1. [4] In the MIPS architecture that we are studying, the expression *one word of memory* has a specific meaning.

1 a. How many bytes are there in one word of memory in this design? 4

1 b. How many bytes are there in a memory address? 4

2 c. Assume that the address of a particular word in memory is word-aligned. Is bit 0 (ie, the low order bit) of that address 0 or is it 1? Why?

0. All addresses are multiples of 4, so both bit 0 and bit 1 are 0.

2 2. [2] Give one example of the simplifications that make a RISC design less complicated than CISC and briefly describe why it makes it easier to design a faster or smaller implementation of the architecture.

all instructions 32 bit - simplifies instruction fetch, decode
limited memory access instructions - simplifies data memory access
regular instruction format - simplifies instruction decode
limited number of instructions - reduces hardware complexity, saving space not wasted on unused instructions

3. [3] The expression *calling conventions* describes the way that procedures interface with each other during a procedure call, including managing the stack, expected register usage, and maintaining the return address. We have been using one consistent set of calling conventions in all the code that we have read or looked at in this class.

1 a. When returning from a call, a called procedure must guarantee that the values of some registers are unchanged from the values they had when the procedure was called. Is $ra one of these registers? (Yes) No

2 b. The stack pointer is set and restored when a procedure needs a stack frame for use during execution. What benefit is gained by maintaining a stack pointer that is a multiple of 8?

The stack pointer is always double word aligned and so any called procedures can do aligned stores and loads on double word quantities.

4. [4] Some procedures create a stack frame when they start executing and discard it when they finish, but other procedures do not. Describe a particular situation in which a procedure author would decide that a stack frame was necessary. What specifically would the stack be used for in your example?

2 idea
2 explain

save register - if you are going to change a register that must be preserved, you can store it on the stack

argument build area - if you are going to call another procedure with arguments

local variables - if you need more vars than can fit in registers

5. [4] Translate the C language statement below into MIPS assembly language instructions. You can assume that `counter` is the label of an integer (one 32-bit word) stored in main memory. When your code snippet completes, the value in memory should be appropriately updated. Note: You are not writing a complete procedure, just the few instructions needed to implement this line of code.

```
counter++;
```

2 memory
2 arithmetic

```
lw   $t0, counter
addi $t0, 1
sw   $t0, counter
```

6. [4] Describe two differences between the representation used for `.asciiz` strings that we are using (also known as C strings) and the representation used for Java strings.

2 each difference

asciiz null terminated, ascii

Java counted, Unicode

7. [2] There are several different instructions that can be used to change the control flow in a MIPS program. Selecting from `j`, `jal`, `jr`, `beq` and `bne`, which instruction has the greatest range? (In other words, which instruction can be used to go the furthest away relative to where the instruction is located?) Why is this true?

2

jr it is the only one with a 32-bit destination address specification

8. [2] Draw separation lines and label the sign bit, biased exponent, and mantissa fields in the following 32-bit floating point representation of 3.14159274 (the closest we can get to π with single-precision floating point).

2

sign exponent mantissa

0|100 0000 0|100 1001 0000 1111 1101 1011

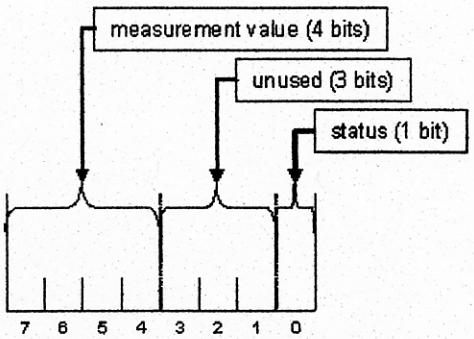
9. [2] What is an important benefit of constructing single precision floating point numbers with the sign bit, biased exponent, and mantissa arranged the way they are in the word?

2

similarity to integers for some operations
- pos/neg recognized the same way
- large fp numbers look like large integers

10. [3] Imagine that you have an electronic sensor that reports its measurements in an 8-bit byte, formatted as shown here.

The low order bit (bit 0) is the status bit. Bits 1, 2, and 3 are not used. Bits 4 through 7 contain the measurement that the device is reporting.



a. How many different measurement values can be reported by this device using this format?

