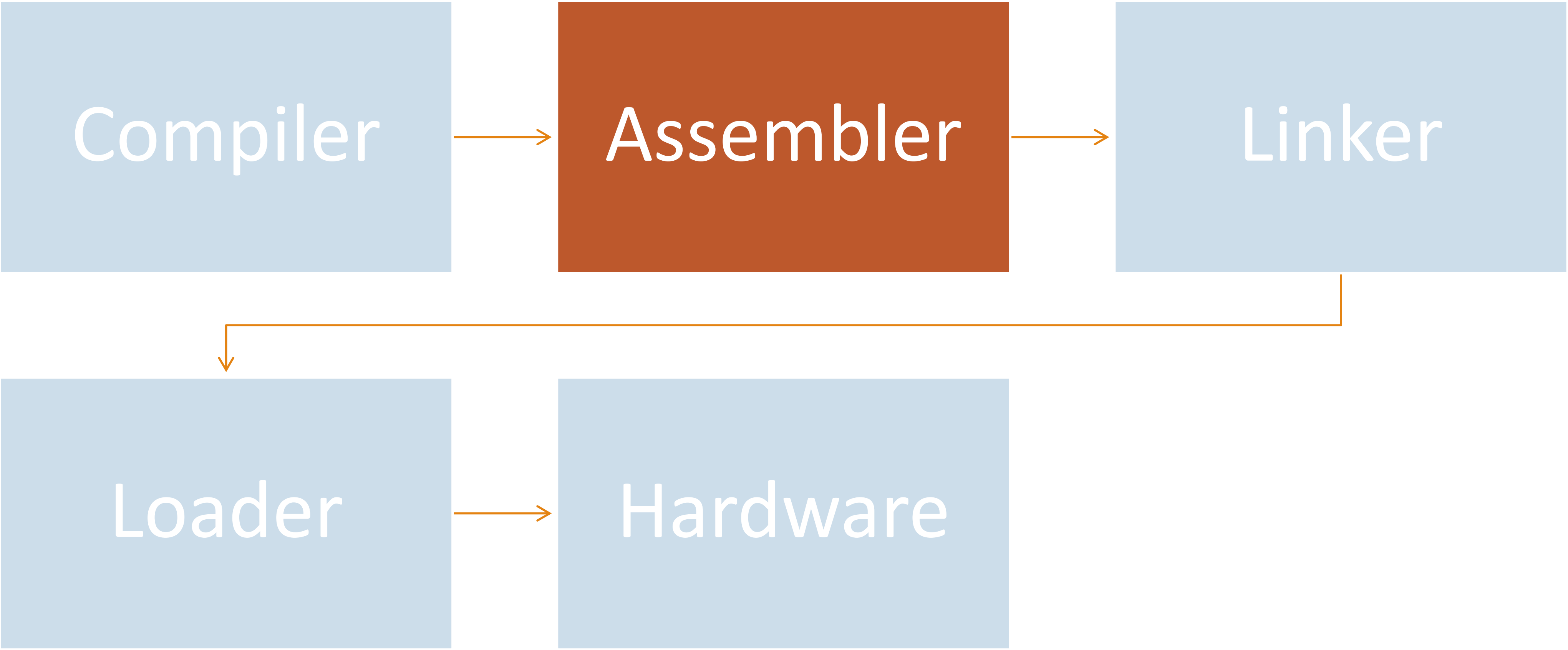
Discussion 3 – RISC-V

WANG RUOYU

WANGRY@SHANGHAITECH.EDU.CN

Where are we?



Assembly Language – RISC-V

ISA: Instruction Set Architecture, has two classes.

- RISC: Reduced Instruction Set Computing, e.g. MIPS, RISC-V
- CISC: Complex Instruction Set Computing, e.g. x86

RISC-V: One of RISC ISA (Instruction Set Architecture)

What makes a good ISA?

- Programmability
- Implementation
- Compatibility

Variables vs. Registers

RISC-V has 32 registers

- Every register is 32-bit.
- Have unique name.
- We should use its name rather than the number, e.g. s5 rather than x21.
- Registers have no type definition, everything is number.

