Misc Topics

Caiwen Ding
Department of Computer Science and Engineering
University of Connecticut

CSE3666: Introduction to Computer Architecture

Outline

- A few more instructions
  - AUIPC
- RISC vs CISC

- Set less than
- Frame pointer (Section 2.8)

How does LA work?

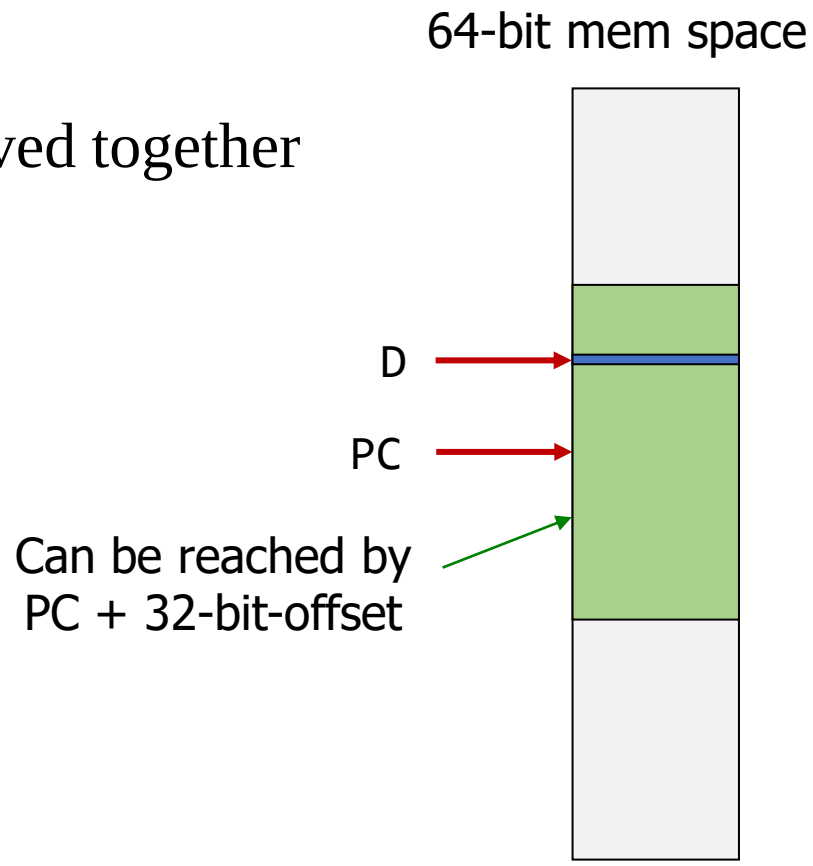
```
la rd, var   # load var's addr into rd
```

- An address is a 32-bit value. We could use LUI and ADDI to get any 32-bit value into a register

- Not easy on 64-bit processors where addresses are of 64 bits
  - More instructions are needed

PC-relative addressing

- Use PC-relative addressing to access data near the instruction
  - Add a 32-bit offset to PC
- Benefit: Data and code can be moved together



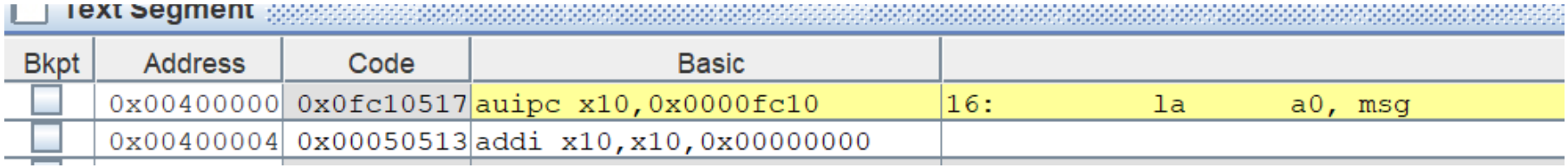
Can only access a tiny area of the memory space, but is good enough for common cases.

