EE 4720 **Homework 2** Solution **Due: 6 March 2002**

Problem 1: Two VAX instructions appear below. VAX documentation can be found via `http://www.ece.lsu.edu/ee4720/doc/vax.pdf`. Don't print it, it's 544 pages. Take advantage of the extensive bookmarking of the manual to find things quickly. Chapter 5 describes the addressing modes and assembler syntax, Chapter 8 summarizes the VAX ISA, and Chapter 9 lists the instructions. For the instructions look up `ext` and `add` then find the mnemonics used below. Pay attention to operand order.

(*a*) Translate the VAX code below to MIPS (without changing what it does, of course). Ignore overflows and the setting of condition codes.

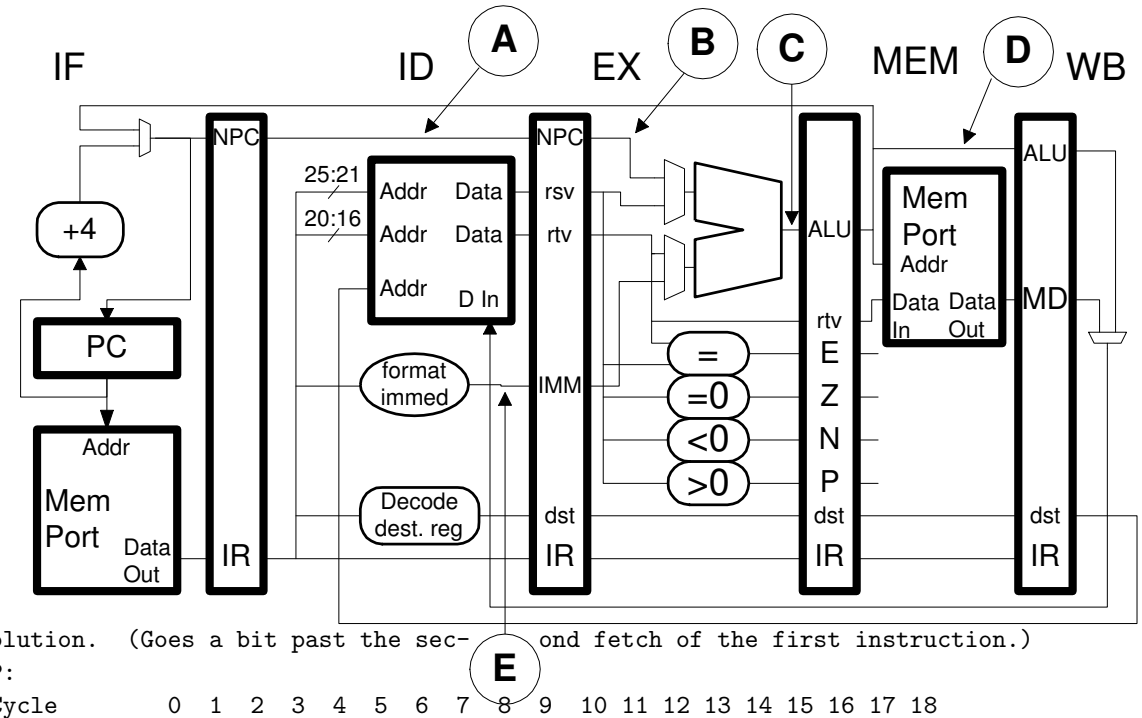
```
extzv    #10, #5, r1, r2

addl2    @0x12034060(r3), (r4)+  # Don't overlook the "@" and "+".
```

The solution appears below. Common mistakes are noted in the comments (the code shown is correct).

```
srl $2, $1, 10
andi $2, $2, 31

lui $10, 0x1203
add $10, $10, $3
lw  $10, 0x4060($10)
lw  $10, 0($10)       # The @ is for indirect, so load again!
lw  $11, 0($4)
add $10, $10, $11
sw $10, 0($4)
addi $4, $4, 4        # Increment r4 by the size of the data item.
```

(*b*) (Extra Credit) Show how the instructions above are coded.

