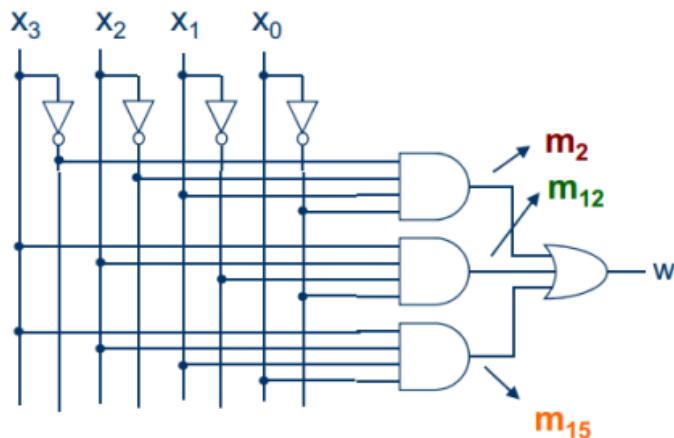


## Exemple: 4 entrades i 1 sortida

$x_3\ x_2\ x_1\ x_0$	w
0 0 0 0	0
0 0 0 1	0
<b>0 0 1 0</b>	<b>1</b>
0 0 1 1	0
0 1 0 0	0
0 1 0 1	0
0 1 1 0	0
0 1 1 1	0
1 0 0 0	0
1 0 0 1	0
1 0 1 0	0
1 0 1 1	0
<b>1 1 0 0</b>	<b>1</b>
1 1 0 1	0
1 1 1 0	0
<b>1 1 1 1</b>	<b>1</b>

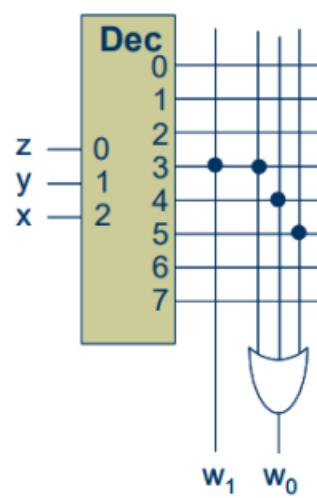


- $w = m_2 + m_{12} + m_{15} = !x_3 \cdot !x_2 \cdot x_1 \cdot !x_0 + x_3 \cdot x_2 \cdot !x_1 \cdot !x_0 + x_3 \cdot x_2 \cdot x_1 \cdot x_0$
- El CLC utilitza 4 portes NOT, 3 portes AND-4 i 1 porta OR-3
- Què faríeu si a la TV hi hagués alguna sortida a X (*Don't care*)?

## Exemple: Síntesi CLC amb dues sortides

- L'ordre de les variables d'entrada a la TV (xyz) ha de correspondre's amb el pes de les entrades al bloc Dec-3-8
  - x amb pes 2, y amb pes 1, z amb pes 0
- Sortides:
  - Com la sortida  $w_1$  només utilitza un *minterm*, no cal porta OR per sintetitzar-lo
  - Per sintetitzar  $w_0$  cal una porta OR-3

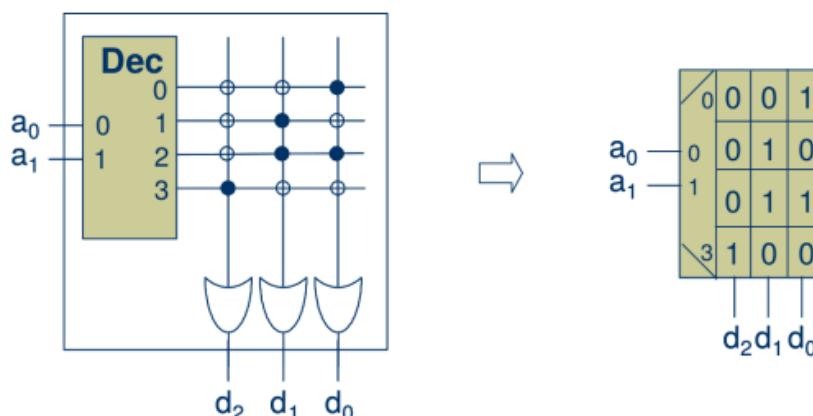
x	y	z	$w_1$	$w_0$
0	0	0	0	0
0	0	1	0	0
0	1	0	0	0
0	1	1	1	1
1	0	0	0	1
1	0	1	0	1
1	1	0	0	0
1	1	1	0	0



- Sintetitzar un CLC amb 2 entrades i 3 sortides amb la següent TV

$a_1$	$a_0$	$d_2$	$d_1$	$d_0$
0	0	0	0	1
0	1	0	1	0
1	0	0	1	1
1	1	1	0	0

- Mostrem el connexionat així com la representació compacta
  - Vigileu amb els pesos dels senyals i l'ordre de les files!
  - A la representació compacta incloem la taula de veritat del CLC



## Exercici 1617Q1-E1

a) ¿Cuántas puertas And y Or y de cuantas entradas cada una hacen falta para implementar directamente la expresión en suma de minterms de la función w de la siguiente tabla de verdad

a	b	c	w
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1

Número puertas AND = 4 de 3 entradas.

Número puertas OR = 1 de 4 entradas.

b) Especificar el tamaño mínimo de la ROM para sintetizar un circuito de 4 entradas y 3 salidas.

Número de palabras =  $2^4 = 16$

Bits por palabra = 3

Dado el esquema del siguiente circuito (incluida la tabla de verdad del bloque C1) completad la tabla de verdad de la salida W y esribid la expresión lógica en suma de minterms.

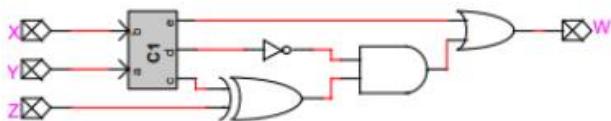


Tabla de verdad de C1

a	b	c	d	e
0	0	1	1	0
0	1	0	0	0
1	0	1	0	1
1	1	1	1	0

Tabla de verdad de W:

X	Y	Z	W
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	0

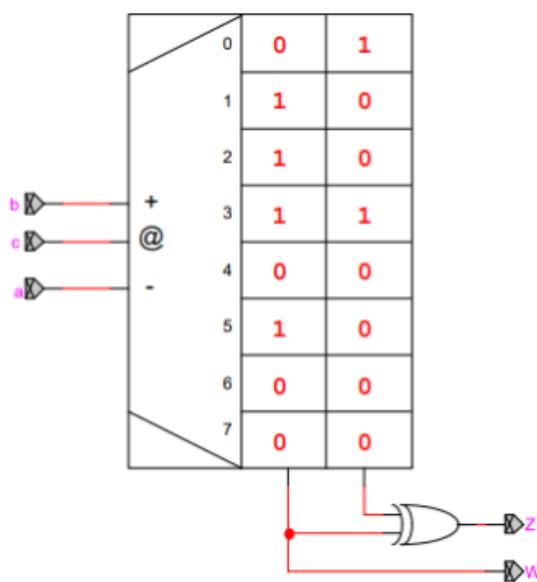
Expresión en suma de minterms de W:  $\bar{X} \cdot \bar{Y} \cdot \bar{Z} + \bar{X} \cdot Y \cdot Z + X \cdot \bar{Y} \cdot Z$

# Exercici 1819Q2-E1

- Queremos implementar un circuito combinacional cuya función/comportamiento está descrito en la siguiente tabla de verdad. Para implementarlo ya tenemos el circuito diseñado. Rellena el contenido de la ROM del circuito para que haga la función descrita en la tabla de verdad

Tabla de verdad

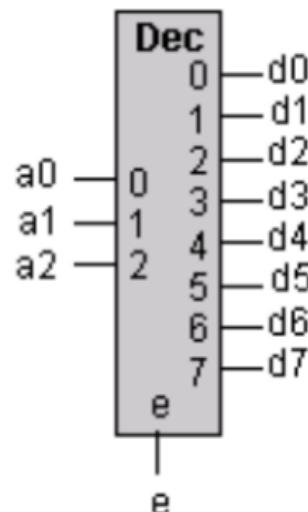
a	b	c	z	w
0	0	0	1	0
0	0	1	1	1
0	1	0	0	0
0	1	1	0	0
1	0	0	1	1
1	0	1	0	1
1	1	0	1	1
1	1	1	0	0



- Afegeix una entrada extra al Dec- $n-2^n$ , la senyal *enable*
  - Si val 0, totes les sortides del Decoder valdran 0
  - Si val 1, el Decoder operarà amb normalitat
- Exemple: Dec-3-8 amb senyal *enable*

