ANTOLOGIA

MICROPROCESADORES

LICENCIATURA EN INGENIERIA EN SISTEMAS COMPUTACIONALES

SÉPTIMO CUATRIMESTRE

Septiembre- Diciembre

Marco Estratégico de Referencia

ANTECEDENTES HISTORICOS

Nuestra Universidad tiene sus antecedentes de formación en el año de 1979 con el inicio de actividades de la normal de educadoras "Edgar Robledo Santiago", que en su momento marcó un nuevo rumbo para la educación de Comitán y del estado de Chiapas. Nuestra escuela fue fundada por el Profesor de Primaria Manuel Albores Salazar con la idea de traer Educación a Comitán, ya que esto representaba una forma de apoyar a muchas familias de la región para que siguieran estudiando.

En el año 1984 inicia actividades el CBTiS Moctezuma Ilhuicamina, que fue el primer bachillerato tecnológico particular del estado de Chiapas, manteniendo con esto la visión en grande de traer Educación a nuestro municipio, esta institución fue creada para que la gente que trabajaba por la mañana tuviera la opción de estudiar por las tarde.

La Maestra Martha Ruth Alcázar Mellanes es la madre de los tres integrantes de la familia Albores Alcázar que se fueron integrando poco a poco a la escuela formada por su padre, el Profesor Manuel Albores Salazar; Víctor Manuel Albores Alcázar en septiembre de 1996 como chofer de transporte escolar, Karla Fabiola Albores Alcázar se integró como Profesora en 1998, Martha Patricia Albores Alcázar en el departamento de finanzas en 1999.

En el año 2002, Víctor Manuel Albores Alcázar formó el Grupo Educativo Albores Alcázar S.C. para darle un nuevo rumbo y sentido empresarial al negocio familiar y en el año 2004 funda la Universidad Del Sureste.

La formación de nuestra Universidad se da principalmente porque en Comitán y en toda la región no existía una verdadera oferta Educativa, por lo que se veía urgente la creación de una institución de Educación superior, pero que estuviera a la altura de las exigencias de los jóvenes que tenían intención de seguir estudiando o de los profesionistas para seguir preparándose a través de estudios de posgrado.

Nuestra Universidad inició sus actividades el 18 de agosto del 2004 en las instalaciones de la 4ª avenida oriente sur no. 24, con la licenciatura en Puericultura, contando con dos grupos de

cuarenta alumnos cada uno. En el año 2005 nos trasladamos a nuestras propias instalaciones en la carretera Comitán – Tzimol km. 57 donde actualmente se encuentra el campus Comitán y el Corporativo UDS, este último, es el encargado de estandarizar y controlar todos los procesos operativos y Educativos de los diferentes Campus, Sedes y Centros de Enlace Educativo, así como de crear los diferentes planes estratégicos de expansión de la marca a nivel nacional e internacional.

Nuestra Universidad inició sus actividades el 18 de agosto del 2004 en las instalaciones de la 4ª avenida oriente sur no. 24, con la licenciatura en Puericultura, contando con dos grupos de cuarenta alumnos cada uno. En el año 2005 nos trasladamos a nuestras propias instalaciones en la carretera Comitán – Tzimol km. 57 donde actualmente se encuentra el campus Comitán y el corporativo UDS, este último, es el encargado de estandarizar y controlar todos los procesos operativos y educativos de los diferentes campus, así como de crear los diferentes planes estratégicos de expansión de la marca.

MISIÓN

Satisfacer la necesidad de Educación que promueva el espíritu emprendedor, aplicando altos estándares de calidad Académica, que propicien el desarrollo de nuestros alumnos, Profesores, colaboradores y la sociedad, a través de la incorporación de tecnologías en el proceso de enseñanza-aprendizaje.

VISIÓN

Ser la mejor oferta académica en cada región de influencia, y a través de nuestra Plataforma Virtual tener una cobertura Global, con un crecimiento sostenible y las ofertas académicas innovadoras con pertinencia para la sociedad.

VALORES

- Disciplina
- Honestidad
- Equidad
- Libertad

ESCUDO

El escudo de la UDS, está constituido por tres líneas curvas que nacen de izquierda a derecha formando los escalones al éxito. En la parte superior está situado un cuadro motivo de la abstracción de la forma de un libro abierto.

ESLOGAN

"Mi Universidad"

ALBORES

Es nuestra mascota, un Jaguar. Su piel es negra y se distingue por ser líder, trabaja en equipo y obtiene lo que desea. El ímpetu, extremo valor y fortaleza son los rasgos que distinguen.

MICROPROCESADORES

Objetivo de la materia:

Valorar la importancia de los microprocesadores en las áreas de la electrónica, comunicaciones, sistemas de control y, en particular, en los sistemas de computadoras. Explicar las diferencias entre los diversos modelos de microprocesadores y destacar las características de cada uno.

INDICE

Unidad 1

Estructura de bits y bytes

1.1.- Sistemas numéricos decimal, binario y hexadecimal.

1.2.- Formatos de datos.

1.3.- Funcionamiento interno de una PC.

1.4.- La evolución de los microprocesadores.

1.5.- Arquitectura del microprocesador 80x86.

1.6.- Estructura de un programa ejecutable cargado en memoria.

1.7.- Arreglo de registros internos.

1.8.- Operación en modo real.

1.9.- Operación en modo protegido.

Unidad 2

Modos de direccionamiento

2.1.- Direccionamiento por registros.

2.2.- Direccionamiento inmediato.

2.3.- Direccionamiento directo.

2.4.- Direccionamiento base más índice.

2.5.- Direccionamiento relativo.

2.5.1.- Instrucciones para transferencia de datos.

2.5.2.- Instrucciones aritméticas y lógicas.

2.5.3.- Instrucciones para control de programa.

Unidad 3

Formato general de un programa en lenguaje ensamblador

3.1.- Procedimiento para generar un programa ejecutable.

3.2.- Debugger.

3.3.- Introducción a las interrupciones.

3.4.1.- Interrupciones de software.

3.4.2.- Interrupciones de hardware

3.5.- Programación modular (MACROS).

Unidad 4

Señales en los pines del microprocesador 80x86

4.1.- Generador de reloj.

4.2.- Temporización del canal.

4.3.- Interfaz de memoria.

4.4.- Interface de entrada/salida.

4.4.1.- Interface programable.

4.5.- Temporizador programable.

Unidad I

Estructura de bits y bytes

1.1.- Sistemas numéricos decimal, binario y hexadecimal.

Bases numérica decimal La forma de interpretar los números decimales es a través de su posición (lugar que ocupa el digito en el número total. unidad, decena, centena, etc), que se pueden relacionar con las denominaciones de unidades, decenas, centenas, etc. Desde un punto de vista analítico, el sistema decimal tiene 10 símbolos representativos (0,1,2,3,4,5,6,7,8,9), para representar símbolos como "12" se usa una combinación de los símbolos mencionados. El principio del cálculo posicional puede hacerse extensivo a los demás sistemas numéricos mediante la siguiente ecuación:

$$\sum_{n=0}^{\infty} D_n . B^n$$

Donde n es la posición del número comenzando desde cero e incrementando con enteros positivos en la posición desde el punto decimal hacia la izquierda y con enteros negativos desde el punto decimal a la derecha, D es el número y B representa la base numérica de la cifra a convertir.

Ejemplo:

El decimal "125" podría representarse con punto decimal así, "125.00" entonces tomando desde el punto decimal hacia la izquierda y comenzando desde cero, el número cinco (5) tendrá la posición cero (0), el numero dos (2) la posición uno (1) y finalmente el número uno la posición (2), siguiendo lo establecido por la ecuación:

= 50 x100 + 21 x101 + 12 x102 = 5x1 + 2x10 + 1x100 = 5+20+100=125

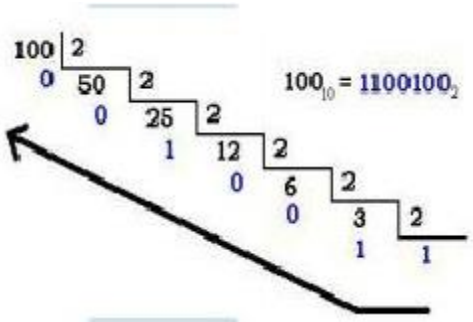
Convertir un binario a decimal

El sistema binario se basa únicamente en dos condiciones: encendido (1) o apagado (0), por lo tanto su base es 2, la anterior formula es aplicable a la conversión de cualquier base a la base decimal cada posición de los dígitos binarios representa un valor particular de "2" elevado a la potencia "n".

Es importante tener claro que el digito ubicado más a la derecha es llamado digito menos significativo (LSB ó de menos valor posicional numérico) y el digito ubicado más a la derecha es el más significativo (MSB ó de más valor posicional numérico).

Convertir un número decimal a binario

Existen dos maneras de convertir un número de decimal a binario, la primera es utilizar una tabla de equivalencias de binario a decimal siguiendo las potencias de 2 (dos), la segunda es válida para convertir de decimal a cualquier base numérica y consiste en dividir sucesivamente la parte entera entre dos tomando sus módulos (residuos) y el último resultado, la parte decimal se multiplica por la base y se toma el numero entero.

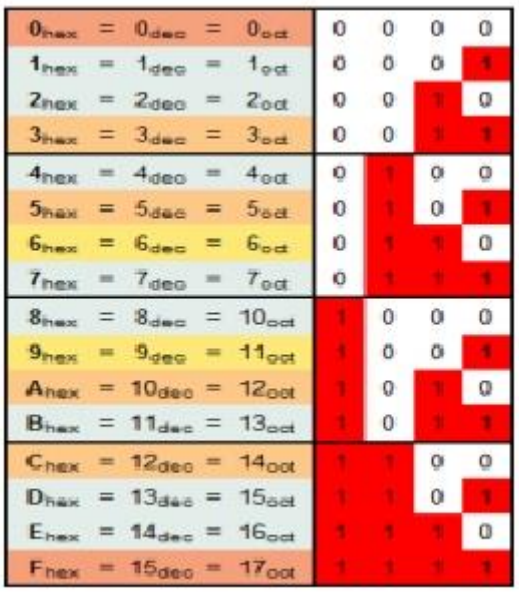



```
0,3125 (decimal)   => 0,0101 (binario).
Proceso:
0,3125 x 2 = 0,625 => 0
0,625  x 2 = 1,25  => 1
0,25   x 2 = 0,5   => 0
0,5    x 2 = 1     => 1
En orden: 0101     -> 0,0101 (binario)
```

Los dígitos son cada una de las cifras que componen un número en cualquier sistemas numérico, para el sistema binario los dígitos binarios hacen referencia a las cifras individuales representadas por un "uno" o un "cero" que en conjunto forman un número binario, en el ejemplo anterior cada residuo y último resultado es una cifra o digito binario, en conjunto forman el numero binario "1100100" que equivale al número decimal "100".

El sistema Hexadecimal

La base decimal tiene 10 dígitos representativos que van del 0 al 9 con lo que puede representarse cualquier valor, en la base hexadecimal se tiene 16 dígitos que van del 0 al 9 y de la letra A hasta la F las que representan los números del 10 al 15. La conversión entre números binarios y hexadecimales, es sencilla notando que cada cuatro binarios equivalen a un hexadecimal y cada hexadecimal equivale a cuatro (4) binarios, esta agrupación de cuatro dígitos binarios se debe comenzar desde la posición menos significativa desde el punto separador de enteros y fracciones a la izquierda o a la derecha.



$0_{hex} = 0_{dec} = 0_{oct}$	0	0	0	0
$1_{hex} = 1_{dec} = 1_{oct}$	0	0	0	1
$2_{hex} = 2_{dec} = 2_{oct}$	0	0	1	0
$3_{hex} = 3_{dec} = 3_{oct}$	0	0	1	1
$4_{hex} = 4_{dec} = 4_{oct}$	0	1	0	0
$5_{hex} = 5_{dec} = 5_{oct}$	0	1	0	1
$6_{hex} = 6_{dec} = 6_{oct}$	0	1	1	0
$7_{hex} = 7_{dec} = 7_{oct}$	0	1	1	1
$8_{hex} = 8_{dec} = 10_{oct}$	1	0	0	0
$9_{hex} = 9_{dec} = 11_{oct}$	1	0	0	1
$A_{hex} = 10_{dec} = 12_{oct}$	1	0	1	0
$B_{hex} = 11_{dec} = 13_{oct}$	1	0	1	1
$C_{hex} = 12_{dec} = 14_{oct}$	1	1	0	0
$D_{hex} = 13_{dec} = 15_{oct}$	1	1	0	1
$E_{hex} = 14_{dec} = 16_{oct}$	1	1	1	0
$F_{hex} = 15_{dec} = 17_{oct}$	1	1	1	1

El sistema Octal

El sistema octal también es utilizado en la representación de información y datos en el microprocesador, se compone de ocho dígitos (0,1,2,3,4,5,6,7) los cuales siguen una regla similar a la de los hexadecimales la cual establece que tres dígitos binarios equivalen a un dígito octal y un octal a tres binarios.

