

## ET14 (Llenguatge màquina i assembler SISA)

### Exercicis per avaluar objectius de nivell B

#### Exercici 14.1

Donat el següent programa en llenguatge assembler del SISA:

```
.data
    v1:    .byte 1,2,3
    v2:    .space 3, 0x64
    res:    .space 3
.text
    CONJUNT_INSTRUCCIONS
.end
```

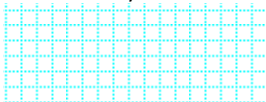


Doneu el valor contingut al registre R0 després d'executar el programa. Assumiu que el primer valor especificat a l'apartat encapçalat per la directiva ".data" es posa a la posició 0x0A47 de la memòria. Contesteu cada apartat assumint que el conjunt d'instruccions és el que s'especifica a cada cas:

- a) El conjunt d'instruccions és:  
    MOVI R0, lo(v2)  
    MOVHI R0, hi(v2)
- b) El conjunt d'instruccions és:  
    MOVI R1, lo(v1)  
    MOVHI R1, hi(v1)  
    LDB R0, 2(R1)
- c) El conjunt d'instruccions és:  
    MOVI R1, lo(res)  
    MOVHI R1, hi(res)  
    LDB R0, 0(R1)
- d) El conjunt d'instruccions és:  
    MOVI R7, 3  
    MOVI R0, lo(v1)  
    MOVHI R0, hi(v1)  
    MOVI R1, lo(v2)  
    MOVHI R1, hi(v2)  
    MOVI R2, lo(res)  
    MOVHI R2, hi(res)  
    bucle: ADDI R7, R7, -1  
    ADD R6, R0, R7  
    LDB R3, 0(R6)  
    ADD R6, R1, R7  
    LDB R4, 0(R6)  
    ADD R3, R3, R4  
    ADD R6, R2, R7  
    STB 0(R6), R3  
    BNZ R7, bucle  
    LDB R0, 1(R6)

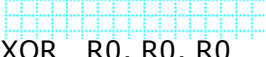



**Exercici 14.2**

En el següents apartat trobareu enunciats de problemes en llenguatge natural per a realitzar programes en SISA que els resolguin. Els programes que els resolen estan parcialment realitzats. Ompliu els forats que falten amb les instruccions adequades en llenguatge ensamblador. Fixeu-vos en els comentaris per a saber quina instrucció és l'adequada a cada lloc.






- a) Escriviu un programa que intercanviï els 8 bits de major pes amb els 8 bits de menor pes del registre R2 del banc de registres.

```
.data
.text
    MOVI R1, 0xFF      ; posem en R1 una màscara de bits 1111111111111111
    MOVHI R1, 0x00     ; posem en R1 una màscara de bits 0000000011111111
    MOVI R6, 0x08      ; posem en R6 el valor 8
    MOVI R7, 0xF8      ; posem en R7 el valor -8
     ; copiem en R5 els 8 bits de menor pes de R2
     ; desplaçem 8 posicions a la dreta el contingut de R2
     ; desplaçem 8 posicions a l'esquerra el contingut de R5
    OR R2, R2, R5      ; tornem a unir els bits
.end
```

- b) Escriviu un programa que llegeixi el valor que es troba en el dispositiu d'Entrada/Sortida número 1 i si el valor llegit és igual que 0 l'escriu a la paraula de memòria 0x05DA i si no a la posició de memòria 0x05DC.

```
.data
.text
    MOVI R6, 0xDA      ; posem en R6 l'adreça de memòria 0xFFDA
    MOVHI R6, 0x05     ; posem en R6 l'adreça de memòria 0x05DA
     ; llegim el valor del port 1 i l'emmagatzemem en el registre R2
    XOR R0, R0, R0     ; posem en R0 un zero
     ; comparem si R2 és igual a zero i deixem el resultat a R5
    BZ R5, 1           ; saltem en funció del valor de R5
    ADDI R6, R6, 2     ; deixem a R6 la posició de memòria final
     ; emmagatzemem el contingut de R2 a la posició de memòria
     indicada
.end
```

