

Nom i cognoms: .....

**PROVA DE PROBLEMES**

**Temps: 1h.** La puntuació està posada al costat de cada exercici. La prova puntua un total d'un punt. La revisió de l'examen es farà el mateix dia després de l'examen.

**1. MÀQUINA SENZILLA – SOFTWARE: La Potència de la MS1. (5 Punts )**

a. Disposem de la Màquina Senzilla modificada amb les instruccions **PUSH D**; **POP D**; **CALL D** i **RET** afegides al joc d'instruccions original: **ADD F, D**; **CMP F, D**; **MOV F, D** i **BEQ D**. Escriu un programa en ensamblador que calculi la següent expressió (4 Punts):

$$Pot(n) = n^n \quad \forall n \geq 0$$

Per exemple, els primers termes de l'expressió són:

<i>n</i>	0	1	2	3	4	5
<i>Pot(n)</i>	0 <sup>0</sup> = 1	1 <sup>1</sup> = 1	2 <sup>2</sup> = 4	3 <sup>3</sup> = 27	4 <sup>4</sup> = 256	...

Disposeu de la funció **Multiplica** implementada a l'adreça amb el mateix nom:

<i>Etiqueta</i>	@	<i>Contingut</i>	<i>Comentari</i>
<b>Multiplica:</b>	30	POP <b>AddrRetorn</b>	Obtenir el PC de retorn empilat per CALL
	31	POP <b>a</b>	Obtenir primer operand a l'adreça <b>a</b>
	32	POP <b>b</b>	Obtenir el segon operand a l'adreça <b>b</b>
<b>Iterar:</b>	33	MOV <b>Zero, c</b>	Inicialment el resultat és 0
	34	CMP <b>Zero, b</b>	S'ha acabat la multiplicació? ...
	35	BEQ <b>FiMultiplica</b>	... si <b>b</b> és 0 s'ha acabat la multiplicació
	36	ADD <b>a, c</b>	Acumular <b>a</b> al resultat
	37	ADD <b>MenysUn, b</b>	Decrementar <b>b</b>
	38	CMP <b>a, a</b>	Salt incondicional a la ...
	39	BEQ <b>Iterar</b>	... propera iteració
<b>FiMultiplica:</b>	40	PUSH <b>c</b>	Empilar el resultat
	41	PUSH <b>AddrRetorn</b>	Tornar l'adreça de retorn a la pila pel RET
	42	RET	Retornar a la rutina invocadora

Les dades (variables i constants) que utilitza la subrutina **Multiplica**:

<i>Etiqueta</i>	@	<i>Contingut</i>	<i>Tipus</i>
<b>Zero:</b>	100 <sub>d</sub>	0000 <sub>h</sub>	Constant
<b>MenysUn:</b>	101 <sub>d</sub>	FFFF <sub>h</sub>	Constant
<b>AddrRetorn:</b>	102 <sub>d</sub>	X	Variable
<b>a:</b>	103 <sub>d</sub>	X	Variable
<b>b:</b>	104 <sub>d</sub>	X	Variable
<b>c:</b>	105 <sub>d</sub>	X	Variable

Exemple de crida (multiplicar el contingut de les adreces 80 i 81 i posar el resultat a la 82):

@	<i>Contingut</i>	<i>Comentari</i>
<b>Exemple:</b>	PUSH 80	A l'adreça 80 hi ha el primer operand
	PUSH 81	A l'adreça 81 hi ha el segon operand
	CALL <b>Multiplica</b>	Fer la multiplicació
	POP 82	Posar el resultat a l'adreça 82

**NOTA:** El programa que heu d'implementar cal posar-lo a partir de l'adreça *Pot*. Podeu suposar que el paràmetre *Pot(n)* es troba a l'adreça *n* i el resultat s'escriu a l'adreça *r*. Heu d'ubicar en alguna adreça de memòria tant el programa com les dades (variables i constants) que definireu respectant les adreces que ja es troben ocupades.

La resolució del problema plantejat es correspon a una estructura iterativa on, a cada iteració, es realitza una multiplicació. Per exemple:

$$\text{Pot}(5) = 5^5 = 5 * 5 * 5 * 5 * 5 = 3125$$

Disposant de la subrutina que multiplica, només cal realitzar una estructura iterativa per obtenir el resultat de multiplicar *n* vegades la variable *n*. El cas particular de *n = 0* cal tenir en compte que el resultat és *1*.

