L'arquitectura CISCA

Miquel Albert Orenga
Gerard Enrique Manonellas

PID_00218253

Universitat Oberta de Catalunya

Índex

Introducció

Atesa la gran varietat de processadors comercials i a causa de la seva complexitat creixent, hem optat per definir una màquina de propòsit general que anomenarem **Complex Instruction Set Computer Architecture** (CISCA) i que utilitzarem en els exemples que ens ajudaran a entendre millor els conceptes tractats en aquesta assignatura, i també en els exercicis que anirem proposant.

L'arquitectura CISCA s'ha definit seguint un model senzill de màquina, del qual només definirem els elements més importants. D'aquesta manera, serà més fàcil entendre les referències als elements del computador i altres conceptes referents al seu funcionament.

S'ha definit una arquitectura per a treballar els conceptes teòrics generals tan semblant com sigui possible a l'arquitectura x86-64, per a facilitar el pas a la programació sobre aquesta arquitectura real. Aquesta serà l'arquitectura sobre la qual es desenvoluparan les pràctiques.

Objectius

Amb els materials didàctics d'aquest mòdul es pretén que els estudiants assoleixin els objectius següents:

1. Conèixer els elements bàsics d'un computador senzill i comprendre'n el funcionament.

2. Conèixer el joc d'instruccions d'una arquitectura concreta amb unes especificacions pròpies.

3. Aprendre els conceptes bàsics de programació a partir d'una arquitectura senzilla però propera a una arquitectura real.

4. Ser capaç de convertir el codi assemblador que genera el programador en codi màquina que pugui interpretar el computador.

5. Entendre què fa cada una de les instruccions d'un joc d'instruccions en assemblador i veure quins efectes tenen sobre els diferents elements del computador.

6. Entendre el funcionament d'una unitat de control microprogramada per a una arquitectura concreta.

1. Organització del computador

El computador s'organitza en unitats funcionals que treballen independentment i que estan interconnectades per línies, habitualment anomenades *busos*. Això ens permet descriure el comportament funcional del computador, que dóna lloc a les especificacions de l'arquitectura del computador.

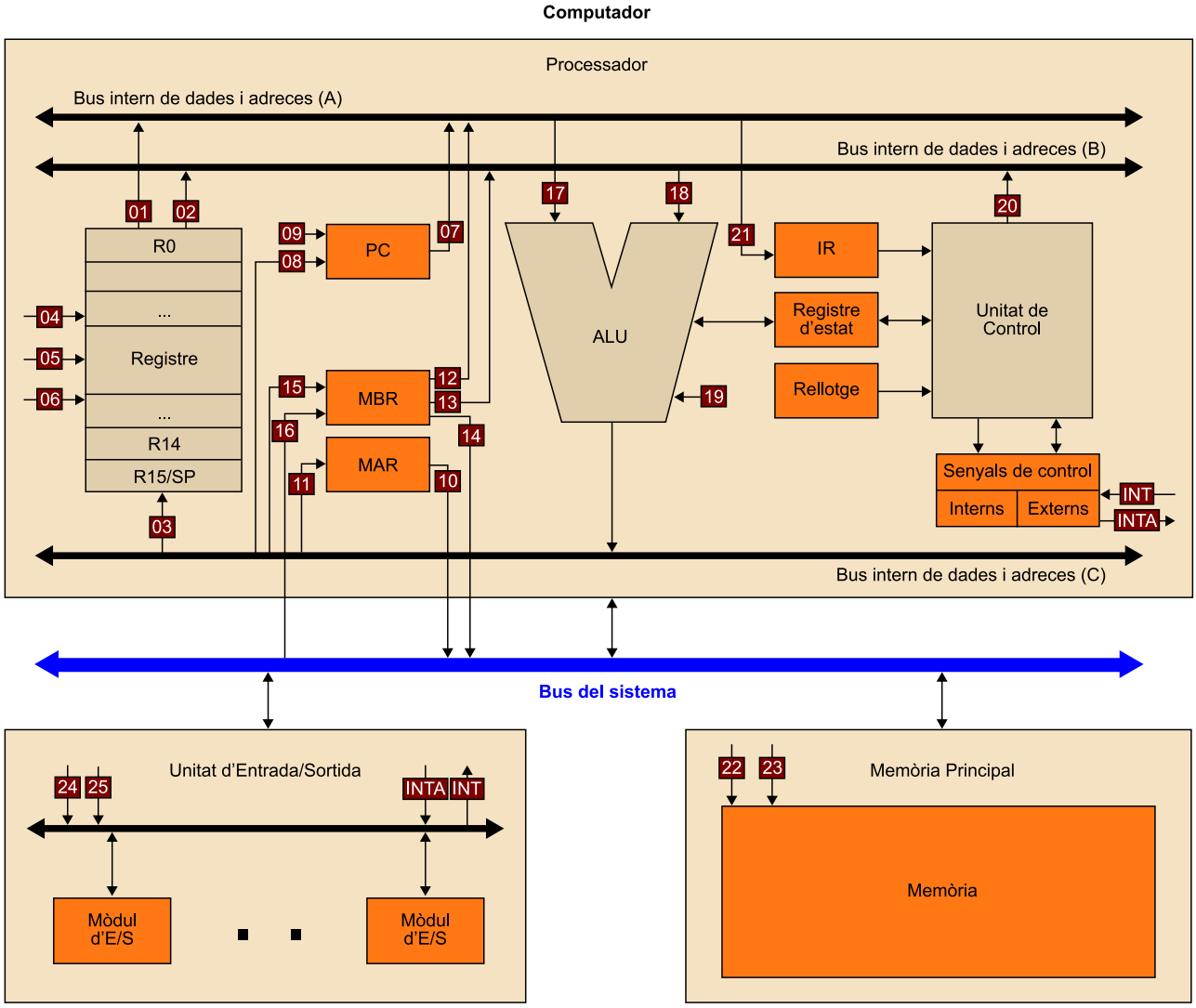
Tal com es veu a la figura següent, les unitats funcionals principals d'un computador són:

- Processador (CPU)
  - Registres
  - Unitat aritmètica i lògica (ALU)
  - Unitat de control (UC)

- Unitat de memòria principal (Mp)

- Unitat d'entrada/sortida (E/S)

- Sistema d'interconnexió (bus)



1.1. Processador

1.1.1. Organització dels registres

Tots els registres són de 32 bits i els bits de cada registre es numeren del 31 (bit de més pes) al 0 (bit de menys pes). Els registres del processador (CPU) són de quatre tipus:

1) **Registres de propòsit general**. Hi ha 16 registres de propòsit general, de R0 a R15. El registre R15 és especial; s'utilitza de manera implícita en les instruccions PUSH, POP, CALL i RET, però també es pot utilitzar com a registre de propòsit general.

Registre R15

El registre R15 es pot anomenar també StackPointer (SP).

2) Registres d'instrucció. Els dos registres principals relacionats amb l'accés a les instruccions són el comptador de programa (PC) i el registre d'instrucció (IR).

Registres visibles al programador

Els registres de propòsit general són els únics registres visibles al programador, la resta de registres que s'expliquen a continuació no són visibles.

El registre PC tindrà un circuit autoincrementador. Dins del cicle d'execució de la instrucció, en la fase de lectura de la instrucció, el PC quedarà incrementat en tantes unitats com bytes tingui la instrucció. El valor del PC a partir d'aquest moment, i durant la resta de fases de l'execució de la instrucció, es denota com a PC_{up} (PC *updated*) i apunta a l'adreça de la instrucció següent en la seqüència.

3) Registres d'accés a memòria. Hi ha dos registres necessaris per a qualsevol operació de lectura o escriptura en memòria: el Registre de Dades de la Memòria (MBR) i el Registre d'Adreces de la Memòria (MAR).

4) Registres d'estat i de control. Els bits del registre d'estat són modificats pel processador com a resultat de l'execució d'instruccions aritmètiques o lògiques. Aquests bits són parcialment visibles al programador, mitjançant les instruccions de salt condicional.

El registre d'estat inclou els bits de resultat de zero, transport, sobreeiximent i signe.

Bits de resultat

Bits de resultat actius

Considerem que els bits de resultat són actius quan valen 1, i inactius quan valen 0.

- **Bit de zero (Z)**: s'activa si el resultat obtingut és zero.
- **Bit de transport (C)**: també anomenat *carry* a la suma i *borrow* a la resta. S'activa si en el darrer bit que operem en una operació aritmètica es produeix transport. S'activa si al final de l'operació en portem una segons l'algorisme de suma i resta tradicional operand en binari.

  Bit de transport a la resta

  El bit de transport, també anomenat *borrow*, es genera segons l'algorisme convencional de la resta. Però si fem la resta A – B sumant-hi el complementari del subtrahend, A + (– B), el bit de transport de la resta (*borrow*) serà el bit de transport (*carry*) negat obtingut de fer la suma amb el complementari (*borrow* = *carry negat*), llevat dels casos en què B = 0, en què aleshores tant el *carry* com el *borrow* són iguals i valen 0.

- **Bit de sobreeiximent (V)**: també anomenat *overflow*. S'activa si l'última operació ha produït un sobreeiximent segons el rang de representació utilitzat. Per a representar el resultat obtingut, en el format de complement a 2 amb 32 bits, necessitaríem més bits dels disponibles.
- **Bit de signe (S)**: actiu si el resultat obtingut és negatiu. Si el bit més significatiu del resultat és 1.
- **Bit per a habilitar les interrupcions (IE)**: si està actiu permet les interrupcions, si està inactiu no es permeten les interrupcions.

- **Bit d'interrupció (IF)**: si hi ha una petició d'interrupció, s'activa.

1.1.2. Unitat aritmètica i lògica

La unitat aritmètica i lògica (ALU) és l'encarregada de fer les operacions aritmètiques i les operacions lògiques, considerant nombres de 32 bits en complement a 2 (Ca2). Per a fer una operació, la unitat agafarà els operands font del bus intern A i B i dipositarà el resultat en el bus intern C.

Quan s'executa una instrucció que fa una operació aritmètica o lògica, la unitat de control haurà de determinar quina operació fa l'ALU, però també quin registre diposita la dada al bus A, quin registre al bus B i a quin registre s'emmagatzemarà el resultat generat sobre el bus C.

Cada una de les operacions que farà l'ALU es pot implementar de diferents maneres i no analitzarem l'estructura de cada mòdul, sinó que ens centrarem solament en la part funcional descrita en les instruccions d'aquesta arquitectura.

1.1.3. Unitat de control

La unitat de control (UC) és la unitat encarregada de coordinar la resta de components del computador per mitjà dels senyals de control.

