CISS360: Computer Systems and Assembly Language
Test t01

Name: aoro1@cougars.ccis.edu Score: $\boxed{}$

Open `main.tex` and enter answers (look for `answercode`, `answerbox`, `answerlong`). Turn the page for detailed instructions. To rebuild and view pdf, in bash shell execute `make`. To build a gzip-tar file, in bash shell execute `make s` and you'll get `submit.tar.gz`.

Instructions

- This is a closed-book, no-discussion, no-calculator, no-browsing-on-the-web no-compiler/no-MIPS-simulator test.
- Cheating is a serious academic offense. If caught you will receive an immediate score of -100%.
- If a question asks for a program output and the program or code fragment contains an error, write `ERROR` as output. When writing output, whitespace is significant.
- If a question asks for the computation of a value and the program or code fragment contains an error, write `ERROR` as value.

Honor Statement

I, $\boxed{\text{[REPLACE WITH YOUR FULLNAME]}}$, attest to the fact that the submitted work is my own and is not the result of plagiarism. Furthermore, I have not aided another student in the act of plagiarism.

Question	Points
1	
2	
3	
4	
5	
6	
7	
8	
9	
10	
11	
12	
13	
14	
15	
16	
17	
18	
19	
20	

Question	Points
21	
22	
23	
24	
25	
26	
27	
28	
29	
30	
31	
32	
33	
34	
35	
36	
37	
38	
39	
40	

TOTAL	

Welcome to your first day of work at Fullabugz Company. Your job is to write MIPS code, debug MIPS code (written by someone else ... ARGHHH), etc.

Do not give me two answers for the same question. I reserve the right to choose one for you ... and I always choose the wrong one (if there is one)!

Here's some useful info:

1. Print int: syscall 1
2. Read int: syscall 5
3. Print string: syscall 4
4. Read string: syscall 8
5. Exit: syscall 10

You should know what to do with `$v0`, `$a0`, `$a1`, right?

Q1.

(a) ISA stands for

Answer:

```
Instruction Set Architecture
```

(b) CISC stands for
Answer:

```
Complex Instruction Set Computer
```

(c) RISC stands for
Answer:

```
Reduced Instruction Set Computer
```

(c) T or F or M. An assembly instruction translates to one machine instruction.
Answer:

```
F
```

(d) The five parts of a computer are
Answer:

```
Input, output, memory, datapath, control
```

(e) The CPU or processor is made up of the following two parts:
Answer:

```
datapath and control
```

(f) Which of the following provides the fastest storage mechanism. Choose one.

a. CPU Cache

b. Register file

c. Memory

d. External storage (such as hard drive)

Answer:

```
b. register file
```

Q2. Joe Cantcode wrote the following code to display the first integer in the data segment to the console. But it doesn't work!!! Correct it. In the code below, the first integer in the data segment is 2010. But you cannot assume that it won't change in the future. You must not change the data segment.

(From the management: This is a warm-up. Just insert one line.)

Answer:

```
          .text
          .globl    main

main:     la        $a0, odyssey
          lw        $a0, 0($a0)
          li        $v0, 1
          syscall
          li        $v0, 10
          syscall

          .data
odyssey:  .word 2010
```

Q3. Tom Toofaz always jumps into his project assignments and hack out code ... and ends up spending his night in the office. He's having problem implementing the following requirement in MIPS with less than four `t`–registers. But his manager, Sloe Arnn Steady, told him that four is *way* too many. Help Tom and he'll buy you lunch.

Write a code segment in MIPS to compute `9 * a - (b + c) + d` where `$s0`, `$s1`, `$s2`, and `$s3` have values of the C/C++ integer variables `a`, `b`, `c`, `d` respective. The result must be placed in `$s4`. Your code must be as efficient as possible and you must use your registers efficiently too. You may assume that the `t`-registers are not used elsewhere, and so you can use the `t`–registers in your code. No such guarantee for the `s`-registers – therefore you cannot change the values in `$s0`, `$s1`, `$s2`, and `$s3`. You also cannot use the `a`– and `v`–registers. Do not use the `mul` or `mult` command. Write down the number of `t`-registers used.

