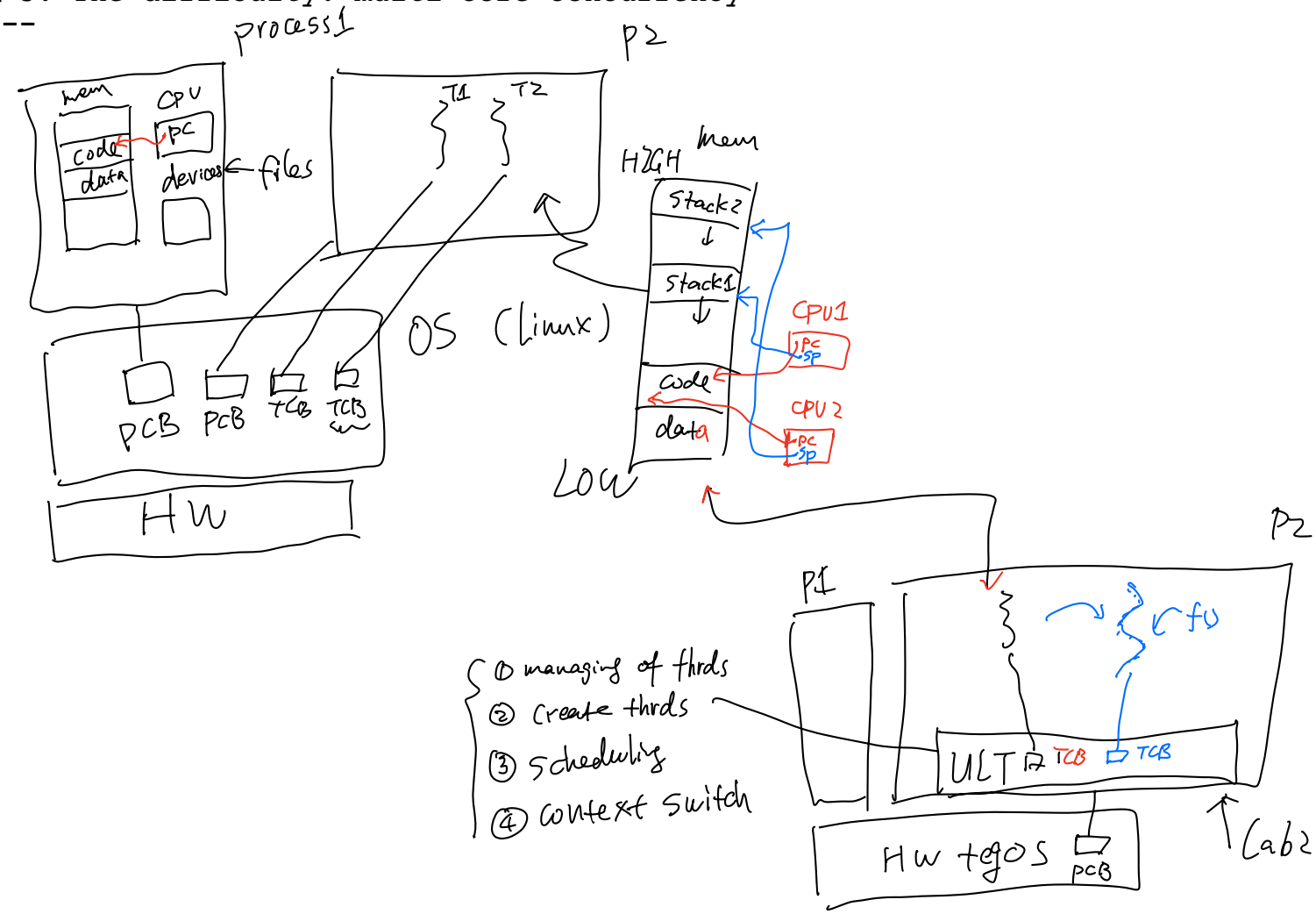
Week 03a
CS6640
01/20 2026
https://naizhengtan.github.io/26spring/

- [x] 0. Admin
- [x] 1. Recap: threading
- [ ] 2. egos assembly I
- [ ] 3. Context switches in user-space
- [ ] 4. Cooperative vs. Preemptive multithreading in user-level
- [ ] 5. The difficulty: multi-core concurrency



Q: what C program "components" do the four carry?
   -- stack: ? local vars
   -- text: ? Code
   -- heap: ? malloc mem
   -- data: ? global vars

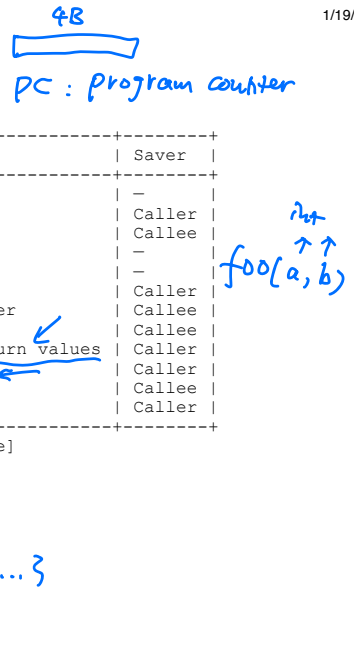
Q: are all four absolutely necessary? Can we get rid of some?

