

Exercicis típics

- Conversió entre paraula de control de 43 bits i mnemotècnic
 - Paraula de control corresponent a accions IN/OUT

	@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	x x x	1 0 1 0	1 0 1 0	x x x x	x x x x	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 x x x	0 1 0	x x x x	x x x x	x x x x	5 0	
ADD R1, R2, R3	0 1 0	0 0 1	1 1 1	0 0 1	0 0 0	0 0 0 0	1 1 0 0	x x x x	x x x x	x x x x	x x x x	

- Afegir E/S asíncrona a PPE
 - Fer l'entrada/sortida utilitzant teclat/impressora

- Ensamblar, desensamblar instruccions SISA

- Ensamblar:

- **XOR R4, R2, R5** \rightsquigarrow 0000 010 101 100 010 \rightsquigarrow 0x0562
- **BZ R3, -2**
- **IN R5, 0x12**
- **MOVI R6, -3**

- Desensamblar:

- 0x0564 \rightsquigarrow **ADD R4, R2, R5**
- 0x2345
- 0x81F4
- 0x1345

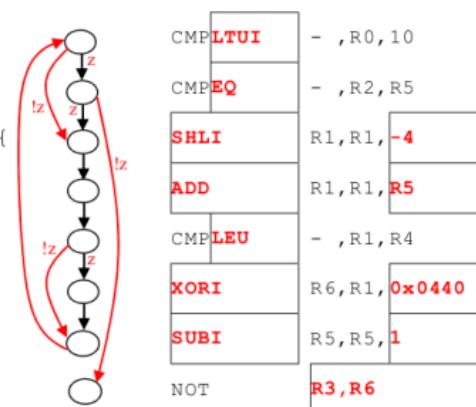
Canvis a l'estat del computador

- Com canvia l'estat del computador a l'executar una instrucció SISA?
 - Quins registres, *ports* canvién de valor un cop executada?
- Si l'estat inicial dels registres és **R0=0x0000, R1=0x0001, ... R7=0x0007, PC=0x12AE** i els ports d'entrada tenen el valor 0x1234, indiqueu quines modificacions es produiran a l'estat del computador després d'executar cadascuna de les següents instruccions SISA (els casos són independents).

- **ADD R7, R2, R4** \rightsquigarrow $R7 \leftarrow 0x0006$; $PC \leftarrow 0x12B0$;
- **SHA R6, R2, R1** \rightsquigarrow $R6 \leftarrow 0x0004$; $PC \leftarrow 0x12B0$;
- **CMPEQ R2, R3, R4** \rightsquigarrow $R2 \leftarrow 0x0000$; $PC \leftarrow 0x12B0$;
- **BZ R1, +3** \rightsquigarrow $PC \leftarrow 0x12B0$;
- **BNZ R1, -2** \rightsquigarrow $PC \leftarrow 0x12AC$;
- **IN R4, 26** \rightsquigarrow $R4 \leftarrow 0x1234$; $PC \leftarrow 0x12B0$;
- **OUT 13, R2** \rightsquigarrow $\text{OutPort}[13] = 0x0002$; $PC \leftarrow 0x12B0$;
- **MOVI R5, -4** \rightsquigarrow $R5 \leftarrow 0xFFFF$; $PC \leftarrow 0x12B0$;
- **MOVHI R5, -4** \rightsquigarrow $R5 \leftarrow 0xFC05$; $PC \leftarrow 0x12B0$;
- **ADDI R4, R5, -7** \rightsquigarrow $R4 \leftarrow 0xFFE$; $PC \leftarrow 0x12B0$;

- Donat un codi escrit en llenguatge C, expressar-lo com a accions de la UPG i després com a instruccions SISA
 - E3-Q1-1718 (assumiu que totes les dades són naturals)
 - Compte amb la traducció a SISA de XOR(R1, 0x0440)
 - Inicialment, assumiu que a 0x0016 podeu fer servir **MOVHI**
 - Després penseu com obtenir el mateix resultat d'una altra forma

```
while ((R0<10) || (R2!=R5)) {
    R1=R1/16+R5;
    if (R1>R4)
        R6=XOR(R1,0x0440);
    R5--;
}
R3=not (R6);
```



@I-Mem	
0x0000	MOVI R7, 10
0x0002	CMP LTU R7, R0, R7
0x0004	BNZ R7, 2
0x0006	CMP EQ R7, R2, R5
0x0008	BNZ R7, 10
0x000A	MOVI R7, -4
0x000C	SHL R1, R1, R7
0x000E	ADD R1, R1, R5
0x0010	CMP LEU R7, R1, R4
0x0012	BNZ R7, 3
0x0014	MOVI R7, 0x44
0x0016	SHL/A R7, R7, R7
0x0018	XOR R6, R1, R7
0x001A	ADDI R5, R5, -1
0x001C	BNZ R7, -15
0x001E	NOT R3, R6

Exercici típic

- Deduir el contingut de la ROM-CTRL-LOGIC
 - Poseu x sempre que sigui possible
 - Els senyals que modifiquen l'estat del computador mai valdran x
 - Bz, Bnz: Actualització registre PC
 - RdIn, WrOut: Lectura/escriptura ports d'entrada/sortida
 - WrD: Actualització registres R0...R7
 - Es pot demanar alguna(es) fila(s), columna(es) o interseccions
 - Practiqueu!!

Exercicis: ensamblar/desensamblar

- **LDB R1, -3(R2)** \rightsquigarrow 0101 010 001 111101 \rightsquigarrow 0x547D
- **ST 5(R6), R7** \rightsquigarrow 0100 110 111 000101 \rightsquigarrow 0x4DC5
- **LD R5, -1(R6)** \rightsquigarrow 0011 110 101 111111 \rightsquigarrow 0x3D7F
- **STB -2(R6), R1** \rightsquigarrow 0110 110 001 111110 \rightsquigarrow 0x6C7E

Exercicis: modificació de l'estat

- Assumim que el contingut inicial de la memòria és tal que els bytes amb adreça parell contenen 0xFA i els d'adreça senar contenen 0x34.
 - Assumim que $R0 = 0x0000$, $R1 = 0x0001$, ..., $R7 = 0x0007$ i $PC = 0x4520$
 - Quines modificacions a l'estat del computador provoquen les següents instruccions SISA (són independents, totes actuen sobre l'estat inicial)?
 - LD R3, 5(R2) $\rightsquigarrow R3 \leftarrow 0x34FA$; $PC \leftarrow 0x4522$;
 - LDB R2, -6(R1) $\rightsquigarrow R2 \leftarrow 0x0034$; $PC \leftarrow 0x4522$;
 - LDB R2, -5(R1) $\rightsquigarrow R2 \leftarrow 0xFFFA$; $PC \leftarrow 0x4522$;
 - ST 2(R5), R4 $\rightsquigarrow Mem_w[0x0006] \leftarrow 0x0004$; $PC \leftarrow 0x4522$;
 - STB 2(R1), R7 $\rightsquigarrow Mem_b[0x0003] \leftarrow 0x07$; $PC \leftarrow 0x4522$;

Exercici: SISA \iff paraula control

- Donada una instrucció SISA, determinar la seva paraula de control
 - LD R2, 10(R6)

@A	@B	Rb/N	OP	F	-/I/a	@D	WrD	Wr-Out	Rd-In	Wr-Mem	Byte	TknBr	Z (hexa)	ADDR-IO (hexa)		
1	1	0	x	x	x	0	0	0	1	0	0	0	1	A	X	X

- Donada una paraula de control, determinar la instrucció SISA

@A		@B		Rb/N	OP	F			-l/a		@D		WrD	Wr-Out	Rd-In	Wr-Mem	Byte	TknBr	Z (hexa)				ADDR-IO (hexa)				
1	0	0	1	1	1	0	0	0	1	0	0	x	x	x	x	x	0	0	0	0	1	F	F	F	D	X	X

