Name:

Problem	Possible	Score
1	15	14
2	15	12
3	15	15
4	15	15
5	20	15
6	20	18
Total	100	89

General information that may be useful sometime during test:

Table of Powers of Two

N	2^N		N	2^N		N	2^N		N	2^N
0	1		8	256		16	65,536		24	16,777,216
1	2		9	512		17	131,072		25	33,554,432
2	4		10	1,024		18	262,144		26	67,108,864
3	8		11	2,048		19	524,288		27	134,217,728
4	16		12	4,096		20	1,048,576		28	268,435,456
5	32		13	8,192		21	2,097,152		29	536,870,912
6	64		14	16,384		22	4,194,304		30	1,073,741,824
7	128		15	32,768		23	8,388,608		31	2,147,483,648

1. The general representation question: Give the correct pattern (in binary) for the following numbers:

Coding method	Base 10 value	Answer
Unsigned Binary (to 10 bits)	801	1100100001
Two's Complement (to 10 bits)	-441	0110111001 / 1001000111
Two's Complement (to 16 bits, 8 fractional bits)	-38.3125	0010011010010000 / 1101100101110000 (-1)
Excess 127 code	41	01010100
Excess 127 code	-41	01010110

2. General knowledge question:

a) Give the register changes involved in the execution of the *lw $s1,100($s2)* instruction. That is, give a RTL version of the work needed to complete the instruction.

Fetch
MAR <= PC
IR <= m[MAR]
PC <= PC+4

Execute
$s1 <= m[SignExtend(100) + $s2]

b) What is meant by the term 'forwarding'?

Making the result of the ALU available to the next instruction if the next instruction requires that result, so that the next instruction doesn't need to stall and wait for the result to have been stored to a register. So your "forwarding" the write back result to just before the execution stage. (and what else -1)

c) Explain what is meant by the term *hazard*, and describe where it is found.

Hazard is when an instruction can't run during the clock cycle it is meant to because it could cause an error. The three hazards are: structural (resource isn't available at a time it is needed), control (conditional branch), and data (RAW, etc.).

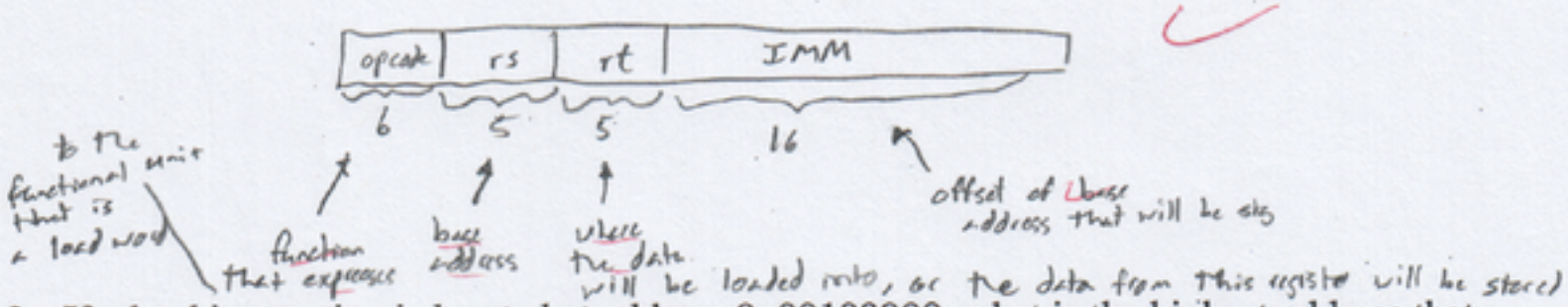
d) When we discussed the various aspects of pipelining, we determined that a pipeline with N stages had a speedup of N under what conditions? That is, what two conditions must be present in the system for the real speedup to approach N?

?? - No hazards - -2

$$\frac{N}{\left(1+\frac{N-1}{m}\right)\left(1+\frac{T_{REG}}{T_{process}}\right)}$$

Answer:
- T_{REG} is negligible
- m is very big

e) Give the format (bit-pattern-wise, not assembly-language-syntax-wise) for a load or store instruction, and explain the function of each of the fields.



f) If a load instruction is located at address 0x00100000, what is the highest address that can be reached by that instruction?

