



UNIVERSITAT
POLITÈCNICA
DE VALÈNCIA



Fonaments de computadors

Tema 6. LLENGUATGE D'ASSEMBLADOR

- Estudiar els conceptes fonamentals per a la programació en llenguatge d'assemblador del processador MIPS R2000.
- Conèixer l'estructura bàsica d'un processador, que permeti l'execució de les diferents instruccions, a partir de circuits digitals ja coneguts.
- Conèixer els elements d'un programa (dades i instruccions) en assemblador i comprendre la seua ubicació en la memòria.
- Conèixer una mostra significativa del repertori d'instruccions d'un processador actual.
- Entendre la codificació dels diferents tipus d'instruccions en llenguatge màquina del MIPS R2000.
- Analitzar i desenvolupar programes en llenguatge assemblador.

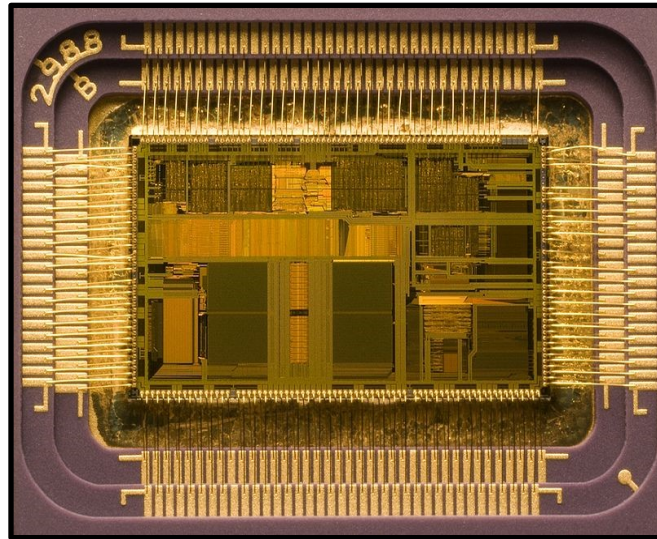
- Poliformat, secció “Recursos”
 - Exercicis sense solució.
 - Solucions als exercicis.
 - **Simulador PCSPIM** (per a instal·lar).
 - **Interfície Web per a l'SPIM** (no necessita instal·lació).
 - Exàmens d'anys anteriors.

- Introducció
- Unitats funcionals del computador
 - Definició
 - Unitat central de procés
- Joc d'instruccions
 - Arquitectura
- MIPS R2000
 - Organització de la memòria
 - Programes en ensamblador
 - Directives
 - Etiquetes
 - Estructura
 - Banc de registres
- Jocs d'instruccions
 - Introducció
 - Instruccions aritmètiques
 - Instruccions lògiques
 - Instruccions de càrrega i emmagatzematge
 - Pseudoinstruccions
 - Instruccions de comparació
 - Instruccions de control de flux
- Codificació d'instruccions
 - Tipus R
 - Tipus I
 - Tipus J
- Unitat de Control
 - Execució d'instruccions
- Connexió amb altres assignatures
 - Llenguatges informàtics
 - Sistemes Operatius
 - Processament avançat, xarxes

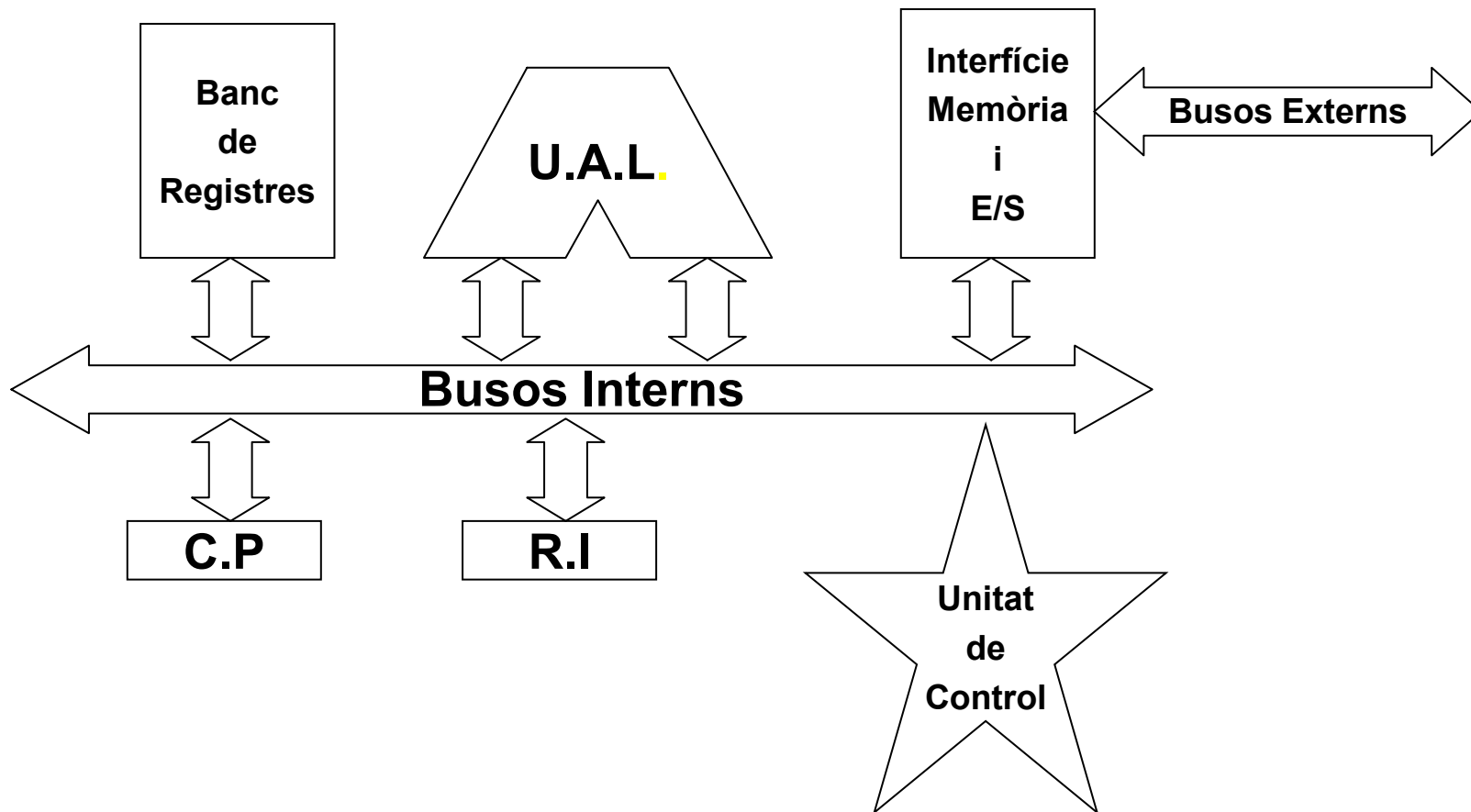
- MIPS Assembly Language Programming. Robert Briton
- See MIPS Run. Dominic Sweetman
- Introducción a los computadores. Julio Sahuquillo i altres

- Unitat funcional
 - Conjunt de circuits que realitzen una funció específica diferenciada d'unes altres funcions
- Unitats funcionals bàsiques del computador
 - Unitat Central de Procés (UCP, CPU)
 - Unitat de Memòria Principal
 - Unitat d'Entrada/Eixida (E/S, I/O)

- Unitat Central de Processament o CPU
 - És el component en un ordinador, que interpreta les instruccions i processa les dades contingudes en els programes de la computadora.
 - És a dir, executa instruccions



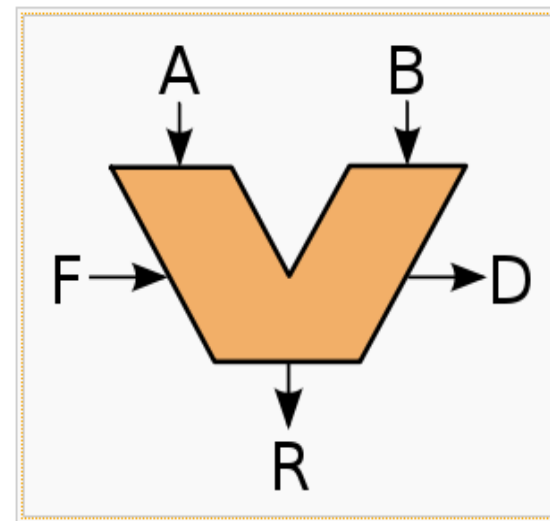
- Unitat Central de Processament (UCP)