Tabla 2. *Conversión y códigos* [16]

Decimal	Binario	Hexadecimal	Octal	BCD	Exceso 3	Gray o Reflejado
0	0	0	0	0	11	0
1	1	1	1	1	100	1
2	10	2	2	10	101	11
3	11	3	3	11	110	10
4	100	4	4	100	111	110
5	101	5	5	101	1000	111
6	110	6	6	110	1001	101
7	111	7	7	111	1010	100
8	1000	8	10	1000	1011	1100
9	1001	9	11	1001	1100	1101
10	1010	A	12	0001 0000		1111
11	1011	B	13	0001 0001		1110
12	1100	C	14	0001 0010		1010
13	1101	D	15	0001 0011		1011
14	1110	E	16	0001 0100		1001
15	1111	F	17	0001 0101		1000

Representación de números negativos

Para representar números negativos en el sistema numérico binario, se realiza mediante la técnica denominada complemento a dos, esta técnica se define como el numero obtenido después de convertir todos los ceros a unos y todos los unos a ceros, al resultado se le suma uno.

Decimal 125 Equivale a 1111101 complemento a dos 0000010 + 0000001 = 0000011

Bits y Bytes

Un bit es la unidad más pequeña de información, pero en su forma más simple un bit representa un digito binario, es decir, un "uno" o un "cero", físicamente los bits requieren de un espacio o celda donde almacenar una cantidad de energía definida en dos estados estables que representen los dígitos binarios, esta energía es llamada voltaje, que es la energía necesaria para mover una carga eléctrica (electrón) de un punto a otro.

La Unidad Central de Proceso o CPU, se encarga de procesar la información dentro de celdas denominadas registros, cada registro se compone de una o varias celdas la cuales almacenan una entidad llamada "bit", en un comienzo las celdas de almacenamiento se fabricaban a partir de flip-flops o también llamados basculadores o biestables, generalmente los registros se disponen en un conjunto de 8 o 16 bits. Los flip-flops tienen la capacidad de almacenar dos niveles de voltaje, uno bajo típicamente 0,5 Volts que es interpretada como cero "0" o apagado y un nivel alto típicamente 5 Volts interpretado como uno "1" o encendido, estos estados son los conocidos como "bit" (BInary digiT o digito binario).

A un grupo de 4 bits se le conoce como Nibble, dos Nibbles u 8 bits se conocen como Byte y dos Bytes o 16 bits se denominan como palabra, existen también, la doble palabra y la cuádruple palabra. Como ejemplo de estos registros en los microprocesadores esta un registro denominado AX el cual es de 16 bits pero puede ser utilizado como de 8 bits, este registro AX puede representar 65536 números binarios entre 0000000000000000 (20) y 1111111111111111(215).

1.2.- Formatos de datos.

Buses en los microprocesadores A partir del desarrollo de la arquitectura Von Newman, se introdujo en los microprocesadores el concepto de programa almacenado y la ejecución secuencial del programa almacenado, para que el microprocesador pueda comunicarse con los diversos periféricos que tiene existen unos medios llamados buses.

Los buses permiten la conexión con los periféricos incluida la memoria y permiten la transmisión de la información, físicamente los buses consisten en líneas conductoras que permiten la circulación de los pulsos eléctricos. Generalmente el número de líneas de entrada es igual al número de líneas de salida, este número define la longitud de la palabra de datos del

microprocesador que comúnmente es 4, 8, 16, 32 y 64 bits aunque la unidad básica de almacenamiento y procesamiento es de un Byte.

Bus de datos

Es el encargado de llevar datos e instrucciones hacia y desde el microprocesador, es un bus bidireccional en donde los datos varían en tamaño desde 8 a 64 bits, por el bus de datos las instrucciones y los datos son transferidos al microprocesador y los resultados de una operación son enviados desde el microprocesador.

Bus de direcciones

Contiene la información digital que envía el microprocesador a la memoria y demás dispositivos direccionales del sistema para seleccionar una posición de memoria, una unidad de entrada/salida o un registro particular de la misma, la cantidad de líneas del bus de direcciones determina la capacidad máxima de la memoria que puede direccionar el sistema basado en microprocesador. Los primeros microprocesadores utilizados en sistemas de cómputo tenían 16 líneas de direcciones con lo que podían direccionar hasta 2 16=210*26 =1024 * 64 = 65536 = 64KB.

Cuantos más bits tenga el bus de direcciones más capacidad de direccionamiento de memoria tendrá el sistema, el número de líneas de dirección se ha incrementado a 20, 24, 32 y 36 bits este último permite direccionar:

2 36= 210*210*210*26 =1024 * 1024 * 1024 * 1024 * 64 = 1K*1K*1K*1K*64 = 1M * 1M * 64 2 36= 1G * 64 = 64 GB.

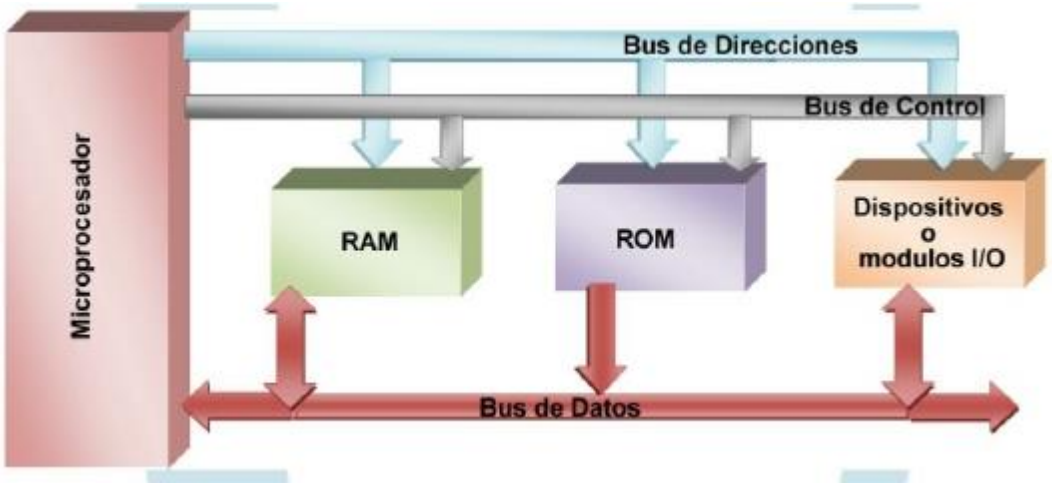
Bus de control, se utiliza para coordinar, sincronizar y comunicarse con los dispositivos externos, también se utiliza para insertar estados de espera para los dispositivos más lentos y evitar congestión en el bus, contiene líneas que seleccionan la memoria o dispositivo E/S y los habilita para que haga operaciones de lectura o escritura, particularmente en los microprocesadores 80x86 existen cuatro líneas que son:

- $\overline{MRDC}$ control de lectura de memoria.
- $\overline{MWTC}$ control de escritura de memoria.
- $\overline{IORC}$ control de lectura en un puerto.
- $\overline{IOWC}$ control de escritura en un puerto.

1.3.- Funcionamiento interno de una PC.

Los sistemas de cómputo basados en microprocesadores se componen de tres bloques fundamentales, la Unidad Central de Procesamiento – CPU, Dispositivos de memoria y Puertos de Entrada / Salida, de igual forma el microprocesador está conformado por varios bloques, entre ellos están la Unidad Aritmético – lógica (ALU), una Unidad de Control (UC) y un bloque o matriz de registros.

La unidad de entrada conformada dispositivos encargados de recibir la información del mundo exterior, estos dispositivos llamado de entrada / salido o I/O (*Input/Output*), la información es llevada hacia la unidad de memoria para ser procesada posteriormente. La información luego es llevada desde la unidad central de proceso o CPU, hacia circuitos o periféricos utilizando el bus de datos. La unidad de memoria se encarga de almacenar los datos y los programas que operan sobre esos datos. La CPU obtiene las instrucciones y los datos colocando una dirección en el bus de direcciones para posteriormente ser transferidos a través del bus de datos cuando la CPU lo solicite.



Existen dos sistemas diferentes de memoria, la de almacenamiento primario y la del almacenamiento secundario. La memoria de almacenamiento primario se refiere a la memoria que almacenan los programas que se van a ejecutar y los datos utilizados durante la ejecución del programa.

La memoria secundaria es la encargada de almacenar grandes cantidades de datos que no se requieren con frecuencia, este tipo de dispositivo son los discos duros y discos 3,5". También se distinguen tres categorías, la ROM (Red Only Memory) o memoria de solo lectura donde se almacenan cierto tipo de programas como la BIOS, la memoria RAM (Random Access Memory) o memoria de lectura y escritura, donde se almacenan datos que los programas en ejecución van generando, como tercera categoría se encuentra la "cache" con tiempo de acceso muy rápido muy cercana al núcleo del procesador.

La ejecución de un programa se realiza de forma secuencial, las instrucciones almacenadas en la memoria de programa modifican los datos almacenados en la memoria u obtenidos a través de un dispositivo de entrada, los datos que se obtienen después del procesamiento pueden ser almacenados en la memoria o ser enviados a los periféricos de salida utilizando el bus de datos, la CPU utiliza el bus de control para establecer la lectura y/o escritura de los datos y con el bus de direcciones determina el destino de los datos.

La Unidad Central de Proceso – CPU

En la práctica la Unidad Central de Proceso o CPU se encuentra en forma de un circuito integrado llamado microprocesador. El microprocesador es un circuito digital que acepta o lee datos aplicados a un cierto número de líneas de entrada, los procesa de acuerdo a las instrucciones secuenciales de un programa almacenado en su memoria y suministra o escribe los resultados del proceso con cierto número de salida. La evolución de los microprocesadores ha hecho que evolucionen los computadores ampliando sus prestaciones, servicios, capacidad y facilidad de interacción con el medio. Los microprocesadores se aplican a otras situaciones de control y supervisión en el control industrial, maquinaria, motores, telemetría, robótica, automatización entre otros.

Los datos de entrada pueden provenir de interruptores, sensores, convertidores A/D, teclados, etc; los datos de salida son dirigidos a display, LCD, CTR, convertidores D/A, impresoras, etc. El soporte físico de las instrucciones del programa es la memoria, la cual almacena los datos para que sean procesados, los niveles lógicos (unos y ceros) en las líneas de salida de un microprocesador no solo dependen del programa, también dependen de la historia o estado anterior de las señales de entrada. Como se observa es clave la cantidad de pines o conexiones de entra/salida que tenga el microprocesador para interactuar con los periféricos y la memoria por eso la mayoría de microprocesadores utilizan las mismas líneas para entrada y salida. Las líneas de control sincronizan las operaciones con los componentes externos conectados y ejercen un control global de los buses de datos y direcciones.

Los programas realizan funciones o aplicaciones limitadas por la imaginación del programador y la capacidad de los dispositivos, el programa que se ejecuta debe alojarse en determinadas posiciones de memoria y ser escrito en un lenguaje que la CPU entienda, es decir, lenguaje binario. La CPU entonces lee de forma secuencial la lista de instrucciones, las interpreta y controla la ejecución de cada una de ellas, para ejecutar cada instrucción la CPU realiza los siguientes pasos:

1. Leer de la memoria la instrucción a ejecutar y guardarla en un registro interno de la CPU.
2. Identificar la instrucción que acaba de leer.
3. Comprobar si la instrucción necesita datos que se encuentran en memoria, si es el caso, determina la localización de los mismos.
4. Buscar los datos en memoria y guardarlos en registro dentro de la CPU.
5. Ejecutar la instrucción.
6. El resultado puede ser almacenado en memoria o retenido esperando la siguiente instrucción o incluso generar comunicación con otros elementos externos a la CPU.
7. Comienza un nuevo ciclo empezando con el primer paso.

La ejecución de cada instrucción implica un movimiento de datos a partir de pulsos electrónicos que siguen una secuencia y orden establecido por una señal generada en un circuito que genera pulsos periódicos, este circuito denominado "reloj", es simple en su construcción pero vital para el funcionamiento del microprocesador.

1.4.- La evolución de los microprocesadores.

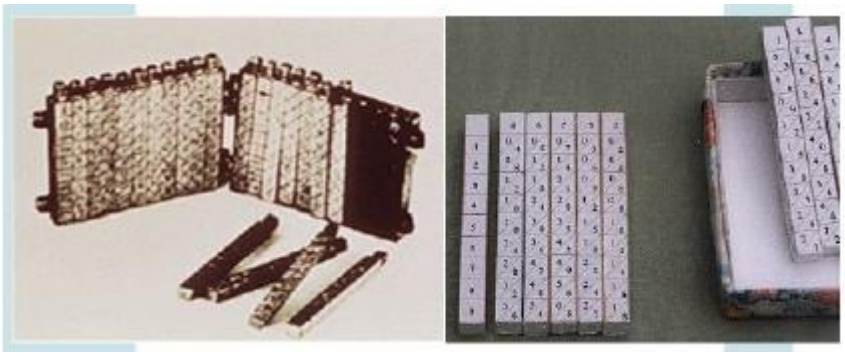
En los comienzos de la implementación de circuitos lógicos el diseñador debía poseer amplios conocimientos tanto en lógica digital como en los componentes y dispositivos que debía acoplar, esto presentaba inconvenientes, la llamada lógica cableada no permitía hacer modificaciones sin afectar la construcción física del circuito, en 1970 esto cambia con la aparición del microprocesador, con lo que aparece lo que se denomina Lógica programable, la cual permite modificar el comportamiento lógico digital del circuito sin tener que cambiar su configuración física.

