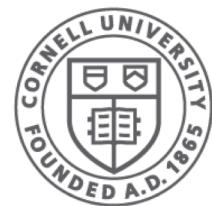


# RISC, CISC, and ISA Variations

CS 3410  
Computer System Organization & Programming



**Cornell CIS**  
COMPUTING AND INFORMATION SCIENCE

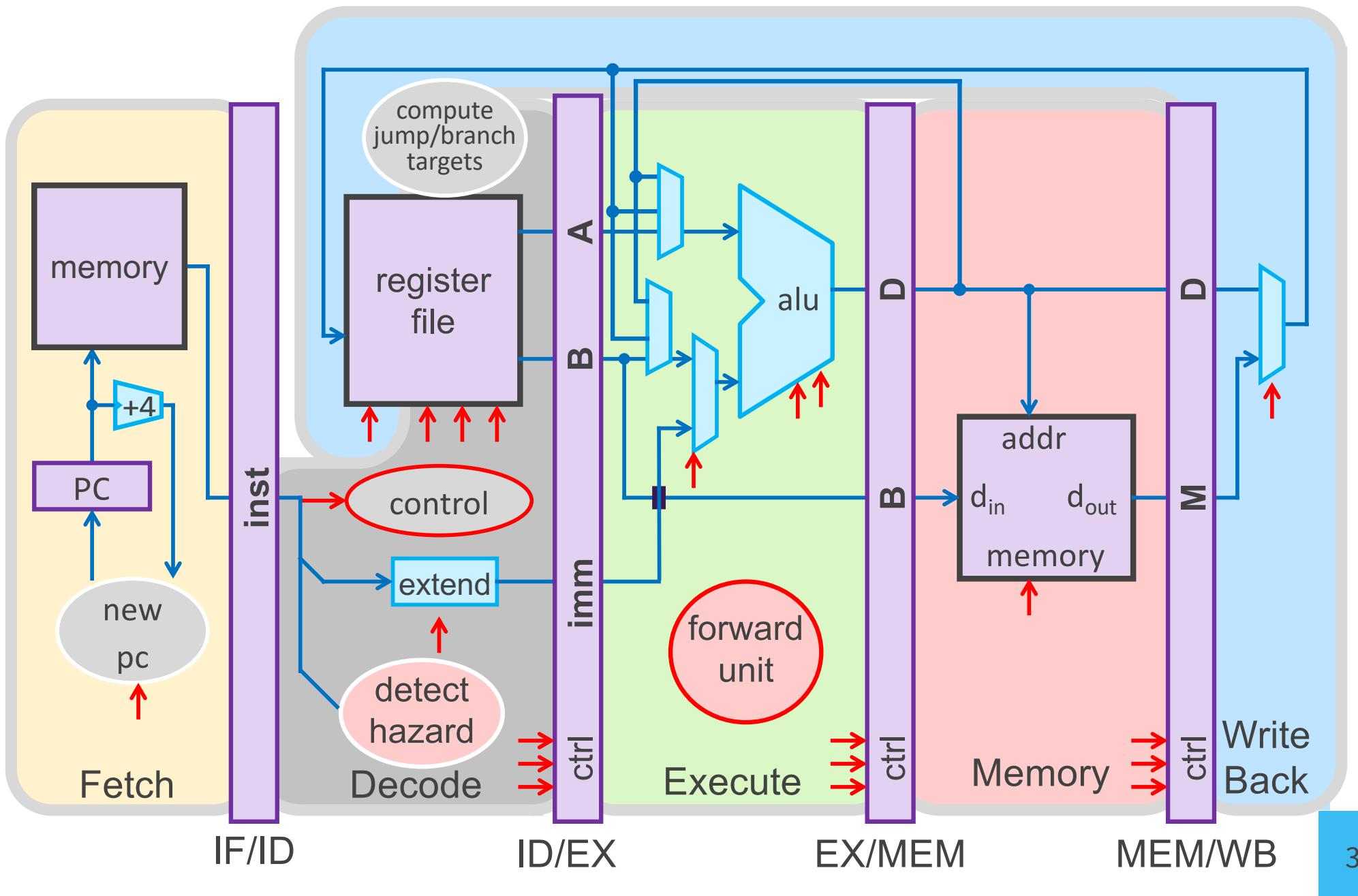
These slides are the product of many rounds of teaching CS 3410 by Professors Weatherspoon, Bala, Bracy, and Sirer.

