

ROM OUT (binària i hexa)



- Implementem les x amb '0'

@ROM	Bnz	Bz	WrMem	RdIn	WrOut	WrD	Ldlr	Byte	R@/Pc	Alu/R@	Pc/Rx	Ry/N	P/I/L/A1	P/I/L/A0	OP1	OP0	MxN1	MxN0	MxF	F2	F1	F0	Mx@D1	Mx@D0		
0	1	1	0	0	0	0	1	0	0	1	1	0	x	x	0	0	1	1	1	1	0	0	x	x	F	0xC260F0
1	0	0	0	0	0	0	0	x	x	x	1	0	x	x	0	0	1	0	1	1	0	0	x	x	D	0x0020B0
2	0	0	0	0	0	1	x	x	x	x	0	1	0	0	0	0	x	x	0	x	x	x	0	0	Al	0x041000
3	0	0	0	0	0	1	x	x	x	x	0	1	0	0	0	1	x	x	0	x	x	x	0	0	Cmp	0x041100
4	0	0	0	0	0	1	x	x	x	x	0	0	0	0	0	0	0	0	1	1	0	0	0	1	Addi	0x040031
5	0	0	0	0	0	0	0	x	x	x	0	0	x	x	0	0	0	0	1	1	0	0	x	x	Addr	0x000030
6	0	0	0	0	0	1	x	0	1	x	x	x	0	1	x	x	x	x	x	x	x	x	0	1	Ld	0x048401
7	0	0	1	0	0	0	x	0	1	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	St	0x208000
8	0	0	0	0	0	1	x	1	1	x	x	x	0	1	x	x	x	x	x	x	x	x	0	1	Ldb	0x058401
9	0	0	1	0	0	0	x	1	1	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	Stb	0x218000
10	1	1	0	0	0	1	x	x	x	1	0	x	1	1	1	0	x	x	1	0	1	1	0	1	Jalr	0xC44E2D
11	0	1	0	0	0	0	x	x	x	0	0	x	x	x	1	0	x	x	1	0	0	0	x	x	Bz	0x400220
12	1	0	0	0	0	0	x	x	x	0	0	x	x	x	1	0	x	x	1	0	0	0	x	x	Bnz	0x800220
13	0	0	0	0	0	1	x	x	x	x	x	0	0	0	1	0	0	1	1	0	0	1	1	0	Movi	0x040266
14	0	0	0	0	0	1	x	x	x	x	0	0	0	0	1	0	0	1	1	0	1	0	1	0	Movhi	0x04026A
15	0	0	0	1	0	1	x	x	x	x	x	x	1	0	x	x	x	x	x	x	x	x	1	0	In	0x140802
16	0	0	0	0	1	0	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	Out	0x080000
17..31	0	0	0	0	0	0	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	Nop	0x000000

