PAUTAS PARA ENSAMBLADOR

# FORMATO DE INSTRUCCIONES:

Se deben poner con este formato

INSTRUCCIÓN OPERANDO, OPERANDO

Se pueden eliminar los espacios tras la coma si es necesario, lo único **importante:**

**instrucción + espacio + operando + coma(espacio si queréis hacerlo más vistoso) + operando**

# REGISTROS

Tenemos 5:

- **AR, BR, CR**: Registros normales

- **RR**: Registro Resultado (Almacena **TODOS** los resultados de operaciones)

- **JR**: Registro de Salto (Se comprueba siempre con la instrucción **JMP**)

- **LR**: Registro de Bucle (Se usa para la instrucción **LOOP** y **SIEMPRE SE REDUCE EN 1 EN CADA ITERACIÓN**)

# PUNTEROS

Tenemos 3:

- **SP**: Puntero de pila (Indica la cantidad de elementos de la pila)

- **CP**: Puntero que código (Indica la posición de la siguiente instrucción a ejecutar)

- **EP**: Puntero de fin (Indica donde acaba el código para decirle a la CPU que pare de ejecutar instrucciones)

# USO DE LA CPU

La CPU está diseñada para que primero cargue el código en memoria de las posiciones más bajas a las más altas. Pensad que no nos conviene utilizar las posiciones más bajas para variables.

Si queremos introducir el final del código en algún momento solo hay que realizar una instrucción JMP a una posición fuera del bloque de código, la instrucción de ejecutarCodigo() se encarga de eso.

Métodos

**cargarEnMemoria(File):** Debe ser la **PRIMERA INSTRUCCIÓN QUE SE EJECUTE EN LA CPU**. Si no hay código, la instrucción **ejecutarCodigo()** no funcionará ya que no tendremos un puntero que defina el final del bloque de código.

**ejecutarCodigo():** Ejecuta todo el código mediante un bucle while, cuando CP alcance el final del código (EP) se acaba la ejecución. Llama a **ejecutarInstruccion()** en bucle, la cual interpreta y ejecuta la instrucción mediante **interpretarLínea()** e incrementa CP.

Instrucciones especiales

**MOV (reg, mem):** El código se almacena dentro de la memoria a partir de la posición 0, Si movemos a memoria en posiciones cercanas a la cantidad de líneas de código de ensamblador tendremos problemas porque podemos borrar líneas de código.

La memoria es un ArrayList, así que, si insertamos elementos en posiciones como, por ejemplo, la 90 todo lo que hay desde el último elemento almacenado hasta el que queremos alcanzar se llenará de 0 para que no haya problemas con ello. Con esto OJO, si se introduce en posiciones de memoria altísimas tardará mucho en almacenarse porque se crearán todas las posiciones de por medio.

**LOOP (mem):** No hace falta decrementar LR, en la instrucción viene incluido

**JMP (mem):** Importante hacer un MOV RR, JR antes de llamar a esta instrucción, si no, dará error. JMP saltará si JR es true, debe almacenarse un valor booleano.