

Nom i cognoms:.....

**PROVA DE PROBLEMES**

Temps: 1h 15m. Obtenint 3 o més punts sobre 10, la prova puntuarà el 50% del total de l'examen. La puntuació està posada al costat de cada exercici. La revisió de l'examen es farà el dia 14 a l'hora de classe (15:00h).

**1.- MÀQUINA SENZILLA – SOFTWARE: La sèrie de Fibonacci. (5 Punts )**

**a.-** Escriu un programa en ensamblador de la Màquina Senzilla que calculi l'*n*-èssim valor de la sèrie de *Fibonacci*. (4 Punts).

Recorda que la sèrie de *Fibonacci* comença per 0 i 1 i el següent element amb posició *n* s'obté a partir de la suma dels dos elements de les posicions anteriors *n - 1* i *n - 2*. Una definició formal de la sèrie pot ser la següent:

$$Fib(n) = \begin{cases} n & n \geq 0 \wedge n \leq 1 \\ Fib(n-1) + Fib(n-2) & \forall n > 1 \end{cases}$$

Per exemple, els primers termes de la sèrie són:

<i>n</i>	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
<i>Fib(n)</i>	0	1	1	2	3	5	8	13	21	34	55	89	144	233	377	610	987	1597	2584	4181	6765	...

**NOTA:** Per implementar el programa, es poden utilitzar, de forma orientativa, les següents dades:

<i>Etiqueta</i>	@	<i>Contingut</i>
<b>zero:</b>	100	0
<b>un:</b>	101	1
<b>n:</b>	102	<i>n</i>
<b>i:</b>	103	X
<b>Fib0:</b>	104	X
<b>Fib1:</b>	105	X
<b>resultat:</b>	106	X

També pot ser d'ajuda el següent algorisme iteratiu en *pseudocodi* que soluciona el problema:

```

i := 0; // Inicialitzar comptador
resultat := 0; // Calcular Fib(0)
Fib0 := 1; // Calcular Fib(1)
Mentre i <> n Fer // Mentre no s'arribi a l'n-èssim
    Fib1 := resultat + Fib0; // Calcular el següent element
    Fib0 := resultat; // Guardar l'anterior
    resultat := Fib1; // A resultat posar-hi el següent
    i := i + 1; // Incrementar el comptador
F Mentre
    
```

Abans d'implementar un algorisme utilitzant assemblador de la MS1 és bo d'implementar-lo utilitzant un **llenguatge d'alt nivell** com, per exemple, el *pseudocodi* ja que resulta més senzill traduir-lo d'aquí que implementar-lo tot de nou directament en assemblador. Com que en aquest exercici ja es troba implementat en *pseudocodi* i, a més a més, hi ha les variables definides, només caldrà traduir **cada instrucció i estructura del pseudocodi** en llenguatge **assemblador de la MS1**.

La idea de l'algorisme implementat en *pseudocodi* és tenir sempre calculat l'element actual i el següent element de la sèrie en dues variables (*Fib0* i *Fib1*). Es calcula si s'ha acabat de computar l'*n*-èssim valor de la sèrie utilitzant un comptador (variable *i*) que es va incrementant des de 0 fins a *n*.

Com que s'han utilitzat les mateixes variables que hi ha definides a l'enunciat, es pot traduir l'algorisme directament del *pseudocodi* a assemblador de la MS1 utilitzant les mateixes etiquetes:

### Programa en assemblador utilitzant etiquetes:

<i>Etiqueta</i>	@	<i>Contingut</i>
<b>inici:</b>	0	mov zero, i
	1	mov zero, resultat
	2	mov un, Fib0
<b>iterar:</b>	3	cmp i, n
	4	beq fi
	5	mov resultat, Fib1
	6	add Fib0, Fib1
	7	mov resultat, Fib0
	8	mov Fib1, resultat
	9	add un, i
	10	cmp zero, zero
	11	beq iterar
<b>fi:</b>	12	beq fi

L'única “*dificultat*” de la traducció es troba en la primera instrucció de l'estructura iterativa (**Mentre – FiMentre**): “*Fib1 := resultat + Fib0;*”

Aquesta instrucció no es pot traduir en una sola instrucció d'assemblador de la MS1, calen el **mov** i l'**add** de les adreces 5 i 6. Existeix una altra alternativa per fer la traducció que, tot i que és menys elegant (ja que modifica *Fib0*), també és correcta:

...	...	...
5	add	resultat, Fib0
6	mov	Fib0, Fib1
...	...	...

