

B-Format Layout

RISC-V Conditional Branches

- E.g., `beq x1, x2, Label`
- Branches read two registers but don't write to a register (similar to stores)
- How to encode label, i.e., where to branch to?

Branching Instruction Usage

- Branches typically used for loops (**if-else**, **while**, **for**)
 - Loops are generally small (< 50 instructions)
 - Function calls and unconditional jumps handled with jump instructions (J-Format)
- **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)

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 $\pm 2^{11}$ 'unit' addresses from the PC
 - (We will see in a bit that we can encode 12-bit offsets as immediates)
- Why not use byte as a unit of offset from PC?
 - Because instructions are 32-bits (4-bytes)
 - We don't branch into middle of instruction

Scaling Branch Offset

- One idea: To improve the reach of a single branch instruction, multiply the offset by four bytes before adding to PC
- This would allow one branch instruction to reach $\pm 2^{11} \times 32$ -bit instructions either side of PC
 - Four times greater reach than using byte offset

Branch Calculation

- If we **don't** take the branch:

$$PC = PC + 4 \quad (\text{i.e., next instruction})$$

- If we **do** take the branch:

$$PC = PC + \text{immediate} * 4$$

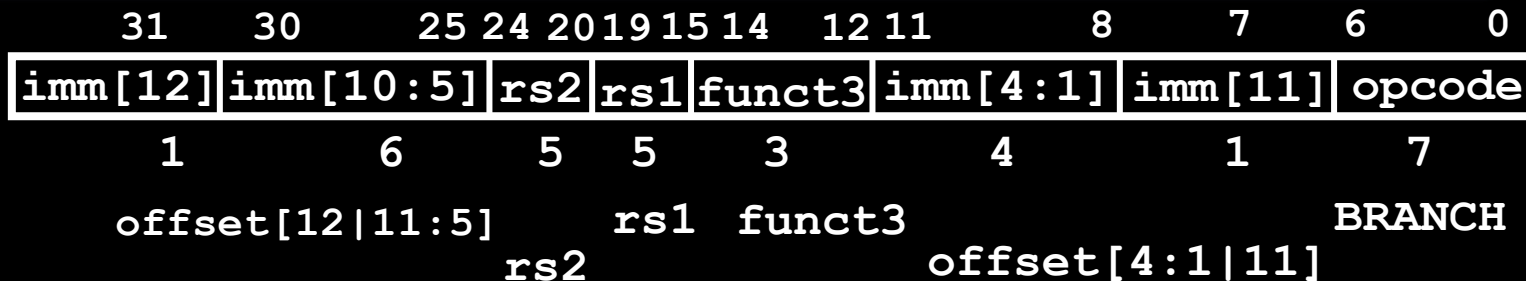
- Observations:

- **immediate** is number of instructions to jump (remember, specifies words) either forward (+) or backwards (−)

RISC-V Feature, $n \times 16\text{-bit}$ Instructions

- Extensions to RISC-V base ISA support 16-bit compressed instructions and also variable-length instructions that are multiples of 16-bits in length
- To enable this, RISC-V scales the branch offset by 2 bytes even when there are no 16-bit instructions
- Reduces branch reach by half and means that $\frac{1}{2}$ of possible targets will be errors on RISC-V processors that only support 32-bit instructions (as used in this class)
- RISC-V conditional branches can only reach $\pm 2^{10} \times 32\text{-bit}$ instructions on either side of PC

RISC-V B-Format for Branches



- B-format is mostly same as S-Format, with two register sources (**rs1/rs2**) and a 12-bit immediate **imm[12:1]**
- But now immediate represents values -4096 to +4094 in 2-byte increments
- The 12 immediate bits encode *even* 13-bit signed byte offsets (lowest bit of offset is always zero, so no need to store it)

Branch Example, Determine Offset

- RISC-V Code:

```

Loop: beq  x19,x10,End
      add  x18,x18,x10
      addi x19,x19,-1
      j    Loop
End:   # target instruction
  
