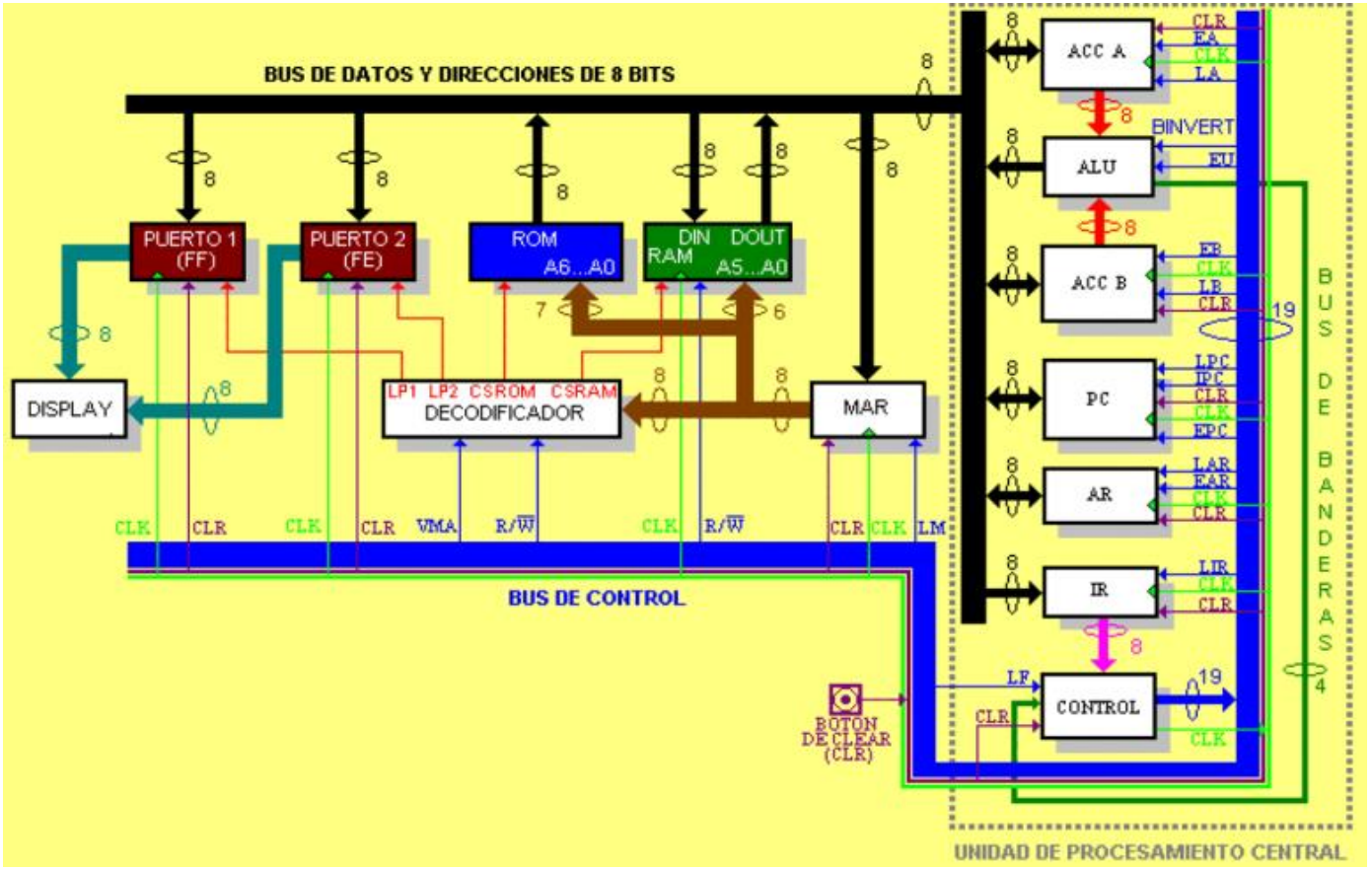
Funcionamiento del microprocesador ESCOMICRO1.

Este microprocesador cuenta con los siguientes bloques:



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 proporcionar direcciones de memoria al registro MAR.

MAR (Registro de Direcciones de Memoria).

El uso de este registro es necesario cuando el procesador tenga un solo BUS para datos y para direcciones (compartido). Este registro almacena la dirección que se va a acceder por las memorias, es decir, asigna las direcciones ya sea en la memoria RAM o en la memoria ROM.

ACC A, ACCB (Acumuladores A y B).

Los registros acumuladores contienen los datos con los que opera directamente la ALU.

ALU (Unidad Aritmética y Lógica).

Se encarga de realizar las operaciones básicas de suma y resta. Si la bandera **BINVERT = 0**, entonces realiza la suma de los acumuladores, por el contrario, se realiza la resta.

IR (Registro de Instrucciones).

En este registro se almacena la instrucción tomada de memoria durante el FETCH. (Sólo lectura)

AR (Registro Auxiliar).

Este registro sirve como depósito momentáneo de los datos para un correcto procesamiento de las instrucciones tipo D. Estos registros son numerosos en cualquier microprocesador.

Bandera VMA.

Válida que la dirección que se coloca en el registro MAR sea una dirección válida.

Unidad de Control

Es la encargada de **decodificar** el formato de la instrucción que se envía al Registro de Instrucciones, es decir, analiza el **código de operación** para determinar la instrucción, el **registro acumulador** que se utilizará y el **tipo de instrucción**. Una vez que decodifica la instrucción, **genera las microinstrucciones** en las fases de ejecución adecuadas para ejecutar cada instrucción del ensamblador.

Memoria ROM.

Esta memoria es denominada como **memoria de programa**, y es la encarga 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.

