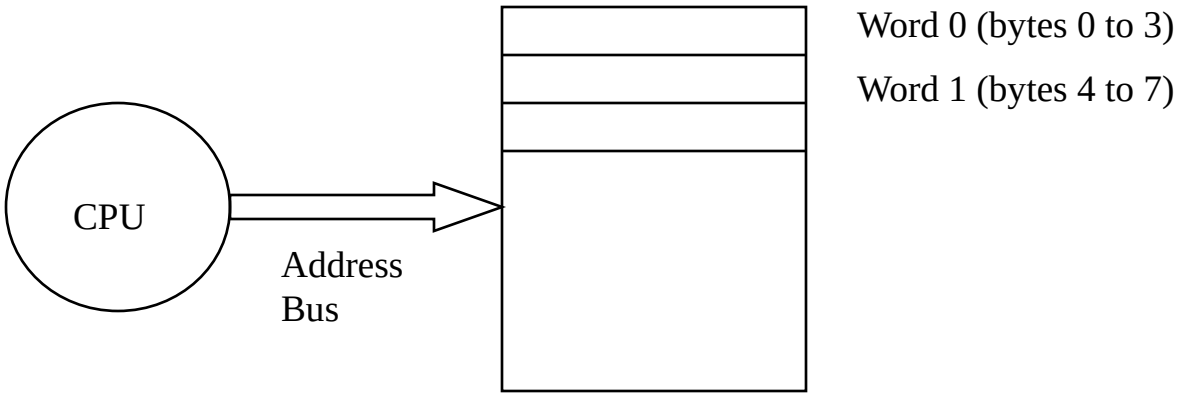
Review

- Instruction Set Overview
  - Classifying Instruction Set Architectures (ISAs)

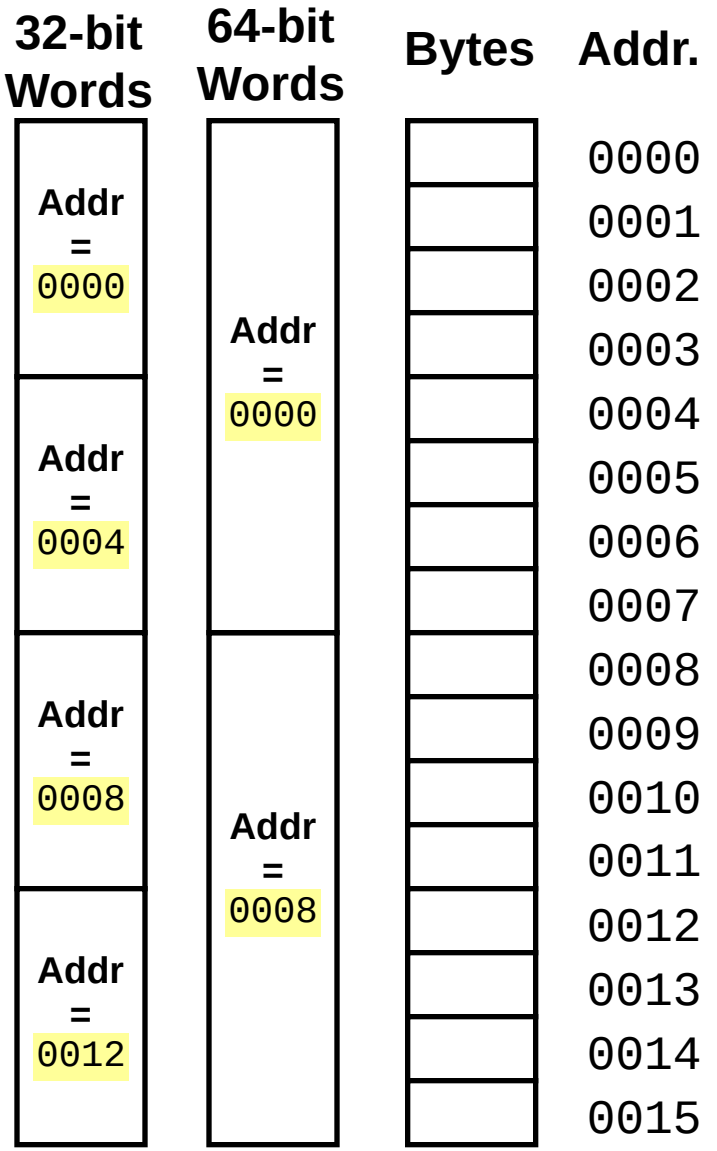
Memory Addressing

- Since 1980 almost every machine uses addresses to level of 8-bits (byte)
- 2 questions for design of ISA:
  - Read a 32-bit word as four loads of bytes from sequential byte addresses or as one load word from a single byte address, how do byte addresses map onto words?
  - Can a word be placed on any byte boundary?



