Universidad de Costa Rica | Facultad de Ingeniería | Escuela de Ingeniería Eléctrica

# Anteproyecto VI

Roberto Sánchez Cárdenas — B77059
Gabriel Jiménez Amador — B73895

San José, 31 de agosto de 2021
IE0424 — Laboratorio de Circuitos Digitales

## Índice



## 1. Introducción

Este laboratorio pretende la implementación de una memoria caché utilizando el lenguaje de descripción de hardware Verilog. En este documento se presentan los diseños del firmware y hardware. Los diseños de hardware se mantienen principalmente en un alto nivel por medio de diagramas.

## 2. Objetivos

- Analizar interfaces de memoria de un CPU e implementar un cache sencillo.
- Recolectar y analizar datos de rendimiento de un cache.

## 3. Anteproyecto

### 3.1. Documente qué significa cada una de las señales que forman la interfaz de memoria de picorv32.

- Mem-ready: En alto habilita la escritura o lectura de un dato.
- Mem-valid: Indica validez de un dato.
- Mem-instr: Cuando está en alto indica que son intruccciones de escritura, en bajo son intrucciones de lectura.
- Mem addr: Contiene la dirección de escritura o lectura.

### 3.2. ¿Qué es el miss/hit rate de un cache?

El miss/hit rate es un parámetro útil para medir la eficiencia de una memoria Caché. Para explicar este concepto primero se debe tener claro que son los aciertos (hits) y (fallos) hits; el miss hace referencia cuando una unidad de procesamiento pide un dato a la memoria caché, pero el dato no se encuentra ahí, mientras que un hit significa que el dato sí estaba. Los datos no encontrados se deben pedir a una memoria de nivel inferior en la jerarquía de memoria, las cuales son más lentas. [2]

Ahora, la tasa de fallos hace referencia a la relación que existe entre el número de aciertos o fallos con el total de datos solicitados. De modo que se puede modelar por medio de la siguiente ecuación:

$$\text{hitrate} = \frac{\text{hits}}{\text{requests}} \cdot 100 \qquad \text{missrate} = \frac{\text{misses}}{\text{requests}} \cdot 100 \tag{1}$$

### 3.3. ¿Qué es un bloque? ¿Cómo se identifica un bloque dentro de un cache?

Las cachés fueron diseñadas de modo que se guarden los datos segmentados en varias posiciones memoria de la caché, sin embargo, en esas posiciones se tiene más que la data de interés. Como se observa en la siguiente figura, cada posición está compuesta por varias partes.

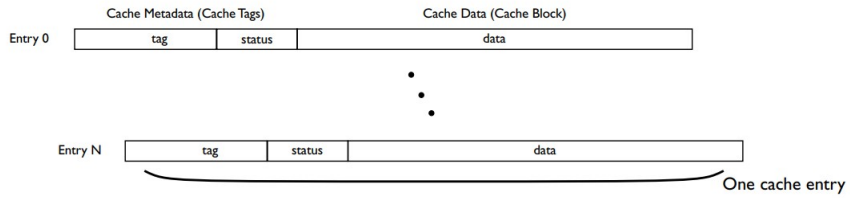


**Figura 1:** Memoria caché

Para identificar cada posición de memoria, se tienen identificadores únicos para cada espacio.

**¿Cuál es la diferencia entre write-back y write-through?**

La política de writeback le indica al procesador que escriba a una posición válida de una memoria caché, mientras que la política de writethrough escribe tanto en la memoria caché como en la memoria principal (RAM). Estas diferenciasen políticas hace que no siempre se tenga la misma data en ambas memorias. Además, la política write through es más lenta. [1]

### 3.4. Documente, cuál es el tamaño del byte offset, el tamaño del índice del bloque y el tamaño del tag para los siguientes tamaños de cache y bloque:

Asumiendo que el tamaño de la dirección física de la caché es de 32 bits y que se utiliza asociatividad directa, se tienen las siguientes fórmulas:

$$\text{offset} = \log_2(\text{tamañoBloque}) \tag{2}$$

$$\text{índice} = log_2\left(\frac{\text{tamañoCache}}{\text{tamañoBloque}}\right) \tag{3}$$

$$\text{tag} = \text{tamañoDirFísica - offset - índice} \tag{4}$$

| Tamaño de bloque | Tamaño del cache | Bytes por bloque | Offset | Índice | Tag |
|---|---|---|---|---|---|
| 2 palabras | 1kB | 8 | 3 | 7 | 22 |
| 2 palabras | 2kB | 8 | 3 | 8 | 21 |
| 4 palabras | 2kB | 16 | 4 | 7 | 21 |
| 2 palabras | 4kB | 8 | 3 | 9 | 20 |
| 4 palabras | 4kB | 16 | 4 | 8 | 20 |