Partiendo de las técnicas digitales consolidadas en los años sesenta, se comienza a desarrollar y profundizar en el estudio de las técnicas y desarrollo de aplicaciones para los microprocesadores junto con la programación en lenguaje máquina y la programación en assembler. Con esta breve introducción es hora de iniciar una exploración por los momentos históricos que han sido claves para la invención, evolución y desarrollo del microprocesador.

Primeros dispositivos para realizar cálculos

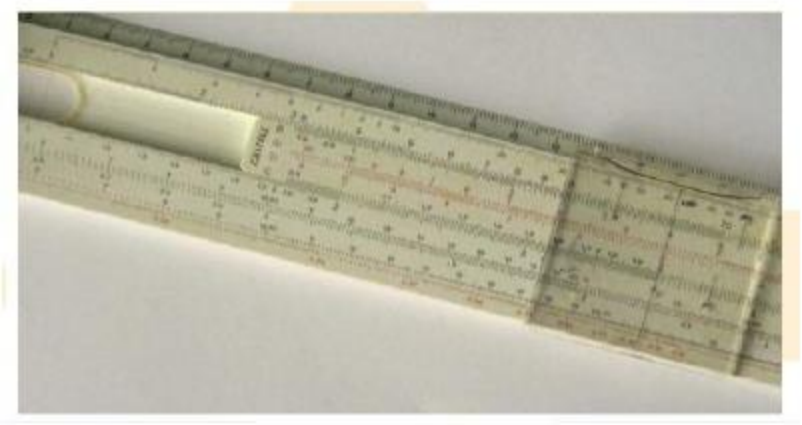
El dispositivo más reconocido es el Abaco, su origen se pierde en el tiempo algunos estiman cerca de 3000 años A.C otros entre 600 y 500 A.C en Egipto o China, su efectividad ha superado el pasar de los años.

A comienzos del siglo XVI, el sistema decimal desplazó al sistema romano en la realización de cálculos complejos, John Naiper (1550-1617), matemático escoses, descubre los logaritmos y construye las primeras tablas de multiplicar.



Dispositivos mecánicos

Basados en los logaritmos, surge las primeras máquinas analógicas de cálculo derivadas de prototipos construidos por Edmund Gunter (1581-1626), matemático y astrónomo inglés y William Ougthred (1574-1660).

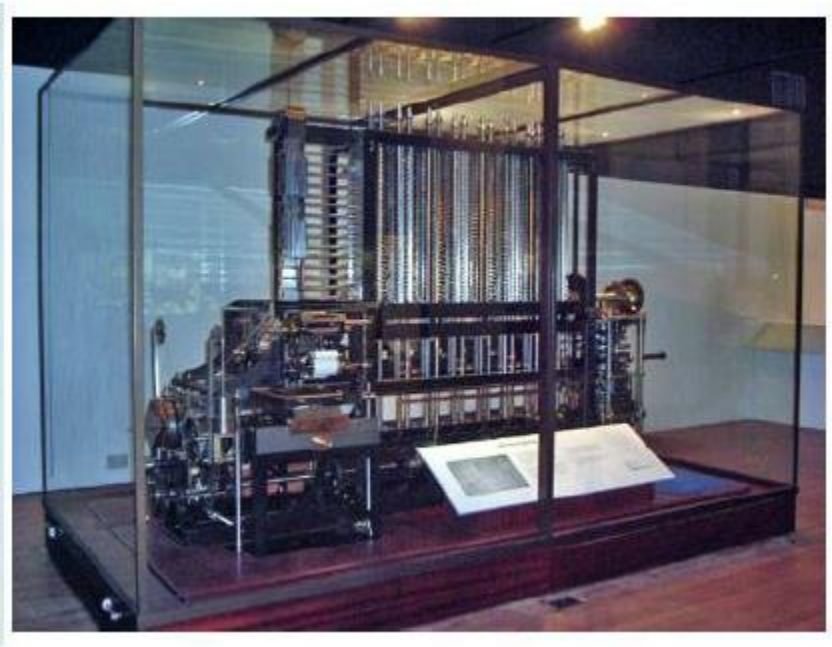


El filósofo y matemático francés Blas Pascal (1623-1662) construye la primera máquina mecánica llamada "Pascalina", su funcionamiento se basaba en engranajes y ruedas con capacidad de sumar.

La pascalina se perfecciono en una maquina llamada "Calculadora Universal", desarrollada por Gottfried Leibniz (1646-1716), tenía la capacidad de realizar las cuatro operaciones aritméticas básicas.
Otros avances significativos fueron la calculadora de 12 dígitos (1774) construida por Philipp-Matthaus Hahn; el telar de Jacquard (1801) controlado por tarjetas perforadas para reproducir patrones de tejidos, por Joseph Jacquard; el Arithmometer (1820), primera calculadora construida a nivel industrial, ideada por el francés Xavier Charles Thomas de Colmar (1785-1870).

Charles Babbage (1791-1871), inventa una maquina llamada "Máquina de Diferencias o Máquina de Babbage", este inventor británico elaboro los principios de la computadora digital moderna y junto con la matemática británica Augusta Ada Byron (1815-1852), se consideran los inventores de la computadora digital moderna. Babbage también ideo la "Maquina Analítica" (1833), como

una máquina de cálculo general aunque no se pudo construir debido a la complejidad y limitaciones tecnológicas de la época.



Era Electromagnética

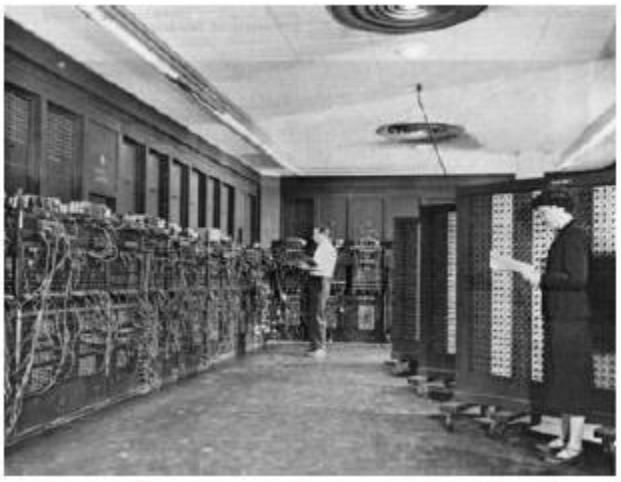
Con el surgimiento de la electricidad y dispositivos como el motor eléctrico, los contactos eléctricos y dispositivos electromagnéticos como los relay o relevos o contactores los cuales al paso de la corriente eléctrica abren o cierran contactos permitiendo el bloqueo o paso de la corriente con lo que se puede representar dos estados estables, encendido y apagado, que en lógica digital son el uno (1) y cero (0), estos estados junto con el trabajo realizado por George Boole (1815-1864), con su teoría de lógica booleana llamada "El álgebra de Boole", son la base matemática en la lógica de los computadores modernos.

El primer computador electromecánico denominado "Mark I" (1944), construido por Howard H. Aiken, tenía 760.000 ruedas y 800 Kilómetros de cable, su funcionamiento se basa en la maquina analítica de Charles Babbage.

Era eléctrica

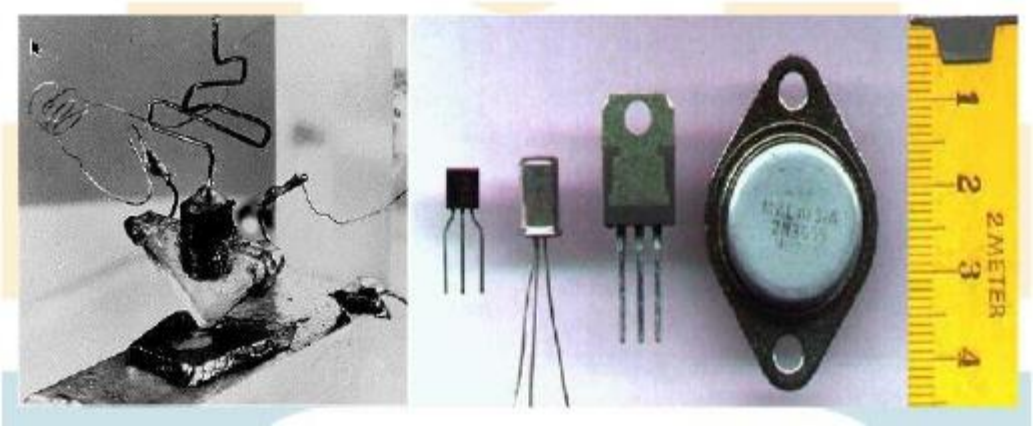
La invención del tubo de rayos catódicos o CTR (1897) por Karl Braun (1850-1918) y el diodo de válvula de vacío (1904) por Jhon A. Fleming que permite el mejoramiento en las comunicaciones, preceden la invención que define la tecnología del microprocesador, se hable del tubo al vacío (1906) por Lee de Forest (1873-1961), basado en el efecto Edison, este dispositivo es un diodo de válvula de vacío con una rejilla de control adicional creando la válvula de tres electrodos, con este dispositivo se hace posible la radio, la telefonía, la telegrafía inalámbrica, la televisión, etc. Impulsando también el desarrollo de la electrónica. Colossus fue el primer dispositivo calculador y sistema de cómputo primitivo que uso tubos al vacío, utilizado por los británicos para descifrar comunicaciones alemanas durante la segunda guerra mundial, de este se fabricaron varios modelos que mejoraban el desempeño del anterior, llegando a utilizar 4200 válvulas.

ENIAC (Electronic Numerical Integrator And Computer), primera computadora de propósito general, desarrollada en la Universidad de Pennsylvania por Jhon Presper Eckert y John William Mauchly, completamente digital ejecutaba sus instrucciones en lenguaje máquina, la ENIAC se reprogramaba recableando sus circuitos, lo que tardaba varios días e incluso semanas, construida con mas de 17000 válvulas, mas de 7000 diodos de cristal, 1500 relés, 70.000 resistencias, 10.000 condensadores, con un peso de 27 toneladas, temperatura de operación de 50°C y consumo de 160 KW.



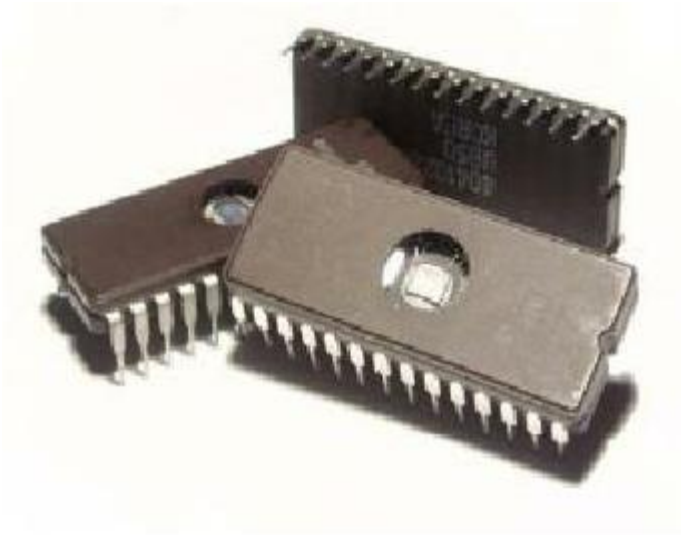
Era Electrónica

La revolución del transistor comienza con su desarrollo e invención en los laboratorios de Bell Telephone en 1948, su utilización comenzó hasta 1950, para la fabricación de aparatos de radio, televisión, sonido, militar y la naciente tecnología de computadores.



En 1958 se desarrolla el Circuito Integrado, por el Ingeniero Jack Kilby (1923-2005), consistía en un dispositivo de germanio que integraba seis transistores en una misma base semiconductora

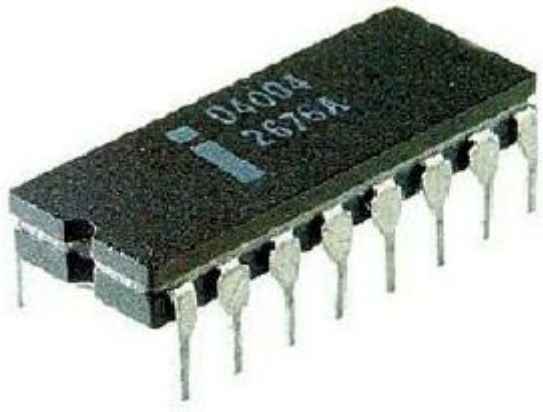
formando un oscilador de rotación de fase. El circuito integrado consiste en una pastilla pequeña de unos cuantos milímetros de área, construida a partir de silicio en la que se fabrican circuitos electrónicos basados en circuitos semiconductores, estos dispositivos se protegen del exterior por una cubierta de cerámica o plástico y se comunica al exterior por medio de unos contactos metálicos, llamados pines, patillas o contactos.