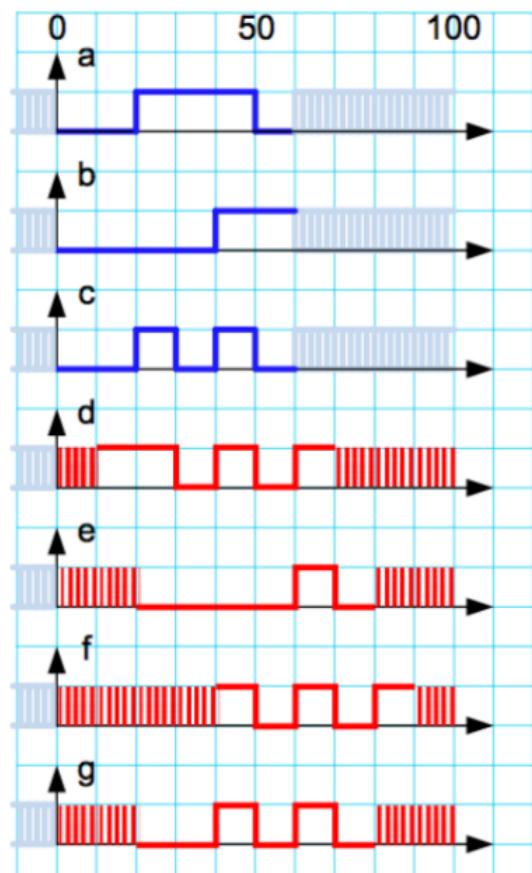
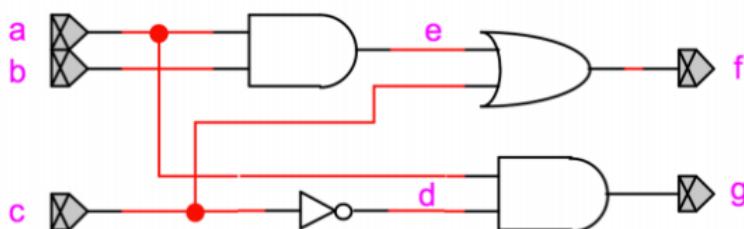
Truth Table (compressed):

e	a <sub>2</sub>	a <sub>1</sub>	a <sub>0</sub>	d <sub>7</sub>	d <sub>6</sub>	d <sub>5</sub>	d <sub>4</sub>	d <sub>3</sub>	d <sub>2</sub>	d <sub>1</sub>	d <sub>0</sub>
0	X	X	X	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	1
1	0	0	1	0	0	0	0	0	0	1	0
1	0	1	0	0	0	0	0	0	1	0	0
1	0	1	1	0	0	0	0	1	0	0	0
1	1	1	0	0	0	0	1	0	0	0	0
1	1	0	1	0	0	1	0	0	0	0	0
1	1	1	0	0	1	0	0	0	0	0	0
1	1	1	1	1	0	0	0	0	0	0	0



## Exercici 1415Q1 E1

Completa el siguiente cronograma de las señales del esquema lógico sabiendo que los tiempos de propagación de las puertas son:  $T_p(\text{Not}) = 10 \text{ u.t.}$ ,  $T_p(\text{And}) = T_p(\text{Or}) = 20 \text{ u.t.}$ . Debéis operar adecuadamente con las zonas sombreadas (no se sabe el valor que tienen) y dibujar la señal sombreada cuando no se pueda saber si vale 0 o 1.

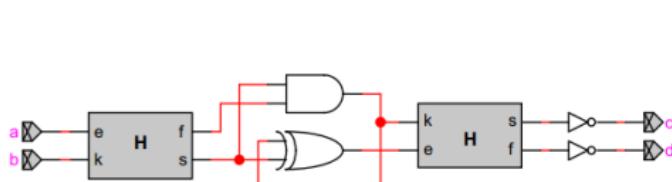


Dado el esquema del siguiente circuito (incluida la tabla de verdad del bloque H),

- Compleudad la tabla de verdad de las salidas c y d y escribid la expresión lógica en suma de minterms de c. (1 punto)
- Escribid el camino crítico (o uno de ellos si hay varios) y el tiempo de propagación desde la entrada a hasta la salida c. Se dan los tiempos de propagación de H (en la tabla) y de las puertas:  $T_p(\text{Not}) = 10$ ,  $T_p(\text{And}) = 20$ ,  $T_p(\text{Or}) = 30$  y  $T_p(\text{Xor}) = 50$  u.t. Por ejemplo, uno de los caminos de b a d se especificaría como: b - k - s - Xor - e - f - Not - d. (1 punto)

T.V. bloque H

e	k	f	s
0	0	1	1
0	1	1	0
1	0	0	1
1	1	1	1



a	b	c	d
0	0	1	0
0	1	0	0
1	0	0	1
1	1	1	0

Tp bloque H

T <sub>p</sub>	f	s
e	60	50
k	90	80

Expresión en suma de minterms de c:  $\neg a \cdot \neg b + ab$

Camino crítico de a a c: a - e - f - And - Xor - e - s - Not - c  $T_{p_{a-c}}: 190$  u.t.

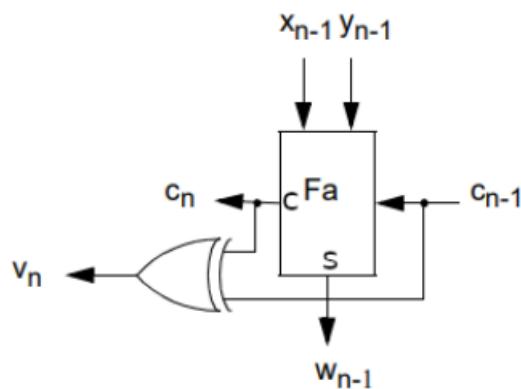
## No representativitat a la suma d'enters



- Afegirem una nova sortida,  $v_n$ , al bloc ADD

- $v_n = 1$  indicarà que el resultat de la suma d'enters amb aritmètica convencional no és representable en Ca2 de  $n$  bits
- Cal detectar si els operands són del mateix signe però el resultat és de signe diferent
  - Els operands són positius però el resultat és negatiu:  $x_{n-1} \cdot y_{n-1} \cdot w_{n-1}$
  - Els operands són negatius però el resultat és positiu:  $x_{n-1} \cdot y_{n-1} \cdot \neg w_{n-1}$
- $v_n = x_{n-1} \cdot y_{n-1} \cdot \neg w_{n-1} + \neg x_{n-1} \cdot y_{n-1} \cdot w_{n-1}$
- Amb la TV es pot demostrar que  $v_n$  és equivalent a  $c_n \text{ XOR } c_{n-1}$

$x_{n-1}$	$y_{n-1}$	$c_{n-1}$	$c_n$	$w_{n-1}$	$v_n$
0	0	0	0	0	0
0	0	1	0	1	1
0	1	0	0	1	0
0	1	1	1	0	0
1	0	0	0	1	0
1	0	1	1	0	0
1	1	0	1	0	1
1	1	1	1	1	0

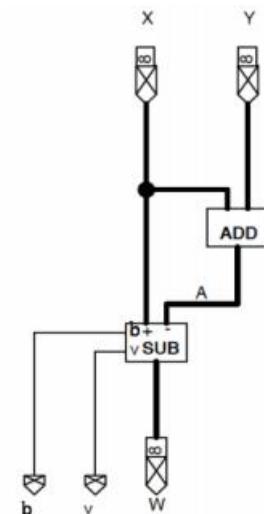


# Exercici 1516Q1E2



Dado el siguiente circuito CLC y los vectores de 8 bits de entrada X y Y, completad la siguiente tabla.

X	Y	A	W	Wu	Ws	b	v
00000000	11111111	11111111	00000001	1	1	1	0
11111111	00000001	00000000	11111111	255	-1	0	0
10000000	10101010	00101010	01010110	86	86	0	1

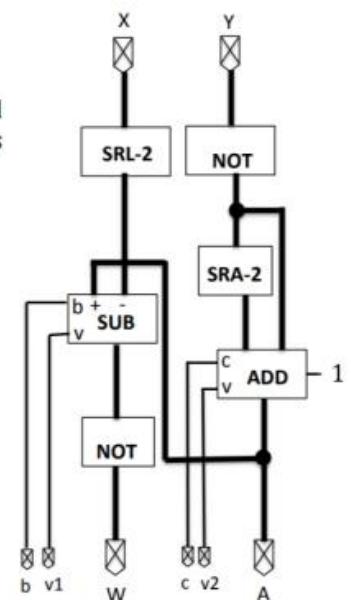


# Exercici 1819Q1E2

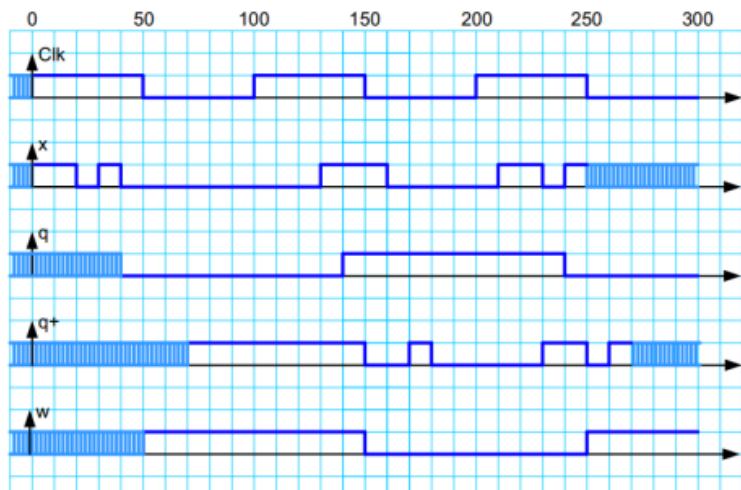


Dado el esquema del CLC a bloques de la derecha, completad la siguiente tabla que indica el valor de las salidas del circuito para cada uno de los cuatro casos concretos de valores de las entradas (un caso por fila).