Aquesta arquitectura disposa d'una unitat de control microprogramada, la funció bàsica de la qual és coordinar l'execució de les instruccions, amb la determinació de les operacions (anomenades *microoperacions*) que es fan i quan es fan i amb l'activació dels *senyals de control* necessaris en cada moment.

Els tipus de senyals que tindrem a la unitat de control són:

1) Senyals d'entrada

a) Temporització
b) Registre d'instrucció (IR)
c) Registre d'estat
d) Senyals externs de la CPU

2) Senyals de sortida i de control

a) Interns a la CPU:

- Accés als busos interns

- Control de l'ALU
- Control d'altres elements de la CPU

b) Externs a la CPU:

- Accés al bus extern
- Control de la memòria
- Control dels mòduls d'E/S

La taula següent mostra els senyals més importants per al control del computador en aquesta arquitectura.

	Senyal	Observacions	Dispositius afectats
01	$R_{outAenable}$		Banc de registres
02	$R_{outBenable}$		
03	$R_{inCenable}$		
04	Ri_{outA}	Són 16 * 3 = 48 senyals. Un senyal de cada tipus i per a cada registre.	
05	Ri_{outB}		
06	Ri_{inC}		
07	PC_{outA}		Registre PC
08	PC_{inC}		
09	$PC_{+\Delta}$	Per a indicar increment del registre PC (0-4).	
10	MAR_{outEXT}		Registre MAR
11	MAR_{inC}		
12	MBR_{outA}		Registre MBR
13	MBR_{outB}		
14	MBR_{outEXT}		
15	MBR_{inC}		
16	MBR_{inEXT}		
17	ALU_{inA}		ALU
18	ALU_{inB}		
19	ALU_{op}	Senyals que codifiquen les operacions de l'ALU. 4 bits → 2^4 = 16 operacions diferents	
20	IR_{outB}	Por a poder posar els valors immediats de la instrucció al bus intern B	Registre IR
21	IR_{inA}		
22	Read		Memòria

	Senyal	Observacions	Dispositius afectats
23	Write		
24	Read E/S		Sistema d'E/S
25	Write E/S		

1.2. Memòria principal

Hi ha 2^{32} posicions de memòria d'un byte cadascuna (4 GB de memòria). Les dades s'accedeixen sempre en paraules de 32 bits (4 bytes). L'ordre dels bytes en una dada de 4 bytes és en format *little-endian*, és a dir, el byte de menys pes s'emmagatzema a l'adreça de memòria més petita de les 4.

Format *little-endian*

Volem desar el valor 12345678h a l'adreça de memòria 00120034h amb la següent instrucció:

MOV [00120034h], 12345678h

Com que aquest valor és de 32 bits (4 bytes) i les adreces de memòria són de 1 byte, necessitarem 4 posicions de memòria per a emmagatzemar-lo, a partir de l'adreça especificada (00120034h). Si el desem en format *little-endian* quedarà emmagatzemat a la memòria de la següent forma:

Memòria	
Adreça	**Contingut**
00120034h	78h
00120035h	56h
00120036h	34h
00120037h	12h

1.2.1. Memòria per a la pila

Es reserva per a la pila una part de la memòria principal (Mp), des de l'adreça FFFF0000h a l'adreça FFFFFFFFh, de manera que es disposa d'una pila de 64 KB. La mida de cada dada que s'emmagatzema a la pila és de 32 bits (4 bytes).

El registre SP (registre R15) apunta sempre al cim de la pila. La pila creix cap a adreces petites. Per a posar un element a la pila, primer decrementarem el registre SP i després desarem la dada a l'adreça de memòria que indiqui el registre SP; si volem treure un element de la pila, primer llegirem la dada de l'adreça de memòria que indica el registre SP i després incrementarem el registre SP.

El valor inicial del registre SP és 0, això ens indicarà que la pila és buida. En posar el primer element a la pila, el registre SP es decrementa en 4 unitats (mida de la paraula de pila) abans d'introduir la dada. D'aquesta manera, en posar

la primera dada a la pila, quedarà emmagatzemada a les adreces FFFFFFFCh – FFFFFFFFh en format *little-endian* i el punter SP valdrà FFFFFFFCh (= 0 – 4 en Ca2 utilitzant 32 bits); la segona dada anirà a les adreces FFFFFFF8h – FFFFFFFBh i SP valdrà FFFFFFF8h, i així successivament.

Memòria principal (4 GB)	
Adreça	**Contingut**
00000000h	Taula de vectors d'interrupció (256 bytes)
...	
00000FFh	
0000100h	Codi i dades
...	
FFFEFFFFh	
FFFF0000h	Pila (64 KB)
...	
FFFFFFFFh	

1.2.2. Memòria per a la taula de vectors d'interrupció

Es reserva per a la taula de vectors una part de la memòria principal, des de l'adreça 00000000h a l'adreça 000000FFh, de manera que disposem de 256 bytes per a la taula de vectors d'interrupció. A cada posició de la taula hi emmagatzemarem una adreça de memòria de 32 bits (4 bytes), adreça d'inici de cada RSI, i podem emmagatzemar fins a 64 (256/4) adreces diferents.

1.3. Unitat d'entrada/sortida (E/S)

La memòria d'E/S disposa de 2^{32} ports d'E/S. Cada port correspon a un registre de 32 bits (4 bytes) ubicat a un dels mòduls d'E/S i cada mòdul d'E/S pot tenir assignats diferents registres d'E/S. Podem utilitzar tots els modes d'adreçament disponibles per a accedir als ports d'E/S.

1.4. Sistema d'interconnexió (bus)

En aquest sistema disposarem de dos nivells de busos: els busos interns del processador per a interconnectar els elements de l'interior del processador i el bus del sistema per a interconnectar el processador, la memòria i el sistema d'E/S:

- Bus intern del processador (tindrem 3 busos de 32 bits, que podrem utilitzar tant per a dades com per a adreces).

- Bus extern del processador (tindrem 1 bus de 32 bits, que el podrem utilitzar tant per a dades com per a adreces).

- Línies de comunicació o d'E/S (tindrem 1 bus de 32 bits que el podrem utilitzar tant per a dades com per a adreces).

2. Joc d'instruccions

Tot i que el joc d'instruccions d'aquesta arquitectura té poques instruccions, segueix moltes de les característiques d'una arquitectura CISC, com ara instruccions de longitud variable, operand destinació implícit igual al primer operand font explícit, possibilitat de fer operacions aritmètiques i lògiques amb operands en memòria, etc.

Nota

El joc d'instruccions d'aquesta arquitectura té poques instruccions per tal que l'arquitectura sigui pedagògica i senzilla.

2.1. Operands

Les instruccions poden ser de 0, 1 o 2 operands explícits, en ser una arquitectura amb un model registre-memòria. En les instruccions amb dos operands explícits, només un dels dos operands pot fer referència a la memòria, l'altre serà un registre o un valor immediat. En les instruccions d'un operand, aquest operand pot fer referència a un registre o a memòria.

Podem tenir dos tipus d'operands:

1) Adreces: Valors enters sense signe [0 .. $(2^{32} - 1)$]; es codificaran utilitzant 32 bits.

2) Nombres: Valors enters amb signe [-2^{31} .. $+(2^{31} - 1)$]; es codifiquen utilitzant Ca2 en 32 bits, un valor serà negatiu si el bit de signe (bit de més pes, bit 31) és 1 i positiu en cas contrari.

2.2. Modes d'adreçament

Els modes d'adreçament que suporta CISCA són els següents:

1) Immediat. La dada està en la instrucció mateixa.

L'operand s'expressa indicant:

- Un número decimal; es pot expressar un valor negatiu afegint el signe '-' al seu davant.
- Un número binari finalitzat amb la lletra 'b'.
- Un número hexadecimal finalitzat amb la lletra 'h'.
- Una etiqueta; el nom de l'etiqueta va sense claudàtors.
- Una expressió aritmètica.

Expressions aritmètiques

En els diferents modes d'adreçament es poden expressar adreces, valors immediats i desplaçaments com a expressions aritmètiques formades per etiquetes, valors numèrics i operadors (+ - * /). Però cal tenir present que el valor que representa aquesta expressió

s'ha de poder codificar en 32 bits si és una adreça o un immediat i en 16 bits si és un desplaçament. L'expressió s'ha d'escriure entre parèntesis. Per exemple:

```
MOV R1, ((00100FF0h+16)*4)
MOV R2, [Vec+(10*4)]
MOV [R8+(25*4)],R1
```

Quan s'expressa un número, no es fa extensió de signe, es completa el número amb zeros. Per aquest motiu, en binari i hexadecimal, els números negatius s'han d'expressar en Ca2 utilitzant 32 bits (32 dígits binaris o 8 dígits hexadecimals respectivament).

L'etiqueta, sense claudàtors, especifica una adreça que s'ha de codificar utilitzant 32 bits, tant si representa el nom d'una variable com el punt del codi on volem anar. Llevat de les instruccions de salt condicional, que l'etiqueta es codifica com un desplaçament i utilitzen adreçament relatiu a PC, com veurem més endavant.

```
MOV R1, 100         ; Carrega el valor 100 (00000064h) en el registre R1.
MOV R1, -100        ; Carrega el valor -100 (FFFFFF9Ch) en el registre R1.
MOV R1, FF9Ch.      ; Carrega el valor F9Ch (0000FF9Ch) en el registre R1.
MOV R1, FFFFFF9Ch   ; És equivalent a la instrucció on carreguem -100 a R1.
MOV R1, 1001 1100b. ; Carrega el valor 9Ch (0000009Ch) en el registre R1.

MOV R1 1111 1111 1111 1111 1111 1111 1001 1100b ; Carrega el valor FFFFFF9Ch en el
                    ; registre R1, és equivalent a la instrucció on carreguem -100 a R1.
MOV R1, var1;       ; Carrega l'adreça, no el valor que conté la variable, en el registre R1.
JMP bucle           ; Carrega l'adreça que correspon al punt de codi on hem posat
                    ; l'etiqueta bucle en el registre PC.
MOV R1, ((00100FF0h+16)*4) ; Carrega a R1 el valor 00404000h.
```

2) Registre. La dada està emmagatzemada en un registre.

L'operand s'expressa indicant el nom d'un registre: R*i*.

```
INC R2     ; El contingut del registre R2 s'incrementa en una unitat.
ADD R2, R3 ; Se suma el contingut del registre R3 al registre R2: R2 = R2 + R3.
```

3) Memòria. La dada està emmagatzemada a la memòria.