El desarrollo del transistor, conduce al desarrollo de la primera familia lógica RTL (Lógica Resistencia Transistor), con la utilización de tecnologías de integración surge la tecnología de circuitos integrados digitales. Robert Noyce después de renunciar a Fairchild en 1968 en compañía de Gordon Moore y Andrew Grove, con el respaldo económico de Arthur Rock funda la compañía INTEL donde se dan los primeros pasos para el desarrollo del microprocesador, comenzando por el desarrollo de una memoria basada en semiconductores y la lógica de un Flip-Flop como elemento de memoria capaz de almacenar un bit, se crea el primer circuito de memoria RAM llamado 1103 (1969) con una capacidad de 1024 bits.

Finalmente aparece el primer microprocesador el Intel 4004, lanzado al mercado en 1971, con una CPU de 4 bits, 2300 transistores, 16 pines (DIP-Dual In-line Package), máxima velocidad de reloj 740 KHz, con arquitectura Harvard, 46 instrucciones, 16 registros de 4 bits cada uno, stack de 3 niveles.

En 1972 se lanza al mercado el Intel 8008, empleaba direcciones de 14 bits, direccionando hasta 16KB de memoria, con la limitante de 18 pines en DIP, tiene un bus compartido de datos y direcciones de 8 bits, velocidad 0,5 a 0,8 MHz.



El primer "PC" o computador personal se construye en IBM y se lanza al mercado en 1981, aunque en el mercado ya existían otros computadores fabricados principalmente por Apple, el modelo IBM PC causa gran impacto debido al concepto de compatibilidad, es decir, la estrategia de fabricar computadores con partes de distintos fabricantes con compatibilidad IBM, en su interior tenía un microprocesador 8088 de Intel.

Microprocesadores de Intel y otros fabricantes

Intel, Con el desarrollo del 8080 de 8 bits lanzado en abril de 1974 y posteriormente el 8085 binariamente compatible con el 8080 pero que tenía menos exigencia de hardware sigue una serie de avances y desarrollo de nuevos microprocesadores con características que mejoran el desempeño con respecto a sus predecesores. En los 80"s aparecen los microprocesadores de 16

bits más potentes con la incursión al mercado del 8086 de Intel en 1978, utilizado por IBM para la fabricación de la gama de IBM PC serie PS/2. Intel prosigue el desarrollo del 8088 para IBM XT, 80186, el 80286 utilizado para IBM AT, el 80386, 80486, Pentium que fue un nombre comercial para evitar la duplicación de su nomenclatura por parte de AMD y Cyrex, más adelante sigue con el desarrollo y producción de Pentium II, Pentium III, Pentium IV, hasta los actuales microprocesadores multinucleo.

Zilog, nace como una empresa fundada por personal que trabajo para Intel en el 8080, construyen el Z-80 que incorpora un conjunto de instrucciones más extenso y compatible con el 8080.

Motorola, en la misma época del 8080 Motorola desarrolla el microprocesador de 8 bits 6800, su diseño es completamente distinto al 8080 pero tiene características similares, Motorola perfecciona el 6800 hacia el 6809 considerado uno de los mejores microprocesadores de la época aunque no tuvo éxito comercial, pero el 6502 derivado el 6800 logro posicionarse como pieza clave en la fabricación de las primeras computadoras personales PET de Commodore y Apple II de Apple Computer Inc. En un consorcio entre Apple, IBM y Motorola, se desarrolla una nueva familia de microprocesadores "los Power PC" los cuales se utilizan en las computadoras Apple de IBM actuales.

AMD, comienza a producir chips en 1969, en 1975 empieza con el desarrollo de memorias RAM, este mismo año introduce un clon del 8080 de Intel utilizando ingeniería inversa, su trabajo se redujo a procesadores embebidos logrando cierto éxito en los 80"s, al no lograr el éxito esperado se concentra en el desarrollo de memorias Flash y procesadores Intel, empezando a desarrollar procesadores como el 386 que superaba ampliamente las prestaciones del Intel 80x386, siguió con el 486 para comenzar fabricando su primer procesador propio AMD K5, K6, K6-II, K6-III, Athlon y la última gamma multinucleo.

Cyrix, comienza a operar en 1988 como proveedor de coprocesadores matemáticos para el 286 y 386, la compañía fue fundada por Jerry Roges y ex trabajadores de Texas Instruments, al igual que AMD comienza clonando procesadores de Intel comenzando con el 386 denominado por Cyrix 486SLC y 486 DLC lanzados en 1992 situando su rendimiento entre un 386 y 486 de Intel, siguen con los 486 compatibles con los de Intel, Cx5X86 compatible con Pentium y el 6x86 primer microprocesador de su línea en superar las prestaciones de Intel Pentium. En 1996 Cyrix lanza el microprocesador MediaGX y en 1997 se fusiona con National Semiconductor.

1.5.- Arquitectura del microprocesador 80x86.

Los procesadores de 16 bits fueron una nueva generación de microprocesadores desarrollados para reemplazar o completar a las microcomputadoras de 8 bits de los años setenta, que fueron las que comenzaron la revolución de las microcomputadoras.

El 8086 fue diseñado para trabajar con lenguajes de alto nivel, disponiendo de un soporte hardware con el que los programas escritos en dichos lenguajes ocupan un pequeño espacio de código y pueden ejecutarse a gran velocidad. Esta concepción, orientada al uso de compiladores, se materializa en un conjunto de facilidades y recursos, y en unas instrucciones entre las que cabe destacar las que permiten efectuar operaciones aritméticas de multiplicar y dividir, con y sin signo; las que manejan cadenas de caracteres, etc.

En su momento, el 8086 junto con el 8088 fueron los microprocesadores más empleados dentro de su categoría, especialmente desde que IBM los adoptó para la construcción de su computadora personal. Muchos fabricantes de microordenadores utilizaron esta familia microcomputadora para fabricar equipos de tipo profesional. Hoy en día, la utilización del 8086 es más reducida, quedando principalmente orientado a la enseñanza, como base de los microprocesadores de la última generación.

Antes de pasar a hacer una descripción más detallada de la arquitectura interna del microprocesador, vamos a destacar brevemente las principales características del 8086:

• El 8086 dispone de instrucciones especiales para el tratamiento de cadenas de caracteres.

• Los registros del 8086 tienen una misión específica, por lo que se podría decir que cada uno de ellos tiene su propia personalidad, aunque varios comparten tareas comunes.

• El encapsulado del 8086 está formado por 40 patillas, simplificando así el hardware, aunque por contra, es necesario la multiplexación del bus de datos con el de direcciones.

• El 8086 dispone de un conjunto de registros, denominados 'cola de instrucciones', en el cual se van almacenando de forma anticipada los códigos de las instrucciones, consiguiendo que este aumente su velocidad de trabajo.

• Las 20 líneas del bus de direcciones sólo permiten direccionar una memoria de 1 Megabyte.

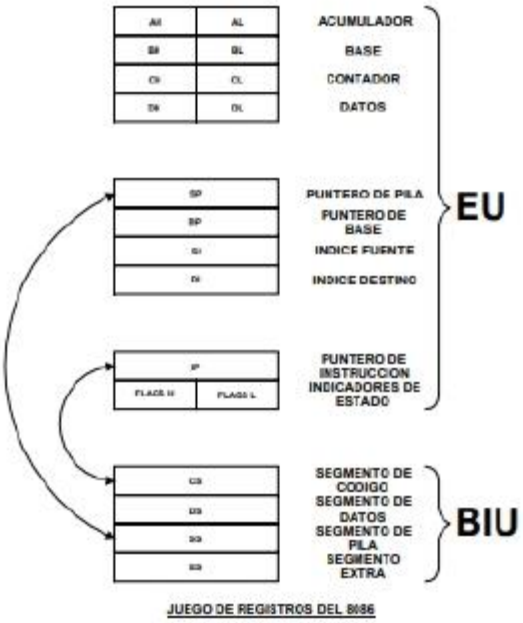
• El 8086 requiere una señal de reloj exterior, siendo 5 y 8 Mhz las frecuencias típicas de funcionamiento.

• El 8086 dispone de una arquitectura "pipe line", es decir, que la CPU puede seguir leyendo instrucciones en los tiempos en que el bus no se utiliza.

ARQUITECTURA INTERNA DEL 8086.

Este microprocesador está dividido en dos sub-procesadores. Por un lado está la "Unidad de Ejecución" (EU) encargada de ejecutar las instrucciones, la cual posee una ALU (unidad aritmético-lógica) con un registro de estado con varios flags asociados y un conjunto de registros de trabajo, y por otro está la "Unidad de Interfaz de bus" (BIU) encargada de la búsqueda de las instrucciones, ubicarlas en la cola de instrucciones antes de su ejecución y facilitar el direccionamiento de la memoria, es decir, encargada de acceder a datos e instrucciones del mundo exterior.

El 8086 contiene 14 registros de 16 bits, de los cuales, unos pertenecen a la EU, que normalmente se suelen usar para direccionamiento, y otros pertenecen a la BIU.



JUEGO DE REGISTROS DEL 8086

Los registros del 8086 podrían clasificarse en tres grupos de acuerdo con sus funciones. El grupo de datos, que es esencialmente el conjunto de registros aritméticos; el grupo de apuntadores, que incluye los registros base e índices y también el contador de programa y el puntero de pila; y el grupo de registros de segmento, que es un conjunto de registros base de propósito especial.

El grupo de registros de datos o registros generales son registros de 16 bits, pudiéndose usar cada uno de ellos como dos registros de 8 bits. Aun siendo registros de uso general tiene asignadas unas operaciones específicas. Así, por ejemplo, el AX es el acumulador de 16 bits y usándolo a veces provoca que el ensamblador produzca un lenguaje máquina codificado en muy

pocos octetos. Se emplea en multiplicaciones, divisiones, entradas/salidas, etc.; el registro BX, se utiliza como registro base para el direccionamiento de memoria; el registro CX, se utiliza como contador y almacenaje de datos y el registro DX, se utiliza para almacenar datos de 16 bits. Puede pensarse que es una extensión del registro AX para multiplicaciones y divisiones con 16 bits. Otra de sus funciones específicas es para almacenar la dirección de E/S durante algunas operaciones de E/S.

El grupo de apuntadores, es decir, punteros e índices está formado por los registros IP, SP, BP, SI, DI.

Los registros puntero son dos:

• IP como registro puntero de instrucciones conocido principalmente como contador de programa. Este contiene un valor de 16 bits que es un desplazamiento sobre la dirección del registro CS (segmento de código) que más adelante detallaremos.

• SP como registro de pila. El registro BP actúa como base de la dirección de la pila.

Los registros puntero de instrucciones (IP) y puntero de pila (SP) se encargan del control de flujo del programa.

Los registros SI y DI actúan como índices asociados al registro DS (segmento de datos).

El grupo de registros de segmento está formado por los registros CS, SS, DS y ES.

• CS (segmento de código).
• DS (segmento de datos).
• SS (segmento de pila).
• ES (segmento extra).

La importancia de dichos registros queda reflejada en la estructura de la memoria con la técnica de segmentación, que principalmente radica en el que el espacio total de memoria se divide en trozos de 64K bytes, que reciben el nombre de “segmentos”.

Las ventajas de utilizar registros de segmento son:

• Permite una capacidad de memoria de hasta 1 megabyte, aunque la dirección asociada a una instrucción sea sólo de 16 bits.

• Permiten que las partes de un programa, instrucciones, datos y pilas, tengan un tamaño mayor de 64K, mediante la utilización de más de un segmento para código, datos o pila.

• Facilitan la utilización de áreas separadas para un programa, sus datos y la pila.

• Permiten colocar un programa y sus datos en diferentes áreas de memoria cada vez que se ejecute.

Aparte de todos estos registros, el 8086 consta de un registro de estado de estado de 16 bits, aunque algunos de ellos no se utilizan. Cada uno de los bits se denomina indicador o flag, que generalmente, se modifican por las operaciones lógicas y aritméticas.

Estos indicadores son:

• SF (indicador de signo).

• ZF (indicador de cero).

• PF (indicador de paridad).

• CF (indicador de acarreo).

• AF (indicador de acarreo auxiliar).

• OF (indicador de desbordamiento).

• DF (indicador de dirección).

• IF (indicador de interrupción).

• TF (indicador de trap).