# iClicker Question (first alone, then pairs)

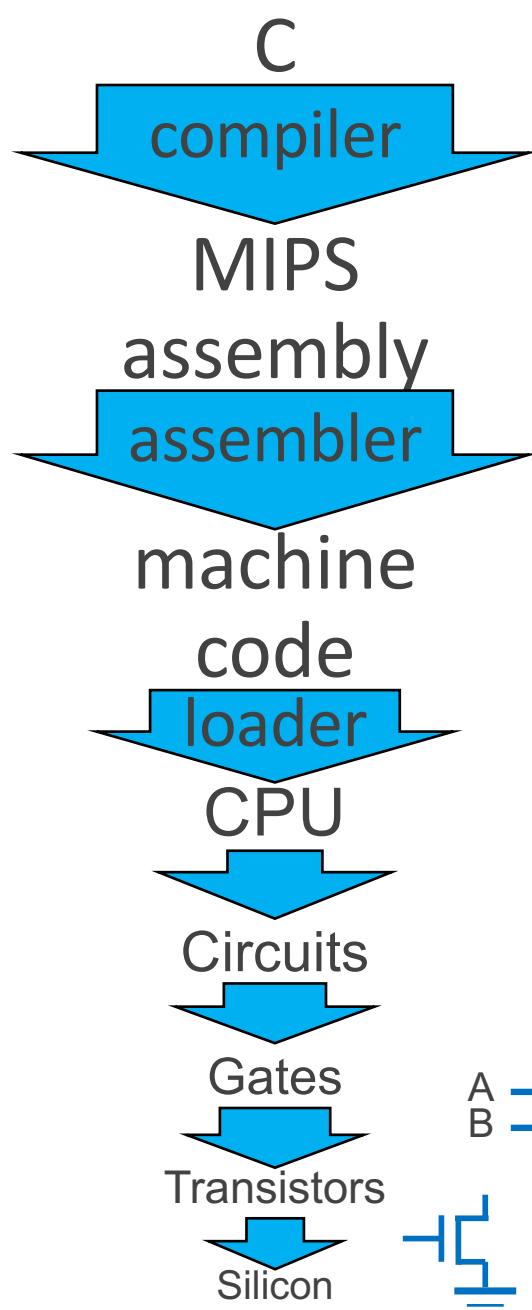
Which is **not** considered part of the ISA?

- A. There is a control delay slot.
- B. The number of inputs each instruction can have.
- C. Load-use stalls will **not** be detected by the processor.
- D. The number of cycles it takes to execute a multiply.
- E. Each instruction is encoded in 32 bits.

# Big Picture: Where are we now?



# Big Picture: Where are we going?

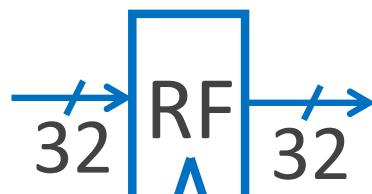
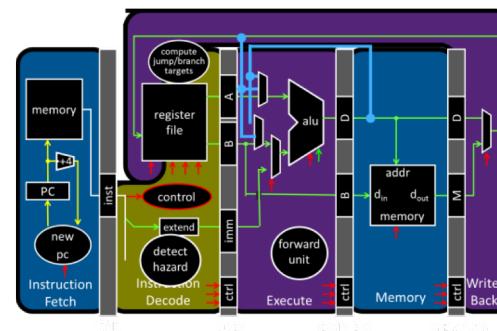


```
int x = 10;  
x = x + 15;
```

