CSE 404: Digital System Design Sessional

4-Bit PC Design & Simulation

Report

Submitted by:

Section: B1

Group: G5

Muhammad Azmain Adel (1405075)

Muhtasim Ulfat (1405086)

Maksudul Alam (1405087)

Sazzad Hossain (1405089)

Sadman Yasar Ridit (1405090)

Introduction

In this report, the design and simulation of a 4-Bit PC we designed for the course CSE 404: Digital System Design Sessional has been discussed.

We had to implement 25 instructions for completing the assignment on 4-bit PC. For designing the circuits and running simulations, Proteus 8.6 has been used.

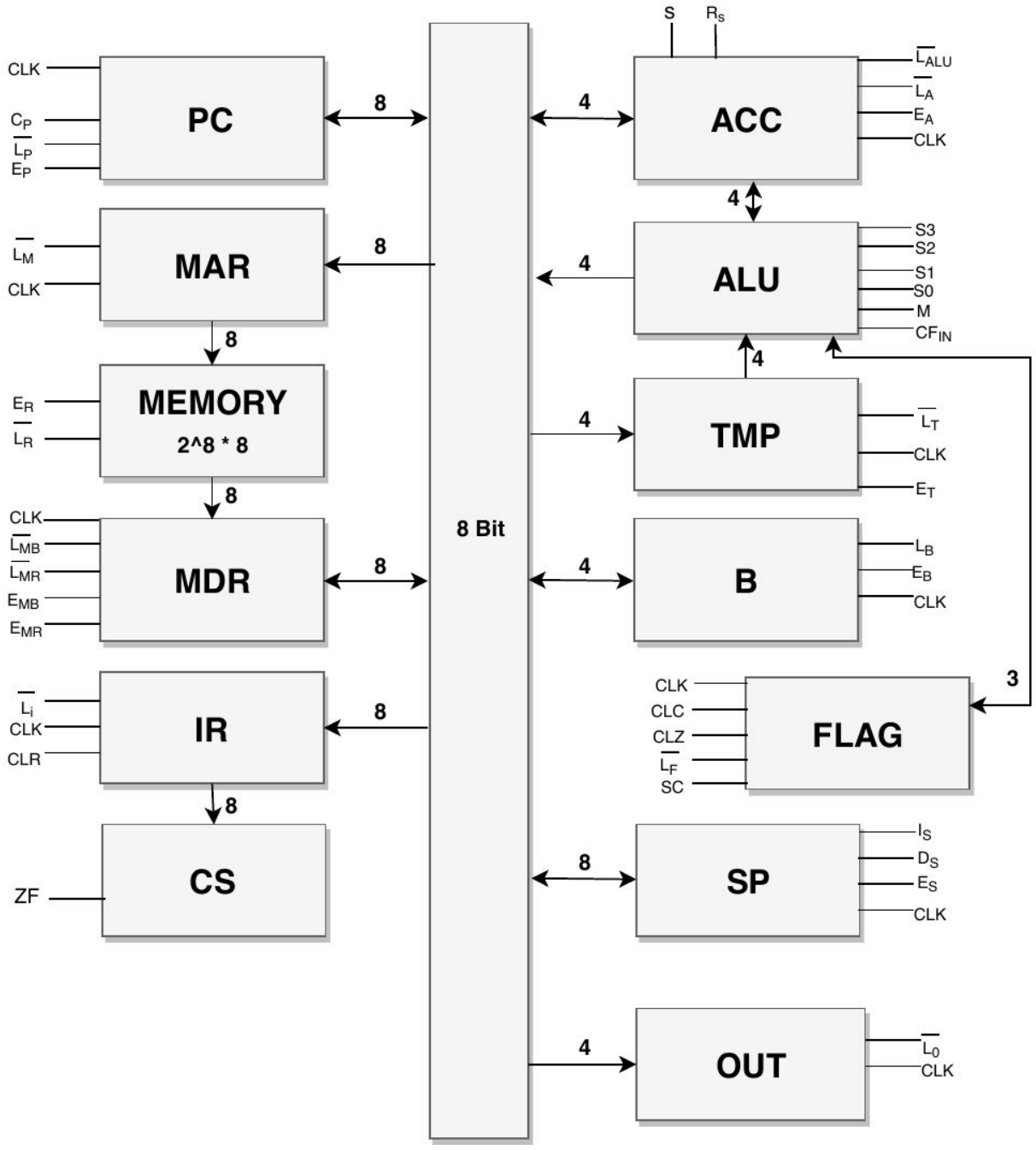
The 4-bit PC is consists with ALU (arithmetic logic unit), Controller-Sequencer, Memory, Accumulator (ACC), Memory Address Register (MAR), Memory Data Register (MDR), Stack Pointer (SP), Program Counter (PC), B, TEMP, FLAG, Input Port and Output Port. As per the requirement of implementation, data bus is 4 bit and address bus is 8 bit, so an 8 bit W-Bus is used.