- STB – 3(R4), R7

Exercici: omplir la ROM_CTRL_LOGIC

- Per files o per columnes o per interseccions

Dirección				Contenido																			
k_{15}	k_{14}	k_{13}	k_{12}	Bnz	Bz	Wr-Mem	RdIn	WrOut	WrD	Byte	Rb/N	-i/l/a1	-i/l/a0	OP1	OP0	MxN1	MxN0	MxF	F2	F1	F0	Mx@D1	Mx@D0
0 0 1 0 x	0 0 0 0 0	0 0 0 0 0	0 0 0 0 1	x	0 0 0 0 0	0 0 0 0 0	0 0 0 0 0	0 0 0 0 0	0 0 0 0 0	0 0 0 0 0	0 0 0 0 0	0 0 0 0 0	0 0 0 0 0	0 0 0 0 0	0 0 0 0 0	0 0 0 0 0	0 0 0 0 0	1 1 0 0 0	0 1 1 0 0	0 1 0 0 0	0 1 0 0 0	0 1 0 0 0	0 1 0 0 0
0 1 0 1 x	0 0 0 0 0	0 0 0 0 0	0 0 0 0 0	1	1 0 0 0 0	1 0 0 0 0	1 0 0 0 0	1 0 0 0 0	1 0 0 0 0	1 0 0 0 0	1 0 0 0 0	1 0 0 0 0	1 0 0 0 0	1 0 0 0 0	1 0 0 0 0	1 0 0 0 0	1 0 0 0 0	1 1 0 0 0	1 1 0 0 0	0 0 0 0 0	0 0 0 0 0	0 0 0 0 0	0 0 0 0 0
1 0 0 1 1	0 0 0 0 0	0 0 0 0 0	0 0 0 0 0	1	x	0 0 0 0 0	0 0 0 0 0	0 0 0 0 0	0 0 0 0 0	0 0 0 0 0	0 0 0 0 0	0 0 0 0 0	0 0 0 0 0	0 0 0 0 0	0 0 0 0 0	0 0 0 0 0	1 0 0 0 0	1 1 0 0 0	1 0 1 0 0	1 0 1 0 0	1 0 1 0 0	1 0 1 0 0	1 0 1 0 0

ADDI

LDB

MOVHI

Exercici



- Calculeu el **temps d'execució** del següent programa als computadors SISC Harvard unicicle i multicicle, i el **percentatge d'augment de la velocitat d'execució** del multicicle respecte al unicicle

```

MOVI   R0 , 0x00
MOVHI  R0 , 0x10
LD     R1 , 0(R0) ; Assumiu que R1 valdrà 0x002B
MOVI   R3 , 0
MOVI   R4 , 0x01
MOVI   R6 , -1
Loop: AND   R2 , R1 , R4
       ADD   R3 , R3 , R2
       SHL   R1 , R1 , R6
       BNZ   R1 , -4
       OUT   3 , R3
; Final del programa
    
```

Exercici



- Analitzant el codi veiem ...
 - Executa 6 instruccions abans d'arribar al bucle (5 ràpides, 1 lenta)
 - El bucle executa 4 instruccions ràpides i itera 6 cops
 - Executa 1 instrucció ràpida després del bucle
- $N_{instruccions} = 6 \text{ inst} + (4 \text{ iter} \times 6 \text{ inst/iter}) + 1 \text{ inst} = 31 \text{ inst}$
 - 30 instruccions ràpides (instR) i 1 instrucció lenta (instL)
- Temps d'execució:
 - $T_{execució \text{ unicicle}} = 31 \text{ inst} \times 3.000 \text{ u.t./inst} = 93.000 \text{ u.t.}$
 - $T_{execució \text{ multicicle}} = 30 \text{ instR} \times 2.250 \text{ u.t./instR} + 1 \text{ instL} \times 3.000 \text{ u.t./instL} = 70.500 \text{ u.t.}$
- Increment de velocitat? ($N=N_{instruccions}=31$)
 - Regla de tres:

$$\begin{array}{rcl} N/93.000 \text{ inst./u.t.} & - & 100 \\ N/70.500 \text{ inst./u.t.} & - & x \end{array}$$
 - $x = 100 \cdot (N/70.500) / (N/93.000) = 100 \cdot 93.000 / 70.500 = 131,9$
 \Rightarrow és un 31,9% més ràpid

- Donat l'estat actual de la UC i el contingut del registre IR, indiqueu la paraula de control i l'estat següent de la UC (assumiu que abans d'executar f) el registre R4 val 0xFFFF).

Apartado	Nodo/Estado (Mnemo Salida)	Instrucción en el IR (en ensamblador)	Nodo/Estado Siguiente (Mnemo Salida)
a	F	(no se sabe)	D
b	D	BNZ R4, -2	Bnz
c	Addr	STB -3 (R1), R7	Stb
d	Al	SUB R3, R1, R2	F
e	Movhi	MOVHI R1, 27	F
f	Bnz	BNZ R4, -5	F

Apartado	@A	@B	Pc/Rx	Rv/N	OP	F	P/I/L/A	@D	WrD	Wr-Out	Rd-In	Wr-Mem	LdIr	LdPc	Byte	Ali/R@	R@/Pc	N (hexa)	ADDR-IO
a	xxx	xxx	1 0	00	100	xx	xxx	0 0	0 0	0 0	1 1	0 1 0	0002	xx					
b	100	xxx	1 0	00	100	xx	xxxx	0 0	0 0	0 0	0 0	x x x	FFFC	xx					
c	xxx	111	0 0	00	100	xx	xxx	0 0	0 0	0 0	0 0	x x x	FFFD	xx					
d	xxx	xxx	0 1	00	101	00	011	1 0	0 0	x 0	x x x	xxxx	xxxx	xx					
e	xxx	xxx	0 0	10	010	00	001	1 0	0 0	x 0	x x x	001B	xx						
f	xxx	xxx	0 x	10	000	xx	xxx	0 0	0 0	x 1	x 0 x	xxxx	xxxx	xx					

- Els valors dels senyals en verd són coneixuts (surten directament de l'IR) però en aquest estat són irrelevants.

Exemple: contingut ROM Q+

- Exemple: contingut de l'adreça 0xAC de la ROM Q+?
 - Les adreces de la ROM Q+ tenen 10 bits:
 - Els 5 de més pes són l'estat actual
 - Els 5 de menys pes són els bits 15,14,13,12 i 8 de la instrucció
 - $0xAC \rightsquigarrow (10101100)_2 \rightsquigarrow (0010101100)_2 \rightsquigarrow (00101\ 01100)_2$
 - Estat actual = $00101_2 \rightsquigarrow 5_{10} \rightsquigarrow Addr$
 - Codi d'operació = $01100_2 \rightsquigarrow (0110\ 0)_2 \rightsquigarrow STB$
 - Mirant el graf d'estats de la UC, si estem a l'estat Addr i la instrucció és STB, l'estat futur és Stb
 - Com el codi de l'estat Stb és 9, el contingut de l'adreça 0xAC de la ROM Q+ és $01001_2 = 09_{16} = 0x09$

- Exemple: quines adreces de la ROM Q+ contenen l'estat Addr?
 - Mirant el graf d'estats de la UC, veiem que l'estat Addr és l'estat futur de l'estat D per a les instruccions LD, ST, LDB, STB
 - D és codifica amb $1 = (00001)_2$
 - Els codis d'operació de LD, ST, LDB, STB són $0011_2, 0100_2, 0101_2, 0110_2$ respectivament
 - Aquestes instruccions no tenen bit extra de codi d'operació
 - Adreces ROM Q+: 5 bits codificació de l'estat + 4 bits codi d'operació + 1 bit codi d'operació extra
 - Adreces involucrades:
 - $(00001\ 0011\ x)_2 \rightsquigarrow (00\ 0010\ 011x)_2 \rightsquigarrow 0x026$ i $0x027$
 - $(00001\ 0100\ x)_2 \rightsquigarrow (00\ 0010\ 100x)_2 \rightsquigarrow 0x028$ i $0x029$
 - $(00001\ 0101\ x)_2 \rightsquigarrow (00\ 0010\ 101x)_2 \rightsquigarrow 0x02A$ i $0x02B$
 - $(00001\ 0110\ x)_2 \rightsquigarrow (00\ 0010\ 110x)_2 \rightsquigarrow 0x02C$ i $0x02D$

