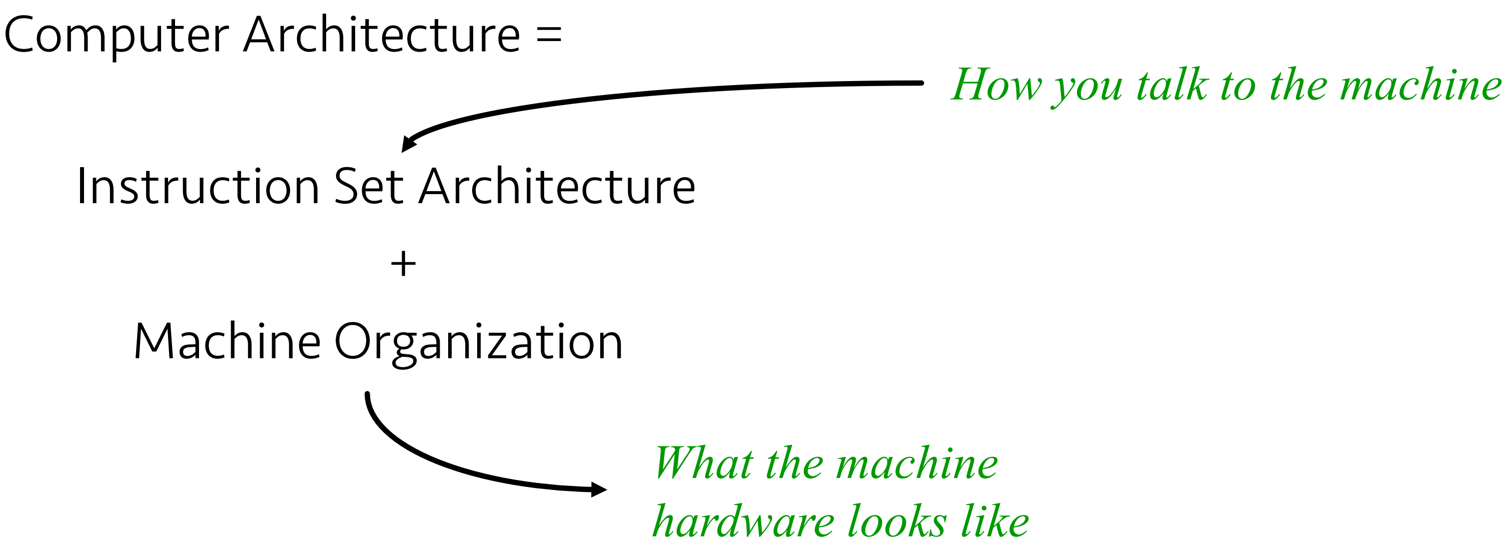
CSE 141: Introduction to Computer Architecture

Instruction Set Architecture (ISA)

Good Morning!

CC BY-NC-ND Pat Pannuto – Many slides adapted from Dean Tullsen

What is Computer Architecture?



