



AEC3 – Sistemas Operativos

ISMA HERNANDEZ

ORGANIZACIÓN DE LA ACTIVIDAD PRÁCTICA

Nombre de la práctica	
Tipo de tarea	Individual
Entregables	PDF a modo de memoria ; fichero del código en el lenguaje de programación decidido por el alumno y link a un video corto (máximo 10 minutos) subido a su onedrive, youtube, Dropbox, etc donde muestre de viva voz la ejecución de su programa y explique el código.

DESCRIPCIÓN DE LA ACTIVIDAD PRÁCTICA

Esta actividad práctica de la unidad 3 consta de 2 partes:

1ª parte es un ejercicio de memoria caché

2ª parte es un ejercicio de programación relacionado con memoria caché.

1º Problema caché:

Sea la siguiente dirección de memoria en hexadecimal: AC05H. Se pide:

- a) ¿Qué tamaño tiene la Memoria principal? Expréselo en Bytes como potencia de dos y expréselo también como una unidad múltiplo del Byte (por ejemplo KB)

Si tenemos una caché con correspondencia directa, tamaño de bloque de 16 bytes y una etiqueta de 4bits

- b) ¿Cuál es el tamaño de línea de caché?
c) ¿Qué tamaño tiene la memoria caché?
d) ¿Cuántas líneas tiene la caché?
e) Indique el valor de los tres campos que conforman la correspondencia directa.

NOTA: No basta con poner el resultado. Hay que justificarlo. No utilizéis calculadora, lo cálculos siempre con potencias de 2.

2º Ejercicio de programación:

Se pide a los alumnos que elijan un lenguaje de programación de sus elección. El objetivo es hacer un programa que presente un menú con las siguientes opciones:

1. Tipo de problema de caché 1
2. Tipo de problema de caché 2
3. Salir

Hemos visto variantes de ejercicios de memoria caché (en algunos nos dan información del tamaño del bus, en otro del tamaño de la memoria, nos dan las direcciones en hexadecimal, decimal o binario, nos dan el tamaño de bloque o el tamaño de línea, etc). Pues bien, el alumno debe escoger dos variantes cualesquiera (1 para la opción 1 y otro para la 2) y realizar un programa que pide los datos necesarios y acabe devolviendo en cada caso la estructura de la dirección (campos de palabra, marco, línea o etiqueta según corresponda) en los tres modos de correspondencia (directa, asociativa o asociativa por conjuntos)

El alumno entregará el código de su programa, documentación con pantallazos o explicaciones que estime convenientes en el pdf (que también incluye el ejercicio 1) y un link a un video donde se le escuche, se le vea y se vea también el código. Hay que explicar el código y realizar ejemplos de ejecución en un tiempo máximo de 10 minutos. Tiempos superiores de video penalizarán.

EJERCICIO 1:

Apartado A)

¿Qué tamaño tiene la memoria principal?

Para calcular el tamaño de la memoria principal, debemos fijarnos en la dirección que nos dan: AC05H. Esto está en hexadecimal, y sabemos que cada dígito hexadecimal equivale a 4 bits.

Dado que la dirección tiene 4 dígitos, hacemos el cálculo:

$$4 \text{ dígitos} * 4 \text{ bits por dígito} = 16 \text{ bits}$$

Con 16 bits podemos representar todas las direcciones desde 0000H hasta FFFFH, lo que equivale a 2 elevado a 16 posiciones de memoria. Cada una de estas posiciones almacena 1 byte, así que:

Tamaño total de la memoria principal:

Tamaño total de la memoria principal = 2 elevado a 16 = 65,536 bytes.

Como es un número grande, lo convertimos a una unidad más común. Sabemos que 1 KB = 2 elevado a 10 bytes, así que:

$$65.536 \text{ bytes son } 64\text{KB}$$

Por tanto, el tamaño de la memoria principal es de 64KB o de otra forma 2 elevado a 16 bytes.

b) ¿Cuál es el tamaño de la línea de caché?

Aquí el problema nos dice que el tamaño del bloque es de 16 bytes. Ahora, en una caché de correspondencia directa, cada línea de caché almacena exactamente un bloque. Esto significa que el tamaño de la línea de caché es igual al tamaño del bloque, porque solo se almacena ese bloque en cada línea.

Entonces:

- Tamaño de la línea de caché: 16 bytes.

Es importante recalcar que no estamos hablando del tamaño total de la caché, sino únicamente del espacio que ocupa una línea.

d) Sabemos que la memoria principal tiene un tamaño total de 2^{16} bytes, y que el tamaño de cada bloque es de 16 bytes o lo que es lo mismo 2 elevado a 4 bytes. En una caché de correspondencia directa, cada bloque de la memoria principal se mapea a una línea única en la caché. El número total de líneas se calcula dividiendo el tamaño de la memoria principal entre el tamaño del bloque:

$$\text{Numero de lineas} = \text{Tamaño memoria principal} / \text{Tamaño del bloque}$$

$$\text{Numero de lineas} = 2 \text{ elevado } 16 / 2 \text{ elevado } 4 = 4096 \text{ lineas}$$

Esto significa que la caché tiene 4096 líneas, cada una capaz de almacenar un bloque de 16 bytes.

c) ¿Qué tamaño tiene la memoria caché?

