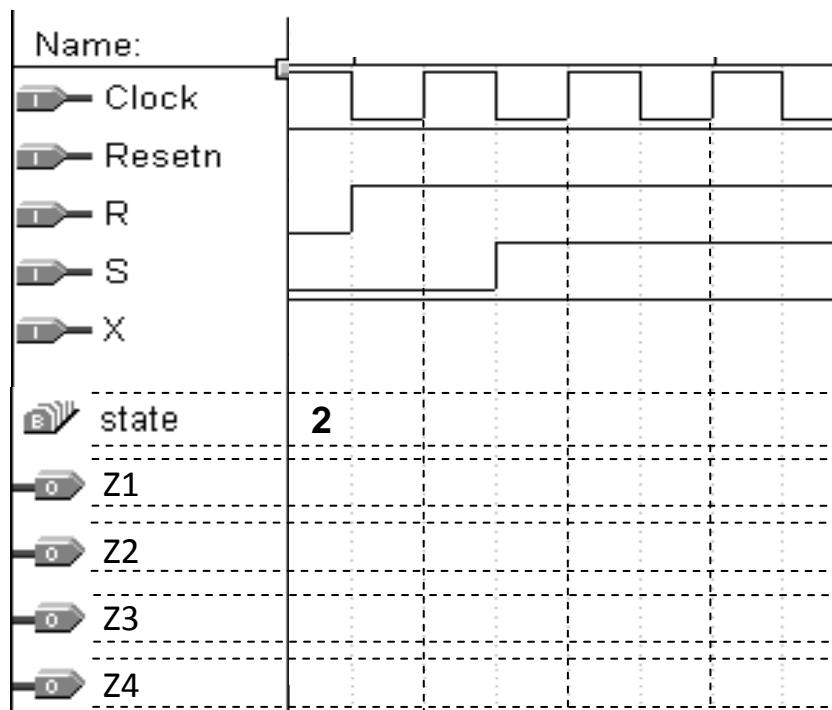
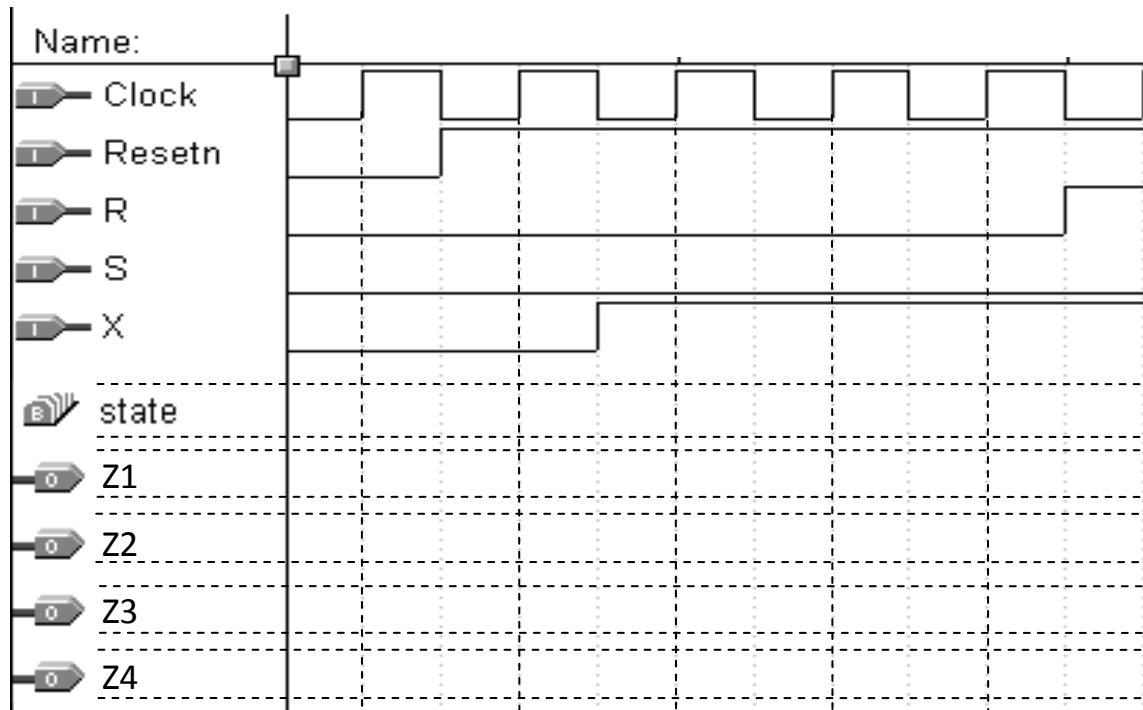


1. **ASM and VHDL.**

20 pts.