X	Y	b	v1	c	v2	W	A
0000 0000	1111 1111	0	0	0	0	11111110	00000001
1000 0001	1111 1100	1	0	0	0	00011011	00000100
1000 0000	0111 1111	0	0	1	1	10111110	01100001
1111 1111	1000 0000	0	1	0	1	10011111	10011111

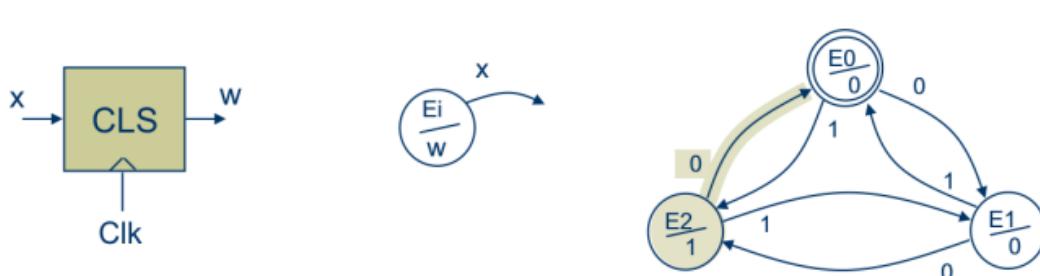


- Per fer el graf d'estats caldria construir les taules de veritat del CLC que calcula les sortides i del CLC que actualitza l'estat



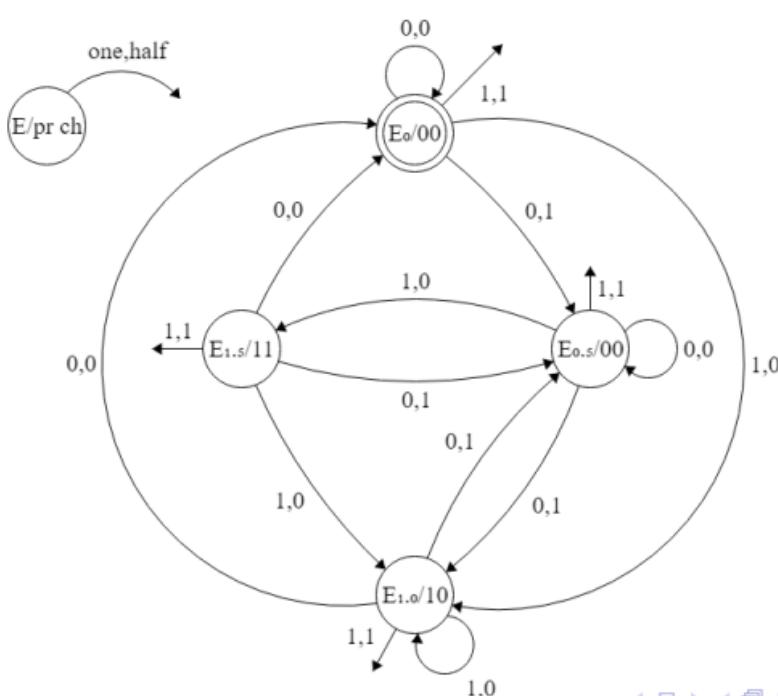
## Seguiment del graf d'estats: exercici 1

- A partir del graf d'estats, podem completar el cronograma simplificat que mostra l'evolució de l'estat i de la sortida a mesura que l'entrada canvia de valor.
  - Exemple on  $n = 1, m = 1, k = 2$



Núm. Ciclo	00	01	02	03	04	05	06	07	08	09	10	11	12	13	14
Estado	E0	E2	E0	E1	E2	E0	E1	E0	E2	E1	E2	E0	E2	E0	E2
Entrada x	1	0	0	0	0	0	1	1	1	0	0	1	0	1	0
Salida w	0	1	0	0	1	0	0	0	1	0	1	0	1	0	1

- Dibuixeu el graf d'estats d'un CLS corresponent al mecanisme de control d'una màquina de venda de productes de 1€ que admeti monedes de 1€ i de 0,5€ tal que:
    - Entrades:
      - *half*: val 1 si en aquest cicle s'ha introduït una moneda de 0,50€
      - *one*: val 1 si en aquest cicle s'ha introduït una moneda de 1€
      - Podeu assumir que *one* i *half* no poden valer 1 simultàniament
    - Sortides:
      - *product* (pr): valdrà 1 durant un cicle si s'ha rebut un mínim de 1€
      - *change* (ch): valdrà 1 durant un cicle si cal retornar canvi (0,50€)
    - Un cop que el CLS notifica que cal servir un producte (i, si cal, retornar canvi), el CLS es disposarà a rebre monedes per a un nou producte.
  - Orientacions:
    - Determineu quins estats necessitaríeu i les transicions entre ells en funció de les entrades rebudes
      - Amb  $n$  senyals d'entrada, de cada node han de sortir  $2^n$  arestes
    - No oblideu dibuixar la llegenda i determinar l'estat inicial



# Construir graf d'estats (E2 Q2 18/19)



Completeu el graf d'estats d'un circuit seqüencial amb dos entrades d'1 bit  $a$  i  $b$  i una sortida  $S$  de 2 bits amb el següent funcionament:

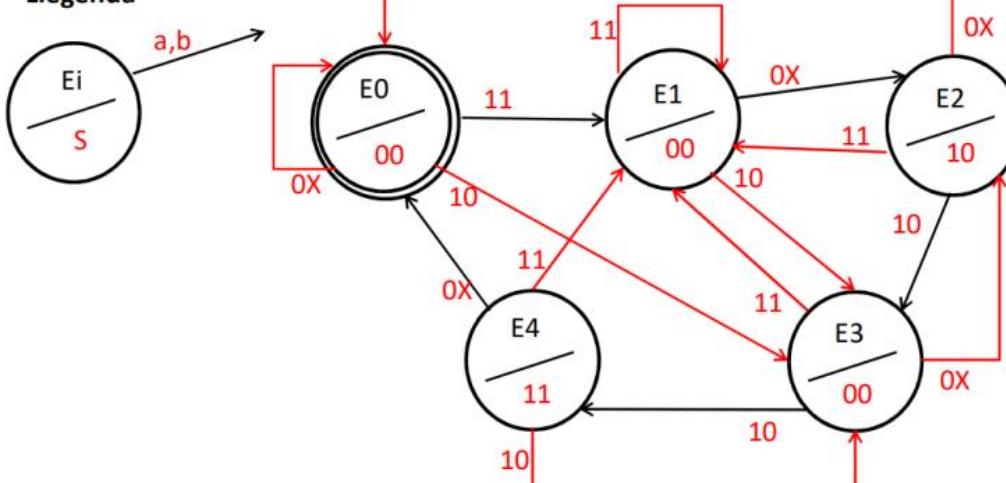
La sortida  $S$ , de dos bits, ha de valdre 10 durant 1 cicle si per l'entrada  $a$  arriba la seqüència 10, i ha de valdre 11 si per l'entrada  $a$  arriba la seqüència 11 al mateix temps que arriba la seqüència 00 per l'entrada  $b$ . En qualsevol altre cas la sortida  $S$  ha de valdre 00.

El reconeixement de les seqüències s'ha de fer sense encavalcament.

