

RISC-V; CALL

CS110 Discussion 5

Suting Chen

RISC-V Instructions

Different types

31	27	26	25	24	20	19	15	14	12	11	7	6	0	
funct7				rs2		rs1		funct3		rd		opcode		R-type
imm[11:0]						rs1		funct3		rd		opcode		I-type
imm[11:5]				rs2		rs1		funct3		imm[4:0]		opcode		S-type
imm[12 10:5]				rs2		rs1		funct3		imm[4:1 11]		opcode		B-type
imm[31:12]										rd		opcode		U-type
imm[20 10:1 11 19:12]										rd		opcode		J-type

RISC-V Instructions

What are those?

- func7, func3, opcode — Help CPU identify type of instruction
- rd — The place to “write to”
- rs1, rs2 — The place to “read from”
- imm — “Numbers”

RISC-V Instructions

R-type

- add, sub, xor, or, and — addition/subtract/bit ops
- sll, srl, sra — bit shiftings
- slt, sltu — comparisons

RISC-V Instructions

R-type — add, sub, xor, or, and

```
add rd rs1 rs2
sub rd rs1 rs2
xor rd rs1 rs2
or  rd rs1 rs2
and rd rs1 rs2
```


RISC-V Instructions

R-type — add, sub, xor, or, and

`t3 = t1 + t2`

`a1 = a6 - a0`

`a7 += a4`

`add t3 t1 t2`

`sub a1 a6 a0`

`add a7 a7 a4`

RISC-V Instructions

R-type — sll, srl, sra

- shift left logical
- shift right logical
- shift right arithmetical

RISC-V Instructions

R-type — What are “logical” and “arithmetical”

- Assume a 4-bit register: **0b1001**
- Shift left: **0b0010**
- Shift right: **0b0100**
- Any other possible shifts?

RISC-V Instructions

Most Significant Bit

- `0b11001010` — 1
- `0b01100011` — 0
- Often have special usages (sign bit)
- Opposed to LSB (Least Significant Bit)

RISC-V Instructions

R-type — What are “logical” and “arithmetical”

- Assume a 4-bit register: **0b1001**
- Shift left: **0b0010**
- Shift right: **0b0100**
- “Arithmetical”: **0b1100**

```
# t1 = 0xFFFFFFFF
```

```
# t2 = 4
```

```
sll t3 t1 t2
```

```
srl t4 t1 t2
```

```
sra t5 t1 t2
```


RISC-V Instructions

R-type — slt, sltu

- Set less than
- Set less than (unsigned)

RISC-V Instructions

R-type — Set less than

```
int32_t a = 0x1234;  
int32_t b = 0x0040;  
  
c = (a < b) ? 1 : 0;
```

```
# t1 = 0x1234  
# t2 = 0x0040  
  
slt t3 t1 t2
```


RISC-V Instructions

R-type — Set less than

```
int32_t a = 0xFFFFFFFF;  
int32_t b = 0x02b66240;  
  
c = (a < b) ? 1 : 0;
```

```
# t1 = 0xFFFFFFFF  
# t2 = 0x02b66240  
  
slt t3 t1 t2
```


RISC-V Instructions

R-type — Set less than (unsigned)

```
uint32_t a = 0x1234;  
uint32_t b = 0x0040;  
  
c = (a < b) ? 1 : 0;
```

```
# t1 = 0x1234  
# t2 = 0x0040  
  
sltu t3 t1 t2
```


RISC-V Instructions

R-type — Set less than (unsigned)

```
uint32_t a = 0xFFFFFFFF;  
uint32_t b = 0x02b66240;  
  
c = (a < b) ? 1 : 0;
```

```
# t1 = 0xFFFFFFFF  
# t2 = 0x02b66240  
  
sltu t3 t1 t2
```


RISC-V Instructions

R-type — Conclusion

add	ADD	R	0110011	0x0	0x00	rd = rs1 + rs2	
sub	SUB	R	0110011	0x0	0x20	rd = rs1 - rs2	
xor	XOR	R	0110011	0x4	0x00	rd = rs1 ^ rs2	
or	OR	R	0110011	0x6	0x00	rd = rs1 rs2	
and	AND	R	0110011	0x7	0x00	rd = rs1 & rs2	
sll	Shift Left Logical	R	0110011	0x1	0x00	rd = rs1 << rs2	
srl	Shift Right Logical	R	0110011	0x5	0x00	rd = rs1 >> rs2	
sra	Shift Right Arith*	R	0110011	0x5	0x20	rd = rs1 >> rs2	msb-extends
slt	Set Less Than	R	0110011	0x2	0x00	rd = (rs1 < rs2)?1:0	
sltu	Set Less Than (U)	R	0110011	0x3	0x00	rd = (rs1 < rs2)?1:0	zero-extends