Shown in Figure 1(next page) is the VHDL specification of a ASM controller. Analyze the VHDL code and complete the following timing diagram: Specify the values for state (0, 1, 2, or 3), and outputs Z1, Z2, Z3 and Z4. Note that there are two timing diagrams, each is **independent** of the other. For the last one, the initial state is given (as state = 2).

(Assume all flipflops are initially '0'.)



```
LIBRARY ieee ;  
USE ieee.std_logic_1164.all ;
```

```
ENTITY T2Prob1 IS  
    PORT (    Clock, Resetn, X, R, S : IN    STD_LOGIC ;  
            Z1, Z2, Z3, Z4      : OUT  STD_LOGIC ) ;  
END T2Prob1 ;
```

**Figure 1. To be used for
problem 1**

```
ARCHITECTURE Behavior OF T2Prob1 IS  
    SIGNAL state : STD_LOGIC_Vector (1 DOWNT0 0);
```

```
BEGIN
```

```
    PROCESS ( Resetn, Clock )
```

```
        IF Resetn = '0' THEN
```

```
            state <= "00" ;
```

```
        ELSIF (Clock'EVENT AND Clock = '1') THEN
```

```
            CASE state IS
```

```
                WHEN "01" =>
```

```
                    IF S = '0' THEN state <= "10" ;
```

```
                    ELSE state <= "00" ;
```

```
                        Z2 <= '1' ;
```

-- Note: Z2 is an output

```
                    END IF ;
```

```
                WHEN "10" =>
```

```
                    state <= "01" ;
```

```
                WHEN "00" =>
```

```
                    IF X = '0' THEN state <= "00";
```

```
                    ELSE state <= "10";
```

```
                    END IF ;
```

```
                WHEN OTHERS =>
```

```
                    state <= "00";
```

```
            END CASE ;
```

```
        END IF ;
```

```
    END PROCESS ;
```

```
    Z1 <= '1' WHEN state = "01" ELSE '0' ;
```

-- Z1 is an output

```
    PROCESS (state, R, S)
```

```
    BEGIN
```

```
        Z3 <= '0' ;
```

```
        Z4 <= '0' ;
```

```
        CASE state IS
```

```
            WHEN "01" =>
```

```
                IF S = '0' THEN Z3 <= '1' ;
```

```
                ELSE Z4 <= '1' ;
```

```
                END IF ;
```

```
            WHEN "10" => IF R = '1'
```

```
                THEN Z3 <= '1' ;
```

```
                END IF ;
```

```
            WHEN OTHERS =>
```

```
        END CASE ;
```

```
    END PROCESS ;
```

```
END Behavior ;
```

2. **VHDL test bench analysis.** Given the following design for a circuit named “myPriority” and a testbench for it. Complete the timing diagram on the next page (functional simulation).

10 pts.

```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;

ENTITY myPriority IS
  PORT (x : IN STD_LOGIC_VECTOR(3 DOWNTO 0) ;
        z : OUT STD_LOGIC_VECTOR(1 DOWNTO 0) ;
        w : OUT STD_LOGIC ) ;
```

END myPriority ;

ARCHITECTURE Behavior OF myPriority IS

```
BEGIN
  PROCESS ( x )
  BEGIN
    z <= "00" ;
    IF x(3) = '1' THEN z <= "11" ; END IF ;
    IF x(1) = '1' THEN z <= "01" ; END IF ;
    IF x(2) = '1' THEN z <= "10" ; END IF ;
```

```
    w <= x(1) OR x(2);
```

```
  END PROCESS ;
```

