Solución examen junio 2020

Estructura de Computadores (Universitat Oberta de Catalunya)

Espacio grapa

Examen 2019/20-2

Asignatura	Código	Fecha	Hora inicio
Estructura de computadores	75.573	10/06/2020	18:30

Ficha técnica del examen

- Comprueba que el código y el nombre de la asignatura corresponden a la asignatura matriculada.
- Tiempo total: **2 horas** Valor de cada pregunta: **Se indica en el enunciado**
- En el caso de que los estudiantes no puedan consultar algún material durante el examen, ¿cuáles son?: **NINGUNO**
- Se puede utilitzar calculadora? **NO** De que tipo? **NINGUNO**
- En el caso de que haya preguntas tipo test: ¿descuentan las respuestas erróneas? **NO** ¿Cuánto?

- Indicaciones específicas para la realización de este examen

Examen 2019/20-2

Asignatura	Código	Fecha	Hora inicio
Estructura de computadores	75.573	10/06/2020	18:30

Enunciados

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 rescribir el código o parte del código según vuestro planteamiento.

1.1 : 10%
1.2 : 10%

Pregunta 2 (35%)

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

Pregunta 3 (35%)

3.1 : 15%
3.1.1 : 10%
3.1.2 : 5%

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

Pregunta 4 (10%)

4.1 : 5%
4.2 : 5%

Examen 2019/20-2

Asignatura	Código	Fecha	Hora inicio
Estructura de computadores	75.573	10/06/2020	18:30

Pregunta 1

1.1 Práctica – 1a Parte

Escribir un fragmento de código ensamblador de la subrutina addPairsRP1 que mira si se encuentra una pareja, dos casillas consecutivas con el mismo número, juntamos la pareja poniendo la suma de la pareja en la casilla de la derecha, un 0 en la casilla de la izquierda y acumularemos esta suma (puntos que se ganan). (No se tiene que escribir el código de toda la subrutina).

```
;;;;;
; Emparejar números iguales des la derecha de la matriz (m) y acumular los puntos en el marcador
; sumando los puntos de las parejas que se hagan.
; Recorrer la matriz por filas de derecha a izquierda y de abajo hacia arriba.
; Cuando se encuentre una pareja, dos casillas consecutivas con el mismo número, juntamos la pareja
; poniendo la suma de los números de la pareja en la casilla de la derecha y un 0 en la casilla
; de la izquierda y acumularemos esta suma (puntos que se ganan).
; Si una fila de la matriz es: [8,4,4,2] y state = 1, quedará [8,0,8,2], p=p+(4+4) y state = '2'.
; Si al final se ha juntado alguna pareja (puntos>0), pondremos la variable (state) a '2' para
; indicar que se ha movido algún número y actualizaremos la variable (score) con los puntós
; obtenidos de hacer las parejas.
; Para recorrer la matriz en ensamblador, el índice va de la posición 30 (posición [3][3]) a la
; (posición [0][0]) con decrementos de 2 porque los datos son de tipo short(WORD) 2 bytes.
; Per a acceder a una posición concreta de la matriz desde ensamblador hay que tener en cuenta que
; el índice es:(index=(fila*DimMatrix+columna)*2), multiplicamos por 2 porque los datos son de tipo
; short(WORD) 2 bytes. Los cambios se tienen que hacer sobre la misma matriz.
; No se tiene que mostrar la matriz.
; Variables globales utilizadas: m     : Matriz 4x4 donde hay los números del tablero de juego.
;                                score : Puntos acumulados hasta el momento.
;                                state : Estado del juego. (2: Se han hecho movimientos).
;;;;;
addPairsRP1:
   push rbp
   mov  rbp, rsp
   ...
   mov r10, 0                       ;p = 0
   mov rax, (SizeMatrix-1)*2        ;index [i][j]
   mov rbx, 0
   mov rcx, (SizeMatrix-1)*2        ;index [i][j-1]
   mov rdx, 0
   mov r8, DimMatrix                ;i = DimMatrix
   dec r8                           ;i = DimMatrix-1
   addPairsRP1_Rows:
   mov r9, DimMatrix                ;j = DimMatrix
   dec r9                           ;j = DimMatrix-1
      addPairsRP1_Cols:
      mov bx, WORD[m+rax]                 ;bx=m[i][j]
      cmp bx, 0                    ;if (m[i][j]!=0)
      je  addPairsRP1_EndIf                                    ; && (m[i][j]==m[i][j-1])) {
                                                               ;    m[i][j]  = m[i][j]*2;
                                                               ;    m[i][j-1]= 0;
                                                               ;    p = p + m[i][j];
                                                               ; }
      mov rcx, rax
        sub rcx, 2
        cmp bx, WORD[m+rcx]            ;m[i][j]==m[i][j-1]
        jne addPairsRP1_EndIf
           shl bx,1                    ;m[i][j]*2
           mov WORD[m+rax], bx         ;m[i][j]  = m[i][j]*2
           mov WORD[m+rcx], 0          ;m[i][j-1]= 0
           add r10w, bx                ;p = p + m[i][j]
   addPairsRP1_EndIf:

      sub rax, 2
```