Un cop traduït el programa es poden substituir les etiquetes de programa i les variables per adreces físiques:

### Programa en assemblador utilitzant adreces:

```

0:   mov 100, 103
     mov 100, 106
     mov 101, 104
3:   cmp 103, 102
     beq 12
     mov 106, 105
     add 104, 105
     mov 106, 104
     mov 105, 106
     add 101, 103
     cmp 100, 100
     beq 3
12:  beq 12

```

Un cop feta la traducció a adreces es pot fer la codificació de les instruccions fàcilment a codi màquina (**Binari i Hexadecimal**) traduint els diferents camps del format d'instrucció: **codi d'operació**, **adreça de l'operand font** i **adreça de l'operand destí**:

**Programa en llenguatge màquina (Binari i Hexadecimal):**

0:	<b>mov</b> 100, 103	0:	<b>1011 0010 0110 0111</b>	0:	<b>B267<sub>h</sub></b>
	<b>mov</b> 100, 106	1:	<b>1011 0010 0110 1010</b>	1:	<b>B26A<sub>h</sub></b>
	<b>mov</b> 101, 104	2:	<b>1011 0010 1110 1000</b>	2:	<b>B2E8<sub>h</sub></b>
3:	<b>cmp</b> 103, 102	3:	<b>0111 0011 1110 0110</b>	3:	<b>73E6<sub>h</sub></b>
	<b>beq</b> 12	4:	<b>11XX XXXX X000 1100*</b>	4:	<b>C00C<sub>h</sub></b>
	<b>mov</b> 106, 105	5:	<b>1011 0101 0110 1001</b>	5:	<b>B569<sub>h</sub></b>
	<b>add</b> 104, 105	6:	<b>0011 0100 0110 1001</b>	6:	<b>3469<sub>h</sub></b>
	<b>mov</b> 106, 104	7:	<b>1011 0101 0110 1000</b>	7:	<b>B568<sub>h</sub></b>
	<b>mov</b> 105, 106	8:	<b>1011 0100 1110 1010</b>	8:	<b>B4EA<sub>h</sub></b>
	<b>add</b> 101, 103	9:	<b>0011 0010 1110 0111</b>	9:	<b>32E7<sub>h</sub></b>
	<b>cmp</b> 100, 100	10:	<b>0111 0010 0110 0100</b>	10:	<b>7264<sub>h</sub></b>
	<b>beq</b> 3	11:	<b>11XX XXXX X000 0011*</b>	11:	<b>C003<sub>h</sub></b>
12:	<b>beq</b> 12	12:	<b>11XX XXXX X000 1100*</b>	12:	<b>C00C<sub>h</sub></b>

\*Com que l'operand font del **beq** no s'utilitza, es pot codificar aquesta instrucció de 128 formes diferents. En la codificació de la taula anterior s'ha utilitzat l'opció **X = 0**. Les altres possibilitats de codificació per les adreces 4, 11 i 12 es resumeixen en les següent expressions regulars:

4: [C|D|E|F] [0-F] [0|8] C<sub>h</sub>  
 11: [C|D|E|F] [0-F] [0|8] 3<sub>h</sub>  
 12: [C|D|E|F] [0-F] [0|8] C<sub>h</sub>

