

# RISC, CISC, and ISA Variations

**Hakim Weatherspoon**

**CS 3410**

Computer Science

Cornell University

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

# Announcements

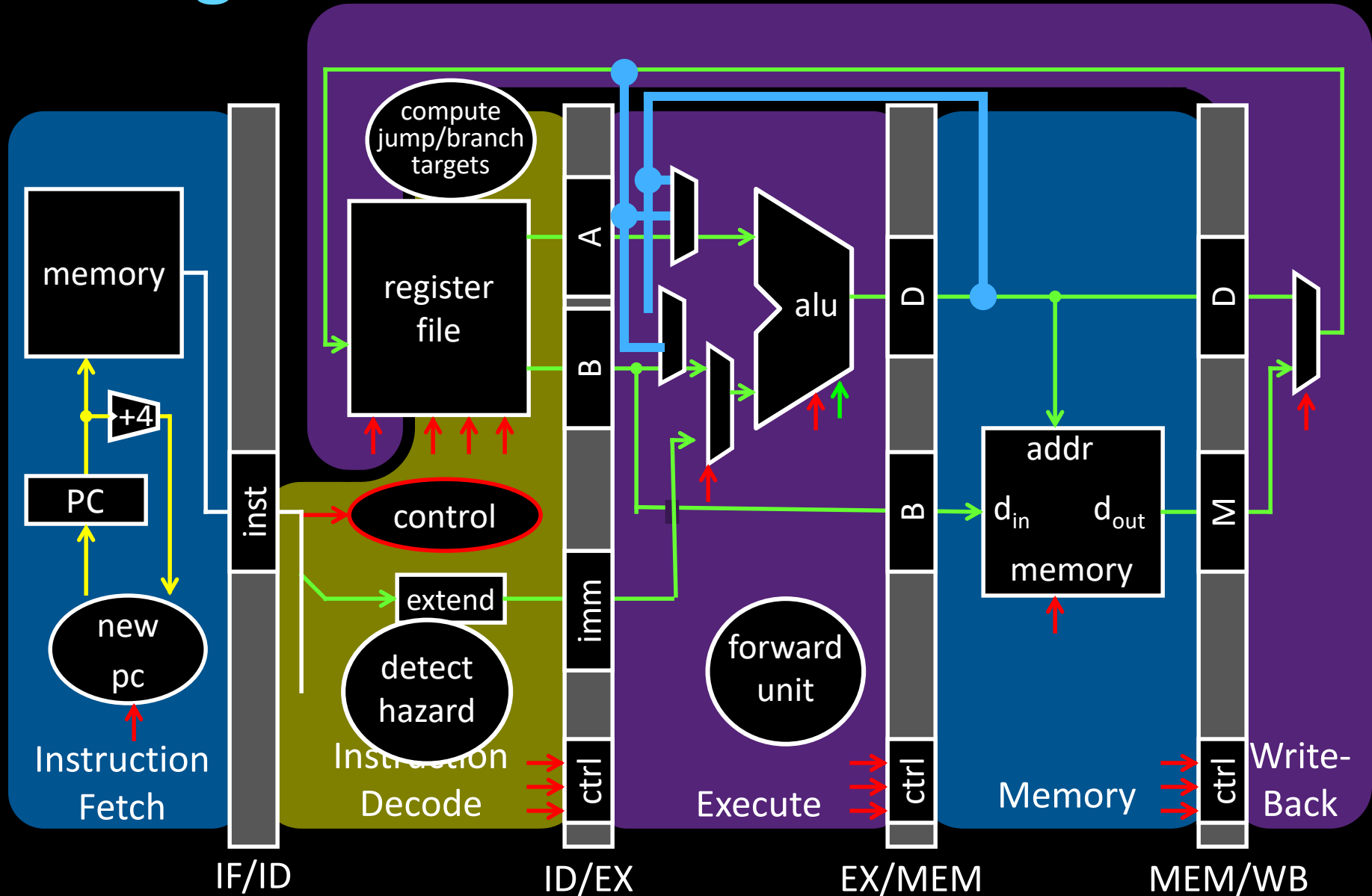
## *Prelim today*

Starts at **7:30pm** sharp

Go to location based on netid

Find locations on piazza

# Big Picture: Where are we now?



# Big Picture: Where are we going?

C

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

compiler

MIPS  
assembly

assembler

machine  
code

CPU

Circuits

Gates

Transistors

Silicon

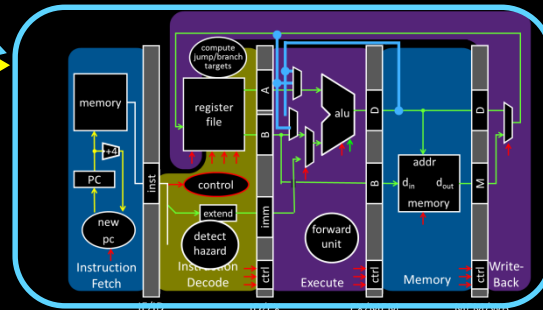
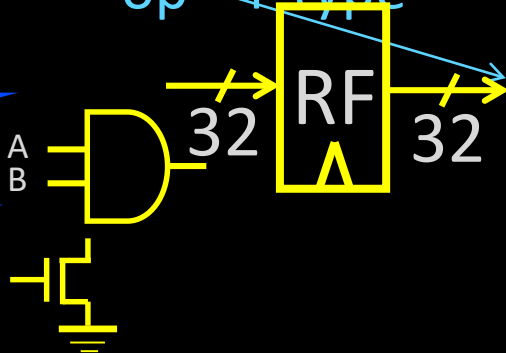
```
addi r5, r0, 10  ← r5 = r0 + 10  
mulr r5, r5, 2   ← r5 = r5 << 1 # r5 = r5 * 2  
addi r5, r5, 15  ← r5 = r5 + 15  
op = addi  r0    r5    10
```

001000	000000	00101	000000000000001010
