C335
Computer Structures

MIPS Instructions (Part #3)

Dr. Liqiang Zhang

Department of Computer and Information Sciences

Adapted from Morgan Kaufmann and others

Review and More: Dealing with Constants

- Small constants are used quite frequently (50% of operands in many common programs)

  e.g., A = A + 5;
  B = B + 1;
  C = C - 18;

- Solutions? Why not?
  - Put “typical constants” in memory and load them
  - Create hard-wired registers (like $zero) for constants like 1, 2, 4, 10, …

- How do we make this work?
  - How do we Make the common case fast !

Review and More: Immediate Operands

- Include constants inside arithmetic instructions
  - Much faster than if they have to be loaded from memory (they come in from memory *with* the instruction itself)

- MIPS immediate instructions

```
addi  $s3, $s3, 4       #$s3 = $s3 + 4
```

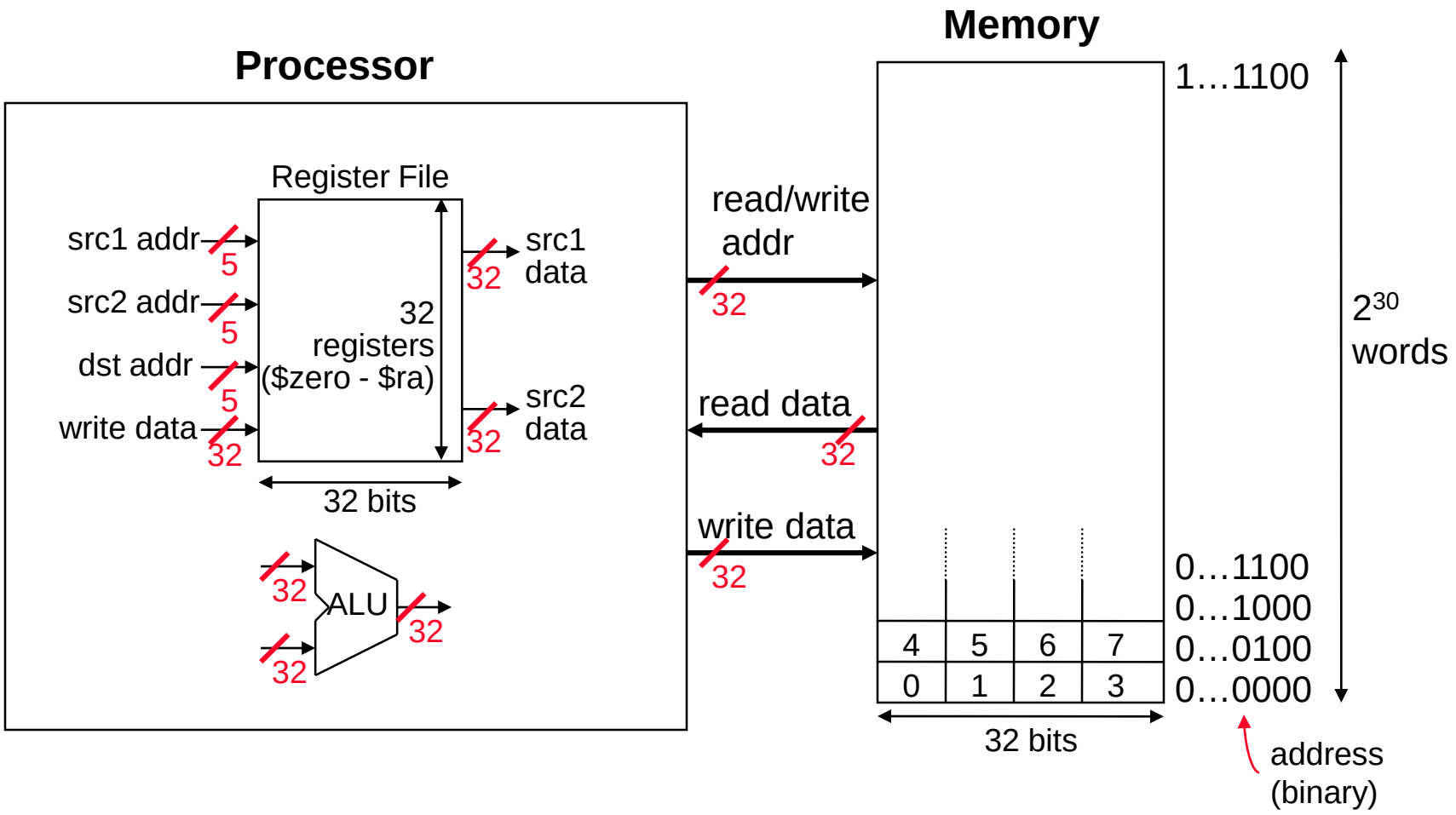
- There is no `subi` instruction, why?

