EECE 417 Computer Systems Architecture

Department of Electrical and Computer Engineering
Howard University

Charles Kim

Spring 2007

Computer Organization and Design (3rd Ed)

-The Hardware/Software Interface

by

David A. Patterson
John L. Hennessy

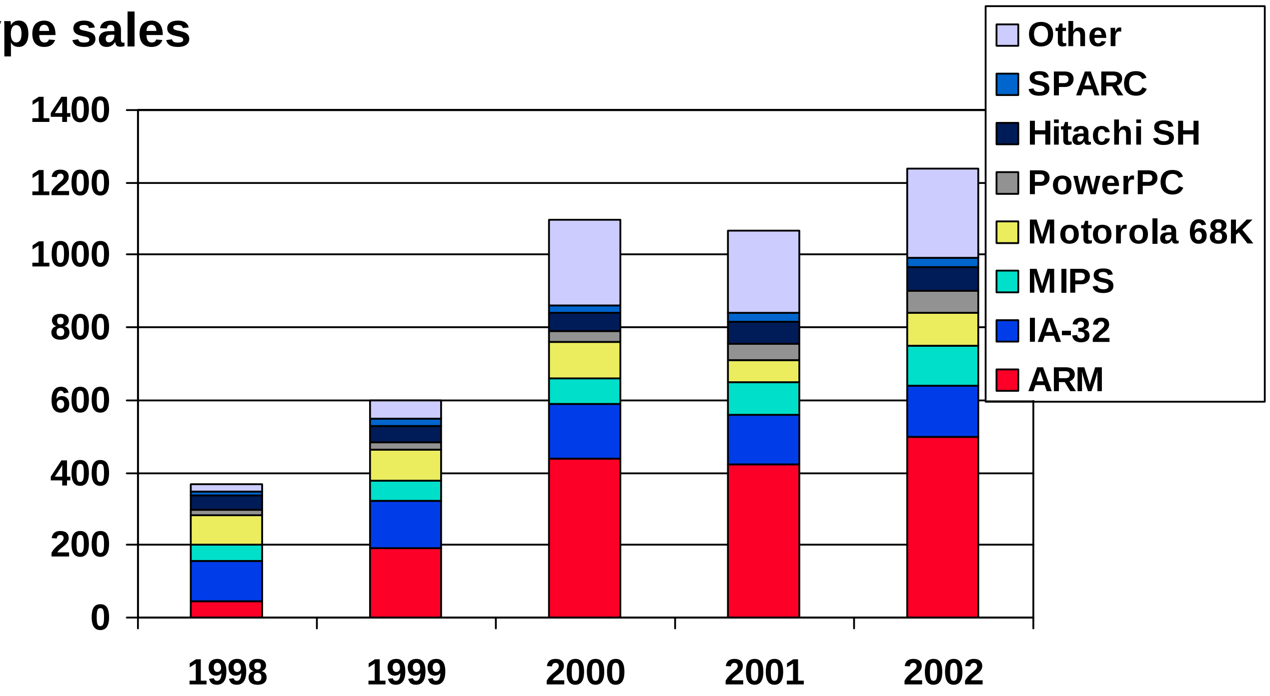
Chapter 2

Instructions: Language of the Computer

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, …**

- **ISA type sales**



MIPS arithmetic

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

Example:

C code:	**a = b + c**	
	dst src1 src2	
MIPS 'code':	**add a, b, c**	

(we'll talk about registers in a bit)

"The natural number of operands for an operation like addition is three…requiring every instruction to have exactly three operands, no more and no less, conforms to the philosophy of keeping the hardware simple"