Problem 2: A pipelined MIPS implementation and some MIPS code appear below. The results computed by the MIPS instructions are shown in the comments.




```
# Solution.  (Goes a bit past the sec-    ond fetch of the first instruction.)
LOOP:
 # Cycle        0  1  2  3  4  5  6  7  8  9  10 11 12 13 14 15 16 17 18
addi $1, $2, 4  IF ID EX ME WB                        IF ID EX ME WB
sub  $3, $0, $3    IF ID EX ME WB                        IF ID EX ME WB
and  $1, $1, $6       IF ID -> EX ME WB                     IF ID -> EX
or   $4, $1, $5          IF -> ID ----> EX ME WB               IF -> ID
bne  $4, $3, LO                IF ----> ID ----> EX ME WB            IF
sw   $4, 7($8)                           IF ----> ID EX ME WB
add  $10, $11,                                    IF IDx
 # Cycle        0  1  2  3  4  5  6  7  8  9  10 11 12 13 14 15 16 17 18
 #
 # A               0x1004   0x100c   0x1010    0x1014  0x1000
 # A                  0x1008   0x1010   0x1014   0x1018   0x1004
 # A                     0x100c   0x1010   0x1014   0x101c   0x1008

 # B                  0x1004   0x100c   0x1010    0x1014  0x1000
 # B                     0x1008   0x1010   0x1014   0x1018   0x1004
 # B                        0x100c   0x1010   0x1014   0x101c   0x1008

 # C                  24 30 ?? 20 ?? ?? 70 ?? ?? 1000
 # C                                          808

 # D                     24 30 ?? 20 ?? ?? 70 ?? ?? 1000
 # D                                             808

 # E               4  ?? ?? ?? ?? ?? ?? -5 -5 -5 7  ??       (Decimal)

 # Cycle        0  1  2  3  4  5  6  7  8  9  10 11 12 13 14 15 16 17 18
```

(*a*) Draw a pipeline execution diagram showing the execution of the code on the implementation. Base your pipeline execution diagram on the illustrated pipeline, do not depend solely on memorized execution timing rules, since they depend on details of the hardware which vary from problem to problem. Show execution until the second fetch of the first instruction.

Diagram shown above.

(*b*) Determine the CPI for a large number of iterations.

CPI $= \frac{13}{6} = 2.16667$.

(*c*) Certain wires in the implementation diagram are labeled with letters. (The circled letters with arrows.) Beneath the pipeline execution diagram show the value on those wires at near the end of each cycle. (Write sideways if necessary.) Do not show values if the corresponding stage holds a bubble or a squashed instruction. Only show immediate values if the corresponding instruction uses one. *Hint: Three instructions above use an immediate.*

Diagram shown above. The immediate holds the branch displacement, which is the number of instructions to skip. Many solutions incorrectly showed the branch target (`0x1000`) in the E row (the immediate value). Some solutions omitted the effective address computed by the `sw` instruction (808 in the C and D rows).

(*d*) *This is a special bonus question that did not appear in the original assignment!* For those students who have taken EE 3755 in Fall 2001, identify the Verilog code in `http://www.ece.lsu.edu/ee4720/v/mipspipe.html` corresponding to each labeled wire.

