSB]
JMP address	Jumps to the address
JNZ address	Jumps to the address if Zero flag is not set
CLC	Clears the Carry flag
CLS	Clears the Sign flag
HLT	Halts execution
NOP	No operation

Table 1: Instruction Set

3 Block Diagram:

The block diagram we first proposed was slightly modified to overcome some problems. The final block digram is shown below. The proposed block diagram is attached with the report.

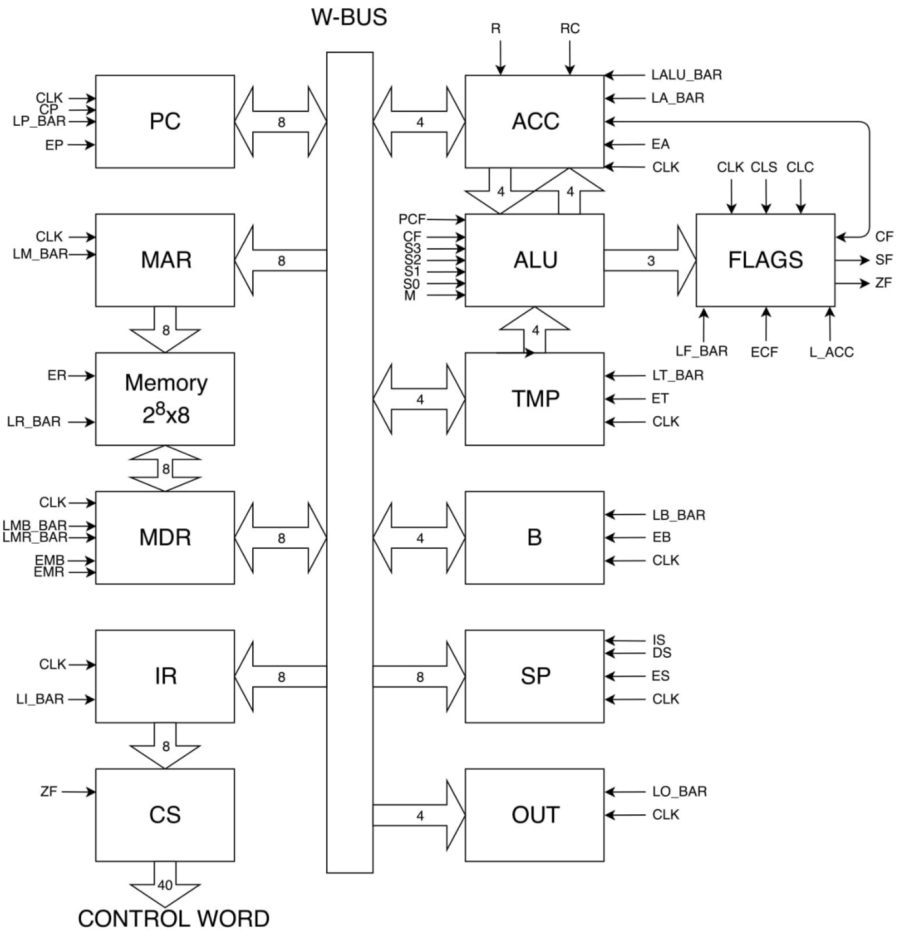

Figure 1: Block Diagram

4 Description of Blocks

4.1 Program Counter (PC)

Program Counter or PC is also known as Instruction Pointer. It points to the next instruction to be executed. PC is connected with the BUS via a bidirectional tristate buffer. Generally in fetch cycle value of PC is first loaded in Memory Address Register(MAR) and then PC is incremented by 1. But during JUMP, CALL or RET instructions PC value is loaded from MDR to PC via bus to jump to a specific address. In our 4-bit PC implementation PC is 8-bit. As PC points to the address of Memory, during execution of two byte instructions like "LDA address" PC does not necessarily point to next instruction to be executed. Rather it is pointed to the operand value. We implemented PC with a up counted which could also load data. As PC is a synchronous block, it has clock pulse as input.

Control Signals Related to PC

- **CP:** When CP is high, value of PC is incremented by 1 in the next positive clock edge.
- **EP:** When EP is high, value of PC is transferred to bus.
- **LP_BAR:** When LP_BAR is low, PC is loaded with the data present in the bus in the next positive clock edge.

4.2 Memory Address Register (MAR)

This 8-bit register holds the address of Memory to be read off or to be written to. This register is connected with the bus via tristate unidirectional buffer and it has a two state line connected with the address line of Memory. MAR is loaded with the value of PC during fetch cycle and with the value of MDR during executing memory referencing instructions. It is a synchronous block. We implemented MAR with register and tristate buffer.

Control Signals Related to MAR

- **LM_BAR:** When LM_BAR is low, MAR is loaded with the data present in the bus in the next positive clock edge.

4.3 Memory and Bootloader

Memory is commonly known as RAM. Program and data of 4-bit PC is stored in a memory which has 8-bit address and one byte is stored in each address. So the memory has total 256 bytes of capacity. Data can be read off and written to the memory. Memory is connected with the Memory Data Register via tristate bidirectional buffer. During read operation data stored in the address specified in MAR is transferred from Memory to MDR. Similarly during write operation data stored in MDR is written in the address specified by MAR of the Memory. Memory was implemented using IC 6116.

Bootloader

Before running the program memory is loaded with the program and data. This program and data is stored in a ROM. When 4-bit PC is started 256 bytes are transferred from ROM to Memory. During this period no clock pulse is sent to any other part of the 4-bit PC. A counter generates addresses from 0 to 255 and the data stored in ROM in that address is loaded in Memory. ROM is initially loaded with a .BIN file in which program is written in HEX format.