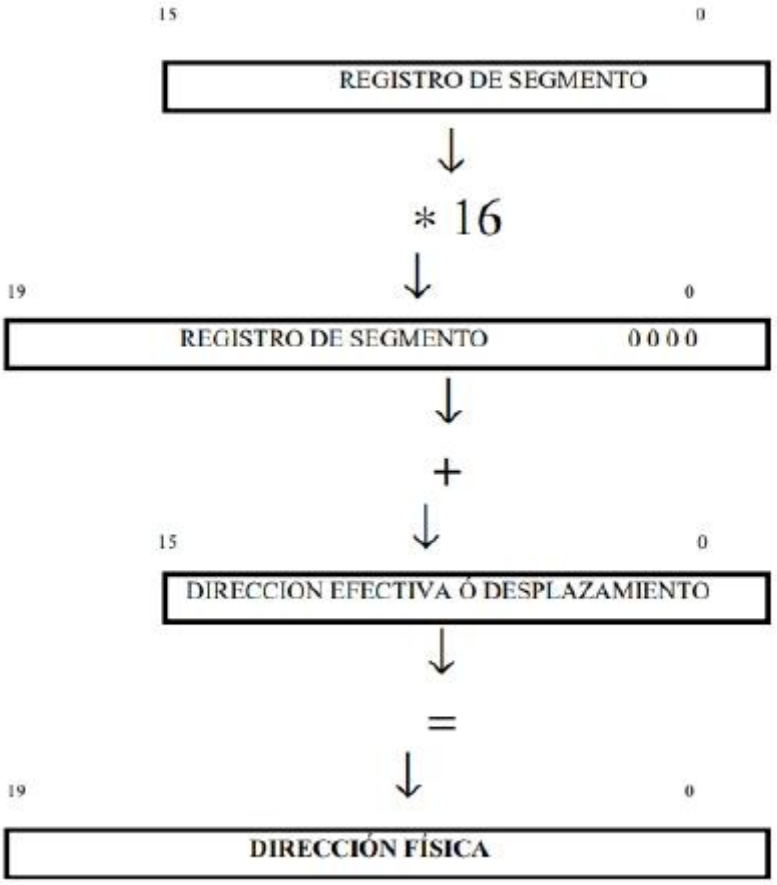
1.6.- Estructura de un programa ejecutable cargado en memoria.

El 8086 usa un esquema llamado segmentación, para acceder correctamente a un megabyte completo de memoria, con referencias de direcciones de sólo 16 bits, y todo esto gracias a la utilización de registros de segmento que dividen esencialmente el espacio de memoria en segmentos de 64K de longitud, que pueden estar separados entre sí, adyacentes o superpuestos, y que comienzan en una dirección divisible por 16.

La forma en que se completan los 20 bits del bus de direcciones, disponiendo en la CPU, solamente, registros de 16 bits, se consigue de la siguiente manera:

Se parte del contenido de uno de los registros de segmento, que actúan como base. Después, se multiplica por 16 el contenido del registro de segmento, lo que, en binario, significa añadirle 4 ceros a la derecha y convertirlo en una magnitud de 20 bits. Finalmente, se suma un desplazamiento al resultado de la multiplicación anterior. Abreviadamente, la fórmula para calcular una dirección de memoria es:

Dirección Física = 16 * (registro de segmento) + desplazamiento.



De esta manera, sobre la dirección base que apunta el registro de segmento multiplicado por 16, existe un margen de 64K bytes, controlado por un desplazamiento de 16 bits.

Después de haber visto de una forma general el funcionamiento de la estructura de la memoria de segmentación vamos a ver cómo, más concretamente, el 8086 y en función de los registros de segmento y el resto de registros, calcula las direcciones completas de las instrucciones, posiciones de pila y datos.

Las direcciones completas de las instrucciones y de las posiciones de la pila se forman sumando el contenido de los registros IP y SP con el segmento de código (CS) y el segmento de pila (SS) respectivamente.

La dirección de un dato puede formarse mediante la suma de los contenidos de los registros BX ó BP, los contenidos de SI ó DI, y un desplazamiento. El resultado de este cálculo se denomina dirección efectiva (EA) ó desplazamiento de segmento.

La dirección definitiva del dato, sin embargo, se determina mediante la EA y el registro de segmento apropiado, el segmento de datos (DS), el segmento extra (ES) ó el segmento de pila (SS).

El uso de los diferentes segmentos significa que hay áreas de trabajo separadas para el programa, la pila y los datos. Cada área de trabajo tiene un tamaño máximo de 64K bytes y un mínimo de 0. Dado que hay cuatro registros de segmento, uno de programa (CS), uno de pila (SS), uno de datos (DS) y uno extra (ES) el área de trabajo puede llegar hasta 256K.

El programador puede determinar la posición de estos segmentos, cargando el apropiado registro de segmentación de 16 bits con la dirección de segmento apropiada. Esto se realiza normalmente al inicio del programa, pero se puede fácilmente hacer en cualquier momento mientras al programa se ejecuta, cambiando dinámicamente la dirección de estos segmentos. Cambiando el desplazamiento, el programador puede acceder a cualquier punto de segmento.

Se puede pensar en el segmento como una ampliación de memoria que forma un área de trabajo. El desplazamiento es la única parte de la dirección que aparece normalmente en los programas en lenguaje ensamblador durante las referencias a direcciones del área de trabajo.

1.7.- Arreglo de registros internos.

El 8086/88 dispone de 4 registros de datos, 4 registros de segmento, 5 registros de índice y 1 registro de estado.

Registros de datos

Los registros de datos son de 16 bits, aunque están divididos. lo que permite su acceso en 8 bits. Estos registros son de propósito general aunque todos tiene alguna función por defecto.

AX (acumulador) se usa para almacenar el resultado de las operaciones, es al único registro con el que se puede hacer divisiones y multiplicaciones. Puede ser accedido en 8 bits como AH para la parte alta (HIGH) y AL (LOW) para la parte baja.

BX (registro base) almacena la dirección base para los accesos a memoria. También puede accederse como BH y BL, parte alta y baja respectivamente.

CX (contador) actúa como contador en los bucles de repetición. CL (parte baja del registro) almacena el desplazamiento en las operaciones de desplazamiento y rotación de múltiples bits.

DX (datos) es usado para almacenar los datos de las operaciones.

Registros de segmento

Los registros de segmento son de 16 bits (como ya se dicho antes) y contienen el valor de segmento.

CS (segmento de código) contiene el valor de segmento donde se encuentra el código. Actúa en conjunción con el registro IP (que veremos más adelante) para obtener la dirección de memoria que contiene la próxima instrucción. Este registro es modificado por las instrucciones de saltos lejanos.

DS (segmento de datos) contiene el segmento donde están los datos.

ES (segmento extra de datos) es usado para acceder a otro segmento que contiene más datos.

SS (segmento de pila) contiene el valor del segmento donde está la pila. Se usa conjuntamente con el registro SP para obtener la dirección donde se encuentra el último valor almacenado en la pila por el procesador.

Registros de índice

Estos registros son usados como índices por algunas instrucciones. También pueden ser usados como operandos (excepto el registro IP).

IP (índice de programa) almacena el desplazamiento dentro del segmento de código. Este registro junto al registro CS apunta a la dirección de la próxima instrucción. No puede ser usado como operando en operaciones aritmético/lógicas.

SI (índice de origen) almacena el desplazamiento del operando de origen en memoria en algunos tipos de operaciones (operaciones con operandos en memoria).

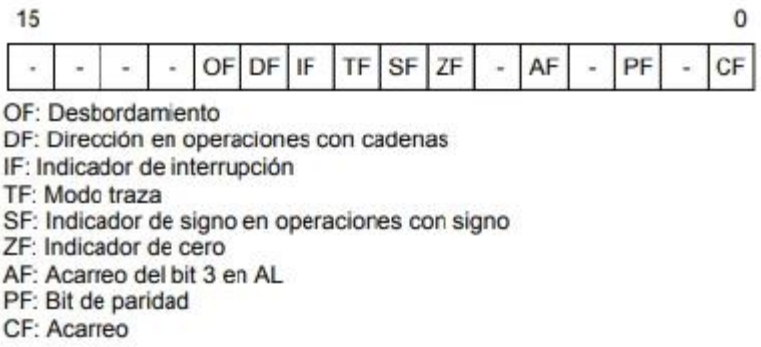
DI (índice de destino) almacena el desplazamiento del operando de destino en memoria en algunos tipos de operaciones (operaciones con operandos en memoria).

SP (índice de pila) almacena el desplazamiento dentro del segmento de pila, y apunta al último elemento introducido en la pila. Se usa conjuntamente con el registro SS.

BP (índice de base) se usa para almacenar desplazamiento en los distintos segmentos. Por defecto es el segmento de la pila

Registro de estado

El registro de estado contiene una serie de banderas que indican distintas situaciones en las que se encuentra el procesador



OF (desbordamiento) es el principal indicador de error producido durante las operaciones con signo. Vale 1 cuando:

• La suma de dos números con igual signo o la resta de dos números con signo opuesto producen un resultado que no se puede guardar (más de 16 bits).

• El bit más significativo (el signo) del operando ha cambiado durante una operación de desplazamiento aritmético.

• El resultado de una operación de división produce un cociente que no cabe en el registro de resultado.

DF (dirección en operaciones con cadenas) si es 1 el sentido de recorrido de la cadena es de izquierda a derecha, si es 0 irá en sentido contrario.

IF (indicador de interrupción) cuando vale 1 permite al procesador reconocer interrupciones. Si se pone a 0 el procesador ignorará las solicitudes de interrupción.
TF (modo traza) indica al procesador que la ejecución es paso a paso. Se usa en la fase de depuración.

SF (indicador de signo) solo tiene sentido en las operaciones con signo. Vale 1 cuando en una de estas operaciones el signo del resultado es negativo.

ZF (indicador de cero) vale 1 cuando el resultado de una operación es cero.

AF (acarreo auxiliar) vale 1 cuando se produce acarreo o acarreo negativo en el bit 3.

PF (paridad) vale 1 si el resultado de la operación tiene como resultado un número con un número par de bits a 1. Se usa principalmente en transmisión de datos.

CF (bit de acarreo) vale 1 si se produce acarreo en una operación de suma, o acarreo negativo en una operación de resta. Contiene el bit que ha sido desplazado o rotado fuera de un registro o posición de memoria. Refleja el resultado de una comparación.

1.8.- Operación en modo real.

El modo real (también llamado modo de dirección real en los manuales de Intel) es un modo de operación del 8086 y posteriores CPUs compatibles de la arquitectura x86. El modo real está caracterizado por 20 bits de espacio de direcciones segmentado (significando que solamente se puede direccionar 1 MB de memoria), acceso directo del software a las rutinas del BIOS y el hardware periférico, y no tiene conceptos de protección de memoria o multitarea a nivel de hardware. Todos los CPUs x86 de las series del 80286 y posteriores empiezan en modo real al encenderse el computador; los CPUs 80186 y anteriores tenían solo un modo operacional, que era equivalente al modo real en chips posteriores.

La arquitectura 286 introdujo el modo protegido, permitiendo, entre otras cosas, la protección de la memoria a nivel de hardware. Sin embargo, usar estas nuevas características requirió instrucciones de software adicionales no necesarias previamente. Puesto que una especificación de diseño primaria de los microprocesadores x86 es que sean completamente compatibles hacia atrás con el software escrito para todos los chips x86 antes de ellos, el chip 286 fue hecho para iniciarse en 'modo real', es decir, en un modo que tenía apagadas las nuevas características de protección de memoria, de modo que pudieran ejecutar sistemas operativos escritos para microprocesadores más viejos. Al día de hoy, incluso los más recientes CPUs x86 se inician en modo real al encenderse, y pueden ejecutar el software escrito para cualquier chip anterior.

Los sistemas operativos DOS (MS-DOS, DR-DOS, etc.) trabajan en modo real. Las primeras versiones de Microsoft Windows, que eran esencialmente un shell de interface gráfica de usuario corriendo sobre el DOS, no eran realmente un sistema operativo por sí mismas, corrían en modo real, hasta Windows 3.0, que podía ejecutarse tanto en modo real como en modo protegido. Windows 3.0 podía ejecutarse de hecho en dos "sabores" de modo protegido - el "modo estándar", que corría usando modo protegido, y el "modo mejorado 386", que además usaba direccionamiento de 32 bits y por lo tanto no corría en un 286 (que a pesar de tener modo protegido, seguía siendo un chip de 16 bits; los registros de 32 bits fueron introducidos en la serie 80386). Con Windows 3.1 se retiró el soporte para el modo real, y fue el primer ambiente operativo de uso masivo que requirió por lo menos un procesador 80286 (no contando con el Windows 2.0 que no fue un producto masivo). Casi todos los sistemas operativos modernos x86 (Linux, Windows 95 y posteriores, OS/2, etc.) cambian el CPU a modo protegido o a modo largo en el arranque.

1.9.- Operación en modo protegido.

El modo protegido es un modo operacional de los CPUs compatibles x86 de la serie 80286 y posteriores.

El modo protegido tiene un número de nuevas características diseñadas para mejorar la multitarea y la estabilidad del sistema, tales como la protección de memoria, y soporte de hardware para memoria virtual como también la conmutación de tarea. A veces es abreviado como p-mode y también llamado Protected Virtual Address Mode (Modo de Dirección Virtual Protegido) en el *manual de referencia de programador del iAPX 286 de Intel*, (Nota, iAPX 286 es solo otro nombre para el Intel 80286). En el 80386 y procesadores de 32 bits posteriores se agregó un sistema de paginación que es parte del modo protegido.

La mayoría de los sistemas operativos x86 modernos se ejecutan en modo protegido, incluyendo GNU/Linux, FreeBSD, OpenBSD, NetBSD, Mac OS X y Microsoft Windows 3.0 y posteriores. (Windows 3.0 también se ejecutaba en el modo real para la compatibilidad con las aplicaciones de Windows 2.x).

