CME341 Nov. 17, 2021 midterm

`Time: 1 hour,`
`Text Books, Notes and Computer Files Only`
`NO CELL PHONES or LAPTOPS`

Each Question starts with the prototype `second_midterm_quartus` and assumes the prototypes instantiated therein are the prototypes used to get the correct answer for the preamble.

However, subquestions within a question start with the prototypes as modified in the previous subquestion. For example the prototypes used to start question 3b) are those used to get the answer for 3a).

The student prototypes that were modified in the preamble should be saved in a separate folder on the H drive so they can be retrieved at the beginning of each question.

The test bench used in this exam is the same as the test bench used in the preamble.

Before starting the exam download the following files from the class website:
`program_memory_preamble.hex`
`CME341_Nov_2021_midterm_Q3.hex`
`CME341_Nov_2021_midterm_Q4.hex`
`CME341_Nov_2021_midterm_Q5.hex`
`CME341_Nov_2021_midterm_Q6.hex`

NB: It is very important to clear, i.e. initialize, all flip/flops and registers constructed as part of the solution to a question. There are a couple of reasons for this. One of which is to ensure the answer for the exam dependent seed is correct. The testbench issues a reset after it changes the seed from 8'HAA to the student's exam dependent. For the second seed to produce the same result all registers and flip/flops must start in the same initial state after a `sync_reset`.

(1)

1. This question tests to see if you have done the preamble properly. It uses the program used in the preamble, i.e. the machine code for the program is in file `program_memory_preamble.hex`. The listing is given at the end of the question.

   Make sure the test bench is changed so that `exam_dependent_seed` is that listed on your answer sheet.

   Make sure `from_PS` is hardwired to `8'H00` and `from_ID` is hardwired to `8'H00`.

```
0000                org 8'H00
0000 00           load x0,#4'H0;
0001 10           load x1,#4'H0;
0002 20           load y0,#4'H0;
0003 30           load y1,#4'H0;
0004 40           load o_reg,#4'H0;
0005 50           load m,#4'H0;
0006 60           load i,#4'H0;
0007 70           load dm,#4'H0;
0008 80           mov x0, ipins;
0009 88           mov x1, x0;
000A 91           mov y0, x1;
000B 9A           mov y1, y0;
000C A3           mov o_reg, y1;
000D AC           mov m, r;
000E B5           mov  i, m;
000F BE           mov dm, i;

0010              org 8'H10
0010 C2           add x0,y0;
0011 E2           jmp 8'H20

0020              org 8'H20
0020 F2           jnz 8'H20;
0021 E3           jmp 8'H30;

0030              org 8'H30
0030 F3           jnz 8'H30
0031 E2           jmp 8'H20;
```

   If your circuit is working correctly the accumulator should read `16'HE5A5` at time 300 μs. Report `accumulator_output` for your exam dependent seed.

(1) 2. (a) This is a new question so remember to start with the set of prototypes used in the preamble.

Modify .hex file `program_memory_preamble.hex` so that the "`load oreg,#4'H0`" instruction becomes a "`load x1,#4'HF`". Then use the modified .hex file to initialize the program memory.

If your circuit is working correctly the accumulator should read `16'H8043` at time 300 μs, which is the answer for seed 8'HAA).

Report `accumulator_output` for **your** exam dependent seed.

(1) (b) This is a continuation question so you start with the same prototypes used to answer Question 2a. Also use the .hex file that was used to answer Question 2a.

Change the operation of the `load x1, #4'HX` instruction so that it not only loads `X1`, but also loads data memory with the least significant 4 bits in the instruction. The new instruction must auto increment the `i` register because it writes to data memory.

Modify `from_ID` so that `from_ID[7:1]=7'H0` and `from_ID[0]= register_enable[6] & i_mux_select`

If your circuit is working correctly the accumulator should read `16'H3A88` at time 300 μs, which is the answer for seed 8'HAA).

Report `accumulator_output` for **your** exam dependent seed.

(1)

3. Modify the instruction decoder as follows:

   - Build an 8-bit register (i.e. a bank of 8-flip/flops) called `counter` inside the instruction decoder. This register is to be clocked with `clk`.
   - Synchronously set `counter` to 8'H00 with `sync_reset`.
   - Change the no operation instruction NOPCF, which has machine code 8'HCF, so that it sets `counter` to 8'H80.
   - Change the no operation instruction NOPDF so that it decrements `counter` by 8'H01.
   - Connect `counter` to `from_ID`.

   The two no-operation instructions as redefined are still to be considered ALU instructions. This means `reg_enables[4]` must be high while either instruction is in the `ir`.