Answer:

```
addu $t0, $s1, $s2    # t0 = b + c
addu $t0, $t0, $s3    # t0 = (b + c) + d
move $t1, $s0         # temp store s0
addu $t2, $s0, $s0    # t2 = 2 * a
addu $t2, $t2, $t2    # t2 = 4 * a
addu $t2, $t2, $t2    # t2 = 8 * a
addu $t2, $t2, $t1    # t2 = 9 * a
subu $s4, $t2, $t0    # s4 = 9 * a - (b + c) + d
```

(Code fragment, not a complete program.)

Number of `t`–registers used:

Answer:

```
3
```

Q4. The following is an attempt by Sue Longlunch to scan an array of words and to add 10 to each word in that array. The array has size 20 and base address is in register `$s0`. Your manager is fuming mad because there are bugs in the code (and Sue is still at lunch) and have asked you to do a code review and writing in some comments. You should write "`should be: xxxxx`" to replace the code or "`insert here: xxxxx`" to insert some code between two lines.

```
                                   COMMENTS TO SUEa
        addi    $t0, $s0, 20       # should be:   li      $t1, 20
                                   #              addi    $t1, $t1, $t1
                                   #              addi    $t1, $t1, $t1
                                   #              addi    $t1, $s0, $t1
                                   #
                                   # now we have 2 pointers:
                                   # s0 = &x[0] & t1 = &x[size]
                                   # we had to multiply size * 4 bytes
                                   # (1 word = 4 bytes) to get the
                                   # address of the  &x[size]

loop:   blt     $s0, $t1, exit:    # now we are looping if &x[0] < &x[size]

        move    $t1, 0($s0)        # lw     $t0, 0($s0)
                                   # addiu  $t0, $t0, 10

        addi    $s0, $s0, 1        # this should be:
                                   # addiu s0 = s0 + 4 (move to the next
                                   # element in the array)

        addi    $t1, $t1, 10       # delete this

        jr      loop
exit:
```

Q5. While working on an artificial neural network, your senior teammate Arnie Ned needs you to write a MIPS code fragment to "clamp" the integer value in `$s0` at `-5` and `5`. In other words if `$s0` is less than `-5`, `$s0` is set to `-5`. If the value of `$s0` is greater than `5`, `$s0` is set to `5`. Otherwise, the value in `$s0` is unchanged.

Answer:

```
        li      $t0, 5
        li      $t1, -5
if:     ble     $s0, $t0, exit          # case when s0 <= 5 return s0
        bge     $s0, $t1, exit          # case when s0 >= -5 return s0

        bgt     $s0, $t0, else          # case when s0 > 5 return 5
        li      $s0, 5
        j       exit

else:   li      $s0, -5                 # case when x0 < -5 return -5
        j       exit

exit:   li      $v0, 1                  # print s0
        move    $a0, $s0
        syscall
```

(Code fragment, not a complete program.)

Q6. What is the MIPS code fragment for the C code

```
f[i + 2] = f[i + 1] + f[i];
```

where `f` is an array of integers. The base address of `f` is associated with `$t0`, `i` is associated with `$t1`.

Answer:

```
        # t0 - &x[0]
        # $t1 - i
        addu  $t1, $t1, $t1
        addu  $t1, $t1, $t1   # t1 = 4 * i
        addu  $t1, $t0, $t1   # t1 = &f[i]

        addu  $t2, $t1, $t1   # t2 = &f[i + 1]
        addu  $t3, $t2, $t1   # t3 = &f[i + 2]
```

Q7. Your project leader Lee Pointa wants you to write some MIPS code to perform the first pass of bubblesort (in ascending order) on an integer array, which will be used in a bubblesort that Bubba Short will use. Lee says that the base address of the array is stored in `$a0` and the number of values in the array is stored in `$a1`. Another thing: Lee tells you he prefers you do not use an index variable, but rather, use a pointer to iterate over the array. (Look at his lastname.)

