



UC Berkeley
Teaching Professor
Dan Garcia

CS61C

Great Ideas in Computer Architecture (a.k.a. Machine Structures)



UC Berkeley
Teaching Professor
Lisa Yan

决策 RISC-V Decision Making and Logical Operations

逻辑操作
cs61c.org

Berkeley
UNIVERSITY OF CALIFORNIA

Garcia, Yan



VIEW FROM THE TOP

Conversations with Global Leaders

Driving Moore's Law:

A Fireside Chat with Ann Kelleher and
Dean Tsu-Jae King Liu

ANN KELLEHER

*EXECUTIVE VICE PRESIDENT &
GENERAL MANAGER OF TECHNOLOGY DEVELOPMENT
INTEL*

**Wednesday, Sept. 21 | 12-1 p.m.
Banatao Auditorium, Sutardja Dai Hall**

Event is free and open to the Berkeley community.
Lunch will be provided.

Co-hosted with the Society of Women Engineers
engineering.berkeley.edu/events



**Berkeley
Engineering**

THE ANIMAL TRANSLATORS

"Researchers are using machine learning (ML) systems to decode animal communication. Scientists at Germany's Max Planck Institute for Brain Research used ML algorithms to analyze 36,000 mole rat chirps in seven colonies, identifying unique vocal signatures for each mole rat, as well as a distinct dialect for each colony. The multi-institutional Project CETI (Cetacean Translation Initiative) hopes to decipher the communication of sperm whales through the efforts of ML specialists, marine biologists, roboticists, linguists, and cryptographers. The project will involve recording whale sounds and movements via underwater microphones, robotic fish, and acoustic tags. Other projects aim to build technologies that enable human-animal communication, with Hunter College's Diana Reiss envisioning "a Google Translate for animals."



- Memory is **byte-addressable**, but **lw** and **sw** access one **word** at a time.
- A pointer (used by **lw** and **sw**) is just a memory address, we can add to it or subtract from it (using offset).
- Big- vs Little Endian
 - Tip: draw lowest byte on the right
- New Instructions:
lw, sw, lb, sb, lbu

Decision Making

- Addition/subtraction

```
add rd, rs1, rs2  
sub rd, rs1, rs2
```

- Add immediate

```
addi rd, rs1, imm
```

- Load/store

```
lw rd, rs1, imm  
lb rd, rs1, imm  
lbu rd, rs1, imm  
sw rs1, rs2, imm  
sb rs1, rs2, imm
```

Computer Decision Making

beq: branch if
equal

bne: branch if
not equal

- Based on computation, do something different
- In programming languages: if-statement
- RISC-V: *if-statement instruction is*
beq reg1, reg2, L1
means: go to statement labeled L1
if (value in reg1) == (value in reg2)
....otherwise, go to next statement
- **beq** stands for *branch if equal*
- Other instruction: **bne** for *branch if not equal*

Types of Branches

breq : ==

bneq : !=

blt : <

bge : >=

bltu

bgeu

- *Branch – change of control flow*

- *Conditional Branch – change control flow depending on outcome of comparison*

- branch if equal (**breq**) or branch if not equal (**bneq**)
 - Also branch if less than (**blt**) and branch if greater than or equal (**bge**)
 - And unsigned versions (**bltu**, **bgeu**)

- *Unconditional Branch – always branch*

- a RISC-V instruction for this: *jump* (**j**), as in **j label**

jump

Example *if* Statement

- Assuming translations below, compile *if* block

$f \rightarrow x_{10}$

$g \rightarrow x_{11}$

$h \rightarrow x_{12}$

$i \rightarrow x_{13}$

$j \rightarrow x_{14}$

bne $x_{13}, x_{14}, \text{Exit}$

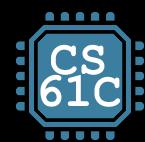
add x_{10}, x_{11}, x_{12} if $(i == j)$
 $f = g + h;$

bne $x_{13}, x_{14}, \text{Exit}$

add x_{10}, x_{11}, x_{12}

Exit:

- May need to ~~negate~~ branch condition



Example *if-else* Statement

- Assuming translations below, compile

$f \rightarrow \mathbf{x10}$

$g \rightarrow \mathbf{x11}$

$h \rightarrow \mathbf{x12}$

$i \rightarrow \mathbf{x13}$

$j \rightarrow \mathbf{x14}$

```
if (i == j)          bne x13,x14,Else
    f = g + h;      add x10,x11,x12
else                  j Exit
    f = g - h; Else:sub x10,x11,x12
                        Exit:
```

Magnitude Compares in RISC-V

- General programs need to test < and > as well.
- RISC-V magnitude-compare branches:
“Branch on Less Than”

Syntax: **blt reg1,reg2, Label**

Meaning: **if (reg1 < reg2) goto Label;**

“Branch on Less Than Unsigned”

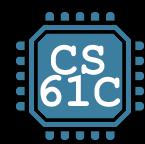
Syntax: **bltu reg1,reg2, Label**

Meaning: **if (reg1 < reg2) // treat registers
as unsigned integers
goto label;**

Also “Branch on Greater or Equal” **bge** and **bgeu**

Note: No ‘**bgt**’ or ‘**ble**’ instructions

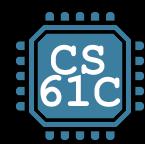
n: treat registers
as unsigned integers



Loops in C/Assembly

- *There are three types of loops in C:*
 - **while**
 - **do ... while**
 - **for**
- *Each can be rewritten as either of the other two, so the same branching method can be applied to these loops as well.*
- *Key concept: Though there are multiple ways of writing a loop in RISC-V, the key to decision-making is conditional branch* 条件分支

58 affair



C Loop Mapped to RISC-V Assembly

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

x9: address of array

x10: sum

x11: i

x12: A[i]

x13: 20

```
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+=x12  
addi x9, x9, 4 # &A[i+1]  
addi x11, x11, 1 # i++  
j Loop
```

Done:

Logical Instructions

- Add/sub

```
add rd, rs1, rs2  
sub rd, rs1, rs2
```

- Add immediate

```
addi rd, rs1, imm
```

- Load/store

```
lw rd, rs1, imm  
lb rd, rs1, imm  
lbu rd, rs1, imm  
sw rs1, rs2, imm  
sb rs1, rs2, imm
```

- Branching

```
beq rs1, rs2, Label  
bne rs1, rs2, Label  
bge rs1, rs2, Label  
blt rs1, rs2, Label  
bgeu rs1, rs2, Label  
bltu rs1, rs2, Label  
j Label
```



RISC-V Logical Instructions

- *Useful to operate on fields of bits within a word*
 - *e.g., characters within a word (8 bits)*
 - *Operations to pack /unpack bits into words*
 - *Called logical operations*

<i>Logical operations</i>	<i>C operators</i>	<i>Java operators</i>	<i>RISC-V instructions</i>
<i>Bit-by-bit AND</i>	&	&	and
<i>Bit-by-bit OR</i>			or
<i>Bit-by-bit XOR</i>	^	^	xor
<i>Shift left logical</i>	<<	<<	sll
<i>Shift right logical</i>	>>	>>	srl

与成氏九叔
古稀

RISC-V Logical Instructions

- Always two variants
 - Register: **and** x5, x6, x7 # x5 = x6 & x7
 - Immediate: **andi** x5, x6, 3 # x5 = x6 & 3
- Used for 'masks'
 - **andi** with 0000 00FF_{hex} isolates the least significant byte
 - **andi** with FF00 0000_{hex} isolates the most significant byte

No NOT in RISC-V

- *There is no logical NOT in RISC-V*
 - Use **xor** with 11111111_{two}
 - Remember - simplicity...

