Examen Final EC

Estructura de Computadores (Universitat Oberta de Catalunya)

Examen *2020/21-1*

Asignatura	Código	Fecha	Hora inicio
Estructura de computadores	*75.573*	*20/1/2021*	*15:30*

Ficha técnica del examen

- *Comprueba que el código y el nombre de la asignatura corresponden a la asignatura de la que te has matriculado.*
- *Tiempo total:* **2 horas** *Valor de cada pregunta:* **Se indica en el enunciado**
- *¿Se puede consultar algún material durante el examen? ¿Qué materiales están permitidos?* **NINGUNO**
- *¿Puede utilizarse calculadora? ¿De qué tipo?*
- *Si hay preguntas tipo test, ¿descuentan las respuestas erróneas? ¿Cuánto?*
- *Indicaciones específicas para la realización de este examen:*

Asignatura	Código	Fecha	Hora inicio
Estructura de computadores	*75.573*	*20/1/2021*	*15:30*

Enunciados

Enunciado: El enunciado del examen estará en formato PDF.
En el ordenador desde donde se realizará el examen debéis tener instalado algún software para poder leer documentos en formato PDF. Por ejemplo, se puede utilizar el software gratuito Adobe Acrobat Reader DC, pero podéis utilizar cualquier otro software.

Identificación del estudiante: No es necesario identificarse, de esta forma se garantiza que el examen será tratado de forma anónima.

Respuestas:
Se deberá identificar cada respuesta dentro del examen. Es ***obligatorio indicar el número de pregunta y el apartado***, opcionalmente también se puede añadir todo o parte del enunciado si esto os ayuda en la resolución de la pregunta.
Si no se identifica correctamente a qué pregunta hace referencia la respuesta no se evaluará.

En caso de ser necesario aplicar un procedimiento para resolver alguna pregunta, mostrad claramente y argumentad el procedimiento aplicado, no solo el resultado. En caso de duda, si no se pueden resolver por los mecanismos establecidos o por falta de tiempo, haced los supuestos que consideréis oportunos y argumentadlos.

Elaboración documento a entregar:
Utilizad cualquier editor de texto para crear el documento con las respuestas, siempre que después os permita exportar el documento a formato PDF para hacer la entrega.

Entrega: Es obligatorio entregar las respuestas del examen en un único documento ***en formato PDF***.
No se aceptarán otros formatos.
Es responsabilidad del estudiante que la información que contenga el documento PDF que se entregue refleje correctamente las respuestas dadas en el examen. Recomendamos que abráis el fichero PDF generado y reviséis atentamente las respuestas para evitar que se haya podido perder, cambiar o modificar alguna información al generar el documento en formato PDF.
La entrega se puede hacer tantas veces como se quiera, se corregirá la última entrega que se haga dentro del horario especificado para realizar el examen.

COMPROMISO DE AUTORESPONSABILIDAD: este examen se tiene que resolver de forma individual bajo vuestra responsabilidad y siguiendo las indicaciones de la ficha técnica (sin utilizar ningún material, ni calculadora).
En caso de que no sea así, el examen se evaluará con un cero. Por otro lado, y siempre a criterio de los Estudios, el incumplimiento de este compromiso puede suponer la apertura de un expediente disciplinario con posibles sanciones.

Asignatura	Código	Fecha	Hora inicio
Estructura de computadores	*75.573*	*20/1/2021*	*15:30*

No se puede utilizar calculadora. Hay que saber interpretar un valor en binario, decimal o hexadecimal para realizar la operación que se pida. Y el resultado se tiene que expresar en el formato correspondiente.

Valoración de las preguntas del examen

Pregunta 1 (20%)

Pregunta sobre la práctica.
Hay que completar las instrucciones marcadas o añadir el código ensamblador que se pide.
Los puntos suspensivos indican que hay más código, pero no se tiene que completar.
NOTA: Si el código propuesto en cada pregunta no se corresponde con la forma en que vosotros plantearíais la respuesta, podéis reescribir el código o parte del código según vuestro planteamiento.