Instruction Set

No	Mnemonic	Description
1	LDA address	Acc ← Memory [address]
2	STA address	Memory [address] ← Acc
3	ADD B	Acc ← Acc + B
4	ADC B	Acc ← Acc + B + C (Contents of Carry Flag)
5	ADD address	Acc ← Acc + Memory [address]
6	OR address	Acc ← Acc | Memory [address]
7	XOR address	Acc ← Acc ⊕ Memory [address]
8	NOT	Acc ← !Acc
9	NEG	Acc ← -Acc
10	TEST B	Set flags according to (Acc.B)
11	SBB immediate	Acc ← Acc – Immediate – Carry

12	INC	Acc ← Acc + 1
13	DEC	Acc ← Acc - 1
14	OUT	Output_port ← Acc
15	AND B	Acc ← Acc.B
16	SHL	Acc ← Acc << 1, Carry ← Acc[MSB], Acc[LSB] ← 0
17	ROR	Acc ← Acc >> 1, Carry ← Acc[LSB], Acc[MSB] ← Acc[LSB]
18	RCR	Acc ← Acc >> 1, Acc[MSB] ←Carry, Carry ← Acc[LSB]
19	JMP address	Jumps to the address
20	JNZ address	Jumps to the address if Zero flag is not set
21	CALL address	Calls a subroutine (at the specified address) unconditionally
22	RET	Returns from current subroutine to the caller unconditionally
23	CLC	Clears the Carry flag
24	CLZ	Clears the Zero flag
25	HLT	Halts execution

Block Diagram



Brief Description of Blocks

Program Counter (PC):

Associated Control Pins: CLK , C_P , $\overline{L_P}$, E_P

The Program Counter (PC) is an 8-bit register structure that contains the address pointer value of the current instruction. After each instruction cycle the PC needs to be updated to point to the next instruction in memory. Generally, instructions are fetched sequentially in memory. Most of the time PC is updated just by incrementing. But some instructions like CALL address, JMP address place a new value in the PC from where instructions are fetched.

In SAP-1 architecture, there was no jump or call instructions, so it was enough just to send the output of PC register to the bus (W0-W7 line). But since we need to execute instructions like JUMP and CALL. we also need to load address to PC. This is also done by interacting with bus (W0-W7 line).

The $\overline{L_P}$ control pin is used to load the address of the instruction to PC.

E_P PC register passes it's output to the bus.

C_P is used to increment the PC to fetch the next instruction.

Memory Address Register (MAR):

Associated Control Pins: CLK , $\overline{L_M}$

Memory Address Register is an 8-bit register that takes 8-bit address as input from PC through W-bus (W0-W7 line) and sends this address to the memory directly (MO0-MO7 line) to read from or write into that address of memory.

When $\overline{L_M}$ is low, address is loaded to MAR through W-bus and when $\overline{L_M}$ is high, the address is passed to the Memory.

Memory:

Associated Control Pins: E_R , $\overline{L_R}$

Memory has two parts: RAM and Bootloader.

1. RAM:

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.

When E_R is high, value of Memory in address specified by MAR is transferred to MDR in the next positive clock edge.

When $\overline{L_R}$ is low, the address of Memory specified by MAR is written with the value of MDR at the end of the clock cycle.

2. 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.

Memory Data Register (MDR):

Associated Control Pins: CLK , $\overline{L_{MB}}$, $\overline{L_{MR}}$, E_{MR} , E_{MB}

MDR is an 8-bit register that can read from or write into memory through data line RM0-RM7. It can also load data from W-bus and send data to W-bus through W0-W7 line.

When E_{MB} is high, value of MDR is transferred to bus.

When $\overline{L_{MB}}$ is low, MDR is loaded with value of the bus in the next positive clock edge.

When E_{MR} is high, value of MDR is transferred to Memory[MAR].