Answer:

```
        # a0 - &x[0]
        # a1 - size of the array
        move   $t0, $a0         # we will use it as our loop counter
        addu   $a1, $a1, $a1
        addu   $a1, $a1, $a1
        addu   $a1, $t0, $a1    # pointer a1 - &x[size]

first pass:
        bgt    $t0, $a1, end_pass # if x[i] > x[size] goto end_pass
        lw     $t1, 0($t0)        # load the element x[i]
        lw     $t2, 4($t0)        # load the element x[i + 1]
        blt    $t1, $t2, no_swap  # if x[i] < x[i + 1] goto no_swap
        lw     $t1, 4($t0)        # x[i] = x[i + 1]
        lw     $t2, 0($t0)        # x[i + 1] = x[i]
no_swap:
        addiu  $t0, $t0, 4        # move to the next element (pointer moves by 4)
end_pass:
        jr      $ra
```

(Code fragment, not a complete program.)

Q8. It's lunchtime ... the company quiz for today is below. The winner is the first to answer all questions correctly. Work fast: Lee Pointa is really good with these type of questions.

(a) What's in `$s1`, `$s2`, and `$s3` at the end of the code fragment (i.e., not complete program)? (Write ERROR if the value is not an integer value.)

```
        .text
        .globl main
        [code omitted]
        la     $s0, L
        lw     $s1, 8($s0)
        lw     $s2, 16($s0)
        la     $s0, L1
        lw     $s3, -4($s0)
        [code omitted]

        .data
L:      .word 42 43 44 45
        .word 46 47 48 49
L1:     .word 50 51 52 53
        .word 54 55 56 57
```

Answer:
`$s1` $\boxed{8}$ `$s2` $\boxed{46}$ `$s3` $\boxed{\text{ERROR}}$

(b) What is the value of `$s2` at the end of the code fragment? (Write ERROR if the value is not an integer value.)

```
        .text
        .globl main
        [code omitted]
        la     $s0, W
        lw     $s1, 4($s0)
        lw     $s2, 8($s1)
        [code omitted]

        .data
X:      .word 1 2 3 4
Y:      .word 5 6 7 8
Z:      .word 9 10 11 12
W:      .word X Y Z
```

Answer:
`$s2` $\boxed{7}$

Q9. Write a MIPS code fragment that, starting with n (assume stored in `$s0`), it continually computes $3n+1$ if n is odd or $n/2$ if n is even and store it back into n until n is 1. (`/` is integer division.) The values of n are stores in an array in the data segment – a label `L` is already created for you (you don't have to create it). For instance if the user enters 3, then the values are

$$3, 10, 5, 16, 8, 4, 2, 1$$

For instance when $n = 3$, since n is odd, the next value of n is $3n+1 = 10$. The next value for $n = 10$, since $n = 10$ is even, is $n/2 = 10/2 = 5$. The values

$$3, 10, 5, 16, 8, 4, 2, 1$$

are stored at the beginning of the data segment. Note that the starting value $n = 3$ is also stored.

Answer:

```
        # $s0 - n
        la    $a1, L                    # a1 - &L[0]
        lw    $s0, 0($a1)               # n is at &x[0]
        li    $t0, 1
        li    $t1, 2
        li    $t3, 1                    # i-th element

loop:   beq   $s0, $t0, exit            # if n == 1 goto exit
        move  $t4, $t3                  # temporary store i
        addu  $t4, $t4, $t4
        addu  $t4, $t4, $t4
        addu  $a1, $t4, $a1             # a1 now holds the address &L[i + 4]

        div   $s0, $t1                  # n = n % 2
        mfhi  $t2                       # remainder of n % 2
        beqz  $t2, even                 # if n % 2 == 0 goto even case
        move  $t2, $s0                  # temporary store n in t2
        addu  $s0, $s0, $s0             # n = 2 * n
        addu  $s0, $s0, $t0             # n = 3 * n
        addiu $s0, $s0, 1               # n = 3 * n + 1
        j     print_n                   # print n and space

        lw    $s0, 0($a1)               # store n in the array L
        addiu $t3, $t3, 1               # increment i (i++)
        j     loop                      # jump back to loop

even:   div   $s0, $t1
        mflo  $s0                       # n = n / 2
        j     print_n                   # print n and space
        lw    $s0, 0($a1)               # store n in the array L
        addiu $t3, $t3, 1               # increment i (i++)
        j     loop                        # jump back to loop

print_n:
        li    $v0, 1
        move  $a0, $s0
        li    syscall

        li    $v0, 4
        move  $a0, SPACE
        li    syscall
        j     $ra

exit:   jr    $ra
```

