CSM 61C
Fall 2019

RISC-V
Mentoring 2: September 23, 2019

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.

```
    add x8, x0, 0
    addi x5, x0, 0
    addi x11, 0, 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
    . . . . . . . . . . . . .

// sizeof(int) == 4
int sum = 0;
```

for (int i = 0; i ¡ 8; i++) sum += a[i]; c[i] = sum;

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

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

Meta: This is a challenging question for students since they are all new to RISC-V. Make sure to walk through and write out each single RISC-V instruction functionality first. Since the lecture will not cover the detailed name for each register, it will be good just going along with x0 - x31. Drawing all the detailed memory diagram will be helpful for this question since it involves a lot of load and store instructions.

3 Linked List Reversals in RISC-V

3.1 Assume we have the following linked list node struct:

```
struct node{
    int val;
    struct node * next;
};
```

Also, recall the function to reverse a linked list iteratively, given a pointer to the head of the linked list.

```
void reverse(struct node * head){
    struct node * prev = NULL;
    struct node * next;
    struct node * curr = head;
    while(curr != NULL){
        next = curr->next;
        curr->next = prev;
        prev = curr;
        curr = next;
    }
}
```

3.2 Now assume `a0` contains the address of the head of a linked list. Fill in the function below to reverse a linked list. Assume ‘reverse’ follows calling conventions. ‘reverse’ doesn’t return anything. You may not need all lines.

```
1.  reverse: ______________
2.  ______________
3.  ______________
4.  ______________
5.  add s0 a0 x0
6.  xor s2 s2 s2 #s2 corresponds to the pointer ‘prev’
7.  loop: ___ s0 x0 exit
8.  ______________
9.  ______________
10. add s2 s0 x0
11. add s0 s1 x0
12. j loop
13. exit: ______________
14. ______________
15. ______________
16. addi sp sp 12
17. j ra
```

```
1.  reverse: addi sp sp -12
2.  sw s0 0(sp)
3.  sw s1 4(sp)
4.  sw s2 8(sp)

7.  beq s0 x0 exit
8.  lw s1 4(s0)
9.  sw s2 4(s0)

13. exit:lw s0 0(sp)
14. lw s1 4(sp)
15. lw s2 8(sp)
```

Lines 1-4 and 13-15 is just following calling conventions for RISC-V, since `s0`-`s11` by convention, must be preserved when someone calls ‘reverse’. We notice that `s0`, `s1`, and `s2` are all being modified.