Report on 4-bit PC

Course:
Digital Systems Design Sessional

Course Teacher:
Md Shariful Islam Bhuiyan
Chowdhury Md. Rakin
Mehnaz Tabassum Mahin
Mahmudur Rahman Hera

Presented by:
Section A1,Group-2
Abdullah Al Mamun (1305003)
Rajesh Debnath (1305017)
Rukshar Alam (1305031)
Mostofa Rafid Uddin (1305039)
Md. Saqib Hasan (1305057)

Introduction:

For this assignment, we had to construct a simplified 4-bit computer simulation using proteus.

The said 4-bit PC is consisted of different registers(ACC, MAR, MDR, etc), counters (PC,SP), ROMs, a RAM and an arithmetic logic unit.

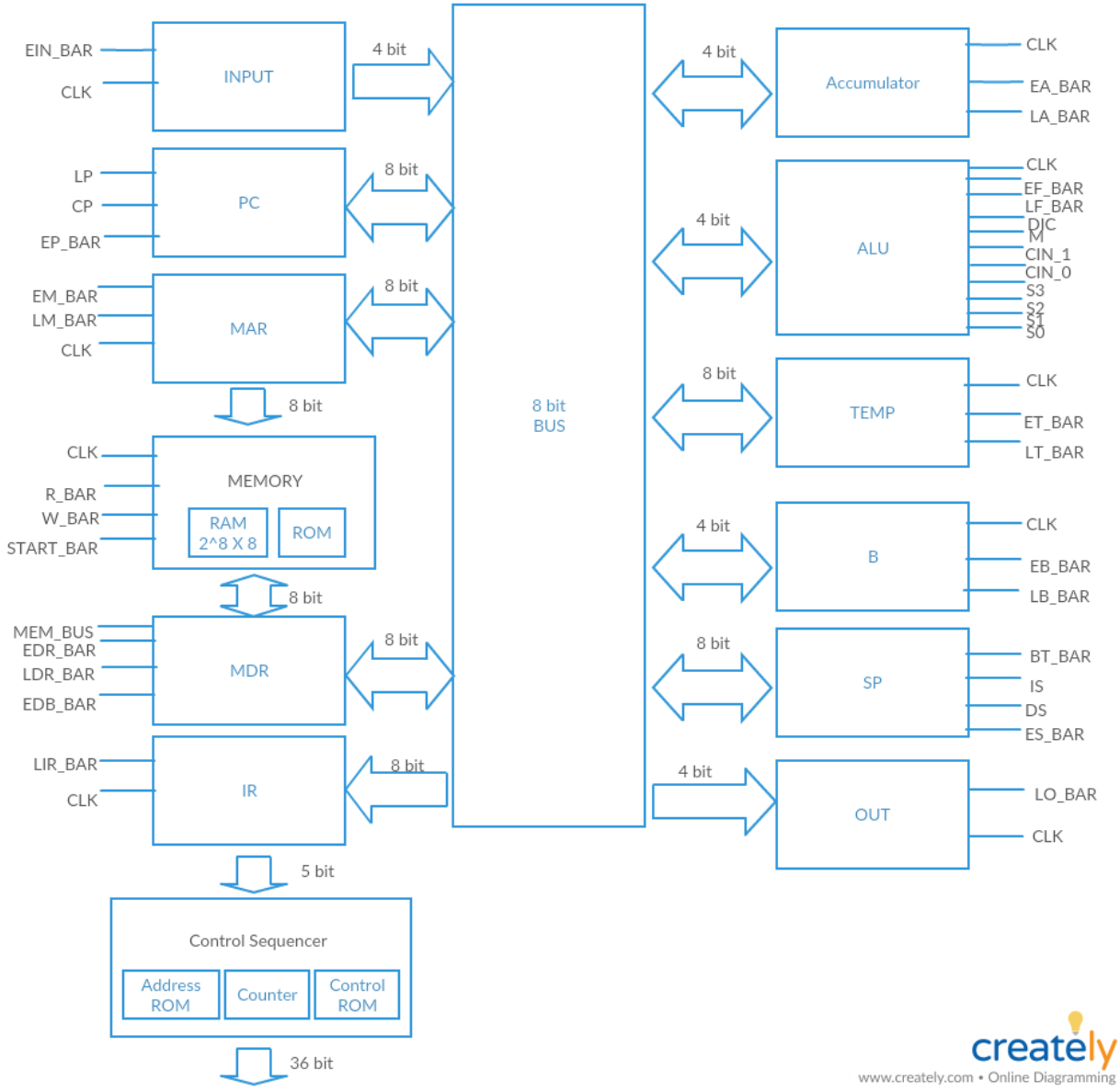
Our PC is able to execute any of the 28 instructions that were assigned to us.

The rest of this report contains implementation details of 4-bit PC on proteus.

Assigned Instruction set:

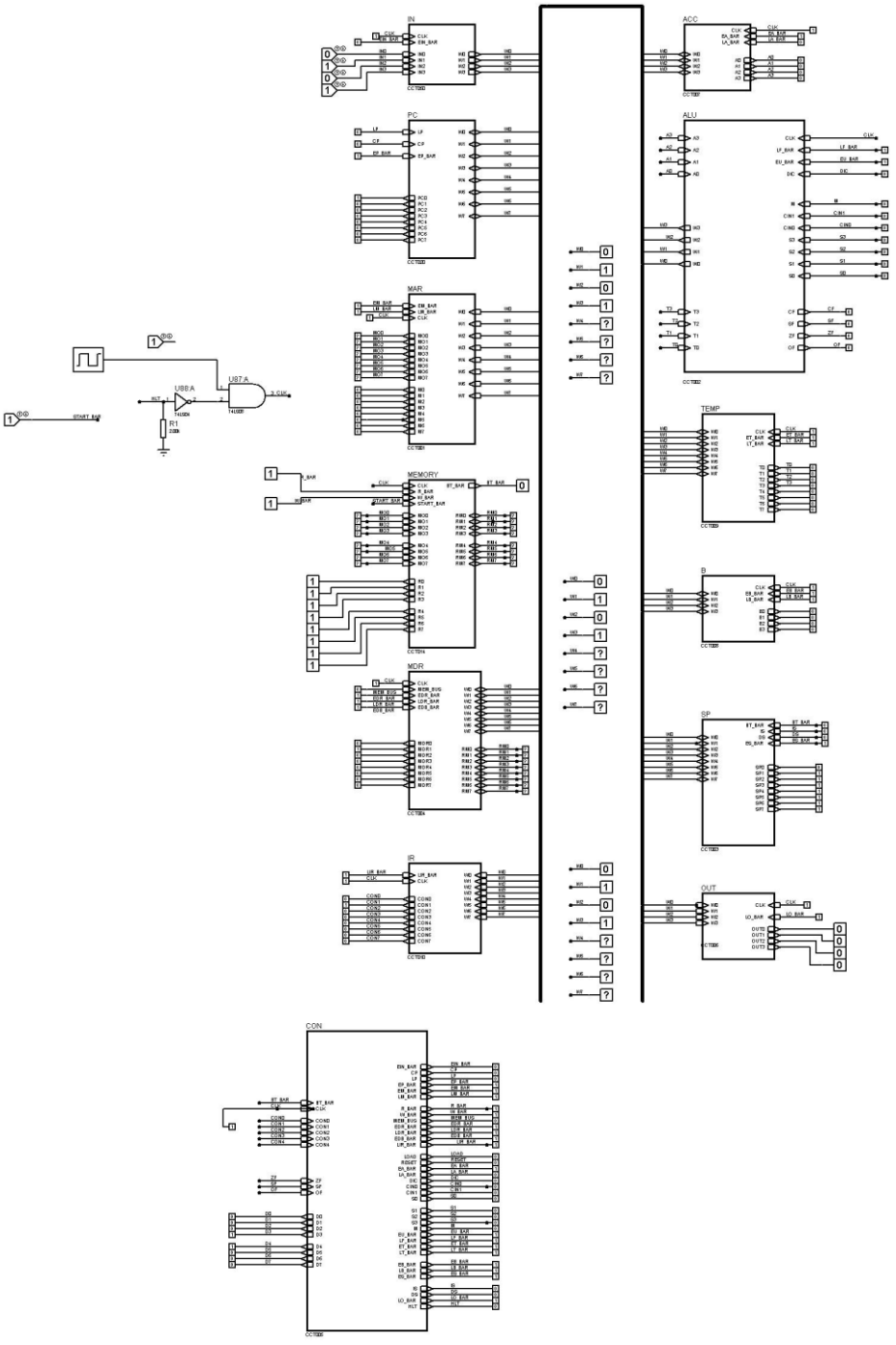
LDA address; STA address; MOV Acc, B; MOV B, Acc;
MOV Acc, immediate; IN; OUT; ADD B; ADC B; SUB B; SBB B;
ADD immediate; SUB immediate; CMP B; NEG; JO address; JNE address;
PUSH; POP; CALL address; RET; JMP; HLT; NOP;
XCHG; CMC; OR [address]; XOR B;