MIPS Instructions, so far

Category	Instr	Example	Meaning
Arithmetic	add	add $s1, $s2, $s3	$s1 = $s2 + $s3
	subtract	sub $s1, $s2, $s3	$s1 = $s2 - $s3
	add immediate	addi $s1, $s2, 4	$s1 = $s2 + 4
Data transfer	load word	lw $s1, 32($s2)	$s1 = Memory($s2+32)
	store word	sw $s1, 32($s2)	Memory($s2+32) = $s1

Review: MIPS Organization

- Arithmetic instructions – to/from the register file
- Load/store instructions - to/from memory



Review: Unsigned Binary Representation

Hex	Binary	Decimal
0x00000000	0…0000	0
0x00000001	0…0001	1
0x00000002	0…0010	2
0x00000003	0…0011	3
0x00000004	0…0100	4
0x00000005	0…0101	5
0x00000006	0…0110	6
0x00000007	0…0111	7
0x00000008	0…1000	8
0x00000009	0…1001	9
	…	
0xFFFFFFFC	1…1100	$2^{32} - 4$
0xFFFFFFFD	1…1101	$2^{32} - 3$
0xFFFFFFFE	1…1110	$2^{32} - 2$
0xFFFFFFFF	1…1111	$2^{32} - 1$

2^{31} 2^{30} 2^{29} . . . 2^3 2^2 2^1 2^0 bit weight

31 30 29 . . . 3 2 1 0 bit position

1 1 1 . . . 1 1 1 1 bit

1 0 0 0 . . . 0 0 0 0 - 1

$2^{32} - 1$

Review: Signed Binary Representation

	2'sc binary	decimal
-2^3 =	1000	-8
$-(2^3 - 1)$ =	1001	-7
	1010	-6
	1011	-5
	1100	-4
	1101	-3
	1110	-2
	1111	-1
	0000	0
	0001	1
	0010	2
	0011	3
	0100	4
	0101	5
	0110	6
$2^3 - 1$ =	0111	7

complement all the bits

0101

and add a 1

0110

1011

and add a 1

1010

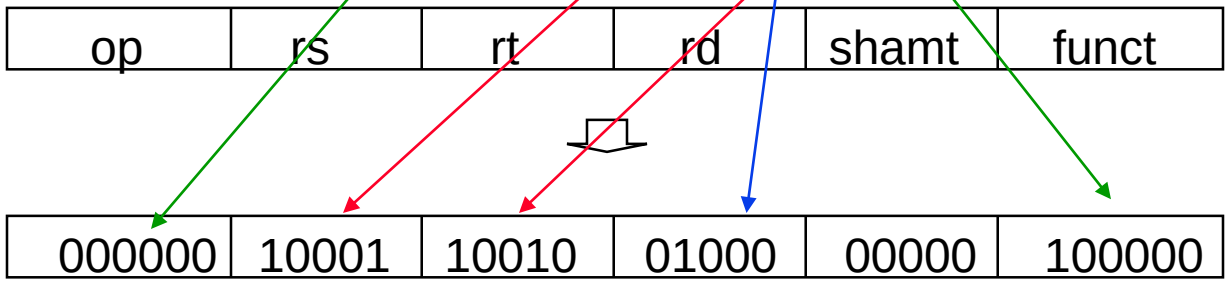
complement all the bits

Machine Language - Arithmetic Instruction

- Instructions, like registers and words of data, are also 32 bits long
  - Example: `add $t0, $s1, $s2`

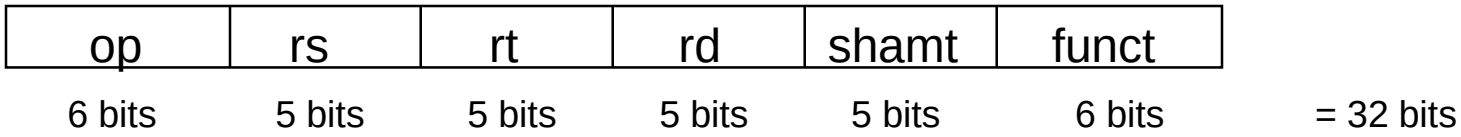
  registers have numbers `$t0=$8, $s1=$17, $s2=$18`

- Instruction Format:



Can you guess what the field names stand for?