You SHOULD NOT use registers as variables.

- Registers are faster but expensive.
- Therefore, the number of them are very limited.
- Store data in memory, only extract them when you want to use them.

Registers

- zero: This register always keep the number of 0

- ra: Return address, used in function call.

- sp: Stack pointer, used to point the stack top.

- s0/fp: Frame pointer, also used in function call, more advanced usage, learn more in CS131 Compiler.

- t0-t6: Temporaries, cannot trust them after function call.

- s1-s11: Saved, should not change after function call, you should maintain them when write a function.

- a0-a1: Function argument and return values, also argument of environment call.
- a2-a7: Function argument, used to pass parameters in function call.

Memory

- RISC-V does not require word alignment.
- But you’d better do this.

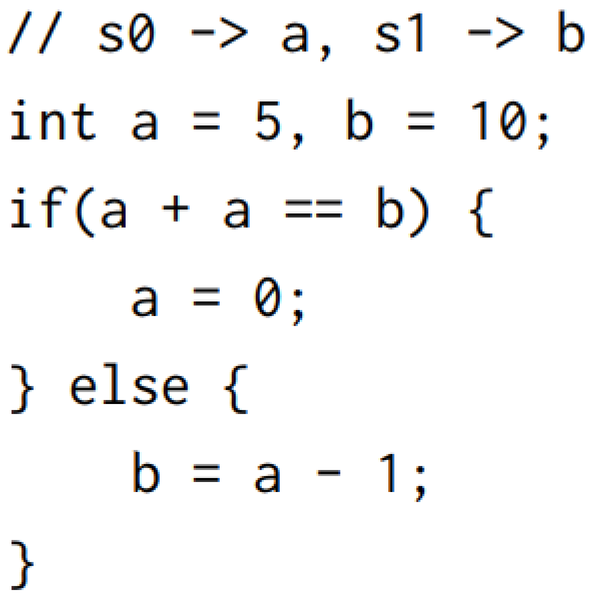
- **sw** stands for store word.
  - sw s2, 4(sp) → store 32 bits (1 word) data into the address store in sp plus 4 bytes.
- **lw** stands for load word.
  - lw sp, -4(sp) → load 32 bits data from the address (sp – 4) into sp.
- There are also sb, sh, sd, lb, etc., but the most useful are these two.

- This two instruction use memory on stack.
- If you want to use memory on heap, use environment call 9.
- sp, s0-s11, ra, which you should maintain them value but need to use now: **push them on stack.**

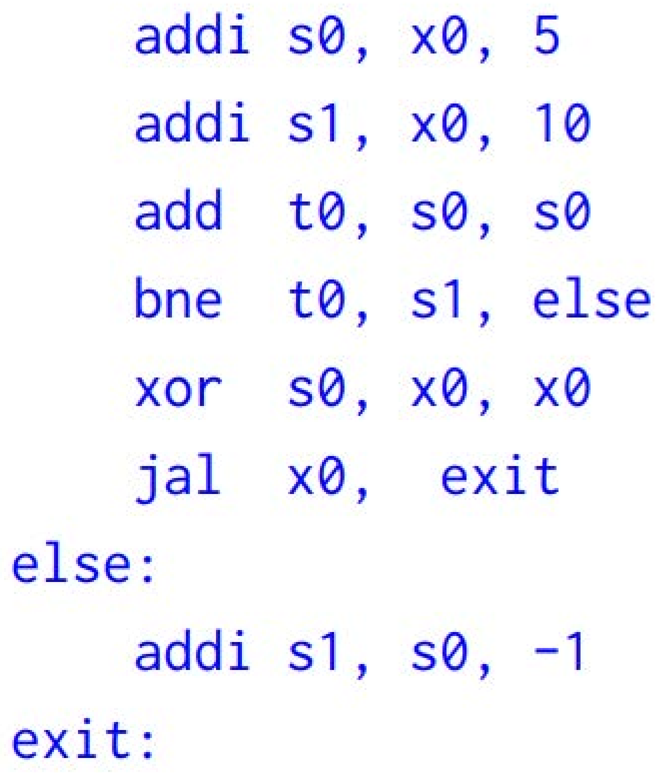
Label and Branch

- Giving a line name by adding label.
- Then, you can go the label by jump or branch.
- You can use label in function call, if-else, loop, etc.
- Let your label easy to understand, that makes you easy to finish the given tasks.

