Computers

What is a computer?

- CPU
- memory (disk drives, DRAM, SRAM, CD)
- input (mouse, keyboard)
- output (display, printer)
- network
- software

We need abstraction ...

Levels of abstraction

Software:

- Application
- Operating system
- Firmware

Instruction set architecture:

- Data type and structures: encodings and machine representation
- Instruction set
- Instruction formats
- Addressing modes and accessing data and instructions

Hardware:

- Instruction set processing
- I/O System
- Digital design
- Circuit design
- Layout

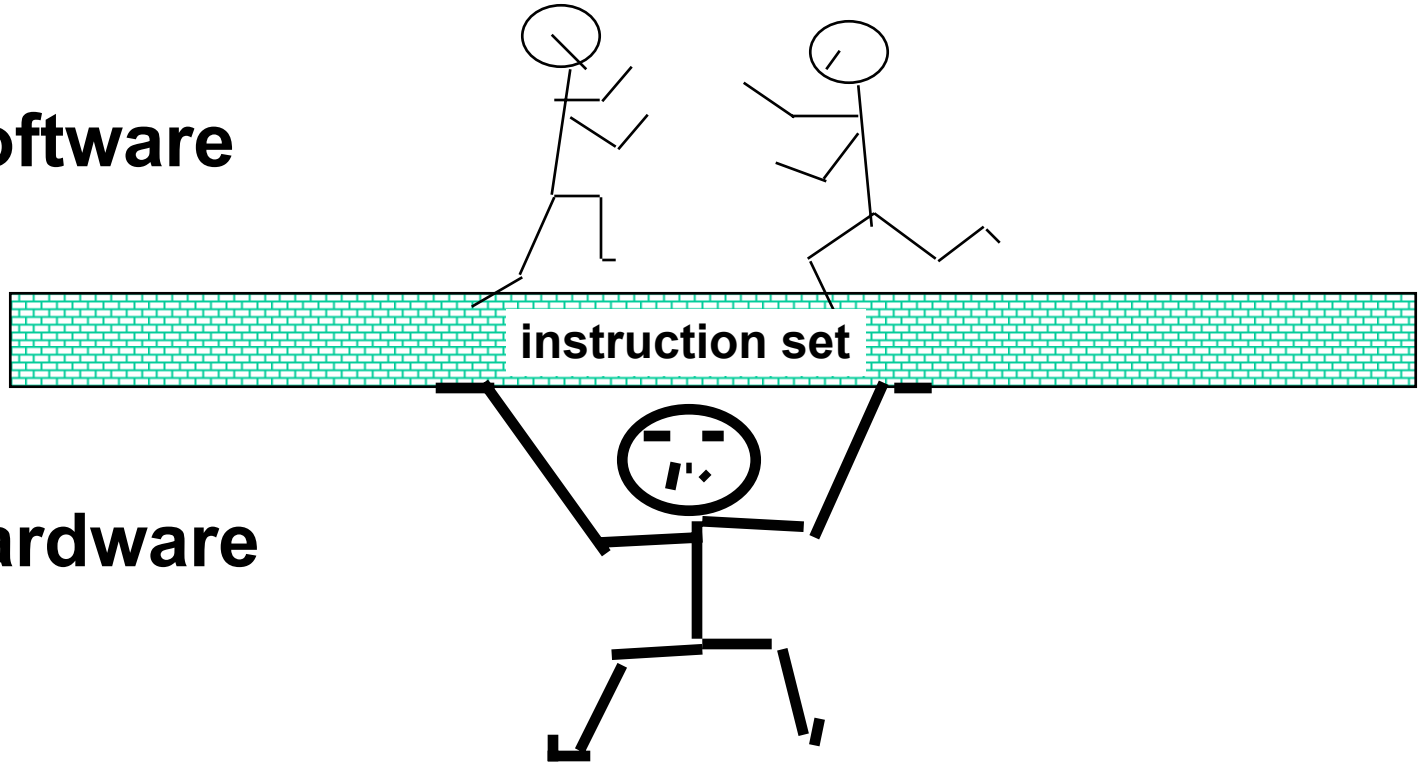
What is “Computer Architecture”