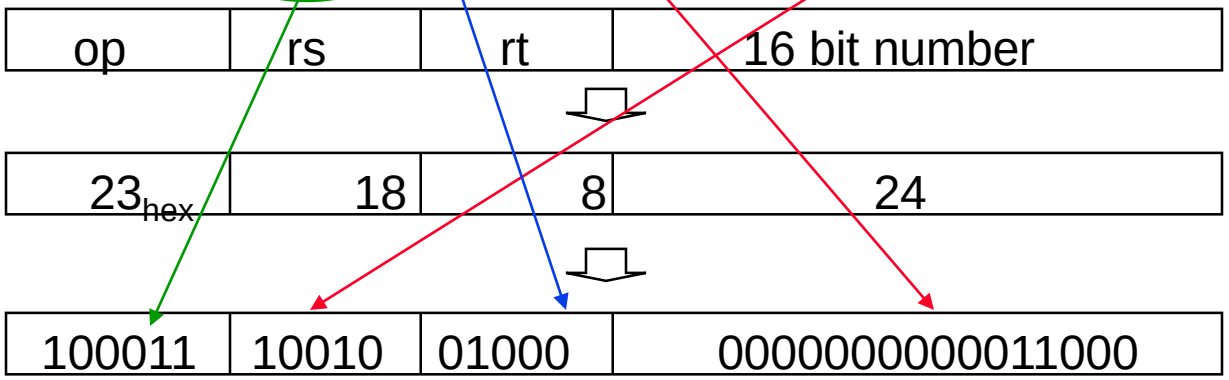
MIPS Instruction Fields



- *op* **op**code indicating operation to be performed
- *rs* address of the first **r**egi**s**ter source operand
- *rt* address of the second **r**egis**t**er source operand
- *rd* the **r**egister **d**estination address
- *shamt* **sh**ift **am**oun**t** (for shift instructions)
- *funct* **funct**ion code that selects the specific variant of the operation specified in the opcode field

Machine Language - Load Instruction

- Consider the load-word and store-word instr's
  - What would the regularity principle have us do?
    - But . . . Good design demands compromise
- Introduce a new type of instruction format
  - I-type for data transfer instructions (previous format was R-type for **r**egister)
- Example: `lw $t0, 24($s2)`



Where's the compromise?

