RISC, CISC, and ISA Variations

CS 3410

Computer System Organization & Programming

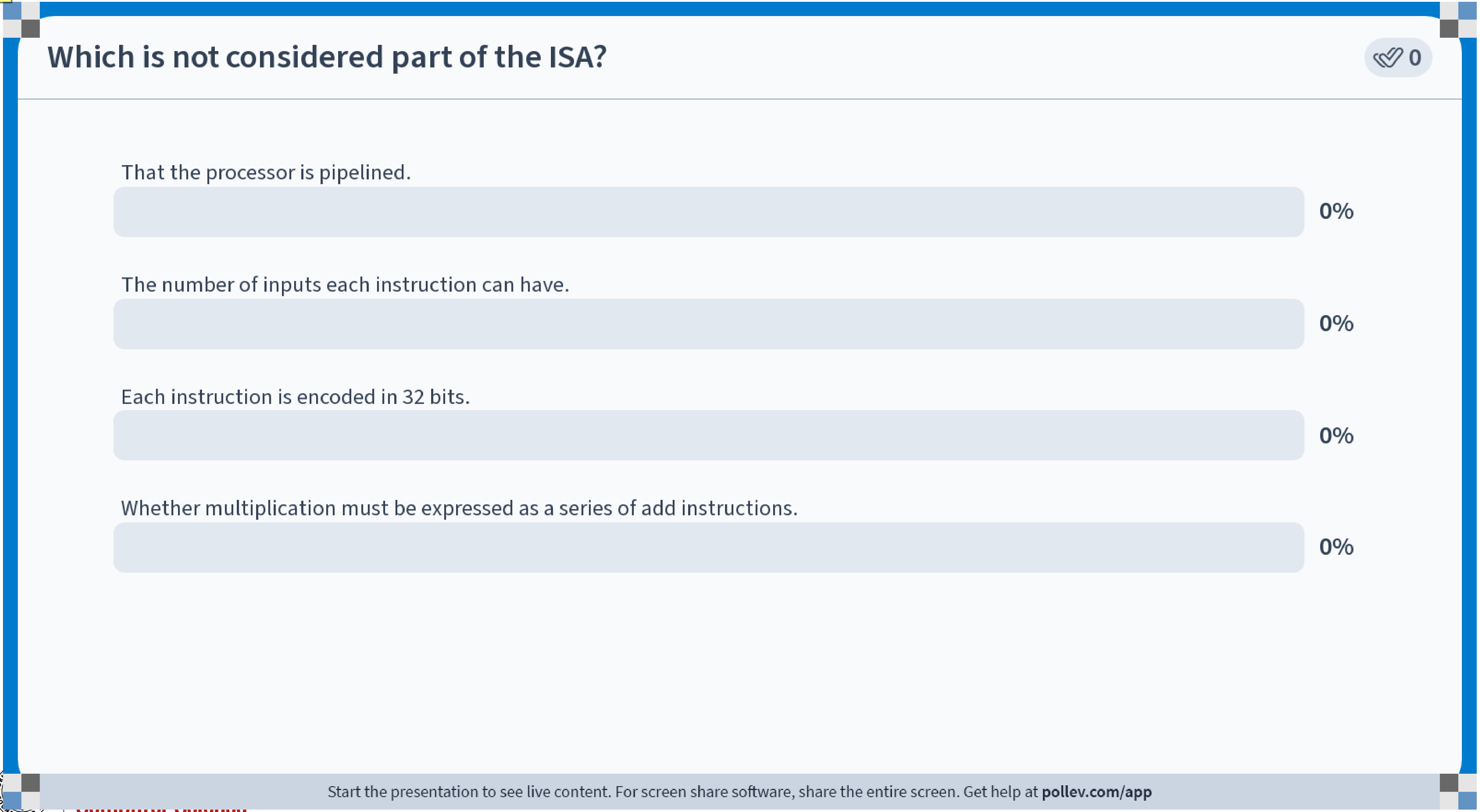
Cornell Bowers CIS
Computer Science

[K. Bala, A. Bracy, E. Sirer, and H. Weatherspoon]

PollEV Question #1

Which is not considered part of the ISA?

A. That the processor is pipelined.
B. The number of inputs each instruction can have.
C. Each instruction is encoded in 32 bits.
D. Whether multiplication must be expressed as a series of add instructions.



Which is not considered part of the ISA?