Computer Architecture =

Instruction Set Architecture +

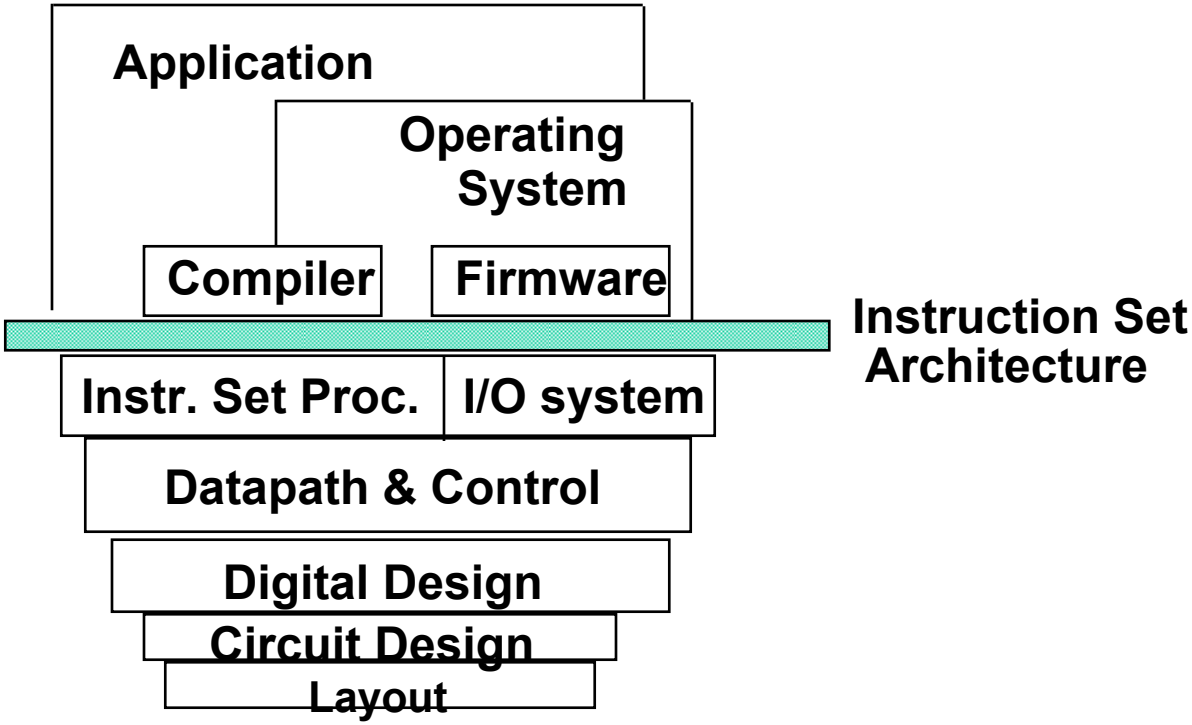
Machine Organization

software

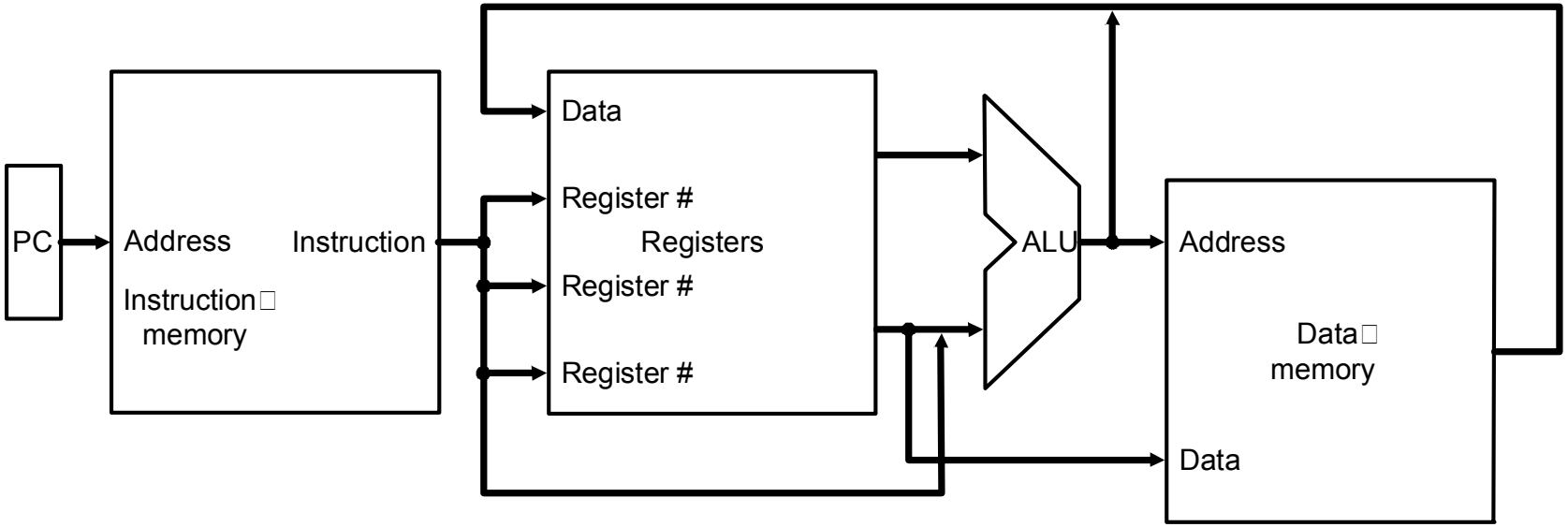


hardware

What is “Computer Architecture”?



