¿Qué necesito para comenzar el diseño de un procesador?
Primero vamos a establecer las instrucciones básicas que va a manejar.

Instrucciones de carga y almacenamiento			
Instr.	Sintaxis	Significado	Tipo
LD	LD ACCx, #n	ACCx = n	I
	LD ACCx, dir	ACCx = [dir]	D
ST	ST ACCx, dir	[dir] = ACCx	D
Instrucciones aritméticas			
ADD	ADD ACCx	ACCx = ACCA+ACCB	R
	ADD ACCx, #n	ACCx = ACCx+n	I
	ADD ACCx, dir	ACCx = ACCx+[dir]	D
SUB	SUB ACCx	ACCx = ACCA - ACCB	R
	SUB ACCx, #n	ACCx = ACCA - n	I
	SUB ACCx, dir	ACCx = ACCA - [dir]	D
Instrucciones de salto incondicional			
B	B #n	PC = #n	I

A estas instrucciones se le conoce como **“Conjunto de instrucciones del lenguaje ensamblador”**. Los lenguajes ensambladores son diferentes de un microprocesador a otro, sin embargo, las funciones que realizan cada uno de ellos son las mismas, es decir, todos los lenguajes ensambladores tienen instrucciones de carga y almacenamiento, aritméticas y lógicas, de salto incondicional, etc.

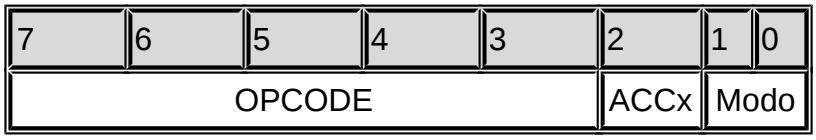
Con estas instrucciones podemos realizar diversos programas. Por ejemplo: hacer un programa que inicie el acumulador A en 7 y lo incremente, el resultado se debe colocar en la dirección de memoria 40H.

```
LD ACCA, #7
LD ACCB, #1
CICLO:
        ADD ACCA
        ST ACCA, 40H
        B CICLO
```

¿Cómo vamos a representar esas instrucciones para que puedan ser ejecutadas?

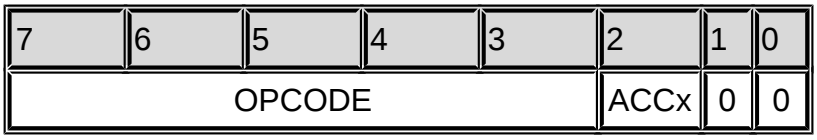
Para poder representarlas necesitamos definir un **formato para las instrucciones**. Este formato debe representar el tipo de instrucción (carga y almacenamiento, aritméticas, salto incondicional, etc.), sus operandos (Acumulador A ó B) y el modo de direccionamiento (tipo R, I, D). El **modo de direccionamiento** nos va a decir cuales son los operandos que puedo manejar con una instrucción del ensamblador, por ejemplo, una instrucción de suma (ADD), puede sumar dos registros, un registro y un número inmediato, un registro y una dirección de memoria (variable).

Un formato que permite representar todo esto, es el siguiente:

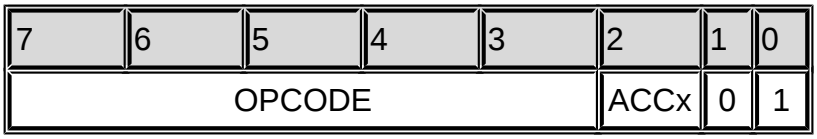


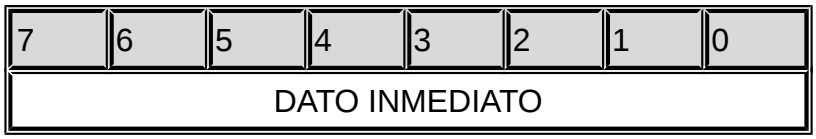
Cada formato de instrucción contiene un código de operación (OPCODE) que permite identificar una instrucción de otra. Este código esta formado por 5 bits que se encuentran de los bits 7 al 3. Con estos 5 bits podemos tener hasta 32 instrucciones diferentes en nuestro procesador. El bit 2 indica el acumulador que va a utilizar la instrucción para colocar el resultado. Bit2 con 1 significa acumulador B, con 0 significa acumulador A. Los bits 1 y 0 permiten especificar el modo de direccionamiento usado en las instrucciones. Con este formato de 8 bits podemos obtener los formatos específicos para cada modo de direccionamiento.

