Introducció als Computadors

Tema 8: Unitat de Procés General (UPG)

http://personals.ac.upc.edu/enricm/Docencia/IC/IC8a.pdf

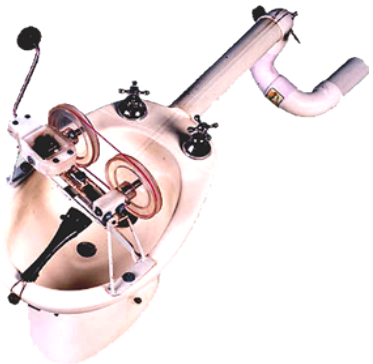
Enric Morancho
(enricm@ac.upc.edu)

Departament d'Arquitectura de Computadors
Facultat d'Informàtica de Barcelona
Universitat Politècnica de Catalunya

UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Facultat d'Informàtica de Barcelona

2020-21, 1^{er} quad.

PPE's vs. UPG, específic vs. genèric

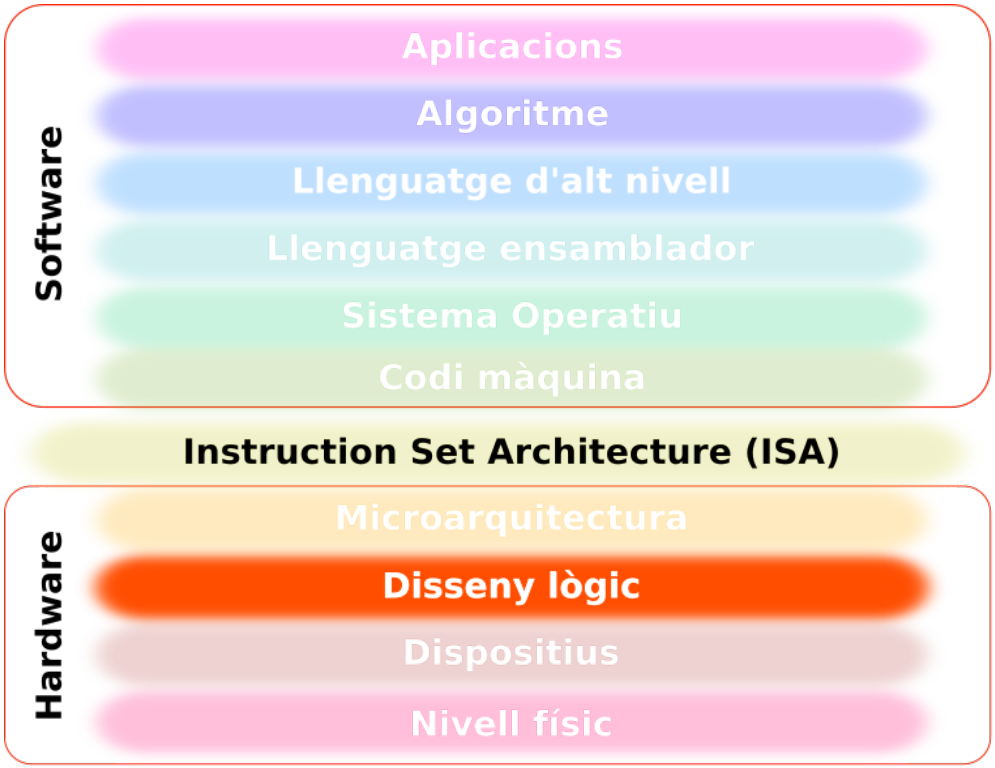


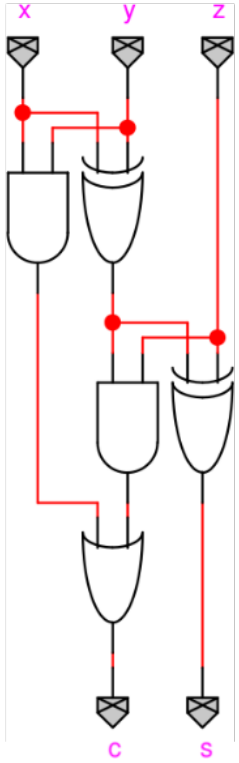
[1]

[2]

Índex

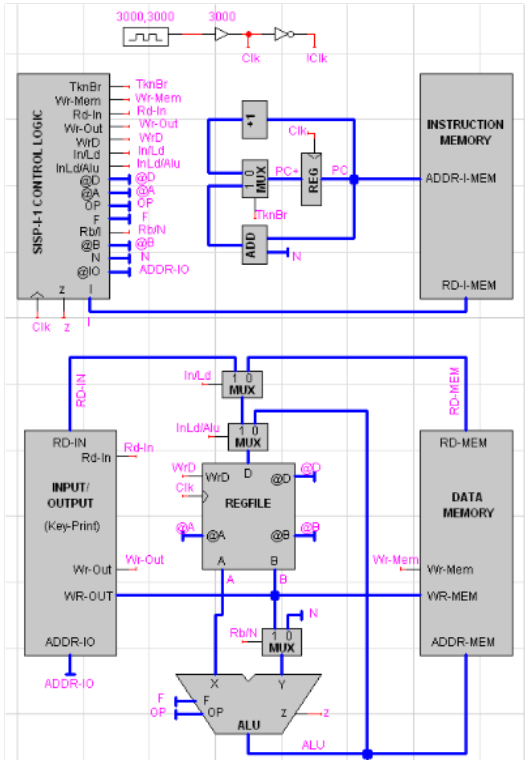
De disseny lògic a microarquitectura





Full-Adder implementat amb 5 portes lògiques

De disseny lògic a microarquitectura



Microarquitectura del processador que dissenyarem a IC

Full de ruta

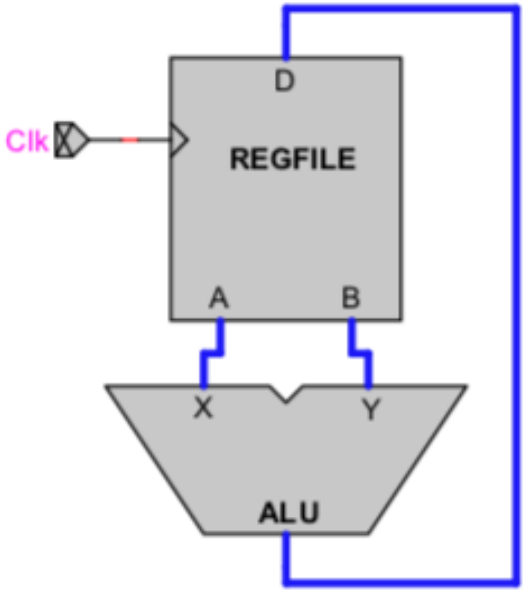
	Tema							
	7	8	9	10	11	12	13	14
Unitat de Control	UCE	UCE	UCE	UCG	UCG	UCG	UCG	UCG
Unitat de Procés	UPE	UPG	UPG	UPG	UPG	UPG	UPG	UPG
Entrada/Sortida	-	-	IO	IO	IO	IO	IO	IO
Memòria RAM	-	-	-	-	MEM	MEM	MEM	MEM
Harvard unicicle	-	-	-	-	-	✓	-	-
Harvard multicicle	-	-	-	-	-	✓	-	-
Von Neumann	-	-	-	-	-	-	✓	✓
Lleng. *assembler*	-	-	-	✓	✓	✓	✓	✓

Unitat de Procés General (UPG)

- Al tema 7 dissenyàrem UPs i UCs específiques per a cada problema
- En aquest tema dissenyarem una UP general
  - La utilitzarem a tots els problemes
    - UPG: Unitat de Procés General
    - Serà el component central del computador que farem a IC
  - Continuarem necessitant una UC específica per a cada problema
- Amb la nostra experiència, la UPG necessitarà:
  - Registres
  - CLC's per a realitzar càlculs aritmètico-lògics
  - La capacitat de modificar dinàmicament les interconnexions entre els registres i els CLC's
  - Comunicar-se amb l'exterior

Esquema bàsic UPG

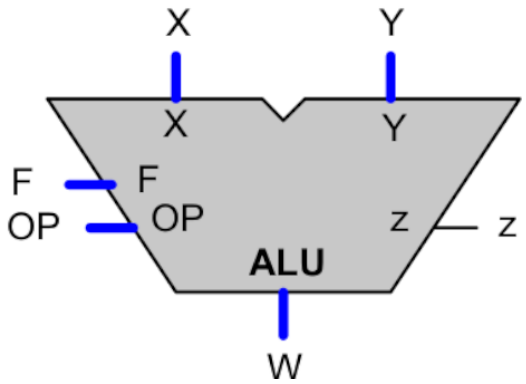
- A la UPG diferenciarem dos grans blocs
  - Banc de registres (REGFILE)
    - CLS que conté els registres
  - Unitat aritmètico-logica (ALU)
    - CLC que realitza els càlculs
- La UPG operarà dos registres i guardarà el resultat a un registre
  - Per decisió de disssey, cada cicle la UPG només podrà fer un càlcul



Índex

Unitat aritmètico-lògica (ALU)

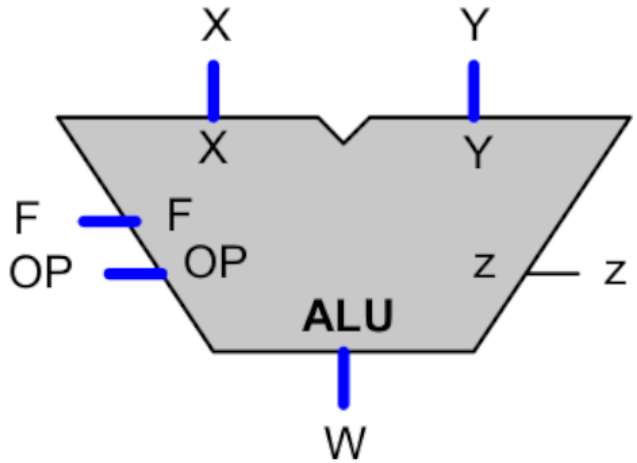
- CLC que implementa totes les operacions aritmètico-lògiques
- Entrades:
  - 2 operands de 16 bits: X i Y
  - 2 bussos OP (de 2 bits) i F (de 3 bits)
    - Codifiquen l'operació a realitzar
- Sortides:
  - resultat de 16 bits: W
  - bit z (zero)
    - $z = 1 \iff W = 0x0000$
- Encapsulat:



ALU: especificació

Description:
The 16-bit output W is the result of an arithmetic/logic operation (AL), a comparison (CMP) or an other action (MISC), applied to the 16-bit inputs X and Y. The type of operation is selected by the 2-bit input OP and by the 3-bit input F.

F	OP = 11	OP = 10	OP = 01	OP = 00
000	---	X	CMPLT(X, Y)	AND(X, Y)
001	---	Y	CMPLE(X, Y)	OR(X, Y)
010	---	---	---	XOR(X,Y)
011	---	---	CMPEQ(X, Y)	NOT(X)
100	---	---	CMPLTU(X, Y)	ADD(X, Y)
101	---	---	CMPLEU(X, Y)	SUB(X, Y)
110	---	---	---	SHA(X ,Y)
111	---	---	---	SHL(X, Y)

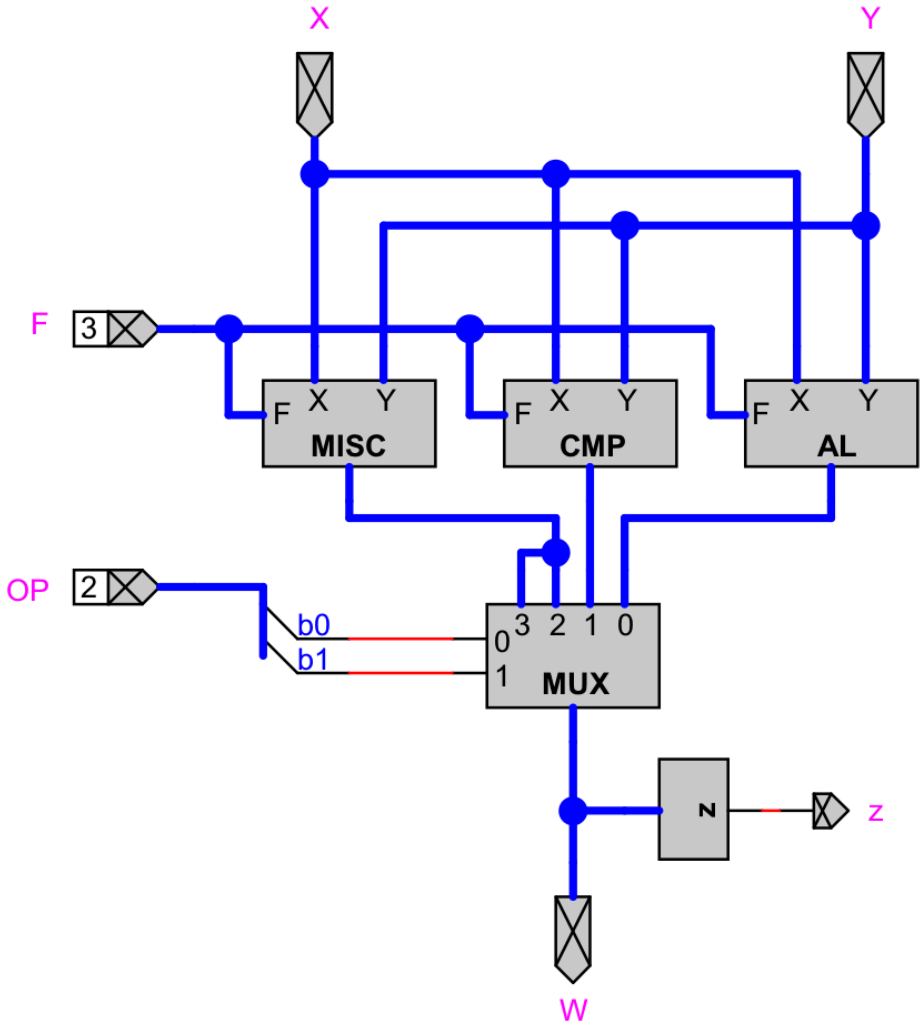


If the ALU performs a comparison the result can be true or false. If the result is true then W(b0) = 1 else W(b0) = 0. Moreover, W(bi) = 0 for i = 1 to 15.
The 1-bit output z indicates when the output W is a vector of 16 zeroes.

