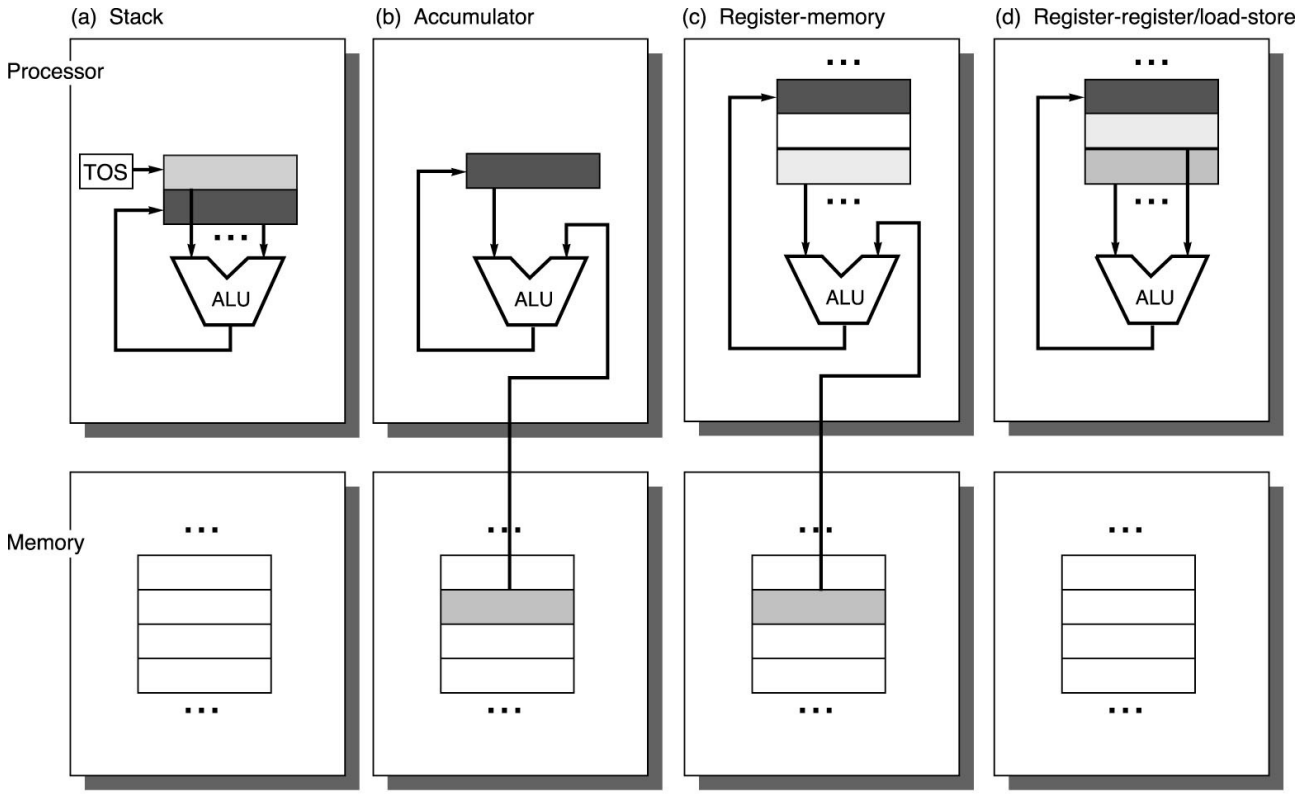
Organización de Computadoras

Principios de Conjuntos de Intrucciones

Clasifición de las ISA



Secuencia de Código para C = A + B

Stack	**Acumulador**	**Registro (Reg-Mem)**	**Registro (load-store)**
Push A	Load A	Load R1, A	Load R1,A
Push B	Add B	Add R3,R1,B	Load R2,B
Add	Store C	Store R3, C	Add R3,R1,R2
Pop C			Store R3,C

Combinaciones de operandos en memoria y operandos totales en instrucciones de ALU

Nro. de direcciones de memoria	Máximo nro. de operandos permitidos	Tipo de arquitectura	Ejemplos
0	3	Registro-registro	Alpha, ARM, MIPS, PowerPC, SPARC.
1	2	Registro-memoria	IBM 360/370, Intel 80x86, Motorola 68000
2	2	Memoria-Memoria	VAX
3	3	Memoria-Memoria	VAX

Ventajas y desventajas de los 3 tipos más comunes de computadoras del tipo registro de propósito general

Tipo	Ventajas	Desventajas
R-R (0,3)	Simple, codificación con instrucciones de tamaño fijo, modelo de generación de código simple, instrucciones toman similares ciclos de reloj para su ejecución	Conteo de instrucciones más alto, programas más grandes
R-M (1,2)	Se pueden acceder datos sin un load separado primero. Tendencia a ser facilmente codificado,	Los operandos no son equivalentes (uno es destuido). Pocos bits disponibles para codificar el registro
M-M (2,2) o (3,3)	Código compacto. No desperdicia registros como temporarios	Gran variación en el tamaño de instrucción. CPI muy variable. Cuello de botella en la memoria

Direccionamiento de Memoria

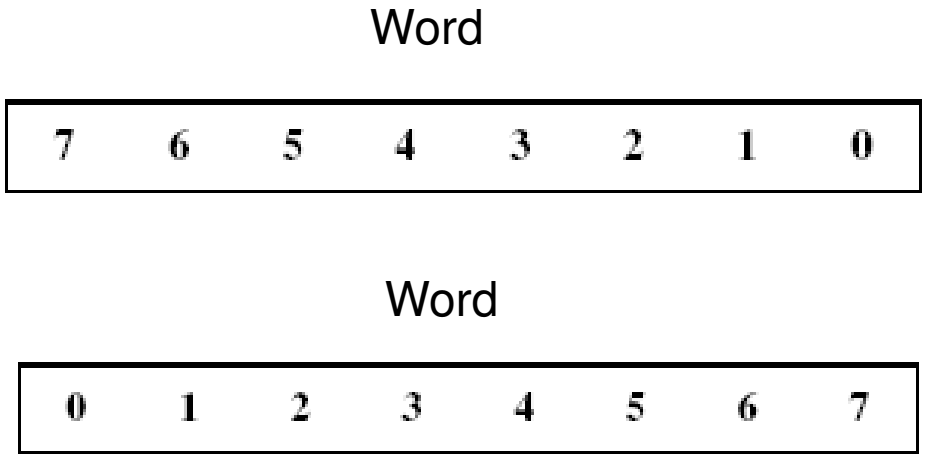
- Direccionamiento por ***byte***, acceso a:
  - Byte
  - Half word
  - Word
  - Double word
- Tópicos
  - Alineación
  - Orden
    - Litttle endian
    - Big endian

Alineación de objetos en memoria

Width of object:	Value of 3 low order bits of byte address: 0	1	2	3	4	5	6	7
1 Byte (Byte)	Aligned	Aligned	Aligned	Aligned	Aligned	Aligned	Aligned	Aligned
2 Bytes (Half word)	Aligned		Aligned		Aligned		Aligned	
2 Bytes (Half word)		Misaligned		Misaligned		Misaligned		Misalig.
4 Bytes (Word)	Aligned				Aligned			
4 Bytes (Word)		Misaligned				Misaligned		
4 Bytes (Word)			Misaligned				Misaligned	
4 Bytes (Word)				Misaligned				Misalig.
8 bytes (Double word)	Aligned							
8 bytes (Double word)		Misaligned						
8 bytes (Double word)			Misaligned					
8 bytes (Double word)				Misaligned				
8 bytes (Double word)					Misaligned			
8 bytes (Double word)						Misaligned		
8 bytes (Double word)							Misaligned	
8 bytes (Double word)								Misalig.

Direccionamiento de Memoria

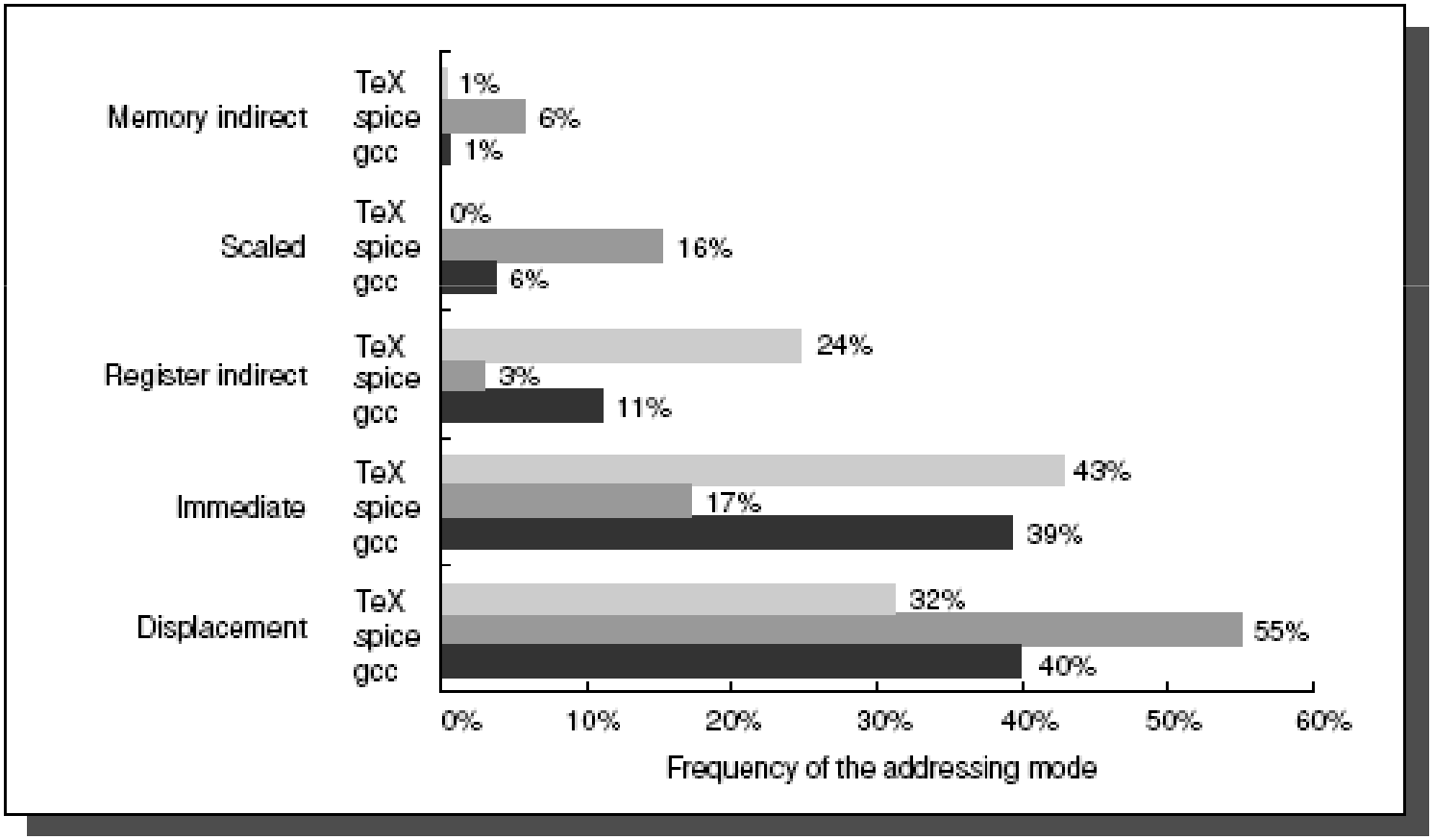
- Interpretación de las direcciones de memoria.
  - "*Little Endian*": pone el byte cuya dirección es XX…000 en la parte más baja de la palabra.
  - "*Big Endian*": pone el byte cuya dirección es XX…000 en la parte más alta de la palabra.