Un algorisme en *pseudocodi* per resoldre aquest problema plantejat pot ser el següent:

```

i := 0;
r := 1;
Mentre i != n Fer
    r := r * n;
    i := i + 1;
FiMentre

```

on *i* és el comptador d'iteracions i *r* és on s'acumula el resultat.

#### Definició de les dades:

Per implementar l'algorisme anterior es necessiten les constants *0* i *1*. En les dades de l'enunciat ja es defineix la constant *0*, per tant, només caldrà afegir la constant *1*.

Quant a les variables, cal reservar espai pel comptador *i*, pel resultat final *r* i pel paràmetre *n*. Per tant, una possible ubicació per les dades pot ser la següent:

Etiqueta	@	Contingut	Tipus
Zero:	100 <sub>d</sub>	0000 <sub>h</sub>	Constant
MenysUn:	101 <sub>d</sub>	FFFF <sub>h</sub>	Constant
AddrRetorn:	102 <sub>d</sub>	X	Variable
a:	103 <sub>d</sub>	X	Variable
b:	104 <sub>d</sub>	X	Variable
c:	105 <sub>d</sub>	X	Variable
n:	106 <sub>d</sub>	Paràmetre n	Variable
i:	107 <sub>d</sub>	X	Variable
r:	108 <sub>d</sub>	X	Variable
Un:	109 <sub>d</sub>	0001 <sub>h</sub>	Constant

#### Implementació de l'algorisme:

Etiqueta	@	Contingut	Comentari
Pot:	0	MOV Zero, i	Posar el comptador d'iteracions a 0
	1	MOV Un, r	El resultat s'incialitza a 1
IterarPot:	2	CMP i, n	Acabar si el comptador ...
	3	BEQ AcabarMult	... és igual al número d'iteracions
	4	PUSH r	Empilar el primer valor a multiplicar
	5	PUSH n	Empilar el segon valor a multiplicar
	6	CALL Multiplica	Cridar a Multiplica
	7	POP r	Obtenir el resultat de la multiplicació a r
	8	ADD Un, i	Incrementar el comptador d'iteracions
	9	CMP Zero, Zero	Salt incondicional ...
	10	BEQ IterarPot	... a la nova iteració
AcabarMult:	11		



a. Quins són els senyals provinents de la **UP** que entren a la Unitat de Control i que serviran per canviar d'estat? Quins són els senyals provinents de la **UC** (el vector de sortides associat a cada estat) que serviran per controlar la Unitat de Procés? (**0.5 Punt**).

Senyals que van des de la UP a la Unitat de Control:

Són 3 línies d'un bit que ja existien abans de la modificació (**FZ**, **CO<sub>1,0</sub>**) més un senyal afegit al codi d'operació: **E**. En total són 3 línies d'un bit.

Senyals que van des de la UC a la Unitat de Procés:

Són 10 línies d'un bit que ja existien abans de la modificació ( **$\bar{L}/E$** , **@+I** → **PC**, **MUX<sub>1,0</sub>**, **M** → **IR**, **M** → **A**, **M** → **B**, **Z** → **FZ**, **ALU<sub>1,0</sub>**) més dos nous senyals pel multiplexor afegit: **MUXA<sub>1,0</sub>**. En total són 12 línies d'un bit.

b. Redefiniu el format d'instrucció original per poder afegir les dues noves instruccions (**1 Punt**).

Cal substituir la instrucció **CMP F, D** per la nova instrucció **SUB F, D**. En aquest cas, com que els operands coincideixen, simplement s'elimina **CMP** i **SUB** adopta el codi d'operació (**01**) que tenia **CMP**.

En el cas de **DEC D**, cal incorporar aquesta nova instrucció i, la Màquina Senzilla original, només té dos bits (**CO<sub>0</sub>** i **CO<sub>1</sub>**) pel codi d'instrucció, per tant, cal utilitzar la tècnica de l'extensió del format per poder-la afegir: a partir de la instrucció **BEQ** que no té operands s'afegirà un nou bit (**E**) per codificar la nova instrucció.