```

Count instructions
from branch

- 1
- 2
- 3
- 4

- Branch offset =

$4 \times 32\text{-bit instructions} = 16 \text{ bytes}$

- (Branch with offset of 0,
branches to itself)





Branch Example, Determine Offset

- RISC-V Code:

```
Loop: beq  x19,x10,End
      add  x18,x18,x10
      addi x19,x19,-1
      j    Loop
End:   # target instruction
```

Count instructions
from branch

- 1
- 2
- 3
- 4

| | | | | | |
|---------|-------|-------|-----|-------|---------|
| ??????? | 01010 | 10011 | 000 | ????? | 1100011 |
|---------|-------|-------|-----|-------|---------|

imm rs2=10 rs1=19 BEQ imm BRANCH

Branch Example, Determine Offset

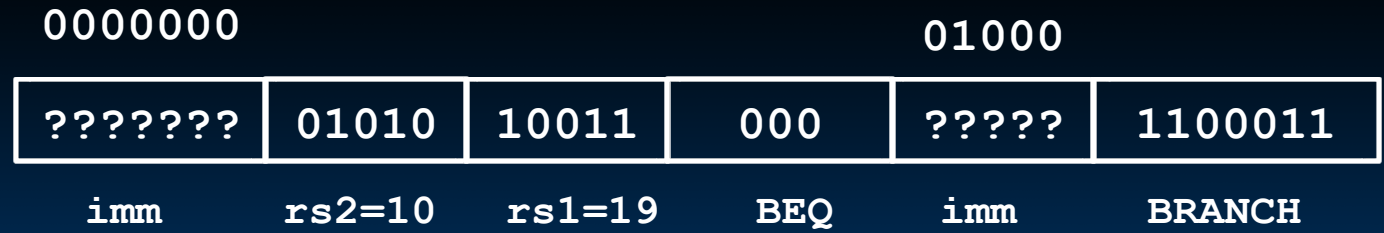
- RISC-V Code:

```

Loop: beq  x19,x10,End
      add  x18,x18,x10
      addi x19,x19,-1
      j    Loop
End:   # target instruction
  
```

Offset = 16 bytes
= 8 x 2

1
2
3
4



RISC-V Immediate Encoding

Instruction encodings, inst[31:0]

| 31 | 30 | 25 | 24 | 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 10:5] | | | | rs2 | | rs1 | | funct3 | | imm[4:1 11] | | opcode | | B-type |

32-bit immediates produced, imm[31:0]

| 31 | 25 | 24 | 12 | 11 | 10 | 5 | 4 | 1 | 0 | | |
|------------|----|----|----|---------|-------------|---|-------------|---|----------|--|--------|
| -inst[31]- | | | | | inst[30:25] | | inst[24:21] | | inst[20] | | I-imm. |
| -inst[31]- | | | | | inst[30:25] | | inst[11:8] | | inst[7] | | S-imm. |
| -inst[31]- | | | | inst[7] | inst[30:25] | | inst[11:8] | | 0 | | B-imm. |