Computer Architecture =

Instruction Set Architecture

+

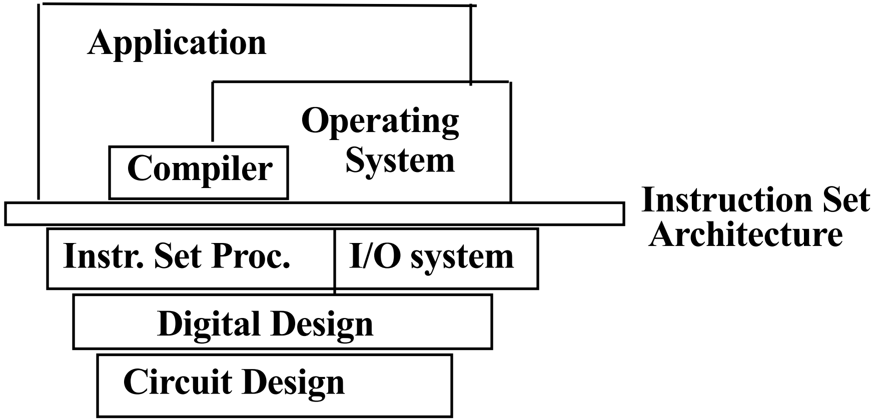
Machine Organization

How you talk to the machine

What the machine hardware looks like

An Instruction Set Architecture is an **abstraction** of a computational machine

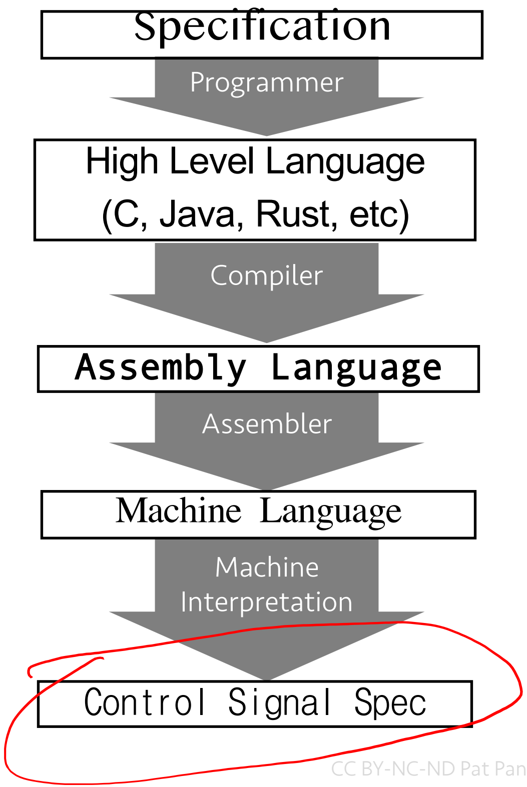
- An ISA is “the agreed-upon interface between all the software that runs on the machine and the hardware that executes it.”



CC BY-NC-ND Pat Pannuto – Many slides adapted from Dean Tullsen

Computers do not speak English

And they do not speak C or Java or Python or Haskell (or…) either



Mach
org - w237

"Swap two array elements."

```
int temp = array[index];
array[index] = array[index + 1];
array[index + 1] = temp;
```

```
lw  $15, 0($2)
lw  $16, 4($2)
sw  $16, 0($2)
sw  $15, 4($2)
```

human

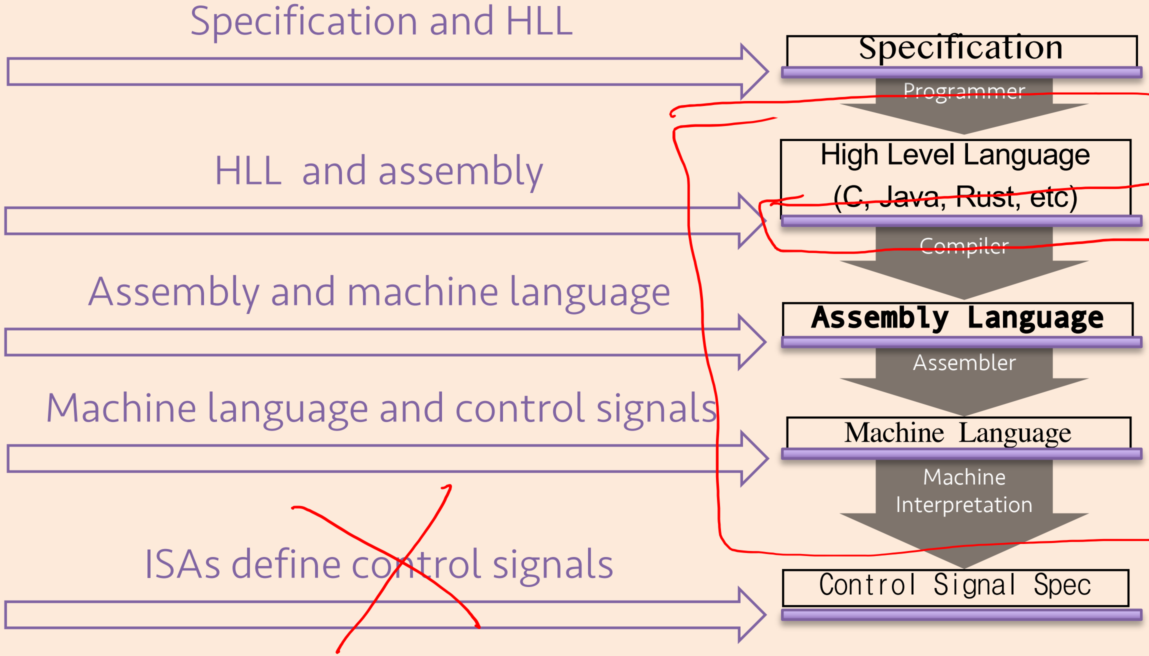
mach

```
10001100011000100000000000000000
10001100111100100000000000000100
10101100111100100000000000000000
10101100011000100000000000000100
```

```
ALUOP[0:3] <= InstReg[9:11] & MASK
```