16 (or 32 if you include the status bit.)

b. If the measurement values are considered to be signed integer numbers represented in two's complement format, what is the largest possible positive measurement value that can be reported?

$0111_2 = 7_{10}$

c. Again assuming two's complement notation, what 8-bit value would be returned when the device reported status=1, unused=0, and measurement = -1?

$1111\ 0001_2 = F1_{16}$

11. [5] Consider the hex value 0x20030007 as representing one MIPS machine language instruction.

a. Convert the hex value to a 32-bit binary value.

0010 0000 0000 0011 0000 0000 0000 0111

b. What is the opcode value for this instruction?

8_{10}

c. What is the name of the instruction with that opcode?

addi

d. What is the format (R, I, or J) of the instruction?

I

e. What is the number of the destination register in this instruction?

3

12. [8] The figure shown here is copied from SPIM and shows part of the HW2 solution after it was assembled and loaded by SPIM. Refer to this figure when answering the following questions.

```
[0x00400090]  0x0008b021  addu $22, $0, $8       ; 81: move    $s6,$t0          # assume x is okay
[0x00400094]  0x0128082a  slt $1, $9, $8        ; 82: ble     $t0,$t1,skip6    # skip if okay
[0x00400098]  0x10200002  beq $1, $0, 8 [skip6-0x00400098]
[0x0040009c]  0x0009b021  addu $22, $0, $9      ; 83: move    $s6,$t1          # clamp to limit
[0x004000a0]  0x0000b821  addu $23, $0, $0      ; 88: move    $s7,$zero        # s7 is the sum
[0x004000a4]  0x3c011001  lui $1, 4097          ; 89: lw      $t0,limit        # t0 == loop index
[0x004000a8]  0x8c280040  lw $8, 64($1)
[0x004000ac]  0x02e8b820  add $23, $23, $8      ; 91: add     $s7,$s7,$t0      # $s7 = $s7 + index
[0x004000b0]  0x2108ffff  addi $8, $8, -1       ; 92: addi    $t0,$t0,-1       # index = index -1
```

2 a. Which one or more of the seven assembly language instructions written by the code's author are actually part of the core instruction set provided by the MIPS architecture?

lw (although the address is pseudo), add, addi

2 b. When the CPU fetches the first instruction in this code, `move $s6, $t0`, what is the value of the PC?

0x00400090

2 c. The second line in the figure includes the instruction `slt $1, $9, $8`. What are the names (not numbers) of the three registers used in this instruction?

$at, $t1, $t0

2 d. The last source instruction in this sequence is `addi $t0, $t0, -1`. The resulting machine language instruction is `0x2108ffff`. Which part of this hex value represents the -1?

13. [4] For the following sequence of MIPS instructions, identify all registers used and their values after the code has executed. The first column of the table is filled in as an example.

1 pt each

```
        li    $t0, 4
        li    $t1, 7
        li    $t2, 3
        sub   $t3, $t1, $t2
        beq   $t0, $t3, next
        add   $s0, $zero, $t3
        j     end
next:
        add   $s0, $t1, $t2
end:
```

Register name:	t0	t1	t2	t3	s0
Register value:	4	7	3	4	10

14. [4] In MIPS assembly language, there exists a pseudo-op called seq (set equal). It compares two source registers, and sets the destination register to 1 if equal, otherwise 0.

Example format: `seq $v0, $s2, $s3`

Write a short sequence of any valid MIPS assembly instructions (except seq itself) to compare the contents of the source registers `$s2` and `$s3`, and set the destination register `$v0` to 1 if equal, otherwise 0.

```
li   $v0, 1
beq  $s2, $s3, skip
li   $v0, 0
skip:
```

4 perfect

3 . . .

15. [4] Suppose $t0 contains the address of the 0th element of an array of 8-bit bytes and $t1 holds the value of index n. Write a short sequence of MIPS instructions to store the value 1 into array[n].

```
la      $t0,byteValues  # address of byte values array
lw      $t1,n           # the index value
```

```
li   $t2, 1
add  $t3, $t0, $t1
sb   $t2, 0($t3)
```