Asignatura	Código	Fecha	Hora inicio
Estructura de computadores	75.573	10/06/2020	18:30

```
      dec r9                        ;j--
      cmp r9, 0                     ;j>0
      jg  addPairsRP1_Cols
   sub rax, 2
   dec r8                           ;i--
   cmp r8, 0                        ;i>=0
   jge  addPairsRP1_Rows
   cmp r10w, 0                      ;p > 0
   je  addPairsRP1_End
      mov BYTE[state], '2'         ;satate='2';
      add DWORD[score], r10d       ;score = score + p;
   addPairsRP1_End:
   ...
   mov rsp, rbp
   pop rbp
   ret
```

1.2 Práctica – 2a parte

Completar el código de la subrutina rotateMatrixRP2. (Sólo completar los espacios marcados, no se pueden añadir o modificar otras instrucciones).

```
;;;;;
; Copia la matriz, que recibimos como segundo parámetro (rsi), en la
; matriz destino, que recibimos como primer parámetro (rdi).
; La matriz origen no se tiene que modificar, los cambios se tienen que hacer en la matriz destino.
; Esto permite copiar 2 matrices después de una rotación y gestionar la opción '(u)Undo' del juego.
; Para recorrer la matriz en ensamblador el índice va de 0 (posición [0][0])
; a 30 (posición [3][3]) con incrementos de 2 porque los datos son de tipo short(WORD) 2 bytes.
; No se muestra la matriz.
; Variables globales utilizadas:
; Ninguna.
; Parámetros de entrada:
; rdi : Dirección de la matriz destino.
; rsi : Dirección de la matriz origen.
; Parámetros de salida :
; Ninguno.
;;;;;
copyMatrixP2:

;;;;;
; Rotar a la derecha la matriz recibida como parámetro (rdi), sobre la matriz (mRotated).
; La primera fila pasa a ser la cuarta columna, la segunda fila pasa
; a ser la tercera columna, la tercera fila pasa a ser la segunda
; columna y la cuarta fila pasa a ser la primera columna.
; En el enunciado se explica con más detalle cómo hacer la rotación.
; NOTA: NO es lo mismo que fer la matriz traspuesta.
; La matriz recibida como parámetro no se tiene que modificar,
; los cambios se tienen que hacer en la matriz (mRotated).
; Para recorrer la matriz en ensamblador el indice va de 0 (posición [0][0]) a 30 (posición [3][3])
; con incrementos de 2 porque los datos son de tipo short(WORD) 2 bytes.
; Para acceder a una posición concreta de la matriz desde ensamblador
; hay que tener en cuenta que el índice es:(index=(fila*DimMatrix+columna)*2),
; multiplicamos por 2 porque los datos son de tipo short(WORD) 2 bytes.
; Una vez se ha hecho la rotación, copiar la matriz (mRotated) a la
; matriz (m) llamando a la subrtuina copyMatrixP2.
; No se tiene que mostrar la matriz.
; Variables globales utilizadas:
; mToRotate : Matriz donde guardamos los números del juego que queremos rotar.
; Parámetros de entrada:
; rdi       : Dirección de la matriz que queremos rotar.
; Parámetros de salida :
; Ninguno.
;;;;;
rotateMatrixRP2:
   push rbp
   mov  rbp, rsp
   ...
```

