CSM 61C
Fall 2019

RISC-V Instruction Formats

Mentoring 3: September 30, 2019

1 More RISC-V

1.1 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;
}
```

1.2 Fill in the following RISC-V code based on the given C code:

```
is _substr:
 mv s1, a0
 mv s2, a1
 jal ra, strlen
 mv s3, a0
 mv a0, s2
 jal ra, strlen
 sub s3, s3, a0
Outer_Loop:
______, ______, ______, False
add t0, x0, x0
Inner_Loop:
add t1, t0, s3
add t1, s1, t1
lbu t1, 0(t1)
____________
____________
______, t1, ______, Update_Offset
addi t0, t0, 1
add t2, t0, s2
____________
beq t2, ______, ______,
jal x0 Inner_Loop
Update_Offset: addi s3, s3, -1
____________
False: xor a0, a0, ______
jal x0, End
True: addi a0, x0, 1
End: ______.
```

2 RISC-y Conversions

2.1 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`

3 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.

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

3.2 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?

3.3 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 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)?

(b) 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 bytes a `blt` instruction can jump forward from the current `pc` using these assumptions? Write your answer in terms of n.

(c) 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)?

(d) 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?

4 Advanced RISC-V

4.1 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.

4.2 As a curious 61C student, you question why there are so many possible `opcodes`, but only 47 instructions. Thus, your 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 halfwords)?

(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?