A continuació es mostra un exemple de funcionament del circuit, en que cada columna correspon a un cicle de rellotge:

<b>a</b>	1	0	0	0	0	0	1	1	0	0	1	1	1	0	1	1	0	1
<b>b</b>	1	1	1	1	0	0	0	0	0	0	0	1	1	0	0	1	1	
<b>S</b>	00	00	10	00	00	00	00	11	00	00	00	11	00	10	00	11	00	

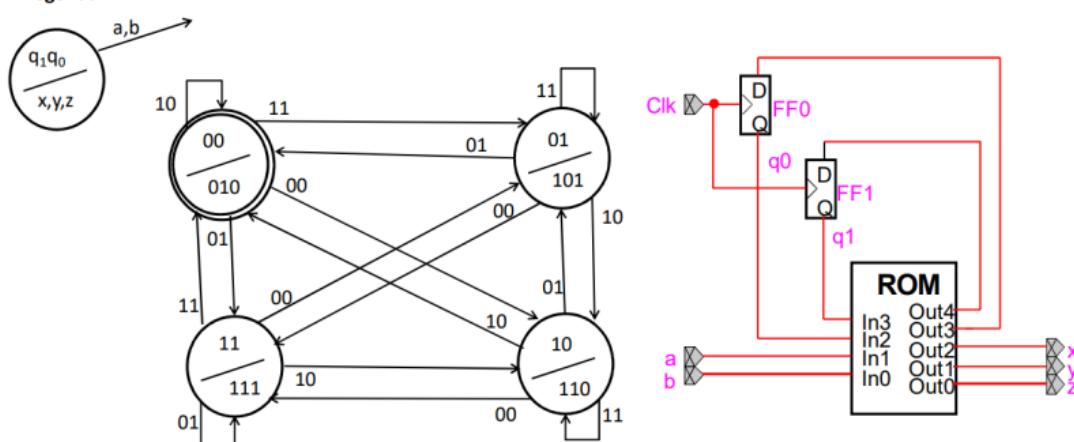
Llegenda



# Síntesi CLS's (E2 Q2 18/19)



- Si es vol implementar el graf d'estats següent amb un circuit seqüencial amb una sola ROM tal com es mostra en la figura.



- Indiqueu en hexa el contingut de les següents adreces de la ROM:

- ROM[0x4]=0x1D
- ROM[0xA]=0x06
- ROM[0xE]=0x17

# Completar graf d'estats i taula de sortides



Grafo incompleto de la UC

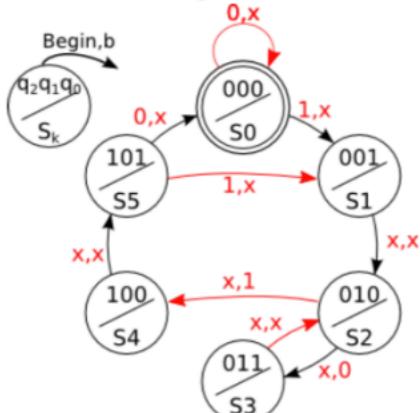


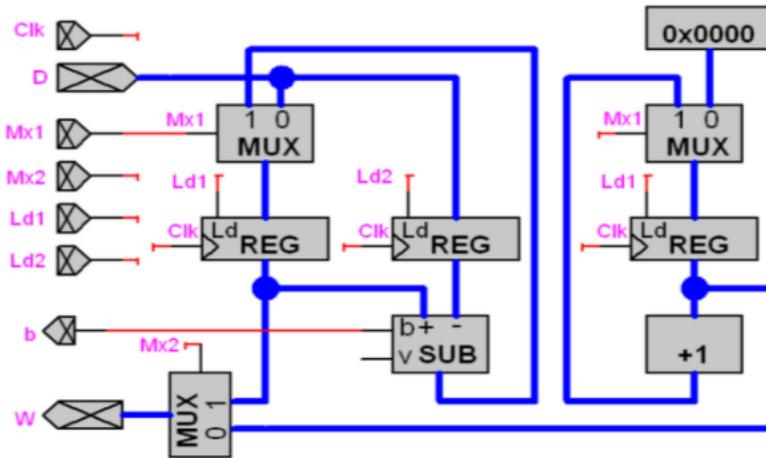
Tabla de salidas

	S0	S1	S2	S3	S4	S5
Mx1	0	x	x	1	x	0
Mx2	x	x	x	x	0	1
Ld1	1	0	0	1	0	1
Ld2	x	1	0	0	x	x
Done	0	0	0	0	1	1

Ejemplo: algoritmo de división implementado por el PPE

$$\begin{array}{l} n = 11 \\ d = 5 \end{array} \rightsquigarrow \left\{ \begin{array}{l} (11 \geq 5) \quad 11 - 5 = 6 \\ (6 \geq 5) \quad 6 - 5 = 1 \\ (1 \not\geq 5) \end{array} \right\} \Rightarrow \begin{array}{l} q = 2 \\ r = 1 \end{array}$$

Esquema completo de la Unidad de Proceso (UP) del PPE



- Hi ha alguna transició impossible al graf d'estats?

## Contingut ROM UC ( $k=3$ , $n=2$ , $m=5$ )

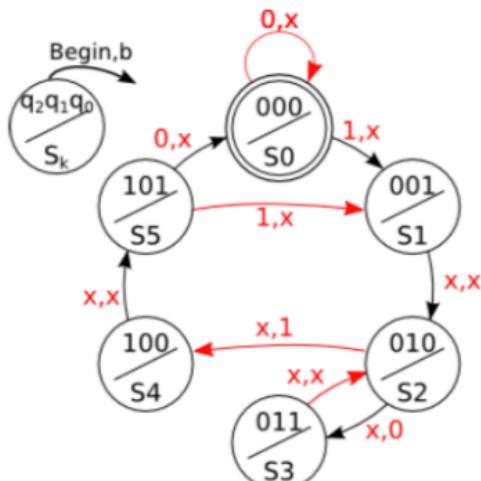
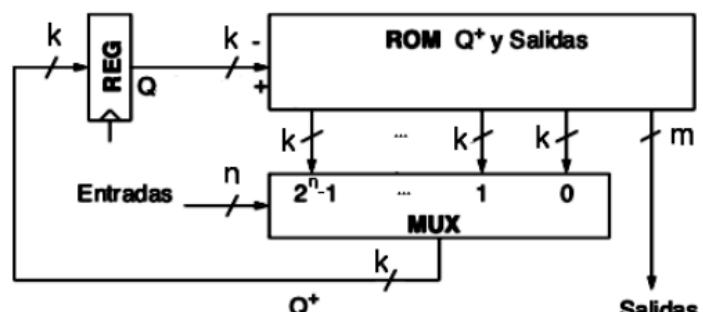


Tabla de salidas

	S0	S1	S2	S3	S4	S5
Mx1	0	x	x	1	x	0
Mx2	x	x	x	x	0	1
Ld1	1	0	0	1	0	1
Ld2	x	1	0	0	x	x
Done	0	0	0	0	1	1



Q	Q+				Salidas
	11	10	01	00	
0 0 0	0 0 1	0 0 1	0 0 0	0 0 0	0 x 1 x 0
0 0 1	0 1 0	0 1 0	0 0 1	0 0 1	x x 0 1 0
0 1 0	1 0 0	0 1 1	1 0 0	0 1 1	x x 0 0 0
0 1 1	0 1 0	0 1 0	0 1 0	0 1 0	1 x 1 0 0
1 0 0	1 0 1	1 0 1	1 0 1	1 0 1	x 0 0 x 1
1 0 1	0 0 1	0 0 1	0 0 0	0 0 0	0 1 1 x 1
1 1 0	x x x	x x x	x x x	x x x	x x x x x
1 1 1	x x x	x x x	x x x	x x x	x x x x x

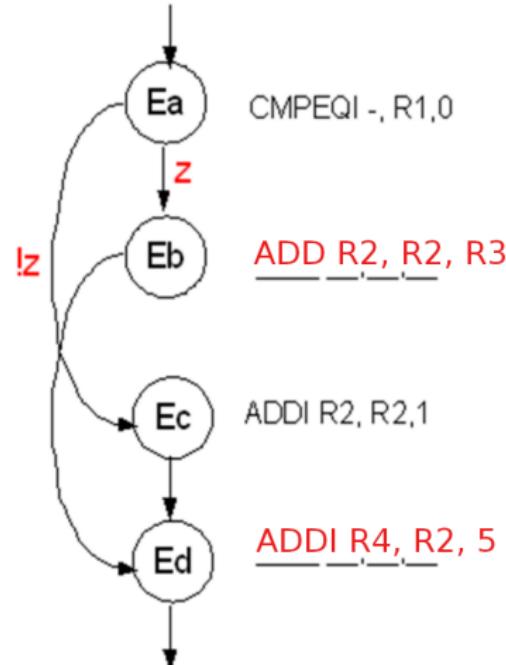
- Completeu els mnemotècnics i les etiquetes  $z$ !/z al graf d'estats:

```

    :
    :
    if (R1 != 0)
        R2 = R2 + R3;
    else
        R2 = R2 + 1;

    R4=R2+5;
    :
    :