- c) Escriviu un programa que compti el número de bits que valen 1 en un número NUM emmagatzemat a memòria. Supposeu que el número NUM està a la paraula d'adreça 0x0004h de la memòria i el resultat el mostrem per un dispositiu mapejat en el port 3 de l'espai d'entrada/sortida.

```
.data
.text
    MOVI R0, 0x04      ; posem en R0 l'adreça de memòria 0x0003
    LD R1, 0(R0)       ; posem en R1 el contingut de l'adreça de memòria 0x0003
    XOR R3, R0, R0     ; fem servir R3 per comptar el número d'uns. L'inicialitzem a 0
     ; posem en R4 una màscara de bits 0000000000000001
    MOVI R5, 16        ; fem servir R5 com a comptador d'iteracions pendents
    MOVI R6, 0xFF      ; fem servir R6 per salvar el valor -1 per a els desplaçaments
    Bucle:  ; obtenim el bit de menor pes de R1 i el deixem en R2
    ADD R3, R3, R2     ; incrementem el comptador d'uns (R3) si el valor de R2 val 1
    SHL R1, R1, R6     ; desplaçem el contingut de R1 un bit cap la dreta
     ; disminuïm el comptador d'iteracions pendents (R5)
     ; si encara queden iteracions tornem a l'inici del bucle
     ; mostrem el resultat pel port 3
.end
```

**Exercici 14.3**

Donat el següent programa en llenguatge ensamblador del SISA:

```
.data
...
v:    .word 1, 24, 3, 24003, -56, 1201, -15056, 111, -4567, -1
w:    .word 100, 100, 100
      .space 6, 0xFF
      .word 0x0001, 0x0001, 0x0001, 0x0001
r:    .space 20

      N=20

.text
MOVHI R0, N           ; inicialitzem el comptador de posicions
_____ R1, _____ ; salvem a R1 l'adreça de v (v[0])
_____ R1, _____
_____                ; salvem a R2 l'adreça de w (w[0])
_____
_____                ; salvem a R3 l'adreça de r (r[0])
_____
ADDHI R0, R0, -2
_____                ; salvem a R4 l'adreça de w[N-1-i]
LD     R5, 0(R1)       ; carreguem el valor de v[i] a R5
LD     R6, 0(R4)       ; carreguem el valor de w[N-1-i] a R6
ADD    R7, R5, R6      ; sumem els dos valors
ST     0(R3), R7       ; emmagatzemem la suma a r[i]
ADDHI R1, R1, 2        ; salvem a R1 l'adreça de v[i+1]
ADDHI R3, R3, 2        ; salvem a R3 l'adreça de r[i+1]
BNZ    R0, -9         ; tornem a iterar si no hem acabat
...                  ; continua el programa...

.end
```

omple les instruccions del llenguatge SISA que falten per a que el programa salvi el resultat de la suma dels elements  $v[i]$  i  $w[N-1-i]$  a l'element  $r[i]$  del vector  $r$ , amb  $i$  des de 0 fins a  $N-1$ . Cada element es una paraula de 16 bits. És a dir, que es comporti com el següent codi d'alt nivell:

```
for (i=0; i<N; i++)
{
    r[i] = v[i] + w[N-1-i]
}
```

**Exercici 14.4**

Per sincronitzar-se amb qualsevol dispositiu extern, el processador SISC Von Neumann utilitza el protocol de comunicació asíncrona *Four-cycle Handshaking* a través dels ports de lectura i escriptura del mòdul d'entrada/sortida INPUT/OUTPUT. D'aquesta manera, llegir una dada provinent del teclat, per exemple, implica més d'una instrucció ja que primer cal testear quin és l'estat del teclat (al port que habitualment anomenem KEY\_STATUS) i només quan aquest ens indica que una nova dada està disponible el processador pot procedir a llegir-la (al port KEY\_DATA).