Examen 2019/20-2

Asignatura	Código	Fecha	Hora inicio
Estructura de computadores	75.573	10/06/2020	18:30

```
mov rdx, rdi                ;short mToRotate[DimMatrix][DimMatrix]
mov r10, 0                  ;[i][j]
mov r8d, 0                  ;i=0;
rotateMatrixRP1_fori:
cmp r8d, DimMatrix          ;i<DimMatrix;
jge rotateMatrixRP1_endfori
   mov r9d, 0               ;j=0;
   rotateMatrixRP1_forj:
   _cmp_ r9d, DimMatrix          ;j<DimMatrix;
   jge rotateMatrixRP1_endforj
   mov r11, r9              ;r11 = j
   shl r11, DimMatrix/2     ;r11 = j*DimMatrix      (DimMatrix=4)
   add r11, DimMatrix       ;r11 = j*DimMatrix+(DimMatrix)
   dec r11                  ;r11 = j*DimMatrix+(DimMatrix-1)
   sub r11, r8              ;r11 = j*DimMatrix+(DimMatrix-1-i)
   _shl_ r11, 1                ;r11 = j*DimMatrix+(DimMatrix-1-i)*2

   mov bx, __WORD[rdx+r10]__          ;mToRotate[i][j];
   mov __WORD[mRotated+r11]__, bx ;mRotated[j][DimMatrix-1-i]=m[i][j]
   add __r10__, 2                     ;index+2

   inc r9                   ;j++.
   jmp  rotateMatrixRP1_forj
   rotateMatrixRP1_endforj:
   inc r8                   ;i++.
   jmp rotateMatrixRP1_fori

rotateMatrixRP1_endfori:

mov  rdi, rdx
mov   _rsi_, mRotated
call copyMatrixP2           ;copyMatrixP2_C(mToRotate, mRotated);
...
mov rsp, rbp
pop rbp
ret
```

Examen 2019/20-2

Asignatura	Código	Fecha	Hora inicio
Estructura de computadores	75.573	10/06/2020	18:30

Pregunta 2

2.1

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

R0 = 00000A10h R1 = 00000B20h R2 = 00000C30h R3 = 00000D40h R4= 00000004h	M(00000A10h) = 0000F00Fh M(00000B20h) = 0000F000h M(00000C30h) = 00000FF0h M(00000D40h) = 00000001h M(00001640h) = 00000F00h	Z = 0, C = 0, S = 0, V = 0

¿Cuál será el estado del computador después de ejecutar cada fragmento de código? (sólo modificaciones, excluyendo el PC).

a)

```
ADD R0, R2
XOR [R0], R0
```

R0= 1640
[00001640h]= 00001940h

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

b)

```
CONT: CMP R4, 0
      JE END
      SUB R4, R0
END:
```

R4= 00000004h – 00000A10h = FFFFF5F4h

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

Examen 2019/20-2

Asignatura	Código	Fecha	Hora inicio
Estructura de computadores	75.573	10/06/2020	18:30

2.2

Dado el siguiente código de alto nivel:

Si (A[j]<=A[i]*2) A[i]= A[i]-A[j];

A es un vector de 8 elementos de 4 bytes cada uno. Se propone la siguiente traducción a CISCA donde hemos dejado 6 espacios para llenar.

```
        MOV R0, [j]
        MUL R0, 4
        MOV R2, [A+R0]
        MOV R1, [i]
        MUL R1, 4
        MOV R3, [A+R1]
        SAL R3, 1
        CMP R2, R3
        JG END
        SUB [A+R1], R2
END:
```

Examen 2019/20-2

Asignatura	Código	Fecha	Hora inicio
Estructura de computadores	75.573	10/06/2020	18:30

2.3

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

```
        MOV R1, [A+R4]
PLUS:   SUB [R10], 10h
        CMP R1,R4
        JNE PLUS
```

Traducidlo a lenguaje máquina y expresadlo en la siguiente tabla. Suponed que la primera instrucción del código se ensambla a partir de la dirección 00006880h (que es el valor del PC antes de empezar la ejecución del fragmento de código). Suponed que la dirección simbólica A vale 00004000h. En la siguiente tabla usad una fila para codificar cada instrucción. Si suponemos que la instrucción comienza en la dirección @, el valor Bk de cada uno de los bytes de la instrucción con direcciones @+k para k=0, 1,… se debe 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 esa fila de la tabla.

A continuación os damos como ayuda las tablas de códigos:

Tabla de códigos de instrucción

B0	Instrucción
10h	MOV
21h	SUB
26h	CMP
42h	JNE

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

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

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

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

		Bk para k=0..10										
@	Ensamblador	0	1	2	3	4	5	6	7	8	9	10
00006880h	MOV R1, [A+R4]	10	11	54	00	40	00	00				
00006887h	SUB [R10], 10h	21	3A	00	10	00	00	00				
0000688Eh	CMP R1, R4	26	11	14								
00006891h	JNE PLUS	42	60	F2	FF							
00006895h												

Examen 2019/20-2

Asignatura	Código	Fecha	Hora inicio
Estructura de computadores	75.573	10/06/2020	18:30

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 bolque 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 utilizar 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, 1, 2, 12, 61, 62, 63, 64, 17, 18, 19, 32, 4, 6, 65, 66, 20, 56, 42, 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 i se indicará el nuevo bloque que es trae a la memoria cache en la línea que le corresponda, expresando de la forma b (a_0 - a_7) donde b:número de bloque, 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.

