University of Calgary
Department of Electrical and Computer Engineering
ENCM 369: Computer Organization
Lecture Instructors: Steve Norman and Norm Bartley

Winter 2023 Practice Test #1A for Sections 02 and 03

Please do *not* write your U of C ID number on this cover page.

Name (printed):

Signature:

Lecture section:

General Instructions

- Marks will be recorded on the **last** page of this question paper. When you are told to start the test, the first thing you should do is to put your name, signature, U of C ID number, and lecture section in the appropriate spaces at the bottom of the last page.
- If you use a **calculator**, it must be one of the following models sanctioned by the Schulich School of Engineering: Casio FX-260, Casio FX-300MS, TI-30XIIS.
- The test is **closed-book**. You may not refer to books or notes during the test, with one exception: you may refer to the *Reference Material* page that accompanies this test paper.
- You are not required to add **comments** to assembly language code you write, but you are strongly encouraged to do so, because writing good comments will improve the probability that your code is correct and will help you to check your code after it is finished.
- Some problems are relatively **easy** and some are relatively **difficult**. Go after the easy marks first.
- To reduce distraction to other students, you are not allowed to leave during the last **ten minutes** of the test.
- Write all answers on the question paper and hand in the question paper when you are done.
- Please print or write your answers **legibly**. What cannot be read cannot be marked.
- If you write anything you do not want marked, put a large X through it and write "rough work" beside it.
- You may use the backs of pages for rough work.

PROBLEM 1 *(12 marks)*
Consider the C code listed to the right. Translate the function quux into RARS assembly language. Follow the usual calling conventions from lectures and labs, and use only instructions from the Midterm Instruction Subset described on the *Reference Material* page.

```
int foo(int x);
int bar(int x);
void quux(int *dest, int n,
          const int *src)
{
  int y;
  int *guard;
  guard = dest + n;
  while (dest != guard) {
    y = *src;
    if (y > -10 && y < 10)
      *dest = foo(y);
    else
      *dest = bar(y);
    dest++;
    src++;
  }
}
```

PROBLEM 2 (*total of 12 marks*). In this problem, you are asked to translate sequences of one or more C statements into sequences of one or more RARS instructions, *not* complete RARS functions. *Use only instructions from the Midterm Instruction Subset.* Use as many t-registers as you wish for intermediate values.

Example. *(No marks.)*

- `s0` is used for `x`, of type `int`.

C code	RARS translation
`x = 42;`	addi s0, zero, 42

Part a. *(4 marks.)*

- `s0` is used for `x`, of type `int*`.
- `s1` is used for `i`, of type `int`.

C code	RARS translation
`x[i] = x[i - 1];`	

Part b. *(5 marks.)*

- `a0` is used for `d`, of type `char*`.
- `a1` is used for `s`, of type `char*`.
- `t5` is used for `i`, of type `int`.
- `t6` is used for `c`, of type `char`.

C code	RARS translation

`do {`
`  c = s[i];`
`  d[i] = c;`
`  i++;`
`} while (c != '\0');`	

Part c. *(3 marks.)*

- `a` is an array of `int` elements on the stack, and the address of `a[0]` is `24(sp)`.
- `j` is a variable of type `int` on the stack, and the address of `j` is `20(sp)`.
- `k` is a variable of type `int` on the stack, and the address of `k` is `16(sp)`.
- The prototype for `f` is `void f(int *y, int n, int *s);`.

C code	RARS translation
`f(a, j, &k);`	

PROBLEM 3 *(14 marks).* The RARS assembly-language program on this page is a *correct* translation of the C program.