```



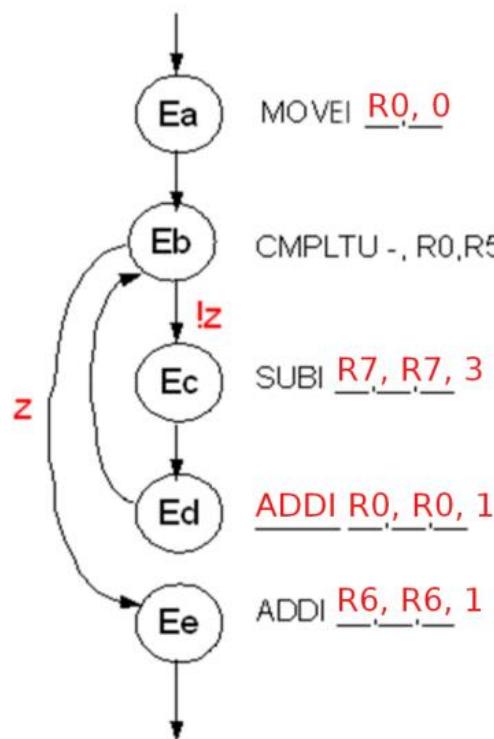
- Completeu els mnemotècnics i les etiquetes  $z$ !/z al graf d'estats:

```

    :
    :
    for (R0 = 0; R0 < R5; R0++)
    {
        R7 = R7 - 3;
    }

    R6++;
    :
    :

```



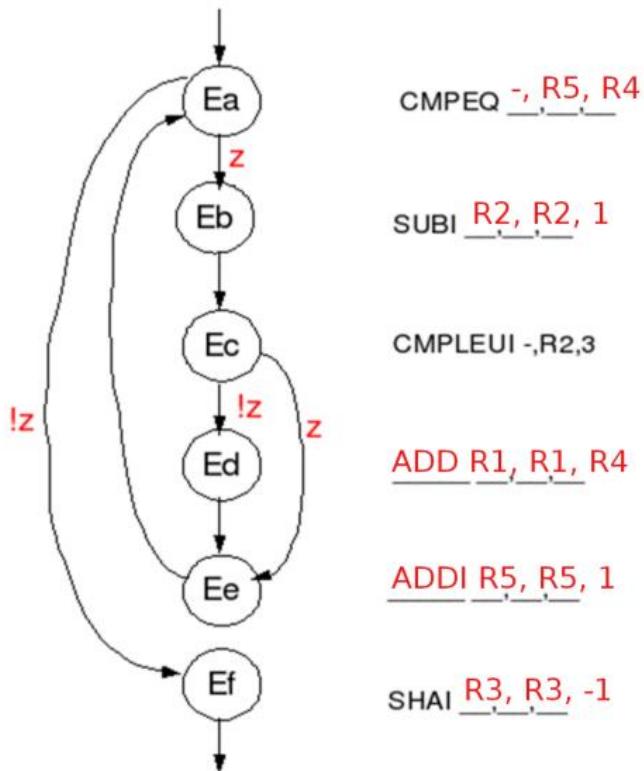
## Exercici 8.5d

- Completeu els mnemotècnics i les etiquetes  $z$  /  $!z$  al graf d'estats:

while ( $R5 \neq R4$ )

```
{
    R2 = R2-1;
    if ( $R2 \leq 3$ ) R1= R1+R4;
    R5++;
}
```

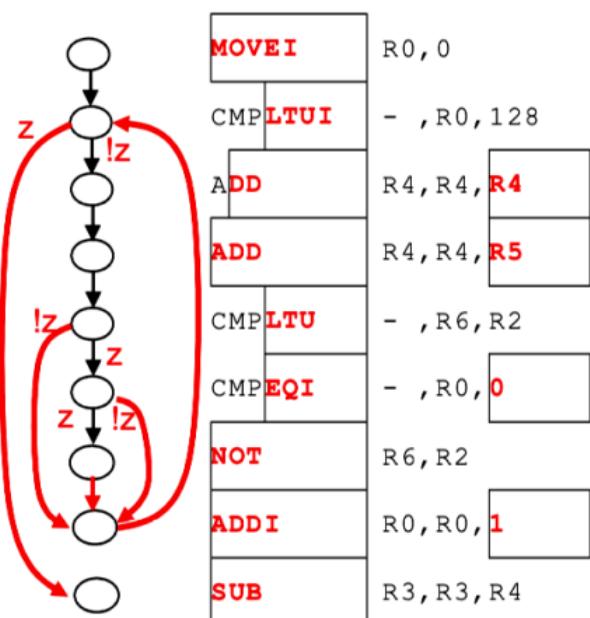
$R3 = R3/2;$



## Parcial E3 Q1 16-17

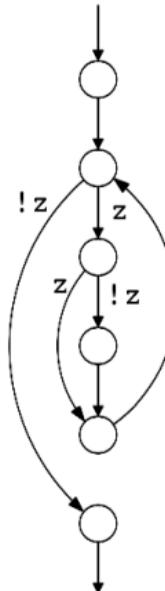
- Completeu (arcs, etiquetes i mnemotècnics) el fragment graf d'estats de la UC perquè, juntament amb la UPG, implementi el fragment de codi indicat.
- Les dades són de tipus natural.

```
for (R0=0; R0<128; R0++) {
    R4=R4*2+R5;
    if ((R6>=R2) && (R0!=0))
        R6=not(R2);
}
R3=R3-R4;
```



- Completeu (arcs, etiquetes i mnemotècnics) el fragment graf d'estats de la UC perquè, juntament amb la UPG, implementi el fragment de codi indicat.
- Les dades són de tipus enter.

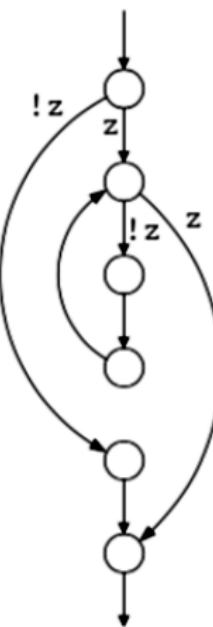
```
for(R1=133; R1>-133; R1--) {
    // R1<0> es el bit
    // de menys pes de R1
    if (R1<0>==1)
        R2=R2+R1
}
R2=R2/4;
```



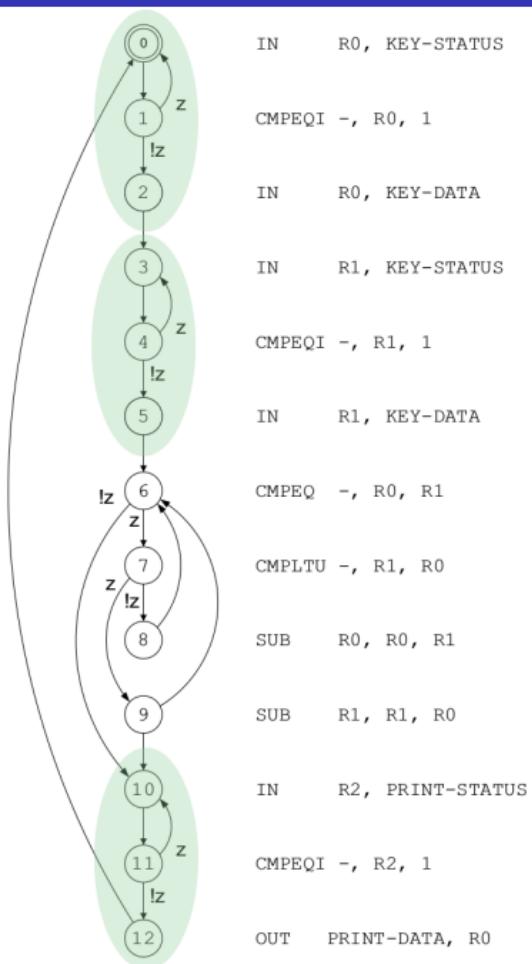
<b>MOVEI</b>	R1,	<b>133</b>
<b>CMPLEI</b>	-,	<b>R1, -133</b>
<b>ANDI</b>	-,	<b>R1, 1</b>
<b>ADD</b>	R2,	<b>R2, R1</b>
<b>SUBI</b>	R1,	<b>R1, 1</b>
<b>SHAI</b>	R2,	<b>R2, -2</b>

- Completeu (arcs, etiquetes i mnemotècnics) el fragment graf d'estats de la UC perquè, juntament amb la UPG, implementi el fragment de codi indicat.
- Les dades són de tipus natural.

```
if (R3 > 50) {
    while (R0 >= R2) {
        R0 = R0 / 2;
        R3 = R3 - 1;
    }
} else {
    // | es OR bit a bit
    R3 = R2 | R3;
}
R4 = R0 + R3;
```



