

# Primer Recuperatorio Organización del Computador

FIUBA - 2024-1C

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

## Assembly

Implementar una función en assembly que dado un array de strings, se cree una lista doblemente enlazada insertando todos los elementos del array de manera **alfabética**. La función debe devolver la lista enlazada.

```
list_t* createListFrom(array_t* array);
```

Tener en cuenta que:

- Las estructuras utilizadas son como las vistas en el taller. Las referencias a las mismas se encuentran al final de este enunciado.
- La lista debe contener **copias** de los elementos del array
- Se supone que el array se encuentra bien formado y es válido. Esto no quiere decir que el array no pueda ser vacío de elementos.
- Siempre se debe devolver una lista enlazada, aunque el array esté vacío.
- Pueden reutilizar cualquier función implementada en el taller.
- Si dos elementos son iguales, es indistinto el orden en que se inserten en la lista.
- *Hint*: usar la función `int32_t strcmp(char* a, char* b)` implementada en el taller para comparar strings.

La firma de de la función de comparación es:

```
int32_t strcmp(char* a, char* b)
/*
 * Compara dos strings con un orden alfabético. Retorna:
 * 0 si son iguales
 * 1 si a < b
 * -1 si b < a
 */
```

Donde la estructura de array es:

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

Y la estructura de lista es:

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

```
typedef struct s_listElem {
    struct s_listElem* next;
    int data;
} listElem_t;
```

Y el enum de tipos es:

```
typedef enum e_type {
    TypeNone = 0,
```

```

TypeInt = 1,
TypeString = 2,
TypeCard = 3
} type_t;

```

## Memoria Cache

Complete la siguiente función en C que simula el seguimiento de escrituras y lecturas en una memoria cache de mapeo directo con las siguientes características:

```

#define ADDRESS_SIZE 32      // Tamaño de dirección en bits
#define SIZE_CACHE 4096     // Tamaño de cache en bytes
#define SIZE_BLOCK 64       // Tamaño de bloque en bytes
#define NSETS 64            // Cantidad de sets

typedef struct line{
    int valido;               // 0: invalido, 1: valido
    int dirty;                // 0: no modificado, 1: modificado
    int tag;
} line_t;

typedef struct acceso{
    uint32_t address;
    int operacion;            // 0: lectura, 1: escritura
} acceso_t;

```

La función debe devolver hits, misses y dirty\_misses por parámetro. La función debe tener la siguiente firma:

```

void simular_cache(acceso_t* accesos, int num_accesos, int* hits, int* misses, int* dirty_misses){

```

```

    *hits = 0;
    *misses = 0;
    *dirty_misses = 0;

```

```

    line_t cache[NSETS];

```

```

    for (int i = 0; i < NSETS; i++) {
        cache[i].valido = 0;
        cache[i].dirty = 0;
        cache[i].tag = 0;
    }

```

```

    for (int i = 0; i < num_accesos; i++) {
        // Completar
    }

```

¿Qué problema puede traer esta implementación si el tamaño de la cache es muy grande? ¿Cómo se podría solucionar?

## Microarquitectura

Se tiene una arquitectura con el siguiente esquema:

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.



