HW4.4. RISC-V Calling Convention

Q1.1: Which of the following statements are true about calling convention (abbreviated as CC)?

- [ ] (a) CC rules apply when jumping within the same function (ex. through loops or branches).
- [x] (b) CC rules apply when jumping to a different function.
- [x] (c) The purpose of CC is to build abstractions within function interactions and regulate assembly code structure.
- [ ] (d) In order to follow CC, we need to store all registers on the stack at the start of a function, even if we don't use a particular register during our function.
- [ ] (e) The RISC-V language mandates that CC be followed, and as such will refuse to assemble or run code that doesn't follow CC.
- [x] (f) Apart from x0 and the starting values of certain registers such as `sp`, all 32 registers are fundamentally identical in behavior.
- [ ] (g) The register `sp` is used to store a pointer to the top of the stack
- [x] (h) CC dictates that anything above the stack pointer at the start of a function call remains unchanged, while anything below the stack pointer is unallocated.
- [x] (i) In order to allocate space on the stack, the stack pointer is moved down. This creates a space which is guaranteed unchangeable by other functions, but modifyable by the current function, and can thus be used for temporary storage.
- [ ] (j) The caller of a function can assume that any "a" register is unchanged after a function call.
- [ ] (k) The caller of a function can assume that the "ra" register is unchanged after a function call.
- [x] (l) The caller of a function can assume that any "s" register is unchanged after a function call.
- [x] (m) The caller of a function can assume that the "sp" register is unchanged after a function call.
- [ ] (n) The caller of a function can assume that any "t" register is unchanged after a function call.
- [x] (o) The callee does not need to restore the original value of any "a" register.
- [x] (p) The callee does not need to restore the original value of the "ra" register.
- [ ] (q) The callee does not need to restore the original value of any "s" register.
- [ ] (r) The callee does not need to restore the original value of the "sp" register.
- [x] (s) The callee does not need to restore the original value of any "t" register.
- [x] (t) It doesn't matter how you ensure that register values get restored (ex. storing on the stack, not using the register, adding and subtracting the same amount), as long as the registers that need to be restored are guaranteed to return to its original value.
- [ ] (u) Efficient code tends to maximize the amount of data stored on the stack, because accessing the stack is faster than accessing registers.
- [ ] (v) You should never use x0 as a destination register.

Select all possible options that apply.

✓ 100%

Try a new variant

Correct answer

Homework 4

Assessment overview

Total points: 42/100

Score: 42%

Question

Value: 10

History: 10

Awarded points: 10/10

Report an error in this question

Previous question

Next question

Attached files

No attached files

Attach a file

Attach text

(b) CC rules apply when jumping to a different function.
(c) The purpose of CC is to build abstractions within function interactions and regulate assembly code structure.
(f) Apart from x0 and the starting values of certain registers such as `sp`, all 32 registers are fundamentally identical in behavior.
(h) CC dictates that anything above the stack pointer at the start of a function call remains unchanged, while anything below the stack pointer is unallocated.
(i) In order to allocate space on the stack, the stack pointer is moved down. This creates a space which is guaranteed unchangeable by other functions, but modifyable by the current function, and can thus be used for temporary storage.
(l) The caller of a function can assume that any "s" register is unchanged after a function call.
(m) The caller of a function can assume that the "sp" register is unchanged after a function call.
(o) The callee does not need to restore the original value of any "a" register.
(p) The callee does not need to restore the original value of the "ra" register.
(s) The callee does not need to restore the original value of any "t" register.
(t) It doesn't matter how you ensure that register values get restored (ex. storing on the stack, not using the register, adding and subtracting the same amount), as long as the registers that need to be restored are guaranteed to return to its original value.

There is a slight distinction as to where calling convention should be applied with regards to labels---we'll refer to different labels as the following categories: loop/iterative, branch/selection, and all other ones (Note that labels handle almost all changes to program flow). Calling convention needs to be followed on everything **but** loop and branch labels, regardless of if they're within a file that solely you will edit or whether they're external files. For loops, they're just acting effectively as points in memory you return to within the same function whilst looping (loops in C aren't a separate function); similarly for branching statements, you're not really declaring a new function, just a place to jump to in memory within the same function. As long as you're within the same function, you can choose how you use any registers, as you are in control of the jumper and the jumpee. Technically, this also applies as long as you're working with functions that aren't going to be used externally (like helper functions), but **you should still follow CC**. The general rule is that you MUST follow CC if there's a chance that someone else will eventually need to interact with your function. Also, "you three hours from now" counts as someone else.

Unlike most other registers, the register `sp` is not initially set to 0 at the start of a program, but instead set to the bottom of the stack (The gp and tp registers have a similar use, but are not covered in this course). This generally means that it is used solely to denote the bottom of the stack, and is thus moved up and down as the program uses temporary memory. The stack is an easy place to put data, since data stored there won't be changed by other functions. This is why there was seemingly random data in the stack in question 2.10; saved registers were placed on the stack to keep their values stored while we use saved registers (C is generally translated to x86, which is another assembly language, but its underlying principles are similar to that of RISC-V).

Some general rules to keep in mind when practising good calling convention are the following: the **caller** is responsible for storing volatile registers (ra, t0-t6, a0-a7) either into memory or on the stack (**note this is NOT the same as storing the values into sp**). The **callee** is responsible for storing the non-volatile registers (sp, s0-s11) in a similar fashion. CC is NOT enforced by the system, so it's a good idea to write out somewhere what registers you plan to use and why.

Generally speaking, register accesses are orders of magnitude faster than memory accesses. While stack accesses tend to be faster than arbitrary memory accesses, it is still generally much faster to keep data in registers. As such, the general rule when deciding which registers to use is to think about what would minimize the number of stack accesses.

While `x0` is hardcoded not to change, that doesn't mean that you shouldn't ever use it as a destination register. Rather, using `x0` as a destination allows for instructions like `jal` to be run without their side effects (indeed, the `j` pseudoinstruction is translated to `jal x0`.

Submitted answer 6 **correct: 100%**

Submitted at 2022-09-20 23:59:16 (PDT)

show

Submitted answer 5 **incorrect: 0%**

Submitted at 2022-09-20 23:58:56 (PDT)

show

Submitted answer 4 **incorrect: 0%**

Submitted at 2022-09-20 23:58:39 (PDT)

show

Show/hide older submissions