



MIPS32 R2000/R3000 INTRODUCCIÓ

- EL MIPS R2000 és l'original
 - Apareix cap el 1985 i funcionava a 8 Mhz
 - L'R3000 és idèntic però funcionant a més velocitat
- És una màquina de 32 Bits
 - És *Bi-Endian* (es pot escollir)
 - En un SGI és Big-Endian
 - L'*Spim* utilitzat sota un *ix86* és *Little-Endian*
 - Pot adreçar fins a 4 GBytes de memòria



MIPS32 R2000/R3000 ARQUITECTURA

- RISC
 - 32 Registres de propòsit general
 - Arquitectura *Load/Store*
 - Caché controlable per Software
 - No té paraula d'estat (*PSW*)
- Exemples
 - SGI (MIPS32, MIPS64)
 - PlayStation
 - Nintendo 64 (MIPS64)



CONTINGUT (I)

- MODEL DE MEMÒRIA
- REGISTRES
- INSTRUCCIONS
 - Tipus i Codificació (R, I, J)
 - Load / Store (Accés L/E a memòria)
 - Control de fluxe (Estructures Iteratives, Condicionals)
 - Aritmètiques (Càlculs Aritmètics: suma, resta, etc...)
 - Lògiques (Càlculs Lògics: AND, OR, etc...)
- MODES D'ADREÇAMENT
- CRIDES A SUBRUTINES

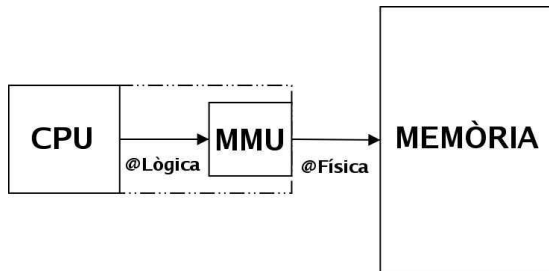


CONTINGUT (II)

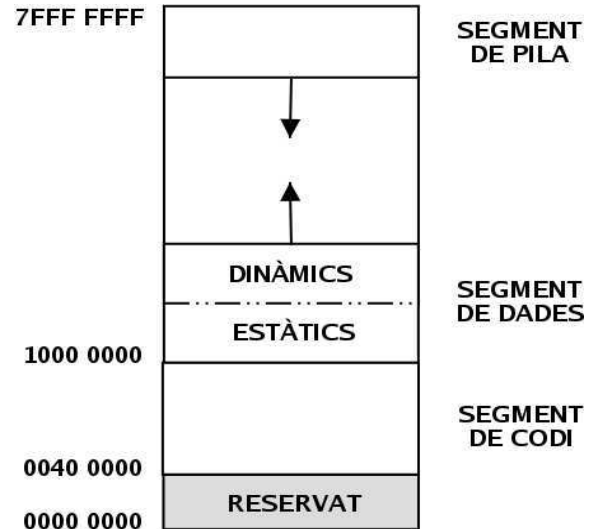
- EXCEPCIONS
- INTERRUPCIIONS
- ENTRADA / SORTIDA

MODEL DE MEMÒRIA

- Tots els programes veuen la memòria de la mateixa manera
 - Memòria Virtual
 - S'utilitzen només 2^{31} bytes



Els programes poden estar a la mateixa @Lògica i a @Físiques diferents



ARQUITECTURA LOAD/STORE (I)

- L'arquitectura de MIPS és *Load/Store*
 - Totes les càlculs es realitzen en els registres
 - Hi haurà instruccions per **obtenir** dades de **memòria** i **copiar-les** als **registres** i viceversa
 - Exemple:
 - Sumar dos valors (*en C*): $a = b + c$;
 - Assemblador de tres adreces amb ordre natural dels operands

add a, b, c

- **a** és el registre destí
- **b** i **c** són els registres font

ARQUITECTURA LOAD/STORE (II)

- Traducció d'un tall de codi en C

```
a = b + c;      |      add a, b, c
                  |
d = a - e;      |      sub d, a, e
                  |
```

- Existeixen 32 registres (0..31)
 - Amb l'assemblador s'anomenen \$0, \$1, ..., \$31
- *Com es calcula $f = (g + h) - (i + j)$?*
(Suposant que les diferents variables són als registres)

ARQUITECTURA LOAD/STORE (III): SOLUCIÓ

- $f = (g + h) - (i + j)$

```
add temp0, g, h      # Càlcul a temp0 del primer terme
add temp1, i, j      # Càlcul a temp1 del segon terme
sub f, temp0, temp1  # Càlcul final a f
```