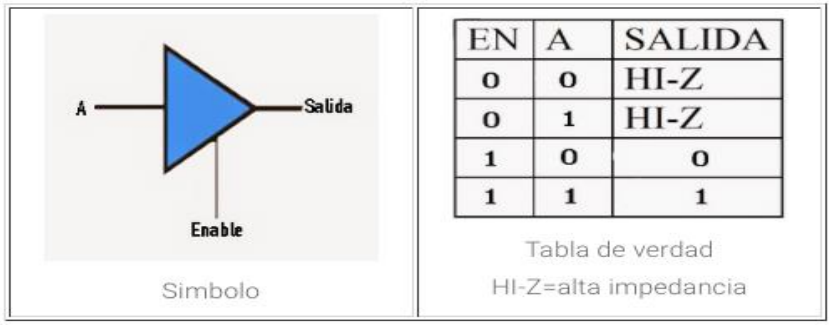
Esta memoria es denominada como **memoria de datos**, y es la encarga de contener las variables usadas por las instrucciones del ensamblador en el microprocesador.

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, están diseñados para manejar dos puertos de 8 bits de salida para el manejo de periféricos.

Al diseñar un microprocesador con un BUS de datos y direcciones compartido puede suceder que en algún momento un bloque del microprocesador intente escribir datos en el BUS de datos y al mismo tiempo otro bloque intente escribir datos en el BUS de direcciones, lo que esto ocasiona es un corto circuito, por lo que, para resolver esto se implementa un **Buffer Triestado** en aquellos bloques del microprocesador que vayan a escribir datos en el BUS compartido. Ya que con esto garantizamos que cuando un bloque vaya a escribir en este BUS compartido los demás bloques se desconecten utilizando el estado de **alta impedancia** (resistencia muy alta/infinita), y de este modo se eviten cortos circuitos.



EN	A	SALIDA
0	0	HI-Z
0	1	HI-Z
1	0	0
1	1	1

Tabla de verdad

HI-Z=alta impedancia

Señales Asíncrona.

Se trata de aquellas señales que dependen del flanco de subida del reloj para su ejecución.

Señales Síncronas.

Se trata de aquellas señales que **no** dependen del flanco de subida del reloj para su ejecución.

Puerto.

Es cada dirección de memoria que se encuentra en el mapa de **Entrada/Salida** de un procesador.

Mapa de memoria.

Es todo el espacio de direcciones que puede generar o manejar un procesador con su bus de direcciones. Generalmente el tamaño del contador de programa **(PC)** determina el BUS de direcciones, para este caso el contador va desde 0 hasta 255 (8 bits), por lo que tenemos un mapa de memoria con tamaño de 256 direcciones.

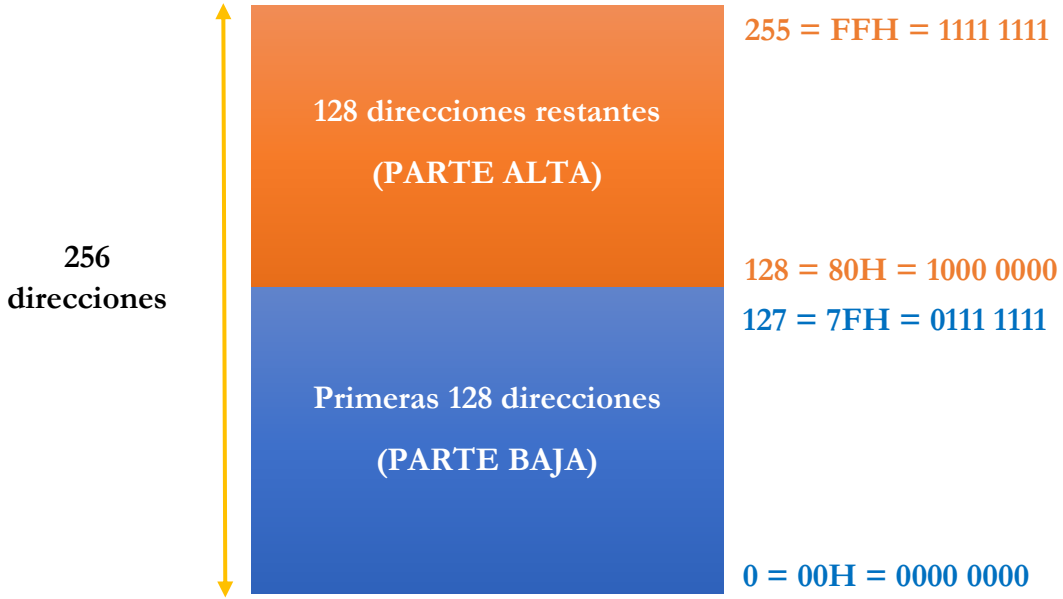
Decodificador.

Es un circuito que se encarga de habilitar una memoria o registro dentro de un rango de direcciones específico.

Ejemplo:

Considere un procesador con **8 bits en el BUS de direcciones** y **8 bits en el BUS de datos**. Usando **memorias con una organización de 128x8**, diseñe un circuito decodificador que permita habilitarla en la parte baja del mapa de memoria.

Como el procesador tiene un bus de direcciones de 8 bits, puede generar $2^8 = 256$ direcciones de memoria, por lo que el mapa de memoria es el siguiente:



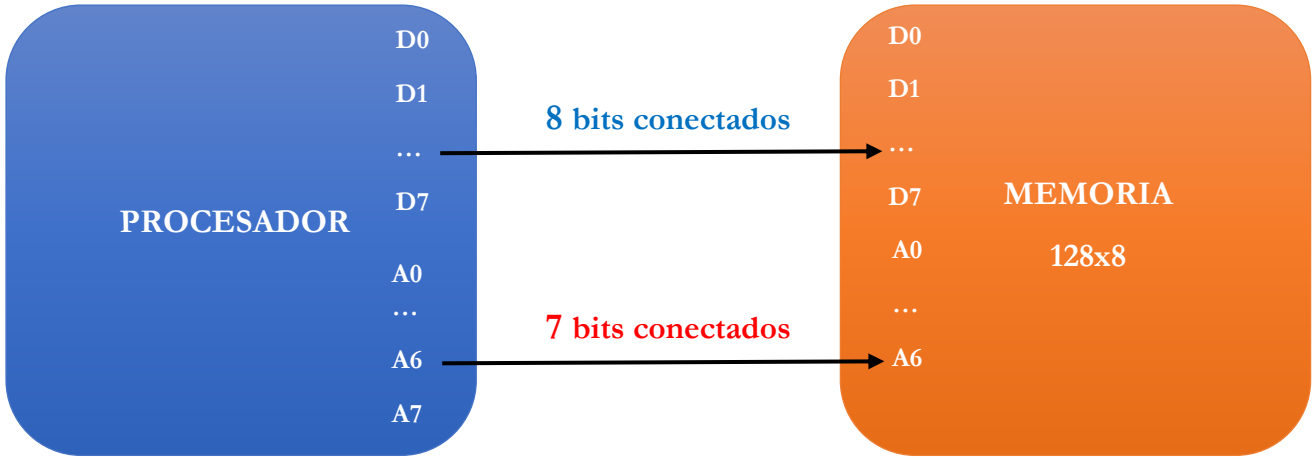
MAPA DE MEMORIA

Para calcular los rangos de la parte alta y de la parte baja, partimos desde la dirección 0000 0000 (**00H**), por lo que para llegar al rango final de la parta baja, basta con que sumemos 127 direcciones más 0111 1111 (**7FH**) a la dirección inicial, para completar un bloque de 128 direcciones.

$$\begin{array}{r} 00H \\ +\ 7FH \\ \hline \end{array}$$

7FH ⟶ Rango Final

La organización de la memoria: 128x8, indica que tenemos $128 = 2^7$ localidades de memoria, es decir, tenemos un **BUS de direcciones de 7 bits**, y el número x8 nos indica que el tenemos un **BUS de direcciones de 8 bits**, es decir, en cada una de las 128 localidades de memoria podemos almacenar un número de 8 bits.



El **BUS de datos** del procesador y de la memoria se conectan directamente, uno a uno.

El **BUS de direcciones de la memoria** se conectará uno a uno con el BUS de direcciones del procesador hasta donde este alcance.

Como podemos ver, en el bus de direcciones del procesador el bit **A7** no esta conectado, debido a que el bus direcciones de la memoria ya está cubierto en su totalidad, por lo que el bit **A7** se utilizará para la construcción del decodificador. Para realizar esto se debe analizar la dirección inicial y final del rango donde queremos mapear la memoria y ver el valor que toma el bit A7 del procesador en ese rango. Para la parte baja, el rango de direcciones es:

MEMORIA POR MAPEAR	Rango	A7	A6	A5	A4	A3	A2	A1	A0
Dirección Inicial	00H	0	0	0	0	0	0	0	0
Dirección Final	7FH	0	0	0	0	0	0	0	0

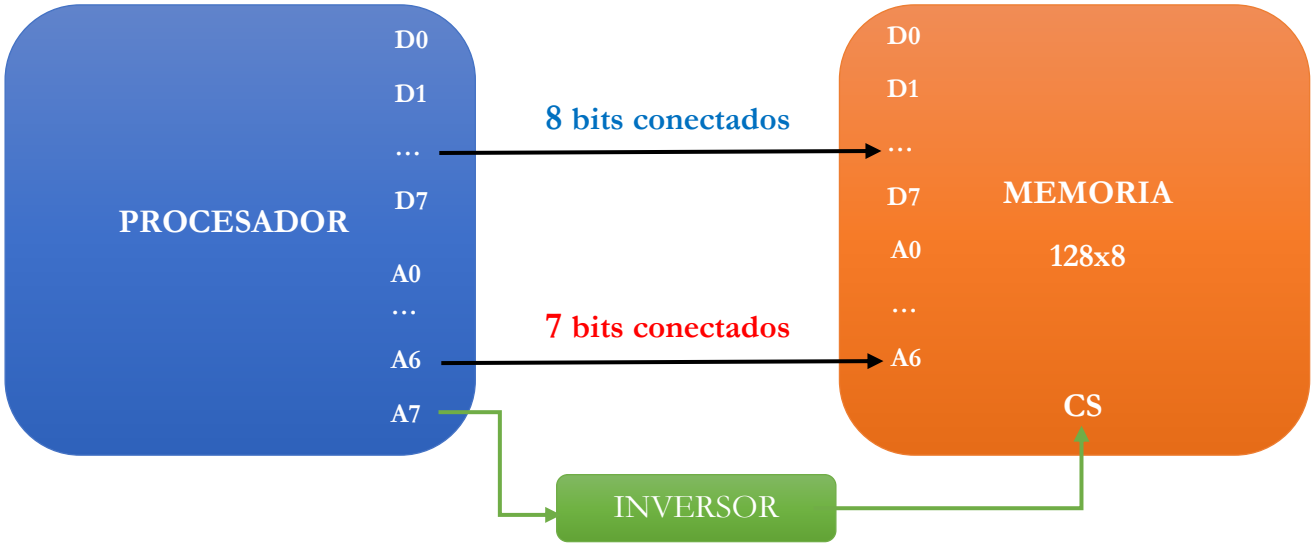
Se puede observar que en la parte baja del mapa de memoria el bit A7 tiene el valor 0. Si queremos que la memoria se habilite en el rango de direcciones de la parte baja del mapa de memoria, debemos conectar un inversor entre el bit A7 del procesador y CS de la memoria, de tal modo que cuando el procesador genere una dirección de memoria que pertenezca a la parte alta, esta entre en modo de alta impedancia (se deshabilite su bus de datos), para que sólo se habilite el bus de la memoria de parte baja. Con el inversor garantizamos que la memoria **sólo funcione en la parte baja** y la parte alta estará sin funcionar en todo momento.

Señal Chip Select (CS modo de bajo consumo) / Chip Enable (CE) de la memoria.