Upper bits sign-extended from **inst[31]** always

Only bit 7 of instruction changes role in immediate between S and B

Branch Example, Complete Encoding

beq **x19,x10**, offset = 16 bytes

13-bit immediate, $imm[12:0]$, with value 16

$imm[0]$ discarded,
always zero

0000000010000

$imm[12]$

$imm[11]$



$imm[10:5]$ **rs2=10** **rs1=19** **BEQ** $imm[4:1]$ **BRANCH**

All RISC-V Branch Instructions

| | | | | | | |
|---------------------------|------------------|------------------|------------------|--------------------------|----------------------|-------------|
| <code>imm[12 10:5]</code> | <code>rs2</code> | <code>rs1</code> | <code>000</code> | <code>imm[4:1 11]</code> | <code>1100011</code> | beq |
| <code>imm[12 10:5]</code> | <code>rs2</code> | <code>rs1</code> | <code>001</code> | <code>imm[4:1 11]</code> | <code>1100011</code> | bne |
| <code>imm[12 10:5]</code> | <code>rs2</code> | <code>rs1</code> | <code>100</code> | <code>imm[4:1 11]</code> | <code>1100011</code> | blt |
| <code>imm[12 10:5]</code> | <code>rs2</code> | <code>rs1</code> | <code>101</code> | <code>imm[4:1 11]</code> | <code>1100011</code> | bge |
| <code>imm[12 10:5]</code> | <code>rs2</code> | <code>rs1</code> | <code>110</code> | <code>imm[4:1 11]</code> | <code>1100011</code> | bltu |
| <code>imm[12 10:5]</code> | <code>rs2</code> | <code>rs1</code> | <code>111</code> | <code>imm[4:1 11]</code> | <code>1100011</code> | bgeu |

Long
Immediates

Questions on PC-addressing

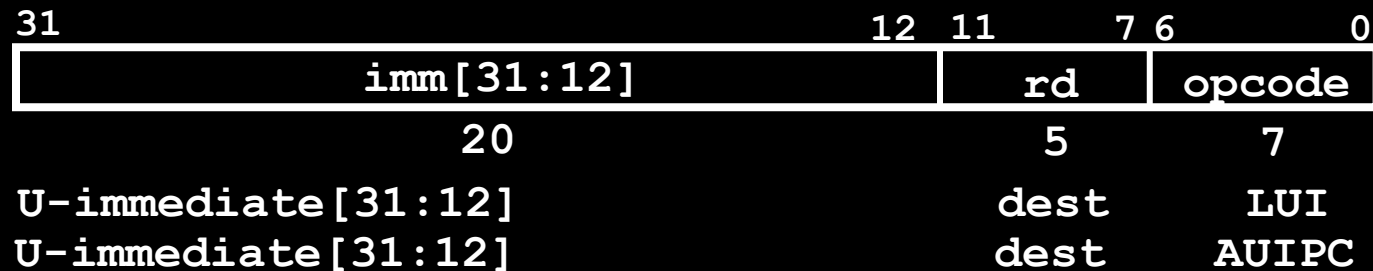
- Does the value in branch immediate field change if we move the code?
 - If moving individual lines of code, then yes
 - If moving all of code, then no ('position-independent code')
- What do we do if destination is $> 2^{10}$ instructions away from branch?
 - Other instructions save us

Questions on PC-addressing

- Does the value in branch immediate field change if we move the code?
 - If moving individual lines of code, then yes
 - If moving all of code, then no ('position-independent code')
- What do we do if destination is $> 2^{10}$ instructions away from branch?
 - Other instructions save us

```
beq x10,x0,far          bne x10,x0,next
# next instr           →      j    far
                        next: # next instr
```

U-Format for “Upper Immediate” Instructions



- Has 20-bit immediate in upper 20 bits of 32-bit instruction word
- One destination register, rd
- Used for two instructions
 - **lui** – Load Upper Immediate
 - **auipc** – Add Upper Immediate to PC

LUI to Create Long Immediates

- LUI writes the upper 20 bits of the destination with the immediate value, and clears the lower 12 bits.
- Together with an **addi** to set low 12 bits, can create any 32-bit value in a register using two instructions (**lui/addi**).

```
lui x10, 0x87654          # x10 = 0x87654000
addi x10, x10, 0x321       # x10 = 0x87654321
```

One Corner Case

How to set 0xDEADBEEF?

```
lui x10, 0xDEADB      # x10 = 0xDEADB000
addi x10, x10, 0xEEF   # x10 = 0xDEADAEEF
```

addi 12-bit immediate is always sign-extended, if top bit is set, will subtract -1 from upper 20 bits

Solution

How to set 0xDEADBEEF?

```
LUI x10, 0xDEADC    # x10 = 0xDEADC000
```

```
ADDI x10, x10, 0xEEF    # x10 =  
                        #0xDEADBEEF
```

Pre-increment value placed in upper 20 bits, if sign bit will be set on immediate in lower 12 bits.

