Instruction Set Architecture I

Instructor: Dmitri A. Gusev

Fall 2007

CS 502: Computers and Communications

Lecture 2, September 10, 2007

SPIM MIPS Simulator

- Download the SPIM simulator from http://pages.cs.wisc.edu/~larus/spim.html
- spim implements almost the entire MIPS32 assembler-extended instruction set. (It omits most floating point comparisons and rounding modes and the memory system page tables.) MIPS compilers also generate a number of assembler directives that spim cannot process. These directives usually can be safely deleted.
- Sample programs: https://www.cs.tcd.ie/John.Waldron/itral/source/source.html

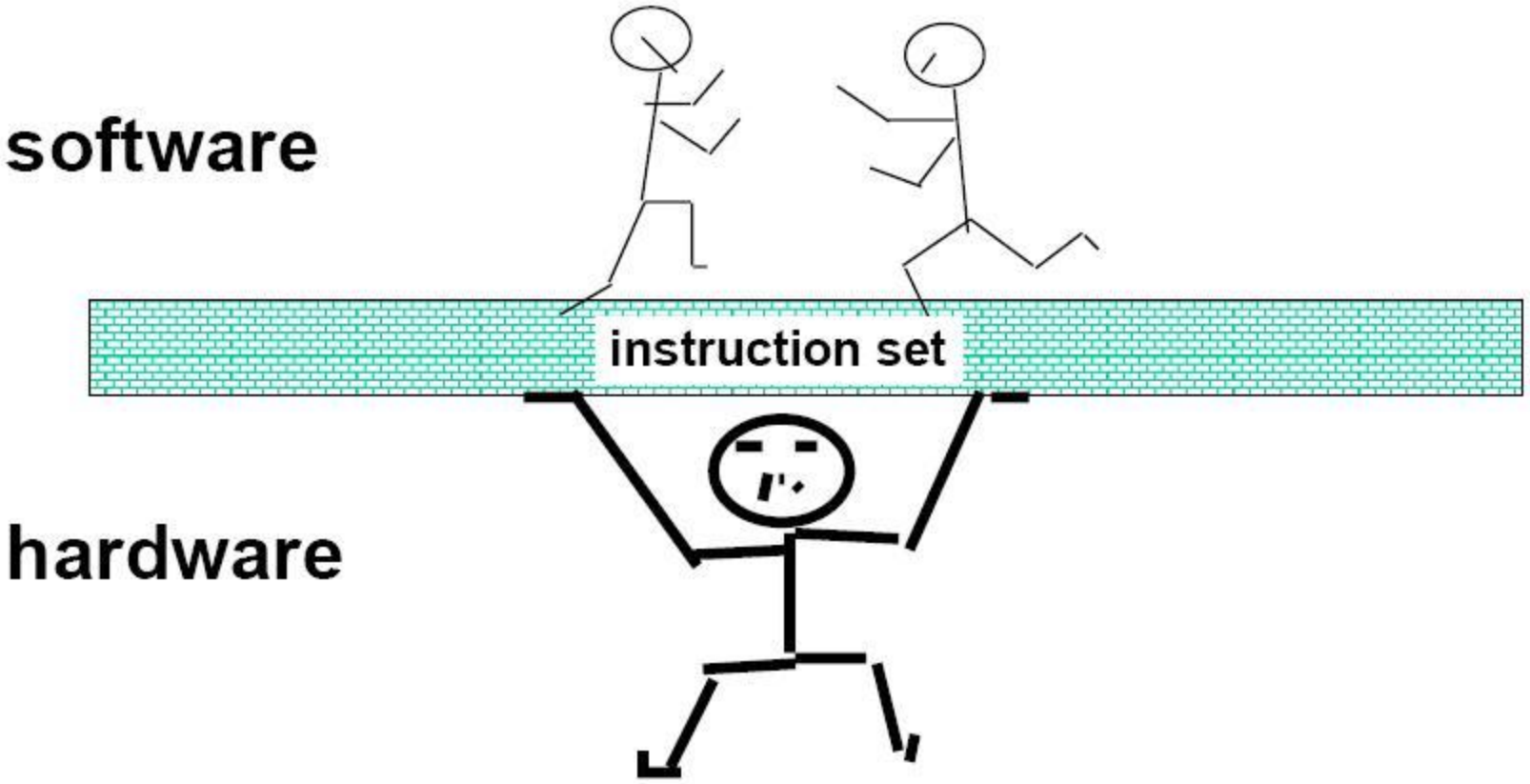
What is "Computer Architecture"

