PEC2

NOMBRE:
APELLIDOS:

Estructura de computadores

2021 s2

Estudios de informática, multimedia y comunicación

EIMT.UOC.EDU

UOC
Universitat Oberta de Catalunya

Presentación

La presente PEC2 contiene 2 preguntas y representa el 50% de la nota de la evaluación continua.

Como podréis ver, los ejercicios son muy parecidos a los que habéis hecho durante estos días, en los que además habéis podido dar las soluciones, comentarlas y plantear dudas en el foro. Esta PEC es **individual**, **evaluable** y por tanto no puede comentarse.

Competencias

Las competencias específicas que persigue la PEC2 son:

- [13] Capacidad para identificar los elementos de la estructura y los principios de funcionamiento de un ordenador.
- [14] Capacidad para analizar la arquitectura y organización de los sistemas y aplicaciones informáticos en red.
- [15] Conocer las tecnologías de comunicaciones actuales y emergentes y saberlas aplicar convenientemente para diseñar y desarrollar soluciones basadas en sistemas y tecnologías de la información.

Objetivos

Los objetivos de la siguiente PEC son:

- Conocer la organización del sistema de memoria de un computador.
- Conocer el funcionamiento de la memoria cache, así como los algoritmos de correspondencia y reemplazamiento.
- Conocer la organización del sistema de entrada/salida.
- Comprender las técnicas de entrada/salida (entrada/salida programada, Interrupciones y DMA).

Enunciado

Responder cada pregunta o apartado en el recuadro correspondiente.

Recursos

Podéis consultar los recursos disponibles en el aula, pero no hacer uso del foro.

Criterios de valoración

La **puntuación** y los **criterios de evaluación** los encontraréis en cada pregunta.

Formato y fecha de entrega

La PEC2 se ha de entregar en el apartado de **entrega de actividades en un documento en formato pdf.** Es necesario que rellenéis en el espacio reservado de la primera página de la PEC vuestro **NOMBRE Y APELLIDOS.** La fecha **límite** de entrega es el **02/05/2022.**

Enunciado

Pregunta 1 (4 puntos)

Tenemos un sistema de memoria en el que todos los accesos se hacen a palabra (no nos importa cuál es el tamaño de una palabra). Supondremos que el espacio de direcciones de memoria se descompone en bloques de B= 8 palabras. Cada bloque comienza en una dirección múltiplo de B. Una fórmula para calcular el identificador numérico del bloque es la siguiente:

Bloque = dirección de memoria (dirección palabra) DIV B (tamaño del bloque en palabras)

Suponemos que el sistema dispone de una memoria cache de L= 4 líneas (donde cada línea tiene el tamaño de un bloque). Estas líneas se identifican como líneas 0, 1, 2 y 3.

Apartado 1.1 (2 puntos) Memoria Cache de Acceso Directo

Suponemos que el sistema utiliza una **política de asignación directa**, de manera que cada bloque de la memoria principal sólo se puede llevar a una línea determinada de la memoria cache. En este caso, el identificador del bloque determina la línea específica donde se puede guardar utilizando la siguiente fórmula (similar a la fórmula para determinar el bloque):

Línea = identificador de bloque MOD L (tamaño de la cache en líneas)

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

12, 13, 14, 22, 2, 23, 3, 28, 43, 44, 45, 14, 46, 28, 54, 55, 56, 14, 44, 45

1.1.a) (1,2 puntos)

La siguiente tabla muestra el estado inicial de la cache, que contiene las primeras palabras de la memoria (organizadas en L bloques). Completar la tabla para mostrar la evolución de la cache durante la ejecución del programa, indicando únicamente los fallos que se producen. Para cada fallo hay que llenar una columna indicando el nuevo bloque que se trae a la memoria cache a la línea que corresponda, expresado de la forma b:e (a_1 – a_B), donde b: número de bloque de memoria principal, e: etiqueta y (a_1 – a_B) son las direcciones del bloque, donde a_1 es la primera dirección y a_B es la última dirección del bloque.