END Behavior ;

```
-- Testbench
library ieee;
use ieee.std_logic_1164.all;
entity Prob2_tb is
end Prob2_tb;
architecture TB of Prob2_tb is
  signal sa : std_logic_vector(3 downto 0);
  signal sb : std_logic_vector(1 downto 0);
  signal sc : std_logic;
begin -- TB
  UUT : entity work.myPriority
    port map (
      x => sa,
      z => sb,
      w => sc );
  process
  begin
    sa <= "0001";
    wait for 10 ns;
    assert(sc = '0');
    assert(sb = "00");
```

```
    sa <= "1001";
    wait for 10 ns;
    assert(sc = '1');
    assert(sb = "11");

    sa <= "1011";
    wait for 10 ns;
    assert(sc = '1');
    assert(sb = "11");

    sa <= "0100";
    wait for 10 ns;
    assert(sc = '1');
    assert(sb = "10");

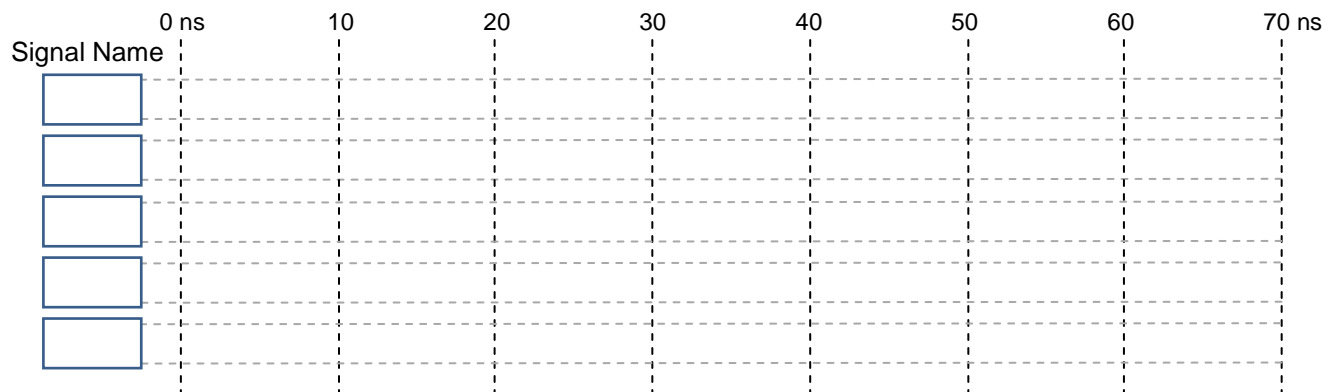
    sa <= "1111";
    wait for 10 ns;
    assert(sc = '1');
    assert(sb = "11");

    report "SIMULATION FINISHED";
    wait;
  end process;
end TB;
```

2. (continue)

(a) Given the design and the testbench from the previous page, complete the following timing diagram (functional simulation). (8 pts)

- For std_logic signals, draw the waveforms.
- For std_logic_vector signals specify the values in binary.
- There may be more “rows” and “columns” than you need. You don’t have to fill them all.