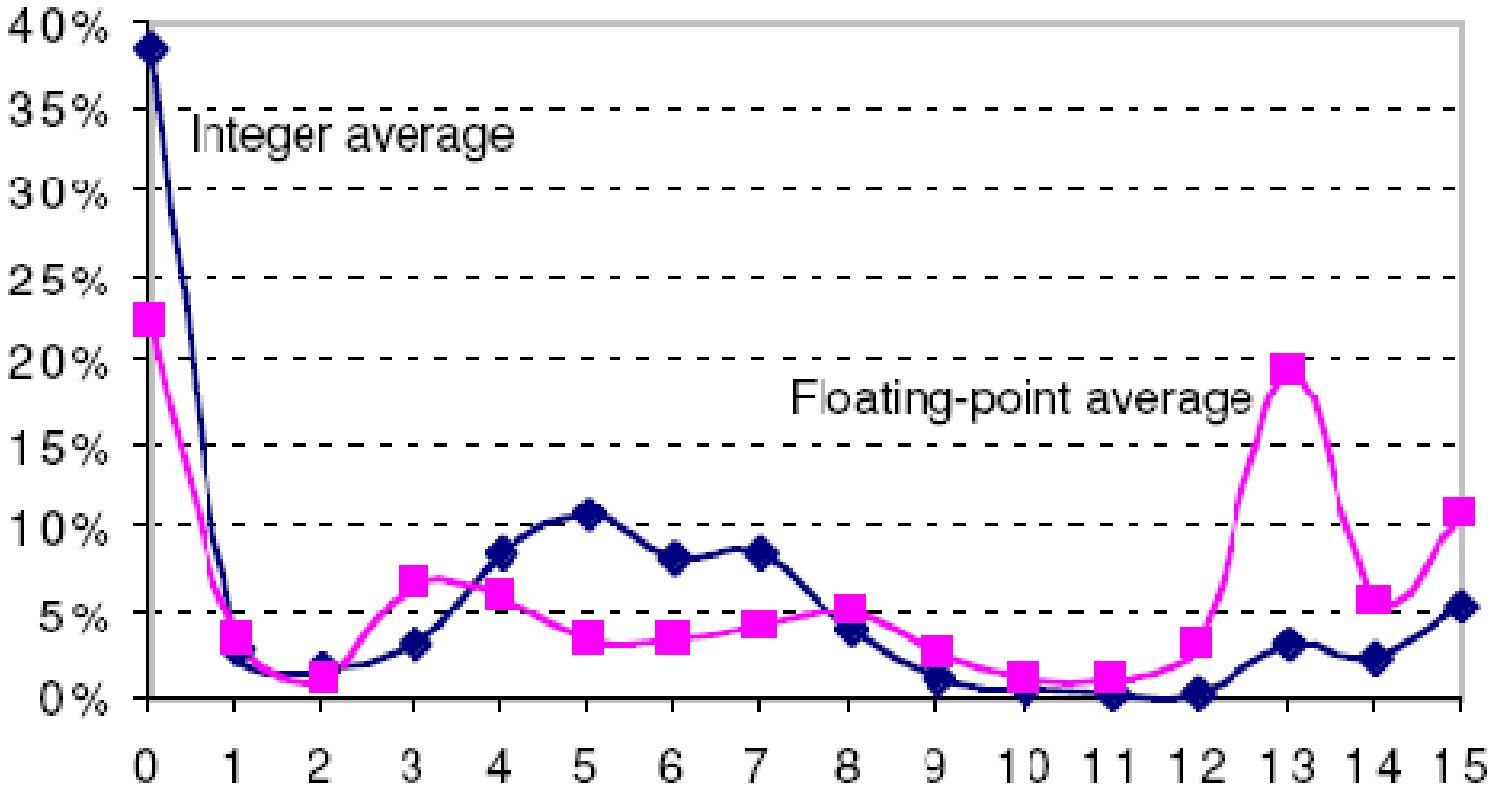
Modos de Direccionamiento

Addressing mode	Example instruction	Meaning	When used
Register	`Add R4,R3`	`Regs[R4]←Regs[R4] + Regs[R3]`	When a value is in a register.
Immediate	`Add R4,#3`	`Regs[R4]←Regs[R4]+3`	For constants.
Displacement	`Add R4,100(R1)`	`Regs[R4]←Regs[R4] + Mem[100+Regs[R1]]`	Accessing local variables (+ simulates register indirect, direct addressing modes)
Register indirect	`Add R4,(R1)`	`Regs[R4]←Regs[R4] + Mem[Regs[R1]]`	Accessing using a pointer or a computed address.
Indexed	`Add R3,(R1 + R2)`	`Regs[R3]←Regs[R3] +Mem[Regs[R1] +Regs[R2]]`	Sometimes useful in array addressing: `R1` = base of array; `R2` = index amount.
Direct or absolute	`Add R1,(1001)`	`Regs[R1]←Regs[R1] + Mem[1001]`	Sometimes useful for accessing static data; address constant may need to be large.
Memory indirect	`Add R1,@(R3)`	`Regs[R1]←Regs[R1] + Mem[Mem[Regs[R3]]]`	If `R3` is the address of a pointer p, then mode yields $*p$.
Autoincrement	`Add R1,(R2)+`	`Regs[R1]←Regs[R1] + Mem[Regs[R2]]` `Regs[R2]←Regs[R2]+d`	Useful for stepping through arrays within a loop. `R2` points to start of array; each reference increments `R2` by size of an element, d.
Autodecrement	`Add R1,-(R2)`	`Regs[R2]←Regs[R2]-d` `Regs[R1]←Regs[R1] + Mem[Regs[R2]]`	Same use as autoincrement. Autodecrement/increment can also act as push/pop to implement a stack.
Scaled	`Add R1,100(R2)[R3]`	`Regs[R1]← Regs[R1]+ Mem[100+Regs[R2] + Regs[R3]*d]`	Used to index arrays. May be applied to any indexed addressing mode in some computers.

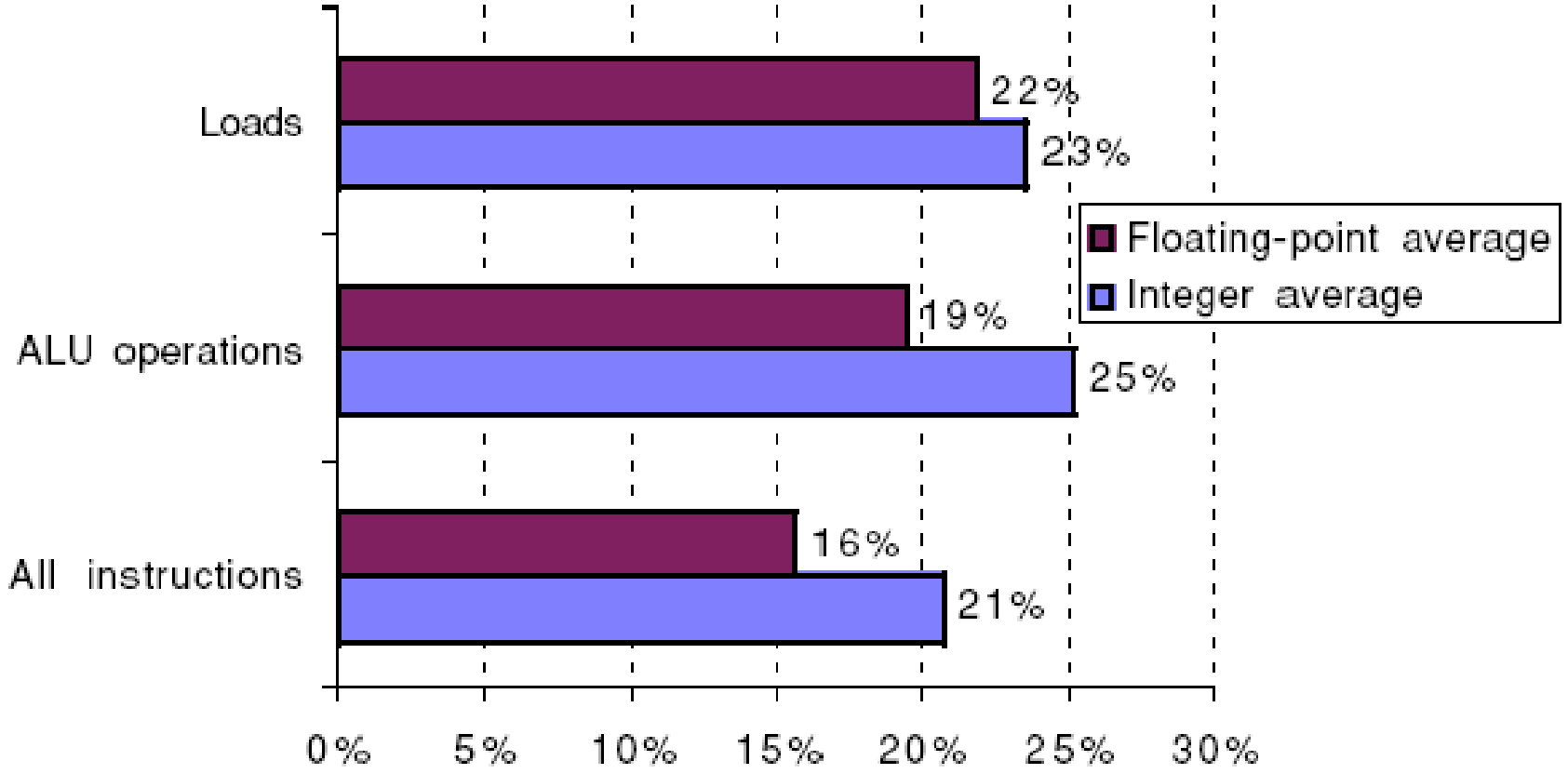
Frecuencia de modos de direccionamiento (VAX, SPEC89)



Número de bits de desplazamiento (Alpha, SPEC CPU2000)



Uso de inmediatos (Alpha, SPEC CPU2000)



Tipo y tamaño de operandos

- Generalmente lo implica el OpCode
  - Carácter
    - ASCII
    - Unicode
  - Enteros
    - Half word
    - Word
  - Coma flotante, precisión simple y doble
    - IEEE 754
  - Decimal
    - BCD

Operaciones en el conjunto de instrucciones

Operator type	Examples
Arithmetic and logical	Integer arithmetic and logical operations: add, subtract, and, or, multiple, divide
Data transfer	Loads-stores (move instructions on computers with memory addressing)
Control	Branch, jump, procedure call and return, traps
System	Operating system call, virtual memory management instructions
Floating point	Floating-point operations: add, multiply, divide, compare
Decimal	Decimal add, decimal multiply, decimal-to-character conversions
String	String move, string compare, string search
Graphics	Pixel and vertex operations, compression/decompression operations

Top 10 instrucciones en 80x86

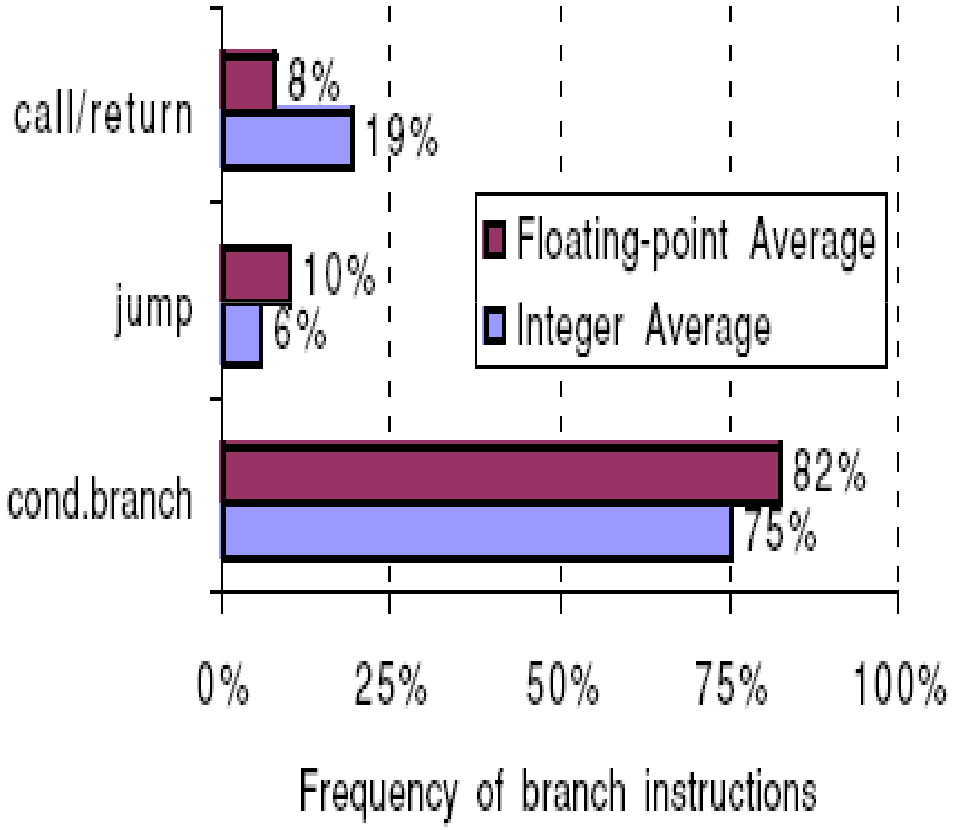
Rank	80x86 instruction	Integer average (% total executed)
1	load	22%
2	conditional branch	20%
3	compare	16%
4	store	12%
5	add	8%
6	and	6%
7	sub	5%
8	move register register	4%
9	call	1%
10	return	1%
Total		96%

Soporte multimedia para desktop RISC

Instruction category	Alpha MAX	HP PA-RISC MAX2	Intel Pentium MMX	Power PC AltiVec	SPARC VIS
Add/subtract		4H	8B,4H,2W	16B, 8H, 4W	4H,2W
Saturating add/sub		4H	8B,4H	16B, 8H, 4W	
Multiply			4H	16B, 8H	
Compare	8B (>=)		8B,4H,2W (=,>)	16B, 8H, 4W (=,>,>=,<,<=)	4H,2W (=,not=,>,<=)
Shift right/left		4H	4H,2W	16B, 8H, 4W	
Shift right arithmetic		4H		16B, 8H, 4W	
Multiply and add				8H	
Shift and add (saturating)		4H			
And/or/xor	8B,4H,2W	8B,4H,2W	8B,4H,2W	16B, 8H, 4W	8B,4H,2W
Absolute difference	8B			16B, 8H, 4W	8B
Maximum/minimum	8B, 4W			16B, 8H, 4W	
Pack (2n bits --> n bits)	2W->2B, 4H->4B	2*4H->8B	4H->4B, 2W->2H	4W->4B, 8H->8B	2W->2H, 2W->2B, 4H->4B
Unpack/merge	2B->2W, 4B->4H		2B->2W, 4B->4H	4B->4W, 8B->8H	4B->4H, 2*4B->8B
Permute/shuffle		4H		16B, 8H, 4W	

Instrucciones de control del flujo del programa

- Salto condicional (branch)
- Salto incondicional (jump)
- Llamado a procedimiento (call)
- Retorno de procedimiento (return)



