COMP3211/COMP9211 Computer Architecture
Week 13 Tutorial Solution
Written By: Andhi Janapsatya

Q01. [P&H 6.3] How could we modify the following code to make use of a delayed branch slot?

```
LOOP:  lw    $2,   100($3)
       addi  $3,   $3,   4
       beq   $3,   $4,   Loop
```

Modified assembly code:

```
LOOP:  addi  $3,   $3,   4
       beq   $3,   $4,   Loop
       lw    $2,   96($3)
```

Q02. [P&H 6.5] For each pipeline register in Figure 6.25 on page 467, label each portion of the pipeline register with the name of the value that is loaded into the register. Determine the length of each field in bits. For example, the IF/ID pipeline register contains two fields, one of which is an instruction field that is 32 bits wide.

IF/ID Register:
- PC + 4 field [size of PC]
- Instruction field [32 bits]

ID/EX Register:
- PC + 4 field [size of PC]
- Read Data 1 [32 bits]
- Read Data 2 [32 bits]
- Sign extended Instruction(15-0) field [32 bits]
- Instruction(20-16) [5 bits]
- Instruction(15-11) [5 bits]

EX/MEM Register:
- jump/branch address [32 bits]
- zero control signal [1 bit]
- ALU result [32 bits]
- Read Data 2 [32 bits]
- Write back register address [5 bits]

MEM/WB Register:
- Data from memory [32 bits]
- ALU Result [32 bits]
- Write back register address [5 bits]

Q03. [P&H 6.11] Consider executing the following code on the pipelined datapath of Figure 6.46 on page 492 (P&H's textbook):

```
add   $1,   $2,   $3
add   $4,   $5,   $6
add   $7,   $8,   $9
add   $10,  $11,  $12
add   $13,  $14,  $15
```

At the end of the fifth cycle of execution, which registers are

being read and which register will be written?

Registers being read: $11 and $12
Register to be written: $1

Q04. [P&H 6.12] With regard to the program of Q03, explain what the
forwarding unit is doing during the fifth cycle of execution.
If any comparisons are being made, mention them.
The forwarding unit is comparing if result to be written back ($1) or result from ALU ($4) is equal to any source register being used for the current instruction ($8 or $9). No data forwarding is required in this case.

Q05. [P&H 6.13] With regard to the program of Q03, explain what the
hazard detection unit is doing during the fifth cycle of execution.
If any comparisons are being made, mention them.
The hazard detection unit is checking if previous instruction is a load instruction to register ($9) is equal to any source register being read for the next instruction ($11 or $12). No error happen in this example.

Q06. Elaborate the VHDL entity and architecture descriptions for
the sequencer-based control unit of a multicycle processor, as depicted
in block diagram form in Slides 38 & 50 of the lecture from Week 11

```
library IEEE;
use IEEE.std_logic_1164.all;

entity sequencer is

  port (
    state_reg        : in  std_logic_vector(3 downto 0);
    address_select   : out std_logic_vector(1 downto 0);
    ALUcontrol       : out std_logic_vector(1 downto 0);
    SRC1             : out std_logic;
    SRC2             : out std_logic_vector(1 downto 0);
    RegisterControl  : out std_logic_vector(2 downto 0);
    Memory           : out std_logic_vector(3 downto 0);
    PCWriteControl   : out std_logic_vector(3 downto 0));

end sequencer;

architecture behav of sequencer is
  signal output_signal : std_logic_vector(17 downto 0);

begin

  --Refer to appendix C.5 page C-29 from textbook

  with state_reg select
    output_signal <=
    "00"&"0"&"01"&"000"&"1001"&"0010"&"11" when "0000",     --Fetch
    "00"&"0"&"11"&"000"&"0000"&"0000"&"01" when "0001",
    "00"&"1"&"10"&"000"&"0000"&"0000"&"10" when "0010",     --Mem1
    "00"&"0"&"00"&"000"&"1010"&"0000"&"11" when "0011",     --LW2
    "00"&"0"&"00"&"101"&"0000"&"0000"&"00" when "0100",
    "00"&"0"&"00"&"000"&"0110"&"0000"&"00" when "0101",     --SW2
```

```
    "10"&"1"&"00"&"000"&"0000"&"0000"&"11" when "0110",     --Rformat1
    "00"&"0"&"00"&"110"&"0000"&"0000"&"00" when "0111",
    "01"&"1"&"00"&"000"&"0000"&"0101"&"00" when "1000",     --BEQ1
    "00"&"0"&"00"&"000"&"0000"&"1010"&"00" when "1001",     --JUMP1
    "0000000000000000" when others;

  ALUcontrol <= output_signal(17 downto 16);
  SRC1 <= output_signal(15);
  SRC2 <= output_signal(14 downto 13);
  RegisterControl <= output_signal(12 downto 10);
  Memory <= output_signal(9 downto 6);      -- bit 9 = MemRead
                                            -- bit 8 = MemWrite
                                            -- bit 7 = IorD
                                            -- bit 6 = IRWrite
  PCWriteControl <= output_signal(5 downto 2);      -- bit 5 = PCSource(1)
                                                    -- bit 4 = PCSource(0)
                                                    -- bit 3 = PCWrite
                                                    -- bit 2 = PCWriteCond
  address_select <= output_signal(1 downto 0);

end behav;
```