**Important:** Please be neat and write (or draw) carefully. If we cannot read it with a reasonable effort, it is assumed wrong.

**Small8-related problem:**

Shown below is a modified architecture of the Small8 computer. Assume that all the components are the same as you had them in your mini-project with the following exceptions:

- The outputs of AR, PC, SP, and IX 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 flag register called tempC (temporary carry), with synchronous LD and CLR.
- Registers Temp1 and Temp2 are connected as shown, both with synchronous LD and synchronous CLR inputs.

---

**Synchronous LD:** all registers and flipflops

**Synchronous CLR:** IX, Temp1, Temp2, tempC

**OE (output enable):** ARH, ARL, PCH, PCL, IXH, IXL

**INC (increments the 16-bit register by 1):** PC, SP.

**DEC (decrements the 16-bit SP register by 1):** SP.

---

(See next page for ALU functions)
Timing requirements for the read and write operations of the asynchronous RAM:

```
CLK
ExAD
Din
Dout
MemEN
MemOE
MemWE

0101 0102 0103 0104 0105
ZZ    ZZ    ZZ    ZZ    ZZ
ZZ    ZZ    ZZ

Read outdata from loc. 0101
Write indata to loc. 0104
```

## ALU functions

<table>
<thead>
<tr>
<th>SEL</th>
<th>Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>Zero (F = 0)</td>
</tr>
<tr>
<td>001</td>
<td>Add with carry (F = X + Y + Cin)</td>
</tr>
<tr>
<td>010</td>
<td>Subtract (X-Y) with borrow (F = X + /Y + Cin)*</td>
</tr>
<tr>
<td>011</td>
<td>Shift right (F = X &gt;&gt; 1), Cin&gt;&gt;F7, F0&gt;&gt;Cout</td>
</tr>
<tr>
<td>100</td>
<td>Shift left (F = X &lt;&lt; 1), F0&lt;&lt;Cin, Cout&lt;&lt;F7</td>
</tr>
<tr>
<td>101</td>
<td>Bit-wise XOR (F = X XOR Y)</td>
</tr>
<tr>
<td>110</td>
<td>Bit-wise AND (F = X AND Y)</td>
</tr>
<tr>
<td>111</td>
<td>Bit-wise OR (F = X OR Y)</td>
</tr>
</tbody>
</table>

* The Cin for subtraction is the borrow input.

1. You are to implement three new instructions for the Small8 computer by completing the ASM diagram on the next page: PUSH A (opcode: $70$), POP A (opcode: $71$), and ADD S (72).

<table>
<thead>
<tr>
<th>16 pts.</th>
</tr>
</thead>
<tbody>
<tr>
<td>PUSH A</td>
</tr>
<tr>
<td>POP A</td>
</tr>
<tr>
<td>ADD S</td>
</tr>
</tbody>
</table>

NOTE: The PUSH A and POP A instructions should use the stack in a way that is consistent with the CALL and RET instructions defined in the Small8 instruction set on the last page of test (i.e., when to increment and decrement the SP).
(Put answer for Problem 1 here.)

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

“Fetch opcode”

PC.INC

(You don’t have to implement the opcode fetch state.)

opcode

(70) PUSH A

(72) ADD S

(71) POP A . . .
2. Complete the following ASM diagram to fully implement the STAA b,X instruction. For maximum credit, optimize your ASM diagram. To refresh your memory, the STAA b,X is a 2-byte instruction and is defined as follows:

\[ \text{MEM}(EA) \iff A \] 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 STAA b,X opcode.

Ex: If IX= $21FF; b= $04. Then EA= $21FF + $04 = $2203

**Important Notes:**
- IX is a 16-bit storage register, with a LD input (cannot be incremented or do addition).
- Both C and tempC flags are flip-flops whose inputs are from the Cout of the ALU. Both the C and tempC have a LD input.
- STAA b,X affects no user flags at the end of its execution.