Block diagram of 4-bit PC:

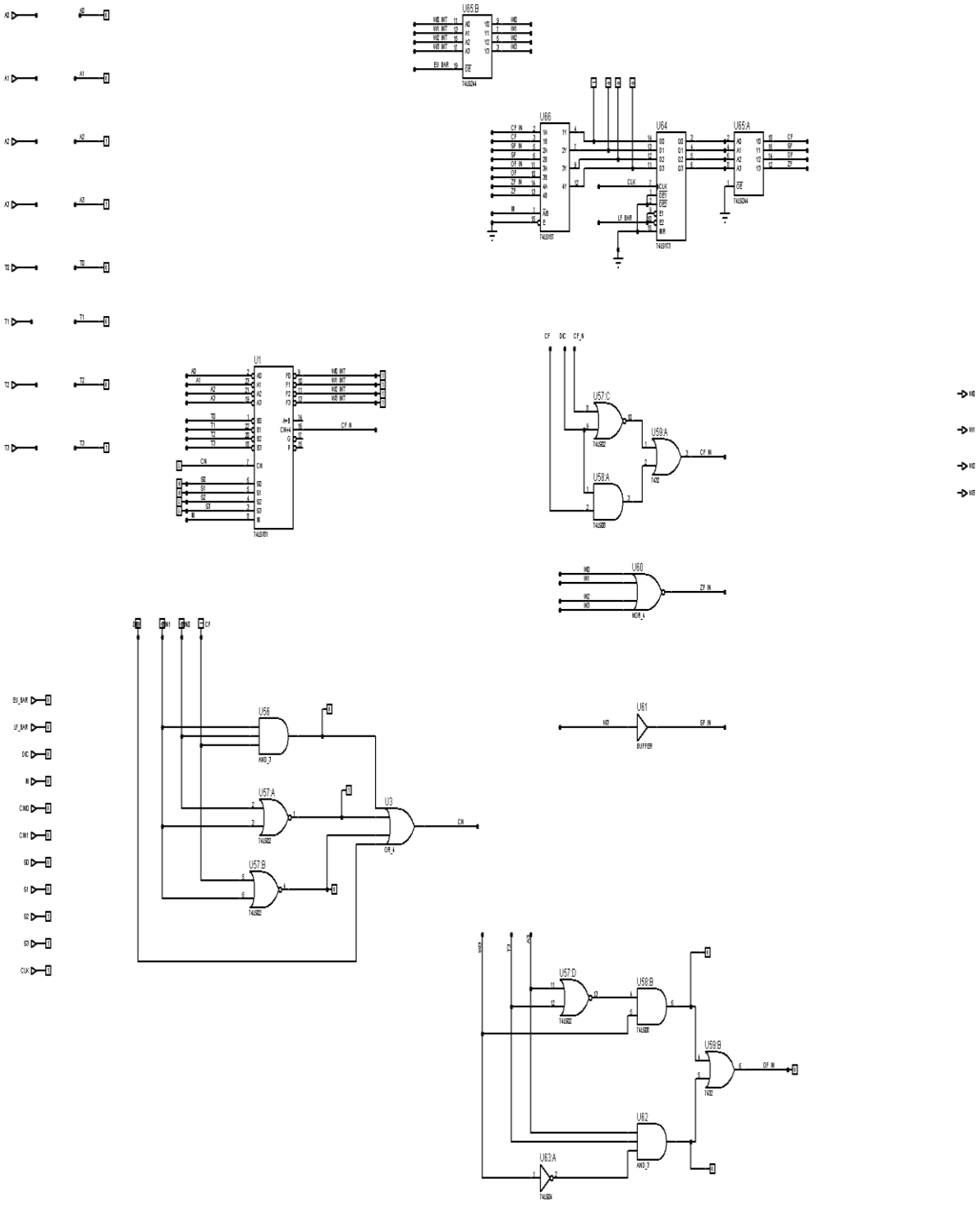


Circuit Diagrams:

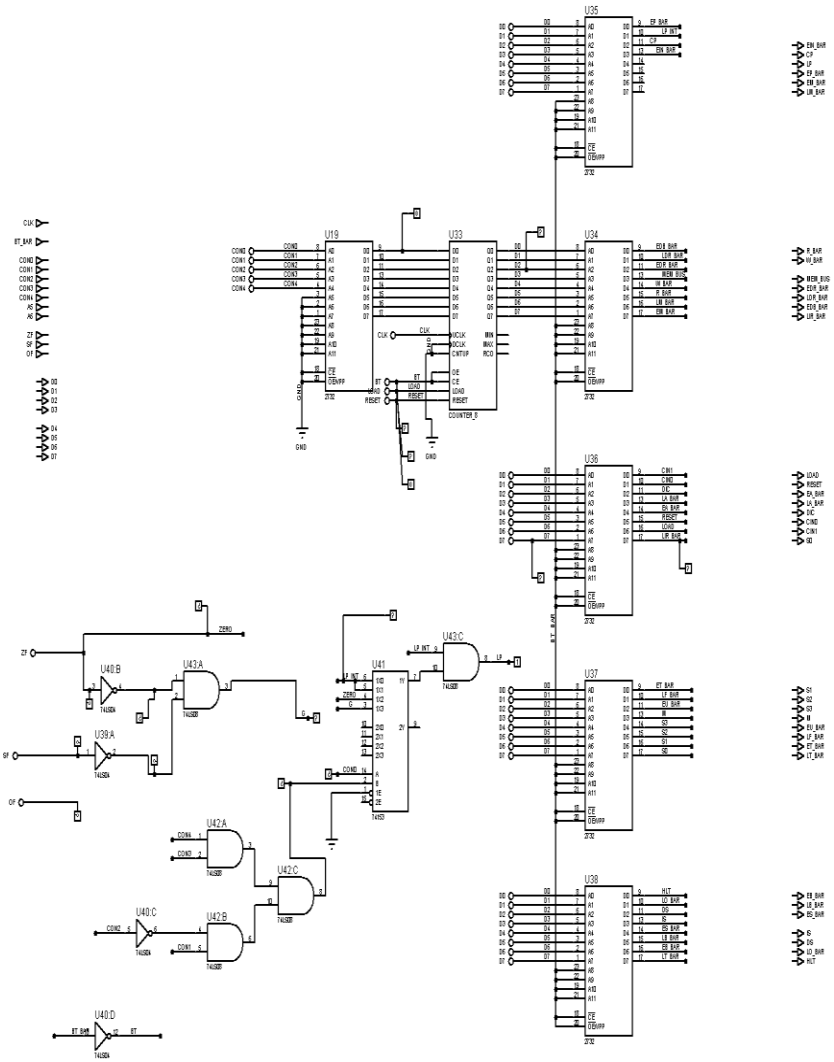
Main Circuit



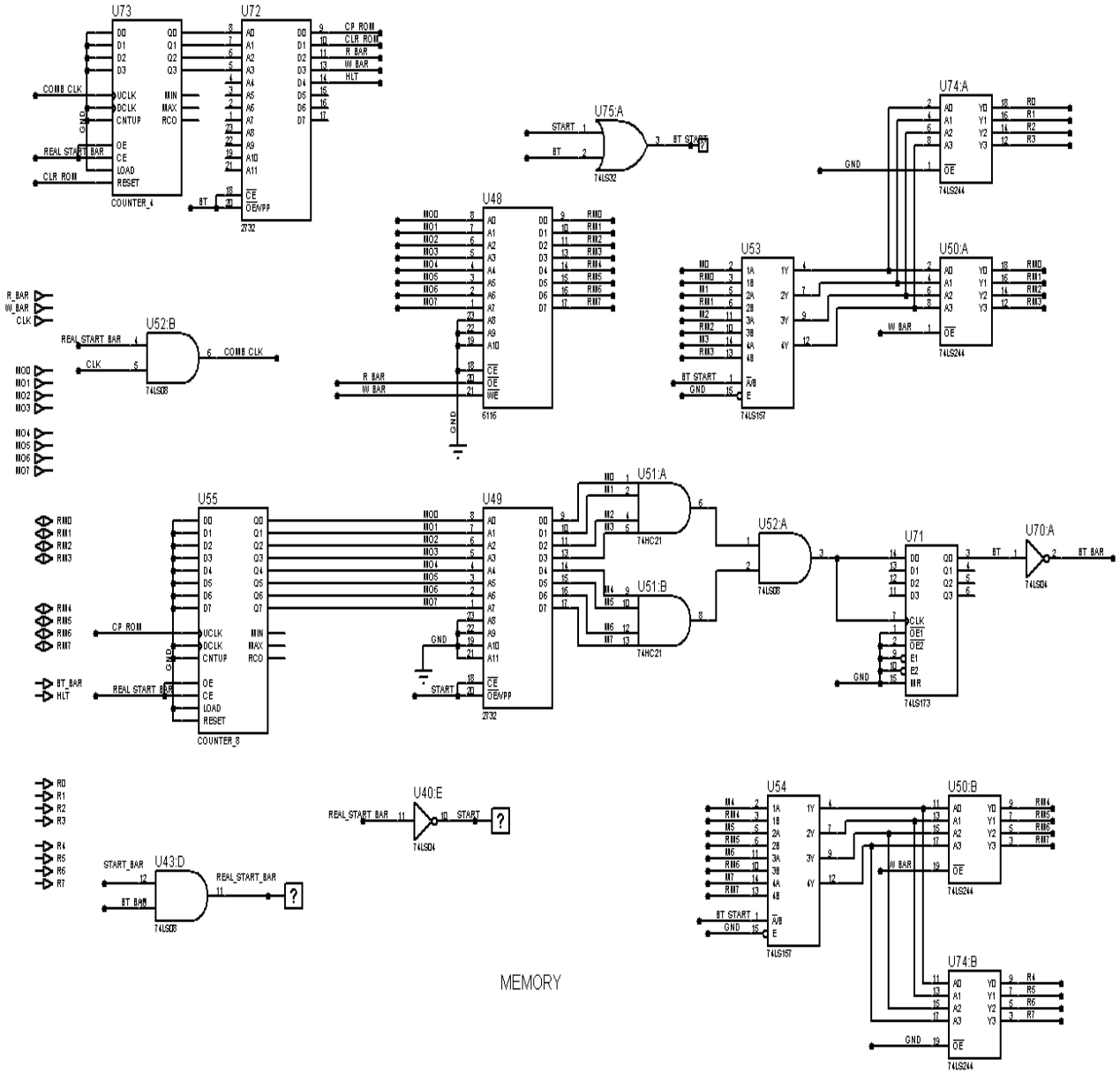
ALU


A0
A1
A2
A3
T0
T1
T2
T3
U65:B
U66
U64
U65:A
U1
U57:C
U59:A
U58:A
U60
U56
U57:A
U57:B
U3
U61
U57:D
U59:B
U62
U63:A
EN_BAR
LF_BAR
M
CIN0
CIN1
S0
S1
S2
S3
CLK
CF
DC
CF_N
ZF_IN
SF_IN
OF_IN
CIN