1: El bus de datos de la memoria está disponible para escribir o leer datos de este.

0: El bus de datos de la memoria se pone en alta impedancia.

Por lo tanto, el diseño de nuestra arquitectura completa, procesador, memoria y circuito decodificador es el siguiente.



SEÑALES DE CONTROL

Señales de Control del Registro PC (Contador de Programa)	
Señal de control	**Función**
LPC	**Carga/Almacena direcciones de memoria** en el Contador de Programa
IPC	**Incrementa** el Contador de Programa a la **siguiente dirección de memoria**
EPC	**Extrae/Saca** las direcciones de memoria que tiene almacenadas el Contador de Programa y **las envía a través del bus de direcciones hacia el Registro MAR**

Señales de Control del Registro MAR (Registro de Direcciones de Memoria)	
Señal de control	**Función**
LM	**Carga/Almacena direcciones de memoria** en el Registro MAR

Señales de Control del Registro de Instrucciones (IR)	
Señal de control	**Función**
LIR	**Carga/Almacena** una instrucción que sale la Memoria ROM en este Registro. La instrucción de nuestro programa se almacena como **formato de instrucción.**

Señales de Control del Decodificador	
Señal de control	**Función**
VMA	**1:** Indica que la **dirección de memoria** proporcionada por el registro MAR es correcta y **activa el decodificador**. **0: No habilita** el circuito decodificador.
R/W	**1:** Se habilita la operación de **lectura**. (Memoria ROM) **0:** Se habilita la operación de **escritura**. (Puertos)

Cuando la señal de control **VMA** y **R/W** tienen el valor de **1**, se activa el Chip Select (CS) de la memoria ROM para que se puedan leer datos de esta memoria.

La señal de control **RW** no se coloca en la ecuación de la memoria RAM porque esta es una memoria de lectura y escritura, y la señal de RW se conecta directamente a la memoria RAM, por lo que, no involucra al circuito decodificador del procesador, ya que está conectada directamente a la memoria.

Señales de Control del Acumulador A (ACCA)	
Señal de control	**Función**
EA	**Extrae/Saca** los **datos/valores** que se encuentren almacenada en el Acumulador **A.**
LA	**Carga/Almacena un dato** en el Acumulador **A**.

Señales de Control del Acumulador B (ACCB)	
Señal de control	**Función**
EB	**Extrae/Saca** los **datos/valores** que se encuentren almacenada en el Acumulador **B.**
LB	**Carga/Almacena un dato** en el Acumulador **B**.

Señales de Control de la ALU (Unidad Aritmética y Lógica)	
Señal de control	**Función**
EU	**Extrae/Saca** los **datos** que se encuentran almacenados en la ALU al bus de direcciones.
BINVERT	**1:** Se realiza la operación de **resta.** **0:** Se realiza la operación de **suma. (Por defecto la suma está activada)**

Señales de Control del Registro Auxiliar (AR)	
Señal de control	**Función**
EAR	**Extrae/Saca** los **datos** que se encuentran almacenados en el Registro Auxiliar.
LAR	**Carga/Almacena datos/valores** en este Registro Auxiliar.

El **Registro de Control** es el encargado de activar las 15 señales de control del procesador, esto lo hace utilizando una instrucción de 15 bits.

Nota: El botón de encendido de las computadoras está conectado a la señal **CLR** (clear) de todos los registros del procesador. En nuestras computadoras el primer programa que se ejecuta cuando la encendemos es la BIOS, el cual esta guardado en la memoria de Programa (ROM), y después la BIOS se encarga de ejecutar al Sistema Operativo.

Microinstrucciones.

Una microinstrucción es cada uno de los comandos que genera la Unidad de Control por ciclo de reloj, para poder ejecutar una instrucción en el Ensamblador.

Cada ciclo de reloj se le conoce como **fase de ejecución** y se denota con la letra T_n

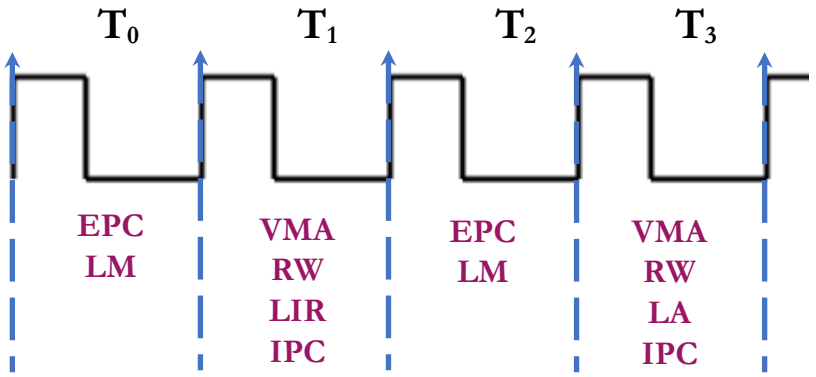
Tabla de microinstrucciones.

En cada una de las fases de nuestra tabla de microinstrucciones (T_0, T_1) todas las señales de control tienen un valor de **0** por defecto, pero cuando el Registro de Control detecta que alguna de estas señales de control se encuentra en la tabla de microinstrucciones, este automáticamente les asigna un valor de **1.**

Se debe realizar una tabla de microinstrucciones por cada una de las instrucciones de nuestro programa.

Construyendo la tabla de microinstrucciones de la instrucción LD ACCA, #7

FASE	LD ACCA, #7
T_0	EPC LM
T_1	VMA RW LIR IPC
T_2	EPC LM
T_3	VMA RW LA IPC



Fase T_0 (EPC LM):

Se carga la dirección de memoria de obtenida del **Contador de Programa** (PC) en el **Registro MAR**, todo eso se realiza en un solo ciclo de reloj, pero la dirección de memoria se carga en el registro MAR en el siguiente flanco de subida.