Assembler pseudo-op handles all of this:

```
li x10, 0xDEADBEEF    # Creates two  
                        #instructions
```

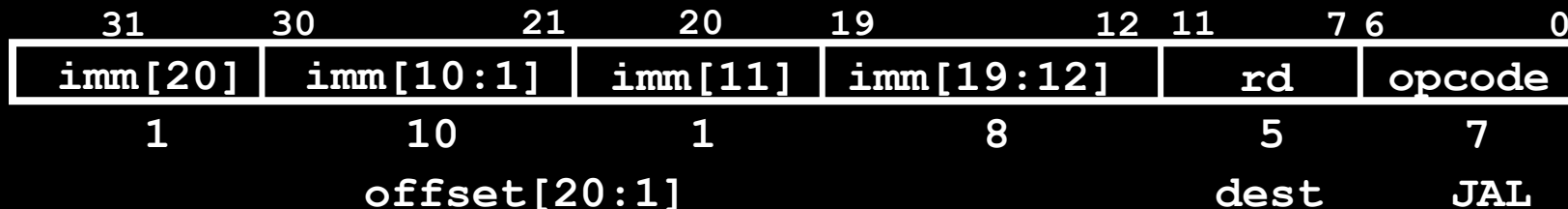
- Adds upper immediate value to PC and places result in destination register
- Used for PC-relative addressing

```
Label: AUIPC x10, 0 # Puts address of
                    # Label in x10
```

J-Format



J-Format for Jump Instructions



- **j**al saves PC+4 in register **rd** (the return address)
 - Assembler “j” jump is pseudo-instruction, uses JAL but sets **rd=x0** to discard return address
- Set PC = PC + offset (PC-relative jump)
- Target somewhere within $\pm 2^{19}$ locations, 2 bytes apart
 - $\pm 2^{18}$ 32-bit instructions
- Immediate encoding optimized similarly to branch instruction to reduce hardware cost



Uses of JAL

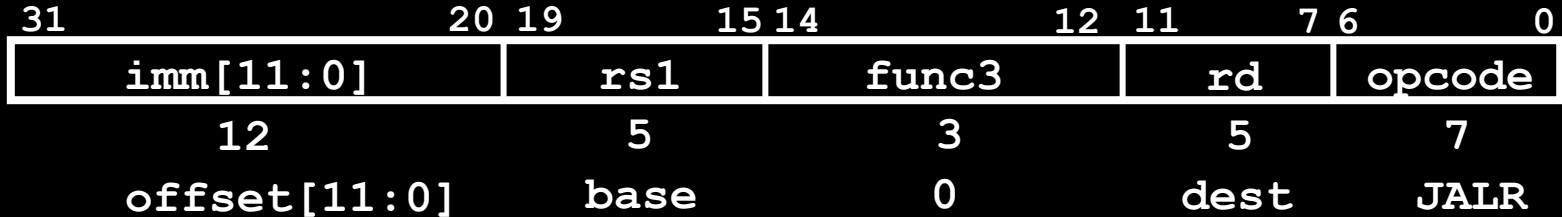
```
# j pseudo-instruction
```

```
j Label = jal x0, Label # Discard return  
address
```

```
# Call function within  $2^{18}$  instructions  
of PC
```

```
jal ra, FuncName
```

JALR Instruction (I-Format)



- **jalr rd, rs, immediate**
 - Writes PC+4 to rd (return address)
 - Sets $PC = rs + immediate$
 - Uses same immediates as arithmetic and loads
 - **no** multiplication by 2 bytes
 - In contrast to branches and **jal**