000000	000000	00101	001010000010000000
001000	00101	00101	000000000000001111

op = addi    r5    r5    15

op = r-type

r5    r5    shamt=1    func=sll



# Big Picture: Where are we going?

C

compiler

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

High Level  
Languages

MIPS  
assembly

```
addi r5, r0, 10  
mulr r5, r5, 2  
addi r5, r5, 15
```

assembler

machine  
code

```
001000000000010100000000000001010  
000000000000001010010100001000000  
001000001010010100000000000001111
```

Instruction Set  
Architecture (ISA)

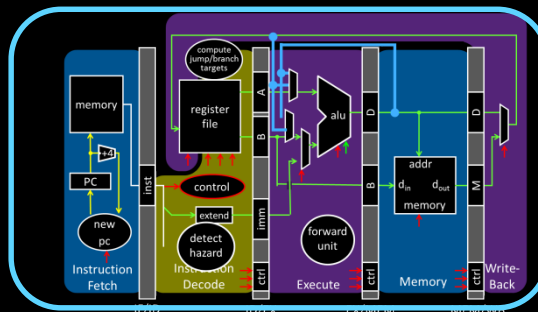
CPU

Circuits

Gates

Transistors

Silicon



# Goals for Today

## Instruction Set Architectures

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

# Next Goal

Is MIPS the only possible instruction set architecture (ISA)?

What are the alternatives?