Per tant, en total, s'obtidran 2 formats d'instrucció diferents:

1. Instruccions originals de dos operands (**F, D**): **ADD** i **MOV** i la instrucció **SUB** que substitueix **CMP**:

CO <sub>1</sub>	CO <sub>0</sub>	F (7 Bits)	D (7 Bits)
-----------------	-----------------	------------	------------

2. Instruccions amb un operand (**D**): l'original **BEQ** i l'afegida **DEC**:

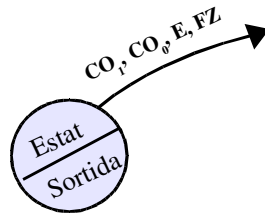
CO <sub>1</sub>	CO <sub>0</sub>	E	X (6 Bits)	D (7 Bits)
-----------------	-----------------	---	------------	------------

Una taula de codificació de les instruccions pot ser la següent:

Instrucció	CO <sub>1</sub>	CO <sub>0</sub>	E
ADD F D	0	0	X
SUB F D	0	1	X
MOV F D	1	0	X
BEQ D	1	1	0
DEC D	1	1	1

c. A partir de la Màquina Senzilla original, amb l'autòmat d'estats sense simplificar, afegiu els estats per dur a terme l'execució de les dues noves instruccions. Cal definir clarament quines són les accions que es realitzen a cada estat així com el vector de sortides associat a cada estat. Cal dibuixar l'autòmat complet per totes les instruccions de la màquina, tant les que s'han modificat com les que no (**3.5 Punts**).

Tenint en compte el problema plantejat, els estats i les transicions es codificaran de la següent manera:



Es realitzarà l'autòmat d'estats afegint les instruccions **DEC D** i **SUB F, D** a partir de l'autòmat d'estats sense simplificar de la Màquina Senzilla original. Les fases per dur a terme l'execució d'aquestes instruccions es descriuen a la següent taula.

Instrucció DEC D:

Per dur a terme l'execució de **DEC**, cal tenir en compte que s'operarà en complement a 2:

$$D - 1 = D + C_2(1)$$

Tenint en compte que el contingut de **D** és de 16 bits:

$$D - 1 = D + FFFF_h$$

La unitat de procés disposa de l'operació de sumar i pot carregar la constant **FFFF<sub>h</sub>** als registres, per tant, s'ha de poder resoldre l'operació plantejada.

FASE	TASCA	ESTATS	ACCIONS
1	Fetch de la instrucció	0	$IR \leftarrow (PC); PC \leftarrow PC + 1$
2	Descodificar la instrucció	1	Avaluar $CO_1, CO_0, E$
3	Lectura d'Operands	$16_a$	$A \leftarrow FFFF_h$
3	Lectura d'Operands	$16_b$	$B \leftarrow (D)$
4	Executar la instrucció	$17^*$	$(D) \leftarrow A + B; FZ \leftarrow (A + B) = 0$

Les tasques que es porten a terme als estats **16<sub>a</sub>** i **16<sub>b</sub>** de la fase de càrrega d'operands es poden fer al mateix temps. Per tant, es poden fusionar aquests estats en un de sol (**16**):

FASE	TASCA	ESTATS	ACCIONS
1	Fetch de la instrucció	0	$IR \leftarrow (PC); PC \leftarrow PC + 1$
2	Descodificar la instrucció	1	Avaluar $CO_1, CO_0, E$
3	Lectura d'Operands	16	$A \leftarrow FFFF_h; B \leftarrow (D)$
4	Executar la instrucció	$7^*$	$(D) \leftarrow A + B; FZ \leftarrow (A + B) = 0$

\*El darrer estat (**17**) de **DEC D** coincideix amb el darrer estat (**7**) d'**ADD F, D**, per tant, s'ha reutilitzat aquest estat d'**ADD**.

Instrucció SUB F, D:

Per dur a terme aquesta instrucció cal tenir en compte que s'operarà en complement a 2:

$$F - D = F + (-D) = F + C_2(D)$$

El complement a 2 es pot resoldre de la següent forma:

$$C_2(D) = C_1(D) + 1$$

El complement a 1 es pot fer amb una **XOR** bit a bit amb una seqüència d'uns. Tenint en compte que **D** és de 16 bits:

$$C_1(D) = (D \oplus FFFF_h) + 1$$

Per tant, la Unitat de Procés de la **MS1** ha de realitzar la següent operació per resoldre el problema:

$$F - D = F + (D \oplus FFFF_h) + 1$$

La **Unitat de Procés** plantejada ja disposa de l'operació de suma, l'operació **XOR** i es poden carregar els operands i les constants **FFFF<sub>h</sub>** i **1** als registres, per tant, s'ha de poder resoldre el problema plantejat.

