Q1. Evaluate the following (10 marks, 2 marks each),

Solutions:

(a) \( 90_{10} = X_2, \Rightarrow X = 1011010_2 = 1 \times 64 + 0 \times 32 + 1 \times 16 + 1 \times 8 + 0 \times 4 + 1 \times 2 + 0 \times 1 \).

(b) \( 23_{16} = Y_5, \Rightarrow Y = 120_5 \)

(c) \( 0111_2 + 111_2 = 1110_2 \)

(d) \( 0101_2 \times 0011_2 = 1111_2 \)

(e) The C-language expression

\[ \text{printf}(''%d
'',X); \] where signed char \( X = 182; \)

\( \text{signed char} \) in C is a signed 8-bit 2’s complement number where the maximum positive number is \( 01111111 = 127_{10} \) and the minimum is \( 11111111 = -128_{10} \). Thus the integer (format \( \%d \) in C) \( 128_{10} = -128_{10} \) and \( 182_{10} = -74_{10} \)

Q2. MU0 Microprocessor (10 marks).

Consider the following MU0 machine language program.

0004 2005 1006 7000 000A 0001 0000

Using the figures on page 2 answer the following.

(a) Describe in words what happens on each of the lines MU0 assembly language program (3 marks).

In the Von Neumann architecture, lines representing program instructions proceed in memory those representing data. In the instructions the first hex number (4-bits) is the opcode and the last three hex numbers (12-bits) are the address in memory of the operand (except for jump instructions where it is the address of the next program instruction in memory).

From the table overleaf, in 0004 the first 0 refers to the LDA instruction and the 004 refers to the memory location at memory location 4 or 000A. 0004 means load 000A into the accumulator.

2005: add the contents of memory location 005 to the contents of the accumulator. Memory address 005 contains 0001.

1006: store the contents of the accumulator in address 006.

7000: stop

(b) What is the contents of address 0006 after execution of the program (2 marks).

Address 0006 contains 000A+0001 = 000B.

(c) Draw a timing diagram showing the contents of the program counter (PC), the instruction register (IR) and the accumulator (ACC) over the two clock periods where the instruction 2005 is being executed. Label the timing diagrams with the trace values in hexadecimal. (5 marks).
Q3. Logic Synthesis and Karnaugh Maps  (10 marks, 2 marks each)

A function generator consists of a counter that drives a decoder. The counter counts up from 0-15 repetitively. The decoder inputs the 4 bit address from the counter and outputs a 2-bit ramp wave. For counter values (0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15)

the decoder produces, (00,01,10,11,00,01,10,11,00,01,10,11,00,01,10,11).

The decoder can be described by a truth table with 16 × 4 bit inputs and 16 × 2 bit outputs. Perform the following.

(1) Draw the truth table of the decoder.

<table>
<thead>
<tr>
<th>XYZW</th>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0001</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0010</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0011</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0100</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0101</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0110</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0111</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1000</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1001</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1010</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1011</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1100</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1101</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1110</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1111</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

(2) Draw the Karnaugh maps from the truth table. (see below)

(3) On the Karnaugh maps loop out the logical adjacencies. (see below)

(4) Write down each of the minimised Boolean logic expressions describing the decoder.

\[ A = Z \text{ and } B = W \]

(5) Draw the circuit schematic of the decoder. (see below)
Q4. VERILOG (10 marks)

(1) Explain the difference between blocking and non-blocking assignments. (4 marks)

A blocking procedural assignment (=) is evaluated and assigned in a single step. Blocking assignments are executed sequentially. Execution flow within the procedure is blocked until the assignment is completed. Evaluation of concurrent statements in the same time step are blocked until the assignment is completed.

In this course we use blocking assignments to model combinational circuits by disallowing the appearance of the same variable on both sides of the = sign. The blocking nature forces the outputs on the left hand side to settle to the inputs on the right hand side as would be expected for combinational logic.

Circuits designed around blocking assignments to and from the same variable on both sides of the = sign are used to produce sequential circuits that transition on propagation delays (controlled race conditions). Care has to be exercised in the design of these circuits.