Quiz1

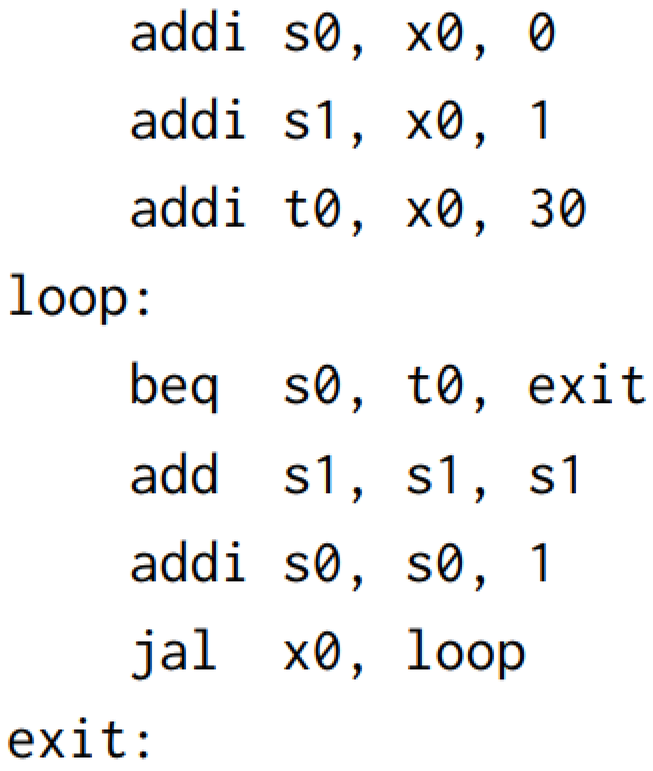


```
// s0 -> a, s1 -> b
int a = 5, b = 10;
if(a + a == b) {
    a = 0;
} else {
    b = a - 1;
}
```

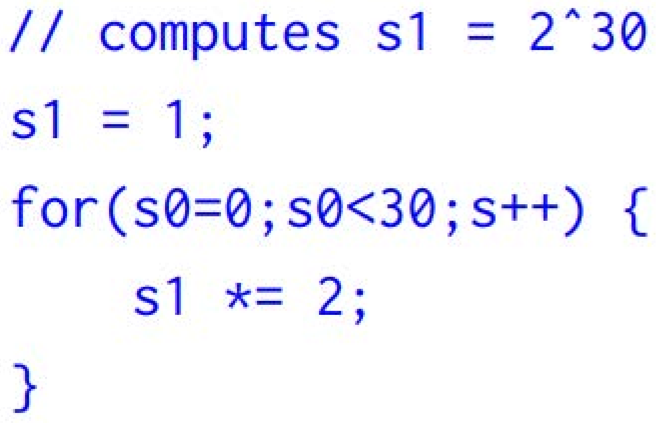


```
    addi s0, x0, 5
    addi s1, x0, 10
    add  t0, s0, s0
    bne  t0, s1, else
    xor  s0, x0, x0
    jal  x0,  exit
else:
    addi s1, s0, -1
exit:
```

Quiz2



```
    addi s0, x0, 0
    addi s1, x0, 1
    addi t0, x0, 30
loop:
    beq  s0, t0, exit
    add  s1, s1, s1
    addi s0, s0, 1
    jal  x0, loop
exit:
```