AUIPC

```
AUIPC rd, immd      # rd = PC + UI
```

- Operations (add upper immediate and PC):
  - Obtain an immediate like LUI:

    The higher 20 bits from machine code, the lower 12 bits are 0
  - Add the immediate and PC PC-relative addressing



Text Segment

Bkpt	Address	Code	Basic	
	0x00400000	0x0fc10517	auipc x10,0x0000fc10	16: la a0, msg
	0x00400004	0x00050513	addi x10,x10,0x00000000	

```
# some assemblers convert LA to the following
auipc   x1, %pcrel_hi(var)
addi    x1, %pcrel_lo(var)       # Add the lower 12 bits

# RARS does not support %pcrel_hi
```

Example

- What is in t0(x5) after auipc?
- What is in t0(x5) after the second ADDI instruction?

main:

Address	Code	Basic	
0x00400000	0x00000013	addi x0,x0,0x00000000	4: nop
0x00400004	0x00000297	auipc x5,0x00000000	5: la t0, main
0x00400008	0xffc28293	addi x5,x5,0xfffffffc	

Encoding of AUIPC

- Which format would you use to encode AUPIC?

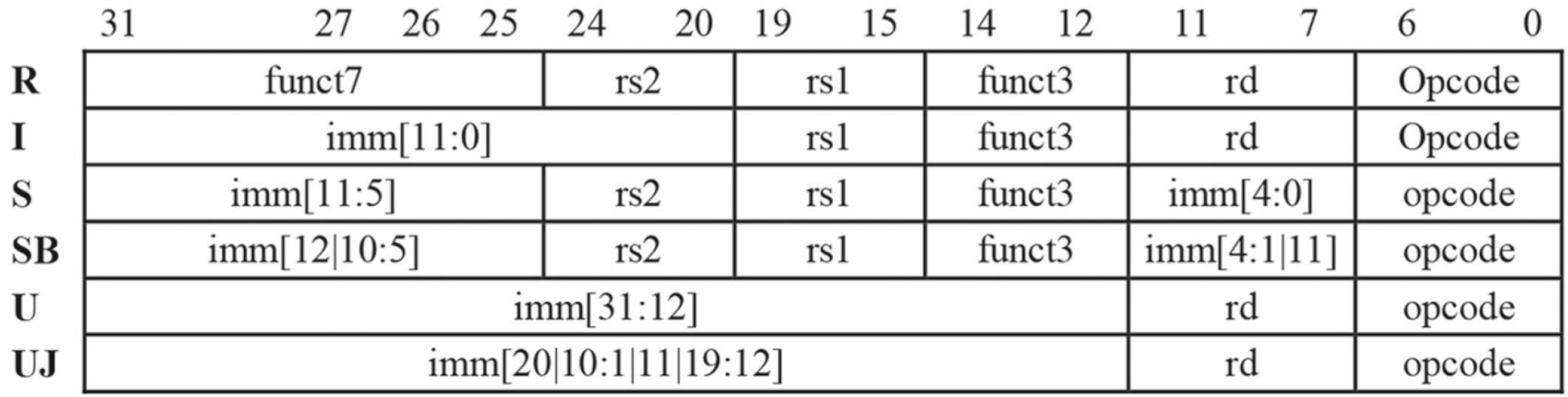
A. I

B. S

C. SB

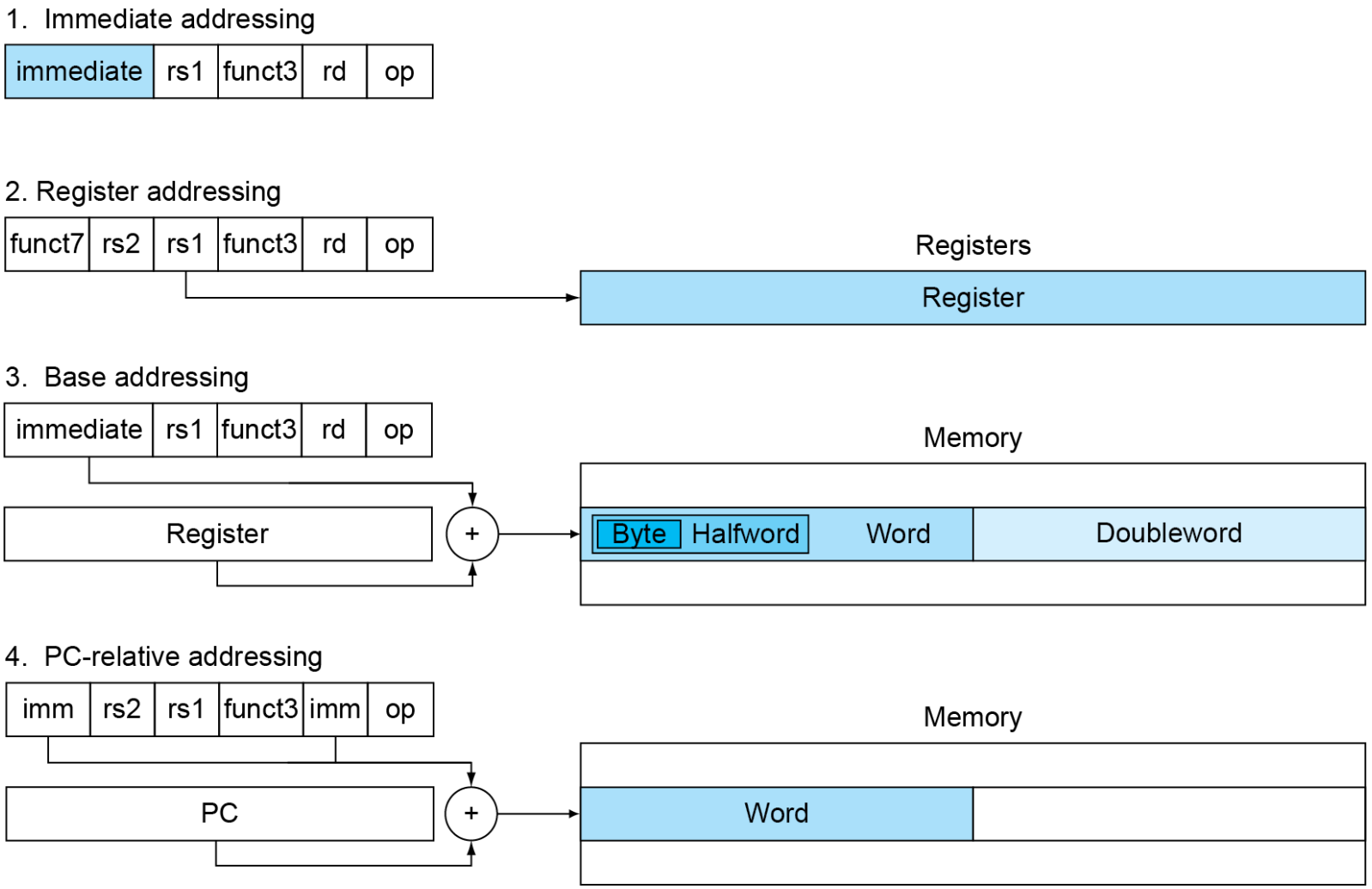
D. U

E. UJ



	31 27	26 25	24 20	19 15	14 12	11 7	6 0
R	funct7		rs2	rs1	funct3	rd	Opcode
I	imm[11:0]			rs1	funct3	rd	Opcode
S	imm[11:5]		rs2	rs1	funct3	imm[4:0]	opcode
SB	imm[12|10:5]		rs2	rs1	funct3	imm[4:1|11]	opcode
U	imm[31:12]					rd	opcode
UJ	imm[20|10:1|11|19:12]					rd	opcode