Fase T_1 (VMA RW LIR IPC):

Con **VMA** indicamos que la dirección de memoria en el Registro MAR es válida y además se activa el decodificador, así mismo se pone en **1** la señal de control **R/W** ya que vamos a leer una instrucción de la memoria ROM (como las señales de control VMA y R/W son **señales asíncronas**, en el momento que reciban el flanco de subida realizarán su operación), una vez VMA y R/W estén en **1**, se activará el **Chip Select (CS)** y se habilitará la parte del mapa de memoria correspondiente a la memoria ROM, una vez leemos de la memoria ROM vemos que se trata de una **instrucción LD,** por lo que tenemos que almacenarla en el Registro de Instrucción mediante la señal de control **LIR (cualquier instrucción LD, ST, ADD, SUB, B que se lea de la ROM se almacenará en el Registro de Instrucción IR como un formato de instrucción '00010 0 00')**, finalmente tenemos que incrementar el contador de programa para que apunte a la siguiente dirección de memoria de nuestro programa, esto lo hacemos con la señal de control **IPC.**

Las fases T_0 y T_1 se conocen como el **FECTH**, en otras palabras, la búsqueda de la instrucción que se ejecutará. La fase de **FETCH** en este procesador que estamos diseñando ocupa dos ciclos de reloj completos.

Fase T_2 (EPC LM):

Nuevamente se carga la dirección de memoria de obtenida del **Contador de Programa** (PC) en el **Registro MAR**, todo eso se realiza en un solo ciclo de reloj, pero la dirección de memoria se carga en el registro MAR en el siguiente flanco de subida.

Fase T_3 (VMA RW LA IPC):

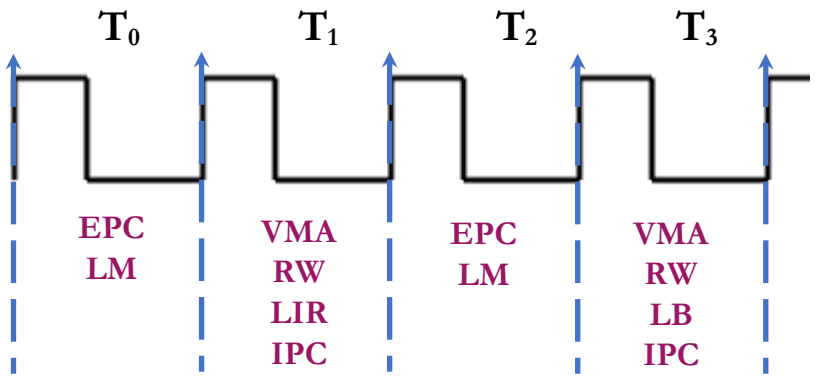
Nuevamente se activa el CS de la memoria ROM y se obtiene el dato, pero esta vez el dato leído de la memoria ROM no es un formato de instrucción, sino que se trata de un número inmediato, en este caso el número 7, entonces al no ser un formato de instrucción este se cargará en el Acumulador que se haya especificado en la instrucción **LD**, para este caso el número inmediato 7 debe cargarse en el Acumulador A, y para poder realizar esto necesitamos activar la señal de control **LA** para poder cargar este número en el Acumulador A, una vez hecho esto, incrementamos el Contador de Programa a la siguiente dirección de memoria de nuestro programa. (**IPC**)

Podemos notar que utilizamos **4 microinstrucciones** para ejecutar una sola instrucción del ensamblador de nuestro programa, es decir, utilizamos 4 ciclos de reloj para ejecutar una instrucción de ensamblador. El desempeño de la velocidad del reloj depende del número de microinstrucciones que utilice para ejecutar una instrucción, **entre menos microinstrucciones necesite para ejecutar una instrucción del ensamblador**, mejor desempeño tendrá.

Construyendo la tabla de microinstrucciones de la instrucción LD ACCB, #1

Al tratarse de una instrucción similar el proceso es el mismo que el que se realizó anteriormente, la única diferencia es que cambia el Registro acumulador que se utilizará.

FASE	LD ACCB, #1
T_0	EPC LM
T_1	VMA RW LIR IPC
T_2	EPC LM
T_3	VMA RW LB IPC



Construyendo la tabla de microinstrucciones de la instrucción ADD ACCA (ACCA = ACCA + ACCB)

FASE	ADD ACCA
T_0	EPC LM
T_1	VMA RW LIR IPC
T_2	EU LA

Fase T_0 y T_1:

Tanto la fase **T_0 y T_1** son las mismas que las dos instrucciones anteriores, debido a que a esas dos fases se les considera el **FETCH,** entonces lo que se hace es obtener la dirección de memoria del Contador de Programa y cargarla en el Registro MAR, seguido de la activación del decodificador mediante **VMA** y activar la opción de lectura en la memoria ROM con **RW**, y una vez que el **formato de instrucción** sale de la memoria ROM este **se carga/almacena** en el Registro de Instrucción, y finalmente tenemos que incrementar el Contador de Programa para que apunte siguiente dirección de memoria de nuestro programa.

Cuando almacenamos el **formato de instrucción** en el Registro de Instrucción, el **Registro de Control** ya sabe cuál es la instrucción que se va a ejecutar, así como los acumuladores que se van a utilizar para realizar la operación (en caso de que se trate de un ADD o SUB).

Cuando el Registro de Instrucción sabe que se trata de una instrucción de suma o resta (**ADD/SUB**) y a su vez sabe cuáles son los acumuladores que se utilizarán para realizar dicha operación, este registro hará que se activen las señales de control de la ALU.

La ALU cuenta con dos buses de direcciones que se conectan directamente con los Registros Acumuladores A y B, por lo que, los datos/valores que se encuentran guardados en estos Registros Acumuladores se pasan directamente a la ALU para que se lleve a cabo la operación que se especificó en el formato de instrucción que se almacenó en el Registro de Instrucción.

