13. INTRODUCCIÓN A LOS SISTEMAS MICROPROGRAMABLES

I TEORÍA

13.1 INTRODUCCIÓN

En el presente tema no nos basaremos en ningún dispositivo microprogramable específico, sino que se verá una visión generalizada con la que se darán a conocer múltiples conceptos que, por lo general, son comunes a dichos dispositivos.

13.1.1 LÓGICA CABLEADA Y LÓGICA PROGRAMADA

Hasta la aparición del circuito microprogramable (CμP), el diseño de los automatismos se realizaba mediante lógica cableada, esto es, mediante la unión de distintos módulos entre sí. En la actualidad tres tecnologías permiten realizar los automatismos cableados:

- Relés electromagnéticos.
- Módulos lógicos neumáticos.
- Tarjetas o módulos electrónicos.

El enorme desarrollo de la tecnología electrónica permitió la alta integración de componentes, lo que trajo consigo la llegada del CμP, elemento clave en todo automatismo programado. Hoy en día podemos encontrarnos con tres sistemas programados diferentes:

- Tarjetas electrónicas estándares y específicas.
- Micro y mini ordenadores.
- Autómatas programables.

A la hora de diseñar un automatismo, la elección entre sistema cableado o sistema programado, dependerá en general de su complejidad, de la cantidad a realizar y, fundamentalmente, de su costo.

Las principales ventajas de un sistema programado son las siguientes:

- Reducción del hardware y, por lo tanto, mayor fiabilidad del sistema.
- Disminución del coste de materiales, mano de obra y mantenimiento.
- Realización fácil y rápida de cambios o adaptaciones cambiando el programa de instrucciones y sin afectar, por lo general, la estructura del sistema.
- Reducción del tiempo de diseño.

13.1.2 CONCEPTO DE CIRCUITO MICROPROGRAMABLE

En general, entenderemos por circuito microprogramable un circuito integrado a gran escala (LSI) o muy alta escala (VLSI) para el control, mediante lógica programada, de sistemas electrónicos. En la práctica nos encontraremos con dos tipos de CμP's: el microprocesador (μP) y el microcontrolador (μC).

En la década de los 80 comienza la ruptura entre la evolución tecnológica de los microprocesadores y la de los microcontroladores, ya que los primeros han ido incorporando cada vez más y mejores capacidades par las aplicaciones en donde se requiere el manejo de grandes volúmenes de información, mientras que por otro lado, los segundos han incorporado más capacidades que les permiten la interacción con el mundo físico en tiempo real, además de mejores desempeños en ambientes de tipo industrial.

Las diferencias y analogías entre ambos dispositivos microprogramables, µP y µC, las iremos viendo en el desarrollo de este tema.

13.1.3 CARACTERÍSTICAS

Un CµP se diferencia de otro por un conjunto de características y algunos aspectos internos de su arquitectura. A continuación se describen las más generales:

Comunes	• **Velocidad**: Determina el número de operaciones por segundo que puede realizar el circuito. Se suele medir en MHz o en GHz. • **Juego de instrucciones**: Número de instrucciones que puede ejecutar el dispositivo. • **Arquitectura interna**: Se refiere a los bloques internos y su interconexión. • **Memoria**: Tipo y cantidad de memoria que posee.
µP	• **`Longitud:`** Número de bits del bus de datos. Las longitudes más comunes son las de 8, 16, 32 y 64 bits. • **Direccionamiento de memoria**: Número de bits del bus de direcciones. Así, diremos que un µP puede direccionar 64 K de memoria si su bus de datos posee 16 líneas (2^{16} = 65.536 = 64 K).
µC	• **Puertos**: Se refiere al número de terminales de entrada/salida.

13.1.4 INTERRUPCIONES

Por interrupción entenderemos una señal de control que obliga al CµP a parar el proceso que estaba realizando para realizar otro proceso al que se le suele denominar subrutina de atención.

Los CµP's disponen generalmente de dos modos de interrupción: **enmascarable** y **no enmascarable**. Una interrupción es enmascarable cuando su aceptación está condicionada a la situación de un bit (el I) del registro de estado. La interrupción no enmascarable, en cambio, es incondicional, aceptándose siempre.

