COGNOMS:

NOM:

EXAMEN PARCIAL D'EC1
Dijous, 27 d'abril de 2006

L'examen consta de 5 preguntes. S'ha de contestar als mateixos fulls de l'enunciat, dins dels requadres, excepte la pregunta 3, que es respon en full a part. No oblideu posar el vostre nom i cognoms a tots els fulls. La durada de l'examen és de **120 minuts**. Les notes sortiran el dia 9 de Maig. El procediment per a la revisió es publicarà al Racó junt amb les notes del parcial.

Pregunta 1. (4 punts)

Donada la següent declaració de dades:

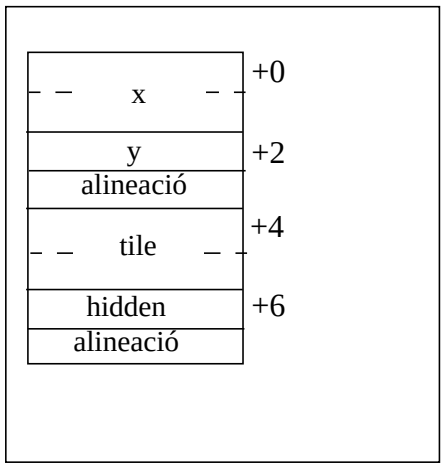
```
#define NUM_TILES 255
#define TILE_SIZE 7
#define MAX_SPRITES 8

struct sprite {
     int x;
     char y;
     int tile;
     unsigned char hidden;
};

char tileTable[NUM_TILES][TILE_SIZE];
struct sprite spriteTable[MAX_SPRITES];

int nextSprite = 4;
unsigned char newData[MAX_SPRITES] = { 1, 0, 1, 1, 0, 1, 0, 0 };
```

a) Dibuixa com s'emmagatzema una tupla de tipus struct sprite a memòria. Quants bytes ocupa, incloent-hi els bytes d'alineació?



Un tupla struct sprite ocupa [8] bytes

b) Tradueix la secció .data (declaració de dades) a assemblador. Pots considerar que les constants ja estan declarades. Quants bytes ocupen totes les variables definides incloent-hi els bytes d'alineació?

```
NUM_TILES = 255
TILE_SIZE = 7
MAX_SPRITES = 8

.data
     tileTable:       .fill NUM_TILES * TILE_SIZE, 1, 0
                      .balign 2
     spriteTable:     .fill MAX_SPRITES * 8, 1, 0
     nextSprite:      .word 4
     newData:         .byte 1, 0, 1, 1, 0, 1, 0, 0
```

Les variables ocupen [1860] bytes

c) Tradueix la següent sentència de codi C-- a assemblador.

```
spriteTable[nextSprite].y = 'Q';
```

```
$MOVEI   R0, nextSprite
LD       R0, 0(R0)
$MOVEI   R1, spriteTable
MOVI     R2, 8
MUL      R0, R0, R2
ADD      R0, R1, R0
MOVI     R2, 'Q'
STB      2(R0), R2
```

COGNOMS:

NOM:

d) Tradueix el següent bucle en C-- a assemblador. Fes servir la tècnica d'accés seqüencial.

```
main()
{
     register int i;
     i = 0;
     do
     {
              spriteTable[i].hidden = newData[i];
              i++;
     }
     while (i < MAX_SPRITES);
}
```

```
    .text

    main:

         ; R0 és la variable i, R1 és el punter a spriteTable
         ; R2 és el punter a newData, R3 conté MAX_SPRITES

         MOVI    R0, 0
         $MOVEI  R1, spriteTable
         $MOVEI  R2, newData
         $MOVEI  R3, MAX_SPRITES

    do:

         LDB     R4, 0(R2)
         STB     6(R1), R4
         ADDI    R0, R0, 1
         ADDI    R1, R1, 8
         ADDI    R2, R2, 1
         CMPLT   R4, R0, R3
         BNZ     R4, do

         HALT
```

e) A quin element (fila i columna) de la matriu tileTable està accedint el següent codi en assemblador?

```
$MOVEI  R1, tileTable
MOVI    R3, TILE_SIZE
MOVI    R4, 3
SHA     R2, R3, R4
$MOVEI  R3, 0xABCD
MOVI    R4, 5
AND     R3, R3, R4
ADD     R3, R3, R1
ADD     R1, R3, R2
LDB     R1, 0(R1)
```

```
R1 = @tileTable
R3 = TILE_SIZE = 7
R4 = 3
R2 = 7 << 3 = 7 * 2^3 = 7 * 8
R3 = 0xABCD = 0b1010101111001101
R4 = 5      = 0b0000000000000101
R3 = 5
R3 = @tileTable + 5
R1 = @tileTable + 7*8 + 5
   = @tileTable[8][5]
```

Aquest codi accedeix a: `tileTable[` [8] `][` [5] `]`

Pregunta 2. (0,5 punts)

La següent taula conté una llista de números binaris que representen nombres reals en coma flotant, en els formats IEEE 754 de 32 bits o SISA-F de 16 bits. Marca amb una **X** la casella corresponent al tipus de valor de cada un d'ells, d'acord amb la següent notació:

NRM = normalitzat DNRM = denormalitzat
0 = zero INF = infinit
NAN = valor "Not a Number" (nombres no representables en coma flotant)

signe	exponent	mantissa	NRM	DNRM	0	INF	NAN
0	00 0000	0 1111 1111	X				
1	00 0000	0 0000 0000			X		
0	10 0001	0 0000 0000	X				
1	1111 1111	000 0000 0000 0000 0000 0000				X	
0	0010 0100	110 0010 0000 1110 1110 1011	X				
0	0000 0000	000 0000 0000 0000 0000 0000			X		
1	0000 0000	100 0000 1000 0001 0000 0000		X			
0	1111 1111	101 0001 0001 0000 1001 0100					X

COGNOMS:

NOM:

Pregunta 3. (3,5 punts)

Donat el següent codi en llenguatge d'alt nivell:

```
int x;
char exam2(int w, char *c, int *z);

int exam1(int *par1, char par2[])
{
     register int i;
     i = *par1;

     if ((par2[i] != 0) && (i > 2))
             par2[i] += '0';
     else
             par2[i] = exam2(i, &par2[i], &x);

     return par2[i];
}
```

Tradueix al llenguatge assemblador del SISA-F la subrutina exam1. Posa comentaris al codi. Respon a aquesta pregunta en un full a part.

```
exam1:
     ; R1 és par1, R2 és par2, R3 és i, R4 és @par2[i], R0 és par2[i]

     LD R3, 0(R1)                ; i = *par1
     ADD R4, R2, R3              ; R4 = @par2[i]
     LDB R0, 0(R4)               ; R0 = par2[i]
     BZ R0, else                 ; (par2[i]!=0) ?
     MOVI R5, 2
     $CMPGT R5, R3, R5           ; (i>2) ?
     BZ R5, else
     MOVI R5, '0'
     ADD R0, R0, R5              ; R0 = par2[i] + '0'
     BNZ R5, fin_else
else:
     $PUSH R6, R4                ; salvem R4 i R6 que es necessiten mes tard
     ADDI R1, R3, 0              ; passem 1er parametre
     ADDI R2, R4, 0              ; passem 2on parametre
     $MOVEI R3, x                ; passem 3er parametre
     $CALL R6, exam2             ; R0 = exam2(i, &par2[i], &x)
     $POP R4, R6                 ; restaurem R4 i R6
fin_else:
     STB 0(R4), R0               ; par2[i] = R0
     JMP R6
```

Pregunta 4. (1 punt)

Donat el següent contingut inicial de la memòria representada en hexadecimal a partir de l'adreça 0x100:

```
@       Contingut (en hexadecimal)
0x100: 02 0A 06 01 09 A0 08 01 FE 01 00 00 FE FF 06 01
```

Determina el valor de R1 (en hexadecimal) després d'executar el codi de cada apartat. Per a cada apartat considera el mateix contingut inicial de la memòria.

a)

```
$MOVEI R1, 0x10E
LD     R1, 0(R1)
LDB    R1, -1(R1)
```

R1 = 0xFFA0

b)

```
$MOVEI R1, 0x106
LD     R0, 0(R1)
LD     R2, 4(R0)
ADD    R1, R1, R2
```

R1 = 0x0104

Pregunta 5. (1 punt)

Suposem que tenim una funció booleana `func` que examina el contingut de R1 i retorna una determinada condició en R0 com a resultat. Inicialment el registre R1 ja té un valor determinat. Suposem que denotem el valor inicial de cada bit de R1 amb una lletra de la següent manera (per ex., `n` és el valor del bit 2):

R1 = `'abcd efgh ijkl mnop'`

Donada la següent traducció a SISA-F de la funció `func`, indica el contingut final dels registres R1 i R2, usant la mateixa notació emprada per al valor inicial de R1 (així per exemple, si multipliquéssim R1 per 2 fent `ADD R1,R1,R1`, el resultat seria R1='bcde fghi jklm nop0'). Indica també, en poques paraules, quin és el significat del resultat que retorna la funció `func` en el registre R0 (que pot valer 1 o 0, és a dir, cert o fals).

```
func:
     MOVI    R2, 0
     MOVI    R3, 0              ; comptador del bucle
     MOVI    R4, 1              ; constant per als desplaçaments
buc:
     MOVI    R5, 8              ; límit del bucle
     $CMPLT  R5, R3, R5
     BZ      R5, fibuc
     AND     R5, R1, R4
     SHL     R2, R2, R4
     OR      R2, R2, R5
     MOVI    R5, -1
     SHL     R1, R1, R5
     ADDI    R3, R3, 1
     BNZ     R5, buc
fibuc:
     $CMPEQ  R0, R1, R2
     JMP     R6
```

`R1 = 0000 0000 abcd efgh`

`R2 = 0000 0000 ponm lkji`

La funció func retorna cert si...
el valor que es passa per R1 es capicua en binari