- Utilització dels registres

Es mapegen les variables f, g, h, i, j als registres **16, 17, 18, 19** i **20**

```
add $8, $17, $18     # Registre 8 hi ha el càlcul g+h
add $9, $19, $20     # Registre 9 hi ha el càlcul i+j
sub $16, $8, $9      # Registre 16 conté (g+h) - (i+j)
```

ARQUITECTURA LOAD/STORE (IV)

- Copiar una dada de Memòria a un Registre
 - Load (Càrrega)
 - En MIPS la instrucció és **lw**
- Copiar una dada d'un Registre a Memòria
 - Store (Emmagatzemament)
 - En MIPS la instrucció és **sw**

ACCÉS A MEMÒRIA (I)

- MIPS pot adreçar a *byte*
 - Pot accedir a mitjes paraules i bytes individuals (no amb **lw** i **sw**)
 - Pot accedir a la paraula (32 Bits) @0 - @3
 - Pot accedir a la mitja paraula (16 Bits) @2 - @3
- Com es tradueix la instrucció C: **$g = h + A[i]$** ?

```
lw    $8, AStart($19)
add   $17, $18, $8
```

ACCÉS A MEMÒRIA (II)

- Instrucció C: $g = h + A[i]$
 - El registre \$8 és temporal (contindrà $A[i]$)
 - Al registre \$19 hi ha el valor de l'índex i
 - Al registre \$18 hi ha la variable h
 - Al registre \$17 correspon a la variable g

```
lw    $8, AStart($19)
add   $17, $18, $8
```

AStart:	A[0]
AStart + 4:	A[1]
AStart + 8:	A[2]
	...

- Què passa amb la variable i (al registre \$19)?
 - Cal que s'incrementi de 4 en 4.

REGISTRES DE PROPÒSIT GENERAL

- REGISTRES DE PROPÒSIT GENERAL
 - Significa que amb qualsevol registre es pot realitzar qualsevol de les operacions


```
add $1, $2, $3
add $3, $4, $5
lw  $8, 50($9)
lw  $9, 50($8)
sw  $1, 50($9)
```
 - En realitat tots els registres, tenen un nom i, en algun moment donat, una funció específica
 - \$0 és el registre **número 0**, s'anomena \$zero, sempre val la **constant 0** i, per tant, no s'hi pot escriure



REPRESENTACIÓ DE LES INSTRUCCIONS

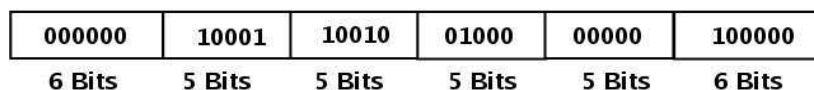
- **TOTES** LES INSTRUCCIONS DE MIPS32 OCUPEN 32 BITS
 - Una instrucció codificada a llenguatge màquina té un **format**
 - El *format d'instrucció* descomposa la instrucció en diferents **camp**s amb significat propi
 - Existeixen 3 formats diferents d'instrucció
 - **Format R**
 - **Format I**
 - **Format J**



FORMAT R DE LES INSTRUCCIONS (I)

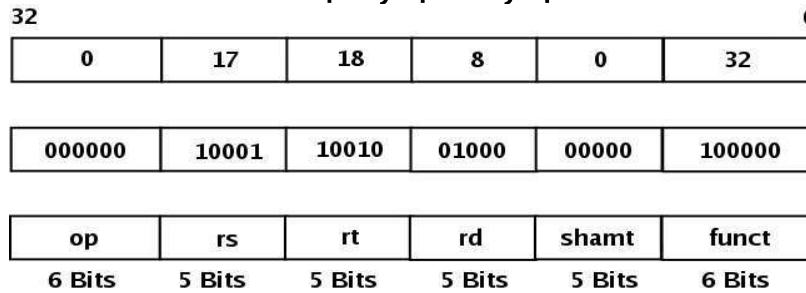
- Format d'instrucció **R**

add \$8, \$17, \$18



FORMAT R DE LES INSTRUCCIONS (II)