- Unitat de control

- Genera els senyals necessaris per a que la unitat de processament (CPU) execute les instruccions de forma adequada:
 - Indica quina operació ha de fer la ALU.
 - Selecciona els registres on estan els operands i on es guardarà el resultat.
 - Genera el senyals d'escriptura quan els registres tenen que emmagatzemar informació.
 - Genera el senyal que controlen la memòria principal i els perifèrics.
 - Resumint, governa tots els circuits de la CPU i del computador.
- Es un sistema seqüencial, i la seua complexitat depèn de la complexitat de la CPU i de les instruccions que s'han d'executar

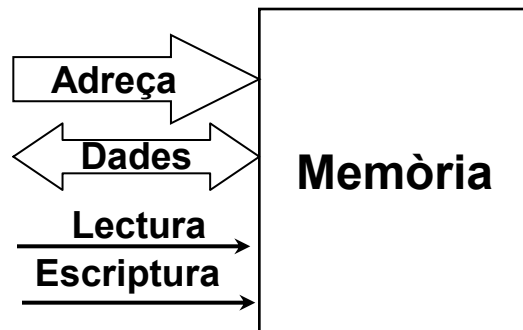
- La Unitat Aritmètic-Lògica (UAL), o Arithmetic Logic Unit (ALU), realitza operacions aritmètiques (com suma, resta, multiplicació, etc.) i operacions lògiques (com igual a, menor que, major que, etc.), entre dos nombres.
 - A y B → Operands
 - F → Operació a realitzar
 - R → Resultat de l'operació
 - D → Indicadors de resultat



- Banc de registres
 - S'usen per a emmagatzemar dades temporalment
 - Memòria amb un temps d'accés molt xicotet
 - Sempre es troba dins del processador
 - Nombre de registres entre 8 i 512, depèn del processador
 - MIPS R2000 32 registres de 32 bits (\$0 .. \$31)
 - 1 port d'escriptura i 1 o 2 ports de lectura
- Registres especials
 - Comptador de programa (C.P.) Conté l'adreça de memòria de la següent instrucció que s'ha d'executar.
 - Registre d'instrucció (R.I.) Conté el codi de la instrucció que s'està executant.

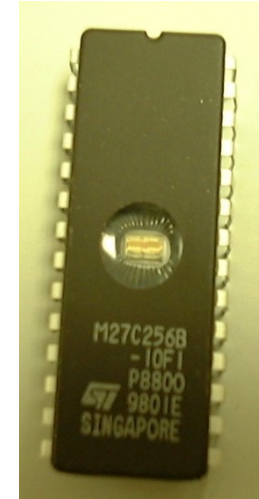
- Característiques del banc de registres del MIPS R2000
 - Banc de registres d'Enters:
 - 32 Registres de 32 bits de propòsit general
 - S'identifiquen com \$0 a \$31
 - Exemple: add \$2, \$3, \$4 \rightarrow ($\$2 \leftarrow \$3 + \4)
 - 2 Registres especials de 32 bits HI i LO.
 - Emmagatzemen els resultats de multiplicacions i divisions.
 - Banc de registres de Reals, que pot utilitzar-se com:
 -  32 Registres de 32 bits amb format IEEE 754 de simple precisió.
S'identifiquen per \$f0 a \$f31
 - 16 registres de 64 bits amb format IEEE 754 de doble precisió.
S'identifiquen per \$f0, \$f2, \$f4, ..., \$f30
- El registre \$0 té permanentment el valor 0

- Memòria: Dispositiu d'emmagatzematge (dades + instruccions)



Unitats de capacitat

- 1K (kilo) = $2^{10} = 1024$
- 1M (mega) = $2^{10}K = 2^{20}$
- 1G (giga) = $2^{10}M = 2^{20}K = 2^{30}$
- 1T (tera) = $2^{10}G = 2^{20}M = 2^{30}K = 2^{40}$
- 1P (peta) = $2^{10}T = 2^{20}G = 2^{30}M = 2^{40}K = 2^{50}$
- 1E (exa) = $2^{10}P = 2^{20}T = 2^{30}G = 2^{40}M = 2^{50}K = 2^{60}$

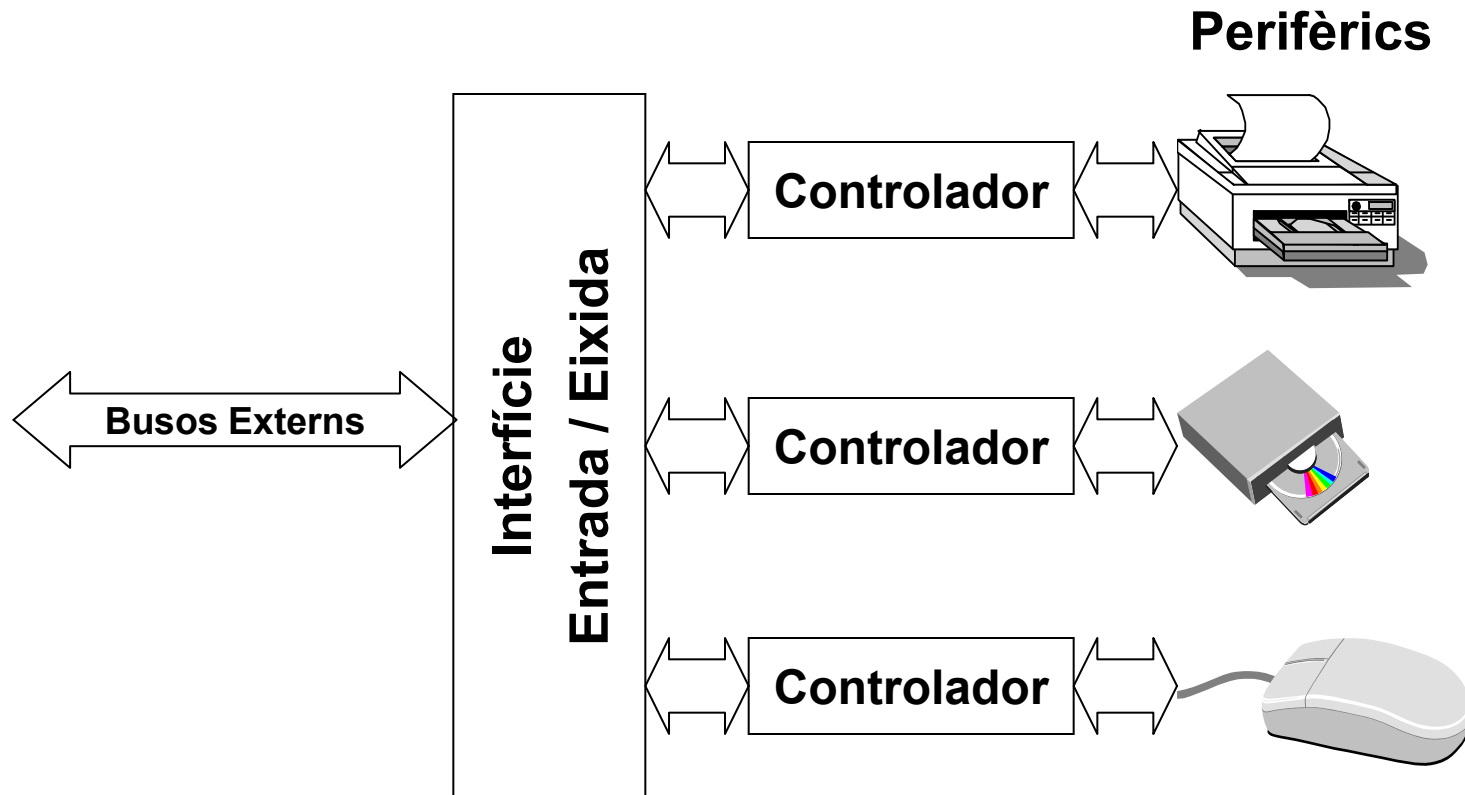


- Espai direccionable en el MIPS R2000
 - Bus d'adreces de 32 bits
 - 4 Gbytes (4G x 8bits) = 2 Ghalf = (2G x 16bits) = 1 Gword = (1G x 32bits).
 - 2^{32} bytes amb adreces des de 0 a $2^{32} - 1$
 - 2^{30} paraules (4 bytes) amb adreces: 0, 4, 8, ..., $2^{32} - 4$

La memòria accessible per l'usuari es troba en el rang [0x00400000, 0x7FFFFFFF]

Sistema Operatiu (Rutines d'interrupció)	0xFFFFFFFF 0x80000000
Memòria de dades	0x7FFFFFFF 0x10000000
Programa (Memòria d'instruccions)	0x0FFFFFFF 0x00400000
Reservat	0x003FFFFFFF 0x00000000

- Sistema d'Entrada/Eixida
 - Permeten la comunicació del sistema UCP-memòria amb l'exterior