Línea	Estado Inicial	Fallo: 43	Fallo: 14	Fallo: 46	Fallo: 54	Fallo: 56
0	0:0 (0 - 7)					
1	1:0 (8 - 15)	5:1 (40 – 47)	1:0 (8 – 15)	5:1 (40 – 47)		
2	2:0 (16 - 23)				6:1 (48 – 55)	
3	3:0 (24 - 31)					7:1 (56 – 63)

Línea	Fallo: 14	Fallo: 44	Fallo:	Fallo:	Fallo:	Fallo:
0						
1	1:0 (8 – 15)	5:1 (40 – 47)				
2						
3						

1.1.b) (0,4 puntos)
¿Cuál es la tasa de aciertos (T_a)?

T_a = (20 accesos – 7 fallos) / 20 accesos = 13 aciertos / 20 accesos = 0,65

1.1.c) (0,4 puntos)
Suponemos que el tiempo de acceso a la memoria cache, o tiempo de acceso en caso de acierto (t_a), es de 10 ns y el tiempo total de acceso en caso de fallo (t_f) es de 40 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 * 32 ns = 2,6 ns + 11,2 ns = 13,8 ns

Apartado 1.2 Memoria cache de acceso completamente asociativo
Ahora suponemos que el mismo sistema utiliza una política de emplazamiento completamente asociativa, de manera que cualquier bloque de la memoria principal se puede llevar a cualquier bloque de la memoria cache. Si encontramos que la cache ya está llena, se utiliza un **algoritmo de reemplazamiento LRU**, de manera que sacaremos de la memoria cache aquel bloque que hace más tiempo que no se referencia.

Consideremos la misma lista de lecturas a memoria:

12, 13, 14, 22, 2, 23, 3, 28, 43, 44, 45, 14, 46, 28, 54, 55, 56, 14, 44, 45

1.2.a) (1,2 puntos)
La siguiente tabla muestra el estado inicial de la cache. Completar la tabla para mostrar la evolución de la cache durante la ejecución del programa, indicando únicamente los fallos que se producen. Para cada fallo hay que llenar una columna indicando el nuevo bloque que se traerá a la memoria cache a la línea que le corresponda, expresado de la forma b (a_1 – a_B), donde b: número de bloque, y (a_1 – a_B) son las direcciones del bloque, donde a_1 es la primera dirección y a_B es la última dirección del bloque.

Línea	Estado Inicial	Fallo: 43	Fallo: 14	Fallo: 54	Fallo: 56	Fallo: 14
0	0 (0 - 7)			6 (48 - 55)		
1	1 (8 - 15)	5 (40 – 47)				1 (8 – 15)
2	2 (16 - 23)		1 (8 – 15)		7 (56 – 63)	
3	3 (24 - 31)					

Línea	Fallo: 44	Fallo:	Fallo:	Fallo:	Fallo:	Fallo:
0						
1						
2						
3	5 (40 – 47)					

1.2.b) (0,4 puntos)
¿Cuál es la tasa de aciertos (T_a)?

T_a = (20 accesos – 6 fallos) / 20 accesos = 14 aciertos / 20 accesos = 0,7

1.2.c) (0,4 puntos)
Suponemos los mismos datos de tiempo de acceso en los casos de acierto y de fallo que en el apartado 1.1.c) anterior. Considerando la tasa de aciertos obtenida en la pregunta anterior, ¿cuál es el tiempo medio de acceso a memoria (t_m)?

$t_m = T_a \times t_a + (1-T_a) \times t_f = 0{,}7 * 4 \text{ ns} + 0{,}3 * 32 \text{ ns} = 2{,}8 \text{ ns} + 9{,}6 \text{ ns} = 12{,}4 \text{ ns}$

Criterios de valoración. Para los apartados 1.1.a y 1.2.a cada error en los fallos o aciertos de la memoria cache o en la colocación de un bloque en la cache resta 0,6. Los apartados restantes se puntuarán con los 0,4 puntos cada uno de ellos si la solución es correcta y coherente con vuestra respuesta a los apartados a) correspondientes.

