CSC 411

Computer Organization (Fall 2024)
Lecture 15: Representing instructions

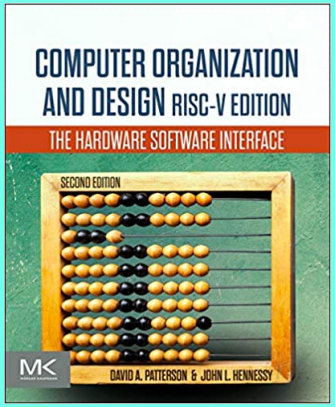
Prof. Marco Alvarez, University of Rhode Island

Disclaimer

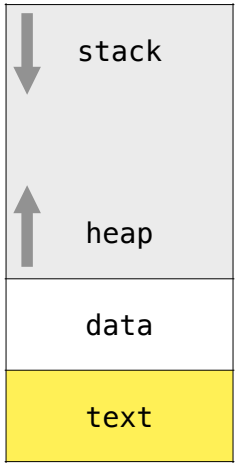
Some figures and slides are adapted from:

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)

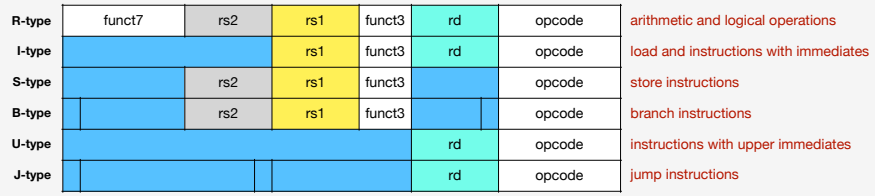
```
00000000001101011001001100010011
00000000011001010000001100110011
00000000000000011001100101000011
00000000100000110011001110000011
00000000011100110011000000100011
00000000101001100110100001000011
00000000000000001000000001100111
```

Instruction representation

- Instructions are encoded in binary
  - machine code
- RISC-V instructions
  - encoded as 32-bit instruction words (RV32I)
  - must be aligned in 4-byte boundaries in memory
- Encoding principles
  - small number of formats for simplicity
  - regularity: keep formats as similar as possible
    - simplifies hardware design and decoding
  - all instructions have an opcode field
  - some instructions use additional funct3 or funct7 fields

Instructions formats

- Core instruction formats
  - R-type, I-type, S-type, U-type
- Immediate encoding variants
  - B-type, J-type
- Key design features
  - opcode, rs1, rs2, rd are always in the same position
  - RV32I includes 47 instructions



Base instruction formats

R-type instructions

- Characteristics
  - used for register-to-register arithmetic/logical instructions
  - three registers operands (rs1, rs2, rd)
  - 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

RV32I Base Integer Instructions

Inst	Name	FMT	Opcode	funct3	funct7	Description (C)	Note
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
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
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]	
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
jal	Jump And Link	J	1101111			rd = PC+4; PC += imm	
jalr	Jump And Link Reg	I	1100111	0x0		rd = PC+4; PC = rs1 + imm	
lui	Load Upper Imm	U	0110111			rd = imm << 12	
auipc	Add Upper Imm to PC	U	0010111			rd = PC + (imm << 12)	
ecall	Environment Call	I	1110011	0x0	imm=0x0	Transfer control to OS	
ebreak	Environment Break	I	1110011	0x0	imm=0x1	Transfer control to debugger	

Practice

rd	s1	x9	01001
rs1	t0	x5	00101
rs2	t1	x6	00110
opcode			0110011
funct3			001
funct7			0000000

```
sll s1, t0, t1
```

0x006294B3

R-type 0000000 00110 00101 001 01001 0110011

0000 0000 0110 0010 1001 0100 1011 0011

0 0 6 2 9 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

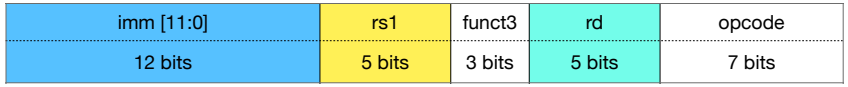
R-type 0000000 00101 01111 000 10000 0110011

0000 0000 0101 0111 1000 1000 0011 0011

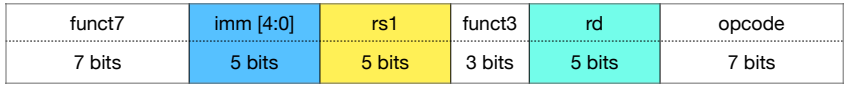
0 0 5 7 8 8 3 3

I-type instructions

- Characteristics
  - used for immediate arithmetic and load instructions
  - two registers (rs1, rd) and 1 12-bit immediate
- Immediates
  - sign-extended to 32 bits, however with limited range (only using 12 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

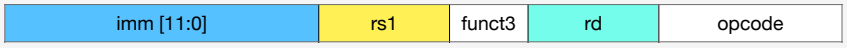


Practice

- Draw arrows from instructions to format

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

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




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

S-type instructions

- Characteristics
  - used for store instructions
  - two source registers (rs1, rs2) and a 12-bit immediate
- Immediates
  - immediate split into two fields

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

U-type instructions

- Characteristics
  - used for upper immediate instructions (lui, auipc)
  - one destination register (rd) and a 20-bit immediate
- Load upper immediate (`lui`)
  - sets `rd` to `imm << 12`
- Add upper immediate to PC (`auipc`)
  - sets `rd` to `(imm << 12) + PC`

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

Practice

- Encode the following instruction
  - opcode: 0110111

```
lui x1, 0x12345
```

imm [31:12]	rd	opcode

Dealing with large immediates

- I-type instructions are limited to 12-bit immediates
- `lui` and `addi` can be used together to load a 32-bit constant into a register
  - `lui` loads 20 bits into the upper 20 bits of the destination register (clears the lower 12 bits)
  - `addi` adds the sign-extended lower 12 bits of the constant to the same register

```
lui x5, 0x12345
addi x5, x5, 238 # 0x0EE
# loads 0x123450EE
```

```
lui x5, 0x12345
addi x5, x5, -274 # 0xEEE
# loads 0x12344EEE
```

Corner case: addi sign-extends the lower 12 bits, which means if the most significant bit is 1, the upper 20 bits are filled with 1s during the addition. Better use the li pseudoinstruction to handle all of this.

Practice

- Run this code using a simulator
  - understand the `li` pseudoinstruction
  - understand `auipc` and PC

```
nop
li x1, 10
nop
li x2, 0xABCD0123
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

- Characteristics
  - used for conditional branches
  - two source registers and a 13-bit 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: 1100011
  - funct3: 101

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

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

J-type instructions

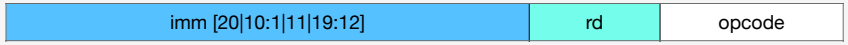
- Characteristics
  - used for unconditional jumps (jal)
  - one destination register and a 21-bit immediate
- Range for `jal` instructions is [-1048576, 1048574]
  - 21 bits are used (bit 0 is not encoded)
  - about +/- 262144 instructions
    - if necessary, greater jumps can be achieved by combining `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`



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