Memory Addressing

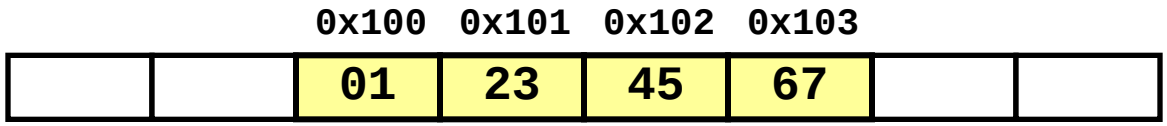
- Memory is byte addressed and provides access for bytes (8 bits), half words (16 bits), words (32 bits), and double words(64 bits).
- Addresses Specify Byte Locations
  - Address of the first byte in word
  - Successive word addresses differ by 4 (32-bit)



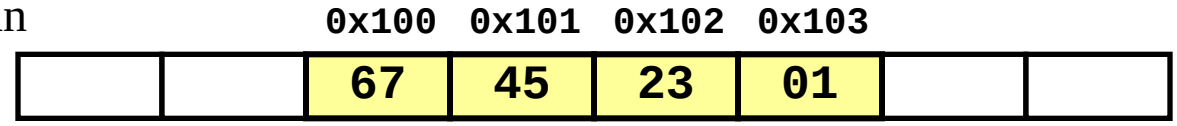
Addressing Objects: Endianess and Alignment

- Big Endian: address of most significant byte = word address
  (xx00 = Big End of word)
  - IBM 360/370, Motorola 68k, MIPS, Sparc, HP PA
- Little Endian: address of least significant byte = word address
  (xx00 = Little End of word)
  - Intel 80x86, DEC Vax, DEC Alpha (Windows NT)

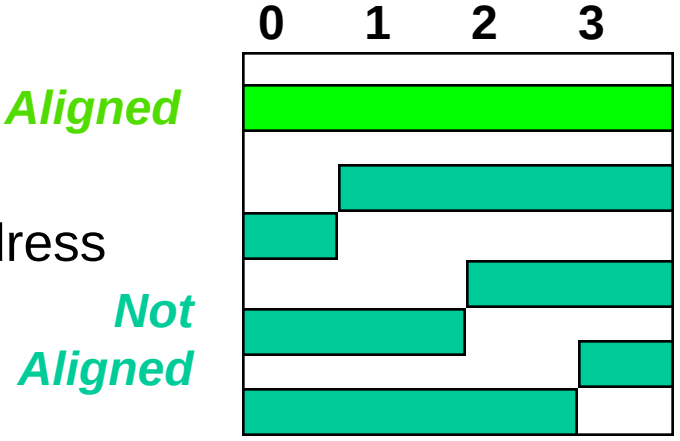
Big Endian



Little Endian



Alignment: require that objects fall on address that is multiple of their size.



Reading Byte-Reversed Listings

- Disassembly
  - Text representation of binary machine code
  - Generated by program that reads the machine code
- Example Fragment

Address	Instruction Code	Assembly Rendition
8048365:	5b	pop %ebx
8048366:	81 c3 ab 12 00 00	add $0x12ab,%ebx
804836c:	83 bb 28 00 00 00 00	cmpl $0x0,0x28(%ebx)

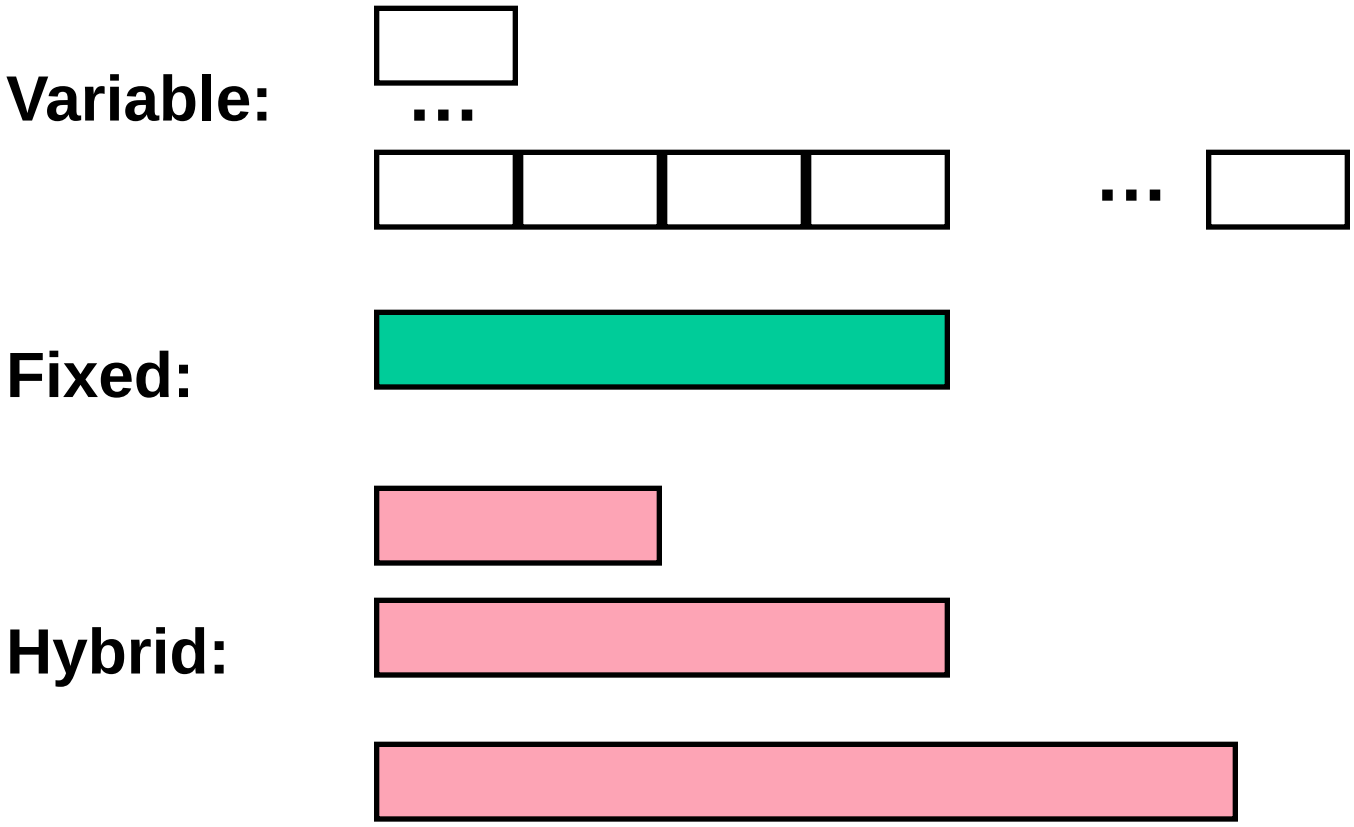
- Deciphering Numbers
  - Value: 0x12ab
  - Pad to 4 bytes: 0x000012ab
  - Split into bytes: 00 00 12 ab
  - Reverse: ab 12 00 00

