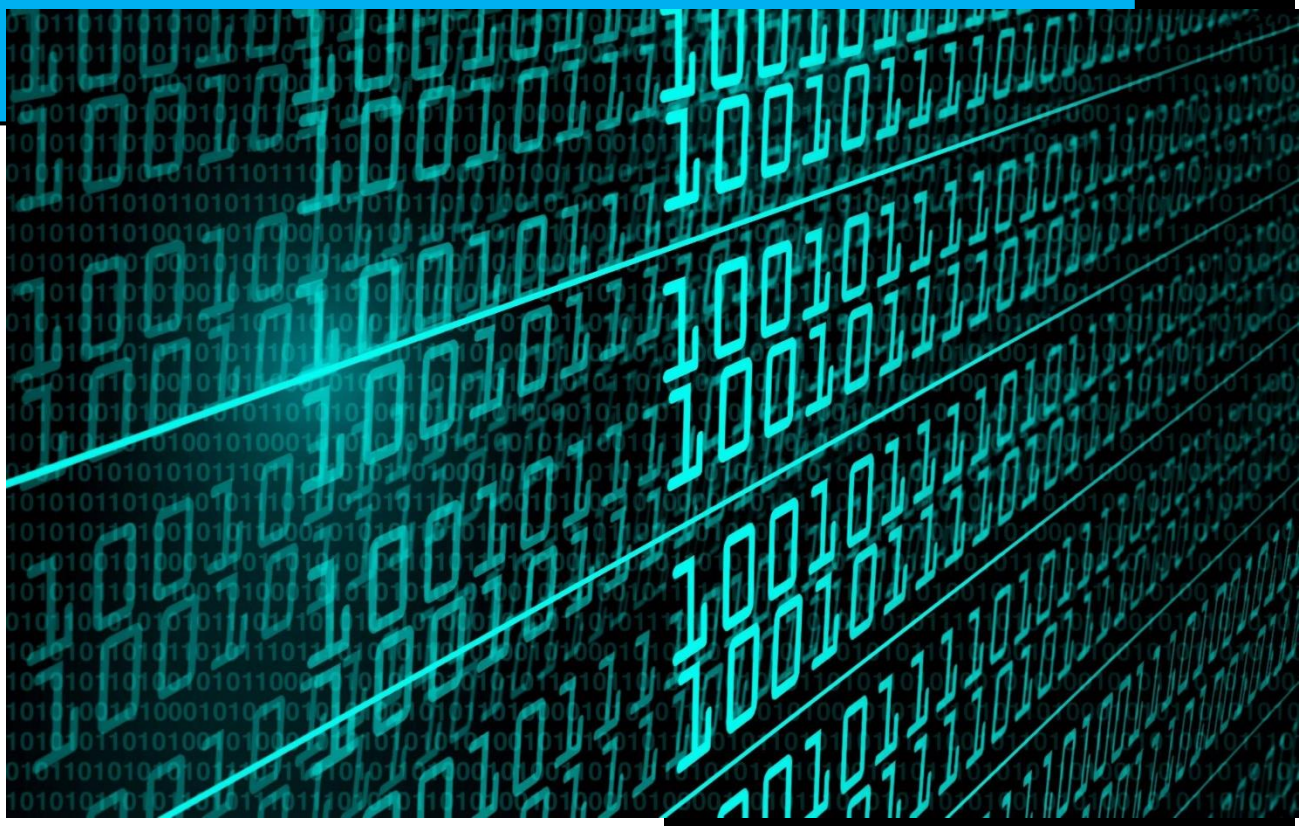
2019

# Práctica 1: Introducción a Lenguaje Ensamblador



**Fernando Bermúdez 100405854, Grupo 80**

**100405854@alumnos.uc3m.es**

**Pablo-Tomás Campos 100406000 , Grupo 80**

**100406000@alumnos.uc3m.es**

**Universidad Carlos III de Madrid**

**17-10-2019**

# Contenido

Decisiones de diseño: 2
Ejercicio 1 2
Método inicializar(int[][] A, int M, int N) 2
Método sumar(int[][] A, int[][] B, int[][] C, int M, int N) 3
Método extraerFila(int[] A, int[][] B, int M, int N, int j) 4
Método masCeros(int[][] A, int[][] B, int M, int N) 5
Ejercicio 2 6
Método extraerValores(float[][] A, int M, int N, int[][] V) 6
Método sumar(float[][] A, float[][] B, float[][] C, int M, int N) 8