Para calcular el tamaño de la caché, necesitamos saber cuántas líneas hay y el tamaño de cada línea. El número de líneas lo voy a calcular en el siguiente apartado, pero podemos adelantar la fórmula:

$$\text{Tamaño de memoria cache} = \text{Número de líneas} * \text{Tamaño de cada línea}$$

Sabemos por el apartado d) que hay 4096 líneas y que cada línea tiene 16 bytes, calculamos el tamaño total de la caché:

$$\text{Tamaño de la caché} = 4096 * 16 = 2 \text{ elevado } 16 \text{ bytes.}$$

Al igual que la memoria principal la caché tiene un tamaño de 64KB.

e) ¿Cuáles son los tres campos que forman la correspondencia directa?

En correspondencia directa, una dirección de memoria se divide en tres campos:

1. Etiqueta: Identifica qué bloque de memoria está almacenado en la línea de caché.
2. Índice: Identifica a qué línea de la caché pertenece la dirección.
3. Desplazamiento: Indica la posición exacta del byte dentro de un bloque.

Para determinar el tamaño de estos campos, debemos dividir los 16 bits de la dirección basándonos en la información que tenemos:

1. Desplazamiento:
 - Cada bloque tiene 16 bytes = $2 \text{ elevado } 4 \text{ bytes}$.
 - Esto significa que necesitamos 4 bits para identificar la posición dentro del bloque.
2. Índice:
 - Hay 4096 líneas = $2 \text{ elevado a } 12 \text{ líneas}$
 - Esto significa que necesitamos 12 bits para identificar la línea de la caché.
3. Etiqueta:
 - Los bits restantes forman la etiqueta. Como ya hemos usado $4 + 12 = 16$ bits, no queda ningún bit para la etiqueta en este caso.

Por tanto, para desplazamiento tendremos 4 bits, para índice 12 bits y para etiqueta 0 bits

EJERCICIO 2: DISEÑAR UN PROGRAMA CON EJERCICIOS DE CACHE

En esta sección voy a explicar los archivos que componen mi proyecto y cómo trabajan en conjunto para resolver los problemas de caché. Mi proyecto está diseñado para simular diferentes problemas de caché y está estructurado de forma modular para mantener el código limpio y fácil de gestionar. Cada carpeta y archivo tiene un propósito claro:

Estructura del Proyecto

- **Raíz del Proyecto**

Aquí tengo el archivo Makefile, que se encarga de automatizar la compilación del programa. El ejecutable resultante se llama `cache_simulator`. También está el directorio de objetos compilados (`obj`), el de código fuente (`src`) y el de encabezados (`include`). Todo está ordenado para facilitar la navegación y el mantenimiento.

Carpetas

- **include**

Contiene los archivos de encabezado, como `cache.h` y `utils.h`, que declaran las funciones principales y las utilidades auxiliares. Esto separa las definiciones de las implementaciones, haciendo que mi código sea más modular.

- **src**

Contiene los archivos fuente `main.c`, `cache.c`, y `utils.c`. Cada archivo tiene funciones específicas:

- `main.c`: Es el punto de entrada del programa y controla el flujo principal.
- `cache.c`: Contiene la lógica para calcular las direcciones en caché (mapeo directo, asociativo y set-asociativo).
- `utils.c`: Incluye utilidades como validación de entrada para garantizar que el usuario proporcione datos válidos.

- **obj**

Almacena los archivos objeto (`.o`) generados durante la compilación. Mantengo esta carpeta separada para no mezclar archivos temporales con el código fuente.

Makefile

Este archivo automatiza la construcción del proyecto.

- Especifico el compilador (`gcc`) y las banderas (`CFLAGS`), aunque por ahora no estoy usando `-Wall` ni `-Wextra` ni `-Werror` porque no son necesarias.
- La regla `all` compila todo, y también incluye un dibujo al final para darle un toque personal al proyecto. Me gusta agregar ese detalle para hacerlo más distintivo.
- Las reglas `clean` y `fclean` eliminan archivos temporales y el ejecutable, y con `re` recompilo desde cero.

Flujo del Código

- main.c

Es donde todo comienza. Tengo un menú que le permite al usuario elegir entre dos tipos de problemas de caché:

- El problema 1 toma una dirección en hexadecimal como entrada.
- El problema 2 permite al usuario ingresar un tamaño de bloque y usa una dirección de ejemplo. Según la elección, se llaman funciones en cache.c para realizar los cálculos correspondientes.