Types of Addressing Modes (VAX)

	Addressing Mode	Example	Action
1.	Register direct	Add R4, R3	R4 <- R4 + R3
2.	Immediate	Add R4, #3	R4 <- R4 + 3
3.	Displacement	Add R4, 100(R1)	R4 <- R4 + M[100 + R1]
4.	Register indirect	Add R4, (R1)	R4 <- R4 + M[R1]
5.	Indexed	Add R4, (R1 + R2)	R4 <- R4 + M[R1 + R2]
6.	Direct	Add R4, (1000)	R4 <- R4 + M[1000]
7.	Memory Indirect	Add R4, @(R3)	R4 <- R4 + M[M[R3]]
8.	Autoincrement	Add R4, (R2)+	R4 <- R4 + M[R2] R2 <- R2 + d
9.	Autodecrement	Add R4, (R2)-	R4 <- R4 + M[R2] R2 <- R2 - d
10.	Scaled	Add R4, 100(R2)[R3]	R4 <- R4 + M[100 + R2 + R3*d]

- Studies by [Clark and Emer] indicate that modes 1-4 account for 93% of all operands on the VAX.

Generic Examples of Instruction Formats



Instruction Formats

- If code size is most important, use variable length instructions:
  (1)Difficult control design to compute next address
  (2) complex operations, so use microprogramming
  (3) Slow due to several memory accesses

- If performance is most important, use fixed length instructions
  (1) Simple to decode, so use hardware
  (2) Works well with pipelining
  (3) Wastes code space because of simple operations

- Recent embedded machines (ARM, MIPS) added optional mode to execute subset of 16-bit wide instructions (Thumb, MIPS16); per procedure decide performance or density

Types of Operations

- Arithmetic and Logic: AND, ADD
- Data Transfer: MOVE, LOAD, STORE
- Control BRANCH, JUMP, CALL
- System OS CALL, VM
- Floating Point ADDF, MULF, DIVF
- Decimal ADDD, CONVERT
- String MOVE, COMPARE
- Graphics (DE)COMPRESS

80x86 Instruction Frequency

Rank	*Instruction*	*Frequency*
1	load	22%
2	branch	20%
3	compare	16%
4	store	12%
5	add	8%
6	and	6%
7	sub	5%
8	register move	4%
9	call	1%
10	return	1%
Total		96%

Relative Frequency of Control Instructions

Operation	SPECint92	SPECfp92
Call/Return	13%	11%
Jumps	6%	4%
Branches	81%	87%

What is the indication of the relative frequency?

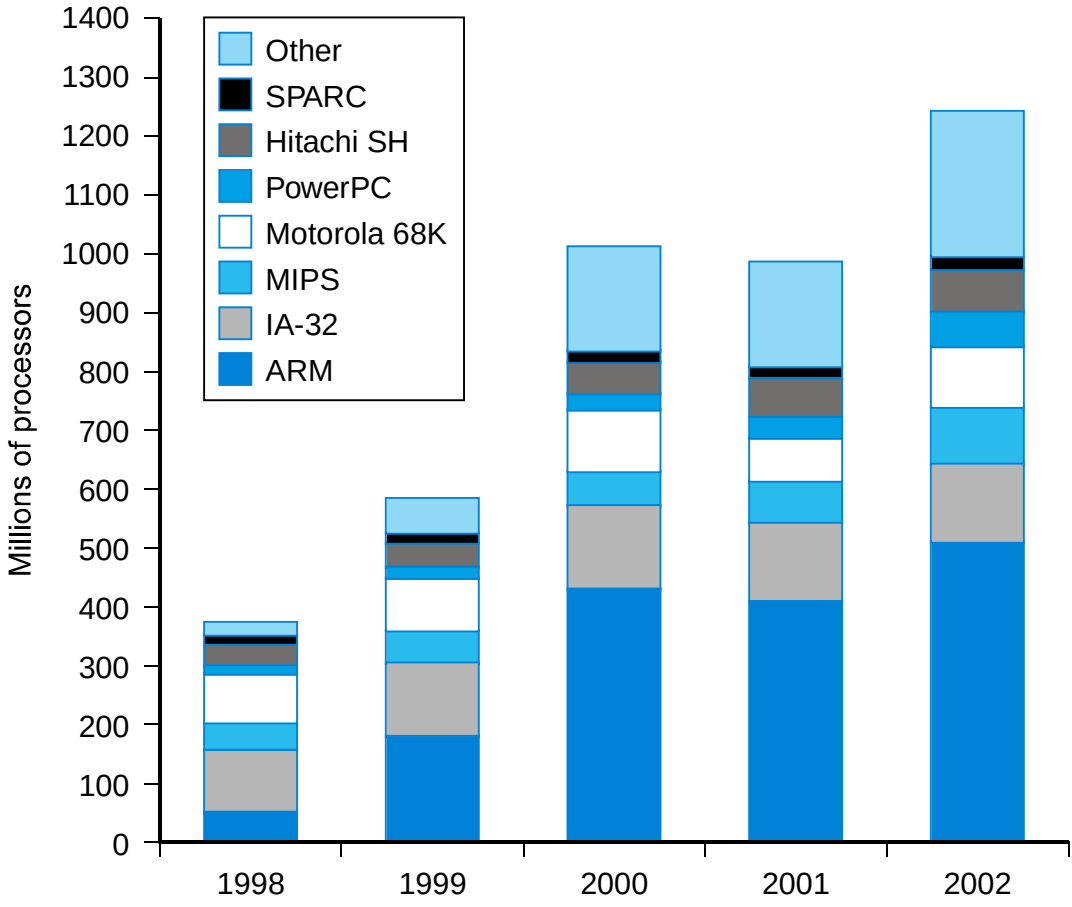
- Design hardware to handle branches quickly, since these occur most frequently