La interrupción enmascarable es de menos prioridad que la no enmascarable, pero es muy flexible. Este tipo de interrupción se puede obtener con diferentes modos operativos, que se describen a continuación:

- Cuando el sistema reconoce la interrupción activa el periférico que la ha solicitado, el cual situará en el bus de datos una instrucción relativa a la dirección de la memoria ROM o EPROM, donde se encuentra el inicio de la subrutina de atención.
- Ante la petición y posterior aceptación de la interrupción, el contador de programa salta a una posición de memoria concreta, que corresponde al inicio de la subrutina de atención.
- El periférico que solicita la interrupción deposita en el bus de direcciones el octeto menos significativo, denominado **vector**, de la dirección de la subrutina de atención, mientras que un registro interno sitúa el octeto más significativo. Con este tipo de interrupción, propia de los µP's, se pueden controlar hasta 256 periféricos, identificándose cada uno de ellos por su vector. Este modo es el denominado interrupción vectoriada.

La interrupción no enmascarable se pide a través del terminal de interrupción no enmascarable (NMI) del CµP. Cuando se acepta, el contador de programa se carga con una posición definida de la ROM (o EPROM), en la que se inicia la subrutina.

13.2 EL MICROPROCESADOR

El µP, también llamado CPU (*Central Proces Unit*), suele tener forma de cuadrado o rectángulo negro, y va o bien sobre un elemento llamado zócalo (*socket*) que se suelda en una placa o metido dentro de una especie de cartucho con un sistema de conexión a la mencionada placa.

En este punto veremos una arquitectura interna básica y las características de un sistema basado en µP

13.2.1 ARQUITECTURA INTERNA

Un µP se diseña con una arquitectura que le permita efectuar las funciones de búsqueda y ejecución de instrucciones.

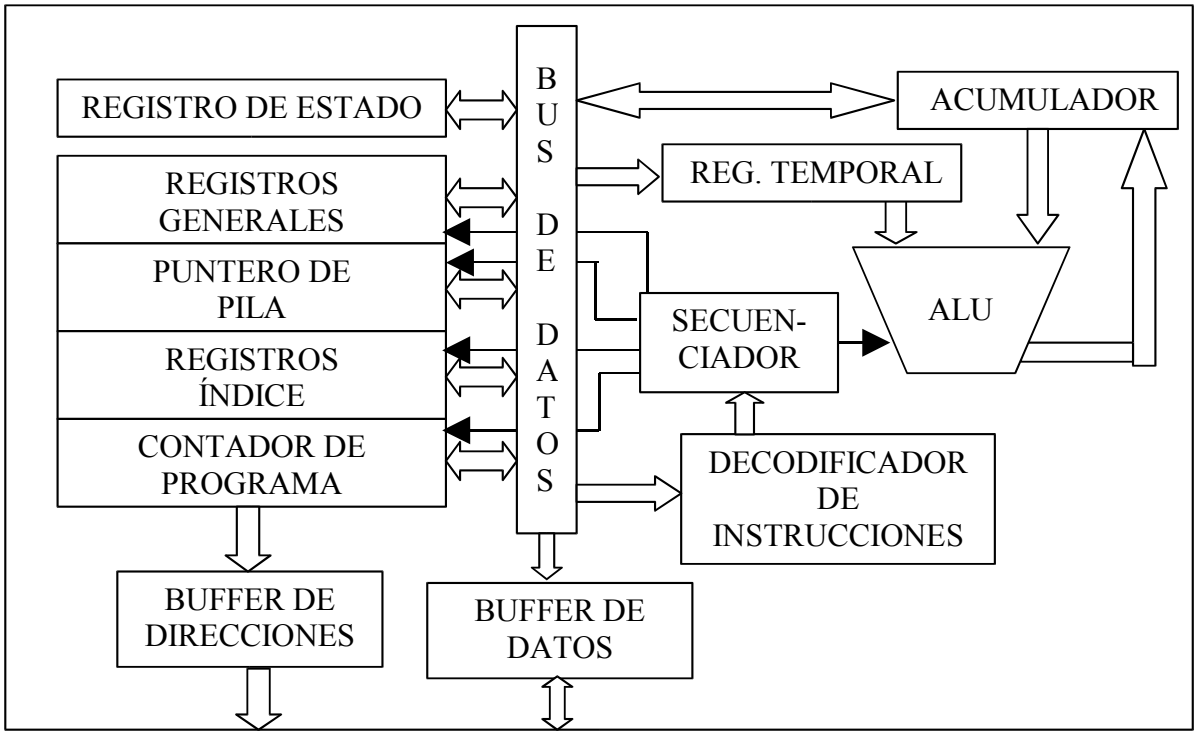
En la figura de la página siguiente se muestra la arquitectura interna generalizada de un µP. Es importante familiarizarse con los distintos bloques que en ella aparecen, y que a continuación se describen.

