CS352H Computer Systems Architecture

Lecture 2: Instruction Set Architectures I

September 1, 2009

ISA is a Contract

- Between the programmer and the hardware:
  - Defines visible state of the system
  - Defines how the state changes in response to instructions
- Programmer obtains a model of how programs will execute
- Hardware designer obtains a formal definition of the correct way to execute instructions
- ISA Specification:
  - Instruction set
  - How instructions modify the state of the machine
  - Binary representation
- Today: MIPS ISA

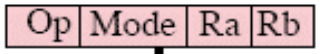
ISA is a Contract

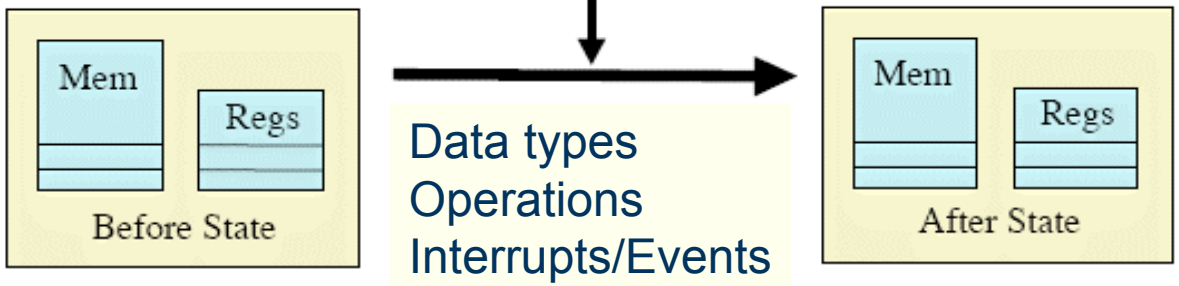
- Between the programmer and the hardware:
  - Defines visible state of the system
  - Defines how the state changes in response to instructions
- Programmer obtains a model of how programs will execute
- Hardware designer obtains a formal definition of the correct way to execute instructions
- ISA Specification:
  - Instruction set
  - How instructions modify the state of the machine
  - Binary representation
- Today: MIPS ISA

ISA Specification

Instruction formats
Instruction types
Addressing modes

Instruction Representation





Data types
Operations
Interrupts/Events

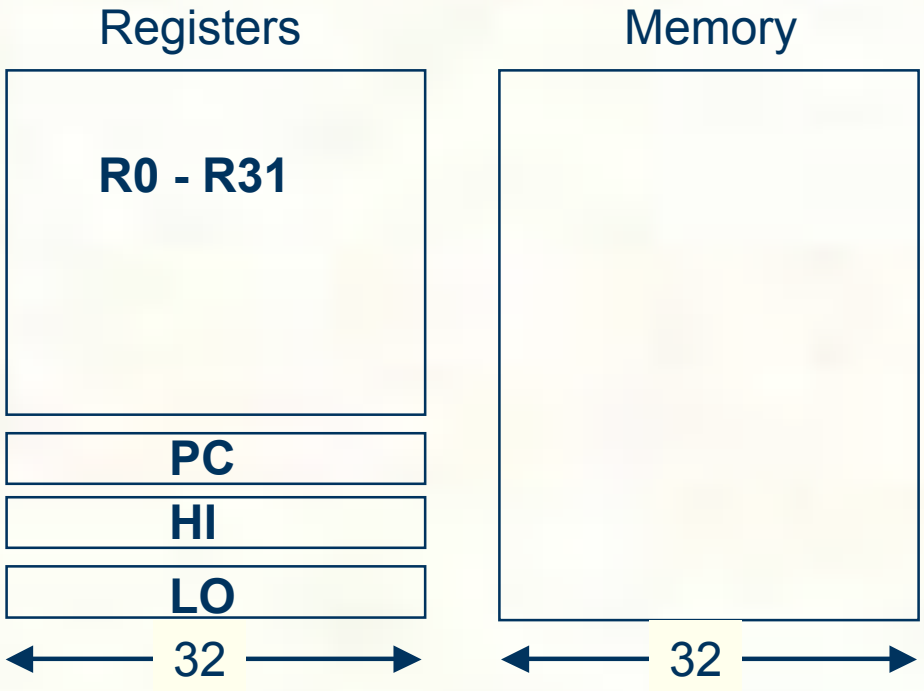
Machine state
Memory organization
Register organization