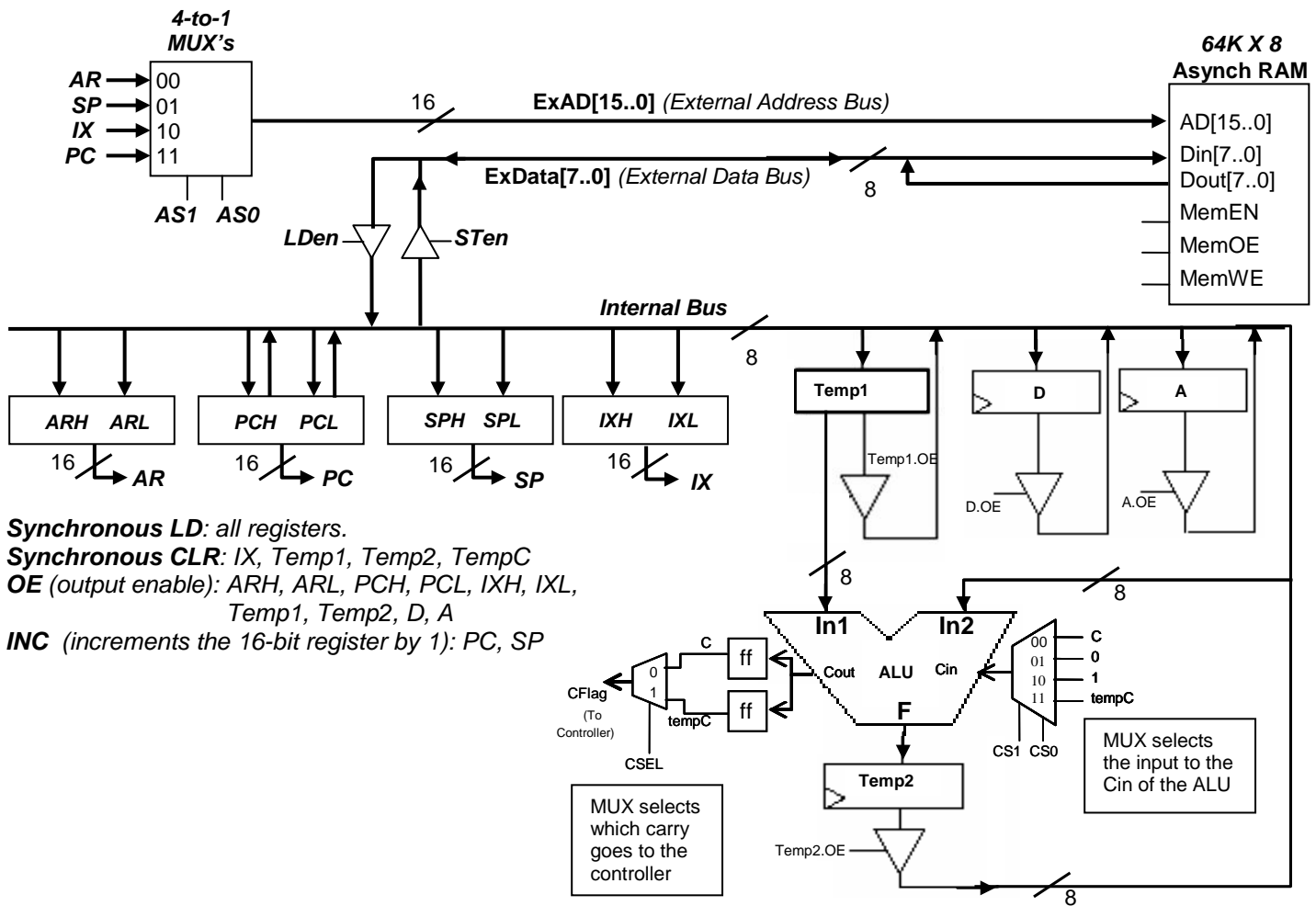


(b) List the assertion errors (if any) – what is the error and at what time (ns)? (2 pts)

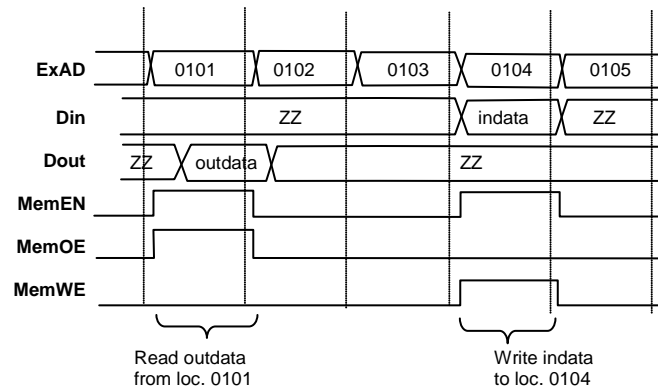
Small8-related problem: to be used for Problems 3, and 4.

Shown below is a modified architecture of the Small8 computer. Unless specified otherwise, assume that all the components are the same as you had them in your mini-project.

- The outputs of AR, SP, IX, and PC are connected to the External Address Bus through a set of 4-to-1 MUX's.
- The RAM is a 64K X 8 **asynchronous** RAM (which means no flipflop delays at the inputs and outputs). The timing requirements of its read and write operations are given in the timing diagram on the next page.
- There is a new flag register called tempC (temporary carry).
- Registers Temp1 and Temp2 are connected as shown, both with a synchronous **LD** input. Temp1 also has a synchronous **CLEAR** input.



Timing requirements for the read and write operations of the asynchronous RAM:



3. You are to implement the following two instructions for the Small8 computer by completing the ASM chart on the next page:

20 pts.

Description

INCA (FA)

A \leq A + 1; Increment Register A.