CC BY-NC-ND Pat Pannuto – Many slides adapted from Dean Tullsen

Poll Q: If you had to put the ISA somewhere in this stack, would you say it sits between…



Specification and HLL

HLL and assembly

Assembly and machine language

Machine language and control signals

~~ISAs define control signals~~

"Swap two array elements."

```
int temp = array[index];
array[index] = array[index + 1];
array[index + 1] = temp;
```

```
lw   $15,  0($2)
lw   $16,  4($2)
sw   $16,  0($2)
sw   $15,  4($2)
```

```
10001100011000100000000000000000
10001100111100100000000000000100
10101100111100100000000000000000
10101100011000100000000000000100
```

```
ALUOP[0:3] <= InstReg[9:11] & MASK
```

CC BY-NC-ND Pat Pannuto – Many slides adapted from Dean Tullsen

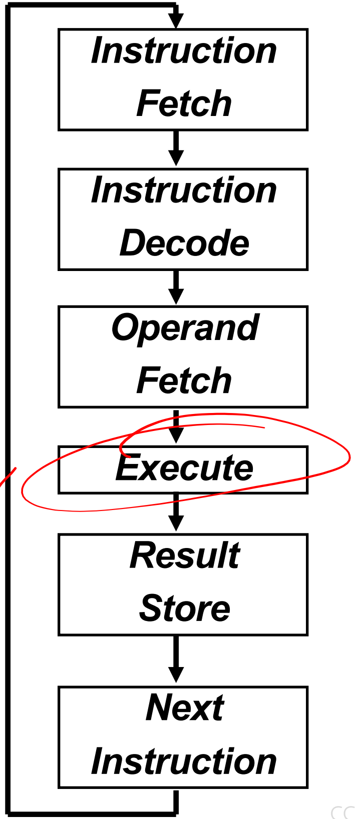
The Instruction Set Architecture

- that part of the architecture that is visible to the programmer
  - available instructions ("opcodes")
  - number and types of registers
  - instruction formats
  - storage access, addressing modes
  - exceptional conditions

Sometimes
+ peripheral access?

CC BY-NC-ND Pat Pannuto – Many slides adapted from Dean Tullsen

The Instruction Execution Cycle



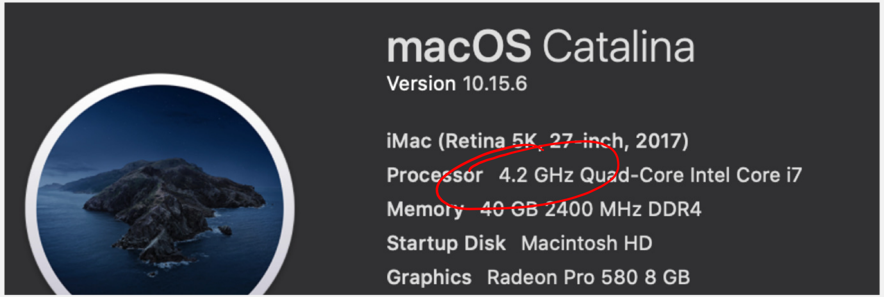
Instruction Fetch	Obtain instruction from program storage
Instruction Decode	Determine required actions and instruction size
Operand Fetch	Locate and obtain operand data
Execute	Compute result value or status
Result Store	Deposit results in storage for later use
Next Instruction	Determine successor instruction