Linea	Estado Inicial	0	1	2	12	61
0	0:0 (0 - 7)	E 0:0 (0 - 7)	E 0:0 (0 - 7)	E 0:0 (0 - 7)	0:0 (0 - 7)	0:0 (0 - 7)
1	1:0 (8 - 15)	1:0 (8 - 15)	1:0 (8 - 15)	1:0 (8 - 15)	E 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)	3:0 (24 - 31)	F 7:1 (56 - 63)

Linea	62	63	64	17	18	19
0	0:0 (0 - 7)	0:0 (0 - 7)	F 8:2 (64 - 71)	8:2 (64 - 71)	8:2 (64 - 71)	8:2 (64 - 71)
1	1:0 (8 - 15)	1:0 (8 - 15)	1:0 (8 - 15)	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)	E 2:0 (16 - 23)	E 2:0 (16 - 23)	E 2:0 (16 - 23)
3	E 7:1 (56 - 63)	E 7:1 (56 - 63)	7:1 (56 - 63)	7:1 (56 - 63)	7:1 (56 - 63)	7:1 (56 - 63)

Examen 2019/20-2

Asignatura	**Código**	**Fecha**	**Hora inicio**
Estructura de computadores	75.573	10/06/2020	18:30

Linea	32		4		6		65		66		20	
0	F	4:1 (32 - 39)	F	0:0 (0 - 7)	E	0:0 (0 - 7)	F	8:2 (64 - 71)	E	8:2 (64 - 71)		8:2 (64 - 71)
1		1:0 (8 - 15)		1:0 (8 - 15)		1:0 (8 - 15)		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)	E	2:0 (16 - 23)
3		7:1 (56 - 63)		7:1 (56 - 63)		7:1 (56 - 63)		7:1 (56 - 63)		7:1 (56 - 63)		7:1 (56 - 63)

Linea	56		42		50							
0		8:2 (64 - 71)		8:2 (64 - 71)		8:2 (64 - 71)						
1		1:0 (8 - 15)	F	5:1 (40 - 47)		5:1 (40 - 47)						
2		2:0 (16 - 23)		2:0 (16 - 23)	F	6:1 (48 - 55)						
3	E	7:1 (56 - 63)		7:1 (56 - 63)		7:1 (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 4 ns y el tiempo total de acceso en caso de fallo (t_f) es de 20 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 * 4 ns + 0,35 * 20 ns = 2,6 ns + 7 ns = 9,6 ns

Examen 2019/20-2

Asignatura	Código	Fecha	Hora inicio
Estructura de computadores	75.573	10/06/2020	18:30

3.2 Sistema de E/S

3.2.1 E/S programada

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

- Velocidad de transferencia del dispositivo d'E/S v_{transf} = 10 MBytes/s = 10000 Kbytes/s
- Tiempo de latencia medio del dispositivo $t_{latencia}$ = 0
- Direcciones de los **registros de estado** y **datos** del controlador de E/S: 0E00h y 0E04h
- El bit del **registro de estado** que indica que el controlador del puerto de E/S está disponible es el bit 4, o el quinto bit menos significativo (cuando vale 1 indica que está disponible)
- Procesador con una frecuencia de reloj de 1 GHz, el tiempo de ciclo t_{ciclo} = 1 ns. El procesador puede ejecutar 2 instrucciones por ciclo de reloj.
- Transferencia de **escritura** desde memoria al puerto de E/S
- Transferencia de N_{datos}= 400000 datos
- El tamaño de una dato es m_{dato} = 4 bytes
- Dirección inicial de memoria donde residen los datos: 80000000h

a) El siguiente código realizado con el repertorio de instrucciones CISCA realiza la transferencia descrita antes mediante la técnica de E/S programada. Completar el código.

```
1.        MOV R3, _400000_
2.        MOV R2, 80000000h
3.Bucle:  _IN_     R0, [0E00h]   ; leer 4 bytes
4.        AND R0, _00010000b_
5.        _JE_Bucle
6.        MOV R0,_[R2]_          ; leer 4 bytes
7.        ADD R2, 4
8.        OUT _[0E04h]_, R0      ; escribir 4 bytes
9.        SUB R3, 1
10.       JNE Bucle
```

