CSC 411

Computer Organization (Spring 2024)
Lecture 15: Representing instructions

Prof. Marco Alvarez, University of Rhode Island

Feedback

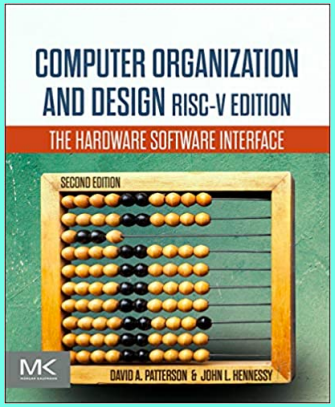
- Add instructor office hours
- Solutions for in-class exercises

Disclaimer

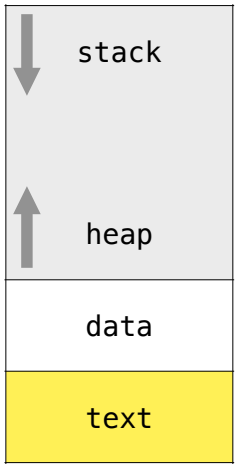
Some figures and slides are adapted from:

Computer Organization and Design (Patterson and Hennessy)

The Hardware/Software Interface



Context



High-level language program (in C)

```
swap(size_t v[], size_t k)
{
   size_t temp;
   temp = v[k];
   v[k] = v[k+1];
   v[k+1] = temp;
}
```

Compiler

Assembly language program (for RISC-V)

```
swap:
      slli x6, x11, 3
      add  x6, x10, x6
      lw   x5, 0(x6)
      lw   x7, 4(x6)
      sw   x7, 0(x6)
      sw   x5, 4(x6)
      jalr x0, 0(x1)
```

Assembler

Binary machine language program (for RISC-V)

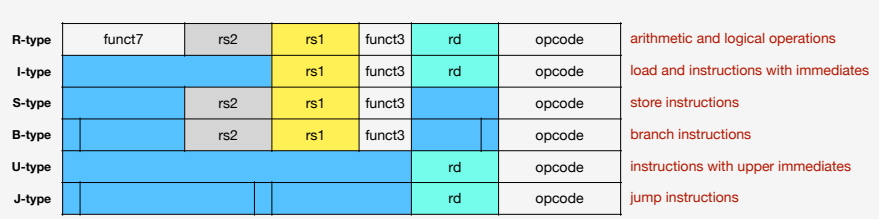
```
00000000001101011001001100010011
00000000011001010000001100110011
00000000000000011001100101010000011
00000000100000110011001110000011
00000000011100110011000000100011
00000000101001100110100001000011
00000000000000001000000001100111
```

Representing instructions

- Instructions are encoded in binary
  - called machine code
- RISC-V instructions
  - encoded as 32-bit instruction words
  - small number of formats
    - encoding operation code (opcode), register numbers, …
  - regularity !
    - keep formats as similar as possible => simpler hardware

Instructions formats

- RV32I includes **47 instructions**
  - must be aligned on a 4-byte boundary in memory
- There are 4 core instruction formats (R/I/S/U)
  - and two immediate encoding variants (B/J)
- The opcode, source register operands (`rs1`, `rs2`), and destination register operand (`rd`) are always kept at the same position
  - simplifies decoding (performance critical)



Base instruction formats

R-type instructions

- Arithmetic/logical instructions
  - 3 registers and no immediate values
  - sets of similar instructions get the same opcode
    - funct3 and funct7 are identifiers used to differentiate instructions with the same opcode

funct7	rs2	rs1	funct3	rd	opcode
7 bits	5 bits	5 bits	3 bits	5 bits	7 bits

Encoding

all instructions have an **opcode** field

not all instructions use the **funct3** or **funct7** fields

Format	Instruction	Opcode	Funct3	Funct6/7
R-type	add	0110011	000	0000000
	sub	0110011	000	0100000
	sll	0110011	001	0000000
	xor	0110011	100	0000000
	srl	0110011	101	0000000
	sra	0110011	101	0000000
	or	0110011	110	0000000
	and	0110011	111	0000000
	lr.d	0110011	011	0001000
	sc.d	0110011	011	0001100
I-type	lb	0000011	000	n.a.
	lh	0000011	001	n.a.
	lw	0000011	010	n.a.
	lbu	0000011	100	n.a.
	lhu	0000011	101	n.a.
	addi	0010011	000	n.a.
	slli	0010011	001	000000
	xori	0010011	100	n.a.
	srli	0010011	101	000000
	srai	0010011	101	010000
	ori	0010011	110	n.a.
	andi	0010011	111	n.a.
	jalr	1100111	000	n.a.
S-type	sb	0100011	000	n.a.
	sh	0100011	001	n.a.
	sw	0100011	010	n.a.
SB-type	beq	1100111	000	n.a.
	bne	1100111	001	n.a.
	blt	1100111	100	n.a.
	bge	1100111	101	n.a.
	bltu	1100111	110	n.a.
	bgeu	1100111	111	n.a.
U-type	lui	0110111	n.a.	n.a.
UJ-type	jal	1101111	n.a.	n.a.