Estado		Acciones	Palabra de control compactada
0	F	Búsqueda de la Instr.: $IR \leftarrow Mem_w[PC] //$ Incremento del PC: $PC \leftarrow PC + 2$	$R@/Pc=0$, Byte=0, Ldlr=1, $Pc/Rx=1$, N=0x0002, Ry/N=0, OP=00, F=100, Alu/R@=1, LdPc=1.
1	D	Decodificación. Calculo @ salto tomado: $R@ \leftarrow PC + SE(N8)*2 //$ Lectura de registros. $(RX \leftarrow Ra) // (RY \leftarrow Rb)$	La decodificación no requiere ninguna acción en la UPG por lo que se usa la UPG en este ciclo para adelantar trabajo que pueda ser útil. Este cálculo solo será útil si la instrucción es BZ o BNZ. $N=SE(IR<7..0>)*2$, $Pc/Rx=1$, Ry/N=0, OP=00, F=100. Estas acciones se realizan sin tener que especificar nada en la palabra de control ya que RX y RY no tienen señal de permiso de escritura y @A y @B se generan directamente de los campos de bits del registro de instrucción $IR<11..9>$ e $IR<8..6>$, respectivamente, en cada ciclo del grafo. Por ello las especificamos aquí entre paréntesis y ya no las especificaremos de ninguna forma en el resto de nodos.
2	Al	$Rd \leftarrow RX \text{ Al } RY$	$Pc/Rx=0$, Ry/N=1, OP=00, F= $IR<2..0>$, P/I/L/A=00, WrD=1, @D= $IR<5..3>$.
3	Cmp	$Rd \leftarrow RX \text{ Cmp } RY$	$Pc/Rx=0$, Ry/N=1, OP=01, F= $IR<2..0>$, P/I/L/A=00, WrD=1, @D= $IR<5..3>$.
4	Addi	$Rd \leftarrow RX + SE(N6)$	N= $SE(IR<5..0>)$, $Pc/Rx=0$, Ry/N=0, OP=00, F=100, P/I/L/A=00, WrD=1, @D= $IR<8..6>$.
5	Addr	$R@ \leftarrow RX + SE(N6)$	N= $SE(IR<5..0>)$, $Pc/Rx=0$, Ry/N=0, OP=00, F=100.
6	Ld	$Rd \leftarrow Mem_w[R@]$	$R@/Pc=1$, Byte=0, P/I/L/A=01, WrD=1, @D= $IR<8..6>$.
7	St	$Mem_w[R@] \leftarrow RY$	$R@/Pc=1$, Byte=0, Wr-Mem=1.
8	Ldb	$Rd \leftarrow Mem_b[R@]$	$R@/Pc=1$, Byte=1, P/I/L/A=01, WrD=1, @D= $IR<8..6>$.
9	Stb	$Mem_b[R@] \leftarrow RY<7..0>$	$R@/Pc=1$, Byte=1, Wr-Mem=1.
10	Jalr	$PC \leftarrow RX \& (\sim 1) // Rd \leftarrow PC$	$Pc/Rx=0$, OP=10, F=011, Alu/R@=1, LdPc=1, P/I/L/A=11, WrD=1, @D= $IR<8..6>$.
11	Bz	if (RX == 0) $PC \leftarrow R@$	$Pc/Rx=0$, OP=10, F=000, Alu/R@=0, LdPc=z.
12	Bnz	if (RX != 0) $PC \leftarrow R@$	$Pc/Rx=0$, OP=10, F=000, Alu/R@=0, LdPc=!z.
13	Movi	$Rd \leftarrow SE(N8)$	N= $SE(IR<7..0>)$, Ry/N=0, OP=10, F=001, P/I/L/A=00, WrD=1, @D= $IR<11..9>$.
14	Movhi	$Rd \leftarrow (N*(2^8)) RX<7..0>$	N= $SE(IR<7..0>)$, $Pc/Rx=0$, Ry/N=0, OP=10, F=010, P/I/L/A=00, WrD=1, @D= $IR<11..9>$.
15	In	$Rd \leftarrow Input[N8]$	ADDR-IO= $IR<7..0>$, Rd-In=1, P/I/L/A=10, WrD=1, @D= $IR<11..9>$
16	Out	$Output[N8] \leftarrow RX$	ADDR-IO= $IR<7..0>$, Wr-Out=1.
17	Nop		
...	...		
31	Nop		

Fig. 13.4 Palabra de Control compacta para cada nodo del grafo de estados de la UC que implementa las fases de ejecución de cada instrucción.

Mnemotécnicos	Palabra de Control de 33 bits									
	@A	@B	Rb/N	OP	F	In/Alu	@D	WrD	N (hexa)	
ADD R6, R3, R5	0 1 1	1 0 1	1	0 0	1 0 0	0	1 1 0	1	X X X X	
CMPLEU R3, R1, R5	0 0 1	1 0 1	1	0 1	1 0 1	0	0 1 1	1	X X X X	
ADDI R7, R1, -1	0 0 1	x x x	0	0 0	1 0 0	0	1 1 1	1	F F F F	
ANDI R2, R3, 0xFF00	0 1 1	x x x	0	0 0	0 0 0	0	0 1 0	1	F F 0 0	
NOT R4, R2	0 1 0	x x x	x	0 0	0 1 1	0	1 0 0	1	X X X X	
MOVE R1, R5	1 0 1	x x x	x	1 0	0 0 0	0	0 0 1	1	X X X X	
MOVEI R3, 0XFA02	x x x	x x x	0	1 0	0 0 1	0	0 1 1	1	F A 0 2	
IN R2	x x x	x x x	x	x x	x x x	1	0 1 0	1	X X X X	
OUT R4	1 0 0	x x x	x	x x	x x x	x	x x x	0	X X X X	
ANDI -, R3, 0x8000	0 1 1	x x x	0	0 0	0 0 0	x	x x x	0	8 0 0 0	
NOP	x x x	x x x	x	x x	x x x	x	x x x	0	X X X X	
IN R2 // OUT R4	1 0 0	x x x	x	x x	x x x	1	0 1 0	1	X X X X	

Fig. 8.9 Ejemplos de acciones que se pueden hacer en la UP en un ciclo y las palabras de control de 33 bits asociadas a cada acción

@A y @B son XXX cuando es un IN o un MOVEI

Rb/N és X → cuando el registro @B es XXX

Rb/N és 0 → cuando el registro N se está utilizando (ADDI, MOVI)

Rb/N és 1 → cuando se utilizan todos los registros.