Pregunta 2 (5 puntos)

Se desea analizar el rendimiento de la comunicación de datos entre la memoria de un procesador y un puerto USB con un dispositivo conectado, que tienen las siguientes características:

- Velocidad de transferencia del dispositivo de E/S ($\mathbf{v}_{transf}$)= 2 MB/s
- Tiempo de latencia promedio del dispositivo ($t_{latencia}$) = 0
- Direcciones de los **registros de datos y de estado** del controlador de E/S: A000h y A004h
- El bit del **registro de estado** que indica que el controlador del puerto de E/S está disponible es el bit 2, o el tercer bit menos significativo (cuando vale 1 indica que está disponible)
- Procesador con una frecuencia de reloj de 4GHz, y el procesador puede ejecutar 1 instrucción cada 4 ciclos de reloj (t_{instr}= 4*t_{ciclo})
- Transferencia de **escritura** desde memoria al puerto de E/S
- En cada escritura de un dato se transfieren 4 Bytes
- Transferencia de N_{datos}= 4.000.000 datos, es decir, 4.000.000 × 4 Bytes = 16.000.000 Bytes
- Dirección inicial de memoria donde residen los datos: B0000000h

Apartado 2.1 (2 puntos) E/S programada

2.1.a) (0,5 puntos)

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

```
1.              MOV   R3, _4000000_
2.              MOV   R2, B0000000h
3. Bucle:       _IN_  R0, [A004h]          ; leer 4 bytes
4.              AND   R0, _00000100b_
5.              _JE__ Bucle
6.              MOV   R0,_[R2]_            ; leer 4 bytes
7.              ADD   R2, _4__
8.              _OUT_ [A000h], R0          ; escribir 4 bytes
9.              SUB   R3, _1_
10.             JNE   Bucle
```

2.1.b) (0,5 puntos)

¿Cuánto tiempo dura la transferencia t_{transf_bloque}?
¿Qué porcentaje de este tiempo dedica la CPU a la transferencia?

$t_{transf\_bloque} = t_{latencia} + (N_{datos} * t_{transf\_dato})$
$t_{latencia} = 0$
$N_{datos} = 4.000.000$
$t_{transf\_dato} = m_{dato} / v_{transf} = 4$ Bytes / 2 MByte/s = 2 us
$t_{transf\_bloque} = 0 + (4.000.000 * 2\ us) = 8$ s

La CPU dedica el 100% del tiempo y, por lo tanto, el tiempo coincide con el tiempo dedicado por el periférico t_{bloc}.

2.1.c) (1 punto)

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

Frecuencia de reloj = 4 GHz, implica un tiempo de ciclo de
$t_{ciclo} = 1/4 * 10^{-9} = 0{,}25$ ns (nanosegundos)
$t_{instr} = 4 * t_{ciclo} = 4 * 0{,}25$ ns = 1 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 y 10. Ejecutar las 8 instrucciones requiere $8 \times (t_{instr}) = 8 * 1$ ns = 8 ns

Por lo tanto, el tiempo mínimo para transferir un dato es 4 ns
Se pueden transferir 4 bytes cada 8 ns, es decir: $4 / (8 \times 10^{-9}) = 500$ MBytes/s

Apartado 2.2 (2 puntos) E/S por Interrupciones

Suponer que el siguiente código CISCA es una rutina de servicio a las interrupciones (RSI) para transferir a través del dispositivo de E/S anterior, el mismo número de datos que antes con E/S programada, pero ahora mediante la técnica de E/S por interrupciones.
Suponer:

- Tiempo de programación y finalización de la transferencia de 1000 ns ($t_{prog} + t_{final}$)
- El tiempo para atender la interrupción (t_{rec_int}), o tiempo adicional desde que la CPU detecta la interrupción hasta que comienza a ejecutarse la primera instrucción de la RSI es de 16 ciclos de reloj.
- Se utiliza una variable global que se representa con la etiqueta **DIR,** y que al principio del programa contiene la dirección inicial de memoria donde residen los datos a transferir.