Contador del programa (PC): Es el encargado de direccionar la instrucción a ejecutar. Normalmente se va incrementando en una unidad a la frecuencia del reloj del sistema, aunque puede cargarse con cualquier valor, lo que permite romper la secuencia ordenada de la ejecución de instrucciones del programa.

Decodificador de instrucciones: Elemento que recibe la instrucción en código binario e interpreta su significado. Consta de un decodificador y de una ROM de microinstrucciones en la que residen los códigos de operaciones que se deben realizar para ejecutar cada una de las instrucciones.

Secuenciador: Bloque encargado de generar y transmitir las señales de gobierno y sincronismo a todo el sistema, para ejecutar la instrucción previamente decodificada. Una vez

que recibe las microinstrucciones de la ROM interna, genera las señales necesarias para poner al resto de los bloques en el modo que corresponda a la instrucción a ejecutar.



Unidad aritmético-lógica (ALU): Es la encargada de realizar las operaciones de carácter lógico y aritmético de los datos que poseen el acumulador y el registro temporal.

Acumulador (A): Registro que contiene uno de los operandos que intervienen en la operación que realiza la ALU, así como el resultado una vez realizada dicha operación. También suele nombrarse como registro de trabajo (W)

Registros de propósito general: Se emplean como registros temporales y pueden operar de modo independiente o unidos por parejas para aumentar así la longitud de palabra. Suelen denominarse con las letras B, C, D, etc.

Registro puntero de pila (SP, *stack pointer*): Está constituido por un contador y su misión es la de controlar una zona de la RAM, con estructura LIFO, en la que se salvan temporalmente el contenido de algunos registros del μP ante un salto a subrutina.
Registros índice: Empleados para direccionar determinadas posiciones de memoria.

Registro de estado: Conjunto de biestables en los que se depositan las condiciones de una determinada operación realizada. Las informaciones de este registro son de un solo bit a los que se denominan banderas (*flags*). Los más importantes son los siguientes:

- C: Arrastre aritmético.
- H: Arrastire del cuarto bit (usado en aritmética BCD).
- Z: Indica si la operación realizada ha dado cero.
- I: Utilizado para enmascarar ciertas interrupciones.
- S: Indica el signo de la operación realizada.
- P: Paridad, para la comprobación de transferencia de datos.

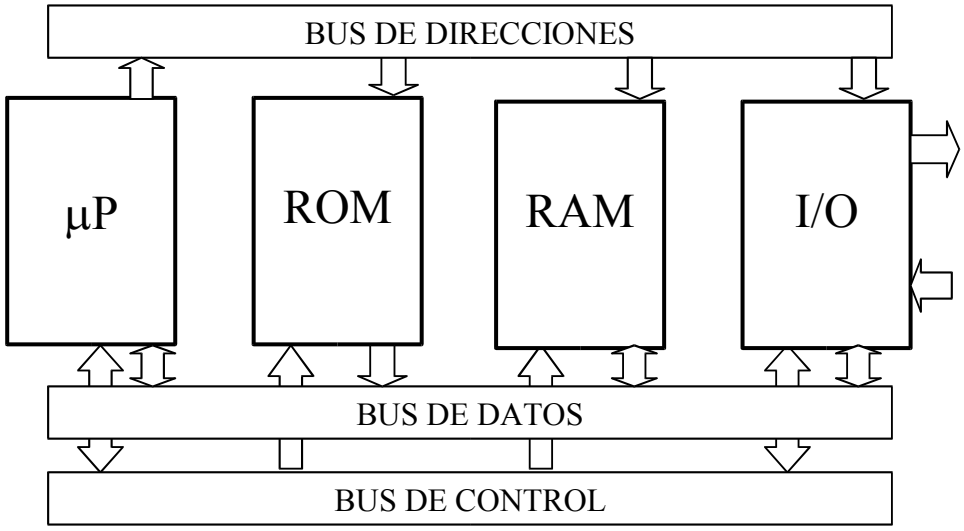
Buffer de instrucciones/datos: Registro (*latch*) bidireccional y triestado conectado al bus de datos para retención temporal.

Buffer de direcciones: Registro (*latch*) triestado que mantiene la dirección de memoria a la que se ha accedido hasta la ejecución de la instrucción, momento en el que recibe una nueva dirección contenida en el contador de programa.

13.2.2 SISTEMA BASADO EN MICROPROCESADOR

Un sistema basado en microprocesador consta, por lo general, de los tres bloques siguientes relacionados entre sí: CPU, Memoria y Unidades de entradalsalida (I/O).

La CPU ejecuta las instrucciones del programa residente en la memoria ROM y procesa los datos recibidos desde las unidades de entrada o desde la memoria, para originar unos datos que guarda en RAM o salen al exterior por las unidades de salida.



