Instructions:

- Print your name in the space provided below.
- This examination is closed book and closed notes, aside from the test supplement and 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 7 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] Why is 2's complement representation used for signed integer values instead of sign-magnitude representation? Be precise and complete.

Sign-magnitude representation represents the magnitude (absolute value) of the integer in 31 bits and uses the high bit to represent the sign. 2's complement represents non-negative integers in the same way as sign-magnitude, and represents negative integers by flipping the bits (of the representation of the non-negative analog) and adding 1.

The primary disadvantage of sign-magnitude representation is that a different algorithm must be used when adding two integers with opposite signs than when adding two integers with the same sign. See slide #4 of the notes from Aug 27 for an example.

While there is an algorithm that will work with opposite-signed numbers, that would require two different hardware devices to handle addition of sign-magnitude numbers. That would waste space on chip, and have at least a small time penalty.

If 2's complement representation is used, a single algorithm will suffice for all cases, and therefore the hardware implementation would be more efficient.

It's true that sign-magnitude allows two different representations for 0, and represents one less negative integer, but those issues are insignificant when compared to the issue of implementing addition.

---

2. [10 points] The IEEE Floating Point Standard specifies that the single-precision floating point type will allocate the 32 bits it uses as follows:

1 bit	8 bits	23 bits (plus phantom bit)
Sign	**Exponent**	**Significand**

As a result, the single-precision type can represent about 7 significant digits of (some) values in the range between a minimum of -3.4×10^{38} and a maximum of -3.4×10^{38}.

If the format was changed as shown below, how would the previous sentence have to be revised (roughly speaking)?

1 bit	10 bits	21 bits (plus phantom bit)
Sign	**Exponent**	**Significand**

The number of bits used for the exponent determines the range (the exponent on 10) and the number of bits used for the significand determines the precision (number of significant digits that can be represented).

If you use two more bits for the exponent, you can represent larger exponents and hence have a larger range; if you use fewer bits for the significand, you cannot represent as many significant digits.

So, the effect (roughly speaking) is that the range increases and precision decreases.

Adopting the (very rough) rule of thumb that it takes a little more than 3 bits to represent one significant digit, you'd get about 6 significant digits. The exponent range would be 4 times as large, so the range would be more or less from -10^{154} to 10^{154}.

3. When the MIPS code shown below is assembled in MARS, we get the memory layout shown below.

```
        .data
Sz:     .word  5
List:   .word  10, 20, 30, 40, 50
        .text
main:
        la   $s0, List      # 1
        lw   $s1,  ($s0)    # 2
        lw   $s2, 4($s0)    # 3

        li   $v0, 10
        syscall
```

	address	word value
Sz	1001 0000	0000 0005
List	1001 0004	0000 000A
	1001 0008	0000 0014
	1001 000C	0000 001E
	1000 0010	0000 0028
	1000 0014	0000 0032

a) [12 points] What value is placed into each of the following registers when the program is executed? State your answers in hexadecimal (as shown above).

```
$s0:  1001 0004  # la loads the address of the label; see the memory layout

$s1:  0000 000A  # lw loads the memory word at the specified address; see
                 #     the memory layout for the value

$s2:  0000 0014  # this loads from address $s0 + 4 == 1001 0008; remember
                 #     that addresses are specified as BYTE addresses
```

b) [4 points] What value would be placed into the register `$s3` if the following instruction was executed?

```
la    $s3, Sz
```

This would load the address of the label `Sz`, which is 1001 0000.

c) [8 points] Assume that statements 1 through 3 given above have been executed. Write MIPS assembly code that will compute the address of the first byte of memory after the end of the array `List` and place that address into the register `$s4`. Your solution should work regardless of the dimension of the array, so just hardwiring the value `0x10000018` into `$s4` is <u>not</u> an acceptable answer. You may use any MIPS assembly instructions supported by MARS.

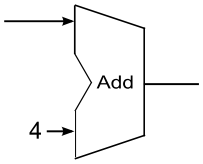
```
#  You need to compute the number of bytes in the array and then add that
#  to the address at which the array begins.

lw    $s4, Sz         # get number of elements in array

sll   $s4, $s4, 2     # multiply by 4 since each array element is 4 bytes

add   $s4, $s0, $s4   # add the array length to the base address, which is
                      #   stored in $s0 from instruction #1 above
```

4. [18 points] Complete the following MIPS assembly program. You may use any MIPS assembly instructions supported by MARS.

```
            .data
X:          .word  1024
Y:          .word  4096

            .text
main:
            # swap the values of the variables X and Y:

            lw    $t0, X      # load value of X and Y into registers
            lw    $t1, Y

            sw    $t0, Y      # store them back at the opposite locations
            sw    $t1, X



            li   $v0, 10
            syscall
```

---

5. [12 points] Refer to the datapath diagram on the test supplement. Explain the purpose of the component shown below. Be clear and concise, and be sure to address why the value 4 is used (although that's not all you should address).

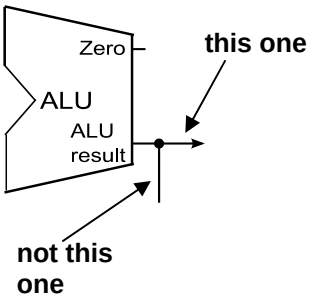


Examining the datapath diagram, we see that the top input is the value from the PC register, which stores the address of the instruction that is being executed, and that the output eventually goes to a multiplexor that may (or may not) pass that value back to the PC register.

So, this unit is computing what may (or may not) be the address of the next instruction to be fetched and executed.

The value 4 is used because all MIPS machine instructions are 4 bytes wide, and so the address of the next instruction in memory would be 4 larger than the address of the current instruction.

6. [12 points]Refer again to the datapath diagram on the test supplement. Consider the connection indicated below. Which of the supported instructions depend on this connection and why? Be precise and complete.



Examining the datapath diagram, we see that this is the ALU and the data line in question goes to the Address port on the data memory unit.

So, this line is used to transfer an address computed in the ALU to the data memory.

The only instructions that require this are the two that access data memory: `lw` and `sw`.

For `lw`, the ALU computes the address that is to be read from; for `sw` it computes the address that is to be written to

---

7. a) [7 points] Why are register numbers stored using 5 bits in MIPS machine language instructions? Be precise.

Using 5 bits, we can represent 32 different register numbers, from 0 to 31. That is appropriate since the MIPS architecture provides exactly 32 user-accessible registers (although some are restricted by convention).

It has absolutely nothing to do with the fact that the word size is 32 bits.

b) [7 points] What is the general purpose of a multiplexor?

A multiplexor takes two or more input signals and passes one of them through, depending on how the multiplexor's select bits are chosen.

The purpose is to be able to select one of two or more candidate values to pass to a datapath element.