0x 00 100000
0x 00007FFF ← might be 0x7FFC
0x 107FFF (C) 2

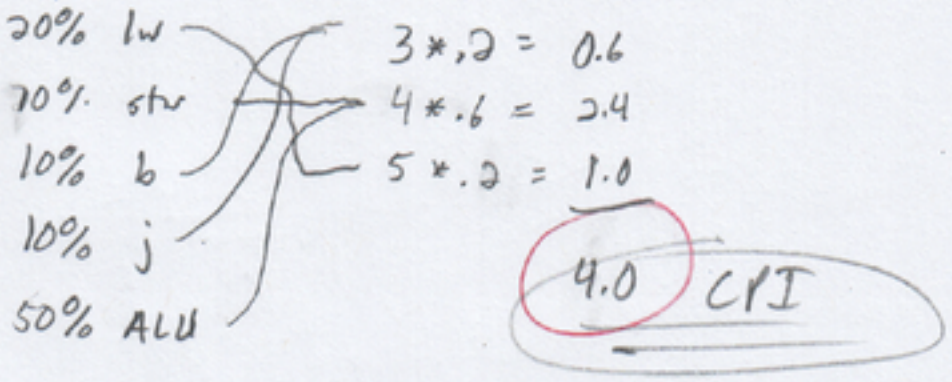
Depending on whether cross-word boundaries are allowed

3. Among the implementations studied this semester is the multi-cycle implementation of the MIPS architecture, which we used as a starting point for moving into the pipelined version of the system. For a MIPS implementation that utilizes a multi-cycle data path, five cycles are required for loads, four cycles for stores and ALU instructions, and 3 cycles for branches and jumps. Assume that a program is executed that has 20% loads, 10% stores, 10% branches, 10% jumps, and the rest ALU instructions. What is the CPI for the machine when it is running this program?

