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.

\[
\text{extzv} \quad \#10, \#5, r1, r2
\]

\[
\text{addl2} \quad @0x12034060(r3), (r4)+ \quad \# \text{Don’t overlook the "@" and "+".}
\]

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

\[
srl \quad $2, \quad \$1, \quad 10
\]

\[
\text{andi} \quad \$2, \quad \$2, \quad 31
\]

\[
lui \quad \$10, \quad 0x1203
\]

\[
\text{add} \quad \$10, \quad \$10, \quad \$3
\]

\[
\text{lw} \quad \$10, \quad 0x4060($10)
\]

\[
\text{lw} \quad \$10, \quad 0($10) \quad \# \text{The @ is for indirect, so load again!}
\]

\[
\text{lw} \quad \$11, \quad 0($4)
\]

\[
\text{add} \quad \$10, \quad \$10, \quad \$11
\]

\[
\text{sw} \quad \$10, \quad 0($4)
\]

\[
\text{addi} \quad \$4, \quad \$4, \quad 4 \quad \# \text{Increment r4 by the size of the data item.}
\]

(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 second 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
add $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;
end

// 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 $\neq$ 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_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 second 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
  add $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
```