Addressing Mode Summary



Use of RISC-V instructions

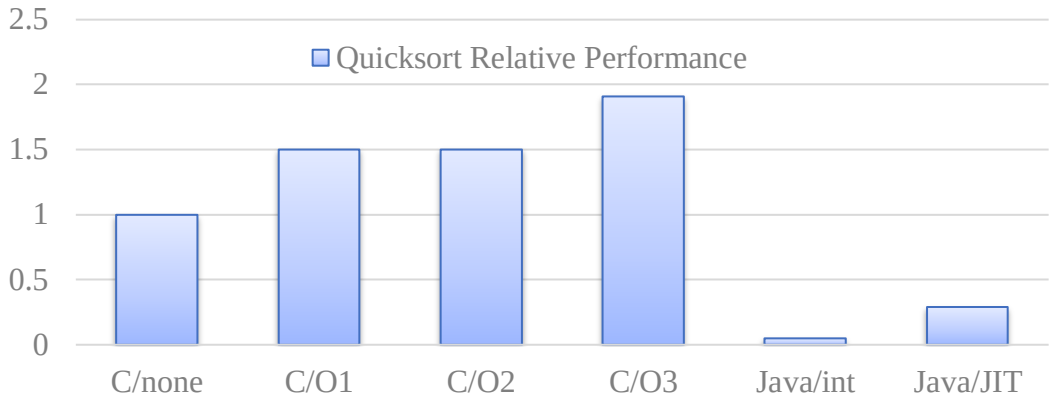
- Measure RISC-V instruction executed in SPEC CPU2006 CPU benchmarks
  - Consider making the common case fast
  - Consider compromises

Instruction class	RISC-V examples	SPEC2006 Int	SPEC2006 FP
Arithmetic	`add, sub, addi`	16%	48%
Data transfer	`lw, sw, lb, sb, lh, sh, lui`	35%	36%
Logical	`and, or, xor, sll, srl, sra`	12%	4%
Branch	`beq, bne, blt, bge, bltu, bgeu`	34%	8%
Jump	`Jal, jalr`	2%	0%

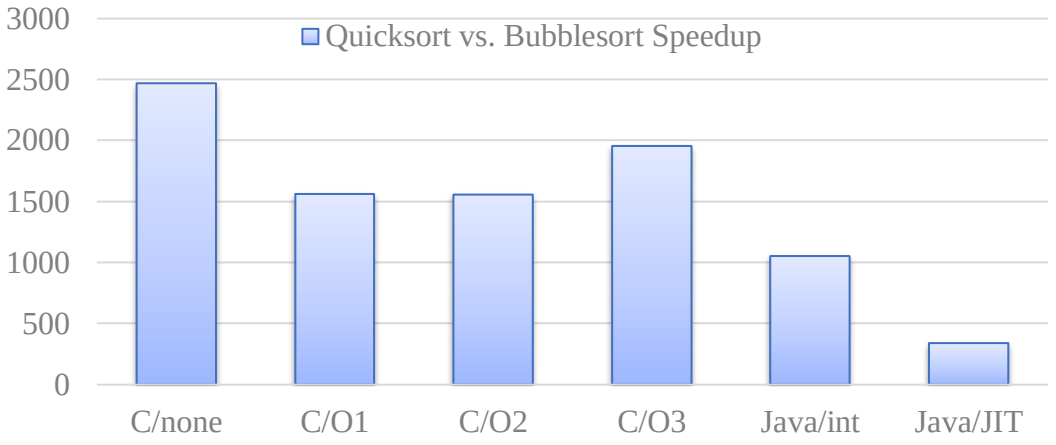
Figure 2.48

Effect of Language and Algorithm

Quicksort
in different languages



Quicksort vs Bubblesort



Nothing can fix a dumb algorithm!