OSI Handout Week03.a: RISC-V assembly I

1. Background I: Registers

Register	ABI Name	Description	Saver
x0	zero	Zero constant	–
x1	**ra**	Return address	Caller
x2	**sp**	Stack pointer	Callee
x3	gp	Global pointer	–
x4	tp	Thread pointer	–
x5–x7	t0–t2	Temporaries	Caller
x8	s0 / fp	Saved / frame pointer	Callee
x9	s1	Saved register	Callee
x10–x11	**a0–a1**	Function args / return values	Caller
x12–x17	**a2–a7**	Function args	Caller
x18–x27	s2–s11	Saved registers	Callee
x28–x31	t3–t6	Temporaries	Caller

[see also "RISC-V registers" in Reference page]



2. Background II: RISC-V assembly I

a) **addi** rd, rs1, immediate

```
rd = rs1 + immediate
```

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

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

c) **mv** rd, rs1

```
rd = rs1
```

d) **call** rd, symbol

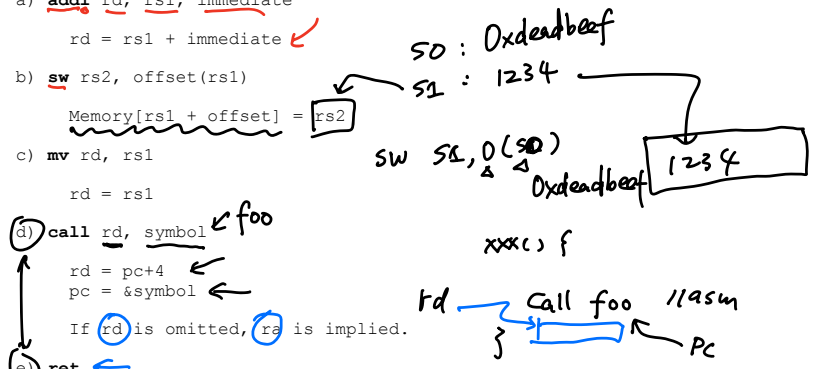
```
rd = pc+4
pc = &symbol
```

If rd is omitted, ra is implied.

e) **ret**

```
pc = ra
```

f) See others in "RISC-V instruction listings"
(on Reference page of OSI homepage)

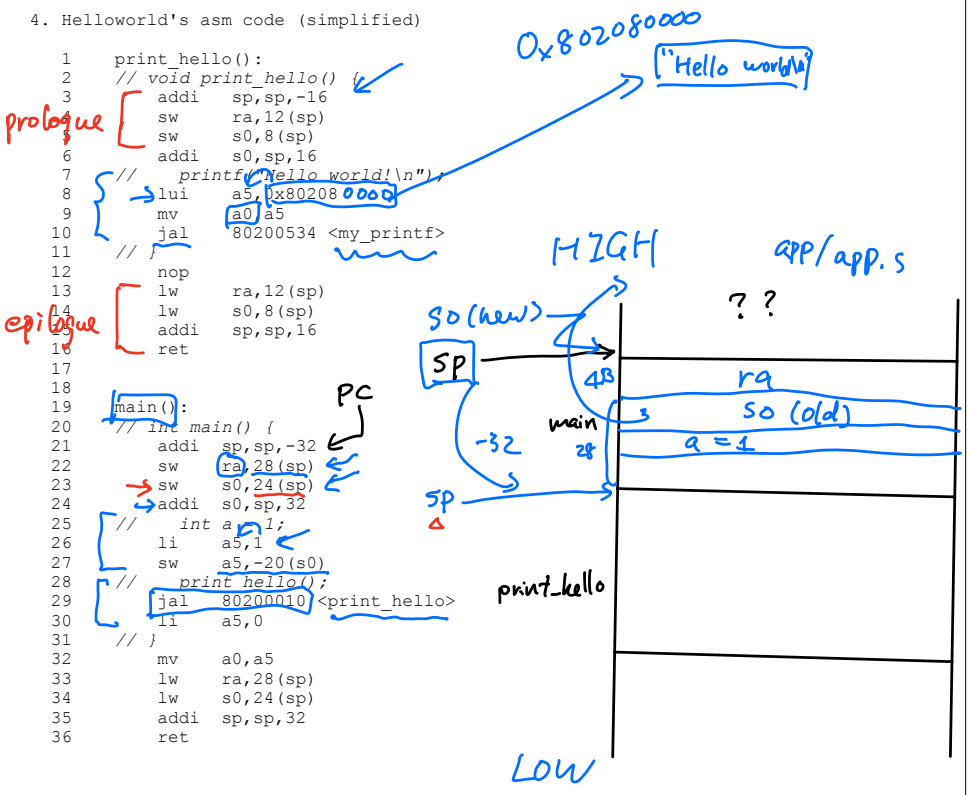


3. Hellworld's C code (headers omitted)

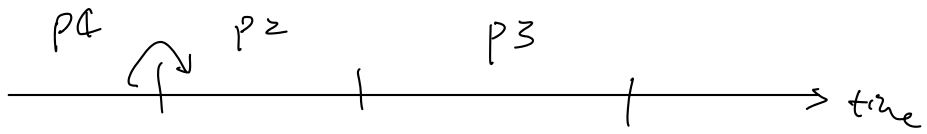
```
1    void print_hello() {
2        printf("Hello world!\n");
3    }
4
5    int main() {
6        int a = 1;
7        print_hello();
8    }
```