L'operand s'expressa indicant l'adreça de memòria on és la dada, una etiqueta com a nom d'una variable o, de forma més genèrica, una expressió aritmètica entre claudàtors: [adreça] [nom_variable] [(expressió)].

```
MOV R1, [C0010020h]  ; Com que cada adreça de memòria es correspon a 1 byte i
                     ; l'operand destinació, R1, és de 4 bytes, els valors emmagatzemats
                     ; a les posicions C0010020h - C0010023h es mouen cap a R1.
MOV R1, [var1];      ; Carrega el contingut de la variable var1 en el registre R1.
```

```
MOV R2, [Vec+(10*4)] ; Carrega el contingut de l'adreça Vec+40 en el registre R2.
```

4) Indirecte. La dada està emmagatzemada a la memòria.

L'operand s'expressa indicant un registre que conté l'adreça de memòria on és l'operand: [Registre].

```
MOV R3, var1   ; Carrega l'adreça, no el valor que conté, en el registre R3.
MOV R1, [R3]   ; R3 conté l'adreça de la posició de memòria de la dada que s'ha de carregar
               ; en el registre R1. R1 = M(R3). Com que a R3 hem carregat l'adreça de var1,
MOV R1, [var1] ; és equivalent a carregar el contingut de la variable var1 en el registre R1.
```

Nota

var1: adreçament immediat; [R3] adreçament indirecte; [var1] adreçament a memòria.

5) Relatiu. La dada està emmagatzemada a la memòria.

L'adreça on és l'operand es determina sumant el valor del registre i el desplaçament indicat de 16 bits. En assemblador s'indica utilitzant [Registre + Desplaçament]. El desplaçament es pot escriure com una expressió aritmètica.

```
MOV R1,[R2 + 100]    ; Si R2 = 1200, l'operand és M(1200 + 100),
                     ; és a dir, la dada és a l'adreça de memòria M(1300).
MOV [R2+(25*4)], R8  ; Carrega el valor del registre R8 a l'adreça de memòria M(1200+100).
```

6) Indexat. La dada està emmagatzemada a la memòria.

L'adreça on és l'operand es determina sumant l'adreça de memòria indicada de 32 bits i el valor del registre. En assemblador s'indica utilitzant: [adreça + registre] [nom_variable+registre] [(expressió)+Registre]

```
MOV R1, [BC000020h + R5]          ; Si R5 = 1Bh, l'operand és a l'adreça de memòria M(BC00003Bh).
MOV R1, [vector + R3]             ; Si R3 = 08h i l'adreça de vector=AF00330Ah, el valor que
                                  ; carreguem a R1 és a l'adreça de memòria M(AF003312h).
MOV R1, [(vector+00100200h) + R3] ; Carreguem a R1 el valor que es troba a M(AF103512h).
```

7) Relatiu a PC. Aquest mode d'adreçament només s'utilitza en les instruccions de salt condicional.

L'operand s'expressa indicant l'adreça de memòria on es vol fer el salt, una etiqueta que indiqui una posició dins del codi o, de forma més genèrica, una expressió aritmètica: etiqueta o (expressió).

```
JE etiqueta ; Es carrega al PC l'adreça de la instrucció indicada per l'etiqueta.
```

8) A pila. L'adreçament a pila és un mode d'adreçament implícit; és a dir, no cal fer una referència explícita a la pila, sinó que treballa implícitament amb el cim de la pila a través del registre SP (R15).

En ser un mode d'adreçament implícit, només s'utilitza en les instruccions PUSH (posar un element a la pila) i POP (treure un element de la pila).

La instrucció **PUSH font** fa el següent:

SP = SP – 4

M[SP] = font

La instrucció **POP destinació** fa el següent:

destinació = M[SP]

SP = SP + 4

2.3. Instruccions

2.3.1. Instruccions de transferència de dades

- **MOV destinació, font**. Mou la dada a què fa referència l'operand font a la ubicació especificada per l'operand destinació (destinació ← font).

- **PUSH font**. Emmagatzema l'operand font (que representa una dada de 32 bits) al cim de la pila. Primer decrementa SP (registre R15) en 4 unitats i, a continuació, desa la dada a què fa referència l'operand font en la posició de la pila apuntada per SP.

- **POP destinació**. Recupera sobre l'operand destinació el valor emmagatzemat al cim de la pila (que representa una dada de 32 bits). Recupera el contingut de la posició de la pila a què apunta SP (registre R15) i ho desa on indica l'operand destinació, després incrementa SP en 4 unitats.

2.3.2. Instruccions aritmètiques

Les instruccions aritmètiques i de comparació operen considerant els operands i el resultat com a enters de 32 bits en Ca2. Activen els bits de resultat segons el resultat obtingut. Aquests bits de resultat els podrem consultar utilitzant les instruccions de salt condicional.

- **ADD destinació, font**. Fa l'operació destinació = destinació + font.

- **SUB destinació, font**. Fa l'operació destinació = destinació – font.

- **MUL destinació, font**. Fa l'operació destinació = destinació * font. Si el resultat no es pot representar en 32 bits, s'activa el bit de sobreeiximent.

- **DIV destinació, font**. Fa l'operació destinació/font, divisió entera que considera el residu amb el mateix signe que el dividend. El quocient es desa a destinació i el residu es desa a R0. Només es produeix sobreeiximent en el cas $-2^{31}/-1$. Si font = 0 no es pot fer la divisió, i per a indicar-ho s'activa el bit de transport (en els processadors reals aquest error genera una excepció que gestiona el sistema operatiu).

- **INC destinació**. Fa l'operació destinació = destinació + 1.

- **DEC destinació**. Fa l'operació destinació = destinació – 1.

- **CMP destinació, font**. Compara els dos operands mitjançant una resta: destinació – font, i actualitza els bits de resultat. Els operands no es modifiquen i el resultat no es desa.

- **NEG destinació**. Fa l'operació destinació = 0 – destinació.

Bits de resultat

Totes les instruccions aritmètiques poden modificar els bits de resultat del registre d'estat segons el resultat obtingut.

Instrucció	Z	S	C	V
ADD	x	x	x	x
SUB	x	x	x	x
MUL	x	x	-	x
DIV	x	x	x	x
INC	x	x	x	x
DEC	x	x	x	x
CMP	x	x	x	X
NEG	x	x	x	x

Notació: x significa que la instrucció modifica el bit de resultat; - significa que la instrucció no modifica aquest bit; 0 indica que la instrucció posa aquest bit a 0.

Exemple

En aquest exemple es mostra com funcionen els bits de resultat d'acord amb les especificacions de l'arquitectura CISCA, però utilitzant registres de 4 bits en lloc de registres de 32 bits per a facilitar-ne els càlculs. Considerem els operands R1 i R2 registres de 4 bits que representen valors numèrics en complement a 2 (rang de valors des de –8 fins a +7). El valor inicial de tots els bits de resultat per cada instrucció és 0.

	Resultat	R1 = 7, R2 = 1				Resultat	R1 = –1, R2 = –7			
		Z	S	C	V		Z	S	C	V
ADD R1, R2	R1 = –8	0	1	0	1	R1 = –8	0	1	1	0
SUB R1, R2	R1 = +6	0	0	0	0	R1 = +6	0	0	0	0
SUB R2, R1	R2 = –6	0	1	1	0	R2 = –6	0	1	1	0
CMP R2, R1	R2 = 1	0	1	1	0	R2 = –7	0	1	1	0
NEG R1	R1 = –7	0	1	1	0	R1 = +1	0	0	1	0
INC R1	R1 = –8	0	1	0	1	R1 = 0	1	0	1	0
DEC R2	R2 = 0	1	0	0	0	R2 = –8	0	1	0	0

Valor decimal	Codificació en 4 bits i Ca2
+7	0111
+6	0110
+5	0101
+4	0100
+3	0011
+2	0010
+1	0001
0	0000
–1	1111
–2	1110
–3	1101
–4	1100
–5	1011
–6	1010
–7	1001
–8	1000

Tingueu en compte que per a obtenir el nombre negat d'un nombre en complement a 2 hem de negar el nombre bit a bit i sumar-hi 1. Per exemple:

- (+6) 0110, el neguem, 1001 i hi sumem 1, 1010 (–6).
- (–3) 1101, el neguem, 0010 i hi sumem 1, 0011 (+3).
- (+0) 0000, el neguem, 1111 i hi sumem 1, 0000 (–0). Queda igual.

- (–8) 1000, el neguem, 0111 i hi sumem 1, 1000 (–8). Queda igual, +8 no es pot representar en Ca2 utilitzant 4 bits.

Una manera ràpida d'obtenir el nombre negat manualment és negar a partir del primer 1.

2.3.3. Instruccions lògiques

Les instruccions lògiques operen bit a bit i el resultat que es produeix en un bit no afecta la resta. Activen els bits de resultat segons el resultat obtingut, bits que podrem consultar utilitzant les instruccions de salt condicional.

- **AND destinació, font**. Fa l'operació destinació = destinació AND font. Fa una 'i' lògica bit a bit.

- **OR destinació, font**. Fa l'operació destinació = destinació OR font. Fa una 'o' lògica bit a bit.

- **XOR destinació, font**. Fa l'operació destinació = destinació XOR font. Fa una 'o exclusiva' lògica bit a bit.

- **NOT destinació**. Fa la negació lògica bit a bit de l'operand destinació.

- **SAL destinació, font**. Fa un desplaçament aritmètic a l'esquerra dels bits de destinació, desplaça tants bits com indiqui font i omple els bits de menys pes amb 0. Es produeix sobreeiximent si el bit de més pes (bit 31) canvia de valor en finalitzar els desplaçaments. Els bits que es desplacen es perden.

- **SAR destinació, font**. Fa un desplaçament aritmètic a la dreta dels bits de destinació, desplaça tants bits com indiqui font. Conserva el bit de signe de destinació; és a dir, copia el bit de signe als bits de més pes. Els bits que es desplacen es perden.

- **TEST destinació, font**. Comparació lògica que realitza una operació lògica AND i actualitza els bits de resultat que correspongui segons el resultat generat, però sense desar el resultat. Els operands no es modifiquen i el resultat no es desa.

Bits de resultat

Totes les instruccions lògiques (excepte NOT) poden modificar els bits de resultat del registre d'estat segons el resultat obtingut.

Instrucció	Z	S	C	V
AND	x	x	0	0
OR	x	x	0	0
XOR	x	x	0	0

Notació: x significa que la instrucció modifica el bit de resultat; - significa que la instrucció no modifica aquest bit; 0 indica que la instrucció posa aquest bit a 0.

Instrucció	Z	S	C	V
NOT	-	-	-	-
SAL	x	x	-	x
SAR	x	x	-	0
TEST	x	x	0	0

Notació: x significa que la instrucció modifica el bit de resultat; - significa que la instrucció no modifica aquest bit; 0 indica que la instrucció posa aquest bit a 0.