MIPS machine



Example: adding two variables

Software level

C:

- A = B + C

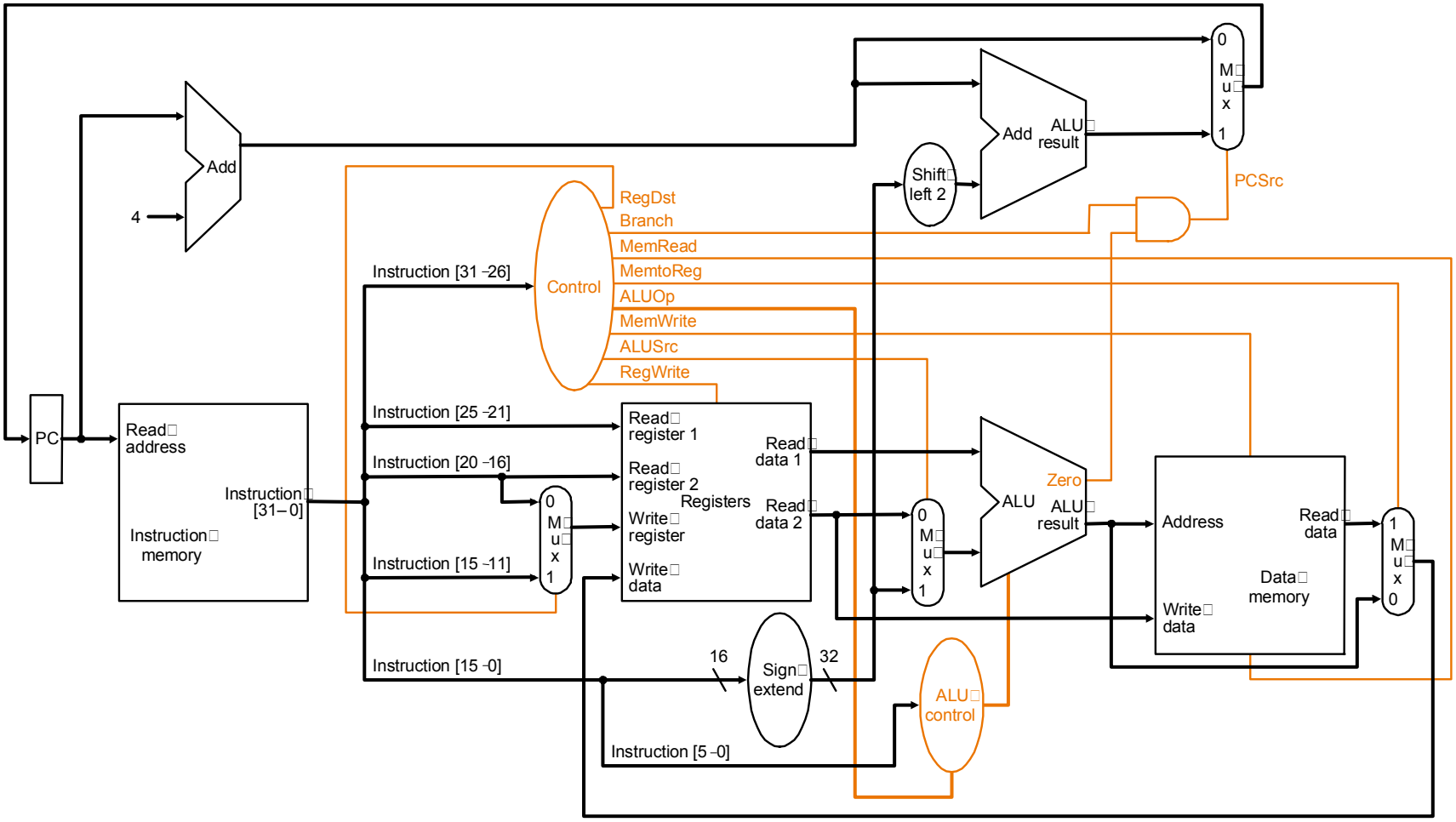
Assembler

- B -> $s1, C -> $s2
- add $t0, $s1, $s2
- $t0 -> A

Machine instruction:

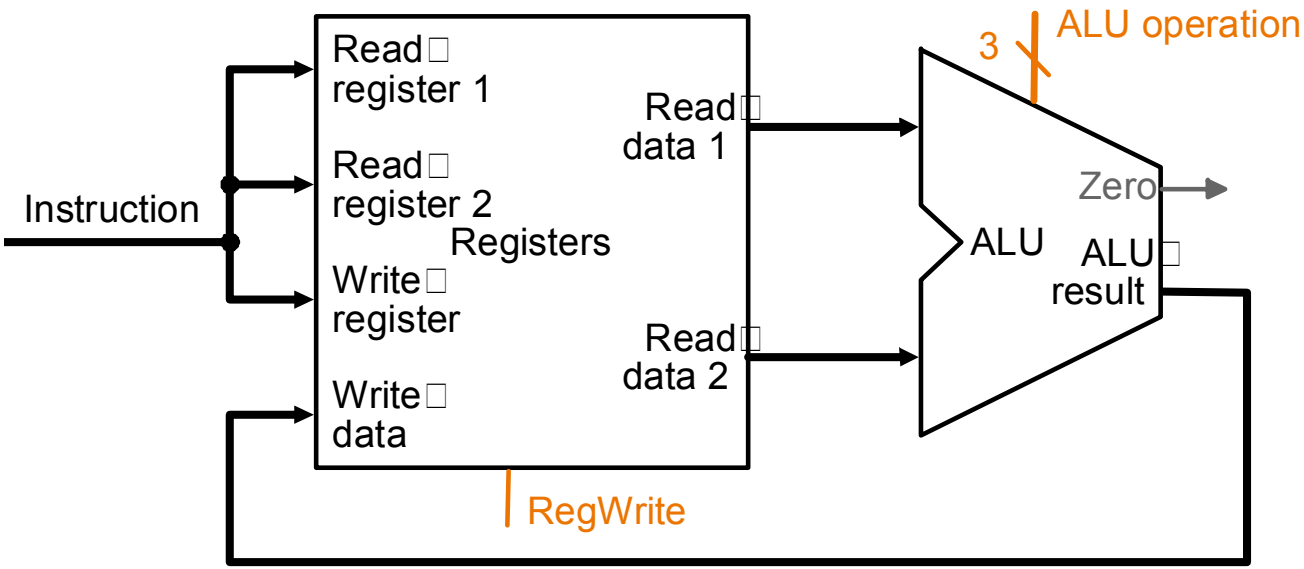
	op	rs	rt	rd	…	funct
• decimal:	0	17	18	8	0	32
• binary:	000000	10001	10010	01000	00000	100000