When $\overline{L_{MR}}$ is low, MDR is loaded with value of the Memory[MAR] in the next positive clock edge.

Instruction Register (IR):

Associated Control Pins: CLR , CLK , $\overline{L_I}$

Instruction Register is an 8-bit register, During Fetch cycle, IR is loaded with 8-bit op-code of an instruction through W-bus (W0-W7 line) from MDR. Since, our 4-bit PC can execute 28 instructions, we only need 5 bits. So, the most significant 3 bits are redundant. IR register then sends the least significant 5 bits of the op-code to Control register using the CON0-C0N4 line.

When $\overline{L_I}$ is low, IR is loaded with the data in bus in the next positive clock edge.

Controller-Sequencer (CS):

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.

When **IC1 = 0 and IC0 = 0**: Go to address 0x00

When **IC1 = 0 and IC0 = 1**: Go to next address

When **IC1 = 1 and IC0 = 0**: If ZF = 0, go to address 0x00

Else, go to first address of jump micro-routine

When **IC1 = 1 and IC0 = 1**: Go to address of IR

Accumulator (ACC):

Associated Control Pins: CLK, $\overline{L_A}$, S, E_A, R_S, $\overline{L_{ALU}}$

This is a 4-bit register that can load data from W-bus or send data to W-bus using W0-W3 line. It also passes data to Arithmetic Logic Unit (ALU) through A0-A3 line to perform different arithmetic and logic operations.

When E_A is high, value of ACC is transferred to the bus.

When $\overline{L_A}$ is low, ACC is loaded with the value of the bus in the next positive clock edge.

When S is high, execute Shift in the next positive clock edge.

When R_S is high, execute ROR in the next positive clock edge.

When $\overline{L_{ALU}}$ is low, ACC is loaded with value of the ALU output in the next positive clock edge.

Arithmetic Logic Unit (ALU):

Associated Control Pins: $S0$, $S1$, $S2$, $S3$, M, CF_{IN}

This unit takes data from Accumulator register through A0-A3 line and from Temp Register through T0-T3 line and performs various arithmetic and logic operation.

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.

M : Arithmetic or logical mode selector.

$S0$, $S1$, $S2$, $S3$: Operation selector.

CF_{IN} : Controls carry input of ALU.

$S3$	$S2$	$S1$	$S0$	CF_{IN}	M	Function
1	1	1	1	1	0	Inc
1	1	1	1	0	0	Transfer
1	0	0	1	0	0	A+B
1	0	0	1	Carry	0	A+B+Cin
0	1	1	0	1	0	A-B
0	1	1	0	0	0	A-B-1
0	0	0	0	0	0	Dec
1	0	0	1	x	1	XOR
0	0	0	0	x	1	NOR
1	0	1	1	x	1	OR

Temporary Register (TMP):

Associated Control Pins: CLK , $\overline{L_T}$, E_T

Temporary Register is an 8-bit register as it needs to store 4-bit data or 8-bit address, whichever is necessary for a certain operation. This register can load data from W-bus or send data to W-bus using W0-W7 line. It also passes data to Arithmetic Logic Unit (ALU) through T0-T3 line (data is always 4-bit) continuously to perform different arithmetic and logic operations.

When $\overline{L_T}$ is low, TMP is loaded with the value of the bus in the next positive clock edge.

E_T enables the register.

B Register (B):

Associated Control Pins: CLK , E_B , L_B

This is a 4-bit register that can load data from W-bus or send data to W-bus using W0-W3 line.

When E_B is high, value of B is transferred to bus.

When L_B is high, B is loaded with the value of the bus in the next positive clock edge.

Stack Pointer (SP):

Associated Control Pins: CLK , I_S , D_S , E_S

Stack Pointer is an 8-bit register that stores the address of the last program request in a stack. Initially after boot loading process is complete, the Stack Pointer points to FF-the last memory location.

When I_S is high, increment SP by 1 in the next positive clock edge.

When D_S is high, decrement SP by 1 in the next positive clock edge.

When E_S is high, send SP value in bus.

Flags Register (FLAG):

Associated Control Pins: CLK , CLC , CLZ , SC , $\overline{L_F}$

Zero flag (ZF), Sign flag (SF) and Carry flag (CF) 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.

When $\overline{L_F}$ is low, load flags according to ALU result in next positive clock edge.

When SC is high, send CF to ACC for RCL operation.

When CLC is high, clear CF in the next positive clock edge.