La señal de control **BINVERT** le indica a la ALU cuál es la operación que se debe efectuar, pero como el valor por defecto de todas las señales de control es cero, entonces **BINVERT = 0** le indica a la ALU que la operación a realizar es la suma, por lo que, no tenemos que agregar la señal de control **BINVERT** a la tabla de microinstrucciones.

Fase T_2 (EU LA):

Para que el resultado de la operación pueda ser extraído de la ALU hacía el Registro Acumulador en donde se debe almacenar, se activa la señal de control **EU** de la ALU, y a su vez activamos la señal de control **LA** del lado del Registro Acumulador A para que el valor/dato que sale de la ALU se almacene en el acumulador A. Por lo que podemos concluir que **una instrucción de tipo R es más rápida que una instrucción inmediata**, ya que sólo utiliza tres microinstrucciones para ejecutar la instrucción del ensamblador.

Como se ocupa una instrucción más después del FETCH, no es necesario volver a incrementar IPC, debido a que esto se realiza en el FETCH, por lo tanto, hace que el PC ya se localicé en la siguiente dirección de memoria.

Nota. Se acostumbra a incrementar el contador de programa en la última fase que tengamos en nuestra tabla de microinstrucciones

Conceptos Fundamentales.

La **organización de una computadora** digital se refiere a las unidades lógicas que la componen (Unidad Aritmética Lógica (ALU), Contador de Programa (PC), Decodificador, la Unidad de Memoria), así como las funciones que realizan, su operación y la forma en que se relacionan y se comunican unas con otras.

La **arquitectura de la computadora** se enfoca en la forma de construir/diseñar cada una de estas unidades lógicas para que realicen las **funciones especificadas en su organización**, así como la manera en que estás unidades van a comunicarse para interactuar entre ellas.

Ejemplo:

El conjunto de instrucciones de máquina y los diferentes registros que tiene el procesador, la forma de manejar la memoria y los diferentes periféricos de la computadora se definen en la **organización de la computadora**.

Los detalles de la construcción de la unidad aritmética y lógica, y la unidad de control para que la computadora ejecute dichas instrucciones, así como el número de bits utilizados para representar los números enteros, los números reales y las instrucciones de máquina (**formato de la instrucción**), los detalles de la construcción de la unidad de memoria y la unidad de entrada/salida son detalles que se definen en la **arquitectura de la computadora**.

CPU (Unidad de Procesamiento Central).

El CPU de un procesador es donde se encuentra la ALU, los Registros, así como la Unidad de Control.

Unidad de Control.

Se encarga de decodificar la instrucción almacenada en el registro de instrucción, para que así pueda identificar el tipo de instrucción a ejecutar. Después realiza la **generación de microinstrucciones**, usando distintas fases de ejecución para ejecutar cada instrucción del ensamblador.

Esquema de Von Neumann Reducido.

Se maneja este concepto cuando se maneja la Unidad de Memoria y la Unidad de Entrada/Salida (periféricos) como un solo bloque dentro de la arquitectura del procesador, generalmente este bloque es conocido como **Memoria de Programa y Datos.**

Arquitectura de Von Neumann Reducido.

Hablamos de una Arquitectura de Von Neumann reducida cuando los periféricos (Entrada/Salida) y la memoria (RAM/ROM) **se encuentran en el mismo mapa de memoria**, es decir, todo está mapeado en el mismo rango de direcciones, a esto se le conoce como **sistema de Entrada/Salida mapeado**.

Sistema de Entrada/Salida mapeado.

La ventaja que tiene este tipo de sistema es que las instrucciones **ST** y **LD** pueden tener acceso tanto a puertos como a memoria dentro del mapa de memoria, lo que se traduce en un menor número de instrucciones y en una Unidad de Control más sencilla.

Sistema de Entrada/Salida aislado.

En este tipo de sistemas se tienen dos mapas de memoria diferentes, uno se ocupa exclusivamente para mapear memoria y el otro se ocupa exclusivamente para mapear periféricos. La ventaja de tenerlos separados es que el mapa de puertos ya no ocupará espacio en el mapa de memoria, y ahora podemos utilizar todo el mapa exclusivamente para la memoria. (Arquitectura x86 de Intel)

¿Cómo saber que tengo una Arquitectura Aislada?

Esto lo podemos comprobar por el conjunto de instrucciones, ya que ahora tendremos una instrucción para manejar memoria y otra instrucción para manejar puertos en el Ensamblador.

Sistema de E/S Aislado (Arquitectura x86 de Intel)

- Se tiene un mapa de memoria para la memoria.
  Para acceder a memoria en el lenguaje C se utilizan **apuntadores**.
  Para acceder a memoria en el lenguaje Ensamblador se usa la instrucción **MOV**.

- Se tiene un mapa de memoria para E/S (puertos).
  Para acceder al mapa de E/S en el lenguaje C se utilizan las funciones **inportb** y **outportb**
  Para acceder al mapa de E/S en el lenguaje ensamblador se usa la instrucción **IN, OUT**.

En conclusión, podemos saber que se trata de un **sistema de E/S Asilado** cuando en el lenguaje ensamblador existen unas instrucciones para acceder a puertos y otras instrucciones diferentes para acceder a memoria.

Sistema de E/S Mapeado (ESCOMICRO1)

- Se tiene un mapa de memoria para Memoria (RAM/ROM) y E/S (puertos)
  Para acceder a este único mapa de memoria en el lenguaje ensamblador se utilizan las instrucciones ST y LD, ya sea para acceder a memoria o puertos.
  Generalmente los microcontroladores tienen una **E/S mapeada**.

USB (Bus Serial Universal) es un bus de comunicación

PCIE (PCI Express) es un bus de comunicación de la tarjeta madre.

Un Bus no tiene asignados puertos, si no es periférico no podemos asignarle puertos, por esta razón hablamos de un '**conector USB'** y no de un puerto USB.

Arquitectura Harvard

La arquitectura Harvard tiene la característica de que su Bus de Direcciones, su Bus de Datos y el Bus de Control **están separados**.