✓ 0

That the processor is pipelined. 0%

The number of inputs each instruction can have. 0%

Each instruction is encoded in 32 bits. 0%

Whether multiplication must be expressed as a series of add instructions. 0%

PollEV Question #1

Which is not considered part of the ISA?

[✓ A. That the processor is pipelined.]
B. The number of inputs each instruction can have.
C. Each instruction is encoded in 32 bits.
D. Whether multiplication must be expressed as a series of add instructions.

Big Picture: How to ~~Design~~ Program a Processor

C

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

High Level Languages

compiler

RISC-V assembly language

```
addi x5, x0, 10
slli x5, x5, 1
addi x5, x5, 15
```

x0 = 0

x5 = x0 + 10

x5 = x5 * 2

x5 = x5 + 15

Instruction Set Architecture (ISA)

assembler

RISC-V machine language

10	x0		x5	op = addi
000000001010	00000	000	00101	0010011

```
00000000000100101001001010011111
```

000000001111	00101	000	00101	0010011
15	x5		x5	op = addi

EVERYTHING IS A NUMBER!

Cornell Bowers CIS
Computer Science

Big Picture: How to ~~Design~~ Program a Processor

C

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

x0 = 0

compiler

RISC-V assembly language

```
addi x5, x0, 10   ← x5 = x0 + 10
slli x5, x5, 1    ← x5 = x5 * 2
addi x5, x5, 15   ← x5 = x5 + 15
```

assembler

RISC-V machine language

```
10            x0        x5    op = addi
000000001010 00000 000 00101 0010011
00000000000100101001001010011111
000000001111 00101 000 00101 0010011
15            x5        x5    op = addi
```

EVERYTHING IS A NUMBER!

Goals for Today

Instruction Set Architectures

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

Iron Law of Processor Performance

How to make *a processor* that runs *programs faster*?

$$\frac{time}{program} = \frac{instructions}{program} * \frac{cycles}{instruction} * \frac{time}{cycle}$$

(CPI)

Clock Frequency

Pipelining/Performance Lecture: tradeoff CPI and clock frequency

This lecture: incorporating instruction count into the equation

Instruction Set Architecture (ISA)

Different CPU architectures specify different instructions

Two classes of ISAs

- Reduced Instruction Set Computers (RISC)
  RISC-V, MIPS, IBM Power PC, Sun Sparc, 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: array[x] = array[y] + array[z]
    1 add ?   OR   2 loads, an add, and a store ?
  - Keeps HW simple → many RISC ISAs are load/store

PollEV Question #2

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.

PollEV Question #2

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

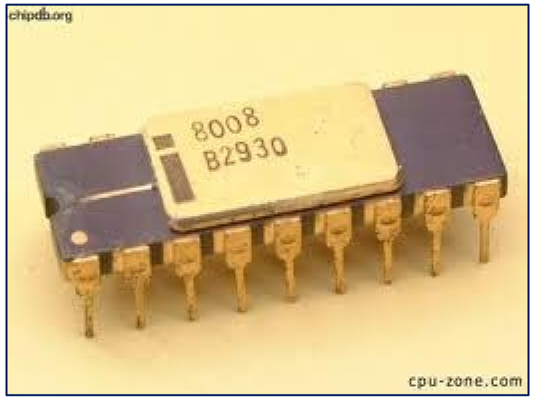
- RISC-V/MIPS: load/store, arithmetic, control flow, …
- ARMv7: similar to MIPS, but more shift, memory, & conditional ops
- ARMv8 (64-bit): even closer to MIPS, no conditional ops
- VAX: arithmetic on memory or registers, strings, polynomial evaluation, stacks/queues, …
- Cray: vector operations, …
- 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]
    - Add the accumulator to the word in memory at address 200
    - Place the sum back in the accumulator

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
  - RISC-V, 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
  - RISC-V, MIPS, ARM

ISAs are a product of current technology

- # of available registers plays huge role in ISA design

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

Takeaway

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

Complex Instruction Set Computers necessary but were very complex

- Necessary to reduce the number of instructions required to fit a program into memory.
- However, also greatly increased the complexity of the ISA as well.

Next Goal

How do we reduce the complexity of the ISA while maintaining or increasing performance?