Control Signals Related to Memory

- **ER:** When ER is high, value of Memory in address specified by MAR is transferred to MDR in the next positive clock edge.
- **LR_BAR:** When LR_BAR is low, the address of Memory specified by MAR is written with the value of MDR at the end of the clock cycle. Memory is asynchronous but MDR is synchronous that's why a clock cycle is needed for read and write operations.

4.4 Memory Data Register (MDR)

Memory Data Register is a 8-bit register. It acts as a bridge between Memory and bus. Read and write operation of Memory has been discussed above in memory block. During loading data in a register from Memory MDR reads data from Memory and sends the data in bus. Storing data is the opposite process. MDR is connected with the bus via tristate bidirectional buffer. MDR is a synchronous block.

Control Signals Related to MDR

- **EMB:** When EMB is high, value of MDR is transferred to bus.
- **LMB_BAR:** When LMB_BAR is low, MDR is loaded with value of the bus in the next positive clock edge.
- **EMR:** When EMR is high, value of MDR is transferred to Memory[MAR].
- **LMR_BAR:** When LMR_BAR is low, MDR is loaded with value of the Memory[MAR] in the next positive clock edge.

4.5 Instruction Register (IR)

This is a 8-bit register which holds the opcode of current instruction which is being executed. This opcode is used by the controller to produce control signals required to execute the instruction. IR is connected with the bus via unidirectional tristate buffer. It is also connected with the controller's address line via unidirectional two state line.

Control Signals Related to Memory

- **LI_BAR:** When LI_BAR is low, IR is loaded with the data in bus in the next positive clock edge.

4.6 Controller (CS)

Controller is also known as Sequencer. It's task is to generate necessary control signals during the execution of an instruction. There are four ways to design a controller. The controller in our 4-bit PC is microprogrammed. Each control word is 40-bit long among them 2 bits are reserved. There are total 71 control words in our 4-bit PC. These control words are stored in 5 ROMs. All 5 ROMs are connected to the same address line. Every instruction in 4-bit PC has similar fetch cycle. So first 4 words are dedicated to fetch cycle in the controller. Instruction specific control routines are loaded using opcode from IR. These opcodes are designed in such a way that they represent the first address of their respective control routine. Along with control signals there are 2 bits to generate the address of the next control word.

Operation of 2 address generating bits

IC1	IC0	Description
0	0	Go to address 0x00
0	1	Go to next address
1	0	If ZF = 1 go to address 0x00 otherwise go to the first address of jump micro-routine
1	1	Go to the address specified in IR

Table 2: Summary of address generating bits

4.7 Accumulator (ACC)

Accumulator is a 4-bit data register. Result after ALU operation is stored in ACC. ACC has bidirectional tristate connection with bus. It has direct outgoing line to ALU. It is the operand A of ALU. Another incoming tristate line from ALU is used to load result in ACC after ALU operation. This line is tristate because all operations in ALU does not store the result in ACC. ACC has connection with carry flag to execute rotate with carry left (RCL) instruction. To enable shifting ACC has been designed as a shift register.

Control Signals Related to ACC

- **EA:** When EA is high, value of ACC is transferred to bus.
- **LA_BAR:** When LA_BAR is low, ACC is loaded with the value of the bus in the next positive clock edge.
- **R:** When R is high, execute ROL in the next positive clock edge.
- **RC:** When R is high, execute RCL in the next positive clock edge.
- **LALU_BAR:** When LALU_BAR is low, ACC is loaded with value of the ALU output in the next positive clock edge.

4.8 Register B and Temporary Register TMP

Register B and TMP are 4-bit data registers. B is used to store data during program execution. TMP acts as temporary register. Both registers have bidirectional tristate connection with bus like ACC. TMP has a unidirectional two state line connected to the input B of ALU.

Control Signals Related to B and TMP

- **EB:** When EB is high, value of B is transferred to bus.
- **ET:** When ET is high, value of TMP is transferred to bus.
- **LB_BAR:** When LB_BAR is low, B is loaded with the value of the bus in the next positive clock edge.
- **LT_BAR:** When LT_BAR is low, TMP is loaded with the value of the bus in the next positive clock edge.

4.9 Arithmetic Logic Unit (ALU)

This is the data processing unit of the 4-bit PC. We used 74LS181 ALU for arithmetic and logical operations. We configured the IC to meet the need of our instruction set. Input A and B of ALU come from ACC and TMP respectively. The result is sent to ACC to be stored. FLAGS register is updated after each ALU operation. This block is asynchronous. ALU takes the carry flag as input from the FLAGS register.

Control Signals Related to ALU

- **M:** Arithmetic or logical mode selector.
- **S3S2S1S0:** Operation selector.
- **PCF:** Controls carry input of ALU. Carry input is the function of PCF and CF.

ALU Operation Summary of our instruction set

Instruction	M	PCF	S3	S2	S1	S0
ADC	0	1	1	0	0	1
SBB	0	1	0	1	1	0
CMP	0	0	0	1	1	0
AND, TEST	1	1	1	1	1	0
OR	1	1	1	0	1	1
NOT	1	1	0	0	0	0

Table 3: ALU operation summary

4.10 FLAGS register

Zero flag, sign flag and carry flag are the three bits of this register. After each ALU operation these flags are updated. Carry flag is also updated after rotate with carry operation. Zero flag is used by controller to determine if jump should be taken during the execution of JNZ (jump not zero) instruction.

Control Signals Related to FLAGS

- **LF_BAR:** When LF_BAR is low, load flags according to ALU result in next positive clock edge.
- **LF_ACC:** When LF_ACC is high, load carry flag according to CF out from ACC in the next positive clock edge.
- **ECF:** When ECF is high, send CF to ACC for RCL operation.
- **CLC:** When CLC is high, clear CF in the next positive clock edge.
- **CLS:** When CLS is high, clear SF in the next positive clock edge.