2.2.a) (0,5 puntos)

Completar el código:

```
1.   CLI
2.   PUSH    R0
3.   PUSH    R1
4.   IN      R0, __[A004h]___     ; leer 4 bytes
5.   _AND_   R0, 00000100b
6.   JE  Error          ; salta a un código no descrito de tratamiento
                        ; de error se ha producido la petición por parte
                        ; del dispositivo, pero el dato no está disponible.
7.   MOV     _R1_, [DIR]
8.   MOV     R0, [R1]            ; leer 4 bytes
9.   _OUT_   [A000h],R0          ; escribir 4 bytes
10.  ADD     R1, _4__
11.  MOV     _[DIR]_ , R1
12.  POP R1
13.  POP R0
14.  STI
15.  _RETI_
```

2.2.b) (1 punto)

¿Cuál es el tiempo total que dedica la CPU a la tarea de Entrada/Salida, t_{cpu}? ¿Cuál es el porcentaje que representa el tiempo de transferencia del bloque t_{transf_bloque} con respecto al tiempo de transferencia del bloque por parte del periférico t_{bloque}?

El tiempo de un ciclo, t_{ciclo}, es 0,25 ns (nanosegundos). Ver solución 2.1.c
Tiempo para atender la interrupción, t_{rec_int}: 16 ciclos × 0,25 ns (t_{cicle}) = 4 ns
Tiempo de ejecución de una instrucción, t_{instr} = 1 ns
Tiempo de ejecución RSI, t_{rsi}: $N_{rsi} \times t_{instr}$ = 15 instrucciones × 1 ns = 15 ns
Tiempo consumido por la CPU en cada interrupción, t_{transf_dato}:
$t_{transf\_dato} = t_{rec\_int} + t_{rsi} = 4 + 15 = 19$ ns

Número de interrupciones producidas (o número total de datos, N_{datos}): 4.000.000 interrupciones.
Tiempo consumido en total en TODAS las interrupciones: $t_{transf\_bloque} = t_{transf\_dato} \times N_{datos}$ = 19 ns × 4.000.000 interrupciones = 76 ms (milisegundos)

El tiempo final de ocupación de la CPU debe incluir el tiempo de programación y finalización de la transferencia:
$t_{cpu} = (t_{prog} + t_{final}) + t_{transf\_bloque}$ = 1000 ns + 76.000.000 ns = 76.001000 ns = 76,001 ms

De los 8 s = 8.000 ms de tiempo total para realizar la transferencia (tiempo calculado en el apartado 2.1.b), la CPU está dedicada a la tarea de E/S:
%ocupación = $t_{transf\_bloque} * 100 / t_{bloque}$ = 76,001 ms* 100 / 8.000 ms => 0,95 % del tiempo.

2.2.c) (0,5 puntos)

Si quisiéramos reducir la frecuencia de reloj del procesador para reducir su consumo energético, hasta qué frecuencia lo podríamos hacer sin reducir la velocidad de transferencia con el dispositivo de E/S?

En la fase de transferencia de datos, el controlador de E/S genera 1.600.000 interrupciones durante 6,4 segundos. (dato calculado en el apartado 2.1).
$N_{datos} * t_{dato}$ = 8 s = 8.000 ms = 8.000.000 us (microsegundos)
Es decir, tenemos una interrupción cada 8.000.000 us / 4.000.000 = 2 us (microsegundos).
Este es el tiempo máximo que debería tardar la gestión de la interrupción (t_{transf_dato}), incluyendo el tiempo adicional para transferir el control a la RSI.

El tiempo consumido por la CPU en cada interrupción es, como hemos visto en el apartado anterior, la suma de los tiempos de transferir el control a la RSI + ejecutar la RSI:
En el enunciado se define que t_{rec_int} = 16 ciclos de reloj = $16 * t_{ciclo}$ y por lo tanto:
$t_{transf\_dato} = t_{rec\_int} + t_{rsi} = t_{rec\_int} + (N_{rsi} * t_{instr}) = 16 * t_{ciclo} + (15 * t_{instr})$

Tal como hemos visto en el apartado anterior y como dice el enunciado, el tiempo de una instrucción es: $t_{instr} = 4 * t_{ciclo}$
Por lo tanto: $t_{transf\_dada} = 16 * t_{ciclo} + (15 * 4 * t_{ciclo}) = 76 * t_{ciclo}$