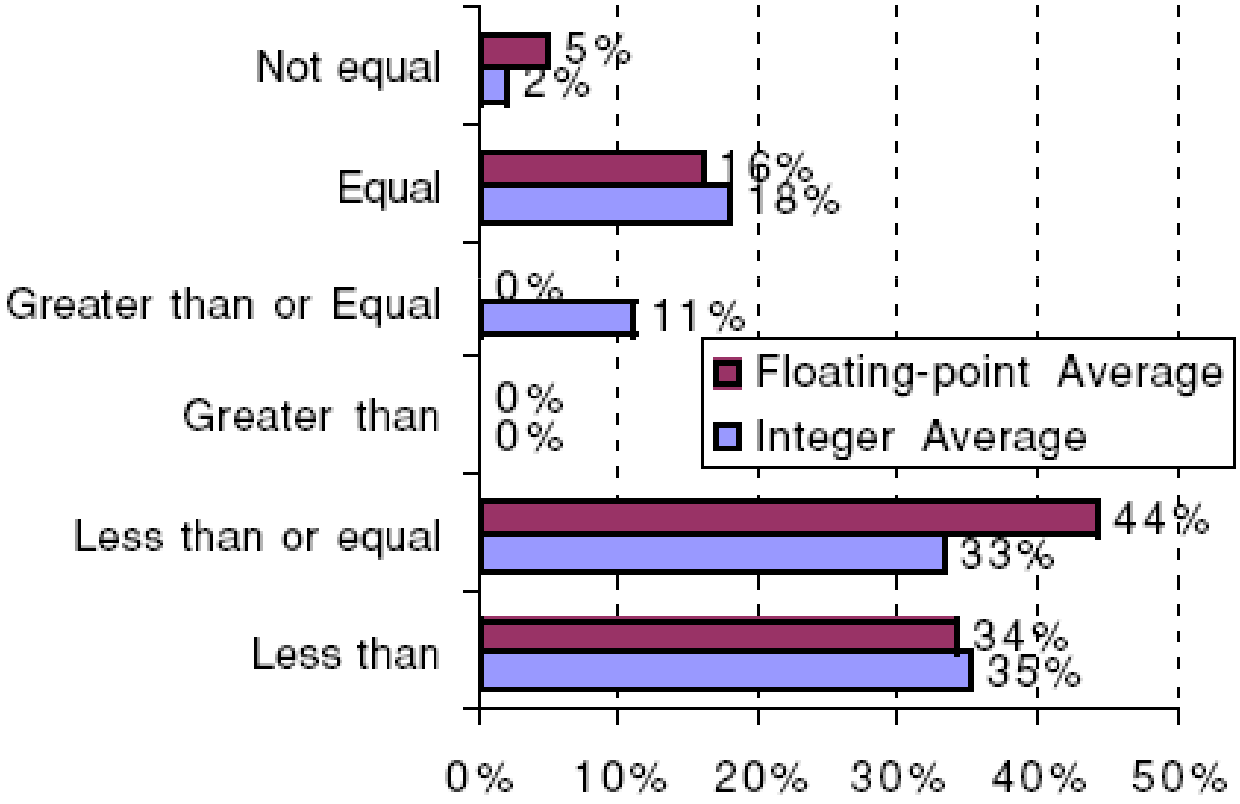
Modos de direccionamiento para instrucciones de control

- Relativas al PC
- Registro indirecto (dirección del salto no conocida en tiempo de compilación)

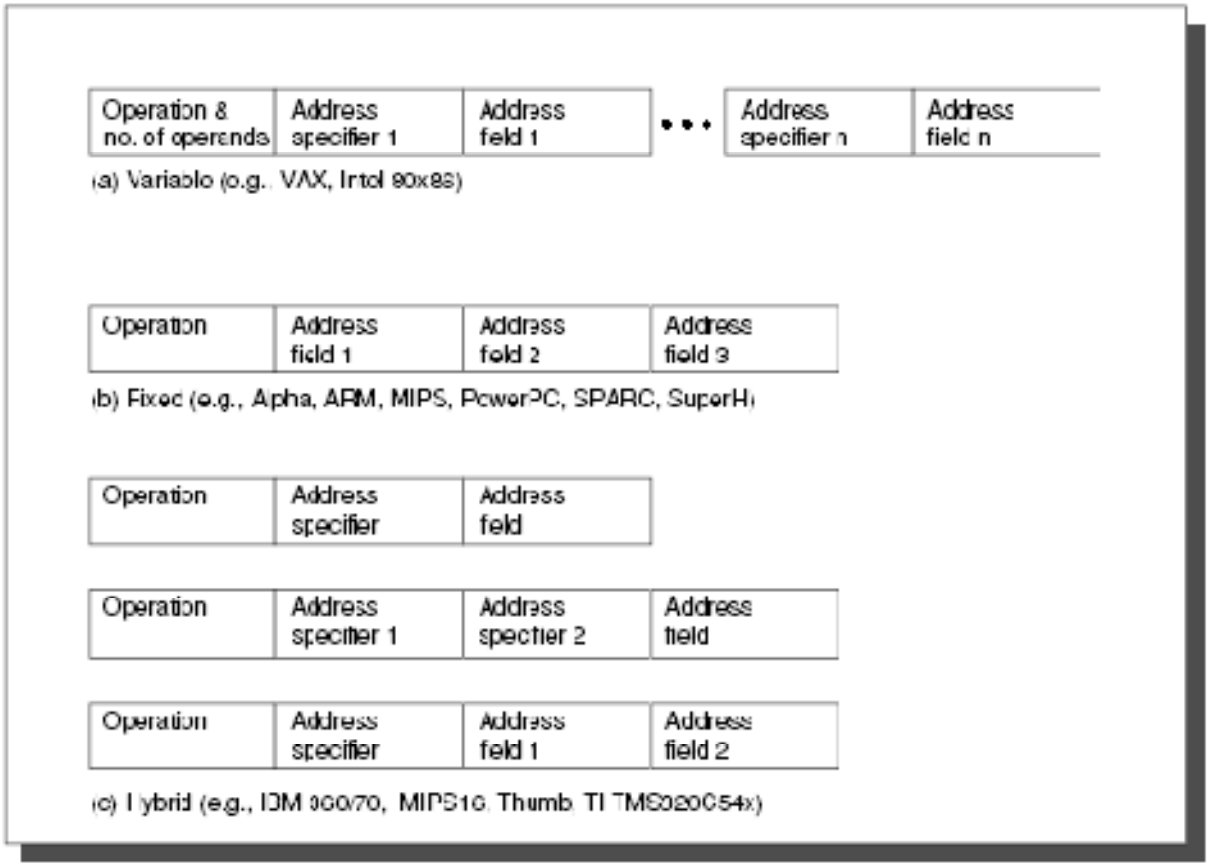
Evaluación de las condiciones de los Branchs

Name	Examples	How condition is tested	Advantages	Disadvantages
Condition code (CC)	80x86, ARM, PowerPC, SPARC, SuperH	Special bits are set by ALU operations, possibly under program control.	Sometimes condition is set for free.	CC is extra state. Condition codes constrain the ordering of instructions since they pass information from one instruction to a branch.
Condition register	Alpha, MIPS	Tests arbitrary register with the result of a comparison.	Simple.	Uses up a register.
Compare and branch	PA-RISC, VAX	Compare is part of the branch. Often compare is limited to subset.	One instruction rather than two for a branch.	May be too much work per instruction for pipelined execution.

Frecuencia de los diferentes tipos de comparaciones



Codificando un conjunto de instrucciones



Programando a nivel de la máquina

Programa en MIPS 32 que calcula la suma de los cuadrados de los numeros de 0 a 100.

```
00100111101111011111111111100000
10101111101111110000000000010100
10101111101001000000000000100000
10101111101001010000000000100100
10101111101000000000000000011000
10101111101000000000000000011100
10001111101011100000000000011100
10001111101110000000000000011000
00000001110011100000000000011001
00100101110010000000000000000001
00101001000000010000000001100101
10101111101010000000000000011100
00000000000000000111100000010010
00000011000011111100100000100001
00010100001000001111111111110111
10101111101110010000000000011000
00111100000001000001000000000000
10001111101001010000000000011000
00001100000100000000000011101100
00100100100001000000010000110000
10001111101111110000000000010100
00100111101111010000000000100000
00000011111000000000000000001000
00000000000000000001000000100001
```

Programando en lenguaje ensamblador (crudo)

Da una sintaxis con mnemónicos para las instrucciones, registros y modos de direccionamiento.

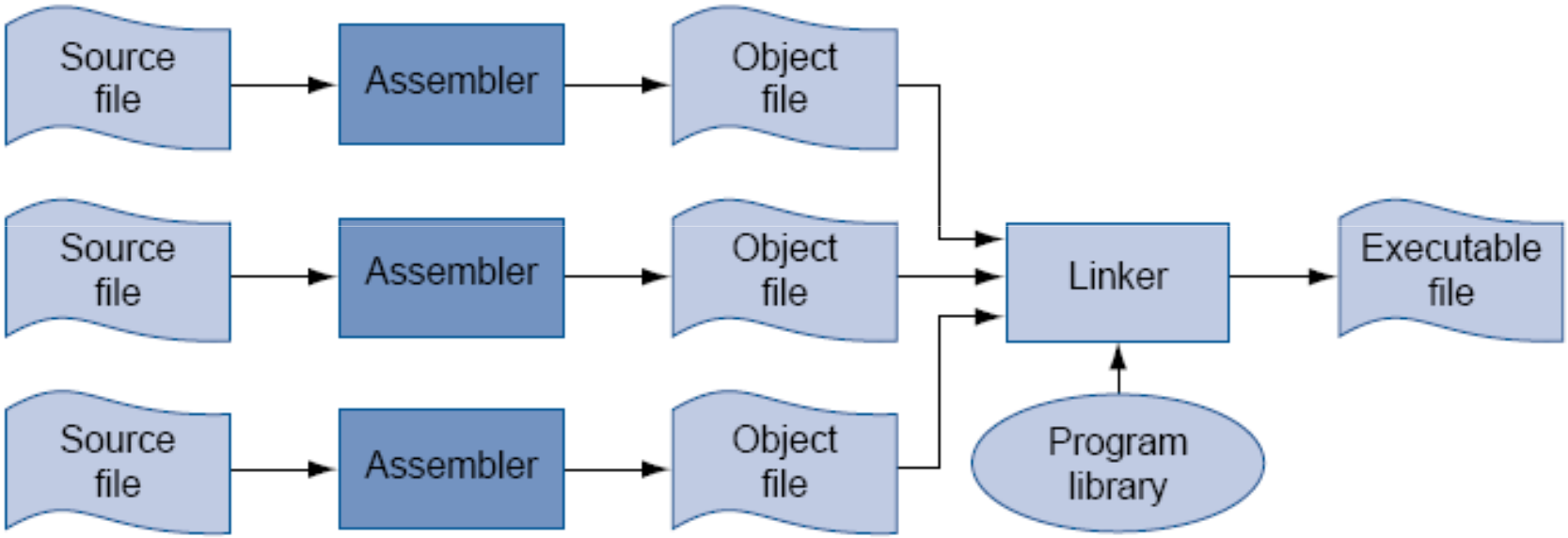
```
addiu    $29, $29, -32
sw       $31, 20($29)
sw       $4,  32($29)
sw       $5,  36($29)
sw       $0,  24($29)
sw       $0,  28($29)
lw       $14, 28($29)
lw       $24, 24($29)
multu    $14, $14
addiu    $8,  $14, 1
slti     $1,  $8, 101
sw       $8,  28($29)
mflo     $15
addu     $25, $24, $15
bne      $1,  $0, -9
sw       $25, 24($29)
lui      $4,  4096
lw       $5,  24($29)
jal      1048812
addiu    $4,  $4, 1072
lw       $31, 20($29)
addiu    $29, $29, 32
jr       $31
move     $2,  $0
```

Programando en lenguaje ensamblador (con abstracciones)

Agrega: directivas, labels, pseudoinstrucciones, modos de direccionamiento no soportados por el procesador.

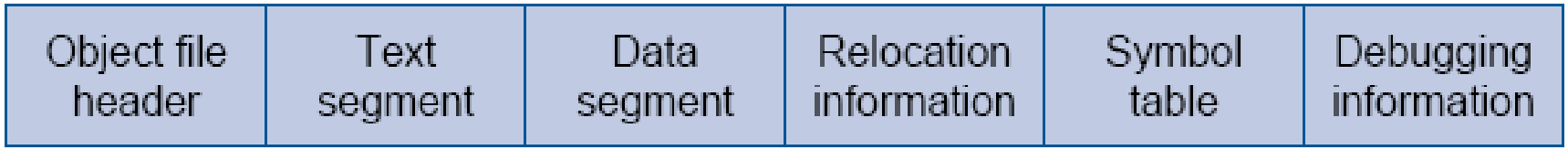
```
        .text
        .align    2
        .globl    main
main:
        subu      $sp, $sp, 32
        sw        $ra, 20($sp)
        sd        $a0, 32($sp)
        sw        $0,  24($sp)
        sw        $0,  28($sp)
loop:
        lw        $t6, 28($sp)
        mul       $t7, $t6, $t6
        lw        $t8, 24($sp)
        addu      $t9, $t8, $t7
        sw        $t9, 24($sp)
        addu      $t0, $t6, 1
        sw        $t0, 28($sp)
        ble       $t0, 100, loop
        la        $a0, str
        lw        $a1, 24($sp)
        jal       printf
        move      $v0, $0
        lw        $ra, 20($sp)
        addu      $sp, $sp, 32
        jr        $ra

        .data
        .align    0
str:
        .asciiz   "The sum from 0 .. 100 is %d\n"
```

Proceso de producir un ejecutable



Ensamblador

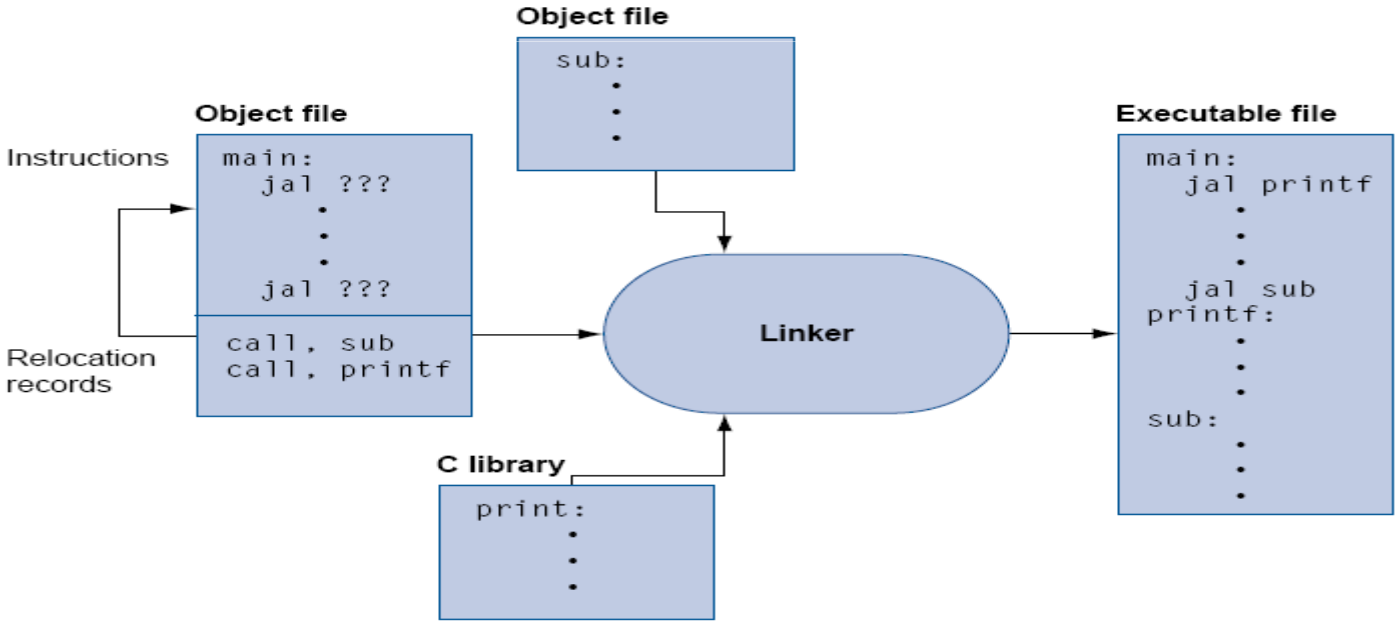
- Programa que traduce el código assembly a binario.
- Genera como salida un archivo objeto.