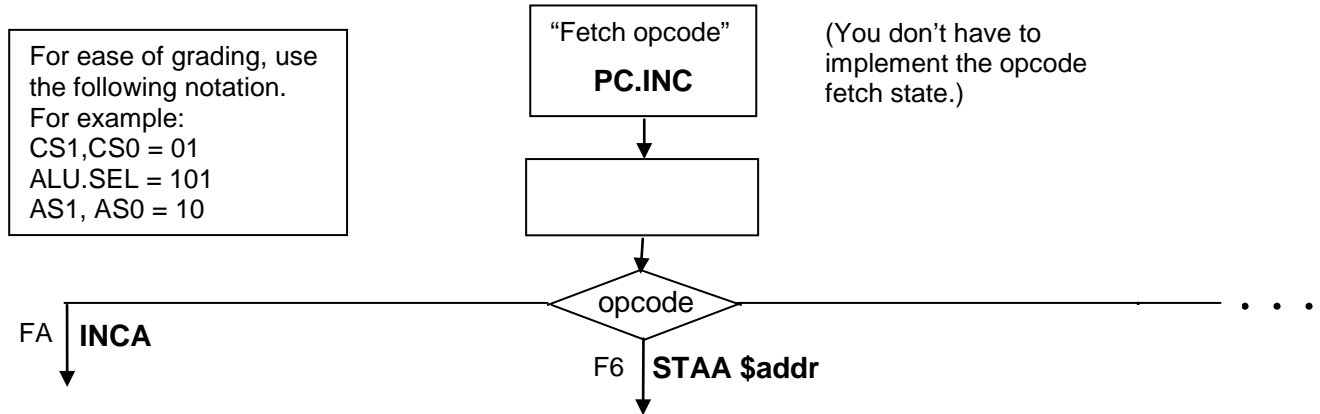
- Note that A is just a storage register. You cannot directly increment Register A.
- This instruction only affects the Z and S flags.
- ALU.SEL = 101 (F = In1 + In2 + Cin)

STAA \$addr (F6) Contents of Reg A is stored in mem[\$addr]

- All 4 flags are affected.
- This is an instruction you implemented for Small 8.

3. (continued) Complete the following ASM chart. Note that you **do not** have to complete the opcode fetch state.

For maximum credit, use the minimum number of states (including the use of conditional outputs). For maximum partial credit, “comment” your ASM chart.



4. Complete the following part of the ASM diagram to implement a new instruction ADDA b,X. For maximum credit, optimize your ASM diagram. **HINT:** The ADDA b,X instruction is very similar to the LDAA b,X and is a 2-byte instruction and defined as follows:

$$\mathbf{EA} = \mathbf{IX} + \mathbf{b} \quad \text{where}$$

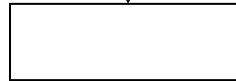
b is the “offset” byte, stored next in memory after the ADDA b,X opcode.

Important Notes:

- IX is a 16-bit storage register; with a LD input (cannot be incremented or do addition).
- ALU.SEL= 101 means F = In1 plus In2 plus Cin; Cout is the carry out.
- You cannot use “user” registers and flags (i.e., A, D, C flag) for the implementation of an instruction like ADDA b,X.
- All 4 user flags are affected.

For example:
CS1,CS0 = 01
ALU.SEL = 101
AS1,AS0 = 10

PC.INC



For maximum partial credit,
comment your ASM chart

opcode
offset byte

ADDA b,X

opcode

5. Assembly language program and .mif file

12 pts.

- (a) Given the following .mif file, analyze it and produce the corresponding assembly language program. All addresses and contents are in hex. (4 pts)

**Put the corresponding
assembly program here:**

Address	Content
0080	F6
0081	8B
0082	00
0083	21
0084	F1
0085	EC
0086	03
0087	FD
0088	B4
0089	80
008A	00
008B	31
008C	41

- (b) Assume that we add one instruction (LDAI \$15) at the beginning of the program, what would be the resulting .mif file. Put the answer there. (8 pts)

Address	Content
0080	
0081	
0082	
0083	
0084	
0085	
0086	
0087	
0088	
0089	
008A	
008B	
008C	
008D	
008E	
008F	
0090	
0091	
0092	
0093	
0094	

6. **Small8 Program Execution** – at the instruction level

18 pts.

Given below is a .mif file containing a “non-sense” program involving the **CALL** and **RET** instructions. Based on the Small8 architecture in your mini-project and the instruction set at the end of this exam, fill in the table at the bottom of this page.