RISC-V Instructions

I-type

- addi, xori, ori, andi, slli, srli, srai — Basis
- slti, sltiu — Set less than
- lb, lw, lu, lbu, lhu — Load from main memory
- jalr, ecall, ebreak

RISC-V Instructions

I-type — addi, xori, ori, andi, slli, srli, srai

```
addi rd rs1 imm  
xori rd rs1 imm  
ori  rd rs1 imm  
andi rd rs1 imm  
slli rd rs1 imm  
srli rd rs1 imm  
srai rd rs1 imm
```



```
rd = rs1 addi imm  
rd = rs1 xori imm  
rd = rs1 ori  imm  
rd = rs1 andi imm  
rd = rs1 slli imm  
rd = rs1 srli imm  
rd = rs1 srai imm
```


RISC-V Instructions

I-type — slti, sltiu

- Set less than imm
- Set less than imm (unsigned)

RISC-V Instructions

I-type — Set less than imm

```
int32_t a = 0x1234;
```

```
c = (a < 0x040) ? 1 : 0;
```

```
# t1 = 0x1234
```

```
slti t3 t1 0x040
```


RISC-V Instructions

I-type — Set less than imm

```
int32_t a = 0xFFFFFFFF;  
  
c = (a < 0x640) ? 1 : 0;
```

```
# t1 = 0xFFFFFFFF  
  
slti t3 t1 0x640
```


RISC-V Instructions

I-type — Set less than imm (unsigned)

```
uint32_t a = 0x1234;  
  
c = (a < 0x040) ? 1 : 0;
```

```
# t1 = 0x1234  
  
sltiu t3 t1 0x040
```


RISC-V Instructions

I-type — Set less than imm (unsigned)

```
uint32_t a = 0xFFFFFFFF;
```

```
c = (a < 0x640) ? 1 : 0;
```

```
# t1 = 0xFFFFFFFF
```

```
sltiu t3 t1 0x640
```


RISC-V Instructions

I-type — lb, lw, lu, lbu, lhu

```
lb  rd imm(rs1)
lw  rd imm(rs1)
lu  rd imm(rs1)
lbu rd imm(rs1)
lhu rd imm(rs1)
```


RISC-V Instructions

I-type — lb, lw, lu, lbu, lhu

load	byte		rd	imm(rs1)
load	word		rd	imm(rs1)
load		unsigned	rd	imm(rs1)
load	byte	unsigned	rd	imm(rs1)
load	halfword	unsigned	rd	imm(rs1)

RISC-V Instructions

I-type — lb, lw, lu, lbu, lhu

```
.data
number:
.word 0x01234567 0x89ABCDEF

.text
la t0 number
lw t1 0(t0)
lb t1 0(t0)
lb t1 3(t0)
lh t1 1(t0)
```


RISC-V Instructions

I-type — lb, lw, lu, lbu, lhu

```
.text
la t0 number
lw t1 0(t0)
lb t1 0(t0)
lb t1 3(t0)
lh t1 1(t0)
```

89	AB	CD	EF
01	23	45	67

RISC-V Instructions

I-type — lb, lw, lu, lbu, lhu

```
.text  
la t0 number  
lw t1 0(t0)  
lb t1 0(t0)  
lb t1 3(t0)  
lh t1 1(t0)
```

89	AB	CD	EF
01	23	45	67

0x01234567

RISC-V Instructions

I-type — lb, lw, lu, lbu, lhu

```
.text
la t0 number
lw t1 0(t0)
lb t1 0(t0)
lb t1 3(t0)
lh t1 1(t0)
```

89	AB	CD	EF
01	23	45	67

0x00000067

RISC-V Instructions

I-type — lb, lw, lu, lbu, lhu

```
.text
la t0 number
lw t1 0(t0)
lb t1 0(t0)
lb t1 3(t0)
lh t1 1(t0)
```