# Decisiones de diseño:

La principal decisión de diseño ha sido la creación de una subrutina bajo la de etiqueta *return*, la cual retorna de cualquier función al punto de su llamada. Consideramos que esta es una decisión importante de diseño pues abrevia el código de las funciones a la vez que aumenta su legibilidad, sobre todo cuando estas se cierran de manera prematura.

Cabe comentar que los bucles y otras estructuras de control se han estructurado de cierta manera. Estas siempre comprueban la condición de permanencia en los mismos, en vez de la condición de salida. El código para efectuar tras su salida está justo debajo de los mismos, lo cual implica que se accede al código asociado con el interior del bucle con un salto de etiqueta desde el mismo, la cual comienza con *then_*. Creemos que esto simplifica el código, pues no tenemos que invertir la condición de permanencia para convertirla en condición de salida.

Otra decisión de diseño relevante ha sido mostrar todos los argumentos que recibe una función y su registro asociado al comienzo de esta. Consideramos que esta ha sido una decisión clave a la hora de hacer legible el código.

Cabe mencionar que todos los argumentos que recibe una función son copiados al comienzo de esta en registros temporales, evitando así su manipulación y su uso en comparaciones.

También cabe comentar que hemos optado por igualar por defecto el output de todas las funciones a su valor en caso de fallo (-1). Consideramos que este pequeño gesto facilita y abrevia el retorno de una función en caso de fallo.

Por último, en todos los métodos que tienen alguna dirección de memoria como parámetro de entrada, comprobamos que dichas direcciones no sean ni negativas ni que apunten al segmento de memoria *.text*. En caso de que alguna de esas condiciones se cumpla, devolvemos el valor por defecto de la función y retornamos. De esta forma evitamos que el programa pueda dar un error al ser una dirección de memoria no válida.

# Ejercicio 1

## Método inicializar(int[][] A, int M, int N)

La función inicializar toma como argumentos la dirección de comienzo de una matriz, así como su número de filas y columnas (M y N respectivamente) e iguala todos sus elementos a cero. M y N empiezan a contar en 1. Los argumentos han sido organizados de la siguiente manera:

| Argumento de entrada | Registro de almacenamiento |
|---|---|
| Dirección de comienzo de A | $a0 ➔ $t0 |
| Número de filas de la matriz, M | $a1 ➔ $t1 |
| Número de columnas de la matriz, N | $a2 ➔ $t2 |

A continuación, mostramos la función inicializar implementada en Java:

```
int inicializar(int[][] A, int M, int N) {
//Comprobamos si el número de filas y de columnas de A es mayor que 0. De no
serlo devolvemos 1
	if(M<=0 || N <=0) return -1;
	//Recorremos A, igualando todos sus elementos a 0
	for(int i = 0; i<= M-1; i++) {
		for(int j = 0; j<=N-1; j++) {
			A[i][j] = 0;
		}
	}
	//Devolvemos 0
	return 0;
}
```

Podemos observar como esta función hace uso de la etiqueta return, dado que no es una función que manipule la pila.