El otro modo operacional principal del 286 y CPUs posteriores es el modo real, un modo de compatibilidad hacia atrás que desactiva las características propias del modo protegido, diseñado para permitir al software viejo ejecutarse en los chips más recientes. Como una especificación de diseño, todos los CPUs x86 comienzan en modo real en el momento de carga (boot time) para asegurar compatibilidad hacia atrás con los sistemas operativos heredados, excepto el Intel 80376 diseñado para aplicaciones en sistemas embebidos. Estos procesadores deben ser cambiados a modo protegido por un programa antes de que esté disponible cualquier característica de este modo. En computadores modernos, este cambio es generalmente una de las primeras tareas realizadas por el sistema operativo en el tiempo de carga.

Mientras que la multitarea en sistemas ejecutándose en modo real es ciertamente posible mediada por software, las características de protección de memoria del modo protegido previenen que un programa erróneo pueda dañar la memoria "propia" de otra tarea o del núcleo del sistema operativo. El modo protegido también tiene soporte de hardware para interrumpir un programa en ejecución y cambiar el contexto de ejecución a otro, permitiendo pre-emptive multitasking.

Unidad 2
Modos de direccionamiento

La forma en que se especifica un operando se denomina modo de direccionamiento, es decir, es un conjunto de reglas que especifican la localización (posición) de un dato usado durante la ejecución de una instrucción.

El 8086 tiene 25 modos de direccionamiento o reglas para localizar un operando de una instrucción.
Los modos de direccionamiento más frecuentes son los que calculan la dirección del operando mediante la suma de la dirección base de un registro segmento, multiplicado por 16 y el valor de un desplazamiento.

La gran variedad de direccionamientos proviene de las muchas formas en que se puede determinar el desplazamiento.

El tipo de direccionamiento se determina en función de los operandos de la instrucción. La instrucción MOV realiza transferencia de datos desde un operando origen a un operando destino (se verá más con más detalle en los siguientes apartados). Su formato es el siguiente:

MOV *destino, origen*

2.1.- Direccionamiento por registros.

Cuando ambos operando son un registro.

Ejemplo:

MOV AX,BX ;transfiere el contenido de BX en AX

2.2.- Direccionamiento inmediato.

Cuando el operando origen es una constante.

Ejemplo:

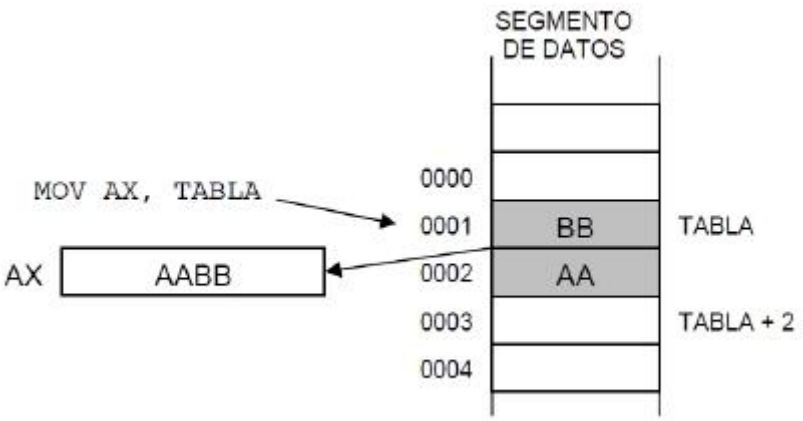
MOV AX,500 ;carga en AX el valor 500.

2.3.- Direccionamiento directo.

Cuando el operando es una dirección de memoria. Ésta puede ser especificada con su valor entre [ ], o bien mediante una variable definida previamente (cómo definir etiquetas se verá más adelante).

Ejemplo:

```
MOV BX,[1000] ; almacena en BX el contenido de la dirección de memoria DS:1000.
MOV AX,TABLA ; almacena en AX el contenido de la dirección de memoria DS:TABLA.
```

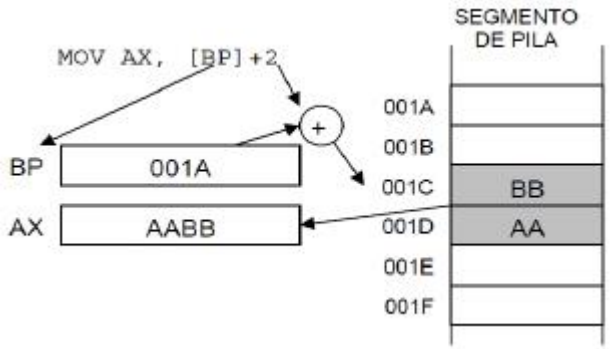


MOV [BP],CX ; almacena en la dirección apuntada por BP en contenido de CX.

2.4.- Direccionamiento base más índice.

Cuando el operando esta en memoria en una posición apuntada por el registro BX o BP al que se le añade un determinado desplazamiento Ejemplo:

MOV AX, [BP] + 2 ; almacena en AX el contenido de la posición de memoria que resulte de sumar 2 al contenido de BP (dentro de segmento de pila). Equivalente a MOV AX, [BP + 2]



Este tipo de direccionamiento permite acceder, de una forma cómoda, a estructuras de datos que se encuentran en memoria.

2.5.- Direccionamiento relativo.

Cuando la dirección del operando se obtiene de la suma de un registro base (BP o BX), de un Índice (DI, SI) y opcionalmente un desplazamiento. Ejemplo:

MOV AX, TABLA[BX][DI] ; almacena en AX el contenido de la posición de memoria apuntada por la suma de TABLA, el contenido de BX y el contenido de DI.

2.5.1.- Instrucciones para transferencia de datos.

Las instrucciones de transferencia de datos copian datos de un sitio a otro y son: MOV, CHG, XLAT, LEA, LDS, LES, LAHF, SAHF, PUSH, PUSHF, POP, POPF.

MOV realiza la transferencia de datos del operando de origen al destino. Como ya hemos visto en la parte de los modos de direccionamiento, MOV admite todos los tipos de direccionamiento. Ambos operandos deben ser del mismo tamaño y no pueden estar ambos en memoria.

MOV reg, reg ; reg es cualquier registro.

MOV mem, reg ; mem indica una posición de memoria

MOV reg, mem

MOV mem, dato ; dato es una constante

MOV reg, dato

MOV seg-reg, mem ;seg-reg es un registro de segmento

MOV seg-reg, reg

MOV mem, seg-reg

MOV reg, seg-reg

XCHG realiza el intercambio entre los valores de los operandos. Puede tener operando en registros y en memoria:

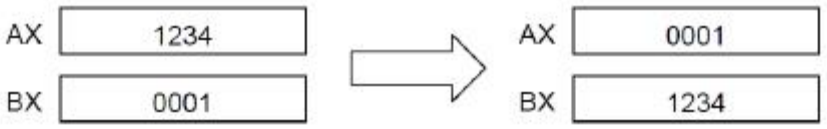
XCHG reg, mem

XCHG reg, reg

XCHG mem, reg

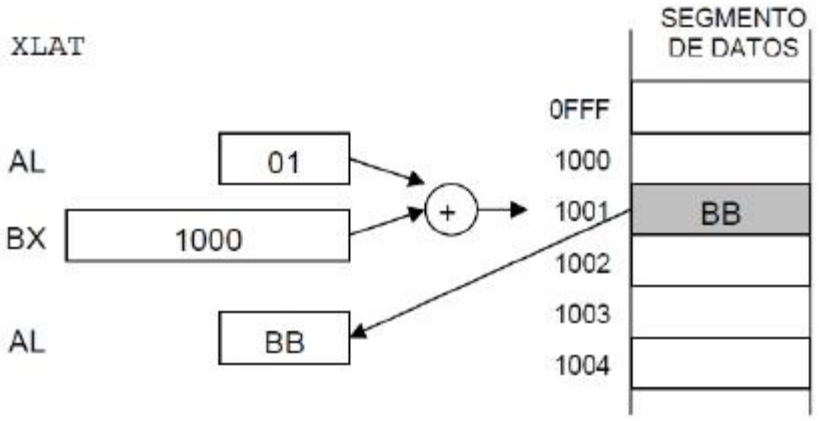
Ejemplo:

```
XCHG AX, BX
```



XLAT carga en AL el contenido de la dirección apuntada por [BX+AL].

Ejemplo:



LEA carga en un registro especificado la dirección efectiva especificada como en el operando origen:

LEA reg, mem

Ejemplos:

LEA AX, [BX] ; carga en AX la dirección apuntada por BX.

LEA AX, 3[BP+SI] ; carga en AX la dirección resultado de multiplicar

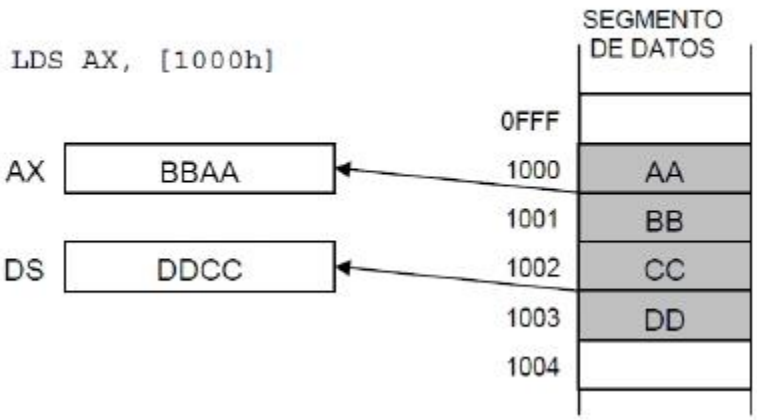
por 3 la suma de los contenido de BP y SI.

Para este tipo de acceso (el del segundo ejemplo) la instrucción LEA es más eficiente, que con la instrucción MOV e instrucciones de multiplicación.

LDS y **LES** carga el contenido de una dirección de memoria de 32 bits de la siguiente manera: la parte baja en el registro especificado y la parte alta en el registro DS y ES respectivamente.

LDS reg, mem32

Ejemplo:



PUSH y **POP** realizan las operaciones de apilado y desapilado en la pila del procesador respectivamente, admiten todos los tipos de direccionamiento (excepto inmediato). Los operandos
deben ser siempre de 16 bits.

PUSH reg

PUSH mem

PUSH seg-reg

POP reg

POP mem

POP seg-reg

Ejemplos:

PUSH AX ; envía a la pila AX

POP DS ; carga el primer elemento de la pila en DS

PUSHF y **POPF** apila y desapila el registro de estado, respectivamente.

LAHF carga la parte baja del registro de estado en AH.

SAHF carga AH en el la parte baja del registro de estado.

2.5.2.- Instrucciones aritméticas y lógicas.

Este tipo de instrucciones realizan operaciones aritméticas con los operandos. Y son: ADD, ADC, DAA, AAA, SUB, SBB, DAS, AAS, NEG, MUL, IMUL, AAM, DIV, IDIV, AAD, CBW, CWB, INC, DEC.

ADD y **ADC** realizan la suma y la suma con acarreo (bit CF del registro de estado) de dos operandos, respectivamente, y guardan el resultado en el primero de ellos. Admiten todos los tipos de direccionamiento (excepto que ambos operando estén en memoria).

ADD/ADC reg, reg

ADD/ADC mem, reg

ADD/ADC reg, mem

ADD/ADC reg, inmediato

ADD/ADC mem, inmediato

Ejemplo:

; J = 34+f

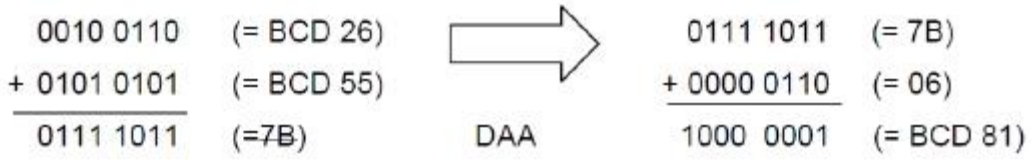
MOV AX, F

ADD AX, 34

MOV J, AX

Estas instrucciones afectan a los bits OF, SF, ZF, AF, PF, CF del registro de estado.

DAA realizan la corrección BCD empaquetado del resultado de una suma en AL.
El 8086/88 realiza las sumas asumiendo que los operados son ambos valores binarios, de manera que se suman dos valores codificados en BCD empaquetado el resultado puede no ser un valor válido en este formato:



Esta instrucción si AF = 1 o el valor de los 4 bits menos significativos del AL es mayor que 9, entonces realiza el primer ajuste BCD. Para ello suma a AL el valor 06h.

Después si CF = 1 o el valor de los 4 bits más significativos de AL es mayor que 9, realizar el segundo ajuste BCD. Para ello suma a AL el valor 60h

Esta instrucción afecta también a los bits OF, SF, ZF, AF y PF del registro de estado.