Summary

- Instruction Set Overview
  - Classifying Instruction Set Architectures (ISAs)
  - Memory Addressing
  - Types of Instructions

MIPS: A Case Study of Instruction Set Architecture

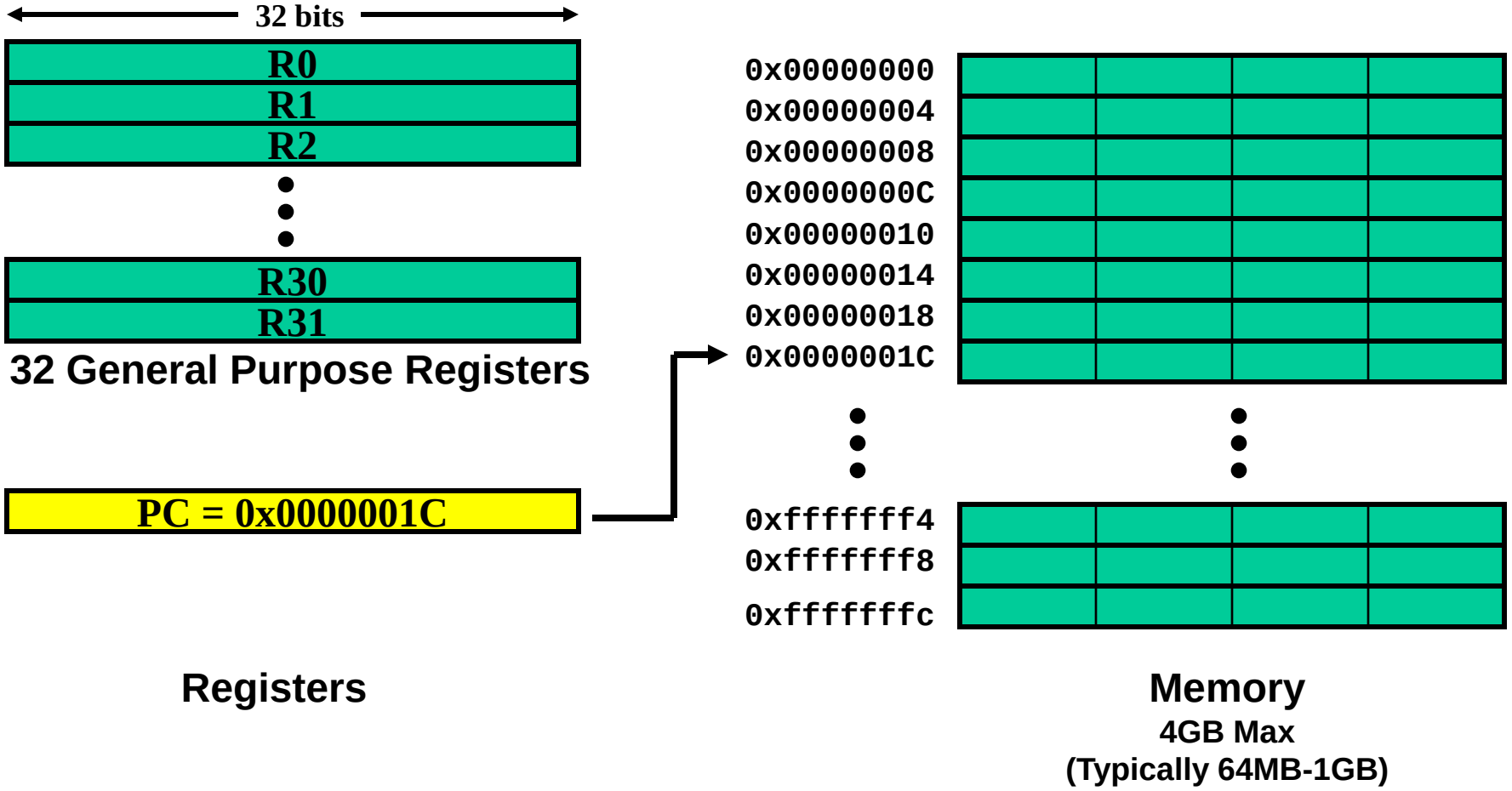
- MIPS: Microprocessor without Interlocked Pipeline Stages
- We'll be working with the MIPS instruction set architecture
  - similar to other architectures developed since the 1980's
  - Almost 100 million MIPS processors manufactured in 2002
  - used by NEC, Nintendo, Cisco, Silicon Graphics, Sony, …