<b>CMPLEUI</b>	-,	<b>R3, 50</b>
<b>CMPLEU</b>	-,	<b>R2, R0</b>
<b>SHLI</b>	<b>R0,</b>	<b>R0, -1</b>
<b>SUBI</b>	<b>R3,</b>	<b>R3, 1</b>
<b>OR</b>	<b>R3,</b>	<b>R2, R3</b>
<b>ADD</b>	<b>R4,</b>	<b>R0, R3</b>



## 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	1   0   1	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	x   x   x   x	0   1   0	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	0   0   0   1	1   0   0	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

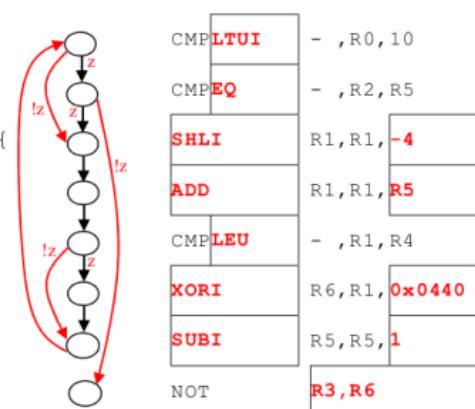
## Canvis a l'estat del computador

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

## 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 $\leftrightarrow$ paraula control

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

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

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

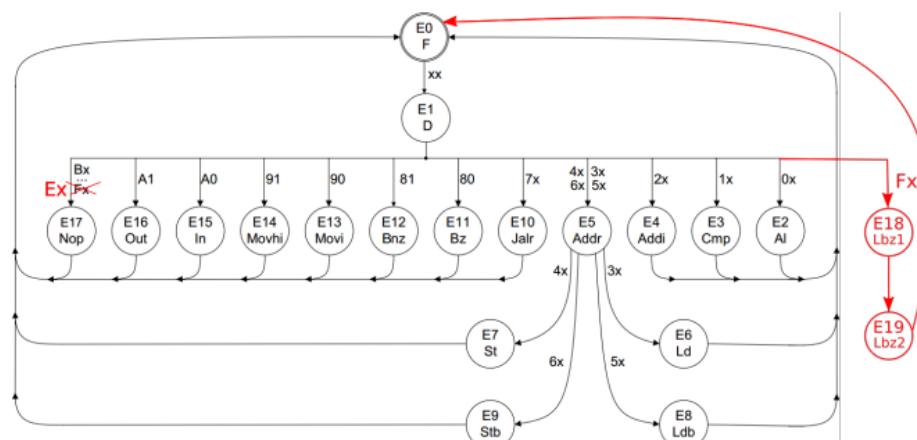
## Exercici: afegir noves instruccions a SISA



- 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@ ← RY RX ← Ra
  - Lbz2: if (RX == 0) PC ← R@;
- Modificacions a les ROM's:
  - ROM Q+: reflectir les noves transicions
  - ROM OUT: per generar les paraules de control

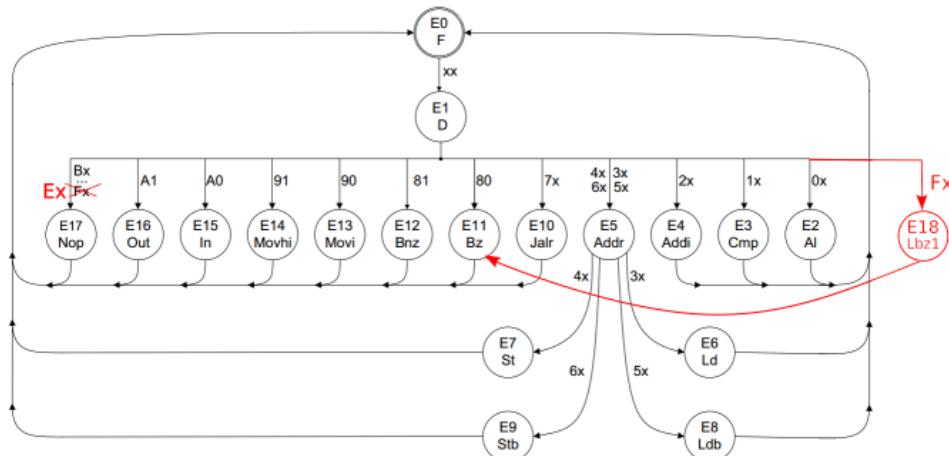
- Graf d'estats UC:



- 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	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	
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

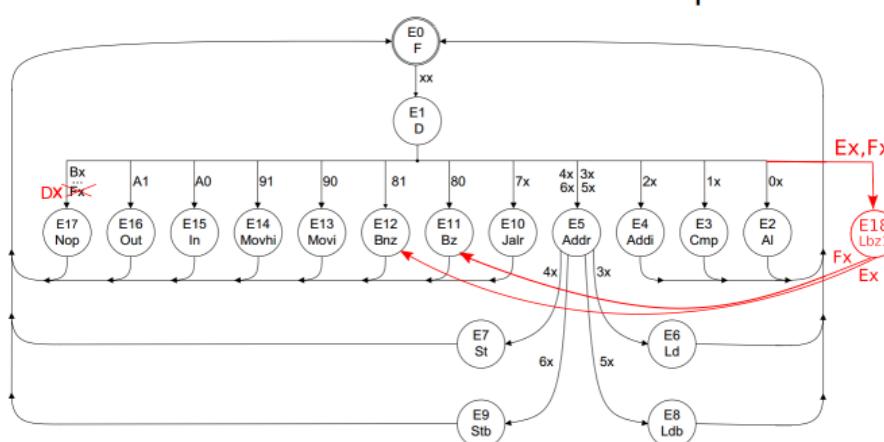
- Els estats Lbz2 i Bz són idèntics!!!
  - Des de Lbz1 podem passar a l'estat Bz, l'estat Lbz2 és redundant
- Graf d'estats UC:



- Modificacions a la ROM Q+
  - $\text{ROM\_Q+}[00001\ 1111\ x] = 0x12$  i  $\text{ROM\_Q+}[10010\ 1111x] = 0x0B$
- Modificacions a la ROM OUT

@ROM	Acciones asociadas al estado (en lenguaje de transferencia de registros)																						
	Bnz	Bz	WrMem	WrOut	WrD	Ldir	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



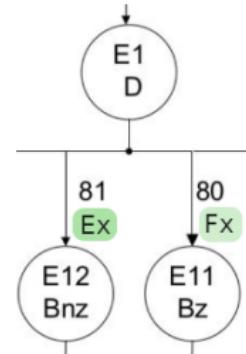
- Modificacions a la ROM Q+ (a més de les anteriors)
  - $\text{ROM\_Q+}[00001\ 1110\ x] = 0x12$  i  $\text{ROM\_Q+}[10010\ 1110x] = 0x0C$
- Modificacions a la ROM OUT (a més de les anteriors)
  - Cap

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

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

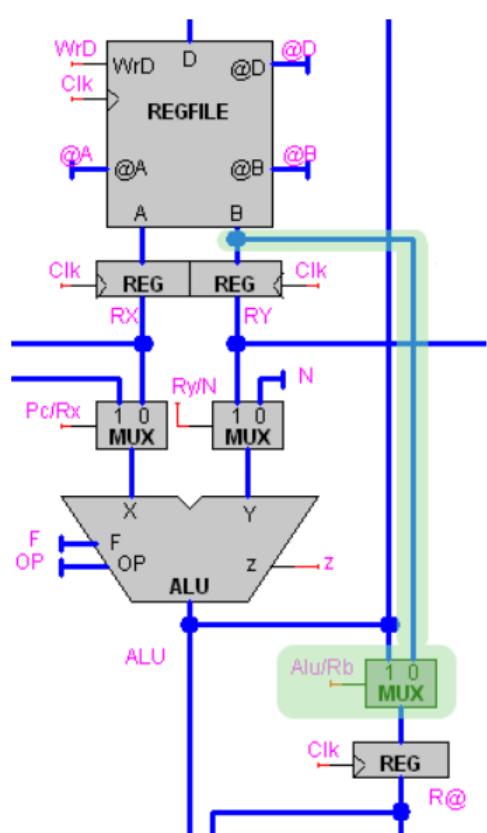
- Canvis al graf d'estats:

- 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
  - $\text{ROM\_Q} + [00001\ 1110\ x] = 0xC$
  - $\text{ROM\_Q} + [00001\ 1111\ x] = 0xB$