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

$t_{transf\_bloque} = t_{latència} + (N_{datos} * t_{transf\_dada})$
$t_{latencia} = 0$
$N_{datos} = 200000$
$t_{transf\_dada} = m_{dato} / v_{transf}$ = 4 Bytes / 10000 Kbytes/s = 0,0004 ms = 0,4 us
t_{transf_bloque} = 0 + (400000 * 0,0004 ms) = 160 ms

Asignatura	**Código**	**Fecha**	**Hora inicio**
Estructura de computadores	75.573	10/06/2020	18:30

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

t_{ciclo} = 1 ns (nanosegundo)

t_{instr} = 1 ns / 2 = 0,50 ns

El mínimo número de instrucciones que ha de ejecutar el programa para cada dato transferido son las 8 instrucciones: 3, 4, 5, 6, 7, 8, 9 i 10. Ejecutar las 8 instrucciones requiere 8 * t_{instr} = 8 * 0,50 ns = 4 ns

Por tanto, el tiempo mínimo para transferir un dato es: 4 ns

Se pueden tranferir 4 bytes cada 4 ns, es decir: $4 / 4 * 10^{-9} =$ 1000 Mbyte/s = 1 Gbytes/s

Examen 2019/20-2

Asignatura	Código	Fecha	Hora inicio
Estructura de computadores	75.573	10/06/2020	18:30

Pregunta 4

4.1

¿Todas las instrucciones modifican los bits de estado Z, V, S y C? Razona la respuesta y pon ejemplos de tu razonamiento

Las instrucciones no tienen por qué modificar los bits de estado. Pueden modificar alguno, todos o ninguno, dependiendo del tipo de instrucción.

Por ejemplo:

JMP: no modifica los bits de estado. Se trata de una instrucción de salto incondicional que simplemente pone en el PC la dirección contenida en su operando.

ADD: es una operación aritmética que puede modificar todos los bits de estado citados dependiendo del resultado.

NOT: se trata de una instrucción de negación lógica que no modifica ninguno de los 4 bits de estado.

4.2

4.2.1 ¿Cuáles son las tres políticas de asignación para almacenar datos dentro de una memoria cache? ¿En qué consisten?

1) Política de asignación directa: un bloque de la memoria principal sólo puede estar en una única línea de la memoria cache. La memoria cache de asignación directa es la que tiene la tasa de fallos más alta, pero se utiliza mucho porqué es la más barata y fácil de gestionar

2) Política de asignación completamente asociativa: un bloque de la memoria principal puede almacenarse en cualquier línea de la memoria cache. La memoria cache completamente asociativa es la que tiene la tasa de fallos más baja. A pesar de eso, no se suele utilizar porqué es la más cara y compleja de gestionar.

3) Política de asignación asociativa por conjuntos: un bloque de la memoria principal puede almacenarse en un subconjunto de las líneas de la memoria cache, pero dentro del subconjunto puede encontrarse en cualquier posición. La memoria cache asociativa por conjuntos es una combinación.

4.2.2 En un sistema de E/S gestionado por DMA. Explica, cuándo i por qué se produce una interrupción. ¿Sirve para indicar el inicio o el final de una transferencia? ¿Quién la genera?

Finalización de la operación de E/S: Cuando ha finalizado la transferencia del bloque el controlador de DMA envía una petición de interrupción al procesador para informar que ha acabado la transferencia de datos.

Examen 2019/20-2

Asignatura	**Código**	**Fecha**	**Hora inicio**
Estructura de computadores	75.573	10/06/2020	18:30

Examen 2019/20-2

Asignatura	**Código**	**Fecha**	**Hora inicio**
Estructura de computadores	75.573	10/06/2020	18:30

Examen 2019/20-2

Asignatura	**Código**	**Fecha**	**Hora inicio**
Estructura de computadores	75.573	10/06/2020	18:30

Examen 2019/20-2

Asignatura	**Código**	**Fecha**	**Hora inicio**
Estructura de computadores	75.573	10/06/2020	18:30

Examen 2019/20-2

Asignatura	**Código**	**Fecha**	**Hora inicio**
Estructura de computadores	75.573	10/06/2020	18:30

Examen 2019/20-2

Asignatura	**Código**	**Fecha**	**Hora inicio**
Estructura de computadores	75.573	10/06/2020	18:30