RISC-V ISA

- RISC-V is a typical of RISC ISAs
  - ARM and MIPS are also in this category
- x86 in Intel's processors is CISC
- Design principles in RISC-V
  1. Simplicity favors regularity
  2. Smaller is faster
  3. Make the common case fast
  4. Good design demands good compromises

RISC: Reduced Instruction Set Computer

CISC: Complex Instruction Set Computer

Fallacies

- Powerful instruction $\Rightarrow$ higher performance
  - Fewer instructions required
  - But complex instructions are hard to implement
    - May slow down all instructions, including simple ones
  - Compilers are good at making fast code from simple instructions

Fallacies

- Use assembly code for high performance
  - Modern compilers are better at dealing with modern processors
  - More lines of code $\Rightarrow$ more errors and less productivity
  - Hard to maintain

Question

- Which of the following ISAs is of a different type from other two?

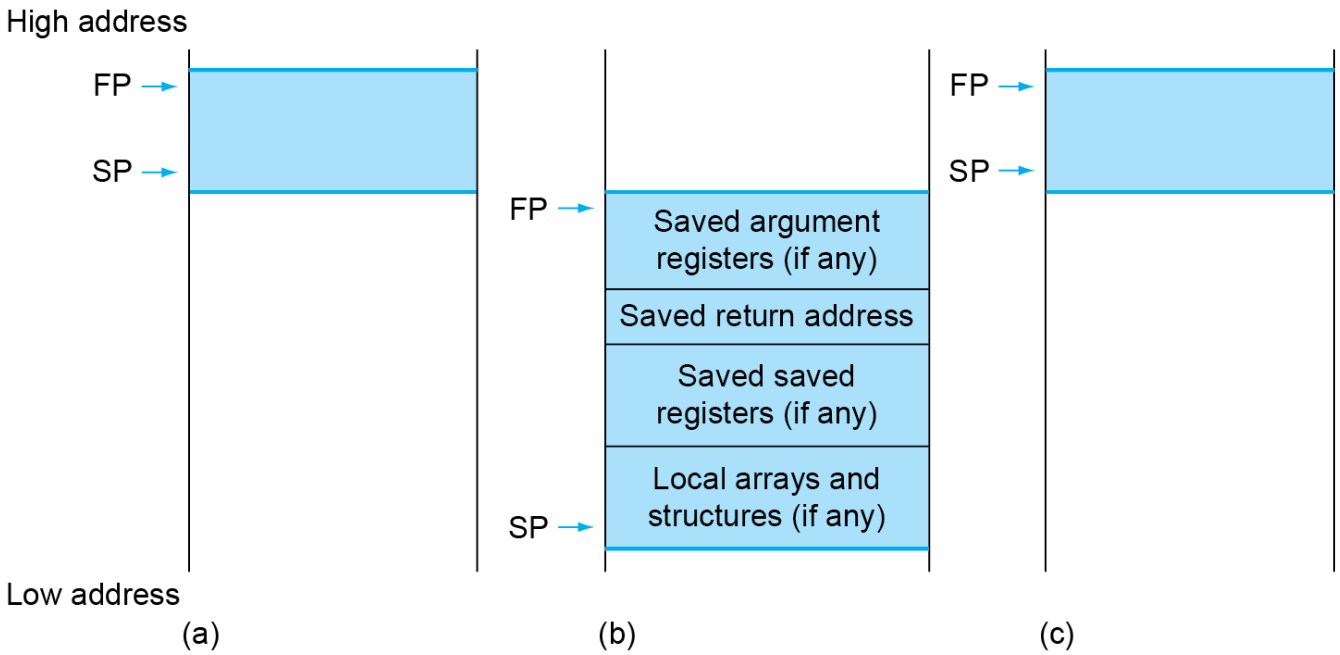
RISC-V, ARM, x86

A. RISC-V
B. ARM
C. x86

Study the remaining slides yourself

Frame pointer

- Every function has a frame: procedure frame/activation record
- fp points to a fixed location in the procedure frame
  - sp may change within the function, but fp does not
  - Use fp as the base register for local variables