- No heu de memoritzar els codis *F* i *OP*, als parcials us subministrarem un xuletari amb aquesta informació així com alguns esquemes lògics
  - `https://personals.ac.upc.edu/enricm/Docencia/IC/chuletario8.pdf`
  - `https://atenea.upc.edu/mod/resource/view.php?id=1673034`

ALU: implementació

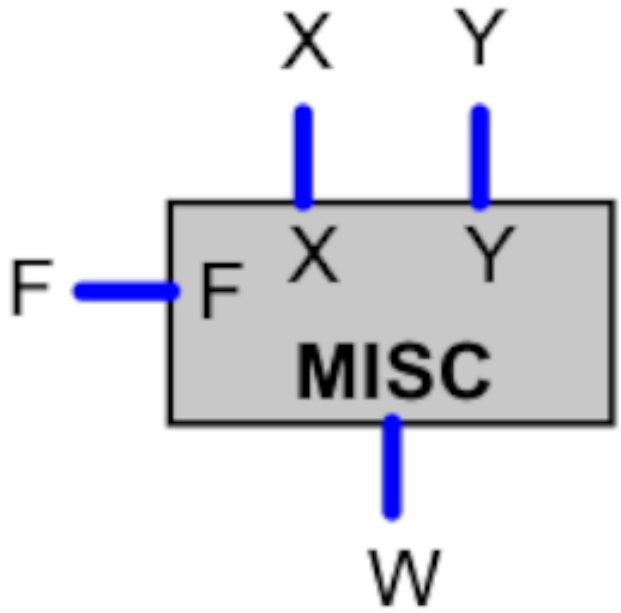
- ALU implementada mitjançant disseny modular multinivell



ALU: especificació bloc MISC

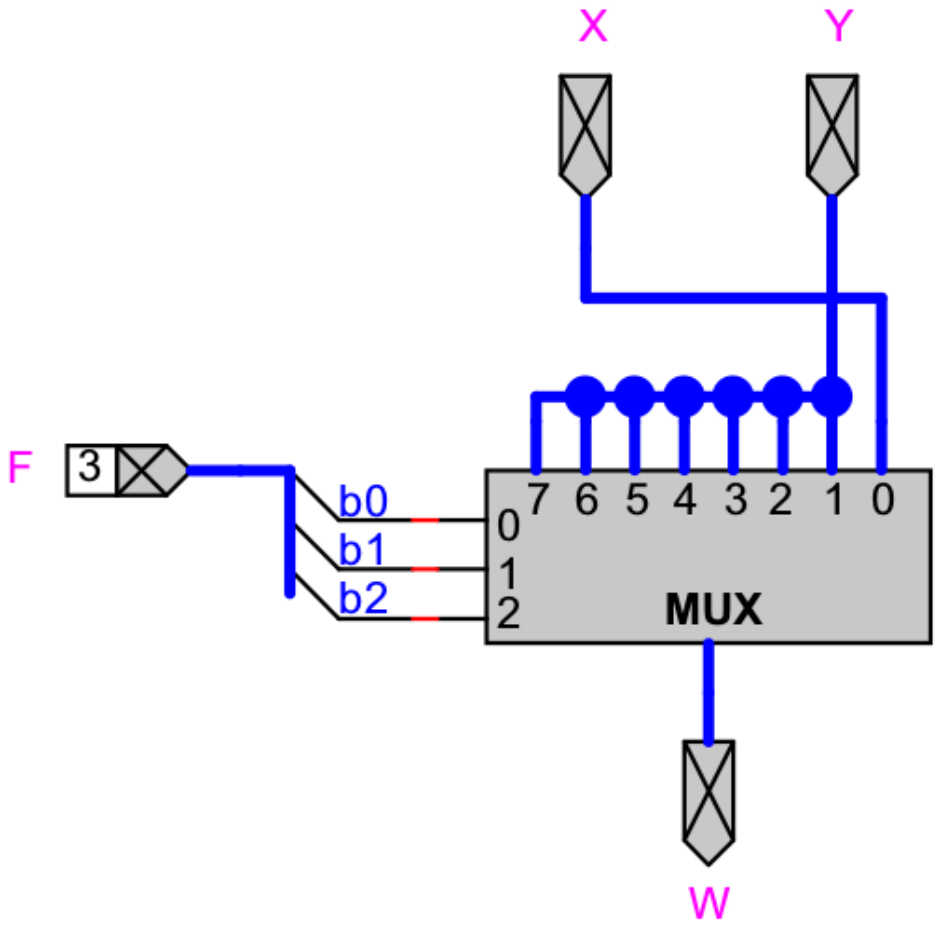
Description:
The 16-bit output W is the function chosen by the 3-bit selection input F, applied to the 16-bit inputs X and Y. The functions are:

F	W
000	X
001	Y
010	---
011	---
100	---
101	---
110	---
111	---



- Posteriorment hi afegirem més funcionalitats

ALU: implementació bloc MISC

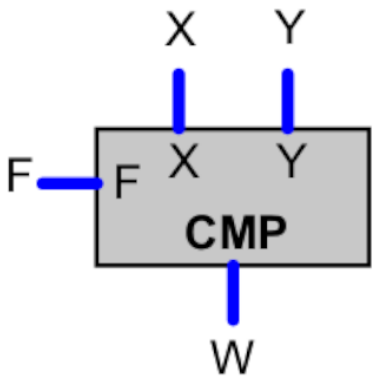


ALU: especificació bloc CMP

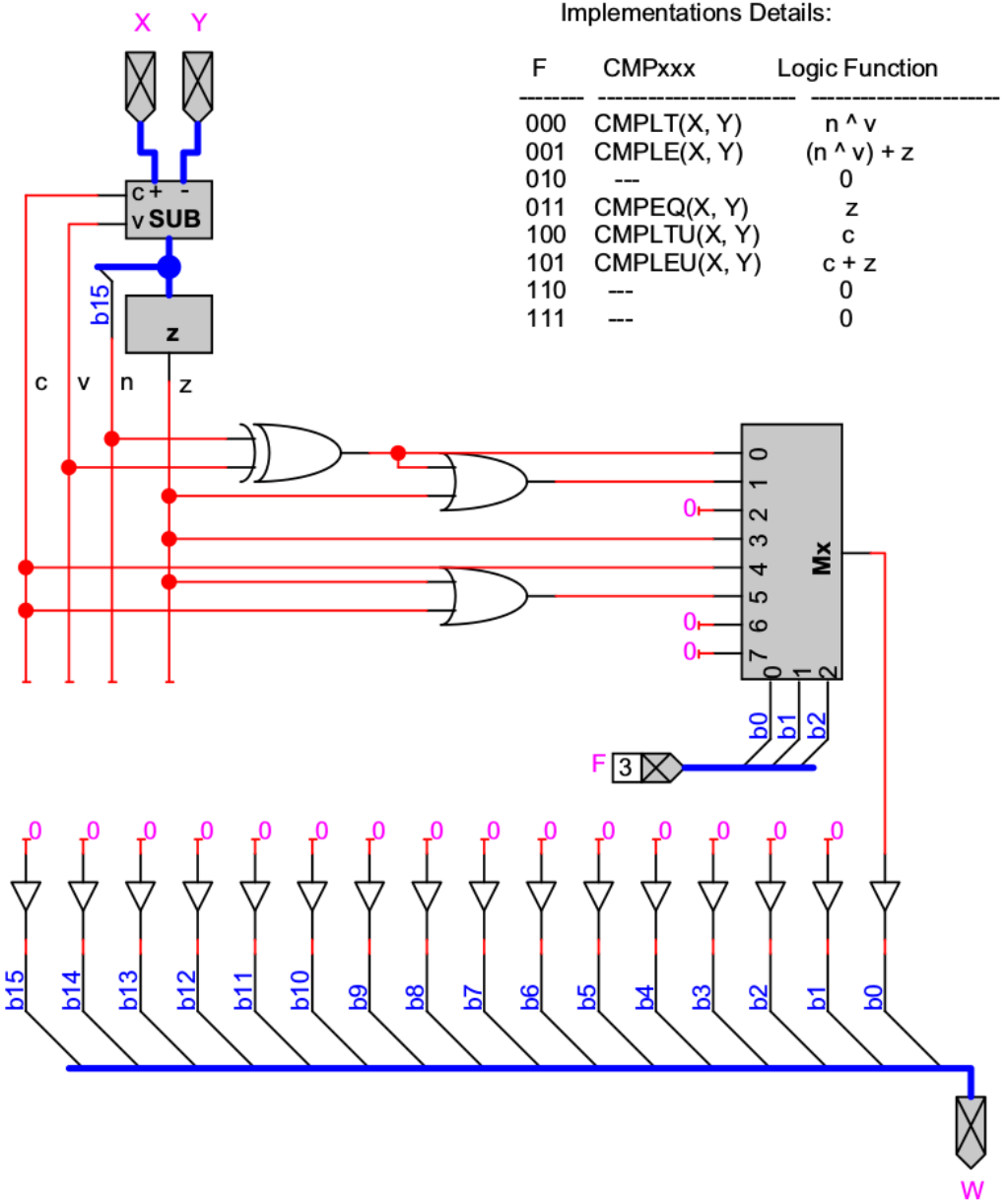
- El resultat TRUE es codifica amb 0x0001
  - $z = 0$
  - **El bit z no segueix el conveni típic per representar booleans**
- El resultat FALSE es codifica amb 0x0000
  - $z = 1$

Description:
The 16-bit output W has only two posible values as the result of a comparison: true or false. True is coded as W(b0) = 1 and false as W(b0) = 0.
For all the cases W(bi) = 0 for i = 1 to 15. The type of comparison and the consideration of the 16-bit inputs X and Y as signed or as unsigned integers is chosen by the 3-bit selection input F according to the next table:

F	W	CMPxx(X, Y)	Name
000	Xs < Ys	CMPLT(X, Y)	Less Than (Signed)
001	Xs <= Ys	CMPLE(X, Y)	Less than or Equal (Signed)
010	---	---	
011	X ==Y	CMPEQ(X, Y)	Equal
100	Xu < Yu	CMPLTU(X, Y)	Less Than Unsigned
101	Xu <= Yu	CMPLEU(X, Y)	Less than or Equal Unsigned
110	---	---	
111	---	---	



ALU: implementació bloc CMP



Implementations Details:

F	CMPxxx	Logic Function
000	CMPLT(X, Y)	n ^ v
001	CMPLE(X, Y)	(n ^ v) + z
010	---	0
011	CMPEQ(X, Y)	z
100	CMPLTU(X, Y)	c
101	CMPLEU(X, Y)	c + z
110	---	0
111	---	0

