

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 $\text{Mem}[\text{R}[\text{x8}]] = 0\text{x}00000180$.

(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 $\text{Mem}[\text{R}[\text{x8}]]$?

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

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;
```

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

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
```