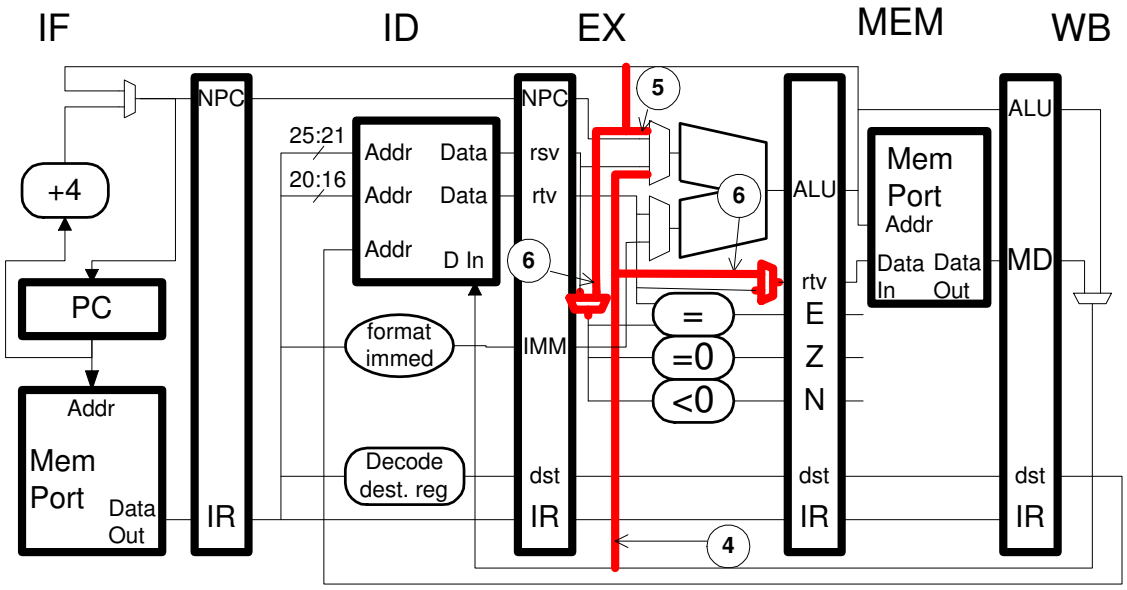
```
// Verilog lines shown below (without much context).

// A:
        id_ex_npc       <= if_id_npc;

// B:  The line with the B comment.

   always @( id_ex_alu_a_src or id_ex_rs_val or id_ex_sa or id_ex_npc )
     case( id_ex_alu_a_src )
       SRC_rs: alu_a = id_ex_rs_val;
       SRC_np: alu_a = id_ex_npc;                       // B
       SRC_sa: alu_a = {27'd0, id_ex_sa};
       default: `UNEXPECTED(alu_a,id_ex_alu_a_src);
     endcase

// C:  The line with the C comment.

   always @( posedge clk ) begin
      ex_me_npc    <= id_ex_npc;
      ex_me_pc     <= id_ex_pc;
      ex_me_alu    <= alu_out;                         // C
      ex_me_rt_val <= id_ex_rt_val;

// D:  The line with the D comment

   always @( posedge clk ) begin

      me_wb_npc      <= ex_me_npc;
      me_wb_pc       <= ex_me_pc;
      me_wb_dst      <= next_me_wb_exc øø reset ? 5'd0 : ex_me_dst;
      me_wb_from_mem <= ex_me_size != 0;
      me_wb_alu      <= ex_me_alu;         // D
```

```
      me_wb_md       <= data_in_2;
      me_wb_exc      <= ex_me_exc ? ex_me_exc : next_me_wb_exc;
      me_wb_occ      <= ~reset & ex_me_occ;
      tb_me_wb_din   <= tb_ex_me_din;
   end

// E:  The case statement and the assignment.

        // E
        case( immed_fmt )
          IMM_s: next_id_ex_imm = { immed[15] ? 16'hffff : 16'h0, immed };
          IMM_l: next_id_ex_imm = { immed, 16'h0 };
          IMM_u: next_id_ex_imm = { 16'h0, immed };
          IMM_j: next_id_ex_imm = { if_id_npc[31:28], ii, 2'b0 };
          IMM_b: next_id_ex_imm = { immed[15] ? 14'h3fff : 14'h0, immed, 2'b0 };
          default: `UNEXPECTED(next_id_ex_imm,immed_fmt);
        endcase

 // Further below

        id_ex_imm       <= next_id_ex_imm;    // E
```

Problem 3: Add **exactly** the bypass paths needed so that the code in the previous problem will run on the implementation below (the same as the one above) with the minimum number of stalls. Indicate the cycles in which the bypass paths will be used and the values bypassed on them.

Solution shown below, added bypass paths are in **red bold**. A pipeline execution diagram is also shown.

Almost all submitted solutions included the bypass path from the MEM stage to the upper ALU mux. Very few properly included the bypass path for the branch conditions. (Some incorrectly showed the bypass path into the ALU, which is used here to compute the branch target.) No submitted solution included a bypass path for the store value.




```
# Solution.  (Goes a bit past the sec-     ond fetch of the first instruction.)
LOOP:
 # Cycle        0  1  2  3  4  5  6  7  8  9  10 11 12 13 14 15 16 17 18
addi $1, $2, 4  IF ID EX ME WB          IF ID EX ME WB
sub  $3, $0, $3    IF ID EX ME WB          IF ID EX ME
and  $1, $1, $6       IF ID EX ME WB          IF ID EX
or   $4, $1, $5          IF ID EX ME WB          IF ID
bne  $4, $3, LO             IF ID EX ME WB          IF
sw   $4, 7($8)                 IF ID EX ME WB
add  $10, $11,                    IF IDx
 # Cycle        0  1  2  3  4  5  6  7  8  9  10 11 12 13 14 15 16 17 18
```