**Tabla 1:** Tamaños de las direcciones en memoria caché

# 4. Diseño

## 4.1. Ejercicio 1

Para este ejercicio se propone diseñar el *firmware* siguiendo el diagrama de flujo de la Figura 2. En este diagrama se tiene un bucle inicial que carga los últimos 48 KB de memoria para la creación de la lista enlazada, escribiendo a las direcciones (espaciadas por 4 bytes). Luego lee la lista enlazada, recorriendo entrada por entrada siguiendo al puntero respectivo y guarda los últimos 5 datos impares y los envía a la salida `out_byte`.

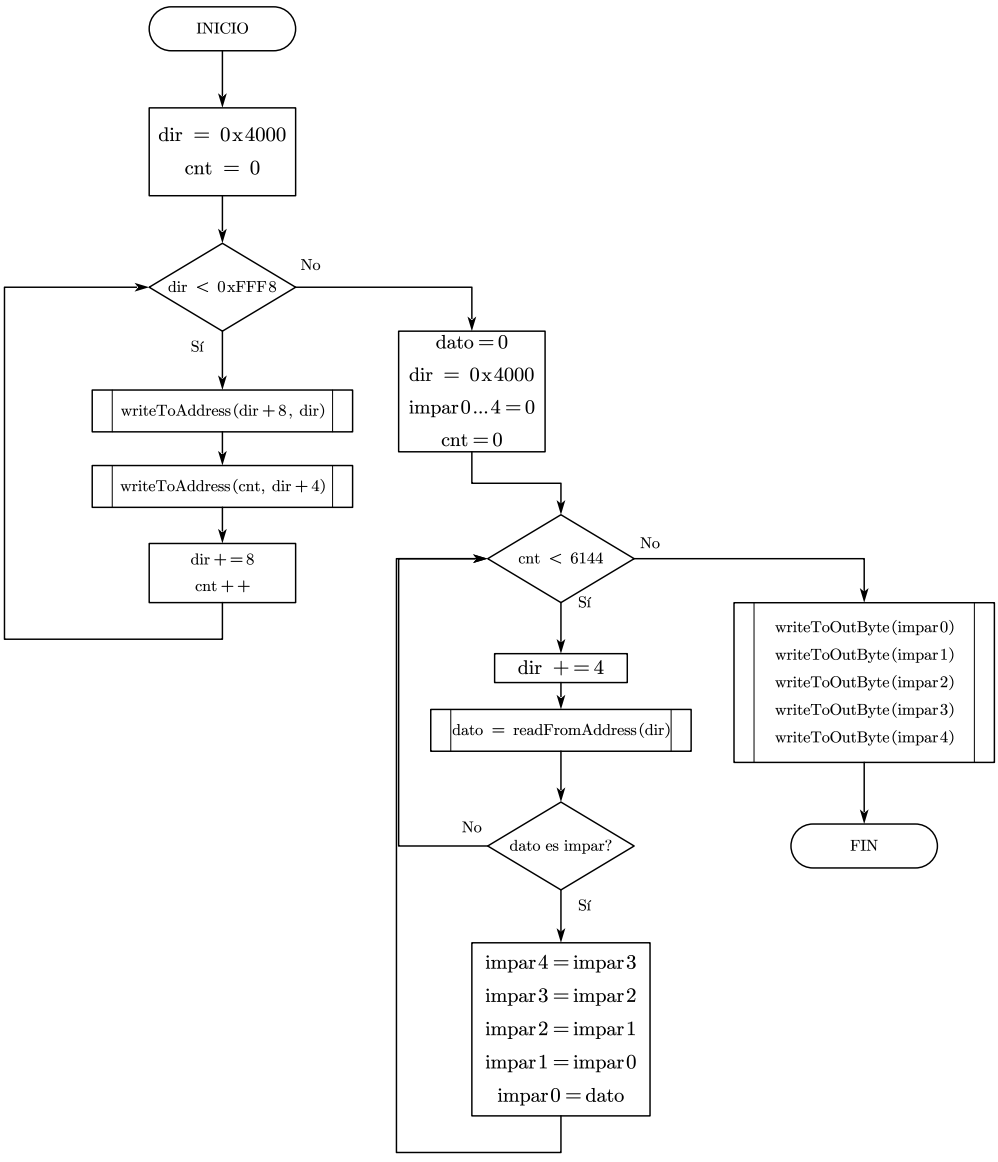


**Figura 2:** Diagrama de flujo del *firmware* a realizar en el Ejercicio 1

El código inicial se muestra a continuación:

```
#include <stdio.h>
#include <stdlib.h>
#include <stdint.h>

#define BASE 0x4000
#define MEMORY_ADD 0x10000000

static void write(uint32_t next_addr, uint32_t index, int addr){
    *((volatile uint32_t *)addr) = next_addr;
    addr = addr + 0x4;
    *((volatile uint32_t *)addr) = index;
}

static uint32_t read(uint32_t addr) {
return *((volatile uint32_t *)addr);
}

void delay(){
    int delay = 50;
    while(delay) delay--;
}

void main(){
    uint32_t address = BASE;
    uint32_t *outbyte = (uint32_t *)MEMORY_ADD;
    uint32_t address_towrite= address+0x8;
    uint32_t address_toread = address+0x4;
    uint32_t data;
    uint32_t odd0 = 0, odd1 = 0, odd2 = 0, odd3 = 0, odd4= 0;

    while(1){
        for(int i = 0; i<6144; i++){
            write(address_towrite, i, address);
            address = address_towrite;
            address_towrite = address+0x8;
        }
        address = BASE;

        for(int i = 0; i < 6144; i++){
            data = read(address+0x4);
            if(!(data%2)){
                odd4 = odd3;
odd3 = odd2;
odd2 = odd1;
odd1 = odd0;
odd0 = data;
            }
            address = read(address);
        }

        *outbyte = odd0;
        delay();
        *outbyte = odd1;
        delay();
        *outbyte = odd2;
        delay();
        *outbyte = odd3;
        delay();
        *outbyte = odd4;
        delay();
    }
}
```

## 4.2. Ejercicio 2

Siguiendo las instrucciones del manejo de la interfaz de memoria PicoRV32 se puede ver que lo necesario para crear el módulo de memoria es en esencia reutilizar el controlador de memoria con la interfaz nativa (ignorando la Look-Ahead) de `system.v`, incluido en el código inicial del laboratorio. Con la salvedad de que se necesitan agregar *delays* en la transferencia