encoding
↓
control signals

least-interesting

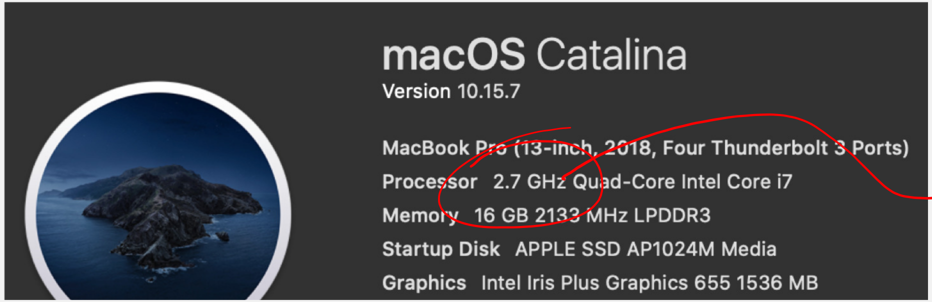
A brief preview of some machine organization concepts: *Cycle*

- The smallest unit of time in a processor



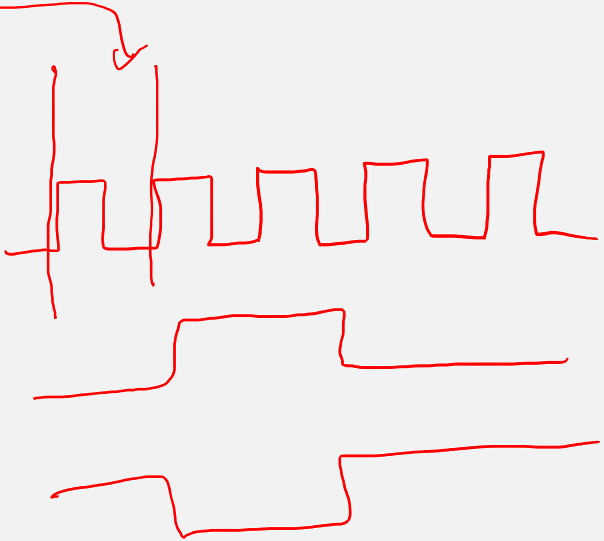
macOS Catalina
Version 10.15.6

iMac (Retina 5K, 27-inch, 2017)
Processor 4.2 GHz Quad-Core Intel Core i7
Memory 40 GB 2400 MHz DDR4
Startup Disk Macintosh HD
Graphics Radeon Pro 580 8 GB



macOS Catalina
Version 10.15.7

MacBook Pro (13-inch, 2018, Four Thunderbolt 3 Ports)
Processor 2.7 GHz Quad-Core Intel Core i7
Memory 16 GB 2133 MHz LPDDR3
Startup Disk APPLE SSD AP1024M Media
Graphics Intel Iris Plus Graphics 655 1536 MB



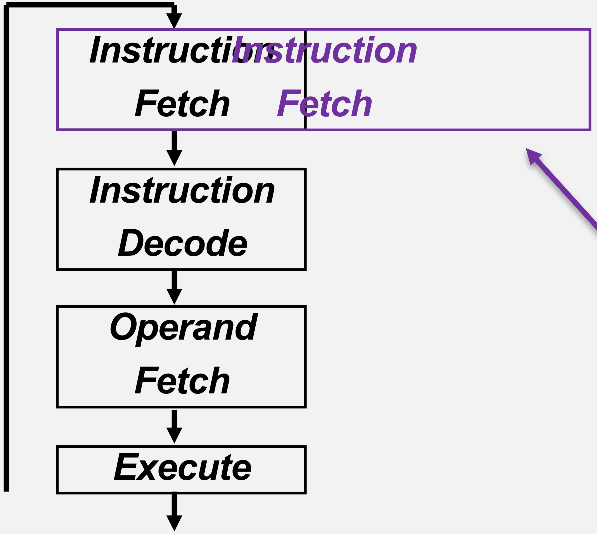
$$\frac{1}{2.7\text{ghz}} = 0.37\text{ns}$$