1.1 : 15%
1.2 : 5%

Pregunta 2 (40%)

2.1 : 10%
2.2 : 15%
2.3 : 15%

Pregunta 3 (40%)

3.1: 20%
- **3.1.1 : 10%**
- **3.1.2 : 10%**

3.2: 20%
- **3.2.1 : 10%**
- **3.2.2 : 5%**
- **3.2.3 : 5%**

Asignatura	Código	Fecha	Hora inicio
Estructura de computadores	*75.573*	*20/1/2021*	*15:30*

Pregunta 1

1.1 Práctica – 1a parte

Modificad la subrutina moveTileP1 para que compruebe que la posición indicada por la variable spacePos de la matriz Tiles tienen un espacio, si hay un espacio mover la ficha en la dirección indicada (funcionamiento normal), si no hay espacio poner la variable state a '5', llamar a la subrutina spacePosScreenP1 y salir. (No se tiene que escribir el código de toda la subrutina, sólo hay que modificar el código para hacer lo que pide el enunciado).

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; Mover la ficha en la dirección indicada por el carácter (charac),
; ('i':arriba, 'k':abajo, 'j':izquierda o 'l':derecha) en la posición
; donde hay el espacio dentro de la matriz indicada por la variable
; (spacePos), controlando los casos donde no se puede hacer el movimiento.
; fila    (i = spacePos / DimMatrix)
; columna (j = spacePos % DimMatrix)
; Si la casilla donde hay el espacio está en los extremos de la matriz
; no se podrá hacer el movimiento desde ese lado.
; Si se puede hacer el movimiento:
; Mover la ficha a la posición donde está el espacio de la matriz (tiles)
; y poner el espacio en la posición donde estaba la ficha movida.
; Para recorrer la matriz en ensamblador el índice va de 0 (posición [0][0])
; a 8 (posición [2][2]) con incrementos de 1 porque los datos son de
; tipo char(BYTE) 1 byte.
;
; No se tiene que mostrar la matriz con los cambios, se hace en UpdateBoardP1().
;
; Variables globales utilizadas:
; (tiles)      : Matriz donde guardamos la fichas del juego
; (charac)     : Carácter a escribir en pantalla.
; (spacePos)   : Posición del espacio en la matriz (tiles).
; (newSpacePos): Nueva posición del espacio en la matriz (tiles).
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
moveTileP1:
   push rbp
   mov  rbp, rsp
   ...
   mov r8d, DWORD[spacePos]   ;spacePos
   mov eax, r8d
   mov edx, 0
   mov ebx, DimMatrix
   div ebx
   mov r10d, eax              ;int i = spacePos / DimMatrix;
   mov r11d, edx              ;int j = spacePos % DimMatrix;
   mov r9d, r8d               ;newspacePos = spacePos;
   mov dil, BYTE[charac]      ;switch(charac)
   { moveTileP1_Switchi:
   cmp dil, 'i'                    ;case 'i':
   jne moveTileP1_Switchk
      cmp r10d, DimMatrix-1        ;if (i < (DimMatrix-1)) {
      jge moveTileP1_SwitchEnd
         add r9d, DimMatrix        ;newspacePos = spacePos+DimMatrix;
         mov dl, BYTE[tiles+r9d]   ;tiles[i+1][j];
```

```
mov BYTE[tiles+r8d], dl   ;tiles[i][j]= tiles[i+1][j];
mov BYTE[tiles+r9d], ' '  ;tiles[i+1][j] = ' ';
jmp moveTileP1_SwitchEnd  ;break
```

```
moveTileP1_Switchk:
cmp dil, 'k'                    ;case 'k':
jne moveTileP1_Switchj
   cmp r10d, 0                  ;if (i > 0) {
   jle moveTileP1_SwitchEnd
      sub r9d, DimMatrix        ;newspacePos = spacePos-DimMatrix;
      mov dl, BYTE[tiles+r9d]   ;tiles[i-1][j];
      mov BYTE[tiles+r8d], dl   ;tiles[i][j]= tiles[i-1][j];
      mov BYTE[tiles+r9d], ' '  ;tiles[i-1][j] = ' ';
      jmp moveTileP1_SwitchEnd  ;break
moveTileP1_Switchj:
cmp dil, 'j'                    ;case 'j':
jne moveTileP1_Switchl
   cmp r11d, DimMatrix-1        ;if (j < (DimMatrix-1)) {
   jge moveTileP1_SwitchEnd
      inc r9d                   ;newspacePos = spacePos+1;
      mov dl, BYTE[tiles+r9d]   ;tiles[i][j+1];
      mov BYTE[tiles+r8d], dl   ;tiles[i][j]= tiles[i][j+1];
      mov BYTE[tiles+r9d], ' '  ;tiles[i][j+1] = ' ';
      jmp moveTileP1_SwitchEnd  ;break
moveTileP1_Switchl:
cmp dil, 'l'                    ;case 'l':
jne moveTileP1_SwitchEnd
   cmp r11d, 0                  ;if (j > 0) {
   jle moveTileP1_SwitchEnd
      dec r9d                   ;newspacePos = spacePos-1;
      mov dl, BYTE[tiles+r9d]   ;tiles[i][j-1];
      mov BYTE[tiles+r8d], dl   ;tiles[i][j]= tiles[i][j-1];
      mov BYTE[tiles+r9d], ' '  ;tiles[i][j-1] = ' ';
moveTileP1_SwitchEnd:
mov DWORD[newSpacePos], r9d

