信息科学与技术学院
School of Information Science and Technology

CS 110
Computer Architecture
RISC-V III

Instructors:

Siting Liu & Yuan Xiao

Course website: https://faculty.sist.shanghaitech.edu.cn/liust/courses/CS110.html

School of Information Science and Technology (SIST)

ShanghaiTech University

2026/3/24

Administrative

- HW 2, due today! HW 3 to be release this Thur.
- Lab 3 check this week; lab 4 released, to check next week
- Proj.1.1 released! Start early! DDL Apr. 7th!
- Discussion this week on RISC-V assembly by TA Chenyang Mao at SPST 4-122, 18:00-19:50; also covers some of the (RISC-V) questions from previous exams;

Outline

- Encoding of RISC-V instructions
  - R-type
  - I-type arithmetic and logic
  - I-type load
  - S-type store
  - B-type
  - J-type
  - U-type
- CALL (compiler, assembler, linker & loader)

Where are we?

High Level Language Program (e.g., C)

temp = v[k];
v[k] = v[k+1];
v[k+1] = temp;

Compiler

Assembly Language Program (e.g., RISC-V)

```
lw   t0, 0(s2)
lw   t1, 4(s2)
sw   t1, 0(s2)
sw   t0, 4(s2)
```

Assembler

We are here!

Machine Language Program (RISC-V)

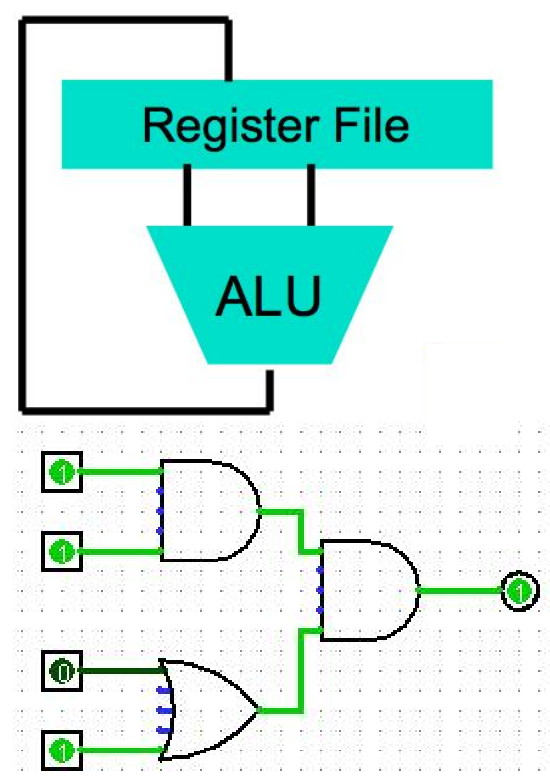
0000 1001 1100 0110 1010 1111 0101 1000
1010 1111 0101 1000 0000 1001 1100 0110
1100 0110 1010 1111 0101 1000 0000 1001
0101 1000 0000 1001 1100 0110 1010 1111

Machine Interpretation

Hardware Architecture Description (e.g., block diagrams)

Architecture Implementation

Logic Circuit Description (Circuit Schematic Diagrams)