Address	Content	
0020	88	LDAA
0021	31	
0022	00	
0023	C8	CALL
0024	27	
0025	00	
0026	F1	STAR D
0027	EC	STAA
0028	03	
0029	C0	RET
002A	00	
002B	00	
002C	14	

Important Notes:

The opcodes for the instructions can be found at the last page of the exam.

Initial values:

- Memory location \$00F1 = 01
- Other initial values are shown below in the table.
- If any initial value is not specified, assume it is 0.

Based on what you know about your Small 8 CPU, analyze the above .mif file and specify the content (in hex) of each of the registers and memory locations. Note that each row in the table below represents the contents at the end of the execution of each instruction. All values should be in HEX.

For ease of grading, leave the box blank to indicate no change to a register or memory location.

Initial
Values

PC	AR	SP	IX	A	D	Memory Location			
						004A	004B	004C	004D
0020	0000	004A	0036	00	00	00	00	00	00

* Fill to the end of the table (i.e., execute only 5 instructions).

INSTRUCTION	OP CODE	C	V	Z	S	DESCRIPTION	SYNTAX
Load Acc (Imm)	84	X	X	✓	✓	$A \leftarrow \text{mem}[\text{PC}]$	LDAI <data>
Load Acc (Abs)	88	X	X	✓	✓	$A \leftarrow \text{mem}(\text{mem}[\text{PC}])$	LDAA <address>
Load Acc (RR)	81	X	X	✓	✓	$A \leftarrow (D)$	LDAD
Store Acc (Abs)	F6	X	X	X	X	$\text{Mem}(\text{mem}[\text{PC}]) \leftarrow (A)$	STAA <address>
Store Acc (RR)	F1	X	X	X	X	$D \leftarrow (A)$	STAR D
Add with Carry	01	✓	✓	✓	✓	$A \leftarrow (A) + (D) + C$	ADCR D
Subtract with Borrow	11	✓	✓	✓	✓	$A \leftarrow (A) + \text{not}(D) + C$	SBCR D
Compare	91	✓	✓	✓	✓	Same as Subtract, but only change Status Flags (A is unchanged)	CMPR D
AND	21	X	X	✓	✓	$A \leftarrow (A) \text{ AND } (D)$	ANDR D
OR	31	X	X	✓	✓	$A \leftarrow (A) \text{ OR } (D)$	ORR D
XOR	41	X	X	✓	✓	$A \leftarrow (A) \text{ XOR } (D)$	XORR D
Shift Left Logical	51	✓	X	✓	✓	$C \leftarrow (A7), A7 \leftarrow A6, \dots, A0 \leftarrow 0$	SLRL
Shift Right Logical	61	✓	X	✓	✓	$C \leftarrow (A0), A0 \leftarrow A1, \dots, A7 \leftarrow 0$	SRRL
Rotate Left through Carry	52	✓	X	✓	✓	$C \leftarrow (A7), A7 \leftarrow A6, \dots, A0 \leftarrow C$	ROL C
Rotate Right through Carry	62	✓	X	✓	✓	$C \leftarrow (A0), A0 \leftarrow A1, \dots, A7 \leftarrow C$	ROR C
Branch on /C (Inh)	B0	X	X	X	X	if $(C=0)$, $\text{PC} \leftarrow \text{mem}[\text{PC}]$ else $\text{PC}++$	BCCA
Branch on C (Inh)	B1	X	X	X	X	if $(C=1)$, $\text{PC} \leftarrow \text{mem}[\text{PC}]$ else $\text{PC}++$	BCSA
Branch on Z (Inh)	B2	X	X	X	X	if $(Z=1)$, $\text{PC} \leftarrow \text{mem}[\text{PC}]$ else $\text{PC}++$	BEQA
Branch on S (Inh)	B3	X	X	X	X	if $(S=1)$, $\text{PC} \leftarrow \text{mem}[\text{PC}]$ else $\text{PC}++$	BMIA
Branch on /Z (Inh)	B4	X	X	X	X	if $(Z=0)$, $\text{PC} \leftarrow \text{mem}[\text{PC}]$ else $\text{PC}++$	BNEA
Branch on /S (Inh)	B5	X	X	X	X	if $(S=0)$, $\text{PC} \leftarrow \text{mem}[\text{PC}]$ else $\text{PC}++$	BPLA
Branch on /V (Inh)	B6	X	X	X	X	if $(V=0)$, $\text{PC} \leftarrow \text{mem}[\text{PC}]$ else $\text{PC}++$	BVCA
Branch on V (Inh)	B7	X	X	X	X	if $(V=1)$, $\text{PC} \leftarrow \text{mem}[\text{PC}]$ else $\text{PC}++$	BVSA
Decrement Acc	FB	X	X	✓	✓	$A \leftarrow (A) - 1$	DECA
Increment Acc	FA	X	X	✓	✓	$A \leftarrow (A) + 1$	INCA
Set Carry Flag	F8	✓	X	X	X	$C \leftarrow 1$	SETC
Clear Carry Flag	F9	✓	X	X	X	$C \leftarrow 0$	CLRC