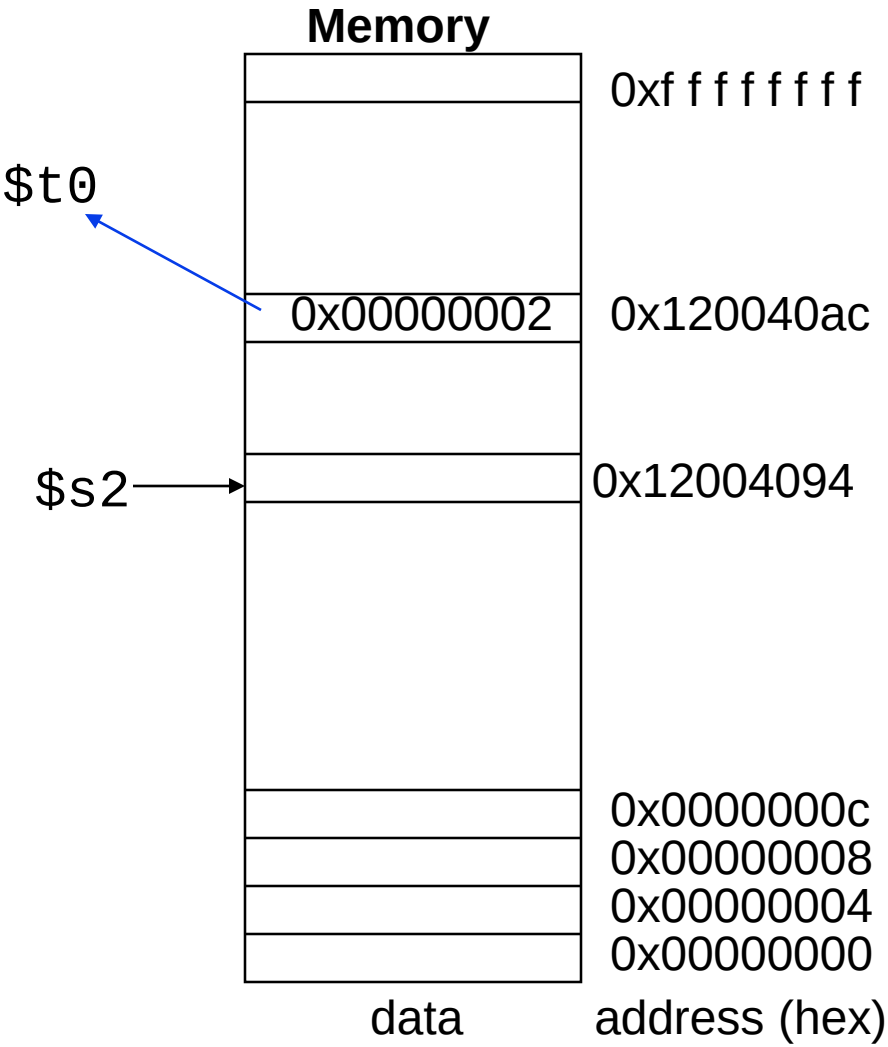
Memory Address Location

- Example: `lw $t0, 24($s2)`

24_{10} + $s2 =

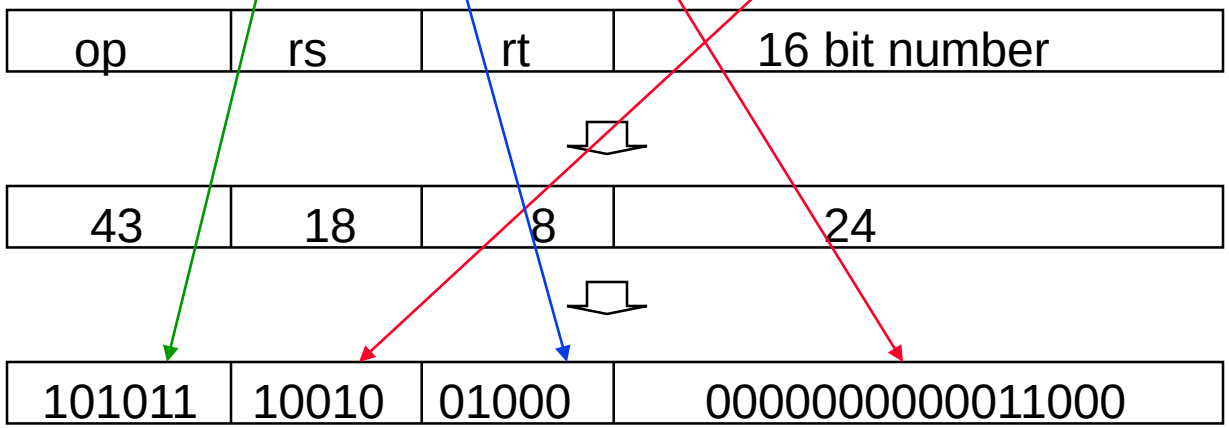
. . . 1001 0100
+ . . . 0001 1000
. . . 1010 1100 =
0x120040ac

Note that the offset can be positive or negative



Machine Language - Store Instruction

- Example: `sw $t0, 24($s2)`



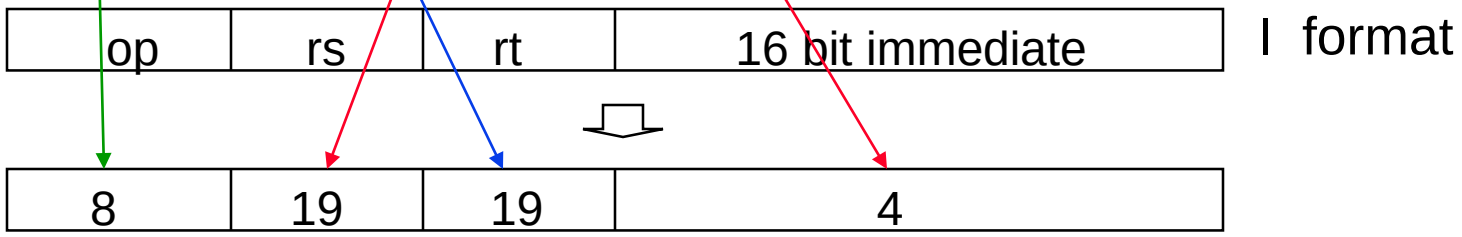
- A 16-bit offset means access is limited to memory locations within a range of $+2^{13}-1$ to -2^{13} (~8,192) words ($+2^{15}-1$ to -2^{15} (~32,768) bytes) of the address in the base register `$s2`
  - 2's complement