When CLZ is high, clear ZF in the next positive clock edge.

Output Register (OUT):

Associated Control Pins: CLK , $\overline{L_O}$

This is a 4-bit register that can load data from W-bus through W0-W3 line, so that it can show result of different operations.

When $\overline{L_O}$ is low, load OUT from bus in the next positive clock edge.

Circuit Diagram

All the circuit diagrams are attached with the report.

Cycle Description

Micro Instruction	Opcode	Total T States	T States	Micro Operation	Control Word
FETCH	00	4	T1	MAR ← PC	C72AF80140
			T2	PC← PC+1, MDR ← RAM[MAR]	BE2AF80140
			T3	IR ← MDR	978AF80140
			T4	LOAD from IR	972AF801C0
LDA	04	9	T5	MAR ← PC	C72AF80140
			T6	PC← PC+1, MDR ← RAM[MAR]	BE2AF80140
			T7	MAR ← MDR	87AAF80140
			T8	MDR ← RAM[MAR]	9E2AF80140
			T9	ACC ← MDR	97A2F80100
STA	09	9	T5	MAR ← PC	C72AF80140
			T6	PC← PC+1, MDR ← RAM[MAR]	BE2AF80140
			T7	MAR ← MDR	87AAF80140
			T8	MDR ← ACC	953AF80140
			T9	RAM[MAR] ← MDR	936AF80100
DEC	0E	5	T5	ACC ← ACC - 1	9728080100
INC	0F	6	T5	ACU ← ACC + 1	972AF80340
			T6	ACC ← ACU	9728F80300
CLC	11	5	T5	CF ← 0	972AF81100
NOT	12	6	T5	ACU ← ~ACC	972B080140
			T6	ACC ← ACU	9729080100
SHIFT	14	5	T5	ACC ← ACC << 1	972E080100
MOV B, Address	15	9	T5	MAR ← PC	C72AF80140

