EEL 4712
Midterm 3 – Spring 2010
VERSION 1

Name: ______________________________

UFID:______________________________

Sign your name here if you would like for your test to be returned in class:

____

IMPORTANT:
• Please be neat and write (or draw) carefully. If we cannot read it with a reasonable effort, it is assumed wrong.
• **As always, the best answer gets the most points.**

COVER SHEET:

Problem#:	**Points**
1 (5 points)	
2 (5 points)	
3 (6 points)	
4 (10 points)	
5 (12 points)	
6 (5 points)	
7 (5 points)	
8 (6 points)	
9 (15 points)	
10 (27 points)	
11 (4 free points)	4

Total:

Regrade Info:

```
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;
BEGIN
-- Process Statement
-- Concurrent Signal Assignment
-- Conditional Signal Assignment
-- Selected Signal Assignment
-- Component Instantiation Statement
END a;

__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;

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

1) (5 points) What is the difference between the carry and overflow status flag in the small8 microprocessor? Note: you *do not* have to specify how to implement the overflow flag.

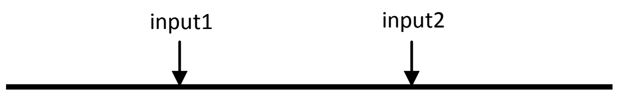
2) (5 points) To guarantee that the subtract with borrow instruction does not perform a borrow, what instruction must precede the subtract instruction in the assembly code?

3) (6 points) Describe the functionality of the instructions required for calling and returning from a function in the Small8 microprocessor. Include a discussion of how the stack pointer is used. For example, how are these instructions different from a normal branch?

4) (10 points) Write the assembly code to implement the following code, using the labels shown below. Assume n is stored at address 0x0200. Make sure to load n from memory before execution and store it back to memory after execution. You only need to show the actual instructions, not the entire assembly program (e.g. lab test cases).

```
if (n < 128) {
   IF_BODY: n = n + 1;
}
AFTER_IF:
```

5) a. (6 points) Create a structural architecture for the bus structure shown below using the provided tri-state components (output enable signals not shown).



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

entity bus_tristate is
  port (
    input1, input2 : in  std_logic_vector(31 downto 0);
    output_enable  : in  std_logic_vector(1 downto 0);
    output         : out std_logic_vector(31 downto 0));
end bus_tristate;

architecture STR of bus_tristate is

  component tristate
    port (
      input  : in  std_logic_vector(31 downto 0);
      enable : in  std_logic;
      output : out std_logic_vector(31 downto 0));
  end component;

begin


  U_TRISTATE1 : tristate port map (

    input  =>

    enable =>

    output =>

    );

  U_TRISTATE2 : tristate port map (

    input  =>

    enable =>

    output =>

    );

end STR;
```

b. (6 points) Create a structural architecture for the bus structure in part a, using a *when-else* or similar construct (i.e., model the bus as a mux).

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

entity bus_mux is

  port (
    input1, input2 : in  std_logic_vector(31 downto 0);
    output_enable  : in  std_logic_vector(1 downto 0);
    output         : out std_logic_vector(31 downto 0));
end bus_mux;

architecture BHV of bus_mux is
begin









end BHV;
```

6) (5 points) Why do FPGA synthesis tools (e.g. Quartus, ISE) convert all tri-state buffers to muxes?

7) (5 points) Describe the purpose of the index addressing instructions. You do not need to explain each instruction. Instead, describe an appropriate usage scenario for the instructions.

8) (6 points) Describe the basic functionality of an opcode fetch, using notation similar to the "description" column in the instruction set table.

9) (15 points) Create a memory initialization file for the following assembly code. Add a comment at the beginning of each instruction. *You will likely need to break your answer up into two columns to fit on the page.*

```
INPORT0      EQU    $FFFE
INPORT1      EQU    $FFFF
OUTPORT0     EQU    $FFFE

BEGIN:
       LDAA   INPORT0
       STAR   D
       LDAA   INPORT1
       ADCR   D
       STAR   D
       LDAA   VALUE1
       ADCR   D
       STAA   OUTPORT0
INFINITE_LOOP:
       CLRC
       BCCA   INFINITE_LOOP

* Data Area
VALUE1:      dc.b   $55

       END    BEGIN
```

```
Depth = 256;
Width = 8;
Address_radix = hex;
Data_radix = hex;
% Program RAM Data %
Content
  Begin
```

```
[       ..00FF] : 00;
End;
```

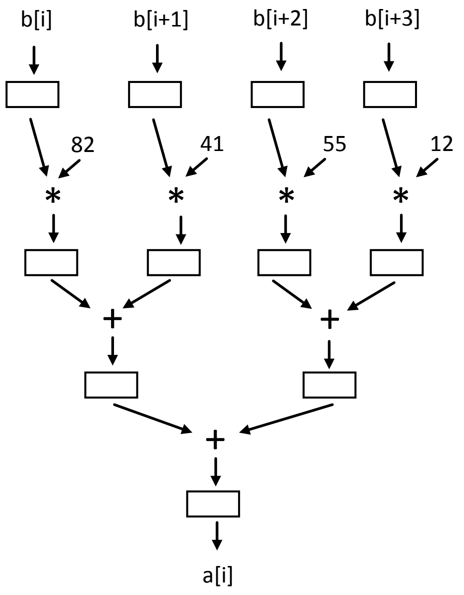
10) a. (5 points) For the following pseudo-code and schedule for the body of the loop (i.e., a set of states in the controller), what is the *minimum* number of adders and multipliers required by the datapath to execute this schedule, assuming each state corresponds to a different cycle? *Do not change the schedule (it is purposely suboptimal).*

```
for (i=0; i < 100000; i++) {
        a[i] = b[i]*82 + b[i+1]*41 + b[i+2]*55 + b[i+3]*12;
}
```

1. Load b[i]
2. Load b[i+1]
3. Load b[i+2], Multiply1 (b[i]*82), Multiply2 (b[i+1]*41)
4. Load b[i+3], Add1 (Multiply1 + Multiply2)
5. Multiply3 (b[i+2] *55), Multiply4 (b[i+3]*12)
6. Add2 (Multiply3 + Multiply4),
7. Add3 (Add1+Add2)
8. Store a[i]

b. (5 points) For the schedule in part a, what is the approximate execution time of the loop in terms of number of cycles? You can ignore initialization states.

c. (6 points) For the following pipelined datapath, what is the approximate performance of the loop in terms of number of cycles? Ignore initialization states. Assume memory bandwidth is sufficient to handle all datapath inputs/outputs.



d. (5 points) For a microprocessor with a 10x faster clock and a CPI (cycles per instruction) of 1.5, what is the approximate speedup of the pipelined circuit from part c? Assume the microprocessor requires 15 instructions for the body of the loop.

e. (6 points) Given an input memory that can deliver 128 bits per cycle, how many iterations of the loop can be done in parallel before a replicated pipeline will stall? Assume each element of b[] is 16 bits.