Programming Assignment #3
RISC-V

CS 3220 / CS 5220 Spring 2024
20 points (grad: 30 points)
due Tuesday, Mar. 26th, 11:59 pm

1 RISC-V Assembly Language

You'll write several functions in RISC-V, using our web-based RISC-V interpreter, with parameter passing, a local variable, and proper stack management. The interpreter is here: https://riscv.jhibbele.w3.uvm.edu

You may work individually or with a partner. Graduate students must work individually.

1.1 First function

Write RISC-V code to implement this function:

```
int f1(int p) {
  int k = 18;
  k = p + k;
  return k;
}
```

And then create a "main" (actually, just a label), and call this function with the parameter 12.

Your function should start at a label named `f1`.

Use the pattern of "one parameter, one local variable" as a framework for your function. Use the `a0` register for the parameter; use `a0` also for the local variable `k`; use `a0` and `a1` to do your calculation; and put the return value in `a0`.

You'll need 16 bytes (four words) of storage on the stack, so the first four instructions in your function should be these:

```
f1:
  addi   sp, sp, -16  # reserve 4 words on the stack
  sw     ra, 12(sp)   # save ra
  sw     s0, 8(sp)    # save s0
  addi   s0, sp, 16   # set s0: top of stack
```

and the last four lines of your function should be:

```
  lw     s0, 8(sp)    # restore s0
  lw     ra, 12(sp)   # restore ra
  addi   sp, sp, 16   # restore the stack pointer
  jalr   x0, ra, 0    # return (jump to ra)
```

Be sure to put the return value in `a0`. You'll also need to save `p` and `k` on the stack, in the region you reserve.

Here's a diagram of the stack:

save ra	this is ra when the function starts
save s0	this is s0 when the function starts
save a0	this is the parameter (p)
local var k	this is the local variable

Call the function by putting 12 into `a0` and executing `jal ra, f1`. This will let `f1` return to the instruction after the function call. After the function returns, you should see the value 30 in the `a0` register.

Before you call the function, set the stack pointer to a non-zero value, such as 64:

```
addi sp, x0, 64
```

and then set the parameter and call the function:

```
addi a0, x0, 12
jal ra, f1
```

1.2 A second function

Write RISC-V code to implement this function:

```
int f2(int p1, int p2) {
  if ( p1 > p2 )
    return p1;
  else if ( p1 < p2 )
    return p2;
  else
    return 0;
}
```

Use two parameters. There are no local variables, so you can use allocate four words for the stack: `ra`, `s0`, and the two parameters.

Your stack should look like this:

save ra	this is ra when the function starts
save s0	this is s0 when the function starts
save a0	this is the first parameter (p1)
save a1	this is the second parameter (p2)

1.3 Local variables

Whenever you want modify a local variable, you should first load the variable from the stack into a register, then make the modification, and then save the updated value back to the stack. So for example, if the local variable `i` is in `sp + 16`, and I want to increment `i`, then I could do the following:

```
lw a0, 16(sp)
addi a0, a0, 1
sw a0, 16(sp)
```

1.4 Code examples

I've put example RISC-V code, showing how various high-level structures are coded in RISC-V, in the class repo in gitlab.

2 Testing

Put your code in a file named `riscv.netid.s`. As you change your code in the interpreter, be sure to save it (there is no autosave feature).

Use comments liberally—comment every line. This will help you organize your thoughts and keep track of what your code is doing. Watch the values of the registers as your code runs. Use breakpoints as appropriate. Coding in assembly language takes practice and concentration and attention to detail.

3 What to Submit

Submit your file `riscv.netid.s` containing your `f1` and `f2` and `f3` and your `main` (which will contain a call to each function).

4 Graduate Students

Students taking the course for graduate credit, and undergraduates for a bit of extra credit: write two additional functions: `intdiv()` and `intsqrt()`.

4.1 Integer division

Write RISC-V code to implement this function:

```
int intdiv(int numer, int denom) {
  if ( denom <= 0 )
    error 8;
  if ( numer < 0 )
    error 8;
  int q = 0;
  while  (numer >= 0 ) {
    numer = numer - denom;
    q = q + 1;
  }
  return q-1;
}
```

Given integers *numer* and *denom*, this will produce the largest integer q such that $q \times denom \leq numer$.

4.2 Integer square root

It's straightforward to approximate the square root of positive real number by using an iterative technique. For an integer p, we can define the integer square root of p to be the largest integer n such that $n^2 \leq p$, and we can employ the same iterative technique, using integer arithmetic.

Write RISC-V code to implement this function:

```
int intsqrt(int p) {
  if ( p < 0 )
    error 8;
  int i0 = p >> 1; // this divides p by 2
  if ( i0 > 0 ) {
    int i1 = ( i0 + p / i0 ) >> 1;  // this divides by 2
    while ( i1 < i0 ) {
      i0 = i1;
      i1 = ( i0 + p / i0 ) >> 1;  // divide by 2
    }
    return i0;
  } else {
    return p;
  }
}
```

Use your `intdiv` function to do the division. To call `intdiv`, put the first parameter in `a0` and the second parameter in `a1`. After the function returns, the result will be in `a0`. Put your two functions in a file named `riscv-grad.netid.s`.