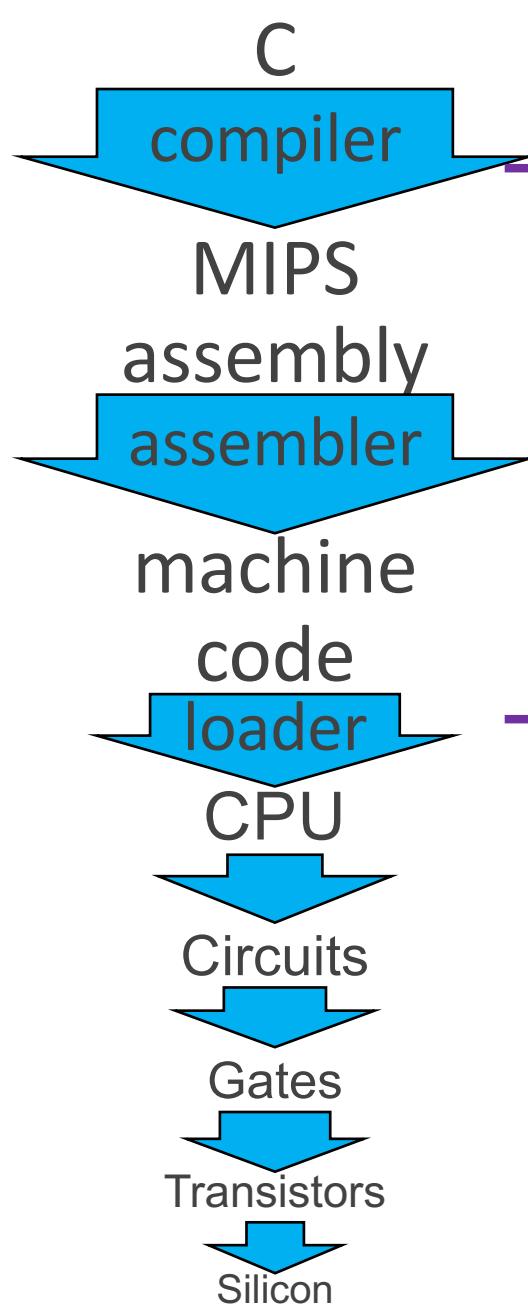
```
addi r5, r0, 10  
addi r5, r5, 15
```

$$\begin{aligned}r0 &= 0 \\r5 &= r0 + 10 \\r5 &= r15 + 15\end{aligned}$$

addi	r0	r5	10
00100000000000101	0000000000000001010	00000000000000001010	
00100000101001010	00000000000000001111		



# Big Picture: Where are we going?



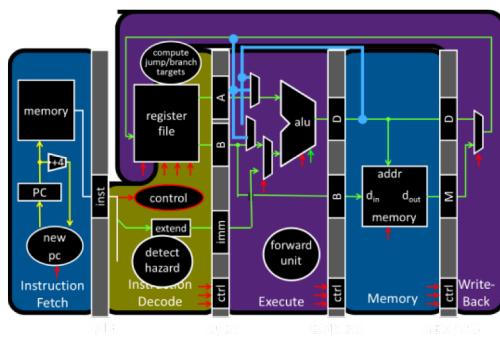
int x = 10;  
x = 2 \* x + 15;

addi r5, r0, 10  
muli r5, r5, 2  
addi r5, r5, 15

0010000000001010000000000001010  
00000000000010100101000100000  
0010000101001010000000000001111

High Level Languages

Instruction Set Architecture (ISA)



# Goals for Today

## Instruction Set Architectures

- ISA Variations, and CISC vs RISC
- Peek inside some other ISAs:
  - X86
  - ARM

# Instruction Set Architecture (ISA)

Different CPU architectures specify different instructions

Two classes of ISAs

- Reduced Instruction Set Computers (RISC)  
IBM Power PC, Sun Sparc, MIPS, Alpha
- Complex Instruction Set Computers (CISC)  
Intel x86, PDP-11, VAX
- Another ISA classification: Load/Store Architecture
  - Data must be in registers to be operated on  
For example:  $\text{array}[x] = \text{array}[y] + \text{array}[z]$   
1 add ?      OR      2 loads, an add, and a store ?
  - Keeps HW simple → many RISC ISAs are load/store

# iClicker Question

What does it mean for an architecture to be called a load/store architecture?

- (A) Load and Store instructions are supported by the ISA.
- (B) Load and Store instructions can also perform arithmetic instructions on data in memory.
- (C) Loads & Stores are the primary means of reading and writing data in the ISA.
- (D) Data must first be loaded into a register before it can be operated on.
- (E) Every load must have an accompanying store at some later point in the program.