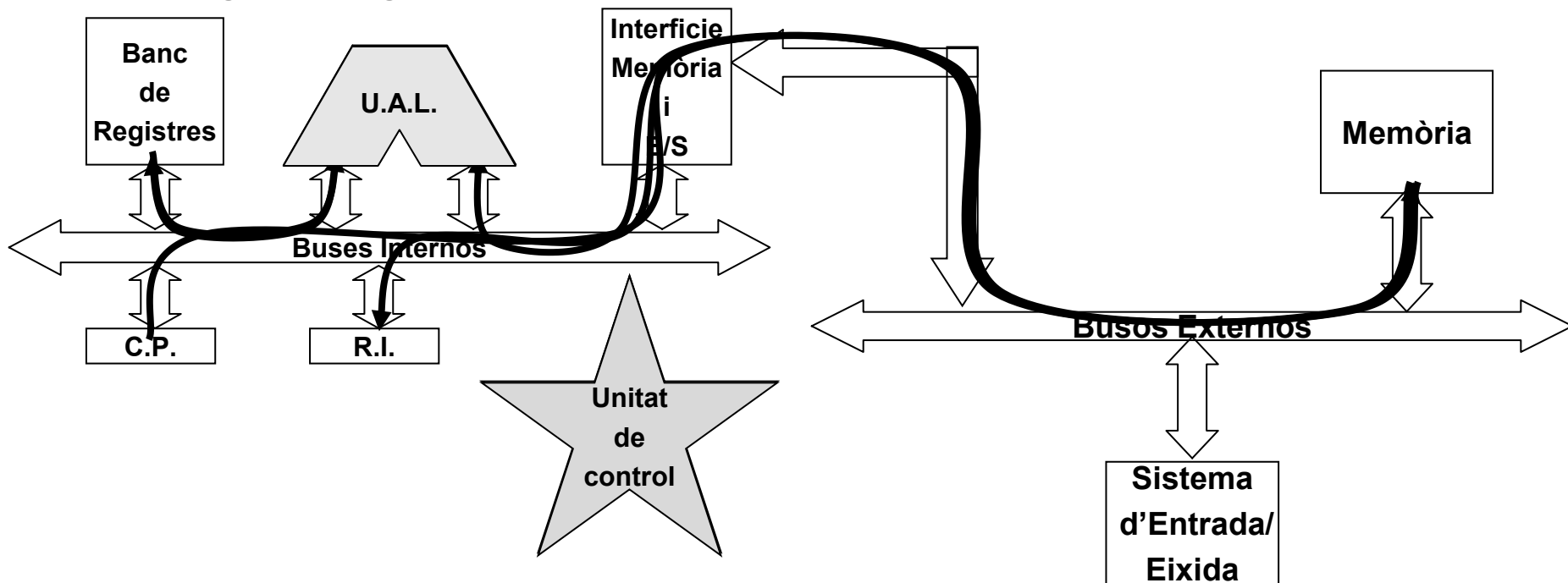


- Perifèric
 - Dispositiu que permet la comunicació amb l'exterior
 - Memòria secundària (emmagatzematge)
 - Entrada/eixida de dades (comunicació)
- Interfície o controlador
 - Dispositiu maquinari/programari que permet la comunicació entre la UCP (o el sistema de memòria) i el perifèric
 - Independitza la UCP del perifèric
 - Pot estar en el perifèric (exemple: discos IDE)

Etapes en l'execució d'una instrucció

FCO

- Cerca de la instrucció a executar
- Decodificació de la instrucció
- Cerca dels operands
- Realització de l'operació
- Emmagatzematge del resultat



- Arquitectura del Joc d'Instruccions
 - Visió de la màquina que té el programador en llenguatge d'assemblador
- L'arquitectura del Joc d'Instruccions està determinada per:
 - Tipus de dades
 - Espai lògic adreçable (organització de la memòria)
 - Conjunt de registres
 - Conjunt d'instruccions (Joc d'instruccions)
 - Modes d'adreçament

- Llenguatge d'ensamblador (Assembly language o asm):
 - És un llenguatge de programació de baix nivell. (prop de la màquina i lluny de tu)
 - Consisteix en un conjunt de mnemònics que representen instruccions que pot executar el processador.
 - Una instrucció d'ensamblador es correspon amb una instrucció màquina.
 - El programes d'ensamblador es tradueixen a codi màquina emprant un ensamblador.
 - El Llenguatge màquina o codi màquina és un conjunt d'instruccions amb les seues dades codificat en codi binari que pot entendre i executar un processador.
 - Tant el llenguatge d'ensamblador com el codi màquina són específics de cada model de processador.

- Estructura d'un programa escrit en ensamblador del MIPS R2000

Directiva que indica que
l'inici del programa està
en aquest codi

```
.globl __start  
.text 0x00400000  
__start:  
    addi $8, $0, 5  
    ...  
.end
```

Directiva que indica el
començament de
l'especificació
d'instruccions

Instruccions

Etiqueta que indica el
començament de
l'execució de les
instruccions

Directiva que indica el
final del codi

- Directives

- Les directives de l'assemblador serveixen per a situar les dades i les instruccions en la memòria del processador
Els seus noms comencen amb un punt i s'empren claudàtors per a indicar que un o diversos arguments són opcionals
 - `.nom_directiva argument1, [, argument2]`
- S'agrupen en tres tipus:
 - Directives per a reserva de posicions de memòria
 - Directives per a indicar l'inici de l'àrea de dades i d'instruccions
 - Directives de propòsit variat
- NO són instruccions, no ocupen memòria.

- Directiva “.text”
 - Sintaxi “.text adreça”
 - Situa les instruccions a partir de l'adreça especificada
 - Si no s'especifica una adreça, es pren 0x00400000
- Directiva “.end”
 - Sintaxi “.end”
 - Indica el final d'un programa

- Etiquetes
 - Sintaxi: “etiqueta:”
 - Indiquen una posició de memòria important, per exemple variables en el segment de dades o punts on saltar en el segment d'instruccions.
- “__start”
 - Etiqueta que indica l'inici del programa, és a dir, la primera instrucció que començarà a executar-se.
 - Només pot haver-hi una etiqueta “__start” en un programa, per a açò s'empra la directiva “. globl etiqueta” que indica que l'etiqueta definida és comuna a tots els arxius del programa.

- Instruccions
 - Llenguatge del Computador
 - El conjunt d'instruccions del MIPS R2000 que s'estudiarà és similar a l'utilitzat en altres architectures com Sony o Nintendo
- Cada instrucció en llenguatge d'assemblador té un format segons el qual es codificarà en llenguatge màquina (0's i 1's)
- Tipus d'instruccions:
 - Instruccions aritmètiques
 - Instruccions lògiques
 - Instruccions de càrrega i emmagatzematge
 - Instruccions de moviment
 - Instruccions de comparació
 - Instruccions de salt condicional
 - Instruccions de salt incondicional

- Instruccions aritmètiques

Sintaxi	Format	Descripció
add rd, rs, rt	R	$rd \leftarrow rs + rt$
addi rt, rs, inm	I	$rt \leftarrow rs + \text{inm}$ (inmediato de 16 bits representat en Ca2)
sub rd, rs, rt	R	$rd \leftarrow rs - rt$
mult rs, rt	R	Multiplica rs per rt deixant els 32 bits de menor pes en el registre LO i els 32 bits de major pes en el registre HI
div rs, rt	R	Divideix rs entre rt deixant el quocient en el registre LO i el residu en el registre HI

- Instruccions de moviment de dades

Sintaxi	Format	Descripció
mfhi rd	R	$rd \leftarrow HI$
mflo rd	R	$rd \leftarrow LO$
mthi rs	R	$HI \leftarrow rs$
mtlo rs	R	$LO \leftarrow rs$

Suma i resta

```
.globl __start
.text 0x00400000
__start:
    addi $8, $0, 5
    addi $9, $0, 6
    add $10, $8, $9
    sub $11, $8, $9
.end
```

Multiplicació

```
.globl __start
.text 0x00400000
__start:
    addi $8, $0, 2
    addi $9, $0, 3
    mult $8, $9
    mflo $10
    mfhi $11
.end
```

Divisió

```
.globl __start
.text 0x00400000
__start:
    addi $8, $0, 4
    addi $9, $0, 11
    div $9, $8
    mflo $10
    mfhi $11
.end
```

- Quin és el contingut dels registres \$10 i \$11 en finalitzar l'execució de cada codi?
- Quin és el contingut dels registres \$10 i \$11 si se substitueixen les següents instruccions?