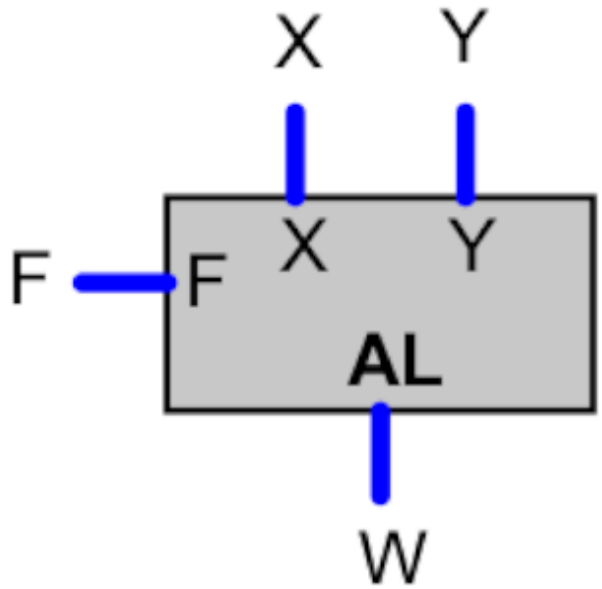
ALU: especificació bloc AL

Description:

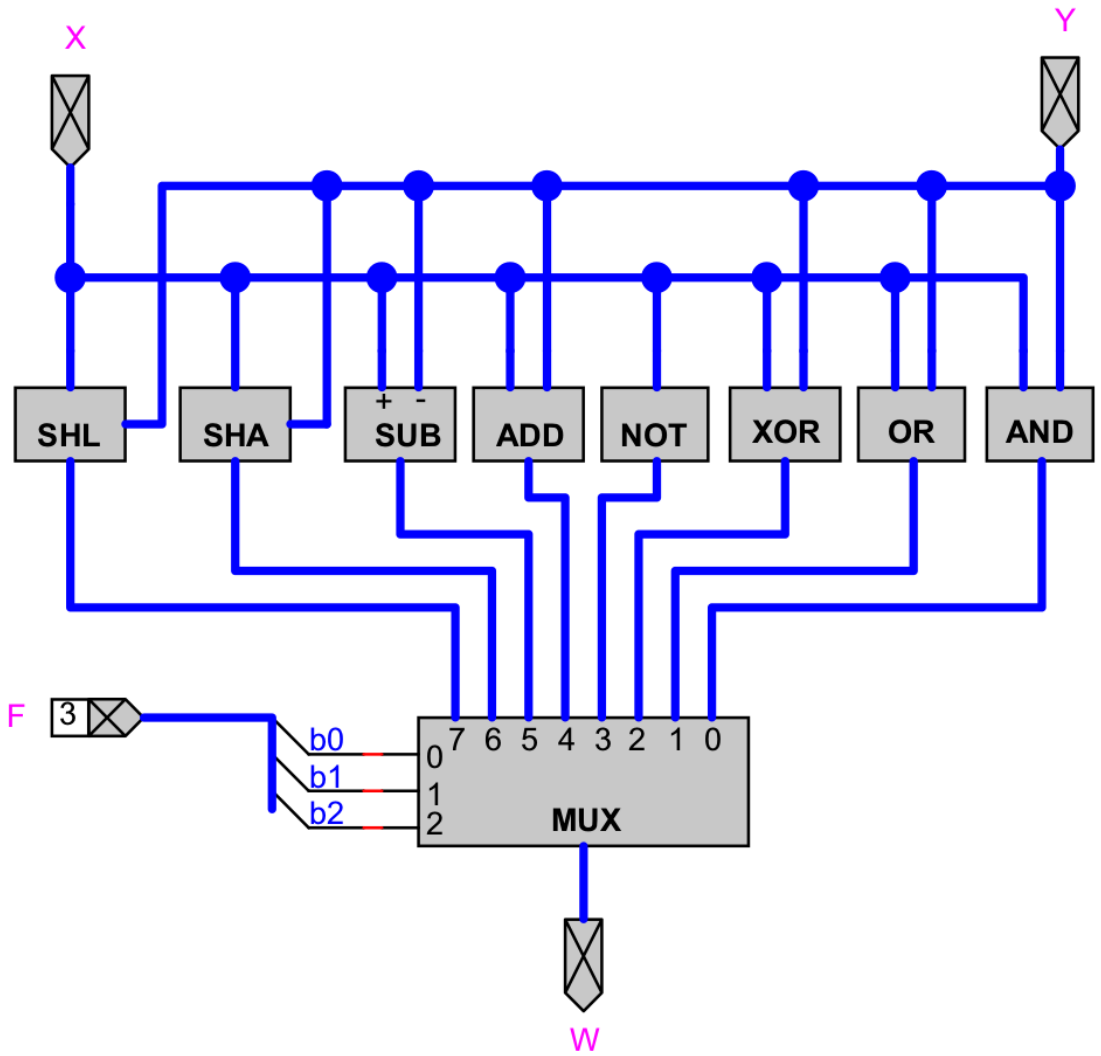
The 16-bit output W is the arithmetic or logic function chosen by the 3-bit selection input F, applied to the 16-bit inputs X and Y.

The functions are:

F	W
000	AND (X, Y)
001	OR (X, Y)
010	XOR(X, Y)
011	NOT (X)
100	ADD (X, Y)
101	SUB (X, Y)
110	SHA (X, Y)
111	SHL (X, Y)



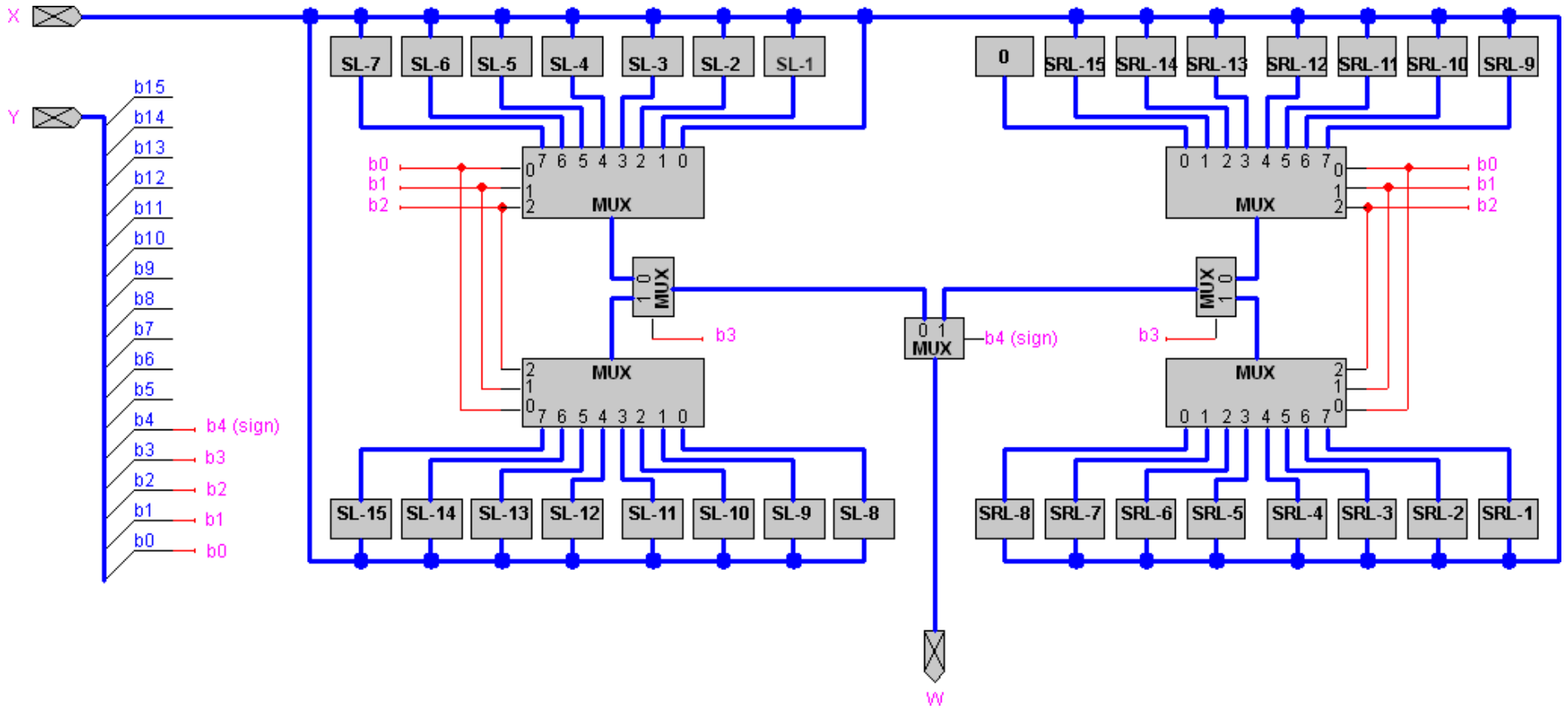
ALU: implementació bloc AL



ALU: blocs SHA i SHL

- Desplaçament (*shift*) aritmètic (SHA) i lògic (SHL) a dreta o esquerra
  - Es diferencien als desplaçaments a la dreta
    - SHL posa "0" als bits de més pes
    - SHA replica bit de més pes
- Els 5 bits baixos de Y codifiquen, en Ca2, quants bits cal desplaçar
  - $-16 \leq shift \leq 15$
  - Si $shift > 0 \implies$ desplaçar X a l'esquerra $shift$ posicions
  - Si $shift < 0 \implies$ desplaçar X a la dreta $\mid shift \mid$ posicions
    - Anàleg a l'aritmètica on $n \cdot 2^{-5} = n/2^5$
- Exemples SHA(X,Y) / SHL(X,Y):
  - SHL(0x8CEF, 0xAAFE) = 0x233B
    - $0xAAFE = (1010\ 1010\ 1111\ 1110)_2 \implies shift = (11110_2)_s = -2$
    - $0x8CEF = (1000\ 1100\ 1110\ 1111)_2 \rightsquigarrow (0010\ 0011\ 0011\ 1011)_2 = 0x233B$
  - SHA(0x840C, 0xA631) = 0xFFFF ($shift = (10001_2)_s = -15$)
  - SHA(0x0F03, 0x5AEE) = 0xC000 ($shift = (01110_2)_s = +14$)

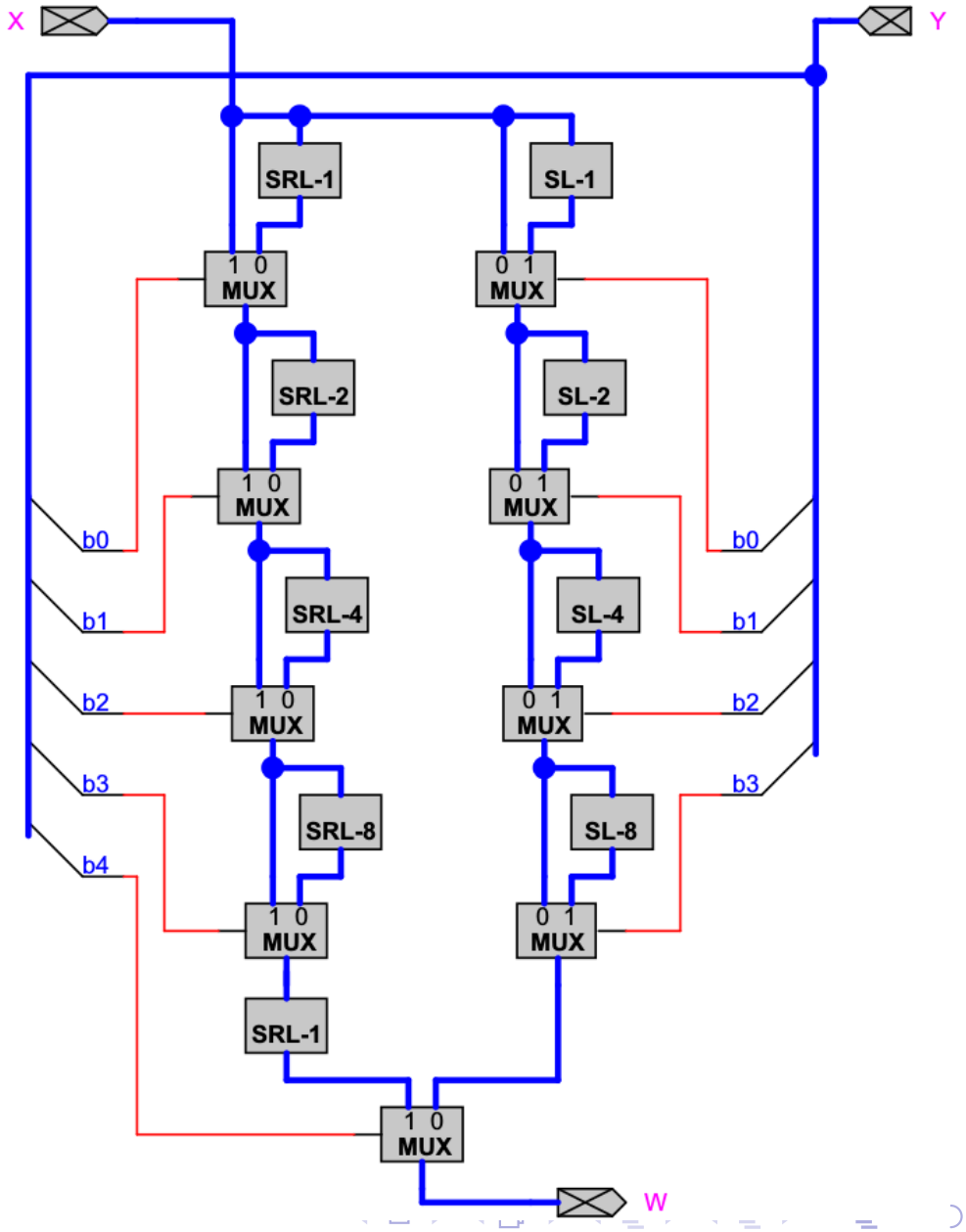
ALU: primera aproximació al bloc SHL



- Calcula els 32 possibles resultats ($-16 \leq shift \leq 15$)
- Un arbre de multiplexors amb 3 nivells selecciona el resultat sol·licitat
- Anàleg per SHA
  - Canviant SRL per SRA i tractament del cas *shift*=-16
- Intentarem una altra implementació amb menys recursos