Finalment, el codi en el format del simulador de la MS1 de <http://eia.udg.es/ms> és el següent (per **n = 25**):

```
@: 0=  mov: 100, 103;
@: 1=  mov: 100, 106;
@: 2=  mov: 101, 104;
@: 3=  cmp: 103, 102;
@: 4=  beq: 0, 12;
@: 5=  mov: 106, 105;
@: 6=  add: 104, 105;
@: 7=  mov: 106, 104;
@: 8=  mov: 105, 106;
@: 9=  add: 101, 103;
@: 10= cmp: 100, 100;
@: 11= beq: 0, 3;
@: 12= beq: 0, 12;
...
@: 100= 0;
@: 101= 1;
@: 102= 25;
@: 103= 0;
@: 104= 0;
@: 105= 0;
@: 106= 0;
...
```

**b.-** Si s'interpreten sense signe els valors emmagatzemats a la memòria de la MS1, quin és el màxim valor d'**n** perquè no hi hagi cap desbordament (*overflow*) i, per tant, el resultat de l'execució del programa implementat sigui correcte? (1 Punt)

Continuant la sèrie de l'enunciat a partir dels valors 19 i 20 fins al valor 29 s'obté la següent taula:

<i>n</i>	18	19	20	21	22	23	24	25	26	27	28	29	30
<i>Fib (n)</i>	...	4181	6765	10946	17711	28657	46368	75025	121393	196418	317811	514229	...

El rang de representació amb **k bits** pels naturals (enters sense signe) és:

$$[0 .. 2^k - 1]$$

Tenint en compte que les cel·les de memòria amb les quals pot operar la MS1 original són de **16 bits** i que s'interpreten sense signe, en una cel·la hi cap (**k = 16**):

$$[0 .. 2^{16} - 1] = [0 .. 65536 - 1] = [0 .. 65535]$$

El darrer terme de la sèrie que es pot representar sense *overflow* és el **25<sup>è</sup>**, el de la **posició n = 24: 46368**.

## 2.- MÀQUINA SENZILLA – HARDWARE: Comparació amb Prefech. (5 Punts )

Es vol modificar la versió de la Màquina Senzilla original amb l'autòmat d'estats de la Unitat de Control sense simplificar per tal d'optimitzar una mica (en temps) l'execució dels programes.

Durant la fase d'execució de la instrucció **CMP** (estat  $S_9$  a les transparències de classe) **no** s'està utilitzant la memòria. Donat que després de l'execució d'un **CMP** sempre cal executar la instrucció de l'adreça següent (que està apuntada pel PC), es pot avançar el Fetch de la següent instrucció i fer-lo a l'estat d'execució del **CMP**.

### Es demana:

- a.- Definir quines són **les accions** que s'han de dur a terme a cada una de les Fases (*Fetch*, Descodificació, Càrrega d'Operands i Execució) de la nova versió de la instrucció **CMP**. (1.5 Punts)

La instrucció **CMP** de la Màquina Senzilla Original amb l'autòmat d'estats sense simplificar té les següent Fases amb els estats associats indicats:

### CMP

<b>Fase 1:</b> (Fetch)	Estat $S_0$	$IR \leftarrow (PC); PC \leftarrow PC + 1$
<b>Fase 2:</b> (Descodificació)	Estat $S_1$	Avaluació de $CO_1$ i $CO_0$
<b>Fase 3:</b> (Càrrega d'Operands)	Estat $S_3$	$B \leftarrow (F)$
<b>Fase 3:</b> (Càrrega d'Operands)	Estat $S_8$	$A \leftarrow (D)$
<b>Fase 4:</b> (Execució)	Estat $S_9$	$A \oplus B; FZ \leftarrow Z$

Quant a les Fases, la nova instrucció **CMP** tindrà les mateixes excepte la d'Execució (4) que canviarà l'estat  $S_9$  d'aquesta per tal d'afegir-hi el *Fetch*:

### CMP

<b>Fase 1:</b> (Fetch)	Estat $S_0$	$IR \leftarrow (PC); PC \leftarrow PC + 1$
<b>Fase 2:</b> (Descodificació)	Estat $S_1$	Avaluació de $CO_1$ i $CO_0$
<b>Fase 3:</b> (Càrrega d'Operands)	Estat $S_3$	$B \leftarrow (F)$
<b>Fase 3:</b> (Càrrega d'Operands)	Estat $S_8$	$A \leftarrow (D)$
<b>Fase 4:</b> (Execució)	Estat $S_9$	$A \oplus B; FZ \leftarrow Z; IR \leftarrow (PC); PC \leftarrow PC + 1$

- b.- Modificar, si cal, el format d'instrucció. (0.25 Punts)

No s'ha afegit cap instrucció nova ni s'ha eliminat cap instrucció. La modificació que s'ha realitzat, no afecta a la codificació de cap instrucció (codi d'operació, operand font o operand destí), per tant, el format d'instrucció no s'ha de modificar per res.

De fet, la modificació és una optimització per l'execució dels programes futurs. Els ja existents per la **Màquina Senzilla** original, s'han d'executar igualment tot i la millora interna de la instrucció **CMP**. En general, els programes no s'haurien de veure afectats en el seu funcionament ni codificació a causa d'una **millora interna** en una CPU.

- c.- Modificar la Unitat de Procés, si cal, per executar la nova versió de **CMP**. (0.25 Punts)

La nova versió de **CMP** només ha de realitzar **més operacions** de les que ja realitza actualment a la seva **Fase 4** (execució). Aquestes noves operacions no són res més que un *Fetch* i això, **ja es pot fer** amb la UP actual de la MS1. Per tant, no hi haurà **cap canvi** a la **Unitat de Procés**.

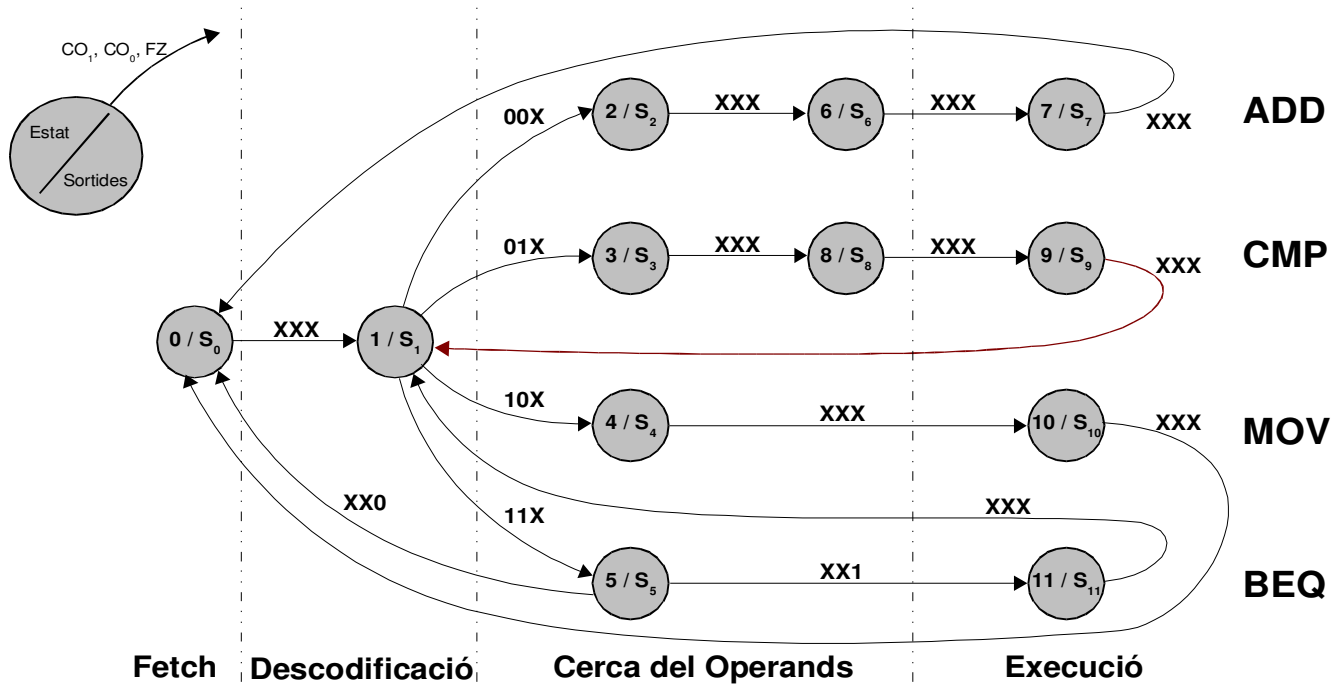
d.- Modificar l'autòmat d'estats de la Unitat de Control, si cal, per executar la nova versió de **CMP**.  
(1.25 Punts)