(For maximum partial credit, “comment” your ASM diagram.)
3. **Small8 Program Execution** – at the *instruction* level

Given the following .mif file and use the architecture shown in Page 1.

<table>
<thead>
<tr>
<th>Address</th>
<th>Content</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>84</td>
</tr>
<tr>
<td>0001</td>
<td>F6</td>
</tr>
<tr>
<td>0002</td>
<td>F6</td>
</tr>
<tr>
<td>0003</td>
<td>07</td>
</tr>
<tr>
<td>0004</td>
<td>10</td>
</tr>
<tr>
<td>0005</td>
<td>01</td>
</tr>
<tr>
<td>0006</td>
<td>70</td>
</tr>
<tr>
<td>0007</td>
<td>FA</td>
</tr>
<tr>
<td>0008</td>
<td>EC</td>
</tr>
<tr>
<td>0009</td>
<td>08</td>
</tr>
<tr>
<td>00A</td>
<td>C0</td>
</tr>
</tbody>
</table>

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

- **PUSH A**: 70 (increment SP before pushing)
- **POP A**: 71 (pop before decrement SP)
- **CALL addr16**: C8
- **RET**: C0

The initial values of memory locations: 1000 = 90, 1001 = 91, 1002 = 92, … 1008 = 98, 1009 = 99. (All values are in hex.)

If any initial values are not specified, assume it is 0.

Analyze the above .mif file and specify the content of each of the register or memory location (in hex). 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, **draw an arrow** to indicate no change to a register (e.g., See register D)

<table>
<thead>
<tr>
<th>PC</th>
<th>AR</th>
<th>SP</th>
<th>IX</th>
<th>A</th>
<th>D</th>
<th>Changed mem location = new value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>1000</td>
<td>1006</td>
<td>1000</td>
<td>00</td>
<td>03</td>
<td>You should put something like:</td>
</tr>
</tbody>
</table>

mem(1004) = 73
4. **VHDL specification**

PD is a D flip-flop (with an exclusive OR gate) and synchronous LD and CLR. CLR has priority over LD.

SR is a 64-bit shift register: If SHF = 1, it will synchronously shift right, else “hold”.

**NOTES:**

- Complete the VHDL code on the next page to define the above circuit.
- All your code in the architecture *must* be inside the *one PROCESS statement*.
- The best answer gets the most points.
4 (continued). Put answer for Problem 4 here:

ENTITY Prob4Circuit IS
    PORT( LD, CLR, INP, CLK, SHF : IN STD_LOGIC;
         PAR, Z : OUT STD_LOGIC);
END Prob4Circuit;
ARCHITECTURE behaviorArch OF Prob4Circuit
BEGIN

PROCESS (  
    -- All your code must be inside this one PROCESS statement.

END PROCESS;
END behaviorArch;
Complete the following VHDL code to define the above circuit.

- You **must** use behavioral VHDL (i.e., you **cannot** use PORT MAP statements.).
- All code should be within one PROCESS block.

ENTITY BusStruc IS
  PORT(
  );
END BusStruc;

ARCHITECTURE behaviorArch OF BusStruc
SIGNAL
BEGIN
PROCESS ( )
END PROCESS;

END behaviorArch;
6. Address decoding

(a) What is the size of RAM-A? __________ RAM-B? __________ RAM-C? __________

(b) What is the 16-bit address of the first location in RAM-C?

______________________________ (in binary) ________ (in hex)

(c) What is the 16-bit address of the last location in RAM-C?

______________________________ (in binary) ________ (in hex)

(d) What are the two 16-bit addresses of the first location in RAM-A?

______________________________ (in binary) ________ (in hex)

(e) What is the 16-bit address of the first location in RAM-B?

______________________________ (in binary) ________ (in hex)

(f) To make the appropriate connections (and logic) to INPORT so that any of the following instructions (and only those addresses) will load data from INPORT to register A, what is the logic expression for OE?

LDAA $4FFF, LDAA $5FFF, LDAA $6FFF, or LDAA $7FFF.