OP y F són XXX cuando es un IN, o un OUT.

In/Alu és X → cuando es un OUT o una funcion que no registra (ANDI -, R3, 0x8000)

In/Alu és 1 → cuando es un IN

@D és X → cuando es un OUT o una funcion que no registra (ANDI -, R3, 0x8000)

WRD és 0 → cuando es un OUT o una funcion que no registra (ANDI -, R3, 0x8000)

- S'afegeixen 10 bits a la paraula de control del tema anterior
 - ADDR-IO: bus de 8 bits amb l'identificador de *port*
 - Wr-Out: senyal binari que indica si en aquest cicle es fa l'acció OUT
 - Rd-In: senyal binari que indica si en aquest cicle es fa l'acció IN
- La nova paraula de control té 43 bits:

	@A			@B			Rb/N	OP		F			In/Alu		@D		WrD	Wr-Out	Rd-In	N (hexa)				ADDR-IO (hexa)	
IN R2, 33	x	x	x	x	x	x	x	x	x	x	x	x	1	0	1	0	1	0	1	X	X	X	X	2	1
OUT 0x50, R1	0	0	1	x	x	x	x	x	x	x	x	x	x	x	x	x	0	1	0	X	X	X	X	5	0
ADD R1, R2, R3	0	1	0	0	1	1	1	0	0	1	0	0	0	0	0	1	1	0	0	X	X	X	X	X	X

Lògica de control: paraula de control



UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Facultat d'Informàtica de Barcelona

- Passarà a tenir 47 bits
 - Afegim senyals Wr-Mem i Byte
 - Wr-Mem mai podrà valer x, valdrà "1" per ST*, "0" per a la resta
 - Byte valdrà "1" per STB i LDB, "0" per ST i LD, x per a la resta
 - Canviem senyal In/Alu (1 bit) per bus -i/l/a (2 bits)

@A			@B			Rb/N	OP		F			-i/l/a		@D		WrD	Wr-Out	Rd-In	Wr-Mem	Byte	TknBr	N (hexa)				ADDR-IO (hexa)	

- Els nous senyals els generarà la ROM_CTRL_LOGIC a partir del codi d'operació de la instrucció en execució

- 51 bits

@A			@B			Pc/Rx	Ry/N	OP		F			P/l/L/A	@D		WrD	Wr-Out	Rd-In	Wr-Mem	Ldlr	LdPc	Byte	Alu/R@	R@/Pc	N (hexa)				ADDR-IO (hexa)	

- Ldlr: senyal de càrrega del registre IR
- Pc/Rx, Alu/R@, R@/Pc: senyals de control dels nous multiplexors
- LdPc: senyal de càrrega del registre PC
 - Eliminem el senyal TknBr

- Lectura instrucció:
 - $REG_Q \rightarrow R@/PC \rightarrow ADDR_MEM \rightarrow RDMEM \rightarrow IR$
 - $T_p = 100 (REG_Q) + 60 (ROM\ OUT) + 50 (\text{mux } R@/PC\ sel) + 840 (\text{Memòria Ld}) + 40 (\text{dada registre càrrega IR}) = 1.090\ u.t.$
- $PC = PC + 2$
 - $REG_Q \rightarrow M \times N \rightarrow N \rightarrow Ry/N \rightarrow ALU \rightarrow Alu/R@ \rightarrow PC$
 - $T_p = 100 (REG_Q) + 60 (ROM\ OUT) + 90 (MUX_{4-1} M \times N\ selecció) + 40 (\text{Mux } Ry/N\ Dada) + 860 (ALU\ ADD) + 40 (MUX_{2-1} Alu/R@\ dada) + 40 (\text{dada registre càrrega PC}) = 1.230\ u.t.$
- Conclusió, l'etapa de *Fetch* imposa que $T_c \geq 1.230u.t.$

