

Processadors actuals

MIPS

MIPS

Versió	nº de bits	Processadors	Comentaris
MIPS 1	32	R2000, R3000	Versió comercial del processador MIPS de la universitat d'Stanford
MIPS II	32	R600	Base estàndard del MIPS32
MIPS III	64	R4000	Primera arquitectura MIPS de 64 bits
MIPS IV	64	R5000, R10000	Actualització del MIPS III
MIPS V	64	R12000, R16000	Base de l'estàndard MIPS64
Arquitectures actuals			
MIPS32	Basada en MIPS II Inclou instruccions del MIPS III, IV i V per augmentar l'eficiència del codi i la transferència de dades		
MIPS64	Basat en el MIPS V És compatible amb el MIPS32		

Aplicacions

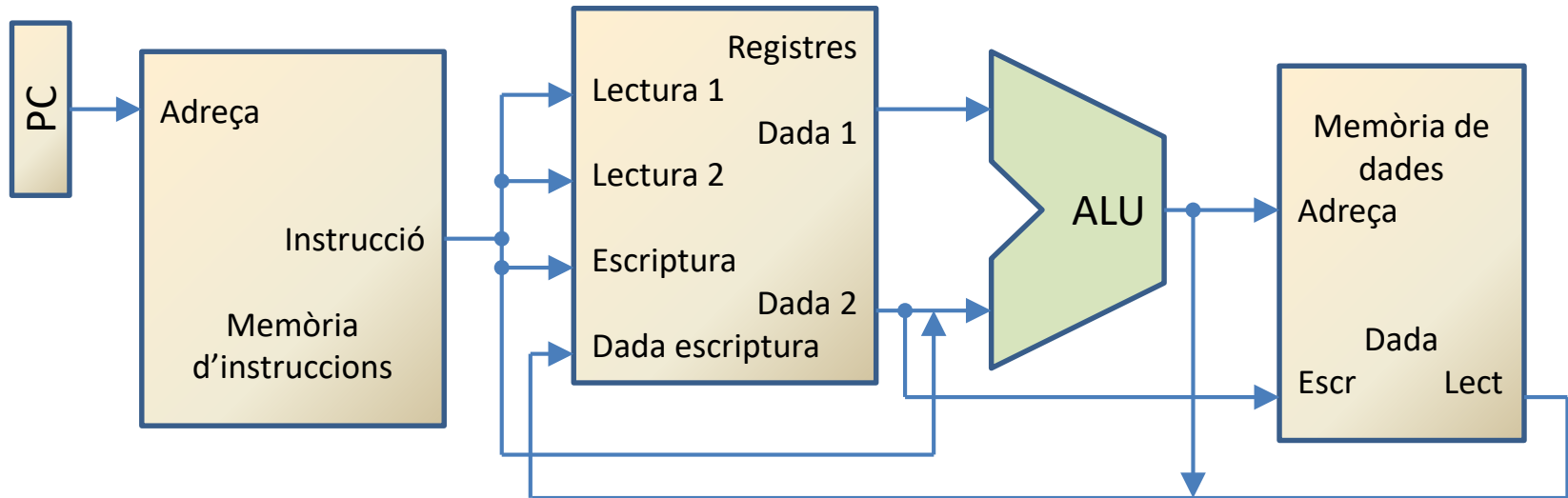
Fabricants d'estacions de treball: SGI, MIPS Computer Systems, Inc., Olivetti, Siemens-Nixdorf, Acer, Digital Equipment Corporation, NEC i DeskStation.

Diversos sistemes operatius van ser portats a l'arquitectura, exemples d'això són el SGI IRIX, Microsoft Windows NT (fins el Windows NT 4.0) i Windows CE, Linux, BSD, UNIX System V, SINIX, MIPS Computer Systems RISC/us. ...

MIPS R2000/R3000

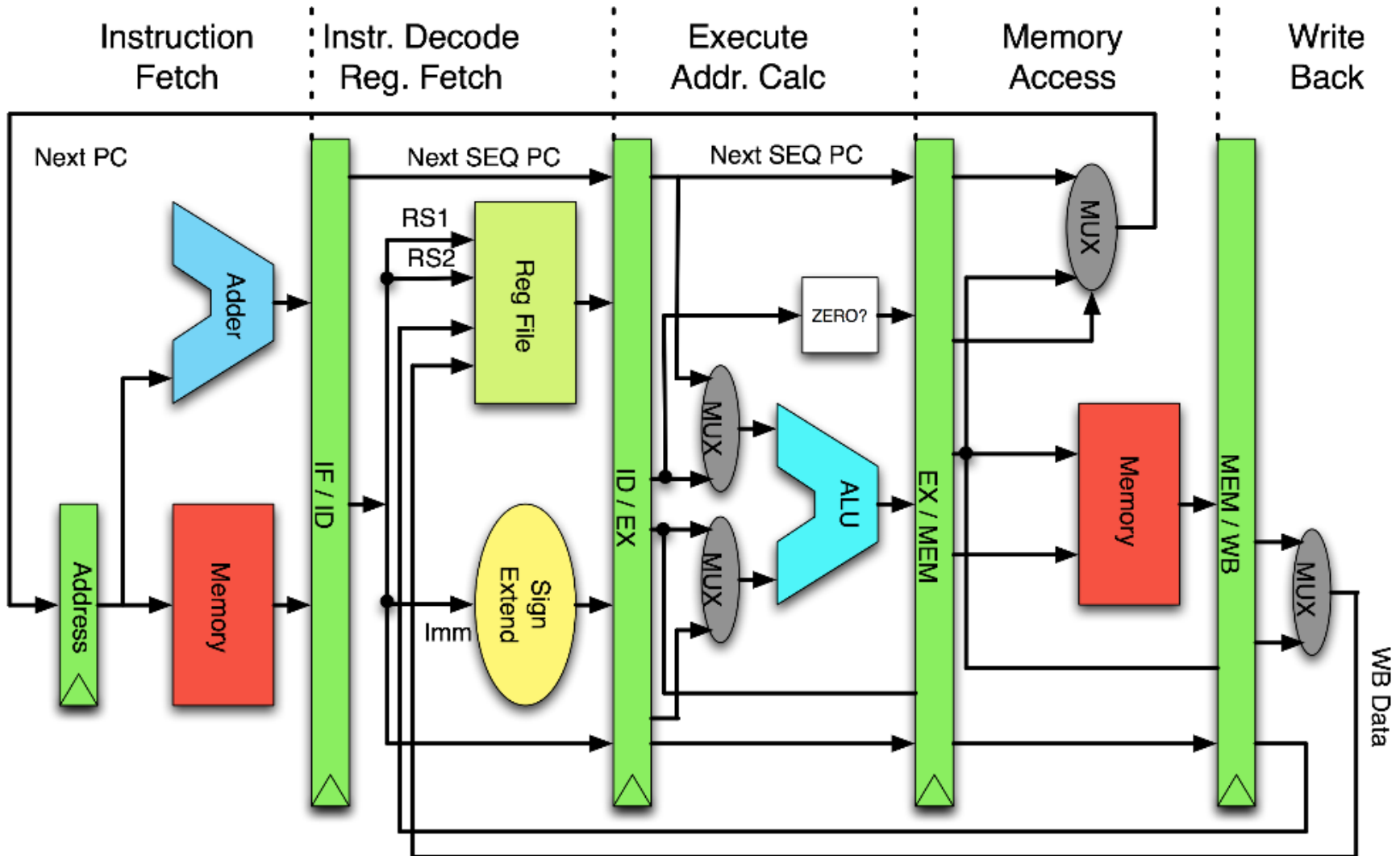
Esquema d'alt nivell del MIPS

- 5 nivells segmentació
- 32 registres de propòsit general i 3 d'especials

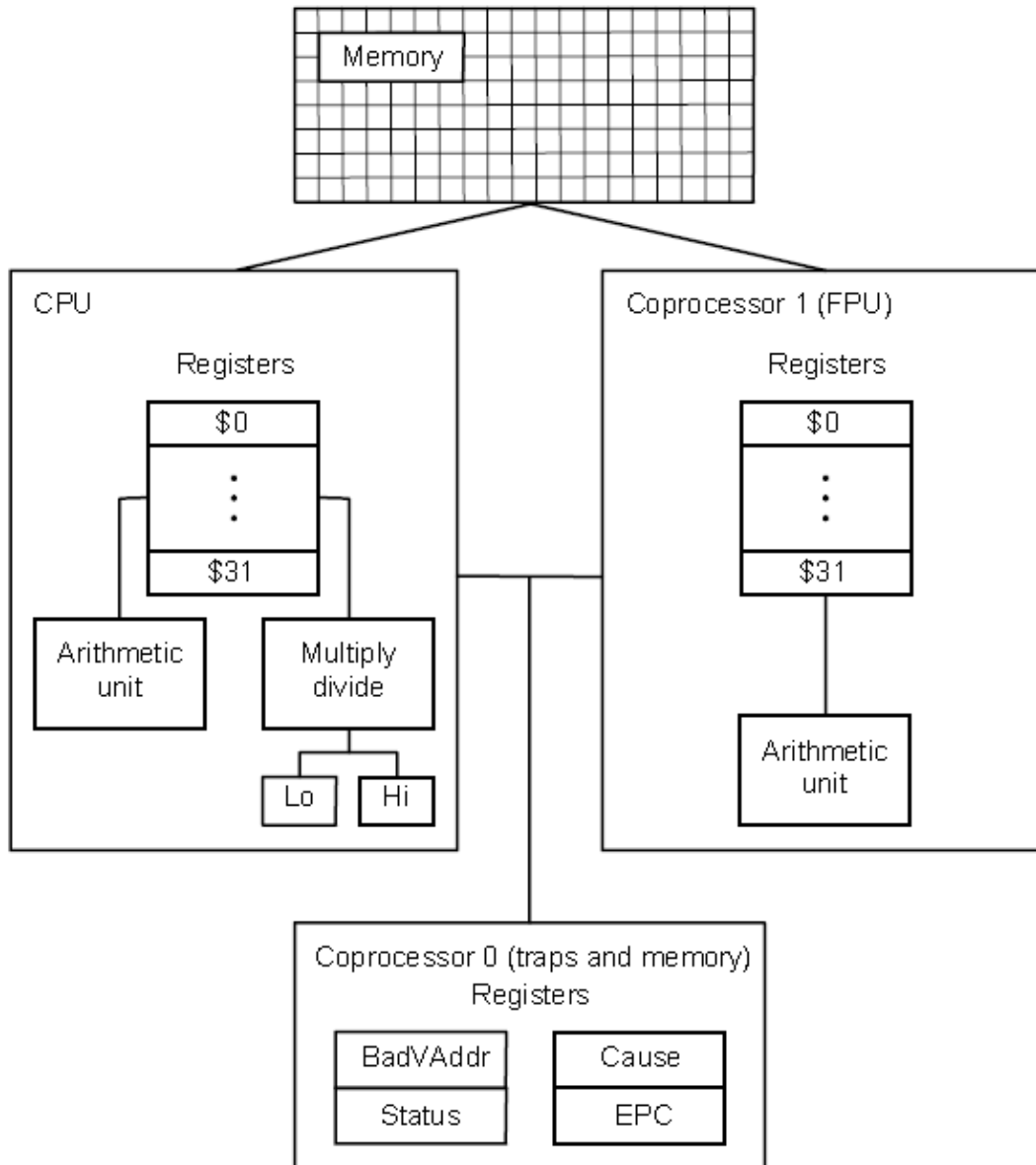


MIPS R2000/R3000

Segmentació:



Organització MIPS



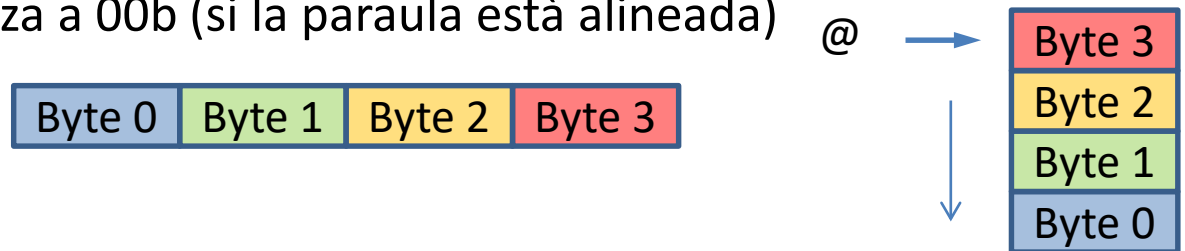
Processador MIPS

- Organització
 - Unitat aritmètica i lògica (ALU).
 - Unitat aritmètica sencera, operacions de multiplicació i divisió.
 - Unitat de coma flotant (FPU).
 - Coprocessador dedicat a la memòria cau i virtual

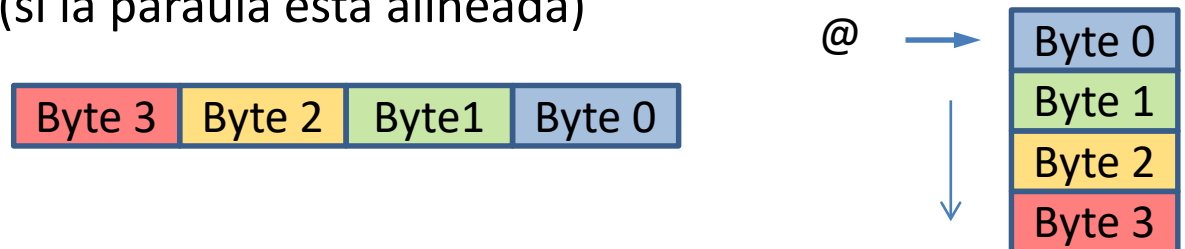
Processador MIPS

- Memòria

- Es denomina paraula (word) al contingut d'una cel·la de memòria.
- Utilitza paraules de 32 bits. Les adreces de memòria es corresponen a dades de 8 bits (byte). En una paraula hi ha quatre bytes.
- Per accedir a una paraula es llegeixen 4 bytes.
- Existeixen dues formes per numerar els bytes dins la paraula:
 - **Big endian** (IBM, Motorola, MIPS): L'adreça del byte **més** significatiu (el gran) finalitza a 00b (si la paraula està alineada)



- **Little endian** (Intel, Dec): L'adreça del byte **menys** significatiu (el petit) finalitza a 00b (si la paraula està alineada)



Processador MIPS

- Tipus de dades:



– Bit: 0,1

– Cadenes de bits:

- 4 bits nibble
- 8 bits byte
- 16 bits half-word
- 32 bits word
- 64 bits double-word

- Noms dels tipus de dades:

- Caràcter
 - ❖ ASCII Codi de 7 bits per símbol
- Decimal (BCD)
 - ❖ Dígits de 0 a 9 codificats de 0000 a 1001
- Sencers
 - ❖ Sense signe i amb signe en complement a dos
- Reals
 - ❖ Precisió simple i doble

Característiques

- Pot moure bytes, mitges paraules i paraules dels registres a memòria i al revés.
- Pot processar números sencers binaris de 32 bits, amb i sense signe
- Té capacitat de processar números binaris reals o de coma flotant en simple i doble precisió.
- **No disposa d'operacions a nivell de bit**

MIPS

Registres especials:

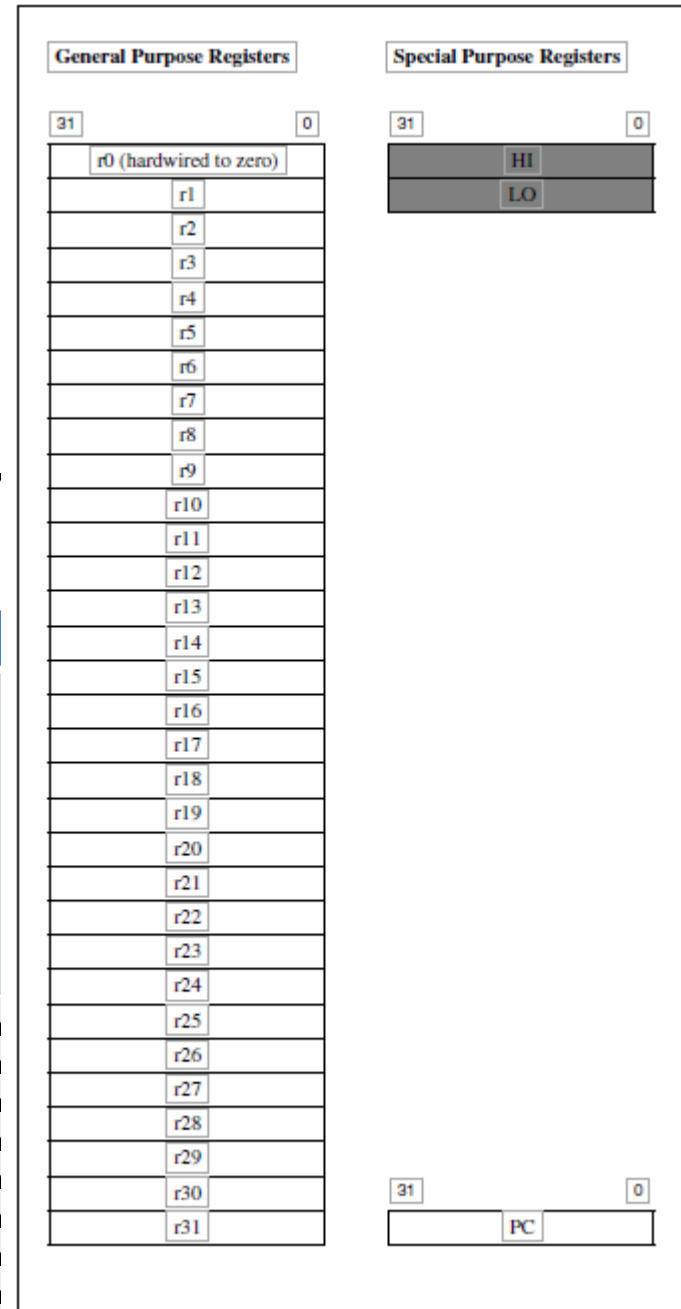
3 registres de 32 bits:

- PC: Comptador de programa.
- HI: Part alta del resultat d'una multiplicació o divisió.
- LO: Part baixa del resultat d'una multiplicació o divisió.

Registres de propòsit general:

32 registres de 32 bits accessibles directament des de l' seu nom de registre o el seu número de registre.

Número	Nom	Significat	Utilització/ Comentaris
\$0	\$z0	Constant zero	Registre de lectura que conté el valor
\$1	\$at	Temporal	Reservat per l'assemblador
\$2	\$v0	Valor 0	Retorn de les funcions
\$3	\$v1	Valor 1	Valor alt en operacions de 64 bits
\$4	\$a0	Argument 0	Primer paràmetre de les funcions
\$5	\$a1	Argument 1	Segon paràmetre de les funcions
\$6	\$a2	Argument 2	Tercer paràmetre de les funcions
\$7	\$a3	Argument 3	Quart paràmetre de les funcions
\$8	\$t0	Temporal 0	Registre temporal 0 (no es guarda en
\$9	\$t1	Temporal 1	Registre temporal 1 (no es guarda en
\$10	\$t2	Temporal 2	Registre temporal 2 (no es guarda en
\$11	\$t3	Temporal 3	Registre temporal 3 (no es guarda en
\$12	\$t4	Temporal 4	Registre temporal 4 (no es guarda en
\$13	\$t5	Temporal 5	Registre temporal 5 (no es guarda en
\$14	\$t6	Temporal 6	Registre temporal 6 (no es guarda en
\$15	\$t7	Temporal 7	Registre temporal 7 (no es guarda en



Registres del processador

Número	Nom	Significat	Utilització/ Comentaris
\$16	\$s0	Temporal Mem. 0	Registre temporal 0 (es guarda en les crides a les funcions)
\$17	\$s1	Temporal Mem. 1	Registre temporal 1 (es guarda en les crides a les funcions)
\$18	\$s2	Temporal Mem. 2	Registre temporal 2 (es guarda en les crides a les funcions)
\$19	\$s3	Temporal Mem. 3	Registre temporal 3 (es guarda en les crides a les funcions)
\$20	\$s4	Temporal Mem. 4	Registre temporal 4 (es guarda en les crides a les funcions)
\$21	\$s5	Temporal Mem. 5	Registre temporal 5 (es guarda en les crides a les funcions)
\$22	\$s6	Temporal Mem. 6	Registre temporal 6 (es guarda en les crides a les funcions)
\$23	\$s7	Temporal Mem. 7	Registre temporal 7 (es guarda en les crides a les funcions)
\$24	\$t8	Temporal 8	Registre temporal 8 (no es guarda en les crides a les funcions)
\$25	\$t9	Temporal 9	Registre temporal 9 (no es guarda en les crides a les funcions)
\$26	\$k0	Nucli 0	Reservat pel nucli del S.O.
\$27	\$k1	Nucli 1	Reservat pel nucli del S.O.
\$28	\$gp	Punter variables globals	Punter a la zona de dades globals
\$29	\$sp	Punter de pila	Punter de pila
\$30	\$fp	Punter trama	Punter a la base de la pila
\$31	\$ra	Adreça de retorn	Guarda l'adreça de retorn en les crides a les funcions

Registres de la unitat de coma flotant

- 32 registres de 32 bits
- En les instruccions de simple precisió, es poden utilitzar tots els registres: \$0, \$1, ..., \$31.
- En les instruccions de doble precisió s'utilitzen per parelles de registres parell/senar: \$0, \$2, \$4,, \$30.