Formato Tipo R (Registro). No existe operando. Los operandos a usar en la instrucción se encuentran en los registros acumuladores (ACCA, ACCB). Esto quiere decir que previamente fueron almacenados los operandos en los registros acumuladores. Solo se usa un byte.

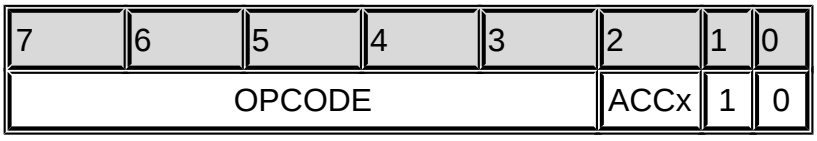


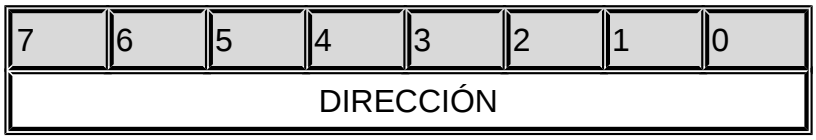
Tipo I (Inmediato). Se usa una byte para el formato de la instrucción y otro para el dato inmediato.





Tipo D (Directo). Se usa un byte para el formato de la instrucción y otro para la dirección de memoria donde se encuentra el dato a usar.





Con estos formatos de instrucción podemos representar las instrucciones con un código binario.

Instrucciones de carga y almacenamiento				
Instrucción	Modos	Significado	Formato de la instrucción	Tipo
LD	LD ACCx, #n	ACCx = n	00000 A 01 nnnn nnnn	I
	LD ACCx, dir	ACCx = [dir]	00000 A 10 dddd dddd	D
ST	ST ACCx, dir	[dir] = ACCx	00001 A 10 dddd dddd	D
Instrucciones aritméticas				
ADD	ADD ACCx	ACCx = ACCA+ACCB	00010 A 00	R
	ADD ACCx, #n	ACCx = ACCx+n	00010 A 01 nnnn nnnn	I
	ADD ACCx, dir	ACCx = ACCx+[dir]	00010 A 10 dddd dddd	D
SUB	SUB ACCx	ACCx = ACCA - ACCB	00011 A 00	R
	SUB ACCx, #n	ACCx = ACCA - n	00011 A 01 nnnn nnnn	I
	SUB ACCx, dir	ACCx = ACCA - [dir]	00011 A 10 dddd dddd	D
Instrucciones de saltos				
B	B #n	PC = #n	00100 x 01 nnnn nnnn	I

Este conjunto de instrucciones y este formato de instrucción dan como resultado la organización mostrada en la figura 1.