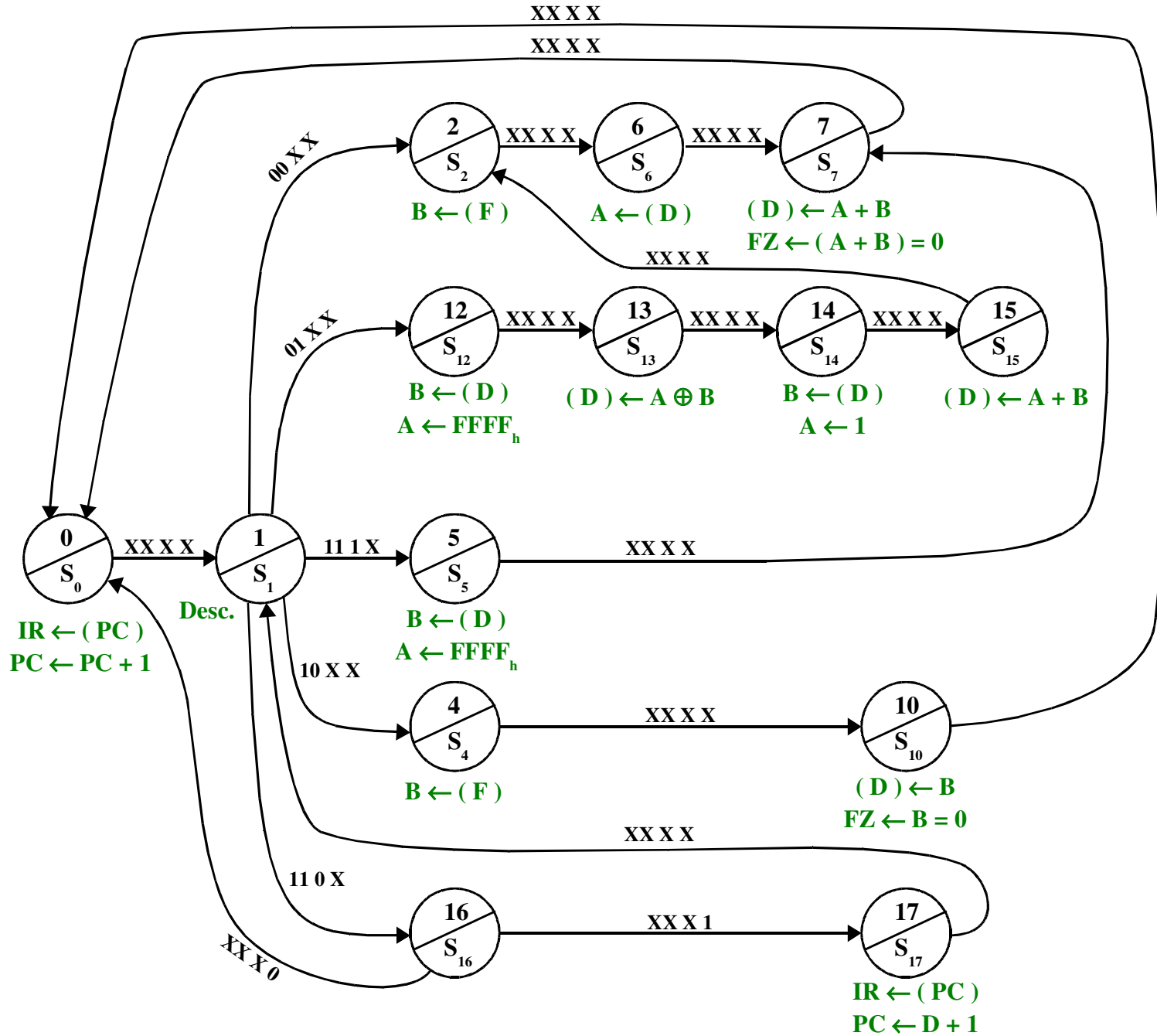
En aquest cas, cal tenir en compte, que s'utilitzarà l'adreça destí (**D**) per deixar resultats intermedis ja que no es disposa de cap registre per fer-ho. Això es pot fer sense problemes ja que, finalment, el resultat s'acaba guardant a **D** i, per tant, aquesta adreça ja s'havia de modificar amb el resultat final de l'operació.