89	AB	CD	EF
01	23	45	67

0x00000001

RISC-V Instructions

I-type — lb, lw, lu, lbu, lhu

```
.text  
la t0 number  
lw t1 0(t0)  
lb t1 0(t0)  
lb t1 3(t0)  
lh t1 1(t0)
```

89	AB	CD	EF
01	23	45	67

0x00002345

RISC-V Instructions

I-type — lb, lw, lu, lbu, lhu (Strange cases)

```
.text  
la    t0 number  
lw    t1 3(t0)  
lhu   t1 5(t0)  
lh    t1 5(t0)  
lbu   t1 7(t0)  
lb    t1 7(t0)
```

89	AB	CD	EF
01	23	45	67

0xABCDEF01

RISC-V Instructions

I-type — lb, lw, lu, lbu, lhu (Strange cases)

```
.text
la    t0 number
lw    t1 3(t0)
lhu   t1 5(t0)
lh    t1 5(t0)
lbu   t1 7(t0)
lb    t1 7(t0)
```

89	AB	CD	EF
01	23	45	67

0x0000ABCD

RISC-V Instructions

I-type — lb, lw, lu, lbu, lhu (Strange cases)

```
.text
la    t0 number
lw    t1 3(t0)
lhu   t1 5(t0)
lh    t1 5(t0)
lbu   t1 7(t0)
lb    t1 7(t0)
```

89	AB	CD	EF
01	23	45	67

0xFFFFABCD

RISC-V Instructions

I-type — lb, lw, lu, lbu, lhu (Strange cases)

```
.text
la    t0 number
lw    t1 3(t0)
lhu   t1 5(t0)
lh    t1 5(t0)
lbu   t1 7(t0)
lb    t1 7(t0)
```

89	AB	CD	EF
01	23	45	67

0x00000089

RISC-V Instructions

I-type — lb, lw, lu, lbu, lhu (Strange cases)

```
.text
la    t0 number
lw    t1 3(t0)
lhu   t1 5(t0)
lh    t1 5(t0)
lbu   t1 7(t0)
lb    t1 7(t0)
```

89	AB	CD	EF
01	23	45	67

0xFFFFFFFF89

RISC-V Instructions

I-type — Conclusion

addi	ADD Immediate	I	0010011	0x0		rd = rs1 + imm	
xori	XOR Immediate	I	0010011	0x4		rd = rs1 ^ imm	
ori	OR Immediate	I	0010011	0x6		rd = rs1 imm	
andi	AND Immediate	I	0010011	0x7		rd = rs1 & imm	
slli	Shift Left Logical Imm	I	0010011	0x1	imm[5:11]=0x00	rd = rs1 << imm[0:4]	
srli	Shift Right Logical Imm	I	0010011	0x5	imm[5:11]=0x00	rd = rs1 >> imm[0:4]	
srai	Shift Right Arith Imm	I	0010011	0x5	imm[5:11]=0x20	rd = rs1 >> imm[0:4]	msb-extends
slti	Set Less Than Imm	I	0010011	0x2		rd = (rs1 < imm)?1:0	
sltiu	Set Less Than Imm (U)	I	0010011	0x3		rd = (rs1 < imm)?1:0	zero-extends
lb	Load Byte	I	0000011	0x0		rd = M[rs1+imm][0:7]	
lh	Load Half	I	0000011	0x1		rd = M[rs1+imm][0:15]	
lw	Load Word	I	0000011	0x2		rd = M[rs1+imm][0:31]	
lbu	Load Byte (U)	I	0000011	0x4		rd = M[rs1+imm][0:7]	zero-extends
lhu	Load Half (U)	I	0000011	0x5		rd = M[rs1+imm][0:15]	zero-extends

RISC-V Instructions

S-type

- sb, sh, sw — Write to main memory

RISC-V Instructions

S-type

```
sb rs2 imm(rs1)  
sh rs2 imm(rs1)  
sw rs2 imm(rs1)
```


RISC-V Instructions

S-type

```
store    byte      rs2 imm(rs1)
store    halfword   rs2 imm(rs1)
store    word       rs2 imm(rs1)
```


RISC-V Instructions

S-type

89	AB	CD	EF
01	23	45	67

RISC-V Instructions

S-type

```
.text
la t0 number
li t1 0x91
sw t1 0(t0)
sw t1 3(t0)
sb t1 2(t0)
sh t1 6(t0)
```