not
can be replaced by
the number xor 11111111

x	y	XOR(x,y)
0	0	0
0	1	1
1	0	1
1	1	0

Logical Shifting

- *Shift Left Logical (**sll**) and immediate (**slli**):*

slli x11,x12,2 #x11=x12<<2

- *Store in **x11** the value from **x12** shifted by 2 bits to the left (they fall off end), inserting 0's on right; << in C.*

- *Before:* **0000 0002_{hex}**
0000 0000 0000 0000 0000 0000 0000 0000 0010_{two}

- *After:* **0000 0008_{hex}**
0000 0000 0000 0000 0000 0000 0000 1000_{two}

- *What arithmetic effect does shift left have?*

- *Shift Right: **srl** is opposite shift; >>*

Arithmetic Shifting

- Shift right arithmetic (**sra**, **srai**) moves n bits to the right (insert high-order sign bit into empty bits)
- For example, if register x10 contained

1111 1111 1111 1111 1111 1111 1111 1110 0111_{two} = -25_{ten}

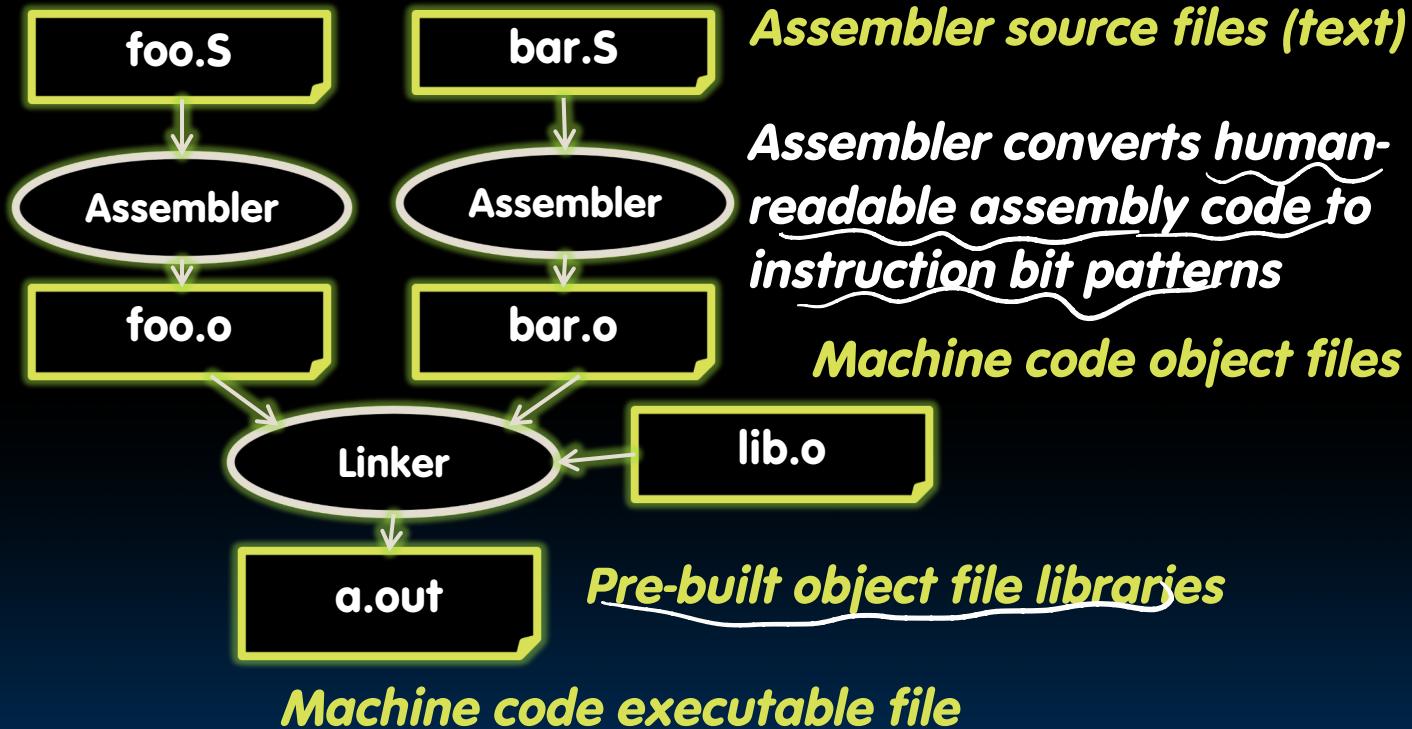
- If execute **srai x10, x10, 4**, result is:

1111 1111 1111 1111 1111 1111 1111 1111 1110_{two} = -2_{ten}

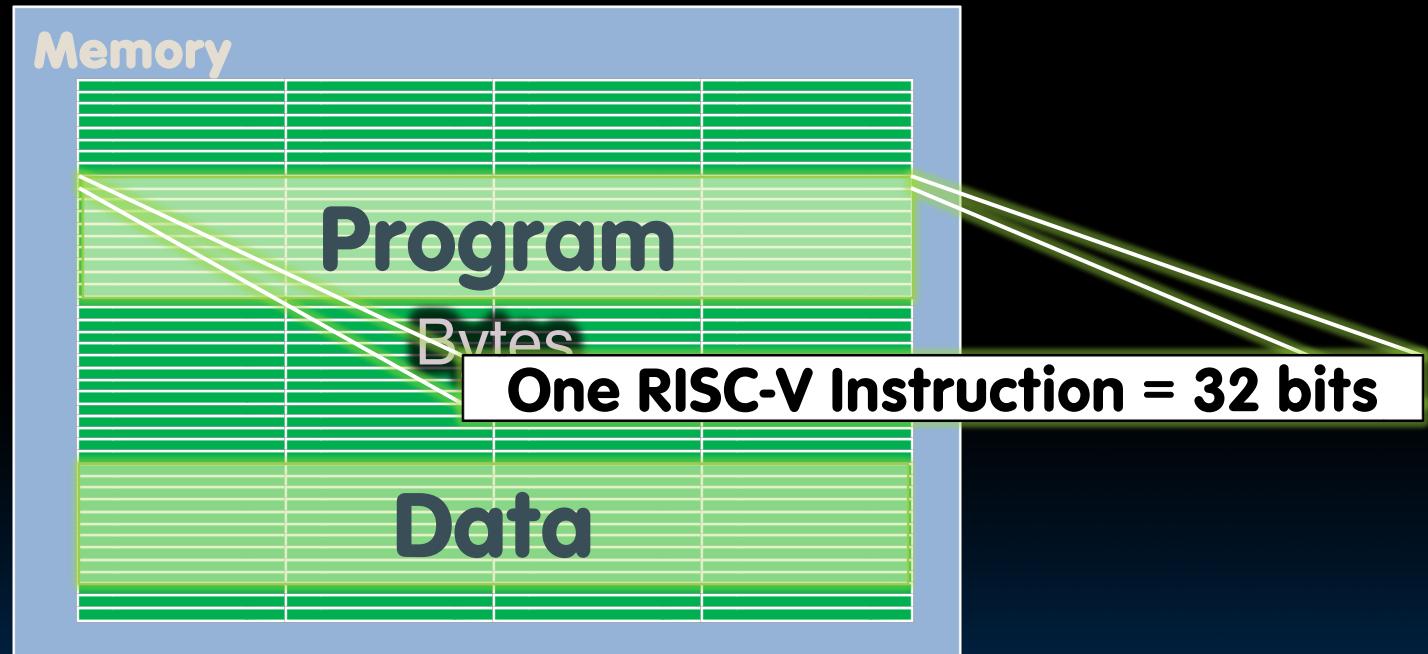
- Unfortunately, this is NOT same as dividing by 2^n
 - Fails for odd negative numbers
 - C arithmetic semantics is that division should round towards 0