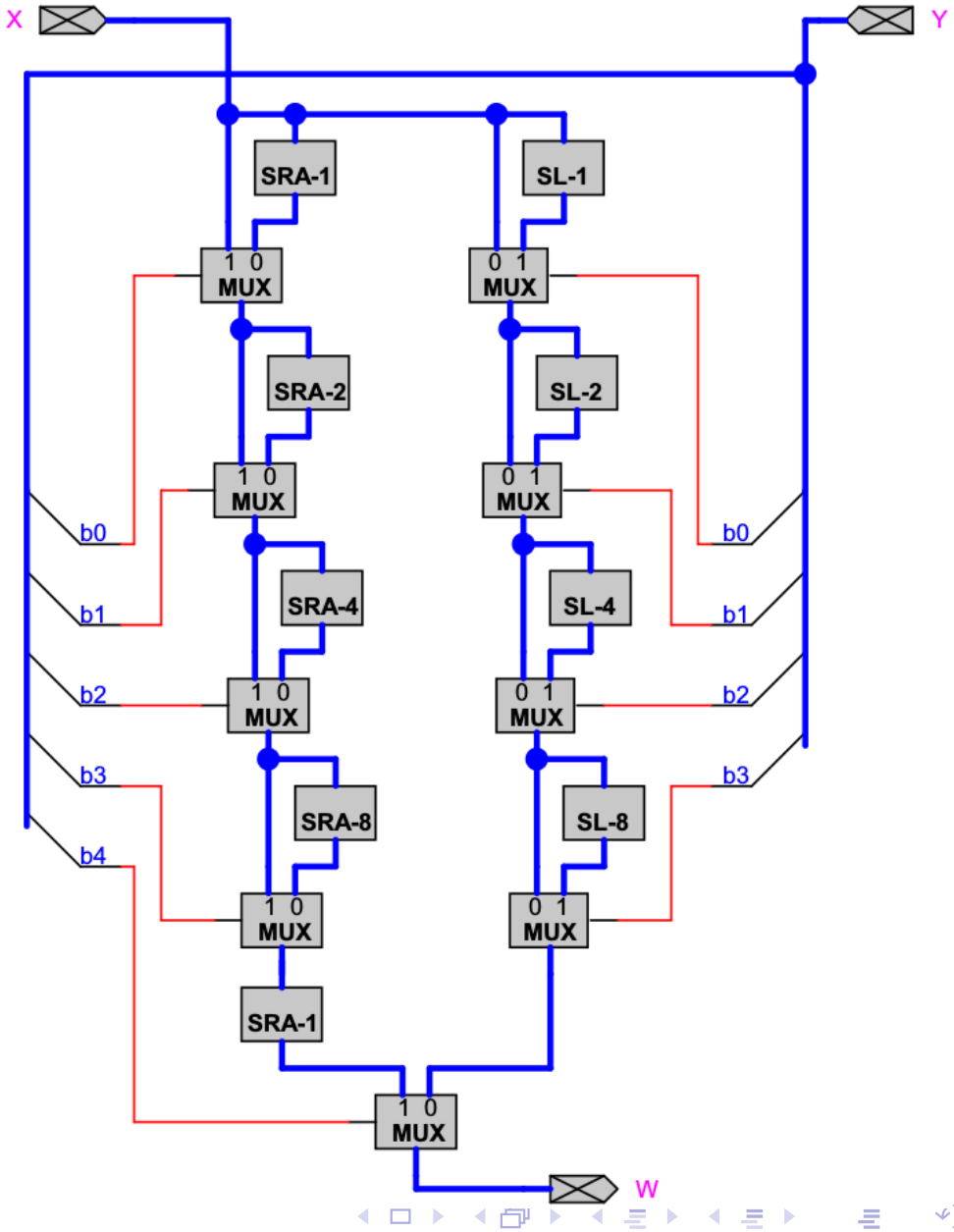
ALU: implementació bloc SHL

- Idea: Descomponem el desplaçament *shift* en una sèrie de desplaçaments que siguin potències de 2
- Exemple: desplaçar +11 és equivalent a desplaçar +1, +2 i +8
- La codificació en Ca2 de *shift* ens dona aquesta descomposició



ALU: implementació bloc SHA

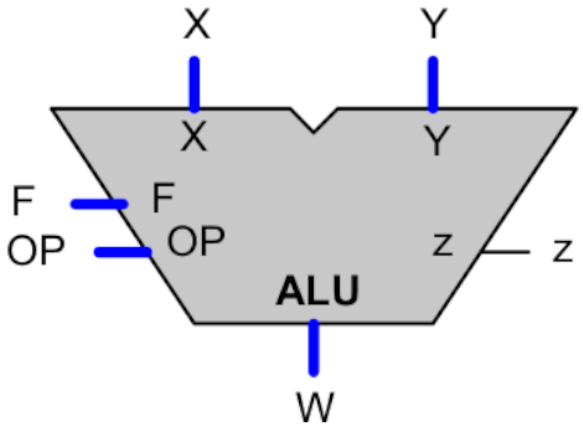
- Anàleg al bloc SHL
- Canvia blocs SRL per SRA



Exercici: quins resultats calcula l'ALU?

Description:
The 16-bit output W is the result of an arithmetic/logic operation (AL), a comparison (CMP) or an other action (MISC), applied to the 16-bit inputs X and Y. The type of operation is selected by the 2-bit input OP and by the 3-bit input F.

F	OP = 11	OP = 10	OP = 01	OP = 00
000	---	X	CMPLT(X, Y)	AND(X, Y)
001	---	Y	CMPLE(X, Y)	OR(X, Y)
010	---	---	---	XOR(X,Y)
011	---	---	CMPEQ(X, Y)	NOT(X)
100	---	---	CMPLTU(X, Y)	ADD(X, Y)
101	---	---	CMPLEU(X, Y)	SUB(X, Y)
110	---	---	---	SHA(X ,Y)
111	---	---	---	SHL(X, Y)



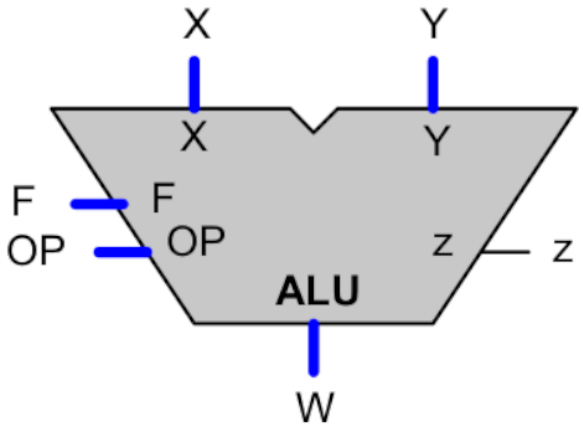
If the ALU performs a comparison the result can be true or false. If the result is true then W(b0) = 1 else W(b0) = 0. Moreover, W(bi) = 0 for i = 1 to 15.
The 1-bit output z indicates when the output W is a vector of 16 zeroes.

F	OP	X	Y	W	z
010	00	0xFA12	0xEDAC		
110	00	0x97AB	0xEEF2		
111	00	0x97AB	0xEEF2		
001	10	0x812D	0x27BF		
000	01	0xFBE6	0xBD56		

Exercici: quins resultats calcula l'ALU?

Description:
The 16-bit output W is the result of an arithmetic/logic operation (AL), a comparison (CMP) or an other action (MISC), applied to the 16-bit inputs X and Y. The type of operation is selected by the 2-bit input OP and by the 3-bit input F.

F	OP = 11	OP = 10	OP = 01	OP = 00
000	---	X	CMPLT(X, Y)	AND(X, Y)
001	---	Y	CMPLE(X, Y)	OR(X, Y)
010	---	---	---	XOR(X,Y)
011	---	---	CMPEQ(X, Y)	NOT(X)
100	---	---	CMPLTU(X, Y)	ADD(X, Y)
101	---	---	CMPLEU(X, Y)	SUB(X, Y)
110	---	---	---	SHA(X ,Y)
111	---	---	---	SHL(X, Y)



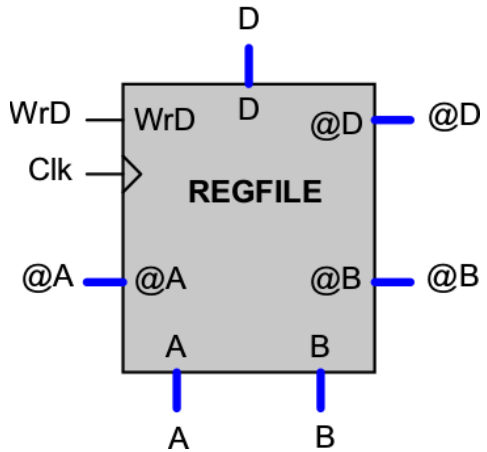
If the ALU performs a comparison the result can be true or false. If the result is true then W(b0) = 1 else W(b0) = 0. Moreover, W(bi) = 0 for i = 1 to 15.
The 1-bit output z indicates when the output W is a vector of 16 zeroes.