Coprocessadors

- Poden haver-hi fins a quatre coprocessadors.
 - De 32 registres cadascun, accessibles mitjançant instruccions específiques.
- Coprocessador 0
 - Coprocessador de control de sistema
 - Incorporat en el xip de la CPU.
 - Controla el subsistema de la memòria cau.
 - Suporta el sistema de la memòria virtual i tradueix les adreces virtuals en físiques.
 - Suporta la gestió de les excepcions
 - Controla els canvis del mode d'execució (usuari, nucli i supervisor).
 - Proporciona el control de diagnòstic i la recuperació per les fallades.
- Coprocessador 1
 - Reservat per la unitat de coma flotant.
- Coprocessador 2
 - Reservat per implementacions específiques.
- Coprocessador 3
 - Reservat per la unitat de coma flotant en el MIPS 64.

Format de les dades de la FPU

Punt flotant de simple precisió
(32 bits) (.fmt tipus S)

Punt flotant de doble precisió
(64 bits) (.fmt tipus D)

Punt fix de precisió simple (32
bits) (.fmt tipus W)

MIPS

- Cicle d'execució: Seqüència repetitiva dels passos:
 1. Cerca de la instrucció (fetch): Obté des de la memòria la instrucció que s'ha d'executar.
 2. Descodificació de la instrucció: Determina el tipus d'instrucció i la seva llargada en cas de que estigui formada per més d'una paraula. S'ha d'especificar com estaran codificats els operants, l'operació i l'adreça de la propera instrucció.
 3. Cerca d'operants: Localitza i aconsegueix les dades que es corresponen als operants de la instrucció.
 4. Execució: Realitza l'operació amb els operants d'entrada per generar un resultat i determinar el seu estat.
 5. Emmagatzemar el resultat: Guarda el resultat a la memòria de dades o als registres. S'especifica en el codi d'operació.

Instruccions (assemblador)

- Te un format de 3 operants, preservant l'ordre natural: $a = b + c \rightarrow \text{add } a, b, c$
- Les instruccions de transferència de dades (load i store) només tenen dos operants:
 - *lw Registre, Adreça de memòria*
 - *sw Registre, Adreça de memòria*
- Arquitectura Load/Store. Totes les operacions aritmètiques i lògiques es realitzen entre registres. Per operar les variables de la memòria primer s'han de carregar als registres (**load**) i per emmagatzemar el resultat s'ha de fer amb les instruccions de guardar a memòria (**store**).
- Accés a memòria. La paraula és de 32 bits i pot adreçar un byte. Les instruccions de transferència de dades que accedeixen a:
 - Mitges paraules (de 16 bits) com lh, sh, ..., només poden adreçar posicions de memòria parelles.
 - A bytes (8 bytes) com lb, sb, ..., poden adreçar qualsevol posició de memòria.

Totes les instruccions ocupen 32 bits -> les instruccions de salt (bne, beq, j, ..) només poden accedir a adreces múltiples de 4. Les instruccions tenen la possibilitat de codificar adreces invàlides, que poden provocar, en temps d'execució, intents d'accés a memòria incorrectes, que generen una excepció d'adreçament incorrecte.

- L'adreça **0000h** conté la paraula formada pels bits **10h, 11h, 12h i 13h**, la paraula **13121110h**.
- Si s'accedeix a l'adreça **000Ah**, es pot llegir la mitja paraula formada pels bytes **1Ah i 1Bh** -> **1B1Ah**.
- Si s'accedeix al byte de l'adreça **0005h** es pot llegir el byte **15h**.

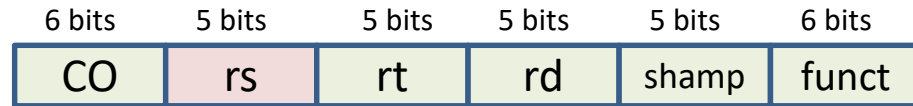
Adreça	Memòria			
@0000h	10h	11h	12h	13h
@0004h	14h	15h	16h	17h
@0008h	18h	19h	1Ah	1Bh
@000Ch	1Ch	1Dh	1Eh	1Fh

Instruccions: Codificació

La senzillesa del repertori d'instruccions del MIPS, juntament amb el fet que totes les instruccions ocupen la mateixa mida (32 bits), fa que hi hagi més d'una forma de codificar i d'interpretar els codis d'instrucció del MIPS32.

Hi ha tres formats:

- **Format R:**



Aquestes instruccions es corresponen al subconjunt d'instruccions del repertori que utilitzen bàsicament adreçaments a registre. Aquest format d'instrucció conté camps que varien entre 5 i 6 bits.

co: Codificació del codi d'operació genèric.

rs: Codificació del registre del primer operand font de la instrucció.

rt: Codificació del registre del segon operand font de la instrucció.

rd: Codificació del registre de l'operand destí de la instrucció.

shamp: Conté un valor que indica el valor del desplaçament. S'utilitza en algunes instruccions de desplaçament de bits a l'esquerra o a la dreta. En les instruccions que no s'utilitza aquest camp, es posa a 0.

funct: Aquest camp indica una funció dins la operació genèrica codificada en el primer camp (o codi d'operació). Per exemple, la suma (add) i la resta (sub) són la mateixa operació (amb codi 0) però amb funcions diferents (la 32 i la 34 respectivament).

Exemple: add \$1, \$2, \$3 -> \$1 := \$2 + \$3

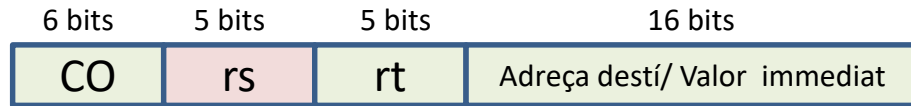


sub \$1, \$2, \$3 -> \$1 := \$2 - \$3



Instruccions: Codificació

- Format I:



Aquestes instruccions es corresponen al subconjunt d'instruccions que utilitzen el mode d'adreçament immediat.

co: Codificació del codi d'operació genèric.

rs: Codificació d'un registre font.

rt: Codificació d'un registre que pot ser font o destí.

Adreça destí / Valor immediat: Codificació d'un valor immediat de 16 bits. Si la instrucció és de salt, es considera part de l'adreça destí (adreçament relatiu al PC). En cas que sigui un altre tipus d'instrucció es un valor immediat corresponent a una constant. Segons el tipus d'operació aquest valor es pot interpretar com a valor sense signe o bé com un valor amb signe representat en C'2. Per exemple, en una instrucció `addi` (sumar) representa que és un valor amb signe codificat en C'2. En canvi, en una instrucció `andi` (and lògica) s'interpreta com un valor sense signe.

Exemple:

`addi $1, $2, 100 -> $1 := $2 + 100`

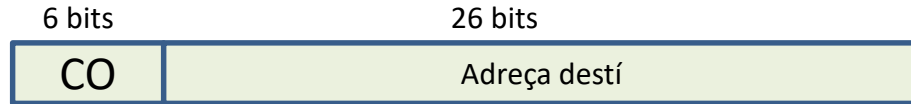


`lw $1, 100($2) -> $1 := Mem[$2 + 100]`



Instruccions: Codificació

- **Format J:**



Aquests format s'utilitza en algunes instruccions de salt.

co: Conté la codificació del codi d'operació genèric.

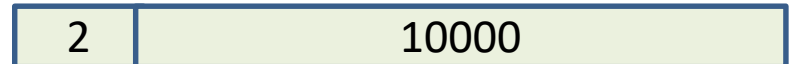
Adreça destí : Conté la codificació d'un valor immediat de 26 bits que representa una adreça absoluta de memòria.

L'adreça no és un valor en bytes, sinó que representa una quantitat de paraules de 32 bits, és a dir, que cal multiplicar per quatre el valor per indicar una posició de memòria.

Els salts absoluts no tenen accés a tota la memòria, només poden saltar a les 2^{26+2} posicions més properes al PC. Els quatre bits més significatius, no es modifiquen.

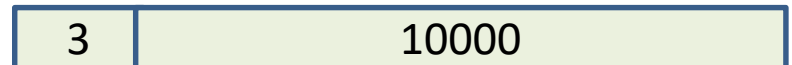
Exemple:

j 10000 -> PC := 10000 << 2 = 40000



jal 10000 -> \$31 = PC + 4

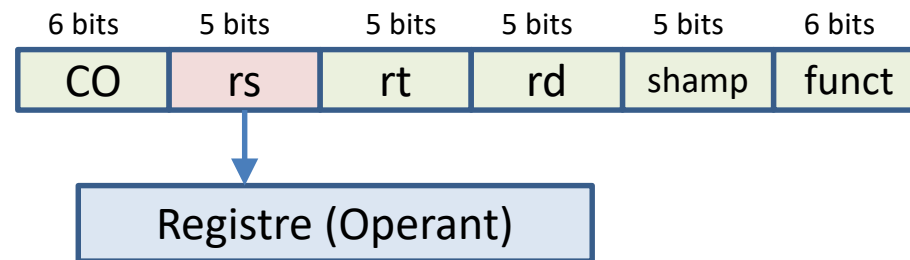
PC := 10000 << 2 = 40000



Instruccions: Modes d'adreçament

Per registre (R)

- Els operants de la instrucció es troben en algun dels registres.
- És el mode d'adreçament més ràpid. No necessita cap accés a memòria per aconseguir els operants.



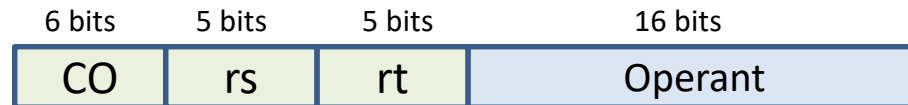
- Exemple: *add \$1, \$0, \$0*

Suma el contingut dels registre \$0 (un zero) amb ell mateix i deixa el resultat (zero) al registre \$1. És la forma més ràpida de posar a zero un registre.

Instruccions: Modes d'adreçament

Immediat (I)

- L'operant està codificat dins de la pròpia instrucció. També és molt ràpid, ja que l'operant és dins de la CPU, al registre IR.



- Exemple: *addi \$1, \$0, 0*

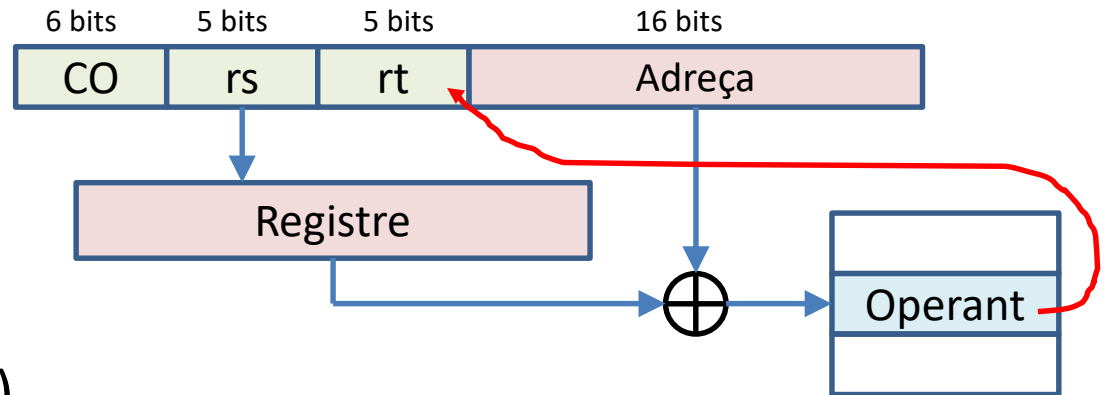
Suma el valor del registre \$0 (un zero) amb el valor immediat 0 i deixa el resultat (zero) al registre \$1. És una altre forma per posar a zero un registre.