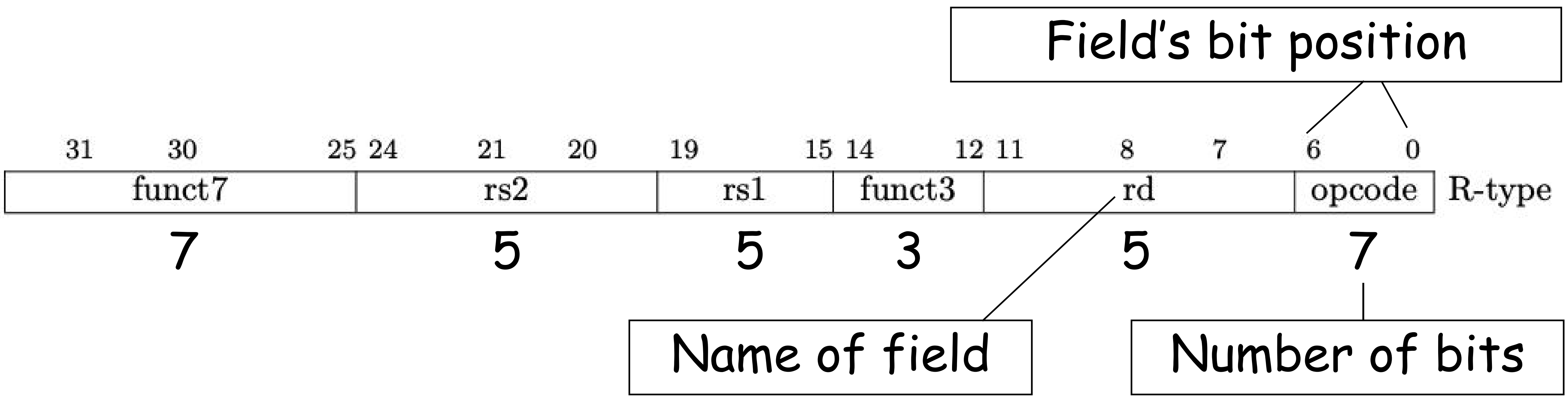
RV32I Instruction Encoding

~ High Bit ***~ Low Bit***

31	30 … 25	24 … 21	20	19 … 15	14 … 12	11 … 8	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]	imm[10:5]	rs2		rs1	funct3	imm[4:1]	imm[11]	opcode	B-type
imm[31:12]						rd		opcode	U-type
imm[20]	imm[10:1]		imm[11]	imm[19:12]		rd		opcode	J-type

- All 32-bit in length
- Generally, fields are aligned if present (rs1, rs2, rd, funct3, funct7, opcode)
- Different number/type of operands/result

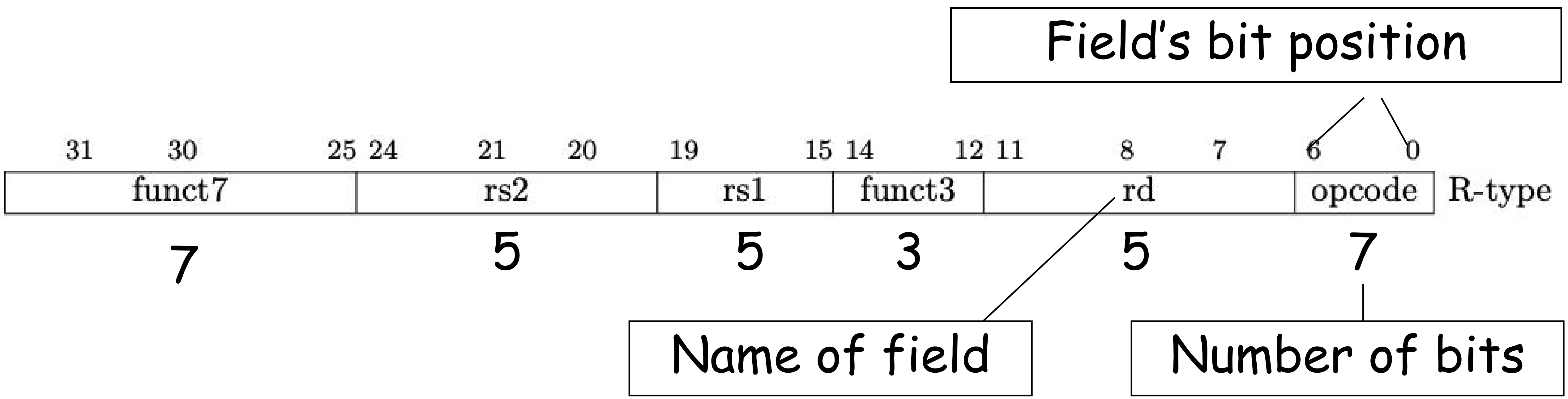
R-type Encoding



Assembly: *Operation rd, rs1, rs2*

rd,rs1,rs2 unsigned numbers, represent the numbers of the regs.