MIPS ISA

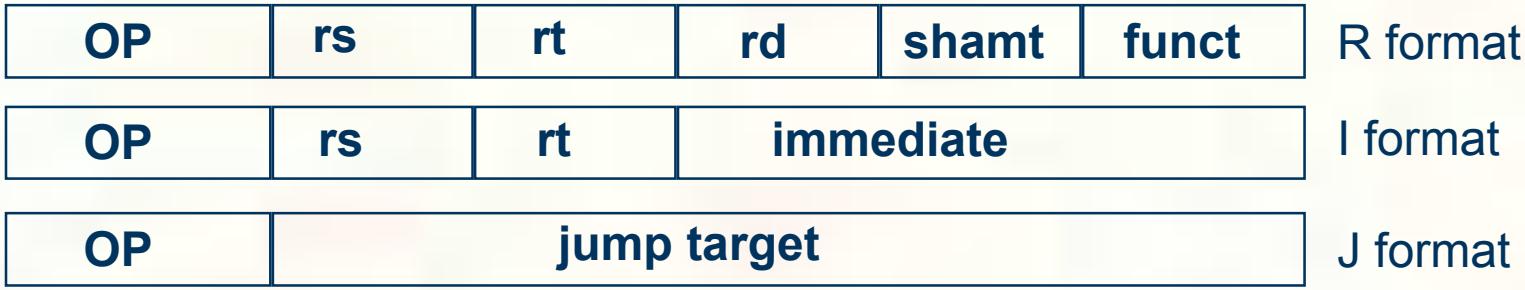
- 32 GP registers (R0-R31) – 32 bits each
- 32 FP registers (F0-F31) – 32 bits each
  - 16 double-precision (using adjacent 32-bit registers)
- 8-, 16-, and 32-bit integer & 32- and 64-bit floating point data types
- Load/Store architecture (no memory access in ALU ops)
- A few, simple addressing modes:
  - Immediate: R1 ← 0x21
  - Displacement: R1 ← 0x100(R2)
- Simple fixed instruction format
  - Three types
  - <100 instructions
- Fused compare and branch
- Pseudo instructions
- Designed for pipelining and ease of compilation

MIPS ISA: A Visual

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



3 Instruction Formats: all 32 bits wide



MIPS Register Convention

Name	Register Number	Usage	Preserve on call?
$zero	0	constant 0 (hardware)	n.a.
$at	1	reserved for assembler	n.a.
$v0 - $v1	2-3	returned values	no
$a0 - $a3	4-7	arguments	yes
$t0 - $t7	8-15	temporaries	no
$s0 - $s7	16-23	saved values	yes
$t8 - $t9	24-25	temporaries	no
$k0 - $k1	26-27	reserved for interrupts/traps	n.a.
$gp	28	global pointer	yes
$sp	29	stack pointer	yes
$fp	30	frame pointer	yes
$ra	31	return addr (hardware)	yes

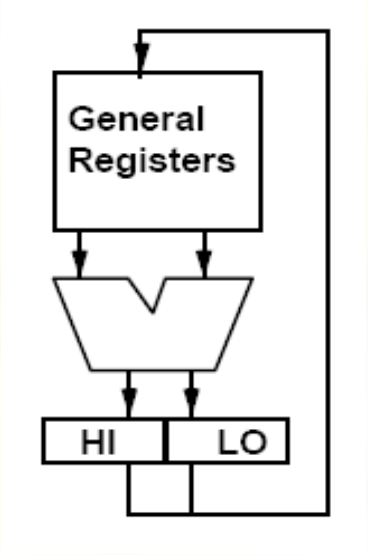
MIPS Arithmetic Instructions

Instruction	Example	Meaning	Comment
add	add $8, $9, $10	$8 = $9 + $10	3 opnds; Exception possible
subtract	sub $8, $9, $10	$8 = $9 - $10	3 opnds; Exception possible
add immediate	addi $8, $9, 100	$8 = $9 + 100	+ const; Exception possible
add unsigned	addu $8, $9, $10	$8 = $9 + $10	3 opnds; No exception
subtract unsigned	subu $8, $9, $10	$8 = $9 - $10	3 opnds; No exception
add imm. Unsig.	addiu $8, $9, 100	$8 = $9 + 100	+ const; No exception
multiply	mult $8, $9	Hi,Lo = $8 * $9	64-bit signed product
multiply unsigned	multu $8, $9	Hi,Lo = $8 * $9	64-bit unsigned product
divide	div $8, $9	Lo = $8 ÷$9 Hi = $8 mod $9	Lo = quotient Hi = remainder
divide unsigned	divu $8, $9	Lo = $8 ÷$9 Hi = $8 mod $9	Unsigned quotient & remainder
move from Hi	mfhi $8	$8 = Hi	
move from Lo	mflo $8	$8 = Lo	

Which add for address arithmetic? Which for integers?