Suma i resta `sub $11, $8, $9` per `sub $11, $9, $8`

Divisió `div $9, $8` per `div $8, $9`

- Instrucciones lògiques

Sintaxi	Format	Descripció
and rd, rs, rt	R	$rd \leftarrow rs \text{ and } rt$, l'operació lògica indicada es realitza bit a bit
nor rd, rs, rt	R	$rd \leftarrow rs \text{ nor } rt$
xor rd, rs, rt	R	$rd \leftarrow rs \text{ xor } rt$
or rd, rs, rt	R	$rd \leftarrow rs \text{ or } rt$
andi rt, rs, inm	I	$rt \leftarrow rs \text{ and } \text{inm}$, (la dada immediata, és de 16 bits i s'estén amb 16 zeros)
ori rt, rs, inm	I	$rt \leftarrow rs \text{ or } \text{inm}$
xori rt, rs, inm	I	$rt \leftarrow rs \text{ xor } \text{inm}$
sll rd, rt, desp	R	$rd \leftarrow rt \ll \text{desp}$, desplaçament a esquerres, conforme desplaça s'emplena amb 0
srl rd, rt, desp	R	$rd \leftarrow rt \gg \text{desp}$, desplaçament a dretes, conforme desplaça s'emplena amb 0

or i and

```
.globl __start
.text 0x00400000
__start:
    ori $8, $0, 0x000a
    ori $9, $0, 0x000c
    or $10, $8, $9
    and $11, $8, $9
.end
```

nor i xor

```
.globl __start
.text 0x00400000
__start:
    ori $8, $0, 0x000a
    ori $9, $0, 0x000c
    nor $10, $8, $9
    xor $11, $8, $9
.end
```

- Realitzeu les operacions bit a bit
- Quin és el contingut dels registres \$10 i \$11 en finalitzar l'execució de cada codi?
- De quina forma pot realitzar-se l'operació NOT?
- De quina forma pot realitzar-se l'operació NAND?

A esquerres

```
.globl __start
.text 0x00400000
__start:
    addi $8, $0, 6
    sll $9, $8, 1
    sll $10, $8, 2
.end
```

A dretes

```
.globl __start
.text 0x00400000
__start:
    addi $8, $0, 6
    srl $9, $8, 1
    srl $10, $8, 2
.end
```

- Quin és el contingut dels registres \$9 i \$10 en finalitzar l'execució de cada codi?
- Quina operació aritmètica està implementant el desplaçament a esquerres?
- Quina operació aritmètica està implementant el desplaçament a dretes?

- Tipus de dades suportades per el MIPS R2000

Tipo de datos	Representación	Tamaño	Nombre
Caracteres	Ascii	8 bits	Ascii
Enteros	Ca2	8 bits	Byte
		16 bits	Half
		32 bits	Word
Reales	IEEE 754	32 bits	Float
		64 bits	Double

- Float : Nombres reals representats segons l'estàndard IEEE 754 de simple precisió
- Double: Nombres reals representats segons l'estàndard IEEE 754 de doble precisió

- Memòria. Pot veure's com un vector unidimensional.

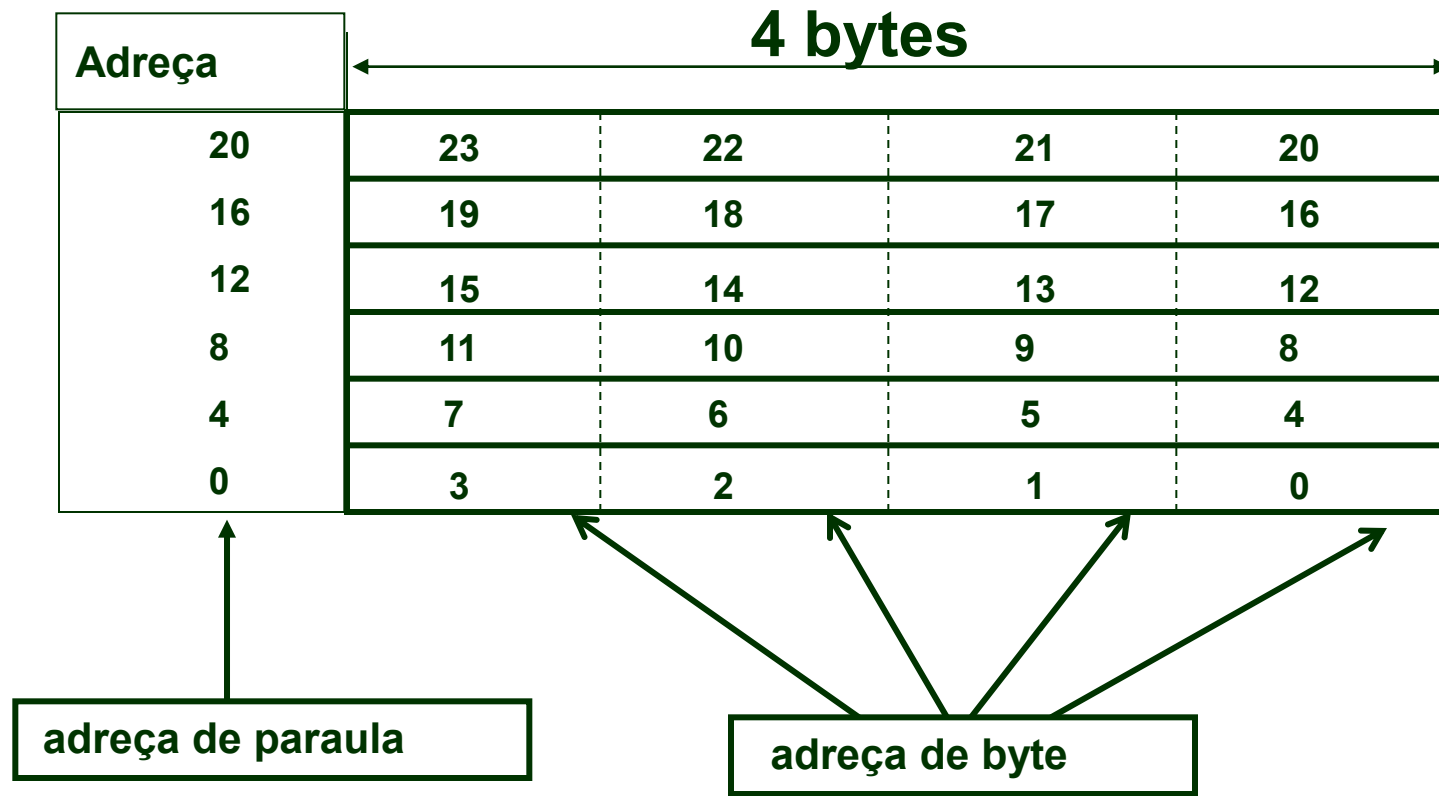
- Adreça de memòria. Índex dins del vector.
- Memòria adreçada byte a byte.
 - Els índexs apunten a cada byte de la memòria. El byte és la mínima unitat adreçable

Adreça	Contingut
3	1 byte de dates
2	1 byte de dates
1	1 byte de dates
0	1 byte de dates

- En el MIPS R2000 la memòria s'organitza en paraules de 4 bytes.