R-type Encoding (Cont'd)



Assembly: Operation rd, rs1, rs2

funct7/funct3/opcode fields:

- opcode: 0b0110011 for RV32I R-format arithmetic/logic operations
- funct7/funct3 together decide the type of operation

R-type Example

31 … 25	24 … 20	19 … 15	14 … 12	11 … 7	6 … 0	
funct7	rs2	rs1	funct3	rd	opcode	R-type

Assembly: add x2,x0,x1

0000000	00001	00000	000	00010	0110011

Look up the green card

See the bottom part of course homepage
Resources: RISC-V Green Card: pdf

Machine code: concatenate all fields

0000_0000_0001_0000_0000_0001_0011_0011

0x00100133

R-type—All Instructions

Assembly: Operation rd,rs1,rs2

0000000	rs2	rs1	000	rd	0110011	ADD
0100000	rs2	rs1	000	rd	0110011	SUB
0000000	rs2	rs1	001	rd	0110011	SLL
0000000	rs2	rs1	010	rd	0110011	SLT
0000000	rs2	rs1	011	rd	0110011	SLTU
0000000	rs2	rs1	100	rd	0110011	XOR
0000000	rs2	rs1	101	rd	0110011	SRL
0100000	rs2	rs1	101	rd	0110011	SRA
0000000	rs2	rs1	110	rd	0110011	OR
0000000	rs2	rs1	111	rd	0110011	AND

funct7/funct3 together decide the operation

I-type Arithmetic & Logic

31 30 25	24 21 20	19 15	14 12	11 8 7	6 0	
funct7	rs2	rs1	funct3	rd	opcode	R-type
7	5	5	3	5	7	
imm[11:0]		rs1	funct3	rd	opcode	I-type
12						

Assembly: *Operation rd,rs1,imm*

Register-immediate type

- *imm*: 12 bits, hold values for [-2048,2047]
- *imm* sign-extended before operations (sign-extension done in hardware)
- Opcode 0b0010011 for I-type arithmetic/logic operations

I-type Example I

12	5	3	5	7	
imm[11:0]	rs1	funct3	rd	opcode	I-type

Assembly: addi x2,x0,1234

010011010010	00000	000	00010	0010011

Look up the green card

Machine code: concatenate all fields

0100_1101_0010_0000_0000_0001_0001_0011

0x4d200113

I-type Example II

12	5	3	5	7	
imm[11:0]	rs1	funct3	rd	opcode	I-type

Assembly: addi x2, x0, -1234

101100101110	00000	000	00010	0010011

2’s complement

Look up the green card

Machine code: concatenate all fields

1011_0010_1110_0000_0000_0001_0001_0011

0xb2e00113

I-type Shift

imm[11:0]	rs1	funct3	rd	opcode	I-type
12	5	3	5	7	

0000000	shamt[4:0]	src	001	dest	0010011	SLLI
0000000	shamt[4:0]	src	101	dest	0010011	SRLI
0100000	shamt[4:0]	src	101	dest	0010011	SRAI

Register-immediate type

- imm: 12 bits, hold values for [-2048,2047]

Not for shift operations

- shamt not sign-extended before operations for shifts
- Opcode 0b0010011 for I-type arithmetic/logic operations
- Shift is specialized, since shift more than 31 bits is meaningless

I-type Arithmetic & Logic

imm[11:0]	rs1	funct3	rd	opcode	I-type
12	5	3	5	7	

imm	src	000	dest	0010011	ADDI
imm	src	010	dest	0010011	SLTI
imm	src	011	dest	0010011	SLTIU
imm	src	100	dest	0010011	XORI
imm	src	110	dest	0010011	ORI
imm	src	111	dest	0010011	ANDI