Exemple

En aquest exemple es mostra com funcionen els bits de resultat d'acord amb les especificacions de l'arquitectura CISCA, però utilitzant registres de 4 bits en lloc de registres de 32 bits per a facilitar-ne els càlculs. Considerem els operands R1 i R2 registres de 4 bits que representen valors numèrics en complement a 2 (rang de valors des de –8 fins a +7). El valor inicial de tots els bits de resultat per cada instrucció és 0.

	Resultat	R1 = 0110, R2 = 0001				Resultat	R1 = 1111, R2 = 1001			
		Z	S	C	V		Z	S	C	V
AND R1, R2	R1 = 0000	1	0	0	0	R1 = 1001	0	1	0	0
OR R1, R2	R1 = 0111	0	0	0	0	R1 = 1111	0	1	0	0
XOR R1, R2	R1 = 0111	0	0	0	0	R1 = 0110	0	0	0	0
SAL R2,1	R2 = 0010	0	0	–	0	R2 = 0010	0	0	–	1
SAR R2,1	R2 = 0000	1	0	–	0	R2 = 1100	0	1	–	0

2.3.4. Instruccions de ruptura de seqüència

Podem distingir entre:

1) Salt incondicional

- **JMP etiqueta**. *etiqueta* indica l'adreça de memòria on es vol saltar, adreça que es carrega en el registre PC. La instrucció que s'executarà després de *JMP etiqueta* sempre és la instrucció indicada per *etiqueta* (JMP es una instrucció de ruptura de seqüència incondicional). El mode d'adreçament que utilitzem en aquesta instrucció és l'adreçament immediat.

Etiqueta

Per a especificar una etiqueta dins d'un programa en assemblador ho farem posant el nom de l'etiqueta seguit de ":". Per exemple:

eti3: JMP eti3

2) Salts condicionals. En les instruccions de salt condicional (JE, JNE, JL, JLE, JG, JGE) s'executarà la instrucció indicada per *etiqueta* si es compleix una condició; en cas contrari, continuarà la seqüència prevista. El mode d'adreçament que utilitzem en aquestes instruccions és l'adreçament relatiu a PC.

- **JE etiqueta** (*jump equal* - salta si igual). Si el bit Z és actiu, carrega en el PC l'adreça indicada per *etiqueta*; en cas contrari continua la seqüència prevista.

- **JNE etiqueta** (*jump not equal* - salta si diferent). Si el bit Z és inactiu, carrega en el PC l'adreça indicada per *etiqueta*; en cas contrari continua la seqüència prevista.

- **JL etiqueta** (*jump less* - salta si més petit). Si S ≠ V, carrega en el PC l'adreça indicada per *etiqueta*; en cas contrari continua la seqüència prevista.

- **JLE etiqueta** (*jump less or equal* - salta si més petit o igual). Si Z = 1 o S ≠ V, carrega en el PC l'adreça indicada per *etiqueta*; en cas contrari continua la seqüència prevista.

- **JG etiqueta** (*jump greater* - salta si més gran). Si Z = 0 i S = V, carrega en el PC l'adreça indicada per *etiqueta*; en cas contrari continua la seqüència prevista.

- **JGE etiqueta** (*jump greater or equal* - salta si més gran o igual). Si S = V, carrega en el PC l'adreça indicada per *etiqueta*; en cas contrari continua la seqüència prevista.

3) Crida i retorn de subrutina

- **CALL etiqueta** (crida la subrutina indicada per etiqueta). *etiqueta* és una adreça de memòria on comença la subrutina. Primer es decrementa SP en 4 unitats, s'emmagatzema en la pila el valor PC_{up} i el registre PC es carrega amb l'adreça expressada per l'etiqueta. El mode d'adreçament que utilitzem en aquestes instruccions és l'adreçament immediat.

- **RET** (retorn de subrutina). Recupera de la pila el valor del PC i incrementa SP en 4 unitats.

4) Crida al sistema i retorn de rutina de servei d'interrupció

- **INT servei** (interrupció de programari o crida a un servei del sistema operatiu). *servei* és un valor que identifica el servei sol·licitat. El mode d'adreçament que utilitzem en aquestes instruccions és l'adreçament immediat.

- **IRET** (retorn d'una rutina de servei d'interrupció). Recupera de la pila del sistema el valor del PC i el registre d'estat, el registre SP queda incrementat en 8 unitats.

2.3.5. Instruccions d'entrada/sortida

- **IN Ri, port**. Mou el contingut del port d'E/S especificat al registre Ri.
- **OUT port, Ri**. Mou el contingut del registre Ri al port d'E/S especificat.

Port

Port fa referència a un port d'entrada/sortida, a un registre d'un mòdul d'E/S. Per a accedir a un port podrem utilitzar els mateixos modes d'adreçament que per a accedir a memòria.

2.3.6. Instruccions especials

- **NOP**. No fa res. El processador passa el temps d'execució d'una instrucció sense fer res.
- **STI**. Habilita les interrupcions, activa (posa a 1) el bit IE del registre d'estat. Si les interrupcions no estan habilitades, el processador no admetrà peticions d'interrupció.
- **CLI**. Inhibeix les interrupcions, desactiva (posa a 0) el bit IE del registre d'estat.

2.4. Estructures de control

En aquest subapartat veurem com diferents estructures de control en llenguatges d'alt nivell, expressades en C, es poden traduir a llenguatge assemblador CISCA.

2.4.1. Estructura if

Estructura condicional expressada en C:

```
if (expressió) sentència;
```

Exemple

```
if (x>y) maxx = 1;
```

Es pot traduir a llenguatge assemblador CISCA de la manera següent:

```
if:   mov r1,[x]
      cmp r1,[y]
      jg set
      jmp endif
set:  mov [maxx],1
endif:
```

Es pot optimitzar el nombre d'instruccions del codi si s'utilitza la condició contrària a la que apareix en el codi C (`x<=y`)

```
if:   mov r1,[x]
      cmp r1, [y]
      jle endif    ;condició contrària
      mov [maxx],1
endif:
```

2.4.2. Estructura if-else

Estructura condicional expressada en C, incloent la condició alternativa:

```
if (expressió) sentència1;
   else sentència2;
```

Exemple

```
if (x>y) max = x;
   else max = y;
```

Es pot traduir a llenguatge assemblador CISCA de la manera següent:

```
if:   mov r1,[x]
      mov r2,[y]
      cmp r1,r2
      jg cert
      jmp else
cert: mov [max],r1
      jmp endif
else: mov [max],r2
endif:
```

Es pot optimitzar el nombre d'instruccions del codi si s'utilitza la condició contrària a la que apareix en el codi C (`x<=y`)

```
if:   mov r1,[x]
      mov r2,[y]
      cmp r1,r2
      jle else ;condició contrària
      mov [max],r1
      jmp endif
else: mov [max],r2
endif:
```

2.4.3. Estructura while

Estructura iterativa en C controlada per una condició expressada al principi:

```
while (expressió)
```

```
   sentència;
```

amb més d'una sentència:

```
while (expressió) {
   sentència1;
   sentència2;
}
```

Exemple

```
while(num > 0){
   i = i*num;
   num = num - 1;
}
resul = i;
```

Es pot traduir a llenguatge assemblador CISCA de la manera següent:

```
       mov r1,[i]
       mov r2,[num]
while: cmp r2,0
       jg cont
       jmp end_w
cont:  mul r1,r2
       sub r2,1
       jmp while
end_w: mov [resul],r1
       mov [i],r1
```

Es pot optimitzar el nombre d'instruccions del codi si s'utilitza la condició contrària a la que apareix en el codi C (`num<=0`)

```
       mov r1,[i]
       mov r2,[num]
while: cmp r2,0
       jle end_w    ;condició contrària
       mul r1,r2
       sub r2,1
       jmp while
end_w: mov [resul],r1
       mov [i],r1
```

2.4.4. Estructura do-while

Estructura iterativa en C controlada per una condició expressada al final:

```
do
   sentència;
```

```
while (expressió);
```

amb més d'una sentència:

```
do {
   sentència1;
   sentència2;
} while (expressió);
```

Exemple

```
do {
   sum = sum + i;
   i = i + 1
} while (i <= valor);
```

Es pot traduir a llenguatge assemblador CISCA de la manera següent:

```
    mov r1,[i]
do: add [sum],r1
    add r1,1
    cmp r1,[valor]
    jle do
    mov [i],r1
```

2.4.5. Estructura for

Estructura iterativa utilitzant l'ordre for:

```
for (expr1; expr2; expr3)
     sentència;
```

és equivalent a:

```
expr1;
while (expr2) {
   sentència;
   expr3;
}
```

on:

```
expr1: inicialització d'un comptador
expr2: condició de sortida de l'estructura iterativa
expr3: actualització del comptador
```

Exemple

```
for (i=0; i<=valor; i++)
   sum = sum + i;
```

Es pot traduir a llenguatge assemblador CISCA de la manera següent:

```
      mov r1,0
for:  cmp r1,[valor]
      jle incr
      jmp endf
incr: add [sum],r1
      add r1,1    ; equivalent a inc r1
      jmp for
endf: mov [i],r1
```

Es pot optimitzar el nombre d'instruccions del codi si s'utilitza la condició contrària a la que apareix en el codi C (`i>valor`)

```
      mov r1,0
for:  cmp r1,[valor]
      jg endf     ; condició contrària
      add [sum],r1
      add r1,1    ; equivalent a inc r1
      jmp for
endf: mov [i],r1
```

2.4.6. Estructura switch-case

Estructura de selecció expressada en C, que amplia el nombre d'opcions de l'estructura if-else, permetent múltiples opcions. S'utilitza una variable que es compara per igualtat successivament amb diferents valors constants, es permet indicar una o diverses sentències per cada cas, finalitzades per la sentència *break*, i especificar una o diverses sentències que s'executin en cas de no coincidir amb cap dels valors indicats, *default*.

```
switch (variable) {
  case valor1:   sentència1;
                 break;
  case valor2:   sentència2;
                 ...
                 break;
                 ...
  default:       sentènciaN;
}
```

Exemple

```
switch (var) {
  case 1:   a=a+b;
            break;
  case 2:   a=a-b;
            break;
  case 3:   a=a*b;
            break;
  default:  a=-a;
}
```

Es pot traduir a llenguatge assemblador CISCA de la manera següent:

```
switch:  mov r1,[var]
         cmp r1, 1
         jne case2
         mov r2, [b]
         add [a], r2
         jmp end_s
case2:   cmp r1, 2
         jne case3
         mov r2, [b]
         sub [a], r2
         jmp end_s
case3:   cmp r1, 3
         jne default
         mov r2, [b]
         mul [a], r2
         jmp end_s
default: neg [a]
end_s:
```

3. Format i codificació de les instruccions

Les instruccions d'aquesta arquitectura tenen un format de longitud variable, com sol succeir en totes les arquitectures CISC. Tenim instruccions de 1, 2, 3, 4, 5, 6, 7, 9 o 11 bytes, segons el tipus d'instrucció i els modes d'adreçament utilitzats.