F	OP	X	Y	W	z
010	00	0xFA12	0xEDAC	0x17BE	0
110	00	0x97AB	0xEEF2	0xFFFE	0
111	00	0x97AB	0xEEF2	0x0002	0
001	10	0x812D	0x27BF	0x27BF	0
000	01	0xFBE6	0xBD56	0x0000	1

Índex

Banc de registres (REGFILE)

- CLS que gestiona els registres de la UPG
  - Per decisió de disseny, tindrà 8 registres
    - Els anomenarem R0, R1,... R7
    - Necessitarem 3 bits per a identificar els registres
- Entrades:
  - 2 identificadors dels registres a llegir (de 3 bits cadascun): @A, @B
  - 1 bit de escriptura: WrD
  - 1 identificador de registre a modificar (de 3 bits): @D
  - 1 bus D (de 16 bits) amb la dada a escriure (si WrD=1) quan es produeixi el pols ascendent de rellotge
- Sortides:
  - 2 busos de 16 bits, A i B, amb el contingut dels registres @A i @B
- Encapsulat:

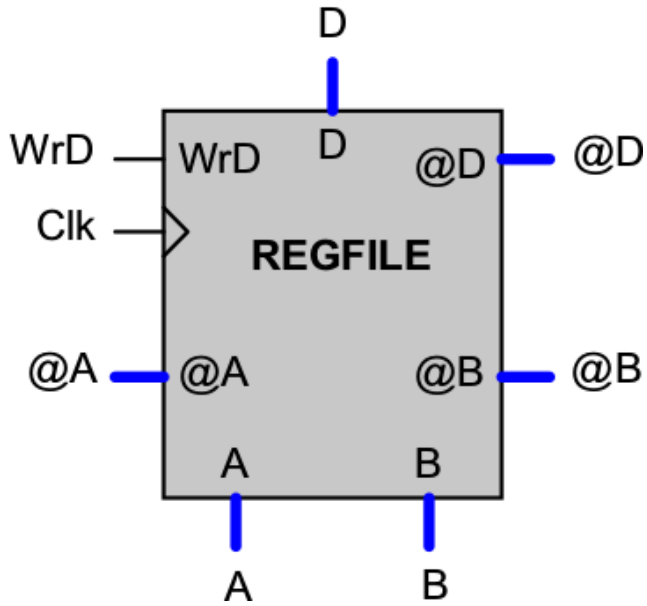


Banc de registres: especificació

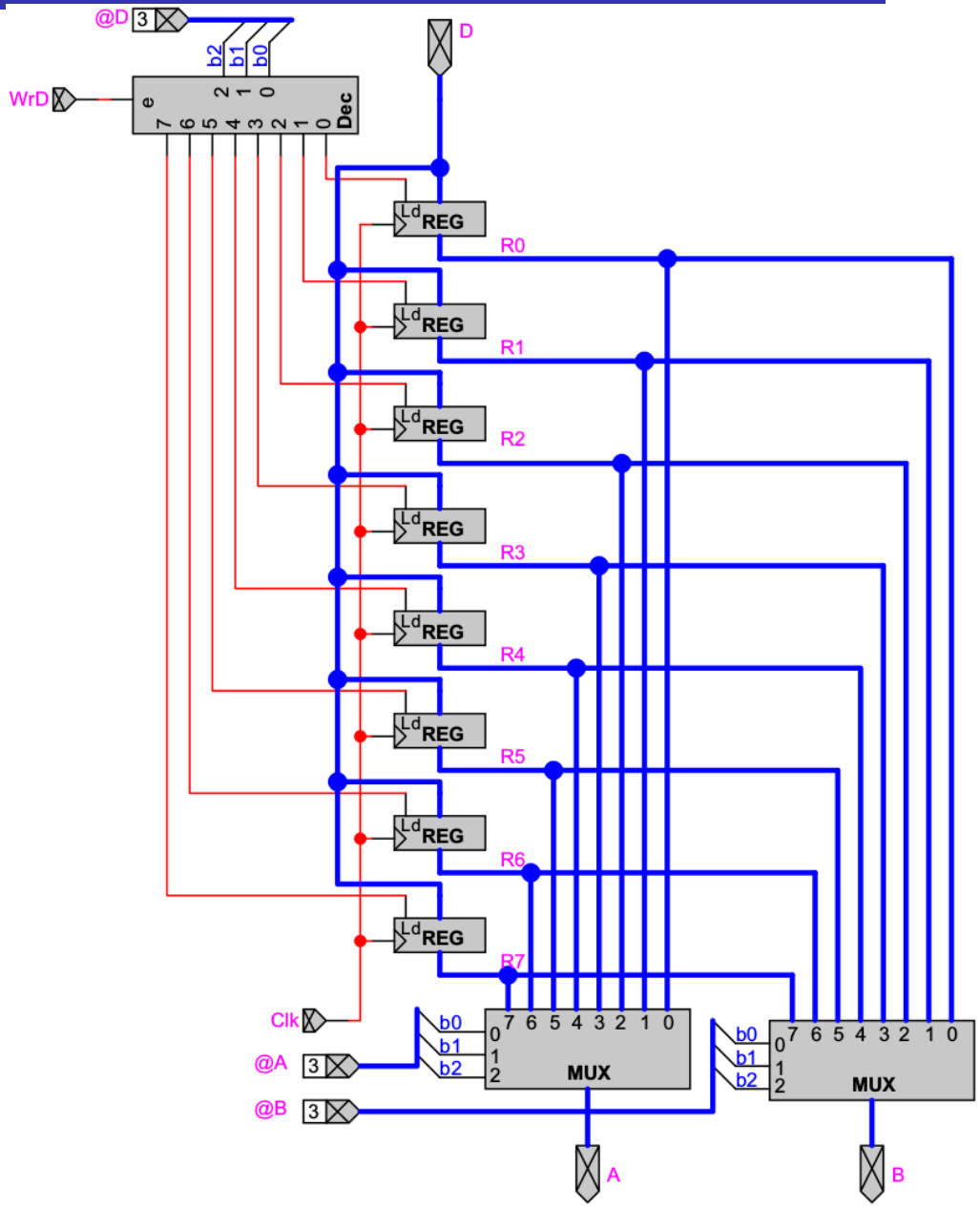
Description:

Register file with 8 16 Bit-wide registers with 2 read ports, A and B, and 1 write port D. @A, @B and @D are the 3-bit addresses of ports A, B and D respectively. If WrD = 1 the port D is written into Register @D when the Clk binary input changes from 0 to 1.
A and B are the contents of Registers @A and @B.

For simulation purposes, a display of the contents of each register is available.



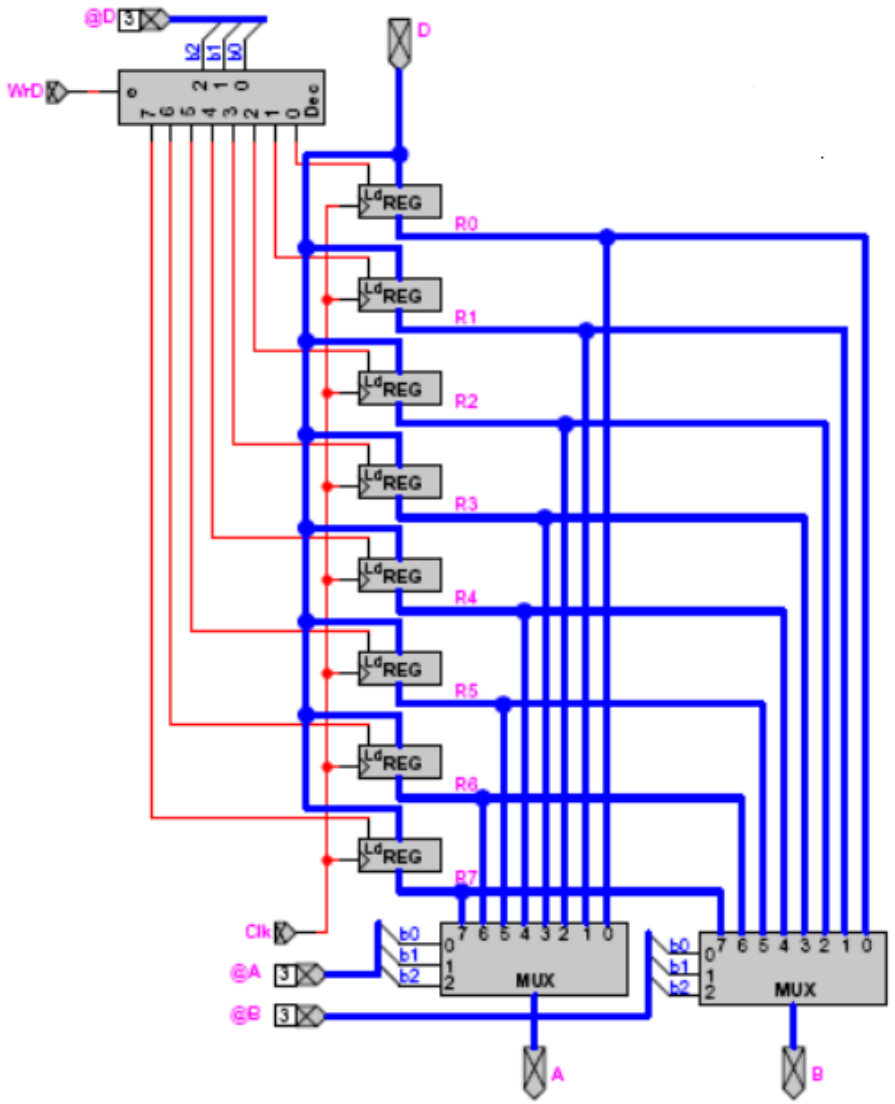
Banc de registres: implementació



Banc de registres: implementació

- *WrD* està connectada a l'entrada *enable* d'un descodificador
  - Si *WrD* val "0", totes les sortides del descodificador valdran 0
    - Com aquestes sortides són els senyals de càrrega dels registres, al final d'aquest cicle cap registre canviarà de valor
  - Si *WrD* val "1", únicament una sortida del descodificador valdrà 1
    - La corresponent al registre @*D*
    - Aquest registre tindrà el senyal de càrrega a "1"
    - Al final d'aquest cicle, s'emmagatzemarà al registre @*D* el valor que hi hagi al bus *D*

Exercici: omplir cronograma



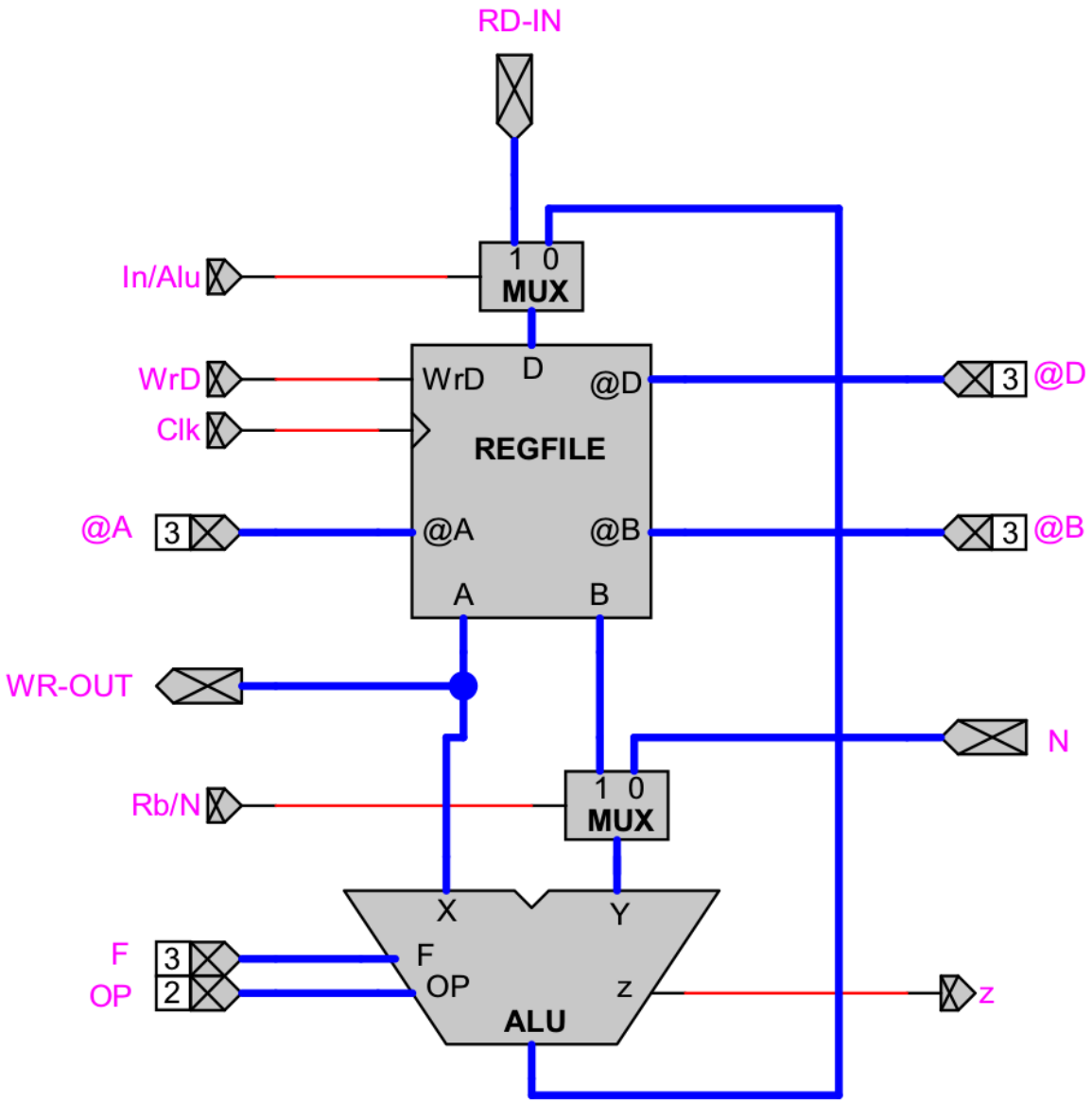
ciclo	c	c+1	c+2	c+3
clk				
WrD				
@D	011	001	100	110
D	0x0020	0x0020	0x0021	0x00105
@A	001	011	011	100
@B	011	111	001	111
R0	0x0031			
R1	0x0000			
R2	0x0001			
R3	0x001B			
R4	0XFF01			
R7	0x0000			
A	0x0000			
B	0x001B			