La figura anterior muestra los diferentes bloques y su estructura de conexión más común, conocida como arquitectura von Newman.

13.2.3 UNIDADES DE ENTRADA/SALIDA

Las unidades de entrada/salida o periféricos permiten la comunicación del sistema con el mundo exterior.

Las unidades de entrada permiten la captura de datos. Algunas permiten la comunicación directa entre los seres humanos y la máquina, mientras que otras requieren la grabación de los datos en la memoria. El teclado de una estación de trabajo conectada directamente a una computadora es un ejemplo de dispositivos de entrada directa.

Al igual que las unidades de entrada, los dispositivos de salida son instrumentos que interpretan información y permiten la comunicación entre los seres humanos y las computadoras. Estos dispositivos convierten los resultados que produce el procesador y que están en código de máquina en una forma susceptible de ser empleada por las personas (por ejemplo, informes impresos o desplegados en pantallas) o como entrada para otras máquinas que formen parte de un ciclo de procesamiento distinto.

Por lo general un periférico se conectará al sistema basado en μP mediante un circuito integrado apropiado al que se le suele denominar interfaz. Entre las interfaces más empleadas podemos destacar las siguientes:

a) Dispositivos para entradas y salidas en formato paralelo:

- Entrada y salida programada (*Programmed Input Output*, PIO).
- Interfaz paralelo (*Programmable Peripheral Interface*, PPI).

b) Dispositivos para entradas y salidas en formato serie:

- Receptor y transmisor Asíncrono Universal (*Universal Asynchronous Receiver Transmitter, Transceptor*,UART).
- Receptor y transmisor Síncrono - Asíncrono Universal (*Universal Synchronous -Asynchronous Receiver/Transmitter*, USART).
- Controlador de comunicaciones serie (*Serial Comunications Controller,* SCC).
- Controlador asícrono de comunicaciones serie (*Asynchronous Serial Comunications Controller,* ASCC).

c) Dispositivos de entradas y salidas con acceso directo a memoria

- Dispositivo de acceso directo a memoria (*Direct Memmory Acess*, DMA)

13.3 EL MICROCOINTROLADOR

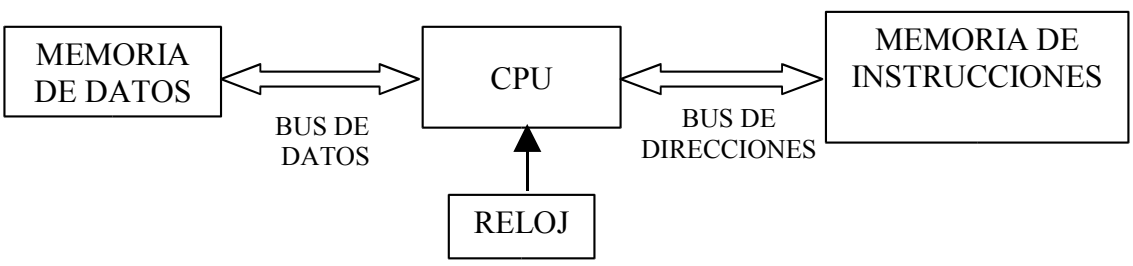
13.3.1 DEFINICIÓN

Un microcontrolador es un microprocesador optimizado para ser utilizado en el control de equipos electrónicos. A diferencia de los microprocesadores, los microcontroladores pueden integrar la totalidad de la arquitectura Von Neuman en su interior.

Los microcontroladores incorporan además de la CPU, cierta cantidad de memoria (ROM, RAM, EPROM, ...) y diversos elementos del subsistema de entrada y salida como puertos y dispositivos periféricos (contadores, temporizadores, conversores A/D,.... De este modo un microcontrolador es un sistema computador limitado, pero con todos sus elementos, en un espacio reducido. Esta característica hace que se empleen en sistemas específicos en los que el espacio ocupado, consumo y precio deben ser mínimos.

13.3.2. ARQUITECTURA INTERNA

En el punto anterior hemos comentado que un μC puede tener integrada una arquitectura Von Newman, pero es más común que integren la arquitectura Harvard.



Como puede observarse en la figura anterior, la arquitectura Harvard independiza la memoria de datos y la de instrucciones para obtener cada una su propio sistema de buses.

En la memoria de instrucciones, también llamada memoria de programa, se almacena el programa de control. Los tipos de memoria adecuados son ROM, PROM, EPROM, E2PROM o FLASH.