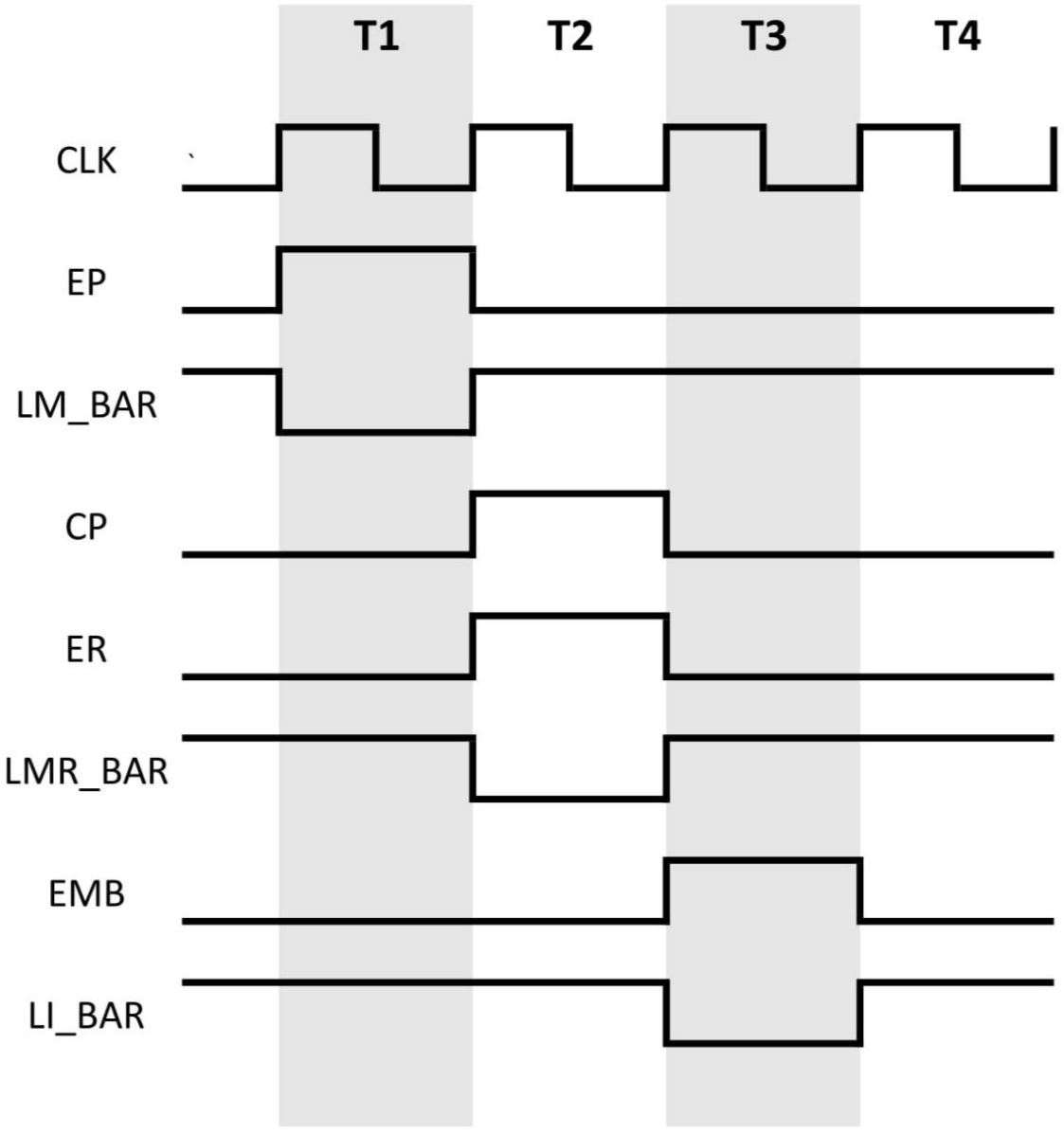
			T6	MDR ← RAM[MAR]	BE2AF80140
			T7	MAR ← MDR	87AAF80140
			T8	MDR ← RAM[MAR]	932AF80140
			T9	B ← MDR	97AAFA0100
ADD B	1A	6	T5	TEMP ← B	972AF10140
			T6	ACC ← ACC + TEMP	9728980100
ADC B	1C	7	T5	TEMP ← B	972AF10140
			T6	CF ← 1	9728980340
			T7	ACC ← ACC + TEMP +CF	9728F90100
ADD Address	1F	10	T5	MAR ← PC	C72AF80140
			T6	MDR ← RAM[MAR]	BE2AF80140
			T7	MAR ← MDR	87AAF80140
			T8	MDR ← RAM[MAR]	9E2AF80140
			T9	TEMP ← MDR	97AAF00140
			T10	ACC ← ACC + TEMP	9728980100
OR Address	25	10	T5	MAR ← PC	C72AF80140
			T6	MDR ← RAM[MAR]	BE2AF80140
			T7	MAR ← MDR	87AAF80140
			T8	MDR ← RAM[MAR]	9E2AF80140
			T9	TEMP ← MDR	97AAF00140
			T10	ACC ← ACC | TEMP	9729B80100
XOR Address	2B	10	T5	MAR ← PC	C72AF80140
			T6	MDR ← RAM[MAR]	BE2AF80140
			T7	MAR ← MDR	87AAF80140
			T8	MDR ← RAM[MAR]	9E2AF80140
			T9	TEMP ← MDR	97AA900140
			T10	ACC ← ACC ⊕ TEMP	9729980100
AND B	31	6	T5	TEMP ← B	972AF10140
			T6	ACC ← ACC.TEMP	9729E80100

SBB Immediate	33	8	T5	MAR ← PC	C72AF80140
			T6	PC ← PC + 1, MDR ← RAM[MAR]	BE2AF80140
			T7	TEMP ← MDR	97AAF00140
			T8	ACC ← ACC - TEMP - CF	9728600100
OUT	37	5	T5	OUT ← ACC	973AF80000
CLZ	38	5	T5	ZF ← 0	972AF80900
NEG	39	8	T5	M ← -1	972B080140
			T6	ACC ← ~ACC	9729080140
			T7	SC ← 1	972AF80340
			T8	ACC ← ACC + 1	9728F80300
ROR	3D	6	T5	ALU ← ROTATE ACC	972AF80160
			T6	ACC ← ALU	972AF80100
TEST B	3F	6	T5	TEMP ← B	972AF10140
			T6	ACC.TEMP	972BE80100
JMP Address	41	7	T5	MAR ← PC	C72AF80140
			T6	PC ← PC + 1, MDR ← RAM[MAR]	BE2AF80140
			T7	PC ← MDR	17AAF80100
CALL Address	44	13	T5	MAR ← PC	C72AF84040
			T6	PC ← PC + 1, MDR ← RAM[MAR]	BE2AF80140
			T7	TEMP ← MDR	97AAF00140
			T8	SP ← SP - 1	782AF82140
			T9	MAR ← SP	972AF80140
			T10	MDR ← PC	D52AF80140
			T11	RAM[MAR] ← MDR	936AF80140
			T12	PC ← TEMP	172AFC0140
			T13	TEST	9E2AF80100