$$\frac{1}{4.2\text{ghz}} = 0.24\text{ns}$$

wh
2

A brief preview of some machine organization concepts: ***Parallelism***

- The ability to do more than one thing at once



Real-world example

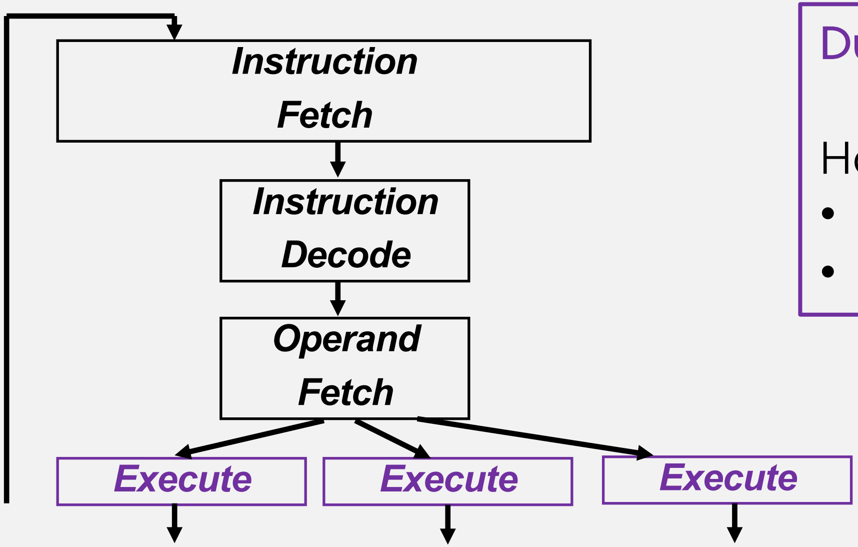
ARM's Thumb instruction set is (mostly) 16-bit instructions on a 32-bit machine

ISA design makes fetch "freely parallel"

CC BY-NC-ND Pat Pannuto – Many slides adapted from Dean Tullsen

A brief preview of some machine organization concepts: *Superscalar Processor*

- Can execute more than one instruction per cycle



Duplication is easy but expensive…

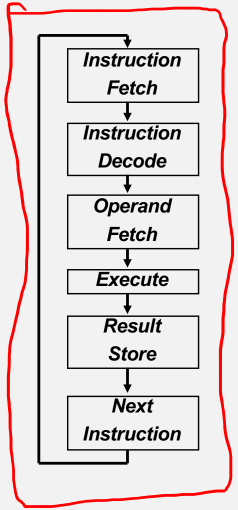
How to do parallelism well?