4.11 Stack Pointer (SP)

SP is implemented as an up-down counter. It always holds the address of the top of the stack. Stack grows downward. During PUSH operation stack pointer value is decremented by 1 and during POP operation it is incremented by 1. It can send data to bus via tristate buffer. This data is intended for MAR.

Control Signals Related to SP

- **IS:** When IS is high, increment SP by 1 in the next positive clock edge.
- **DS:** When DS is high, decrement SP by 1 in the next positive clock edge.
- **ES:** When ES is high, send SP value in bus.

4.12 Output Register OUT

It is a 4-bit register. It can take input from bus via unidirectional tristate buffer. When OUT instruction is executed data in ACC is loaded into OUT.

Control Signals Related to OUT

- **LO_BAR:** When LO_BAR is low, load OUT from bus in the next positive clock edge.

5 Circuit Diagram

All the circuit diagrams are attached with the report.

6 Writing and Executing Program

Program of the 4-bit PC is written in assembly language. There are mnemonics associated with each instructions which are listed in table 1. This program is written in a .TXT file. This file is then compiled using the compiler we have designed. Output of the compiler is .TXT file which contains the machine code written in HEX format. Then the output file is saved as .BIN file so that it can be loaded in bootloader. After loading the .BIN file in bootloader

clock of the 4-bit PC is started. First Memory is loaded from bootloader and then the program is executed. A sample program in Assembly format and its corresponding machine code generated by the compiler is shown below.

Assembly code	Machine Code
LDA 07	04
	07
MOV B, ACC	0E
MOV ACC, 2	0F
	02
ADC B	12
HLT	45
09	09

Table 4: A sample program

When this program is executed it adds the value of ACC and B with CF. As at the start of the program CF is 0 the value stored in ACC at the end of the program is 11.

Summary

- Write assembly program in .TXT file.
- Compile the program and save the output.TXT file as .BIN file
- Load the .BIN file in bootloader.
- Start clock.

7 Additional Feature

We have created a compiler along with the 4-bit PC. The compiler is written in java language. It takes the assembly file as input and generates machine

code. Thus it eliminates the need of hand assembling which is prone to error. The compiler and its codes are available in project's git-hub link.

8 Used IC's and their Count

IC number	Description	IC count
74LS04	Hex Inverter	7
74LS08	Quad 2 input AND gate	7
74LS21	Dual 4 input AND gate	4
74LS32	Quad 2 input OR gate	3
4075	Triple 3 input OR gate	1
4072	Dual 4 input OR gate	1
74LS386	Quad 2 input XOR gate	1
74LS157	Quad 2:1 MUX	4
74LS153	Dual 4:1 MUX	6
COUNTER_8	8-bit up-down counter	4
74LS244	4-bit Buffer with Tristate Output	23
74LS126	1-bit Quad BUS Buffer with Tristate Output	1
74LS173	Quad D-FF with Tristate Output	14
74LS181	Arithmetic logic unit IC	1
2732	EEPROM	6
6116	Static RAM	1

Table 5: IC count

9 Cycle Description of Instructions

Cycle description is listed below. All opcodes and control words are represented in HEX format.

Sequence of control signals

CP LP_BAR EP LM_BAR	ER LR_BAR LMB_BAR LMR_BAR
EMB EMR LI_BAR R	RC LA_BAR EA PCF
S3 S2 S1 S0	CIN LT_BAR LB_BAR
EB CLC CLS LF_BAR	ECF ES IS DS
LO_BAR LALU_BAR IC1 IC0	ET L_ACC RESERVED0 RESERVED1

Op Code	Macro Instruction	T state	Micro Opration	Total T State	Active Control Signals	Control Word
00	Fetch	T1	MAR<–PC	4	EP, LM_BAR	67240310D0
		T2	PC<-PC+1 MDR<-RAM[MAR]		CP, ER, LMR_BAR	DE240310D0
		T3	IR<–MDR		EMB, LI_BAR	57840310D0
		T4	CSA<–IR			57240310F0
04	LDA address	T5	MAR<–PC	9	EP, LM_BAR	67240310D0
		T6	PC<-PC+1 MDR<-RAM[MAR]		CP, ER, LMR_BAR	DE240310D0
		T7	MAR<–MDR		EMB, LM_BAR	47A40310D0
		T8	MDR<–RAM[MAR]		ER, LMR_BAR	5E240310D0
		T9	Acc<–MDR		LA_BAR, EMB	57A00310C0
09	STA address	T5	MAR<–PC	9	EP, LM_BAR	67240310D0
		T6	PC<-PC+1 MDR<-RAM[MAR]		CP, ER, LMR_BAR	DE240310D0
		T7	MAR<–MDR		EMB, LM_BAR	47A40310D0
		T8	MDR<–Acc		EA, LMB_BAR	55260310D0
		T9	RAM[MAR]<–MDR		LR_BAR, EMB	53640310C0
0E	MOV B, Acc	T5	B<–Acc	5	EA, LB_BAR	57260210C0
0F	MOV Acc, Immediate	T5	PC<–MAR	7	EP, LM_BAR	67240310D0
		T6	PC<-PC+1 MDR<-RAM[MAR]		CP, ER, LMR_BAR	DE240310D0
		T7	Acc<–MDR		EMB, LA_BAR	57A00310C0
12	ADC B	T5	TMP<–B	6	EB, LT_BAR	57240190D0
		T6	Acc<-Acc + TMP + Carry		LALU_BAR, PCF, S3, S0	5725930080