Computer Architecture =

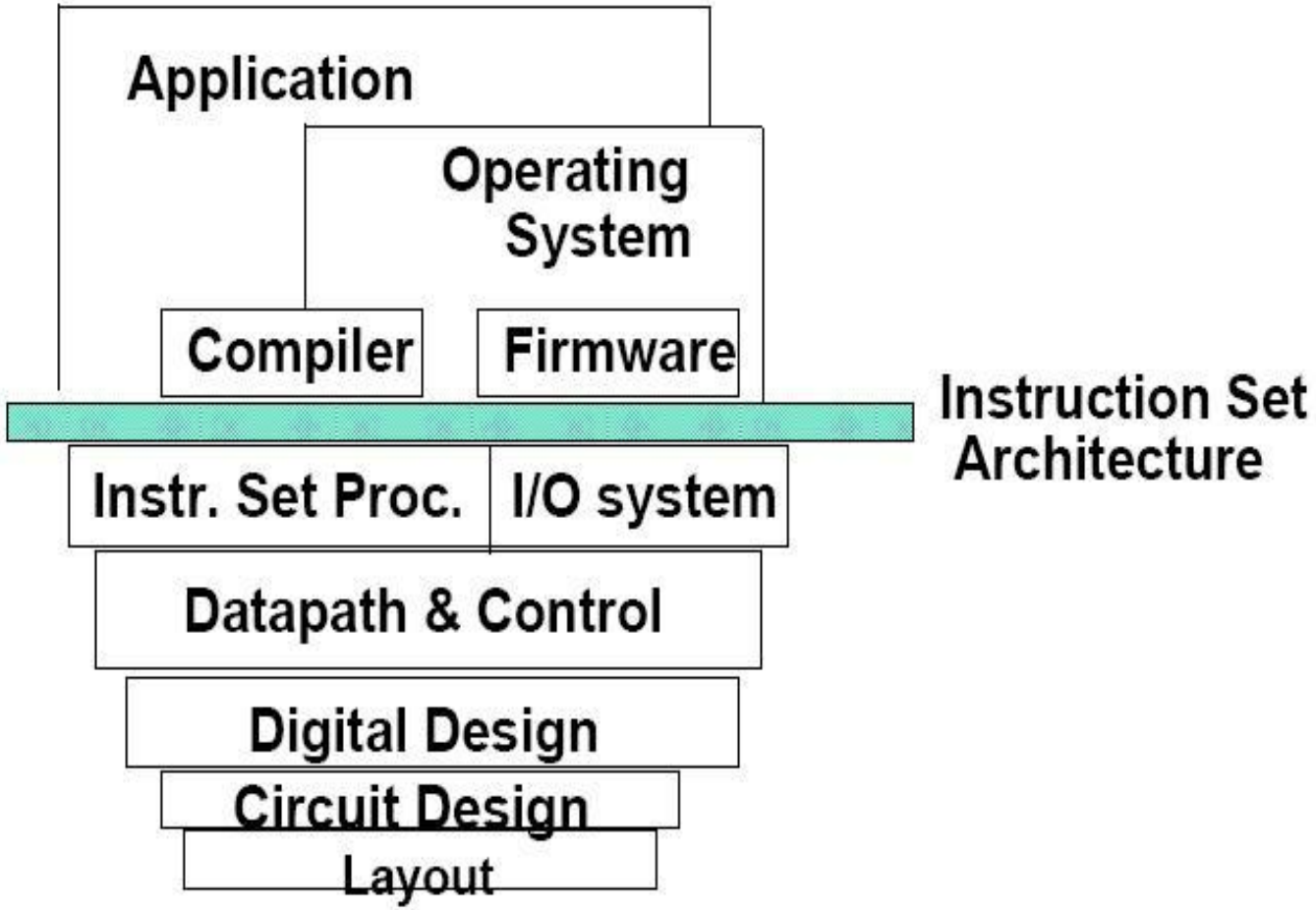
Instruction Set Architecture +

Machine Organization

Where The Instruction Set Belongs

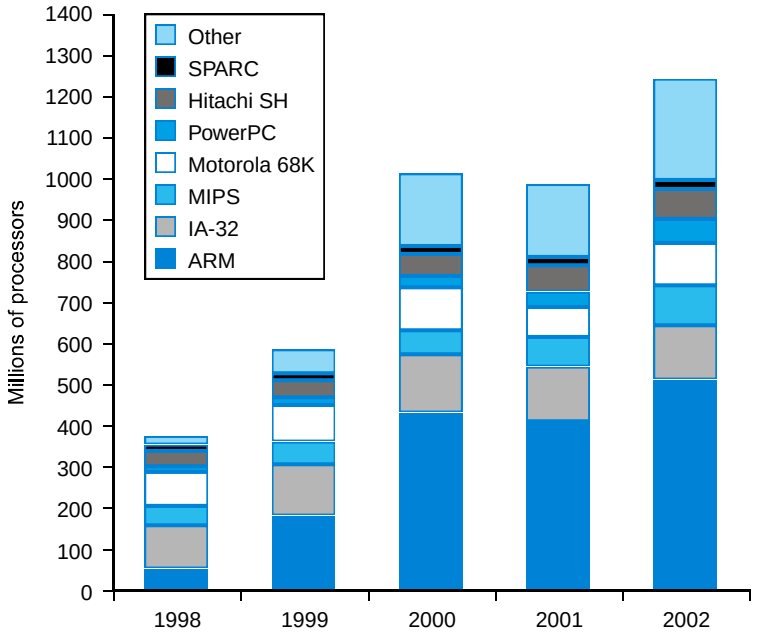


Development: Down & Up From The Instruction Set Architecture

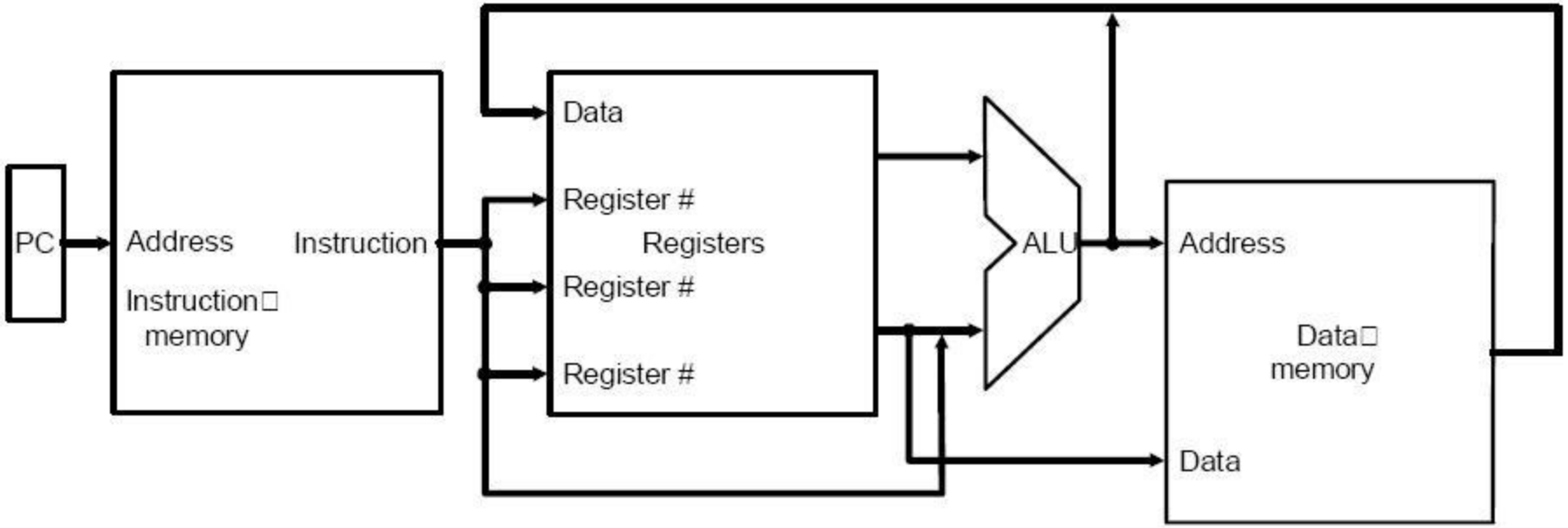


Instructions:

- Language of the Machine
- 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, **...**



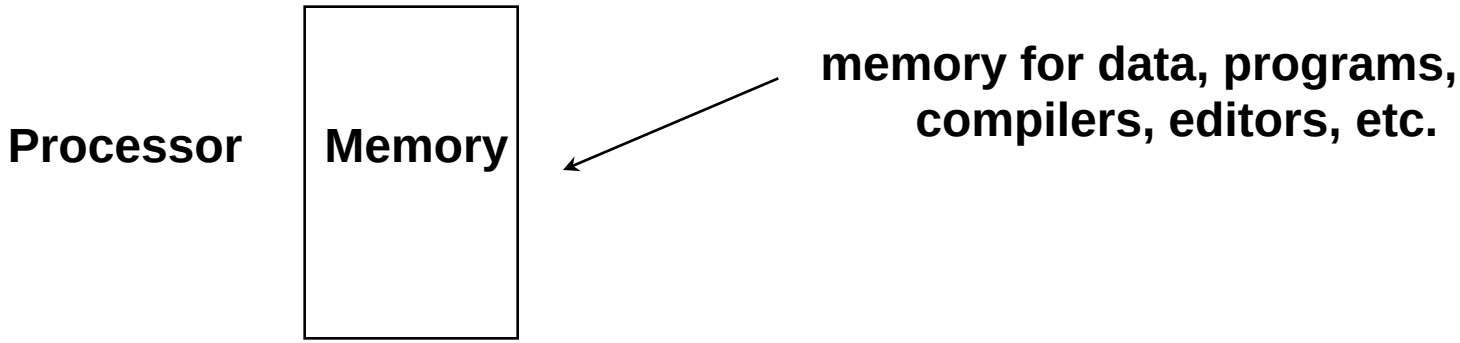
MIPS machine



PC = Program Counter; ALU = Arithmetic & Logic Unit

Stored Program Concept

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



Fetch & Execute Cycle:

- Instructions are fetched and put into a special register
- Bits in the register "control" the subsequent actions
- Fetch the “next” instruction and continue

MIPS Arithmetic

- All *arithmetic instructions* have three operands
- Operand order is fixed (destination first)
- *Arithmetic instructions*' operands **must** be registers
- Only 32 registers provided
- Each register contains 32 bits (1 word)
- 2^{30} memory words are available… but they are accessed *only by data transfer instructions* in MIPS! MIPS uses byte addresses, so sequential word addresses differ by 4

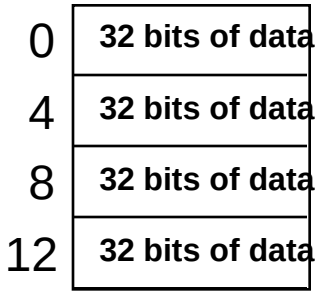
Sample Registers

- $s0, $s1, …, $s7
- $t0, $t1, …, $t7
- $zero

Registers $s0 - $s7 map to 16-23 and $t0 - $t7 map to 8-15. MIPS register $zero always equals 0.

