Assembly (Slides T2-T3)

VE370FA22 TA Yuxuan Tang

Syntax

- Make use of “RISC-V Reference Data.pdf”
- Get familiar with each type of instruction
  R-type: `and rd, rs1, rs2`
  I-type: `ori rd, rs1, imm` , `lw rd, imm(rs1)`
  S-type: `sw r2, imm(rs1)`
  B-type: `blt rs1, rs2, TARGET_LABEL`
  U-type: `lui rd, imm`
  J-type: `jal x1, imm` , `jalr x0, 0(x1)`
- Don’t mistake the positions of rs1 and rs2

Memory Arrangement

- Byte addressable memory (remember we have instructions like `lb` , `sb`)
- 4 bytes = 1 word
- So word address is a multiple of 4 (including data and instruction memory) !!!
- Big/Little endian: Big endian is that most-significant byte is at the
  smallest address and little endian is opposite to that.
  Example of small endian arrangement data in memory: Suppose we have data 0x12345678 and 0xAABBCCDD.

	0xfffe_0004	**0xfffe_0005**	**0xfffe_0006**	**0xfffe_0007**
0xfffe_0004	78	56	34	12

	0xfffe_0008	**0xfffe_0009**	**0xfffe_000a**	**0xfffe_000b**
0xfffe_0008	DD	CC	BB	AA

- How an instruction access the data memory
  eg: `lw` , `lb` , `lbu` , `lh` , `lhu`
  e.g. Use the memory arrangement above, suppose x5 stores 0xfffe_0004,

```
lw x6, 4(x5)
lb x7, 1(x5)
```

After it is executed, x6 == 0xAABBCCDD, x7 == 0x00000056 (What if we use `lbu` here?)

Stack Usage

- Data arrangement here is like that in the data memory (4 bytes=1 word)
- We use stack pointer (`sp`) to reserve stack space, access data in stack, and release stack space
  e.g.

```
addi sp, sp, -20 # means we reserve 5-word space in the stack
sw x1, 0(sp) # we store important data into it first
sw x10, 4(sp)
sw x11, 8(sp)
...
lw x1, 0(sp) # after finishing some code execution, we load them back
lw x10, 4(sp)
lw x11, 8(sp)
...
addi, sp, sp, 20 # means we release the 5-word space
```

Function Call

- Leaf/Non-leaf function: Non-leaf function means the function that does not call (jal) other functions and leaf function is the opposite case.
- For Non-leaf function, you need to store x1 , argument registers (e.g. x10)… in the stack. For leaf function, take care of saved registers.
- Translate c program to assembly. Please go through all the examples in the course slides and your assignment, especially for recursion and loops:)
  Small exercise: Assume the function declaration for `g` is `int g(int a, int b)`

```
int f(int a, int b, int c, int d) {
return g(g(a, b), c+d);
}
```

Solution:

```
f:
addi x2, x2, -8 // Allocate stack space for 2 words
sw x1, 0(x2) // Save return address
add x5, x12, x13 // x5 = c+d
sw x5, 4(x2) // Save c+d on the stack
jal x1, g // Call x10 = g(a,b)
lw x11, 4(x2) // Reload x11= c+d from the stack
jal x1, g // Call x10 = g(g(a,b), c+d)
lw x1, 0(x2) // Restore return address
addi x2, x2, 8 // Restore stack pointer
jalr x0, 0(x1)
```

Reference

[1] VE370 FA22 Slides T2

[2] VE370 FA22 Slides T3

[3] VE370SU22_MID_RC1