Reduced Instruction Set Computer (RISC)

John Cock

- IBM 801, 1980 (started in 1975)
  - Name 801 came from the building 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

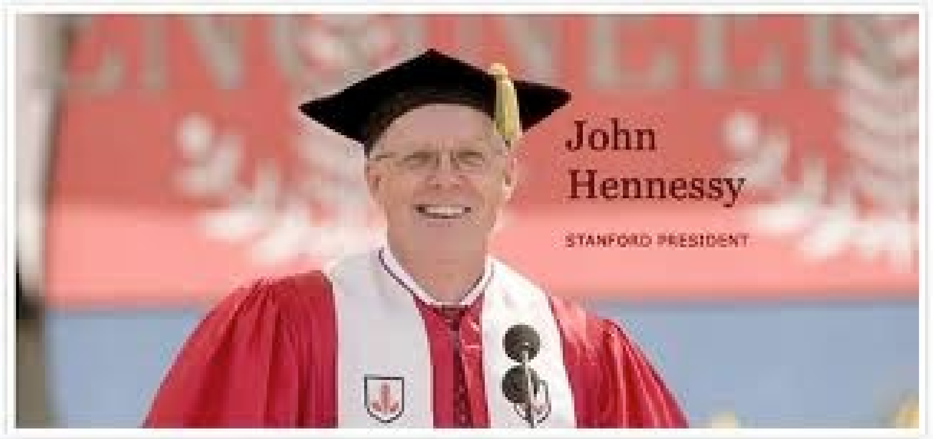
Reduced Instruction Set Computer (RISC)

Dave Patterson

- RISC Project, 1982
- UC Berkeley
- RISC-I: ½ 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

RISC-V = Reduced Instruction Set Computer (RISC)

- ≈ 200 instructions, $\boxed{\text{32 bits each, 4 formats}}$
- all operands in registers
  - almost all are 32 bits each
- ≈ 1 addressing mode: Mem[reg + 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. Mem[segment + reg + reg*scale + offset]

Cornell Bowers CIS
Computer Science

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!

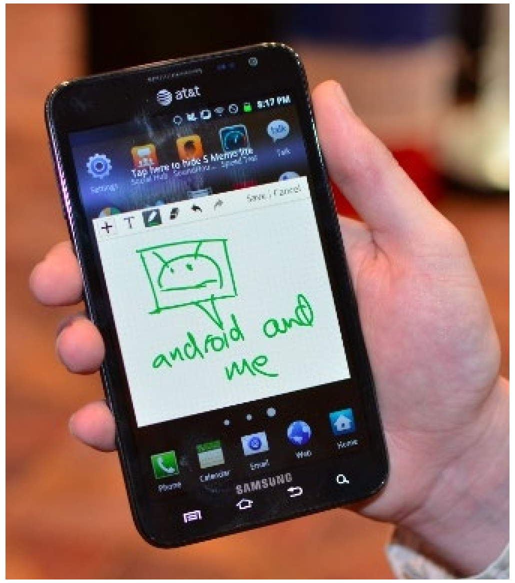
“RISC Inside”