SUB y **SBB** realizan la resta y la resta con acarreo, respectivamente, de dos operandos y guardan el resultado en el primero de ellos. Admiten todos los modos de direccionamiento, excepto dos operando en memoria.
SUB/SBB reg, reg
SUB/SBB mem, reg
SUB/SBB reg, mem
SUB/SBB reg, inmediato
SUB/SBB mem, inmediato
Ejemplo:

```
; J = F-34
MOV AX, F
SUB AX, 34
MOV J, AX
```

Estas instrucciones afectan a los bits OF, SF, ZF, AF, PF, CF del registro de estado.
DAS realizan la corrección BCD empaquetado del resultado de una resta en AL. Actúan de manera similar a la instrucción de ajuste de la suma
NEG realiza la operación aritmética de negado de un operando y guarda el resultado en el mismo operando. Admite todos los tipos de direccionamiento, excepto inmediato.

NEG reg

NEG mem

La operación que realiza es: 0 – operando.

Afecta a todos los bits del registro de estado, poniendo el bit AF a 1.

MUL e **IMUL** realizan la multiplicación y multiplicación con signo, respectivamente, de contenido de AX y del operando indicado, guardando el resultado en AX, para operaciones de 8 bits y en DX:AX para operaciones de 16 bits. Los formatos son:

MUL/IMUL reg

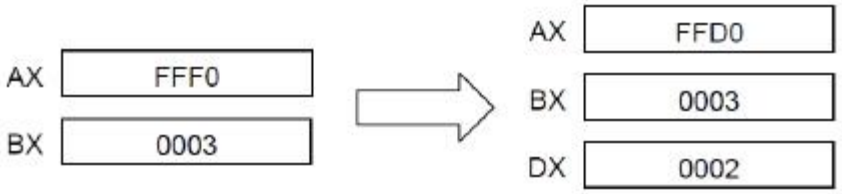
MUL/IMUL mem

Ejemplo:

MOV AX, FFF0h

MOV BX, 3

MUL BX ;DX:AX = FFF0h*3



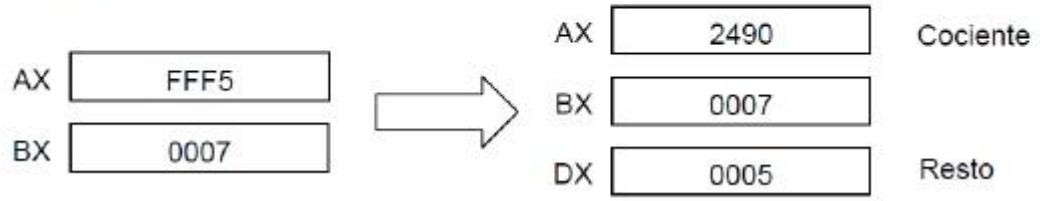
DIV e **IDIV** realizan la división y la división con signo, respectivamente. De AX entre el operando para operaciones de 8 bits, guardando el cociente en AL y el resto en AH; y DX:AX entre el operando para operaciones de 16 bits guardando el cociente en AX y el resto en DX.

DIV/IDIV reg

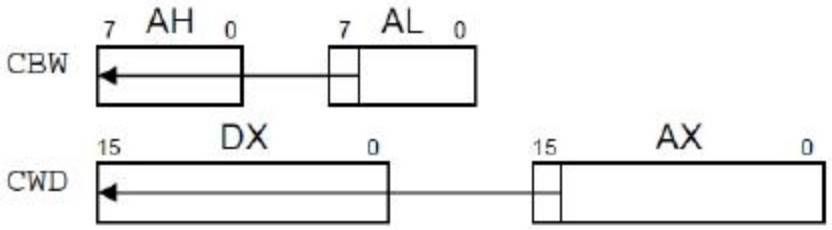
DIV/IDIV mem

Ejemplo:

```
MOV AX, FFF1h
MOV BX, 7
DIV BX
```



CBW y **CWD** realizan la extensión del bit de signo de byte a WORD y de WORD a DWORD, actuando sobre AX y DX:AX, respectivamente. Tal y como se muestra en el figura. Tras esta operación el contenido de AH es FFh.



Ejemplo:

MOV AL, -3

CBW ;AX=-3

INC y **DEC** realizan las operaciones de incremento y decremento, respectivamente, de un operando, guardando el resultado en el mismo operando. Admiten todos los modos de direccionamiento excepto el inmediato.

INC/DEC reg

INC/DEV mem

Afectan a todos los bits de estado del registro de estado.

Instrucciones lógicas

Realizan las operaciones lógicas y son: OR, XOR, AND, NOT, TEST, CMP.

OR, XOR y **AND** realizan las operaciones lógicas *"O", "O exclusiva"* e *"Y"*, respectivamente, de dos operandos, guardando el resultado en el primero de ellos. Estas operaciones son bit a bit. La tabla de verdad de estas funciones es:

Operandos		AND	OR	XOR
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	1	0

Admiten todos los modos de direccionamiento excepto los dos operandos en memoria.

AND/OR/XOR reg, reg

AND/OR/XOR reg, mem

AND/OR/XOR mem, reg

AND/OR/XOR reg, inmediato

AND/OR/XOR mem, inmediato

Afectan a los bits SF, ZF, PF del registro de estado. Además ponen a cero los bits CF y OF.

NOT realiza la operación de negado lógico de los bits del operando, guardando el resultado en el mismo operando. Admite todos los modos direccionamiento excepto inmediato.

NOT reg

NOT mem

No afecta a ningún bit del registro de estado.

Instrucciones de comparación

Estas instrucciones realizan funciones de comparación no guardando el resultado, pero si afecta al registro de estado (no cambian a los operandos). Son muy útiles en las instrucciones de salto que se verán más adelante.

TEST realiza la operación lógica *"Y"* de dos operandos, pero NO afecta a ninguno de ellos, SÓLO afecta al registro de estado. Admite todos los tipos de direccionamiento excepto los dos operandos en memoria.

TEST reg, reg

TEST reg, mem

TEST mem, reg

TEST reg, inmediato

TEST mem, inmediato

Afecta a todos los bits del registro de estado, de la misma manera que la instrucción AND.

CMP realiza la resta de los dos operandos (como la instrucción SUB) pero NO afecta a ninguno de ellos, SÓLO afecta al registro de estado. Admite todos lo modos de direccionamiento, excepto los dos operando en memoria.

CMP reg, reg

CMP reg, mem

CMP mem, reg

CMP reg, inmediato

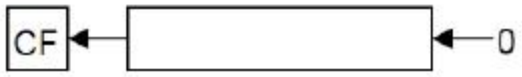
CMP mem, inmediato

Se usa con las instrucciones de salto que veremos más adelante.

Instrucciones de desplazamiento y rotaciones

Realizan operaciones de desplazamiento y rotaciones de bits, y son: SAL/SHL, SAR, SHR, ROL, ROR, RCL, RCR.

SAL/SHL realiza desplazamiento a la izquierda del primer operando tantos bits como indique el segundo operando, introduciendo un 0 y guardando el bit que sale en el bit CF del registro de estado.



Admite los siguientes formatos:

SAL/SHL reg, 1; desplaza 1 vez el contenido de reg

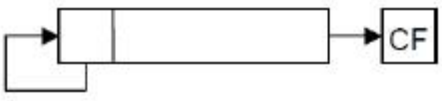
SAL/SHL mem, 1

SAL/SHL reg, CL; desplaza tantas veces el contenido de reg como indique CL.

SAL/SHL mem, CL

Afecta a los bit OF, CF del registro de estado.

SAR realiza el desplazamiento a la derecha del operando, repitiendo el bit de signo y guardando el resultado en el bit CF del registro de estado.



Admite los siguientes formatos:

SAR reg, 1; desplaza 1 vez el contenido de reg

SAR mem, 1

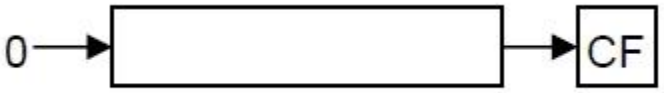
SAR reg, CL; desplaza tantas veces el contenido de reg como indique

CL.

SAR mem, CL

Afecta a todos los bit del registro de estado.

SHR realiza el desplazamiento a la derecha del operando, introduciendo un 0 y guardando el resultado en el bit CF del registro de estado.



Admite los siguientes formatos:

SHR reg, 1; desplaza 1 vez el contenido de reg

SHR mem, 1

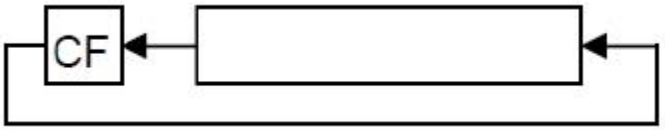
SHR reg, CL; desplaza tantas veces el contenido de reg como indique

CL.

SHR mem, CL

Afecta a los bit OF, CF del registro de estado.

RCL realiza la rotación a la izquierda de los bits del operando a través del bit CF (acarreo) del registro de estado.



Admite los siguientes formatos:

RCL reg, 1; desplaza 1 vez el contenido de reg

RCL mem, 1

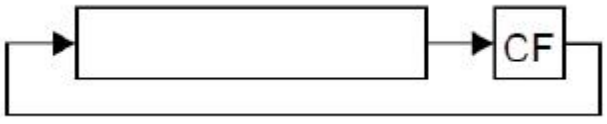
RCL reg, CL; desplaza tantas veces el contenido de reg como indique

CL.

RCL mem, CL

Afecta a los bit OF, CF del registro de estado.

RCR realiza la rotación a la derecha de los bits de operando a través del bit CF del registro de estado.



Admite los siguientes formatos:

RCR reg, 1; desplaza 1 vez el contenido de reg

RCR mem, 1

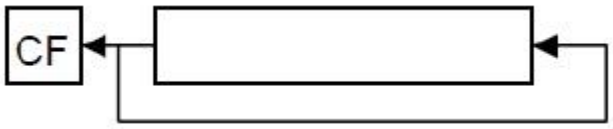
RCR reg, CL; desplaza tantas veces el contenido de reg como indique

CL.

RCR mem, CL

Afecta a los bit OF, CF del registro de estado.

ROL realiza la rotación a la izquierda de los bits del operando, ignorando el bit CF del registro de estado, aunque en CF se almacena el bit que se rota.



Admite los siguientes formatos:

ROL reg, 1; desplaza 1 vez el contenido de reg

ROL mem, 1

ROL reg, CL; desplaza tantas veces el contenido de reg como indique

CL.

ROL mem, CL

Afecta a los bit OF, CF del registro de estado.

ROR realiza la rotación a la derecha de los bits del operando, ignorando el bit CF del registro de estado, aunque en CF se almacena el bit que se rota.

Admite los siguientes formatos:

ROL reg, 1; desplaza 1 vez el contenido de reg

ROL mem, 1

ROL reg, CL; desplaza tantas veces el contenido de reg como indique

CL.

ROL mem, CL

Afecta a los bit OF, CF del registro de estado.

NOTA: Las instrucciones SHL/SAL y SAR se suelen usar para hacer divisiones y multiplicaciones, respectivamente, por números potencia de dos (2, 4, 8,16, 32, 64 y 128), de manera más eficiente que las instrucciones DIV y MUL.

Ejemplo:

MOV AX, 40h

MOV CL, 2

SHL AX, CL

; es equivalente y más eficiente que

MOV AX, 40h
MOV DX, 00h
MOV BX, 04h
DIV BX

Instrucciones de E/S

Se usan para la comunicación con los dispositivos periféricos. Y son IN, OUT.

IN lee de un puerto (sólo si la dirección del puerto es menor que 255). Admite las siguientes formas:

IN AX, inmediato; obtiene un WORD del puerto especificado y lo guarda en AX

IN AX, DX; obtiene un WORD del puerto especificado en DX y lo guarda en AX

OUT escribe en un puerto (sólo si la dirección del puerto es menor que 255). Admite las siguientes formas:

OUT inmediato, AX; escribe un WORD (contenido en AX) en el puerto especificado

OUT DX, AX; escribe un WORD (contenido en AX) en el puerto especificado en DX.

2.5.3.- Instrucciones para control de programa.

Se utilizan para el control del programa, son instrucciones de salto, bucles y llamadas a procedimientos.

Instrucciones de salto

Estas instrucciones permiten saltar a otras partes del código. Todas cambian el registro IP (contador de programa) y el registro CS (segmento de código) si es un salto lejano. Un salto es lejano cuando la dirección a la que se salta no está en el mismo segmento de código.

Existen dos tipos de saltos: los absolutos; en lo que se especifica la dirección absoluta a la que se salta; y los relativos; que son saltos hacia delante o hacia atrás desde el valor de IP.

JMP realiza un salto incondicional a la dirección especificada. La siguiente tabla relaciona los tipos de saltos y los argumentos que puede tomar esta instrucción.

	Cercano	Lejano
Relativo	8 ó 16 bits	-
Absoluto	Mem reg	Inmediato mem

Saltos condicionales estas instrucciones realizan el salto a la dirección especificada en función de si se cumple o no una condición. Para evaluar la condición se considera el registro de estado, esto quiere decir que la condición depende directamente de la instrucción anterior. En la siguiente tabla se presentan estas instrucciones en función del tipo de operandos y la condición que se quiere evaluar.

