

1 Calling Convention

1.1 For parts a and b, answer either Caller, Callee or Neither as applicable. The Caller is the function passing these values to a new call, and the Callee is the function being called. That is, if we invoke a function `doStuff()` from `main()`, `doStuff()` is the Callee and `main()` is the Caller.

- (a) Whose responsibility is it to save the return address (ra) in a function call?
- (b) Whose responsibility is it to save the temporary registers (t0-t6)? What about the saved registers (s0-s11)?
- (c) Whose responsibility is it to save the argument registers (a0-a7)? Why?

2 RISC-V Functions

2.1 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;
    }
}
```

- 2.2 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 #s0 corresponds to curr
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. jr ra

```

3 RISC-y Conversions

3.1 Convert the different RISC-V commands into their hex form or convert the hex form into RISC-V. The instructions are in order of when they would be executed.

(a) `0x005004B3`

(b) `lw t5, 17(t6)`

(c) `sll s9, x9, t0`

(d) `0x03CE2283`

(e) `jalr a0, x11, 8`

(f) `ori t1, t2, 5`

(g) `lui a7, 0xCF61C`

4 Instruction Format Design

Prof. Wawrzynek decides to design a new ISA for his ternary neural network accelerator. He only needs to perform 7 different operations with his ISA: `xor`, `add`, `lw`, `sw`, `lui`, `addi`, and `blt`. He decides that each instruction should be 17 bits wide, as he likes the number 17. There are no `funct7` or `funct3` fields in this new ISA.

- 4.1 What is the minimum number of bits required for the opcode field?
- 4.2 Suppose Prof. Wawrzynek decides to make the `opcode` field 6 bits. If we would like to support instructions with 3 register fields, what is the maximum number of registers we could address?

Given that the `opcode` field is 6 bits wide and each register field is 2 bits wide in the 17 bit instruction, answer the following questions:

- 4.3 Using the assumptions stated in the above description, how many bits are left for the immediate field for the instruction `blt` (Assume it takes `opcode`, `rs1`, `rs2`, and `imm` as inputs)?
- 4.4 Let n be your answer in part (a). Suppose that `blt`'s branch immediate is in units of instructions (i.e. an immediate of value 1 means branching 1 instruction away). What is the maximum number of bits a `blt` instruction can jump forward from the current `pc` using these assumptions? Write your answer in terms of n .
- 4.5 Using the assumptions stated in the description, what is the most negative immediate that could be used in the `addi` instruction (Assume it takes `opcode`, `rs1`, `rd`, and `imm` as inputs)?
- 4.6 For `LUI`, we need `opcode`, `rd`, and `imm` as inputs. Using the assumptions stated in the description, how many bits can we use for the immediate value?

5 Advanced RISC-V

5.1 You are given the following RISC-V code:

```
Loop:  andi t2 t1 1
      srli t3 t1 1
      bltu t1 a0 Loop
      jalr s0 s1 MAX_POS_IMM
```

(a) What is the value of the byte offset that would be stored in the immediate field of the bltu instruction?

(b) What is the binary encoding of the bltu instruction? Please use hexadecimal to represent your answer.

5.2 As a curious 61C student, you question why there are so many possible opcodes, but only 47 instructions. Thus, you propose a revision to the standard 32-bit RISC-V instruction formats where each instruction has a unique opcode (which still is 7 bits). You believe this justifies taking out the funct3 field from the R, I, S, and SB instructions, allowing you to allocate bits to other instruction fields except the opcode field.

(a) What is the largest number of registers that can now be supported in hardware?

(b) With the new register size, how far can a jal instruction jump to (in half-words)?

(c) Assume register `s0 = 0x1000 0000`, `s1 = 0x4000 0000`, `PC = 0xA000 0000`. Let's analyze the instruction `jalr s0, s1, MAX_POS_IMM` where `MAX_POS_IMM` is the maximum possible positive immediate for `jalr`. Using the register sizes defined above, what are the values in registers `s0`, `s1`, and `pc` after the instruction executes?