- Indiqueu el contingut de les adreces $0xC$ i $0x26$?
 - $0xC = 0000000110_2 = (00000\ 0110\ 0)_2 \Rightarrow$ Correspon a la transició Fetch (00000) amb codi d'operació Stb (0110x) per tant, ha d'anar a parar a Decode (00001) \Rightarrow el contingut és $00001_2 = 0x01$
 - $0x26 = 0000100110_2 = (00001\ 0011\ 0)_2 \Rightarrow$ Correspon a la transició Decode (00001) amb codi d'operació Ld (0011x) per tant, ha d'anar a parar a Addr (00101) \Rightarrow el contingut és $00101_2 = 0x05$
- Quina(es) adreça(ces) de la ROM Q+ porta(en) a l'estat In?
 - Arribem a In des de Decode (00001) amb codi d'operació 10100 per tant, l'adreça $(00001\ 1010\ 0)_2 = 0000110100_2 = 0x034$
- Quina(es) adreça(ces) de la ROM Q+ porta(en) a l'estat AI?
 - Arribem a AI des de Decode (00001) amb codi d'operació 0000x per tant, les adreces $(00001\ 0000\ x)_2 = 000010000x_2 = 0x020$ i $0x021$

- Indiqueu el contingut d'un seguit de files/columnes/interseccions de la ROM OUT

- Posseu x sempre que sigui possible

@ROM	Bz	MxF	WrD	R@/PC	Pc/Rx	LdIR	Ry/N	Estado
3	0	0	1	x	0	x	1	Cmp
5	0	1	0	x	0	0	0	Addr
9	0	x	0	1	x	x	x	Stb
12	0	1	0	x	0	x	x	Bnz

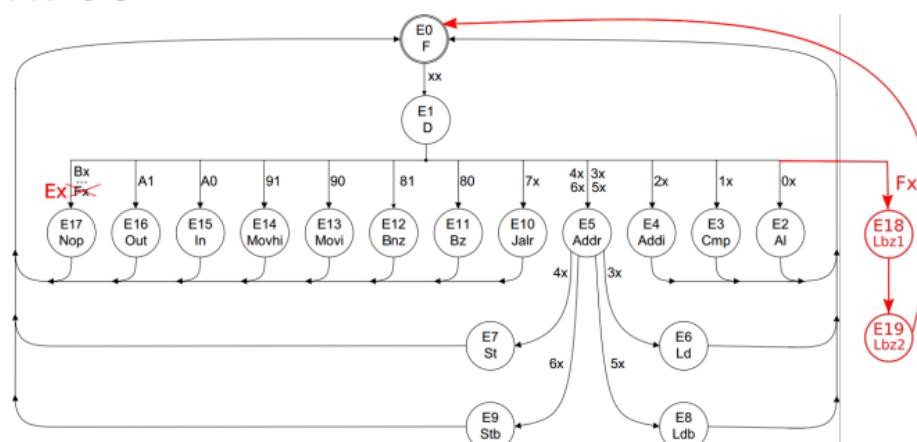
Exercici: afegir noves instruccions a SISA



- Instruccions a afegir: salts condicionals a direccions absolutes
 - Long Branch on Zero:*
 - Sintaxi: LBZ Ra, Rb
 - Codificació: 1111 aaa bbb xxxxxxx
 - Semàntica: if (Ra == 0) PC ← Rb; else PC ← PC + 2;
 - Long Branch on No Zero:*
 - Sintaxi: LBNZ Ra, Rb
 - Codificació: 1110 aaa bbb xxxxxxx
 - Semàntica: if (Ra != 0) PC ← Rb; else PC ← PC + 2;
- De moment, sense modificar *hardware* de la UPG ni de la UCG
 - Més endavant ens permetrem modificar el *hardware*
- Haurem de modificar el contingut de la ROM Q+ i de la ROM OUT
 - Afegir nous estats al graf d'estats de la UC
 - Definir les seves paraules de control
- S'assumeix que la resta d'instruccions SISA han de continuar funcionant com fins ara**

- Estratègia d'implementació: Imitarem BZ
 - Al primer cicle de càlcul guardarem a R@ la direcció de salt Rb deixant passar RY (que conté Rb) per la ALU
 - Al cicle següent avaluarem Ra==0 deixant passar RX (que conté Ra) per la ALU; si z=1 actualitzarem PC amb R@
- Calen dos estats per a completar l'execució:
 - Lbz1: R@ \leftarrow RY RX \leftarrow Ra
 - Lbz2: if (RX == 0) PC \leftarrow R@;
- Modificacions a les ROM's:
 - ROM Q+: reflectir les noves transicions
 - ROM OUT: per generar les paraules de control

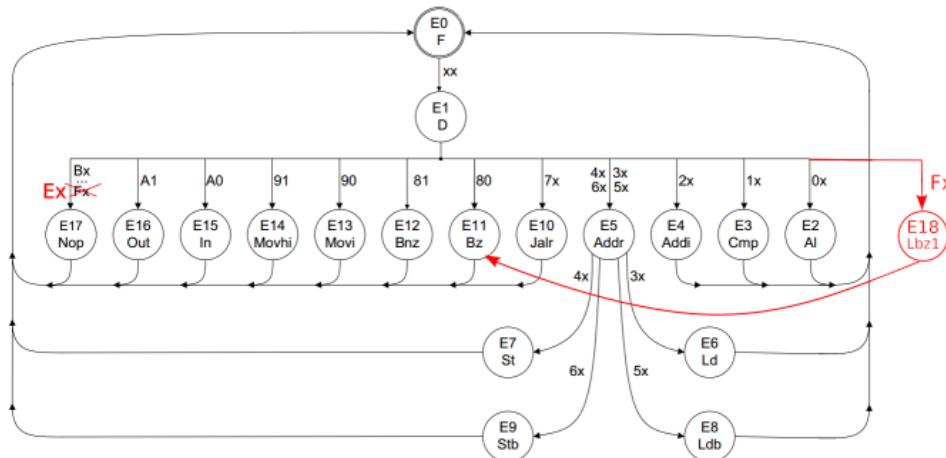
- Graf d'estats UC:



- Modificacions a la ROM Q+
 - ROM_Q+[00001 1111 x] = 0x12, ROM_Q+[10010 1111x] = 0x13 i ROM_Q+[10011 xxxxx] = 0x00
- Modificacions a la ROM OUT

@ROM	Acciones asociadas al estado (en lenguaje de transferencia de registros)																							
	Bnz	Bz	WrMem	RdIn	WrOut	WrD	Ldr	Byle	R@/Pc	Alu/R@	Pc/Rx	Ry/N	P/I/LA1	P/I/LA0	OP1	OP0	MxN1	MxN0	MxF	F2	F1	F0	Mx@D1	Mx@D0
18	0	0	0	0	0	0	x	x	x	x	1	x	x	1	0	x	x	1	0	0	1	x	x	
19	0	1	0	0	0	0	x	x	x	0	0	x	x	x	1	0	x	x	1	0	0	0	x	x

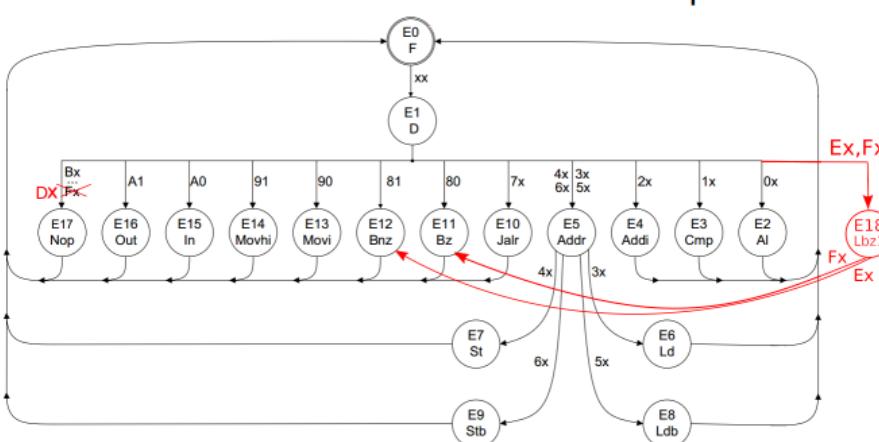
- Els estats Lbz2 i Bz són idèntics!!!
 - Des de Lbz1 podem passar a l'estat Bz, l'estat Lbz2 és redundant
- Graf d'estats UC:



- Modificacions a la ROM Q+
 - ROM_Q+[00001 1111 x] = 0x12 i ROM_Q+[10010 1111x] = 0x0B
- Modificacions a la ROM OUT

@ROM	Acciones asociadas al estado (en lenguaje de transferencia de registros)																						
	Bnz	Bz	WrMem	WrOut	WrD	LdR	Byte	R@/Pc	Alu/R@	Pc/Rx	Ry/N	P/I/LA1	P/I/LA0	OP1	OP0	MxN1	MxN0	MxF	F2	F1	F0	Mx@D1	Mx@D0
18	0	0	0	0	0	0	x	x	x	x	1	x	x	1	0	x	x	1	0	0	1	x	x