Multiply & Divide

- Start multiply/Divide
  - mult rs, rt
  - multu rs, rt
  - div rs, rt
  - divu rs, rt
- Move result from Hi or Lo
  - mfhi rd
  - mflo rd
- Move to Hi or Lo (Why?)
  - mthi rd
  - mtlo rd



MIPS Logical Instructions

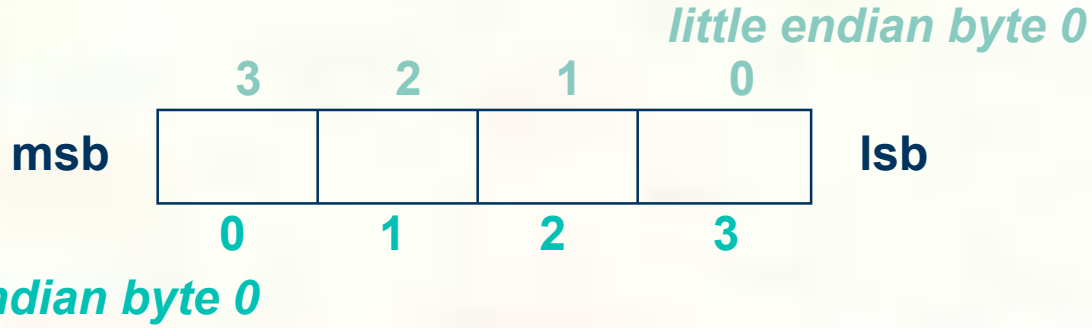
Instruction	Example	Meaning	Comment
and	and $8, $9, $10	$8 = $9 & $10	3 opnds; Logical AND
or	or $8, $9, $10	$8 = $9 | $10	3 opnds; Logical OR
xor	xor $8, $9, $10	$8 = $9 ⊕ $10	+ const; Logical XOR
nor	nor $8, $9, $10	$8 = ~($9 | $10)	3 opnds; Logical NOR
and immediate	andi $8, $9, 10	$8 = $9 & 10	Logical AND; Reg, Const
or immediate	ori $8, $9, 10	$8 = $9 | 10	Logical OR; Reg, Const
shift left logical	sll $8, $9, 10	$8 = $9 << 10	Shift left by constant
shift right logical	srl $8, $9, 10	$8 = $9 >> 10	Shift right by constant
shift right arith.	sra $8, $9, 10	$8 = $9 >> 10	Const srl with sign extension
shift left logical	sllv $8, $9, $10	$8 = $9 << $10	Shift left by variable
shift right logical	srlv $8, $9, $10	$8 = $9 >> $10	Shift right by variable
shift right arith.	srav $8, $9, $10	$8 = $9 >> $10	Var. srl with sign extension

Byte Addresses

- Since 8-bit bytes are so useful, most architectures address individual bytes in memory
  - The memory address of a word must be a multiple of 4 (alignment restriction)
- Big Endian: leftmost byte is word address

  IBM 360/370, Motorola 68k, MIPS, Sparc, HP PA
- Little Endian: rightmost byte is word address

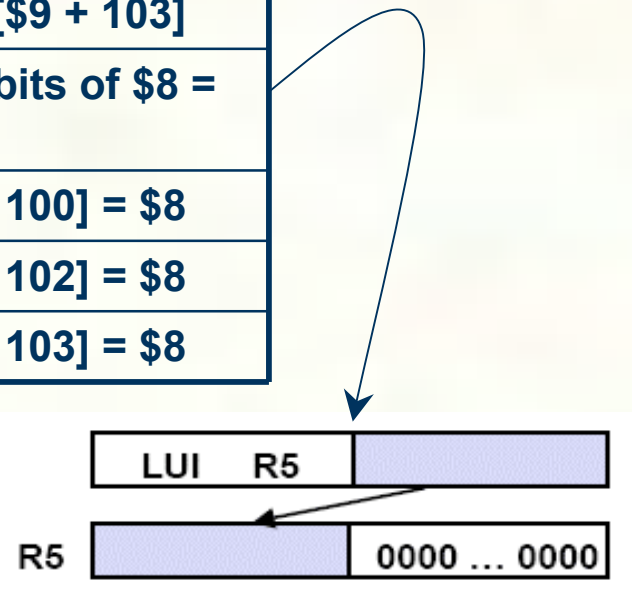
  Intel 80x86, DEC Vax, DEC Alpha (Windows NT)