Example: Datapath and control



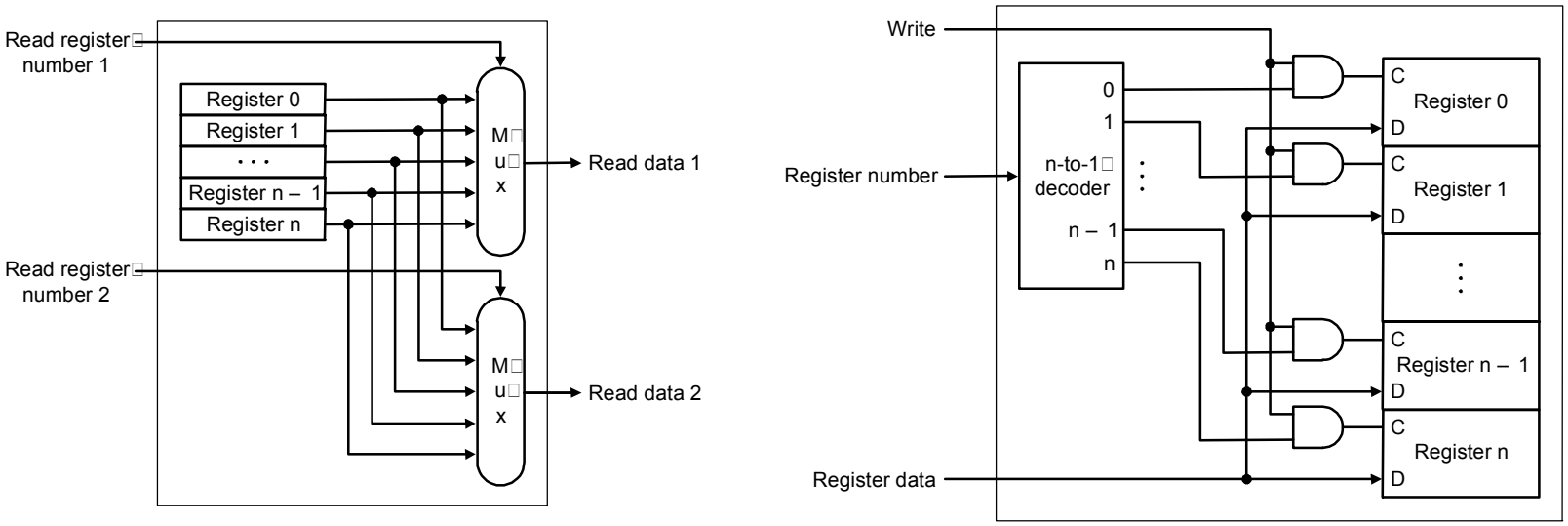
Example: from instructions to gates

Register file and ALU



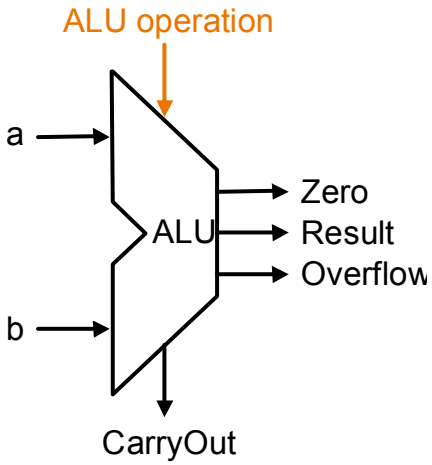
Example: from instructions to gates

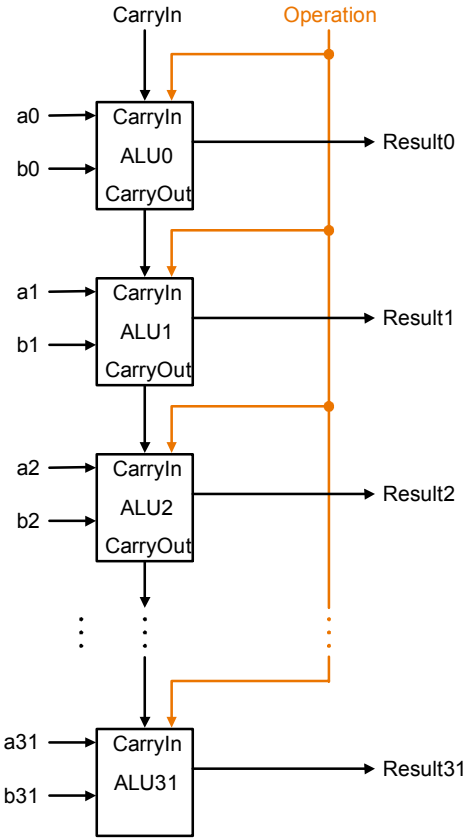
Inside register file

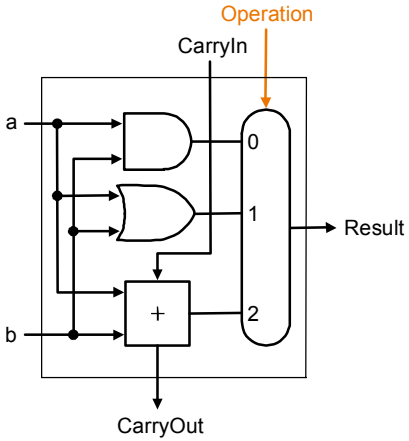


Example: from instructions to gates

Arithmetic Logic Unit (ALU)







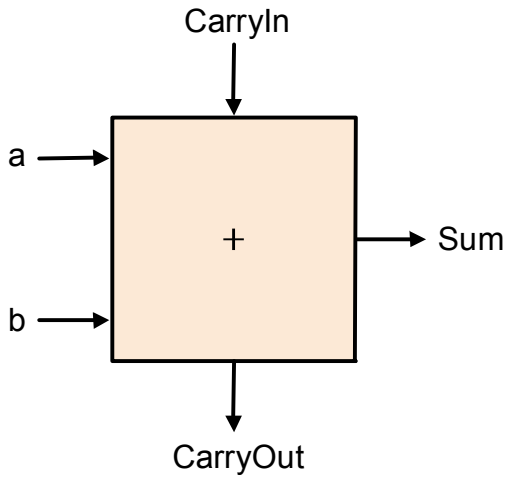
Example: from instructions to gates

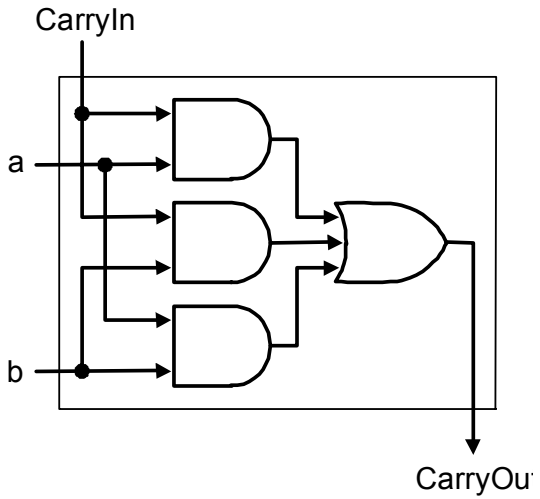
ALU: Carry Out logic

$$CarryOut = b.CarryIn + a.CarryIn + a.b + a.b.CarryIn$$

or

$$CarryOut = b.CarryIn + a.CarryIn + a.b$$

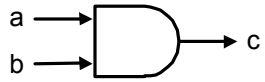




Example: from instructions to gates

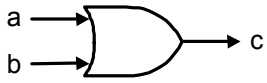
Logic gates

1. AND gate (c = a . b)



a	b	c = a . b
0	0	0
0	1	0
1	0	0
1	1	1

2. OR gate (c = a + b)

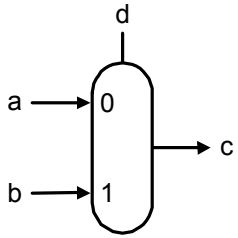


a	b	c = a + b
0	0	0
0	1	1
1	0	1
1	1	1

3. Inverter (c = $\bar{a}$)

a	c = $\bar{a}$
0	1
1	0

4. Multiplexor
   (if d = = 0, c = a;
   else c = b)



d	c
0	a
1	b

Instructions:

- Language of the Machine
- More primitive than higher level languages
  e.g., no sophisticated control flow
- Very restrictive
  e.g., MIPS Arithmetic Instructions

- We'll be working with the MIPS instruction set architecture
  - similar to other architectures developed since the 1980's
  - used by NEC, Nintendo, Silicon Graphics, Sony

Design goals: maximize performance and minimize cost, reduce design time

MIPS arithmetic

- All instructions have 3 operands
- Operand order is fixed (destination first)

Example:

C code: `A = B + C`

MIPS code: `add $s0, $s1, $s2`

(associated with variables by compiler)

MIPS arithmetic

- Design Principle: simplicity favors regularity. Why?
- Of course this complicates some things...