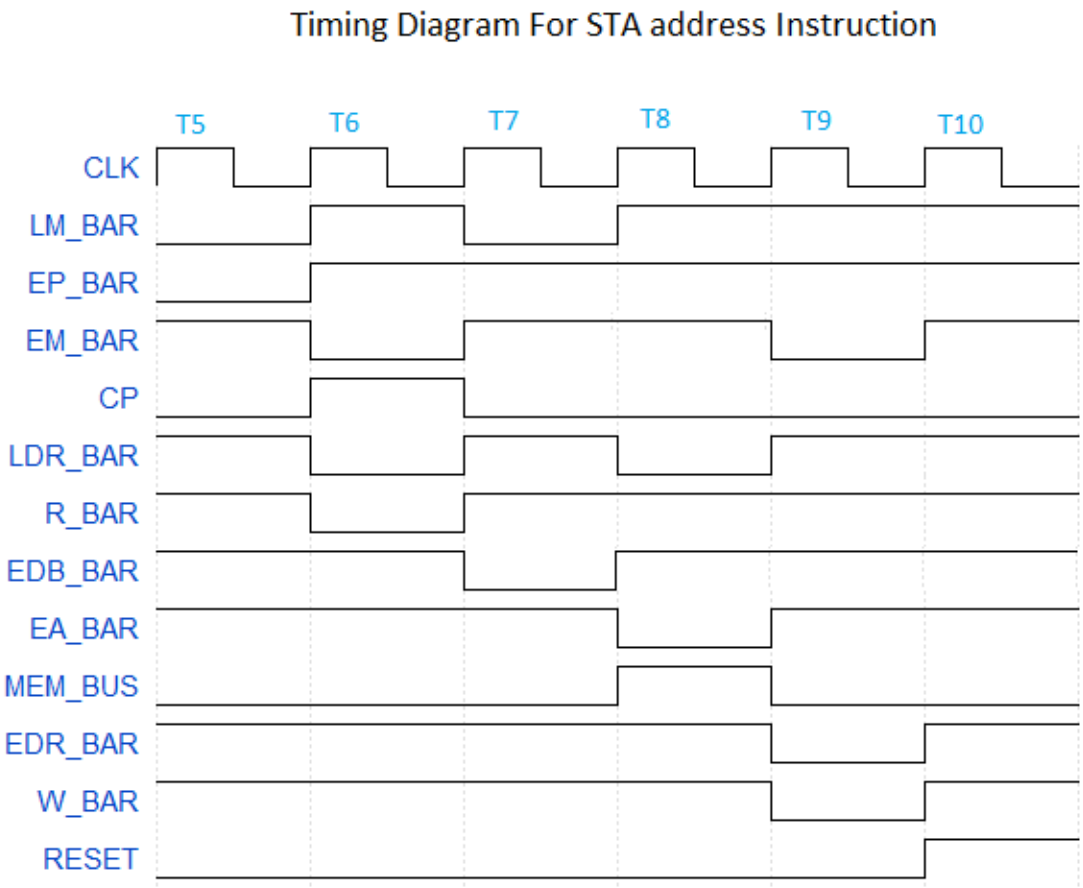
Control Sequencer



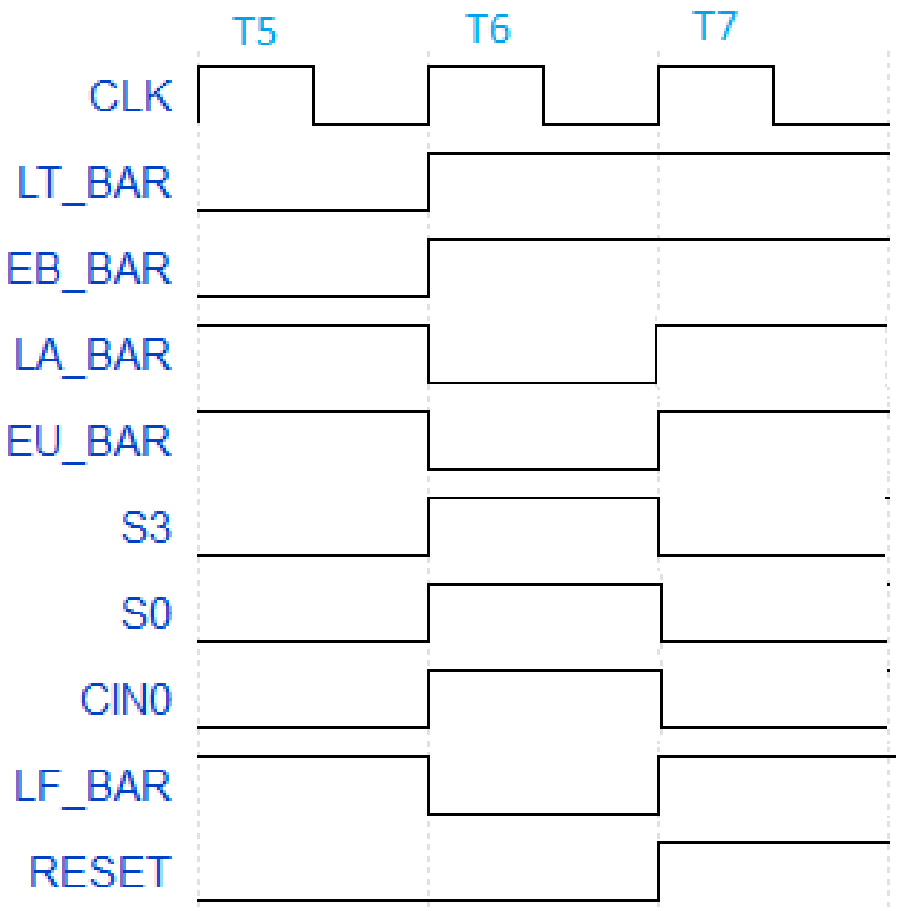
Memory


U73
U72
U75:A
U74:A
U48
U53
U50:A
U52:B
U55
U49
U51:A
U51:B
U52:A
U71
U70:A
U54
U50:B
U74:B
U40:E
U43:D
MEMORY

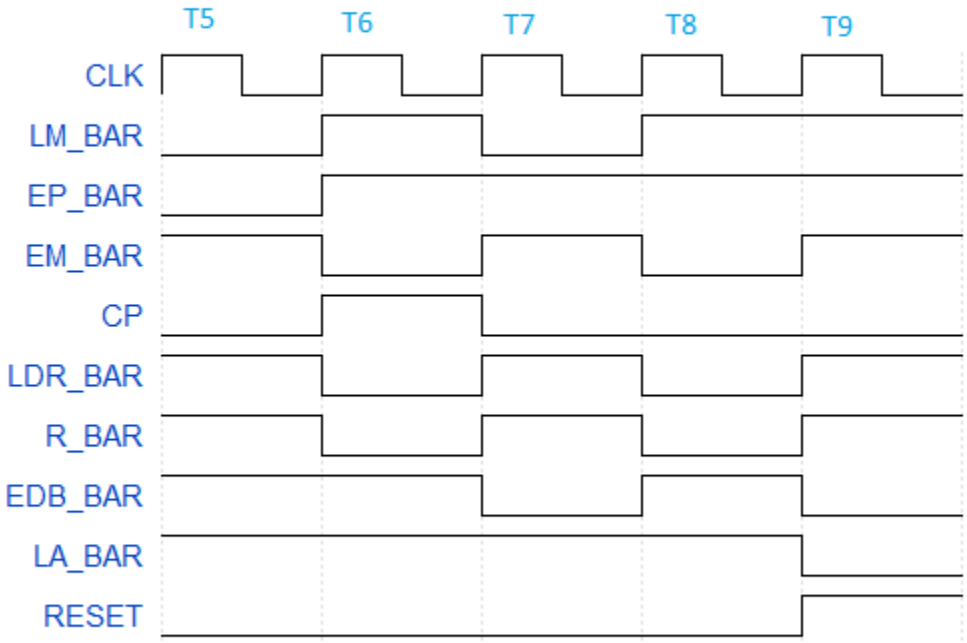
Timing Diagrams:



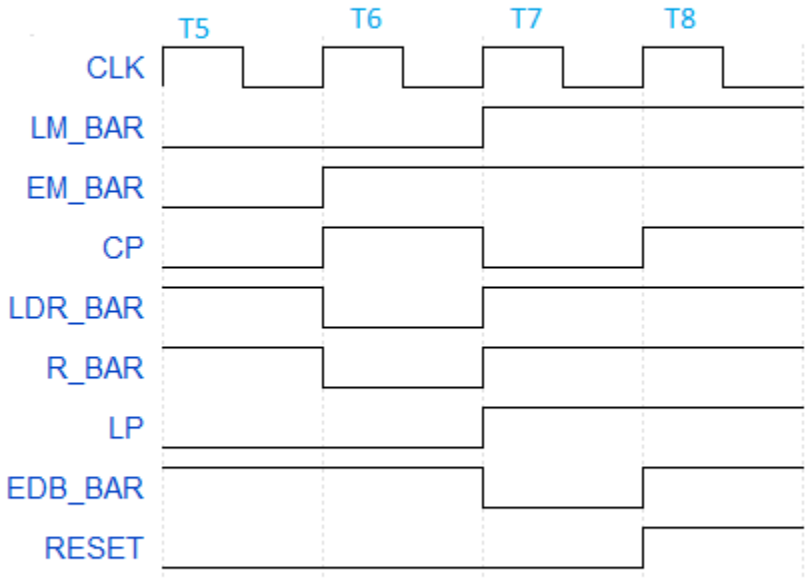
Timing Diagram For ADC B Instruction



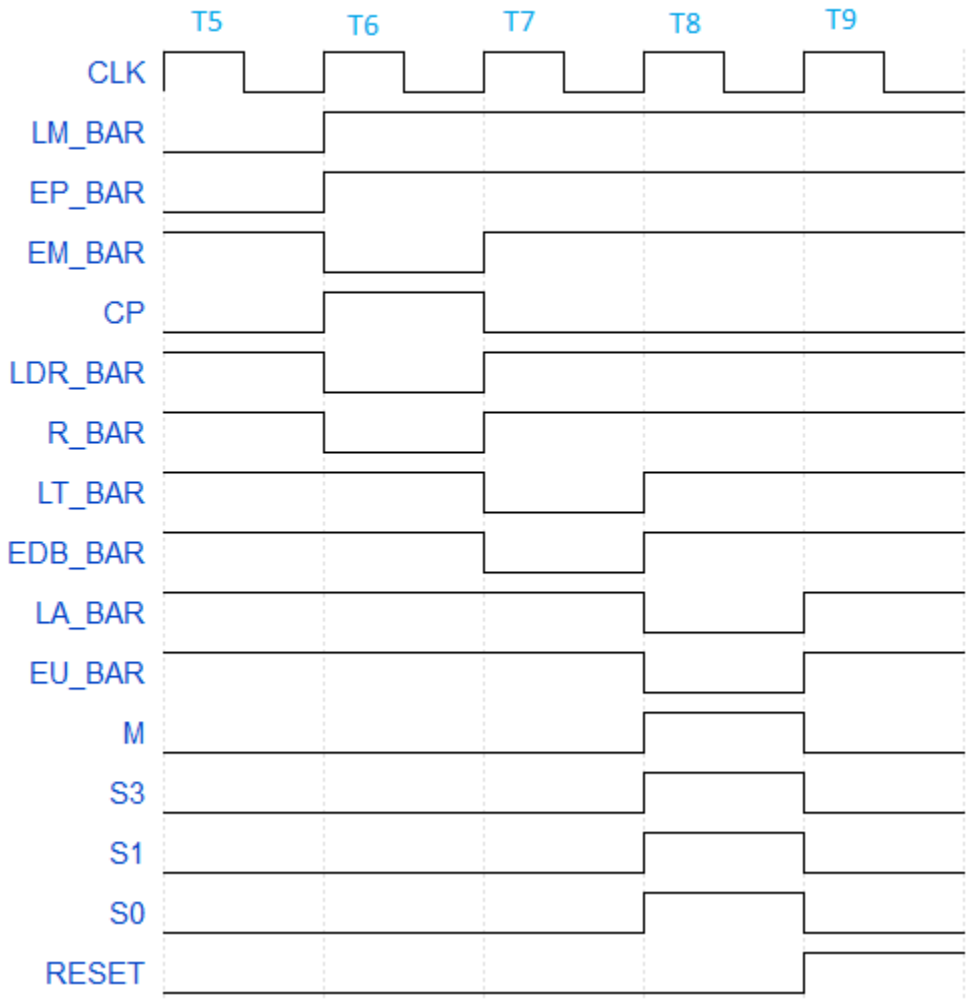
Timing Diagram For LDA address Instruction



Timing Diagram For JZ address Instruction



Timing Diagram For AND immediate Instruction



Explanation of all blocks:

Input Register (IN):

Input register is used to take user input into accumulator. The user input at any moment is stored in a tri-state buffer (74244).

This 4-bit Input register sends data to the bus. It doesn't get data form bus. So, it has a one way communication with the bus.

EIN_BAR is the associated signal with the tri-state buffer. When EIN_BAR gets a low signal, the output of IN register is passed to the W0-W3 lines via the buffer.

Program Counter (PC):

Program counter works as an instruction pointer. It is mainly a register that contains the address of the instruction being executed currently. After each instruction the value of PC is updated to the next instruction address. Instructions are usually fetched sequentially from memory, but some control instructions (namely JMP, CALL, RET) breaks the sequence by placing a new value in the PC.

PC can communicate with other registers only via bus. PC can give the address of the instruction to bus only and in control instructions as JMP, CALL, RET the next instruction address is stored in the PC via W-Bus.

CP is used to increment PC value after each instruction is fetched. EP_BAR is used to give address to bus and LP is used to load the address of the control instructions to PC. For example, when a JMP function is executed, the next instruction address or the operand address of JMP is stored at MDR register. The value of the register is stored at PC via W-Bus.

Memory Address Register (MAR):

Memory Address Register (MAR) is an 8-bit register. This register is used to point at the memory address from where we want to read or write content in memory.

This register is connected with the W-Bus and the Memory. It uses lines W0-W7 to connect with the W-Bus and lines MO0-MO7 to connect with the memory. It gets the address from the W-Bus through lines W0-W7 and passes this address to the memory through lines MO0-MO7. 2 tri-state 4-bit buffers were used for this.

EM_BAR and LM_BAR are the associated control signals both of which are active low. That means if we set EM_BAR to low, we can enable this register and pass data to W-Bus. And when we set LM_BAR to low, we can load address into this register from W-Bus.

RAM:

We can write content to and read content from RAM.

During the boot-loading period, instructions (hex-code for opcode and operand) is loaded into RAM. And, during the run of the PC, RAM is frequently accessed for instruction fetch, data store, data load etc.

RAM has the following associated signals.

WR_BAR, R_BAR, CLK.

RAM has the following input lines, MO0-MO7, these lines connect RAM and MAR. These lines are used to provide address to RAM. And, bidirectional data line, RM0-RM7. These lines connect MDR and RAM, using these lines either data is fed into RAM using the W_BAR signal, or Data is read from RAM using the R_BAR signal.

To write content to RAM at a particular address, we must point at the interested address of RAM using MAR, we provide data in RM0-RM7 lines via MDR, then we control W_BAR signal to write the data.

To read content from RAM at a particular address, we point that address using MAR, then we control the R_BAR signal to read content from RAM, later on this data is sent to MDR.