Desktops/Servers

ARMDroid vs WinTel

Android OS on ARM processor

Windows OS on Intel (x86) processor

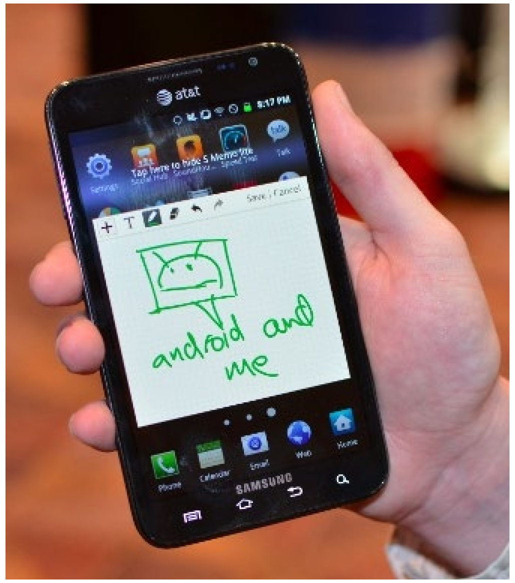


ARMDroid vs WinTel vs MacBook

Android OS on ARM processor

Windows OS on Intel (x86) processor

Mac OS X on M* processor

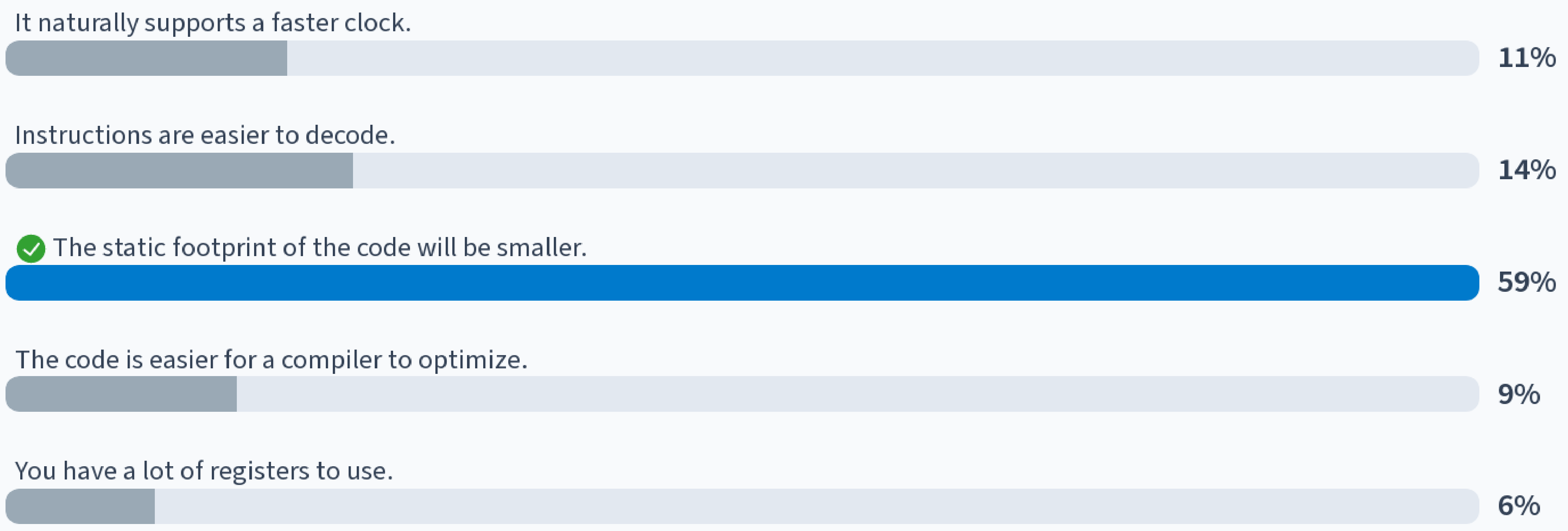


PollEV Question #3

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.

What is one advantage of a CISC ISA?



PollEV Question #3

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.

Takeaway

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

Complex Instruction Set Computers were very complex

- Necessary to reduce the number of instructions required to fit a program into memory.
- However, also 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.

Next Goal

How does RISC-V and ARM compare to each other?

RISC-V instructions

32 bits long and 4 possible formats:

R-type

funct7	rs2	rs1	funct3	rd	op
7 bits	5 bits	5 bits	3 bits	5 bits	7 bits

I-type

imm	rs1	funct3	rd	op
12 bits	5 bits	3 bits	5 bits	7 bits

S-type

imm	rs2	rs1	funct3	imm	op
7 bits	5 bits	5 bits	3 bits	5 bits	7 bits

U-type

imm	rd	op
20 bits	5 bits	7 bits

ARMv7 instruction formats

All ARMv7 instructions are 32 bits long, 3 formats

R-type

opx	op	rs	rd	opx	rt
4 bits	8 bits	4 bits	4 bits	8 bits	4 bits

I-type

opx	op	rs	rd	immediate
4 bits	8 bits	4 bits	4 bits	12 bits

J-type

opx	op	immediate (target address)
4 bits	4 bits	24 bits

ARMv7 Conditional Instructions

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

ARM: avoids delays with conditional instructions

New: 1-bit condition registers (CR)

= ≠ < >

```
Loop: CMP Ri, Rj
```

// set condition registers

// Example: 4, 3 → CR =
0101

// 5,5 → CR = 1000

```
      SUBGT Ri, Ri, Rj
```

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

```
      SUBLE Rj, Rj, Ri
```

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

```
      BNE loop
```

// 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.

PollEV Answers

1. A (That the processor is pipelined.)
2. D (Data must first be loaded into a register before it can be operated on.)
3. C (The static footprint of the code will be smaller.)