MIPS arithmetic

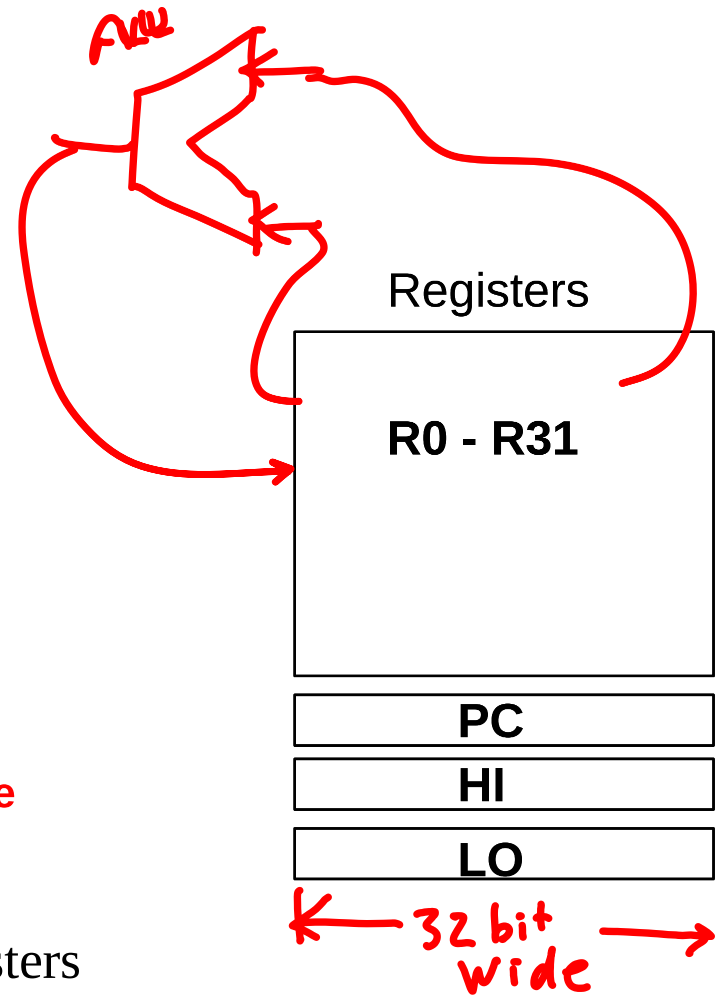
- **Design Principle: simplicity favors regularity.**
- **Of course this complicates some things...**

C code: `a = b + c + d;`

MIPS code:
```
add a, b, c
add a, a, d
```

- **Operands must be registers, only 32 registers provided**
- **Each register contains 32 bits**

- **Design Principle: smaller is faster. Why?**
  - **Large number of registers may increase the clock cycle time**
  - **Then why not 30 registers?**
    - Balance between cries for more registers from programmer vs. fast clock cycle from hardware designer



32 Registers & Policy of Use Conventions

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

Name	Register number	Usage
$zero	0	the constant value 0
$at	1	Assembler temporary (reserved)
$v0-$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
$k0-$k1	26-27	reserved for OS kernel
$gp	28	global pointer
$sp	29	stack pointer
$fp	30	frame pointer
$ra	31	return address

Simple Example with MIPS Assembly Language using *SPIM*

- **F = (g+h) – (i+j)**
- **G=5, h=10, i=20, and j=1**

```
#p53.asm
#f=(g+h)-(i+j)
#f=$s0, g=$s1, h=$s2, i=$s3, j=$s4
        .text                               #program code
main:
        ori $s0, $zero, 0                   #initiate with num
        ori $s1, $zero, 5                   #immediate number
        ori $s2, $zero, 10
        ori $s3, $zero, 20
        ori $s4, $zero, 1                   #$s for variables

        add $t0, $s1, $s2                   #$t for temprary r
        add $t1, $s3, $s4
        sub $t0, $t0, $t1
```

- **See the result of SPIM. Where is the result?**

SPIM's Human Interface (syscall)

- SPIM provides a small set of operating system-like services through the system call (syscall) instruction
- To request a service, a program loads the system call code into register $v0 and arguments into registers $a0~$a3
- System calls that return values put their results in register $v0

Service	System Call Code	Arguments	Result
Print_int	1	$a0 = integer	
Print_float	2	$f12 = float	
Print_double	3	$f12 = double	
Print_string	4	$a0 = string	
Read_int	5		Integer (in $v0)
Read_float	6		Float (in $f0)
Read_double	7		Double (in $f0)
Read_string	8	$a0 = buffer, $a1 = length	
Sbrk	9	$a0 = amount	Address (in $v0)
exit	10		