Com que la nova versió de la instrucció del **CMP** ha de realitzar un **Fetch** a l'estat  $S_9$  de la seva Fase 4 (d'execució), cal fer modificacions a l'autòmat d'estats de la UC per dur a terme aquesta tasca correctament.

L'estat d'execució ja existia i no s'ha afegit cap tasca que no es pugui fer en aquest mateix estat existent d'execució ( $S_9$ ). A més a més, no s'ha modificat la unitat de procés, ni la codificació de les instruccions, per tant, **les arestes del graf encara continuen essent a partir dels dos bits de codi d'operació** ( $CO_1$  i  $CO_0$ ) i del **Flag de Zero** (FZ).

Com que en aquest estat  $S_9$  d'execució s'hi ha afegit un **Fetch**, caldrà moure l'aresta del graf a l'estat  $S_1$  de **descodificació** enlloc d' $S_0$  de **Fetch** que hi havia a l'autòmat de la MS1 abans de la modificació.

**Nou autòmat d'estats:**



e.- Definir el vector de sortides de la Unitat de Control pels nous estats que hagin aparegut i redefinir el vector pels estats que hagin canviat. (0.75 Punts)

Només ha canviat l'estat  $S_9$  que resulta ser una **fusió** de l'estat  $S_9$  anterior i l'estat  $S_0$  de **Fetch**:

	$S_0$	$S_1$	$S_3$	$S_8$	$S_9$
$MX_1$	0	X	1	1	0
$MX_0$	0	X	0	1	0
$ALU_1$	X	X	X	X	0
$ALU_0$	X	X	X	X	1
$\bar{L} / E$	0	0	0	0	0
$PC \leftarrow @+1$	1	0	0	0	1
$IR \leftarrow M$	1	0	0	0	1
$A \leftarrow M$	0	0	0	1	0
$B \leftarrow M$	0	0	1	0	0
$FZ \leftarrow Z$	0	0	0	0	1

- f.- La modificació realitzada estalvia cicles de relloige en l'execució d'un sol **CMP**? En cas afirmatiu, quants? I en l'execució d'un **CMP** al mig d'altres instruccions? En cas afirmatiu, quants?. (1 Punt)

Aquesta millora no optimitza en cap cicle l'execució d'un sol **CMP** ja que, per executar-se, cal que passi pels estats:

$S_0, S_1, S_3, S_8$  i  $S_9$

Per tant, la seva execució tarda igual que abans de la modificació: **5 cicles de relloige**.

En el cas de l'execució d'un **CMP** entre altres instruccions, concretament en l'execució d'una instrucció després del **CMP**, s'estalvia un cicle ja que **s'avança el Fetch** de la següent instrucció i aquesta no cal que executi aquest estat ( $S_0$ ). Es pot comprovar això contrastant els cicles que executava la MS1 abans de la millora i després d'aquesta.