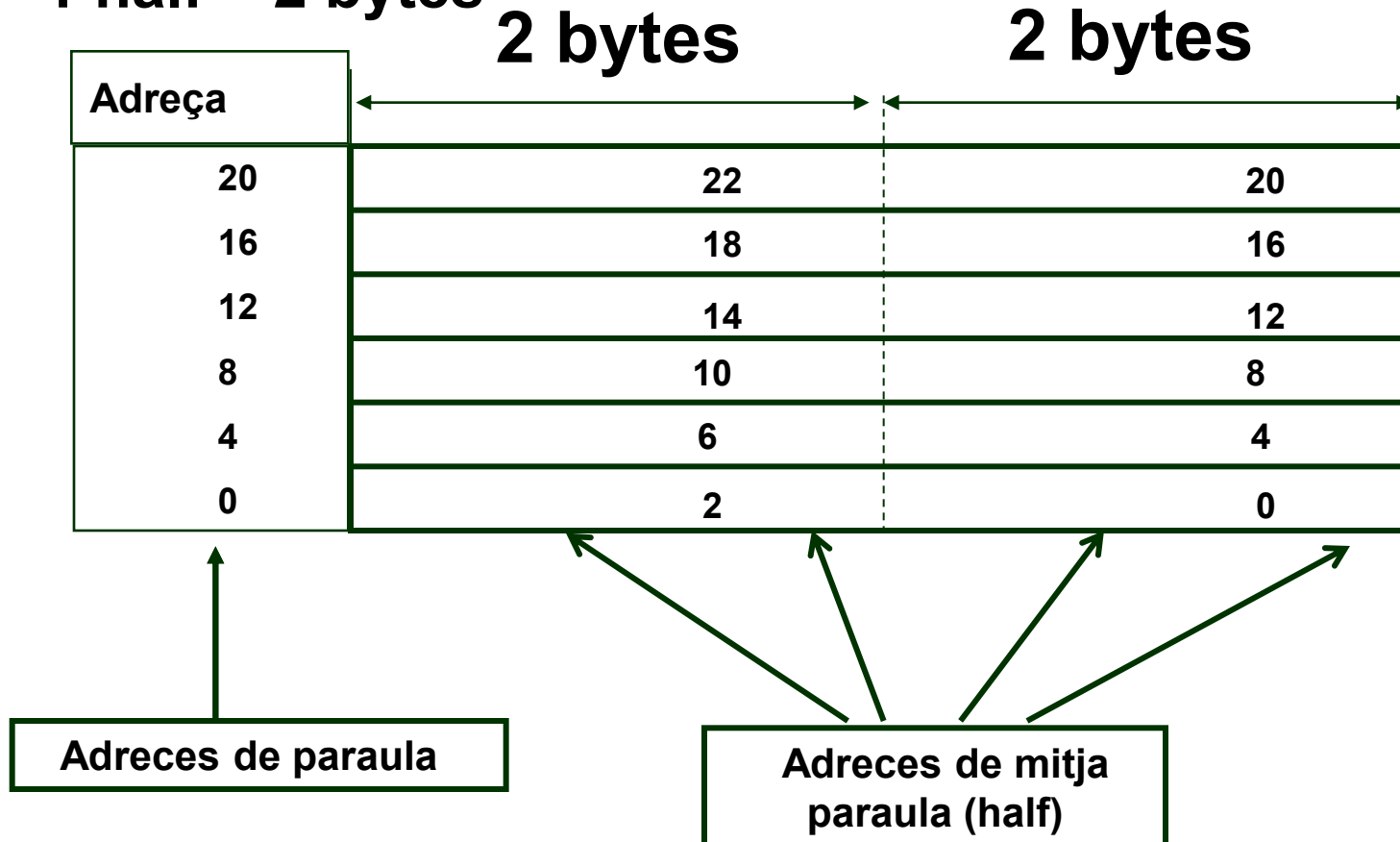
- Cada paraula (word) ocupa 4 posicions (bytes) o adreces de memòria
- És possible accedir a la memòria de les següents formes:
 - Per Byte → Qualsevol adreça
 - Per Word → Adreces múltiples de 4 (0, 4, 8, ...)
 - Per mitja paraula o Half → Adreces múltiples de 2 (0, 2, 4, ...)

- **Accés per paraula (word)**
 - 1 word = 4 bytes



- **Accés per mitja paraula (half)**

- 1 half = 2 bytes



- **Existeixen 2 formes d'emmagatzemar la informació en memòria:**
 - **Format big endian**
 - El byte de major pes de la dada o instrucció s'emmagatzema en l'adreça de memòria més baixa.
 - **Format little endian**
 - El byte de menor pes de la dada o instrucció s'emmagatzema en l'adreça de memòria més baixa.

- Exemple 1. Emmagatzemar en memòria les dues paraules següents utilitzant tots dos formats:

1ª paraula:

0XA1	0XB2	0X33	0X22
31	24 23	16 15	8 7 0

2ª paraula:

0X55	0X55	0XC1	0X00
31	24 23	16 15	8 7 0

Adreces de paraula

Little Endian

0X55	0X55	0XC1	0X00	4
0XA1	0XB2	0X33	0X22	0
31	24 23	16 15	8 7	0

Big Endian

0X00	0XC1	0X55	0X55	4
0X22	0X33	0XB2	0XA1	0
31	24 23	16 15	8 7	0

- Directiva “.data”
 - Sintaxi: “.data adreça”
 - Situa les dades que s'especifiquen en les directives de dades a partir de l'adreça especificada
 - Si no s'especifica una adreça, es pren 0x10000000

- Directiva “.space”
 - Sintaxi: .space n
 - Reserva n bytes de memòria i els inicialitza a 0x00
- Directives “.ascii” i “.asciiz”
 - Sintaxi: .ascii ‘cadena1’ , [‘cadena2’, ... ‘cadena n’]
 .asciiz ‘cadena1’ , [‘cadena2’, ... ‘cadena n’]
 - S'empra per a emmagatzemar cadenes de caràcters codificats en ASCII en posicions consecutives de memòria .
 - La directiva “.asciiz” afegeix un caràcter NULL (0x00) al final de la cadena.

- Directiva “.byte”
 - Sintaxi: “.byte b1, [b2, b3, ...]”
 - Inicialitza posicions consecutives de memòria amb els enters d'1 byte codificats en Ca2
- Directiva “.half”
 - Sintaxi: “.half h1, [h2, h3, ...]”
 - Inicialitza posicions consecutives de memòria amb els enters de 2 bytes codificats en Ca2
- Directiva “.word”
 - Sintaxi: “.word w1, [w2, w3, ...]”
 - Inicialitza posicions consecutives de memòria amb els enters de 4 bytes codificats en Ca2
- En tots els casos es poden especificar les dades en decimal o hexadecimal.

- Directiva “.float”
 - Sintaxi: “.float f1, [f2, f3, ...]”
 - Inicialitza posicions consecutives de memòria amb els reals de 4 bytes codificats en *IEEE 754 de simple precisió
- Directiva “.double”
 - Sintaxi: “.double d1, [d2, d3, ...]”
 - Inicialitza posicions consecutives de memòria amb els reals de 8 bytes codificats en *IEEE 754 de doble precisió
- En tots dos casos es poden especificar les dades en notació decimal o en notació científica.

- Alineació de dades en memòria
 - .space, .byte, .ascii i .asciiz reserven memòria a partir de qualsevol adreça
 - .half reserva memòria a partir d'adreces múltiples de 2 .
 - word i .float reserven memòria a partir d'adreces múltiples de 4
 - .double reserva memòria a partir d'adreces múltiples de 8
- El simulador del MIPS emmagatzema les dades en format Little Endian

Exemple d'alineació de dades:

.data 0x10000020

.word 0x4A038D01

.space 1

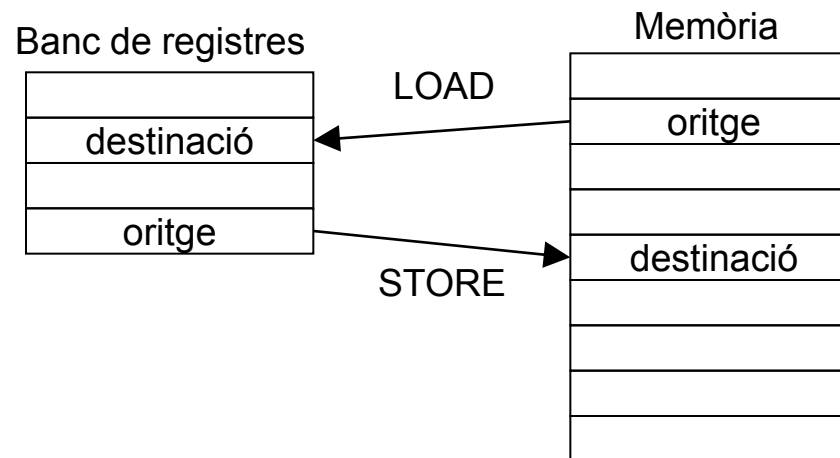
.word 0x10

.byte 0x1, 0x2, 0x3

.double 1.0 #0x3FF0000000000000 IEEE doble precisió

Contingut en hexadecimal				Adreça mem.
31 24	23 16	15 8	7 0	
0x4A	0x03	0x8D	0x01	0x10000020
			0x00	0x10000024
0x00	0x00	0x00	0x10	0x10000028
	0x03	0x02	0x01	0x1000002C
0x00	0x00	0x00	0x00	0x10000030
0x3F	0xF0	0x00	0x00	0x10000034

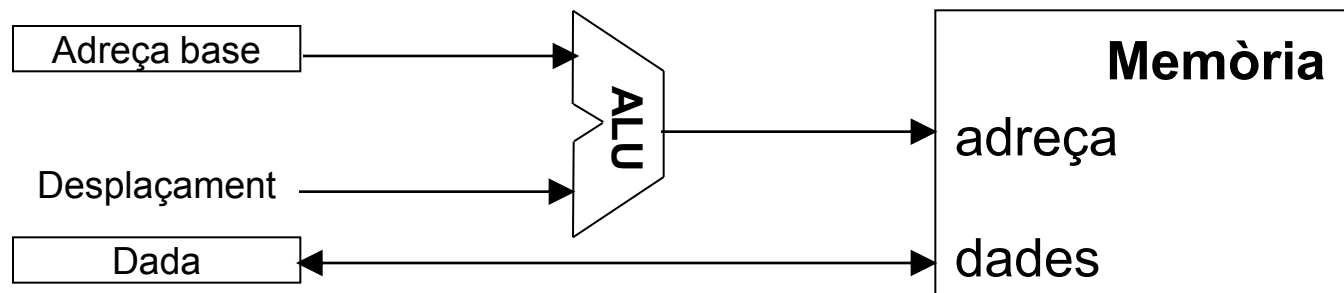
- Instruccions de càrrega i emmagatzematge
 - Són les úniques que accedeixen a la memòria
 - Load: llig el contingut que indica una adreça de memòria i ho emmagatzema en un registre del banc de registres.
 - Store: llig el contingut d'un registre del banc de registres i ho emmagatzema en la memòria, en l'adreça indicada .