   NB: Use hex file `CME341_Nov_2021_midterm_Q3.hex` **to initialize your program memory.** The listing for which is given below

```
  Listing for Question 3
        org 8'H00
0000   C8 start:  NOPC8;    no operation
0001   CF                    NOPCF;   set counter to 8'H80
0002   DF                    NOPDF;   decrement counter
0003   D8                    NOPD8;   no operation
0004   DF                    NOPDF;   decrement counter
0005   DF                    NOPDF;   decrement counter
0006   DF                    NOPDF;   decrement counter
0007   CF                    NOPCF;   set counter to 8'H80
0008   E1                    jmp trap;
                             Align
0010   E1                    trap:    jmp trap;
```

   The answer for `exam_dependent_seed` = `8'HAA` is **16'HF461**.

(1)

4. Modify the instruction decoder so that whenever an ALU instruction is executed the `i` register is cleared.

   Make `from_ID = {7'H0, source_select[3]}` when an ALU instruction is in the `ir` register. Otherwise make `from_ID = 8'H00`. Just to be perfectly clear, `source_select[3]` is the most significant of the 4 select lines for the multiplexer that drives the data bus.

   NB: Use hex file `CME341_Nov_2021_midterm_Q4.hex` to initialize your program memory. The listing for which is given below:

```
  Listing for Question 4
                  org 8'H00;
0000  01   start: load  x0,#4'H1;
0001  23         load y0,#4'H3;
0002  C2         add x0,y0;
0003  E1         jmp done;
                  align
0010  E1   done: jmp done;
```

   The answer for `exam_dependent_seed` = `8'HAA` is **16'H3946**.

(1) 5. (a) Modify the instruction decoder to make `NOPD8` change the instruction that immediately follows it to be a jump instruction. The least significant 4 bits of that instruction is to be the jump address field. For example, if the instruction that follows `NOPD8` has machine code 8'HA4, it will act as a jump to 8'H40.

To simply the design of the circuit no program will be allowed to have a `NOPD8` instruction immediately follow a `NOPD8` instruction.

Also, the instruction decoder must make both `jmp` and `jmp_nz` equal to 1'b1 while the instruction that follows a NOPD8 is in the instruction register.

The instruction that immediately follows NOPD8 must act solely as a jump instruction. That is, its original function should be ignored and no registers should be enabled while it is in the instruction register.

Note: The `NOPD8` instruction must be treated as a ALU instruction, which means `reg_enables[4]` must be 1'b1 while it is in the instruction register.

Note: As explained at the beginning of the exam, be sure to clear all flip/flops and registers built as part of the solution with `sync_reset`.

Note: The source select (the 4 bit select for the multiplexer that selects what is on the data bus) does not affect the scrambler in this question so may be any value.

Note: Your solution to this question can **not** modify the operation of the `ir`. The instruction register is connected to the test bench and affects the scrambler. The scrambler is expecting the instruction register to behave as follows:

```
always @ (posedge clk)
ir = pm_data;
```

NB: Use hex file `CME341_Nov_2021_midterm_Q5.hex` **to initialize your program memory.** The listing for which is given below:

```
  Listing for Question 5
                 org 8'H00;
0000  D8   start: NOPD8
0001  22         load y0,#4'H2; for part a this is jump to A
                              ; for part b this is jumpt to B
0002  E1         jmp error;    should never execute this instruction
                 align
0010  E1   error: jmp error;
                 align
0020  E2   A:     jmp A;
0021  E0         jmp start;  should never execute this instruction
0022  D8   B:     NOPD8;      only part B gets to this instruction
0023  22         load y0,#4'H2; jump to B
```

The answer for `exam_dependent_seed` = `8'HAA` is **16'H2E9B** .

(1) (b) Modify the program sequencer so that the address for the jump instruction implemented in Part a is the entire 8 bits of the machine code that follows NOPD8. For example, if the instruction that follows `NOPD8` has machine code 8'HA4, the jump will be to 8'HA4.

Remember the jump implemented in Part a) is distinguished from a regular jump in that both `jmp` and `jmp_nz` are 1'b1.

The answer for `exam_dependent_seed` = `8'HAA` is **16'H80CA** .

(1)

6. Modify the program sequencer to start executing the instruction stored in program memory at address 8'HFF upon a `sync_reset`. Also change the order of execution to be descending addresses. In other words, the instructions in program memory are executed in the order 8'HFF, 8'HFE, 8'HFD, ..., until a jump or conditional jump instruction is executed.

   The functions of the jump and conditional jump instructions remain the same.

   NB: Use hex file `CME341_Nov_2021_midterm_Q6.hex` **to initialize your program memory.** The listing for which is given below:

```
  Listing for Question 6
                     org 8'H00
0000  E0   trap:    jmp trap;
                    org 8'HFD
00fd  E0            jmp trap;
00fe  7F            load dm, #4'HF;
00ff  B7   start:   mov i,dm;
```

   The answer for `exam_dependent_seed` = `8'HAA` is **16'HE3B0**.