**Màquina Senzilla original sense la millora:**

**ADD:**

<i>Adreça</i>	<i>Instrucció</i>	<i>Estats</i>	<i>Cicles</i>
i	<b>CMP</b>	$S_0, S_1, S_3, S_8$ i $S_9$	5
i + 1	<b>ADD</b>	$S_0, S_1, S_2, S_6$ i $S_7$	5

*Total = 10*

**CMP:**

<i>Adreça</i>	<i>Instrucció</i>	<i>Estats</i>	<i>Cicles</i>
i	<b>CMP</b>	$S_0, S_1, S_3, S_8$ i $S_9$	5
i + 1	<b>CMP</b>	$S_0, S_1, S_3, S_8$ i $S_9$	5

*Total = 10*

**MOV:**

<i>Adreça</i>	<i>Instrucció</i>	<i>Estats</i>	<i>Cicles</i>
i	<b>CMP</b>	$S_0, S_1, S_3, S_8$ i $S_9$	5
i + 1	<b>MOV</b>	$S_0, S_1, S_4, S_{10}$	4

*Total = 9*

**BEQ (amb FZ = 0):**

<i>Adreça</i>	<i>Instrucció</i>	<i>Estats</i>	<i>Cicles</i>
i	<b>CMP</b>	$S_0, S_1, S_3, S_8$ i $S_9$	5
i + 1	<b>BEQ</b>	$S_0, S_1, S_5, S_{11}$	4

*Total = 9*

**BEQ (amb FZ = 1):**

<i>Adreça</i>	<i>Instrucció</i>	<i>Estats</i>	<i>Cicles</i>
i	<b>CMP</b>	$S_0, S_1, S_3, S_8$ i $S_9$	5
i + 1	<b>BEQ</b>	$S_0, S_1, S_5$	3

*Total = 8*

### Màquina Senzilla original amb la millora del CMP:

#### ADD:

<i>Adreça</i>	<i>Instrucció</i>	<i>Estats</i>	<i>Cicles</i>
i	<b>CMP</b>	S <sub>0</sub> , S <sub>1</sub> , S <sub>3</sub> , S <sub>8</sub> i S <sub>9</sub>	5
i + 1	<b>ADD</b>	S <sub>1</sub> , S <sub>2</sub> , S <sub>6</sub> i S <sub>7</sub>	4

*Total = 9*

#### CMP:

<i>Adreça</i>	<i>Instrucció</i>	<i>Estats</i>	<i>Cicles</i>
i	<b>CMP</b>	S <sub>0</sub> , S <sub>1</sub> , S <sub>3</sub> , S <sub>8</sub> i S <sub>9</sub>	5
i + 1	<b>CMP</b>	S <sub>1</sub> , S <sub>3</sub> , S <sub>8</sub> i S <sub>9</sub>	4

*Total = 9*

#### MOV:

<i>Adreça</i>	<i>Instrucció</i>	<i>Estats</i>	<i>Cicles</i>
i	<b>CMP</b>	S <sub>0</sub> , S <sub>1</sub> , S <sub>3</sub> , S <sub>8</sub> i S <sub>9</sub>	5
i + 1	<b>MOV</b>	S <sub>1</sub> , S <sub>4</sub> , S <sub>10</sub>	3

*Total = 8*

#### BEQ (amb FZ = 0):

<i>Adreça</i>	<i>Instrucció</i>	<i>Estats</i>	<i>Cicles</i>
i	<b>CMP</b>	S <sub>0</sub> , S <sub>1</sub> , S <sub>3</sub> , S <sub>8</sub> i S <sub>9</sub>	5
i + 1	<b>BEQ</b>	S <sub>1</sub> , S <sub>5</sub> , S <sub>11</sub>	3

*Total = 8*

#### BEQ (amb FZ = 1):

<i>Adreça</i>	<i>Instrucció</i>	<i>Estats</i>	<i>Cicles</i>
i	<b>CMP</b>	S <sub>0</sub> , S <sub>1</sub> , S <sub>3</sub> , S <sub>8</sub> i S <sub>9</sub>	5
i + 1	<b>BEQ</b>	S <sub>1</sub> , S <sub>5</sub>	2

*Total = 7*

En tots els casos ens estalviem el cicle de *Fetch* de la instrucció que segueix el **CMP**, per tant, la millora, en general, estalvia un cicle per cada **CMP** que es trobi en un programa.

**NOTA:** Cal raonar adequadament cada una de les respostes.