CSM 61C Spring 2020

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?

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

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

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

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

2 More RISC-V

2.1 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;
}
```

2.2 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: addi s3, s3, -1
25. ____________
26. False: xor a0, a0, ______
27. jal x0, End
28. True: addi a0, x0, 1
29. End: ______.
```

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
```