MIPS Design Principles

1. Simplicity Favors Regularity
   - Keep all instructions a single size
   - Always require three register operands in arithmetic instructions
2. Smaller is Faster
   - Has only 32 registers rater than many more
3. Good Design Makes Good Compromises
   - Comprise between providing larger addresses and constants instruction and keeping instruction the same length
4. Make the Common Case Fast
   - PC-relative addressing for conditional branches
   - Immediate addressing for constant operands

MIPS Registers and Memory



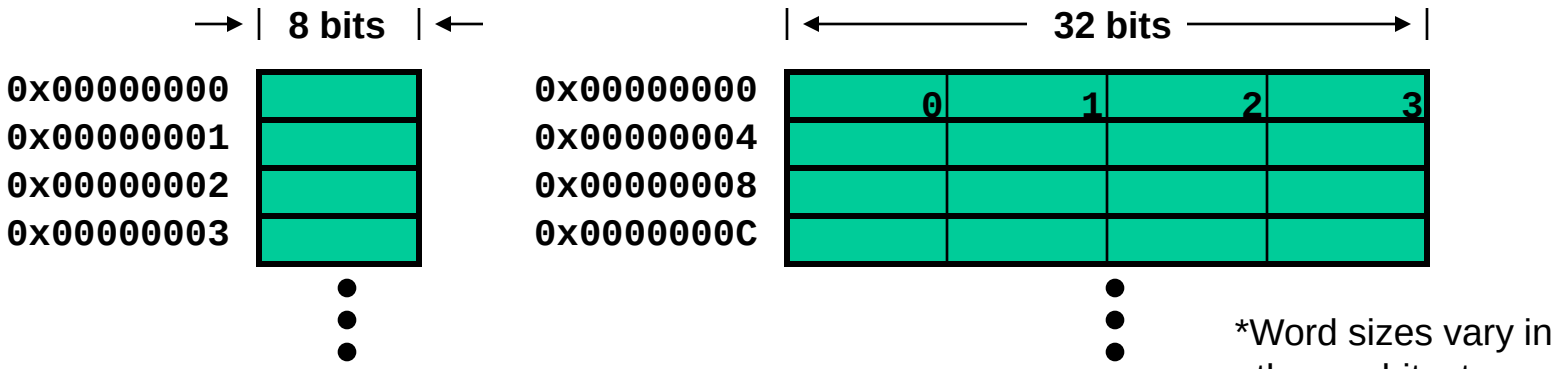
MIPS Registers and Usage

Name	Register number	Usage
$zero	0	the constant value 0
$at	1	reserved for assembler
$v0-$v1	2-3	values for results and expression evaluation
$a0-$a3	4-7	arguments
$t0-$t7	8-15	temporary registers
$s0-$s7	16-23	saved registers
$t8-$t9	24-25	more temporary registers
$k0-$k1	26-27	reserved for Operating System kernel
$gp	28	global pointer
$sp	29	stack pointer
$fp	30	frame pointer
$ra	31	return address

Each register can be referred to by number or name.

MIPS Memory Organization

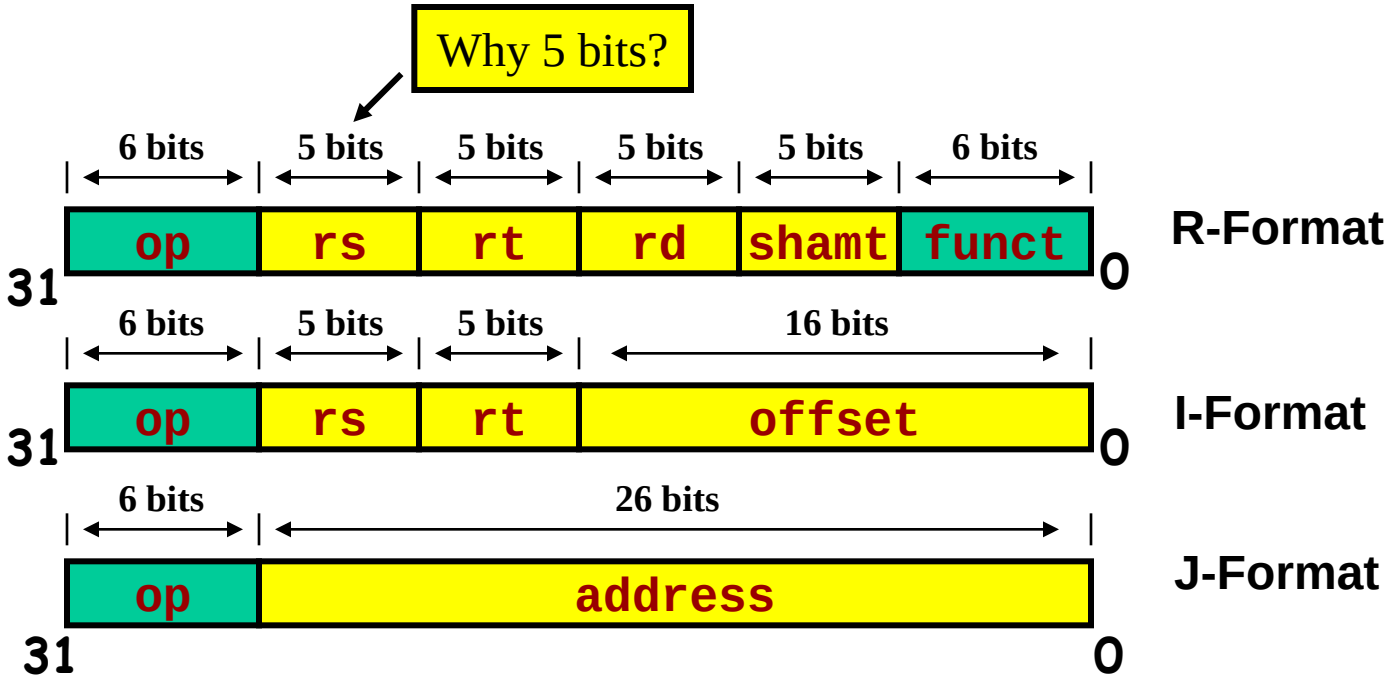
- Two views of memory:
  - 2^{32} bytes with addresses 0, 1, 2, …, 2^{32}-1
  - 2^{30} 4-byte words* with addresses 0, 4, 8, …, 2^{32}-4
- Both views use byte addresses
- Word address must be multiple of 4 (aligned)