de read/write. Esto se puede realizar con un contador que asigna relojes de lectura/escritura. El código inicial para la construcción del módulo se ve a continuación:

```
module memory #(parameter integer MEM_SIZE = 16384)
               (input clk,
                input resetn,
                input mem_valid,
                input mem_instr,
                input [31:0] mem_addr,
                input [31:0] mem_wdata,
                input [3:0] mem_wstrb,
                output reg mem_ready,
                output reg [31:0] mem_rdata);

    `ifdef SYNTHESIS
    initial $readmemh("../firmware/firmware.hex", mem);
    `else
    initial $readmemh("firmware.hex", mem);
    `endif

    wire clk_r, clk_w;
    reg m_read_en;
    reg [3:0] cnt;

    reg [31:0] mem [0:MEM_SIZE-1];

    // R/W clock generation
    assign clk_r = cnt == 9? 1:0;
    assign clk_w = cnt == 14? 1:0;

    // Counter
    always @(posedge clk) begin
        if (-resetn) begin
            cnt <= 0;
        end
        else begin
            cnt <= cnt + 1;
        end
    end

    // Read transfer
    always @(posedge clk_r) begin
        if (-resetn) begin
            mem_ready <= 0;
            mem_rdata <= 0;
            end else begin
            m_read_en <= 0;
            mem_ready <= mem_valid && !mem_ready && m_read_en;
            mem_rdata <= mem[mem_addr >> 2];
            if (mem_valid && !mem_ready && !mem_wstrb && (mem_addr >> 2) < MEM_SIZE) begin
                m_read_en <= 1;
            end
        end
    end

    // Write transfer
    always @(posedge clk_w) begin
        if (-resetn) begin
            mem_ready <= 0;
            end else begin
            mem_ready <= mem_valid && !mem_ready;
            if (mem_valid && !mem_ready && |mem_wstrb && (mem_addr >> 2) < MEM_SIZE) begin
                if (mem_wstrb[0]) mem[mem_addr >> 2][7: 0]  <= mem_wdata[7: 0];
                if (mem_wstrb[1]) mem[mem_addr >> 2][15: 8] <= mem_wdata[15: 8];
                if (mem_wstrb[2]) mem[mem_addr >> 2][23:16] <= mem_wdata[23:16];
                if (mem_wstrb[3]) mem[mem_addr >> 2][31:24] <= mem_wdata[31:24];
                mem_ready                                   <= 1;
            end
        end
    end
endmodule
```