# Instruction Set Architecture Variations

ISA defines the permissible instructions

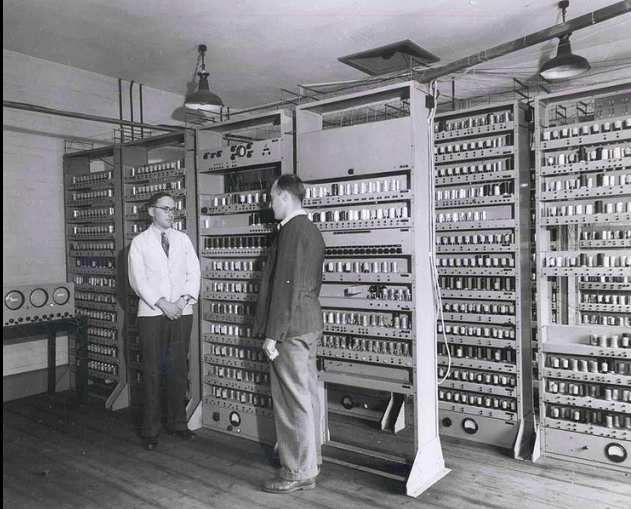
- **MIPS**: load/store, arithmetic, control flow, ...
- ARMv7: similar to MIPS, but more shift, memory, & conditional ops
- ARMv8 (64-bit): even closer to MIPS, no conditional ops
- VAX: arithmetic on memory or registers, strings, polynomial evaluation, stacks/queues, ...
- Cray: vector operations, ...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



# Brief Historical Perspective on ISAs

## Accumulators

- Early stored-program computers had ***one*** register!



EDSAC (Electronic Delay Storage Automatic Calculator) in 1949



Intel 8008 in 1972  
was an accumulator

- One register is two registers short of a MIPS instruction!
- Requires a memory-based operand-addressing mode
  - Example Instructions: `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
  - E.g. indices for array references in data transfer instructions, separate accumulators for multiply or divide instructions, top-of-stack pointer.



Intel 8086  
“extended accumulator”  
Processor for IBM PCs

- Extended Accumulator
  - One operand may be in memory (like previous accumulators).
  - Or, all the operands may be registers (like MIPS).

# Brief Historical Perspective on ISAs

Next step, more registers...

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

# Takeaway

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

Machine	Num 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

# Next Goal

How to compute with limited resources?

i.e. how do you design your ISA if you have limited resources?

# 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

# In the Beginning...

People programmed in assembly and machine code!

E.g. x86

- > 1000 instructions!
  - 1 to 15 bytes each
  - E.g. dozens of add instructions
- 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.  $\text{Mem}[\text{segment} + \text{reg} + \text{reg} * \text{scale} + \text{offset}]$

E.g. VAX

- Like x86, arithmetic on memory or registers, but also on strings, polynomial evaluation, stacks/queues, ...

# Complex Instruction Set Computers (CISC)



# 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.

# 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 bldg that housed the project
- Idea: Possible to make a very small and very fast core
- Influences: Known as “the father of RISC Architecture”. Turing Award Recipient and National Medal of Science.