- Graf d'estats UC:
 - Reutilitzarem estat Lbz1
 - Transicionarà a Bz o Bnz en funció del codi d'operació



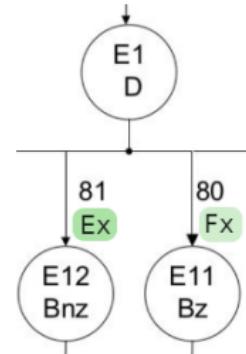
- Modificacions a la ROM Q+ (a més de les anteriors)
 - ROM_Q+[00001 1110 x] = 0x12 i ROM_Q+[10010 1110x] = 0x0C
- Modificacions a la ROM OUT (a més de les anteriors)
 - Cap

- La primera estratègia de modificació *hardware* serà que el $R@$ calculat a *Decode* depengui del codi d'operació de la instrucció a IR:

$$R@ = \begin{cases} Rb & , \text{ si LBZ (1111) o LBNZ (1110)} \\ PC + SE(N8) \cdot 2 & , \text{ altrament (com fins ara)} \end{cases}$$

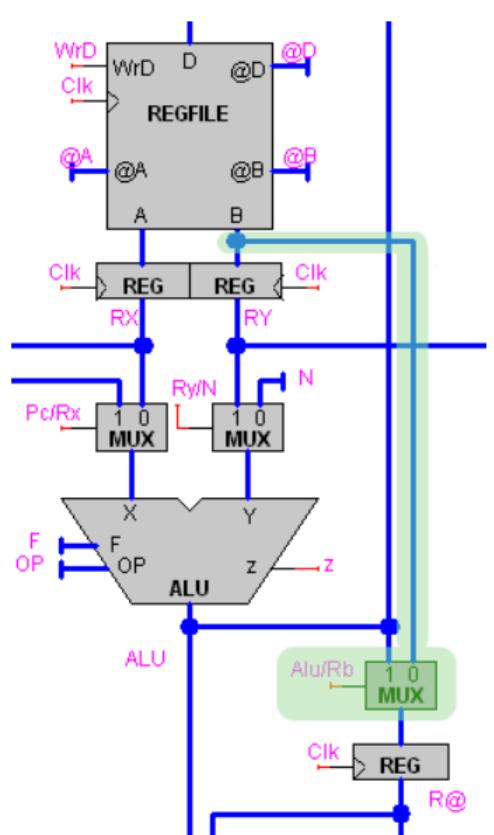
- Canvis al graf d'estats:

- Decode* continua sent comú (tot i que farà accions diferents en funció del codi d'operació)
- Com en acabar *Decode* tindrem a $R@$ l'adreça destí del salt (tant si és BZ, BNZ com si és LBZ, LBNZ), podrem utilitzar els estats Bz i Bnz per a les noves instruccions
 - No calen nous estats ni modificar ROM_OUT
 - Ara les noves instruccions trigaran 3 cicles
- Cal afegir transicions de *Decode* a Bz/Bnz pels nous codis d'operació
 - $\text{ROM_Q} + [00001\ 1110\ x] = 0xC$
 - $\text{ROM_Q} + [00001\ 1111\ x] = 0xB$



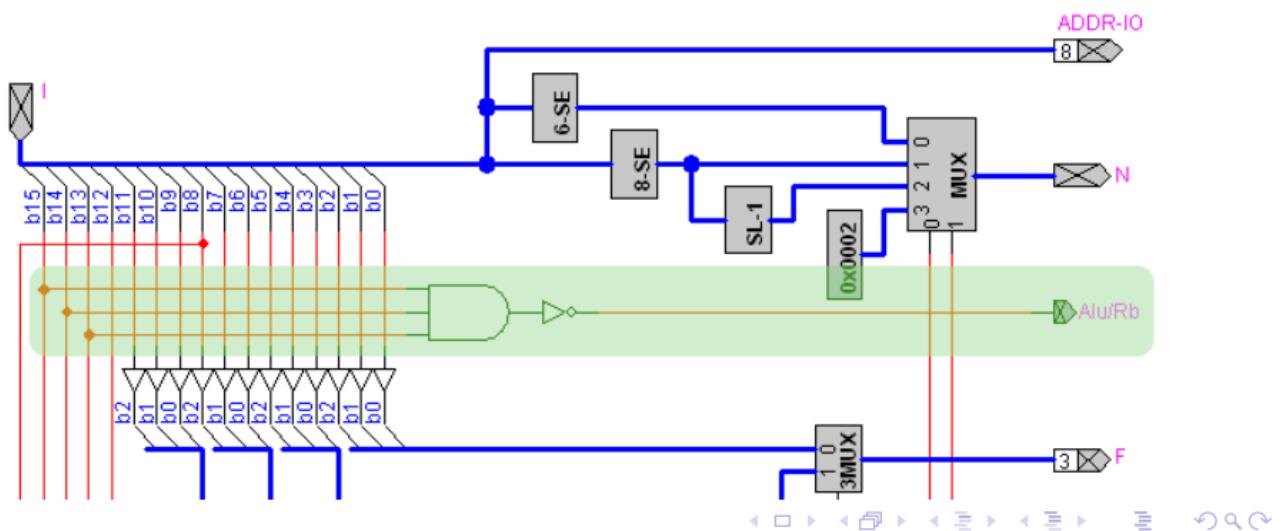
Modificacions a la UPG

- Nou camí des de Rb a $R@$
- Cal multiplexor a l'entrada de $R@$
 - Tria entre les dues opcions
- Nou senyal a paraula de control (Alu/Rb)
 - Si Alu/Rb=1, $R@ \leftarrow ALU$ (com fins ara)
 - Si Alu/Rb=0, $R@ \leftarrow Rb$ (per LBZ i LBNZ)

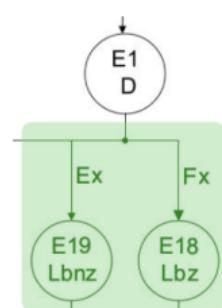


- Modificacions a la Lògica de Control:

- Alu/Rb, un nou senyal a la paraula de control
- Es pot calcular a partir dels bits 15, 14 i 13 de la instrucció a IR
 - Tots tres bits valen 1 a LBZ i a LBNZ
- El més elegant seria afegir sortida a la ROM_OUT
 - 0 per LBZ/LBNZ, 1 per BZ/BNZ, x altrament
 - Així no exposem els codis d'operació SISA fora de la ROM_OUT

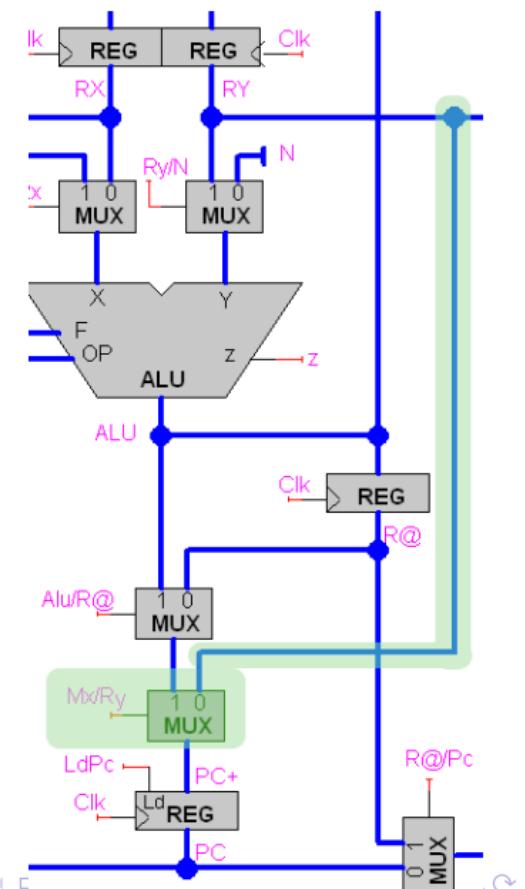


- La segona estratègia de modificació *hardware* serà afegir un camí des de RY (o Rb) al registre PC sense passar per la ALU.
 - D'aquesta forma, if ($RX == 0$) $PC \leftarrow RY$; es podrà fer en un cicle perquè l'ALU només ha de calcular bit z
- Canvis al graf d'estats:
 - Necessitarem dos estats nous (Lbz i Lbnz) per a activar el nou camí i actuar en funció de z
 - Lbz: if ($RX == 0$) $PC \leftarrow RY$;
 - Lbnz: if ($RX != 0$) $PC \leftarrow RY$;
 - Cal afegir transicions de *Decode* a Lbz (18=0x12)/Lbnz (19=0x13) pels nous codis d'operació
 - $ROM_Q+[00001\ 1110\ x] = 0x13$
 - $ROM_Q+[00001\ 1111\ x] = 0x12$
 - Aquests nous estats transicionaran a *Fetch*
 - Si les files sense utilitzar de la ROM_Q+ estaven inicialitzades a 0, no cal fer res