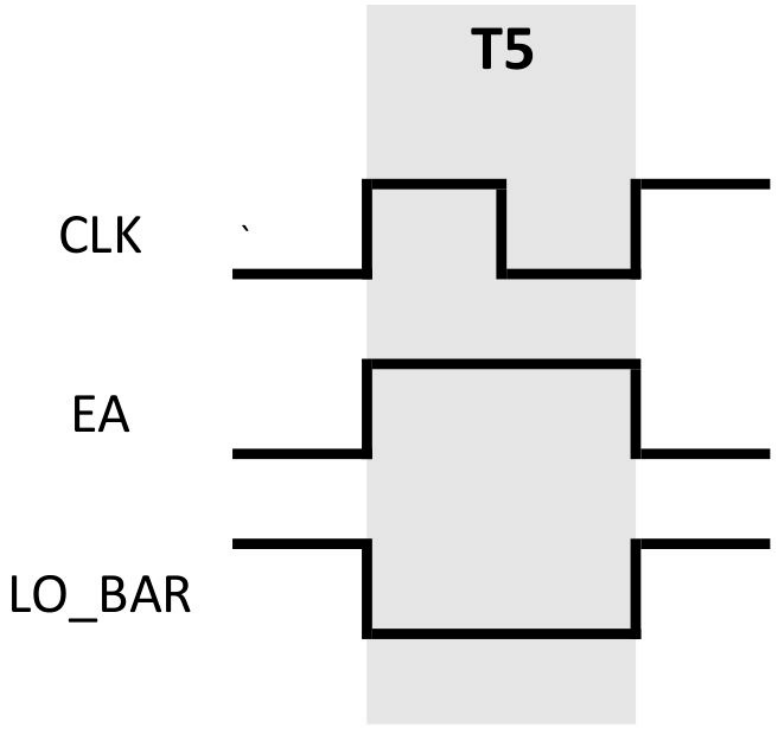
RET	4D	8	T5	MAR ← SP	872AF82140
			T6	MDR ← RAM[MAR]	972AF88140
			T7	SP ← SP + 1	9E2AF80140
			T8	PC ← MDR	17AAF80100

