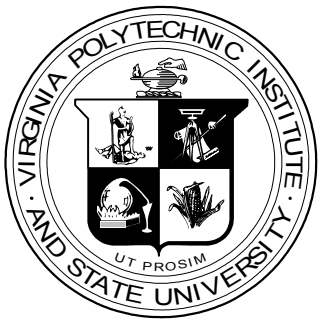

Instructions:

- Print your name in the space provided below.
- This examination is closed book and closed notes, aside from the permitted one-page formula sheet and the MIPS reference card. 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 8 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 SOLUTION

printed

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

signed

1. [10 points] Explain the differences and relationship between the instruction register and the program counter.

Depending on when you look at it during the fetch-execute cycle, the program counter stores the address of the current instruction being executed, or of the next instruction to be fetched.

The instruction register stores the instruction that is currently being executed.

---

2. [24 points] Each part below provides a pseudo-code description of an arithmetic or logical operation that could be performed on the MIPS hardware. For each, write a short snippet of MIPS assembly code that would perform the given operation.

   a) `$t7 = $t3 + $t4;`

```
add    $t7, $t3, $t4
```

   b) `$t3 = $t4 + $t5 – $t6;`

```
add    $t3, $t4, $t5
sub    $t3, $t3, $t6
```

   c) `$s3 = $t2 / ($s1 - 5432);`

```
addi   $s3, $s1, -5432
div    $s3, $t2, $s3
```

---

3. [8 points] Format R for MIPS machine language instructions is shown below. Why are the fields for the register operands 5 bits wide?

op	rs	rt	rd	shamt	funct

The MIPS architecture incorporates 32 general-purpose registers (and 32 floating-point registers). Therefore, we need to be able to distinguish among 32 different registers, and that requires 5 bits.

4. [10 points] The MIPS assembly code below shows the declaration of an array and a subsequent instruction that is intended to store a value at some location within the array. However, just as in C, there is the peril that the register `$s0` may contain a value that does not specify a suitable address within the array. Show how to add a check to the given code that will prevent the store instruction from being executed if the address in `$s0` is logically invalid.

Validating the address in $s0 requires two checks. Is the address below the

```
        .data
array:  .space 200
size:   .word   50

        .text
main:
        la      $t0, array          # get lower bound of array addresses
        blt     $s0, $t0, bad_addr  # check for violation of lower bound
        lw      $t1, size           # calculate address of highest word
        addi    $t1, $t1, -1        # note: this is 4 bytes BELOW end of
        muli    $t1, $t1, 4         #       array
        add     $t1, $t0, $t1
        bgt     $s0, $t1, bad_addr  # check for violation of upper bound

        sw      $s7, ($s0)
bad_addr:                           # corrective action wasn't specified
```

---

5. [12 points] The author of the code shown in the previous question wants to add a procedure call, within `main`. In order to communicate with the procedure, the stack must be modified as shown below:

```
|-----------------|
|                 | <-- sp    addi    $sp, $sp, -12  # alloc room on stack
|-----------------|           la      $t2, array     # get address of array
|  addr of array  |           sw      $t2, 8($sp)    #    and put on stack
|-----------------|           lw      $t2, size      # get size of array
|  value of size  |           sw      $t2, 4($sp)    #    and put on stack
|-----------------|           sw      $t0,  ($sp)    # put $t0 on stack
| value in reg t0 |
|-----------------|
```

Initially, the stack pointer is at the location shown above. Write the necessary instructions to transform the stack to hold the specified data, and leave the stack pointer at the proper location.

---

6. [6 points] Recalling what we know about PC-relative addressing in MIPS assembly, why might the assembler need to take special action in order to translate the first instruction below into machine language? Be specific.

```
here:   beq  $s0, $s2, there
. . .
there:  add  $s0, $s0, $s0
```

The assembler must translate the code into machine language instructions, which are 32 bits wide. The label must be resolved to a relative address; if the distance from "here" to "there" is very large, the relative address will not fit into the 16 bits available in the obvious format. In that case, the assembler must translate the instruction in some other way.

7. [24 points] Consider the MIPS assembly procedure below, which is somewhat similar to the C Library function `strcmp()`. The comments explain WHAT each statement does, or should do (in the case of the missing statements). The code that is shown is logically and syntactically correct. Your job is to complete the implementation by filling in the empty blanks with statements that conform to the comments.

```
################################################################ strcmp
# Pre:
#       a0:  address of zero-terminated string 0
#       a1:  address of zero-terminated string 1
# Post:
#      v0:  0 if the strings are not equal, 1 if they are equal
strcmp:

        ori     $v0, $zero, 1        # default to equality

        lb      $t0, ($a0)           # get first character of string 0

        lb      $t1, ($a1)           # get first character of string 1


loop:   beqz    $t0, ck_second       # check for terminator in string 0

        beqz    $t1, not_equal       # check for terminator in string 1

        bne     $t0, $t1, not equal  # check for mismatch


        addi    $a0, $a0, 1          # step to next characters

        addi    $a1, $a1, 1

        SAME AS FIRST BLANK ABOVE    # get next character of string 0

        SAME AS FIRST BLANK ABOVE    # get next character of string 1

        j       loop                 # restart loop


ck_second:

        beqz    $t1, stop            # check for length mismatch

not_equal:

        or      $v0, $zero, $zero    # strings are NOT equal

stop:

        jr      $ra
```

8. [6 points] Consider the short MIPS assembly program below:

```
            .data
value1:     .word     42                  #  1
value2:     .word     38                  #  2

            .text
main:
            lw        $t0, value1         #  3
            lw        $t1, value2         #  4
            add       $t2, $t1, $t0       #  5
            // exit syscall not shown
            .data
msg:        .asciiz "The sum is "         #  6

            .text
            la        $a0, msg            #  7
            li        $v0, 4              #  8
            syscall                       #  9
            move      $a0, $t2            # 10
            li        $v0, 1              # 11

            syscall                       # 12
            li        $v0, 10             # 13
            syscall                       # 14
```

a) Which of the numbered lines correspond to things that would be stored in the text segment in the MIPS memory architecture? List individual line numbers or ranges (e.g., 87-105).

Lines 3 – 5 and 7 - 14

b) Which of the numbered lines correspond to things that would be stored in the data segment in the MIPS memory architecture?

Lines 1, 2 and 6