• Modificacions a la UPG

- Nou camí des de RY (o Rb) a PC
- Cal nou multiplexor a l'entrada del PC
 - També es podria fer convertint el multiplexor Alu/R@ en un Mux 4-1
- Nou senyal a paraula de control (Mx/Ry)
 - Si $Mx/Ry=1$, $PC \leftarrow Mux\ Alu/R@$ (com fins ara)
 - Si $Mx/Ry=0$, $PC \leftarrow RY$ (per LBZ i LBNZ)

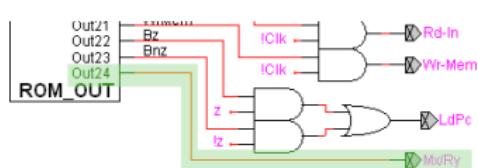


Afegir instruccions LBZ i LBNZ v4.0

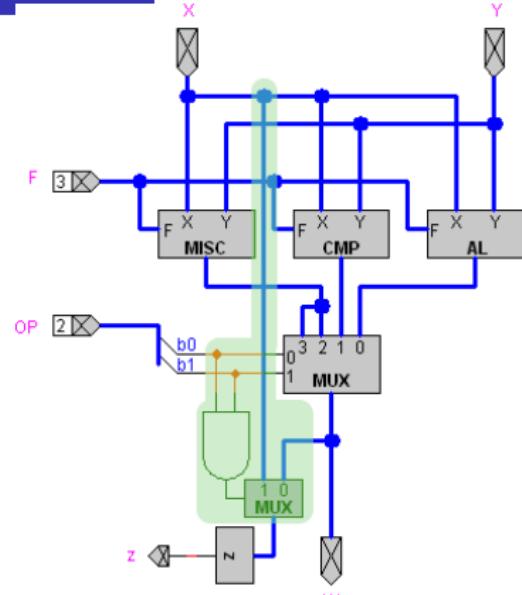
• Modificacions a la Lògica de Control:

- Mx/Ry, un nou senyal a la paraula de control
 - Afegirem columna a ROM_OUT amb el seu valor a cada estat
- Cal definir a ROM_OUT la nova columna i els nous estats

@ROM	Mx/Ry	Bnz	Bz	WlMem	RdIn	WlOut	WrD	LdIn	Byte	R@/Pc	Alu/R@	Pc/Rx	Ry/N	P//LA1	P//LA0	OP1	OP0	MxN1	MxN0	MxF	F2	F1	F0	Mx@D1	Mx@D0			
0	1 1	0 0 0	0	0 1	0	0 1	1 0	x x	0 0 1 1	1 1 0 0	x x	0 0 1 1	1 1 0 0	x x	x x	x x	x x	x x	x x	x x	x x	x x	x x	x x	x x			
1	x 0 0	0 0 0	0	0 0	0	0 0	x x	x x x 1 0	x x	x x	0 0 1 0	1 1 0 0	x x	0 0 1 0	1 1 0 0	x x	x x	x x	x x	x x	x x	x x	x x	x x	x x			
2	x 0 0	0 0 0	0	1	x x	x x x x	0 1	0 0	0 0 0 0	0 0 1 0	x x	0 0 0 0	x x	0 x x x	0 x x x	0 x x x	0 x x x	0 x x x	0 x x x	0 x x x	0 x x x	0 x x x	0 x x x	0 x x x	0 x x x	0 x x x		
3	x 0 0	0 0 0	0	1	x x	x x x x	0 1	0 0	0 0 0 0	0 0 1 0	x x	0 0 0 0	x x	0 1 x x	0 x x x	0 x x x	0 x x x	0 x x x	0 x x x	0 x x x	0 x x x	0 x x x	0 x x x	0 x x x	0 x x x	0 x x x		
4	x 0 0	0 0 0	0	1	x x	x x x x	0 0	0 0	0 0 0 0	0 0 0 0	x x	0 0 0 0	x x	0 0 0 0	0 0 0 0	0 1 1 0 0	0 1 1 0 0	0 1 1 0 0	0 1 1 0 0	0 1 1 0 0	0 1 1 0 0	0 1 1 0 0	0 1 1 0 0	0 1 1 0 0	0 1 1 0 0			
5	x 0 0	0 0 0	0	0	0	0	0	0	x x x x	0 0	x x	x x	x x	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	1 1 0 0	1 1 0 0	1 1 0 0	1 1 0 0	1 1 0 0	1 1 0 0	1 1 0 0	1 1 0 0	1 1 0 0		
6	x 0 0	0 0 0	0	0	1	x 0	1 x x x	x 0	0 1 x x x	x x	x x	x x	x x	0 1 x x x	0 1 x x x	x x x x	x x x x	x x x x	x x x x	x x x x	x x x x	x x x x	x x x x	x x x x	0 1	0 1		
7	x 0 0	0 1 0	0	0	0	0	x 0	1 x x x	x x x x	x x x x	x x x x	x x x x	x x x x	x x x x	x x x x	x x x x	x x x x	x x x x	x x x x	x x x x	x x x x	x x x x	x x x x	x x x x	x x x x	x x x x		
8	x 0 0	0 0 0	0	1	x 1	1 x x x	x x x x	0 1	x 0 1 x x x	x x x x	x x x x	x x x x	x x x x	x x x x	x x x x	x x x x	x x x x	x x x x	x x x x	x x x x	x x x x	x x x x	x x x x	x x x x	x x x x	0 1		
9	x 0 0	0 1 0	0	0	0	x 1	1 x x x	x x x x	x x x x	x x x x	x x x x	x x x x	x x x x	x x x x	x x x x	x x x x	x x x x	x x x x	x x x x	x x x x	x x x x	x x x x	x x x x	x x x x	x x x x	x x x x		
10	1 1 1	0 0 0	1	x x	x x x 1 0	x	1 1 1 0	x x	1 0 1 1 0	x x	1 0 1 1 0	x x	1 0 1 1 0	x x	1 0 1 1 0	1 0 1 1 0	1 0 1 1 0	1 0 1 1 0	1 0 1 1 0	1 0 1 1 0	1 0 1 1 0	1 0 1 1 0	1 0 1 1 0	1 0 1 1 0	1 0 1 1 0	1 0 1 1 0		
11	1 0 1	0 0 0	0	x x	x x x x	0 0	x x x x	x x x x	x x x x	x x x x	x x x x	x x x x	x x x x	x x x x	x x x x	x x x x	x x x x	x x x x	x x x x	x x x x	x x x x	x x x x	x x x x	x x x x	x x x x	x x x x	x x x x	
12	1 1 0	0 0 0	0	x x	x x x x	0 0	x 0 0 x	x x x x	x x x x	x x x x	x x x x	x x x x	x x x x	x x x x	x x x x	x x x x	x x x x	x x x x	x x x x	x x x x	x x x x	x x x x	x x x x	x x x x	x x x x	x x x x	x x x x	
13	x 0 0	0 0 0	1	x x	x x x x	x x x x	0 0	0 0 0 0	1 0 0 1	1 0 0 1	1 0 0 1	1 0 0 1	1 0 0 1	1 0 0 1	1 0 0 1	1 0 0 1	1 0 0 1	1 0 0 1	1 0 0 1	1 0 0 1	1 0 0 1	1 0 0 1	1 0 0 1	1 0 0 1	1 0 0 1	1 0 0 1		
14	x 0 0	0 0 0	1	x x	x x x x	x x x x	0 0	0 0 0 0	1 0 0 1	1 0 0 1	1 0 0 1	1 0 0 1	1 0 0 1	1 0 0 1	1 0 0 1	1 0 0 1	1 0 0 1	1 0 0 1	1 0 0 1	1 0 0 1	1 0 0 1	1 0 0 1	1 0 0 1	1 0 0 1	1 0 0 1	1 0 0 1		
15	x 0 0	0 0 1	0	1	x x	x x x x	x x x x	1 0	x x x x x	x x x x x	x x x x x	x x x x x	x x x x x	x x x x x	x x x x x	x x x x x	x x x x x	x x x x x	x x x x x	x x x x x	x x x x x	x x x x x	x x x x x	x x x x x	x x x x x	x x x x x		
16	x 0 0	0 0 0	1	0	x x	x x x x	x x x x	x x	x x x x x	x x x x x	x x x x x	x x x x x	x x x x x	x x x x x	x x x x x	x x x x x	x x x x x	x x x x x	x x x x x	x x x x x	x x x x x	x x x x x	x x x x x	x x x x x	x x x x x	x x x x x		
17	x 0 0	0 0 0	0	x x	x x x x	x x x x	x x	x x x x x	x x x x x	x x x x x	x x x x x	x x x x x	x x x x x	x x x x x	x x x x x	x x x x x	x x x x x	x x x x x	x x x x x	x x x x x	x x x x x	x x x x x	x x x x x	x x x x x	x x x x x	x x x x x		
18	0 0 1	0 0 0	0	x x	x x x x	x x x x	x x	0 x x x x	x x x x x	1 0 x x x x	1 0 x x x x	1 0 x x x x	1 0 x x x x	1 0 x x x x	1 0 x x x x	1 0 x x x x	1 0 x x x x	1 0 x x x x	1 0 x x x x	1 0 x x x x	1 0 x x x x	1 0 x x x x	1 0 x x x x	1 0 x x x x	1 0 x x x x	1 0 x x x x		
19	0 1 0	0 0 0	0	x x	x x x x	x x x x	x x	0 x x x x	x x x x x	1 0 x x x x	1 0 x x x x	1 0 x x x x	1 0 x x x x	1 0 x x x x	1 0 x x x x	1 0 x x x x	1 0 x x x x	1 0 x x x x	1 0 x x x x	1 0 x x x x	1 0 x x x x	1 0 x x x x	1 0 x x x x	1 0 x x x x	1 0 x x x x	1 0 x x x x		
20..31	x 0 0	0 0 0	0	x x	x x x x	x x x x	x x	x x x x x	x x x x x	x x x x x	x x x x x	x x x x x	x x x x x	x x x x x	x x x x x	x x x x x	x x x x x	x x x x x	x x x x x	x x x x x	x x x x x	x x x x x	x x x x x	x x x x x	x x x x x	x x x x x	x x x x x	Nop