Timing Diagram

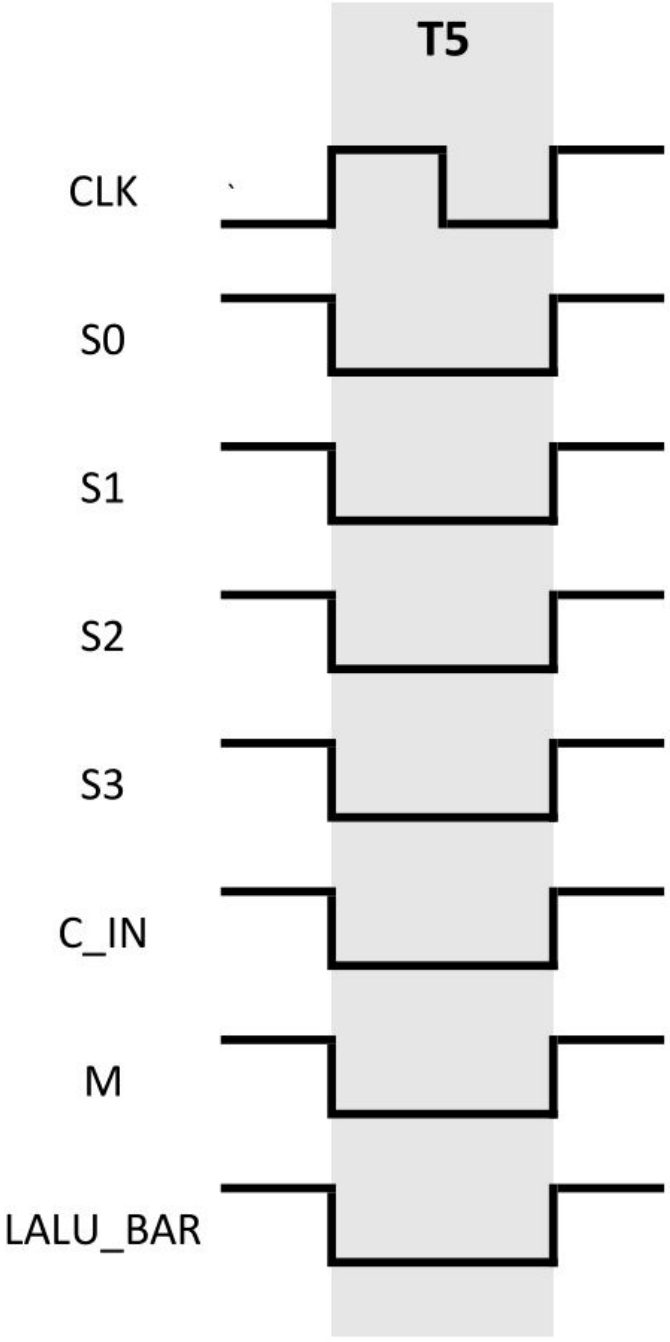
Instruction: **FETCH**



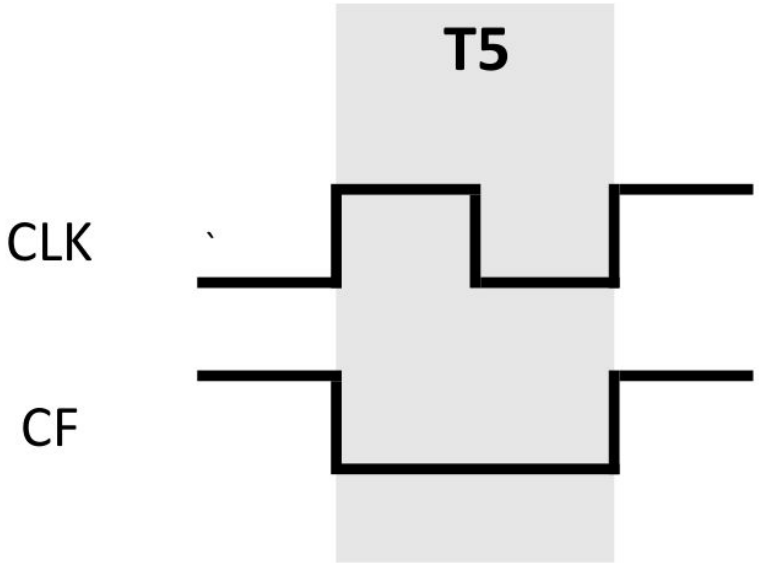
Instruction: **OUT**



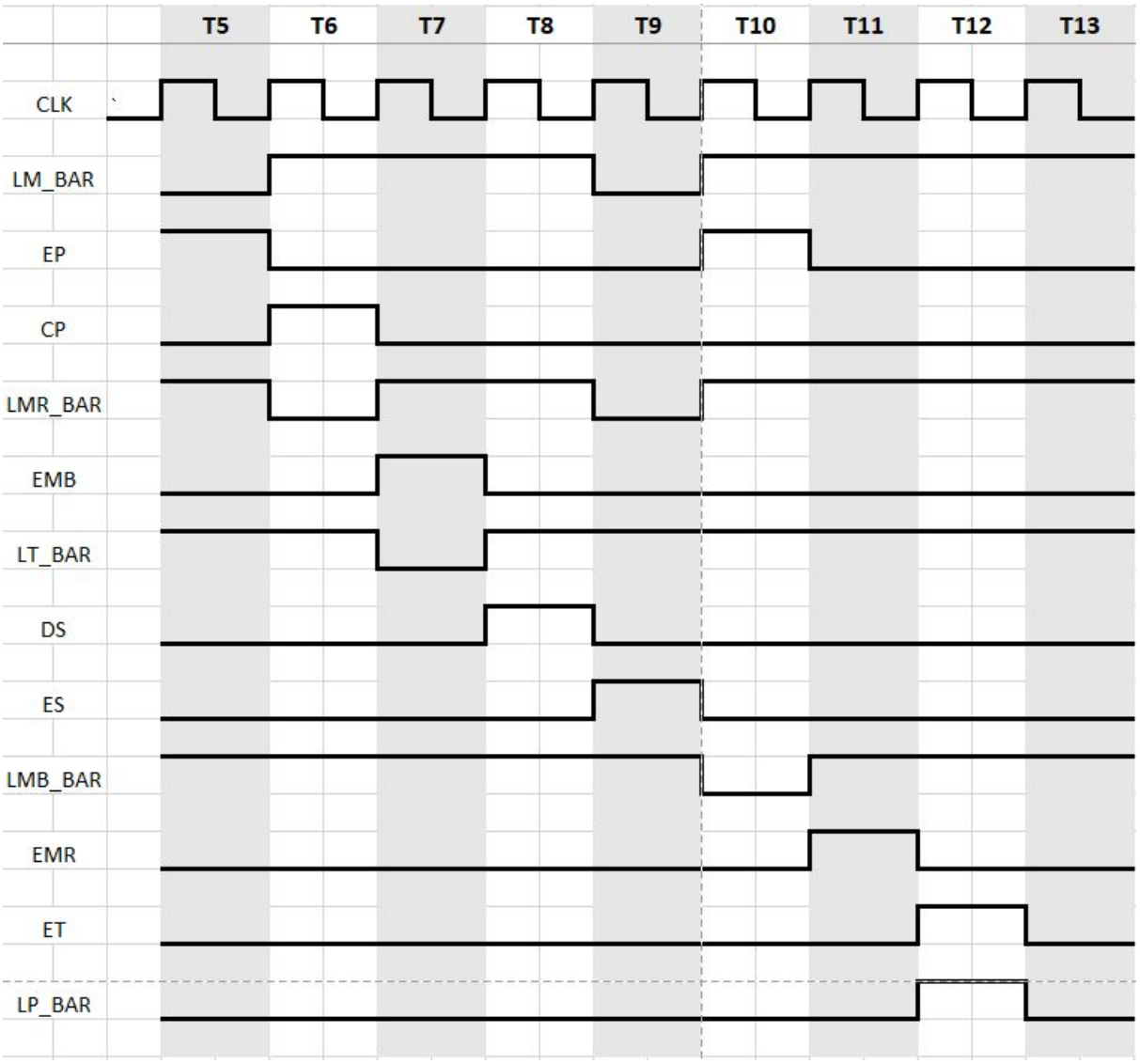
Instruction: **DEC**



Instruction: **CLC**



Instruction: **CALL Address**



Running a Program

Writing

We convert the mnemonic form to hex code. This is done by merging the hex opcode of the instruction and the hex value of the operand (address or immediate value, if exists).

Each instruction will become a hex value representing 1 or 2 bytes of information, given whether they use address or immediate value operands or not.

Next, we arrange the hex values in a BIN file line by line such that each line has a two-digit hex code (1 byte).

Executing

We store these hex codes in a BIN file. We add, FF at the final line to denote end of FILE.

We load this BIN file in the program ROM. When the PC starts, during the boot loader phase, each of these instructions from the program ROM is loaded into the RAM, afterwards during the fetch cycle, OP is fetched from the RAM and it is eventually sent to the instruction register and the execution phase starts.

Used IC

IC NUMBER	DESCRIPTION	COUNT
74LS173	4-bit D-type register with 3 state output	15
74LS244	Octal 3 state buffer	37
74LS157	Quad 2-input Multiplexer	6