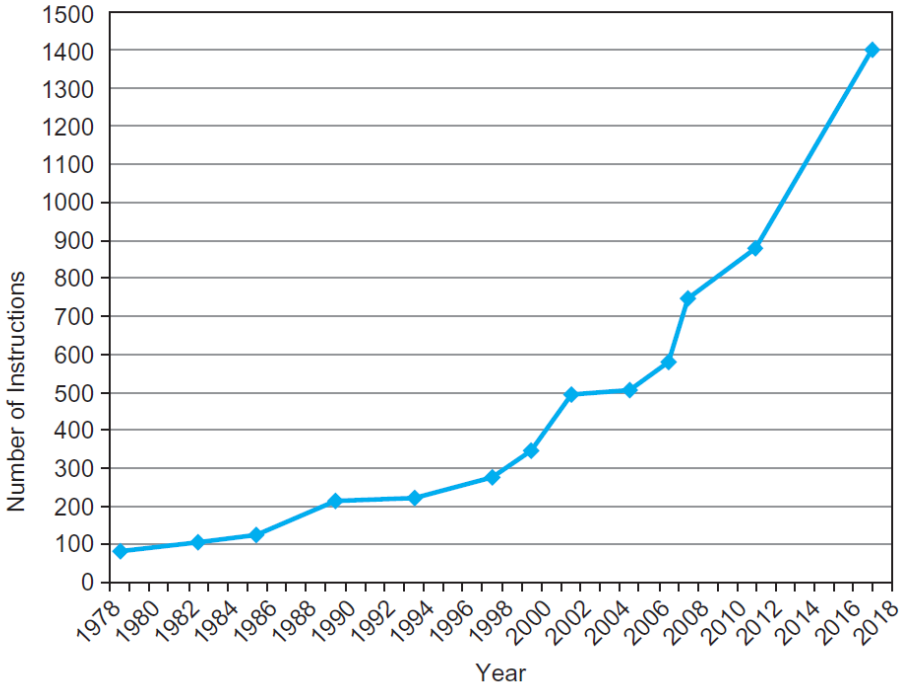
Example: set and restore frame pointer

```
addi    sp, sp, -4          # push fp
sw      fp, (sp)
addi    fp, sp, 0           # set fp

# function body is here
# use fp as the base register, e.g., -40(fp)
# sp may change, but fp does not


# before return, restore fp, and then sp
lw      fp, (sp)
addi    sp, sp, 4
ret
```

An example of compiled code, where fp is set to the old sp
cse3666/add.md at master · zhijieshi/cse3666 (github.com)

Fallacies

- Backward compatibility ⇒ instruction set doesn't change
  - But they do accrete more instructions



x86 instruction set

Saving comparison results to a register

- Save the comparison result in a register
  - Can be used in combination with `beq, bne`

```
# if (rs1 < rs2) rd = 1; else rd = 0
slt   rd, rs1, rs2    # signed
sltu  rd, rs1, rs2    # unsigned


# compare with immediate
# if (rs1 < immd) rd = 1; else rd = 0
slti  rd, rs1, immd   # signed
sltiu rd, rs1, immd   # unsigned
```

immd is 12 bits long and sign extended, even for unsigned comparison!

Example

```
# pseudoinstruction seqz
seqz  t0, t1        # t0 = (t1 == 0)



# t0 = (s1 >= '0') && (s1 <= '9')
t0 = is_digit(s1)
```

Example

```
# pseudoinstruction seqz
seqz  t0, t1        # t0 = (t1 == 0)

sltui  t0, t1, 1

# t0 = (s1 >= '0') && (s1 <= '9')
t0 = is_digit(s1)

addi  t0, s1, -48       # '0' is 48
sltui t0, t0, 10
```