A continuación mostramos la batería de pruebas realizadas para comprobar el funcionamiento de la función:

| Datos a introducir | Descripción de la prueba | Resultado esperado | Resultado obtenido |
|---|---|---|---|
| $a0<0 \|\| $a1<0 \|\| $a2<0 | Introducción de argumentos negativos | -1 | -1 |
| 0<$a0<0x00200000 | Dirección de la matriz apunta a .text | -1 | -1 |
| 0x00200000<$a0 && 0<$a1 && 0<$a2 | Introducción de valores válidos | 0 | 0 |

## Método sumar(int[][] A, int[][] B, int[][] C, int M, int N)

Este método toma como parámetros la dirección de comienzo de tres matrices (A, B y C) y las dimensiones de las tres (M y N). La matriz A almacena el resultado de la suma de las matrices B y C. Los argumentos han sido distribuidos de la siguiente manera:

| Argumento de entrada | Registro de almacenamiento |
|---|---|
| Dirección de comienzo de A | $a0 ➔ $t0 |
| Dirección de comienzo de B | $a1 ➔ $t1 |
| Dirección de comienzo de C | $a2 ➔ $t2 |
| Número de filas de la matriz, M | $a3 ➔ $t3 |
| Número de columnas de la matriz, N | ($sp) ➔ $t4 |

A continuación, se muestra el método sumar implementado en Java:

```
int sumar(int[][] A, int[][] B, int[][] C, int M, int N) {
//Comprobamos si el número de filas y de columnas de A es mayor que 0
		if(M<=0 || N <=0) return -1;
	//Recorremos A, B y C, igualando los elementos de A a la suma de los
correspondientes de B y C
		for(int i = 0; i<= M-1; i++) {
			for(int j = 0; j<=N-1; j++) {
				A[i][j] = B[i][j] + C[i][j];
			}
		}
		//Devolvemos 0
		return 0;
	}
```

A continuación, se puede ver la batería de pruebas realizadas para comprobar el funcionamiento del método:

| Datos a introducir | Descripción de la prueba | Resultado esperado | Resultado obtenido |
|---|---|---|---|
| $a0<0 \|\| $a1<0 \|\| $a2<0 \|\| $a3<1 \|\| ($sp)<1 | Introducción de argumentos negativos | -1 | -1 |
| 0<$a0,$a1,$a2<0x00200000 | Dirección de las matrices apuntan a .text | -1 | -1 |
| 0x00200000<$a0,$a1,$a2 && $a3,($sp)>0 | Introducción de valores válidos | 0 | 0 |
| $a0 = $a1 = $a2 && $a3,($sp)>0 | Suma de una matriz con si misma | 0 | 0 |

## Método extraerFila(int[] A, int[][] B, int M, int N, int j)

Este método toma como parámetros la dirección de comienzo de la matriz A en donde se almacenará la fila j de la matriz MxN. M y N empiezan a contar en uno, mientras que j lo hace en 0. Los argumentos han sido distribuidos de la siguiente manera:

| Argumento de entrada | Registro de almacenamiento |
|---|---|
| Dirección de comienzo de A | $a0 ➔ $t0 |
| Dirección de comienzo de B | $a1 ➔ $t1 |
| Número de filas de la matriz, M | $a2 ➔ $t2 |
| Número de columnas de la matriz, N | $a3 ➔ $t3 |
| Fila para extraer, j | ($sp) ➔ $t4 |

A continuación, mostramos la función extraerFila implementada en Java:

```
int extraerFila(int[] A, int[][] B, int M, int N, int j) {
//Comprobamos si el número de filas y de columnas de A es mayor que 0, y que
la fila a copiar (j) existe dentro de B
	if(M<=0 || N<=0 || j>=M || j<=0) return -1;
	//Recorremos la fila escogida de B, copiando sus elementos en A
	for(int i = 0; i<=M-1; i++) {
		A[i]=B[j][i];
	}
	//Devolvemos 0
	return 0;
}
```

A pesar de que la función se podría haber implementado de forma más eficiente y simple en Java, hemos escogido esta implementación pues es la que más se asemeja a la forma de implementarla en ensamblador, como puede observarse en el código.