- En la descripció de les instruccions d'accés a memòria s'utilitzarà la següent nomenclatura:
 - rd, rs i rt representen registres de propòsit general
 - $rt_{31..16}$: Part alta del registre rt
 - $rt_{15..0}$: Part baixa del registre rt
 - $desp+rs$, dóna una adreça de memòria, resultat de sumar al contingut del registre rs una quantitat anomenada desplaçament
 - $M[desp+rs]$ és el contingut de la posició de memòria $desp+rs$

dades	adreces		
0x00000032	0x10000010	rs = 0x10000004 desp = 8	} Desp+rs = 0x1000000C M[desp+rs] = 0x7654
0x00007654	0x1000000C		
0x00000543	0x10000008		
0x00000000	0x10000004		
0x00000100	0x10000000		


- Instruccions de càrrega i emmagatzematge
 - Càlcul de l'adreça, tant per a les instruccions de càrrega com les d'emmagatzematge:
 - Adreça base: registre amb l'adreça de memòria base.
 - Desplaçament: enter de 16 bits amb el desplaçament. Va inclòs en la instrucció.
 - Adreça efectiva de memòria = Adreça base + Desplaçament
 - Dada: registre amb les dades per a carregar (load) des de la memòria (estenenent el signe) o emmagatzemar (store) a la memòria



- Instruccions de càrrega i emmagatzematge



Sintaxi	Format	Descripció
lw rt, desp(rs)	I	$rt \leftarrow M[\text{desp}+rs]$,
lh rt, desp(rs)	I	$rt \leftarrow M[\text{desp}+rs]$, càrrega mitjana paraula (16 bits) i estén el signe
lb rt, desp(rs)	I	$rt \leftarrow M[\text{desp}+rs]$, carrega 1 byte (8 bits) i estén el signe
sw rt, desp(rs)	I	$M[\text{desp}+rs] \leftarrow rt$
sh rt, desp(rs)	I	$M[\text{desp}+rs] \leftarrow rt$ Emmagatzema la part baixa de rt en memòria
sb rt, desp(rs)	I	$M[\text{desp}+rs] \leftarrow rt$ Emmagatzema el byte menys significatiu en memòria
lui rt, inm	I	$rt_{31..16} \leftarrow \text{inm}$ $rt_{15..0} \leftarrow 0$



Lectura de memòria

```
.globl __start
.data 0x10000000
var_a: .byte 2
var_b: .half -4
var_c: .word 6

.text 0x00400000
__start:
    lui $8, 0x1000
    lb $9, 0($8)
    lh $10, 2($8)
    lw $11, 4($8)
.end
```

- Quin és el contingut de la memòria abans d'executar el codi?
- En quines adreces de memòria estan emmagatzemades les variables “var_a”, “var_b” i “var_c”?
- Quin és el contingut dels registres \$8, \$9, \$10 i \$11 en finalitzar l'execució del codi?

- Què s'hauria de modificar (canviar o ampliar) en el codi si se substituïra la directiva de dades `.data 0x10000000` per `.data 0x10000010` ?

Esriptura en memòria

```
.globl __start
.data 0x10000000
var_a: .space 1
var_b: .space 2
var_c: .space 4

.text 0x00400000
__start:
    addi $8, $0, -1
    addi $9, $0, 2
    addi $10, $0, 4
    lui $11, 0x1000
    sb $8, 0($11)
    sh $9, 2($11)
    sw $10, 4($11)
.end
```

- Quin és el contingut de la memòria abans de llançar el codi?
- Quin és el contingut de la memòria després de llançar el codi?
- Quin és el contingut dels registres \$8, \$9, \$10 i \$11 en finalitzar l'execució del codi?

- Definició:
 - Conjunt d'ordres, d'aparença similar a les instruccions, que el llenguatge d'assemblador incorpora a més de les instruccions màquina del processador.
 - Una pseudoinstrucció es tradueix en instruccions: d'1 a 4 instruccions màquina.
 - El programador pot usar les pseudoinstruccions com si es tractara d'instruccions màquina suportades pel processador.
 - El programa assemblador és l'encarregat de substituir aquestes pseudoinstruccions per les instruccions màquina corresponents.
 - El SPIM reserva el registre \$1, per a emmagatzematge auxiliar en cas de necessitar més d'una instrucció
- Exemple:
 - Carregar l'adreça de memòria d'una etiqueta en un registre
 - Inicialitzar una dada en un registre
 - Cobrir totes les possibles comparacions o salts condicionals

- li: Load Immediate:
 - Sintaxi: li R, inm
 - Descripció: Carrega en el registre indicat per R l'enter amb signe en complement a 2 de 32 bits inm
 - Traducció: pot traduir-se en lui, lui i ori, o ori, depenent de la dada que s'ha de carregar en el registre.
 - Exemples
 - li \$8, 1 → ori \$8, \$0, 1
 - li \$9, 0x10000001 → lui \$1, 0x1000; ori \$9, \$1, 0x0001
 - li \$10, 0x10000000 → lui \$10, 0x1000

- la: Load Address:
 - Sintaxi: la R, etiqueta
 - Descripció: Carrega en el registre R l'adreça de memòria a la qual fa referència etiqueta
 - Traducció: pot traduir-se en lui, lui i ori, depenent de l'adreça de l'etiqueta
 - Exemples
 - Davant aquesta definició de dades →

```
.data 0x10000000
var_a: .word 1
var_b: .word 2
```
 - la \$8, var_a → lui \$8, 0x1000
 - la \$9, var_b → lui \$1, 0x1000; ori \$9, \$1, 4

- Instruccions de comparació

Sintaxi	Format	Descripció
slt rd, rs, rt	R	Si $rs < rt$ llavors $rd \leftarrow 1$, si no $rd \leftarrow 0$
slti rt, rs, inm	I	Si $rs < inm$ llavors $rt \leftarrow 1$, si no $rt \leftarrow 0$

Comparació immediata

```
.globl __start
.text 0x00400000
__start:
    addi $8, $0, 1
    addi $9, $0, 3
    slti $10, $8, 2
    slti $11, $9, 2
.end
```

Comparació

```
.globl __start
.text 0x00400000
__start:
    addi $8, $0, 1
    addi $9, $0, 3
    slt $10, $8, $9
    slt $11, $9, $8
.end
```

- Quin és el contingut dels registres \$10 i \$11 en finalitzar l'execució de cada codi?

- Instruccions de salt condicional

Sintaxi	Format	Descripció
beq rs, rt, etiqueta	I	Si $rs = rt$ llavors salta a l'adreça etiqueta $PC \leftarrow \text{etiqueta}$
bne rs, rt, etiqueta	I	Si $rs \neq rt$ llavors salta a l'adreça etiqueta $PC \leftarrow \text{etiqueta}$

- Instruccions de salt incondicional

Sintaxi	Format	Descripció
j etiqueta	J	$PC \leftarrow \text{etiqueta}$, salta a l'adreça etiqueta
jal etiqueta	J	Salta a l'adreça etiqueta guardant-se prèviament l'adreça de tornada en \$31 $\$31 \leftarrow PC$ actualitzat $PC \leftarrow \text{etiqueta}$
jr rs	R	$PC \leftarrow rs$, salta a l'adreça continguda en el registre rs

Salt d'igualtat

```
.globl __start
.text 0x00400000
__start:
    addi $8, $0, 1
    addi $9, $0, -1
    beq $8,$0,fin
    addi $9, $0, 1
fin:
.end
```

Salt de desigualtat

```
.globl __start
.text 0x00400000
__start:
    addi $8, $0, 1
    addi $9, $0, -1
    bne $8,$0,fin
    addi $9, $0, 1
fin:
.end
```

- Quina és l'adreça de memòria a la qual apunta l'etiqueta “fin”?
- Quin és el contingut del registre \$9 en finalitzar l'execució dels programes anteriors?

Salt incondicional

```
.globl __start
.text 0x00400000
__start:
    addi $8, $0, 3
    addi $9, $0, 1
bucle:
    beq $8, $0, fin
    mult $9, $8
    mflo $9
    addi $8, $8, -1
    j bucle
fin:
.end
```

- Quina és l'adreça de memòria a la que fa referència l'etiqueta “bucle”?
- Quin és el contingut dels registres \$8 i \$9 en finalitzar l'execució del programa?
- El programa exemple realitza alguna operació coneguda? quina?
- En l'operació que implementa el programa exemple, què representen \$8 i \$9?