0000000	shamt[4:0]	src	001	dest	0010011	SLLI
0000000	shamt[4:0]	src	101	dest	0010011	SRLI
0100000	shamt[4:0]	src	101	dest	0010011	SRAI

Same as corresponding R-type funct3

I-type Load

imm[11:0]	rs1	funct3	rd	opcode	I-type
12	5	3	5	7	

Assembly: lw/lhu/lh/lb/lbu rd, (imm)rs1

- Opcode: 0b0000011 for RV32I I-format load operations
- funct3:
  - First bit indicates signed(0)/unsigned(1)
  - Last 2 bits indicates w(10)/h(01)/b(00)

I-type Load

imm[11:0]	rs1	funct3	rd	opcode	I-type
12	5	3	5	7	

Assembly: lw/lhu/lh/lb/lbu rd, imm(rs1)

imm	src	000	dest	0000011	LB
imm	src	001	dest	0000011	LH
imm	src	010	dest	0000011	LW
imm	src	100	dest	0000011	LBU
imm	src	101	dest	0000011	BHU

2’s complement

I-type Load Example

imm[11:0]	rs1	funct3	rd	opcode	I-type
12	5	3	5	7	

Assembly: lw/lhu/lh/lb/lbu rd,imm(rs1)

imm	src	000	dest	0000011	LB
imm	src	001	dest	0000011	LH
imm	src	010	dest	0000011	LW
imm	src	100	dest	0000011	LBU
imm	src	101	dest	0000011	BHU

2's complement

Assembly: lbu x18, -1(x17)

FFF	10001	100	10010	0000011

Machine code

1111_1111_1111_1000_1100_1001_0000_0011

0xFFF8C903

S-type Encoding

31 30 25	24 21 20	19 15	14 12	11 8 7	6 0	
funct7	rs2	rs1	funct3	rd	opcode	R-type
imm[11:0]		rs1	funct3	rd	opcode	I-type
12		5	3	5	7	
imm[11:5]	rs2	rs1	funct3	imm[4:0]	opcode	S-type
7	5					

Assembly: sw/sh/sb rs2,imm(rs1)

- Opcode: 0b0100011 for RV32I S-format store operations
- funct3:
  - Last 2 bits indicates w(10)/h(01)/b(00)
  - First bit 0

S-type Store Instructions

7	5	5	3	5	7	
imm[11:5]	rs2	rs1	funct3	imm[4:0]	opcode	S-type

Assembly: sw/sh/sb rs2,imm(rs1)

imm[11:5]	rs2	rs1	000	imm[4:0]	0100011	SB
imm[11:5]	rs2	rs1	001	imm[4:0]	0100011	SH
imm[11:5]	rs2	rs1	010	imm[4:0]	0100011	SW

S-type Example

imm[11:5]	rs2	rs1	funct3	imm[4:0]	opcode	
7	5	5	3	5	7	S-type

Assembly: *sw x20,-31(x21)*

1111111	10100	10101	010	00001	0100011

Machine code:

1111_1111_0100_1010_1010_0000_1010_0011

0xFF4AA0A3

B-type Conditional Branch

			5	3	5		7	
imm[11:5]		rs2	rs1	funct3	imm[4:0]		opcode	S-type
7		5						
imm[12]	imm[10:5]	rs2	rs1	funct3	imm[4:1]	imm[11]	opcode	B-type

Assembly: bne/beq/blt/bltu/beg/begu rs1,rs2,label

- Opcode: 0b1100011 for RV32I B-format branch operations
- How to represent label?