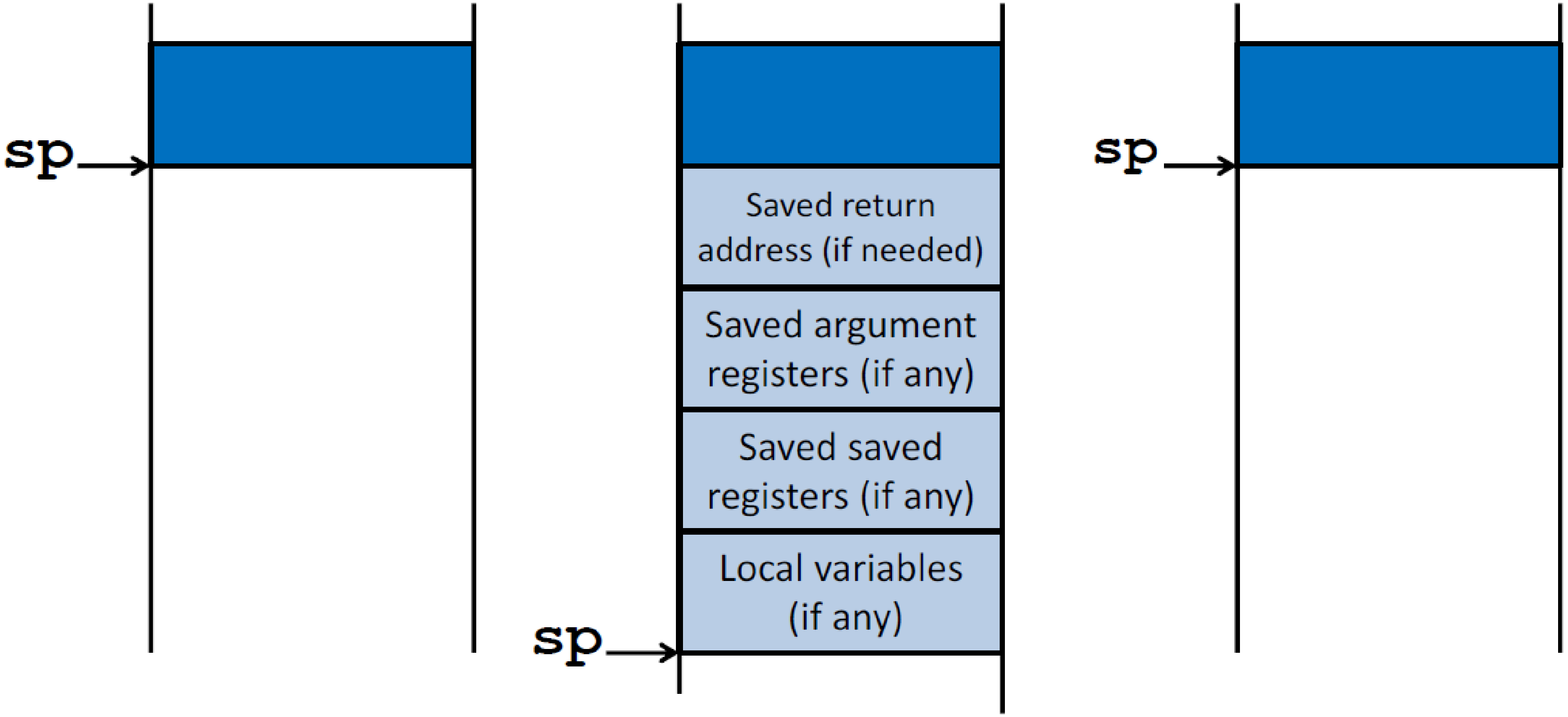
```
// computes s1 = 2^30
s1 = 1;
for(s0=0;s0<30;s++) {
    s1 *= 2;
}
```

Function Call

- Caller & Callee
  - Caller invoke callee.
  - Callee should make sure he haven’t change caller saved registers.

- Steps of function call
  - Caller put parameters into registers a0-a7.
  - Caller put next line’s address into ra and jump to the function label. (using jal)
  - Callee pushes s0-s11, sp onto stack. (attention: ra’s saver is not callee) ← Why?
  - Callee execution.
  - Callee extract value from stack.
  - Callee jump to ra’s address.

The Stack’s Condition


sp
sp
Saved return address (if needed)
Saved argument registers (if any)
Saved saved registers (if any)
Local variables (if any)
sp

Before call During call After call