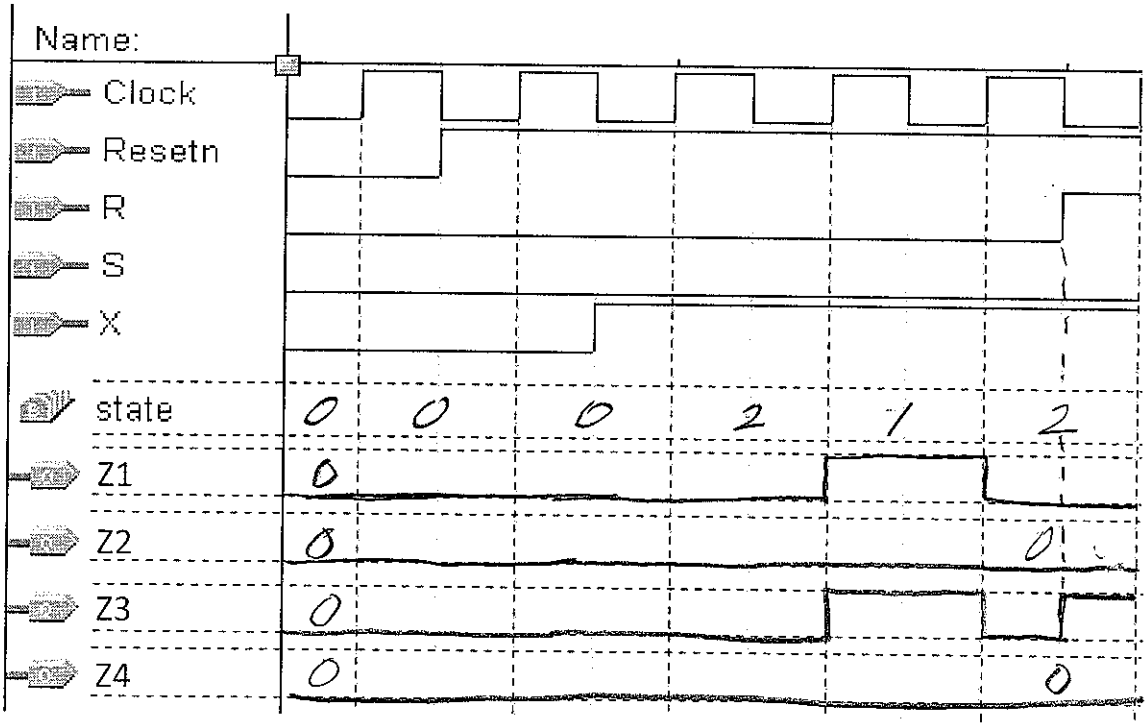
EEL 4712 – Digital Design
Final Exam – Fall Semester 2013