- El format de les instruccions indica la forma en què es codifiquen aquestes instruccions en codi màquina.
- Totes les instruccions del *MIPS R2000 tenen una grandària de 32 bits.
- En el MIPS R2000 es distingeixen tres tipus de formats en funció dels components del processador que utilitzen les instruccions:
 - Tipo R o de tipus registre.
 - Tipo I o de tipus immediat.
 - Tipo J o de salt incondicional.
- Tots els tipus tenen en comú els primers sis bits, formant un camp conegut com CO (Codi d'Operació)

Tipus R

Codi d'operació de la instrucció, imposat pel dissenyador, indica el tipus d'instrucció

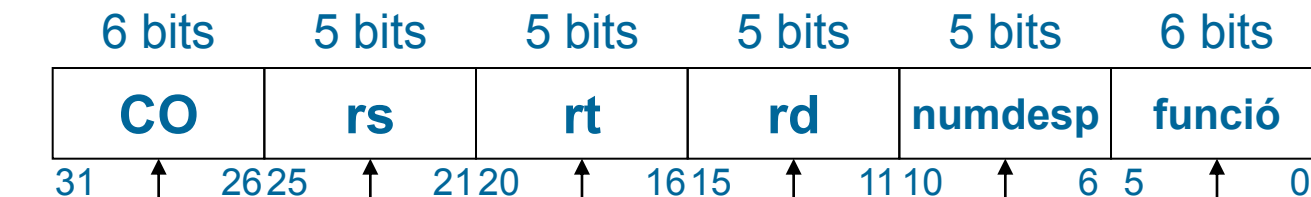
Codificació binària del primer registre font

Codificació binària del segon registre font

Codificació binària del registre destí

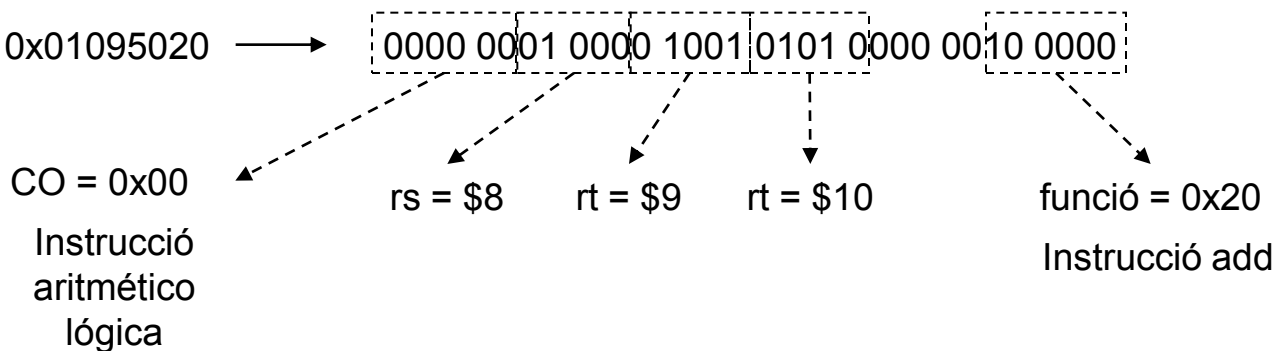
Codificació binària del nombre de desplaçaments (instruccions srl, sll, i sra)

Indica el tipus d'operació aritmètica o lògica.

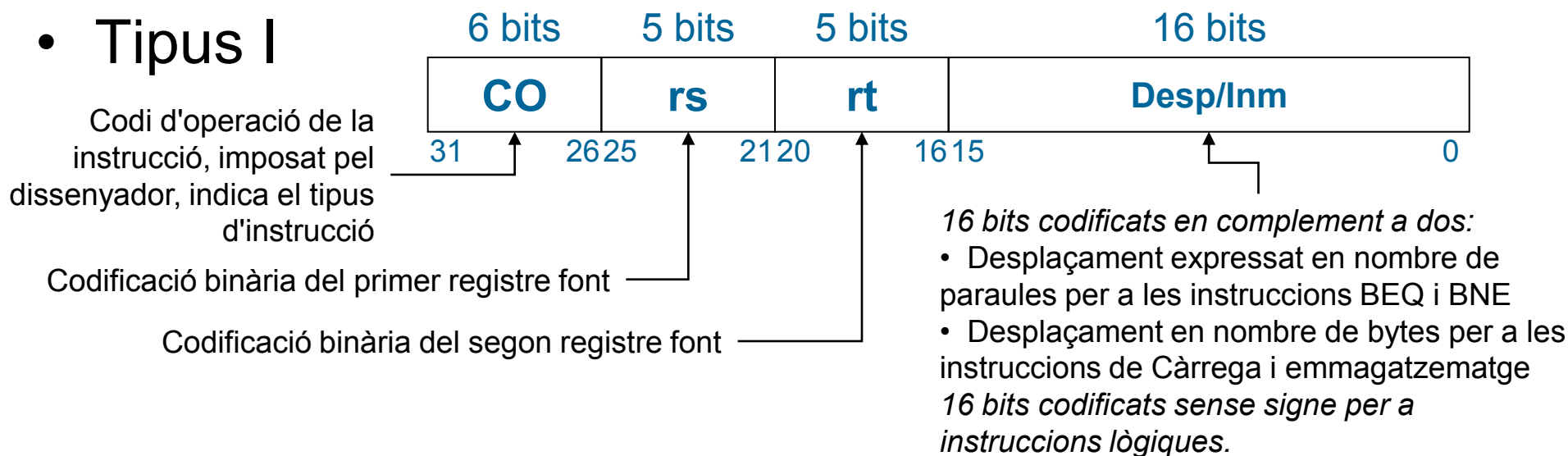


Exemple

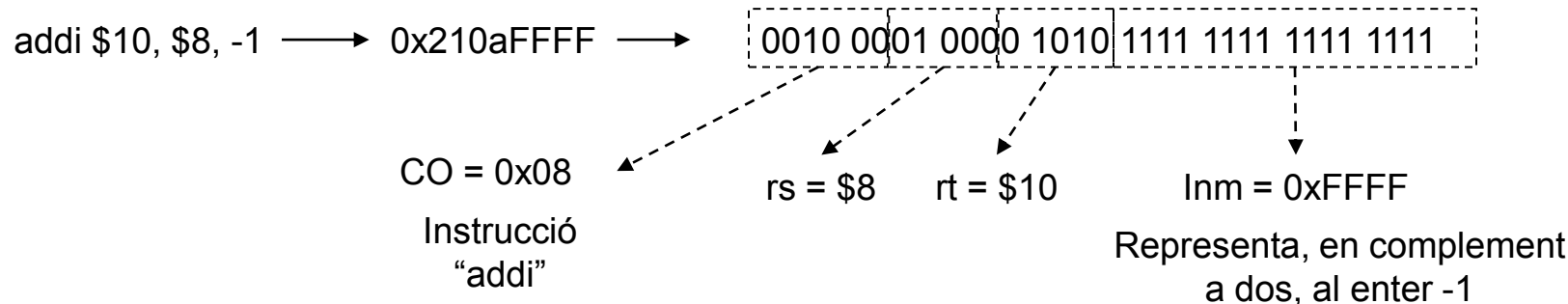
add \$10, \$8, \$9 → 0x01095020 →



• Tipus I



• Exemple amb una dada immediata



- Tipus I: Exemple amb un desplaçament condicional

```
.text 0x0040000
```

```
_start:
```

```
lw $4, 8($10)
```

```
bne $4, $5, distinto
```

```
sub $4, $4, $5
```

```
j salida
```

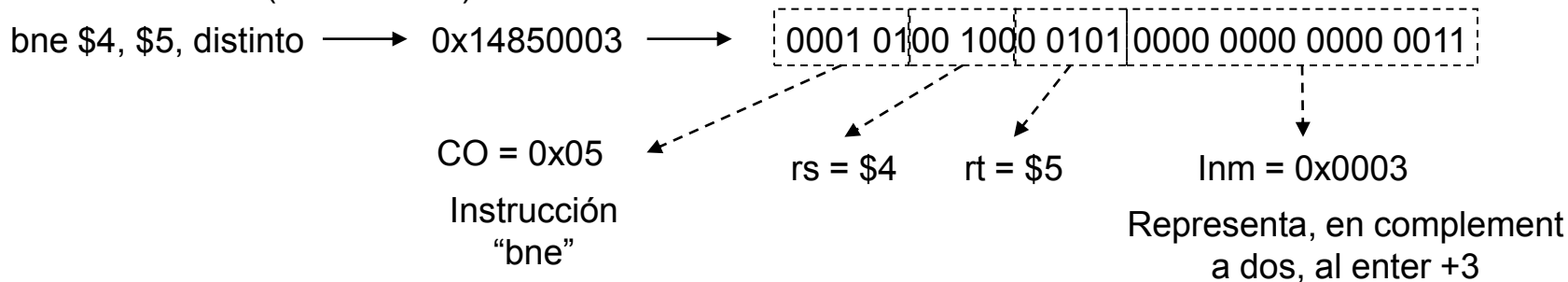
```
distinto: add $4, $4, $5
```

```
salida: addi $5, $5, 1
```