La memoria de datos maneja información que el programa suelen variar continuamente, por lo que debe ser del tipo de lectura y escritura. La SRAM suele ser la más empleada

13.3.3. TERMINALES Y RECURSOS DE UN µC

Según las aplicaciones a las que haya destinado el fabricante el µC, nos encontraremos con una serie de terminales y recursos. A continuación se describen los más frecuentes.

Terminales

- Alimentación: Dos terminales para polarizar el circuito.
- Entrada de reloj: Dos terminales para conectar el cristal de cuarzo.
- Reset: Terminal de puesta a cero del sistema.
- Puertos: Terminales de entrada/salida para el control de periféricos.

Recursos

- Circuito de reloj: Se encarga de generar los impulsos de sincronización del sistema.
- Temporizadores: Encargados de controlar tiempos
- Conversores A/D y D/A: Para el tratamiento de señales analógicas.
- Perro Guardián (*Watchdog*): Sistema de inicialización cuando el programa queda bloqueado.
- Estado de reposo: Sistema para reducir el consumo de energía.

13.4 PROGRAMACIÓN

13.4.1 INTRODUCCIÓN

Un CµP es capaz de interpretar niveles altos y bajos de tensión. Mediante abstracción, dichos niveles los identificamos con los valores binarios 1 y 0. Por lo tanto para el sistema son "inteligibles" aquellos conjuntos de bits, denominados programas, correctamente codificados.

Por programación entenderemos la tarea de obtener programas que el CµP sea capaz de interpretar.

El programa se encontrará en una zona de memoria, generalmente de sólo lectura, y el CµP irá "leyendo" e "interpretando" las instrucciones. La lectura del programa se realiza a partir de posiciones consecutivas de memoria, aunque es posible realizar saltos mediante las instrucciones apropiadas.

El modo de realizar un programa dependerá del lenguaje que utilicemos. A continuación se describen las características de los distintos lenguajes de programación.

13.4.2 LENGUAJES DE PROGRAMACIÓN

Al lenguaje que utiliza un conjunto de bits que es capaz de interpretar el CµP se le denomina **lenguaje máquina** y al programa escrito en este lenguaje se le conoce como **programa objeto**.

Dado que el lenguaje máquina es de laboriosa programación y de difícil interpretación, existen otros lenguajes más operativos. Dichos lenguajes se pueden dividir inicialmente en **lenguajes de bajo nivel** y **lenguajes de alto nivel**, dependiendo si están más o menos cerca, respectivamente, del lenguaje máquina.

Al programa realizado tanto en lenguaje de alto como de bajo nivel, se le denomina **programa fuente**, y para convertirlo en programa objeto se necesita un "traductor" que recibirá el nombre de **ensamblador** si se ha programado en un lenguaje de bajo nivel o de **compilador** o **intérprete** si se ha programado en alto nivel.

Seguidamente nos centraremos en un lenguaje de bajo nivel denominado **simbólico**, **nemóníco** o simplemente **lenguaje ensamblador**. Aunque la mayoría de CµP's pueden programarse en lenguajes de alto nivel (Basic, C, C++, Java, etc.), el lenguaje ensamblador es que el que más potencia ofrece si bien a costa de una mayor complejidad.

Para sacar todo el partido en la programación de un CµP en lenguaje ensamblador, será necesario conocer su arquitectura interna, y para ello es necesario tener ciertos conocimientos en electrónica digital.

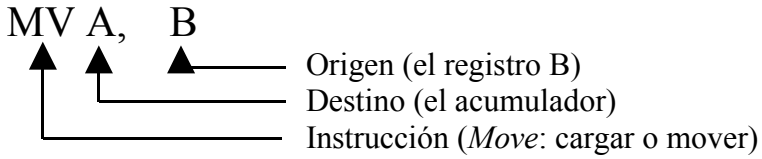
13.4.3 LENGUAJE ENSAMBLADOR

En lenguaje ensamblador, a cada uno de los códigos que es capaz de interpretar el CµP se le asigna un símbolo. De esta forma se hace más fácil su interpretación; por ejemplo, para la instrucción de parar en el µP Z80, es más fácil recordar, y por supuesto interpretar la palabra HALT, que el octeto (*byte*) 01110110 en binario (76H en hexadecimal).

En general, una instrucción en ensamblador podrá tener uno, dos o tres conjuntos de bits. Por ejemplo, si el CµP es de 8 bits, una instrucción en ensamblador podrá tener uno, dos o tres octetos, ocupando el mismo número de posiciones de memoria.