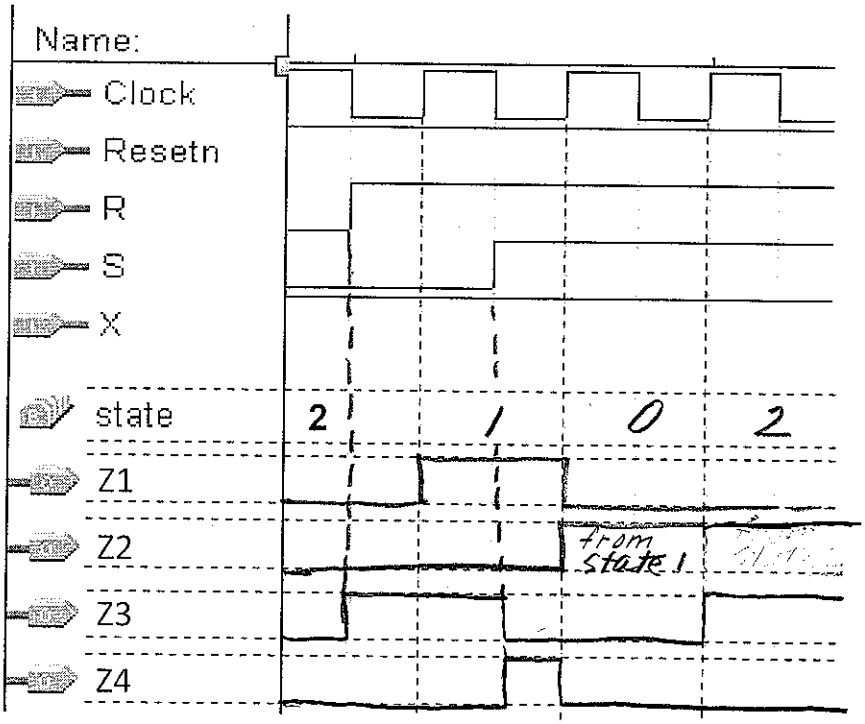
Name Solution

1. ASM and VHDL.

22 pts.

(a) 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).
Please show delays. (Assume all flipflops are initially '0'.)

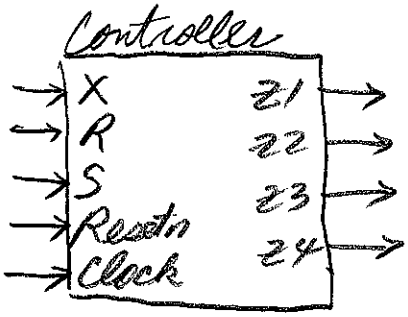


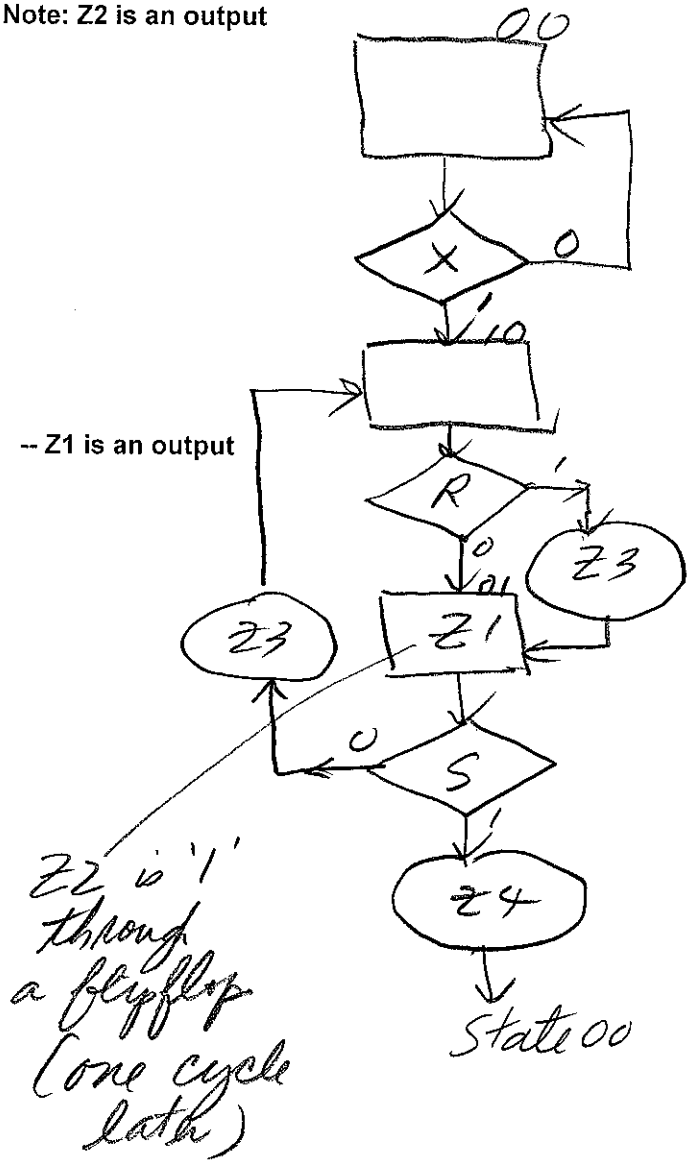


Name _____

```
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 ;

ARCHITECTURE Behavior OF T2Prob1 IS
     SIGNAL state : STD_LOGIC_Vector (1 DOWNTO 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 ;
```