El comptador de programa (CP) apunta a la instrucció que s'està executant ("bne")

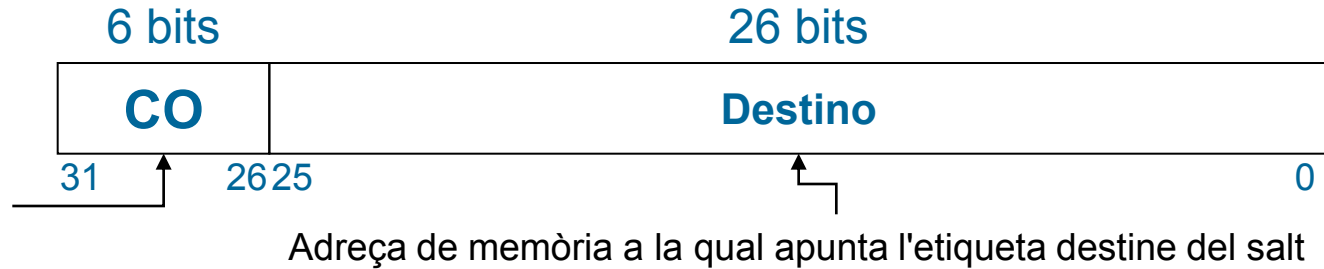
L'etiqueta "diferent" apunta a la instrucció "add" que es troba a dues instruccions del CP

Cada instrucció es codifica en 32 bits, per la qual cosa ocupa una paraula en memòria. Per tant, les adreces de memòria on es troben les instruccions sempre seran múltiples de quatre: 0, 4, 8, 12, 16, etc. El camp desplaçament per a la instrucció condicional ("bne" en aquest cas) expressa el salt en nombre de paraules de memòria (instruccions) a saltar

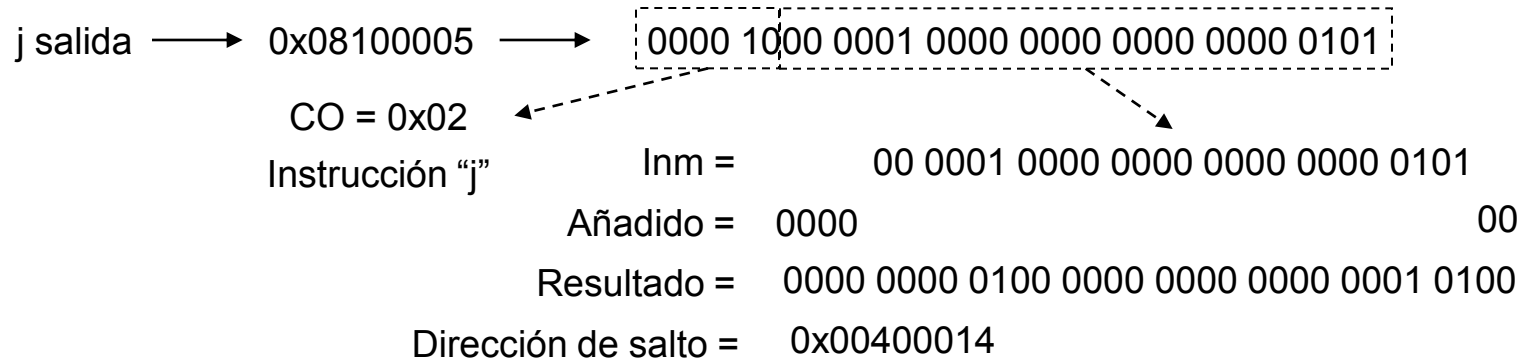


- Tipus j

Codi d'operació de la instrucció, imposat pel dissenyador, indica el tipus d'instrucció



- Exemple (del codi de la transparència anterior)



Només 26 bits dels 32 del Comptador de Programa (PC) són modificats per j romanent els altres 6 sense canvis.

L'adreça de salt (32 bits) ha de codificar-se en el camp destine de 26 bits, per la qual cosa es prescindeix dels dos bits de menor pes i dels 4 bits de major pes per ser sempre zeros

- Un llenguatge de programació és un idioma artificial dissenyat per a expressar computacions que poden ser dutes a terme per màquines com les computadores.
 - Depenent de la màquina
 - Independent de la màquina

LLenguatge	Codi
Llenguatge Java	a=b+c;
Llenguatge d'assemblador MIPS	add \$3,\$2,\$9
Llenguatge màquina del MIPS	0000 0000 0100 1001 0001 1000 0010 0000

- Software (programari): és el conjunt dels programes de còmput, procediments, regles, documentació i dades associades que formen part de les operacions d'un sistema de computació.
- Tipus de Software IEEE 729
 - Software de Sistema
 - Software de Programació
 - Software de Aplicació
- El software adopta diverses formes en diferents moments del seu cicle de vida
 - Codi Font
 - Codi objecte
 - Codi Executable

- Què és un Sistema Operatiu?
 - Un programa que actua com a intermediari entre l'usuari d'un computador i el maquinari del mateix
- Objectius del sistema operatiu:
 - Executar programes i facilitar la solució dels problemes de l'usuari
 - Fer un ús convenient del computador
- Usar el computador de forma eficient
- Proporcionar una màquina virtual estesa

- Per a computadors de taula:
 - Linux
 - Windows XP, Vista, 7, 8, 10
 - Mac OS X
- Per a Smartphones (antigament PDAS)
 - Linux,
 - Symbian
 - Windows Phone
 - Palm OS
 - Android
 - iOS
- Per a sistemes empotrats:
 - FreeRTOS
 - Microcos
 - eCos

- És el mecanisme usat per una aplicació per a sol·licitar un servei al sistema operatiu.
- Interfície entre aplicacions i SO.
 - Generalment disponibles com a funcions en assembler.
- Serveis típics del sistema operatiu.
 - Gestió de processos.
 - Gestió de processos lleugers.
 - Gestió de senyals, temporitzadors.
 - Gestió de memòria.
 - Gestió de fitxers i directoris.
- Exemples de cridades.
 - read: permet llegir dades d'un fitxer.
 - fork: permet crear un nou procés.

- Per a augmentar la potència i l'eficiència en el processament es tenen diferents models de processament més avançat.
- L'optimització es fonamenta en diversos conceptes
 - Paralelització de processos.
 - Utilització de recursos compartits.
 - Especialització de components.
- Existeixen diversos models
 - Multiprocessadors.
 - Supercomputadors.
 - Xarxes de Computadors.
 - Sistemes Distribuïts

- Multiprocessadors
 - Computador amb dos o més processadors.
 - Permeten el processament en paral·lel processant simultàniament diversos “fils” pertanyents a un mateix procés o bé de processos diferents.
 - Generen certes dificultats de coordinació especialment en l'accés a memòria.
 - Precisen que el Sistema Operatiu estiga preparat per a poder obtenir tota la potència que ofereixen.
- Supercomputadors
 - Sistemes amb múltiples computadors connectats directament per mitjà de connexions dedicades.
 - Proporcionen una potència de còmput molt més elevada que els computadors comuns.

- Xarxes de Computadors
 - Conjunt de computadors connectats per mitjans de comunicació (cables, ones, etc.) que els permet compartir recursos (unitats d'emmagatzematge, impressores, etc.)
 - Existeixen diversos models d'implementació: ISO/OSI, TCP/IP, ATM, etc.
- Sistemes Distribuïts
 - Sistemes basats en xarxes de computadors on els computadors es comuniquen a través de serveis (model client-servidor).
 - Un servidor també pot ser client i viceversa
 - Han de ser confiables: si un component falla, un altre component deu ser capaç de reemplaçar-ho (tolerància a fallades).

- Poliformat, secció “Recursos”
 - Exercicis sense solució.
 - Solucions als exercicis.
 - **Simulador PCSPIM** (per a instal·lar).
 - **Interfície Web per a l'SPIM** (no necessita instal·lació).
 - Exàmens d'anys anteriors.



UNIVERSITAT
POLITÈCNICA
DE VALÈNCIA



etsinf



Fonaments de computadors

Tema 6. LLENGUATGE D'ASSEMBLADOR