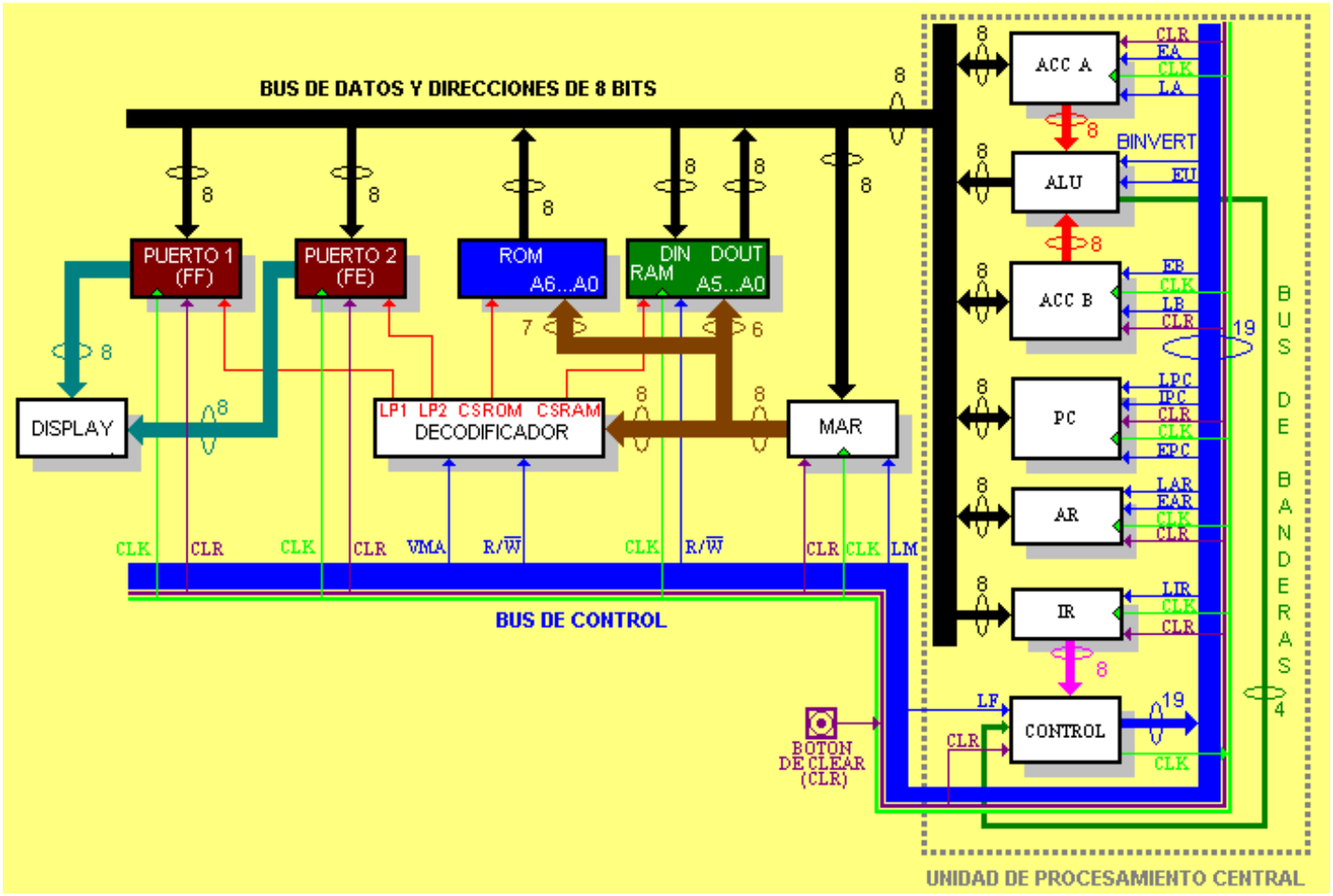


Figura 1 Organización del ESCOMICRO1.

Este microprocesador lo llamamos el ESCOMICRO1 y esta formado de los bloques siguientes:

Acc A, Acc B (Acumuladores A y B). Los registros acumuladores tiene los datos con los que opera directamente la ALU.

ALU (Unidad Aritmética y Lógica). Se encarga de las operaciones básicas de suma y resta. Si Binvert = 0, entonces se realiza la suma de los acumuladores, por el contrario se realiza la resta.

PC (Contador de Programa). El PC tiene la dirección de la siguiente instrucción a ejecutar por el microprocesador. Es un registro capaz de contar desde 0 hasta 255(8 bits). Su función es la de emitir direcciones de memoria.

IR (Registro de Instrucciones). Este registro es donde se tiene que almacenar la instrucción tomada de memoria durante el FETCH.

AR (Registro Auxiliar). Los registros auxiliares, siempre numerosos en cualquier microprocesador. Este registro sirve como depósito momentáneo de los datos para un correcto procesamiento de las instrucciones tipo D.

MAR. (Registro de Direcciones de Memoria). Este registro es necesario sólo cuando nos encontremos en presencia de un microprocesador con un solo BUS para datos y direcciones (compartido). Con este tipo de bus, resulta evidente que no es posible usar el mismo BUS al mismo tiempo para los datos y para las direcciones. El registro MAR resuelve exactamente este problema ya que almacena la dirección que se va acceder por las memorias.

Unidad de Control. Es la encargada de realizar la decodificación de la instrucción, es decir, analizar el código de operación para determinar la instrucción, su acumulador operando y el tipo de instrucción. Una vez decodificada la instrucción genera las microinstrucciones en las fases de ejecución adecuadas para ejecutar cada instrucción del ensamblador.

Decodificador

Decodifica las direcciones de memoria para activar las señales de CS de cada memoria dependiendo de los rangos de direcciones asignadas a la memoria ROM, RAM o periféricos.

Memoria ROM

Esta memoria se le denomina memoria de programa y es la encargada de contener las instrucciones a ejecutar por el microprocesador, esta mapeada en la parte baja del mapa de memoria en el rango de direcciones de 00H-7FH.

Memoria RAM

Esta memoria se le denomina memoria de datos y es la encargada de contener las variables usadas por las instrucciones del ensamblador en el microprocesador, esta mapeada en la parte media baja del mapa de memoria en el rango de direcciones de 80H-BFH.

Puertos

Son dos registros mapeados en la parte alta del mapa de memoria, en las direcciones Ffh y Feh, y están diseñados para manejar dos puertos de 8 bits de salida para el manejo de periféricos.