Amb l'objectiu de reduir el nombre d'instruccions dels programes, hem afegit al joc d'instruccions del llenguatge màquina SISA la següent instrucció:

**Semàntica:** **if** (INPUT[N] == 0) **then** PC  $\leftarrow$  PC  
**else** PC  $\leftarrow$  PC + 2

**Assemblador:**     INTEST INPUT[N];

**Descripció:** La instrucció INTEST recull el valor del port d'entrada INPUT[N] i incrementa el PC en dos només en el cas que el contingut del port d'entrada llegit no sigui zero. En altres paraules, la instrucció es queda llegint el port d'entrada indicat indefinidament, fins que el seu valor deixa de ser zero.

Sense utilitzar la nova instrucció INTEST, un programa senzill que reculli una dada del teclat, la incrementi en una unitat i la imprimeixi a través de la impressora es podria escriure de la següent manera:

bucle0:	IN R0, KEY-STATUS	o també	IN R0, 0
	BZ R0, bucle0	o també	BZ R0, -2
	IN R0, KEY-DATA	o també	IN R0, 1
	ADDI R0, R0, 1		
bucle1:	IN R1, PRINT_STATUS	o també	IN R1, 2
	BZ R1, bucle1	o també	BZ R1, -2
	OUT PRINT_DATA, R0	o també	OUT 0, R0

Omple el següent llistat d'instruccions, amb les instruccions que calen per a que aquest nou llistat d'instruccions sigui equivalent utilitzant la nova instrucció INTEST sempre que calgui esperar que el contingut d'un port sigui diferent de zero.

Llenguatge SISA		
ADDI	R0, R0, 1	

**NOTA 1:** La instrucció INTEST **no** guarda el valor que arriba del port d'entrada a cap registre del banc de registres.

**NOTA 2:** El teclat es comunica a través de dos ports d'entrada KEY\_STATUS i KEY\_DATA. El primer s'utilitzarà com a port d'estat per saber si el teclat ja té una nova dada disponible i el segon s'utilitzarà per rebre la dada. La impressora es comunica amb el processador a través del port d'entrada PRINT\_STATUS, que indica quan la impressora està disponible per rebre un nou valor, i un port de sortida PRINT\_DATA, per on li podem enviar les dades. Els ports KEY\_STATUS, KEY\_DATA i PRINT\_STATUS es corresponen als ports d'entrada 0, 1 i 2 respectivament. El port PRINT\_DATA es correspon al port de sortida 0.

## ET14 (Llenguatge màquina i assembler SISA)

### Exercicis per avaluar objectius de nivell A

(Recordeu que l'objectiu amb l'asterisc cal fer-lo a casa i portar-lo resolt a classe)

#### (\*) Exercici 14.5

Donada la següent especificació parcial d'un programa en assembler SISA:

```
.data
    SiM: .space 2
    Ca2: .space 2
.text
    INSTRUCCIONS
.end
```

Escriu un llistat d'instruccions que agafi el nombre enter emmagatzemat a la posició *SiM* de la memòria, representat en signe i magnitud, i l'emmagatzemi a la posició *Ca2* de la memòria, representat en *Ca2*.

#### Exercici 14.6

Suposeu que hem afegit al joc d'instruccions del llenguatge màquina SISA una nova instrucció anomenada ANDBR, la sintaxi de la qual és la següent:

ANDBR Rd, Ra, Rb, Rc ;Rd=(Ra AND (0x0001<<Rc)) AND (Rb AND (0x0001<<Rc))

On Rd, Ra, Rb i Rc són registres del banc de registres (fixa't que caldria ampliar el banc de registres per a que tingués tres busos de lectura A, B i C).