Archivo objeto generado por un assembler en Unix

Linker

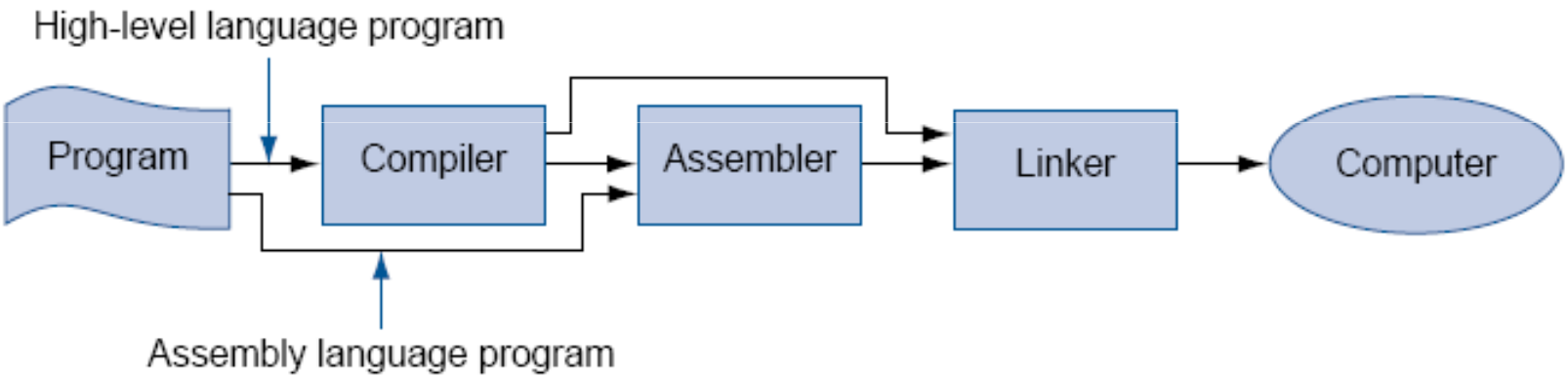
- Enlaza los archivos objeto y genera el ejecutable.
- Resuelve las referencias a bibliotecas externas.
- Asigna las posiciones finales en memoria.



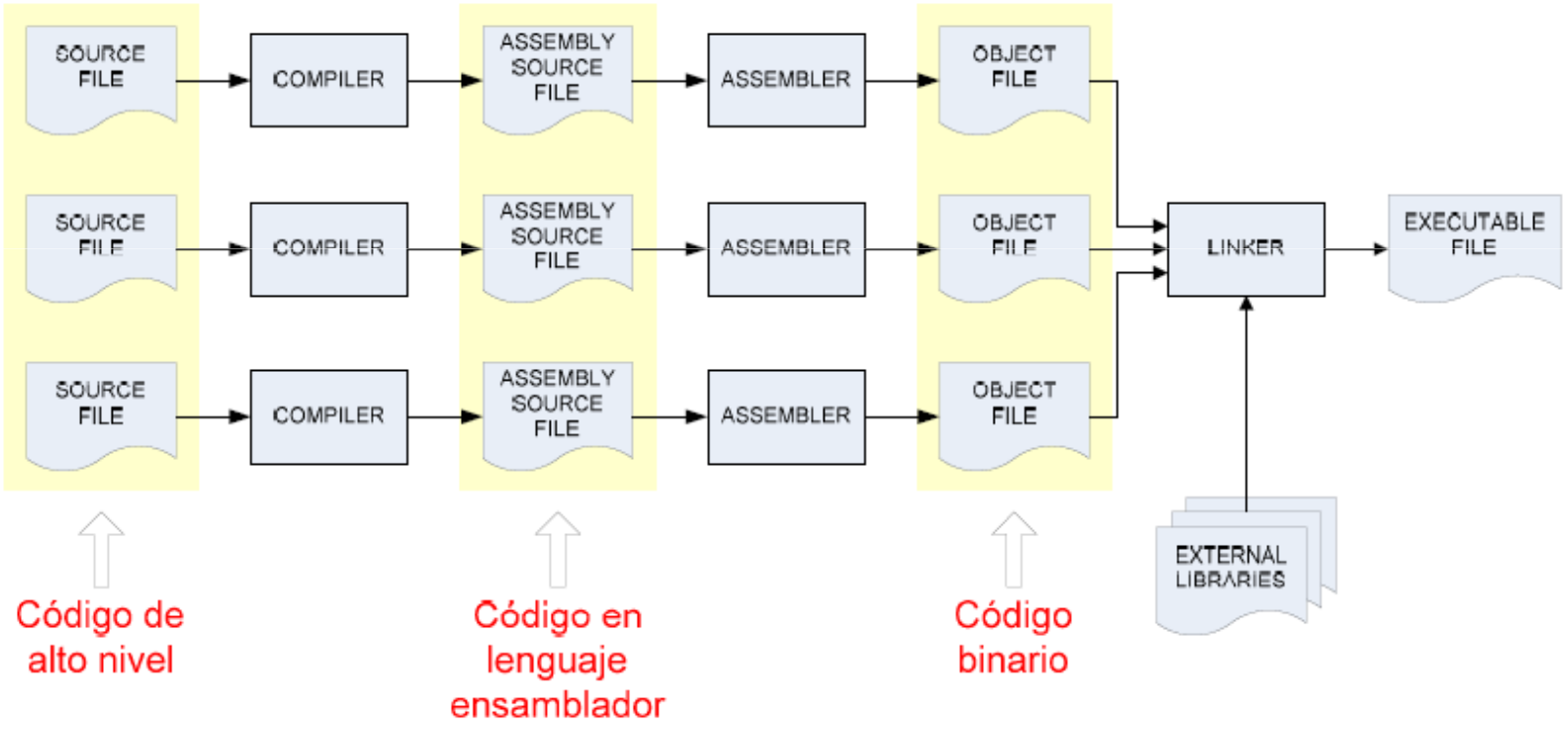
Necesidad de mayores de abstracciones: lenguajes de alto nivel

```
#include <stdio.h>

int
main (int argc, char *argv[])
{
     int i;
     int sum = 0;

     for (i = 0; i <= 100; i = i + 1) sum = sum + i * i;
     printf ("The sum from 0 .. 100 is %d\n", sum);
}
```

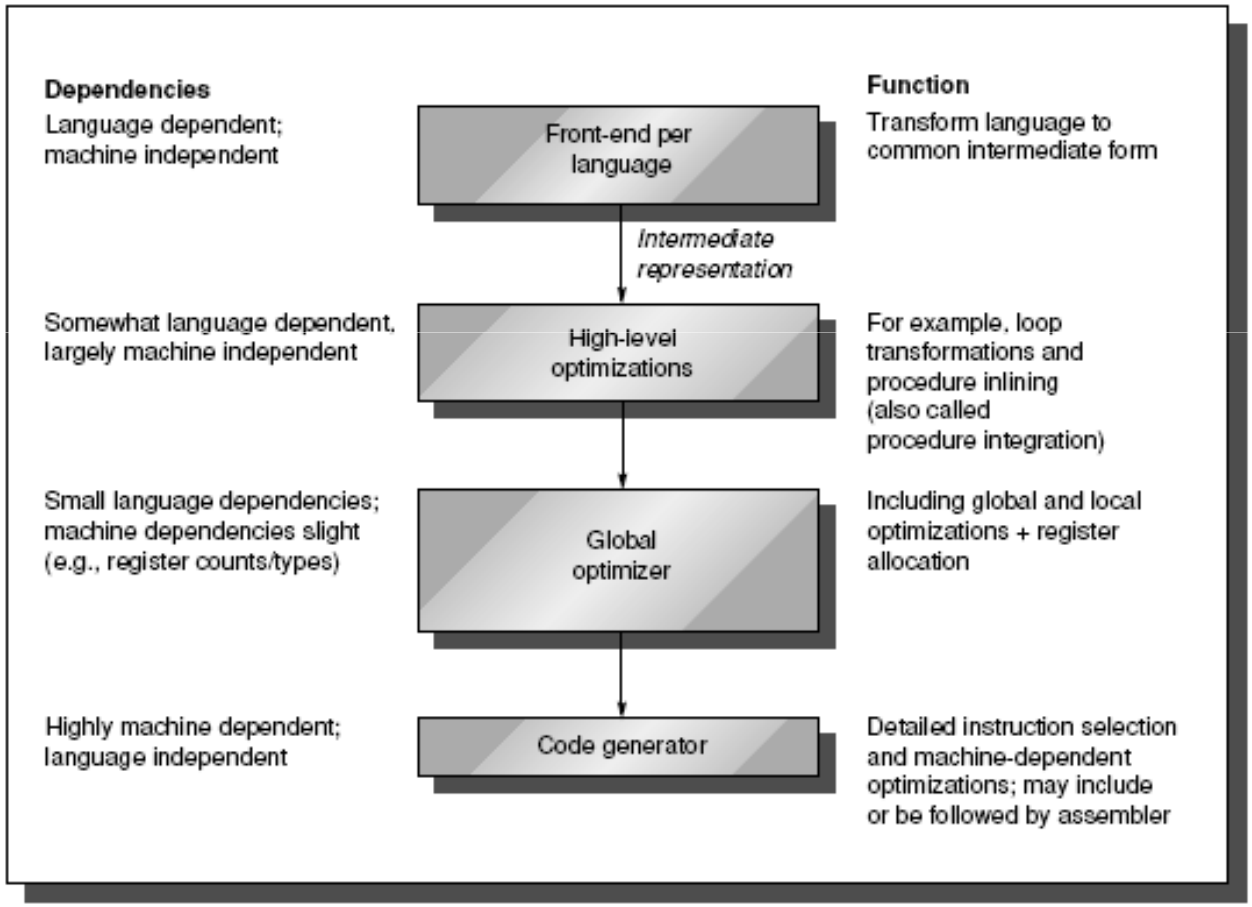
El compilador



Generación de un ejecutable



El Roll de los Compiladores



Expectativas para aplicaciones del tipo de escritorio

- Arquitectura carga/almacenamiento
- Registros de propósito general
- Modos de direccionamiento
  - Desplazamiento
  - Inmediato
- Tipos de datos
  - Enteros, 8, 16, 32 y 64 bits
  - Coma flotante, IEEE 754
- Instrucciones simples
  - load, store, add, subtract, move reg-reg, shift
  - Compare, branch, jump, call, return
- Codificación de instrucción fija