*Word sizes vary in other architectures

MIPS Instructions

- All instructions exactly 32 bits wide
- Different formats for different purposes
- Similarities in formats ease implementation



MIPS Instruction Types

Name	Register number	Usage
$zero	0	the constant value 0
$at	1	reserved for assembler
$v0-$v1	2-3	values for results and expression evaluation
$a0-$a3	4-7	arguments
$t0-$t7	8-15	temporary registers
$s0-$s7	16-23	saved registers
$t8-$t9	24-25	more temporary registers
$k0-$k1	26-27	reserved for Operating System kernel
$gp	28	global pointer
$sp	29	stack pointer
$fp	30	frame pointer
$ra	31	return address

- Arithmetic & Lo registers

  add $s1, $s2, $s3
  or $s3, $s4, $s5

- Data Transfer - move register data to/from memory

  lw $s1, 100($s2) $s1 = Memory[$s2 + 100]
  sw $s1, 100($s2) Memory[$s2 + 100] = $s1

- Branch - alter program flow

  beq $s1, $s2, 25 if ($s1==$s2) PC = PC + 4 + 4*25

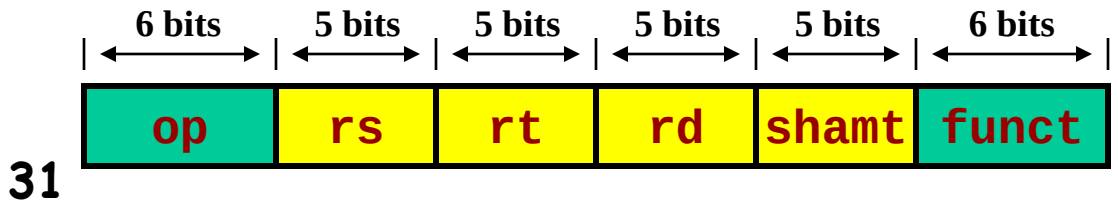
MIPS Arithmetic & Logical Instructions

- Instruction usage (assembly)

add dest, src1, src2	dest=src1 + src2
sub dest, src1, src2	dest=src1 - src2
and dest, src1, src2	dest=src1 AND src2

- Instruction characteristics
  - Always 3 operands: destination + 2 sources
  - Operand order is fixed
  - Operands are always general purpose registers
- Design Principles:
  - Design Principle 1: Simplicity favors regularity
  - Design Principle 2: Smaller is faster

Arithmetic & Logical Instructions - Binary Representation



add dest, src1, src2

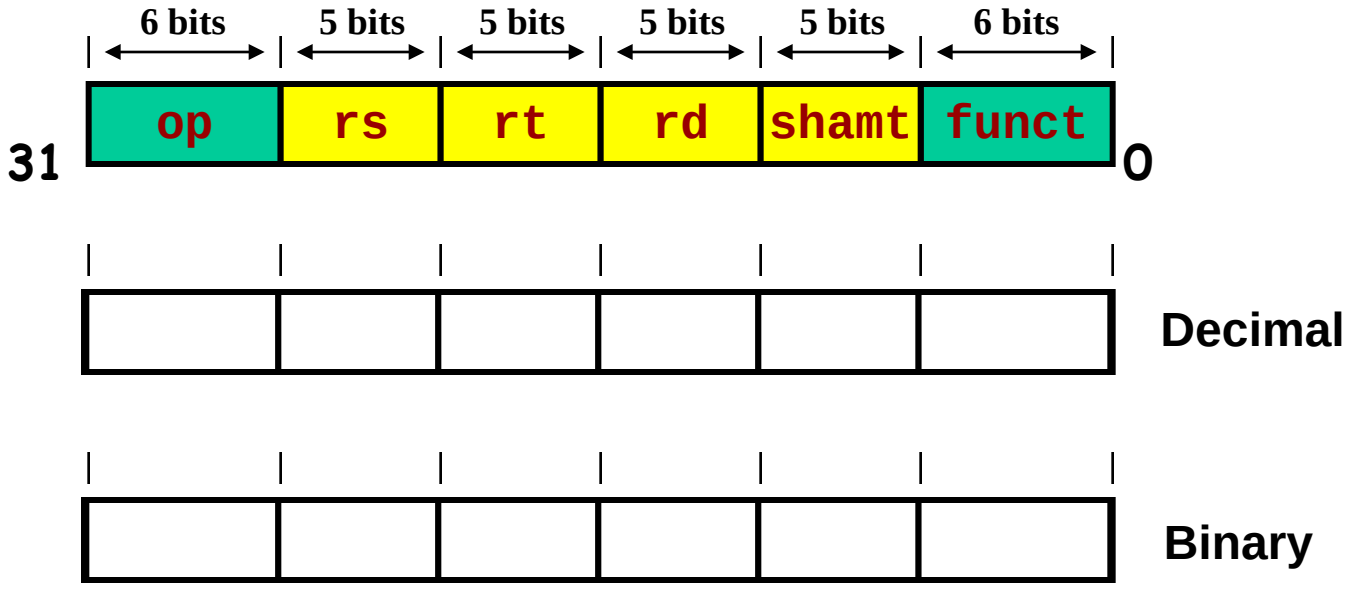
- Used for arithmetic, logical, shift instructions
  - `op`: Basic operation of the instruction (*opcode*)
  - `rs`: first register source operand
  - `rt`: second register source operand
  - `rd`: register destination operand
  - `shamt`: shift amount (more about this later)
  - `funct`: function - specific type of operation
- Also called "R-Format" or "R-Type" Instructions

