Lab alarm

RISC-V Assembly

- For this lab, good to understand the basics (go through the questions!)
- Has important differences with x86
  - Load (l*) and store (s*) instructions -- load from memory to register and store to memory from register
  - What is the instruction for jump?
  - Pay attention to what the operands are
- Register naming and conventions are different
- Consult textbook or cheat sheet: https://www.cl.cam.ac.uk/teaching/1617/ECAD+Arch/files/docs/RISCVGreenCardv8-20151013.pdf

The Stack (Review)

- Grows downwards. So to expand it, need to subtract stack pointer (sp)
- Frame pointer (s0/fp) points to the base of the stack
- Return address saved right below fp
- Last fp saved right below return address
- Local variables live below the last fp

Answer the four questions in the lab description before attempting the lab -- it helps to review the calling conventions and to familiarize with RISC-V

Alarm

- Some programs want to run stuff periodically
- We (the OS) provide the mechanism to do so without blocking
- Interface:
  - Process P registers a callback/handler function to be called every X ticks using syscall SIGALARM
  - Every X ticks, the OS interrupts the execution of P, saves the context, and jumps to the callback
  - At the end of the callback, the process calls syscall SIGRETURN
  - The OS restores the original execution context of P, jumps back to where P was interrupted, and resumes execution

Consider:

- How does the user unregister a handler?
- How do we make the process execute the handler (and return to the original point later)? (hint: modify a register in the process' trap frame)
- It is necessary to have a SIGRETURN syscall? Can we have the user directly return from the handler? How? (this is one of the challenges)

Implementation

- The tests are divided into two parts
- First part makes sure the handler is called at all (test0)
- Second part makes sure that:
  - The handler is called multiple times (test1)
  - The interrupted code is resumed correctly (test1)
  - No reentrant handler calls; i.e. if handler hasn't returned, don't call it again (test2)
- We recommend that you work in stages, in the above order

Questions?