Op Code	Macro Instruction	T state	Micro Opration	Total T State	Active Control Signals	Control Word
14	ADC address	T5	MAR<–PC	10	EP, LM_BAR	67240310D0
		T6	PC<-PC+1 MDR<-RAM[MAR]		CP, ER, LMR_BAR	DE240310D0
		T7	MAR<–MDR		EMB, LM_BAR	47A40310D0
		T8	MDR<–RAM[MAR]		ER, LMR_BAR	5E240310D0
		T9	TMP<–MDR		EMB, LT_BAR	57A40110D0
		T10	Acc<–Acc + TMP + Carry		LALU_BAR, PCF, S3, S0	5725930080
1A	SBB address	T5	MAR<–PC	10	EP, LM_BAR	67240310D0
		T6	PC<-PC+1 MDR<-RAM[MAR]		CP, ER, LMR_BAR	DE240310D0
		T7	MAR<–MDR		EMB, LM_BAR	47A40310D0
		T8	MDR<–RAM[MAR]		ER, LMR_BAR	5E240310D0
		T9	TMP<–MDR		EMB, LT_BAR	57A40110D0
		T10	Acc<–Acc - TMP - Carry		LALU_BAR, PCF, S2, S1	5725630080
20	OUT	T5	OUT<–Acc	5	EA, LO_BAR	5726031040
21	PUSH	T5	SP<-SP - 1 MDR<-Acc	7	DS, EA, LMB_BAR	55260311D0
		T6	MAR<–SP		ES, LM_MAR	47240314D0
		T7	RAM[MAR]<–MDR		EMR, LR_BAR	53640310C0
24	POP	T5	MAR<–SP	7	ES, LM_BAR	47240314D0
		T6	SP<-SP+1 MDR<-RAM[MAR]		IS, ER, LMR_BAR	5E240312D0
		T7	Acc<–MDR		EMB, LA_BAR	57A00310C0
27	AND B	T5	TMP<–B	6	EB, LT_BAR	57240190D0
		T6	Acc<–Acc & TMP		LALU_BAR, M, PCF, S3, S2, S1	5725EB0080

Op Code	Macro Instruction	T state	Micro Opration	Total T State	Active Control Signals	Control Word
29	OR address	T5	MAR<–PC	10	EP, LM_BAR	67240310D0
		T6	PC<-PC+1 MDR<-RAM[MAR]		CP, ER, LMR_BAR	DE240310D0
		T7	MAR<–MDR		EMB, LM_BAR	47A40310D0
		T8	MDR<–RAM[MAR]		ER, LMR_BAR	5E240310D0
		T9	TMP<–MDR		EMB, LT_BAR	57A40110D0
		T10	Acc<–Acc — TMP		LALU_BAR, PCF, S3, S1, S0	57A4BB0080
2F	OR Immediate	T5	MAR<–PC	8	EP, LM_BAR	67240310D0
		T6	PC<-PC+1 MDR<-RAM[MAR]		CP, ER, LMR_BAR	DE240310D0
		T7	TMP<–MDR		EMB, LT_BAR	57A40110D0
		T8	Acc<–Acc — TMP		LALU_BAR, PCF, S3, S1, S0	57A4BB0080
33	NOT	T5	Acc<– !Acc	5	LALU_BAR, M, PCF	57250B0080
34	XCHG	T5	TMP<–Acc	7	EA, LT_BAR	57260110D0
		T6	Acc<–B		EB, LA_BAR	57200390D0
		T7	B<–TMP		ET, LB_BAR	57240210C8
37	CMP B	T5	TMP<–B	6	EB, LT_BAR	57240190D0
		T6	Acc - TMP		S2, S1	57246300C0
39	TEST B	T5	TMP<–B	6	EB, LT_BAR	57240190D0
		T6	Acc & TMP		M, PCF, S3, S2, S1	5725EB00C0
3B	ROL	T5	Acc<- Acc<<1 Acc[LSB]<- Acc[MSB]	5	R	57340310C0
3C	RCL	T5	Acc<-Acc<<1 Carry<- Acc[MSB] Acc[LSB]<-Carry	7	RC, ECF	572C0318D0
		T6	LOAD FLAG FROM Acc		LF_BAR,L_ACC	57240300C4

Op Code	Macro Instruction	T state	Micro Opration	Total T State	Active Control Signals	Control Word
3E	JMP address	T5	MAR<–PC	7	EP, LM_BAR	67240310D0
		T6	PC<-PC+1 MDR<-RAM[MAR]		CP, ER, LMR_BAR	DE240310D0
		T7	PC<–MDR		EMB, LP_BAR	17A40310C0
41	JNZ address	T5	MAR<–PC	7	EP, LM_BAR	67240310D0
		T6	PC<-PC+1 MDR<-RAM[MAR]		CP, ER, LMR_BAR	DE240310E0
		T7	PC<–MDR		EMB, LP_BAR	17A40310C0
43	CLC	T5	Clears the Carry flag	5	CLC, LF_BAR	57240340C0
44	CLS	T5	Clears Sign flag	5	CLS, LF_BAR	57240320C0
45	HLT	T5	Halts execution	5		57240310F0
46	NOP	T5	No operation	5		57240310C0