La Arquitectura MIPS

MIPS 32

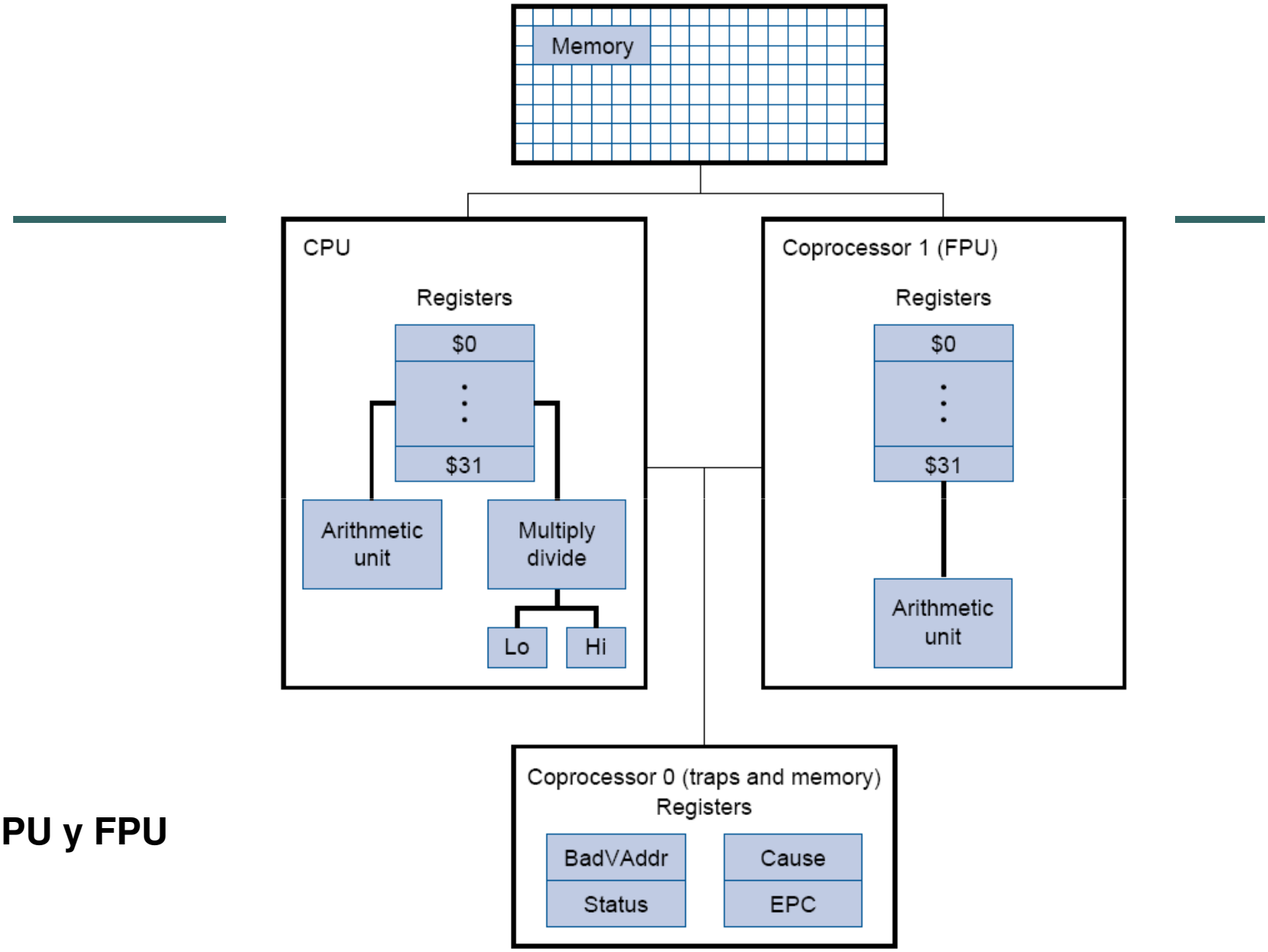
La Arquitectura MIPS

- Conjunto de instrucciones simple L/S
- Diseñada para eficiencia del *pipeline*
- Eficiencia para el compilador

La Arquitectura MIPS

- Registros
  - 32 registros de 32 bits: $0, $1, $2, $31 (int)
  - 32 registros de 32 bits: $f0, $f1, $f2, $f31 (fp)
- Tipos de datos
  - Bytes (8 bits), Half Word (16 bits), Words (32 bits)
  - Double words (fp)
- Modos de direccionamiento
  - Inmediato
  - Desplazamiento

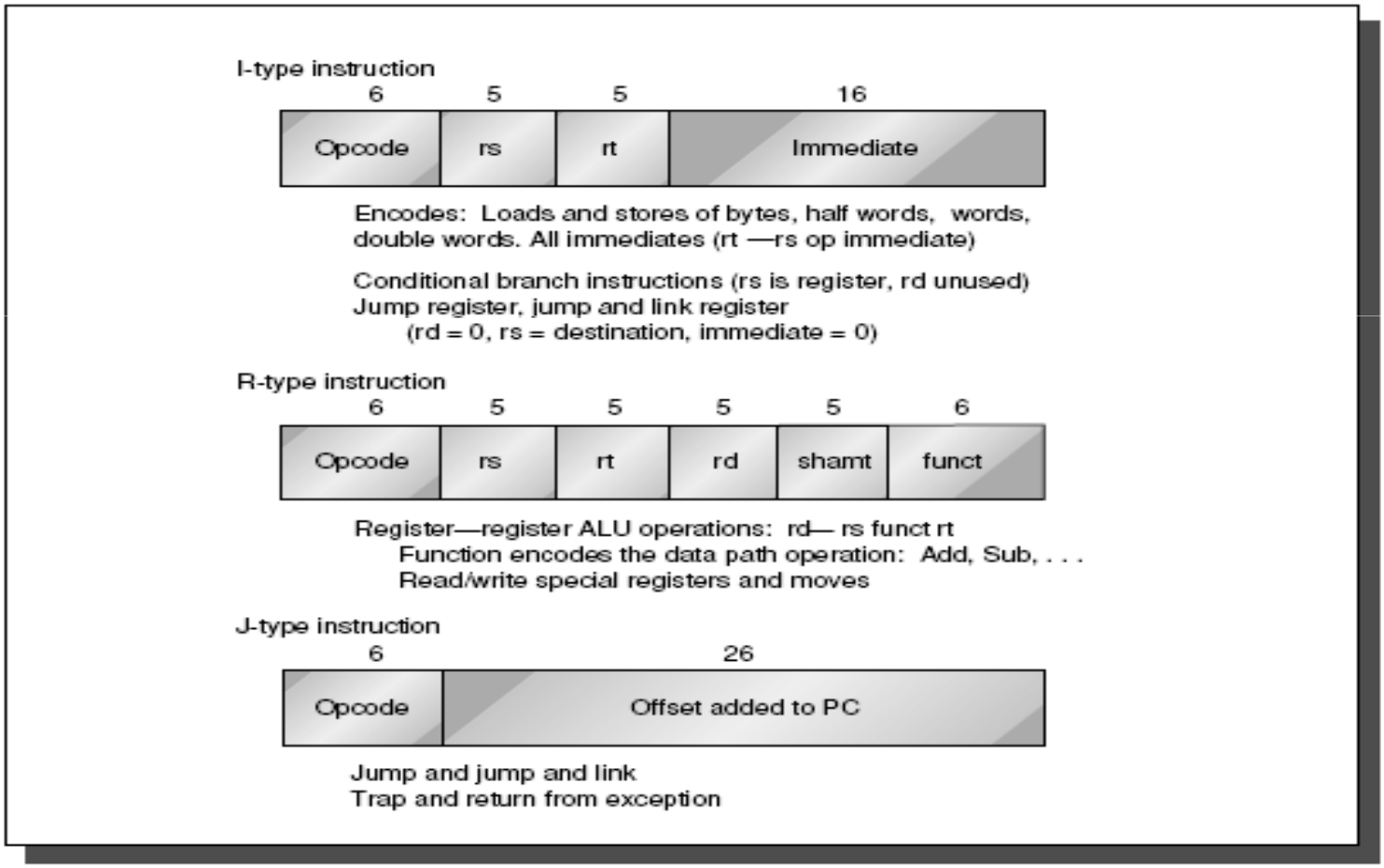
CPU y FPU


Memory
CPU
Registers
$0
$31
Arithmetic unit
Multiply divide
Lo
Hi
Coprocessor 1 (FPU)
Registers
$0
$31
Arithmetic unit
Coprocessor 0 (traps and memory)
Registers
BadVAddr
Cause
Status
EPC

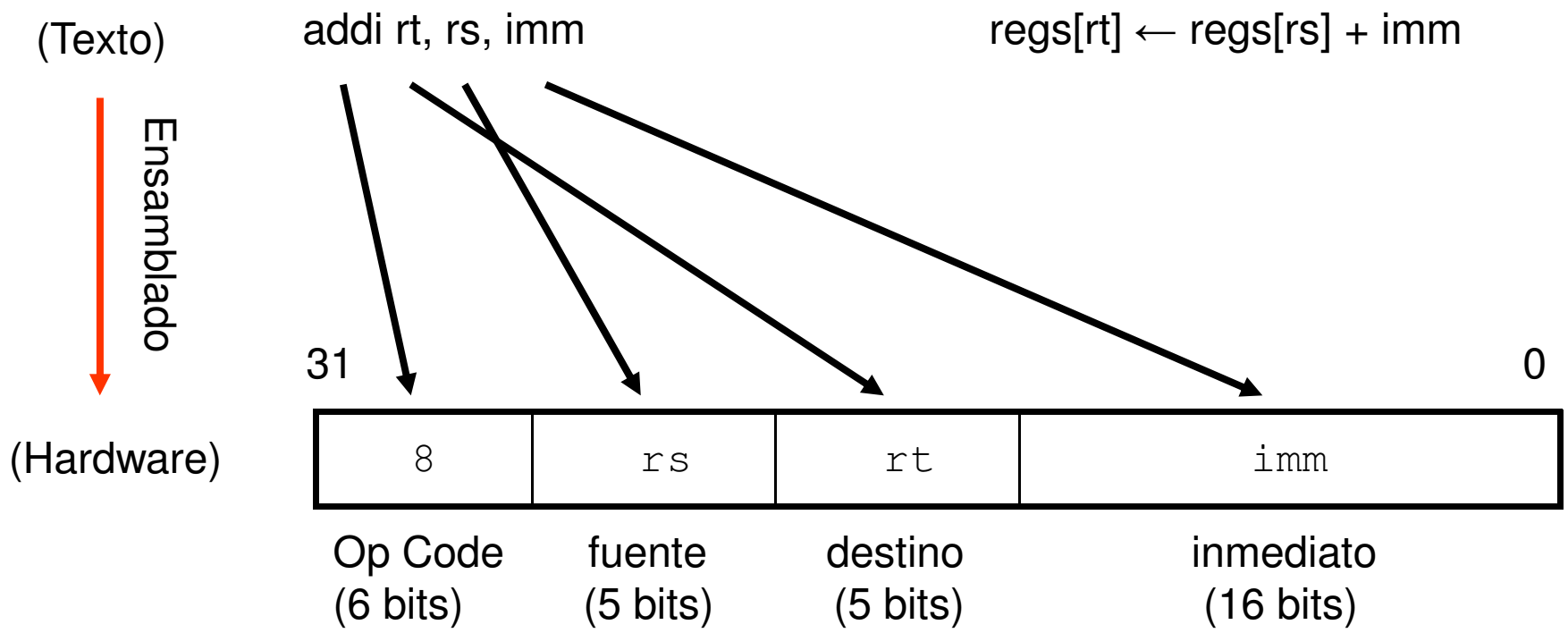
Convención de uso de registros

Registers			
Name	Number	Use	Callee must preserve?
$zero	$0	constant 0	N/A
$at	$1	assembler temporary	No
$v0–$v1	$2–$3	values for function returns and expression evaluation	No
$a0–$a3	$4–$7	function arguments	No
$t0–$t7	$8–$15	temporaries	No
$s0–$s7	$16–$23	saved temporaries	Yes
$t8–$t9	$24–$25	temporaries	No
$k0–$k1	$26–$27	reserved for OS kernel	No
$gp	$28	global pointer	Yes
$sp	$29	stack pointer	Yes
$fp	$30	frame pointer	Yes
$ra	$31	return address	N/A