Simple Example with *Syscall*

p53A.asm - Notepad

```
#p53A.asm
#Print Out Syscall
#f=(g+h)-(i+j)
#f=$s0, g=$s1, h=$s2, i=$s3, j=$s4
        .text                   #program code
main:
        ori $s0, $zero, 0       #initiate with number
        ori $s1, $zero, 5       #immediate number loading
        ori $s2, $zero, 10
        ori $s3, $zero, 20
        ori $s4, $zero, 1       #$s for variables

        add $t0, $s1, $s2       #$t for temprary results
        add $t1, $s3, $s4
        sub $t0, $t0, $t1
#syscall
#calling system call using $v0
# print intger   $v0=1  with argument in $a0
# read integer   $v0=5  with saved in $v0
        ori $v0, $zero,1        #request for print
        or  $a0,  $zero, $t0    #what is in t0
        syscall
```

- **Result**

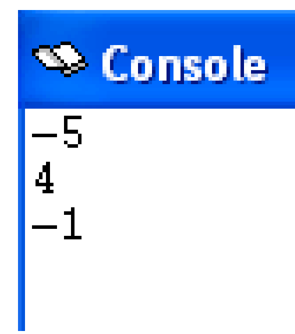
Console

```
-6
```

Let’s read values from keyboard

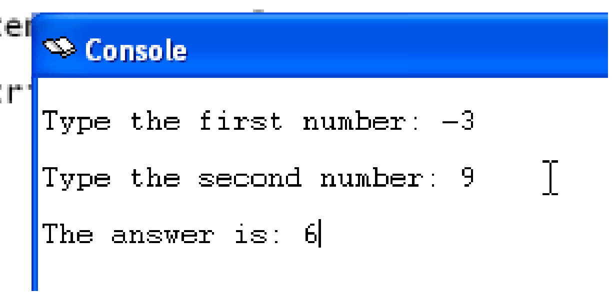
```
#p53B.asm
#Read 4 values from Keyboard
#And Print Out the result using SYSCALL
#a=b+c
#a=$s0, b=$s1, c=$s2
        .text                           #program code
main:
        ori $s0, $zero, 0               #initiate with number 0
# Read Input (b)
        ori $v0, $zero, 5
        syscall                         #now type-in is in v0
        or  $s1, $zero,$v0

# Read Input (c)
        ori $v0, $zero, 5
        syscall                         #now type-in is in v0
        or  $s2, $zero,$v0

#Add (b) and (c)
        add $s0, $s1, $s2               #$t for temprary results
#syscall
#calling system call using $v0
# print intger    $v0=1  with argument in $a0
# read integer    $v0=5  with saved in $v0
        ori $v0, $zero,1                #request for print
        or  $a0,  $zero, $s0            #what is in t0
        syscall
```

- **Result**
- **a=b+c**



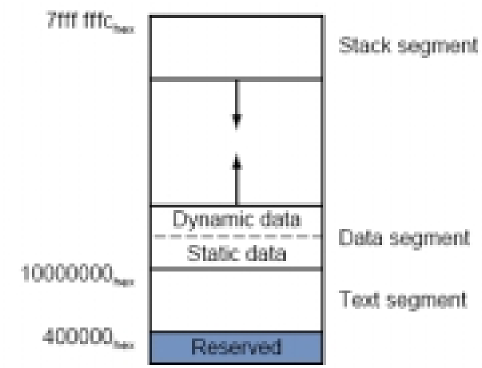
Let’s add bells and whistles

```
#p53C.asm
#Read 4 values from Keyboard
#with strings
#And Print Out the result using SYSCALL
#a=b+c
#a=$s0, b=$s1, c=$s2
main:
        .data                       #data part
msg1:   .asciiz "\nType the first number: "  #string to print
msg2:   .asciiz "\nType the second number: "
msg3:   .asciiz "\nThe answer is: "
        .text                               #code part
        ori $v0, $zero, 4           #msg1
        la $a0,  msg1               # takes the address of string as an argument
        syscall
        ori $v0, $zero, 5           #read input (b)
        syscall                     #now type-in is in v0
        or  $s1, $zero,$v0
        ori $v0, $zero, 4           #msg2
        la $a0,  msg2               # takes the address of string as an argument
        syscall
        ori $v0, $zero, 5           #read input (c)
        syscall                     #now type-in is in v0
        or  $s2, $zero,$v0
        add $s0, $s1, $s2           #add (b) and (c) $t for te
        ori $v0, $zero, 4           #msg3
        la $a0,  msg3               # takes the address of str
        syscall
        ori $v0, $zero,1            #request for print
        or  $a0,  $zero, $s0        #result
        syscall
```