A continuación mostramos la batería de pruebas realizadas para comprobar el funcionamiento de la función:

| Datos a introducir | Descripción de la prueba | Resultado esperado | Resultado obtenido |
|---|---|---|---|
| $a0<0 \|\| $a1<0 \|\| $a2<0\|\|$a3<0 \|\| $s0<0 | Introducción de argumentos negativos | -1 | -1 |
| 0<$a0<0x00200000 | Dirección de la matriz A apunta a .text | -1 | -1 |
| 0x00200000<$a0 && 0<$a1 && 0<$a2 | Introducción de valores válidos | 0 | 0 |
| 0<$a1<0x00200000 | Dirección de la matriz B apunta a .text | -1 | -1 |
| $a0∈$a1 | Matriz A contenida en matriz B | 0 | 0 |

## Método masCeros(int[][] A, int[][] B, int M, int N)

Este método toma como parámetros las direcciones de inicio de dos matrices (A y B) y las dimensiones de ambas (M y N). Devuelve un valor en función de la matriz con un mayor número de elementos iguales a 0. Los argumentos han sido organizados de la siguiente manera:

| Argumento de entrada | Registro de almacenamiento |
|---|---|
| Dirección de comienzo de A | $a0 ➔ $t0 |
| Dirección de comienzo de B | $a1 ➔ $t1 |
| Número de columnas de las matrices, N | $a2 ➔ $t2 |
| Número de filas de las matrices, M | $a3 ➔ $t3 |

A continuación, se muestra la implementación del método masCeros en Java:

```
int masCeros(int[][] A, int[][] B, int M, int N) {
//Comprobamos si el número de filas y de columnas de A es mayor que 0
            if(M<=0 || N <=0) return -1;
//Llamamos a la función calcular, la cual devuelve el número de apariciones
de un determinado valor dentro de una matriz
            int a = calcular(A, M, N, 0);
            int b = calcular(B, M, N, 0);
            //Comparamos a y b
            if(a==b)return 2;
            if(b>a) return 1;
            //Devolvemos 0
            return 0;

      }
```

A continuación, se puede ver la batería de pruebas realizadas para comprobar que el funcionamiento del método es correcto:

| Datos a introducir | Descripción de la prueba | Resultado esperado | Resultado obtenido |
|---|---|---|---|
| $a0<0 \|\| $a1<0 \|\| $a2<1 \|\| $a3<1 | Introducción de argumentos negativos | -1 | -1 |
| 0<$a0,$a1<0x00200000 | Dirección de las matrices apuntan a .text | -1 | -1 |
| 0x00200000<$a0,$a1 && $a2,$a3>0 | Introducción de valores válidos | 0 | 0 |

# Ejercicio 2

## Método extraerValores(float[][] A, int M, int N, int[][] V)

El método extraerValores toma como argumentos la dirección de una matriz de números flotantes (A), sus dimensiones (M y N) y la dirección de un vector de 6 posiciones (V). El vector V almacena, en cada una de sus posiciones, el número de elementos de la matriz A de la siguiente manera:

- V[0] almacena el número de elementos de la matriz cuyo valor es 0.
- V[1] almacena el número de elementos de la matriz cuyo valor es más infinito.
- V[2] almacena el número de elementos de la matriz cuyo valor es menos infinito.
- V[3] almacena el número de elementos de la matriz con valor NaN.
- V[4] almacena el número de elementos de la matriz que se codifican como no normalizados.
- V[5] almacena el número de elementos de la matriz normalizados.

Los argumentos de entrada han sido organizados de la siguiente manera:

| Registro de entrada | Registro de Almacenamiento |
|---|---|
| Dirección de comienzo de la matriz A | $a0 ➔ $t0 |
| Número de filas de la matriz (M) | $a1 ➔ $t1 |
| Número de columnas de la matriz (N) | $a2➔ $t2 |
| Dirección de comienzo del vector V | $a3➔ $t3 |