Figure 1. To be used for problem 1



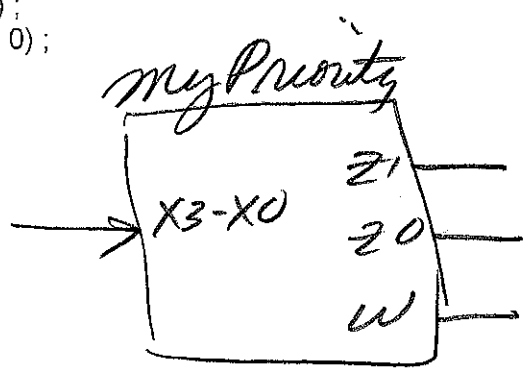


 Name _____

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).

12 pts.

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

ENTITY myPrioity 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 ;
```



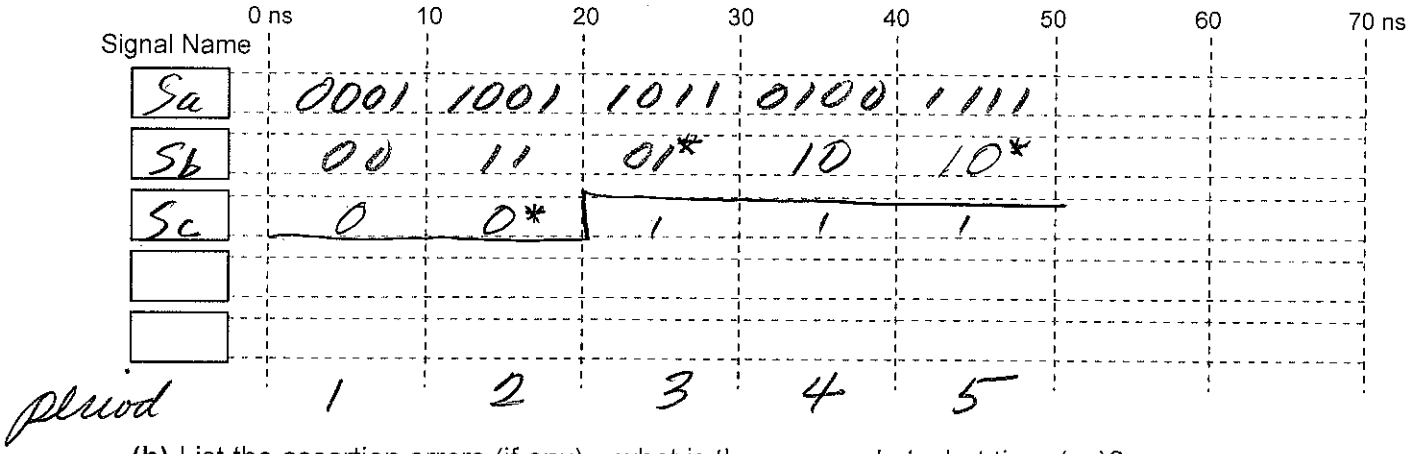