10 Timing Diagram

Timing Diagram of Fetch cycle

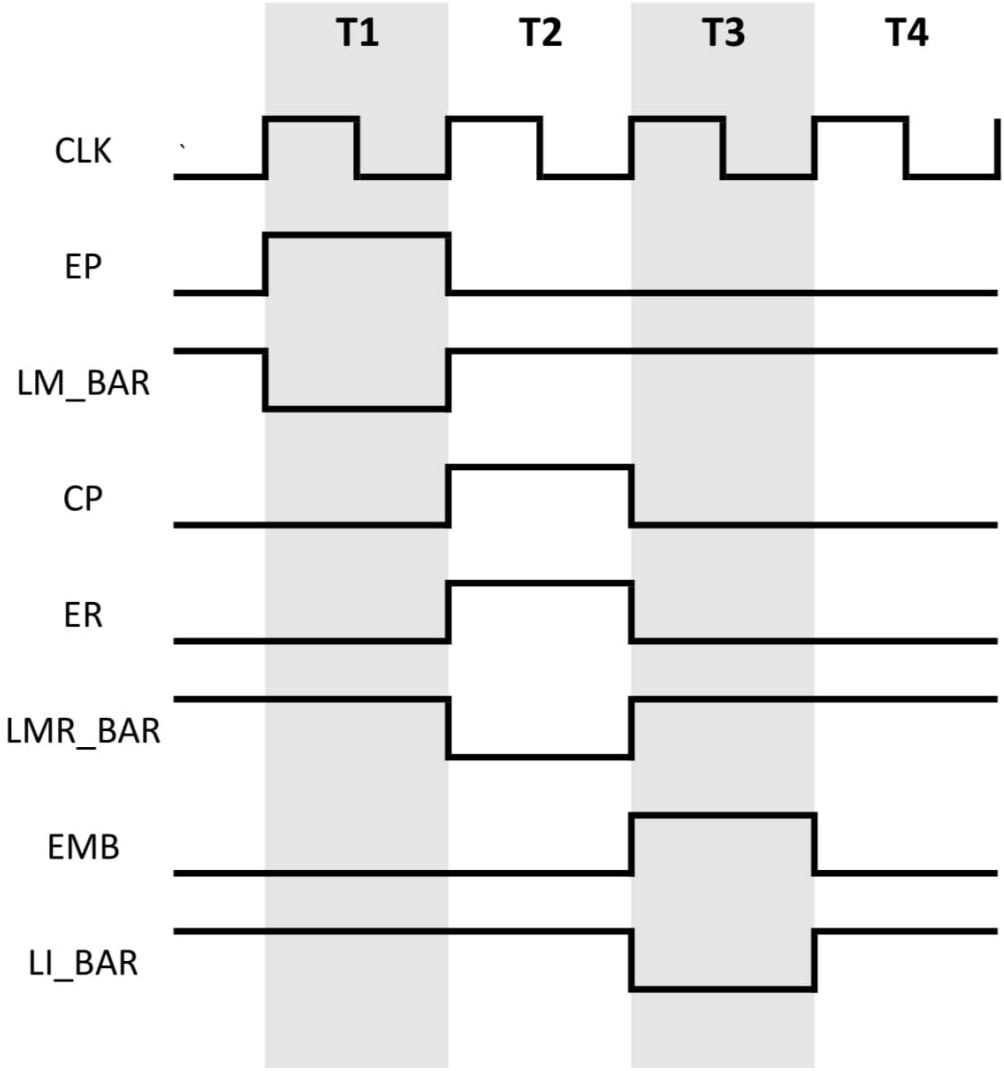

Figure 2: Fetch Cycle

Timing Diagram of ADC address Instruction

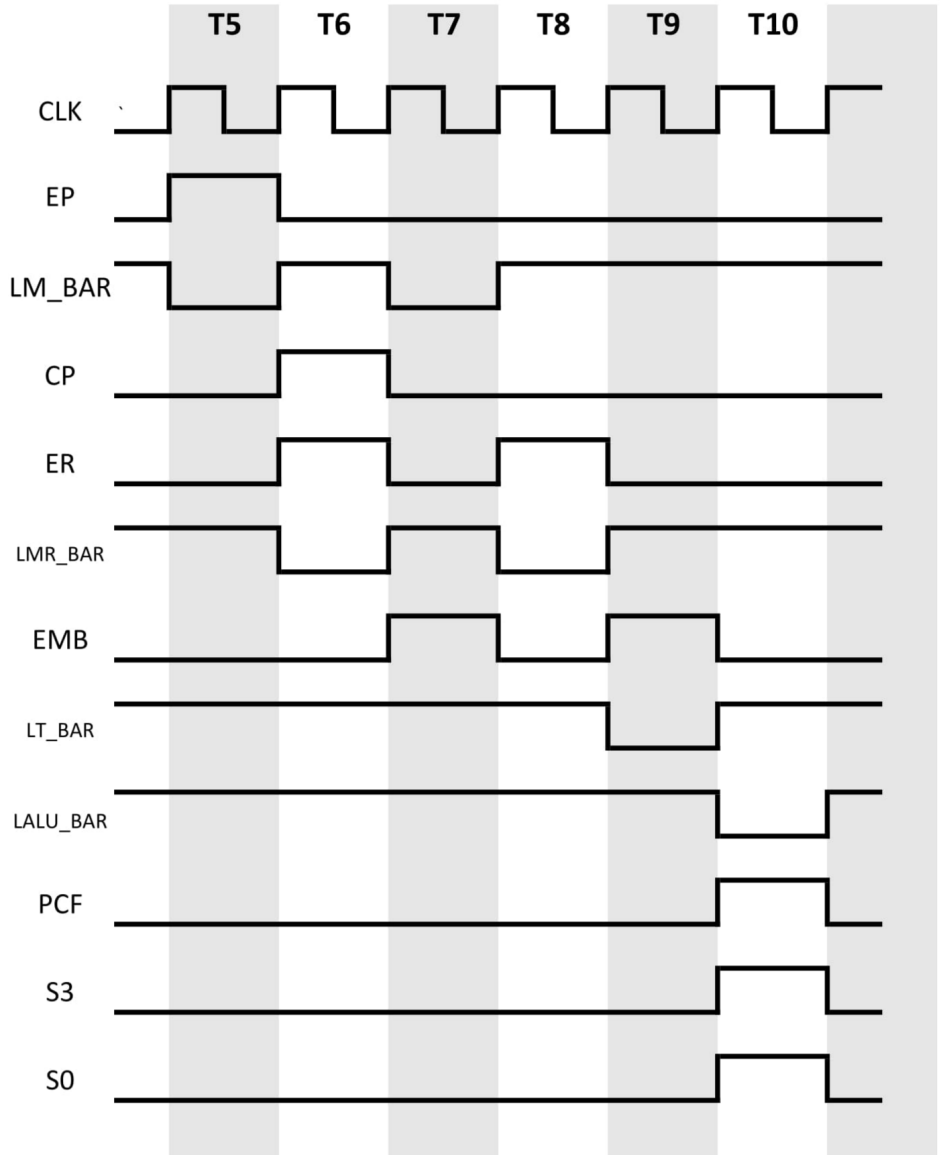

Figure 3: ADC address

Timing Diagram of OR Immediate Instruction

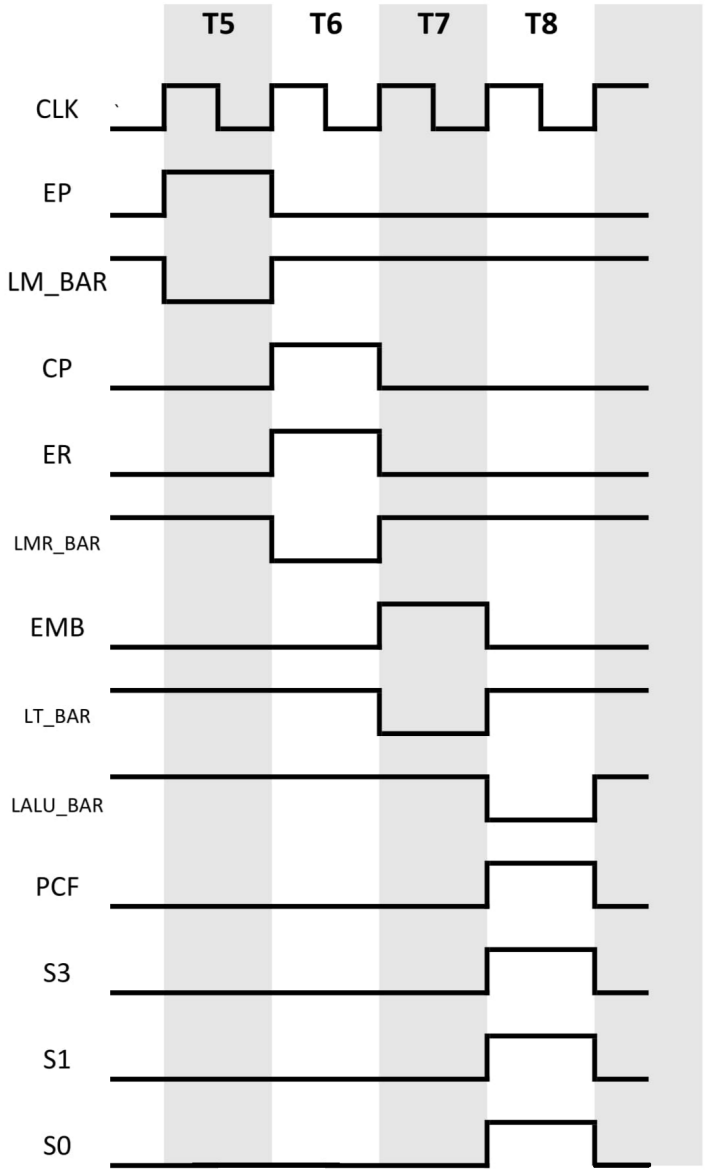

Figure 4: OR Immediate

Timing Diagram of PUSH Instruction

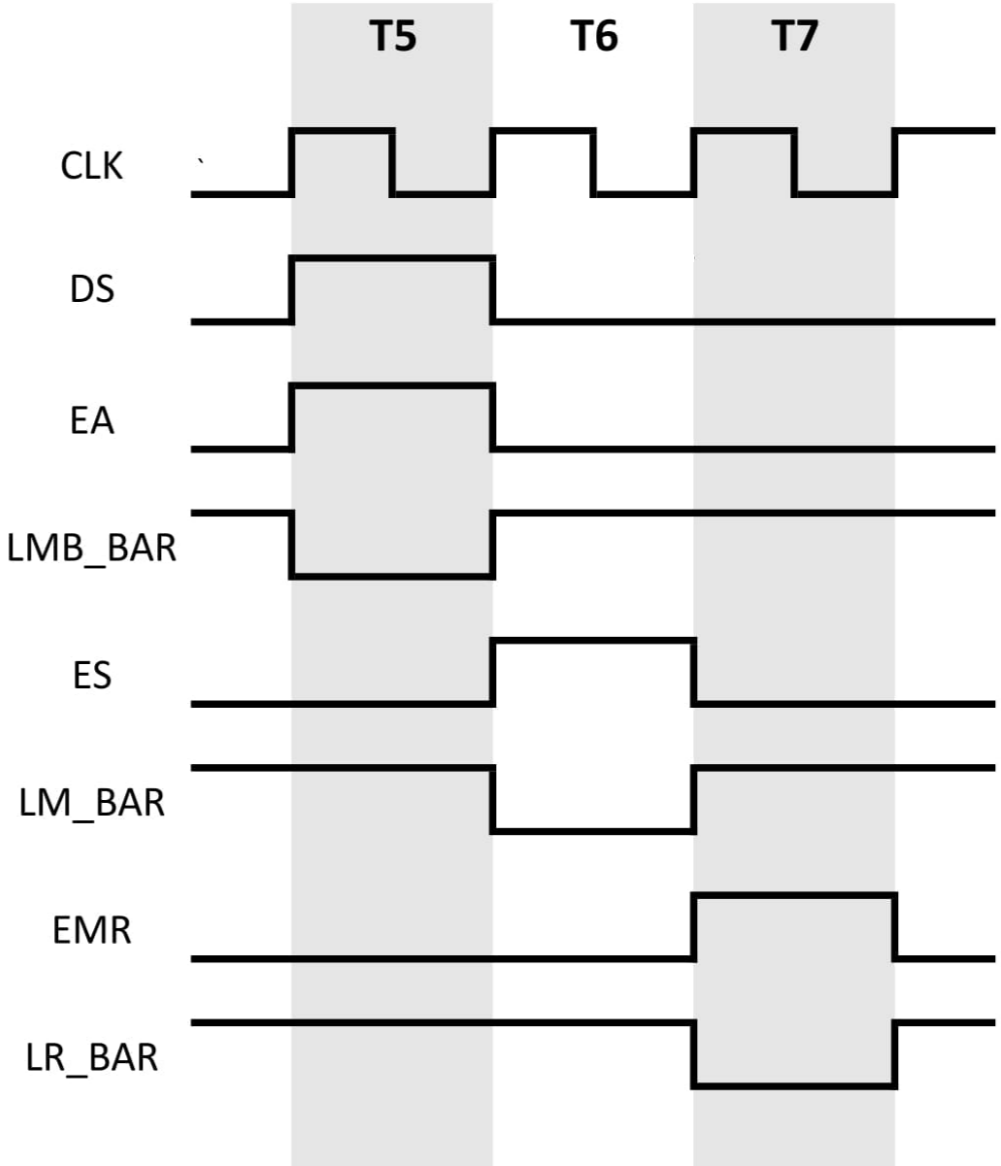

Figure 5: PUSH

Timing Diagram of XCHG Instruction

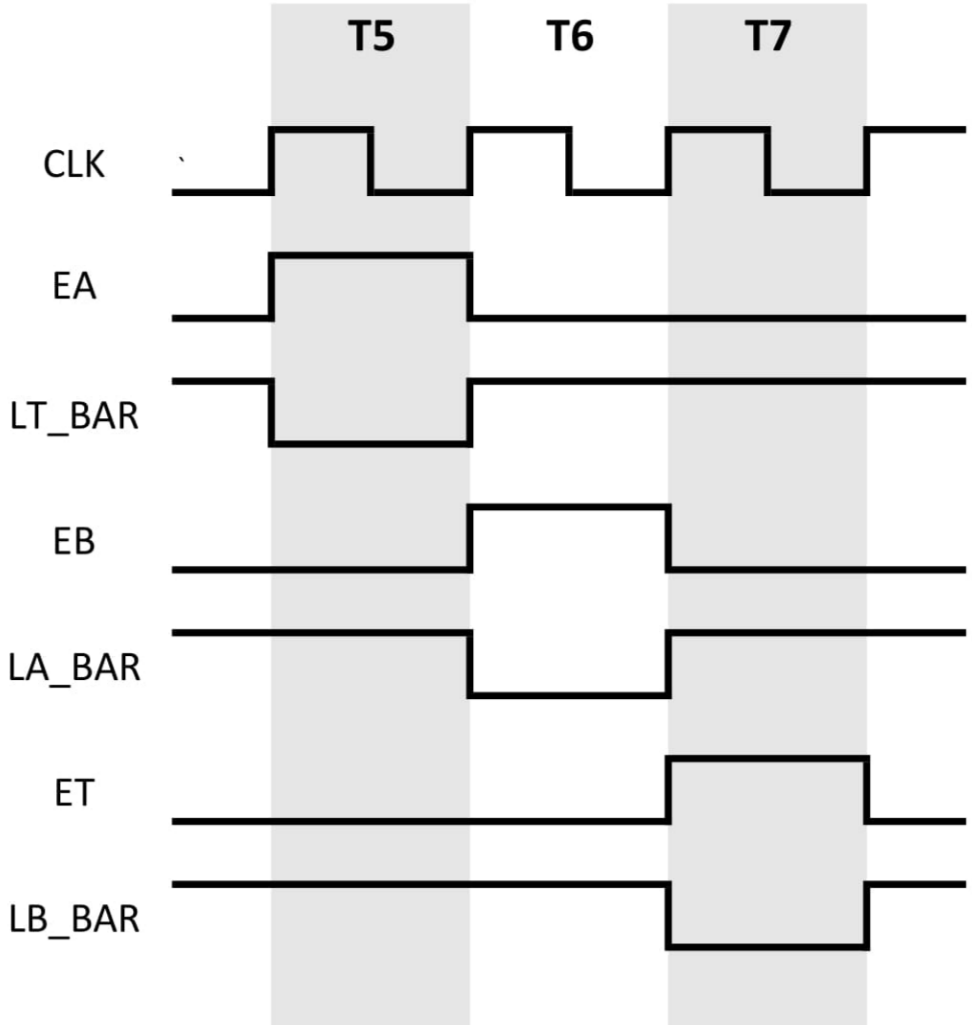

Figure 6: XCHG

Timing Diagram of JMP Instruction

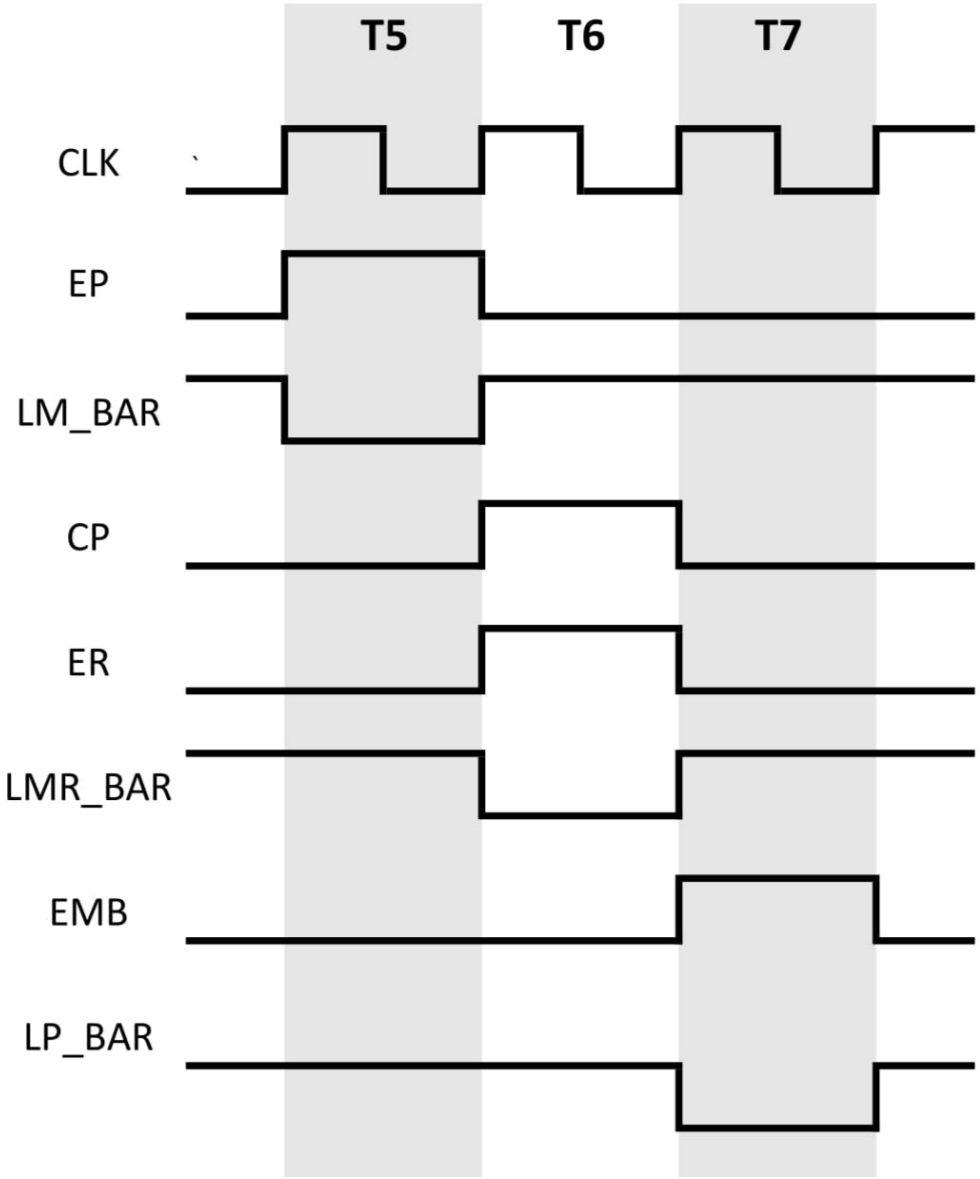

Figure 7: JMP

11 Discussion

While completing this project we faced a lot of problems. Some issues are discussed here.