Índex

Estructura de la UPG

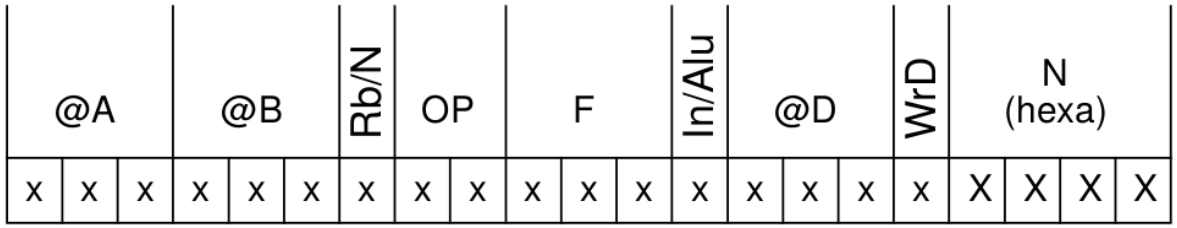
- A més de la ALU i del RF, la UPG es podrà comunicar amb l'exterior
  - Tindrà bus d'entrada RD-IN de 16 bits
    - El seu valor es podrà emmagatzemar a un registre
  - Tindrà un bus de sortida WR-OUT de 16 bits
    - Hi enviarem el contingut del registre @A
- També podrem definir l'operand Y de la ALU sense passar pel RF
  - "Valor immediat"
  - Útil per introduir valors constants a la UPG
    - No malbaratarem registres per emmagatzemar valors constants
  - La UPG tindrà un bus d'entrada N de 16 bits
- Caldran multiplexors abans de les entrades D del RF i Y de la ALU
  - Amb els seus senyals de control
    - El nom del senyal de control d'un multiplexor seguirà un conveni

Estructura de la UPG



Entrades/sortides de la UPG

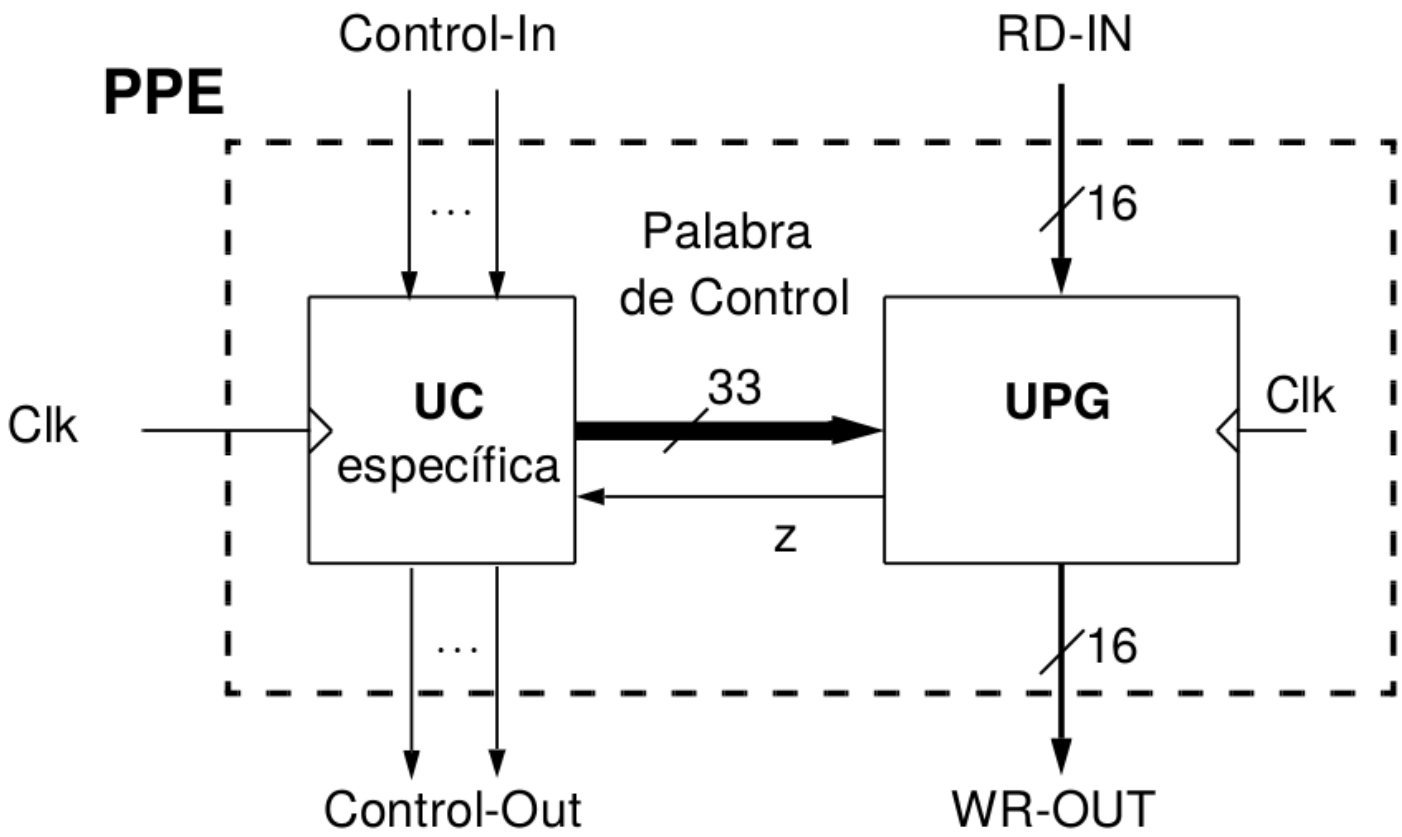
- Entrades:
  - De control (paraula de control):
    - 33 bits (3+3+1+2+3+1+3+1+16)



@A			@B			Rb/N	OP		F			In/Alu	@D			WrD	N (hexa)			
x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	X	X	X	X

  - De dades:
    - Bus RD-IN de 16 bits
- Sortides:
  - De control (paraula de condició):
    - 1 bit (z)
  - De dades:
    - Bus WR-OUT de 16 bits

Comunicació UC - UPG



Índex

Accions de la UPG

- Classificarem les accions que pot fer la UPG en un cicle:
  1. Accions aritmètico-lògiques i comparacions de dos operands
  2. Accions aritmètico-lògiques d'un operand
  3. Accions de moviment
  4. Accions d'entrada de dades
  5. Accions de sortida de dades
- Cada acció tindrà associada una paraula de control
  - 33 bits
  - Per comoditat, assignarem un mnemotècnic a cada acció
    - "011 101 1 00 100 0 110 1 XXXX" vs. "ADD R6, R3, R5"
    - Està composat pel nom de l'acció i els registres implicats
    - És la llavor del llenguatge màquina amb el que programarem la UPG

Accions de la UPG (1/3)

1.- Accions aritmètico-lògiques i comparacions de dos operands

- És possible realitzar 12 tipus de càlculs
  - Nom del càlcul:
    - Aritmètico-lògics: `AND`, `OR`, `XOR`, `ADD`, `SUB`, `SHA`, `SHL`
    - Comparacions: `CMPLT`, `CMPLE`, `CMPEQ`, `CMPLTU`, `CMPLEU`
  - Operen amb dos registres font i el resultat es guarda a un registre destí
    - Als mnemotècnics, primer indicarem el registre destí
    - El registre destí també pot ser un dels registres font
  - Si el segon operand és un valor immediat (entra pel bus N), afegirem una `I` al mnemotècnic
- Exemples:
  - `ADD R7, R1, R2` $\rightsquigarrow$ `R7 = R1 + R2`
  - `SUBI R3, R6, 0xA314` (constant en hexa) $\rightsquigarrow$ `R3 = R6 - 0xA314`
  - `XORI R3, R3, -1` (constant en decimal) $\rightsquigarrow$ `R3 = R3` $\oplus$ `0xFFFF`

Accions de la UPG (2/3)

2.- Accions aritmètico-lògiques d'un operand

- Operació NOT bit a bit (*bitwise*)
- Exemple:
  - `NOT R3, R5` $\rightsquigarrow$ `R3 = !R5`

3.- Accions de moviment

- Copiar el valor d'un registre sobre un altre o inicialitzar un registre amb un valor
  - `MOVE R3, R2` $\rightsquigarrow$ `R3 = R2`
  - `MOVEI R5, 0x14` (constant en hexa) $\rightsquigarrow$ `R5 = 0x0014`
  - `MOVEI R5, -2` (constant en decimal) $\rightsquigarrow$ `R5 = 0xFFFE`

Accions de la UPG (3/3)

4.- Accions d'entrada de dades

- Assignar a un registre el valor que hi ha al bus RD-IN
- L'assignació es realitza al final de cicle
- Exemple:
  - `IN R3`

5.- Accions de sortida de dades

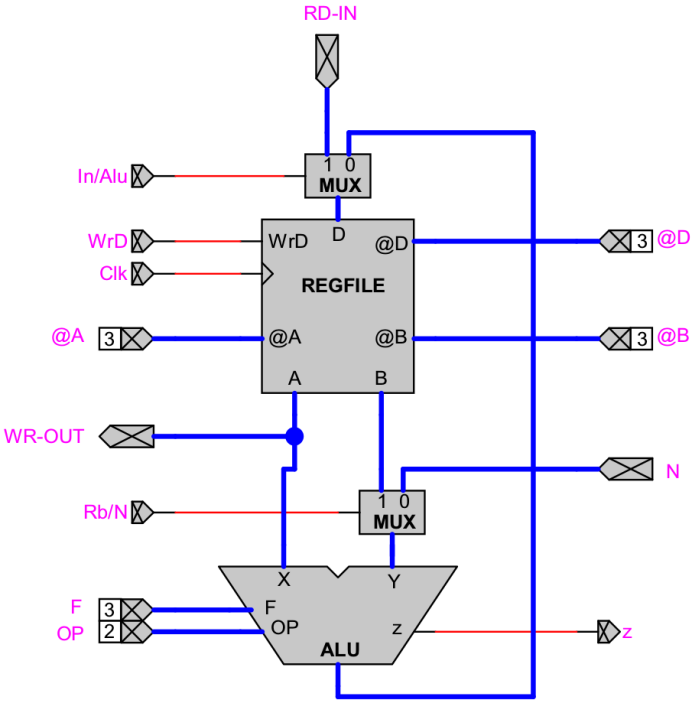
- Mostrar pel bus de sortida WR-OUT el valor d'un registre
- El valor està disponible des de l'inici de cicle (un cop transcorregut el T_p del banc de registres)
- Exemple:
  - `OUT R2`

Casos particulars

- Accions que no modifiquem cap registre destí
  - Exemple: `ADD -, R2, R3`
  - Només actualitzen el bit *z*
  - El bit de control *WrD* tindrà el valor "0"
- NOP - *No operation*
  - Deixa passar un cicle sense modificar l'estat de la UPG
  - Exemple: `NOP`
  - El bit de control *WrD* tindrà el valor "0"
- Accions en paral·lel
  - És possible realitzar algunes parelles d'accions al mateix cicle
    - `IN R3` $\|$ `OUT R4`
    - `ADD R3, R4, R5` $\|$ `OUT R4`
    - `ADD -, R4, R7` $\|$ `IN R5`
  - Altres combinacions **NO** es podran fer al mateix cicle
    - UPG només pot llegir 2 registres, escriure'n un i fer un càlcul per cicle
    - `ADD R3, R4, R5` $\nparallel$ `OUT R1` (caldria llegir 3 regs)
    - `ADD R3, R4, R7` $\nparallel$ `IN R5` (caldria escriure 2 regs)
    - `ADD R3, R4, R7` $\nparallel$ `NOT R2, R5` (caldria fer dos càlculs amb l'ALU)

Exercici: mnemotècnic i paraula de control

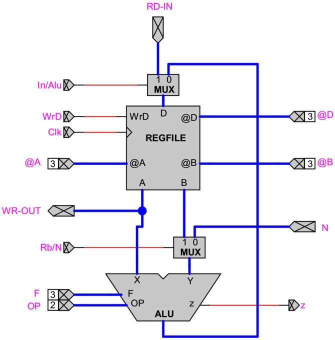
- Cal saber passar del mnemotècnic a la paraula de control i a l'inrevés
- Sempre que sigui possible, poseu *x* a la paraula de control