```
-- Testbench
library ieee;
use ieee.std_logic_1164.all;

entity Prob2_tb is
end Prob_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;
```

 Name _____

2. (continue)

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

- For std_logic signals, show the waveforms.
- For std_logic_vector signals specify the values in binary.
- There may be more "rows" and "columns" than you need.



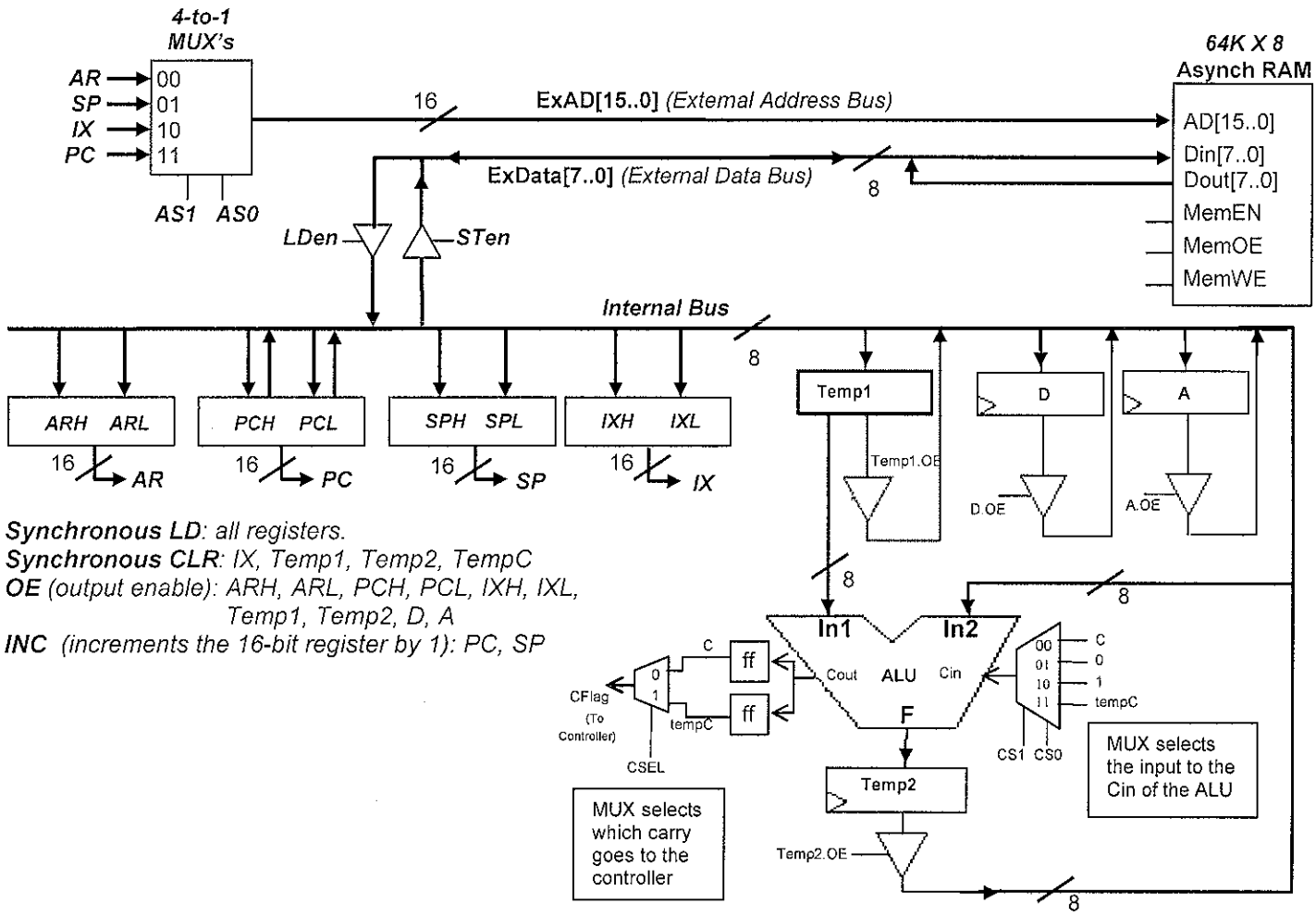
(b) List the assertion errors (if any) – what is the error and at what time (ns)?

20ns: period 2, Sc asserted to be a '1'
30ns: period 3, Sb asserted to be "11"
50ns: period 5, Sb asserted to be "11"

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

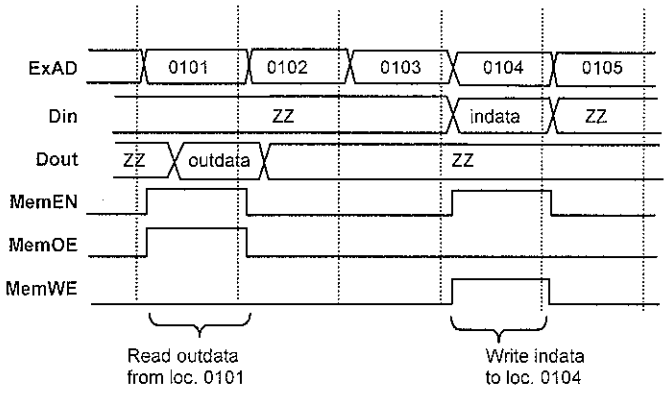
Shown below is a modified architecture of the Small8 computer. Unless specified otherwise, assume that all the components are the same as you had them in your mini-project.

- 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. 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 syncrhronous **LD** input. Temp1 also has a synchronous **CLEAR** input.



