CSM 61C
Spring 2019

RISC-V Instruction Formats

Mentoring 3: February 3, 2019

1 RISC-y Conversions

1.1 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`

`add s1, x0, t0`

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

`0x011FAF03`

(c) `sll s9, x9, t0`

`0x00549CB3`

(d) `0x03CE2283`

`lw t0, 60(t3)`

(e) `jalr a0, x11, 8`

`0x00858567`

(f) `label: ori t1, t2, 5`

`0x0053E313`

(g) `lui a7, 0xCF61C`

`0xCF61C8B7`

2 Reverse Linked List in RISC-V

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;
    }
}
```

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 doesnt 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)
```

3 RISC-V Potpourri

3.1 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, LD, 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.

What is the minimum number of bits required for the opcode field?

$\lceil \log_2 7 \rceil = 3$

Binary encoding, which requires least number of bits, is used here. In order to represent 7 operations, we need at least $\lceil \log_2 7 \rceil = 3$ bits.

(b) 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?

The instruction is 17 bit wide, 6 bits are used for opcode, we have 11 bits left for register indexing. Given we need 3 register fields, we can have $11/3 = 3$ bits per register field which means we could address 8 registers.

(c) 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:

(a) Using the assumptions stated in the description of part (d), how many bits are left for the immediate field for the instruction BLT (Assume it takes opcode, rs1, rs2, and imm as inputs)?

$17 - 6 - 2 - 2 = 7$

BLT has 1 opcode field (-6), 2 register fields $(-2-2)$, we can use the rest $17 - 6 - 2 - 2 = 7$ bits for expressing jump offset.

(b) Let n be your answer in part (i). Suppose that BLTs 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.

$(2^{n-1}1)17$

In 2s complement, the range of an n-bit number is $[2n1, 2n11]$. jumping forward means that the offset is positive. With n-bit 2s complement offset, we can jump forward $2n11$ instructions, which is $(2n11)17$ bits since each instruction is 17 bits wide.

(c) Using the assumptions stated in the description of part (d), 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 RISC-V to C Translations

4.1 Translate the following RISCV code into C code.

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, x0
    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
    .

// sizeof(int) == 4
```

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

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.