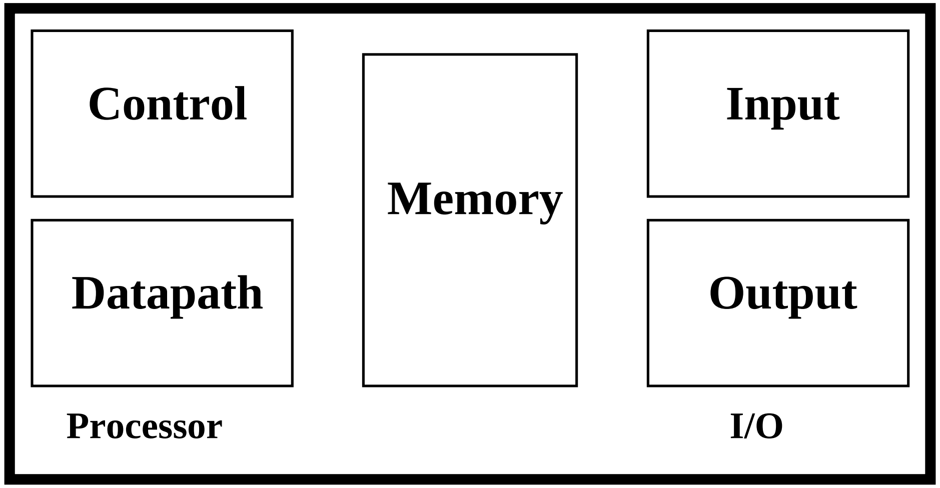
What if we want to use only core instructions

```
#p53D.asm
# Without using pseudo instructions
#such as la "load address", li "load immediate"  etc
main:
        .data   0x10010000              #starting address of first string
        .asciiz "\nType the first number: "  #msg1
        .data   0x10010100              #starting addres of next
        .asciiz "\nType the second number: " #msg2
        .data   0x10011000              #starting address of the third
        .asciiz "\nThe answer is: "          #msg3
        .text                           #code part
        ori $v0, $zero, 4       #msg1
        lui $a0, 0x1001         # Upper part of msg1 addr (ie a0=10010000)
        ori $a0, $a0, 0         # now a0 has 1 word addr 10010000
        syscall
        ori $v0, $zero, 5       #read input (b)
        syscall                 #now type-in is in v0
        or  $s1, $zero,$v0
        ori $v0, $zero, 4       #msg2
        lui $a0,  0x1001        #a0=10010000
        ori $a0, $a0, 0x0100    #a0=10010100
        syscall
        ori $v0, $zero, 5       #read input (c)
        syscall                 #now type-in is in v0
        or  $s2, $zero,$v0
        add $s0, $s1, $s2       #add (b) and (c) $t for tempr
        ori $v0, $zero, 4       #msg3
        lui $a0, 0x1001         #a0=10010000
        ori $a0, $a0, 0x1000    #$a0=10011000
        syscall
        ori $v0, $zero,1        #request for print
        or  $a0,  $zero, $s0    #result
        syscall
```