Arithmetic & Logical Instructions - Binary Representation Example

- Machine language for

  add $8, $17, $18

- See reference card for `op`, `funct` values



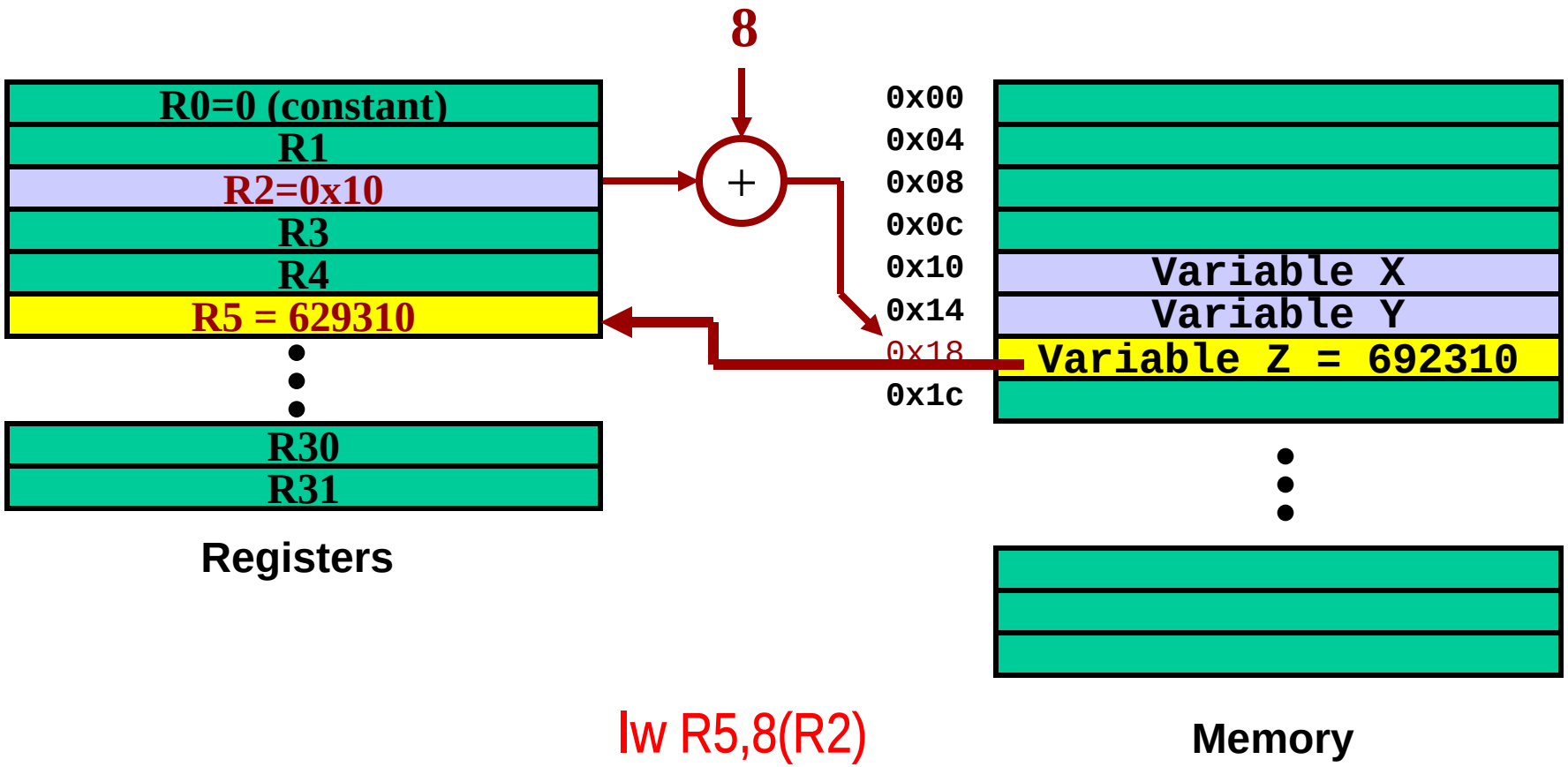
MIPS Data Transfer Instructions

- Transfer data between registers and memory
- Instruction format (assembly)