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?

Some Arithmetic Instructions

Instruction	Example	Meaning	Comments
add	add $s1,$s2,$s3	$s1 = $s2 + $s3	3 operands; data in registers
subtract	sub $s2,$s3,$s7	$s2 = $s3 - $s7	3 operands; data in registers

Some Data Transfer Instructions

Instruction	Example	Meaning	Comments
load word	lw $s1,100($s2)	$s1 = Memory[$s2+100]	Data from memory to register
store word	sw $s3,100($s2)	Memory[$s2+100] = $s3	Data from register to memory

Example

C code: `A[12] = h + A[8];`

MIPS code:

```
lw $t0, 32($s3)
add $t0, $s2, $t0
sw $t0, 48($s3)
```

- Can refer to registers by name (e.g., $s2, $t2) instead of number
- Store word has destination last
- Remember: Arithmetic operands are registers, not memory!

Can’t write: `add 48($s3), $s2, 32($s3)`

Logical Instructions

Instruction	Example	Meaning	Comments
and	and $s1,$s2,$s3	$s1 = $s2 & $s3	3 reg. operands; bit-by-bit AND
or	or $s4,$s2,$s6	$s4 = $s2 | $s6	3 reg. operands; bit-by-bit OR
nor	nor $s2,$s5,$s6	$s2 = ~ ($s5 | $s6)	3 reg. operands; bit-by-bit NOR
and immediate	andi $s1,$s2,100	$s1 = $s2 & 100	Bit-by-bit AND reg with constant
or immediate	ori $s2,$s1,100	$s2 = $s1 | 100	Bit-by-bit OR reg with constant
shift left logical	sll $s3,$s4,10	$s3 = $s4 << 10	Shift left by constant
shift right logical	srl $s1,$s7,5	$s1 = $s7 >> 5	Shift right by constant

MIPS Fields

R-type (for “register”):

op	rs	rt	rd	shamt	funct
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits
opcode (“operation code”)	1st register source operand	2nd register source operand	Register destination operand	Shift amount	Function

I-type (for “immediate”):

op	rs	rt	constant or address
6 bits	5 bits	5 bits	16 bits

Some Conditional Branch Instructions

Instruction	Example	Meaning	Comments
branch on equal	beq $s1,$s2,L	if($s1 == $s2) go to L; else fall through	Equal test and branch
branch on not equal	bne $s3,$s2,L	if($s3 != $s2) go to L; else fall through	Not equal test and branch
set on less than	slt $s1,$s2,$s3	if($s2<$s3) $s1=1; else $s1=0	Used with beq, bne
set on less than immediate	slti $s1,$s2,100	if($s2<100) $s1=1; else $s1=0	Used with beq, bne

An Unconditional Jump Instruction

Instruction	Example	Meaning	Comments
jump	j L	go to L	jump to target address

Note: In the tables above, L stands for “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?*

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

Register 1 ($at) reserved for assembler, 26-27 for operating system

SPIM: I/O via System Calls (Section A.9)

```
        .data
str:    .asciiz "You typed the number = "
        .text
main:   li      $v0, 5          # system call code for read_int
        syscall                 # read the integer input
        li      $s1, 0          # load zero to $s1
        add     $s1, $s1, $v0   # copy the input
        li      $v0, 4          # system call for print_str
        la      $a0, str        # load address of string to print
        syscall                 # print the string
        li      $v0, 1          # system call for print_int
        li      $a0, 0          # load zero to $a0
        add     $a0, $a0, $s1   # set $a0 to the integer to print
        syscall                 # print the integer
        jr      $ra
```