CSC 230
Spring 2022 (A01: CRN 20772; A02: CRN 23816)
Test #2: Wednesday, 9 March 2022

Marking Key

Exam duration: 50 minutes

Instructor: Michael Zastre

Students must check the number of pages in this examination paper before beginning to write, and report any discrepancy immediately.

- **All answers are to be written on this exam paper.**
- The exam is closed book. Other than the MIPS32 reference provided to you, no books or notes are permitted.
- When answering questions, please do not detach any exam pages!
- ***Electronic devices are not permitted***. Cellphones must be turned off.
- Partial marks are available for the questions in sections B and C.
- There are nine (9) printed pages in this document, including this cover page.
- We strongly recommend you read the entire exam through from beginning to end before starting on your answers.
- **Please have your UVic ID card available for inspection by an exam invigilator.**

Question 1 (20 marks)

On this page are 20 terms, numbered from 1 to 20.

On the opposite page are 25 definitions, lettered from A to Y.

For each term on this page (page 2) choose the correct / best definition by writing the definition's letter beside the term.

Term	**Definition (letter): *your answers***
1. computer system	B
2. CPU	Y
3. data memory	L
4. directive	K
5. endianness	F
6. epilog	C
7. execute	X
8. frame pointer	V
9. hexadecimal	T
10. I-format instruction	M
11. instruction decode	E
12. jump-and-link	W
13. local variables	S
14. named register	J
15. null-terminated string	O
16. pass-by-reference	P
17. pseudo-instruction	Q
18. R-format instruction	N
19. word	G
20. writeback	R

Definition (letter)	Definition (text)
A	Region of memory set aside specifically for storing MIPS32 instructions that access data.
B	Collection of CPU, memory, devices.
C	After this code is complete, the CPU may safely execute the `jr $ra` instruction.
D	Core Programming Utility.
E	The phase of CPU cycle where the meaning of the operation is determined.
F	The MIPS32 architecture supports either the "big" or "little" version of this (although MARS supports only one of them).
G	Same size in bits as a MIPS32 instruction.
H	A special system call used at the end of program to indicates that the program is indeed finished.
I	Are local to a loop body (i.e., assembler automatically eliminates these variables when it detects code has exited a loop).
J	An example would be `$a3`.
K	An example of this in the assembler would be .word
L	Region of memory that includes the run-time stack.
M	The resulting encoding has 16-bit immediate field.
N	Needed to encode an operation such `slt` or `xor`.
O	Used in assignment #2 as a way to represent the original text of messages.
P	A better choice for procedure arguments where the value of particular data argument may not fit within a single register.
Q	A convenience to the programmer who would otherwise be forced to write all code using core operations.
R	The fourth phase of a CPU cycle.
S	Can facilitate recursive-style programming when stored in a stack frame.
T	One of these digits represents four bits of information.
U	The result of placing two octal digits side-by-side.
V	Utilized in fairly advanced techniques for managing stack frames.
W	Equivalent to a procedure call.
X	The phase that occurs after instruction decode.
Y	Another name for the computational core of computer system.

Question 2 (8 marks)

Consider the following encoded MIPS32 R-format instruction.

`0x01356020`

For this given machine instruction, answer the following.

a) What are the `Rs`, `Rt`, and `Rd` values for this encoding? Explain your answer as completely as possible.

The bit representation is:

0000 00**01 001**1 **0101 0110 0**000 0010 0000

The five bits for Rs, Rt, and Rd are shown in **blue**, **orange**, and **green**

Therefore Rs is $9 (or $t1), Rt is $21 (or $s5), and Rd is $12 (or $t4)

[4 marks]

b) What is the actual R-format instruction (including registers) in MIPS32 assembler? Explain your answer as completely as possible.

Given that this is an R-format instruction, the left-most six bits (opcode field) are always zero. Therefore the actual operation intend for the instruction is determined by the shamt and funct fields (in orange and green respectively).

0000 0001 0011 0101 0110 0**000 00**10 **0000**

The value in funct is 0x20 which corresponds to an add (see MIPS green sheet, under "Core Instruction Set").

Therefore the full instruction is:

add $12, $9, $21 or **add $t4, $t1, $s5**

[4 marks]

(This page intentionally left blank.)

Question 3 (10 marks)

Consider the code fragment given to you below.