Para programar un CµP en lenguaje ensamblador, será necesario conocer su juego de instrucciones, cuyo número y potencia dependerá en gran medida de su arquitectura. Las dos arquitecturas típicas son la **RISC** (*Reduced Instruction SetCcomputere,* computador de juego de instrucciones reducido) y la **CISC** (C*omplex Instruction Set Computer,* computador de juego de instrucciones complejo). [WB_Enlace_12_1].

Veamos a continuación la interpretación de una instrucción en ensamblador en la que el valor contenido en el acumulador (A) se carga en el registro de propósito general B:



13.4.4 TIPOS DE INSTRUCCIONES

Las instrucciones que interpreta y ejecuta un CμP se pueden reagrupar por funciones. A continuación se indican las más importantes.

a) Instrucciones de transferencia: Empleadas para transferir datos de fuente a destino, por ejemplo cargar el acumulador con el dato residente en una posición de memoria, leer o escribir en un periférico de entrada/salida, etc.

b) Instrucciones aritméticas: Aquellas que realizan sumas o restas con o sin acarreo, ajuste de un valor a notación decimal, incremento o decremento de un registro, etc.

c) Instrucciones lógicas: Tales como la AND, OR, X-OR, etc.

d) Instrucciones de rotación y desplazamiento: Las que rotan a izquierda o derecha los bits de un registro.

e) Instrucciones de bifurcación o salto: Permiten efectuar saltos de modo condicional o incondicional.

f) Instrucciones de llamada y retorno a subrutinas: Para ir a subrutina y volver al programa principal.

g) Instrucciones de interrupción: Empleadas para ir a la subrutina de interrupción, así como del retorno al programa principal.

13.5 MODOS DE DIRECCIONAMIENTO

Las instrucciones de un CμP disponen de diferentes formas de localizar los datos con los que debe operar. Estas variantes reciben el nombre de **modos de direccionamiento**. La potencia y capacidad del juego de instrucciones de un CμP viene determinada no sólo por el número de instrucciones, sino por los distintos modos de direccionamiento de que disponga. En general se prefiere un CμP con pocas instrucciones pero con grandes posibilidades de direccionamiento.

A continuación se describen los modos de direccionamiento más característicos (se considera un bus de datos de 8 bits y uno de direcciones de 16 bits).

a) Direccionamiento inmediato: El octeto que sigue al código de operación es la dirección donde se encuentra el operando o dato.

b) Direccionamiento directo: Los dos octetos siguientes al código de operación, indican la dirección de memoria donde se encuentra el operando.

c) Direccionamiento indirecto: Los dos octetos siguientes al código de operación, indican la dirección de memoria cuyo contenido y el de la siguiente, forman la dirección de otra posición de memoria donde se encuentra el operando.

d) Direccionamientb relativo: Detrás del código de la instrucción se proporciona un valor que, sumado al contenido del contador del programa, proporciona la dirección donde se encuentra el operando.

e) Direccionamiento por registro: El CμP dispone de uno o varios registros cuyos contenidos sirven para localizar las posiciones de memoria donde se encuentran los operandos.

f) Direccionamiento indexado: Detrás del código de operación se encuentra un valor que, sumado al contenido del registro índice, facilita la dirección donde se encuentra el operando o dato.

13.6 PROGRAMACIÓN EN ENSAMBLADOR

En general, para la programación en ensamblador se seguirán las mismas fases que se siguen en la programación en lenguajes de alto nivel:

1.- Definición y estudio: se deberá examinar con detalle todos los aspectos del proyecto, así como los dispositivos que han de gobernarse.

2.- Ordinograma o diagrama de flujo: Se trata de un gráfico, realizado con símbolos establecidos, que representa las operaciones elementales que de forma ordenada deberá realizar el sistema.

3.- Programación: Realización mediante instrucciones de las operaciones presentadas en el ordinograma. Dentro de esta fase existe una etapa de depuración en la que verifica y corrige el programa hasta obtener su correcto funcionamiento.

Para programar en ensamblador es indispensable comenzar aprendiendo el juego de instrucciones del CμP y conocer la arquitectura del mismo. A partir de aquí habrá que comenzar realizando programas sencillo para después ir incrementando el grado de dificultad.

Para iniciarnos en la programación en ensamblador utilizaremos un μP virtual: **SimuProc**, un programa cuyo autor, Vadimir Yepes, lo ofrece gratuitamente desde Internet. En el Anexo D podrás encontrar las especificaciones relativas a su arquitectura y su juego de instrucciones.

A continuación ilustraremos los pasos anteriores con un sencillo ejemplo de programación que resolveremos para el SimuProc.

