Primer Parcial Organización del Computador

FIUBA - 2024-1C

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

Assembly

Implementar una función en assembly que dado un array, aplique una función pasada por parámetro a cada uno de los datos y devuelva un nuevo array. La función a aplicar se pasa por parámetro y tiene la siguiente firma:

```
void* func(void*);
```

Es decir, es una función que recibe un puntero a void y devuelve un puntero a void. Tener en cuenta que: - La función `map` debe aplicar la función `func` a cada uno de los elementos del array - Se supone que la función `func` podrá procesar el tipo de dato que el array contiene. - La función `func` no modifica el array original, sino que devuelve un nuevo dato ya alocado en memoria dinámica.

Se debe devolver un array NUEVO con los datos resultantes de aplicar la función `func` a cada uno de los elementos del array original.

Se puede suponer que los argumentos pasados por parámetros son válidos.

La firma de la función es:

```
array_t* map(array_t* array, void* (*func)(void*))
```

Donde la estructura de array es:

```
typedef struct s_array {
    type_t  type;
    uint8_t size;
    uint8_t capacity;
    void** data;
} array_t;

typedef enum e_type {
    TypeNone = 0,
    TypeInt = 1,
    TypeString = 2,
    TypeCard = 3
} type_t;
```

Memoria Cache

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

- Memoria de 256 bytes y direccionable a byte.
- 4 sets
- 2 vías por set
- 4 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.

op	address	t	s	b	hit/miss/dirty-miss
W	0xC1				
W	0xC5				

op	address	t	s	b	hit/miss/dirty-miss
W	0x57				
W	0x64				
R	0x1A				
W	0xD2				
R	0x6F				
R	0xC2				
R	0x11				
W	0x10				
R	0xC4				
R	0xCE				

¿Cambiaría algo si la política de desalojo fuera FIFO?

Microarquitectura

Se tiene una arquitectura con el siguiente esquema:

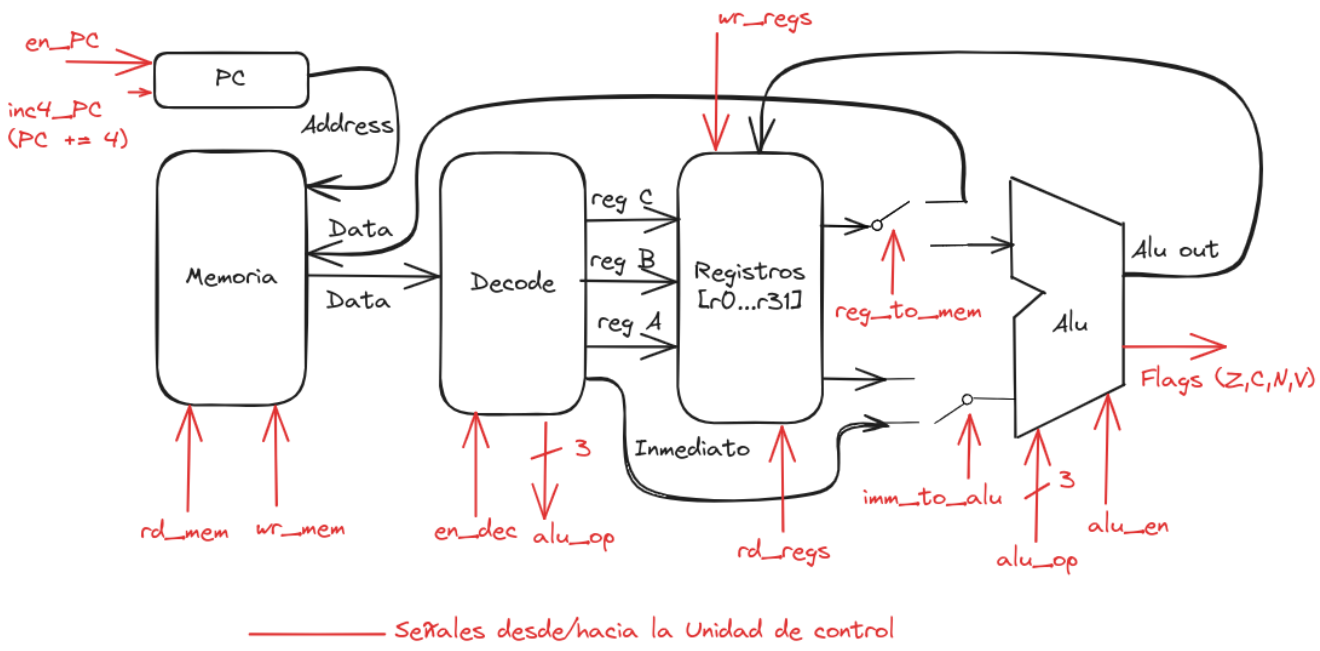

Figure 1: Esquema de la arquitectura

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:

1. Modificar la microarquitectura para que soporte saltos, i.e.:

   ```
   loop:
       ...
       jmp loop
   ```

   Dónde `loop` es un offset de memoria. Es decir, las posiciones de memoria se calculan relativas al `PC` dónde el ensamblador resuelve dicho offset. En otras palabras, el salto se ejecuta sumando (o restando) un número correspondiente al offset al `PC` actual. Dicho offset ya nos viene codificado adecuadamente en la instrucción.

2. Dar la secuencia de microinstrucciones adecuada para ejecutar la instrucción de salto incondicional `jmp`. Indicar a que parte del ciclo de instrucción pertenece cada microinstrucción.

Paginación

Dar las traducciones a memoria física de las siguientes direcciones virtuales, según el esquema de paginación presentado a continuación:

Direcciones:

`0x0300B042; 0x0300BFFA; 0x0200A5F8`

Estructuras de paginación:

Page Directory:

Entry	Page Table
8	0x00010
12	0x00020
15	0x00021

Page Tables:

0x00010:

Entry	Page Table
1	0x00022
2	0x00023
7	0x00024

0x00020:

Entry	Page Table
10	0x00025
11	0x00026
12	0x00027

0x00021:

Entry	Page Table
0	0x00000

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 = {0x7F, 0x01, 0xBD, 0xFF};
pixel_t unPixel = {.r=0x21, .g=0x61, .b=0x82, .a=0xFA};

<tipo mult> mult[4];
<tipo acum> acumulador = 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];
mult[0] = (<tipo cast>)unPixel.r * coefs[0];

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

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? Justificar adecuadamente.

c. Escribir el resultado obtenido en una forma aritmética similar a punto flotante. No se pide representar en bits el número, sino dar una forma aritmética adecuada.