# ISA Variations

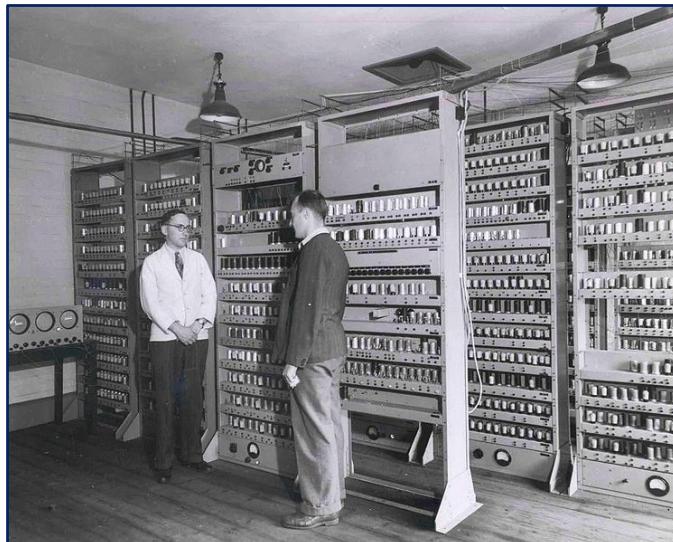
ISA defines the permissible instructions

- **MIPS**: load/store, arithmetic, control flow, ...
- **ARMv7**: similar to MIPS, but more shift, memory, & conditional ops
- **ARMv8 (64-bit)**: even closer to MIPS, no conditional ops
- **VAX**: arithmetic on memory or registers, strings, polynomial evaluation, stacks/queues, ...
- **Cray**: vector operations, ...
- **x86**: a little of everything

# Brief Historical Perspective on ISAs

## Accumulators

- Early computers had **one** register!



EDSAC (Electronic Delay Storage Automatic Calculator) in 1949



Intel 8008 in 1972

- Two registers short of a MIPS instruction!
- Requires memory-based addressing mode
  - Example: add 200 // ACC = ACC + Mem[200]

# Brief Historical Perspective on ISAs

## Next step: More Registers

- Dedicated registers
  - separate accumulators for mult/div instructions
- General-purpose registers
  - Registers can be used for any purpose
  - MIPS, ARM, x86
- *Register-memory* architectures
  - One operand may be in memory (e.g. accumulators)
  - x86 (i.e. 80386 processors)
- *Register-register* architectures (**aka load-store**)
  - All operands **must** be in registers
  - MIPS, ARM

# ISAs are a product of current technology

Machine	# General Purpose Registers	Architectural Style	Year
EDSAC	1	Accumulator	1949
IBM 701	1	Accumulator	1953
CDC 6600	8	Load-Store	1963
IBM 360	18	Register-Memory	1964
DEC PDP-8	1	Accumulator	1965
DEC PDP-11	8	Register-Memory	1970
Intel 8008	1	Accumulator	1972
Motorola 6800	2	Accumulator	1974
DEC VAX	16	Register-Memory, Memory-Memory	1977
Intel 8086	1	Extended Accumulator	1978
Motorola 6800	16	Register-Memory	1980
Intel 80386	8	Register-Memory	1985
ARM	16	Load-Store	1985
MIPS	32	Load-Store	1985
HP PA-RISC	32	Load-Store	1986
SPARC	32	Load-Store	1987
PowerPC	32	Load-Store	1992
DEC Alpha	32	Load-Store	1992
HP/Intel IA-64	128	Load-Store	2001
AMD64 (EMT64)	16	Register-Memory	2003

# In the Beginning...

People programmed in assembly and machine code!

- Needed as many addressing modes as possible
- Memory was (and still is) slow

CPUs had relatively few registers

- Register's were more “expensive” than external mem
- Large number of registers requires many bits to index

Memories were small

- Encouraged highly encoded microcodes as instructions
- Variable length instructions, load/store, conditions, etc

# Reduced Instruction Set Computer (RISC)

## John Cock

- IBM 801, 1980 (started in 1975)
- Name 801 came from the bldg that housed the project
- Idea: Can make a very small and very fast core
- Known as “the father of RISC Architecture”
- Turing Award and National Medal of Science