***Synchronous LD**: all registers.*
***Synchronous CLR**: IX, Temp1, Temp2, TempC*
***OE** (output enable): ARH, ARL, PCH, PCL, IXH, IXL, Temp1, Temp2, D, A*
***INC** (increments the 16-bit register by 1): PC, SP*

Timing requirements for the read and write operations of the <u>asynchronous</u> RAM:

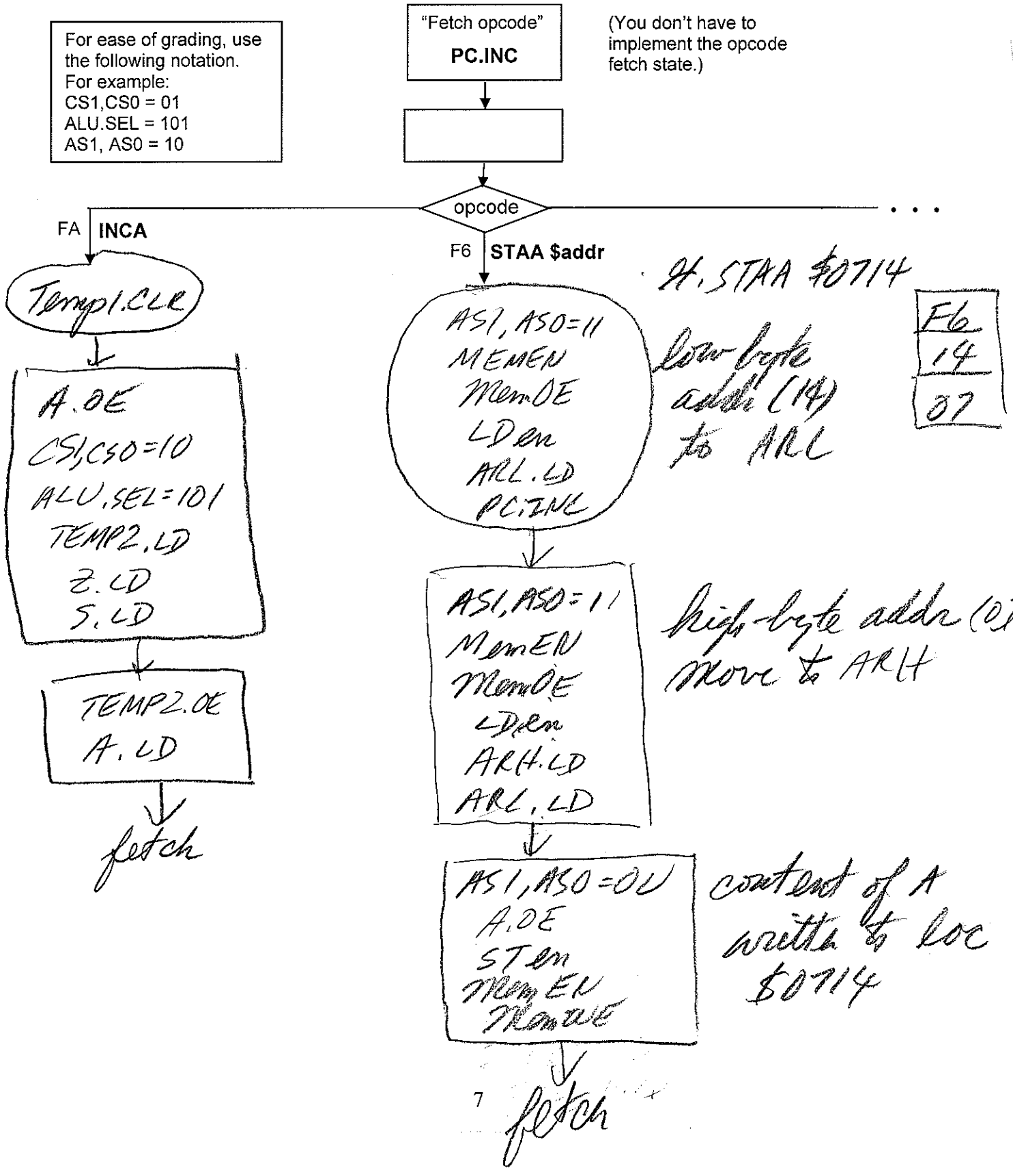


3. You are to implement the following two instructions for the Small8 computer by completing the ASM chart on the next page:

	<u>Description</u>
INCA (FA)	A <= (A) + 1; Increment Register A. • Note that A is just a storage register. You cannot directly increment Register A. • This instruction only afects only the Z and S flags. • ALU.SEL = 101 is In1 + In2 + Cin
STAA $addr (F6)	Contents of Reg A is stored in mem[$addr] • ~~All 5 flags are effected~~. • 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 is defined as follows:

20 pts.

ADDA b, X ; **Reg A <= mem(EA) + Reg A + Carry C;** where EA is the "effective address"

EA = IX + b where
IX is the 16-bit content of the IX register and
b is the "offset" byte, stored next in memory after the ADDA b,X opcode.

Ex: If IX= $20FF; b= $05. Then EA= $20FF + $0005 = $2104 **(Hint: 16-bit addition)**

Important Notes:

- IX is a 16-bit storage register; with a LD input (cannot be incremented or do addition).
- For the ALU: S2,S1,S0 = 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.

More Notes: Both C and tempC flags are flipflops whose inputs are from the Cout of the ALU. Both the C and tempC have synchronous LD and CLR inputs.

For ease of grading, use the following notation.
For example:
CS1,CS0 = 01
ALU.SEL = 101
AS1,AS0 = 10

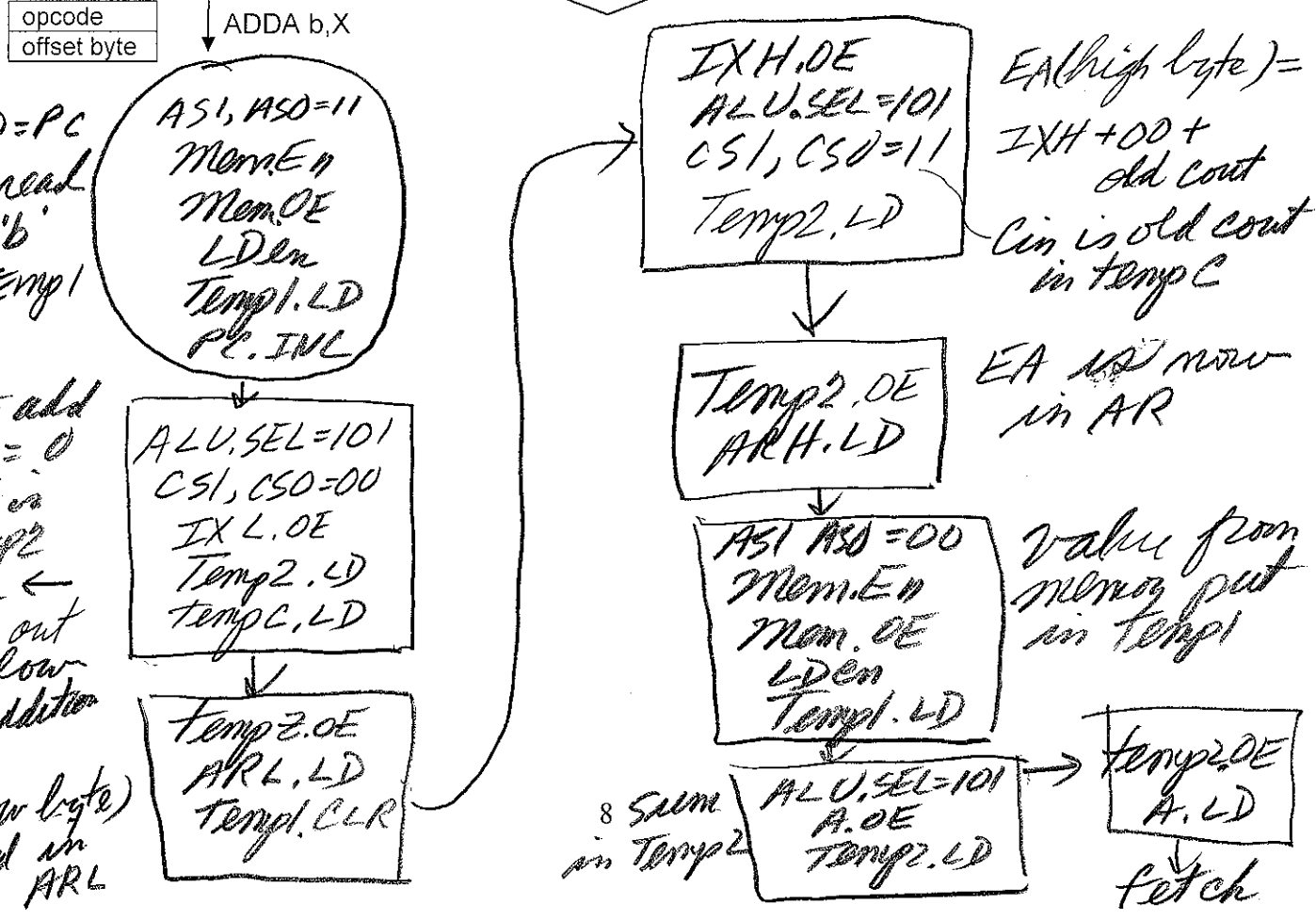
"Fetch opcode"
PC.INC

You don't have to implement the opcode fetch state.

For maximum partial credit, comment your ASM chart

A 2-byte Instruction:

opcode
offset byte



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

STAA $008B
ANDR D
STAR D
STAA 03,X
DECX
BNEA $0080
ORRD
XORRD

(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	84	LDAI $15
0081	15	
"L1" 0082	F6	STAA "THERE"
0083	8D	
0084	00	
0085	21	ANDR D
0086	F1	STAR D
0087	EC	STAA 03,X
0088	03	
0089	FD	DECX
008A	24	BNEA "L1"
008B	F2	
008C	00	
"L2" 008D	31	ORRD
008E	41	XORRD
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: 0031 = 14

- 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.

	PC	AR	SP	IX	A	D	Memory Location			
							004A	004B	004C	004D