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	
	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 1 1	x x x	0	0 0	0 0 0	0	0 1 0	1	F F 0 0	
NOT R4, R2										
	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	x	x x	x x x	1	0 1 0	1	X X X X	
OUT R4										
	0 1 1	x x x	0	0 0	0 0 0	x	x x x	0	8 0 0 0	
NOP										
	1 0 0	x x x	x	x x	x x x	1	0 1 0	1	X X X X	

Exercici: mnemotècnic i paraula de control

- Cal saber passar del mnemotècnic a la paraula de control i a l'inrevés
- Sempre que sigui possible, poseu x a la paraula de control



Mnemotècnics	@A	@B	Rb/N	OP	F	In/Alu	@D	WrD	N (hexa)
	Palabra de Control de 33 bits								
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

Índex

Del codi C al graf d'estats de la UC

- Dividirem el codi C en accions que es puguin fer a la UPG en un cicle
  - Cada acció serà un node del graf d'estats de la UC
- Indicarem les transicions entre estat
  - Codi seqüencial
    - Transicions incondicionals, independents del bit z
    - Accions en el mateix ordre que en el codi C
  - Codi condicional (ruptures de seqüència)
    - El bit z determinarà quin és el següent estat

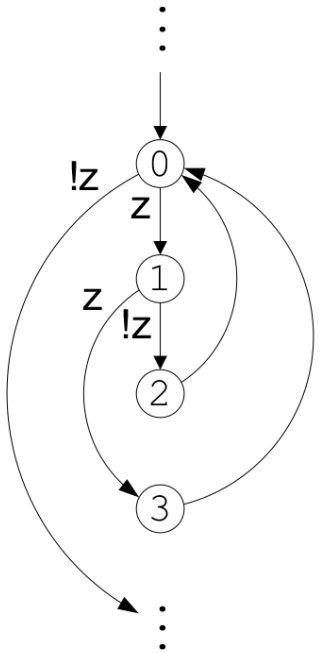
Representació de les comparacions a la UC

- Donades les limitacions de la UPG, és possible que calgui reescriure les comparacions del codi C al transformar-les en accions de la UPG
  - La UPG només té accions per comparacions tipus $=, <, \leq$
    - $A \neq B \iff !(A = B)$
    - $A > B \iff !(A \leq B) \iff B < A$
    - $A \geq B \iff !(A < B) \iff B \leq A$
  - Si un operand és constant, s'ha d'encaminar per l'entrada Y de la ALU
    - $constant < A \iff !(A \leq constant)$
    - $constant \leq A \iff !(A < constant)$
- Cal determinar el valor z adient per a cada transició del graf
  - $z = 0 \iff$ el resultat de la comparació feta a la UPG és TRUE
  - $z = 1 \iff$ el resultat de la comparació feta a la UPG és FALSE
  - z **no es comporta com el conveni típic per a booleans**

Exemple: UC per al càlcul del MCD

- Crearem el graf d'estats a partir del codi C
  - Assumint que els nombres estan a R0 i R1
  - Dades de tipus natural

```
while (R0 != R1) {
  if (R0 > R1) R0 = R0 - R1;
  else R1 = R1 - R0;
}
```




```
CMPEQ  -, R0, R1
CMPLTU -, R1, R0
SUB    R0, R0, R1
SUB    R1, R1, R0
```

- Cada node del graf d'estats representa una acció a executar a la UPG
  - La sortida serà el mnemotècnic o els 33 bits de la paraula de control
- El bit *z* permet implementar condicionals i bucles a la UC

Accions sinònimes a la UPG

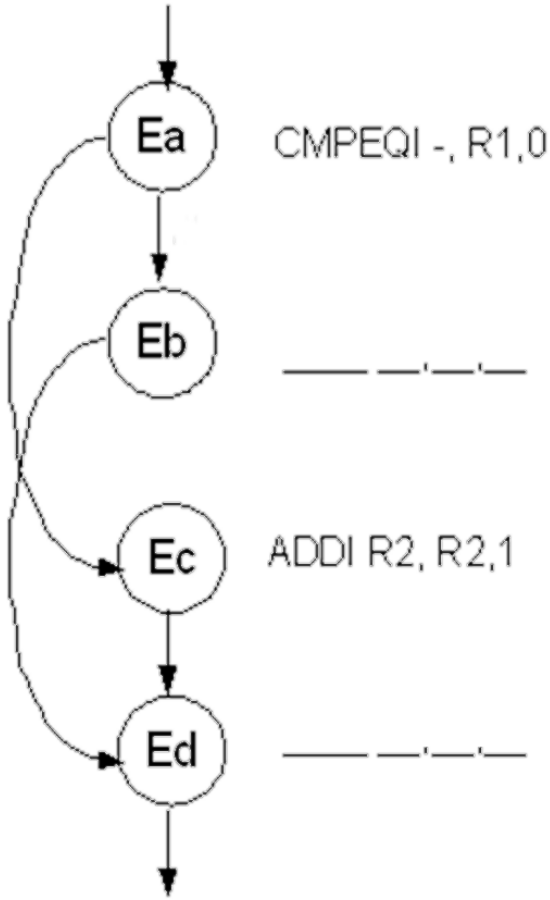
- És possible diverses accions provoquin el mateix canvi d'estat a la UPG
- Exemples:
  - Inicialitzar R1 a 0
    - `MOVEI R1, 0` ≡ `XOR R1, R1, R1` ≡ `SUB R1, R1, R1`
      `ANDI R1, R1, 0` ≡ `SHLI R1, R1, -16`
  - Copiar R2 sobre R1
    - `MOVE R1, R2` ≡ `ADDI R1, R2, 0` ≡ `SUBI R1, R2, 0`
      `ANDI R1, R2, 0xFFFF` ≡ `ORI R1, R2, 0` ≡ `SHLI R2, R1, 0`
  - Guardar a R2 el doble de R1
    - `ADD R2, R1, R1` ≡ `SHL R2, R1, +1` ≡ `SHA R2, R1, +1`
  - Incrementar en 1 R1
    - `ADDI R1, R1, 1` ≡ `SUBI R1, R1, -1`
  - Fer la NOT bit a bit de R4 i guardar-ho a R3
    - `NOT R3, R4` ≡ `XORI R3, R4, 0xFFFF`
- A UPG's més complexes, pot ser més eficient una acció que una altra

Índex

Exercici 8.5b

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

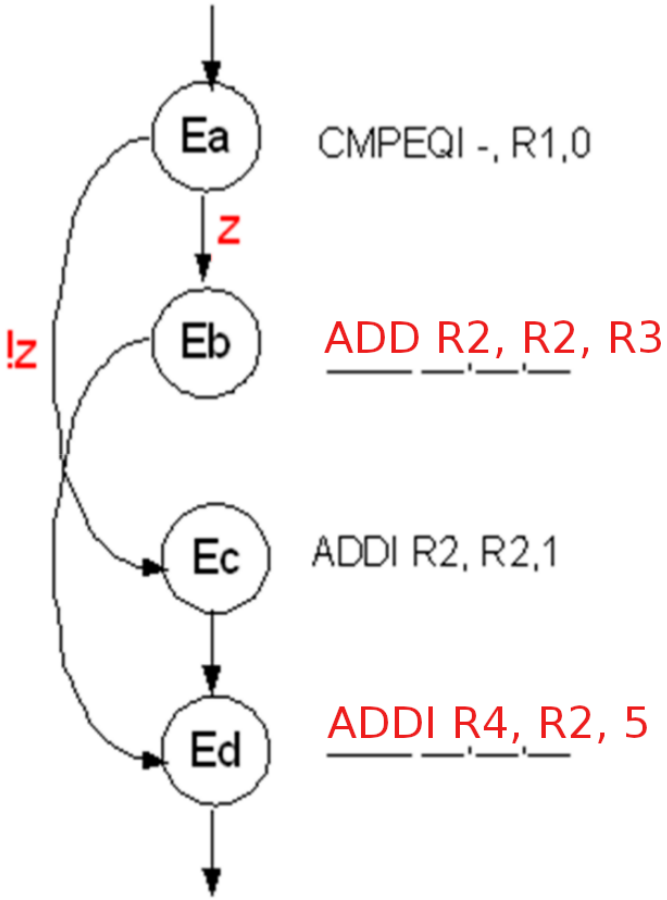
```
    .
    .
    .
if (R1 != 0)
  R2 = R2 + R3;
else
  R2 = R2 + 1;

R4=R2+5;
    .
    .
    .
```



Exercici 8.5b

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

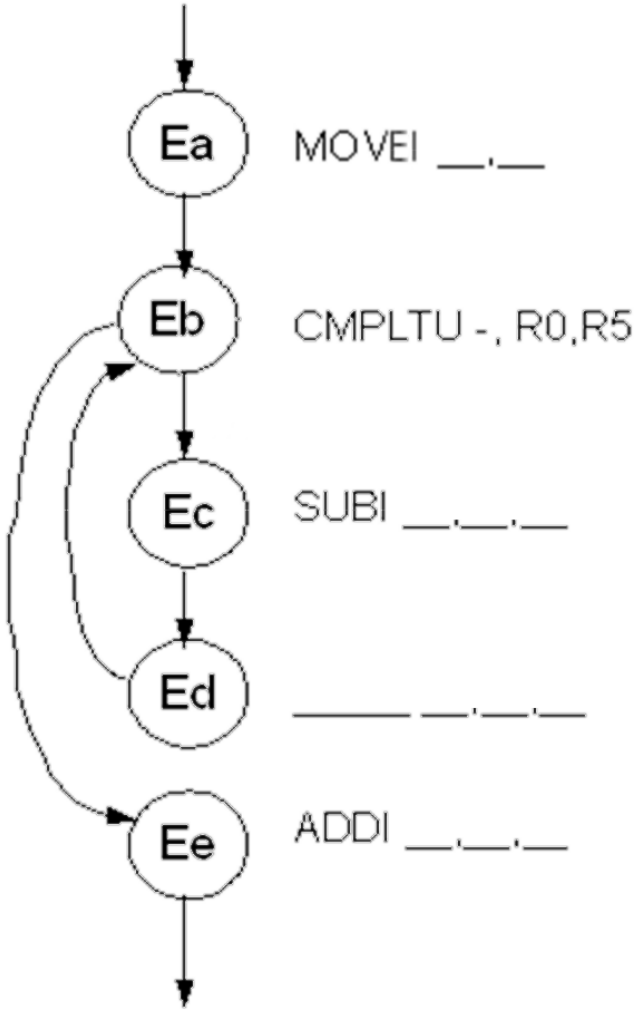
```
    .
    .
    .
if (R1 != 0)
  R2 = R2 + R3;
else
  R2 = R2 + 1;

R4=R2+5;
    .
    .
    .
```



Exercici 8.5c

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

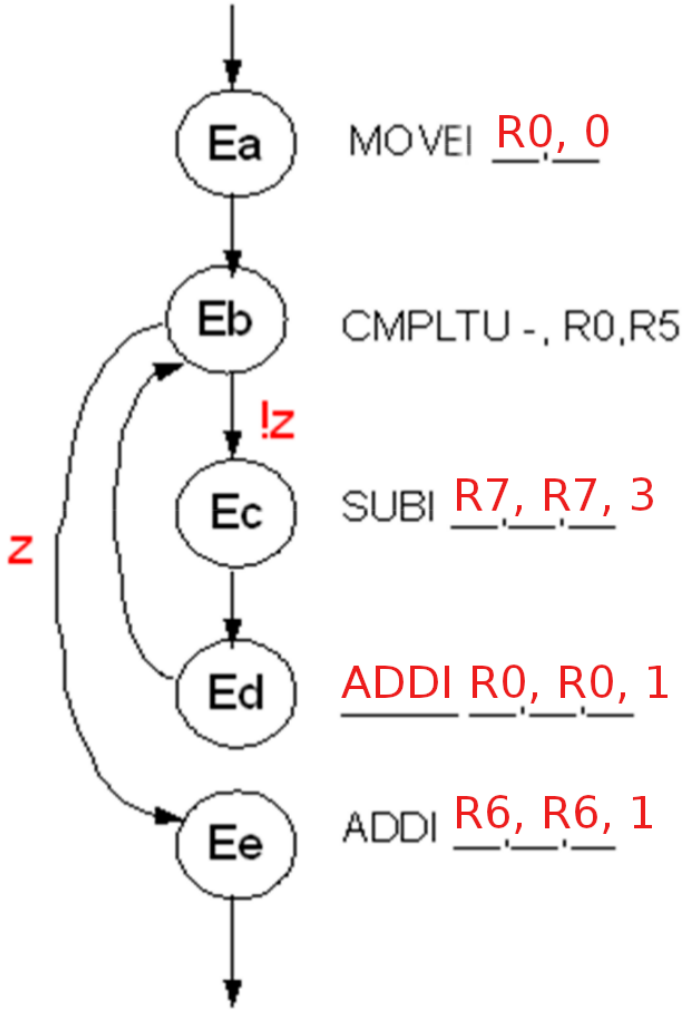
```
    .
    .
    .
for (R0 = 0; R0 < R5; R0++)
  {
    R7 = R7 - 3;
  }

R6++;
    .
    .
    .
```