- cache.c

Aquí está la lógica central. Implementé tres tipos de mapeos de caché:

- Mapeo Directo: Calcula el bloque, la línea y la etiqueta.
- Mapeo Asociativo: Calcula el bloque y la etiqueta, ya que no hay líneas específicas en este caso.
- Mapeo Set-Asociativo: Similar al mapeo directo, pero con conjuntos en lugar de líneas.

Estas funciones imprimen los resultados y ayudan a entender cómo funcionan estos esquemas en un nivel más técnico.

- utils.c

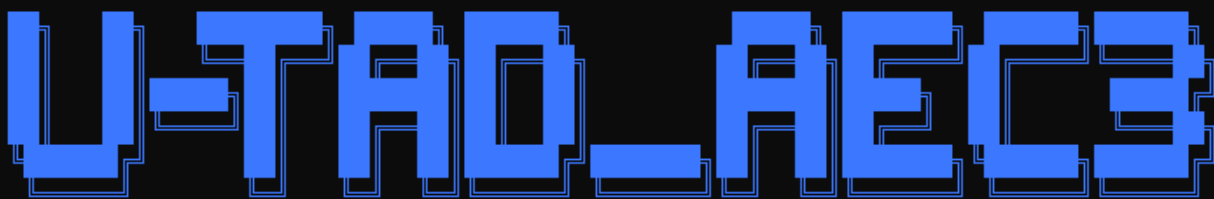
Incluye funciones auxiliares para validar entradas.

- validate_hex_input: Se asegura de que el usuario proporcione una dirección hexadecimal válida.
- validate_int_input: Verifica que los números enteros ingresados sean correctos. Esto es importante para evitar errores de entrada y mejorar la experiencia del usuario.

Capturas

makefile

```
nirmata@nirmata-pc:~/U-Tad/2nd_Course/Q1/Operating_systems/Unit_3/AEC3$ make
Compilando src/main.c...
gcc -Iinclude -c src/main.c -o obj/main.o
Compilando src/cache.c...
gcc -Iinclude -c src/cache.c -o obj/cache.o
Compilando src/utils.c...
gcc -Iinclude -c src/utils.c -o obj/utils.o
Compilando el ejecutable cache_simulator...
gcc -o cache_simulator obj/main.o obj/cache.o obj/utils.o -lm
¡Compilación completada!
```



23/11/2024 ismael.hernandez@live.u-tad.com

```
nirmata@nirmata-pc:~/U-Tad/2nd_Course/Q1/Operating_systems/Unit_3/AEC3$ |
```

```
nirmata@nirmata-pc:~/U-Tad/2nd_Course/Q1/Operating_systems/Unit_3/AEC3$ make re
Eliminando archivos objeto...
Eliminando el ejecutable cache_simulator...
Compilando src/main.c...
gcc -Iinclude -c src/main.c -o obj/main.o
Compilando src/cache.c...
gcc -Iinclude -c src/cache.c -o obj/cache.o
Compilando src/utils.c...
gcc -Iinclude -c src/utils.c -o obj/utils.o
Compilando el ejecutable cache_simulator...
gcc -o cache_simulator obj/main.o obj/cache.o obj/utils.o -lm
¡Compilación completada!
```

U-TAD-AEC3

23/11/2024 ismael.hernandez@live.u-tad.com

menú

```
nirmata@nirmata-pc:~/U-Tad/2nd_Course/Q1/Operating_systems/Unit_3/AEC3$ ./cache_simulator
Menu:
1. Cache problem type 1
2. Cache problem type 2
3. Exit
Choose an option: |
```

Caso problema 1 con numero hexa 3A

```
Menu:
1. Cache problem type 1
2. Cache problem type 2
3. Exit
Choose an option: 1
Enter an address in hexadecimal (without 0x): 3A
The entered address is: 0x3A
Direct Mapping:
Block Offset: 10
Line Number: 3
Tag: 0
Associative Mapping:
Block Offset: 10
Tag: 3
Set-Associative Mapping:
Block Offset: 10
Set Number: 3
Tag: 0
Menu:
1. Cache problem type 1
2. Cache problem type 2
3. Exit
Choose an option: |
```


Caso problema 2 bloque 234

```
Menu:
1. Cache problem type 1
2. Cache problem type 2
3. Exit
Choose an option: 2
Enter the block size: 234
The entered block size is: 234
Direct Mapping:
Block Offset: 54
Line Number: 125
Tag: 5098
Associative Mapping:
Block Offset: 54
Tag: 1305213
Set-Associative Mapping:
Block Offset: 54
Set Number: 1
Tag: 326303
Menu:
1. Cache problem type 1
2. Cache problem type 2
3. Exit
Choose an option: |
```