Boot loader:

Boot loader resides inside the MEMORY module. It is responsible for loading program data from a ROM to RAM during boot period.

Boot loader is consisted of two ROMs, and two counters. One of the ROMs contain the instructions and data and a special sentinel value FF, which is used to terminate the boot-loading process. The second ROM contains control words to drive the boot-loading process.

Using one counter, we provide same address to both the program ROM and RAM, so that content in ROM at a particular address can be sent to that exact address in RAM.

Using the second counter, we drive the boot control ROM.

Basically, using the boot control ROM, we execute the write cycle of RAM, and content from ROM to RAM is transferred one byte at a time. When the final line in the program ROM is reached, boot-loading process is terminated and a special signal BT_BAR is generated to let other key components know that boot has completed and PC can run now.

Memory Data Register (MDR):

MDR is an 8 bit register and it is used to read and write data to and from RAM, and it also relays RAM data to other registers.

MDR has bidirectional data line RM0-RM7. These lines connect MDR and RAM. These lines are used to read data from RAM to MDR and write MDR data to RAM. Given the appropriate signals, MDR can also load data from, and provide data to the BUS.

MEM_BUS, LDR_BAR, EDR_BAR, EDB_BAR are the associated signals with MDR.

The signal MEM_BUS is used in a multiplexer, to decide whether to load data from RAM or load data from BUS.

When MEM_BUS is LOW, data is loaded to MDR from RAM. This is used to read data from RAM.

When MEM_BUS is HIGH, data is loaded to MDR from the BUS.

When the LDR_BAR is LOW data is loaded to MDR either from RAM or BUS, depending on the MEM_BUS signal.

When EDB_BAR is LOW data from the MDR is sent to BUS via W0-W7.
When EDR_BAR is LOW data from the MDR is sent to RAM via RM0-RM7.

Instruction Register (IR):

Instruction Register is used to store the fetched Instruction from physical memory/RAM.

This 8-bit register receives instruction's opcode from memory through MDR register via BUS. This register holds the instruction's opcode and then passes the opcode to the control sequencer.

LIR_BAR is the associated signal. Setting LIR_BAR to low, we can load content from MDR via W0-W7 to this register. Least significant 5 bit of this value is available to the Control register via CON0-CON4 bus. We took only 5 bits because it can cover all our 28 different instructions (2^5 = 32 > 28).

Controller-Sequencer:

The controller-sequencer unit produces the control words for microinstructions that coordinate and direct the rest of the computer. The control word or microinstruction determines how the registers react to the next positive clock edge.

This supervisor unit contains two types of ROM, namely address ROM and control ROM. The control ROM contains the control word for each micro-instruction in order to execute a macro-instruction. The starting address of execution cycle of each macro-instruction is listed in address ROM. The index of address ROM is the op-code of a macro-instruction. We collect the op-code bits $\mathbf{CON_4CON_3CON_2CON_1CON_0}$ from the instruction register. These bits drive the address ROM and starting address of that particular routine is generated. Since our control word is 36-bit length, we need five control ROMs. One control word for any micro-instruction is listed in the same index of those ROMs. The outputs of the

control ROMs are the outputs of this block.

We use an internal counter to generate the required indices for control ROM. After getting BT_BAR as low, i.e., boot loading is done, this counter generates zero. In the zeroth index, the control word for first micro-instruction of fetch cycle is written. Thus, the corresponding signals are generated and the PC value is transferred to MAR. Similarly, for each count, the rest of the micro-instructions of fetch cycle are executed. After three clocks, the op-code for a macro-instruction is available in the input of the address ROM. At the last micro-instruction of fetch cycle, we generate a special signal named as "LOAD", which loads the content of Instruction Register, i.e., op-code of that macro-instruction to the counter. On the next clock, the counter starts counting from that address, which is the starting address of that macro-instruction's execution routine. Thus, sequentially the rest of the micro-instructions of that routine are executed, i.e., the control words are generated, which drive the rest of the computer. At the last micro-instruction of execution cycle, we generate a special signal named as **"**RESET**"**, which resets the counter. Hence, on the next clock the zeroth index's content are generated from the control ROM that means the fetch cycle is started again. Thus, another macro-instruction's execution is started immediately after the first one. Since, we use RESET signal to reset the internal counter for every execution routine, we do not need to waste a single clock. Hence, we have variable machine cycle. Thus, we avoid the hardware complexity by micro-programming through the RESET and LOAD signals.

In order to execute conditional jump instructions (**JG** and **JZ**), we have given zero flag, sign flag and overflow flag as input to the controller sequencer. We assigned opcode 1A to JZ instruction and opcode 1B to JG. In binary, 1A is 11010 and 1B is 11011. So, if **CON4 CON3 CON2' CON1** is true, it is either JG or JZ instruction. So, we used an mask, where if this value is false then the usual LP value is passed and if this is true, then it is a conditional jump instruction. Hence, if CON0 is true, then it is JG instruction. If CON0 is false, then it is JZ instruction. So, we passed the value of zero flag to the selector for JZ and passed the value of ZF'(SF XNOR OF) to the selector for JG. If the corresponding value is true, then LP is returned True that means the condition is true and the value will be loaded to PC. Otherwise, LP is returned false that means the condition is false and PC will be incremented sequentially.

Accumulator Register (ACC):

Accumulator register (ACC) is one of the most used blocks of 4-bit PC. It is a 4-bit register. It is used to perform data related operations. It can store and provide with data when necessary.

This register is connected with the W-Bus using the bidirectional lines W0-W3. It is also connected with the ALU using lines A0-A3. Using the bidirectional lines W0-W3 it can read or send data from or through W-Bus. It can also pass data to the ALU using the lines A0-A3. While taking such actions it uses 4-bit buffers also.

LA_BAR and EA_BAR are the control signals for this register both of which are active low. By setting EA_BAR to low, we can provide register data to the BUS. And when we set LA_BAR to low, we can load data into this register.

Arithmetic Logic Unit (ALU):

This asynchronous unit performs the required arithmetic as well as logical operations of our micro-computer.
We have used ***74LS181*** IC as an ALU. It has five control bits to determine the arithmetic or logic operation performed on words ***A*** and ***B***. Here, word A comes from Accumulator Register and B comes from Temporary Register. The output of ALU is provisioned to go to the ***W-BUS*** by enabling ***EU_BAR*** signal.

We cannot perform ADC and ***SBB*** instructions easily by mode selector bits and carry in (***CN)*** bit, since, it only uses the content of carry flag, not constant logic ***one***. Besides, the datasheet of that IC shows that, in order to execute ***SUB***, we provide logic ***one*** to the ***CN*** bit. In summary, there is no ready-made operation by which ***ADC, SUB,*** and ***SBB*** operations can be performed. Hence, we generate the following function table that relates the inputs of the ***CN*** bit of ALU to the external input signals. We need to provide two signals for controlling the carry such as ***CIN1*** and ***CIN0***. This function table yields a combinational circuit equation, i.e., ***CIN1'. CIN0. CF + CIN1. CIN0'***

+ CIN1. CF'.

CIN1	CIN0	CN (ALU input)	Required Operation
0	0	0	ADD
0	1	Content of carry flag (CF)	ADC
1	0	1	SUB
1	1	Inverted Content of carry flag (CF')	SBB

In order to keep track of a changing condition during a computer run, we use a flip-flop and a register named as flag register. We store carry flag, sign flag, overflow

flag, and zero flag. Except overflow flag, rest of the flags are readily available as ALU's output. To determine the overflow flag, we use simple intuition, that is, an overflow can only occur when two numbers added are both positive or both negative. We test the inputs' and output's sign flag for determining the overflow condition.

We know that the contents of flag register are changed only in arithmetic operations. We ensure it by the load signal of flag flip-flop, i.e., **LF_BAR**. Besides, only addition operation can change the carry. We prevent the carry contents from unwanted changes by using another flip-flop. Besides, in order to execute DEC instruction, a **DIC** control signal is used to so that it doesn't effect the carry flag whereas normal decrement affects the carry flag.

Temporary Register (TEMP):

Temporary register (TEMP) can be used to store both data and address whichever is needed in various operations. It is an 8-bit register.

This register interacts with the W-Bus and the ALU. With the bidirectional lines W0-W7 it keeps contact with the W-Bus. It is connected with the ALU with the lines T0-T3. With the lines W0-W7 it can receive or send data or address information through the W-Bus. It passes data to the ALU using lines T0-T3. It also uses 4-bit buffers to perform these actions.

LT_BAR and ET_BAR are the associated control signals which are active low. When ET_BAR is set to LOW, data is provided to the BUS. When LT_BAR is low data or address is loaded into this register.

B Register (B):

B register is used to store data operands for various computations.

This 4-bit register interacts with the BUS. This register can both load data from, and provide data to the BUS, depending on its input control signals.