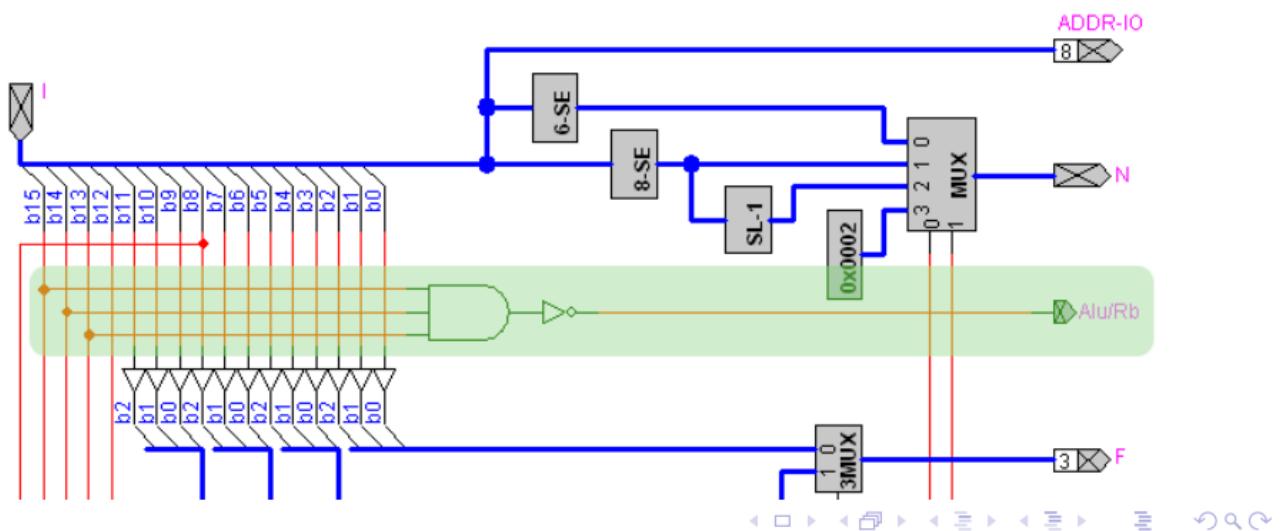
## Modificacions a la UPG

- 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)

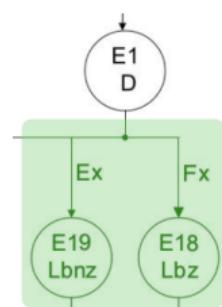


- Modificacions a la Lògica de Control:

- 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

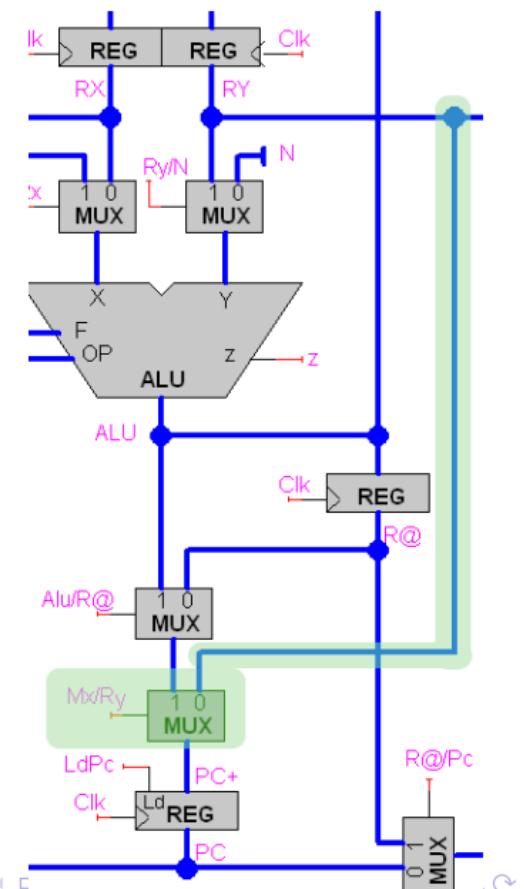


- La segona estratègia de modificació *hardware* serà afegir un camí des de RY (o Rb) al registre PC sense passar per la ALU.
  - D'aquesta forma, if (RX == 0) PC ← RY; es podrà fer en un cicle perquè l'ALU només ha de calcular bit z
- Canvis al graf d'estats:
  - Necessitarem dos estats nous (Lbz i Lbnz) per a activar el nou camí i actuar en funció de z
    - Lbz: if (RX == 0) PC ← RY;
    - Lbnz: if (RX != 0) PC ← 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



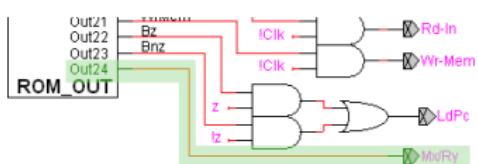
#### • Modificacions a la UPG

- 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)

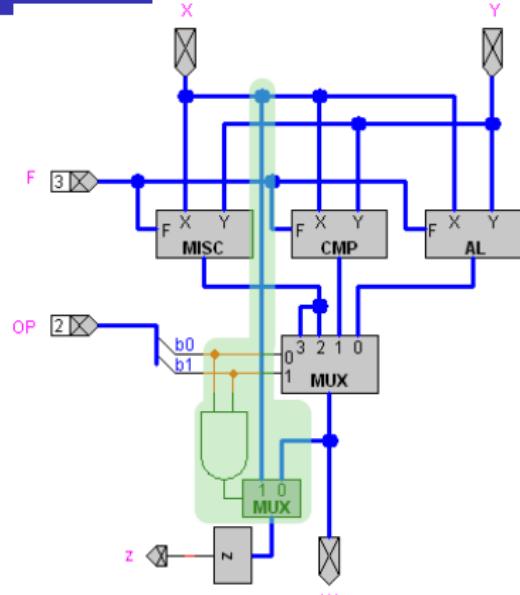


#### • Modificacions a la Lògica de Control:

- 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



- 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



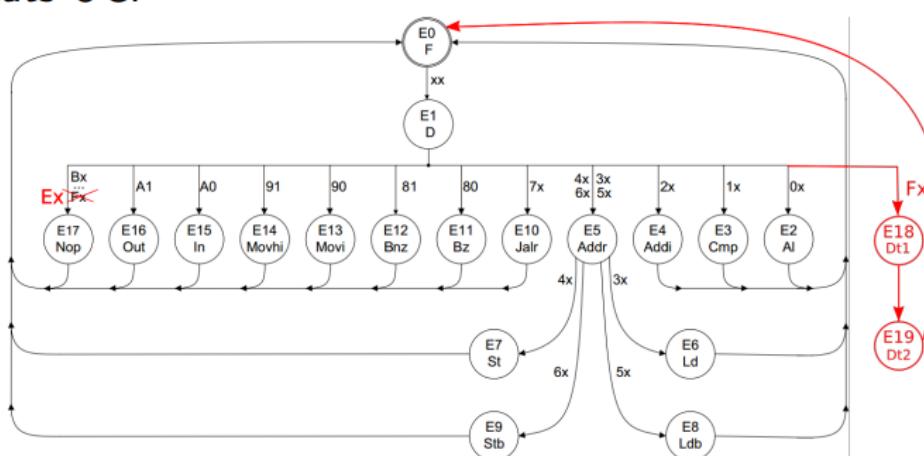
## DECTEST1: especificació

- 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

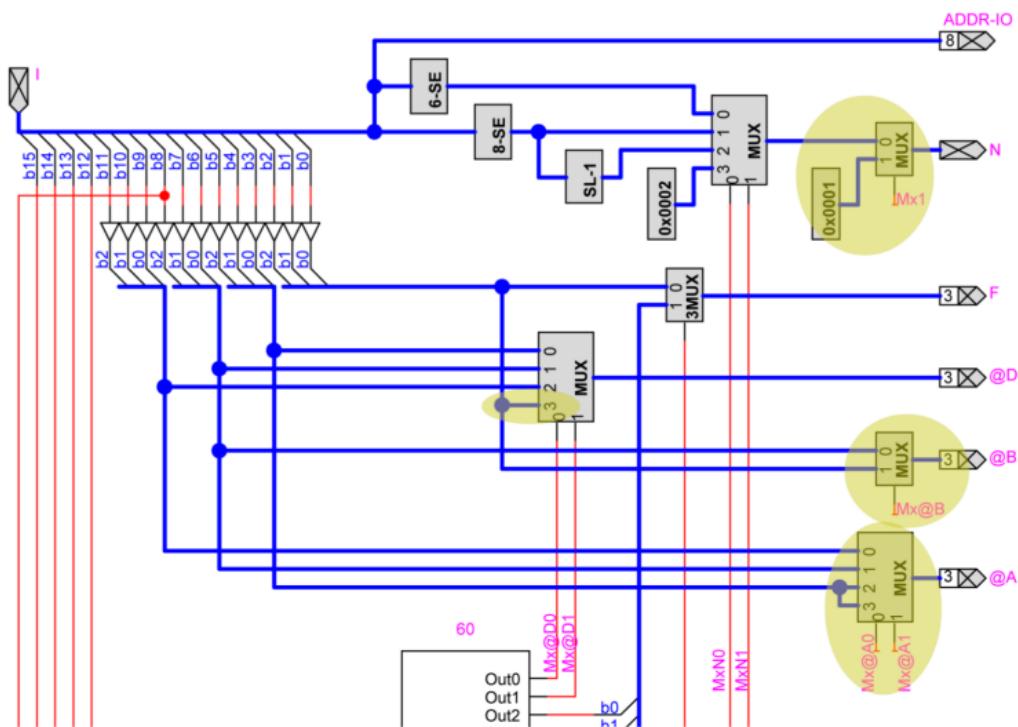
@ROM	Bnz	Bz	WnMem	WrIn	WrOut	WrD	Ldir	Byte	R@/Pc	Alu/R@	Pc/Rx	Ry/N	P1/LA1	P1/LA0	OP1	OP0	MxN1	MxN0	MxF	F2	F1	F0	Mx@D1	Mx@D0
18	0	0	0	0	0	0	0	x	x	x	1	0	x	x	0	0	1	1	1	1	0	1	x	x
19	1	0	0	0	0	1	x	x	x	0	0	0	0	0	0	0	0	1	1	1	0	0	1	0