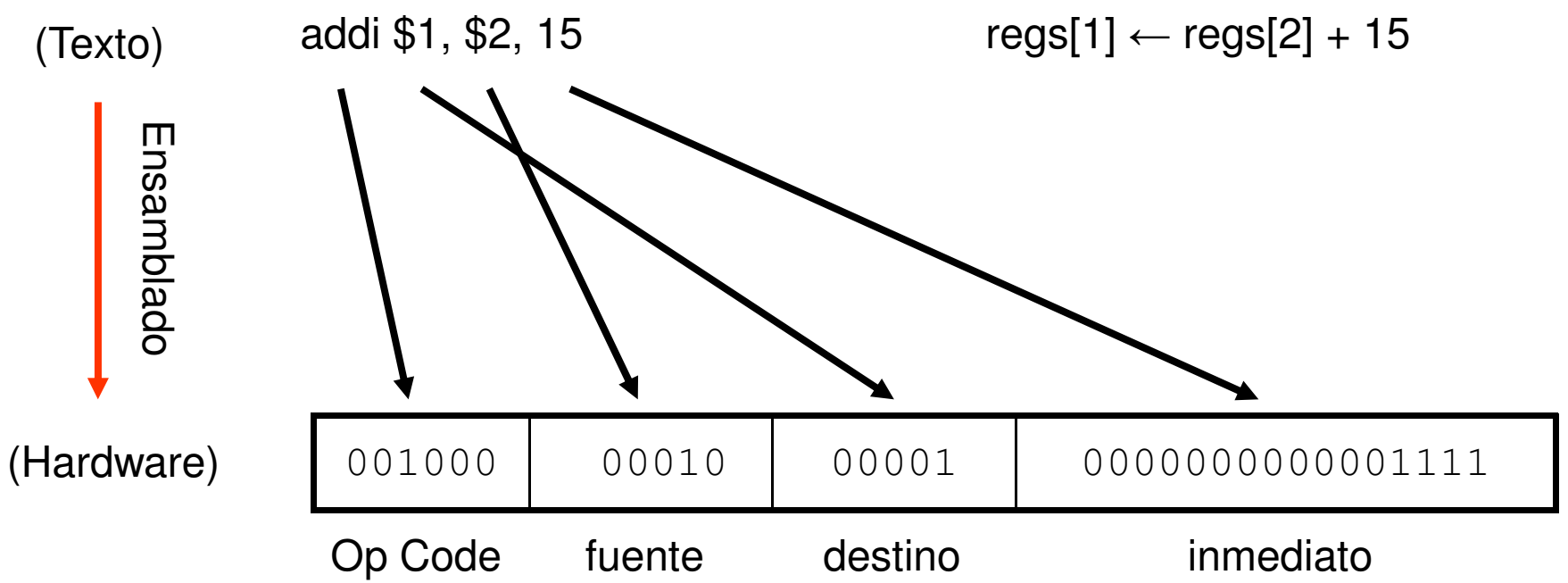
Formatos de Instrucciones MIPS



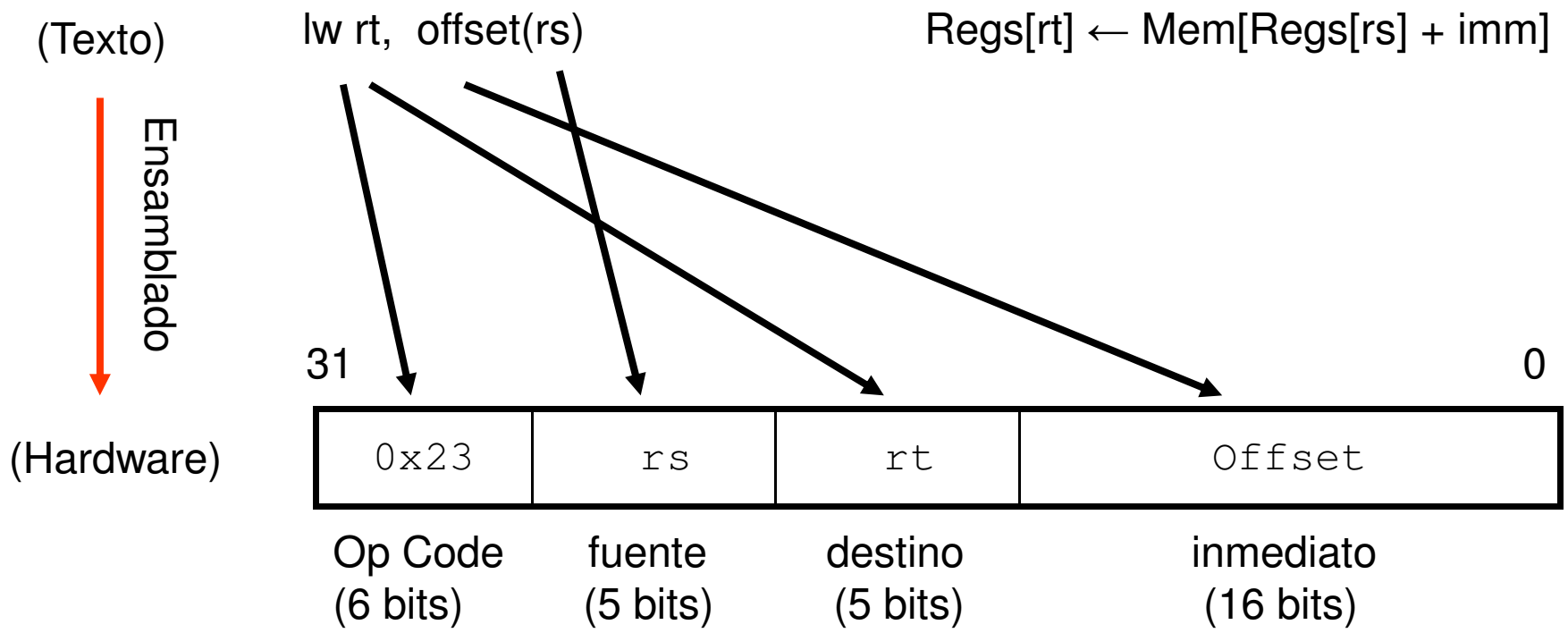
Instrucción del Tipo I: addi



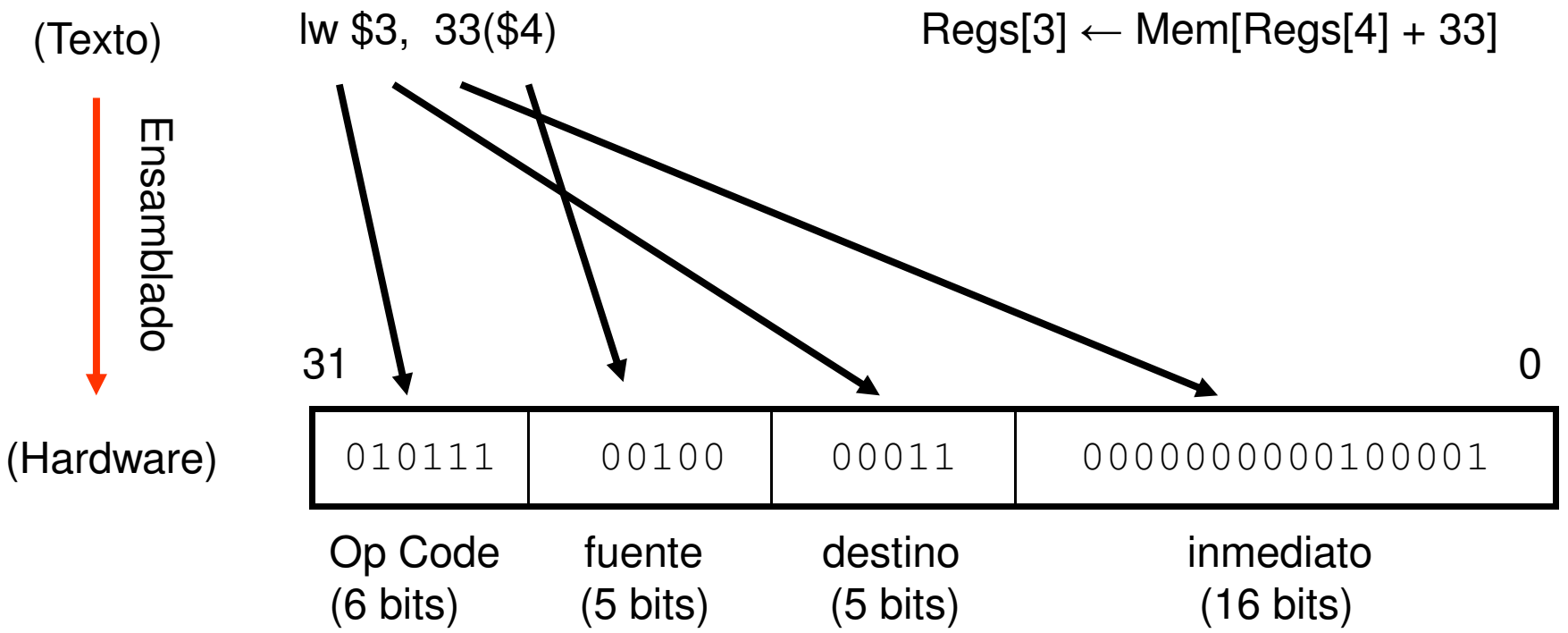
Ejemplo de addi



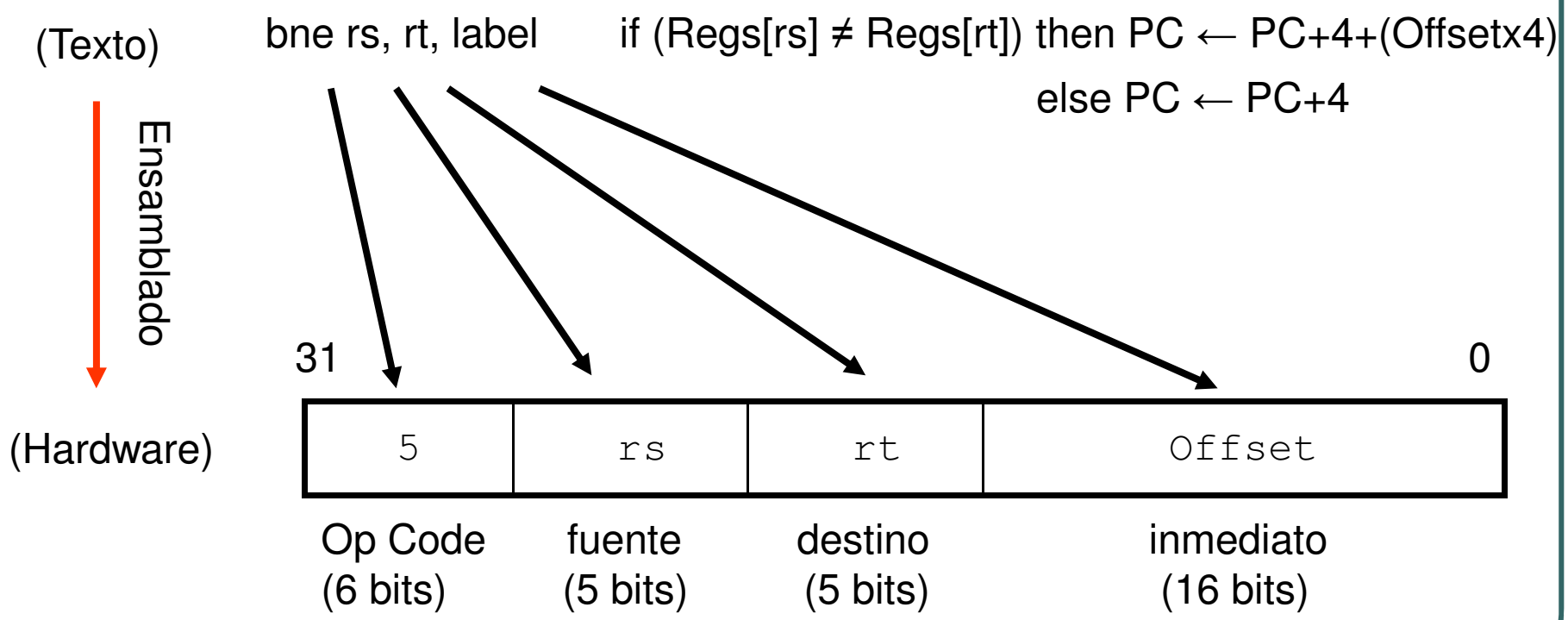
Instrucción del Tipo I: lw



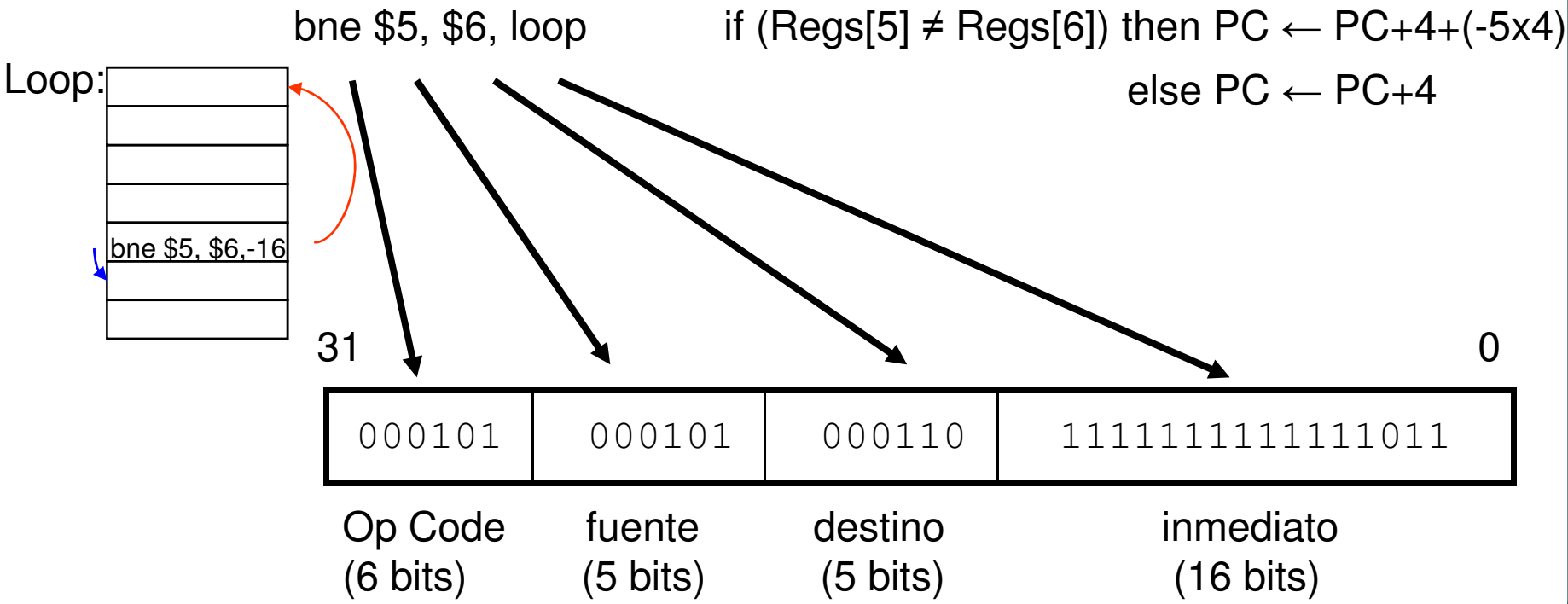
Ejemplo de lw



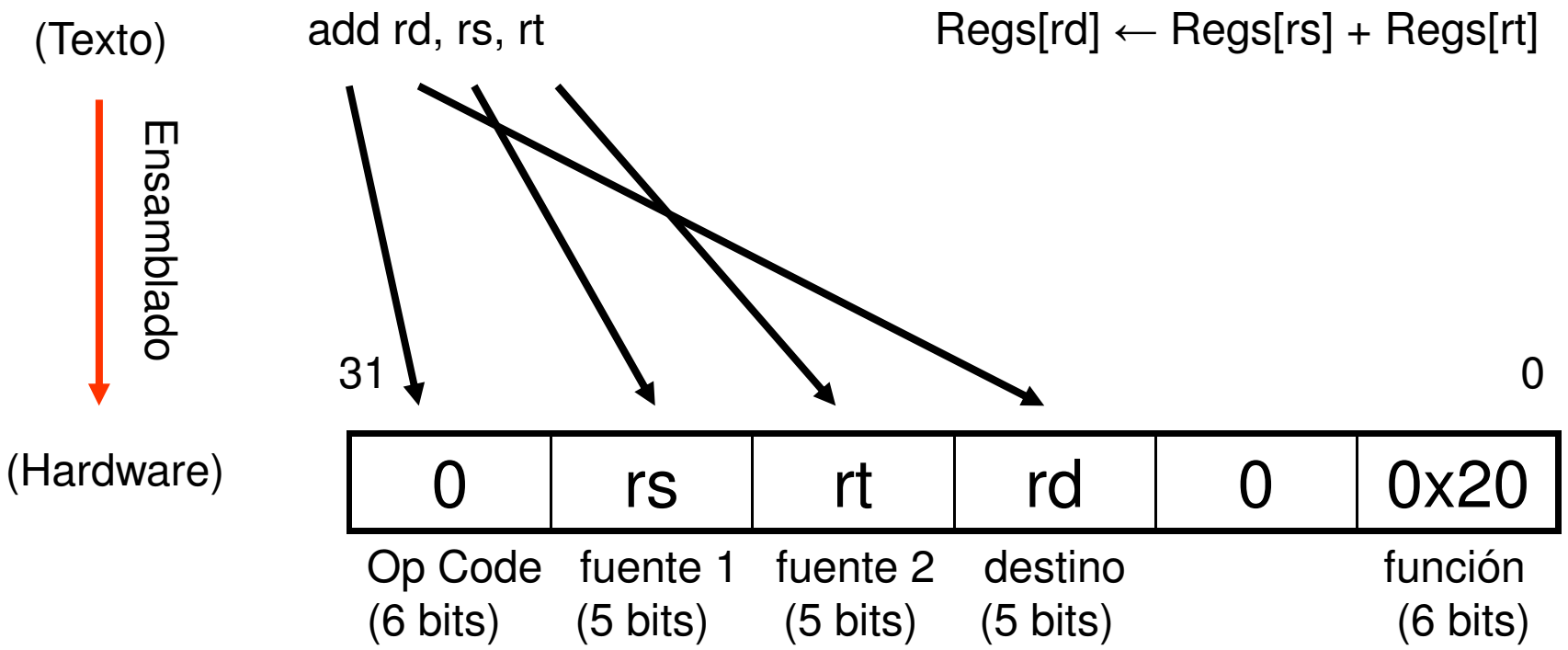
Instrucción del Tipo I: bne



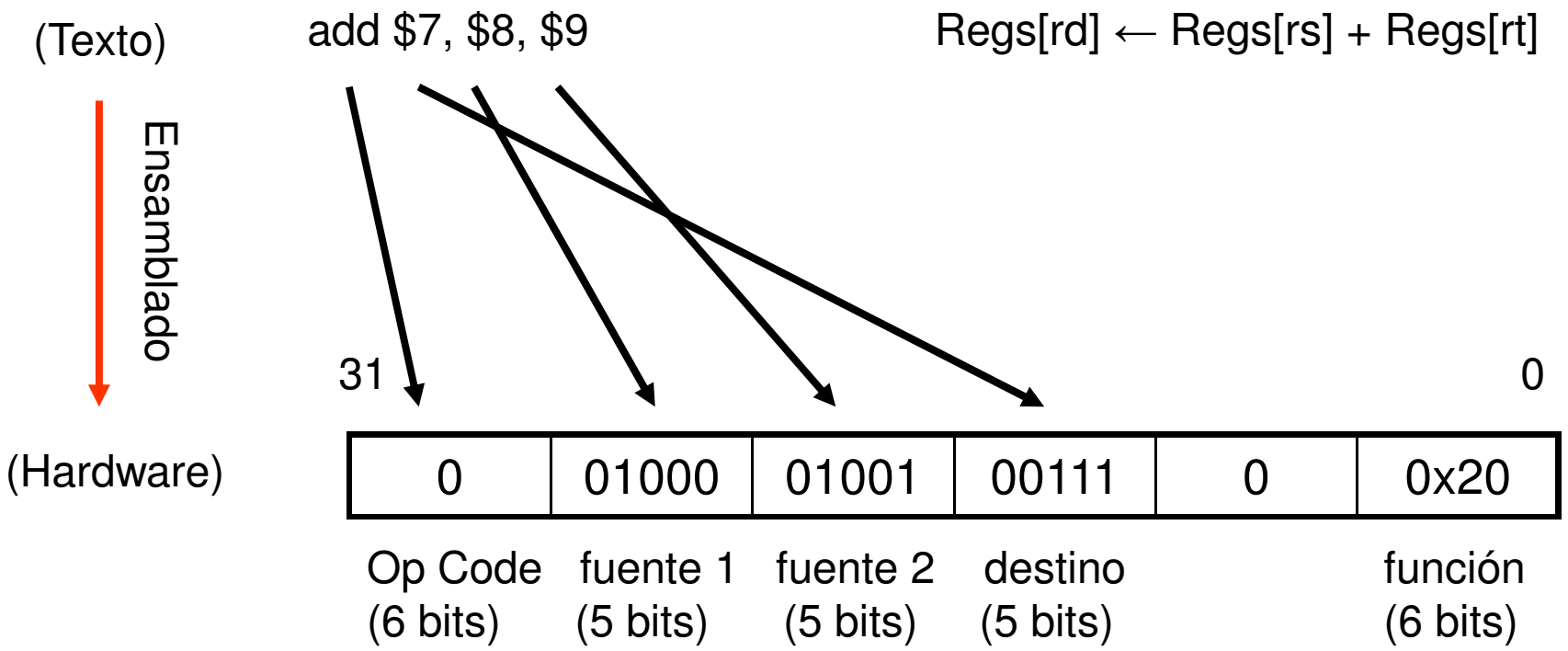
Ejemplo de bne



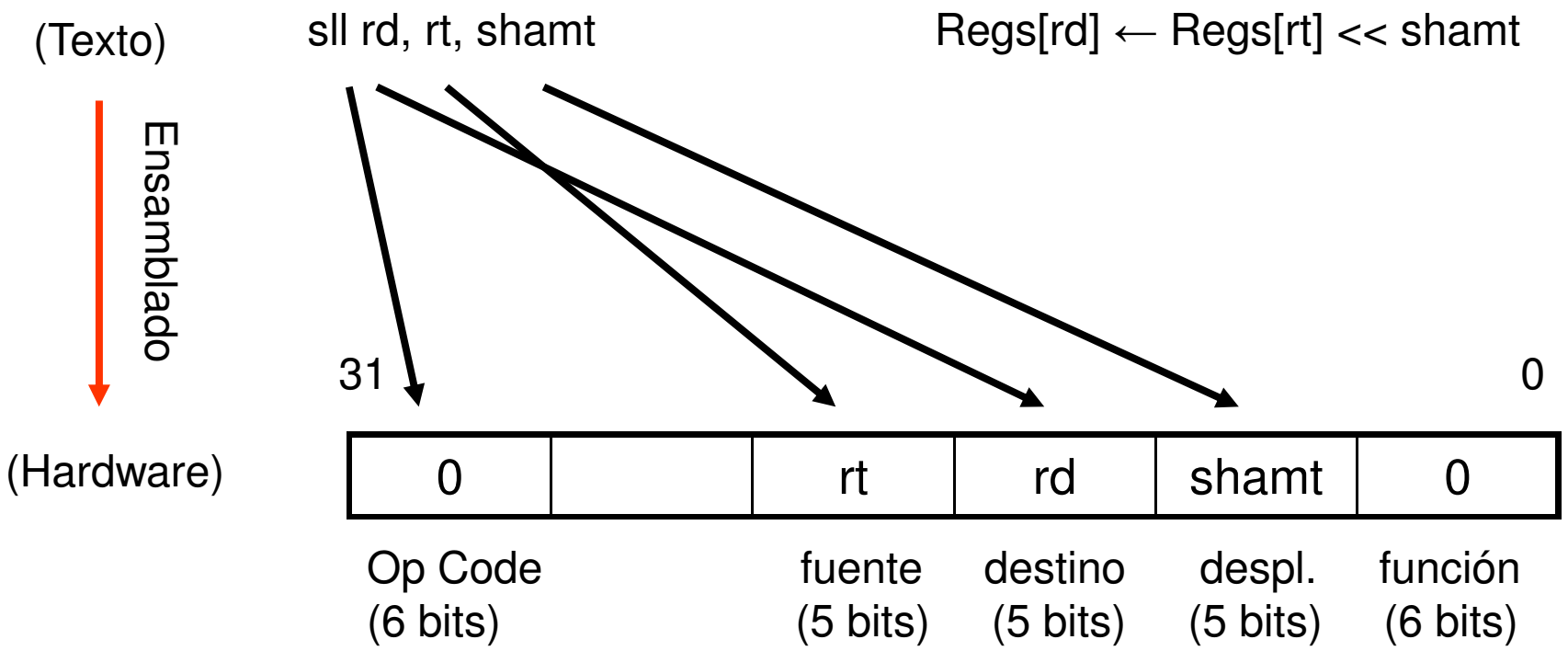