Arquitectura RISC

Una forma de la organización de la unidad de procesamiento es la denominada con **Conjunto Reducido de Instrucciones** (RISC), esta arquitectura presenta las siguientes características:

Conjunto limitado y simple de instrucciones. Por **simple** podemos entender que se trata de operaciones que tardan pocos ciclos de reloj en ejecutarse.

Instrucciones que realizan operaciones sencillas

- Sumar de dos registros.
- Instrucciones de tipo R

Instrucciones orientadas a los registros con acceso muy limitado a la memoria. Podemos decir que en esta arquitectura no hay instrucciones tipo D, sólo tipo R y tipo I y solamente las instrucciones **LOAD** y **STORE** van a acceder a memoria.

Modos limitados de direccionamiento. Muchas computadoras de tipo RISC ofrecen un solo modo para direccionar la memoria, generalmente un direccionamiento directo o indirecto de registros con un desplazamiento. Particularmente tienen 3 modos de direccionamiento, el tipo R, tipo I y el tipo J.

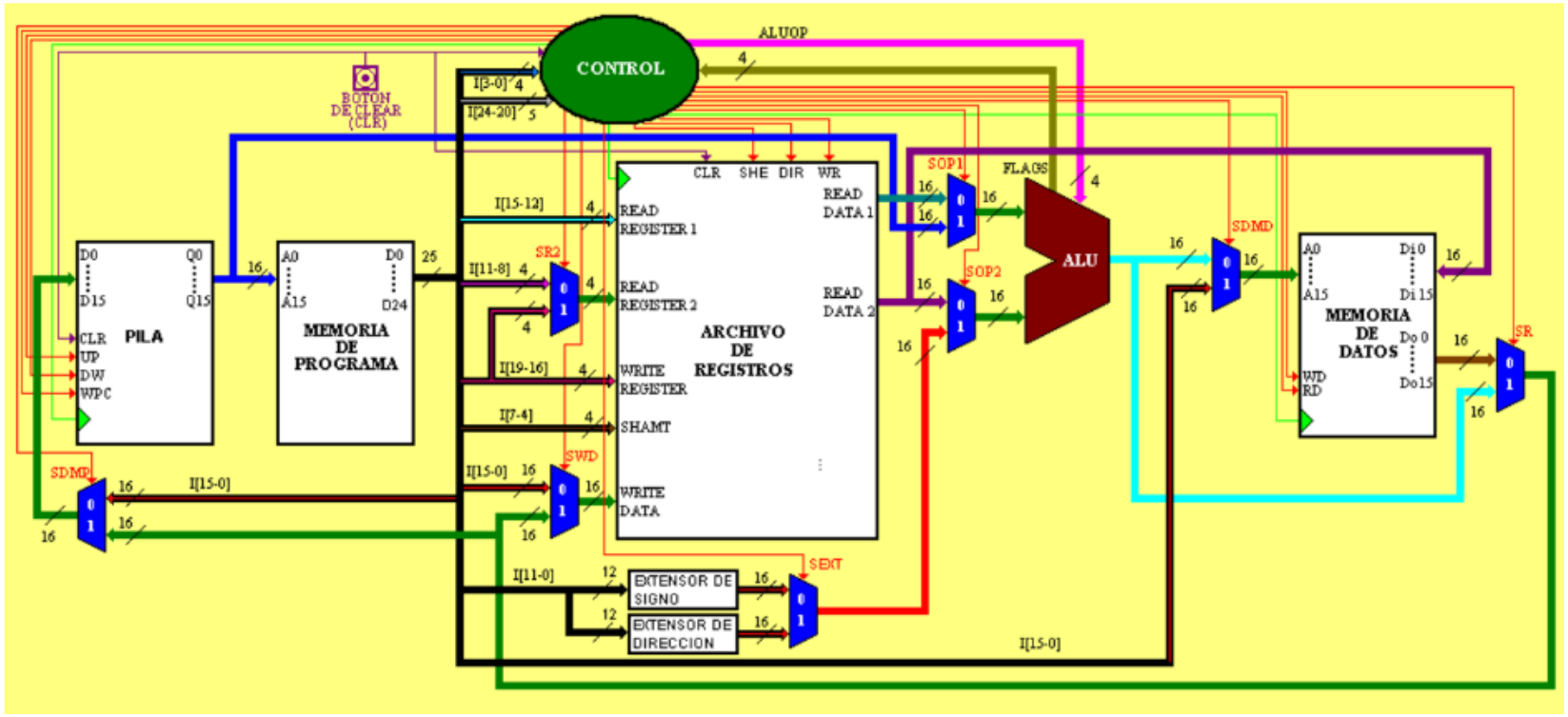
Un gran banco de registros. Manejamos varios registros en el procesador para que podamos guardar la mayor cantidad de datos dentro de ellos y evitemos tener que acceder a la memoria para obtenerlos.

Palabra de la instrucción con extensión y formatos fijos. El formato de instrucción es el mismo para todas las instrucciones. En la Arquitectura CISC el formato de instrucción es variable, ya que algunas instrucciones ocupan un byte y otras 2 bytes.

Microprocesador RISC de 16 bits.

Microprocesador de arquitectura RISC de 16 bits tipo MIPS (Microprocessor without Interlocked Pipeline Stages). La arquitectura MIPS es la base de los procesadores superescalares actuales, por lo que su estudio es fundamental para entender las arquitecturas de cómputo modernas. A este microprocesador le llamamos **ESCOMIPS**, el cual cuenta con una organización de seis componentes y cuenta con las siguientes características:

- Formato de instrucción de 25 bits para todas las instrucciones. Los formatos son tipo R, I y J.
- Cada instrucción se ejecuta en un ciclo de reloj.
- Archivo de 16 registros de trabajo; es el banco de registros del procesador.
- Pila en hardware de 8 niveles.
- Memoria de programa y memoria de datos separada, es decir, **Arquitectura Harvard**. El contador de programa puede direccionar hasta 64kwords. En memoria de datos se puede direccionar hasta 64kwords+2kwords usando brincos relativos.
- Ejecución de brincos condicionales en un solo ciclo de reloj.