C code:

```
A = B + C + D;
E = F - A;
```

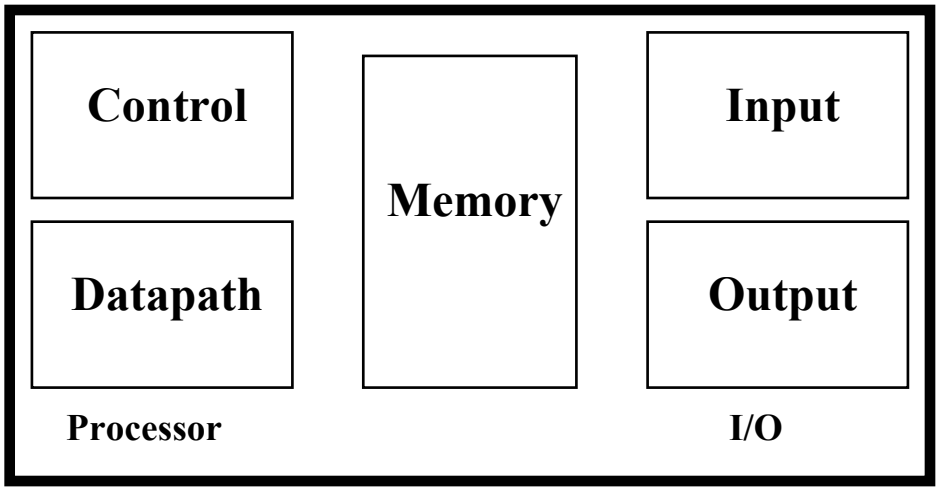
MIPS code:

```
add $t0, $s1, $s2
add $s0, $t0, $s3
sub $s4, $s5, $s0
```

- Operands must be registers, only 32 registers provided
- Design Principle: smaller is faster. Why?

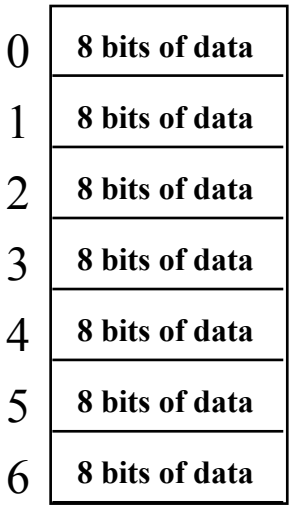
Registers vs. Memory

- Arithmetic instructions operands must be registers,
  — only 32 registers provided
- Compiler associates variables with registers
- What about programs with lots of variables



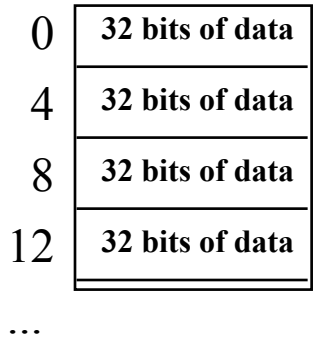
Memory Organization

- Viewed as a large, single-dimension array, with an address.
- A memory address is an index into the array
- "Byte addressing" means that the index points to a byte of memory.



Memory Organization

- Bytes are nice, but most data items use larger "words"
- For MIPS, a word is 32 bits or 4 bytes.



Registers hold 32 bits of data

- 2^{32} bytes with byte addresses from 0 to 2^{32}-1
- 2^{30} words with byte addresses 0, 4, 8, ... 2^{32}-4
- Words are aligned

  i.e., what are the least 2 significant bits of a word address?

Instructions

- Load and store instructions
- Example:

C code:

```
A[8] = h + A[8];
```

MIPS code:

```
lw $t0, 32($s3)
add $t0, $s2, $t0
sw $t0, 32($s3)
```

- Store word has destination last
- Remember arithmetic operands are registers, not memory!

Our First Example

- Can we figure out the code?

```
swap(int v[], int k);
{ int temp;
     temp = v[k]
     v[k] = v[k+1];
     v[k+1] = temp;
}
```

```
swap:
     muli $2, $5, 4
     add $2, $4, $2
     lw $15, 0($2)
     lw $16, 4($2)
     sw $16, 0($2)
     sw $15, 4($2)
     jr $31
```

So far we've learned:

- MIPS
  - — loading words but addressing bytes
  - — arithmetic on registers only

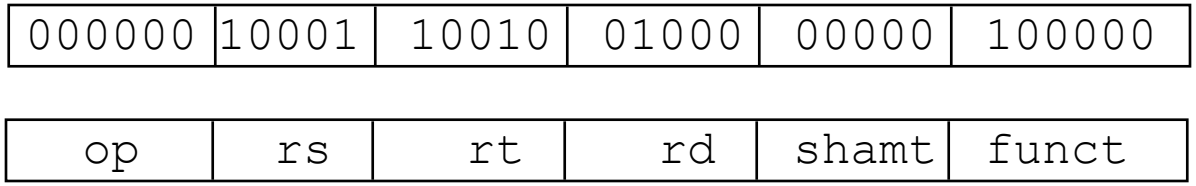
- Instruction / Meaning

```
add $s1, $s2, $s3        $s1 = $s2 + $s3
sub $s1, $s2, $s3        $s1 = $s2 - $s3
lw $s1, 100($s2)         $s1 = Memory[$s2+100]
sw $s1, 100($s2)         Memory[$s2+100] = $s1
```

Machine Language

- Instructions, like registers and words of data, are also 32 bits long
  - Example: `add $t0, $s1, $s2`
  - registers have numbers, `$t0=9, $s1=17, $s2=18`