Queremos encontrar el tiempo de ciclo tal que el tiempo de transferencia de un dato sea 4 us:
2 us = $76 * t_{ciclo}$ => t_{ciclo} = 2 us / 76 = 0,0263 us = 26,31ns
$1 / 26{,}31 * 10^{-9}$ = 38 MHz

Apartado 2.3 (1 punto) E/S por DMA

Supondremos que el controlador de E/S puede funcionar en modo DMA (Acceso Directo a Memoria). La suma del tiempo de cesión del bus y del tiempo de recuperación del bus es de 10 ns ($\mathbf{t}_{cesión}$ + $\mathbf{t}_{recup}$ = 10 ns). El tiempo de la transferencia de un dato por el bus es de 1 ns ($\mathbf{t}_{mem}$ = 1 ns).

2.3.a) (0,5 puntos)

Consideremos que en la transferencia por DMA, los datos se envían entre el controlador de DMA y la memoria, en modo ráfaga, y dispone de un buffer de un tamaño $\boldsymbol{m}_{buffer}$ = 4000 bytes. Calcular el tiempo total de ocupación del bus por parte del controlador de DMA para llevar a cabo la transferencia que venimos analizando.

Medida de les ráfagas $\boldsymbol{N}_{ráfaga}$: $\boldsymbol{m}_{buffer}$ / $\boldsymbol{m}_{dato}$ = 4000 / 4 = 1000
Tiempo ocupación Bus, $\boldsymbol{t}_{transf\_ráfaga}$: $\boldsymbol{t}_{cesión}$ + 1000×$\boldsymbol{t}_{mem}$ + $\boldsymbol{t}_{recup}$ = 10 + 1000 * 1 = 1.010 ns
Número de peticiones del Bus, $\boldsymbol{N}_{datos}$ / $\boldsymbol{N}_{ráfaga}$: 4.000.000 / 1.000 = 4.000

Tiempo total de ocupación del Bus $\boldsymbol{t}_{transf\_bloque}$: $\boldsymbol{t}_{transf\_ráfaga}$×($\boldsymbol{N}_{datos}$/$\boldsymbol{N}_{ráfaga}$) =1.010 ns * 4.000 = 4.040.000 ns = 4.040 us = 4,04 ms

2.3.b) (0,5 puntos)

La CPU no puede hacer ninguna tarea durante todo el tiempo en que el bus está ocupado por parte del controlador de DMA. ¿Cuál es el porcentaje de tiempo que tiene disponible la CPU para ejecutar código efectivo de otros programas durante la transferencia?

$\boldsymbol{t}_{transf\_bloque}$ = 4,044 ms
$\boldsymbol{t}_{bloque}$ = 8.000 ms
%ocupación = (t_{transf_bloque} * 100) / t_{bloque}

Porcentaje de tiempo disponible: 100 – % ocupación = 100 - ($\boldsymbol{t}_{transf\_bloque}$ * 100 / $\boldsymbol{t}_{bloque}$) = 100 – (4,04 * 100 / 8.000) = 100 - 0,0505 = 99,9495%

Criterios de valoración. En los apartados 2.1.a y 2.2.a cada valor erróneo resta 0,25. El resto de apartados están bien o están mal: no hay gradación.

Pregunta 3 (1 punto)

Preguntas teóricas

3.a) (0,25 puntos)

¿Por qué el sistema de memoria se organiza en una estructura jerárquica? ¿Cuáles son los niveles de esta estructura?

El objetivo final de la jerarquía de memorias es conseguir que, cuando el procesador acceda a un dato, éste se encuentre en el nivel más rápido de la jerarquía. Conseguimos tener una memoria con un coste moderado, una velocidad cercana a la del nivel más rápido y la capacidad del nivel mayor.

Los niveles son:

Registros del procesador
Memoria caché
Memoria Principal
Memoria Secundaria

3.b) (0,25 puntos)

En la memoria caché, ¿Cuándo debe utilizarse un algoritmo de reemplazo? ¿Cuáles son los principales algoritmos de reemplazo?

