CS6640 Handout Week2.b

1. Background: RISC-V assembly I

a) registers
   [see “RISC-V registers” in reference page]

b) **addi** rd, rs1, immediate

```
rd = rs1 + immediate
```

c) **sw** rs2, offset(rs1)

```
Memory[rs1 + offset] = rs2
```

d) **mv** rd, rs1

```
rd = rs1
```

e) **call** rd, symbol

```
rd = pc+4
pc = &symbol
```

If rd is omitted, ra is implied.

f) **ret**

```
pc = ra
```

2. Context switch in user-space:

a) void ctx_start(void** old_sp, void* new_sp);

This will be used when starting a new thread.
It will save registers on the old stack,
      store current stack pointer to “old_sp”,
      switch stack to the “new_sp”,
      and
      finally call ctx_entry().

b) void ctx_switch(void** old_sp, void* new_sp);

This will be used for context switch.
It will save registers on the old stack,
      store current stack pointer to “old_sp”,
      switch stack to the “new_sp”,
      and
      restore registers from the new stack,
      finally return (to ra).

3. grass/context.S

```
ctx_start:
    addi sp,sp,-64
    sw s0,4(sp)        /* Save callee-saved registers */
    sw s1,8(sp)
    sw s2,12(sp)
    sw s3,16(sp)
    sw s4,20(sp)
    sw s5,24(sp)
    sw s6,28(sp)
    sw s7,32(sp)
    sw s8,36(sp)
    sw s9,40(sp)
    sw s10,44(sp)
    sw s11,48(sp)
    sw ra,52(sp)       /* Save return address */
    sw sp,0(a0)        /* Save the current stack pointer */
    mv sp,a1           /* Switch the stack */
    call ctx_entry     /* Call ctx_entry() */

ctx_switch:
    addi sp,sp,-64
    sw s0,4(sp)        /* Save callee-saved registers */
    sw s1,8(sp)
    sw s2,12(sp)
    sw s3,16(sp)
    sw s4,20(sp)
    sw s5,24(sp)
    sw s6,28(sp)
    sw s7,32(sp)
    sw s8,36(sp)
    sw s9,40(sp)
    sw s10,44(sp)
    sw s11,48(sp)
    sw ra,52(sp)       /* Save return address */
    sw sp,0(a0)        /* Save the current stack pointer */
    mv sp,a1           /* Switch the stack */
    lw s0,4(sp)        /* Restore callee-saved registers */
    lw s1,8(sp)
    lw s2,12(sp)
    lw s3,16(sp)
    lw s4,20(sp)
    lw s5,24(sp)
    lw s6,28(sp)
    lw s7,32(sp)
    lw s8,36(sp)
    lw s9,40(sp)
    lw s10,44(sp)
    lw s11,48(sp)
    lw ra,52(sp)       /* Restore return address */
    addi sp,sp,64
    ret
```

4. An example use of ctx_start+ctx_entry

```
void thread_create(void (*f)(void *), void *arg, unsigned int stack_size) {

  tcb = create_thread_control_block();
  old_tcb = current_running_thread_control_block();

  … // do something necessary

  void **old_sp = …   // old stack pointer’s address in old_tcb
  void *new_sp =  …   // new stack pointer in tcb

  ctx_start(old_sp, new_sp);
}


void ctx_entry(void){

    … // do something useful

    (*f)(arg);  // run function “f” received by “thread_create”

    … // wrap up
}
```

5. How do context switches interact with I/O?

This assumes a user-level threading package.

The thread calls something like "fake_blocking_read()". This looks to the thread as though the call blocks, but in reality, the call is not blocking:

```
int fake_blocking_read(int fd, char* buf, int num) {

    int nread = -1;

    while (nread == -1) {

        /* this is a non-blocking read() syscall */
        nread = read(fd, buf, num);

        if (nread == -1 && errno == EAGAIN) {
            /*
             * read would block. so let another thread run
             * and try again later (next time through the
             * loop).
             */
            yield();
        }
    }

    return nread;
}
```