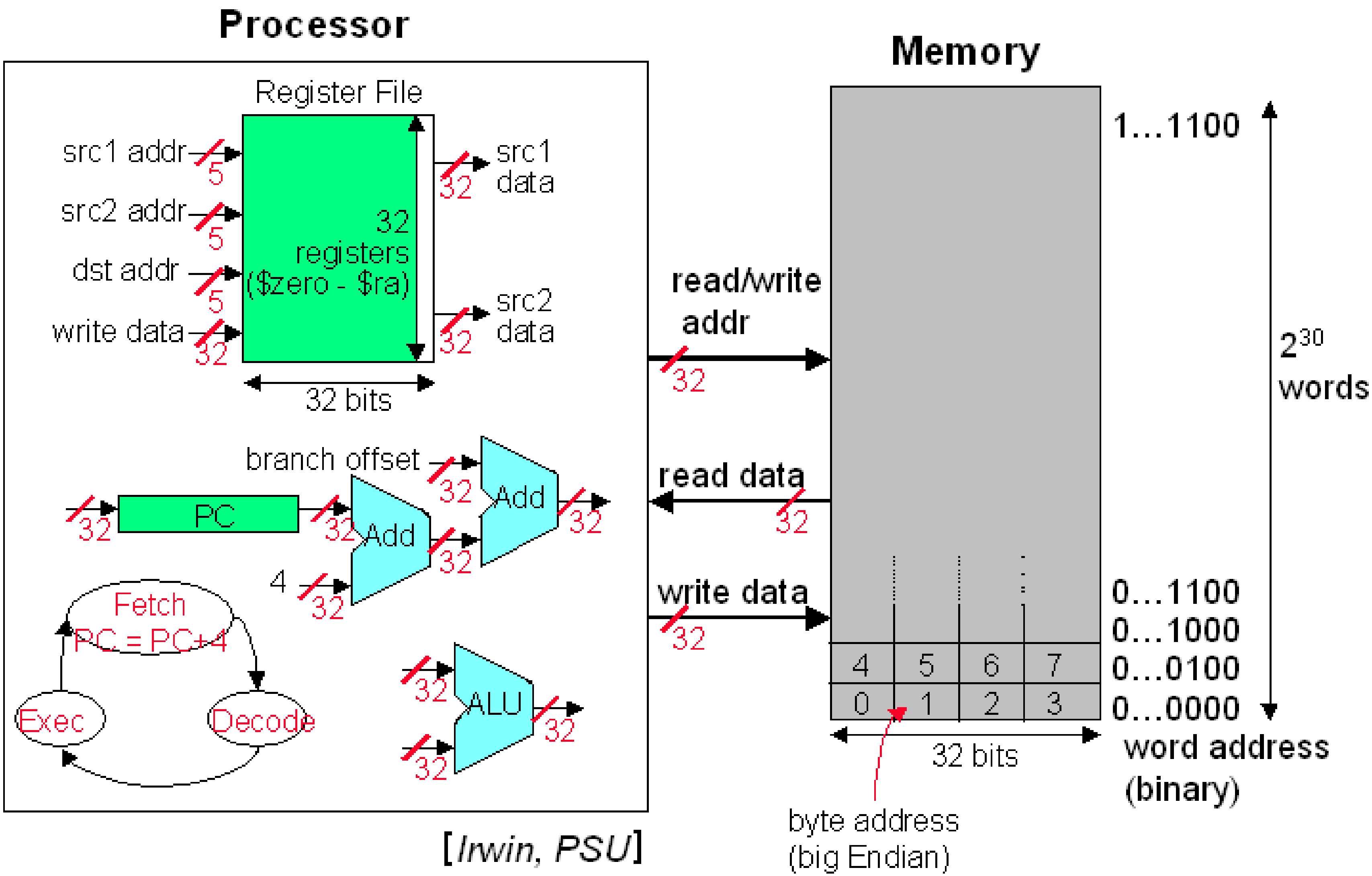


Registers vs. Memory

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



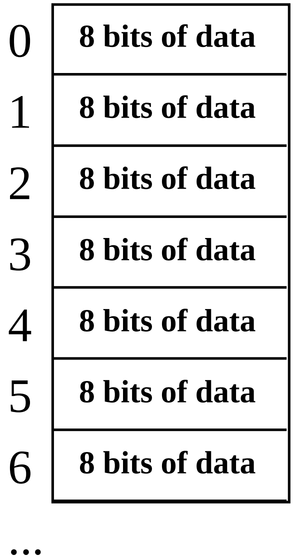
MIPS Register & Memory

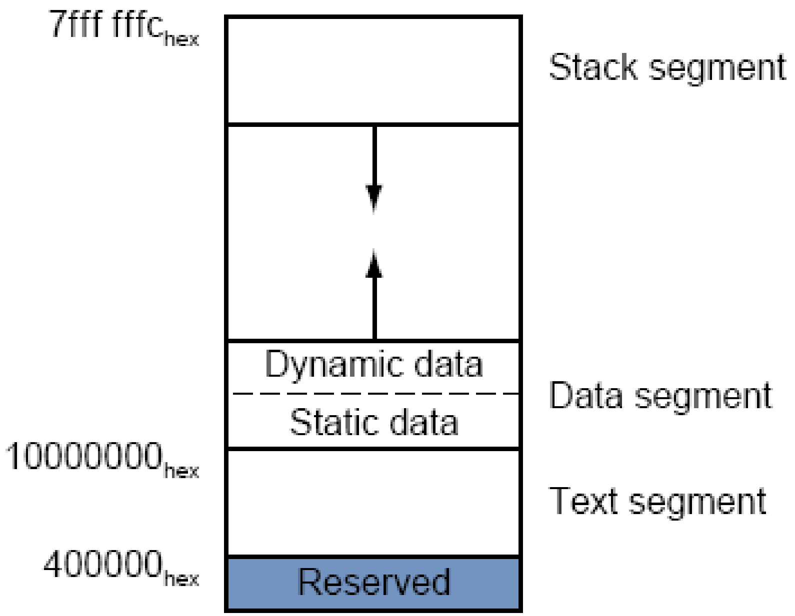


[*Irwin, PSU*]

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.

0	32 bits of data
4	32 bits of data
8	32 bits of data
12	32 bits of data
...	

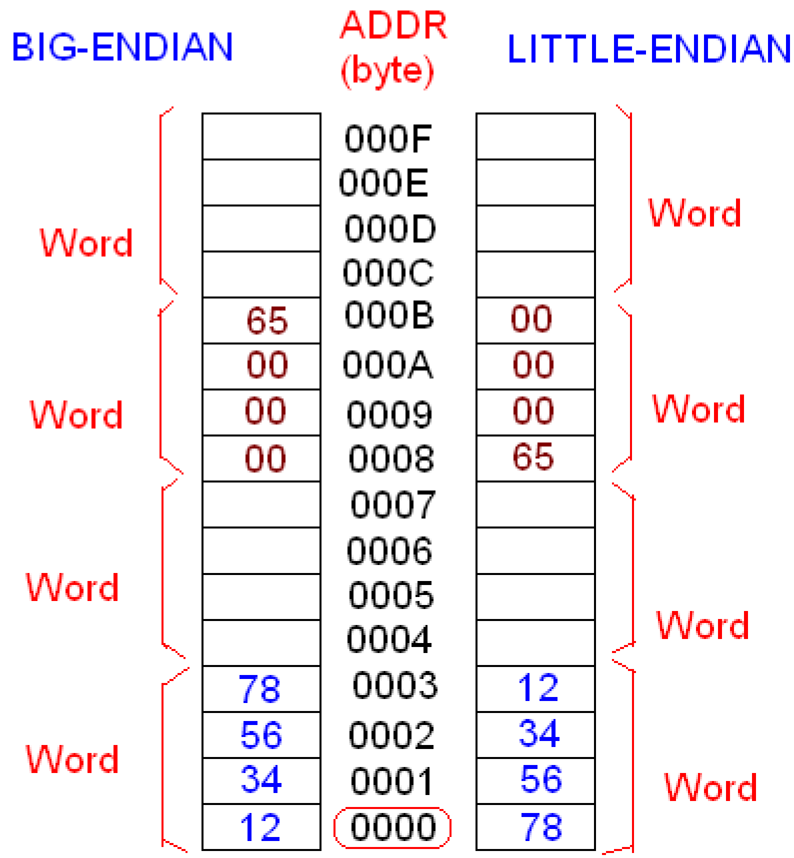
Registers hold 32 bits of data

- 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

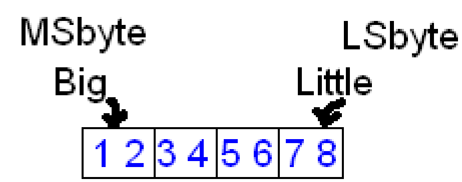
  i.e., what are the least 2 significant bits of a word address?

WORD
0 0000 0000
4 0000 0100
8 0000 1000
12 0000 1100
16 0001 0000

Byte Order and Two Camps



"Store 0x00000065 to adress 0x0008"



"Store 0x12345678 to address 0x0000"

- Big-Endian: Fill the MSbyte first at 0x0000
- Little-Endian: Fill the LSbyte first at 0x0000