La quantitat de bits dels valors immediats queda limitat per la mida de les instruccions del MIPS32 que és de 32 bits. Mai es podran carregar valors de 32 bits amb una sola instrucció.

Instruccions: Modes d'adreçament

Base més desplaçament o indexat

- L'operant és a la memòria. L'adreça s'obté sumant un valor fix i un de variable que s'utilitza com a índex.
- S'utilitza per accedir als valors d'un vector. El valor fixa sol ser l'adreça inicial del vector, i el valor variable l'índex.



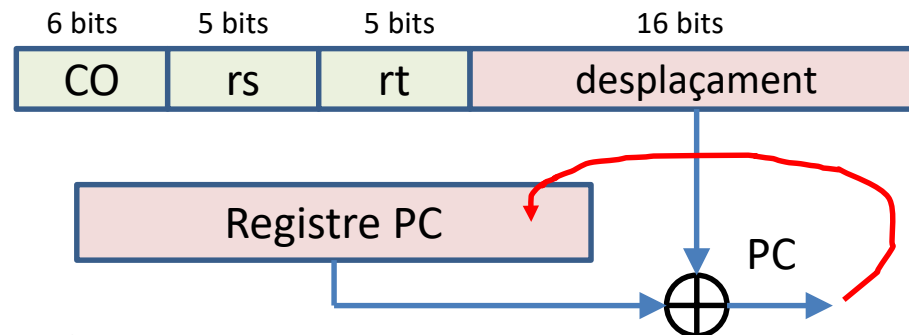
- Exemple: `lw $1, 4($2)`

Copia al registre al registre \$1, el valor que hi ha a la posició de memòria que s'obté de la suma de 4 amb el contingut del registre \$2.

Instruccions: Modes d'adreçament

Relatiu al PC

- L'operant s'obté de forma indexada (base o desplaçament) utilitzant un valor fix més el valor variable del registre PC.
- Normalment s'utilitza en les instruccions de control de flux (bifurcacions i salts condicionals).



- Exemple: `bne $0, $1, 100`

S'efectua un salt a l'adreça $PC + 100$, si el valor del registre `$0` és diferent al valor del registre `$1`.

El salt relatiu està codificat en `C'2`. Aquest desplaçament, senyala el nombre d'instruccions que s'ha d'avançar o retrocedir.

El salt es calcula: $PC := (PC+4) + desp * 4$

Instruccions: Tipus

- Instruccions de transferència de dades
- Instruccions aritmètiques per nombres sencers
- Instruccions lògiques
- Instruccions d'activació condicional
- Instruccions de rotació i desplaçament
- Instruccions de control de programa
- Instruccions de control de sistema

Instruccions: MIPS32

ABS.D	ABS.S	ADD	ADD.D	ADD.S	ADDI	ADDIU	ADDU
AND	ANDI	BC1F	BC1FL	BC1T	BC1TL	BC2F	BC2FL
BC2T	BC2TL	BEQ	BEQL	BGEZ	BGEZAL	BGEZALL	BGEZL
BGTZ	BGTZL	BLEZ	BLEZL	BLTZ	BLTZAL	BLTZALL	BLTZL
BNE	BNEL	BREAK	C.cond.D	C.cond.S	CACHE	CEIL.W.D	CEIL.W.S
CFC1	CFC2	CLO	CLZ	COP2	CTC1	CTC2	CVT.D.S
CVT.D.W	CVT.S.D	CVT.S.W	CVT.W.D	CVT.W.S	DIV	DIV.D	DIV.S
DIVU	ERET	FLOOR.W.D	FLOOR.W.S	J	JAL	JALR	JR
LB	LBU	LDC1	LDC2	LH	LHU	LL	LUI
LW	LWC1	LWC2	LWL	LWR	MADD	MADDU	MFC0
MFC1	MFC2	MFHI	MFLO	MOV.D	MOV.S	MOVF	MOVF.D
MOV.F.S	MOVN	MOVN.D	MOVN.S	MOVT	MOVT.D	MOVT.S	MOVZ
MOVZ.D	MOVZ.S	MSUB	MSUBU	MTC0	MTC1	MTC2	MTHI
MTLO	MUL	MUL.D	MUL.S	MULT	MULTU	NEG.D	NEG.S
NOR	OR	ORI	PREF	ROUND.W.D	ROUND.W.S	SB	SC
SDC1	SDC2	SH	SLL	SLLV	SLT	SLTI	SLTIU
SLTU	SQRT.D	SQRT.S	SRA	SRAV	SRL	SRLV	SSNOP
SUB	SUB.D	SUB.S	SUBU	SW	SWC1	SWC2	SWL
SWR	SYNC	SYSCALL	TEQ	TEQI	TGE	TGEI	TGEIU
TGEU	TLBP	TLBR	TLBWI	TLBWR	TLT	TLTI	TLTIU
TLTU	TNE	TNEI	TRUNC.W.D	TRUNC.W.S	WAIT	XOR	XORI

21 * 8 = 168 instruccions

Instruccions: MIPS32

Format de les dades de la FPU

Punt flotant de simple precisió (32 bits) (.fmt tipus S)

Punt flotant de doble precisió (64 bits) (.fmt tipus D)

Punt fix de precisió simple (32 bits) (.fmt tipus W)

ABS.D	ABS.S	ADD	ADD.D	ADD.S	ADDI	ADDIU	ADDU
AND	ANDI	BC1F	BC1FL	BC1T	BC1TL	BC2F	BC2FL
BC2T	BC2TL	BEQ	BEQL	BGEZ	BGEZAL	BGEZALL	BGEZL
BGTZ	BGTZL	BLEZ	BLEZL	BLTZ	BLTZAL	BLTZALL	BLTZL
BNE	BNEL	BREAK	C.cond.D	C.cond.S	CACHE	CEIL.W.D	CEIL.W.S
CFC1	CFC2	CLO	CLZ	COP2	CTC1	CTC2	CVT.D.S
CVT.D.W	CVT.S.D	CVT.S.W	CVT.W.D	CVT.W.S	DIV	DIV.D	DIV.S
DIVU	ERET	FLOOR.W.D	FLOOR.W.S	J	JAL	JALR	JR
LB	LBU	LDC1	LDC2	LH	LHU	LL	LUI
LW	LWC1	LWC2	LWL	LWR	MADD	MADDU	MFC0
MFC1	MFC2	MFHI	MFLO	MOV.D	MOV.S	MOVF	MOVF.D
MOVF.S	MOVN	MOVN.D	MOVN.S	MOVT	MOVT.D	MOVT.S	MOVZ
MOVZ.D	MOVZ.S	MSUB	MSUBU	MTC0	MTC1	MTC2	MTHI
MTLO	MUL	MUL.D	MUL.S	MULT	MULTU	NEG.D	NEG.S
NOR	OR	ORI	PREF	ROUND.W.D	ROUND.W.S	SB	SC
SDC1	SDC2	SH	SLL	SLLV	SLT	SLTI	SLTIU
SLTU	SQRT.D	SQRT.S	SRA	SRAV	SRL	SRLV	SSNOP
SUB	SUB.D	SUB.S	SUBU	SW	SWC1	SWC2	SWL
SWR	SYNC	SYSCALL	TEQ	TEQI	TGE	TGEI	TGEIU
TGEU	TLBP	TLBR	TLBWI	TLBWR	TLT	TLTI	TLTIU
TLTU	TNE	TNEI	TRUNC.W.D	TRUNC.W.S	WAIT	XOR	XORI

Instruccions: MIPS32

Transferència de dades

ABS.D	ABS.S	ADD	ADD.D	ADD.S	ADDI	ADDIU	ADDU
AND	ANDI	BC1F	BC1FL	BC1T	BC1TL	BC2F	BC2FL
BC2T	BC2TL	BEQ	BEQL	BGEZ	BGEZAL	BGEZALL	BGEZL
BGTZ	BGTZL	BLEZ	BLEZL	BLTZ	BLTZAL	BLTZALL	BLTZL
BNE	BNEL	BREAK	C.cond.D	C.cond.S	CACHE	CEIL.W.D	CEIL.W.S
CFC1	CFC2	CLO	CLZ	COP2	CTC1	CTC2	CVT.D.S
CVT.D.W	CVT.S.D	CVT.S.W	CVT.W.D	CVT.W.S	DIV	DIV.D	DIV.S
DIVU	ERET	FLOOR.W.D	FLOOR.W.S	J	JAL	JALR	JR
LB	LBU	LDC1	LDC2	LH	LHU	LL	LUI
LW	LWC1	LWC2	LWL	LWR	MADD	MADDU	MFC0
MFC1	MFC2	MFHI	MFLO	MOV.D	MOV.S	MOV.F	MOV.F.D
MOV.F.S	MOVN	MOVN.D	MOVN.S	MOVT	MOVT.D	MOVT.S	MOVZ
MOVZ.D	MOVZ.S	MSUB	MSUBU	MTC0	MTC1	MTC2	MTHI
MTLO	MUL	MUL.D	MUL.S	MULT	MULTU	NEG.D	NEG.S
NOR	OR	ORI	PREF	ROUND.W.D	ROUND.W.S	SB	SC
SDC1	SDC2	SH	SLL	SLLV	SLT	SLTI	SLTIU
SLTU	SQRT.D	SQRT.S	SRA	SRAV	SRL	SRLV	SSNOP
SUB	SUB.D	SUB.S	SUBU	SW	SWC1	SWC2	SWL
SWR	SYNC	SYSCALL	TEQ	TEQI	TGE	TGEI	TGEIU
TGEU	TLBP	TLBR	TLBWI	TLBWR	TLT	TLTI	TLTIU
TLTU	TNE	TNEI	TRUNC.W.D	TRUNC.W.S	WAIT	XOR	XORI