FASE	TASCA	ESTATS	ACCIONS
1	Fetch de la instrucció	0	IR ← ( PC ); PC ← PC + 1
2	Descodificar la instrucció	1	Avaluar CO <sub>1</sub> , CO <sub>0</sub> , E
3	Lectura d'Operands	12 <sub>a</sub>	B ← ( D )
3	Lectura d'Operands	12 <sub>b</sub>	A ← FFFF <sub>h</sub>
4	Executar	13	( D ) ← A ⊕ B
3	Lectura d'Operands	14 <sub>a</sub>	B ← ( D )
3	Lectura d'Operands	14 <sub>b</sub>	A ← 1
4	Executar	15	( D ) ← A + B
3	Lectura d'Operands	3*	B ← ( F )
3	Lectura d'Operands	8*	A ← ( D )
4	Executar la instrucció	9*	( D ) ← A + B; FZ ← ( A + B ) = 0

Les tasques que es porten a terme en els estats **12<sub>a</sub>** i **12<sub>b</sub>**, de la fase de càrrega d'operands es poden fer al mateix temps, per tant, es poden fusionar aquests estats. Això mateix passa amb els estats **14<sub>a</sub>** i **14<sub>b</sub>**, per tant, també es poden fusionar.

FASE	TASCA	ESTATS	ACCIONS
1	Fetch de la instrucció	0	IR ← ( PC ); PC ← PC + 1
2	Descodificar la instrucció	1	Avaluar CO <sub>1</sub> , CO <sub>0</sub> , E
3	Lectura d'Operands	12	B ← ( D ); A ← FFFF <sub>h</sub>
4	Executar	13	( D ) ← A ⊕ B
3	Lectura d'Operands	14	B ← ( D ); A ← 1
4	Executar	15	( D ) ← A + B
3	Lectura d'Operands	2*	B ← ( F )
3	Lectura d'Operands	6*	A ← ( D )
4	Executar la instrucció	7*	( D ) ← A + B; FZ ← ( A + B ) = 0

\*Els tres darrers estats (**3**, **8** i **9**) de **SUB F, D** coincideixen amb els tres darrers estats (**2**, **6** i **7**) d'**ADD F, D**, per tant, s'han reutilitzat aquests tres estats de la instrucció **ADD**.



**ADD F, D**

**SUB F, D**

**DEC D**

**MOV F, D**

**BEQ D**

El vector de sortides associat a cada estat  $S_i$  és de 12 bits: els 10 originals més els dos afegits per la modificació ( $MUXA_1$  i  $MUXA_0$ ). La següent taula especifica els seus valors:

	Fetch	Desc.		Operands		Operands				Op.	Op.	Exec.	Exec.	Exec.
	$S_0$	$S_1$	$S_{16}$	$S_2$	$S_6$	$S_{12}$	$S_{13}$	$S_{14}$	$S_{15}$	$S_5$	$S_4$	$S_7$	$S_{10}$	$S_{17}$
$MUX_1$	0	X	X	1	1	1	1	1	1	1	1	1	1	0
$MUX_0$	0	X	X	0	1	1	1	1	1	1	0	1	1	0
$ALU_1$	X	X	X	X	X	X	0	X	0	X	X	0	1	X
$ALU_0$	X	X	X	X	X	X	1	X	0	X	X	0	0	X
$L' / E$	0	0	0	0	0	0	1	0	1	0	0	1	1	0
$PC \leftarrow @ + 1$	1	0	0	0	0	0	0	0	0	0	0	0	0	1
$IR \leftarrow M$	1	0	0*	0	0	0	0	0	0	0	0	0*	0*	1
$A \leftarrow M$	0*	0*	0*	0*	1	1	0*	1	0*	1	0*	0*	0*	0*
$B \leftarrow M$	0*	0*	0*	1	0	1	0*	1	0*	1	1	0*	0*	0*
$FZ \leftarrow Z$	0	0	0	0*	0*	0*	0*	0*	0*	0*	0*	1	1	0
$MUXA_1$	X	X	X	X	0	1	X	0	X	1	X	X	X	X
$MUXA_0$	X	X	X	X	0	0	X	1	X	0	X	X	X	X

\*Usualment s'utilitza un 0, però en realitat el seu valor és indiferent (X).

Com que s'ha partit de la MS1 sense simplificar, en aquest autòmat es poden fusionar els estats  $S_1$  i  $S_{16}$  del **BEQ** per completar la seva descodificació només amb l'estat  $S_1$ : anar directament a l'estat  $S_{17}$  en cas que calgui saltar ( $CO_1 CO_0 E FZ = 1101$ ) o bé anar directament a l'estat  $S_0$  en cas que no calgui saltar ( $CO_1 CO_0 E FZ = 1100$ ). D'aquesta forma es pot suprimir totalment l'estat  $S_{16}$ .