COUNTER_8	8 bit Binary up/down counter	4
74LS386	Quad 2-input Exclusive OR gate	1
74LS04	NOT gate	22
AND_4	4 input AND gate	2
AND_3	3 input AND gate	3
74LS32	OR gate	1
74LS181	4-bit Arithmetic Logic Unit	1
74LS08	Quad 2-input AND gate	6
OR_3	3 input OR gate	1
2732	EPROM	8
COUNTER_4	4 bit binary up/down counter	1
6116	CMOS Static RAM 16K (2K * 8 bit)	1
74HC21	Dual 4 input AND gate	2

Discussion

- At first, starting the design was very difficult since we had minimum knowledge about it. Gradually by the passage of time, we learned from the mistakes and the process of designing became easier.

- Designing the registers, PC and SP were easier. But, CS and MEMORY were quite difficult in this regards.

- Due to different patched versions of Proteus, we had difficulty in working in different machines.

- We had to change our initial Block Diagram as instructed by our teachers. We did not have any Stack Pointer but had to add that to our design later on.

- We faced problems while designing micro-operations for each instruction. Sometimes, we thought that particular operation could be done in single cycle, but performing them in one cycle resulted in failure. Such as, loading PC from TEMP or MDR took 2 cycles, but initially, we thought this could be done in one cycle.

- 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.