Q10. Your boss just did a code review and found that Joe Cantcode wrote some MIPS code implementing the following idea in a critical part of the code to auto-correct the tilt actuators on a high speed train:

```
if (t0 == 0)
{
    t1 = 12;
}
else if (t0 == 1)
{
    t1 = 34;
}
else if (t0 == 2)
{
    t1 = 56;
}
else if (t0 == 3)
{
    t1 = 78;
}
```

where variables `t0`, `t1` and associated with registers `$t0`, `$t1` respectively. He has asked you rewrite the code to implement a switch in MIPS using the jump table method

```
switch (t0)
{
    case 0:
        t1 = 12;
        break;
    case 1:
        t1 = 34;
        break;
    case 2:
        t1 = 56;
        break;
    case 3:
        t1 = 78;
}
```

so that he can do some performance analysis and see if the jump table method is faster.

Answer. Implement the switch here

```
        .text
        .globl main
main:
        li $t0, 0           # your code must work for different values of t0
        li $t2, 3
        # implement switch below
        blt  $t0, $0, exit      # if t0 < 0 goto exit
        bgt  $t0, $t2, exit     # if t0 > 3 goto exit

        la   $a1, labels        # a1 - holds the base address
        addu $t0, $t0, $t0
        addu $t0, $t0, $t0
        addu $t0, $a0, $t0
        lw   $t0, 0($t0)        # t0 - hold the base addres + t0 * 4
        jr    $t0

L0:     li   $t1, 12
        move $a0, $t1
        li   $v0, 1
        syscall
        j    exit

L1:     li   $t1, 34
        move $a0, $t1
        li   $v0, 1
        syscall
        j     exit

L2:     li   $t1, 56
        move $a0, $t1
        li   $v0, 1
        syscall
        j    exit

L3:     li   $t1, 78
        move $a0, $t1
        li   $v0, 1
        syscall
        j    exit

        # At this point $t1 must be set to the right value based on your
        # implementation of switch
exit:   li $v0, 10          # exit
        syscall

        .data
labels: .word   L0 L1 L2 L3
```

Q11. Implement the following C/C++ code fragment in MIPS code fragment

```
y2 = x * x * x + 2 * x * x + 3 * x + 4;
```

where `y2` is `$s0` and `x` is `$s1`. Assume that the cost of multiplication is higher than the cost of addition. Say the cost of multiplication is 20 and addition is 1. You want your MIPS code to be as fast as possible since the above is used in performing error correction codes in signal processing. The project leader, Zig Nell, claims that it can be done with two multiplications.

Answer:

```
        move    $t0, $s1         # temp store x
        addu    $s1, $s1, $s1    # 2 * x
        addu    $s2, $s1, $t0    # 3 * x
        addui   $s2, $s2, 4      # 3 * x + 4
        mul     $s3, $t0, $t0    # x^2
        mul     $s4, $s3, $t0    # x^3
        addu    $s3, $s3, $s3    # 2 * x ^ 2
        addu    $s0, $s3, $s2    # y = 2 * x ^ 2 + 3 * x + 4
        addu    $s0, $s0, $s4    # y = x ^ 3 + 2 * x ^ 2 + 3 * x + 4
```

Q12. Just great. Bubba Short is not back from lunch. Lee Pointa said you have to write the bubblesort as a function for him. Remember that the base address of the array is stored in `$a0` and the number of values in the array is stored in `$a1`. Just to be somewhat organized Lee told you to include at least a `swap` function that swaps the integer values at addresses stored in `$a0` and `$a1`. (You can add more functions if you like. For instance you might want to have a function that performs one pass of bubblesort.)