- La tercera estratègia de modificació *hardware* serà ampliar la ALU.
- Afegirem una funcionalitat que deixi passar l'operand *Y* però que calculi el bit *z* en funció de l'operand *X*
 - Triem codificació $OP=11$, $F=001$ (simplifica implementació ALU)
- Graf d'estats i accions com a v4.0
- No cal afegir senyals ni a ROM_OUT ni a paraula de control
- El contingut de les noves files de la ROM_OUT serà:



@ROM	Bnz	Bz	WrMem	RdIn	WrOut	WrD	Ldr	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		
18	0	1	0	0	0	0	x	x	x	1	0	x	x	x	x	1	1	x	x	1	0	0	1	x	x	Lbz
19	1	0	0	0	0	0	x	x	x	1	0	x	x	x	x	1	1	x	x	1	0	0	1	x	x	Lbnz

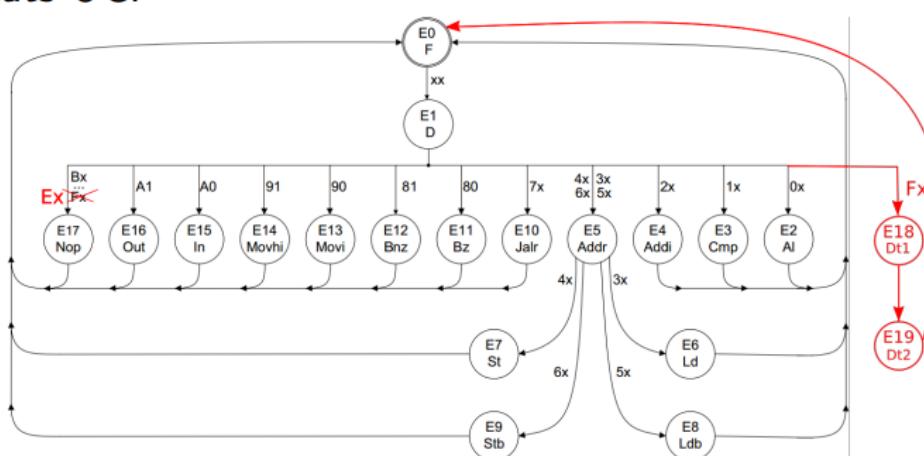
DECTEST1: especificació

- Decrement 1 and test*: permetrà implementar una espera
 - Sintaxi: DECTEST1 Ra
 - Codificació: 1111 aaa x 11111111
 - Semàntica: PC = PC+2; Ra = Ra-1; if (Ra!=0) PC = PC-2;
- La instrucció decrementa Ra. Si el resultat no és 0, modifica el PC de forma que la següent instrucció a executar torni a ser el DECTEST1
 - Permet implementar bucles dins d'una instrucció de LM
- D'on traurem el -1?
 - Dels 8 bits baixos de la codificació de la instrucció :-)
 - SE(N8)

- Estratègia d'implementació: Imitarem BZ
 - A un cicle guardarem a R@ la direcció de salt PC-2
 - Al cicle següent guardem (RX + -1) a Rd; com el bit z reflecteix si $(RX + -1) \neq 0$, si $z=0$ actualitzarem PC amb R@
- Calen dos estats per a completar l'execució:
 - Dt1: $R@ \leftarrow PC - 2$
 - També necessitem $RX \leftarrow Ra$ però ja es fa a tots els cicles
 - Dt2: $Ra \leftarrow RX + -1$; if ($!z$) $PC \leftarrow R@$
- Modificacions a les ROM's:
 - ROM Q+: reflectir les noves transicions
 - ROM OUT: per generar les paraules de control

DECTEST1: ROM Q+ i ROM OUT

- Graf d'estats UC:



- Modificacions a la ROM Q+
 - $ROM_Q+[00001\ 1111\ x] = 0x12$, $ROM_Q+[10010\ 1111x] = 0x13$ i $ROM_Q+[10011\ xxxxx] = 0x00$
- Modificacions a la ROM OUT

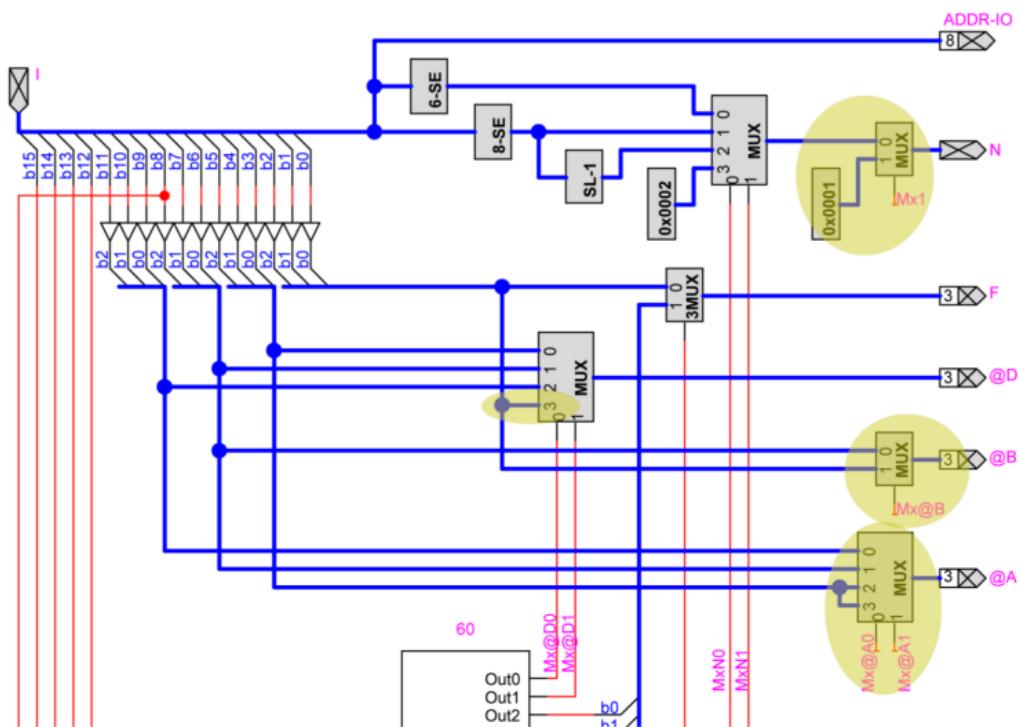
@ROM	Bnz	Bz	WnMem	WrIn	WrOut	WrD	Ldir	Byte	R@/Pc	Alu/R@	Pc/Rx	Ry/N	Pm/LA1	Pm/LA0	OP1	OP0	Mxn1	Mxn0	MxF	F2	F1	F0	Mx@D1	Mx@D0
18	0	0	0	0	0	0	0	x	x	x	1	0	x	x	0	0	1	1	1	1	0	1	x	x
19	1	0	0	0	0	1	x	x	x	0	0	0	0	0	0	0	0	1	1	1	0	0	1	0