LB_BAR and EB_BAR are the associated signals. Setting LB_BAR to low, we can load content from W0-W3 to B register and setting EB_BAR to low, we can load data to W0-W3.

Stack Pointer (SP):

Stack pointer is a register that stores the address of the last program request in a stack. A stack is a specialized memory segment that stores data in last in first out manner. The most recently entered request resides at the top of the stack and the program always takes request from the top.

At the starting of boot loader, we initialize stack pointer with FF to point last address of RAM. When a PUSH instruction is requested, SP is decremented and then this SP is loaded to the MAR to point new memory location. As SP always holds the recent request, when a POP instruction is requested, value of SP is loaded to MAP without increment and decrement.

When a function is called by CALL instruction, then the next instruction address is stored on the SP. After returning from this function by RET instruction, SP value is loaded to MAR to execute the instruction following function CALL instruction.

ES_BAR is used to load the data from SP to bus. BT_BAR is used to initialize SP at the starting of boot loader and it is initialized to FF to point last memory location. IS and DS is used to increment and decrement the value of SP respectively.

Output Register (OUT):

Output register can be used to display results of different computation, for instance by adding a hex-output converter with it or LEDs.

This 4-bit output register interacts with the BUS.

LO_BAR is the associated signal.

When LO_BAR gets a low signal, the output register loads the content of W0-W3.

Control Word:

HLT – Halt
DIC - Decrement Accumulator value

LF_BAR - Load Flag

M - Mode selector for ALU

S3 - ALU operation selection

S2 - ALU operation selection

S1 - ALU operation selection

S0 - ALU operation selection

CIN1 - Selector for CIN

CIN0 - Selector for CIN

EF_BAR - Enable Flag

EU_BAR - Enable Accumulator Register

LS - Load Stack

IS - Increment Stack

DS - Decrement Stack

ES_BAR - Enable Stack

LT_BAR - Load Temp Register

ET_BAR - Enable Temp Register

LO_BAR - Load Output Register

EB_BAR - Enable B
register LB_BAR -
Load B Register

EA_BAR - Enable Accumulator
Register LA_BAR - Load
Accumulator Register

EIN_BAR - Enable Input Port

LIR_BAR - Load Instruction
Register

MEM_BUS - Memory Bus

LDR_BAR - Load Data Register

EDR_BAR - Enable Data Register

EDB_BAR - Enable Data Bus

W_BAR - Write RAM

R_BAR - Read RAM

LM_BAR - Load MAR

EM_BAR - Enable MAR

CLEAR

CP - Counter for PC

EP_BAR - Enable PC

LP - Load PC

LOAD

RESET

Explanation of all instructions:

Here, MO stands for Micro operation and HLO stands for High Level Overview.

MO: Micro operations. Each minimal operation(done in one T state) needed to execute an instruction(macro) .

All the instructions share the fetch cycle. The fetch cycle has the following micro operations. Maximum number of Micro operations determine the maximum number of T states needed.

Fetch cycle:

1st cycle: load PC value to MAR

2nd cycle: RAM to MDR and PC= PC + 1

3rd cycle: MDR to IR

Micro operations for individual instructions(execution cycle):

1. LDA address
Opcode: 00H

HLO: ACC←Memory[Address]

MO: i) MAR ← Address[operand]
ii) ACC ← MDR

4th cycle:load load PC value to MAR

5th cycle: load RAM value to MDR and
PC++ 6th cycle: load MDR to MAR

7th cycle: load RAM value to MDR

8th cycle: load MDR data to Accumulator

2. STA address
Opcode: 01H

HLO: Memory[Address] ← ACC

MO: 4) load PC to MAR

5) load RAM to MDR, Increment PC //MDR holds the operand

6) load MDR to MAR

7) load ACC to MDR

8) Write MDR data to RAM

3. MOV Acc, B

Opcode: 05H

HLO: ACC ← B

MO: 4th cycle: load data from B to Accumulator

4. MOV B, Acc

Opcode: 04H

HLO: B← ACC
MO:

4th cycle: load Accumulator to B

5. MOV Acc, immediate
Opcode: 07H

HLO: ACC← immediate
MO:

4th cycle: load PC value to MAR

5th cycle: load RAM to MDR and PC++
6th cycle: load MDR to Accumulator

6. IN

Opcode: 02H

HLO: ACC← Input port.

MO: 4th cycle: load Input port to Accumulator..

7. OUT
Opcode: 03H
HLO: Output port← ACC.

MO: 4th cycle: Load Accumulator to Output Port.

8. ADD B

Opcode: 0AH

HLO: ACC ← ACC+
B MO:

4th cycle: Load register B to
TEMP_REGISTER 5th cycle: Select ALU
Operation for ADD

6th cycle: Load ALU output to Accumulator.

9. ADC B
Opcode:
0BH
HLO: ACC← ACC+ B + C (Carry)
MO:

4th cycle: Load register B data to TEMP_REGISTER
5th cycle: Select ALU Operation for ADC

6th cycle: Load ALU output to Accumulator.

10. SUB B

Opcode: 0CH

HLO: ACC← ACC- B
MO:

4th cycle: load B data to TEMP_REGISTER
5th cycle: select ALU operation for SUB 6th
cycle: Load ALU output to Accumulator.

11. SBB B
Opcode:

0DH
HLO: ACC← ACC - B -Bo(Borrow)
MO:

4th cycle: load B data to TEMP_REGISTER
5th cycle: select ALU operation for SBB 6th cycle: Load ALU output to Accumulator.

12. ADD immediate

Opcode: 0EH

HLO: ACC← ACC + immediate

MO: Cycle 4: Load PC to MAR

Cycle 5: Write RAM data to MDR ,Increment PC

Cycle 6: Load MDR to TEMP REGISTER

Cycle 7: Select ALU operation for ADD

Cycle 8: Load ALU output to Accumulator.

13. SUB immediate
Opcode: 10H

HLO: ACC← ACC - immediate
MO: Cycle 4: Load PC to MAR
Cycle 5: Write RAM data to MDR ,Increment PC
Cycle 6: Load MDR data to TEMP REGISTER
Cycle 7: Select ALU operation for SUB
Cycle 8: Load ALU output to Accumulator.

14. CMP B

Opcode: 11H
HLO: ACC not change, but flags will be changed according to (ACC-B)
MO:

4th cycle: load B to TEMP_REGISTER

5th cycle: select ALU operation for SUB (no load is needed to Accumulator)

15. RCR

Opcode: 12H

HLO: Acc ← Acc >> 1, Acc [MSB] ← C (Carry) ,C (Carry) ← Acc [LSB]

MO:

4th cycle: ACC to Temp
5th cycle: ACC + Temp + C to ACC
6th cycle: ACC to Temp
7th cycle: ACC + Temp + C to ACC
8th cycle: ACC to Temp
9th cycle: ACC + Temp + C to ACC
10th cycle: ACC to Temp

16. JZ address
Opcode: 1AH

HLO: Jump to address when zero flag is 1. MO:
If zero flag is 1,

4th cycle: load PC to MAR

5th cycle: load RAM data to MDR and PC+=1
6th cycle: load MDR to PC (if z =1)

17. JG address

Opcode: 1BH

HLO: Jump to address when A>B
MO: If zero flag is 0 and sign flag and
overflow have the same bit,

4th cycle: load PC to MAR

5th cycle: load RAM data to MDR and
PC+=1
6th cycle: load MDR data to PC (if z = 0 &&
(s == v))

18. PUSH
Opcode: 15H

HLO: STACK← ACC

MO: 4th cycle: Decrement SP and load accumulator to
MDR. 5th cycle: Load SP to MAR

6th cycle: Write MDR data to RAM location

19. POP
Opcode: 16H
HLO: ACC← STACK
MO:

4th cycle: load SP to MAR

5th cycle: load RAM value to MDR

6th cycle: increment SP and load MDR to Accumulator

20. CALL address

Opcode: 17H

HLO: Calls a subroutine(at the specified address) unconditionally.

i) Save next_instruction address to stack

ii) Jump to the operand address

MO:

4th cycle: Load PC to MAR

5th cycle: Write RAM data to MDR and increment PC

6th cycle: Load MDR data to TEMP_REGISTER and decrement SP

7th cycle: Load SP to MAR

8th cycle: Load PC to MDR

9th cycle: Write MDR data to RAM

10th cycle: Load TEMP REGISTER to PC

21. RET
Opcode: 18H

HLO: Returns from current subroutine to the caller unconditionally. MO:

4th cycle: Load SP to MAR

5th cycle: Write RAM data to MDR

6th cycle: Load MDR to PC and Increment SP

22. JMP address Opcode:
09H
HLO: Jumps unconditionally to address. MO:

4th cycle: Load PC to MAR

5th cycle: Write RAM data to MDR, increment PC 6th cycle:
Load MDR to PC

23. HLT

Opcode: 0FH

HLO: Halts execution.

MO: 4th cycle: activate halt pin to stop the clock.