moveTileP1_End:
...
mov rsp, rbp
pop rbp
ret
```

1.2 Práctica – 2a parte

Haced los cambios necesarios al código ensamblador de esta subrutina considerando que las variables moves y sorted se han declarado de tipo char (1 byte). No se pueden añadir instrucciones, sólo hay que modificar las instrucciones que sea necesario.

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; Verificar el estado del juego.
; Si se han terminado los movimientos (moves == 0) poner (status='3').
; Si no se ha podido hacer el movimiento (spacePos == newSpacePos) poner (status='2').
; Si no estamos en ninguno de estos casos:
; Comprobar si la matriz (tiles) es igual a la matriz objetivo (tilesEnd).
; Recorrer toda la matriz (tiles) por filas de izquierda a derecha y de arriba a abajo
; comparando cada posición con la misma posición de la matriz (tilesEnd).
; Si hay una posición diferente (sorted=0) y detener la búsqueda.
```

```
; Si se ha recorrido toda la matriz y todas las posiciones son iguales
; (sorted=1), poner (status = '4') para indicar que se ha ganado y salir.
```

Asignatura	Código	Fecha	Hora inicio
Estructura de computadores	*75.573*	*20/1/2021*	*15:30*

```
; Retornar (status).
; Para recorrer la matriz en ensamblador el índice va de 0 (posición [0][0])
; a 8 (posición [2][2]) con incrementos de 1 porque los datos son de
; tipo char(BYTE) 1 byte.
;
; Variables globales utilizadas: Ninguna.
; Parámetros de entrada:
; rdi (edi) moves       : Movimientos que quedan para ordenar les fichas.
; rsi (esi) spacePos    : Posición del espacio dentro de la matriz tiles.
; rdx (edx) newSpacePos : Nueva posición del espacio dentro de la matriz tiles.
; Parámetros de salida :
; rax (al): (status): Estado del juego.
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
checkStatusP2:
   push rbp
   mov  rbp, rsp
   ...
                                ;al - char status
   mov r9d, 1                   ;int  sorted = 1;
   mov r10d, 0                  ;i=0
   mov r11d, 0                  ;j=0
   mov r12d, 0                  ;k=0;

   cmp edi, 0                   ;if (moves == 0) {
   jne checkStatusP2_ElseIf1
      mov al, '3'               ;status= '3';
      jmp checkStatusP2_End
   checkStatusP2_ElseIf1:
   cmp esi, edx                 ;} else if (spacePos == newSpacePos) {
   jne checkStatusP2_While
      mov al, '2'               ;status= '2';
      jmp checkStatusP2_End
   checkStatusP2_While:         ;} else {
      cmp r12d, SizeMatrix       ;while ((k<SizeMatrix)
      jge checkStatusP2_EndWhile
         cmp r9d, 1             ;&& (sorted==1)){
         jne checkStatusP2_EndWhile
          mov dl, BYTE[tiles+r12d]
          cmp dl, BYTE[tilesEnd+r12d]   ;if(tiles[i][j]!=tilesEnd[i][j])
          je  checkStatusP2_EndIf
               mov r9d, 0       ;sorted=0;
            checkStatusP2_EndIf:;}
            inc r12d            ;k++;
      jmp checkStatusP2_While
   checkStatusP2_EndWhile:
   cmp r9d, 1                   ;if (sorted==1)
   jne checkStatusP2_Else
      mov al, '4'               ;status = '4'
      jmp checkStatusP2_End
   checkStatusP2_Else:          ;} else {
      mov al, '1'               ;status = '1'
   checkStatusP2_End:
   ...
   mov rsp, rbp
   pop rbp
   ret
```

Asignatura	Código	Fecha	Hora inicio
Estructura de computadores	*75.573*	*20/1/2021*	*15:30*

Pregunta 2

2.1

El estado inicial del computador CISCA justo antes de empezar la ejecución de cada fragmento de código (en cada apartado) es el siguiente:

R2 = 00000010h R4 = 00000020h R8 = 00000030h	M(00000020h) = 11223344h M(00000030h) = 77665544h M(00000200h) = 8899AABCh	Z = 0, C = 0, S = 0, V = 0

Completad el estado del computador después de ejecutar cada código (indicad los valores de los registros en hexadecimal). Suponed que la dirección simbólica A vale 200h.

a)

```
MOV R2,
[R4] SAL
R2, 1 ADD
[R8], R2
```

2a)
R2 =
00002244
6688h

[R8] =
99AA
BBCCh

Z=0 , C=0 ,
S=1 , V=1

b)

```
ADD    R2,
R4     MOV
R8,    [A]
ADD    R8,
[R2]
```

R2=
77665544h

R8=
0000
0000h

Z=1 , C=1 ,
S=0 , V=0

Asignatura	Código	Fecha	Hora inicio
Estructura de computadores	*75.573*	*20/1/2021*	*15:30*

2.2

Dado el siguiente código de alto nivel:

```
do {
  C = C * A;
  B = B / A;
}
while B <= C
```

Se propone la siguiente traducción a CISCA donde hemos dejado 7 espacios para llenar.

```
      MOV     R0, [A]
DO :          [C]     ,
      MUL     R0
      DIV     [B], R0
      MOV      R2, [B]
      CMP     R2, [C]
      JLE     DO
```

Asignatura	Código	Fecha	Hora inicio
Estructura de computadores	*75.573*	*20/1/2021*	*15:30*

2.3

Dado el siguiente fragmento de código de un programa en lenguaje ensamblador de CISCA:

```
        TEST  R0,[M+R4]
        JE    BEGIN
        INC   [M]
BEGIN:  CMP   [R2], 1h
```

Traducidlo a lenguaje máquina y expresadlo en la siguiente tabla. Suponed que la primera instrucción del código se ensambla a partir de la dirección 00023C00h (que es el valor del PC antes de empezar la ejecución del fragmento de código). Suponed que la dirección simbólica M vale 00000400h. En la siguiente tabla usáis una fila para codificar cada instrucción. Si suponemos que la instrucción empieza en la dirección @, el valor Bk de cada uno de los bytes de la instrucción con direcciones @+k para k=0, 1,… se tiene que indicar en la tabla en hexadecimal en la columna correspondiente (recordad que los campos que codifican un desplazamiento en 2 bytes o un inmediato o una dirección en 4 bytes lo hacen en formato little endian, esto hay que tenerlo en cuenta escribiendo los bytes de menor peso, de dirección más pequeña, a la izquierda y los de mayor peso, dirección mayor, a la derecha). Completad también la columna @ que indica para cada fila la dirección de memoria del byte B0 de la instrucción que se codifica en esta fila de la tabla.

A continuación, os damos como ayuda las tablas de códigos:

Tabla de códigos de instrucción

B0	Instrucción
43h	JE
26h	CMP
33h	TEST
24h	INC

Tabla de modos de direccionamiento (Bk<7..4>)

Camp modo Bk<7..4>	Mode
0h	Inmediato
1h	Registro
2h	Memoria
3h	Indirecto
4h	Relativo
5h	Indexado
6h	Relativo a PC

Tabla de modos de direccionamiento (Bk<3..0>)

Camp modo Bk<3..0>	Significado
Num. registro	Si el modo tiene que especificar un registro
0	No se especifica registro.

@	Ensamblador	Bk para k=0..10										
		0	1	2	3	4	5	6	7	8	9	10
00023C00h	TEST R0,[M+R4]	33	10	54	00	04	00	00				
00023C07h	JE BEGIN	43	60	06	00							
00023C0Bh	INC [M]	24	20	00	04	00	00					
00023C11h	CMP [R2], 1h	26	32	00	01	00	00	00				

00023C18h												

Asignatura	Código	Fecha	Hora inicio
Estructura de computadores	*75.573*	*20/1/2021*	*15:30*

Pregunta 3

3.1. Memoria cache

Memoria cache de asignación directa
Tenemos un sistema de memoria en el que todos los accesos se realizan a palabra (no nos importa cuál es el tamaño de la palabra). Supondremos que el espacio de direcciones de memoria se descompone en bloques de 8 palabras. Cada bloque empieza en una dirección múltiplo de 8. Así, el bloque 0 contiene las direcciones 0, 1, 2, 3, 4, 5, 6, 7, el bloque 1 las direcciones 8, 9, 10, 11, 12, 13, 14, 15, y el bloque N las direcciones 8*N, 8*N+1, 8*N+2, 8*N+3, 8*N+4, 8*N+5, 8*N+6, 8*N+7.

