MIPS

MIPS is a “computer family”

- R2000/R3000 (32-bit)
- R4000/4400 (64-bit)
- R8000 (for scientific & graphics applications)
- R10000 (64-bit)

MIPS originated as a Stanford research project
Microprocessor without **I**nterlocked **P**ipe **S**tages

MIPS was bought by Silicon Graphics (SGI) & is now independent

MIPS is a RISC

MIPS Registers

Part of the state of a process

Thirty-two 32-bit **general purpose registers** (**GPR**s): \$0, \$1, ..., \$31

- integer arithmetic
- address calculations
- temporary values

By convention software uses different registers for different purposes *(next slide)*

A 32-bit **program counter** (**PC**)

Two 32-bit registers **HI** and **LO** used specifically for multiply and divide

- HI & LO concatenated for the product
- LO for the quotient; HI for the remainder

Thirty-two 32-bit registers: \$**f**0, \$**f**1, ..., \$**f**31 used for floating-point arithmetic

- often used as 16 64-bit registers for double precision FP

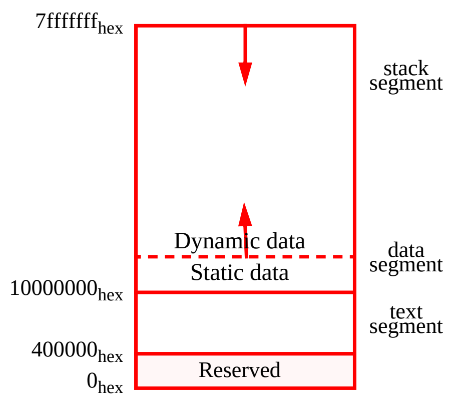
Other special-purpose registers *(later)*

MIPS Register Names and gcc Conventions

Register	Name	Use	Comment
$0	zero	always 0	cannot be written
$1	$at	reserved for assembler	don't use it!
$2, $3	$v0, $v1	function return	
$4 - $7	$a0 - $a3	pass first 4 procedure/ function arguments	
$8 - $15	$t0 - $t7	temporaries	not saved across a call (caller saved)
$16 - $23	$s0 - $s7	temporaries	saved across a call (callee saved)
$24, $25	$t8, $t9	temporaries	not saved across a call (caller saved)
$26, $27	$k0, $k1	reserved for the OS	don't use them!
$28	$gp	pointer to global static memory	points to the middle of a 64KB block in static data *(next slide)*
$29	$sp	stack pointer	points to the last allocated stack location *(next slide)*
$30	$fp	frame pointer	points to the activation record *(later)*
$31	$ra	procedure/function return address	

Memory Usage

A software convention



text segment: the code

data segment

- **static data**: objects whose size is known to the compiler & whose lifetime is the whole program execution
- **dynamic data**: objects allocated as the program executes (`malloc`)

stack segment: FIFO process-local storage

MIPS Load-Store Architecture

Most instructions compute on operands stored in registers

- load data into a register from memory
- compute in registers
- the result is stored into memory

For example,

```
a = b + c
d = a + b
```

is “compiled” into:

load b into register $x

load c into register $y

$z ⇐ $x + $y

store $z into a

$z ⇐ $z + $x

store $z into d

MIPS Information Units

Data types and sizes

- byte
- half-word (2 bytes)
- word (4 bytes)
- float (4 bytes using single-precision floating-point format)
- double (8 bytes using double-precision floating-point format)

Memory is byte-addressable

A data type must start on an address evenly divisible by its size in bytes

Big & Little Endian Byte Order

Every word starts at an address that is divisible by 4.
Which byte in the word is byte 0?
How is the data in **`.byte 0,1,2,3`** stored?

Big Endian

31 24	23 16	15 8	7 0	byte address
0	1	2	3	0
4	5	6	7	4
8	9	10	11	8
0’s	0’s	0’s	110_2	12

Most significant byte is the lowest byte address.
Word is addressed by the byte address of the **most** significant byte.

Little Endian

31 24	23 16	15 8	7 0	
3	2	1	**0**	0
7	6	5	**4**	4
11	10	9	**8**	8
0’s	0’s	0’s	110_2	12

Least significant byte is the lowest byte address.
Word is addressed by the byte address of the **least** significant byte.

Big & Little Endian Byte Order

Problems when transferring between big & little endian computers data structures that contain a mixture of integers & characters

Big Endian

byte address				
0	J	I	M	
4	S	M	I	T
8	H	0	0	0
12	0’s	0’s	0’s	21

Little Endian

				byte address
	M	I	J	0
T	I	M	S	4
0	0	0	H	8
0’s	0’s	0’s	21	12

Transfer from big endian computer to little endian computer

Little Endian

				byte address
	M	I	J	0
T	I	M	S	4
0	0	0	H	8
21	0’s	0’s	0’s	12

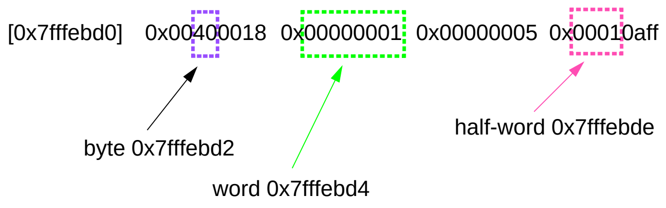
MIPS Information Units

MIPS support both **big-** and **little-endian** byte orders

SPIM uses the byte order of the machine its running on

- Intel: little-endian
- Alpha, SPARC, Mac: big-endian

Words in SPIM are listed from left to right, but byte addresses are little-endian within a word



```
[0x7fffebd0]   0x00400018  0x00000001  0x00000005  0x00010aff
```