Table 1: Small8 Instruction Set

Addendum to the instruction set

INSTRUCTION	OP CODE	C	V	Z	S	DESCRIPTION	SYNTAX
Subroutines:							
Load SP (Imm)	89	X	X	X	X	$\text{SP} \leftarrow \text{mem}[\text{PC}+1], \text{mem}[\text{PC}]$	LDSI <data>
Call	C8	X	X	X	X	$\text{SP} \leftarrow (\text{SP}) + 1; \text{mem}[\text{SP}] \leftarrow (\text{PC}_L);$ $\text{SP} \leftarrow (\text{SP}) + 1; \text{mem}[\text{SP}] \leftarrow (\text{PC}_H)$	CALL <address>
Return	C0	X	X	X	X	$\text{PC}_H \leftarrow \text{mem}[\text{SP}]; \text{SP} \leftarrow (\text{SP}) - 1;$ $\text{PC}_L \leftarrow \text{mem}[\text{SP}]; \text{SP} \leftarrow (\text{SP}) - 1$	RET
Index addressing:							
Load X (Imm)	8A	X	X	X	X	$X \leftarrow \text{mem}[\text{PC}+1], \text{mem}[\text{PC}]$	LDXI <data>
Load Acc (Indx)	BC	X	X	X	X	$A \leftarrow \text{mem}[(X) + b]$	LDAA b,X
Store Acc (Indx)	EC	X	X	X	X	$\text{mem}[(X) + b] \leftarrow (A)$	STAA b,X
Increment X	FC	X	X	X	X	$X \leftarrow (X) + 1$	INCX
Decrement X	FD	X	X	X	X	$X \leftarrow (X) - 1$	DECX

```

ENTITY __entity_name IS
PORT(__input_name, __input_name      : IN  STD_LOGIC;
      __input_vector_name            : IN  STD_LOGIC_VECTOR(__high downto __low);
      __bidir_name, __bidir_name      : INOUT STD_LOGIC;
      __output_name, __output_name    : OUT   STD_LOGIC);
END __entity_name;

```

```

ARCHITECTURE a OF __entity_name IS
    SIGNAL __signal_name : STD_LOGIC;
    SIGNAL __signal_name : STD_LOGIC;
BEGIN
    -- Process Statement
    -- Concurrent Signal Assignment
    -- Conditional Signal Assignment
    -- Selected Signal Assignment
    -- Component Instantiation Statement
END a;

```

```

<generate_label>:
    FOR <loop_id> IN <range> GENERATE
        -- Concurrent Statement(s)
    END GENERATE;

```

```

__instance_name: __component_name PORT MAP (__component_port => __connect_port,
                                              __component_port => __connect_port);

```

```

WITH __expression SELECT
    __signal <= __expression WHEN __constant_value,
              __expression WHEN __constant_value,
              __expression WHEN __constant_value,
              __expression WHEN __constant_value;

```

```

__signal <= __expression WHEN __boolean_expression ELSE
    __expression WHEN __boolean_expression ELSE
    __expression;

```

```

IF __expression THEN
    __statement;
    __statement;
ELSIF __expression THEN
    __statement;
    __statement;
ELSE
    __statement;
    __statement;
END IF;

```

```

CASE __expression IS
    WHEN __constant_value =>
        __statement;
        __statement;
    WHEN __constant_value =>
        __statement;
        __statement;
    WHEN OTHERS =>
        __statement;
        __statement;
END CASE;

```

```

WAIT UNTIL __expression;

```

Problem: Points:

1 (20 pts)	
2 (10 pts)	
3 (20 pts)	
4 (20 pts)	
5 (12 pts)	
6. (18 pts)	
Total:	