Cadascun dels bytes que formen una instrucció els denotarem com a B0, B1, ... (Byte 0, Byte 1, ...). Si la instrucció està emmagatzemada a partir de l'adreça @ de memòria, B0 és el byte que es troba en l'adreça @; B1, en l'adreça @+1, etc. D'altra banda, els bits dins d'un byte es numeren del 7 al 0, essent el 0 el de menor pes. Denotem com a B*k*<*j*..*i*> amb *j* > *i* el camp del byte *k* format pels bits del *j* a l'*i*. Escriurem els valors que prenen cadascun dels bytes en hexadecimal.

Per a codificar una instrucció, primer codificarem el codi d'operació en el byte B0 i, a continuació, en els bytes següents (els que calguin), els operands de la instrucció amb el seu mode d'adreçament.

En les instruccions de dos operands, es codificarà un operand a continuació de l'altre en el mateix ordre que s'especifica en la instrucció. Cal recordar que, en ser una arquitectura registre-memòria, només un operand pot fer referència a memòria; d'aquesta manera tindrem les combinacions de modes d'adreçament que es mostren a la taula següent.

Observació

En les instruccions de dos operands, l'operand destinació no podrà utilitzar un adreçament immediat.

Operand destinació	Operand font
Registre (adreçament directe a registre)	Immediat
	Registre (adreçament directe a registre)
	Memòria (adreçament directe a memòria)
	Indirecte (adreçament indirecte a registre)
	Relatiu (adreçament relatiu a registre base)
	Indexat (adreçament relatiu a registre índex)
Memòria (adreçament directe a memòria)	Immediat
	Registre (adreçament directe a registre)
Indirecte (adreçament indirecte a registre)	Immediat
	Registre (adreçament directe a registre)

Operand destinació	Operand font
Relatiu (adreçament relatiu a registre base)	Immediat
	Registre (adreçament directe a registre)
Indexat (adreçament relatiu a registre índex)	Immediat
	Registre (adreçament directe a registre)

3.1. Codificació del codi d'operació. Byte B0

El byte B0 representa el codi d'operació de les instruccions. Aquesta arquitectura no utilitza la tècnica d'expansió de codi per al codi d'operació.

Codificació del byte B0

B0 (Codi d'operació)	Instrucció
Especials	
00h	NOP
01h	STI
02h	CLI
Transferència	
10h	MOV
11h	PUSH
12h	POP
Aritmètiques	
20h	ADD
21h	SUB
22h	MUL
23h	DIV
24h	INC
25h	DEC
26h	CMP
27h	NEG
Lògiques	
30h	AND
31h	OR
32h	XOR
33h	TEST

B0 (Codi d'operació)	Instrucció
34h	NOT
35h	SAL
36h	SAR
Ruptura de seqüència	
40h	JMP
41h	JE
42h	JNE
43h	JL
44h	JLE
45h	JG
46h	JGE
47h	CALL
48h	RET
49h	INT
4Ah	IRET
Entrada/sortida	
50h	IN
51h	OUT

3.2. Codificació dels operands. Bytes B1-B10

Per a codificar els operands, haurem d'especificar el mode d'adreçament si aquest no és implícit i la informació per a expressar directament una dada, l'adreça, o la referència a l'adreça on tenim la dada.

Per a codificar un operand podem necessitar 1, 3 o 5 bytes, que denotarem com a Bk, Bk+1, Bk+2, Bk+3, Bk+4.

En el primer byte (Bk) codificarem el mode d'adreçament (Bk<7..4>) i el número de registre si aquest mode d'adreçament utilitza un registre, o zeros si no utilitza un registre (Bk<3..0>).

Bk<7..4>	Mode d'adreçament
0h	Immediat
1h	Registre (adreçament directe a registre)

Bk<7..4>: mode d'adreçament
Bk<3..0>: 0 (no utilitza registres)
Bk+1, Bk+2, Bk+3, Bk+4: valor immediat de 32 bits en Ca2

Bk<7..4>	Mode d'adreçament
2h	Memòria (adreçament directe a memòria)
3h	Indirecte (adreçament indirecte a registre)
4h	Relatiu (adreçament relatiu a registre base)
5h	Indexat (adreçament relatiu a registre índex)
6h	A PC (adreçament relatiu a registre PC)
De 7h a Fh	Codis no utilitzats en la implementació actual. Queden lliures per a futures ampliacions del llenguatge.

Bk<7..4>: mode d'adreçament
Bk<3..0>: 0 (no utilitza registres)
Bk+1, Bk+2, Bk+3, Bk+4: valor immediat de 32 bits en Ca2

Quan haguem de codificar un valor immediat o una adreça on tenim la dada, ho farem utilitzant els bytes (Bk+1, Bk+2, Bk3, Bk+4) en format *little-endian*. Si hem de codificar un desplaçament utilitzarem els bytes (Bk+1, Bk+2) en format *little-endian*.

Expressions aritmètiques

Si s'ha expressat un operand utilitzant una expressió aritmètica, abans de codificar-lo s'haurà d'avaluar l'expressió i representar-la en el format corresponent: una adreça utilitzant 32 bits, un valor immediat utilitzant 32 bits en Ca2 i un desplaçament utilitzant 16 bits en Ca2.

Recordeu que en les instruccions amb dos operands explícits, només un dels dos operands pot fer referència a la memòria, l'altre serà un registre o un valor immediat. Cal codificar cada operand segons el mode d'adreçament que utilitzi:

1) Immediat

Format: número decimal, binari o hexadecimal o etiqueta

La dada es codifica en Ca2 utilitzant 32 bits en format *little-endian*. No es fa extensió de signe, es completa el nombre amb zeros; per aquest motiu, en binari i hexadecimal, els números negatius s'han d'expressar en Ca2 utilitzant 32 bits (32 dígits binaris o 8 dígits hexadecimals, respectivament).

A les instruccions de transferència en què s'especifica una etiqueta sense claudàtors que representa el nom d'una variable o el punt del codi on volem anar, es codifica una adreça de memòria de 32 bits (adreça de la variable o adreça on volem anar, respectivament). Atès que no hem de fer cap accés a memòria per a obtenir l'operand, considerarem que aquestes instruccions utilitzen una adreçament immediat.

Exemple

Sintaxi	Valor a codificar	Codificació operand					
		Bk<7..4>	Bk<3..0>	Bk+1	Bk+2	Bk+3	Bk+4
0	00000000h	0h	0h	00h	00h	00h	00h
100	00000064h	0h	0h	64h	00h	00h	00h
-100	FFFFFF9Ch	0h	0h	9Ch	FFh	FFh	FFh
156	0000009Ch	0h	0h	9Ch	00h	00h	00h
9Ch	0000009Ch	0h	0h	9Ch	00h	00h	00h
FF9Ch	0000FF9Ch	0h	0h	9Ch	FFh	00h	00h
FFFFFF9Ch	FFFFFF9Ch	0h	0h	9Ch	FFh	FFh	FFh
1001 1100b	0000009Ch	0h	0h	9Ch	00h	00h	00h
1111 1111 1111 1111 1111 1111 1001 1100b	FFFFFF9Ch	0h	0h	9Ch	FFh	FFh	FFh
var1 L'etiqueta *var1* val 00AB01E0h	00AB01E0h	0h	0h	E0h	01h	ABh	00h
bucle L'etiqueta *bucle* val 1FF00230h	1FF00230h	0h	0h	30h	02h	F0h	1Fh

Bk<7..4>: mode d'adreçament
Bk<3..0>: 0 (no utilitza registres)
Bk+1, Bk+2, Bk+3, Bk+4: valor immediat de 32 bits en Ca2

2) Registre (adreçament directe a registre)

Format: Ri

El registre es codifica utilitzant 4 bits.

Exemple

Sintaxi	Valor a codificar	Codificació operand	
		Bk<7..4>	Bk<3..0>
R0	0h	1h	0h
R10	Ah	1h	Ah

Bk<7..4>: mode d'adreçament
Bk<3..0>: número de registre

3) Memòria (adreçament directe a memòria)

Format: [adreça] o [nom_variable]

L'adreça de memòria es codifica utilitzant 32 bits en format *little-endian* (8 dígits hexadecimals), necessaris per a accedir als 4 GB de la memòria principal. Si posem nom_variable, per a poder fer la codificació cal conèixer a quina adreça de memòria fa referència.

Exemple

Sintaxi	Valor a codificar	Codificació operand					
		Bk<7..4>	Bk<3..0>	Bk+1	Bk+2	Bk+3	Bk+4
[00AB01E0h]	00AB01E0h	2h	0h	E0h	01h	ABh	00h
[var1] L'etiqueta *var1* val 00AB01E0h	00AB01E0h	2h	0h	E0h	01h	ABh	00h

Bk<7..4>: mode d'adreçament
Bk<3..0>: 0 (no utilitza registres)
Bk+1, Bk+2, Bk+3, Bk+4: adreça de 32 bits

4) Indirecte (adreçament indirecte a registre)

Format: [Ri]

El registre es codifica utilitzant 4 bits.

Exemple

Sintaxi	Valor a codificar	Codificació operand	
		Bk<7..4>	Bk<3..0>
[R0]	0h	3h	0h
[R10]	Ah	3h	Ah

Bk<7..4>: mode d'adreçament
Bk<3..0>: número de registre

5) Relatiu (adreçament relatiu a registre base)

Format: [Registre + Desplaçament]

El registre es codifica utilitzant 4 bits. El desplaçament es codifica en Ca2 utilitzant 16 bits en format *little-endian*. No es fa extensió de signe, es completa el número amb zeros; per aquest motiu, en hexadecimal els números negatius s'han d'expressar en Ca2 utilitzant 16 bits (4 dígits hexadecimals).