4. Helloworld's asm code (simplified)

```
1    print_hello():
2    // void print_hello() {
3        addi   sp,sp,-16
4        sw     ra,12(sp)
5        sw     s0,8(sp)
6        addi   s0,sp,16
7    //     printf("Hello world!\n");
8        lui    a5,0x80208000
9        mv     a0,a5
10       jal    80200534 <my_printf>
11   // }
12       nop
13       lw     ra,12(sp)
14       lw     s0,8(sp)
15       addi   sp,sp,16
16       ret
17
18
19   main():
20   // int main() {
21       addi   sp,sp,-32
22       sw     ra,28(sp)
23       sw     s0,24(sp)
24       addi   s0,sp,32
25   //     int a = 1;
26       li     a5,1
27       sw     a5,-20(s0)
28   //     print_hello();
29       jal    80200010 <print_hello>
30       li     a5,0
31   // }
32       mv     a0,a5
33       lw     ra,28(sp)
34       lw     s0,24(sp)
35       addi   sp,sp,32
36       ret
```



- Context Switch:



Q: which are in process's context?
- A. %pc
- B. %sp
- ✓ C. stack
- D. heap
- E. other processes' memory

Q: which are in thread's context?
- ✓ A. %pc
- ✓ B. %sp
- C. stack
- D. heap
- E. other processes' memory

OSI Handout Week03.a: Context Switch

1. Context switch in user-space:

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

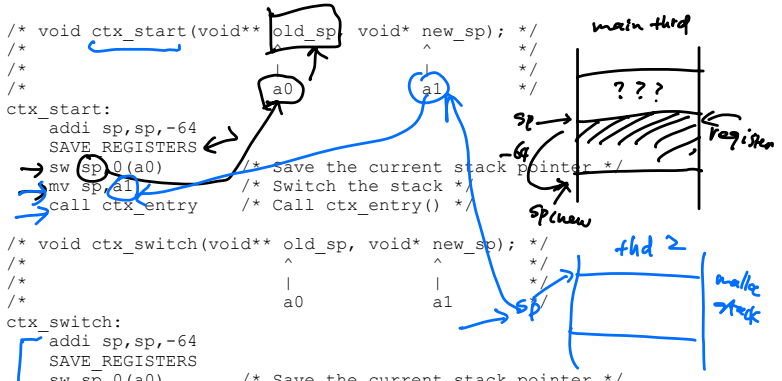
This will be used when starting a new thread.
It will
1. save registers on the old stack,
2. store current stack pointer to "old_sp",
3. switch stack to the "new_sp",
4. and
5. finally call ctx_entry().

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).

2. user/apps/thread.s

```
1       .macro SAVE_REGISTERS
2           sw s0,4(sp)        /* Save callee-saved registers */
3           sw s1,8(sp)
4           sw s2,12(sp)
5           sw s3,16(sp)
6           sw s4,20(sp)
7           sw s5,24(sp)
8           sw s6,28(sp)
9           sw s7,32(sp)
10          sw s8,36(sp)
11          sw s9,40(sp)
12          sw s10,44(sp)
13          sw s11,48(sp)
14          sw ra,52(sp)       /* Save return address */
15      .endm
16
17      .macro RESTORE_REGISTERS
18          lw s0,4(sp)        /* Restore callee-saved registers */
19          lw s1,8(sp)
20          lw s2,12(sp)
21          lw s3,16(sp)
22          lw s4,20(sp)
23          lw s5,24(sp)
24          lw s6,28(sp)
25          lw s7,32(sp)
26          lw s8,36(sp)
27          lw s9,40(sp)
28          lw s10,44(sp)
29          lw s11,48(sp)
30          lw ra,52(sp)       /* Restore return address */
31      .endm
32
33      /* void ctx_start(void** old_sp, void* new_sp); */
34      /*                      ^            ^          */
35      /*                      |            |          */
36      /*                      a0           a1         */
37      ctx_start:
38          addi sp,sp,-64
39          SAVE_REGISTERS
40          sw sp,0(a0)        /* Save the current stack pointer */
41          mv sp,a1           /* Switch the stack */
42          call ctx_entry     /* Call ctx_entry() */
43
44      /* void ctx_switch(void** old_sp, void* new_sp); */
45      /*                      ^            ^          */
46      /*                      |            |          */
47      /*                      a0           a1         */
48      ctx_switch:
49          addi sp,sp,-64
50          SAVE_REGISTERS
51          sw sp,0(a0)        /* Save the current stack pointer */
52          mv sp,a1           /* Switch the stack */
53          RESTORE_REGISTERS
54          addi sp,sp,64
55          ret
```



3. 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
}
```