Sense tenir en compte els canvis que això suposaria a la UPG i la UCG, escriu una seqüència d'instruccions del llenguatge assembler SISA ampliat amb la nova operació ANDBR que, donat el contingut dels registres R0 i R1, emmagatzemi a R2 el nombre de posicions que valen 1 alhora a tots dos registres. És a dir, si R0=0001 0101 0000 1110 i R1=1101 0101 0000 0000, el contingut del registre R2 després d'executar aquesta seqüència d'instruccions serà 0x0003, ja que les posicions 8, 10 i 12 valen 1 a tots dos registres.

**NOTA 1:** no pots utilitzar l'antiga instrucció AND.

**NOTA 2:** pots modificar el contingut de qualsevol dels registres del banc de registres sempre que utilitzis correctament el valor inicial dels registres R0 i R1 i que el registre R2 contingui el resultat correcte de l'operació al final de l'execució del programa.

#### Exercici 14.7

Suposeu que hem ampliat el joc d'instruccions del llenguatge màquina SISA amb dues noves instruccions anomenades BBZ i BBNZ que ens permetin saltar si un bit concret d'un cert registre del banc de registres val 0 o 1, respectivament. Definim aquestes dues instruccions de la següent manera:

**Assembler:** BBZ Ra, K, N;

**Semàntica:** if (Ra<K> == 0) then PC ← PC + N  
else PC ← PC + 2

**Descripció:** La instrucció BBZ (Branch if Bit is Zero) salta tantes instruccions com indiqui N, interpretada com a un nombre enter en *Ca2*, si el bit de la posició K del registre Ra val 0. Altrament, la instrucció no provocarà cap ruptura de seqüència i s'executarà la següent instrucció del programa.

**Assemblador:** BBNZ Ra, K, N;

**Semàntica:** if (Ra<K> == 1) **then** PC ← PC + N  
                                           **else** PC ← PC + 1

**Descripció:** La instrucció BBNZ (Branch if Bit is Not Zero) salta tantes instruccions com indiqui N, interpretada com a nombre enter en Ca2, si el bit de la posició K del registre Ra val 1. Altrament, la instrucció no provocarà cap ruptura de seqüència i s'executarà la següent instrucció del programa. En tots dos casos, el rang de valors possibles per K és de 0 a 15, ambdós inclosos. El rang de valors possibles per N és de -32 a 31, ambdós inclosos.

Volem realitzar una multiplicació entre dos nombres naturals X i Y que ens arriben a través del teclat. X s'emmagatzema inicialment al registre R6 i Y al registre R7. L'algoritme de multiplicació es descriu a través del següent codi en llenguatge d'alt nivell:

```

1:    R5 = 0;
2:    do {
3:        if (R7<0> == 1)
4:            R5 = R5 + R6;
5:        if (R6<15> == 1)
6:            break;
7:        R6 = R6 * 2;
8:        R7 = R7 / 2;
9:    } while R7 ≠ 0

```

NOTA: la instrucció **break** provoca la finalització immediata del bucle i el salt a la primera instrucció fora d'aquest.

Com podeu observar, la multiplicació pot acabar per dos motius:

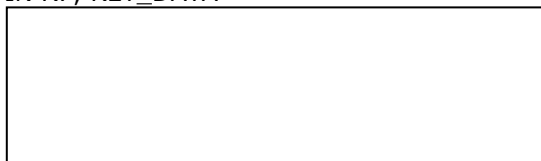
- (i) R7 val zero i per tant ja no té sentit seguir multiplicant, o
- (ii) el bit de més pes de R6 (R6<15>) val 1 i per tant el proper desplaçament de R6 produirà un sobreiximent i perdrem informació necessària d'X per continuar multiplicant, fet que fa que deixi de tenir sentit seguir amb la multiplicació.