Transferència de dades

Sintaxi		T	Descripció	Exemple
lb	rt, desp(rs)	I	$rt = \text{ext_signe}(\text{Mem}[\text{desp}+(\text{rs})]_{7..0}, 32)$	lb \$1, 100(\$2); Carregar a \$1 un byte amb signe
lbu	rt, desp(rs)	I	$rt = \text{ext_zeros}(\text{Mem}[\text{desp}+(\text{rs})]_{7..0}, 32)$	lbu \$1, 100(\$2); Carregar a \$1 un byte sense signe
lh	rt, desp(rs)	I	$rt = \text{ext_signe}(\text{Mem}[\text{desp}+(\text{rs})]_{15..0}, 32)$	lh \$1, 100(\$2); Carregar a \$1 16 bits amb signe
lhu	rt, desp(rs)	I	$rt = \text{ext_zeros}(\text{Mem}[\text{desp}+(\text{rs})]_{15..0}, 32)$	lhu \$1, 100(\$2); Carregar a \$1 16 bits sense signe
lui	rd, imm16	I	$rd_{31..16} = \text{imm16}; rd_{15..0} = 0;$	lui \$1, 100; Carregar a \$1 (H) el valor de 16 bits
lw	rt, desp(rs)	I	$rt = \text{Mem}(\text{desp}+(\text{rs}))$	lw \$1, 100(\$2); Carregar a \$1 una paraula de 32 bits
lwl	rt, desp(rs)	I	$rt_{31..16} = \text{Mem}[\text{desp}+(\text{rs})]$	
lwr	rt, desp(rs)	I	$rt_{16..0} = \text{Mem}[\text{desp}+(\text{rs})]$	
sb	rt, desp(rs)	I	$\text{Mem}[\text{desp}+(\text{rs})] = rt_{7..0}$	sb \$1, 100(\$2); Emmagatzemar el byte baix de \$1
sh	rt, desp(rs)	I	$\text{Mem}[\text{desp}+(\text{rs})] = rt_{15..0}$	sh \$1, 100(\$2); Emmagatzemar els 16 bits (L) de \$1
sw	rt, desp(rs)	I	$\text{Mem}[\text{desp}+(\text{rs})] = rt$	sw \$1, 100(\$2); Emmagatzemar \$1
swl	rt, desp(rs)	I	$\text{Mem}[\text{desp}+(\text{rs})] = rt_{31..16}$	
swr	rt, desp(rs)	I	$\text{Mem}[\text{desp}+(\text{rs})] = rt_{15..0}$	
mfhi	rd	I	$rd = hi$	mfhi \$1;
mflo	rd	I	$rd = lo$	mflo \$1;
mthi	rd	I	$hi = rd$	mthi \$1;
mtlo	rd	I	$lo = rd$	mtlo \$1;
movn	rd, rs, rt	R	Si $(rt \lt; 0) \rightarrow rd = rs$	movn \$1, \$2, \$3; Copiar si no és zero
movz	rd, rs, rt	R	Si $(rt = 0) \rightarrow rd = rs$	movz \$1, \$2, \$3; Copiar si és zero

- LWL rt, desplaçament(base) **Load Word Left**

Propòsit: És per carregar la part **més** significativa d'una paraula com un valor amb signe d'una adreça de memòria no alineada.

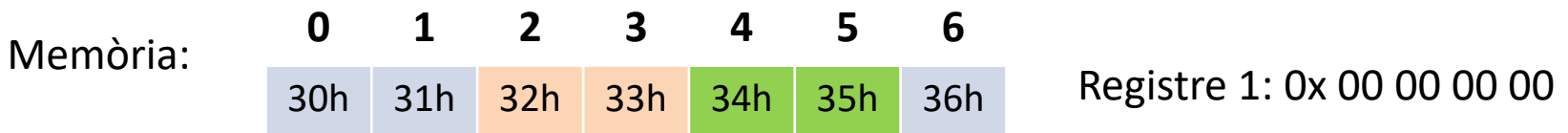
Descripció: $rt \leftarrow rt \text{ combinat Mem}(\text{base} + \text{desplaçament})$

- LWR rt, desplaçament(base) **Load Word Right**

Propòsit: És per carregar la part **menys** significativa d'una paraula com un valor amb signe d'una adreça de memòria no alineada.

Descripció: $rt \leftarrow rt \text{ combinat Mem}(\text{base} + \text{desplaçament})$

Exemple: Paraula al segon byte de la memòria. Big-endian



Instrucció: `lwl $1, 2($0)`  Registre 1: 0x **32 33** 00 00

Instrucció: `lwr $1, 5($0)`  Registre 1: 0x 32 33 **34 35**

Instruccions: MIPS32 Aritmètiques, desplaçament i lògiques

ABS.D	ABS.S	ADD	ADD.D	ADD.S	ADDI	ADDIU	ADDU
AND	ANDI	BC1F	BC1FL	BC1T	BC1TL	BC2F	BC2FL
BC2T	BC2TL	BEQ	BEQL	BGEZ	BGEZAL	BGEZALL	BGEZL
BGTZ	BGTZL	BLEZ	BLEZL	BLTZ	BLTZAL	BLTZALL	BLTZL
BNE	BNEL	BREAK	C.cond.D	C.cond.S	CACHE	CEIL.W.D	CEIL.W.S
CFC1	CFC2	CLO	CLZ	COP2	CTC1	CTC2	CVT.D.S
CVT.D.W	CVT.S.D	CVT.S.W	CVT.W.D	CVT.W.S	DIV	DIV.D	DIV.S
DIVU	ERET	FLOOR.W.D	FLOOR.W.S	J	JAL	JALR	JR
LB	LBU	LDC1	LDC2	LH	LHU	LL	LUI
LW	LWC1	LWC2	LWL	LWR	MADD	MADDU	MFC0
MFC1	MFC2	MFHI	MFLO	MOV.D	MOV.S	MOV.F	MOV.F.D
MOV.F.S	MOV.N	MOV.N.D	MOV.N.S	MOVT	MOVT.D	MOVT.S	MOVZ
MOVZ.D	MOVZ.S	MSUB	MSUBU	MTC0	MTC1	MTC2	MTHI
MTLO	MUL	MUL.D	MUL.S	MULT	MULTU	NEG.D	NEG.S
NOR	OR	ORI	PREF	ROUND.W.D	ROUND.W.S	SB	SC
SDC1	SDC2	SH	SLL	SLLV	SLT	SLTI	SLTIU
SLTU	SQRT.D	SQRT.S	SRA	SRAV	SRL	SRLV	SSNOP
SUB	SUB.D	SUB.S	SUBU	SW	SWC1	SWC2	SWL
SWR	SYNC	SYSCALL	TEQ	TEQI	TGE	TGEI	TGEIU
TGEU	TLBP	TLBR	TLBWI	TLBWR	TLT	TLTI	TLTIU
TLTU	TNE	TNEI	TRUNC.W.D	TRUNC.W.S	WAIT	XOR	XORI

Aritmètiques, desplaçament i lògiques

Sintaxi		T	Descripció	Exemple
add	rd, rs, rt	R	$rd = rs + rt$	add \$1, \$2, \$3; Suma paraules amb signe
addi	rd, rs, imm16	I	$rd = rs + \text{ext_signe}(\text{imm16}, 32)$	add \$1, \$2, 100; Suma paraules amb signe
addu	rd, rs, rt	R	$rd = rs + rt$	add \$1, \$2, \$3; Suma paraules sense signe
addiu	rd, rs, imm16	I	$rd = rs + \text{ext_zeros}(\text{imm16}, 32)$	add \$1, \$2, 100; Suma paraules sense signe
div	rt, rs	R	$lo = rs / rt; hi = \text{mod}(rs, rt);$	div \$2, \$3; Divisió amb signe
divu	rt, rs	R	$lo = rs / rt; hi = \text{mod}(rs, rt);$	div \$2, \$3; Divisió sense signe
mult	rt, rs	R	$hi-lo = rt * rs$	mult \$2, \$3; Producte amb signe
multu	rt, rs	R	$hi-lo = rt * rs$	mult \$2, \$3; Producte sense signe
sub	rd, rs, rt	R	$rd = rs - rt$	sub \$1, \$2, \$3; Diferència amb signe
subu	rd, rs, rt	R	$rd = rs - rt$	subi \$1, \$2, \$3; Diferència sense signe
and	rd, rs, rt	R	$rd = rs \text{ AND } rt$	and \$1, \$2, \$3
andi	rd, rs, imm16	I	$rd = rs \text{ AND } \text{ext_zeros}(\text{imm16}, 32)$	andi \$1, \$2, 100
nor	rd, rs, rt	R	$rd = rs \text{ NOR } rt$	
or	rd, rs, rt	R	$rd = rs \text{ OR } rt$	
ori	rd, rs, imm16	I	$rd = rs \text{ OR } \text{ext_zeros}(\text{imm16}, 32)$	
xor	rd, rs, rt	R	$rd = rs \text{ XOR } rt$	
xori	rd, rs, imm16	I	$rd = rs \text{ XOR } \text{ext_zeros}(\text{imm16}, 32)$	
sll	rd, rt, shamt5	R	$rd = \text{desp_lògic}(rt, \text{shamt5}, \text{esquerra})$	sll \$1, \$2, 3; $\$1 = \$2 \ll 3$
sllv	rd, rt, rs	R	$rd = \text{desp_lògic_var}(rt, rs_{4..0}, \text{esquerra})$	sllv \$1, \$2, \$3; $\$1 = \$2 \ll \$3_{4..0}$
sra	rd, rt, shamt5	R	$rd = \text{desp_aritmètic}(rt, \text{shamt5}, \text{dreta})$	sra \$1, \$2, 3; $\$1 = \$2 \text{ div } 2^3$
srav	rd, rt, rs	R	$rd = \text{desp_aritmètic_var}(rt, rs_{4..0}, \text{dreta})$	srav \$1, \$2, \$3; $\$1 = \$2 \text{ div } 2^{(\$3 \text{ and } 1Fh)}$
srl	rd, rt, shamt5	R	$rd = \text{desp_lògic}(rt, \text{shamt5}, \text{dreta})$	srl \$1, \$2, 3; $\$1 = \$2 \gg 3$
srlv	rd, rt, rs	R	$rd = \text{desp_lògic_var}(rt, rs_{4..0}, \text{dreta})$	srlv \$1, \$2, \$3; $\$1 = \$2 \gg \$3_{4..0}$

Instruccions: MIPS32 Activació condicional, control de programa i varis

ABS.D	ABS.S	ADD	ADD.D	ADD.S	ADDI	ADDIU	ADDU
AND	ANDI	BC1F	BC1FL	BC1T	BC1TL	BC2F	BC2FL
BC2T	BC2TL	BEQ	BEQL	BGEZ	BGEZAL	BGEZALL	BGEZL
BGTZ	BGTZL	BLEZ	BLEZL	BLTZ	BLTZAL	BLTZALL	BLTZL
BNE	BNEL	BREAK	C.cond.D	C.cond.S	CACHE	CEIL.W.D	CEIL.W.S
CFC1	CFC2	CLO	CLZ	COP2	CTC1	CTC2	CVT.D.S
CVT.D.W	CVT.S.D	CVT.S.W	CVT.W.D	CVT.W.S	DIV	DIV.D	DIV.S
DIVU	ERET	FLOOR.W.D	FLOOR.W.S	J	JAL	JALR	JR
LB	LBU	LDC1	LDC2	LH	LHU	LL	LUI
LW	LWC1	LWC2	LWL	LWR	MADD	MADDU	MFC0
MFC1	MFC2	MFHI	MFLO	MOV.D	MOV.S	MOV.F	MOV.F.D
MOV.F.S	MOV.N	MOV.N.D	MOV.N.S	MOVT	MOVT.D	MOVT.S	MOVZ
MOVZ.D	MOVZ.S	MSUB	MSUBU	MTC0	MTC1	MTC2	MTHI
MTLO	MUL	MUL.D	MUL.S	MULT	MULTU	NEG.D	NEG.S
NOR	OR	ORI	PREF	ROUND.W.D	ROUND.W.S	SB	SC
SDC1	SDC2	SH	SLL	SLLV	SLT	SLTI	SLTIU
SLTU	SQRT.D	SQRT.S	SRA	SRAV	SRL	SRLV	SSNOP
SUB	SUB.D	SUB.S	SUBU	SW	SWC1	SWC2	SWL
SWR	SYNC	SYSCALL	TEQ	TEQI	TGE	TGEI	TGEIU
TGEU	TLBP	TLBR	TLBWI	TLBWR	TLT	TLTI	TLTIU
TLTU	TNE	TNEI	TRUNC.W.D	TRUNC.W.S	WAIT	XOR	XORI