# Reduced Instruction Set Computer (RISC)

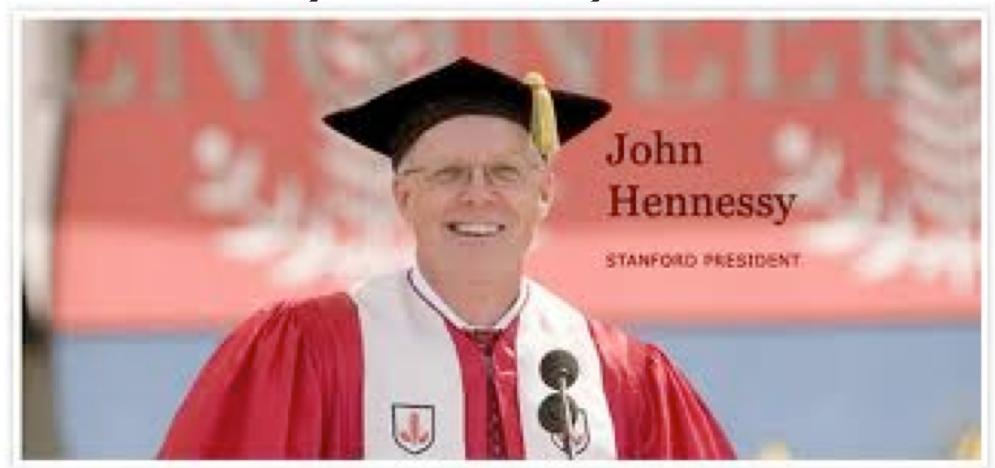
## Dave Patterson

- RISC Project, 1982
- UC Berkeley
- RISC-I:  $\frac{1}{2}$  transistors & 3x faster
- Influences: Sun SPARC, namesake of industry



## John L. Hennessy

- MIPS, 1981
- Stanford
- Simple, *full* pipeline
- Influences: MIPS computer system, PlayStation, Nintendo



# RISC vs. CISC

MIPS = Reduced Instruction Set Computer (RISC)

- $\approx 200$  instructions, 32 bits each, 3 formats
- all operands in registers
  - almost all are 32 bits each
- $\approx 1$  addressing mode:  $\text{Mem}[\text{reg} + \text{imm}]$

x86 = Complex Instruction Set Computer (CISC)

- $> 1000$  insns, 1-15 bytes each (*dozens of add insns*)
- operands in dedicated registers, general purpose registers, memory, on stack, ...
  - can be 1, 2, 4, 8 bytes, signed or unsigned
- 10s of addressing modes
  - e.g.  $\text{Mem}[\text{segment} + \text{reg} + \text{reg}^*\text{scale} + \text{offset}]$

# The RISC Tenets

## RISC

- Single-cycle execution
- Hardwired control
- Load/store architecture
- Few memory addressing modes
- Fixed-length insn format
- Reliance on compiler optimizations
- Many registers (compilers are better at using them)

## CISC

- many multicycle operations
- microcoded multi-cycle operations
- register-mem and mem-mem
- many modes
- many formats and lengths
- hand assemble to get good performance
- few registers

# RISC vs CISC

## RISC Philosophy

Regularity & simplicity  
Leaner means faster  
Optimize common case

Energy efficiency  
Embedded Systems  
Phones/Tablets

## CISC Rebuttal

Compilers can be smart  
Transistors are plentiful  
Legacy is important  
Code size counts  
Micro-code!  
“RISC Inside”