Cuando se produce un fallo de memoria caché y se tiene que llevar a la memoria caché un bloque de memoria principal determinado, si este bloque de memoria se puede almacenar en más de una línea de la memoria caché, hay que decidir en qué línea se emplaza, de todas las posibles, y sobrescribir los datos que se encuentran en aquella línea. El algoritmo de reemplazo se encarga de esta tarea.

Los principales algoritmos de reemplazo son:
1) FIFO (*first in first out*). Para elegir la línea se utiliza una cola, de manera que la línea que hace más tiempo que está almacenada en la memoria caché será la reemplazada. Este algoritmo puede reducir el rendimiento de la memoria caché porque la línea que se encuentra almacenada en la memoria caché desde hace más tiempo no tiene que ser necesariamente la que se utilice menos. Se puede implementar fácilmente utilizando técnicas de *buffers* circulares (o *round-robin*): cada vez que se debe sustituir una línea se utiliza la línea del *buffer* siguiente, y cuando se llega a la última, se vuelve a empezar desde el principio.
2) LFU (*least frequently used*). En este algoritmo se elige la línea que hemos utilizado menos veces. Se puede implementar añadiendo un contador del número de accesos a cada línea de la memoria caché.
3) LRU (*least recently used*). Este algoritmo elige la línea que hace más tiempo que no se utiliza. Es el algoritmo más eficiente, pero el más difícil de implementar, especialmente si hay que elegir entre muchas líneas. Se utiliza habitualmente en memorias caché asociativas por conjuntos, con conjuntos pequeños de 2 o 4 líneas. Para memorias cachés 2-asociativas, se puede implementar añadiendo un bit en cada línea; cuando se hace referencia a una de las dos líneas, este bit se pone a 1 y el otro se pone a 0 para indicar cuál de las dos líneas ha sido la última que se ha utilizado.
4) Aleatorio. Los algoritmos anteriores se basan en factores relacionados con la utilización de las líneas de la caché; en cambio, este algoritmo elige al azar la línea que se debe

reemplazar al azar. Este algoritmo es muy simple y se ha demostrado que tiene un rendimiento solo ligeramente inferior a los algoritmos que tienen en cuenta factores de utilización de las líneas.

3.c) (0,25 puntos)
¿Cuáles son las tres partes básicas de un módulo de E/S? ¿Qué tipos de registros incluye un módulo de E/S?

En un módulo de E/S distinguimos tres partes básicas:
1) Una interfaz interna normalizada con el resto del computador mediante el bus de sistema que nos da acceso al banco de registros del módulo de E/S.
2) Una interfaz externa específica para el periférico que controla. Habitualmente la conexión con el periférico se realiza mediante un sistema de interconexión normalizado de E/S.
3) La lógica necesaria para gestionar el módulo de E/S. Es responsable del paso de información entre la interfaz interna y externa.

Los registros que incluye un módulo de E/S son:
• Registros de control.
• Registros de estado.
• Registros de datos.

3.d) (0,25 puntos)
En un sistema de E/S gestionada por DMA ¿Qué diferencia hay entre el funcionamiento normal y el modo ráfaga, qué ventajas tiene un modo con respecto al otro?

Una manera de optimizar las operaciones de E/S por DMA consiste a reducir el número de cesiones y recuperaciones del bus. Para hacerlo, en lugar de solicitar y liberar el bus para cada dato que se tiene que transferir, se solicita y se libera el bus para transferir un conjunto de datos de manera consecutiva. Esta modalidad de transferencia se denomina modo ráfaga.

Para hacer la transferencia de este conjunto de datos, que denominamos ráfaga, el controlador de DMA hace falta que disponga de una memoria intermedia (buffer), de forma que la transferencia de datos entre la memoria y el controlador de DMA es pueda hacer a la velocidad que permita la memoria y no quedando limitada a la velocidad del periférico.

Este modo de funcionamiento no afecta la programación ni la finalización de la operación de E/S descrita anteriormente, pero sí que modifica la transferencia de datos