Ahora, hay que considerar tres cosas en este diseño:

1. Es probable que `system.v` tenga que modificarse para incluir ese módulo de memoria.

2. Se deben realizar pruebas para validar que el módulo funciona como esperado.

### 4.3. Ejercicio 3

El diagrama de bloques con el pinout del cache y su interacción con el *core* y el módulo de memoria del Ejercicio 2 se ve en la Figura 3.

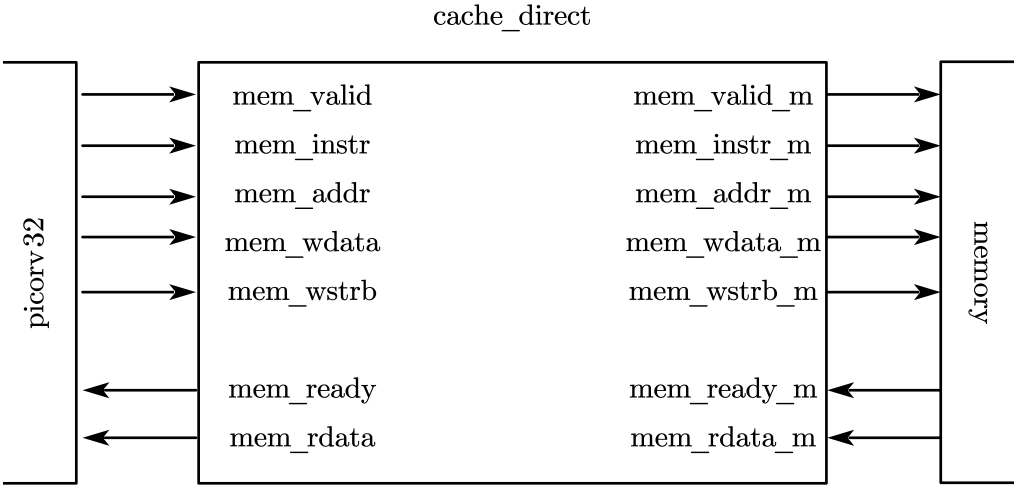


**Figura 3:** Pinout del cache a realizar en el Ejercicio 3

Ahora, el controlador de este módulo se plantea diseñarlo como una máquina de estados siguiendo las indicaciones de la Figura 4 [3]. Se tienen 5 estados principales:

- IDLE: El cache espera a una solicitud lectura/escritura de parte del controlador de memoria.
- TAG CHECK: El cache verifica el tag de la dirección solicitada y revisa sus flags de `valid, dirty` y si es un `hit` o `miss`.
- EVICTION: Si la dirección en el cache es válida y `dirty` y la consulta en el cache resulta en un `miss` se debe actualizar la memoria (según la política write-back) para remover la entrada y reemplazarla en la siguiente etapa.
- REFILL: Se incluye la entrada en el cache. Se puede llegar a este estado de dos maneras, luego de actualizar memoria reemplazada (EVICTION) o luego de la revisión del tag (TAG CHECK) sólo si en este último la entrada era o inválida o no estaba `dirty` pero si era un `miss`. Es decir la memoria principal estaba actualizada con el cache.
- DATA R/W: Se procesa la solicitud R/W del inicio pero desde el cache. Luego de pasar por los estados anteriores o si ya estaba en el cache en primera instancia.