Suponemos que el sistema también dispone de una memoria cache de 4 líneas (donde cada línea tiene el tamaño de un bloque, es decir, 8 palabras). Estas líneas se identifican como líneas 0, 1, 2 y 3. Cuando se hace una referencia a una dirección de memoria principal, si esta dirección no se encuentra en la memoria cache, se trae todo el bloque correspondiente desde la memoria principal a una línea de la memoria cache (así si hacemos referencia a la dirección 2 de memoria principal traeremos el bloque formado por las palabras 0, 1, 2, 3, 4, 5, 6, 7).

Suponemos que el sistema utiliza una **política de asignación directa**, de manera que cada bloque de la memoria principal sólo se puede traer a una línea determinada de la memoria cache.

La ejecución de un programa genera la siguiente lista de lecturas a memoria:

0, 18, 2, 22, 15, 23, 24, 50, 17, 3, 18, 19, 32, 4, 6, 65, 51, 20, 56, 50

3.1.1. La siguiente tabla muestra el estado inicial de la cache, que contiene las primeras 32 palabras de la memoria (organizadas en 4 bloques).

Completar la tabla para mostrar la evolución de la cache durante la ejecución del programa. Para cada acceso se debe rellenar una columna indicando si se trata de un acierto o un fallo.

Si es un acierto escribiremos E en la línea correspondiente delante de las direcciones del bloque, si es un fallo escribiremos F y se indicará el nuevo bloque que se trae a la memoria cache en la línea que le corresponda, expresando de la forma b:e (a_0 - a_7) donde b:número de bloque, e:etiqueta y (a_0 - a_7) son las direcciones del bloque, donde a_0 es la primera dirección del bloque y a_7 es la octava (última) dirección del bloque.

Línea	Estado Inicial	12		4		14		28		5	
0	0:0 (0 - 7)		0:0(0-7)	E	0:0(0-7)		0:0(0-7)		0:0(0-7)	E	0:0(0-7)
1	1:0 (8 - 15)	E	1:0 (8 - 15)		1:0 (8 - 15)	E	1:0 (8 - 15)		1:0 (8 - 15)		1:0 (8 - 15)
2	2:0 (16 - 23)		2:0 (16 - 23)		2:0 (16 - 23)		2:0 (16 - 23)		2:0 (16 - 23)		2:0 (16 - 23)
3	3:0 (24 - 31)		3:0 (24 - 31)		3:0 (24 - 31)		3:0 (24 - 31)	E	3:0 (24 - 31)		3:0 (24 - 31)

Asignatura	Código	Fecha	Hora inicio
Estructura de computadores	*75.573*	*20/1/2021*	*15:30*

Línea	20	6	44	22	7	8
0	0:0 (0 - 7)	E 0:0 (0 - 7)	0:0 (0 - 7)	0:0 (0 - 7)	E 0:0 (0 - 7)	0:0 (0 - 7)
1	1:0 (8 - 15)	1:0 (8 - 15)	F 1:1 (40 - 47)	1:1 (40 - 47)	1:1 (40 - 47)	F 1:0 (8 - 15)
2	E 2:0 (16 - 23)	2:0 (16 - 23)	2:0 (16 - 23)	E 2:0 (16 - 23)	2:0 (16 - 23)	2:0 (16 - 23)
3	3:0 (24 - 31)	3:0 (24 - 31)	3:0 (24 - 31)	3:0 (24 - 31)	3:0 (24 - 31)	3:0 (24 - 31)

Línea	30	45	55	10	43	0
0	0:0 (0 - 7)	0:0 (0 - 7)	0:0 (0 - 7)	0:0 (0 - 7)	0:0 (0 - 7)	E 0:0 (0 - 7)
1	1:0 (8 - 15)	F 1:1 (40 - 47)	1:1 (40 - 47)	F 1:0 (8 - 15)	F 1:1 (40 - 47)	1:1 (40 - 47)
2	2:0 (16 - 23)	2:0 (16 - 23)	F 2:1 (48 - 55)	2:1 (48 - 55)	2:1 (48 - 55)	2:1 (48 - 55)
3	E 3:0 (24 - 31)	3:0 (24 - 31)	3:0 (24 - 31)	3:0 (24 - 31)	3:0 (24 - 31)	3:0 (24 - 31)

Línea	56	46	1			
0	0:0 (0 - 7)	0:0 (0 - 7)	E 0:0 (0 - 7)			
1	1:1 (40 - 47)	E 1:1 (40 - 47)	1:0 (8 - 15)			
2	2:1 (48 - 55)	2:1 (48 - 55)	2:1 (48 - 55)			
3	F 3:1(56-63)	3:1(56-63)	3:1(56-63)			

3.1.2 a) ¿Cuál es la tasa de aciertos (T_a)?