24. NOP Opcode: 08H
MO:

4th cycle: Nothing else to do, return to the start of fetch cycle.

25. RCL

Opcode: 06H

HLO: Acc ← Acc >> 1, Acc [MSB] ← C (Carry) ,C (Carry) ← Acc [LSB]

MO:

4th cycle: ACC to Temp
5th cycle: ACC + Temp + C to ACC

26. DEC
Opcode: 19H

HLO: Acc ← Acc – 1.
MO:

4th cycle: Acc – 1 to Acc

27. OR B
Opcode: 13H
HLO: Acc ← Acc | B
MO:

4th cycle: Load PC to MAR

5th cycle: Write RAM to MDR, increment
PC 6th cycle: Load MDR to MAR

7th cycle: Write RAM to MDR

8th cycle: Load MDR to TEMP_REGISTER

9th cycle: SELECT ALU operation OR and Load ALU output to Accumulator

27. AND immediate

Opcode: 14H

HLO: ACC←
ACC.immediate
value
MO:

4th cycle: load PC value to MAR

5th cycle: RAM to MDR and PC= PC + 1
6th cycle: MDR to Temp
7th cycle: Acc &Temp to Acc

Explanation of all control signals:

Cycle Description:

				T-							
Macro-Instr	Op-C	Description	Total	St	Micro-Operati	Active			CON		
uction	ode		T-States	at	on						
				e							
				T1	MAR ← PC	LM_BAR, EP_BAR,	10	18	FF	DF	0
					PC← PC+1,	CP, LDR_BAR,					
ALL	-	Fetch	4	T2	MDR ←		10	18	FF	CE	98
						R_BAR, EM_BAR					
					RAM[MAR]						
				T3	IR ← MDR	EDB_BAR,	10	18	FF	9B	C8
						LIR_BAR					
				T4	LOAD from IR	LOAD	10	18	FF	DF	CA
				T5	MAR ← PC	LM_BAR, EP_BAR,	10	18	FF	DF	0
					PC← PC+1,	CP, LDR_BAR,					
				T6	MDR ←		10	18	FF	CE	98
						R_BAR, EM_BAR					

					RAM[MAR]						
LDA		ACC ←									
	00H		9			LM_BAR,				D	
address		RAM[address]									
				T7	MAR ← MDR	EDB_BAR,	10	18	FF		8
										B	
				T8	MDR ←	LDR_BAR, R_BAR,	10	18	FF	CE	88
					RAM[MAR]	EM_BAR					
				T9	ACC ← MDR	LA_BAR,	10	18	FE	D	C9
						EDB_BAR, RESET				B	
HALT	0FH	Halts	5	T5	HALT	HLT	50	18	FF	DF	C8
		execution									
IN	02H	ACC ←	5	T5	ACC ←	LA_BAR, EIN_BAR,	10	18	FE	5F	C9
		input_port			input_port	RESET					
				T5	MAR ← PC	LM_BAR, EP_BAR,	10	18	FF	DF	0
					PC← PC+1,	CP, LDR_BAR,					
				T6	MDR ←		10	18	FF	CE	98
						R_BAR, EM_BAR					
					RAM[MAR]						
						LM_BAR,				D	

				T7	MAR ← MDR	EDB_BAR,	10	18	FF		8
STA		RAM[address]								B	
	01H		9								
address		← ACC									
						LDR_BAR,					
				T8	MDR ← ACC	EA_BAR,	10	18	FD	EF	88
						MEM_BUS,					
				T9	RAM[MAR] ←	EDR_BAR,	09	63	98	07	F2
						EM_BAR,W_BAR,					
					MDR						
OUT	03H	output_port	5	T5	output_port	LO_BAR, EA_BAR,	10	18	ED	DF	C9
		← ACC			← ACC	RESET					
MOV B,	04H	B ← ACC	5	T5	B ← ACC	LB_BAR, EA_BAR,	10	18	F9	DF	C9
ACC						RESET					
MOV ACC,	05H	ACC ← B	5	T5	ACC ← B	LA_BAR, EB_BAR,	10	18	F6	DF	C9
B						RESET					
				T5	TEMP ← ACC	LT_BAR, EA_BAR	8	B7	98	07	F2
RCL	06H	ACC ← ACC<<1+CF	7	T6	ACC ← ACC + TEMP + C	EU_BAR, LF_BAR,LA_BAR,Cin0,S0	D	55	98	07	F2
				T7	NOP		09	F7	B8	07	F2
						RESET					
				T5	MAR ← PC	LM_BAR, EP_BAR,	10	18	FF	DF	0
						EM_BAR					

MOV ACC,		ACC ←			PC← PC+1,	CP, LDR_BAR,					
	07H		7	T6	MDR ←		10	18	FF	CE	98
immediate		immediate				R_BAR, EM_BAR					
					RAM[MAR]						
				T7	ACC ← MDR	LA_BAR,	10	18	FE	D	C9
						EDB_BAR, RESET				B	
NOP	08H	No Operation	5	T5	NOP	NOP, RESET	10	18	FF	DF	C9
				T5	MAR ← PC	LM_BAR, EP_BAR,	10	18	FF	DF	0
						EM_BAR					
					PC← PC+1,	CP, LDR_BAR,					
				T6	MDR ←		10	18	FF	CE	98
JMP		Jump to the				R_BAR, EM_BAR					
	09H		8								
address		address			RAM[MAR]						
				T7		LP, EDB_BAR	10	18	FF	D	CC
					PC ← MDR					B	
				T8		CP, LP, RESET	10	18	FF	DF	D
											D
				T5	TEMP ← B	LT_BAR, EB_BAR,	14	98	B7	DF	C8
						S3, S0					
		ACC ← ACC +									
ADD B	0AH		7	T6		S3, S0	14	98	FF	DF	C8
		B									
					ACC ← ACC +						
						LA_BAR, EU_BAR,					
					TEMP						
				T7		S3, S0, LF_BAR,	4	90	FE	DF	C9
						RESET					

				T5	TEMP ← B	LT_BAR, EB_BAR,	14	B8	B7	DF	C8
						S3, S0, CIN0					
		ACC ← ACC +									
ADC B	0BH		7	T6		S3, S0, CIN0	14	B8	FF	DF	C8
		B + C									
					ACC ←ACC +						
						LA_BAR, EU_BAR,					
					TEMP + C						
				T7		S3, S0, CIN0,	4	B0	FE	DF	C9
						LF_BAR, RESET					
				T5	TEMP ← B	LT_BAR, EB_BAR,	13	58	B7	DF	C8
						S2, S1, CIN1					
SUB B	0CH	ACC ← ACC - B	7	T6		S2, S1, CIN1	13	58	FF	DF	C8
					ACC ← ACC -						
						LA_BAR, EU_BAR,					
					TEMP						
				T7		S2, S1, CIN1,	3	50	FE	DF	C9
						LF_BAR, RESET					
				T5	TEMP ← B	LT_BAR, EB_BAR,	13	78	B7	DF	C8
						S2, S1, CIN1, CIN0					
		ACC ← ACC - B									
SBB B	0DH		7	T6		S2, S1, CIN1, CIN0	13	78	FF	DF	C8
		- BO									
					ACC←ACC -						
					TEMP - BO	LA_BAR, EU_BAR,					
				T7		S2, S1, CIN1, CIN0,	3	70	FE	DF	C9
						LF_BAR, RESET					
				T5	MAR ← PC	LM_BAR, EP_BAR,	10	18	FF	DF	0
						EM_BAR					

					PC← PC+1,	CP, LDR_BAR,					
ADD	0EH	ACC ← ACC +	9	T6	MDR ←		10	18	FF	CE	98
immediate		immediate				R_BAR, EM_BAR					
					RAM[MAR]						
				T7	TEMP ← MDR	LT_BAR,	14	98	BF	D	C8
						EDB_BAR, S3, S0				B	
				T8		S3, S0	14	98	FF	DF	C8
					ACC ← ACC +						
					TEMP	LA_BAR, EU_BAR,					
				T9		S3, S0, LF_BAR,	4	90	FE	DF	C9
						RESET					
				T5	MAR ← PC	LM_BAR, EP_BAR,	10	18	FF	DF	0
					PC← PC+1,	CP, LDR_BAR,					
				T6	MDR ←		10	18	FF	CE	98
						R_BAR, EM_BAR					
					RAM[MAR]						
SBB	10H	ACC ← ACC -	10								
RAM[address]		RAM[address]-Borrow		T7	MAR← MDR	EDB_BAR,LM_BAR					
				T8		LDR_BAR,					
					MDR← RAM[MAR]	R_BAR, EM_BAR	09	55	98	07	F2
				T9	TEMP← MDR	EDB_BAR,LT_BAR	09	F6	98	07	72