Instrucción del Tipo R: add



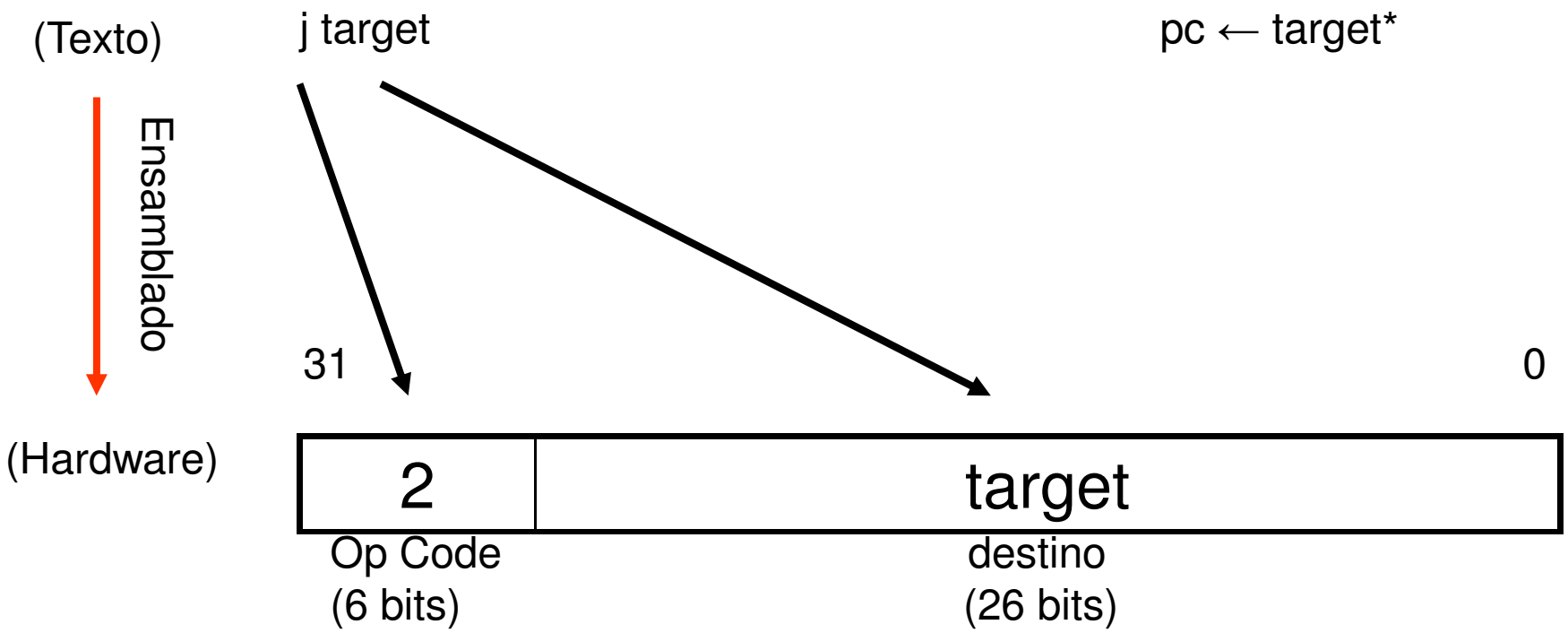
Ejemplo de add:



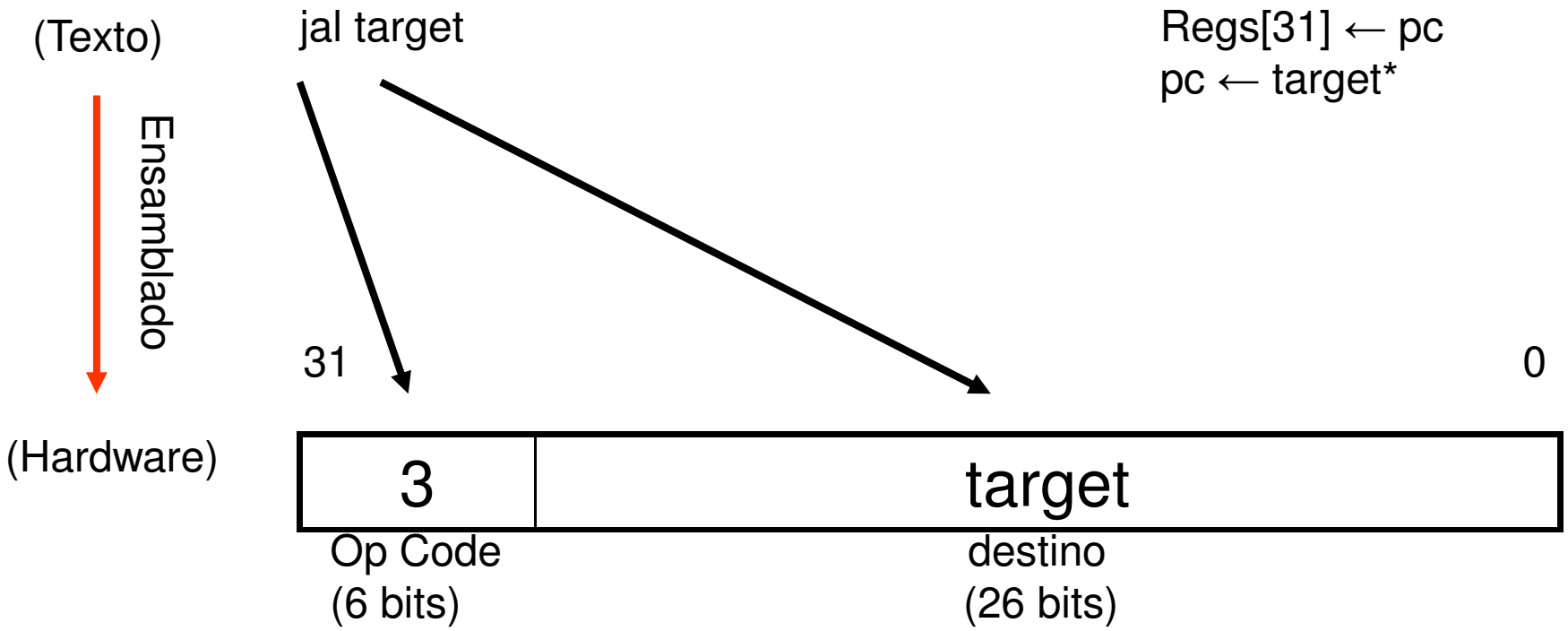
Instrucción del Tipo R: sll



Instrucción del Tipo J: j



Instrucción del Tipo J: jal



Instrucciones Aritmético Lógicas

- add, addu, addi, addiu, sub
- and, andi, xor, xori, nor, or, ori
- clo, clz
- div (cociente en lo, resto en hi), divu
- mult, multu (resultado en hi lo)
- mul (especifica rdest)
- madd, maddu, msub, msubu (resultado en hi lo)
- sll, sllv, sra, srav, srl, srlv