Run the following modules to see the operation of blocking statements in a general setting.

```verilog
module blocking;

reg clk;
reg X;
reg Y;
reg Z;

initial begin
    X = 0;
    clk = 0;
    #1clk = 0;
    #1clk = 1;
    #1clk = 0;
    #1clk = 1;
    #1clk = 0;
    #1clk = 1;
    #1clk = 0;
    #1clk = 1;
end

initial begin
    $dumpfile("blocking.vcd");
    $dumpvars;
end

//Try swapping the combinational always blocks
always @(*) begin
    Y = X;
end
```
always @(*) begin
  Z = Y;
end
always @(posedge clk) begin
  X <= ~X;
end
endmodule

module blocking;

reg clk;
reg X;
reg Y;
reg Z;

initial begin
  X = 0;
  clk = 0;
  #1clk = 0;
  #1clk = 1;
  #1clk = 0;
  #1clk = 1;
  #1clk = 0;
  #1clk = 1;
  #1clk = 0;
  #1clk = 1;
end

initial begin
  $dumpfile("blocking.vcd");
  $dumpvars;
end

always @(*) begin
  //Try swappinng these...
  Y = X;
  Z = Y;
end
always @(posedge clk) begin
  X <= ~X;
end
endmodule
Nonblocking assignments ($<=$) are evaluated and assigned in two steps: the right-hand side is evaluated immediately. The assignment to the left-hand side is postponed until other evaluations in the current time step are completed.

We use non-blocking assignments to model sequential always blocks in which the sensitive list has a posedge or negedge. In this way the left hand sides are assigned to the value the right hand sides had before the active edge transition as would be expected for a sequential circuit.

Question 4 part 2 below shows another example.

(2) Draw a timing diagram starting at t = 0 and running for four clock periods showing the values of X, Y, Z and clk produced by the following VERILOG. (6 marks)

```verilog
datamodule bmb;
  reg clk;
  reg X;
  reg Y;
  reg Z;
  initial begin
    X = 0;
    clk = 0;
    forever #1 clk = clk;
  end
  always @(posedge clk) begin
    Y <= X;
    X <= ~X;
  end
  always @(*) begin
    Z = X;
  end
endmodule
```
ADD

Fetch
Data Out
Address
Data In

Decode/Execute
Data Out
Address
Data In

Timing and Control

FSM State Transition Table

<table>
<thead>
<tr>
<th>State</th>
<th># state</th>
<th>Next</th>
<th>BEn</th>
<th>PEn</th>
<th>AccEn</th>
<th>M[10]</th>
<th>Xsel</th>
<th>Ysel</th>
<th>Asel</th>
<th>Rtn</th>
<th>Wtn</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>xxx</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>10</td>
<td>0</td>
<td>x</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>000</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>00</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>001</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>x</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>010</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>01</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>011</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>100</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>00</td>
<td>0</td>
<td>0</td>
<td>x</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>101</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>00</td>
<td>0</td>
<td>0</td>
<td>x</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>110</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>00</td>
<td>0</td>
<td>0</td>
<td>x</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>111</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>xx</td>
<td>0</td>
<td>x</td>
<td>x</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Notes:
- N and Z are the Negative and Zero state of the Accumulator, respectively.
- (used to reduce the size of the table, as drawn)
- If a value is not going to be latched it doesn’t matter what it is!
  (e.g. ALU output for STO)
- STP operates by remaining in its evaluation state.

MU0 Instruction Set

<table>
<thead>
<tr>
<th>F</th>
<th>Mnemonic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>LDA</td>
<td>Acc := [S]</td>
</tr>
<tr>
<td>1</td>
<td>STO</td>
<td>[S] := Acc</td>
</tr>
<tr>
<td>2</td>
<td>ADD</td>
<td>Acc := Acc + [S]</td>
</tr>
<tr>
<td>3</td>
<td>SUB</td>
<td>Acc := Acc − [S]</td>
</tr>
<tr>
<td>4</td>
<td>JMP</td>
<td>PC := S</td>
</tr>
<tr>
<td>5</td>
<td>JGE</td>
<td>If Acc ≥ 0, PC := S</td>
</tr>
<tr>
<td>6</td>
<td>JNE</td>
<td>If Acc ≠ 0, PC := S</td>
</tr>
<tr>
<td>7</td>
<td>STP</td>
<td>Stop</td>
</tr>
</tbody>
</table>