Fill in the table of GPRs and the blank boxes in the diagram of memory with *numbers* to show the state of the assembly-language program, at the *last time* it will get to POINT ONE.

label	address
`foo`	`0x0040_0078`
`times10`	`0x0040_00f4`
`main`	`0x0040_0104`
`gs`	`0x1001_0000`

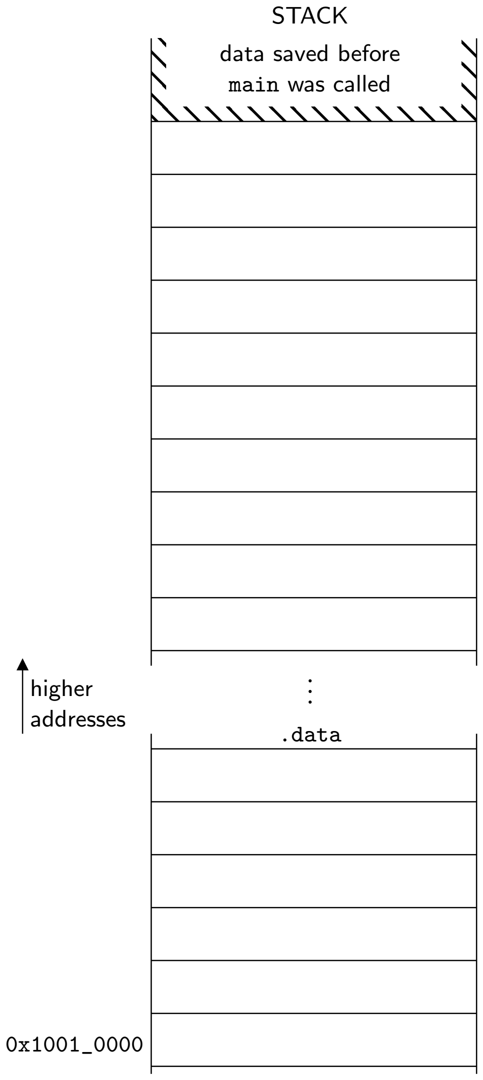
GPR	GPR value when `main` starts
`s0`	100
`s1`	300
`s2`	500
`s3`	700
`sp`	`0x7fff_ef80`
`ra`	`0x0040_0064`

Use base ten or hexadecimal format for numbers, whichever is more convenient for any particular number. Note that some of the memory words in the diagram might not be used by the program.

GPR values, *last time* at POINT ONE

GPR	value
`a0`	
`t2`	
`t5`	
`s0`	
`s3`	
`sp`	
`ra`	

Memory, *last* time at POINT ONE

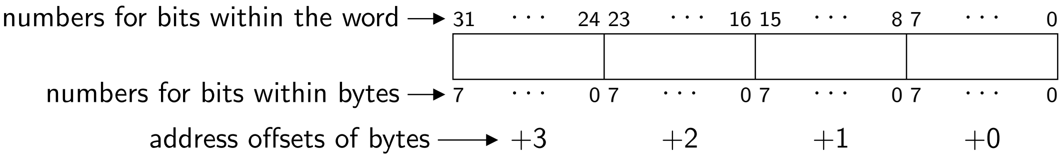


```
int times10(int x);

void foo(int *d,
         const int *s,
         int n)
{
  int i = 0, v;
  for ( ; i < n; i++) {
    if (s[i] > 0)
      v = times10(s[i]);
    else
      v = 0;
    d[i] = v;
  }
}

int times10(int x)
{
  return 10 * x;
}

int gs[ ] =
  {2, -4, 6, -8};

int main(void)
{
  int x[4];
  foo(x, gs, 4);

  // ... more code ...

  return 0;
}
```

```
     .text
     .globl  foo
foo: addi  sp, sp, -20
     sw    ra, 16(sp)
     sw    s3, 12(sp)
     sw    s2, 8(sp)
     sw    s1, 4(sp)
     sw    s0, 0(sp)
     addi  s0, a0, 0
     addi  s1, a1, 0
     addi  s2, a2, 0

     addi  s3, zero, 0
L1:  bge   s3, a2, L4
     slli  t1, s3, 2
     add   t2, s1, t1
     lw    t3, (t2)
     ble   t3, zero, L2
     addi  a0, t3, 0
     jal   times10
     j     L3
L2:  addi  a0, zero, 0
L3:  slli  t5, s3, 2
     add   t6, s0, t5
     sw    a0, (t6)
     addi  s3, s3, 1
     j     L1
L4:
     lw    s0, 0(sp)
     lw    s1, 4(sp)
     lw    s2, 8(sp)
     lw    s3, 12(sp)
     lw    ra, 16(sp)
     addi  sp, sp, 20
     jr    ra

     .globl times10
times10:
     slli  t5, a0, 3

     # POINT ONE

     slli  t6, a0, 1
     add   a0, t5, t6
     jr    ra

     .data
     .globl  gs
gs:  .word   2, -4, 6, -8
     .text
     .globl  main
main:
     addi  sp, sp, -20
     sw    ra, 16(sp)

     addi  a0, sp, 0
     la    a1, gs
     addi  a2, zero, 4
     jal   foo

     # ... more code ...

     addi  a0, zero, 0

     lw    ra, 16(sp)
     addi  sp, sp, 20
     jr    ra
```