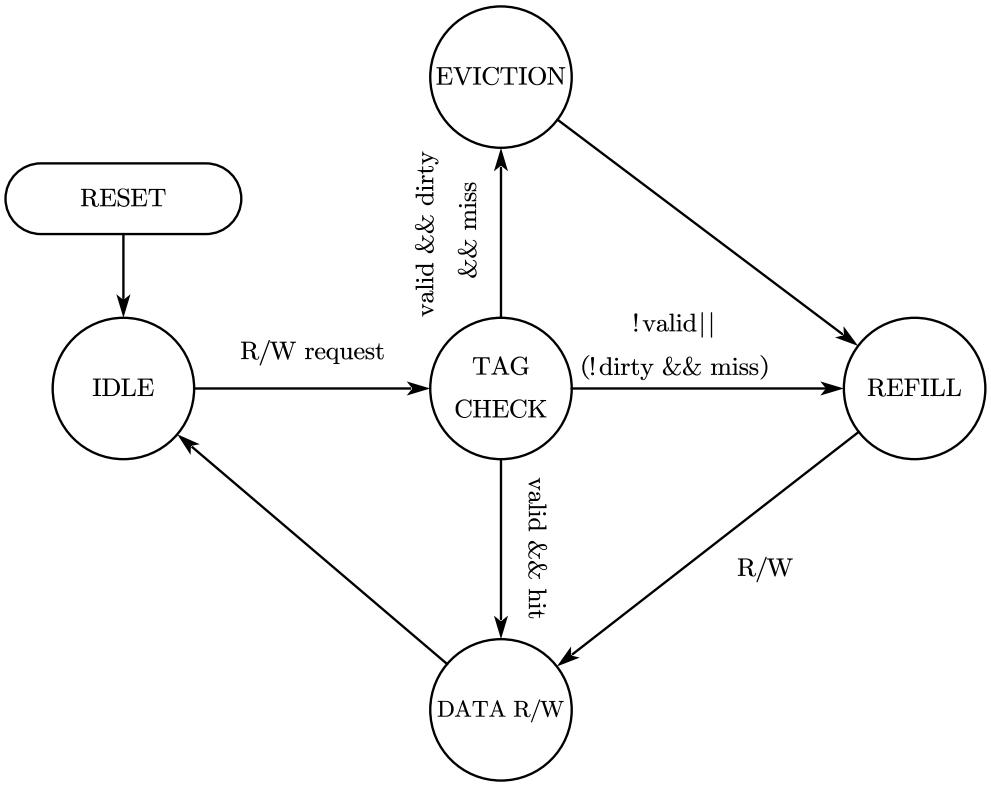


**Figura 4:** Máquina de estados del controlador de cache a realizar en el Ejercicio 3

Las solicitudes R/W de memoria deben seguirse según el estándar impuesto por la interfaz de memoria del PicoRV32.

La memoria del cache en sí debe diseñarse parametrizable pero inicialmente de 1 KB y un tamaño de bloque de 2 palabras, según las especificaciones. Entonces siguiendo la Tabla 1 se requiere el formato de la Figura 5 para entradas de 32 bits.

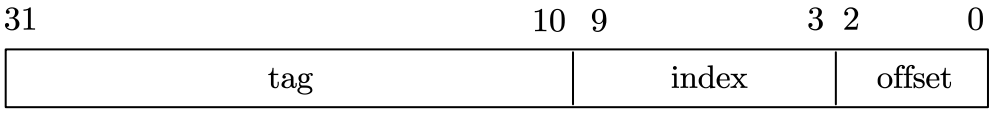


**Figura 5:** Formato de direcciones a implementar con el cache

### 4.4. Ejercicio 4

Se plantea realizar el ejercicio con el modulo anterior instanciado y cambiando los parámetros a los solicitados y anotando los resultados requeridos. Ya que el modulo de cache se realizó de forma parametrizable.

## 5. Observaciones y recomendaciones

- Se recomienda implementar el controlador del módulo de cache como una máquina de estados siguiendo los estados propuestos.
- A la hora de escribir firmware que debe escribir posiciones de memoria específicas, es conveniente realizarlo de manera manual con punteros. El uso de estructuras complica la alocación de la memoria.

## Referencias

[1] Chris Wright Andrew Sloss Dominic Symes. *ARM System Developer's Guide: Designing and Optimizing System Software*. 1.ª ed. The Morgan Kaufmann Series in Computer Architecture and Design. Morgan Kaufmann, 2004. ISBN: 1558608745,9781558608740,9780080490496.

[2] Philip J. Hanlon y col. "The Combinatorics of Cache Misses during Matrix Multiplication". En: *Journal of Computer and System Sciences* 63.1 (2001), págs. 80-126. ISSN: 0022-0000. DOI: https://doi.org/10.1006/jcss.2001.1756. URL: https://www.sciencedirect.com/science/article/pii/S0022000001917568.

[3] Xuan Zhang. *ECE 566A Modern System-on-Chip Design, Lab 3: Cache Design*. 2017. URL: https://classes.engineering.wustl.edu/ese566/Lab/Lab3_description.pdf.