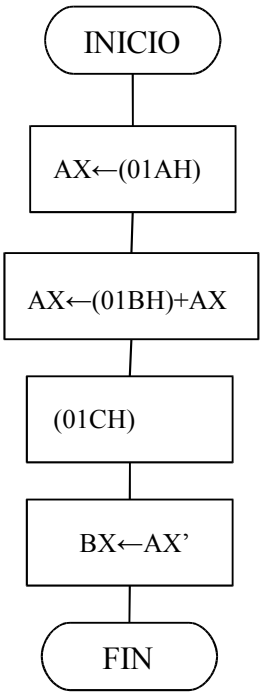
Ejemplo:

Sumar los valores de los datos almacenados en las posiciones de memoria 01AH y 01BH, y almacenar el resultado en la posición de memoria 01CH y su complemento a unos en el registro de propósito general BX.

Estudio:

Almacenaremos el primer valor en el acumulador (AX) y lo sumaremos con el segundo valor. El resultado y su negado lo almacenaremos en las posiciones pedidas.

Ordinograma:



Programa:

DIR	Ensamblador	Descripción
000	MOV AX, 01A	Primer dato al acumulador
001	ADD 01B	Suma con el segundo dato
002	MOV 01C, AX	Carga en (01CH)
003	NOT AX	Negación del acumulador
004	MOV BX, AX	Carga a BX
005	HLT	Fin de programa

Recuerde que los resultados de las operaciones de suma y negación quedan almacenados en el acumulador (AX).

II IMPROVE YOUR TECHNICAL ENGLISH

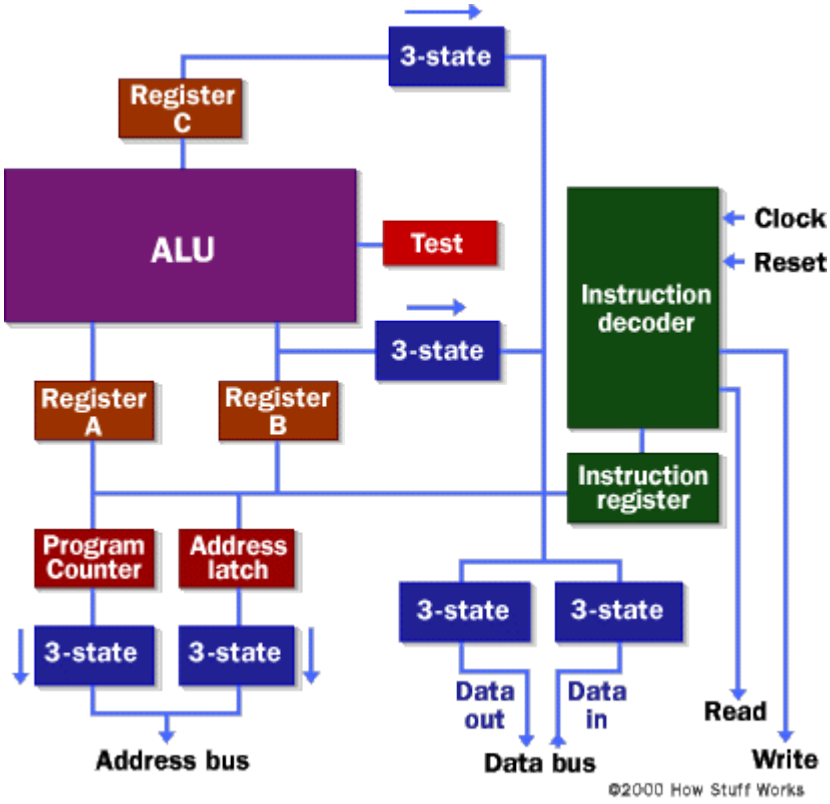
1 Inside a Microprocessor

To understand how a microprocessor works, it is helpful to look inside and learn about the logic used to create one. In the process you can also learn about assembly language -- the native language of a microprocessor -- and many of the things that engineers can do to boost the speed of a processor.

A microprocessor executes a collection of machine instructions that tell the processor what to do. Based on the instructions, a microprocessor does three basic things:

- Using its ALU (Arithmetic/Logic Unit), a microprocessor can perform mathematical operations like addition, subtraction, multiplication and division. Modern microprocessors contain complete floating point processors that can perform extremely sophisticated operations on large floating point numbers.
- A microprocessor can move data from one memory location to another.
- A microprocessor can make decisions and jump to a new set of instructions based on those decisions.