Aquí se puede ver el pseudo código del método extraerValores:

```
int extraerValores(float[][] A, int M, int N, int[] V) {
	//Comprobamos si el número de filas y columnas de A es al menos 1
	if(M<=0 || N<=0) return -1;
	//Recorremos A, dejando en V los valores correspondientes
	for(int i=0; i<M; i++) {
		for(int j=0; j<N; j++) {
			float elem = A[i][j];
			int mascara_exponente = 0x7F800000;
			int mascara_mantisa = 0x00700000;
			int mascara_signo = 0x80000000;
			int exponente = (int)(elem * mascara_exponente);
			int mantisa = (int)(elem * mascara_mantisa);
			int signo = (int)(elem * mascara_signo);
			if(exponente == 255) {
				if(mantisa == 0) {
					if(signo == 0) {
					//Si el exponente es 255, la mantisa 0 y el
signo 0 entonces es un -infinito
					V[2] = V[2] + 1;
					}else {
					//Si el exponente es 255, la mantisa 0 y el
signo 1 entonces es un +infinito
					V[1] = V[1] + 1;
					}
				}else {
					//Si el exponente es 255 y la mantisa no es 0
entonces es un NaN
					V[3] = V[3] + 1;
				}
			}else if(exponente == 0) {
				if(mantisa == 0) {
					//Si el exponente y la mantisa son 0 entonces
el elemento es 0
					V[0] = V[0] + 1;
				}else {
					//Si el exponente es 0 y la mantisa no es 0
entonces es un no normalizado
					V[4] = V[4] + 1;
				}
			}else {
				//Si el exponente no es ni 0 ni 255 entonces se
trata de un número normalizado
				V[5] = V[5] + 1;
			}
		}
	}
	//Si todo ha funcionado correctamente, devolvemos 0
	return 0;
}
```

Una vez más, aunque el método se podría haber implementado de forma más eficiente y simple, hemos optado por llevar a cabo esta debido a que es la que más se asemeja a la implementación en ensamblador.

A continuación, se puede observar la batería de pruebas realizadas para comprobar que el método funciona adecuadamente:

| Datos a introducir | Descripción de la prueba | Resultado esperado | Resultado obtenido |
|---|---|---|---|
| $a0<0 \|\| $a1<1 \|\| $a2<1 \|\| $a3<0 | Introducción de argumentos negativos | -1 | -1 |
| 0<$a0,$a3<0x00200000 | Dirección de la matriz y el vector apuntan a .text | -1 | -1 |
| 0x00200000<$a0,$a3 && $a1,$a2>0 | Introducción de valores válidos | 0 | 0 |

## Método sumar(float[][] A, float[][] B, float[][] C, int M, int N)

El método sumar toma como argumentos las direcciones de tres matrices de números flotantes (A, B, C), y las dimensiones de estas (M & N). La matriz A debe almacenar la suma de las matrices B y C. Los argumentos de entrada han sido organizados de la siguiente manera:

| Registro de entrada | Registro de Almacenamiento |
|---|---|
| Dirección de comienzo de A | $a0 ➔$t0 |
| Dirección de comienzo de B | $a1 ➔$t1 |
| Dirección de comienzo de C | $a2➔$t2 |
| Número de filas en las matrices, M | $a3➔$t3 |
| Número de columnas en las matrices, N | ($sp) ➔$t4 |

De esta forma fue implementado el método sumar en Java:

```
int sumar(float[][] A, float[][] B, float[][] C, int M, int N) {
		//Comprobamos si el número de filas y columnas de las matrices es al menos 1
		if(M<=0 || N<=0) return -1;
		//Recorremos A, B y C, dejando en los elementos de A la suma de los elementos de B y C
		for(int i=0; i<M; i++) {
			for(int j=0; j<N; j++) {
				A[i][j] = B[i][j] + C[i][j];
			}
		}
		//Devolvemos 0
		return 0;
	}
```

A continuación mostramos la batería de pruebas realizadas para comprobar el funcionamiento adecuado de la función:

| Datos a introducir | Descripción de la prueba | Resultado esperado | Resultado obtenido |
|---|---|---|---|
| $a0<0 \|\| $a1<0 \|\| $a2<0\|\|$a3<0 \|\| $s0<0 | Introducción de argumentos negativos | -1 | -1 |
| 0<$a0,$a1,$a2<.text | Dirección de las matrices apuntan a .text | -1 | -1 |
| 0<.text<$a0,$a1,$a2 && 0<$a3, ($sp) | Introducción de valores válidos | 0 | 0 |
| $a0 = $a1 = $a2 | Suma de una matriz con si misma | 0 | 0 |