Podemos ver que la **PILA** está conectada directamente con la Memoria de Programa, esto se debe a que el Contador de Programa se encuentra dentro de la PILA, pero al ser una pila podemos tener hasta 16 Contadores de Programa, lo cual nos va a permitir realizar **llamadas a subrutinas**, es decir, manejar funciones.

En este microprocesador tenemos definidos los siguientes bloques funcionales:

Pila. Este bloque funcional contiene 8 registros Contadores de Programa (PC's). Un registro contador de programa es aquel que contiene la dirección de la instrucción a ejecutarse por el microprocesador. Al tener un conjunto de 8 registros podemos implementar lo que conoce como PILA en Hardware.

Memoria de programa. Este bloque funcional guarda todas las instrucciones del programa a ejecutarse por el microprocesador. Tiene una organización de 64kx25. (Bus de direcciones de 16 bits con un bus de datos de 25 bits, de acuerdo con el formato de instrucción)

Memoria de datos. Este bloque funcional guarda todos los datos o variables del programa a ejecutarse por el microprocesador. Tiene una organización de 64kx16. (Bus de direcciones de 16 bits con un bus de datos de 16 bits).

Archivo de registros. Este bloque funcional contiene 16 registros de 16 bits. Estos registros guardan los operandos que necesitan las instrucciones del procesador para ser ejecutadas.

ALU. Este bloque funcional está destinado a realizar las operaciones aritméticas y lógicas del microprocesador, es decir, es la unidad que hace todo el procesamiento. Esta unidad además de realizar las operaciones aritméticas y lógicas. También realiza el cálculo de las direcciones para el manejo de bloques de datos o arreglos en la memoria de datos.

Unidad de control. Este bloque funcional es el "cerebro" del microprocesador. Este bloque realiza la decodificación de los códigos de operación y de los códigos de función para poder identificar la instrucción que se va a ejecutar y su tipo (Tipo I, Tipo R y Tipo J). Una vez identificada la instrucción, la unidad de control activa o no cada una de las señales de control de todo el microprocesador

Nota. Cuando hablamos de un Sistema Operativo o un procesador de una determinada cantidad de bits, se refiere como tal

Diseño de la ALU

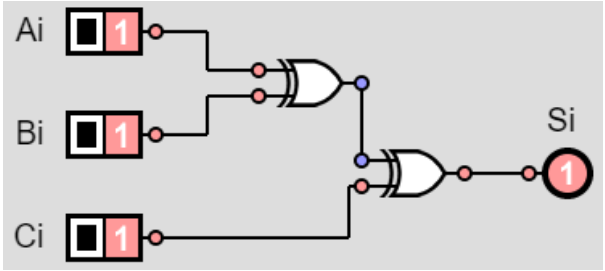
Para comenzar con el diseño de la ALU, comenzaremos diseñando un circuito sumador de 3 bits, en donde las entradas serán **A_i**, **B_i**, **C_i**, y como salidas tendremos lo que se conoce como la suma **S_i** y el acarreo siguiente para el **siguiente bit más significativo**, el acarreo lo denotamos como **C_{i+1}**

donde **Ci** es el acarreo, estás serán las entradas del circuito.

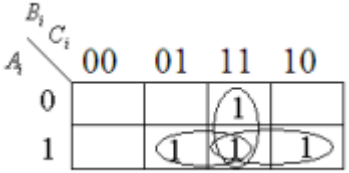
A_i	B_i	C_i	C_{i+1}	S_i
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Una vez obtuvimos la tabla, procedemos a generar las ecuaciones para nuestras salidas, para **S_i** basta con utilizar la compuerta lógica que nos permite realizar una suma binaria, la cual es la **XOR**, de tal modo que la ecuación resultante es:

$\boldsymbol{S_i} = A_i \oplus B_i \oplus C_i$, entonces nuestro circuito lógico resultante es el siguiente:

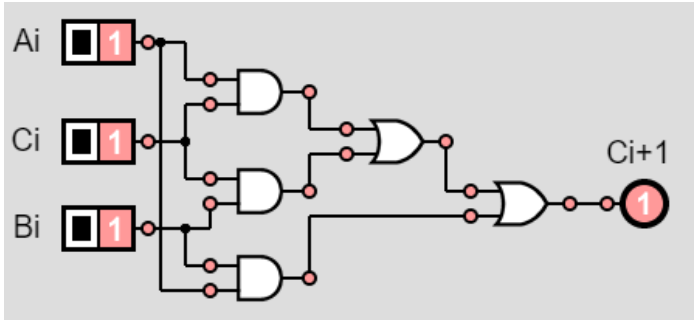


Para obtener la ecuación resultante del acarreo siguiente **C_{i+1}** (el cual le asigna un valor al bit más significativo), se obtiene realizando un Mapa de Karnaugh, el cual se construirá con los mini-términos de la tabla de verdad, es decir, donde el valor de **C_{i+1}** sea igual a uno.



Una que hayamos realizado la simplificación obtenemos la siguiente ecuación para el acarreo siguiente:

$\boldsymbol{C_{i+1}} = A_i\,C_i + B_i\,C_i + A_i\,B_i$, entonces nuestro circuito lógico resultante es el siguiente:



Retardo de Propagación: Un retardo de propagación son los niveles que tenemos de nuestro circuito lógico, para el circuito de C_{i+1}, tenemos 3 retardos de propagación, ya que el primer nivel esta dado por compuertas **AND** que tenemos, el segundo nivel es la compuerta **OR**, y finalmente tenemos otra compuerta **OR** que nos devuelve el valor final de nuestro acarreo siguiente. Cada retardo de propagación se denota como **td**.