89	AB	CD	EF
00	00	00	91

RISC-V Instructions

S-type

```
.text  
la t0 number  
li t1 0x91  
sw t1 0(t0)  
sw t1 3(t0)  
sb t1 2(t0)  
sh t1 6(t0)
```

89	00	00	00
91	00	00	91

RISC-V Instructions

S-type

```
.text  
la t0 number  
li t1 0x91  
sw t1 0(t0)  
sw t1 3(t0)  
sb t1 2(t0)  
sh t1 6(t0)
```

89	00	00	00
91	91	00	91

RISC-V Instructions

S-type

```
.text  
la t0 number  
li t1 0x91  
sw t1 0(t0)  
sw t1 3(t0)  
sb t1 2(t0)  
sh t1 6(t0)
```

00	91	00	00
91	91	00	91

RISC-V Instructions

S-type

sb	Store Byte	S	0100011	0x0		M[rs1+imm][0:7] = rs2[0:7]	
sh	Store Half	S	0100011	0x1		M[rs1+imm][0:15] = rs2[0:15]	
sw	Store Word	S	0100011	0x2		M[rs1+imm][0:31] = rs2[0:31]	

RISC-V Instructions

U-type

- lui — Load Upper Immediate
- auipc — Add Upper Immediate to PC

RISC-V Instructions

U-type

```
lui t0 0x3ab85
```

```
t0 = 0x3ab85000
```

```
auipc t0 0x3
```

```
rd = PC + imm << 12
```

```
t0 = PC + 0x3 << 12
```


RISC-V Instructions

U-type

lui	Load Upper Imm	U	0110111			rd = imm << 12	
auipc	Add Upper Imm to PC	U	0010111			rd = PC + (imm << 12)	

RISC-V Instructions

B-type

- beq, bne, blt, bge, bltu, bgeu
- How label becomes immediates?

RISC-V Instructions

B-type — beq, bne, blt, bge, bltu, bgeu

```
beq  rs1 rs2 imm  
bne  rs1 rs2 imm  
blt  rs1 rs2 imm  
bge  rs1 rs2 imm  
bltu rs1 rs2 imm  
bgeu rs1 rs2 imm
```



```
if (rs1 beq  rs2) goto label;  
if (rs1 bne  rs2) goto label;  
if (rs1 blt  rs2) goto label;  
if (rs1 bge  rs2) goto label;  
if (rs1 bltu rs2) goto label;  
if (rs1 bgeu rs2) goto label;
```


RISC-V Instructions

B-type

beq	Branch ==	B	1100011	0x0		if(rs1 == rs2) PC += imm	
bne	Branch !=	B	1100011	0x1		if(rs1 != rs2) PC += imm	
blt	Branch <	B	1100011	0x4		if(rs1 < rs2) PC += imm	
bge	Branch ≥	B	1100011	0x5		if(rs1 ≥ rs2) PC += imm	
bltu	Branch < (U)	B	1100011	0x6		if(rs1 < rs2) PC += imm	zero-extends
bgeu	Branch ≥ (U)	B	1100011	0x7		if(rs1 ≥ rs2) PC += imm	zero-extends

How labels become immediates?

RISC-V Instructions

How labels become immediates?

- Calculate the offset

```
0 nop
1 blt x0 x0 label_1
2 nop
  label_1:
3 nop
```



```
0x0 addi x0 x0 0
0x4 blt x0 x0 8
0x8 addi x0 x0 0
  label_1:
0xC addi x0 x0 0
```


RISC-V Instructions

How labels become immediates?

```
0x0 addi x0 x0 0
0x4 blt  x0 x0 8
0x8 addi x0 x0 0
    label_1:
0xC addi x0 x0 0
```



```
0x0 0x00000013
0x4 0x00004463
0x8 0x00000013
0xC 0x00000013
```


RISC-V Instructions

How labels become immediates?