There may be very sophisticated things that a microprocessor does, but those are its three basic activities. The following diagram shows an extremely simple microprocessor capable of doing those three things:



This is about as simple as a microprocessor gets. This microprocessor has:

- An address bus (that may be 8, 16 or 32 bits wide) that sends an address to memory

- A data bus (that may be 8, 16 or 32 bits wide) that can send data to memory or receive data from memory
- An RD (read) and WR (write) line to tell the memory whether it wants to set or get the addressed location
- A clock line that lets a clock pulse sequence the processor
- A reset line that resets the program counter to zero (or whatever) and restarts execution

Let's assume that both the address and data buses are 8 bits wide in this example. Here are the components of this simple microprocessor:

- Registers A, B and C are simply latches made out of flip-flops. (See the section on "edge-triggered latches" in How Boolean Logic Works for details.)
- The address latch is just like registers A, B and C.
- The program counter is a latch with the extra ability to increment by 1 when told to do so, and also to reset to zero when told to do so.
- The ALU could be as simple as an 8-bit adder (see the section on adders in How Boolean Logic Works for details), or it might be able to add, subtract, multiply and divide 8-bit values. Let's assume the latter here.
- The test register is a special latch that can hold values from comparisons performed in the ALU. An ALU can normally compare two numbers and determine if they are equal, if one is greater than the other, etc. The test register can also normally hold a carry bit from the last stage of the adder. It stores these values in flip-flops and then the instruction decoder can use the values to make decisions.
- There are six boxes marked "3-State" in the diagram. These are tri-state buffers. A tri-state buffer can pass a 1, a 0 or it can essentially disconnect its output (imagine a switch that totally disconnects the output line from the wire that the output is heading toward). A tri-state buffer allows multiple outputs to connect to a wire, but only one of them to actually drive a 1 or a 0 onto the line.
- The instruction register and instruction decoder are responsible for controlling all of the other components.

Although they are not shown in this diagram, there would be control lines from the instruction decoder that would:

- Tell the A register to latch the value currently on the data bus
- Tell the B register to latch the value currently on the data bus
- Tell the C register to latch the value currently on the data bus
- Tell the program counter register to latch the value currently on the data bus
- Tell the address register to latch the value currently on the data bus
- Tell the instruction register to latch the value currently on the data bus
- Tell the program counter to increment
- Tell the program counter to reset to zero
- Activate any of the six tri-state buffers (six separate lines)
- Tell the ALU what operation to perform
- Tell the test register to latch the ALU's test bits
- Activate the RD line
- Activate the WR line

Coming into the instruction decoder are the bits from the test register and the clock line, as well as the bits from the instruction register.

III PRÁCTICA Y EJERCICIOS

1. APARTADOS PRÁCTICOS

Realice en mnemónico cada uno de los programas propuestos a continuación y compruebe su funcionamiento con el simulador **SimuProc**.

1. Instrucciones de transferencia.

1.1 Mediante la opción "Modificar una Posición de Memoria" introduzca los siguientes datos: (00A) = _ _ _ _ (00B)= _ _ _ _ (00C) = _ _ _ _

1.2 Cargue en cada uno de los registros generales el siguiente dato: _ _ _ _H.

1.3 Guarde dos datos diferentes en dos posiciones de memoria cualesquiera y cree un programa que permita intercambiarlos.

2. Instrucciones aritméticas, lógicas y de rotación.

2.1 A partir de los datos de las siguientes posiciones de memoria: (010) = _ _ _ _H y (015) = _ _ _ _H, se desea un programa que realice las siguientes operaciones lógicas de modo que los resultados se vayan almacenando en la pila:

(010) OR (015) (010) AND (015) (010) XOR (015) (010) OR (015)'

2.2 Realice un programa que permita sumar los valores de dos posiciones de memoria. ¿Qué *flag* detecta el rebosamiento en la suma y en qué registro queda almacenado?

2.3 A partir de las instrucciones LDT y EAP realice un programa que permita multiplicar dos números introducidos por el teclado y muestre el resultado por pantalla.

2.4 Se desea sumar el valor _ _H con el resultado de rotarlo 5 veces a la derecha y guardar el negado del resultante de la operación en la dirección de memoria 111. Escriba el programa adecuado y razone el resultado.

3. Instrucciones de bifurcación y salto.

3.1 Se desea restar el contenido de dos posiciones de memoria almacenando el resultado en los registro BX o CX según sea positivo o negativo, respectivamente. Escriba el programa y compruebe su funcionamiento.

3.2 Escriba un programa que sume el contenido de dos posiciones de memoria de modo que el resultado salga por pantalla indicando si hubo o no desbordamiento.

3.3 Dado un dato por teclado, se desea saber el número de veces que dicho dato está contenido en la zona de memoria comprendida entre las direcciones 020h y 030h. El resultado se dará por pantalla.