Instrucciones que manipulan constantes y de comparación

- lui
- slt, sltu, slti, sltiu,

Instrucciones de branch

- beq
- bgez, bgezal
- bgtz
- blez
- bltz, bltzal
- bne

Instrucciones de jump

- j, jal
- jr, jalr

Instrucciones de carga-almacenamiento

- lb, lbu, lh, lhu, lw
- sb, sh, sw

Instrucciones de movimiento de datos

- mfhi, mflo
- mthi, mtlo
- movn
- movz

Instrucción de llamada al sistema

- eret
- syscall
- break

Instrucción que no hace nada

- nop

Pseudoinstrucciones

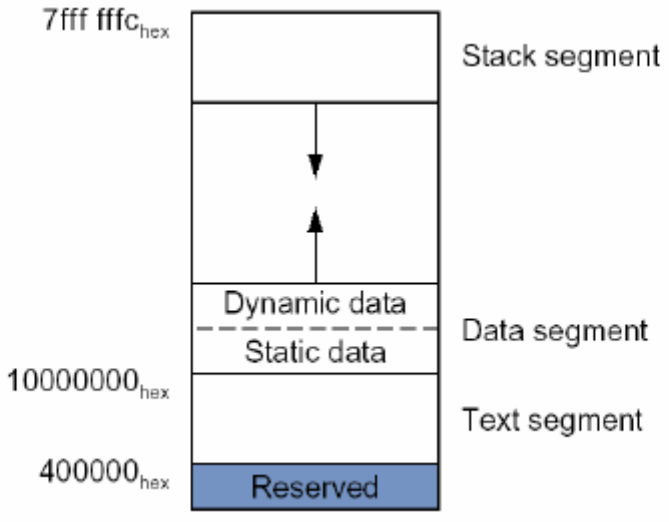
Name	instruction syntax	Real instruction translation	meaning
Move	move $rt,$rs	addi $rt,$rs,0	R[rt]=R[rs]
Clear	clear $rt	add $rt,$zero,$zero	R[rt]=0
Load Address	la $at, LabelAddr	lui $at, LabelAddr[31:16]; ori $at,$at, LabelAddr[15:0]	$at = Label Address
Load Immediate	li $at, IMMED[31:0]	lui $at, IMMED[31:16]; ori $at,$at, IMMED[15:0]	$at = 32 bit Immediate value
Branch if greater than	bgt $rs,$rt,Label	slt $at,$rt,$rs; bne $at,$zero,Label	if(R[rs]>R[rt]) PC=Label
Branch if less than	blt $rs,$rt,Label	slt $at,$rs,$rt; bne $at,$zero,Label	if(R[rs]<R[rt]) PC=Label
Branch if greater than or equal	bge $rs,$rt,Label	slt $at,$rs,$rt; beq $at,$zero,Label	if(R[rs]>=R[rt]) PC=Label
Branch if less than or equal	ble $rs,$rt,Label	slt $at,$rt,$rs; beq $at,$zero,Label	if(R[rs]<=R[rt]) PC=Label
Branch if greater than unsigned	bgtu $rs,$rt,Label		if(R[rs]=>R[rt]) PC=Label
Branch if greater than zero	bgtz $rs,$rt,Label		if(R[rs]>0) PC=Label
Multiplies and returns only first 32 bits	mul $1, $2, $3	mult $2, $3; mflo $1	$1 = $2 * $3

Instruction	gap	gcc	gzip	mcf	perl	Integer average
load	44.7%	35.5%	31.8%	33.2%	41.6%	37%
store	10.3%	13.2%	5.1%	4.3%	16.2%	10%
add	7.7%	11.2%	16.8%	7.2%	5.5%	10%
sub	1.7%	2.2%	5.1%	3.7%	2.5%	3%
mul	1.4%	0.1%				0%
compare	2.8%	6.1%	6.6%	6.3%	3.8%	5%
cond branch	9.3%	12.1%	11.0%	17.5%	10.9%	12%
cond move	0.4%	0.6%	1.1%	0.1%	1.9%	1%
jump	0.8%	0.7%	0.8%	0.7%	1.7%	1%
call	1.6%	0.6%	0.4%	3.2%	1.1%	1%
return	1.6%	0.6%	0.4%	3.2%	1.1%	1%
shift	3.8%	1.1%	2.1%	1.1%	0.5%	2%
and	4.3%	4.6%	9.4%	0.2%	1.2%	4%
or	7.9%	8.5%	4.8%	17.6%	8.7%	9%
xor	1.8%	2.1%	4.4%	1.5%	2.8%	3%
other logical	0.1%	0.4%	0.1%	0.1%	0.3%	0%
load FP						0%
store FP						0%
add FP						0%
sub FP						0%
mul FP						0%
div FP						0%
mov reg-reg FP						0%
compare FP						0%
cond mov FP						0%
other FP						0%

FIGURE 2.32 MIPS dynamic instruction mix for five SPECint2000 programs. Note that integer register-register move instructions are included in the or instruction. Blank entries have the value 0.0%.

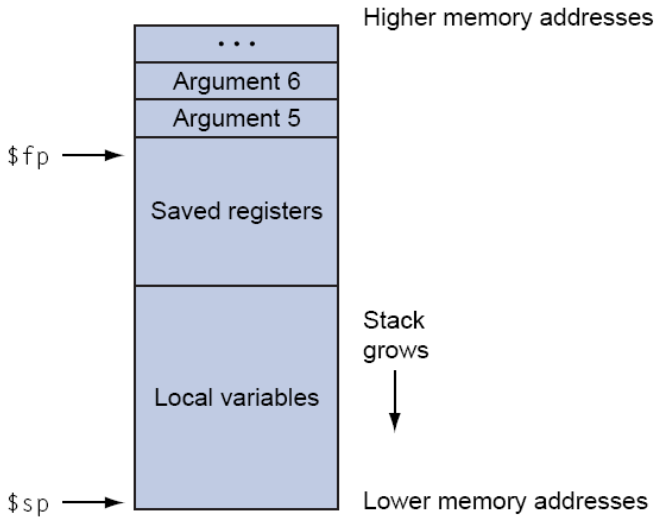
Modelo de memoria

- La arquitectura MIPS divide la memoria en tres secciones:
  - Segmento de texto.
  - Segmento de datos (estáticos y dinámicos).
  - Pila *(stack)*.



Lamados a funciones

- Debe respetarse la convención para el uso de registros.
- Debe tomarse una porción de la pila (*stack frame*):
  - Para almacenar los argumentos pasados.
  - Para almacenar registros de largo plazo.
  - Para almacenar las variables locales al procedimiento.



MIPS Microprocessors

Model	Frequency (MHz)	Year	Process (µm)	Transistors (millions)	Die Size (mm^2)	Pin Count	Power (W)	Voltage (V)	D. cache (KB)	I. cache (KB)	L2 Cache	L3 Cache
R2000	8–16.67	1985	2.0	0.11	?	?	?	?	32	64	None	None
R3000	12–40	1988	1.2	0.11	66.12	145	4	?	64	64	0-256 KB External	None
R4000	100	1991	0.8	1.35	213	179	15	5	8	8	1 MB External	None
R4400	100–250	1992	0.6	2.3	186	179	15	5	16	16	1-4 MB External	None
R4600	100–133	1994	0.64	2.2	77	179	4.6	5	16	16	512 KB External	None
R4650	133–180	1994	0.64	2.2?	77?	179	4.6?	5	16	16	512 KB External	None
R4700	100–200	1996	0.5	2.2?	?	179	?	?	16	16	External	none
R5000	150–200	1996	0.35	3.7	84	223	10	3.3	32	32	1 MB External	None
R8000	75–90	1994	0.7	2.6	299	591+591	30	3.3	16	16	4 MB External	None
R10000	150–250	1996	0.35, 0.25	6.7	299	599	30	3.3	32	32	512 KB–16 MB external	None
R12000	270–400	1998	0.25, 0.18	6.9	204	600	20	4	32	32	512 KB–16 MB external	None
RM7000	250–600	1998	0.25, 0.18, 0.13	18	91	304	10, 6, 3	3.3, 2.5, 1.5	16	16	256 KB internal	1 MB external
R14000	500–600	2001	0.13	7.2	204	527	17	?	32	32	512 KB–16 MB external	None
R16000	700–1000	2002	0.11	?	?	?	20	?	32	32	512 KB–16 MB external	None
R24K	750+	2003	65 nm	?	0.83	?	?	?	64	64	4-16 MB external	

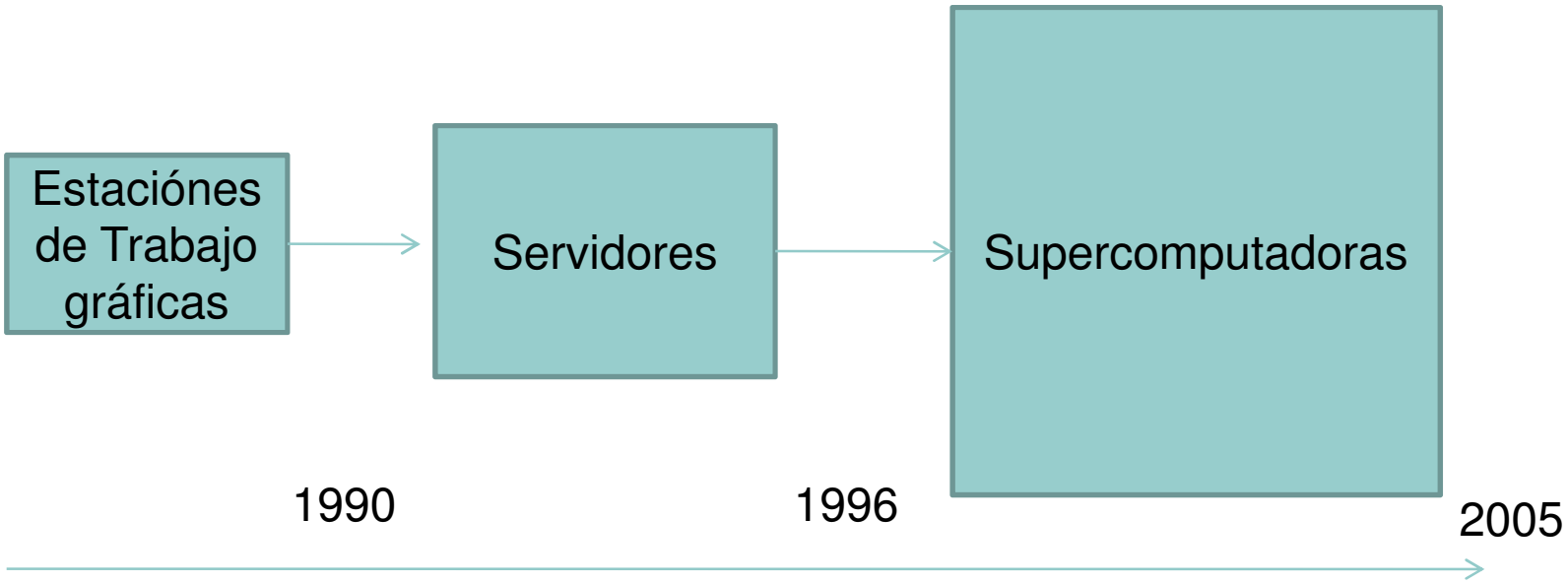
Sistemas basados en MIPS

- Routers
- Cablemodem
- ADSL
- Smart Cards
- Impresoras laser
- Decodificadores
- PlaySation 2
- Robots
- Celulares

Arquitectura MIPS presente en los siguientes procesadores

- IDT RC32438; ATI Xilleon; Alchemy Au1000, 1100, 1200; Broadcom Sentry5; RMI XLR7xx, Cavium Octeon CN30xx, CN31xx, CN36xx, CN38xx and CN5xxx; Infineon Technologies EasyPort, Amazon, Danube, ADM5120, WildPass, INCA-IP, INCA-IP2; Microchip Technology PIC32; NEC EMMA and EMMA2, NEC VR4181A, VR4121, VR4122, VR4181A, VR5432, VR5500; Oak Technologies Generation; PMC-Sierra RM11200; QuickLogic QuickMIPS ESP; Toshiba *Donau*, Toshiba TMPR492x, TX4925, TX9956, TX7901.

Supercomputadoras basadas en MIPS.



Evolución SGI

Supercomputadoras basadas en MIPS.

SGI Origin 2000

Supercomputadoras basadas en MIPS. Sircortex startup (2007)

Sircortex SC5832

CHIP (nodo): multinucleo 6 MIPS 64
Topologia Kautz graph
Controlador de memoria crossbar
interconnect DMA engine,
Gigabit Ethernet y PCI Express
en un chip que consume 10 watts
Desempeño pico de 6 Gigaflops.
Configuración tope (un solo gabinete):
972 nodos, total de 5832 nucleos
MIPS64 y 8.2 teraFLOPS de desempeño
pico.

FIN