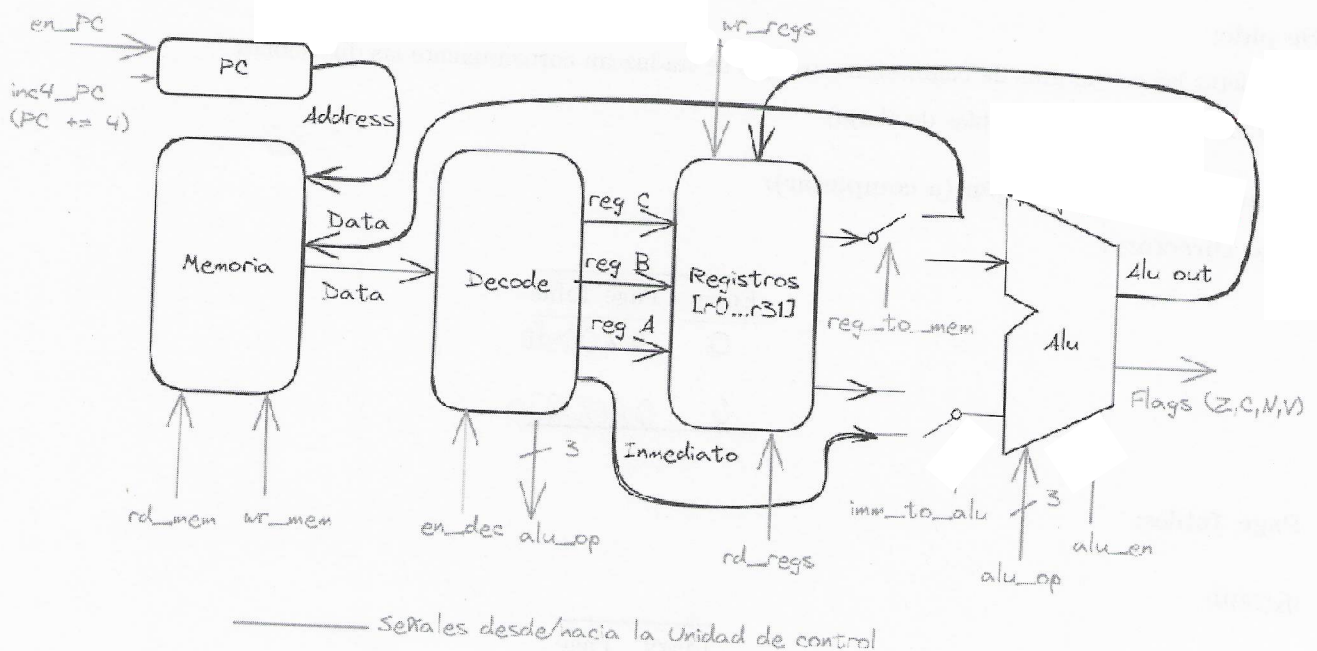


Figure 1: Esquema de la arquitectura

Se pide:

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

```

loop:
...
sub r1, r2
jz 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 un número (que puede ser negativo) correspondiente al offset al PC actual. Dicho offset ya nos viene codificado adecuadamente en la instrucción.

Pueden graficar sobre el esquema entregado.

2. Justificar completa y concisamente las modificaciones introducidas en el punto 1. Aclarar el porqué de cada modificación y dar una idea del funcionamiento general de la arquitectura cuándo ejecuta dicha instrucción.

Para responder pueden usar lenguaje coloquial técnico (es decir describiendo el proceso) o microcódigo en lenguaje de señales como el el taller de OrgaSmall

## Paginación

Estamos diseñando un sistema y necesitamos realizar las siguientes traducciones de direcciones virtuales a físicas:

Direcciones:

Virtual	Física
0x00000001	0x00000001
0x0180F007	0x00002007
0x01901042	0x00003042


**Se pide:**

Completar las estructuras de paginación para que se traduzcan correctamente las direcciones.

(Pueden completar en las tablas de abajo).


**Estructuras de paginación (a completar):**

**Page Directory:**


Entry	Page Table
	

**Page Tables:**

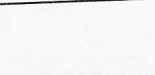
0x00010:

Entry	Page
	

0x00020:

Entry	Page
	

0x00021:

Entry	Page
	

**Representación de la información**

Según el formato que se almacene una imagen, existen las siguientes formas de definir un pixel:

```
typedef struct{
    uint8_t r;
    uint8_t g;
    uint8_t b;
    uint8_t a;
} rgba_t;
```

```
typedef struct{
    uint8_t y;
    uint8_t u;
    uint8_t v
} yuv_t;
```

Para convertir de un formato a otro se usa la siguiente transformación:

```

rgba_t YUV_to_RGBA(yuv_t in) {
    <tipo> Y = in.y;
    <tipo> U = in.u - 128;
    <tipo> V = in.v - 128;
    return (rgba_t) {
        .R = Y + 1.370705 * V,
        .G = Y - 0.698001 * V - 0.337633 * U,
        .B = Y + 1.732446 * U,
        .A = 255
    };
}

```

**Se pide:**

- Definir un tipo adecuado para Y, U, V dentro de la función y explicar claramente los casteos que se van haciendo para obtener el resultado adecuado.
- Dar un ejemplo de conversión de un pixel yuv\_t a rgba\_t mostrando los pasos intermedios y sus conversiones.