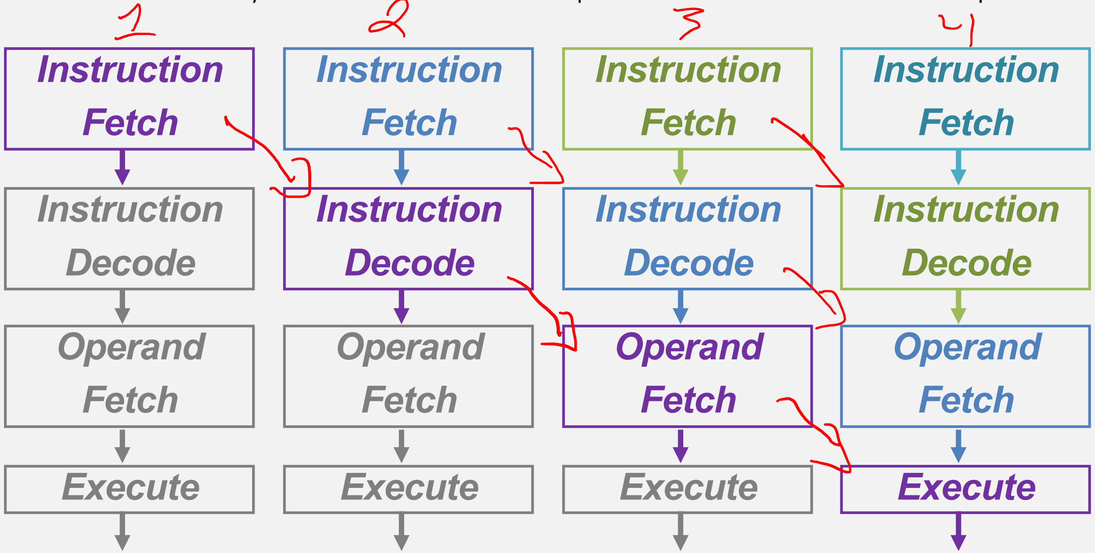
- Second half of this class
- CSE148

CC BY-NC-ND Pat Pannuto – Many slides adapted from Dean Tullsen

A brief preview of some machine organization concepts: *Pipelining*

- Overlapping parts of a large task to increase throughput without decreasing latency
  - Key insight: The less work you do in one step, the faster each step can finish





CC BY-NC-ND Pat Pannuto – Many slides adapted from Dean Tullsen

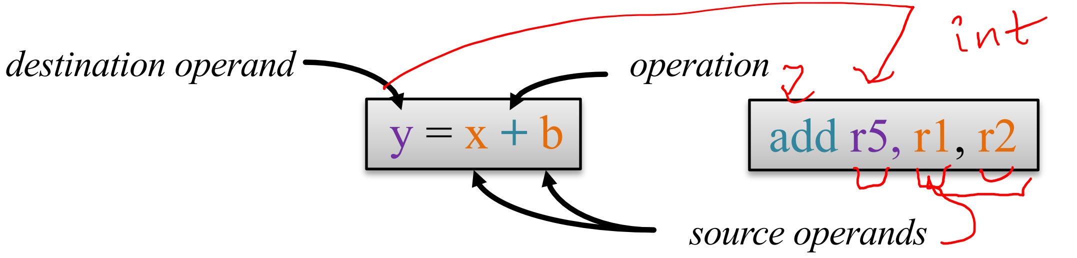
Key questions to ask when designing an ISA

add r5, r1, M[100]

int

float point

- operations
  - how many?
  - which ones?
- operands
  - how many?
  - location
  - types
  - how to specify?
- instruction format
  - size
  - how many formats?