Initial Values	0020	0000	004A	0036	00	00	00	00	00	00
LDAA $0031	0023	0031			14					
CALL $0027	0027		004C				~~24~~	26	00	
STAA 07,X	0029	0039							1	
RET	0026		004A							
STAR D	0027					14				

* 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	×	×	✓	✓	A ← mem[PC]	LDAI <data>
Load Acc (Abs)	88	×	×	✓	✓	A ← mem(mem[PC])	LDAA <address>
Load Acc (RR)	81	×	×	✓	✓	A ← (D)	LDAD
Store Acc (Abs)	F6	×	×	×	×	Mem(mem[PC]) ← (A)	STAA <address>
Store Acc (RR)	F1	×	×	×	×	D ← (A)	STAR D
Add with Carry	01	✓	✓	✓	✓	A ← (A) + (D) + C	ADCR D
Subtract with Borrow	11	✓	✓	✓	✓	A ← (A) + not(D) + C	SBCR D
Compare	91	✓	✓	✓	✓	Same as Subtract, but only change Status Flags (A is unchanged)	CMPR D
AND	21	×	×	✓	✓	A ← (A) AND (D)	ANDR D
OR	31	×	×	✓	✓	A ← (A) OR (D)	ORR D
XOR	41	×	×	✓	✓	A ← (A) XOR (D)	XORR D
Shift Left Logical	51	✓	×	✓	✓	C ← (A7), A7 ← A6, …, A0 ← 0	SLRL
Shift Right Logical	61	✓	×	✓	✓	C ← (A0), A0 ← A1, …, A7 ← 0	SRRL
Rotate Left through Carry	52	✓	×	✓	✓	C ← (A7), A7 ← A6, …, A0 ← C	ROLC
Rotate Right through Carry	62	✓	×	✓	✓	C ← (A0), A0 ← A1, …, A7 ← C	RORC
Branch on /C (Inh)	B0	×	×	×	×	if (C=0), PC ← mem[PC] else PC++	BCCA
Branch on C (Inh)	B1	×	×	×	×	if (C=1), PC ← mem[PC] else PC++	BCSA