Desktops/Servers

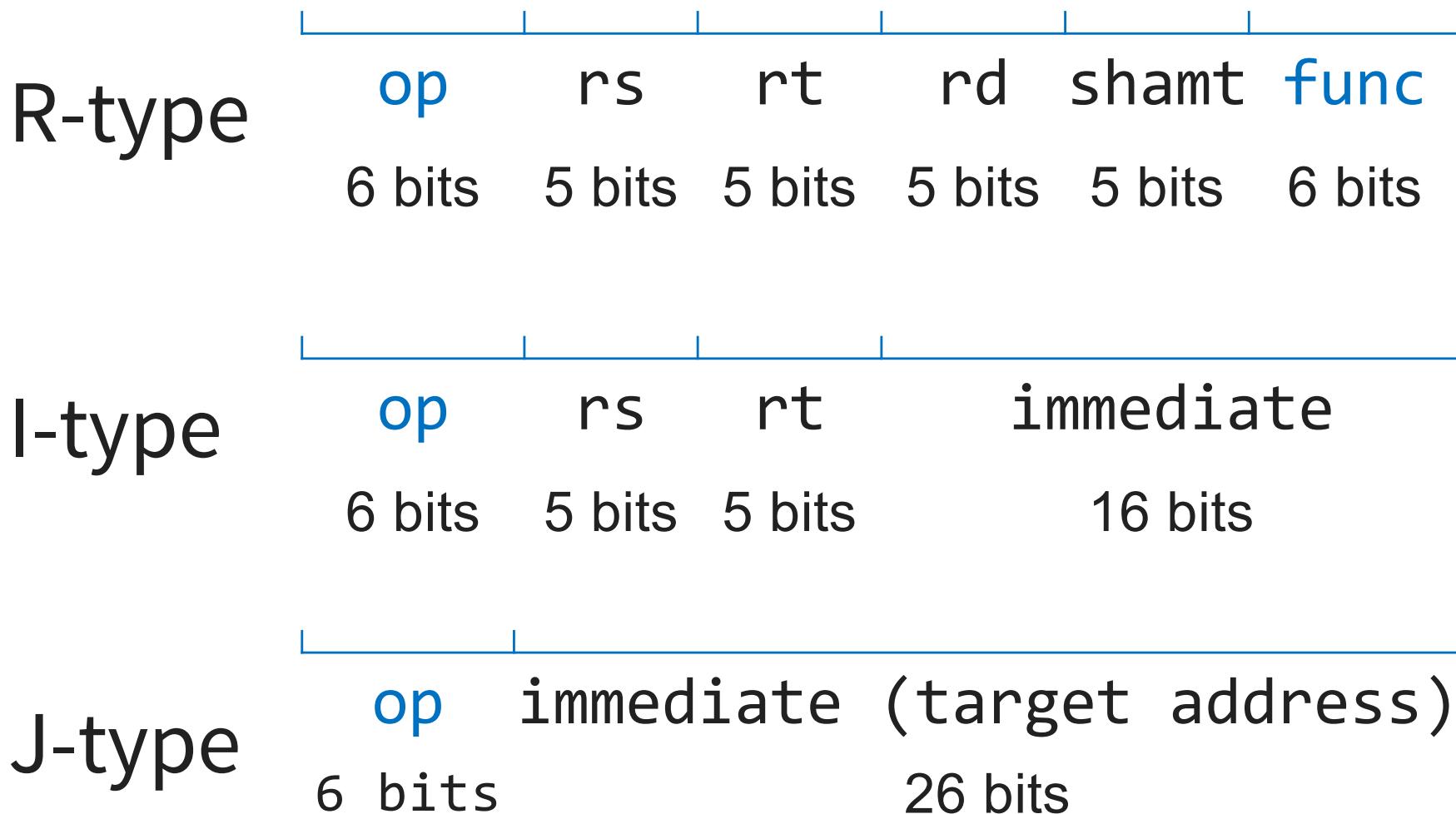
# iClicker Question (alone first, then pairs)

What is one advantage of a CISC ISA?

- A. It naturally supports a faster clock.
- B. Instructions are easier to decode.
- C. The static footprint of the code will be smaller.
- D. The code is easier for a compiler to optimize.
- E. You have a lot of registers to use.

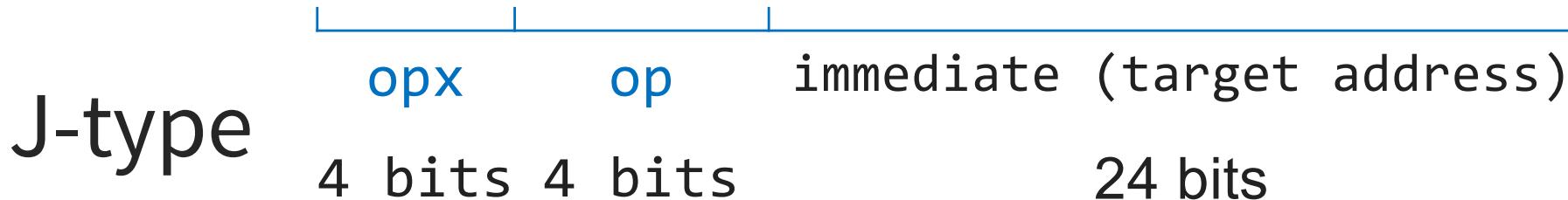
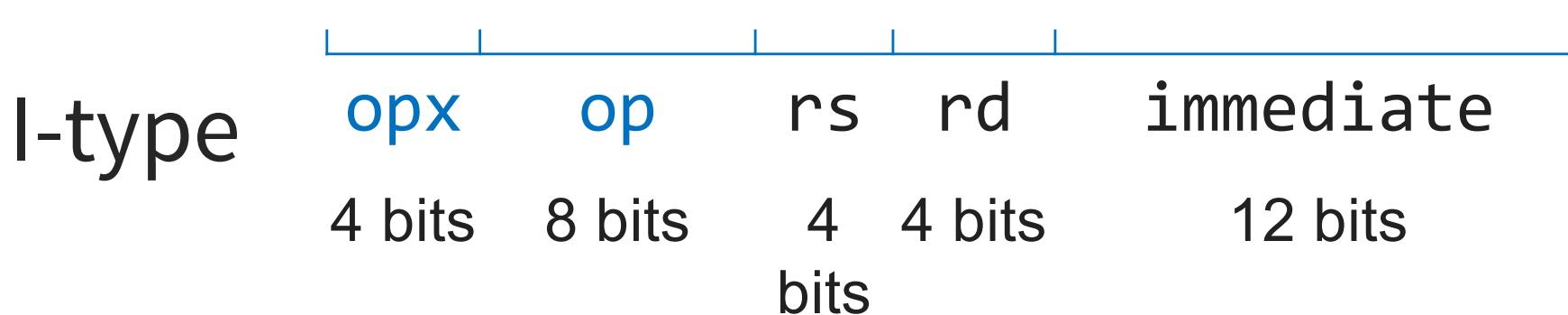
# MIPS instruction formats