Syntax choice	Design choice
add r5, r1, r2	add r5, r1– r4
add [r1, r2], r5	

how does the computer know what `0001 0101 0001 0010` *means?*

add r5 r1 r2

CC BY-NC-ND Pat Pannuto – Many slides adapted from Dean Tullsen

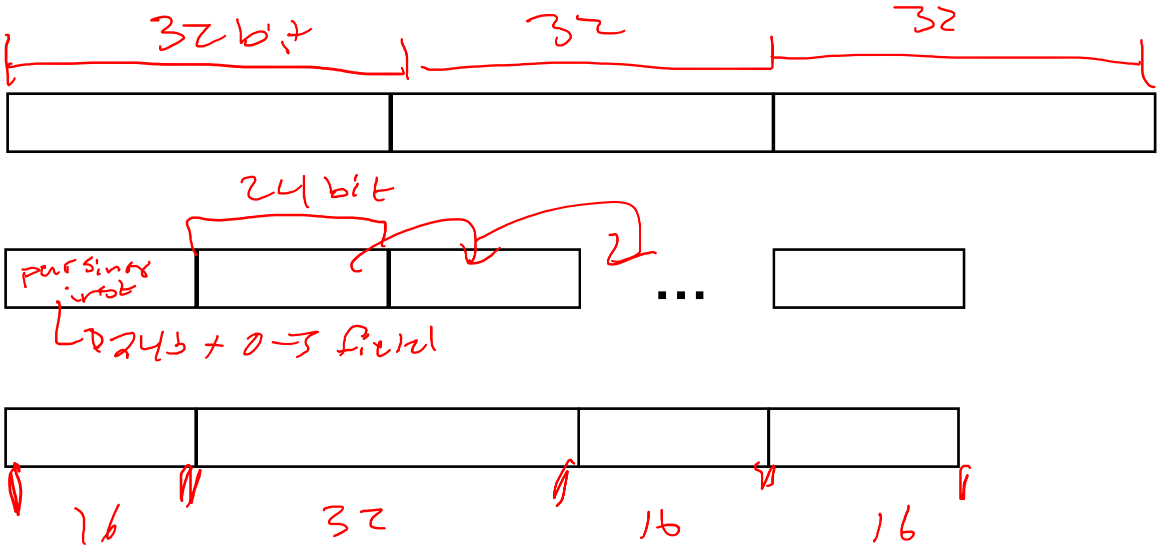
Let us design MIPS together

- We will look at several of the key ISA design decisions
- To succeed in 141 you need to understand the how and the why of MIPS
  - The rest of the course builds on MIPS, so need to be comfortable with it
  - But also need to understand the architectural tradeoffs of MIPS
- To succeed in 141L you need to understand the tradeoffs in ISA design

CC BY-NC-ND Pat Pannuto – Many slides adapted from Dean Tullsen

How long should an instruction be?

- Fixed
- Variable
- Hybrid



add r5, r1, r2

→ 00100100

CC BY-NC-ND Pat Pannuto – Many slides adapted from Dean Tullsen

Instruction length tradeoffs

- Fixed-length instructions (MIPS)
  - easy fetch and decode
  - simplify pipelining and parallelism.
- Variable-length instructions (Intel 80x86, VAX)
  - multi-step fetch and decode
  - much more flexible and compact instruction set.
- Hybrid instructions (ARM)
  - Middle ground

⇨ All MIPS instructions are 32 bits long.

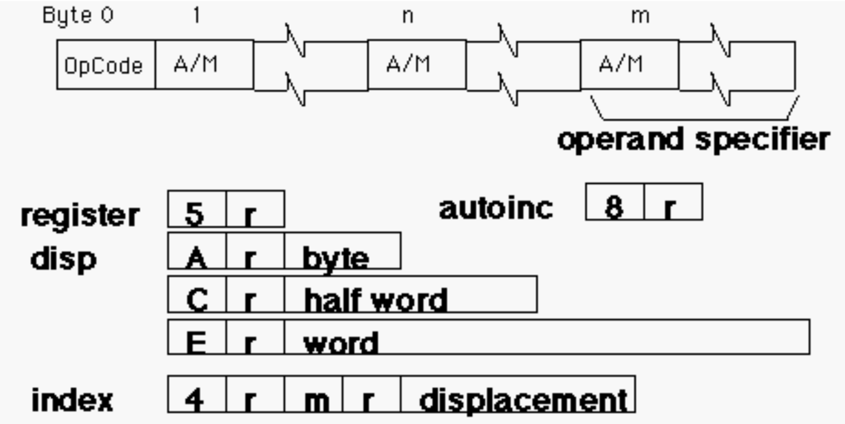
- this decision impacts every other ISA decision we make because it makes instruction bits scarce.

CC BY-NC-ND Pat Pannuto – Many slides adapted from Dean Tullsen

Instruction Formats: What does each bit mean?

- Having many different instruction formats…
  - complicates decoding
  - uses more instruction bits (to specify the format)
  - Could allow us to take full advantage of a variable-length ISA

VAX 11 instruction format



The MIPS Instruction Format

	6 bits	5 bits	5 bits	5 bits	5 bits	6 bits
Register `R-type`	**opcode**	**rs**	**rt**	**rd**	**sa**	**funct**
Immediate `I-type`	**opcode**	**rs**	**rt**	**immediate**		
Jump `J-type`	**opcode**	**target**				

- the opcode tells the machine which format

Example of instruction encoding:

f1 f2 f3 4 5 6

	6 bits	5 bits	5 bits	5 bits	5 bits	6 bits
Register R-type	opcode	rs	rt	rd	sa	funct
Immediate I-type	opcode	rs	rt	immediate		
Jump J-type	opcode	target				

rs = source

rt = target

rd = dest

r2-6

f.46

add r5, r1, r2

R

add

```
opcode=0,     rs=1,  rt=2,  rd=5,  sa=0,  funct=32
000000        00001  00010  00101  00000  100000

00000000001000100010100000100000
0x00222420
```

$2^6 = 64$ opcodes