Uses of JALR

ret and jr psuedo-instructions

ret = jr ra = jalr x0, ra, 0

Call function at any 32-bit absolute address

lui x1, <hi20bits>

jalr ra, x1, <lo12bits>

Jump PC-relative with 32-bit offset

auipc x1, <hi20bits>

jalr x0, x1, <lo12bits>


**“And In
Conclusion...”**

Summary of RISC-V Instruction Formats

| | | | | | | | | | | | | | | | | | | |
|------------------|----|----|----|-----|-----|----|------------|-----|----|--------|--------|----|-------------|--------|--------|--------|--------|--------|
| 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 10:5] | | | | rs2 | | | rs1 | | | funct3 | | | imm[4:1 11] | | | opcode | | B-type |
| imm[31:12] | | | | | | | | | | rd | | | opcode | | U-type | | | |
| imm[20 10:1 11]] | | | | | | | imm[19:12] | | | | | rd | | opcode | | J-type | | |



Complete RV32I ISA!

| Open  RISC-V Reference Card .. | | | | | | | | | |
|---|-------------------------|-----|-------------------|--------|--------------------|---|----------------|--|--|
| Base Integer Instructions: RV32I | | | | | | | | | |
| Category | Name | Fmt | RV32I Base | | | | | | |
| Shifts | Shift Left Logical | R | SLL rd,rs1,rs2 | | | | | | |
| | Shift Left Log. Imm. | I | SLLI rd,rs1,shamt | | | | | | |
| | Shift Right Logical | R | SRL rd,rs1,rs2 | | | | | | |
| | Shift Right Log. Imm. | I | SRLI rd,rs1,shamt | | | | | | |
| | Shift Right Arithmetic | R | SRA rd,rs1,rs2 | | | | | | |
| | Shift Right Arith. Imm. | I | SRAI rd,rs1,shamt | | | | | | |
| Arithmetic | ADD | R | ADD rd,rs1,rs2 | | | | | | |
| | ADD Immediate | I | ADDI rd,rs1,imm | | | | | | |
| | SUBtract | R | SUB rd,rs1,rs2 | | | | | | |
| | Load Upper Imm | U | LUI rd,imm | | | | | | |
| Logical | Add Upper Imm to PC | U | AUIPC rd,imm | Loads | Load Byte | I | LB rd,rs1,imm | | |
| | XOR | R | XOR rd,rs1,rs2 | | Load Halfword | I | LH rd,rs1,imm | | |
| | XOR Immediate | I | XORI rd,rs1,imm | | Load Byte Unsigned | I | LBU rd,rs1,imm | | |
| | OR | R | OR rd,rs1,rs2 | | Load Half Unsigned | I | LHU rd,rs1,imm | | |
| | OR Immediate | I | ORI rd,rs1,imm | | Load Word | I | LW rd,rs1,imm | | |
| | AND | R | AND rd,rs1,rs2 | Stores | Store Byte | S | SB rs1,rs2,imm | | |
| Compare | AND Immediate | I | ANDI rd,rs1,imm | | Store Halfword | S | SH rs1,rs2,imm | | |
| | Set < | R | SLT rd,rs1,rs2 | | Store Word | S | SW rs1,rs2,imm | | |
| | Set < Immediate | I | SLTI rd,rs1,imm | | | | | | |
| | Set < Unsigned | R | SLTU rd,rs1,rs2 | | | | | | |
| | Set < Imm Unsigned | I | SLTIU rd,rs1,imm | | | | | | |
| | Branch = | B | BEQ rs1,rs2,imm | | | | | | |
| Branches | Branch ≠ | B | BNE rs1,rs2,imm | | | | | | |
| | Branch < | B | BLT rs1,rs2,imm | | | | | | |
| | Branch ≥ | B | BGE rs1,rs2,imm | | | | | | |
| | Branch < Unsigned | B | BLTU rs1,rs2,imm | | | | | | |
| | Branch ≥ Unsigned | B | BGEU rs1,rs2,imm | | | | | | |
| | | | | | | | | | |
| Jump & Link | J&L | J | JAL rd,imm | | | | | | |
| | Jump & Link Register | I | JALR rd,rs1,imm | | | | | | |

Synch

Synch thread

I

FENCE

Environment

CALL

I

ECALL

BREAK

I

EBREAK

Not in 61C