PROBLEM 4 (*total of 9 marks*)

Part a. *(3 marks.)* The following diagram shows the arrangement of bytes within a RARS memory word:



Suppose that `t0` contains `0x8090_a0b0` and `t1` contains `0x1001_0004` when the following code fragment starts. *Using hexadecimal format for numbers,* fill in the table to indicate register values when the fragment has finished execution.

```
    sw    t0, (t1)
    addi  t2, zero, 0x789
    sb    t2, 2(t1)
    lw    t4, (t1)
    lb    t5, 1(t1)
    lbu   t6, 3(t1)
```

GPR	value
t4	
t5	
t6	

Part b. *(3 marks.)* Consider the RARS code fragment given below. *Again using hexadecimal format for numbers,* fill in the table to indicate register values when the fragment has finished execution.

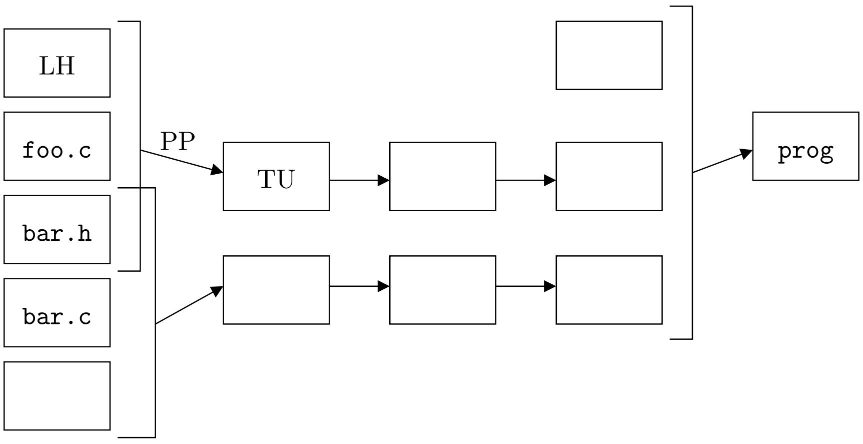
```
    lui   t0, 0x50000
    lui   t1, 0xc0000
    and   t4, t0, t1
    xor   t5, t0, t1
    srli  t6, t1, 3
```

GPR	value
t4	
t5	
t6	

Part c. *(3 marks.)* A programmer writes three source files: `foo.c`, `bar.h`, and `bar.c`, then builds an executable called `prog` using the command

```
gcc foo.c bar.c -o prog
```

In the diagram below, blank rectangles represent *files* involved in the build process, and each arrow represents use of one of the *tools* in the C development *toolchain.* Put labels on all the blank rectangles and arrows, using the key given to the right of the diagram. (A few labels have been given to you as examples.)



A	assembler
ALF	assembly language file
C	compiler
L	linker
LH	library header file
LMC	library machine code
OF	object file
PP	preprocessor
TU	translation unit

MARKS: The space below will be used to record your marks for each question and your overall test mark. Please put your name, signature, and U of C ID number in the appropriate places.

Name (printed): ________________________

Signature: ________________________

UCID number: ____________

Lecture section: ____

Problem	Mark
1	/ 12
2	/ 12
3	/ 14
4	/ 9
TOTAL	/ 47