Branching Instruction Usage

- Branches typically used for loops (*if-else*, *while*, *for*)
  - Loops are generally small (< 50 instructions)
- Recall: Instructions stored in a localized area of memory (Code/Text)
  - Largest branch distance limited by size of code
  - Address of current instruction stored in the program counter (*PC*)

C Loop Mapped to RISC-V Assembly

```
int A[20];
int sum = 0;
for (int i=0; i < 20; i++)
    sum += A[i];
```

```
# Assume x8 holds pointer to A
# Assign x10=sum
  add x9, x8, x0  # x9=&A[0]
  add x10, x0, x0 # sum=0
  add x11, x0, x0 # i=0
  addi x13,x0, 20 # x13=20
Loop:
  bge x11,x13,Done
  lw x12, 0(x9)   # x12=A[i]
  add x10,x10,x12 # sum+=
  addi x9, x9,4   # &A[i+1]
  addi x11,x11,1  # i++
  j Loop
Done:
```

PC-Relative Addressing

- PC-relative addressing: use the immediate field as a two's-complement offset to PC
  - Branches generally change the PC by a small amount
  - Can specify ± 2^{11} 'unit' addresses from the PC
- Recall
  - Each instruction is 4-byte wide (4-byte aligned)
  - Address is multiple of 4, least significant 2 bits "00"
  - Could have used bits [13:2] for *imm*
  - Can specify ± 2^{13} 'unit' addresses from the PC
  - But, to support RVC (16-bit/2-byte instruction) extension, [12:1] for *imm/offset*, can specify ± 2^{12} 'unit' addresses from the PC
- Opposite to it, absolute addressing (use full address)

```
Disassembly of section

0000000000000000 <ltmp0
      0: ff c3 00 d1
      4: fd 7b 02 a9
      8: fd 83 00 91
      c: 08 00 80 52
     10: e8 0f 00 b9
     14: bf c3 1f b8
     18: 48 9a 80 52
     1c: a8 83 1f b8
```

B-type Conditional Branch

7		5	5	3	5		7	
imm[12]	imm[10:5]	rs2	rs1	funct3	imm[4:1]	imm[11]	opcode	B-type

Assembly: bne/beq/blt/bltu/beg/begu rs1,rs2,label

- Opcode: 0b1100011 for RV32I B-format branch operations
- Label: PC-relative addressing, concatenate imm[12], imm[11], then imm[10:5] and imm[4:1] as offset (sign-extended)

B-type Conditional Branch Example

7		5	5	3	5		7	
imm[12]	imm[10:5]	rs2	rs1	funct3	imm[4:1]	imm[11]	opcode	B-type

rs1 = 01011

rs2 = 01101

opcode = 1100011

funct3 = 101

imm/offset

= 6 instructions

= 24 bytes

↓

0|0|000000|1100|0

Bit 12 Bit 0

```
# Assume x8 holds pointer to A
# Assign x10=sum
  add x9, x8, x0  # x9=&A[0]
  add x10, x0, x0 # sum=0
  add x11, x0, x0 # i=0
  addi x13,x0, 20 # x13=20
Loop:
PC ⟶ bge x11,x13,Done
  lw x12, 0(x9)   # x12=A[i]
  add x10,x10,x12 # sum+=
  addi x9, x9,4   # &A[i+1]
  addi x11,x11,1  # i++
  j Loop
Done: # some instruction
```

B-type Conditional Branch Example

7		5	5	3	5		7	
imm[12]	imm[10:5]	rs2	rs1	funct3	imm[4:1]	imm[11]	opcode	B-type
0	000000	01101	01011	101	1100	0	1100011	