- Wait ... What is the imm field?

```
0x0 addi x0 x0 0
0x4 blt  x0 x0 8
0x8 addi x0 x0 0
      label_1:
0xC addi x0 x0 0
```



```
0x0 0x00000013
0x4 0x00004463
0x8 0x00000013
0xC 0x00000013
```

- Extract immediate field -> 4

RISC-V Instructions

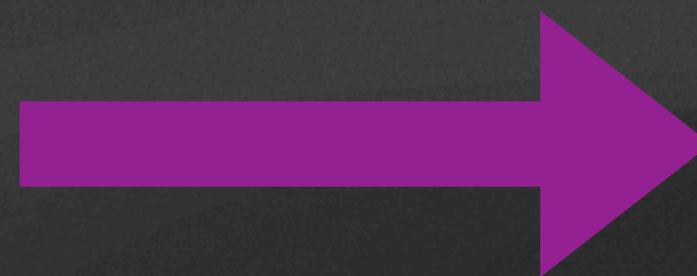
Jal, Jalr

- Jump And Link
- Jump And Link Register

RISC-V Instructions

Jump And Link

```
0 nop
1 jal ra label_1
2 nop
3 nop
      label_1:
4 nop
```



```
0x0  addi x0 x0 0
0x4  jal  x1 12
0x8  addi x0 x0 0
0xC  addi x0 x0 0
      label_1:
0x10 addi x0 x0 0
```


RISC-V Instructions

Jump And Link

- The same thing about imm field happened again

```
0x0  addi x0 x0 0
0x4  jal  x1 12
0x8  addi x0 x0 0
0xC  addi x0 x0 0
      label_1:
0x10 addi x0 x0 0
```



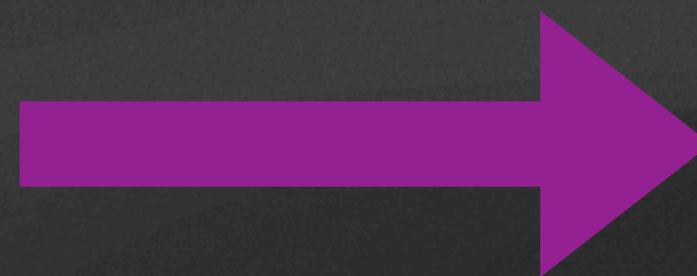
```
0x0  0x00000013
0x4  0x00C000EF
0x8  0x00000013
0xC  0x00000013
0x10 0x00000013
```

- Extract immediate field -> 6

RISC-V Instructions

Jump And Link Register

```
0  la t0 label_2
1  jalr ra t0 4
2  nop
3  nop
   label_2:
4  nop
5  nop
```



```
# load address
0x8  jalr x1 x5 4
0xC  addi x0 x0 0
0x10 addi x0 x0 0
      label_2:
0x14 addi x0 x0 0
```


RISC-V Instructions

Jump And Link Register

- This time we have the corresponding offset

```
# load address
0x8  jalr x1 x5 4
0xC  addi x0 x0 0
0x10 addi x0 x0 0
      label_2:
0x14 addi x0 x0 0
```



```
0x8  0x004280E7
0xC  0x00000013
0x10 0x00000013
0x14 0x00000013
```

- Extract immediate field -> 4

Project 1.1

Get started early!

Recommended Readings

- Why JALR encodes the LSB?
- <https://github.com/jameslzhou/riscv-card>

CALL review

What are those?

- C: Compiler
 - A: Assembler
 - L: Linker
 - L: Loader
-
- These parts have a demo, so come to the discussion if you need that.

Compiler — Before compiling

The C PreProcessor (cpp)

- Deal with directives — Starting with “#” in C
- “Pull” the declarations from #include
- “Expand/Replace” the macros #define

~ `cpp main.c main.i`

Compiler

GNU C compiler (gcc)

- Compile the source code to assembly

```
~ gcc -S main.i
```


Assembler

- Converts assembly level language code into machine language code.

~ `as main.s -o main.o`

~ `hexdump -C main.o`

~ `readelf -a main.o`

~ `objdump -r main.o`

Linker, Loader

- Combines the object files, generated by the assembler to generate an executable.
- Load executable to main memory.

Relocation table

- Information about addresses referenced in this object file that the linker must adjust once it knows the final memory allocation.

Symbol table

- Name and current location of variables or functions that can potentially be referenced in other object files.

What problem should linkers solve?

- Assembler doesn't know the addresses of external objects.
 - Puts zeroes in the object file for each unknown address
- Assembler doesn't know where the things it's assembling will go in memory
 - Assume that things start at address zero, leave for linker re-arrange.

Question

When does the instruction is finally determined?

1. `add x6, x7, x8`

2. `jal x1, fprintf`

A. After compile

B. After assemble

C. After link

D. After load