Activació condicional, control de programa i varis

Sintaxi		T	Descripció	Exemple
slt	rd, rs, rt	R	Si (rs < rt), rd = 1; sinó rd = 0 (C'2)	slt \$1, \$2, \$3;
slti	rd rs, inm16	I	Si (rs < ext_signe(im16, 32), rd = 1; sinó rd = 0; (C'2)	slti \$1, \$2, 100;
sltu	rd, rs, rt	R	Si (rs < rt), rd = 1; sinó rd = 0	sltu \$1, \$2, \$3;
sltiu	rd, rs, inm16	I	Si (rs < ext_zeros(im16, 32), rd = 1; sinó rd = 0;	slti \$1, \$2, 100;
beq	rt, rs, etiq	I	Si (rs = rt) PC = PC + etiq * 4; sinó PC = PC + 4	beq \$1, \$2, 100
bgez	rs, etiq	I	Si (rs >= 0) PC = PC + etiq; sinó PC = PC + 4	bgez \$1, 100
bgezal	rs, etiq	I	Si (rs >= 0) PC = PC + etiq, \$31 = PC + 4; sinó PC = PC + 4	bgezal \$1, 100
bgtz	rs, etiq	I	Si (rs > 0) PC = PC + etiq; sinó PC = PC + 4	
blez	rs, etiq	I	Si (rs <= 0) PC = PC + etiq; sinó PC = PC + 4	
bltz	rs, etiq	I	Si (rs < 0) PC = PC + etiq; sinó PC = PC + 4	
bltzal	rs, etiq	I	Si (rs < 0) PC = PC + etiq, \$31 = PC + 4; sinó PC = PC + 4	
bne	rs, rt, etiq	I	Si (rs <> rt) PC = PC + etiq * 4; sinó PC = PC + 4	
J	etiqueta	J	PC = PC _{31..28} (etiqueta << 2)	J 1000
jal	etiqueta	J	\$31 = PC + 4; PC = PC _{31..28} (etiqueta << 2)	
jalr	rd, rs	R	rd = PC + 4; PC = rs	jalr \$31, \$1
jr	rs	R	PC = rs	Jr \$1
syscall		R	Crida al sistema	
break	codi	R	Excepció de punt de parada. Codi: bits 25..6	
nop		R	No operació	

Instruccions: MIPS32

ABS.D	ABS.S	ADD	ADD.D	ADD.S	ADDI	ADDIU	ADDU
AND	ANDI	BC1F	BC1FL	BC1T	BC1TL	BC2F	BC2FL
BC2T	BC2TL	BEQ	BEQL	BGEZ	BGEZAL	BGEZALL	BGEZL
BGTZ	BGTZL	BLEZ	BLEZL	BLTZ	BLTZAL	BLTZALL	BLTZL
BNE	BNEL	BREAK	C.cond.D	C.cond.S	CACHE	CEIL.W.D	CEIL.W.S
CFC1	CFC2	CLO	CLZ	COP2	CTC1	CTC2	CVT.D.S
CVT.D.W	CVT.S.D	CVT.S.W	CVT.W.D	CVT.W.S	DIV	DIV.D	DIV.S
DIVU	ERET	FLOOR.W.D	FLOOR.W.S	J	JAL	JALR	JR
LB	LBU	LDC1	LDC2	LH	LHU	LL	LUI
LW	LWC1	LWC2	LWL	LWR	MADD	MADDU	MFC0
MFC1	MFC2	MFHI	MFLO	MOV.D	MOV.S	MOVF	MOVF.D
MOVF.S	MOVN	MOVN.D	MOVN.S	MOVT	MOVT.D	MOVT.S	MOVZ
MOVZ.D	MOVZ.S	MSUB	MSUBU	MTC0	MTC1	MTC2	MTHI
MTLO	MUL	MUL.D	MUL.S	MULT	MULTU	NEG.D	NEG.S
NOR	OR	ORI	PREF	ROUND.W.D	ROUND.W.S	SB	SC
SDC1	SDC2	SH	SLL	SLLV	SLT	SLTI	SLTIU
SLTU	SQRT.D	SQRT.S	SRA	SRAV	SRL	SRLV	SSNOP
SUB	SUB.D	SUB.S	SUBU	SW	SWC1	SWC2	SWL
SWR	SYNC	SYSCALL	TEQ	TEQI	TGE	TGEI	TGEIU
TGEU	TLBP	TLBR	TLBWI	TLBWR	TLT	TLTI	TLTIU
TLTU	TNE	TNEI	TRUNC.W.D	TRUNC.W.S	WAIT	XOR	XORI

Instruccions en coma flotant

Accés a memòria

Instrucció	Operació	Comentari
lwc1 ft, desp(rs)	$ft_{0..31} = \text{Mem}[\text{desp}+(\text{rs})]$	Lectura d'un real de 32 bits (COP1)
swc1 ft, desp(rs)	$\text{Mem}[\text{desp}+(\text{rs})] = ft_{0..31}$	Escriptura d'un real de 32 bits (COP1)

Aritmètiques

Instrucció	Operació	Comentari
add.fms fd, fs, ft	$fd = fs + ft$	Fms = s -> Simple precisió
sub.fms fd, fs, ft	$fd = fs - ft$	Fms = d -> Doble precisió
mul.fms fd, fs, ft	$fd = fs * ft$	
div.fms fd, fs, ft	$fd = fs / ft$	

Comparació

Instrucció	Operació	Casos
c.con.fmt fs, ft	$cc = (\text{fs con ft})$	c.lt.d fs, ft -> $CC = (\text{fs} < \text{ft})$ c.le.d fs, ft -> $CC = (\text{fs} \leq \text{ft})$ c.eq.d fs, ft -> $CC = (\text{fs} = \text{ft})$

Instruccions: MIPS32

ABS.D	ABS.S	ADD	ADD.D	ADD.S	ADDI	ADDIU	ADDU
AND	ANDI	BC1F	BC1FL	BC1T	BC1TL	BC2F	BC2FL
BC2T	BC2TL	BEQ	BEQL	BGEZ	BGEZAL	BGEZALL	BGEZL
BGTZ	BGTZL	BLEZ	BLEZL	BLTZ	BLTZAL	BLTZALL	BLTZL
BNE	BNEL	BREAK	C.cond.D	C.cond.S	CACHE	CEIL.W.D	CEIL.W.S
CFC1	CFC2	CLO	CLZ	COP2	CTC1	CTC2	CVT.D.S
CVT.D.W	CVT.S.D	CVT.S.W	CVT.W.D	CVT.W.S	DIV	DIV.D	DIV.S
DIVU	ERET	FLOOR.W.D	FLOOR.W.S	J	JAL	JALR	JR
LB	LBU	LDC1	LDC2	LH	LHU	LL	LUI
LW	LWC1	LWC2	LWL	LWR	MADD	MADDU	MFC0
MFC1	MFC2	MFHI	MFLO	MOV.D	MOV.S	MOVF	MOVF.D
MOVF.S	MOVN	MOVN.D	MOVN.S	MOVT	MOVT.D	MOVT.S	MOVZ
MOVZ.D	MOVZ.S	MSUB	MSUBU	MTC0	MTC1	MTC2	MTHI
MTLO	MUL	MUL.D	MUL.S	MULT	MULTU	NEG.D	NEG.S
NOR	OR	ORI	PREF	ROUND.W.D	ROUND.W.S	SB	SC
SDC1	SDC2	SH	SLL	SLLV	SLT	SLTI	SLTIU
SLTU	SQRT.D	SQRT.S	SRA	SRAV	SRL	SRLV	SSNOP
SUB	SUB.D	SUB.S	SUBU	SW	SWC1	SWC2	SWL
SWR	SYNC	SYSCALL	TEQ	TEQI	TGE	TGEI	TGEIU
TGEU	TLBP	TLBR	TLBWI	TLBWR	TLT	TLTI	TLTIU
TLTU	TNE	TNEI	TRUNC.W.D	TRUNC.W.S	WAIT	XOR	XORI

Instruccions en coma flotant

De salt condicional

Instrucció	Operació	Comentari
bc1f desp	Si CC = 0 -> Saltar	Salt restringit a una zona de 128 KB
bc1t desp	Si CC = 1 -> Saltar	

Altres

Instrucció	Operació	Comentari
mtc1 rt, fs	fs <- rt	
mfc1 rt, fs	rt = ext_signe(fs _{31..0})	Amb extensió de signe
mov.fmt fd, fs	fd = fs	fmt = s , d
cvt.d.fmt fd, fs	fd = Doble(fs)	Converteix a doble precisió
cvt.s.fmt fd, fs	fd = Simple(fs)	Converteix a simple precisió
cvt.w.fmt fd, fs	fd = PuntFix(fs)	Converteix a punt fix



Espai d'adreces de memòria dels MIPS

- El mapa de memòria està dividit en varies zones o segments
- Dels 4 GB disponibles de memòria, només 2 GB estan disponibles pels programes de l'usuari (zona kuseg). La resta serveix pel sistema operatiu i per la memòria virtual
- La zona de l'usuari se divideix en:
 - Segment de text, on s'emmagatzema el programa
 - Segment de dades estàtiques, que emmagatzema les dades necessàries definides en temps de compilació del programa
 - El punter global (\$gp) apunta al centre de l'espai màxim per variables estàtiques. El seu valor inicial l'ha d'assignar al codi al inici del programa.
 - Segment per les dades dinàmiques, és a dir, que es defineixen en temps d'execució del programa
 - El punter de pila (\$sp) apunta inicialment a la posició 0x7FFFFFFF (“fons de la pila”)

Espai d'adreces de memòria dels MIPS

0x00000000	Espai de memòria d'usuari (kuseg) 2 GB
0x7FFFFFFF 0x80000000	Nucli no mapejat (kseg0) 512 MB
0x9FFFFFFF 0xA0000000	Nucli no mapejat i no ubicat a la cau (kseg1) 512 MB
0xBFFFFFFF 0xC0000000	Supervisor mapejat (kseg2) 512 MB
0xDFFFFFFF 0xE0000000	Nucli mapejat (kseg2) 512 MB
0xFFFFFFFF	