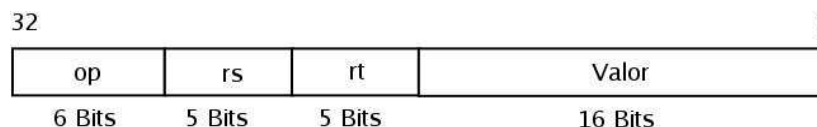
add \$8, \$17, \$18



- **op** : Codi d'operació de la instrucció
- **rs** : Codi del registre del primer operand font
- **rt** : Codi del registre del segon operand font
- **rd** : Codi del registre de l'operand destí
- **shamt** : *Quantitat de desplaçament (no usat a add)*
- **funct** : Funció (variant de l'operació)

FORMAT I DE LES INSTRUCCIONS (I)

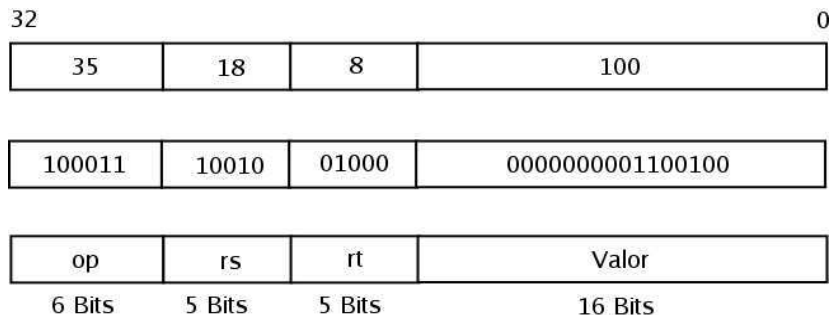
- Què passa amb *lw \$8, Astart(\$19)*?
 - No necessita el registre destí
 - Hi ha un *valor immediat*
- **Per tant, cal un nou format d'instrucció: I**



- Els camps *rs* i *rt* continuen indicant registres
- El camp *valor* té significat diferent segons la instrucció (un *valor*, un *offset*, ...)

FORMAT / DE LES INSTRUCCIONS (II)

lw \$8, 100(\$18)



- **op** : Codi d'operació de la instrucció
- **rs** : Codi de registre (varia de significat segons la instrucció)
- **rt** : Codi de registre (varia de significat segons la instrucció)
- **valor** : Varia segons instrucció: normalment un *offset* o un *valor*

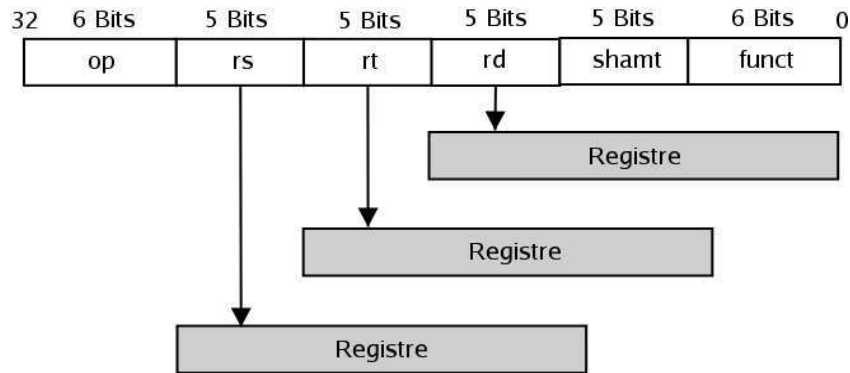
FORMATS D'INSTRUCCIÓ DE MIPS32 (I)

• RESUM DELS FORMATS CONEGUTS

Instrucció	Format	6 Bits	5 Bits	5 Bits	5 Bits	5 Bits	6 Bits
	R	op	rs	rt	rd	shamt	funct
	I	op	rs	rt	Offset		
add	R	0	Reg.	Reg.	Reg.	0	32
sub	R	0	Reg.	Reg.	Reg.	0	34
lw	I	35	Reg.	Reg.	Offset		
sw	I	43	Reg.	Reg.	Offset		

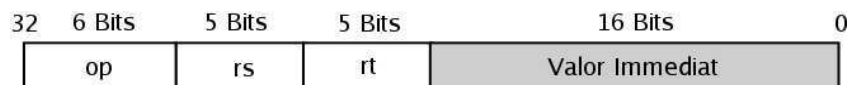
- Quina restricció produeix el **camp Offset**?
- Com es decideix el format d'instrucció?