- Nowadays endian-ness is configurable

MIPS Data Transfer Instructions

Instruction	Example	Meaning
load word	lw $8, 100($9)	$8 = Mem[$9 + 100]
load half word	lh $8, 102($9)	$8 = Mem[$9 + 102]
load half word unsigned	lhu $8, 102($9)	$8 = Mem[$9 + 102]
load byte	lb $8, 103($9)	$8 = Mem[$9 + 103]
load byte unsigned	lbu $8, 103($9)	$8 = Mem[$9 + 103]
load upper immediate	lui $8, 47	Upper 16 bits of $8 = 47
store word	sw $8, 100($9)	Mem[$9 + 100] = $8
store half word	sh $8, 102($9)	Mem[$9 + 102] = $8
store byte	sb $8, 103($9)	Mem[$9 + 103] = $8



- Where do half words & bytes get placed on lh & lb?
- What happens to the rest of the word on sb?

MIPS Compare and Branch

- Compare and branch
  - beq rs, rt, offset if R[rs] == R[rt] then PC-relative branch
  - bne rs, rt, offset !=
- Compare to zero and branch
  - blez rs, offset if R[rs] <= 0 then PC-relative branch
  - bgtz rs, offset >
  - bltz <
  - bgez >=
  - bltzal rs, offset if R[rs] < 0 then branch and link (into R31)
  - bgezal >=
- Remaining compare and branches take two instructions
- Almost all comparisons are against zero
  - Hence $0 is always 0!

MIPS Jump, Branch & Compare Instructions

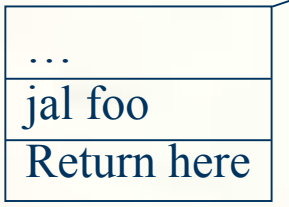
Instruction	Example	Meaning
branch on equal	beq $8, $9, 100	If ($8 == $9) goto PC+4+100
branch on not equal	bne $8, $9, 100	If ($8 != $9) goto PC+4+100
set on less than	slt $8, $9, $10	If ($9 < $10) then $8 = 1 else $8 = 0;
set less than immed.	slti $8, $9, 100	If ($9 < 100) then $8 = 1 else $8 = 0;
set less than unsig.	sltu $8, $9, $10	If ($9 < $10) then $8 = 1 else $8 = 0;
set less than immed. unsig.	sltiu $8, $9, 100	If ($9 < 100) then $8 = 1 else $8 = 0;
jump	j 10004	goto 10004
jump register	jr $31	goto $31
jump and link	jal 10004	$31 = PC + 4; goto 10004

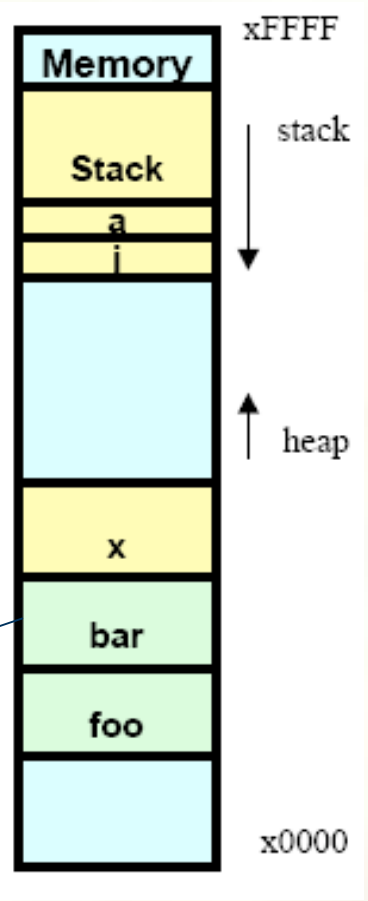
Procedure Support

- Caller
  - Save $t0 - $t9, if needed
  - Save $fp
  - Push arguments on stack (and leave in $a0-$a3)
  - Set $fp to point to arg's
  - Adjust $sp
  - jal
- Callee
  - Save $s0-$s7, $ra as needed
  - Get arg's
  - Do its thing
  - Put return values in $v0, $v1
  - jr $ra