Machine Language – Immediate Instructions

- What instruction format is used for the `addi` ?

```
addi $s3, $s3, 4      #$s3 = $s3 + 4
```

- Machine format:



- The constant is kept inside the instruction itself!
  - So must use the I format – Immediate format
  - Limits immediate values to the range $+2^{15}-1$ to -2^{15}

Instruction Format Encoding

- Can reduce the complexity with multiple formats by keeping them as similar as possible
  - First three fields are the same in R-type and I-type
- Each format has a distinct set of values in the op field

Instr	Frmt	op	rs	rt	rd	shamt	funct	address
`add`	R	0	reg	reg	reg	0	32_{ten}	NA
`sub`	R	0	reg	reg	reg	0	34_{ten}	NA
`addi`	I	8_{ten}	reg	reg	NA	NA	NA	constant
`lw`	I	35_{ten}	reg	reg	NA	NA	NA	address
`sw`	I	43_{ten}	reg	reg	NA	NA	NA	address

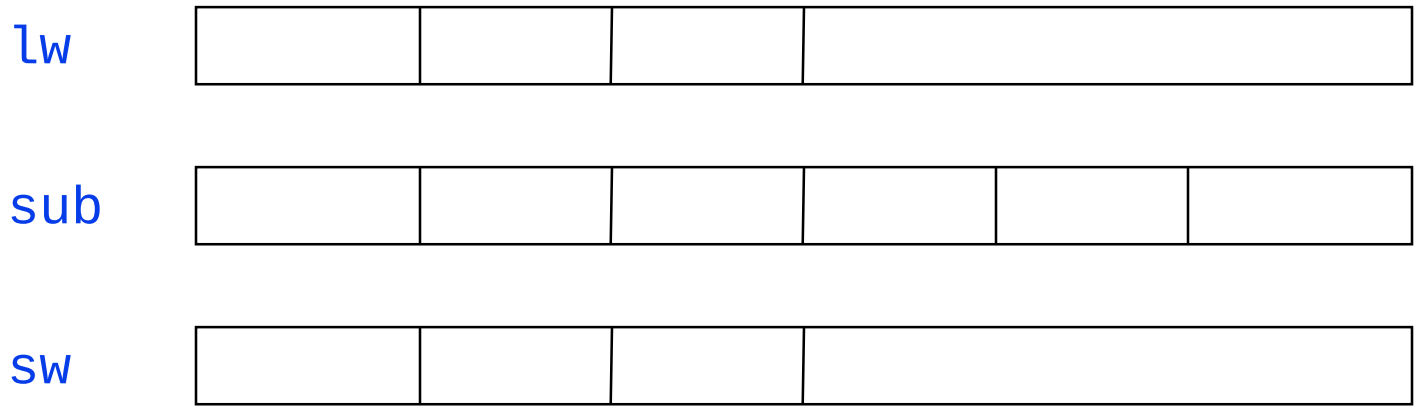
Assembling Code

- Remember the assembler code we compiled last lecture for the C statement

```
A[8] = A[2] - b
```

```
lw    $t0, 8($s3)        #load A[2] into $t0
sub   $t0, $t0, $s2      #subtract b from A[2]
sw    $t0, 32($s3)       #store result in A[8]
```

- Assemble the MIPS object code for these three instructions (decimal is fine)