- Modes del processador:

Hi ha tres formes diferents d'execució de la CPU:

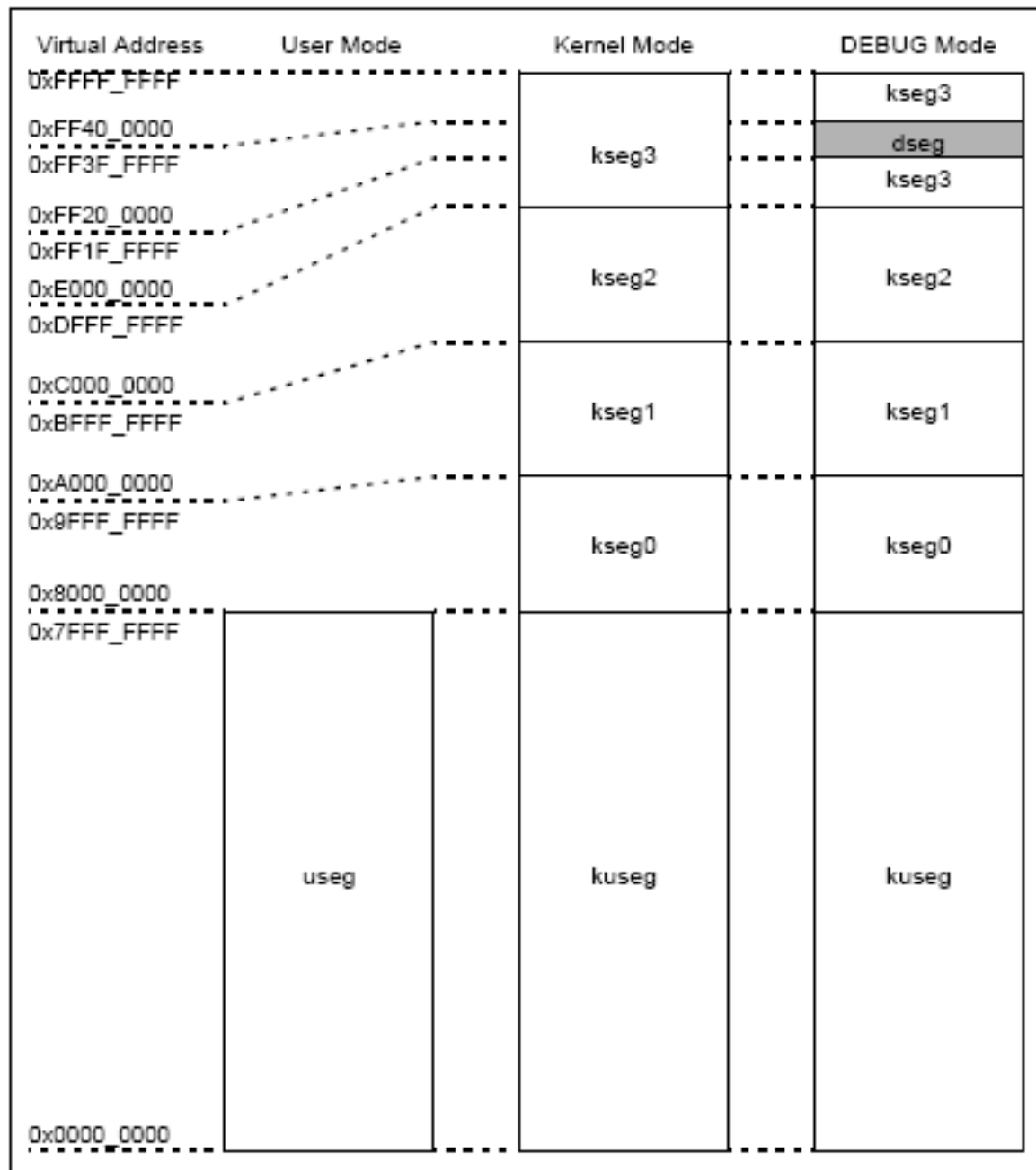
1. mode usuari
2. mode nucli
3. mode Debug.

El processador comença l'execució en el mode nucli i pot quedar-se en aquest mode durant l'operació normal.

El mode usuari és un mode opcional que permet al dissenyador partir el codi entre programari privilegiat i no privilegiat.

El mode DEBUG només s'usa mitjançant un depurador.

Una de les principals diferències entre els diferents modes d'operació és el direccionament de la memòria, de manera que el programari només permet l'accés a determinades regions. Per exemple els perifèrics no són accessibles en el mode usuari.



Mode Nucli

- Per poder accedir a tots els recursos maquinari, el processador ha d'estar en aquest estat.
- Permet l'accés del programari a tot l'espai d'adreces del processador així com també a les instruccions privilegiades.
- El processador operarà amb aquest mode quan el bit DM del registre DEBUG val "0" i el registre STATUS conté els següents valors: UM = 0; ERL = 1; EXL = 1.

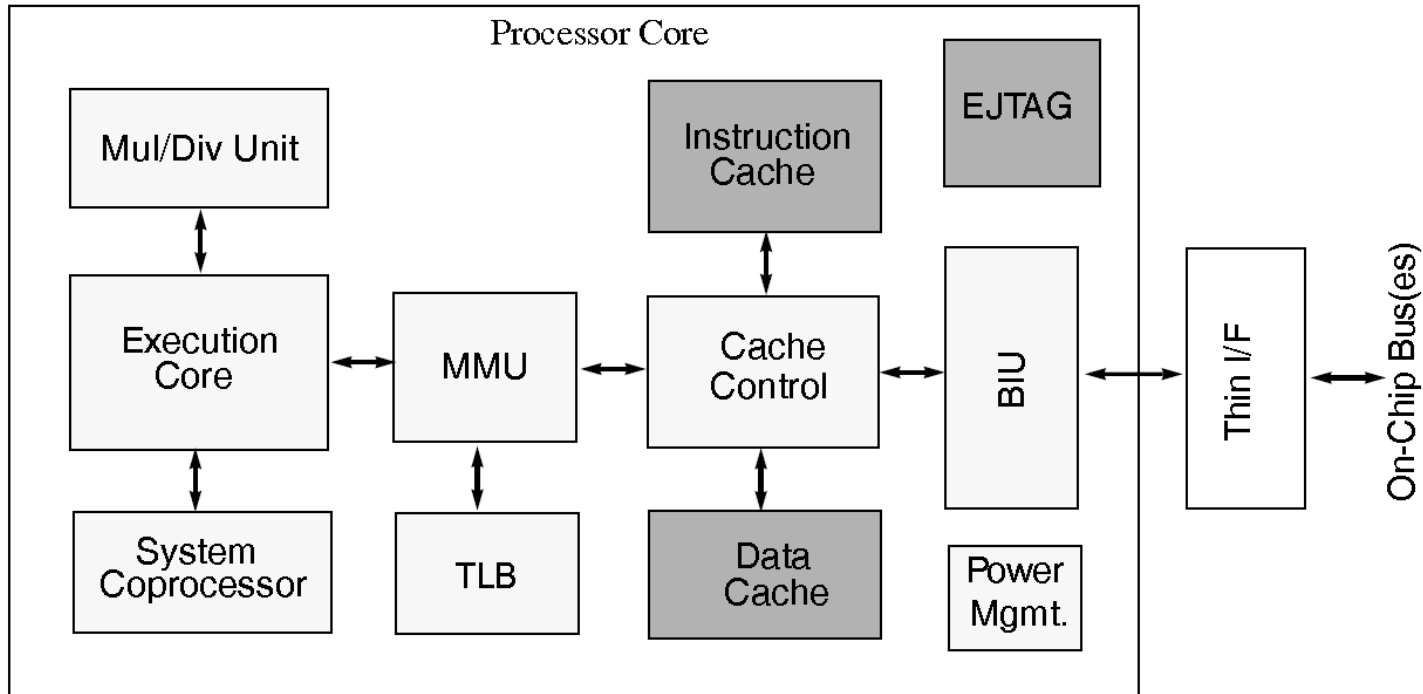
Mode Usuari

- Registre STATUS: UM = 1; EXL = 0; ERL = 0.
- El programari es restringeix a una sèrie de recursos del processador. De tal manera, que en aquest mode només té accés a l'àrea de memòria USEG.

Mode Degug

- Aquest mode especial del processador, requereix que s'executi una excepció debug.
- Es podrà accedir a tots els recursos com en el mode nucli i als recursos de maquinari especials usats en les aplicacions de debug.
- Quan s'entra aquest mode el bit DM del registre DEBUG val 1.
- Per sortir d'aquest mode només cal executar la instrucció DERET.

Diagrama de blocs d'un nucli 4 kc del MIPS32



Blocs necessaris:

- Unitat d'execució
- Unitat de multiplicar/dividir (MDU)
- Sistema de Control del coprocessador (CPO)
- Unitat de control de la memòria (MMU)
- Taula de pàgines actives de la memòria virtual (TLB)
- Controlador de la memòria cau
- Unitat d'interfície amb el Bus (BIU)
- Control de la potència

Blocs opcionals:

- Memòria cau d'instruccions
- Memòria cau de dades
- Controlador Enhanced JTAG (EJTAG) (test)

Estructures de control

- Es poden destacar les següents instruccions:
 - beq r1, r2, Desp (Saltar si és igual)
 - Compara els valors dels registres r1 i r2
 - Si son iguals, el flux del programa salta a la instrucció que es correspon al desplaçament indicat. ($PC = PC + Desp$)
 - Si son diferents, s'executa la següent instrucció.
 - bne r1, r2, Desp (Saltar si és diferent)
 - Si el valor dels dos registres son diferents, al programa salta a l'etiqueta especificada.
 - Si son iguals, continua l'execució.
 - slt r1, r2, r3 (menor)
 - Si r2 és menor que r3, r1 es carrega amb un 1.
 - Si r2 és més gran o igual que r3, r1 valdrà 0.