Branch on Z (Inh)	B2	×	×	×	×	if (Z=1), PC ← mem[PC] else PC++	BEQA
Branch on S (Inh)	B3	×	×	×	×	if (S=1), PC ← mem[PC] else PC++	BMIA
Branch on /Z (Inh)	B4	×	×	×	×	if (Z=0), PC ← mem[PC] else PC++	BNEA
Branch on /S (Inh)	B5	×	×	×	×	if (S=0), PC ← mem[PC] else PC++	BPLA
Branch on /V (Inh)	B6	×	×	×	×	if (V=0), PC ← mem[PC] else PC++	BVCA
Branch on V (Inh)	B7	×	×	×	×	if (V=1), PC ← mem[PC] else PC++	BVSA
Decrement Acc	F8	×	×	✓	✓	A ← (A) - 1	DECA
Increment Acc	FA	×	×	✓	✓	A ← (A) + 1	INCA
Set Carry Flag	F8	✓	×	×	×	C ← 1	SETC
Clear Carry Flag	F9	✓	×	×	×	C ← 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	SP ← mem[PC+1],mem[PC]	LDSI <data>
Call	C8	X	X	X	X	SP ← (SP) + 1; mem[SP] ← (PC_L); SP ← (SP) + 1; mem[SP] ← (PC_H)	CALL <address>
Return	C0	X	X	X	X	PC_H ← mem[SP]; SP ← (SP) – 1; PC_L ← mem[SP]; SP ← (SP) – 1	RET
Index addressing:							
Load X (Imm)	8A	X	X	X	X	X ← mem[PC+1],mem[PC]	LDXI <data>
Load Acc (Indx)	BC	X	X	X	X	A ← mem[(X) + b]	LDAA b,X
Store Acc (Indx)	EC	X	X	X	X	mem[(X) + b] ← (A)	STAA b,X
Increment X	FC	X	X	X	X	X ← (X) + 1	INCX
Decrement X	FD	X	X	X	X	X ← (X) – 1	DECX

 Name _____

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