- Càlcul següent estat UC:
 - $REG_Q \rightarrow Q^+ \rightarrow REG_Q$
 - $T_p = 100 (REG\ Q) + 120 (ROM\ Q+) = 220\ u.t.$
- Lectura de registres
 - $IR \rightarrow REGFILE \rightarrow RX/R_Y$
 - $T_p = 100 (IR) + 130 (REGFILE\ MUX_{8-1}\ selecció) = 230\ u.t.$
- Càlcul adreça destí del salt
 - $REG_Q \rightarrow M \times N \rightarrow N \rightarrow Ry/N \rightarrow ALU \rightarrow R@$
 - $T_p = 100 (REG_Q) + 60 (ROMOUT) + 90 (MUX_{4-1} M \times N\ selecció) + 40 (\text{Mux } Ry/N\ Dada) + 860 (ALU\ ADD) = 1.150\ u.t.$
- Conclusions:
 - L'estat *Decode* imposa que $T_c \geq 1.150u.t.$
 - És menys restrictiu que *Fetch*

- Ldb: Accés a memòria i escriptura a REGFILE:
 - $REG_Q \rightarrow R@/PC \rightarrow ADDR_MEM \rightarrow RD_MEM \rightarrow P/I/L/A \rightarrow REGFILE$
 - $T_p = 100 (REG_Q) + 60 (ROMOUT) + 50 (MUX_{2-1} R@/PC \text{ selecció}) + 880 (RAM \text{ Ldb}) + 80 (MUX_{4-1} P/I/L/A) + 40 (\text{dada registre càrrega Rd}) = 1.210 \text{ u.t.}$
- Cmp (instrucció **CMPLE**)
 - $REG_Q \rightarrow Pc/Rx, Ry/N \rightarrow X, Y \rightarrow ALU \rightarrow P/I/L/A \rightarrow REGFILE$
 - $T_p = 100 (REG_Q) + 60 (ROMOUT) + 50 (PC/Rx, Ry/N \text{ MUX}_{2-1} \text{ selecció}) + 1.020 (ALU \text{ CMPLE}) + 80 (MUX_{4-1} P/I/L/A \text{ dada}) + 40 (\text{dada registre càrrega Rd}) = 1.350 \text{ u.t.}$
- Conclusions:
 - L'estat més restrictiu és Cmp (cas **CMPLE**)
 - Cmp imposa $T_c \geq 1.350 \text{ u.t.}$
 - Arrodonim i determinem $T_c = 1.400 \text{ u.t.}$

Seccions

- A un codi font *assembler* trobarem instruccions i dades
 - Les instruccions s'agrupen en una o varies seccions de codi
 - Contenen les instruccions SISA
 - Comencen amb la directiva `.text`
 - Les dades s'agrupen en una o varies seccions de dades
 - Contenen la reserva d'espai de memòria i la seva inicialització
 - Comencen amb la directiva `.data`
- El codi font *assembler* conté una seqüència de seccions
 - L'inici d'una secció comporta la finalització de la secció anterior
- La directiva `.end` indica el final de la darrera secció i del codi font
 - El contingut posterior del fitxer font és ignorat

- És una cadena alfanumèrica seguida del caràcter :
- Poden aparèixer tant a seccions de codi com de dades
- Permet identificar de forma simbòlica una adreça de memòria
 - Podrem fer referència a adreces de memòria que encara no són conegudes
- Exemple:

```
LD    R1, 0(R3)
BZ    R1, et1      ; Saltem a etiqueta et1
ADD   R2, R0, R1
ADD   R3, R1, R2
et1:  AND  R1, R2, R3 ; et1 representa adreça de la instr
```

- L'*assembler* calcularà el desplaçament corresponent al BZ
 - En aquest cas, +2
- El programador es despreocuparà de fer aquest càlcul
- Codi font més llegible i fàcil de mantenir
 - Si modifiquem el programa i augmenta la distància entre el BZ i el destí, l'*assembler* recalculerà el desplaçament



Reserva d'espai i inicialització de dades

- El llenguatge *assembler* ofereix directives per a dimensionar i inicialitzar les seccions de dades
 - *.space size, fill*
Reserva *size* bytes consecutius i els inicialitza amb el *byte fill*. El paràmetre *fill* és opcional; si no hi és, s'inicialitzarà amb el *byte* 0
 - *.byte fill-1, fill-2, ..., fill-n*
Reserva i inicialitza *n* bytes consecutius amb els valors *fill-1*, *fill-2*, ..., *fill-n*
 - *.word fill-1, fill-2, ..., fill-n*
Reserva i inicialitza *n* words consecutius amb els valors *fill-1*, *fill-2*, ..., *fill-n* (byte de menys pes a l'adreça parell)
 - *.even*
Si volem tenir la garantia que la següent sentència/directiva del codi font *assembler* s'ubiqui a una adreça parell, aquesta directiva insereix, si cal, una directiva *.byte*