The number of bytes used to grow the stack and to shrink the stack is not yet shown (i.e., `????` is used as a placeholder for this number).

```
    .text

    # ... <snip> ...

    # Assume p6_procedure is a leaf procedure.
    #

    # The stack frame must be the correct size to hold
    # a local array of 6 integers and a local array
    # of 16 characters, in addition to the other uses
    # of the stack as shown.
    #
    # To help with word-alignment of memory, the size of the
    # stack must be a multiple of four (i.e., there may
    # anywhere from 1 to 3 unusable bytes in the stack frame).
    #

p6_procedure:
    addi $sp, $sp, ????
    sw $s0, 0($sp)
    sw $s1, 4($sp)
    sw $s2, 8($sp)
    sw $s3, 12($sp)
    sw $s4, 16($sp)

    #
    # CODE BODY NOT SHOWN
    #

    lw $s0, 0($sp)
    lw $s1, 4($sp)
    lw $s2, 8($sp)
    lw $s3, 12($sp)
    lw $s4, 16($sp)

    addi $sp, $sp, ????
    jr $ra
```

a) Given the requirements for the stack frame in the comment before the procedure `p6_procedure` – that is, for a local integer array and a local char array – what number of bytes is needed to grow the stack? To shrink the stack? Briefly explain your answer, including any assumptions.

Given that the stack must also hold two local arrays, 24 bytes are needed for the integer arrays (i.e. 6 times 4 bytes), 16 bytes for chars (as we can stored four chars in a word), and then 20 bytes for saved registers. That makes a total of 60 bytes.

[3 marks]

b) What addresses on the stack, relative to the stack pointer, will you make available for the integer array? Briefly explain your answer, including any assumptions.

Many answers are possible, but each answer should reflect that the local arrays will not overlap. The address given must be fairly precise (i.e. offset from the top of the stack). **Refer to slides 146 and 147 from the "Functions" section for some related diagrams.**

[2 marks]

c) What addresses on the stack, relative to the stack pointer, will you make available for the char array? Briefly explain your answer, including any assumptions.

Many answers are possible, but each answer should reflect that the local arrays will not overlap. The address given must be fairly precise (i.e. offset from the top of the stack).

[2 marks]

d) Relative to the stack pointer, what is the address of the integer indexed in array location 0? And what is the address of the integer indexed in array location 4? Briefly explain your answer, including any assumption.

This answer will depend upon choices described in parts (b) and (c).

[3 marks]

Question 4 (16 marks)

Write a procedure named “scramble". It takes a single 32-bit value in `$a0`, and this value is treated as a packed sequence of bytes. For example, if the value in `$a0` is `0xa59ae8d7`, then this value consists of the following bytes:

- byte 0: `0xd7`
- byte 1: `0xe8`
- byte 2: `0x9a`
- byte 3: `0xa5`

The procedure `scramble` is to return in `$v0` the result of the following arithmetic calculation:

(byte 3) - (byte 0) + (byte 1) + (byte 2)

Your solution, of course, must work for more 32-bit values than just the one provided as an example. You need not worry about little- vs. big-endian order. Write only the code for the procedure (that is, do not write code for calling the procedure from some `main` program).

Hint: The mask `0x00ff` is identical to `0x000000ff` in MIPS32.

Some marks will be given for the quality of your answer.

Your final answer must be written within the box provided on page 9 (opposite). However, you may use this page 8 to work out some ideas.

```
scramble:     # One possible solution....
    addi $sp, $sp, -16
    sw $s0, 0($sp)
    sw $s1, 4($sp)
    sw $s2, 8($sp)
    sw $s3, 12($sp)

    add $s0, $zero, $a0
    add $s1, $zero, $a0
    add $s2, $zero, $a0
    add $s3, $zero, $a0

    srl $s3, $s3, 24
    andi $s3, $s3, 0x0ff
    andi $s0, $s0, 0x0ff
    srl $s1, $s1, 8
    andi $s1, $s1, 0x0ff
    srl $s2, $s2, 16
    andi $s2, $s2, 0x0ff

    sub $s3, $s3, $s0
    add $s3, $s3, $s1
    add $s3, $s3, $s2

    add $v0, $zero, $s3

    lw $s0, 0($sp)
    lw $s1, 4($sp)
    lw $s2, 8($sp)
    lw $s3, 12($sp)
    addi $sp, $sp, 16
    jr $ra
```

The answer should reflect a properly-written procedure/subroutine. This is a leaf procedure and therefore there will not be a need to save or restore $ra. However, marks will not be deducted for omitting (or for including) the saving/restoring of $ra.

- Proper procedure name? **2 marks**
- Grow and shrink stack to an appropriate amount? **2 marks**
- Save and restore any used callee-save registers? **2 marks**
- Appropriate masking and extraction of bytes? **4 marks**
- Appropriate arithmetic? **4 marks**
- Storing result in correct location? Placement of jr instruction? **2 marks**