Practice

rd	s1	x9	01001
rs1	t0	x5	00101
rs2	t1	x6	00110
opcode			0110011
funct3			000
funct7			0000000

```
add s1, t0, t1
```

0x006284B3

R-type 0000000 00110 00101 000 01001 0110011

0000	0000	0110	0010	1000	0100	1011	0011
0	0	6	2	8	4	B	3

Practice

rd	a6	x16	10000
rs1	a5	x15	01111
rs2	t0	x5	00101
opcode			0110011
funct3			000
funct7			0000000

```
add x16, x15, x5
```

0x00578833

R-type 0000000 00101 01111 000 10000 0110011

0000	0000	0101	0111	1000	1000	0011	0011
0	0	5	7	8	8	3	3

I-type instructions

- Arithmetic instructions with immediates and load instructions
  - 2 registers and 1 immediate
  - store instructions use a separate format (need rs2)
- Immediates
  - sign-extended to 32 bits, however with limited range (only using 12 bits)

imm [11:0]	rs1	funct3	rd	opcode
12 bits	5 bits	3 bits	5 bits	7 bits

- shift instructions use a modified I-type with a funct7 field, they only need 5 bits in the immediate — can't shift more than 31 bits

funct7	imm [4:0]	rs1	funct3	rd	opcode
7 bits	5 bits	5 bits	3 bits	5 bits	7 bits

Immediates

- Immediates are always sign-extended
  - except for the 5-bit immediates used in CSR (control and status register) instructions
  - the sign bit for all immediates is always in bit 31 of the instruction to speed sign-extension

Practice

- Draw arrows from instructions to format

```
addi x6, x5, -3
```

```
lw x8, 20(x4)
```

imm [11:0]	rs1	funct3	rd	opcode
12 bits	5 bits	3 bits	5 bits	7 bits

```
slli x2, x2, 2
```

funct7	imm [4:0]	rs1	funct3	rd	opcode
7 bits	5 bits	5 bits	3 bits	5 bits	7 bits

S-type instructions

- Store instructions
  - base register in `rs1` and offset in `imm`
  - destination register not used
- Immediates
  - immediate split into two fields
  - rs1 and rs2 are kept in the same locations

imm [11:5]	rs2	rs1	funct3	imm [4:0]	opcode
7 bits	5 bits	5 bits	3 bits	5 bits	7 bits

Practice

- Encode the following instruction
  - opcode: 0100011
  - funct3: 010

```
sw x8, 20(x4)
```

imm [11:5]	rs2	rs1	funct3	imm [4:0]	opcode
7 bits	5 bits	5 bits	3 bits	5 bits	7 bits

U-type instructions

- Load upper immediate (`lui`)
  - sets `rd` to `imm << 12`
- Add upper immediate to PC (`auipc`)
  - sets `rd` to `(imm << 12) + PC`
- Corner case
  - bit 11 is 1 (sign extends)
  - better use `li` instead

```
lui t0, 0x12345
addi t0, t0, 0xFFF
# this is not 0x12345FFF
```

imm [31:12]	rd	opcode
20 bits	5 bits	7 bits

Practice

- Encode the following instruction
  - opcode: 0110111

```
lui t0, 0x12345
```

imm [31:12]	rd	opcode
20 bits	5 bits	7 bits

Practice

- Run this code using a simulator
  - understand pseudo-instructions
  - understand PC and `auipc`

```
nop
li x1, 10
nop
li x2, 0xDEADBEEF
nop
auipc x1, 0xFFFFF
nop
```

Control instructions

Labels

- When translating RISC-V to binary, labels are converted into explicit values
  - use PC-relative addressing to convert labels into offsets

```
    addi t0, x0, 0
    addi t1, x0, 5
loop:
    bge t0, t1, break1   # +6 instructions, offset = 24 bytes
    li  t2, 3            # pseudo instruction, involves 1 instruction if
                         # constant is small, 2 if constant is large
    beq t0, t2, break2   # +6 instructions, offset = 24 bytes
    addi t0, t0, 1
    addi t1, t1, -1
    j loop               # -5 instructions, offset = -20 bytes
break1:
    addi t0, x0, 1
    j done               # 2 instructions, offset = 8 bytes
break2:
    addi t0, x0, 0
done:
    sw t0, 0(s2)
```