T_a = 13 aciertos / 20 accesos = 0,65

3.1.2 b) Suponemos que el tiempo de acceso a la memoria cache, o tiempo de acceso en caso de acierto (t_a), es de 5 ns y el tiempo total de acceso en caso de fallo (t_f) es de 25 ns. Considerando la tasa de aciertos obtenida en la pregunta anterior, ¿cuál es el tiempo medio de acceso a memoria (t_m)?

$t_m = T_a \times t_a + (1-T_a) \times t_f = 0,65*5ns + 0,35*25ns = 3,25ns + 8,75ns = 12ns$

Asignatura	Código	Fecha	Hora inicio
Estructura de computadores	*75.573*	*20/1/2021*	*15:30*

3.2 Sistema d'E/S

E/S programada

Se quiere analizar el rendimiento de la comunicación de datos entre la memoria de un procesador y un puerto USB, utilizando E/S programada con las siguientes características:

- Velocidad de transferencia del dispositivo de E/S ($\mathbf{v}_{transf}$)= 4 MBytes/s = 4000 KBytes/s
- Tiempo de latencia media del dispositivo ($t_{latencia}$) = 0
- Direcciones de los **registros de datos** y **de estado** del controlador de E/S: 1F00h y 1F04h
- El bit del **registro de estado** que indica que el controlador del puerto de E/S está disponible es el bit 5, o el sexto bit menos significativo (cuando vale 1 indica que está disponible)
- Procesador con una frecuencia de reloj de 4 GHz; el tiempo de ciclo es de 0,25 ns. El procesador puede ejecutar una instrucción en 8 ciclos de reloj.
- Transferencia de **lectura** desde el puerto de E/S hacia la memoria
- Transferencia de N_{datos}= 800.000 datos
- La medida de un dato es m_{dato} = 4 Bytes
- Dirección inicial de memoria donde residen los datos: A0000000h

3.2.1 El siguiente código que utiliza el repertorio de instrucciones CISCA realiza la transferencia descrita antes mediante la técnica de E/S programada. Completad el código.

```
1.       MOV R3, 800000
2.       MOV R2, A0000000
3.Bucle: IN  R0, [1F00h]      ; leer 4 bytes
4.AND R0, 00100000b
5.       JE Bucle
6.       MOV R0, [R2]         ; leer 4 bytes
7.       ADD R2, 4
8.       OUT [1F04h], R0      ; escribir 4 bytes
9.       SUB  R3, 1
10.      JNE Bucle
```

3.2.2 Cuánto tiempo dura la transferencia del bloque de datos t_{transf_bloque}?

Asignatura	Código	Fecha	Hora inicio
Estructura de computadores	*75.573*	*20/1/2021*	*15:30*

3.2.3 Si quisiéramos utilizar el mismo procesador y el mismo programa, pero con un dispositivo de E/S más rápido, ¿cuál es la tasa o velocidad máxima de transferencia del nuevo dispositivo que se podría soportar sin que el dispositivo tuviera que esperar?

ttransf = mdato / vtransf = 4 Bytes / 4 Mbytes/s = 1 Bytes/106 Bytes/s=110-6s= 0,001ms

Ttansf = 0 + (800.000 0,001 ms) = 800 ms = 0,8 s

Máxima velocidad de transferencia del dispositivo

Frecuencia de reloj = 4 GHz

Tiempo de ciclo: Tciclo = 1/4*109 oscilaciones= 0,25/10-9s=0,25ns

Tiempo de cada instrucción: Tinst = 8 * tciclo = 2ns

El bucle que tenemos tiene 8 instrucciones. Por lo tanto 8 x Tinstr = 8 x 2 ns= 16 ns , mdato=4, se pueden transferir 4 bytes en cada escritura de un dato, por lo tanto se pueden transferir 4 bytes cada 16 ns.

máxima velocidad de transferencia : Vmax = 4 bytes / 16 ns = 1/4 bytes*109 /s = 250 * 106 bytes/s = 250 Mbytes/s