rs1 = 01011

rs2 = 01101

opcode = 1100011

funct3 = 101

imm/offset

= 6 instructions

= 24 bytes

↓

0|0|000000|1100|0

Bit 12 Bit 0

Machine code

0x00d5dc63

B-type Branch Instructions

7		5	5	3	5		7	
imm[12]	imm[10:5]	rs2	rs1	funct3	imm[4:1]	imm[11]	opcode	B-type

Assembly: bne/beq/blt/bltu/beg/begu rs1,rs2,imm/offset

imm[12|10:5]	rs2	rs1	000	imm[4:1|11]	1100011	BEQ
imm[12|10:5]	rs2	rs1	001	imm[4:1|11]	1100011	BNE
imm[12|10:5]	rs2	rs1	100	imm[4:1|11]	1100011	BLT
imm[12|10:5]	rs2	rs1	101	imm[4:1|11]	1100011	BGE
imm[12|10:5]	rs2	rs1	110	imm[4:1|11]	1100011	BLTU
imm[12|10:5]	rs2	rs1	111	imm[4:1|11]	1100011	BGEU

Further on Conditional Branch

- To support RVC (16-bit/2-byte instruction) extension, [12:1] for `imm/offset`, can specify ± 2^{12} 'unit' addresses from the PC
- Equivalent to ± 2^{10} 32-bit instructions
- **What if jump to farther away?**

```
beq x10, x0, far
# next instruction
```

```
bne x10, x0, next
j far
next: # next instruction
```

j gets 20-bit imm

J-type Jump Instruction

1	10	1	8	5	7	
imm[20]	imm[10:1]	imm[11]	imm[19:12]	rd	opcode	J-type

Assembly: jal rd, label

- Recall jal does 2 things:
  - Store PC+4 to rd as return address
  - Jump to label = PC + offset(imm)

- Label translated by assembler to a 20-bit offset (encoded similar to Branch offset)
- Can access ± 2^{20} 'unit' addresses from the PC
- ± 2^{18} 32-bit instructions

I-type Jump Instruction

imm[11:0]	rs1	funct3	rd	opcode	I-type
12	5	3	5	7	

Assembly: jalr rd,rs1,imm

- Recall jal does 2 things:
  - Store PC+4 to rd as return address
  - Jump to label = rs + offset(imm)

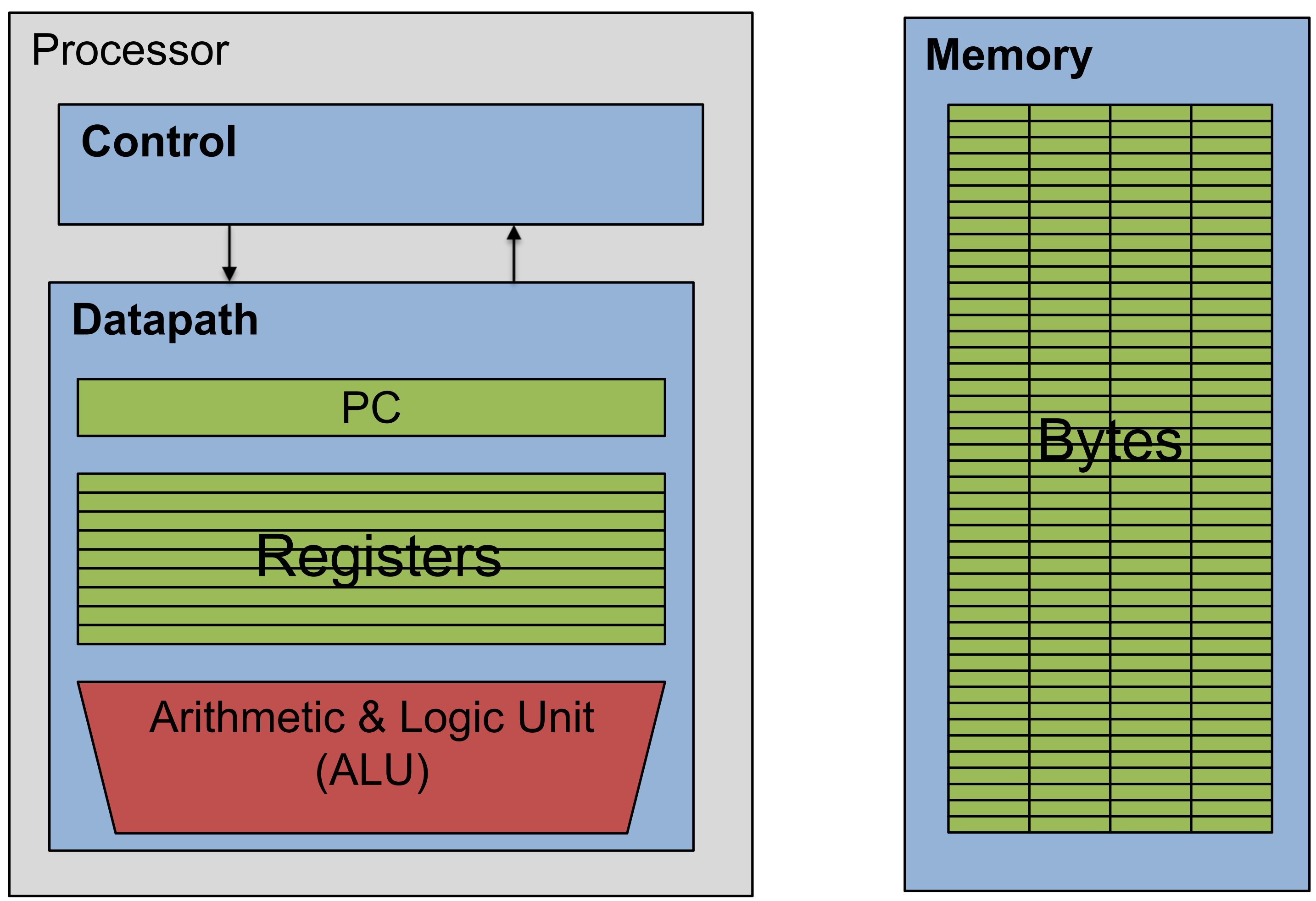
- imm can hold values between [-2048,2047]
- Unlike JAL, include the last 0 using I-format