We used Proteus Design Suite to implement the 4-bit PC. Initially all of the group members did not have same version installed in their personal computer. Thats why merging took some time. We were also stalled by the lack of shortcuts in the software.

Initially we proposed a block diagram and our lab teachers verified that and made some slight modifications. Later we had to modify the diagram again to overcome some problems. Initial diagram is attached with the report and final diagram is shown in the report. The first modification was making TMP register bidirectional for executing XCHG instruction.Second modification was adding the control signal LF_ACC which is used to change carry flag after RCL operation in ACC. Third modification was renaming EALU to PCF. Like our lab teacher suggested we directly connected ALU with ACC rather than connecting ALU with bus. Thus EALU was no longer needer. We used this bit as PCF to control carry input which is the function of CF and PCF.

We faced problems in loading program in Memory from bootloader. First we were sending constant 0 in WR_BAR of RAM chip during loading data from ROM. Nothing was being loaded. Then we changed WR_BAR to clock pulse during loading. That solved most of the problems except loading in address zero. Later we loaded the counter (address generator for ROM) with 0x00 and after a clock pulse started counting up. Initially we started counting without loading counter with 0x00 which caused the problem.

Most of our trouble was caused by control signals. We were loading the first address of control routine of the execution cycle of the instruction before loading the opcode in IR. That's why wrong control signals were being generated. In few instructions we loaded the ALU before data was available. In both cases we had to add an extra clock cycle which solved the problem but finding the source of these problems were very challenging.

12 Conclusion

This project was very educative and challenging at the same time. We had to put a lot of effort and team work to complete the 4-bit PC. This project gave us better insight of the working principle of computer. It helped us to better understand the topic which we are taught in class. Due to the lack of tutorial on this project it is little difficult to get going.

GitHub repository of project

https://github.com/Rafid013/4-Bit-PC