Acciones asociadas al estado
(en lenguaje de transferencia
de registros)

R@ ← PC - 2
Ra ← RX+1; if ((RX+1) != 0) PC ← R@

- *Accumulate memory vector*: Acumula a Rd la suma dels elements d'un vector, on l'adreça inicial del vector és Ra i el vector té Rb elements.
 - Sintaxi: ACCUMV Rd, Ra, Rb, Rc
 - Codificació: 1011 aaa bbb ddd ccc
 - Semàntica:
 $PC=PC+2; \quad Rc=Mem_w[Ra]; \quad Rd=Rd+Rc; \quad Ra=Ra+2; \quad Rb=Rb-1;$
 $\text{if } (Rb \neq 0) \quad PC=PC-2;$
- Observacions:
 - La instrucció utilitza Rc com a registre temporal
 - La instrucció també modifica Ra i Rb.
- Cal afegir hardware?
 - Sí, perquè hem de poder llegir els registres identificats per IR_{543} i IR_{210}
 - També hem de poder escriure el registre identificat per IR_{210}
 - També caldrà poder generar la constant "1"
 - Caldrà afegir multiplexors per a generar $@A$ i $@B$
 - Tindran senyals de control que haurà de generar la ROM OUT
 - Haurem de determinar el seu valor per als estats ja existents

- L'enunciat ens indica com queda la UC
 - No modifica UP
 - Quatre nous senyals a la ROM OUT: $Mx1$, $Mx@B$, $Mx@A0$, $Mx@A1$



- L'enunciat ens diu que caldran 6 estats de càlcul
- Cal omplir els forats amb una acció per forat

Nodo/Estado		Acciones	
Número	Mnem.		
E0	F	IR \leftarrow MEMw[PC] // PC \leftarrow PC+2	
E1	D	R@ \leftarrow PC+SE(N8)*2 // RX \leftarrow Ra	// RY \leftarrow Rb
E17	Acc1	R@ \leftarrow RX	
E18	Acc2	Rc \leftarrow MEMw[R@] // RX \leftarrow Ra	
E19	Acc3	Ra \leftarrow RX + 0x0002 // RX \leftarrow Rd	// RY \leftarrow Rc
E20	Acc4	Rd \leftarrow RX + RY	
E21	Acc5	R@ \leftarrow PC - 2 // RX \leftarrow Rb	
E22	Acc6	Rb \leftarrow RX - 1 // if(RX-1 != 0x0000) PC \leftarrow R@	

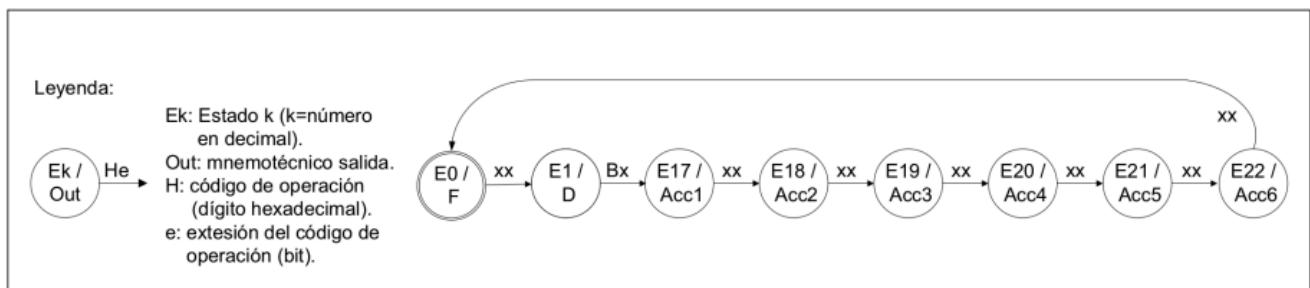
ACCUMV: ROM OUT



- Cal omplir les files i columnes indicades

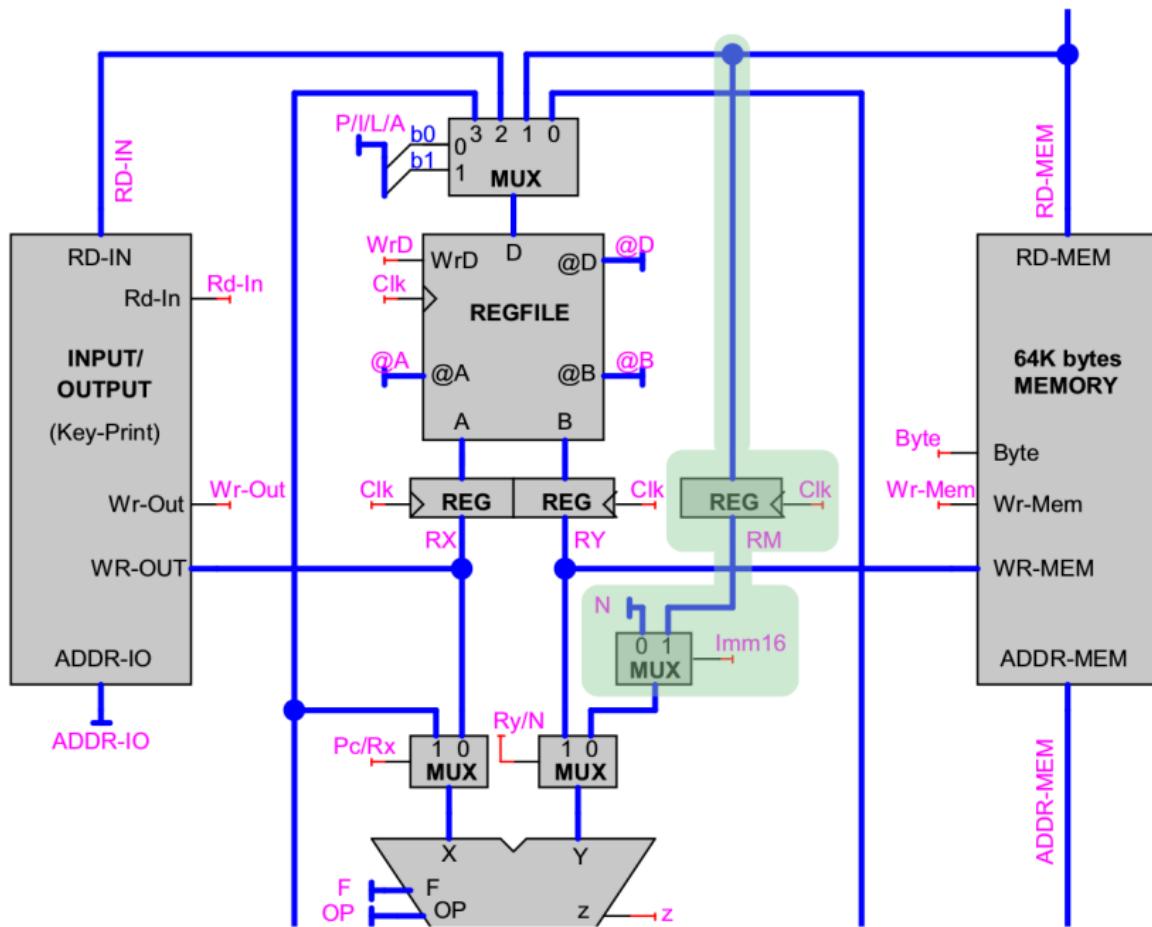
@ROM	Mx@A1	Mx@A0	Mx@B	Mx1	Bnz	Bz	WnMem	Rdn	WrOut	WrD	Ldir	Byte	R@/Pc	Alu/R@	Pc/Rx	Ry/N	P/I/L/A1	P/I/L/A0	OP1	OP0	MxN1	MxN0	MxF	F2	F1	F0	Mx@D1	Mx@D0																											
0	x	0																																																					
1	0	0																																																					
2	x	x																																																					
3	x	x																																																					
4	x	0																																																					
5	0	0																																																					
6	x	x																																																					
7	x	x																																																					
8	x	x																																																					
9	x	x																																																					
10	x	x																																																					
11	x	x																																																					
12	x	x																																																					
13	x	0																																																					
14	x	0																																																					
15	x	x																																																					
16	x	x																																																					
17	x	x																																																					
18	0	0	x	x	0	0	0	0	0	0	1	0	0	1	x	x	x	0	1	x	x	x	x	x	x	1	1																												
19	1	x	1	0	0	0	0	0	0	1	0	x	x	x	0	0	0	0	0	1	1	1	1	0	0	1	0																												
20		x	x																																																				
21	0	1	x	0	0	0	0	0	0	0	x	x	x	1	0	x	x	0	0	1	1	1	1	0	1	x	Acc5																												
22	x	x	x	1	1	0	0	0	0	1	x	x	x	0	0	0	0	0	0	x	x	1	1	0	1	0	1	23..31	x	x																									
23..31	x	x																																																					