Estructuras de control

- Exemple:

- C:

```
if (i==j)
    f = f - i;
else
    f = g + h;
```

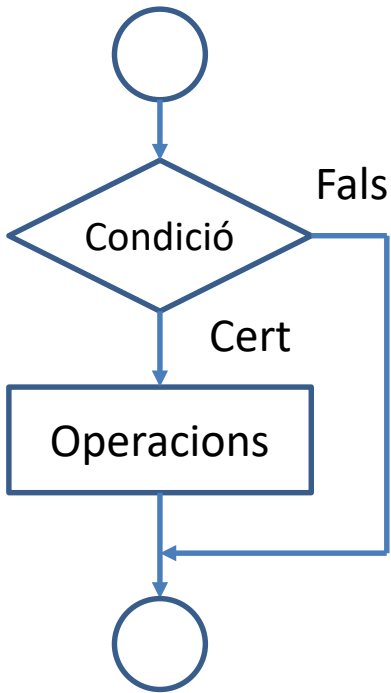
- MIPS:

```
# Suposem que les 5 variables es corresponen als registres $s:
# f : $s0; g : $s1; h : $s2; i : $s3; j : $s4;
        beq $s3, $s4, THEN # ($s3 == $s4) saltar a THEN
        add $s0, $s1, $s2; # f = g + h
        j Fi_IF
THEN:   sub $s0, $s0, $s3   # f = f - i
Fi_IF:
```

Estructures de control

- Exemple:

Condió if-then



- C:
(Variables senceres)
if (x >= y) {
 x = x + 2;
 y = y - 2;
}

- MIPS:

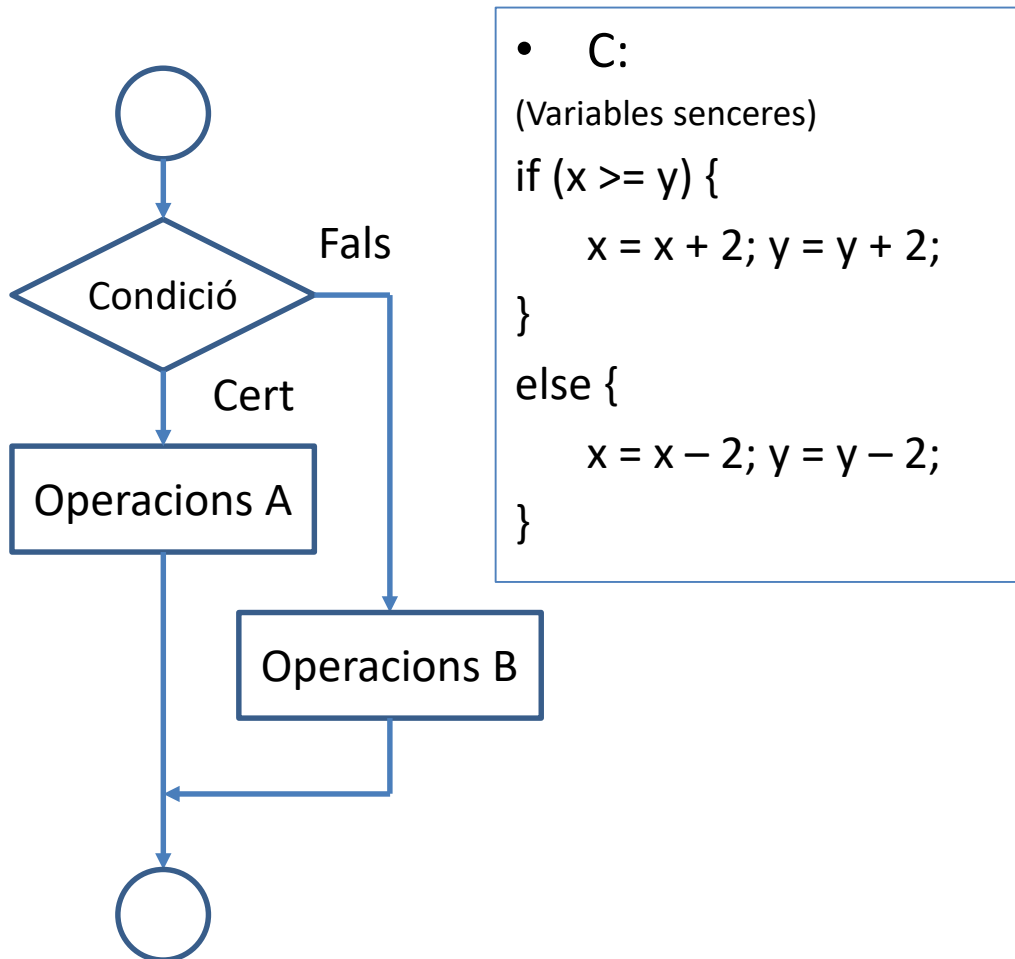
```
IF:      lw      $s0, X
         lw      $s1, Y
         slt     $s0, $s1, Fi_if  # s0 < s1?
         addi   $s0, $s0, 2
         addi   $s1, $s1, -2
         sw     $s0, X
         sw     $s1, Y
```

Fi_if:

Estructures de control

- Exemple:

Condió if-then-else



- C:
(Variables senceres)
if (x >= y) {
 x = x + 2; y = y + 2;
}
else {
 x = x - 2; y = y - 2;
}

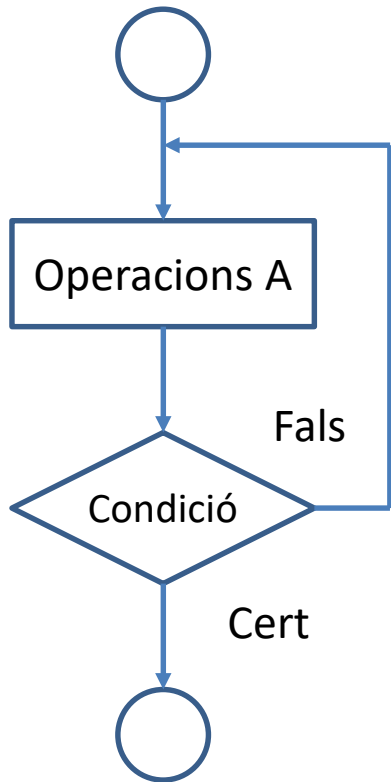
- MIPS:
IF: lw \$s0, X
 lw \$s1, Y
 bge \$s0, \$s1, Then
Else: addi \$s0, \$s0, -2
 addi \$s1, \$s1, -2
 j Fi_if
Then: addi \$s0, \$s0, 2
 addi \$s1, \$s1, 2
Fi_if: sw \$s0, X
 sw \$s1, Y

Estructures de control

- Exemple:

Condió repeat-until

Algoritme que calcula el màxim comú divisor



- C:
(Variables senceres)
a = 81;
b = 18;
do {
 mcd = b;
 res = a % b;
 a = b;
 b = res;
} while (res <> 0);

- MIPS:

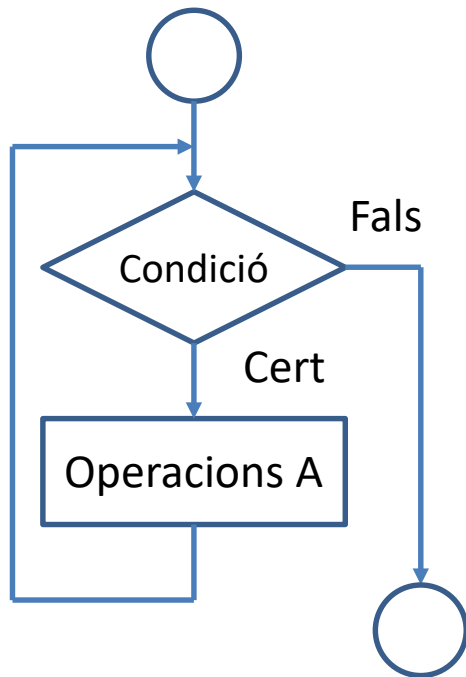
```
addi    $s0, $0, 81    # a = 81
addi    $s1, $0, 18    # b = 18
Repeat: add    $s2, $0, $s1    # mcd = b
div     $s0, $s1        # a % b
addi    $s0, $s1, 0    a = b
mfhi    $s1             #b = res
bnez    $s1, Repeat
..
```

Estructures de control

- Exemple:

Condició While-do

Càlcul del terme anéssim de la sèrie de Fibonacci



- C:
(Variables senceres)
n = 5; fant = 1;
f = 1; i = 2;
while (i <= n) {
 faux = f;
 f = f + fant;
 fant = faux;
 i = i + 1;
}

- MIPS:

```
# n : $s0; f : $s1; fant : $s2;
```

```
# i : $s3; faux : $s4
```

```
li $s0, 5
```

```
li $s2, 1
```

```
li $s1, 1
```

```
li $s3, 2
```

```
While: subu $s5, $s3, $s0
```

```
bgtz $s5, Fi
```

```
addu $s4, $s1, $0
```

```
addu $s1, $s1, $s2
```

```
addu $s2, $s4, $0
```

```
addiu $s3, $s3, 1
```

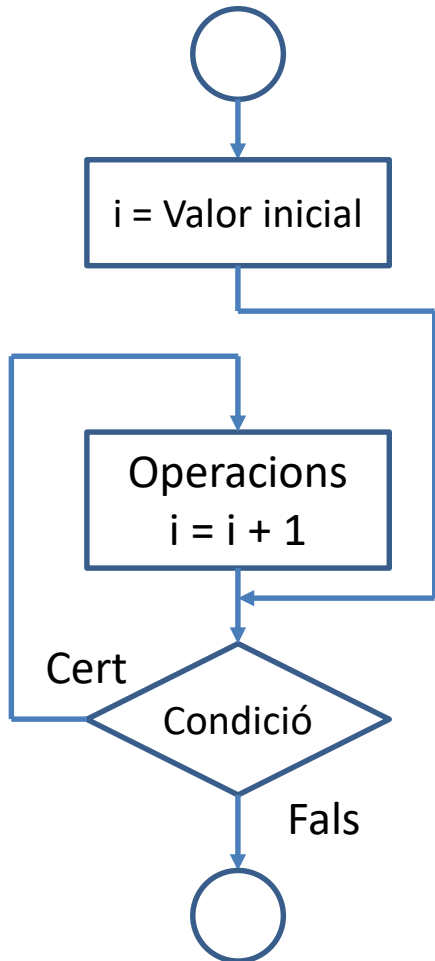
```
j While
```

```
..
```

Estructures de control

- Exemple:

Condició For



Càlcul del terme anéssim de la sèrie de Fibonacci

- C:
(Variables senceres)
n = 5; fant = 1;
f = 1;
for(i = 2; i <= n; i++) {
 faux = f;
 f = f + fant;
 fant = faux;
}

- MIPS:

n : \$s0; f : \$s1; fant : \$s2;

i : \$t0; faux : \$s4

li \$s0, 5

li \$s2, 1

li \$s1, 1

li \$t0, 2

For: slt \$t1, \$s0, \$t0

bltz \$t1, Fi_For

addu \$s4, \$s1, \$0

addu \$s1, \$s1, \$s2

addu \$s2, \$s4, \$0

j For

Fi_For: ..

Estructures de control

- Exemple:

- C:
(Variables senceres)
Int a, V[100];
Int i, j, k;
i = 0; j = 1;
while (V[i] == k)
{
 i = i + j;
}

- MIPS:
L'adreça de V[0] : \$s1;
les variables i, j i k a \$s2, \$s3 i \$s4
 addu \$s2, \$0, \$0 # i = 0
 addui \$s3, \$0, 1 # j = 1
While: addu \$t1, \$s2, \$s2 # t1 = 2 * i
 addu \$t1, \$t1, \$t1 # t1 = 4 * i
 addu \$t1, \$t1, \$s1 # t1 = @ V[i]
 lw \$t0, 0(\$t0) # t0 = V[i]
 bne \$t0, \$s4, Fi # V[i] == k ?
 addu \$s2, \$s2, \$s3 # i = i + j
Fi: ..