1. ADREÇAMENT A REGISTRE



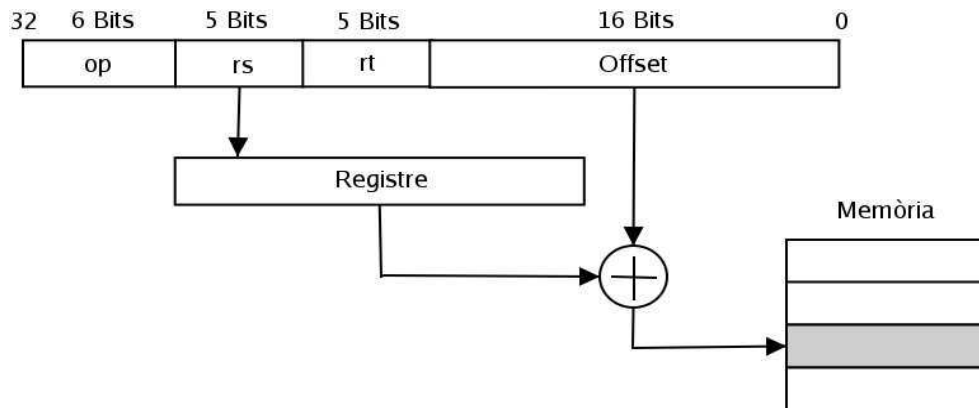
add \$8, \$17, \$18

2. ADREÇAMENT IMMEDIAT



addi \$8, \$17, 100

3. ADREÇAMENT BASE O DESPLAÇAMENT (INDEXAT)



lw \$8, 100(\$19)

ELS SALTS

- L'assemblador necessita poder realitzar **salts**
 - Trencar la seqüencialitat del programa
 - Implementar estructures de control de fluxe pels llenguatges d'alt nivell
 - Alternatives (*if-then-else*)
 - Alternatives de múltiples casos (*switch-case*)
 - Iteratives (*for, while-do, do-while, repeat-until*)
- Dos tipus bàsics de salts
 - Salts incondicionals (bifurcacions de programa)
 - Salts condicionals

SALTS CONDICIONALS(I)

- El MIPS no utilitza una paraula d'estat (**PSW**)
 - No té *flags* de *Zero*, de *Més Gran* ni de *Més Petit*
- Cada salt condicional opera amb registres
 - Saltar si el contingut del registre **\$8** és el mateix que el contingut del registre **\$9**:

beq \$8, \$9, 100

- Saltar si el contingut del registre **\$8** és diferent que el contingut del registre **\$9**:

bne \$8, \$9, 200

SALTS CONDICIONALS(II)

- Els salts condicionals del MIPS
 - Utilitzen adreçament immediat per especificar l'adreça destí
 - Les instruccions de salt condicional són del **format I**
 - El valor immediat és un *offset relatiu* al PC
 - Mode d'adreçament relatiu al PC
- La distancia del salt és limitada: 16 bits
 - Per poder saltar endavant i endarrera: **Offset** en **C₂**
- Totes les instruccions son fixes de 32 bits
 - L'**offset** s'especifica en paraules (**valor offset * 4**).

SALTS INCONDICIONALS

- Els salts incondicionals del MIPS
 - Poden utilitzar adreçament a registre (*format R*)

jr \$1

- Poden utilitzar adreçament immediat

j 100000

- Cal poder saltar més lluny del que es pot especificar utilitzant 16 bits en C_2
 - Nou format d'instrucció: *format J*

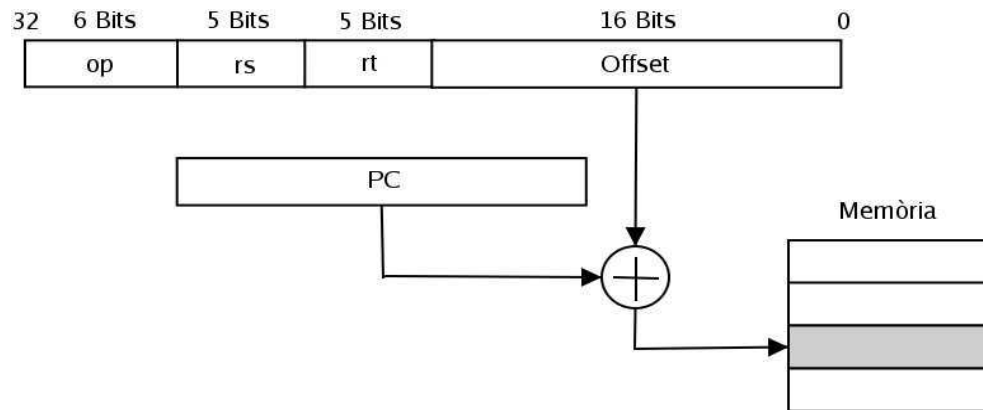
FORMAT J DE LES INSTRUCCIONS

j 100000



- **op** : Codi d'operació de la instrucció
- **rs** : Adreça destí absoluta de memòria

