Recuperatorio Parcial Organización del Computador

FIUBA - 2023-2C

- Nombre y apellido:
- Padrón:
- Hojas adicionales entregadas:

Representación de la información

Cada píxel de una imagen se almacena en con el siguiente formato:

```
struct pixel {
    uint8_t r;
    uint8_t g;
    uint8_t b;
    uint8_t a;
} pixel_t;
```

Si dentro de un algoritmo de procesamiento de imágenes se le aplica la siguiente operación a cada píxel:

```
int8_t coefs = {0x3F, 0x01, 0xFF, 0xC0};
pixel_t unPixel = {.r=0x21, .g=0x61, .b=0x82, .a=0x8A};

<tipo mult> mult[4];
<tipo acum> acumulador = 0;

mult[0] = (<tipo cast>)unPixel.r * coefs[0];
mult[1] = (<tipo cast>)unPixel.g * coefs[1];
mult[2] = (<tipo cast>)unPixel.b * coefs[2];
mult[3] = (<tipo cast>)unPixel.a * coefs[3];

for (size_t i = 0; i < 4; i++) {
    acumulador += mult[i];
}
```

a. ¿Qué tipo de datos debería ser `mult` y `acumulador` para que no haya saturación? (Definir los casteos necesarios en las operaciones).

b. Para el ejemplo anterior y utilizando los tipos de datos elegidos en el punto a. ¿Cuál sería el resultado de la operación?

(**Ayuda**: realizar las operaciones en decimal, *no en binario*, justificando adecuadamente los valores)

c. Si el valor de acumulador lo vamos a interpretar como un número entre -1 y 1, y además necesitamos convertirlo a `float` para su posterior uso en otra función:

- Con sus palabras y brevemente (pueden usar pseudocódigo para claridad) expliquen si conviene primero normalizar y luego *castear* o viceversa y porqué. Además, ¿cómo convierte el procesador el valor de acumulador a `float`?.

Microarquitectura

Se tiene una arquitectura con el esquema de la figura, con las siguientes características:

- Tamaño de palabra: 32 bits
- Tamaño de direcciones: 32 bits
- Tamaño de todas las instrucciones: 32 bits

Se asume que todas los módulos internos tardan un clock en actualizar sus salidas.

Se pide:

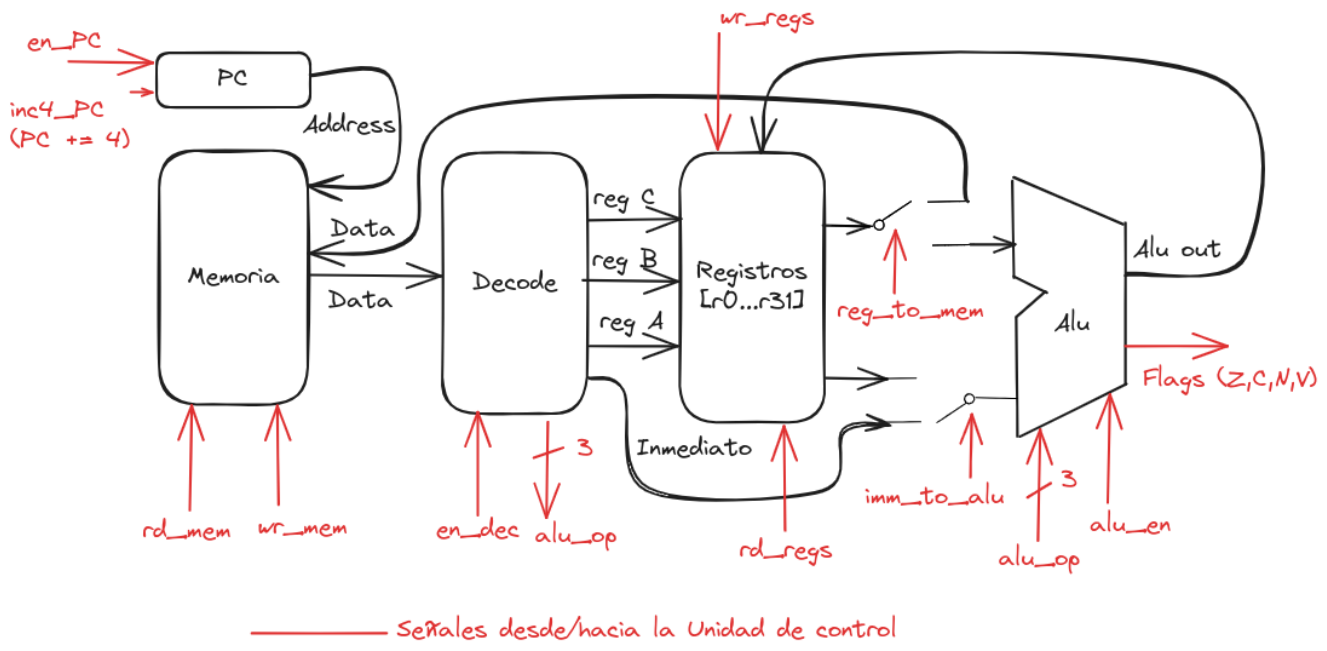

Figure 1: Esquema de la arquitectura

1. Modificar la arquitectura para que el micro pueda ejecutar la siguiente instrucción:

   `sw r4, 8(r5)`

   Dónde el operando de la izquierda es el registro a guardar en memoria, el número fuera de los paréntesis es el offset que se añade a la dirección almacenada en el registro entre paréntesis. (equivalente en `nasm` a hacer `mov [r5+8], r4`).

   (Pueden sobrescribir el diagrama del enunciado)

2. Con la modificación planteada en el punto anterior, escribir la secuencia de microinstrucciones para realizar dicha instrucción. Indicar claramente a qué etapa del ciclo de instrucción se corresponde cada grupo de micro-instrucciones.

ASM

Un sistema de manejo de proyectos mantiene todos los proyectos registrados en una lista enlazada. Cada proyecto tiene un array de tareas a ser completadas y cada tarea posee una dificultad numérica (la dificultad de realizarla). Es decir, cada elemento del array corresponde a la dificultad de la tarea en esa posición.

La dificultad de un proyecto es la suma de las dificultades de sus tareas.

Para mejorar el rendimiento y bajar los costos de alojamiento del sistema se solicita implementar las operaciones más usadas en lenguaje ensamblador. La estructura utilizada en C es:

```
typedef struct s_list {
    struct s_listElem* first;
    struct s_listElem* last;
} list_t;

typedef struct s_listElem {
    struct s_listElem* next;
    uint32_t *tareas;
    uint32_t cantidadTareas;
} listElem_t;
```

Notar que:

- Si `first` es 0 (`NULL`) entonces la lista está vacía y deberá devolverse 0.
- La función `proyecto_mas_dificil` puede recibir un puntero no válido (`NULL`), en tal caso deberá devolver 0.
- Los proyectos que no tienen tareas definidas tienen dificultad 0.
- El campo `next` del último proyecto de la lista es 0 (`NULL`).

Implementar:

```
uint32_t proyecto_mas_dificil(lista_t*)
```

que dada una lista de proyectos devuelve la dificultad del más difícil de ellos.

• La solución debe recorrer la lista pasada como parámetro y encontrar el valor de la dificultad del proyecto más difícil. • La solución no debe modificar la lista pasada como parámetro

Ejemplos:

- `proyecto_mas_dificil([1, 2, 3] -> [98, 99] -> [9] -> NULL) = 197`
- `proyecto_mas_dificil([] -> [] -> [] -> NULL) = 0`
- `proyecto_mas_dificil([3] -> [2, 1] -> [1, 1, 1] -> NULL) = 3`
- `proyecto_mas_dificil([] -> [1, 1, 1] -> [100] -> NULL) = 100`
- `proyecto_mas_dificil(NULL) = 0`

Memoria Cache

Hacer el seguimiento de escrituras y lecturas de la siguiente secuencia de direcciones, en una memoria cache con las siguientes características:

- dirección de 7 bits y memoria direccionable a byte.
- 2 sets
- 4 líneas por set
- 8 bytes por bloque
- 1 byte por lectura/escritura

Suponga que la memoria cache está vacía al inicio y utiliza como política write-back / write-allocate. La política de desalojo es LRU.

		tag	indice set	offset del bloque	
op	address	t	s	b	hit/miss/dirty-miss
W	0x15				
W	0x63				
R	0x2E				
R	0x58				
R	0x10				
W	0x0F				
R	0x7F				
W	0x2A				
R	0x6F				
R	0x1B				
R	0x65				
R	0x29				

¿Cambiaría algo si la política de desalojo fuera FIFO?