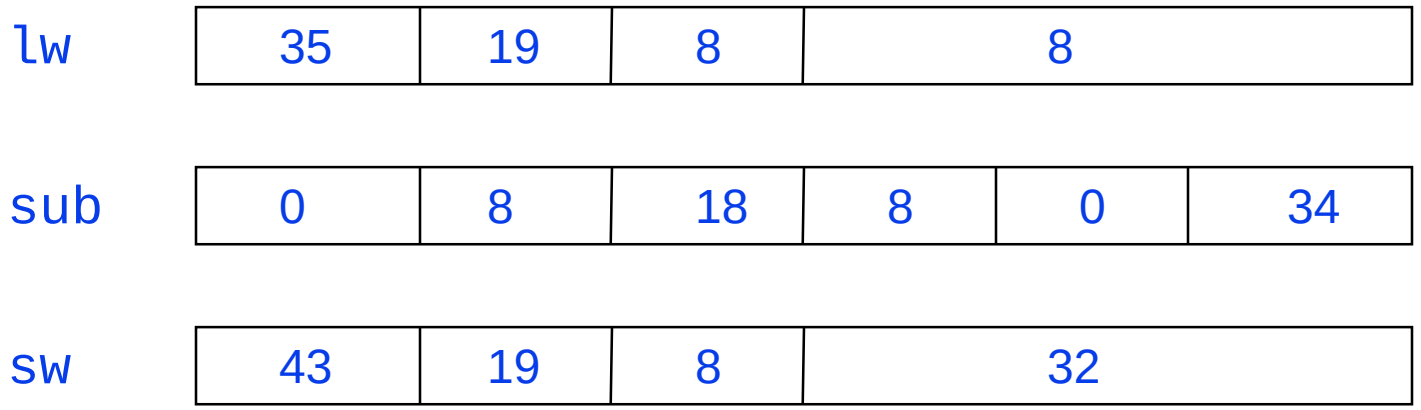
Assembling Code

- Remember the assembler code we compiled for the C statement

```
A[8] = A[2] - b

lw    $t0, 8($s3)        #load A[2] into $t0
sub   $t0, $t0, $s2      #subtract b from A[2]
sw    $t0, 32($s3)       #store result in A[8]
```

- Assemble the MIPS object code for these three instructions (decimal is fine)



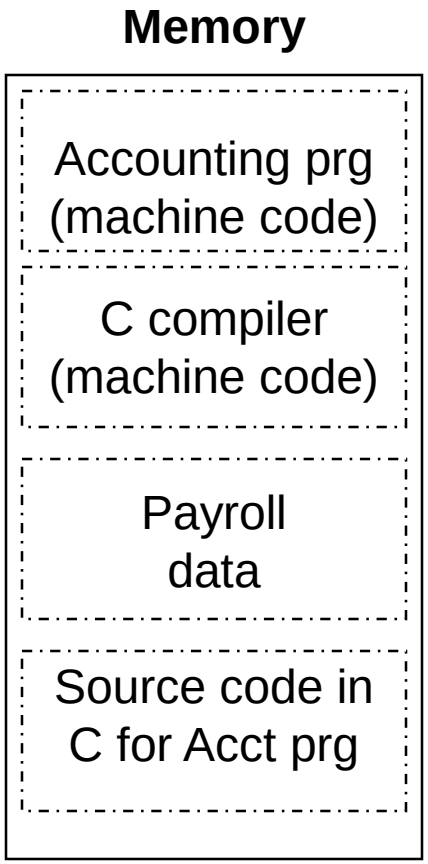
Review: MIPS Instructions, so far

Category	Instr	Op Code	Example	Meaning
Arithmetic (R format)	add	0 & 32	add $s1, $s2, $s3	$s1 = $s2 + $s3
	subtract	0 & 34	sub $s1, $s2, $s3	$s1 = $s2 - $s3
Arithmetic (I format)	add immediate	8	addi $s1, $s2, 4	$s1 = $s2 + 4
Data transfer (I format)	load word	35	lw $s1, 100($s2)	$s1 = Memory($s2+100)
	store word	43	sw $s1, 100($s2)	Memory($s2+100) = $s1

Two Key Principles of Machine Design

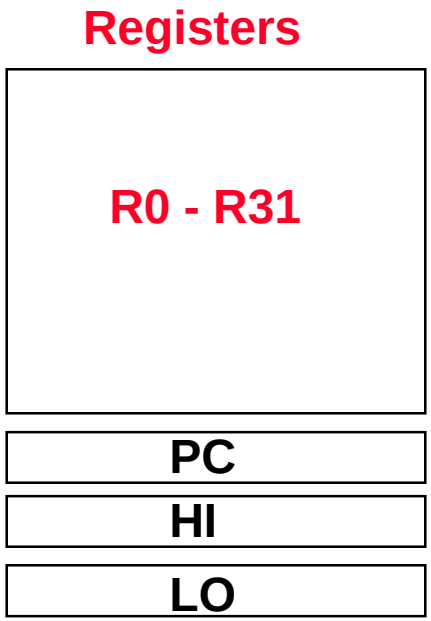
1. Instructions are represented as numbers
2. Programs are stored in memory to be read or written, just like numbers

- Stored-program concept
  - Programs can be shipped as files of binary numbers
  - Computers can inherit ready-made software provided they are compatible with an existing ISA – leads industry to align around a small number of ISAs



Review: MIPS R3000 ISA

- Instruction Categories
  - Load/Store
  - Computational
  - Jump and Branch
  - Floating Point
    - coprocessor
  - Memory Management
  - Special



- 3 Instruction Formats: all 32 bits wide