CSM 61C Fall 2020

RISC-V

Mentoring 2: September 23, 2020

1 Load and Store

1.1 For each line of RISC-V code, answer what will be the value saved into the registers involved or the effect on memory. Assume that x8 contains a valid address in memory, such that Mem[R[x8]] = 0x00000180.

(a) lw x9, 0(x8) What value does x9 now store?

0x00000180

(b) lb x10, 1(x8) What value does x10 store?

0x00000001

(c) lb x11, 0(x8) What value does x11 store?

0xFFFFFF80

(d) sb x11, 3(x8) What's the effect on Mem[R[x8]]?

Mem[R[x8]] = 0x80000180

(e) lbu x12, 0(x8) What value does x12 store?

0x00000080

2 RISC-V to C

2.1 Assume we have two arrays input and result. They are initialized as follows:

```
int *input = malloc(8*sizeof(int));
int *result = calloc(8, sizeof(int));
for (int i = 0; i < 8; i++) {
    input[i] = i;
}
```

You are given the following RISC-V code. Assume register x10 holds the address of input and register x12 holds the address of result.

```
    addi x8, x0, 0
    addi x5, x0, 0
    addi x11, x0, 8
Loop:
    beq x5, x11, Done
    lw x6, 0(x10)
    add x8, x8, x6
    slli x7, x5, 2
    add x7, x7, x12
    sw x8, 0(x7)
    addi x5, x5, 1
    addi x10, x10, 4
    j Loop
Done:
    // exit
    . . . . . .
```

Translate this RISC-V code into C, assuming `sizeof(int) == 4`.

```
int sum = 0;

for (int i = 0; i < 8; i++) {
    sum += input[i];
    result[i] = sum;
}
```

2.2 What is the end array stored starting at register x12?

$[0, 1, 3, 6, 10, 15, 21, 28]$

3 Jumpin' Around

3.1 Consider the following RISC-V code:

```
0x00000000:          Anakin:   add x0, x0, x0
0x00000004:                    jal ra, TryIt
0x00000008:                    addi t0, t0, 16
0x0000000C:                    bne t0, x0, Power
0x00000010:                    j HighGround
0x00000014:                    add x0, x0, x0
0x00000018:    HighGround:     addi t0, t0, 32
0x0000001C:                    jal t0, Exit
0x00000020:          TryIt:    add t0, t0, t0
0x00000024:                    jr ra
0x00000028:                    add t0, x0, t0
0x0000002C:          Power:    jalr ra, t0, 0
0x00000030:                    add t0, x0, x0
0x00000034:           Exit:    add x0, x0, x0
```

Write out each instruction in the order it's executed, and the contents of the registers below at each step. Assume the registers are all zeroed out at first.

pc	ra	t0
0x00000000	0x00000000	0x00000000

pc	ra	t0
0x00000000	0x00000000	0x00000000
0x00000004	0x00000000	0x00000000
0x00000020	0x00000008	0x00000000
0x00000024	0x00000008	0x00000000
0x00000008	0x00000008	0x00000000
0x0000000C	0x00000008	0x00000010
0x0000002C	0x00000008	0x00000010
0x00000010	0x00000030	0x00000010
0x00000018	0x00000030	0x00000010
0x0000001C	0x00000030	0x00000030
0x00000034	0x00000030	0x00000020

4 More RISC-V

4.1 You wish to speed up one of your programs by implementing it directly in assembly. Your partner started translating the function `is_substr()` from C to RISC-V, but didn't finish. Please complete the translation by filling in the lines below with RISC-V assembly. The prologue and epilogue have been written correctly but are not shown.

Note: `strlen()`, both as a C function and RISC-V procedure, takes in one string as an argument and returns the length of the string (not including the null terminator).

```
/* Returns 1 if s2 is a substring of s1, and 0 otherwise. */
int is_substr(char* s1, char* s2) {
    int len1 = strlen(s1);
    int len2 = strlen(s2);
    int offset = len1 - len2;
    while (offset >= 0) {
        int i = 0;
        while (s1[i + offset]  == s2[i]) {
            i += 1;
            if (s2[i] == '\0')
                return 1;
        }
        offset -= 1;
    }
    return 0;
}
```

Fill in the following RISC-V code based on the given C code:

```
1. is_substr:
2.      mv s1, a0
3.      mv s2, a1
4.      jal ra, strlen
5.      mv s3, a0
6.      mv a0, s2
7.      jal ra, strlen
8.      sub s3, s3, a0
9. Outer_Loop:
10.     ______ ______, ______, False
11.     add t0, x0, x0
12. Inner_Loop:
13.     add t1, t0, s3
14.     add t1, s1, t1
15.     lbu t1, 0(t1)
16.     ____________
17.     ____________
18.     ______ t1, ______, Update_Offset
19.     addi t0, t0, 1
20.     add t2, t0, s2
21.     ____________
22.     beq t2, ______, ______,
23.     jal x0 Inner_Loop
24. Update_Offset:
25.     addi s3, s3, -1
26.     ____________
27. False:
28.     xor a0, a0, ______
29.     jal x0, End
30. True:
31.     addi a0, x0, 1
32. End: ______
```

```
10.     blt s3, x0, False
16.     add t2 s2 t0
17.     lbu t2 0(t2)
18.     bne t1, t2, Update Offset
21.     lbu t2 0(t2)
22.     beq t2, x0, True
26.     jal x0 Outer_Loop
28.     xor a0, a0, a0
32. End: ret
```