Exemple

Sintaxi	Valors a codificar	Codificació operand			
		Bk<7..4>	Bk<3..0>	Bk+1	Bk+2
[R0+8]	0h i 0008h	4h	0h	08h	00h
[R10+001Bh]	Ah i 001Bh	4h	Ah	1Bh	00h
[R11-4]	Bh i FFFCh	4h	Bh	FCh	FFh
[R3+FFFCh]	3h i FFFCh	4h	3h	FCh	FFh

Bk<7..4>: mode d'adreçament
Bk<3..0>: número de registre
Bk+1, Bk+2: desplaçament de 16 bits en Ca2

6) Indexat (adreçament relatiu a registre índex)

Format: [adreça + registre] [nom_variable + registre] o [(expressió) + Registre]

El registre es codifica utilitzant 4 bits. L'adreça de memòria es codifica utilitzant 32 bits en format *little-endian* (8 dígits hexadecimals), necessaris per a accedir als 4 GB de la memòria principal. Si posem nom_variable, per a poder fer la codificació cal conèixer a quina adreça de memòria fa referència.

Exemple

Sintaxi	Valors a codificar	Codificació operand					
		Bk<7..4>	Bk<3..0>	Bk+1	Bk+2	Bk+3	Bk+4
[0ABC0100h+R2]	0ABC0100h i 2h	5h	2h	00h	01h	BCh	0Ah
[vector1+R9] L'etiqueta *vector1* val 0ABC0100h	0ABC0100h i 9h	5h	9h	00h	01h	BCh	0Ah

Bk<7..4>: mode d'adreçament
Bk<3..0>: número de registre
Bk+1, Bk+2, Bk+3, Bk+4: adreça de 32 bits

7) A PC (adreçament relatiu a registre PC)

Format: etiqueta o (expressió).

En aquest mode d'adreçament no es codifica l'etiqueta, sinó que es codifica el nombre de bytes que cal desplaçar-se per a arribar a la posició de memòria indicada per l'etiqueta. Aquest desplaçament es codifica en Ca2 utilitzant 16 bits en format *little-endian* (4 dígits hexadecimals).

Per a determinar el desplaçament que hem de codificar (*desp16*) cal conèixer a quina adreça de memòria fa referència l'etiqueta especificada en la instrucció (*etiqueta*) i l'adreça de memòria de la instrucció següent (adreça del byte B0 de la instrucció següent, PC_{up}).

$$desp16 = etiqueta - PC_{up}$$

Si *etiqueta* < PC_{up}, llavors *desp16* serà negatiu; per tant farem un salt endarrere en el codi.

Si *etiqueta* ≥ PC_{up}, llavors *desp16* serà positiu, per tant farem un salt endavant en el codi.

Exemple

Sintaxi	Valor a codificar	Codificació operand			
		Bk<7..4>	Bk<3..0>	Bk+1	Bk+2
Inici L'adreça de l'etiqueta *Inici* val 0AB00030h i PC_{up} = 0AB00150h	FEE0h	6h	0h	E0h	FEh
Fi L'adreça de l'etiqueta *Fi* val 0AB00200h i PC_{up} = 0AB00150h	00B0h	6h	0h	B0h	00h

Bk<7..4>: mode d'adreçament
Bk<3..0>: 0 (no utilitza registres)
Bk+1, Bk+2: desplaçament de 16 bits en Ca2

3.3. Exemples de codificació

Vegem a continuació com codifiquem algunes instruccions.

1) PUSH R3

Codi d'operació: PUSH

- El camp B0 = 11h

Operand: R3 (adreçament a registre)

- El camp Bk: (Bk<7..4>) mode d'adreçament = 1h i (Bk<3..0>) registre = 3h

La codificació en hexadecimal d'aquesta instrucció serà:

B0	B1
11h	13h

2) JNE etiqueta

Codi d'operació: JNE

- El camp B0 = 42h

Operand: etiqueta (adreçament relatiu a PC)

Suposem que etiqueta té el valor 1FF000B4h i PC_{up} val 1FF00030h.

- El camp Bk: (Bk<7..4>) mode d'adreçament=6h i (Bk<3..0>) sense registre=0h
- El camp Bk+1, Bk+2: *desp16* = *etiqueta* - PC_{up}.

desp16 = 1FF000B4h - 1FF00030h = 0084h

1FF000B4h > 1FF00030h → desp16 serà positiu → salt endavant

La codificació en hexadecimal d'aquesta instrucció serà:

B0	B1	B2	B3
42h	60h	84h	00h

3) JL etiqueta

Codi d'operació: JL

- El camp B0 = 43h

Operand: etiqueta (adreçament relatiu a PC)

Suposem que etiqueta té el valor 1FF000A0h i PC_{up} val 1FF00110h.

- El camp Bk: (Bk<7..4>) mode d'adreçament = 6h i (Bk<3..0>) sense registre = 0h
- El camp Bk+1, Bk+2: *desp16* = *etiqueta* - PC_{up}.

desp16 = 1FF000A0h - 1FF00110h = FF90h

1FF000A0h <1FF00110h => desp16 és negatiu = FF90h (–112 decimal), per tant fem un salt endarrere en el codi.

La codificació en hexadecimal d'aquesta instrucció serà:

B0	B1	B2	B3
43h	60h	90h	FFh

4) NOT [0012005Bh]

Codi d'operació: NOT

- El camp B0 = 34h

Operand: [0012005Bh] (adreçament directe a memòria).

- El camp Bk: (Bk<7..4>) mode d'adreçament = 2h i (Bk<3..0>) sense registre = 0h

- El camp Bk+1, Bk+2, Bk+3, Bk+4: adreça de memòria codificada amb 32 bits en format *little-endian*.

La codificació en hexadecimal d'aquesta instrucció serà:

B0	B1	B2	B3	B4	B5
34h	20h	5Bh	00h	12h	00h

En la taula següent es mostra la codificació d'algunes instruccions. Suposem que totes les instruccions comencen a l'adreça @=1FF000B0h (no s'ha d'entendre com a programa sinó com a instruccions individuals). En els exemples, on calgui, *etiqueta1*=0012005Bh i *etiqueta2*=1FF00030h. El valor de cada un dels bytes de la instrucció amb adreces @ + *i* per a *i* = 0, 1, ... es mostra en hexadecimal a la taula (recordeu que els camps que codifiquen un desplaçament en 2 bytes, un valor immediat o una adreça en 4 bytes ho fan en format *little-endian*, aspecte que cal tenir en compte i escriure els bytes d'adreça més petita a l'esquerra i els d'adreça més gran a la dreta):

	Bk per a k=0..10										
Assemblador	**B0**	**B1**	**B2**	**B3**	**B4**	**B5**	**B6**	**B7**	**B8**	**B9**	**B10**
PUSH R3	11	1D									
JMP etiqueta2	40	00	30	00	F0	IF					
JNE etiqueta2	42	60	7Ch	FFh							
CALL etiqueta2	47	00	30	00	F0	1F					
NOT [etiqueta1]	34	20	5B	00	12	00					
DEC [R4]	25	34									
XOR [R13 + 3F4Ah], R12	32	4D	4A	3F	1C						
ADD [8A5165F7h + R1], -4	20	51	F7	65	51	8A	00	FC	FF	FF	FF
RET	48										
MOV [R4+32], 100	10	44	20	00	00	64	00	00	00		

En el byte B0 es codifica el codi d'operació i les caselles ombrejades són aquelles en què està codificat el mode d'adreçament (Bk<7..4>) i el número de registre, si aquest mode d'adreçament utilitza un registre, o zero si no n'utilitza cap (Bk<3..0>); per tant, ens indica on comença la codificació de cada operand de la instrucció.

A la instrucció JMP etiqueta2 que és una instrucció de salt incondicional i que utilitza adreçament immediat, es codifica l'adreça on es vol saltar utilitzant 4 bytes, en aquest cas etiqueta2 = 1FF00030h.

En canvi, a la instrucció JNE etiqueta2 que és una instrucció de salt condicional i que utilitza adreçament relatiu a PC, es codifica un desplaçament utilitzant 2 bytes, per a obtenir el desplaçament es resta PCup – etiqueta2.

En aquest cas, PCup = adreça de la instrucció + mida en bytes de la instrucció codificada = 1FF000B0h + 4 = 1FF000B4h, i etiqueta2 = 1FF00030h. D'aquesta forma PCup – etiqueta2 = 1FF000B4h – 1FF00030h = FFFFFF7Ch, però com només es consideren 2 bytes, el desplaçament serà FF7Ch.

4. Execució de les instruccions

L'execució d'una instrucció consisteix a realitzar el que anomenem un *cicle d'execució* i aquest cicle d'execució és una seqüència d'operacions que es divideixen en 4 fases principals:

1) Lectura de la instrucció
2) Lectura dels operands font
3) Execució de la instrucció i emmagatzematge de l'operand destinació
4) Comprovació d'interrupcions

Les operacions que realitza el processador en cada fase estan governades per la unitat de control i s'anomenen *microoperacions*. Per a determinar aquesta seqüència de microoperacions, la unitat de control haurà de descodificar la instrucció, és a dir, llegir i interpretar la informació que tindrem en el registre IR.

La nomenclatura que utilitzarem per a denotar les microoperacions serà la següent:

Registre destinació ← Registre origen

Registre destinació ← Registre origen <operació> Registre origen / Valor

Analitzem, doncs, la seqüència de microoperacions que es produeix habitualment en cada fase del cicle d'execució de les instruccions.

4.1. Lectura de la instrucció

Llegim la instrucció que volem executar. Aquesta fase consta bàsicament de 4 passos:

```
MAR ← PC, read ; Posem el contingut de PC al registre MAR.
MBR ← Memòria  ; Llegim la instrucció.
PC ← PC + Δ    ; Incrementem el PC en Δ unitats (Δ = mida de la instrucció en bytes)
IR ← MBR       ; Carreguem la instrucció al registre IR.
```

Com que en aquesta arquitectura s'accedeix a la memòria en paraules de 4 bytes, si la instrucció té una mida superior a una paraula de memòria (4 bytes), s'hauria de repetir el procés de lectura de memòria tants cops que fos necessari per a llegir tota la instrucció. Per a simplificar el funcionament de la unitat de

control es considerarà que es pot llegir tota la instrucció fent un únic accés a memòria, i per tant s'especificarà l'increment del PC (Δ) segons la mida en bytes de la instrucció.

La informació emmagatzemada al registre IR es descodificarà per a identificar les diferents parts de la instrucció i determinar, així, les operacions necessàries que caldrà realitzar en les fases següents.

4.2. Lectura dels operands font

Llegim els operands font de la instrucció. El nombre de passos que caldrà realitzar en aquesta fase dependrà del nombre d'operands font i dels modes d'adreçament utilitzats en cada operand.

El mode d'adreçament indicarà el lloc on és la dada, bé una adreça de memòria, bé un registre. Si és a memòria haurem de dur la dada al registre MBR, i si és en un registre no caldrà fer res, perquè ja la tindrem disponible en el processador. Com que aquesta arquitectura té un model registre-memòria, només un dels operands podrà fer referència a memòria.