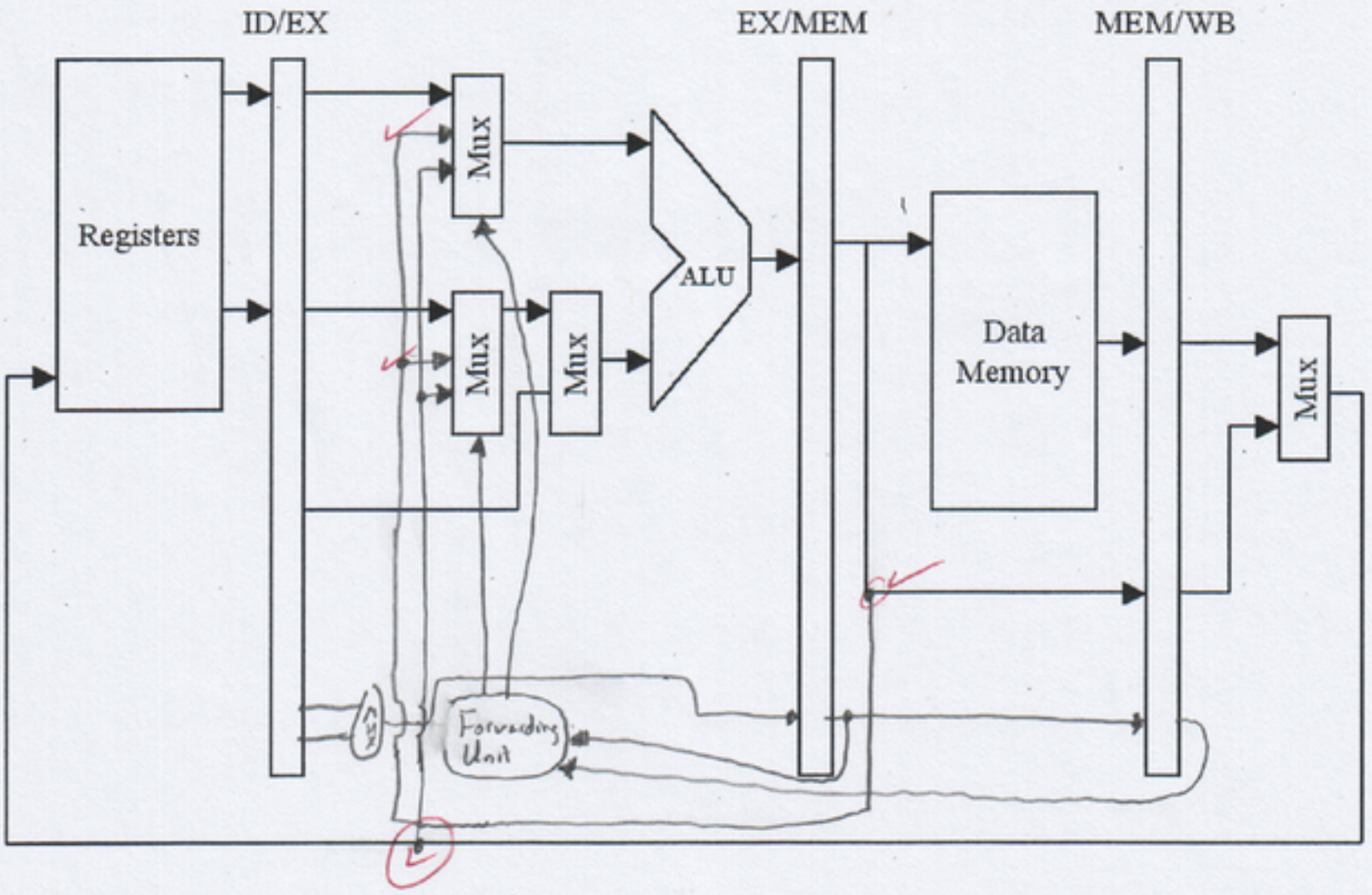
3 cycles - b, j

4 cycles - stw, ALU

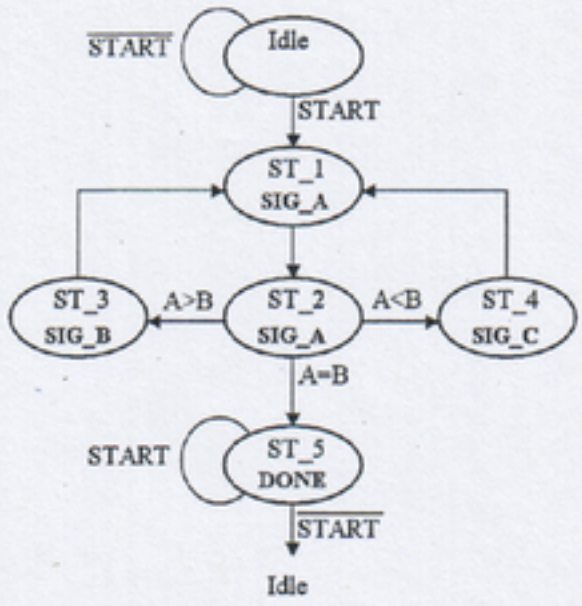
5 " - lw



4. System design question: We have discussed pipelining and the need for making available as quickly as possible the results of instructions. In particular, we need to make available the results of instructions ahead of the current instruction so that the pipeline doesn't stall. Below is a version of Figure 4.54 in the text, which shows how to do forwarding – the name of this technique. However, there are some buses missing. On the figure below, add the necessary data paths so that the system will be capable of forwarding values from one instruction ahead and two instructions ahead.



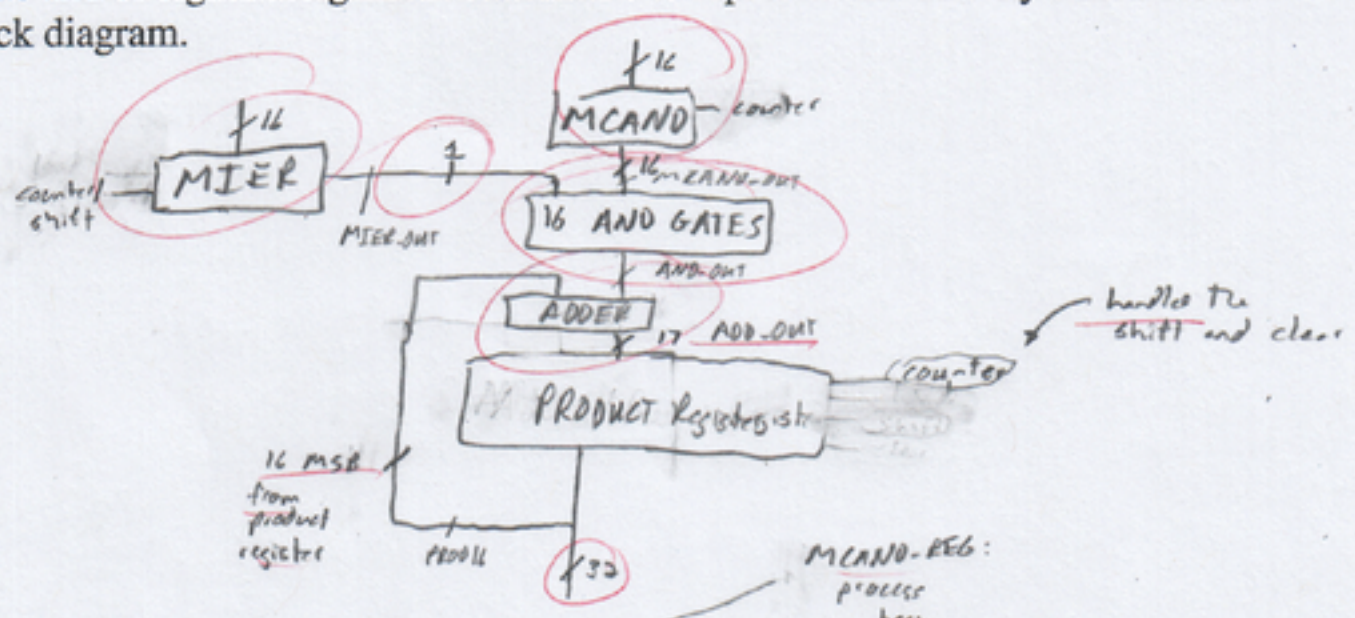
5. In our study of the MIPS architecture this semester, we utilized a multi-cycle mechanism in order to study one implementation method. The control system for that implementation used a state machine method for implementation. In the space provided below, give a VHDL architecture (don't worry about entity) for the state diagram shown. Included in the actual entity are the clock and reset signals not indicated on the state diagram, as well as the inputs shown (START – 1 bit – and A and B – both 8 bits). The output signals are indicated in the diagram (SIG_A, SIG_B, SIG_C, DONE). So, what is needed is an architecture that contains the VHDL description of a state machine that does this work.




```
type PSR_type is (idle, st_1, st_2, st_3, st_4, st_5);
signal PSR : PSR_type;

architecture of FOR_CLASS is
begin

PSR_FUNC:
    process (SYS_CLK, SYS_RST, A, B, Start) is
    begin
        if (SYS_RST = '1') then
            PSR <= idle;
            SIG_A <= (others => '0');
            SIG_B <= (others => '0');
            SIG_C <= (others => '0');
            DONE <= '0';
    elsif RISING_EDGE (SYS_CLK) then
        case PSR is
            when Idle => if Start = '1' then
                             PSR <= st_1;
                         else
                             PSR <= idle;
                         end if;
            when st_1 => sig_A <= '1';
                         PSR <= st_2;
            when st_2 => sig_A <= '1';
                         if A > B then
                             PSR <= st_3;
                         elsif A < B then
                             PSR <= st_4;
                         else
                             PSR <= st_5;
                         end if;
            when st_3 => sig_B <= '1';
                         PSR <= st_1;
            when st_4 => sig_C <= '1';
                         PSR <= st_1;
            when st_5 => Done <= '1';
                         if Start = '0' then
                             PSR <= idle;
                         else
                             PSR <= st_5;
                         end if;
        end case;
    end if;
    end process;

end architecture;
```