- Preserved state
  - $s0 - $s7
  - $sp
  - $ra

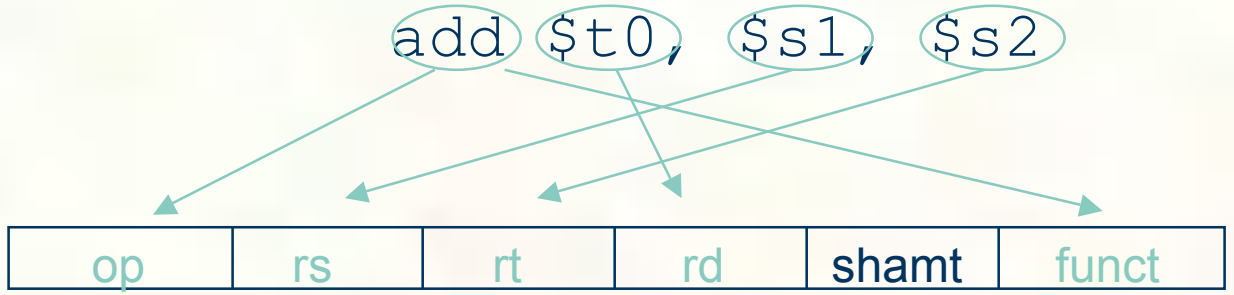
- Not preserved
  - $t0 - $t9
  - $a0 - $a3
  - $v0, $v1





Machine Language: R-format

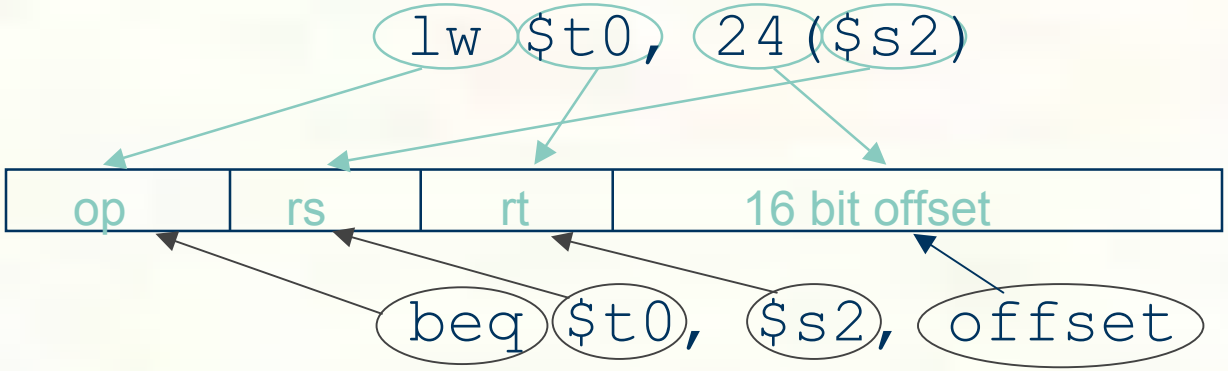
- R format instructions: three operands



op	6-bits	opcode that specifies the operation
rs	5-bits	register file address of the first source operand
rt	5-bits	register file address of the second source operand
rd	5-bits	register file address of the result's destination
shamt	5-bits	shift amount (for constant shift instructions)
funct	6-bits	function code augmenting the opcode

Machine Language: I-format

- I-format Instructions:
  - Load/store
  - Branches
  - Arithmetic with an immediate operand



op	6-bits	opcode that specifies the operation
rs	5-bits	register file address of the first source operand
rt	5-bits	register file address of the second source operand
offset	16-bits	a constant value