4. ADREÇAMENT RELATIU AL PC



beq \$8, \$9, 100

FORMATS D'INSTRUCCIÓ DE MIPS32 (II)

- RESUM DELS FORMATS DEL MIPS

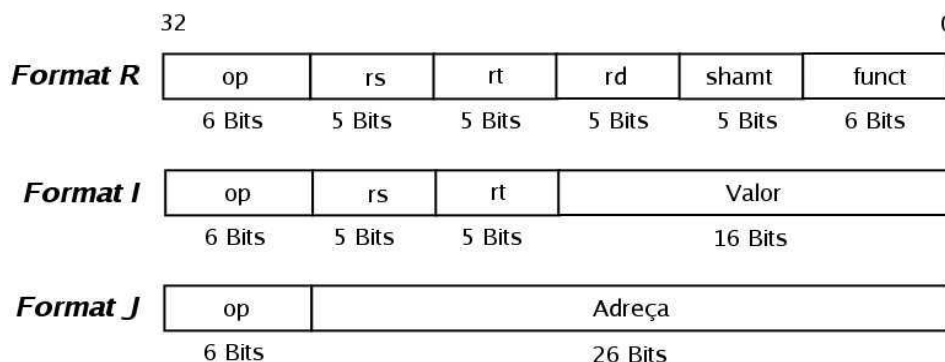
Instrucció	Format	6 Bits	5 Bits	5 Bits	5 Bits	5 Bits	6 Bits
	R	op	rs	rt	rd	shamt	funct
	I	op	rs	rt	Offset		
	J	op	Adreça absoluta				
add	R	0	Reg.	Reg.	Reg.	0	32
sub	R	0	Reg.	Reg.	Reg.	0	34
lw	I	35	Reg.	Reg.	Offset		
sw	I	43	Reg.	Reg.	Offset		
j	J	2	Adreça				



FORMATS D'INSTRUCCIÓ vs. MODES D'ADREÇAMENT (I)

		Modes d'Adreçament			
		Registre	Immediat	Base o desplaçament	Relatiu al PC
Formats d'Instrucció	R	<i>add</i> \$1, \$2, \$3 <i>sub</i> \$4, \$5, \$6	<i>sll</i> \$10, \$11, 8	-	-
	I	-	<i>addi</i> \$1, \$2, 100 <i>ori</i> \$1, \$2, 200	<i>lw</i> \$8, 100(\$9) <i>sw</i> \$4, 40(\$16)	<i>beq</i> \$25, \$26, 100 <i>bne</i> \$18, \$20, 80
	J	-	<i>j</i> 100000	-	-

El mode d'adreçament que indica cada columna de la taula es troba remarcant en *cursiva* a la instrucció de cada cel·la.



FORMATS D'INSTRUCCIÓ vs. MODES D'ADREÇAMENT (II)

- El *mode d'adreçament* fa referència a **com s'accedeix als operands** de la instrucció
- Els modes d'adreçament es poden combinar en una mateixa instrucció
 - Per exemple, adreçaments a **registre** i **immediats**

addi \$1, \$2, 100

- En MIPS les instruccions de desplaçament de bits (com *sll*) utilitzen el camp **shamt** del *format R* per especificar un *valor immediat de 5 bits*

ADREÇAMENTS INVÀLIDS EN MIPS

- Cal accedir a memòria de forma alineada
 - Una paraula (*lw, sw, ...*) a **adreces múltiples de 4**
 - Mitja paraula (*lh, sh, ...*) a **adreces parelles**
 - Un byte (*lb, sb, ...*) a **qualsevol adreça**
 - En els salts (*beq, bne, j, ...*) a **adreces múltiples de 4**
- Les instruccions permeten generar adreces invàlides

lw \$8, 3(\$0)

lh \$8, 1(\$0)

- Un adreçament invàlid provoca una excepció

PSEUDOINSTRUCCIONS

- Les *pseudoinstruccions* són instruccions que **no existeixen** en el llenguatge màquina de MIPS però l'assemblador les **interpreta i tradueix**

li \$8, Immediat_{15..0}

ori \$8, \$0, Immediat_{15..0}

la \$8, Immediat_{31..0}

lui \$8, Immediat_{31..16}

ori \$8, \$8, Immediat_{15..0}