Donat el següent fragment de codi assemblador, on ja s'ha programat la recollida dels valors que arriben del teclat i el retorn del resultat per la impressora, i sense tenir en compte els canvis que afegir aquesta nova instrucció suposaria a la UPG i la UCG, es demana que indiqueu, al full de respostes, les instruccions en assemblador que caldria posar al requadre buit que us marquem més avall necessàries per implementar aquest algoritme de multiplicació, utilitzant les noves operacions BBNZ i BBZ. Heu de resoldre les operacions expressades a les línies 3 i 5 del pseudocodi que us proporcionem amb una única instrucció:

```

begin:    IN R0, KEY_STATUS
          BZ R0, -2
          IN R6, KEY_DATA
          IN R0, KEY_STATUS
          BZ R0, -2
          IN R7, KEY_DATA

```



```

fiblucle: IN R0, PRINT_STATUS
          BZ R0, -2
          OUT PRINT_STATUS, R5
          BNZ R0, begin

```

### Exercici 14.8

A la implementació actual del computador SISC Von Neumann no hi ha cap instrucció que ens permeti saber directament si el resultat d'una suma és representable o no en 16 bits, ni per a naturals ni per a enters. Supposeu que hem afegit al joc d'instruccions del llenguatge màquina SISA una nova instrucció anomenada ADDOVFU, que es defineix de la següent manera:

**Semàntica:** **if**  $(Ra_u + Rb_u \leq 2^{16}-1)$  **then**  $Rd \leftarrow 0$   
**else**  $Rd \leftarrow 1$

**Assemblador:** ADDOVFU Rd, Ra, Rb;

**Descripció:** La instrucció ADDOVFU realitza la suma entre els dos valors continguts als registres Ra i Rb i retorna un 1 al registre Rd si el resultat de la suma no és representable en 16 bits, o un 0 si ho és. Tant el contingut dels registres font com el resultat de la suma s'interpreten com nombres naturals codificats en binari amb 16 bits. Tingues en compte que aquesta instrucció en cap cas retorna el resultat de l'operació aritmètica.

Volem realitzar la suma entre dos nombres naturals codificats en binari: X de 32 bits i Y de 16 bits. La dada X està ja emmagatzemada al banc de registres. Els 16 bits de menys pes de X són al registre R0, i els 16 de més pes són al registre R1. La dada Y ens arribarà a través del teclat, pel port KEY-DATA.

Sense tenir en compte els canvis que això suposaria a la UPG i la UCG, escriu un programa en llenguatge assemblador que agafi la dada Y del teclat, la sumi amb X i imprimeixi el resultat a través de la impressora. Per retornar el resultat de la suma caldrà imprimir primer els 16 bits de menor pes i a continuació els 16 bits de major pes. El programa ha d'usar la nova instrucció ADDOVFU per realitzar la suma de X i Y.

### Exercici 14.9

Donat la següent definició de dades en llenguatge ensamblador SISA:

```
.data  
LLISTA: .space 200  
.text
```

Escriu un llistat d'instruccions que llegeixin 100 nombres enters de 16 bits del teclat i els guardi de forma ordenada (de major a menor) a partir de l'adreça simbòlica LLISTA, fent servir el següent procediment:

- 1-Lectura dada teclat.
- 2-Busca seqüencialment les dades entrades anteriorment i ja ordenades fins trobar la posició on s'ha de posar la nova dada.
- 3-Dewsploçar una posició endavant els nombres més petits que la dada llegida.
- 4-Escriure la nova dada al seu lloc.
- 4-Repetir el procés per els 100 nombres.



## Solucions ET14 (Llenguatge Màquina i Assemblador SISA)

### ET14

#### Exercici 14.1.

- a) R0=0x0A4A
- b) R0=0x0003
- c) R0=0x0000
- d) R0=0x0066

#### Exercici 14.2.

a)