				T10	ACC←ACC -	LA_BAR, EU_BAR,					
						S2, S1, CIN1, CIN0,	3	70	FE	DF	C9
					TEMP - BO	LF_BAR, RESET					
				T5	TEMP ← B	LT_BAR, EB_BAR,	13	58	B7	DF	C8
						S2, S1, CIN1					
CMP B	11H	ACC - B	7								
				T6		S2, S1, CIN1	13	58	FF	DF	C8
					ACC - TEMP						
				T7		S2, S1, CIN1,	3	58	FF	DF	C9
						LF_BAR, RESET					
				T5	ACC ← ACC-1	LA_BAR, EU_BAR, DIC,S0,S1,S2,S3	09	F7	94	F1	F2
DEC	19H	ACC ← ACC -1	6								
					NOP						
				T6		RESET	09	F7	B8	07	F2
				T5	Temp← B	LT_BAR, EB_BAR	09	F7	98	07	32
OR		ACC ← ACC |									
	13H		6								
B		B									
						EU_BAR,LA_BAR,					
				T6	ACC ← ACC |		09	F8	90	7F	F2
					Temp	S1, S2, S3, M					
		ACC ← ACC & B		T5	MAR ← PC	LT_BAR, EB_BAR	8	B7	98	07	F2

AND immediate	14H		8								
				T6	PC++,MDR ← RAM[MAR]	LA_BAR, EU_BAR,	D	55	98	07	F2
						M, S3, S0, RESET					
				T7	MDR ← TEMP	ET_BAR,LDR_BAR, MEM_BUS	9	F6	98	07	72
				T8	ACC ← ACC & TEMP	EU_BAR, LA_BAR, S0, S1, S3,M	9	f7	90	DB	F2
					SP ← SP - 1,	DS, LDR_BAR,					
				T5		EA_BAR,	10	19	FD	EF	88
					MDR ← ACC	MEM_BUS,					
						EM_BAR					
				T6	MAR ← SP	LM_BAR, ES_BAR,	10	18	7F	DF	8
						EM_BAR					
PUSH	15H	RAM[SP] ←	10	T7		EDR_BAR,	10	18	FF	D7	88
		ACC				EM_BAR					
				T8		W_BAR, EDR_BAR,	10	18	FF	D5	88

					RAM[MAR] ←	EM_BAR					
					MDR						
				T9		EDR_BAR,	10	18	FF	D7	88
						EM_BAR					
				T1		R_BAR, RESET,	10	18	FF	DE	89
				0		EM_BAR					
				T5	MAR ← SP	LM_BAR, ES_BAR,	10	18	7F	DF	8
POP	16H	ACC ←	7			EM_BAR					
		RAM[SP]									
				T6	MDR ←	LDR_BAR, R_BAR,	10	18	FF	CE	88
					RAM[MAR]	EM_BAR					
					ACC ← MDR,	LA_BAR,				D	
				T7		EDB_BAR, IS,	10	1A	FE		C9
					SP ← SP + 1					B	
						RESET					
				T5	MAR ← PC	LM_BAR, EP_BAR,	10	18	FF	DF	0
					PC← PC+1,	CP, LDR_BAR,					
				T6	MDR ←		10	18	FF	CE	98
						R_BAR, EM_BAR					
					RAM[MAR]						
				T7	TEMP ← MDR,	LT_BAR,	10	19	BF	D	C8
					SP ← SP - 1	EDB_BAR, DS				B	
				T8	MAR ← SP	LM_BAR, ES_BAR	10	18	7F	DF	48

CALL		Calls a				LDR_BAR,					
	17H		14	T9	MDR ← PC	EP_BAR,	10	18	FF	EF	C0
address		subroutine									
						MEM_BUS					
				T1		EDR_BAR,	10	18	FF	D7	88
				0		EM_BAR					
					RAM[MAR] ←						
				T1		W_BAR, EDR_BAR,	10	18	FF	D5	88
				1	MDR	EM_BAR					
				T1		EDR_BAR,	10	18	FF	D7	88
				2		EM_BAR					
				T1		LP, ET_BAR	10	18	DF	DF	CC
				3	PC ← TEMP						
				T1		CP, LP, RESET	10	18	FF	DF	D
				4							D
				T5	MAR ← SP	LM_BAR, ES_BAR	10	18	7F	DF	48
				T6	MDR ←	LDR_BAR, R_BAR,	10	18	FF	CE	88
		Returns from			RAM[MAR]	EM_BAR					
RET	18H	current	8								
				T7		LP, EDB_BAR, IS	10	1A	FF	D	CC
		subroutine			PC ← MDR,						
										B	
					SP ← SP + 1						
				T8		CP, LP, RESET	10	18	FF	DF	D
											D
RCR	12H	ACC ← ACC>>1(with carry	12	T5	TEMP ← ACC	EA_BAR, LT_BAR	09	F7	88	07	72

				T6	ACC ← ACC+TEMP+C	LA_BAR,EU_BAR, LF_BAR, Cin0, S0	09	F7	92	81	F2
				T7	TEMP ← ACC	EA_BAR, LT_BAR	09	F7	88	07	72
				T8	ACC ← ACC+TEMP+C	LA_BAR,EU_BAR, LF_BAR, Cin0, S0	09	F7	92	81	F2
				T9	TEMP ← ACC	EA_BAR, LT_BAR	09	F7	88	07	72
				T10	ACC ← ACC+TEMP+C	LA_BAR,EU_BAR, LF_BAR, Cin0, S0	09	F7	92	81	F2
				T11	TEMP ← ACC	EA_BAR, LT_BAR	09	F7	88	07	72
				T12	ACC ← ACC+TEMP+C	LA_BAR,EU_BAR, LF_BAR, Cin0, S0	09	F7	92	81	F2
				T5	MAR ← PC	LM_BAR, EP_BAR,	08	B7	98	07	F2
						EM_BAR					
					PC← PC++,	CP, LDR_BAR,					
				T6	MDR ←		0D	55	98	07	F2
						R_BAR, EM_BAR					
		Jump if ZERO=1									
JZ address	1AH		8		RAM[MAR]						
				T7		LP, EDB_BAR	0A	F6	98	07	F2
					PC ← MDR						
				T8		CP, LP, RESET	OF	F7	B8	07	F2
				T5	MAR ← PC	LM_BAR, EP_BAR,	08	B7	98	07	F2

					PC← PC+1,	CP, LDR_BAR,					
				T6	MDR ←		0D	55	98	07	F2
						R_BAR, EM_BAR					
		Jump if									
JG address	1BH		8		RAM[MAR]						
		greater									
				T7		LP, EDB_BAR	0A	F6	98	07	F2
					PC ← MDR						
				T8		CP, LP, RESET	0F	F7	B8	07	F2

How to Write and Execute a program in this computer:

Writing:

Writing a program for this computer can be done in two steps,

i) Writing in mnemonics form of instructions.
ii) Converting the mnemonic form to hex code. This is done by merging the hex op code of the instruction and the hex value of the operand(address or immediate value, if exist

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).

Each program must be terminated with the HALT instruction. Which has the hex opcode, F.

To denote the end of the program file, we use a special value of FF, we put this value at the last line of the program BIN file.

A sample program:

Mnemonic form:
IN
STA F3
LDA F3
HLT

Hex code formulation:
IN has opcode 2
STA has opcode 1
LDA has opcode 0,
So the program becomes,
02 01

F3
00
F3
0F

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

IC used with Count:

IC Number	IC Name	Number of ICs used
74LS173	4 bit D-type register with 3	14

74LS02	NOR gate	8
74LS04	NOT gate	4
AND_4	4 input AND gate	2
AND_3	3 input AND gate	4
74LS32	OR gate	4
74LS181	4 bit Arithmetic Logic Unit	1
74LS08	Quad 2-input AND gate	11
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
	BUFFER	1
	state output	
74LS244	Octal 3 state buffer	35
74LS157	Quad 2-input Multiplexer	5
COUNTER_8	8 bit Binary up/down counter	4
74153	Dual Line Multiplexer	1
74LS386	Quad 2-input Exclusive OR gate	1

Discussion:

We have completed implementing the simplified 4-bit PC with 28 instructions. This particular assignment from the Digital System Design lab has taught us in depth about the execution sequences of a computer in general.

During the development of the 4-bit PC, we have encountered some difficulties. Such as, we had made an assumption that given a valid address at ROM or RAM, we immediately get the content, which doesn't reflect reality. There is an invalid period during which even though there is a valid address at address line, we do not get the valid data at that address.

Another problem we faced was during working with memory write operation, by going through a lot of trial and error, we realized that, we need to provide valid data to RAM for three consecutive clock cycles, during which we give logic_high, logic_low, logic_high to the W_BAR signal for a successful write operation.

We faced problems with data contention while establishing communication between two components only to realize that if we contain the register value in buffers and do not pass it to BUS when it's not necessary, we will not have contention.

We have faced race conditions while implementing the ALU circuit, which was resolved using latches.

We have constructed the flag register in such a way that, its value remains unaffected during logical operations.

We have labeled the connecting wires so that one can easily identify flow of data through the circuit.