Vegem ara com es resoldria això per a diferents modes d'adreçament:

1) Immediat: tenim la dada en la pròpia instrucció; no cal fer res.

2) Directe a registre: tenim la dada en un registre; no cal fer res.

3) Directe a memòria:

```
MAR ← IR(Adreça), read
MBR ← Memòria
```

IR(camp)

En IR(camp) considerarem que *camp* és un dels operands de la instrucció que acabem de llegir i que tenim desat en el registre IR.

4) Indirecte a registre:

```
MAR ← Contingut de IR(Registre), read
MBR ← Memòria
```

Read E/S

En la instrucció IN, l'operand font fa referència a un port d'E/S. En aquest cas, en lloc d'activar el senyal *read* caldrà activar el senyal *read E/S*.

5) Relatiu a registre índex:

```
MAR ← IR(Adreça operand) + Contingut de IR(Registre índex), read
MBR ← Memòria
```

6) Relatiu a registre base:

```
MAR ← Contingut de IR(Registre base) + IR(Desplaçament), read
MBR ← Memòria
```

7) Relatiu a PC:

No cal fer cap operació per a obtenir l'operand. Només farem el càlcul en el cas que calgui fer el salt, però es farà en la fase d'execució i emmagatzematge de l'operand destinació.

8) A pila:

S'utilitza de forma implícita el registre SP. És semblant al mode indirecte a registre, però l'accés a la pila es resoldrà en aquesta fase quan fem un POP (llegim dades de la pila) i es resoldrà en la fase d'execució i emmagatzematge de l'operand destinació quan fem un PUSH (desem dades a la pila).

```
MAR ← SP, read
MBR ← Memòria
SP ← SP + 4   ; 4 és la mida de la paraula de pila.
              ; '+', sumem perquè creix cap a adreces baixes.
```

4.3. Execució de la instrucció i emmagatzematge de l'operand destinació

Quan iniciem la fase d'execució i emmagatzematge de l'operand destinació, tindrem els operands font en registres del processador Ri o en el registre MBR si hem llegit l'operand de memòria.

Les operacions que haurem de realitzar dependran de la informació del codi d'operació de la instrucció i del mode d'adreçament utilitzat per a especificar l'operand destinació.

Un cop feta l'operació especificada, per a emmagatzemar l'operand destinació es poden donar els casos següents:

a) Si l'operand destinació és un registre, en fer l'operació ja deixarem el resultat en el registre especificat.

b) Si l'operand destinació fa referència a memòria, en fer l'operació deixarem el resultat en el registre MBR i després haurem d'emmagatzemar-lo en la memòria en l'adreça especificada pel registre MAR:

- Si l'operand destinació ja s'ha fet servir com a operand font (com passa en les instruccions aritmètiques i lògiques), encara tindrem l'adreça en el MAR.

- Si l'operand destinació no s'ha fet servir com a operand font (com passa en les instruccions de transferència i d'entrada/sortida) caldrà primer *resoldre el mode d'adreçament* com s'ha explicat anteriorment en la fase de lectura de l'operand font, deixant l'adreça de l'operand destinació en el registre MAR.

4.3.1. Operacions de transferència

1) MOV destinació, font

- Si l'operand destinació és un registre i l'operand font és un immediat:

```
Ri ← IR(valor Immediat)
```

- Si l'operand destinació és un registre i l'operand font també és un registre:

```
Ri ← Rj
```

- Si l'operand destinació és un registre i l'operand font fa referència a memòria:

```
Ri ← MBR
```

- Si l'operand destinació fa referència a memòria i l'operand font és un immediat:

```
MBR ← IR(valor Immediat)
(Resoldre mode d'adreçament),write
Memòria ← MBR
```

- Si l'operand destinació fa referència a memòria i l'operand font és un registre:

```
MBR ← Ri,
(Resoldre mode d'adreçament),write
Memòria ← MBR
```

2) PUSH font

- Si l'operand font és un registre, primer caldrà dur-lo al registre MBR:

```
MBR ← Ri
SP ← SP - 4 ; 4 és la mida de la paraula de pila.
; '-', restem perquè creix cap a adreces baixes.
MAR ← SP, write
Memòria ← MBR
```

- Si l'operand font fa referència a memòria ja serà en el registre MBR:

```
SP ← SP - 4 ; 4 és la mida de la paraula de pila.
; '-', restem perquè creix cap a adreces baixes.
MAR ← SP, write
Memòria ← MBR
```

3) POP destinació

- Si l'operand destinació és un registre:

```
Ri ← MBR
```

- Si l'operand destinació fa referència a memòria:

```
(Resoldre mode d'adreçament), write
Memòria ← MBR
```

4.3.2. Operacions aritmètiques i lògiques

1) ADD destinació, font; SUB destinació, font; MUL destinació, font; DIV destinació, font; CMP destinació, font; TEST destinació, font; SAL destinació, font; SAR destinació, font:

Operand destinació

Recordeu que l'operand destinació també és operand font.

- Si l'operand destinació és un registre i l'operand font és un immediat:

```
Ri ← Ri<operació>IR(valor Immediat)
```

- Si l'operand destinació és un registre i l'operand font també és un registre:

```
Ri ← Ri<operació>Ri
```

- Si l'operand destinació és un registre i l'operand font fa referència a memòria:

```
Ri ← Ri<operació>MBR
```

- Si l'operand destinació fa referència a memòria i l'operand font és un immediat:

```
MBR← MBR<operació>IR(valor Immediat), write
Memòria ← MBR
```

- Si l'operand destinació fa referència a memòria i l'operand font és un registre:

```
MBR ← MBR<operació>Ri, write
Memòria ← MBR
```

Per a totes les combinacions de modes d'adreçament cal un tractament especial per a la instrucció ***DIV destinació, font***, ja que genera dos resultats: el quocient i el residu.

Les microoperacions que hi ha especificades guarden el resultat de la operació en l'operand ***destinació***, per la instrucció DIV, aquest valor és el quocient de la divisió.

Cal afegir per a cada una de les combinacions de modes d'adreçament la microoperació següent que realitza l'operació *mòdul* que genera el residu de la divisió i el guarda en el registre R0:

R0 ← destinació <mod> font.

Exemple

Si estem executant la instrucció ADD R3, 7 la microoperació que s'executaria en aquesta fase seria:

R3 ← R3 + IR(valor immediat)=7

Si estem executant la instrucció DIV R1,[var] les microoperacions que s'executarien en aquesta fase serien:

R1 ← R1 / MBR
R0 ← R1 <mod> MBR

2) INC destinació; DEC destinació; NEG destinació; NOT destinació

- Si l'operand destinació és un registre:

```
Ri ← Ri<operació>
```

- Si l'operand destinació fa referència a memòria (ja tindrem l'adreça en el MAR):

```
MBR ← MBR<operació>, write
Memòria ← MBR
```

4.3.3. Operacions de ruptura de seqüència

1) JMP etiqueta

```
PC ← IR(Adreça)
```

2) JE etiqueta, JNE etiqueta, JL etiqueta, JLE etiqueta, JG etiqueta, JGE etiqueta:

```
PC ← PC + IR(Desplaçament)
```

3) CALL etiqueta:

```
MBR ← PC
SP ← SP - 4   ; 4 és la mida de la paraula de pila.
              ; '-', restem perquè creix cap a adreces baixes.
MAR ← SP, write
Memòria ← MBR ; Desem a la pila.
PC ← IR(Adreça)
```

4) RET

```
MAR ← SP, read
MBR ← Memòria
SP ← SP + 4   ; 4 és la mida de la paraula de pila.
              ; '+', sumem perquè creix cap a adreces baixes.
PC ← MBR      ; Restaurem el PC.
```

5) INT servei

```
MBR ← Registre d'Estat
```

```
SP ← SP - 4
MAR ← SP, write
Memòria ← MBR             ; Desem el registre d'estat a la pila.
MBR ← PC
SP ← SP - 4
MAR ← SP, write
Memòria ← MBR             ; Desem el PC a la pila.
MAR ← IR(servei)*4 , read ; 4 és la mida de cada adreça de la taula de vectors.
MBR ← Memòria             ; Llegim l'adreça de la RSI de la taula de vectors.
                          ; La taula de vectors comença a l'adreça 0 de memòria.
PC ← MBR                  ; Carreguem al PC l'adreça de la rutina de servei.
```

6) IRET

```
MAR ← SP, read
MBR ← Memòria
SP ← SP + 4             ; 4 és la mida de la paraula de pila.
                        ; '+', sumem perquè creix fins a adreces baixes.
PC ← MBR                ; Restaurem el PC.
MAR ← SP, read
MBR ← Memòria
SP ← SP + 4
Registre d'estat ← MBR  ; Restaurem el registre d'estat.
```

4.3.4. Operacions d'entrada/sortida

1) IN Ri, port

```
Ri ← MBR   ; Al registre MBR tindrem la dada llegida del port
           ; en el cicle de lectura de l'operand font.
```

2) OUT port, Ri

```
MBR ← Ri         ; Posem la dada que tenim al registre especificat
(Resoldre mode d'adreçament), write E/S
Memòria ← MBR    ; per a emmagatzemar com a operand destinació.
```

4.3.5. Operacions especials

1) NOP. No cal fer res (fa un cicle de no-operació de l'ALU).

2) STI. Activa el bit IE del registre d'estat per a habilitar les interrupcions.

3) CLI. Desactiva el bit IE del registre d'estat per a inhibir les interrupcions.

4.4. Comprovació d'interrupcions

En aquesta fase es comprova si s'ha produït una interrupció (per fer això no cal executar cap microoperació): si no s'ha produït cap interrupció es continua amb la següent instrucció, i si s'ha produït una interrupció s'ha de fer el canvi de context, en què cal desar certa informació i posar al PC l'adreça de la rutina que dóna servei a aquesta interrupció. Aquest procés pot variar molt d'una màquina a una altra; aquí només presentem la seqüència de microoperacions per a actualitzar el PC quan es produeix el canvi de context.

Bit d'interrupció (IF)

Bit IF actiu (IF=1): s'ha produït una interrupció.
Bit IF inactiu (IF=0): no s'ha produït una interrupció.

```
MBR ← Registre d'Estat
SP ← SP - 4
MAR ← SP, write
Memòria ← MBR              ; Desem el registre d'estat a la pila.
MBR ← PC
SP ← SP - 4
MAR ← SP, write
Memòria ← MBR              ; Desem el PC a la pila.
MAR ← (Índex RSI)*4,read   ; 4 és la mida de cada adreça de la taula de vectors.
MBR ← Memòria              ; Llegim l'adreça de la RSI de la taula de vectors.
                           ; La taula de vectors comença a l'adreça 0 de memòria.
PC ← MBR                   ; Carreguem al PC l'adreça de la rutina de servei.
```

4.5. Exemples de seqüències de microoperacions

Presentem a continuació diversos exemples de seqüències de microoperacions per a alguns casos concrets. En cada cas, s'indiquen les microoperacions que cal executar en cada fase i en quin ordre.