# Reduced Instruction Set Computer (RISC)

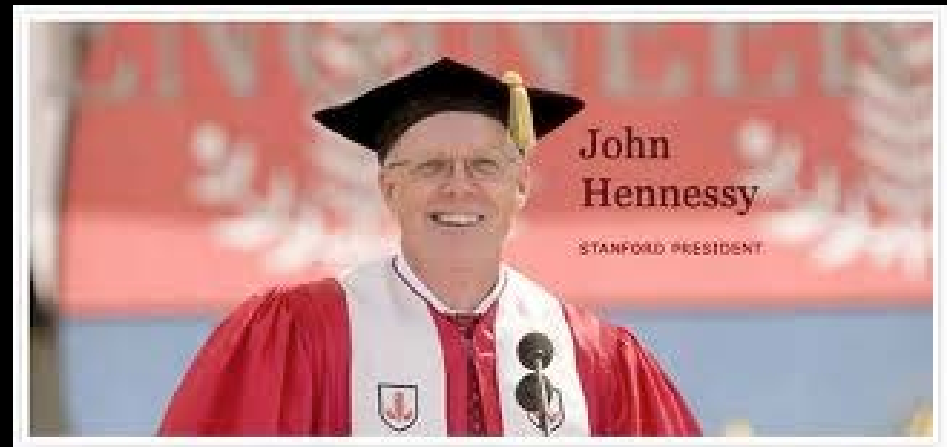
## Dave Patterson

- 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



# Reduced Instruction Set Computer (RISC)

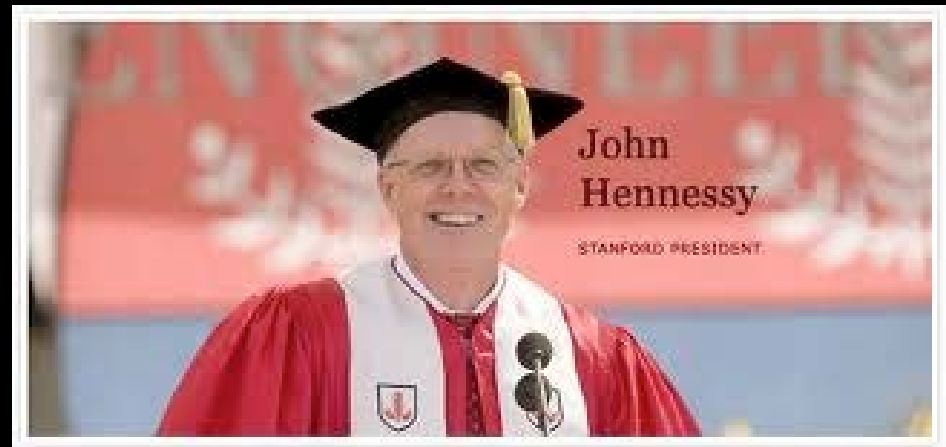
## Dave Patterson

- 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 pipelining, keep full
- Influences: MIPS computer system, PlayStation, Nintendo



# Reduced Instruction Set Computer (RISC)

## MIPS Design Principles

Simplicity favors regularity

- 32 bit instructions

Smaller is faster

- Small register file

Make the common case fast

- Include support for constants

Good design demands good compromises

- Support for different type of interpretations/classes

# Reduced Instruction Set Computer

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: Mem[reg + imm]

x86 = Complex Instruction Set Computer (CISC)

- $>$  1000 instructions, 1 to 15 bytes each
- 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]

# 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 the  
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!

Desktops/Servers

# ARMDroid vs WinTel

- Android OS on ARM processor
- Windows OS on Intel (x86) processor



# 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 MIPS and ARM compare to each other?

# MIPS instruction formats

All MIPS instructions are 32 bits long, has 3 formats



# ARMv7 instruction formats

All ARMv7 instructions are 32 bits long, has 3 formats



# ARMv7 Conditional Instructions

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

In MIPS, performance will be slow if code has a lot of branches

```
Loop: BEQ Ri, Rj, End    // if "NE" (not equal), then stay in loop
      SLT Rd, Rj, Ri     // "GT" if (i > j),
      BNE Rd, R0, Else   // ...
      SUB Ri, Ri, Rj     // if "GT" (greater than), i = i-j;
      J Loop
Else:  SUB Rj, Rj, Ri     // or "LT" if (i < j)
      J Loop             // if "LT" (less than), j = j-i;
End:
```

# ARMv7 Conditional Instructions

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

In ARM, can avoid delay due to Branches with conditional instructions

LOOP: CMP Ri, Rj 

0	1	0	0
=	≠	<	>

 // set condition "NE" if (i != j)

// "GT" if (i > j),

// or "LT" if (i < j)

0	0	0	1
=	≠	<	>

 SUBGT Ri, Ri, Rj // if "GT" (greater than), i = i-j;

1	0	1	0
=	≠	<	>

 SUBLE Rj, Rj, Ri // if "LE" (less than or equal), j = j-i;

0	1	0	0
=	≠	<	>

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



# 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

$Ra = Rb + Rc \ll 4$

$Ra = Rb + Rc \times 16$

# ARMv7 Instruction Set Architecture

All ARMv7 instructions are 32 bits long, has 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

All ARMv8 instructions are 64 bits long, has 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

## ***NO MORE*** Complex Instruction Set Computer (CISC) properties

- NO Conditional execution
- NO Multiple words can be accessed from memory with a single instruction (SIMD: single instr multiple data)

# Instruction Set Architecture 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

# 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.