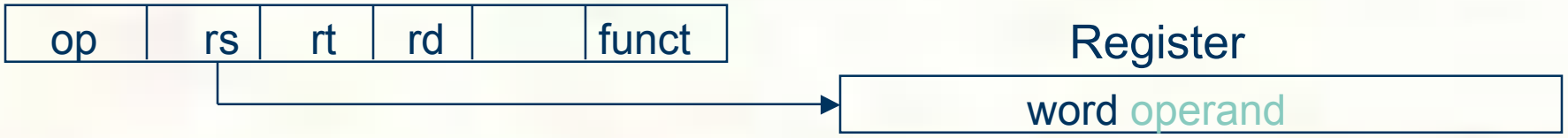
Machine Language: J-format

- J-format instructions:
  - Jump
  - Jump and Link

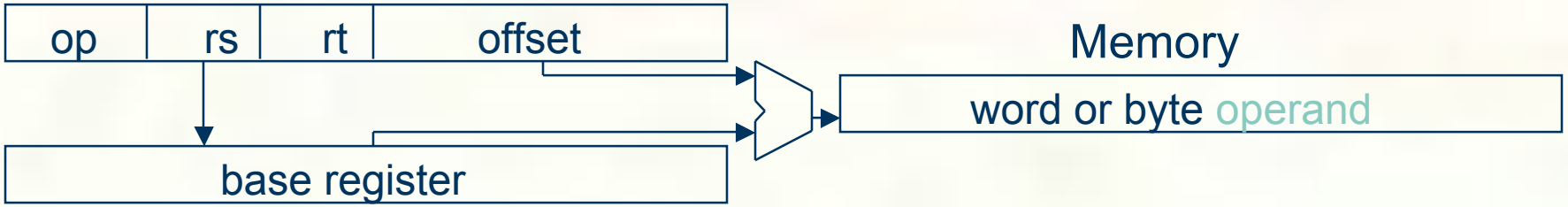
- What about Jump Register?

MIPS Operand Addressing Modes

- Register addressing – operand is in a register



- Base (displacement) addressing – operand is at the memory location whose address is the sum of a register and a 16-bit constant contained within the instruction

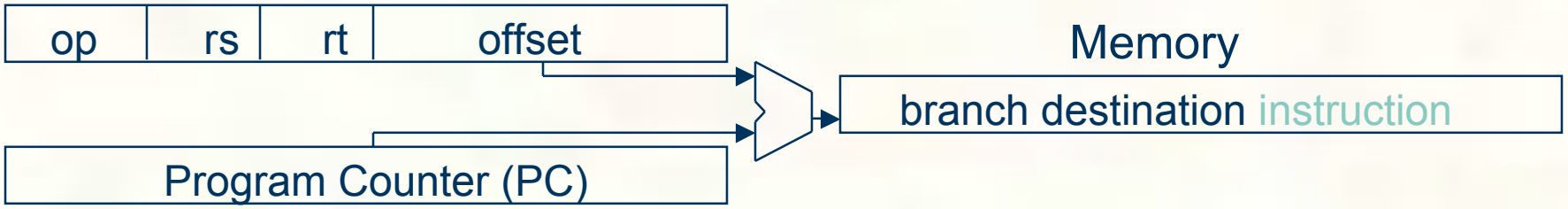


  - Register relative (indirect) with 0($a0)
  - Pseudo-direct with addr($zero)

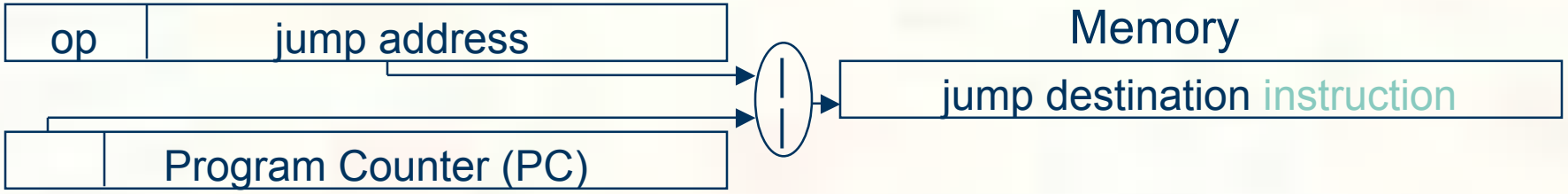
- Immediate addressing – operand is a 16-bit constant contained within the instruction