.data  
.text

```

MOVI R1, 0xFF ; posem en R1 una màscara de bits 1111111111111111
MOVHI R1, 0x00 ; posem en R1 una màscara de bits 0000000011111111
MOVI R6, 0x08 ; posem en R6 el valor 8
MOVI R7, 0xF8 ; posem en R7 el valor -8
AND R5, R2, R1 ; copiem en R5 els 8 bits de menor pes de R2
SHL R2, R2, R7 ; desplaçem 8 posicions a la dreta el contingut de R2
SHL R5, R5, R6 ; desplaçem 8 posicions a l'esquerra el contingut de R5
OR R2, R2, R5 ; tornem a unir els bits

```

.end

b)

.data  
.text

```

MOVI R6, 0xDA ; posem en R6 l'adreça de memòria 0xFFDA
MOVHI R6, 0x05 ; posem en R6 l'adreça de memòria 0x05DA
IN R2, 1 ; llegim el valor del port 1 i l'emmagatzemem en el registre R2
XOR R0, R0, R0 ; posem en R0 un zero
CMPEQ R5, R2, R0 ; comparem si R2 és igual a zero i deixem el resultat a R5
BZ R5, 1 ; saltem en funció del valor de R5
ADDI R6, R6, 2 ; deixem a R6 la posició de memòria final
ST 0(R6), R2 ; emmagatzemem el contingut de R2 a la posició de memòria
indicada

```

.end

c)

.data  
.text

```

MOVI R0, 0x04 ; posem en R0 l'adreça de memòria 0x0003
LD R1, 0(R0) ; posem en R1 el contingut de l'adreça de memòria 0x0003
XOR R3, R0, R0 ; fem servir R3 per comptar el número d'uns. L'inicialitzem a 0
MOVI R4, 0x01 ; posem en R4 una màscara de bits 0000000000000001
MOVI R5, 16 ; fem servir R5 com a comptador d'iteracions pendents
MOVI R6, 0xFF ; fem servir R6 per emmagatzemar el valor -1 per a els
desplaçaments

```

```

Bucle: AND R2, R1, R4 ; obtenim el bit de menor pes de R1 i el deixem en R2
      ADD R3, R3, R2 ; incrementem el comptador d'uns (R3) si el valor de R2 val 1
      SHL R1, R1, R6 ; desplaçem el contingut de R1 un bit cap la dreta
      ADDI R5, R5, -1 ; disminuïm el comptador d'iteracions pendents (R5)
      BNZ R5, Bucla ; si encara queden iteracions tornem a l'inici del bucle
      OUT 3, R3 ; mostrem el resultat pel port 3

```

.end

### Exercici 14.3.

MOVI R0, N	; inicialitzem el comptador de posicions
<u>MOVI</u> R1, <u>lo(v)</u>	; salvem a R1 l'adreça de v (v[0])
<u>MOVHI</u> R1, <u>hi(v)</u>	
MOVI R2, lo(w)	; salvem a R2 l'adreça de w (w[0])
MOVHI R2, hi(w)	
MOVI R3, lo(r)	; salvem a R3 l'adreça de r (r[0])
MOVHI R3, hi(r)	
ADDI R0, R0, -2	
<b>ADD</b> R4, R2, R0	; salvem a R4 l'adreça de w[N-1-i]
LD R5, 0(R1)	; carreguem el valor de v[i] a R5
LD R6, 0(R4)	; carreguem el valor de w[N-1-i] a R6
ADD R7, R5, R6	; sumem els dos valors
ST 0(R3), R7	; emmagatzemem la suma a r[i]
ADDI R1, R1, 2	; salvem a R1 l'adreça de v[i+1]
ADDI R3, R3, 2	; salvem a R3 l'adreça de r[i+1]
BNZ R0, -9	; tornem a iterar si no hem acabat
...	; continua el programa...

### Exercici 14.4.

<b>Llenguatge SISA-I</b>
INTEST KEY-STATUS
IN R0, KEY-DATA
ADDI R0, R0, 1
INTEST PRINT-STATUS
OUT PRINT-DATA, R0

o també

<b>Llenguatge SISA-I</b>
INTEST 0
IN R0, 1
ADDI R0, R0, 1
INTEST 2
OUT 0, R0