From venus.cs61c.org

PC	Machine Code	Basic Code	Original Code
0x0	0x00000293	addi x5 x0 0	addi t0, x0, 0
0x4	0x00500313	addi x6 x0 5	addi t1, x0, 5
0x8	0x0062DC63	bge x5 x6 24	bge t0, t1, break1
0xc	0x00300393	addi x7 x0 3	li t2, 3
0x10	0x00728C63	beq x5 x7 24	beq t0, t2, break2
0x14	0x00128293	addi x5 x5 1	addi t0, t0, 1
0x18	0xFFF30313	addi x6 x6 -1	addi t1, t1, -1
0x1c	0xFEDFF06F	jal x0 -20	j loop
0x20	0x00100293	addi x5 x0 1	addi t0, x0, 1
0x24	0x0080006F	jal x0 8	j done
0x28	0x00000293	addi x5 x0 0	addi t0, x0, 0
0x2c	0x00592023	sw x5 0(x18)	sw t0, 0(s2)

Note all offsets are multiples of 4 (32-bit instructions). We wouldn't need to encode the last 2 bits. However some RISC-V extensions use 16-bit instructions. Therefore, the **lowest bit of an offset is not encoded.**

B-type instructions

- Used for branch instructions
  - format similar to S-type
  - use 2 source registers and an immediate
- Range for branch instructions is [-4096, 4094]
  - 13 bits are used (bit 0 is not encoded)
  - about +/- 1024 instructions
    - if necessary, greater branches can be achieved by inverting the branch condition and using a `j` instruction
- Target address uses PC relative addressing
  - target is (PC + imm << 1)

imm [12|10:5]	rs2	rs1	funct3	imm [4:1|11]	opcode
7 bits	5 bits	5 bits	3 bits	5 bits	7 bits

Practice

- Encode the following instruction
  - assume label done is at offset 24
  - opcode: 1100111
  - funct3: 101

```
bge x5, x6, done
```

imm [12|10:5]	rs2	rs1	funct3	imm [4:1|11]	opcode
7 bits	5 bits	5 bits	3 bits	5 bits	7 bits

J-type instructions

- Used for `jal`
  - format similar to U-type
  - use 1 destination register and an immediate
- Range for `jal` instructions is [-1048576, 1048574]
  - 21 bits are used (bit 0 is not encoded)
  - about +/- 262144 instructions
    - if necessary, greater jumps can be achieved by combining `lui` or `auipc` with `jalr` instructions

imm [20|10:1|11|19:12]	rd	opcode
20 bits	5 bits	7 bits

Practice

- Encode the following instruction
  - `j loop` is a pseudoinstruction for `jal x0, loop`
  - assume label loop is at offset -20
  - opcode: 1101111

```
j loop
```

imm [20|10:1|11|19:12]	rd	opcode
20 bits	5 bits	7 bits

Summary of RISC-V instructions

RISC-V Instructions	Name	Format
Add	add	R
Subtract	sub	R
Add immediate	addi	I
Load word	lw	I
Load word, unsigned	lwu	I
Store word	sw	S
Load halfword	lh	I
Load halfword, unsigned	lhu	I
Store halfword	sh	S
Load byte	lb	I
Load byte, unsigned	lbu	I
Store byte	sb	S
Load reserved	lr.d	R
Store conditional	sc.d	R
Load upper immediate	lui	U
And	and	R
Inclusive or	or	R
Exclusive or	xor	R
And immediate	andi	I
Inclusive or immediate	ori	I
Exclusive or immediate	xori	I
Shift left logical	sll	R
Shift right logical	srl	R
Shift right arithmetic	sra	R
Shift left logical immediate	slli	I
Shift right logical immediate	srli	I
Shift right arithmetic immediate	srai	I
Branch if equal	beq	SB
Branch if not equal	bne	SB
Branch if less than	blt	SB
Branch if greater or equal	bge	SB
Branch if less, unsigned	bltu	SB
Branch if greatr/eq, unsigned	bgeu	SB
Jump and link	jal	UJ
Jump and link register	jalr	I

Additional Instructions in RISC-V Base Architecture

Instruction	Name	Format	Description
Add upper immediate to PC	auipc	U	Add 20-bit upper immediate to PC; write sum to register
Set if less than	slt	R	Compare registers; write Boolean result to register
Set if less than, unsigned	sltu	R	Compare registers; write Boolean result to register
Set if less than, immediate	slti	I	Compare registers; write Boolean result to register
Set if less than immediate, unsigned	sltiu	I	Compare registers; write Boolean result to register