Answer:

```
main:           # a0 - base address
                # a1 - size
                addu $a1, $a1, $a1
                addu $a1, $a1, $a1
                addu $a1, $a0, $a1   # hold the address of x[size]
                move $t0, $a1
                jal    bubblesort

swap:
                ble     $t3, $t2, no_swap
                lw      $t3, 4($a0)
                lw      $t4, 0($a0)
                jr       $ra

bubblesort:
                addui   $t0, $t0, -2       # outer loop counter *end = *size - 2
outer_loop      ble     $t0, $0, sorted      # if j <= 0 goto sorted
                li      $t2, 0               # inner loop counter
inner_loop:     bge     $t2, $t0, exit_inner_loop    # if i >= j exit_inner_loop                I
                lw      $t4, 4($a0)
                j       swap

no_swap:        j        inner_loop

exit_inner_loop:
                addiu   $t0, $t0, -1
                j       outer_loop

sorted:         j       $ra
```

Instructions

In `main.tex` change the email address in

```
\renewcommand\AUTHOR{jdoe5@cougars.ccis.edu}
```

to yours. In the bash shell, execute "`make`" to recompile `main.pdf`. Execute "`make v`" to view `main.pdf`. Execute "`make s`" to create `submit.tar.gz` for submission.

For each question, you'll see boxes for you to fill. You write your answers in `main.tex` file. For small boxes, if you see

```
1 + 1 = \answerbox{}.
```

you do this:

```
1 + 1 = \answerbox{2}.
```

`answerbox` will also appear in "true/false" and "multiple-choice" questions.

For longer answers that needs typewriter font, if you see

```
Write a C++ statement that declares an integer variable name x.
\begin{answercode}
\end{answercode}
```

you do this:

```
Write a C++ statement that declares an integer variable name x.
\begin{answercode}
int x;
\end{answercode}
```

`answercode` will appear in questions asking for code, algorithm, and program output. In this case, indentation and spacing is significant. For program output, I do look at spaces and newlines.

For long answers (not in typewriter font) if you see

```
What is the color of the sky?
\begin{answerlong}
\end{answerlong}
```

you can write

```
What is the color of the sky?
\begin{answerlong}
The color of the sky is blue.
\end{answerlong}
```

For students beyond 245: You can put LaTeX commands in `answerbox` and `answerlong`.

A question that begins with "T or F or M" requires you to identify whether it is true or false, or meaningless. "Meaningless" means something's wrong with the statement and it is not well-defined. Something like "$1+_2$" or "$\{2\}^{\{3\}}$" is not well-defined. Therefore a question such as "Is $42 = 1+_2$ true or false?" or "Is $42 = \{2\}^{\{3\}}$ true or false?" does not make sense. "Is $P(42) = \{42\}$ true or false?" is meaningless because $P(X)$ is only defined if X is a set. For "Is 1 + 2 + 3 true or false?", "1 + 2 + 3" is well–defined but as a "numerical expression", not as a "proposition", i.e., it cannot be true or false. Therefore "Is 1 + 2 + 3 true or false?" is also not a well-defined question.

When writing results of computations, make sure it's simplified. For instance write 2 instead of $1 + 1$. When you write down sets, if the answer is $\{1\}$, I do not want to see $\{1, 1\}$.

When writing a counterexample, always write the simplest.

Here are some examples (see `instructions.tex` for details):

1. T or F or M: $1 + 1 = 2$ .... $\boxed{\text{T}}$

2. T or F or M: $1 + 1 = 3$ .... $\boxed{\text{F}}$

3. T or F or M: $1+^2 =$ .... $\boxed{\text{M}}$

4. $1 + 2 = \boxed{3}$

5. Write a C++ statement to declare an integer variable named `x`.

```
int x;
```

6. Solve $x^2 - 1 = 0$.

   Since $x^2 - 1 = (x - 1)(x + 1)$, $x^2 - 1 = 0$ implies $(x - 1)(x + 1) = 0$. Therefore $x - 1 = 0$ or $x = -1$. Hence $x = 1$ or $x = -1$.

7. Which is true? .... $\boxed{\text{C}}$

   (A) $1 + 1 = 0$
   (B) $1 + 1 = 1$
   (C) $1 + 1 = 2$
   (D) $1 + 1 = 3$
   (E) $1 + 1 = 4$