- Cal indicar transicions entre els nous estats



- Indiqueu el contingut de l'adreça(ces) de la ROM Q+ que implementa(en) la transició de E1 a E17 ?
 - L'adreça 0x036 (00001 1011 0₂, 54₁₀) contindrà 0x11 (10001₂, 17₁₀)
 - L'adreça 0x037 (00001 1011 1₂, 55₁₀) contindrà 0x11 (10001₂, 17₁₀)

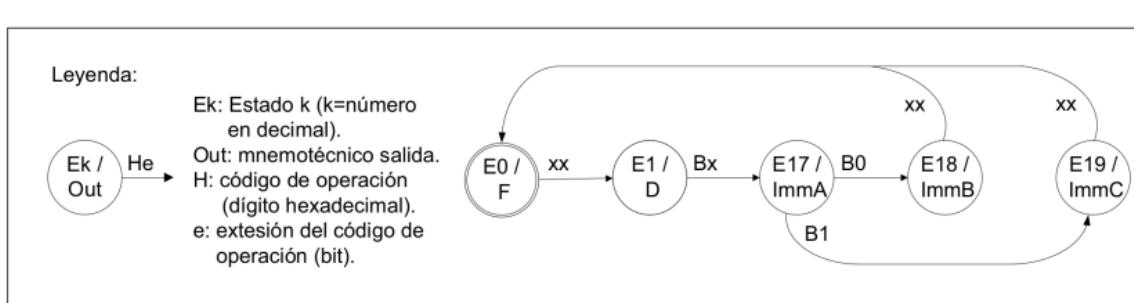
- Codificación: 1011 aaa e xx ddd fff nnnnnnnnnnnnnnnn con e = 0 para las operaciones AL y con e = 1 para las CMP. El campo fff codifica la operación a realizar de la misma manera que se codifica para las familias de instrucciones AL y CMP originales
- Sintaxis: mnemoI16 Rd, Ra, N16 siendo mnemo cualquiera de los mnemotécnicos de las operaciones AL o CMP originales.
 - Ejemplo de instrucción AL: SUBI16 Rd, Ra, N16.
 - Ejemplo de instrucción CMP: CMPEQUI16 Rd, Ra, N16.
- Semántica: Rd = Ra op N16 siendo op la operación AL o CMP que corresponde al mnemo (de la instrucción en ensamblador) o al campo fff (de la instrucción en lenguaje máquina).



mnemoI16: acciones a cada estat

- Completad el contenido de las cajas vacías de la siguiente tabla que indica, mediante una fila para cada nodo del grafo de estados de la unidad de control, la acción (o acciones en paralelo) que se realiza en el computador en cada uno de los nodos que se requieren para ejecutar las nuevas instrucciones (Fetch, Decode, y los 3 nodos nuevos): F, D, ImmA, ImmB e ImmC.
- Para especificar las acciones se usa el mismo lenguaje de transferencia de registros que en la documentación.

Nodo/Estado	Número	Mnem.	Acciones
E0		F	IR \leftarrow MEMw[PC] // PC \leftarrow PC+2
E1		D	R@ \leftarrow PC+SE(N8)*2 // RX \leftarrow Ra // RY \leftarrow Rb
E17		ImmA	RM \leftarrow MEMw[PC] // PC \leftarrow PC+2 // RX \leftarrow Ra
E18		ImmB	Rd \leftarrow RX A1 RM
E19		ImmC	Rd \leftarrow RX Cmp RM



- Indicad la direcció o les direccions (en binari amb x quan sigui possible per referir-nos a més d'una direcció) de la memòria ROM_Q+ i el seu contingut (en hexa) per implementar correctament el pas del node/estat E1 (D) al E17 (ImmA) i del E17 (ImmA) al E18 (ImmB).
 - D a E17: A les adreces 00001 1011 x, el contingut ha de ser 0x11
 - E17 a E18: a l'adreça 10001 1011 0 el contingut ha de ser 0x12
 - Observació: també seria vàlida intercanviar el paper d'E18 i E19, amb el que existiria una altra solució vàlida a tots els apartats

- Assumint que la secció de dades es carrega a 0xA000 i a continuació la de codi, indiqueu el valor de l'etiqueta vector i el contingut de l'adreça 0xA06C del següent programa?

```

N = 24225

.data
0xA000 .space 2
vector: .space 100, 0xFF

.text
0xA066 MOVI R0, lo(N)
0xA068 MOVHI R0, hi(N)

0xA06A MOVI R1, lo(vector)
0xA06C MOVHI R1, hi(vector)

```

- $\text{vector} = 0xA002$
- $\text{Mem}_w[0xA06C] = 0x93A0$ (codificació de `MOVHI R1, 0xA0`)

201819Q1-E4

El programa ensamblador de la derecha se ha traducido a lenguaje máquina para ser ejecutado en el SISC Von Neumann, situando la sección .data a partir de la dirección 0xA000 de memoria y justo a continuación la sección .text.

a) Una vez cargado el programa en memoria:

- ¿A qué dirección de memoria corresponden las etiquetas, o direcciones simbólicas, L1 y V2? (0,5 puntos)

L1 = 0xA026	V2 = 0xA010
-------------	-------------

- ¿Cuál es el word almacenando en las siguientes direcciones de memoria? (0,5 puntos)

Mem _w [0xA010] = 0x0607
Mem _w [0xA02A] = 0x8B06

b) Una vez ejecutado el programa en el computador SISC Von Neumann ¿Cuál es la dirección de memoria escrita por la instrucción ST? ¿Cuál es el valor escrito? (0,75 puntos)

Mem _w [0xA018] = 0x0380

c) Indicad el número total de instrucciones que ejecuta el programa, así como cuántas son lentas y cuántas son rápidas (0,25 puntos)

$N_{total} = 75$	$N_{lentas} = 16$	$N_{rápidas} = 59$
------------------	-------------------	--------------------

```

.data
V1: .word 1, 2, 4, 8
      .word 16, 32, 64
      .word -1
V2: .byte 7, 6, 5, 4
      .byte 3, 2, 1
      .even
V3: .word 0

.text
      MOVI R0, lo(V1)
      MOVHI R0, hi(V1)
      MOVI R1, lo(V2)
      MOVHI R1, hi(V2)
      MOVI R2, 0
      MOVI R3, 0xFF
L1:   LD R4, 0(R0)
      CMPEQ R5, R3, R4
      BNZ R5, L2
      LDB R6, 0(R1)
      SHL R4, R4, R6
      ADD R2, R2, R4
      ADDI R0, R0, 2
      ADDI R1, R1, 1
      BZ R5, L1
      MOVI R7, lo(V3)
      MOVHI R7, hi(V3)
      ST 0(R7), R2
.end

```