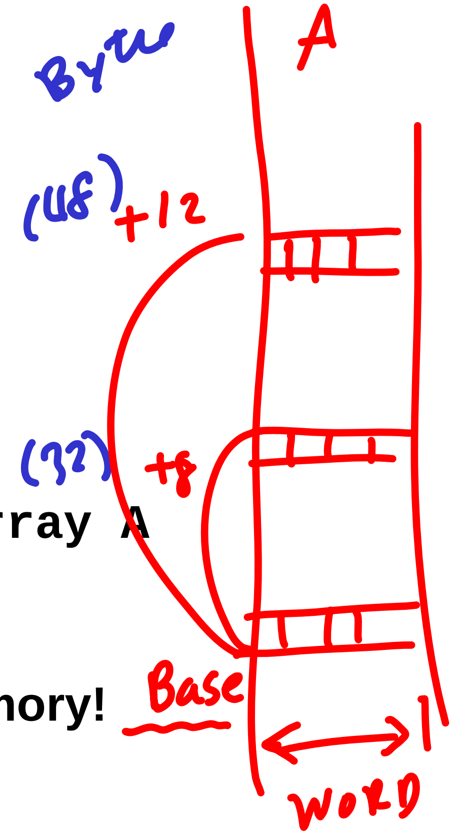
Instructions for Memory Access

- **Load** and **store** instructions
- **Example:**

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

MIPS code:

```
lw $t0, 32($s3)
add $t0, $s2, $t0
sw $t0, 48($s3)
#where $s3 holds the base address of array A
```



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

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

Load and Save Basics in *SPIM* (1/2)

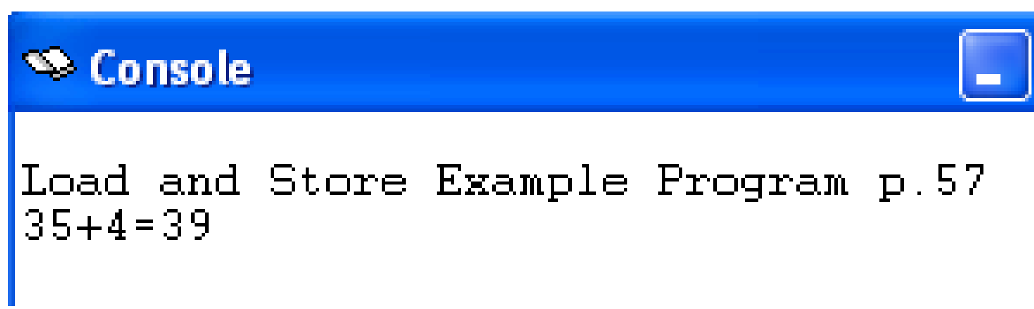
p57.asm - Notepad

File Edit Format View Help

```
#p57.asm
#load and store
#the base address of array A is in $s3
#A[12]=A[1]+A[8]
main:
        .data   0x10010000                          #starting address of the Array A
        .word   0x00000003, 0x00000004, 0x00000005, 0x00000006
        .word   0x00000013, 0x00000014, 0x00000015, 0x00000016
        .word   0x00000023, 0x00000024, 0x00000025, 0x00000026
        .data   0x10011000                          #starting addres of next
        .asciiz "\nLoad and Store Example Program p.57\n"
        .data   0x10011100
        .asciiz "+"
        .asciiz "="
        .text                                       #code part
        ori $v0, $zero, 4           #msg1
        lui $a0, 0x1001             # Display the banner
        ori $a0, $a0, 0x1000
        syscall
        lui $s3, 0x1001
        ori $s3, $s3, 0
        lw  $t0, 32($s3)            #t0=A[8]
#print A[8]
        ori $v0, $zero,1            #request for print
        or  $a0, $zero, $t0
        syscall
#pring + sign
        ori $v0, $zero,4            #request for print
        lui $a0, 0x1001
        ori $a0, $a0, 0x1100
        syscall
```

Load and Save Basics in *SPIM* (2/2)

```
          lw   $t1, 4($s3)           #t1=A[1]
#print A[1]
          ori  $v0, $zero,1          #request for print
          or   $a0, $zero, $t1
          syscall
#Operation of ADD
          addu $t2, $t1, $t0
          sw   $t2, 48($s3)
#Print = sign
          ori  $v0, $zero,4          #request for print
          lui  $a0, 0x1001
          ori  $a0, $a0, 0x1102
          syscall
#print the A[12]
          ori  $v0, $zero,1          #request for print
          lw   $t3, 48($s3)          #result
          or   $a0, $zero, $t3
          syscall
```



Check if we know this:

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

- **Instruction** / **Meaning**

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`