En cada seqüència, les fases s'identifiquen de la manera següent:

- Fase 1: Lectura de la instrucció.
- Fase 2: Lectura dels operands font.
- Fase 3: Execució de la instrucció i emmagatzematge de l'operand destinació.

1) MOV [R1+10], R3 ; adreçament relatiu i a registre

Fase	Microoperació
1	MAR ← PC, read MBR ← Memòria PC ← PC + 5 IR ← MBR
2	(no cal fer res, l'operand font és en un registre)

Fase	Microoperació
3	MBR ← R3 MAR ← R1 + 10, write Memòria ← MBR

2) PUSH R4 ; adreçament a pila

Fase	Microoperació
1	MAR ← PC, read MBR ← Memòria PC ← PC + 2 IR ← MBR
2	(no cal fer res, l'operand font és en un registre)
3	MBR ← R4 SP ← SP – 4 MAR ← SP,write Memòria ← MBR

3) ADD [R5], 8 ; adreçament indirecte a registre i immediat

Fase	Microoperació
1	MAR ← PC, read MBR ← Memòria PC ← PC + 7 IR ← MBR
2	MAR ← R5, read MBR ← Memòria
3	MBR ← MBR + 8, write Memòria ← MBR

4) SAL [00120034h+R3],R2 ; adreçament indexat i a registre

Fase	Microoperació
1	MAR ← PC, read MBR ← Memòria PC ← PC + 7 IR ← MBR
2	MAR ← 00120034h+R3, read MBR ← Memòria
3	MBR ← MBR <desp. esquerra> R2, write Memòria ← MBR

5) JE etiqueta ; adreçament relatiu a PC on *etiqueta* està codificada com un desplaçament

Fase	Microoperació
1	MAR ← PC, read MBR ← Memòria PC ← PC + 4 IR ← MBR
2	(no cal fer res, entenent etiqueta com a op. font)
3	Si bit de resultat Z=1 PC ← PC + etiqueta; sinó, no fer res

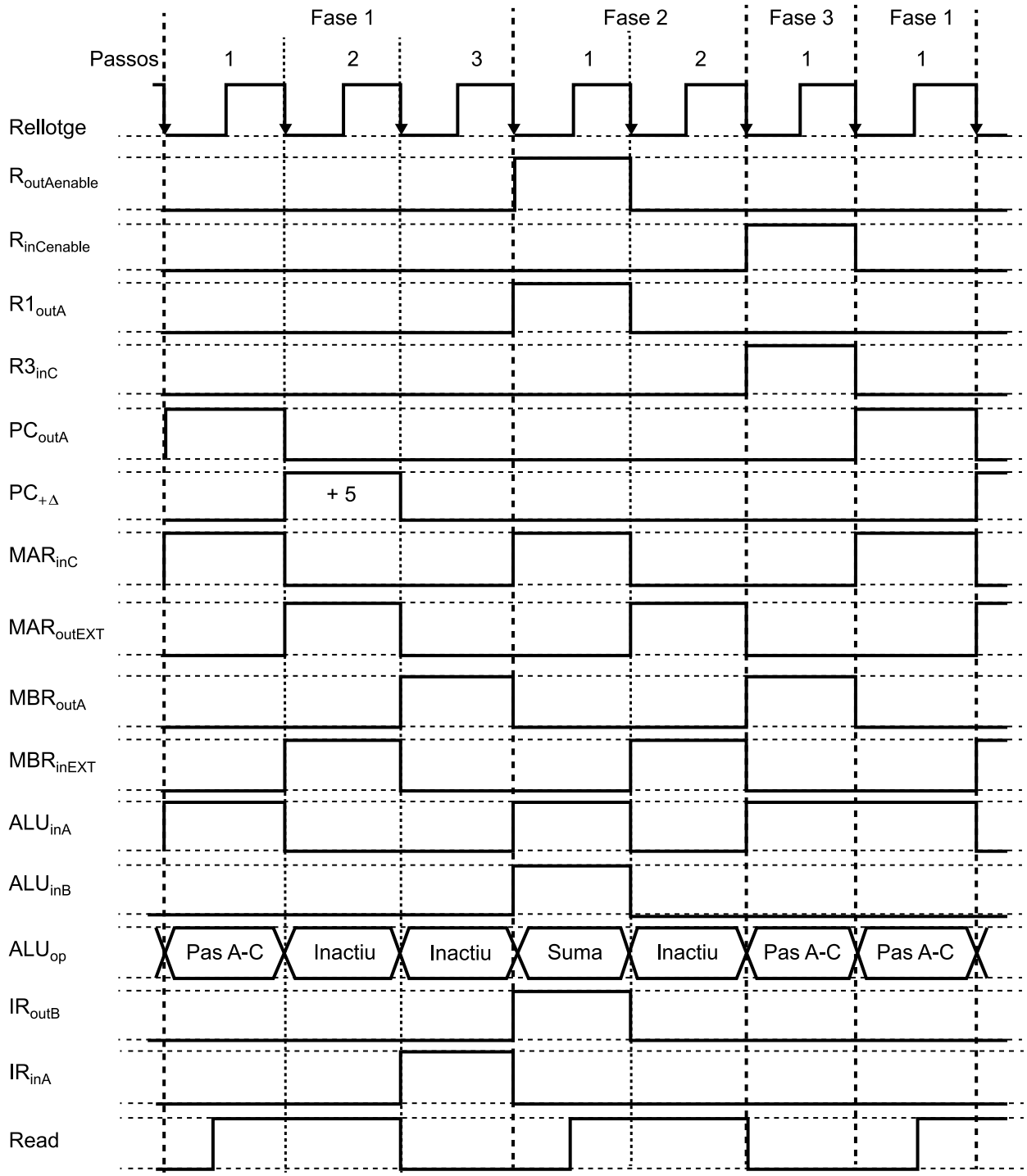
4.6. Exemple de senyals de control i temporització

Vegem el diagrama de temps de l'execució en CISCA d'una instrucció:

MOV R3, [R1+10]

L'operand destinació utilitza adreçament a registre i l'operand font, adreçament relatiu a registre base. Si codifiquem aquesta instrucció, veurem que ocupa 5 bytes.

Fase	Pas	Microoperació	
1	1	MAR ← PC, read	
	2	MBR ← Memòria	PC ← PC + 5
	3	IR ← MBR	
2	1	MAR ← R1 + 10, read	
	2	MBR ← Memòria	
3	1	R3 ← MBR	
1	1	MAR ← PC, read	Instrucció següent



F1.P1: MAR ← PC, read

Fem la transferència des del PC al MAR. Per a fer aquesta transferència connectem la sortida del PC al bus A activant el senyal PC_{outA} i fem passar el valor del PC al bus C a través de l'ALU, seleccionant l'operació Pas A-C i activant el senyal ALU_{inA} per a connectar l'entrada a l'ALU des del bus A, i connectem l'entrada del MAR al bus C activant el senyal MAR_{inC}.

També activem el senyal de lectura de la memòria Read, per a indicar a la memòria que iniciem un cicle de lectura.

F1.P2: MBR ← Memòria, PC ← PC + 5

Finalitzem la lectura de la memòria mantenint actiu el senyal de Read durant tot el cicle de rellotge i transferim la dada a l'MBR. Per a fer aquesta transferència s'activa el senyal MAR_{outEXT}, la memòria posa la dada d'aquesta adreça al bus del sistema i la fem entrar directament al registre MBR activant el senyal MBR_{inEXT}.

Com que ja tenim la instrucció a l'IR i l'hem començat a descodificar, en sabem la longitud. En aquest cas és 5; per tant, incrementem el PC en 5. Per a fer-ho utilitzem el circuit autoincrementador que té el registre PC, per mitjà de l'activació del senyal $PC_{+\Delta}$ indicant un increment de 5.

F1.P3: IR ← MBR

Transferim el valor llegit de la memòria que ja tenim a l'MBR a l'IR. Connectem la sortida de l'MBR al bus A activant el senyal MBR_{outA} i connectem l'entrada de l'IR al bus A activant el senyal IR_{inA}.

Ara ja hem llegit la instrucció. Amb això finalitza la fase de lectura de la instrucció i comença la fase de lectura dels operands font.

F2.P1: MAR ← R1 + 10, read

Calculem l'adreça de memòria on està emmagatzemat l'operand font. Com que és un adreçament relatiu a registre base, hem de sumar el contingut del registre base R1 amb el desplaçament, que val 10, que tenim a l'IR.

Connectem la sortida de R1 al bus A activant els senyals $R_{outAenable}$ per a connectar el banc de registres al bus i $R1_{outA}$ per a indicar que el registre que posarà la dada al bus A és l'R1. També hem de posar el 10 que tenim en el registre IR al bus B activant el senyal IR_{outB}. Amb això ja tenim les dades preparades; per a fer la suma a l'ALU seleccionarem l'operació de suma, activarem el senyal ALU_{inA} per a connectar l'entrada a l'ALU des del bus A i activarem el senyal ALU_{inB} per a connectar l'entrada a l'ALU des del bus B. El resultat quedarà al bus C, que podrem recollir connectant l'entrada del MAR al bus C activant el senyal MAR_{inC}.

També activem el senyal de lectura de la memòria Read.

F2.P2: MBR ← Memòria

Finalitzem la lectura de la memòria mantenint actiu el senyal de Read durant tot el cicle de rellotge i transferim la dada a l'MBR. Per a fer aquesta transferència s'activa el senyal MAR_{outEXT}, la memòria posa la dada d'aquesta adreça de memòria al bus extern i la fem entrar directament al registre MBR activant el senyal MBR_{inEXT}.

Finalitza així la fase de lectura dels operands font i comença la fase d'execució de la instrucció i emmagatzematge de l'operand destinació

F3.P1: R3 ← MBR

Transferim el valor llegit de la memòria que ja tenim a MBR cap a R3. Connectem la sortida de l'MBR al bus A activant el senyal MBR_{outA}, fem passar la dada al bus intern C a través de l'ALU seleccionant l'operació Pas A-C, activant el senyal ALU_{inA} per a connectar l'entrada a l'ALU des del bus A i activant el senyal $R_{inCenable}$ per a connectar el banc de registres al bus C i $R3_{inC}$ per a indicar que el registre que rep la dada del bus C és l'R3.

Amb això finalitza l'execució d'aquesta instrucció i ara començaria l'execució de la instrucció següent.