MIPS Instruction Addressing Modes

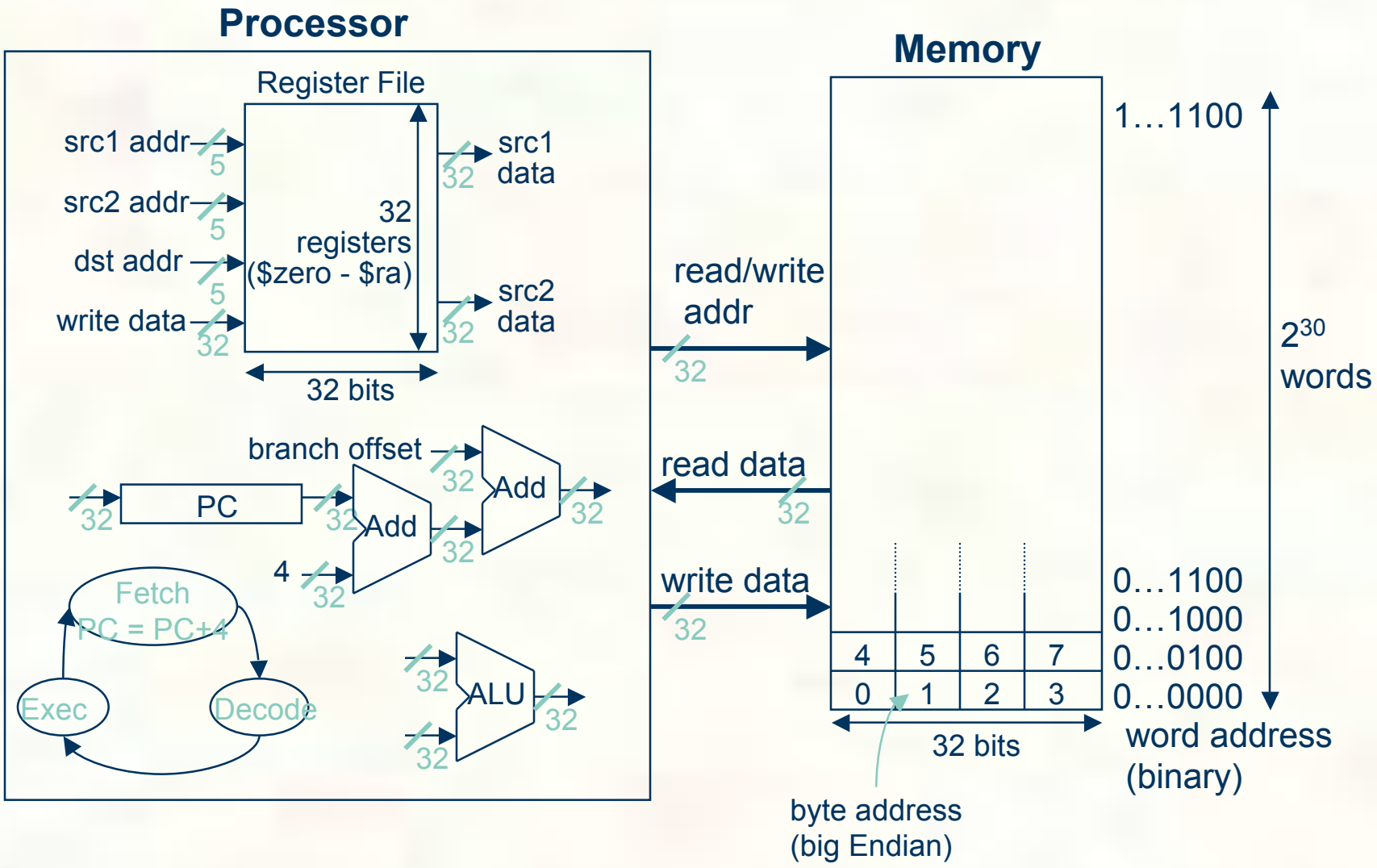
- PC-relative addressing –instruction address is the sum of the PC and a 16-bit constant contained within the instruction



- Pseudo-direct addressing – instruction address is the 26-bit constant contained within the instruction concatenated with the upper 4 bits of the PC



MIPS Organization So Far



MIPS (RISC) Design Principles

- Simplicity favors regularity
  - fixed size instructions – 32-bits
  - small number of instruction formats
  - opcode always the first 6 bits
- Good design demands good compromises
  - three instruction formats
- Smaller is faster
  - limited instruction set
  - limited number of registers in register file
  - limited number of addressing modes
- Make the common case fast
  - arithmetic operands from the register file (load-store machine)
  - allow instructions to contain immediate operands

Next Lecture

- ISA Principles
- ISA Evolution
- Make sure you're comfortable with the contents of Ch. 2
- You will need to read Appendix B to do the programming part of the assignment