Exercici 8.5c

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

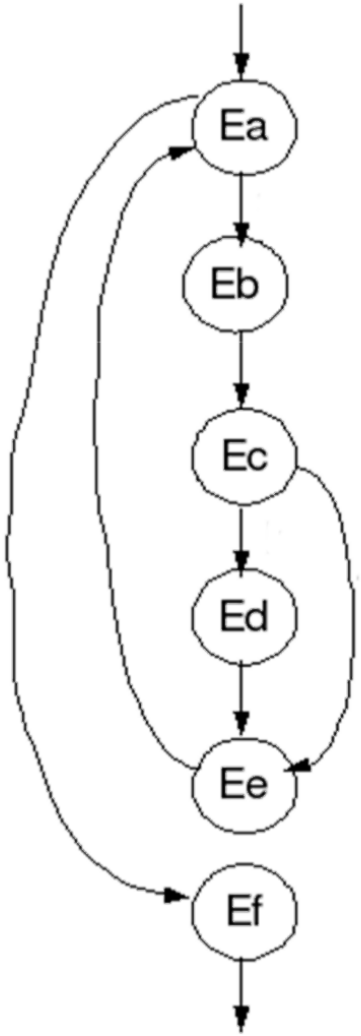
```
    .
    .
    .
for (R0 = 0; R0 < R5; R0++)
  {
    R7 = R7 - 3;
  }

R6++;
    .
    .
    .
```



Exercici 8.5d

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

```
  .
  .
  .
while (R5 != R4)
   {
     R2 =R2-1;
     if (R2 <= 3)  R1= R1+R4;
     R5++;
   }
R3 = R3/2;
  .
  .
  .
```



CMPEQ __,__,__

SUBI __,__,__

CMPLEUI -,R2,3

_____ __,__,__

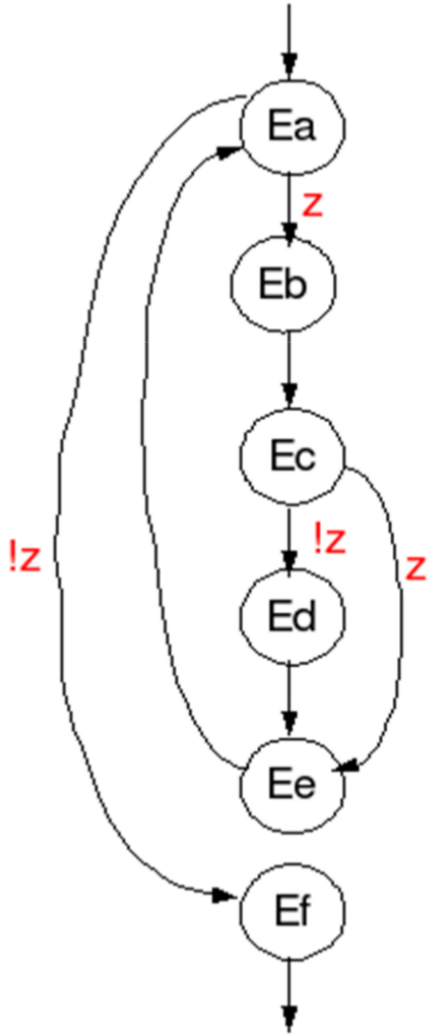
_____ __,__,__

SHAI __,__,__

Exercici 8.5d

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

```
  .
  .
  .
while (R5 != R4)
  {
    R2 =R2-1;
    if (R2 <= 3)  R1= R1+R4;
    R5++;
  }
R3 = R3/2;
  .
  .
  .
```



CMPEQ -, R5, R4

SUBI R2, R2, 1

CMPLEUI -,R2,3

ADD R1, R1, R4

ADDI R5, R5, 1

SHAI R3, R3, -1

Exercici 8.6a

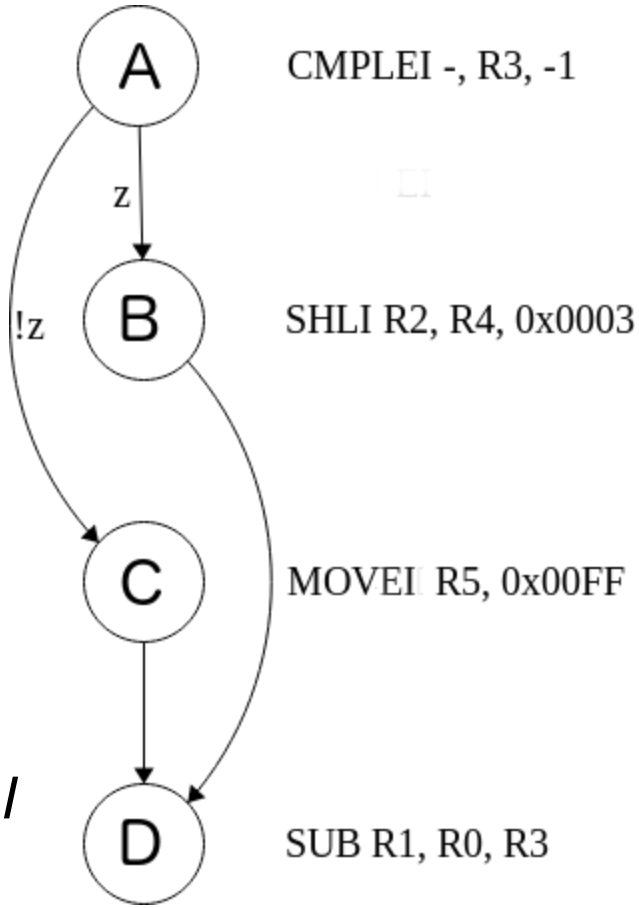
- Dibuixeu el fragment del graf d'estats de la UC que executaria aquest codi font sobre la UPG:

```
// dades de tipus enter
if (R3 > -1)
  R2 = R4 * 8;
else
  R5 = 0x00FF;
R1 = R0 - R3;
```

Exercici 8.6a

- Dibuixeu el fragment del graf d'estats de la UC que executaria aquest codi font sobre la UPG:

```
// dades de tipus enter
if (R3 > -1)
  R2 = R4 * 8;
else
  R5 = 0x00FF;
R1 = R0 - R3;
```



- També es podria haver utilitzat *SHAI*

Exercici 8.6b

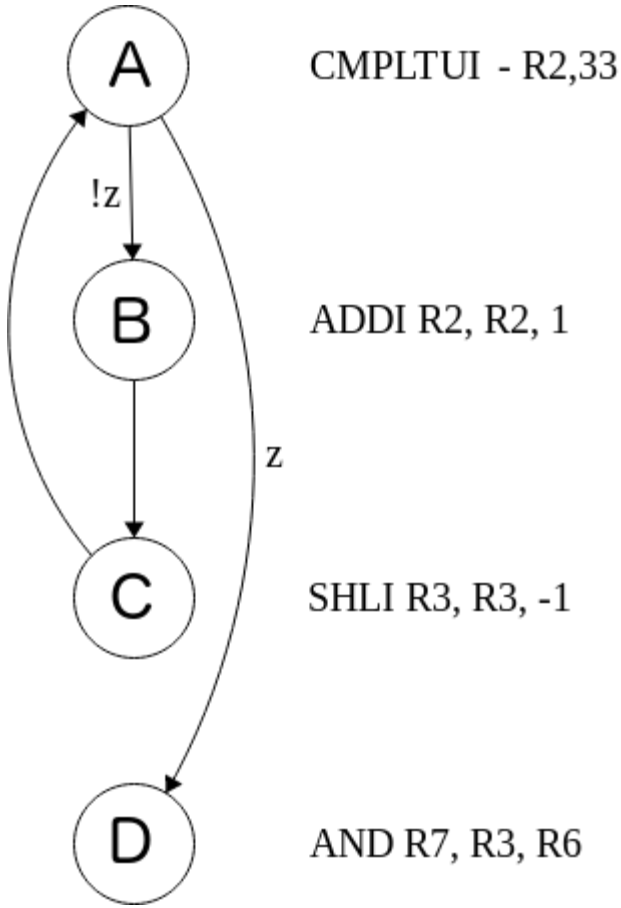
- Dibuixeu el fragment del graf d'estats de la UC que executaria aquest codi font sobre la UPG:

```
// dades de tipus natural
while (R2 < 33) {
  R2 = R2 + 1;
  R3 = R3 / 2;
}
R7 = R3 & R6; // AND bit a bit
```

Exercici 8.6b

- Dibuixeu el fragment del graf d'estats de la UC que executaria aquest codi font sobre la UPG:

```
// dades de tipus natural
while (R2 < 33) {
  R2 = R2 + 1;
  R3 = R3 / 2;
}
R7 = R3 & R6; // AND bit a bit
```

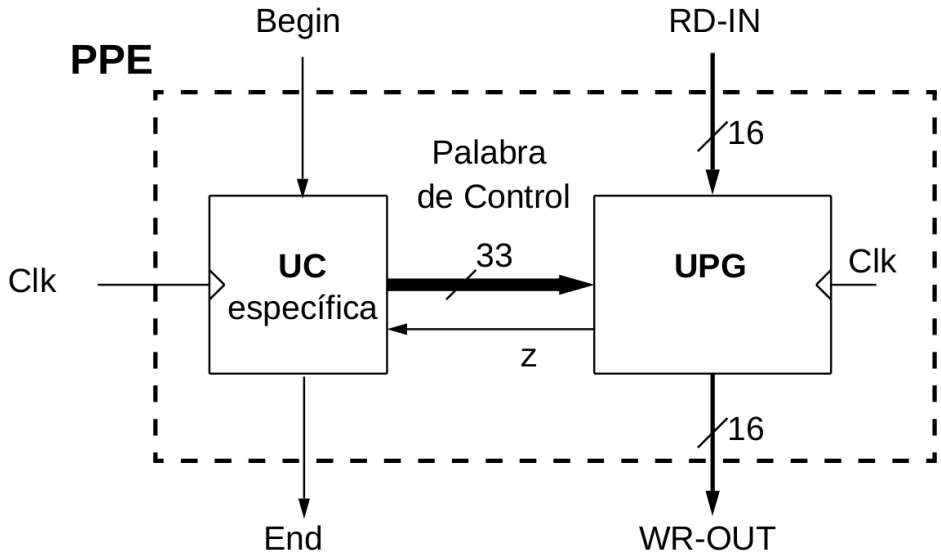


- Usem *SHL* perquè la dada és natural

Índex

Conclusions

- Hem creat una UPG per poder-la utilitzar a tots els problemes



  - La UPG té una paraula de control amb 33 bits
  - Utilitzarem mnemotècnics per representar les paraules de control
  - Cal saber convertir paraules de control en mnemotècnics i a l'inrevés
- Haurem de crear el graf d'estats de la UC:
  - Cada node representa una acció a executar a la UPG
  - Les transicions indiquen en quin ordre s'executen els nodes
  - El bit *z* permetrà implementar condicionals i bucles
- No oblideu realitzar el qüestionari d'Atenea ET8a

Referències I

Llevat que s'indiqui el contrari, les figures, esquemes, cronogrames i altre material gràfic o bé han estat extrets de la documentació de l'assignatura elaborada per Juanjo Navarro i Toni Juan, o corresponen a enunciats de problemes i exàmens de l'assignatura, o bé són d'elaboració pròpia.

[1] (1936). Les Luthiers: Instrumentología, [Online]. Available: http://www.lesluthiers.com/instrumentologia.php.

[2] [Online]. Available: https://www.makerfaireorlando.com/exhibits/the-one-man-band-marc-dobson/.

Introducció als Computadors

Tema 8: Unitat de Procés General (UPG)

http://personals.ac.upc.edu/enricm/Docencia/IC/IC8a.pdf

Enric Morancho
(enricm@ac.upc.edu)

Departament d'Arquitectura de Computadors
Facultat d'Informàtica de Barcelona
Universitat Politècnica de Catalunya

UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Facultat d'Informàtica de Barcelona

2020-21, 1^{er} quad.