but you also have to undo sig A

6. System design question: We have discussed during the semester that all non-trivial digital systems have a data path part and a control part. This question is a data path only question. In the space provided below, provide a block diagram of a data path for implementing a basic shift-and-add unsigned multiply, using the simplest algorithm. The inputs to the module are the multiplier (16 bits) and the multiplicand (16 bits), and the output is the product (32 bits). Also to be included on the block diagram are the control points needed to carry out the work of the algorithm. On the next page, write the VHDL needed for the data path (with its control lines) as an entity-architecture pair. This should be a behavioral implementation, not a structural one. That is, do not use external components for the blocks of your diagram. Rather, use process statements and/or signal assignment statements to represent the activity called for in your block diagram.




```
entity multiplier is
    ( MCAND : in std_logic_vector (15 downto 0);
      MIER  : in std_logic_vector (15 downto 0);
      PROD  : out std_logic_vector (31 downto 0);
      SYS_CLK : in std_logic;
      SYS_RST : in std_logic
    );

signal load, shift, clear : std_logic;
signal counter : std_logic_vector (3 downto 0);
signal MIER_SHF, MCAND_OUT, AND_OUT, PROD16 : std_logic_vector (15 downto 0);
signal ADD_OUT : std_logic_vector (16 downto 0);
signal MIER_OUT : std_logic;
architecture of FOR_CLASS is

begin

  MIER_REG:
      process
      begin
        if SYS_RST = '1' then
            MIER <= (others => '0');
        elsif RISING_EDGE(SYS_CLK) then
            MIER <= '0' & MIER(15 downto 1);
            MIER_OUT <= MIER(0);
        end if;
      end process;

  AND_OUT <= MCAND_OUT when MIER_OUT = '1' else
             (others => '0');
```

```
MCAND_REG:
  process
  begin
    if SYS_RST = '1' then
        MCAND <= (others => '0');
    elsif RISING_EDGE(SYS_CLK) then
        if counter = "0000" then
            MCAND_OUT <= MCAND;
        end if;
    end if;
  end process;

PROD_REG:
  process
  begin
    if SYS_RST = '1' then
        PROD <= (others => '0');
    elsif RISING_EDGE(SYS_CLK) then
        if counter = "0000" then
            PROD <= (others => '0');
        else
            PROD <= '0' & PROD(31 downto 1);
            PROD <= ADD_OUT & PROD(14 downto 1);
        end if;
  end process;

  ADD_OUT <= PROD16 + AND_OUT;
  PROD16 <= PROD(31 downto 16);

COUNT:
  process
  begin
    if SYS_RST = '1' then
        counter <= (others => '0');
    elsif RISING_EDGE(SYS_CLK) then
        if counter = "1111" then
            counter <= (others => '0');
        else
            counter <= counter + 1;
        end if;
    end if;
  end process;
end architecture;
```