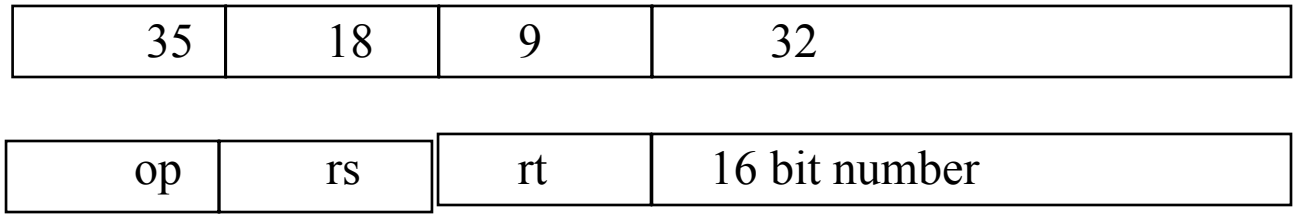
- Instruction Format:



- *Can you guess what the field names stand for?*

Machine Language

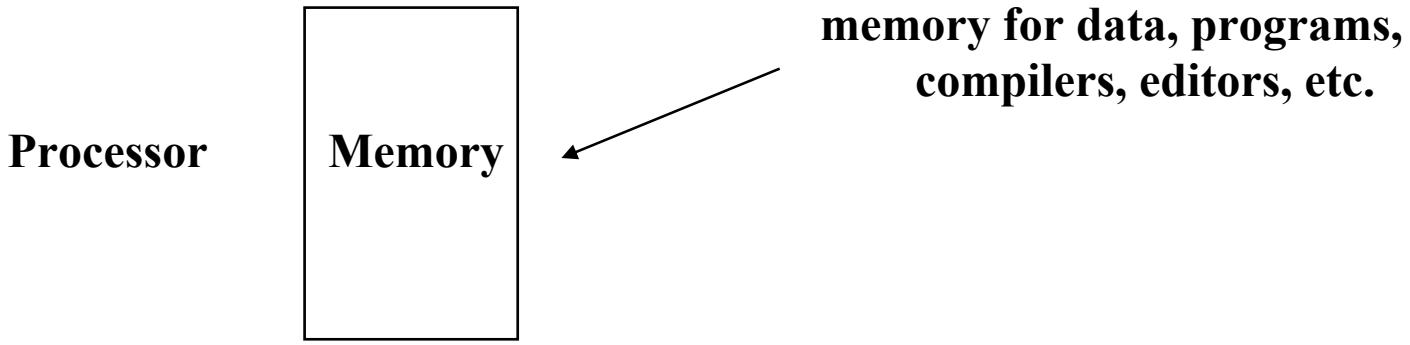
- Consider the load-word and store-word instructions,
  - What would the regularity principle have us do?
  - New principle: Good design demands a compromise
- Introduce a new type of instruction format
  - I-type for data transfer instructions
  - other format was R-type for register
- Example: `lw $t0, 32($s2)`



- Where's the compromise?

Stored Program Concept

- Instructions are bits
- Programs are stored in memory
  — to be read or written just like data



Control

- Decision making instructions
  - alter the control flow,
  - i.e., change the "next" instruction to be executed

- MIPS conditional branch instructions:

```
bne $t0, $t1, Label
beq $t0, $t1, Label
```

- Example: if (i==j) h = i + j;

```
        bne $s0, $s1, Label
        add $s3, $s0, $s1
Label: ....
```

Control

- MIPS unconditional branch instructions:

```
j  label
```

- Example:

```
if (i!=j)              beq $s4, $s5, Lab1
    h=i+j;             add $s3, $s4, $s5
else                   j Lab2
    h=i-j;             Lab1: sub $s3, $s4, $s5
                       Lab2: ...
```

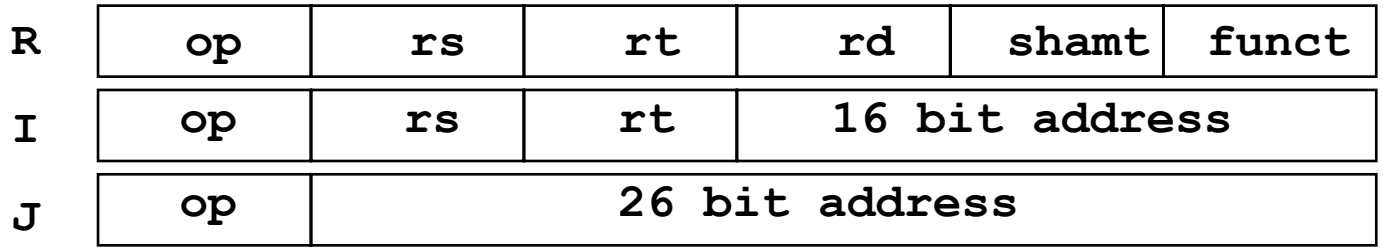
- *Can you build a simple for loop?*

So far:

- Instruction — Meaning

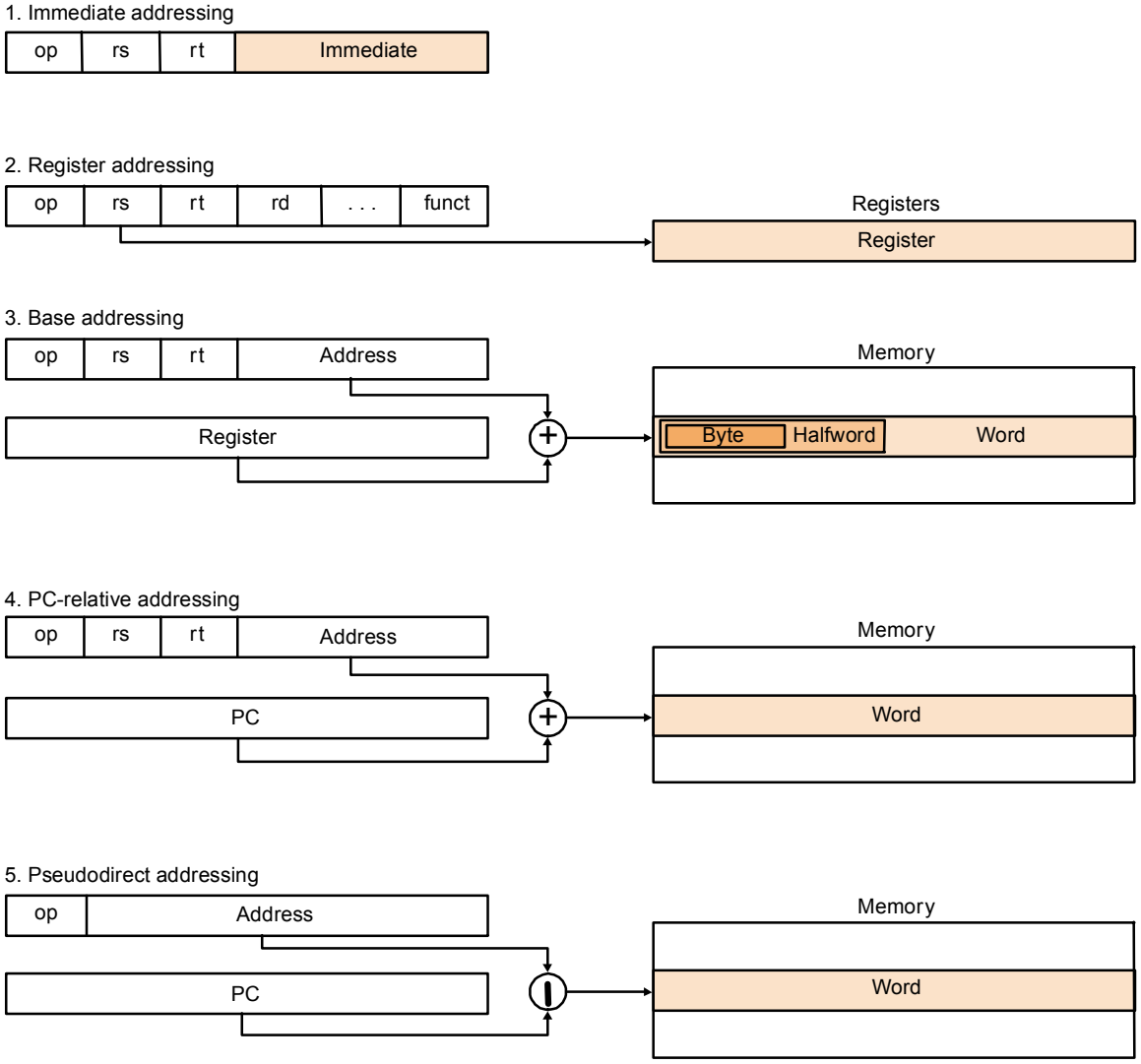
```
add $s1,$s2,$s3     $s1 = $s2 + $s3
sub $s1,$s2,$s3     $s1 = $s2 - $s3
lw $s1,100($s2)     $s1 = Memory[$s2+100]
sw $s1,100($s2)     Memory[$s2+100] = $s1
bne $s4,$s5,L       Next instr. is at Label if $s4 ° $s5
beq $s4,$s5,L       Next instr. is at Label if $s4 = $s5
j Label             Next instr. is at Label
```

- Formats:



Policy of Use Conventions

Name	Register number	Usage
`$zero`	0	the constant value 0
`$v0-$v1`	2-3	values for results and expression evaluation
`$a0-$a3`	4-7	arguments
`$t0-$t7`	8-15	temporaries
`$s0-$s7`	16-23	saved
`$t8-$t9`	24-25	more temporaries
`$gp`	28	global pointer
`$sp`	29	stack pointer
`$fp`	30	frame pointer
`$ra`	31	return address


1. Immediate addressing
op
rs
rt
Immediate
2. Register addressing
op
rs
rt
rd
. . .
funct
Registers
Register
3. Base addressing
op
rs
rt
Address
Register
Memory
Byte
Halfword
Word
4. PC-relative addressing
op
rs
rt
Address
PC
Memory
Word
5. Pseudodirect addressing
op
Address
PC
Memory
Word