- Exemple (assumint `.data` comença a adreça parell):

```
.data
v: .space 15, 20      ; 15 bytes inicialitzats a 20
   .even             ; Inserta un byte
w: .word 0x178A
   .byte 12, 0xFA, -1 ; 3 bytes init. a 12, 0xFA i -1
   .even             ; Inserta un byte
z: .word 0xABCD
```

- El primer `.even` insereix un *byte* per garantir que `.word` estigui ben alineat a una adreça parell
- El segon `.even` també perquè hem declarat 3 *bytes* després del *word*
- Si les dades es carreguem a partir de 0x3000 llavors `v=0x3000`, `w=0x3010`, `z=0x3016`
 - Mateix resultat si la primera directiva fos `.space 16, 20`

Definició de constants numèriques



- Dues possibles sintaxis:
 - `nom_constant = valor`
 - `.set nom_constant, valor`
- Exemple:

```
Mida = 100
.data
vector: .space Mida      ; Reserva Mida bytes
```

- Associa al símbol `Mida` el valor 100
- A partir d'aquest moment, l'*assembler* substituirà totes les aparicions del símbol `Mida` pel valor 100
 - La definició d'una constant no ocuparà espai a memòria**
 - Seria equivalent a un "Buscar i reemplaçar totes" al codi font o a un `#define` de C/C++
- Avantatges:
 - Codi més llegible
 - Facilita el manteniment de codi
 - Si el programador canvia el valor de la constant, l'*assembler* propagarà el canvi a tots els llocs on es referencia

- Permeten obtenir la part alta/baixa d'una dada de 16 bits
 - Siguin adreces de memòria, etiquetes, o constants numèriques
- Estalvia al programador haver de fer el càlcul
- Exemple:
 - Assumim que la secció .data es carrega a partir de 0xCAFE

```

        N = 24225                                ; 24225 = 0x5EA1

.data
        .space 2                                ; 0xCAFE
vector: .space 100, 0xFF                        ; 0xCB00 (0xCAFE+2)

.text
        MOVI    R0, lo(N)                       ; lo(N) = 0xA1
        MOVHI   R0, hi(N)                       ; hi(N) = 0x5E

        MOVI    R1, lo(vector)                   ; lo(vector) = 0x00
        MOVHI   R1, hi(vector)                   ; hi(vector) = 0xCB

```

Sintaxi/Semàntica noves instruccions

- **Load Byte:** **LDB** Rd, N6(Ra)
 - $Rd \leftarrow SE(MEM_b[Ra + SE(N6)]); \quad PC \leftarrow PC + 2;$
 - Fa extensió de signe a 16 bits del *byte* llegit de memòria
- **Store Byte:** **STB** N6(Ra), Rb
 - $MEM_b[Ra + SE(N6)] \leftarrow Rb < 7..0 >; \quad PC \leftarrow PC + 2;$
 - Només s'emmagatzema a memòria el *byte* baix de Rb
- **Load Word:** **LD** Rd, N6(Ra)
 - $Rd \leftarrow MEM_w[(Ra + SE(N6)) \& \sim 1]; \quad PC \leftarrow PC + 2;$
 - $\sim 1 = 0xFFFF$
 - $\&$ és l'operació AND bit a bit
 - El bit de menys pes del resultat de la suma es posa a 0 perquè, com accedim a *word*, l'adreça ha de ser un nombre parell
- **Store Word:** **ST** N6(Ra), Rb
 - $MEM_w[(Ra + SE(N6)) \& \sim 1] \leftarrow Rb; \quad PC \leftarrow PC + 2;$

- Afegirem la darrera instrucció de LM al repertori SISA
- **JALR** = *Jump Address and Link Register*
 - Imprescindible per a expressar en LM crides/retorns a rutines
 - Guarda el valor actual del PC a un registre i assigna al PC un nou valor
 - Permet saltar incondicionalment a una adreça arbitrària
- Sintaxi ensamblador:
 - **JALR** Rd, Ra
 - Format 2-R
- Semàntica:
 - $PC = PC + 2$; $tmp = Ra \& (\sim 1)$; $Rd = PC$; $PC = tmp$;
 - Variable tmp per si el registre font i el destí són el mateix
 - Amb $Ra \& (\sim 1)$ força que el nou valor del PC sigui parell
- Codificació en llenguatge màquina:
 - 0111 aaa ddd xxxxxx
 - Els 6 bits baixos de la codificació són irrelevants