True or False

- If we move all of code, the branch immediate field does not change.

True

Because it utilizes PC-relative addressing/offset



U-Format (Something New)

20	5	7	
imm[31:12]	rd	opcode	U-type

imm[20]	imm[10:1]	imm[11]	imm[19:12]	rd	opcode	
						J-type

Load upper immediate: lui rd,imm

rd = imm<<12

- Can be used to create long immediates to registers along with addi
  - Previously, it was 12-bit, e.g., addi x1, x0, 2047

Registers

lui x5, 0xABCDE — 0xABCDE000 — x5

+

addi x5, x5, 0x123 — 0xABCDE123 — x5

↓

li x5, 0xABCDE123 — A pseudo-instruction

Corner Cases

20	5	7	
imm[31:12]	rd	opcode	U-type

li x5,0xDEADBEEF

This is automatically handled by li without considering the details

Registers

lui x5, 0xDEADB

0xDEADB000	x5

addi x5, x5, 0xEEF

(0xEEF as imm)

	x5

✗

lui x5, 0xDEADC

addi x5, x5, 0xEEF

(0xEEF as imm)

U-Format

20				5	7	
imm[31:12]				rd	opcode	U-type
imm[20]	imm[10:1]	imm[11]	imm[19:12]	rd	opcode	J-type

Add upper immediate PC: auipc rd,imm

- rd = PC + (imm. << 12)

auipc x5, 0xABCDE

0xABCDE000 + PC x5

- lui opcode: 0b0110111
- auipc opcode: 0b0010111

LUI and AUIPC

- Call function with 32-bit absolute address

  lui x6, <hi20bits>

  jalr ra, x6, <lo12bits>

- Jump PC-relative with 32-bit offset

  auipc x6, <hi20bits>

  jalr ra, x6, <lo12bits>

- Obtain PC value

  auipc x6, 0

- Store/load with PC-relative 32-bit offset/32-bit absolute address

  auipc x6,<hi20bits>/lui x6,<hi20bits>

  sw/lw rd, (<lo12bits>)x6

Instruction Decoding

- Given an instruction in binary, how to interpret
- Reverse the procedure translating an instruction to machine code
  - Look up `opcode/funct3/funct7` to identify type & operation
  - Find out `rs1/rs2/rd/imm` value, whichever presents
  - More in hardware design