OE =
7. EEL4712 Trivia

(a) Explain how ModelSim, logic state analyzer, and Quartus SignalTap are similar. (2 pts)

(b) Explain the difference between ModelSim, logic state analyzer, and SignalTap. (3 pts)

(a) Compare FPGA-based reconfigurable computing (RC) to design/implementation using microprocessor (uP) and ASIC; i.e., rank them 1 (highest), 2 or 3 (lowest) - (3 pts)

<table>
<thead>
<tr>
<th></th>
<th>RC</th>
<th>uP</th>
<th>ASIC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Performance:</td>
<td>___</td>
<td>___</td>
<td>_____</td>
</tr>
<tr>
<td>Power consumption:</td>
<td>___</td>
<td>___</td>
<td>_____</td>
</tr>
<tr>
<td>Development cost:</td>
<td>___</td>
<td>___</td>
<td>_____</td>
</tr>
<tr>
<td>Part cost:</td>
<td>___</td>
<td>___</td>
<td>_____</td>
</tr>
<tr>
<td>Design flexibility:</td>
<td>___</td>
<td>___</td>
<td>_____</td>
</tr>
<tr>
<td>Fast time to market:</td>
<td>___</td>
<td>___</td>
<td>_____</td>
</tr>
</tbody>
</table>

(b) What is synthesizable VHDL? Give me an example where VHDL code is not synthesizable. (2 pts)
### Table 1: Small/S Instruction Set

#### Addendum to the instruction set

<table>
<thead>
<tr>
<th>INSTRUCTION</th>
<th>OP CODE</th>
<th>C</th>
<th>V</th>
<th>Z</th>
<th>S</th>
<th>DESCRIPTION</th>
<th>SYNTAX</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Subroutines:</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Load SP (Imm)</td>
<td>89</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td>SP ← mem[PC+1],mem[PC]</td>
<td>LDSI &lt;data&gt;</td>
</tr>
<tr>
<td>Call</td>
<td>C8</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td>SP ← (SP) + 1; mem[SP] ← (PC); SP ← (SP) + 1; mem[SP] ← (PCₙ)</td>
<td>CALL &lt;address&gt;</td>
</tr>
<tr>
<td>Return</td>
<td>C0</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td>PCₙ ← mem[SP]; SP ← (SP) - 1; PC ← mem[SP]; SP ← (SP) - 1</td>
<td>RET</td>
</tr>
<tr>
<td><strong>Index addressing:</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Load X (Imm)</td>
<td>8A</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td>X ← mem[PC+1],mem[PC]</td>
<td>LDXI &lt;data&gt;</td>
</tr>
<tr>
<td>Load Acc (Indx)</td>
<td>BC</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td>A ← mem(X + b)</td>
<td>LDA A b,X</td>
</tr>
<tr>
<td>Store Acc (Indx)</td>
<td>EC</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td>mem(X + b) ← (A)</td>
<td>STA A b,X</td>
</tr>
<tr>
<td>Increment X</td>
<td>FC</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td>X ← (X) + 1</td>
<td>INCX</td>
</tr>
<tr>
<td>Decrement X</td>
<td>FD</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td>X ← (X) - 1</td>
<td>DECX</td>
</tr>
</tbody>
</table>
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;

SIGNAL __signal_name : __type_name;
__instance_name: __component_name
GENERIC MAP (__component_par => __connect_par)
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;

WAIT UNTIL __expression;

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

Problem: Points:

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>(16 pts)</td>
</tr>
<tr>
<td>2</td>
<td>(16 pts)</td>
</tr>
<tr>
<td>3</td>
<td>(16 pts)</td>
</tr>
<tr>
<td>4</td>
<td>(14 pts)</td>
</tr>
<tr>
<td>5</td>
<td>(14 pts)</td>
</tr>
<tr>
<td>6</td>
<td>(14 pts)</td>
</tr>
<tr>
<td>7</td>
<td>(10 pts)</td>
</tr>
<tr>
<td>Total:</td>
<td></td>
</tr>
</tbody>
</table>