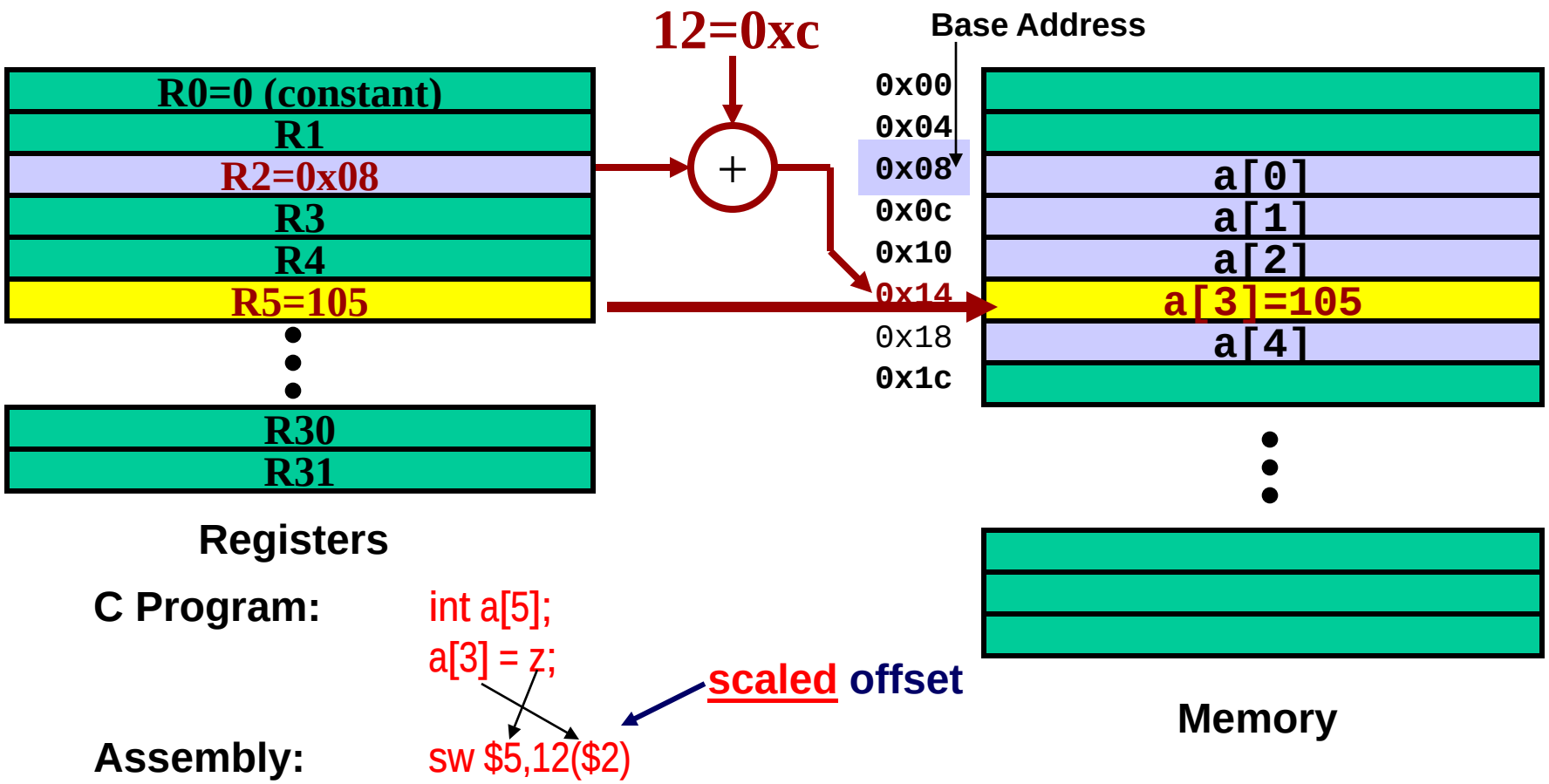
  lw $dest, offset($addr) load word
  sw $src, offset($addr) store word

- Uses:
  - Accessing a variable in main memory
  - Accessing an array element

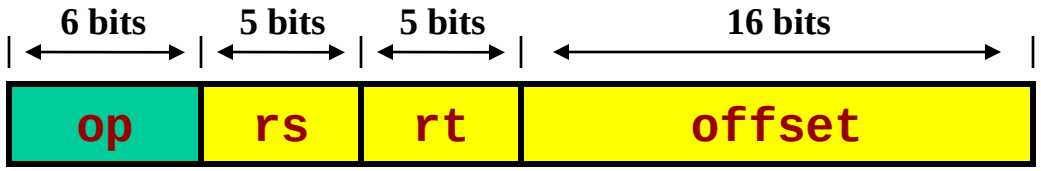
Example - Loading a Simple Variable



Data Transfer Example - Array Variable



Data Transfer Instructions - Binary Representation



- Used for load, store instructions
  - `op`: Basic operation of the instruction (*opcode*)
  - `rs`: first register source operand
  - `rt`: second register source operand
  - `offset`: 16-bit signed address offset (-32,768 to +32,767)
- Also called "I-Format" or "I-Type" instructions

I-Format vs. R-Format Instructions

- Compare with R-Format