Summary

31	30 25	24 21	20	19 15	14 12	11 8	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]	imm[10:5]	rs2		rs1	funct3	imm[4:1]	imm[11]	opcode	B-type
imm[31:12]						rd		opcode	U-type
imm[20]	imm[10:1]		imm[11]	imm[19:12]		rd		opcode	J-type

imm[31:12]				rd	0110111	LUI
imm[31:12]				rd	0010111	AUIPC
imm[20|10:1|11|19:12]				rd	1101111	JAL
imm[11:0]		rs1	000	rd	1100111	JALR
imm[12|10:5]	rs2	rs1	000	imm[4:1|11]	1100011	BEQ
imm[12|10:5]	rs2	rs1	001	imm[4:1|11]	1100011	BNE
imm[12|10:5]	rs2	rs1	100	imm[4:1|11]	1100011	BLT
imm[12|10:5]	rs2	rs1	101	imm[4:1|11]	1100011	BGE
imm[12|10:5]	rs2	rs1	110	imm[4:1|11]	1100011	BLTU
imm[12|10:5]	rs2	rs1	111	imm[4:1|11]	1100011	BGEU
imm[11:0]		rs1	000	rd	0000011	LB
imm[11:0]		rs1	001	rd	0000011	LH
imm[11:0]		rs1	010	rd	0000011	LW
imm[11:0]		rs1	100	rd	0000011	LBU
imm[11:0]		rs1	101	rd	0000011	LHU
imm[11:5]	rs2	rs1	000	imm[4:0]	0100011	SB
imm[11:5]	rs2	rs1	001	imm[4:0]	0100011	SH
imm[11:5]	rs2	rs1	010	imm[4:0]	0100011	SW
imm[11:0]		rs1	000	rd	0010011	ADDI
imm[11:0]		rs1	010	rd	0010011	SLTI
imm[11:0]		rs1	011	rd	0010011	SLTIU
imm[11:0]		rs1	100	rd	0010011	XORI
imm[11:0]		rs1	110	rd	0010011	ORI
imm[11:0]		rs1	111	rd	0010011	ANDI

0000000	shamt	rs1	001	rd	0010011	SLLI
0000000	shamt	rs1	101	rd	0010011	SRLI
0100000	shamt	rs1	101	rd	0010011	SRAI
0000000	rs2	rs1	000	rd	0110011	ADD
0100000	rs2	rs1	000	rd	0110011	SUB
0000000	rs2	rs1	001	rd	0110011	SLL
0000000	rs2	rs1	010	rd	0110011	SLT
0000000	rs2	rs1	011	rd	0110011	SLTU
0000000	rs2	rs1	100	rd	0110011	XOR
0000000	rs2	rs1	101	rd	0110011	SRL
0100000	rs2	rs1	101	rd	0110011	SRA
0000000	rs2	rs1	110	rd	0110011	OR
0000000	rs2	rs1	111	rd	0110011	AND
0000 pred	succ	00000	000	00000	0001111	FENCE
0000 0000	0000	00000	001	00000	0001111	FENCE.I
000000000000		00000	000	00000	1110011	ECALL
000000000001		00000	000	00000	1110011	EBREAK
csr		rs	[illegible]	rd	1110011	CSRRW
csr		rs	[illegible]	rd	1110011	CSRRS
csr		rs	[illegible]	rd	1110011	CSRRC
csr		zimm	101	rd	1110011	CSRRWI
csr		zimm	110	rd	1110011	CSRRSI
csr		zimm	111	rd	1110011	CSRRCI

Not in CS110