CSM 61C Spring 2019 Advanced RISC-V and Intro to Caching

Mentoring 4: February 10, 2019

1 Substrings

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 didnt 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:

```
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: ______.
```

2 Advanced RISC-V

2.1 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.

2.2 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`. Lets 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?

3 Cache Rules Everything Around Me

3.1 You are given a RISCV machine with a single level of 2KiB direct-mapped cache with 512B cache blocks. It has 1MiB of physical address space.

The function `foo()` is ran on the system with a cold cache and as the only process:

```
#define ARRAY_LEN 4096
#define STEP_SIZE 64
// A starts at 0x10000
// B starts at 0x20000
void foo (int* A, int* B) {
    int total = 0;
    for ( int i = 0; i < ARRAY_LEN; i += STEP_SIZE )
        {
            total += A[ i ];
            total -= B[ i ];
        }
}
```

(a) Calculate the number of Tag, Index, and Offset bits for this cache.

(b) Calculate the hit percentage for this cache after running foo.

(c) The cache is now cleared and the code is run again. This time, `A` and `B` are pointing to the same array, which starts at `0x10000`. Calculate the new hit percentage.

(d) Assume `A` and `B` starts once again at `0x10000` and `0x20000`. What is the new hit percentage if we ran foo on a fully associative cache, with all other parameters staying the same?