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( $RX-1 \neq 0x0000$ ) $PC \leftarrow R@$

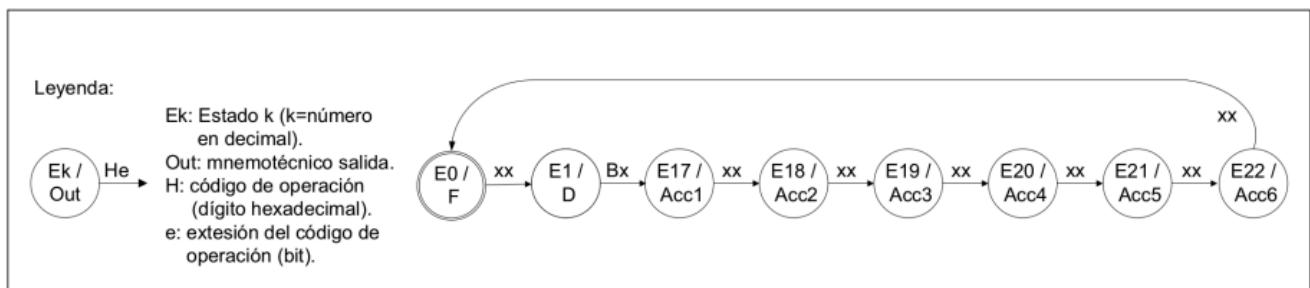
## ACCUMV: ROM OUT



- Cal omplir les files i columnes indicades

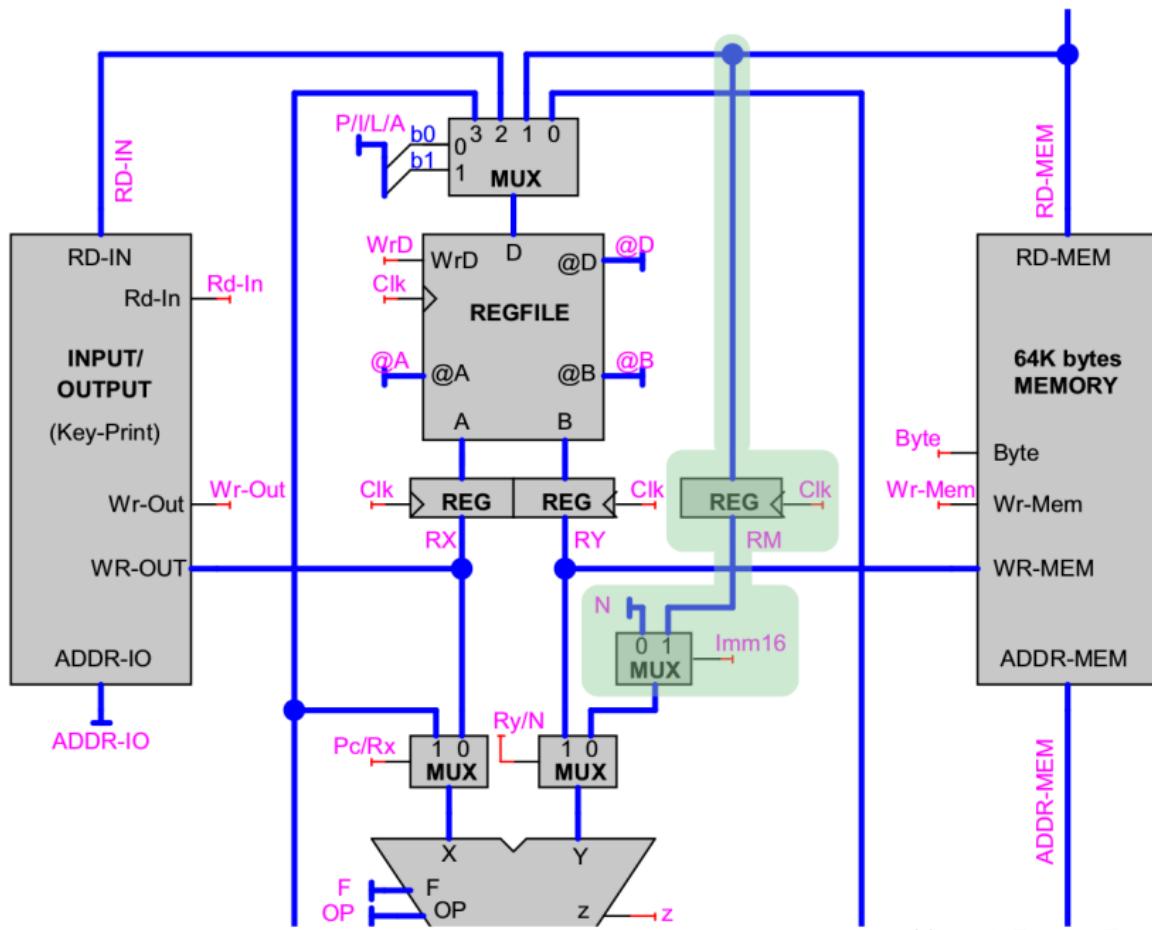
@ROM	Mx@A1	Mx@A0	Mx@B	Mx1	Bnz	Bz	WnMem	Rdn	WrOut	WrD	Ldir	Byte	R@/Pc	Alu/R@	Pc/Rx	Ry/N	P/I/L/A1	P/I/L/A0	OP1	OP0	MxN1	MxN0	MxF	F2	F1	F0	Mx@D1	Mx@D0
0		x	0																									
1		0	0																									
2		x	x																									
3		x	x																									
4		x	0																									
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	Acc6
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 0<sub>2</sub>, 54<sub>10</sub>) contindrà 0x11 (10001<sub>2</sub>, 17<sub>10</sub>)
  - L'adreça 0x037 (00001 1011 1<sub>2</sub>, 55<sub>10</sub>) contindrà 0x11 (10001<sub>2</sub>, 17<sub>10</sub>)

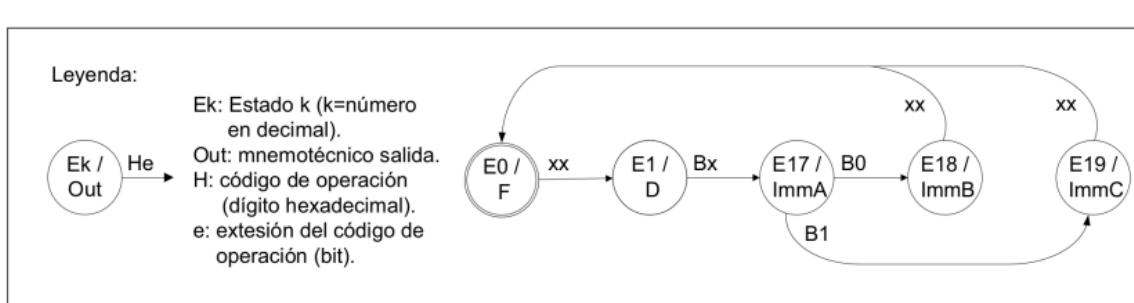
- Codificación: 1011 aaa e xx ddd fff nnnnnnnnnnnnnnnn con e = 0 para las operaciones AL y con e = 1 para las CMP. El campo fff codifica la operación a realizar de la misma manera que se codifica para las familias de instrucciones AL y CMP originales
- Sintaxis: mnemoI16 Rd, Ra, N16 siendo mnemo cualquiera de los mnemotécnicos de las operaciones AL o CMP originales.
  - Ejemplo de instrucción AL: SUBI16 Rd, Ra, N16.
  - Ejemplo de instrucción CMP: CMPEUI16 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 <sub>w</sub> [0xA010] = 0x0607
Mem <sub>w</sub> [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 <sub>w</sub> [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

```