Con este procesador podemos realizar el análisis de cómo se ejecutan las instrucciones definidas previamente. Podemos escribir el programa de ejemplo anterior, usando los formatos de instrucción definidos, dentro de una memoria de la siguiente forma:

Dirección de memoria	Formato de instrucción	Instrucción
0	00000 0 01	LD ACCA, #7
1	0000 0111	
2	00000 1 01	LD ACCB, #1
3	0000 0001	
4	00010 0 00	CICLO: ADD ACCA
5	00001 0 10	ST ACCA, 40h
6	0010 1000	
7	00100 0 01	B CICLO
8	0000 0100	

La ejecución de cada instrucción de este programa se determina mediante su tabla de microinstrucciones.

Microinstrucciones

Una microinstrucción es cada uno de los comandos que genera la unidad de control, por ciclo de reloj, para poder ejecutar una instrucción del ensamblador. Cada ciclo de reloj se le conoce como fase de ejecución y se le denota con la letra Tn. Las dos primeras fases de ejecución (T0 y T1) de todas las instrucciones definidas en este procesador se le conoce como el FETCH. El FETCH es la búsqueda de la instrucción que se va a ejecutar en la memoria de programa.

Cada microinstrucción la podemos representar en una tabla de la siguiente forma:

Instrucción	Fase	Microinstrucciones
LD ACCA, #n	T0	EPC LM
	T1	VMA RW LIR IPC
	T2	EPC LM
	T3	VMA RW LA IPC
LD ACCB, #n	T0	EPC LM
	T1	VMA RW LIR IPC
	T2	EPC LM
	T3	VMA RW LB IPC
LD ACCA, dir	T0	
	T1	
	T2	
	T3	
	T4	
	T5	
LD ACCB, dir	T0	

	T1	
	T2	
	T3	
	T4	
	T5	
ST ACCA, dir	T0	EPC LM
	T1	VMA RW LIR IPC
	T2	EPC LM
	T3	VMA RW LAR
	T4	EAR LM
	T5	VMA EA IPC
ST ACCB, dir	T0	
	T1	
	T2	
	T3	
	T4	
	T5	
ADD ACCA	T0	EPC LM
	T1	VMA RW LIR IPC
	T2	EU LA
ADD ACCB	T0	
	T1	
	T2	
ADD ACCA, #n	T0	
	T1	
	T2	
	T3	
ADD ACCB, #n	T0	
	T1	
	T2	
	T3	
ADD ACCA, dir	T0	
	T1	
	T2	
	T3	
	T4	
	T5	
ADD ACCB, dir	T0	
	T1	
	T2	
	T3	

	T4	
	T5	
SUB ACCA	T0	
	T1	
	T2	
SUB ACCB	T0	
	T1	
	T2	
SUB ACCA, #n	T0	
	T1	
	T2	
	T3	
SUB ACCB, #n	T0	
	T1	
	T2	
	T3	
SUB ACCA, dir	T0	
	T1	
	T2	
	T3	
	T4	
	T5	
SUB ACCB, dir	T0	
	T1	
	T2	
	T3	
	T4	
	T5	
B #n	T0	EPC LM
	T1	VMA RW LIR IPC
	T2	EPC LM
	T3	VMA RW LPC

Como podemos observar el número de fases de ejecución (T0, ... Tn) depende del tipo de instrucción.

- Las instrucciones tipo R ocupan una fase de ejecución (T2) después del fetch.
- Las instrucciones tipo I ocupan dos fases de ejecución (T2, T3) después del fetch.
- Las instrucciones tipo D ocupan cuatro fases de ejecución (T2, T3, T4, T5) después del fetch.

Las instrucciones tipo D que tiene como operando a una dirección de memoria (variable) son las más costosas en ejecución.

Este Microprocesador tiene una arquitectura llamada **Conjunto de Instrucciones Complejas** (CISC - Complex Instruction Set Computers) y presenta las siguientes características:

- Los **formatos de instrucción son de longitud variable.** Las instrucciones tipo R en este microprocesador tienen un formato de 1 byte, mientras que las instrucciones tipo I y Tipo D ocupan dos bytes.

- Existen instrucciones tipo D, lo cual quiere decir que las instrucciones pueden tener como operandos a direcciones de memoria (variables), estas instrucciones se les denomina **instrucciones complejas** y tardan muchos ciclos de reloj (T0, ..., T5) en ejecutar su tarea por el acceso a memoria. Es decir, **se tienen instrucciones con acceso a memoria**.

- En microprocesadores modernos como INTEL, existen **muchos modos de direccionamiento**.

- **La memoria de programa y datos se encuentran usando el mismo bus de datos y direcciones** por lo que el acceso es secuencial.

- **La unidad de control se hace mas compleja** porque debe manejar distintas fases de ejecución dependiendo del tipo de instrucción.