All MIPS instructions are 32 bits long, 3 formats



# ARMv7 instruction formats

All ARMv7 instructions are 32 bits long, 3 formats



# MIPS Control Dependence

```
while(i != j) {  
    if (i > j)  
        i -= j;  
    else  
        j -= i;  
}
```

```
Loop: BEQ Ri, Rj, End // if "NE" (not equal), stay in loop  
      SLT Rd, Rj, Ri // (i > j) → Rd=1, (i ≤ j) → Rd = 0  
      BEQ Rd, R0, Else // Rd == 0 means (i ≤ j) → Else  
      SUB Ri, Ri, Rj // i = i-j;  
  
      J Loop  
  
Else:   SUB Rj, Rj, Ri // j = j-i;  
  
      J Loop  
  
End:
```

In MIPS, performance suffers if code has a lot of branches

3 NOP injections due to delay slot

# ARMv7 Conditional Instructions

```
while(i != j) {  
    if (i > j)  
        i -= j;  
    else  
        j -= i;  
}
```

Loop: CMP Ri, Rj

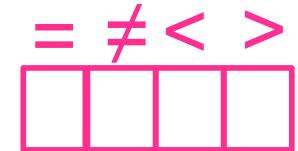
SUBGT Ri, Ri, Rj

SUBLE Rj, Rj, Ri

BNE loop

ARM: avoids delays with conditional instructions

New: 1-bit condition  
registers (CR)



// set condition registers

// Example: 4, 3 → CR = 0101

// 5,5 → CR = 1000

// i = i-j only if CR & 0001 != 0

// j = j-i only if CR & 1010 != 0000

// if "NE" (not equal), then loop

**Control Independence!**

# ARMv7: Other Cool operations

Shift one register (e.g., Rc) any amount

Add to another register (e.g., Rb)

Store result in a different register (e.g. Ra)

ADD Ra, Rb, Rc LSL #4

Ra = Rb + Rc << 4

Ra = Rb + Rc x 16

# ARMv7 Instruction Set Architecture

ARMv7 instructions are 32 bits long, 3 formats

## Reduced Instruction Set Computer (RISC) properties

- Only Load/Store instructions access memory
- Instructions operate on operands in processor registers
- 16 registers

## Complex Instruction Set Computer (CISC) properties

- Autoincrement, autodecrement, PC-relative addressing
- Conditional execution
- Multiple words can be accessed from memory with a single instruction (SIMD: single instr multiple data)

# ARMv8 (64-bit) Instruction Set Architecture

ARMv8 instructions are **64 bits long**, 3 formats

**Reduced Instruction Set Computer (RISC) properties**

- Only Load/Store instructions access memory
- Instructions operate on operands in processor registers
- **32 registers** and r0 is always 0

**Complex Instruction Set Computer (CISC) properties**

- Conditional execution
- Multiple words can be accessed from memory with a single instruction (SIMD: single instr multiple data)

# ISA Takeaways

The number of available registers greatly influenced the instruction set architecture (ISA)

**Complex Instruction Set Computers** were very complex

- + Small # of insns necessary to fit program into memory.
- greatly increased the complexity of the ISA as well.

Back in the day... CISC was necessary because everybody programmed in assembly and machine code! Today, CISC ISA's are still dominant due to the prevalence of x86 ISA processors. However, RISC ISA's today such as ARM have an ever increasing market share (of our everyday life!).  
ARM borrows a bit from both RISC and CISC.