Virginia Tech
1 8 7 2

Instructions:

- Print your name in the space provided below.
- This examination is closed book and closed notes, aside from the permitted one-page formula sheet. No calculators or other computing devices may be used.
- Answer each question in the space provided. If you need to continue an answer onto the back of a page, clearly indicate that and label the continuation with the question number.
- If you want partial credit, justify your answers, even when justification is not explicitly required.
- There are 6 questions, priced as marked. The maximum score is 100.
- When you have completed the test, sign the pledge at the bottom of this page and turn in the test.
- Note that either failing to return this test, or discussing its content with a student who has not taken it is a violation of the Honor Code.

Do not start the test until instructed to do so!

Name _____
printed

Pledge: On my honor, I have neither given nor received unauthorized aid on this examination.

_____
signed

Questions 1 through 3 refer to the datapath diagram, Figure 4.24 from P&H, that was distributed with the test.

1. [15 points] For which of the supported instructions <u>must</u> the ALUSrc control signal, labeled **Q1** on the datapath diagram, be set to zero?

   ALUSrc must be set to 0 for any instruction that requires passing two values from the register file to the ALU; that would include all of the R-type instructions and beq.

---

2. [15 points] Carefully describe the purpose and operation of the AND gate located near the label **Q2** on the datapath diagram. Be complete.

   The purpose is to control the MUX that receives the output from the AND gate, in order to pass forward the computed address if a beq instruction is being executed and the two registers involved are equal.

   The AND gate receives two input values:

   > **Branch**   **signal set by the main Control unit to 1 if the current instruction's opcode indicates it's beq, and to 0 otherwise**

   > **Zero**   **signal set by the ALU, to 1 if the result of the current computation is zero, and to 0 otherwise**

   So, if the instruction is beq and the result of the computation is zero, then the AND gate receives two 1s and sets its output to 1, instructing the MUX to pass through the value from the branch address calculation rather than the value PC + 4.

   And, otherwise the AND gate will receive at least one 0 as input and will therefore output a 0, instructing the MUX to pass through the address PC + 4.

3. [15 points] Suppose the MemtoReg control signal, labeled Q3 on the datapath diagram, was stuck at 0. Assume that all the other control signals operate correctly. Which of the supported instruction(s), if any, would not execute correctly? Justify your answer carefully; in particular, describe carefully just what would go wrong for each such instruction.

If MemtoReg is stuck at 0, the MUX it controls will invariably pass the ALU output value to the Write Data port on the Register File.

The most obvious effect is that the Read Data value from the Data memory unit can never be passed to the Register File, which means that the `lw` instruction must always fail.

R-type instructions are unaffected, since MemtoReg must be set to 0 for them in any case.

`j` and `sw` will also not be affected, so long as the RegWrite signal is properly set to 0 for both.

---

4. Recall the various MIPS machine instruction formats.

a) [10 points] What does the number of bits that are used to specify a register number imply about the underlying hardware? Why?

Five bits are used to specify a register number, so there are 2^5 or 32 different patterns that can be formed. This implies that the underlying hardware will not have more than 32 general-purpose registers (or it will have registers that cannot be selected in machine instructions).

b) [10 points] What role does the funct field play, and for which instructions? Be precise.

The funct field is used by R-type instructions to specify exactly which arithmetic-logical operation is to be carried out.

5. [20 points] The native MIPS assembly language does not include a memory-to-memory data transfer instruction:

```
mcopy   ($rd), ($rs), $rt       # copy the contents of the memory word at
                                #   the address in $rs to $rt consecutive
                                #   memory words, starting at the address
                                #   in $rd
                                #
                                # That is:
                                #       Mem[$rd:$rd+$rt-1] <-- Mem[$rs]
```

Show how an assembler might replace the pseudo-instruction above with a sequence of <u>basic</u> (native) instructions to achieve the same effect. Include comments to explain the logic of your design. You should not <u>leave</u> the value in any register modified, except possibly for $at. It is OK to modify other registers as long as they have their original values restored at the end of the operation.

```
        addi    $sp, $sp, -12       # back up some temp registers
        sw      $t0, 8($sp)
        sw      $t1, 4($sp)
        sw      $t2, 0($sp)

        or      $t0, $zero, $zero   # count number of copies made
        lw      $t1, 0($rs)         # get value to be copied
        or      $t2, $rd, $zero     # get pointer to target memory


mloop:  beq     $t0, $rt, mdone     # done when counter reaches $rt

        sw      $t1, 0($t2)         # write copy of value to target

        addi    $t0, $t0, 1         # count this copy

        addi    $t2, $t2, 4         # step to next word in target range

        j       mloop


mdone:  lw      $t0, 8($sp)         # restore values to temp pointers
        lw      $t1, 4($sp)
        lw      $t2, 0($sp)
        addi    $sp, $sp, 12        # restore stack pointer
```
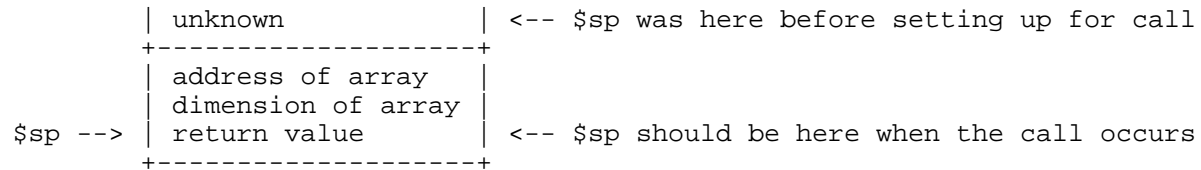
6. [15 points] Assume that a MIPS procedure F takes two parameters via the runtime stack, and places its return value onto the stack. The procedure is designed so that it expects the stack to be organized as shown below when it begins to execute:

```
        | unknown              | <-- $sp was here before setting up for call
        +---------------------+
        | address of array    |
        | dimension of array  |
$sp --> | return value        | <-- $sp should be here when the call occurs
        +---------------------+
```

Given the data segment below for the calling procedure, write the code the calling procedure would need in order to set up the stack before the call; you may use any valid MIPS instructions you like.

```
        .data
Size:   .word  100        # dimension of array to be passed
Array:  .word  400        # array to pass to procedure
RetVal: .word   42        # location for return value from procedure

# code to set up stack goes here:

        addi $sp, $sp, -12     # make room on the stack

        la   $at, Array        # get address of the array
        sw   $at, 8($sp)       # write address of array to the stack

        lw   $at, Size         # get dimension of the array
        sw   $at, 4($sp)       # write array dimension to the stack

      # Since the return value is written to the stack by the called
      # procedure, there is no reason to write a value to that location;
      # all you need to do is reserve the space on the stack.
```

```
# procedure call would go here:
```