A Bit About Machine Program

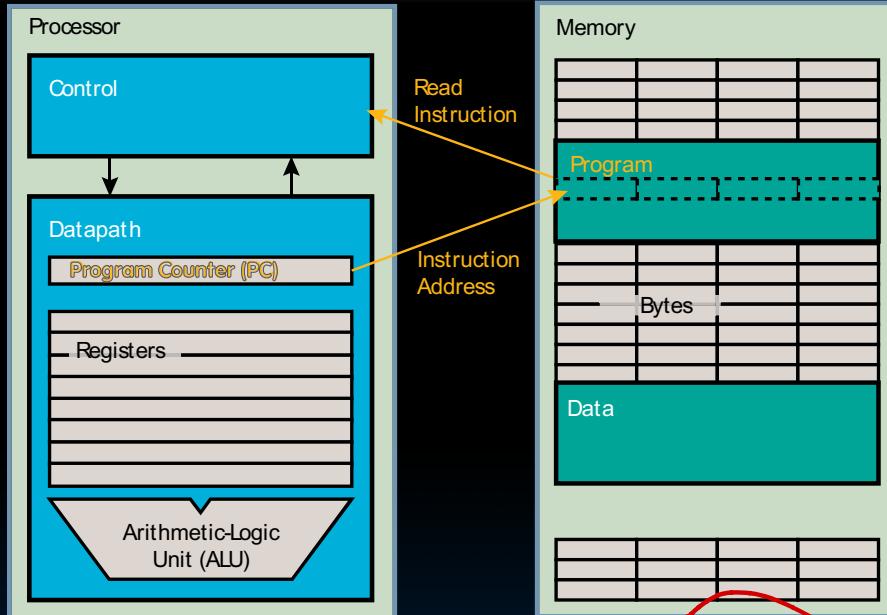
Assembler to Machine Code (More Later in Course)



How Program is Stored



Program Execution



- *PC (program counter) is a register internal to the processor that holds byte address of next instruction to be executed*

- Instruction is fetched from memory, then control unit executes instruction using datapath and memory system, and updates PC (default add +4 bytes to PC, to move to next sequential instruction; branches, jumps alter)

Helpful RISC-V Assembler Features

- Symbolic register names
 - E.g., a0-a7 for argument registers (x10-x17) for function calls
 - E.g., zero for x0

- Pseudo-instructions 伪指令

mv

li

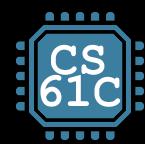
nop

- Shorthand syntax for common assembly idioms

- E.g., `mv rd, rs = addi rd, rs, 0`

- E.g., `li rd, 13 = addi rd, x0, 13`

- E.g., `nop = addi x0, x0, 0`



Example: Translate $*x = *y$

We want to translate $*x = *y$ into RISC-V
 x, y ptrs stored in: $x3$ $x5$

```
1: add x3, x5, zero
2: add x5, x3, zero
3: lw x3, 0(x5)
4: lw x5, 0(x3)
5: lw x8, 0(x5)
6: sw x8, 0(x3)
7: lw x5, 0(x8)
8: sw x3, 0(x8)
```

1	
2	
3	
4	
5→6	
6→5	
7→8	

L08b Translate $*x = *y;$

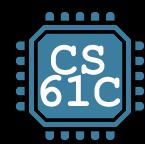
We want to translate $*x = *y$ into RISC-V
 x, y ptrs stored in: **x3** **x5**

```
1: add x3, x5, zero
2: add x5, x3, zero
3: lw x3, 0(x5)
4: lw x5, 0(x3)
5: lw x8, 0(x5)
6: sw x8, 0(x3)
7: lw x5, 0(x8)
8: sw x3, 0(x8)
```

1
2

3
4

5 → 6
6 → 5
7 → 8



Your Turn. What is in x12?

x10 holds 0x34FF

slli x12, x10, 0x10

srl x12, x12, 0x08

and x12, x12, x10 0x0

0x3400

0x4F0

0xFF00

0x34FF

L09a Logical Operations. What is in x12?

x10 holds 0x34FF

slli x12, x10, 0x10

srli x12, x12, 0x08

and x12, x12, x10

0x0
0x3400
0x4F0
0xFF00
0x34FF