Condición	Sin signo	Con signo
=	**JE/JZ**	**JE/JZ**
>	**JA/JNBE**	**JG**
<	**JB/JNAE**	**JL**
≥	**JAE/JNB**	**JGE**
≤	**JBE/JNA**	**JLE**
≠	**JNE/JNZ**	**JNE/JNZ**

También existen instrucciones que evalúan sólo un bit del registro de estado.

BIT	= 0	= 1
ZF	**JZ**	**JNZ**
CF	**JC**	**JNC**
OF	**JO**	**JNO**
SF	**JS**	**JNS**
SP	**JP**	**JNP**

Estos saltos son siempre relativos, es decir, una dirección de 8 ó 16 bits.

Instrucción de llamada a procedimiento CALL y RET

La instrucción CALL se usa para realizar una llamada a un procedimiento y la instrucción RET se usa para volver de un procedimiento. Cuando se realiza una llamada a procedimiento con CALL, se guardan en la pila el valor de IP en caso de un salto corto, y de CS e IP en caso de un salto lejano.

Cuando se ejecuta la instrucción RET se recuperan de la pila los valores de IP o de CS e IP dependiendo del caso.

Al salir de un procedimiento es necesario dejar la pila como estaba; para ello podemos utilizar la instrucción pop, o bien ejecutar la instrucción RET n donde n es el número de posiciones que deben descartarse de la pila.

Bucles

Las instrucciones de bucle se usan para realizar estructuras repetitivas, y utilizan el registro CX como contador.

LOOP esta instrucción hace que el programa salte a la dirección especificada (salto dentro del segmento), mientras que CX sea distinto de 0 y decrementa CX en 1 cada vez.

LOOP *salto*

Ejemplo:

MOV CX, 100

COMIENZO:

…

…

LOOP COMIENZO; este bucle se repite 100

LOOPNE/LOOPNZ esta instrucción salta a la dirección especificada mientras que CX sea distinto

de 0 y si ZF = 0.

LOOPNE/LOOPNZ *salto*

Esta instrucción proporciona una ruptura del bucle adicional.

LOOPE/LOOPZ esta instrucción actúa como la anterior pero la condición adicional es ZF = 1.

LOOPE/LOOPZ *salto*

JCXZ esta instrucción realiza un salto si CX = 0.

JCXZ *salto*

Ninguna de estas instrucciones afecta al registro de estado.

Instrucciones de cadena de caracteres

Realizan operaciones con cadenas de caracteres. Antes de ver las instrucciones que manipulan adenas, es necesario comentar el uso de los prefijos de repetición, modificadores que sólo se pueden usar con las instrucciones de manipulación de cadenas.

REP este modificador repite la instrucción a la que acompaña mientras que CX sea distinto de 0 (decrementa CX cada vez). Las instrucciones con las que se puede usar son MOVS, MOVDSW o STOS.

MOVS *destino, fuente*

REP MOVS *destino, fuente*

REPE/REPZ este modificador repite la instrucción a la que acompaña mientras que CX sea distinto de 0 y ZF = 1 (decrementa CX cada vez). Las instrucciones con las que se puede usar son CMPS o SCAS.

CMPS *destino, fuente*

REPE/REPZ CMPS *destino, fuente*

REPNE/REPNZ este modificador repite la instrucción a la que acompaña si CX es distinto de 0 y

ZF = 0 (decrementa CX cada vez). Las instrucciones con las que se puede usar son CMPS o SCAS.

REPNE/REPN CMPS *destino, fuente*

MOVS/MOVSW copia un byte o un WORD de una parte a otra de la memoria.

MOVS destino, fuente.

Donde destino es ES:DI y fuente es DS:SI, lo que quiere decir que antes de utilizar la instrucción hay que cargar en SI y DI los valores apropiados.

Ejemplo:

LEA SI, fuente

LEA DI, ES:destino

MOV CX, 100

REP MOVS destino, fuente

Por lo tanto, para usar esta instrucción hay que seguir los siguientes pasos:

1.- Colocar el bit DF (dirección de recorrido) al valor correcto (lo veremos más adelante).

2.- Cargar en SI el desplazamiento de la fuente.

3.- Cargar en DI es desplazamiento del destino.

4.- Cargar en CX el número de elementos a mover.

5.- Ejecutar la instrucción MOVS/MOVSB con el prefijo REP.

Esta instrucción no afecta al registro de estado.

CMPS realiza la comparación de dos cadenas, devuelve el resultado en el registro de estado.

CMPS destino, fuente

Hay que realizar los mismos pasos para usar esta instrucción que en el caso de la instrucción MOVS/MOVSB, la única diferencia es que el modificador que usa es REPE/REPZ o REPNE/REPNZ.

Esta instrucción afecta a todos los bits del registro de estado.

SCAS/SCASW localiza el valor contenido en AL o AX (según sea byte o WORD) en una cadena, si encuentra el elemento, devuelve en DI el desplazamiento del siguiente elemento.

SCAS/SCASW destino

Al igual que las instrucciones anteriores es necesario cargar en DI el desplazamiento del primer elemento de la cadena.

Ejemplo

```
; busca en CADENA un espacio en blanco
LEA DI, ES:CADENA
MOV AL, ' '
MOV CX, 100
REP SCAS CADENA
```

Esta instrucción afecta a todos los bits del registro de estado.

LODS/LODSW transfiere un elemento de una cadena (fuente) a AL o AX, respectivamente.

LODS/LODSW *fuente*

Esta instrucción también necesita que la fuente esté cargada en SI.

STOS/STOSW transfiere el contenido de AL o AX, respectivamente, a una cadena (destino).

STOS/STOSW *destino*

También debe cargarse en DI el desplazamiento de la cadena, y puede usarse con el modificador REP.

Ejemplo:

```
; busca en una cadena un 0 y si lo encuentra rellena las siguientes
5 posiciones con ceros.
LEA DI, ES:CADENA
MOV AX, 0
```

```
MOV CX, 200
REPNE SCASW
JCXZ no_encon
SUB DI, 2
MOV CX, 6
REP STOS CADENA
no_encon:
…
…
```

Otras instrucciones

HLT parada del procesador, solo es posible salir de esta estado reiniciando o por medio de una interrupción externa.
HLT
LOCK bloquea el acceso al bus por parte de otro dispositivo mientras dure la ejecución de la instrucción a la que acompaña.
LOCK instrucción
WAIT genera estados de espera en el procesador hasta que se active la línea TEST, generalmente usada por el coprocesador.
WAIT
CLC/STC pone a 0 ó a 1, respectivamente, el bit CF del registro de estado.
CLC/STC
CMC cambia el valor del bit CF del registro de estado.
CMC
CLI/STI pone a 0 ó a 1, respectivamente, el bit IF del registro de estado.
CLI/STI
CLD/STD pone a 0 ó a 1, respectivamente, el bit DF del registro de estado. Este bit es el que se usa para recorrer una cadena de manera ascendente o descendente en memoria.
CLD/STD
NOP no operación, hace que el procesador ejecute NADA.
NOP.

Etiquetas, comentarios y directivas

Las **etiquetas** asignan un nombre a una instrucción. Esto permite hacer referencia a ellas en el resto del programa. Pueden tener una máximo de 31 caracteres y deben terminar en ":".

Los **comentarios** permiten describir las sentencias de un programa, facilitando su comprensión. Comienzan por ";", el ensamblador ignora el resto de la línea.

Ejemplo:
INI_CONT: MOV CX, DI ; inicia el contador

Las **directivas** son comandos que afectan al ensamblador, no al procesador. Se puede usar para preparar segmentos y procedimientos, definir símbolos, reservar memoria, etc. La mayoría de las directivas no generan código objeto.

Las directivas más comunes son:
Las **directivas simplificadas** se utilizan para la definición de segmentos.
.MODEL para usar las directivas simplificadas es necesario incluir esta directiva que define el modelo de memoria que debe usarse. Algunos de los argumentos que puede tomar son:
TINY: para programa con un solo segmento para datos y código (tipo .COM)
SMALL: para programas con un solo segmento de datos (64K, incluida la pila) y otro de código (64K).
LARGE: varios segmentos de datos y código (1Mb para cada uno).
MEDIUM: Varios segmentos de código y 1 de datos.
COMPACT: 1 segmento de código y varios de datos.
Con esta directiva se preparan todos los segmentos y el ensamblador reconoce, a partir de este momento, las directivas .DATA, .STACK y .CODE.
.STACK n sirve para fijar un tamaño **n** del segmento de pila, por defecto 1K.
.DATA abre el segmento de datos.
.CODE abre el segmento de código, al final código debe aparecer **END**.
Una vez inicializado los segmento se permite usar los símbolos @CODE y @DATA en lugar del nombre de los segmentos de código y datos respectivamente.
Justo después de la directiva .CODE hay que inicializar el segmento de datos (ya que la directiva o genera código):
MOV AX, @DATA
MOV DS, AX
PROC y **ENDP** definen un procedimiento (o subprograma).
EQU asigna nombre a números, combinaciones de direccionamiento y a otras cosas que se vayan a usar repetidas veces en el código.
Ejemplo:
K EQU 1024 ; especifica una constante
TABLA EQU DS:[BP][SI] ;especifica una combinación de direc.
VELOCIDAD EQU TOCINO ; da un nombre alternativo

DB, DW y **DD** se usan para asignar espacio a las variables en memoria. DB tamaño byte, DW tamaño WORD y DD tamaño DWORD.

Ejemplo:

MSG_ERROR DB 'has cometido un error, zoquete' ;reserva para MSG constante de tamaño byte con valor 255, este es un caso especial de la directiva DB, que también pude ser usada para declarar cadenas de caracteres.

PESO_MEDIO DW ? ; Reserva para PESO_MEDIO tamaño DWORD pero no inicializa el valor (?)

n DUP reserva tantas posiciones del tamaño que se indique (DB, DW, DD) como indique n.

Ejemplo:

```
.MODEL SMALL
.STACK 100H
.DATA
max EQU 100
cad DB max DUP ?
dac DB max DUP ?
.CODE
MOV AX, @DATA
MOV DS, AX
…
…
END
```

Anexo

SEGMENT y **ENDS** definen los límites de un segmento. Las definiciones de SEGMENT deben terminar con la sentencia ENDS. El formato es:

nom-seg SEGMENT *[tipo-alin] [tipo-combi] ['clase']*

…

…

…

nom-seg ENDS

tipo-alin indica en qué tipo de zona debe comenzar el segmento cuando se almacene en memoria: así se hace que comience en cualquier lugar con BYTE, en una dirección par con WORD, en una posición divisible entre 16 con PARA o divisible por 256 con PAGE. *tipo-combi* indica el tipo de

combinación con otros segmentos con el mismo nombre, así con PUBLIC indica concatenación y con COMMON indica solape, el de pila debe ser forzosamente STACK.

'clase' afecta al orden en que se almacenan los segmentos, el ensamblador almacena de forma contigua todos los segmentos que tengan la misma clase.
ASSUME indica al ensamblador qué registro de segmento (CS, DS, ES o SS) corresponde a cada segmento declarado. El formato es:
n*ombre* ASSUME reg-seg:nom-seg, …
ASSUME reg-seg:NOTHING ; cancela cualquier ASSUME anterior para el registro especificado

Unidad 3

Formato general de un programa en lenguaje ensamblador

El programa ensamblador es el programa que realiza la traducción de un programa escrito en ensamblador a lenguaje máquina. Esta traducción es directa e inmediata, ya que las instrucciones en ensamblador no son más que nemotécnicos de las instrucciones máquina que ejecuta directamente la CPU.

3.1.- Procedimiento para generar un programa ejecutable.

Un programa en ensamblador está compuesto por líneas, conteniendo cada una de ellas un comentario, una única instrucción o una directiva. En lo que sigue se utilizan los símbolos <> para encerrar un identificador o un número que el programador debe escribir; los símbolos [] encierran elementos opcionales; los símbolos {} encierran elementos que se puede escribir consecutivamente varias veces; el carácter | separa elementos opcionales.

Comentarios
Una línea con un asterisco (*) en la primera columna se interpretará como un comentario.
Ej:
* Esto son dos líneas de comentario porque el carácter
* De la primera columna es un asterisco

Un comentario también puede escribirse al final de la misma línea en que se escribe una instrucción. Ej:

ADD D0,D1 Comentario sobre la instrucción ADD

Instrucciones

```
[<etiq>] <Cód_ope>.<tamaño>  <Fuente>,<Dest.>  [<Coment.>]
```

La etiqueta etiq es opcional; si existe debe empezar en la primera columna; si no existe debe haber por lo menos un espacio en blanco antes el código de operación. El comentario también es opcional.

No todas las instrucciones tienen dos operandos. Las hay con uno solo, e incluso sin ningún operando. Ejemplos:

```
Etiq1    MOVE.W D0,D1    Copiar el contenido de D0 en D1
         ADD.W    #3,D2
         NEG.W    D2
         BEQ.S    Etiq1
         NOP
```

Directivas

Controlan acciones auxiliares que realiza el programa ensamblador. No son traducibles a código máquina, sino que indican al ensamblador las preferencias del programador de cómo ha de efectuarse la traducción a lenguaje máquina. Ejemplo:

```
Variable    DS.B    1
```

`Variable` es la etiqueta, `DS.B` es la directiva y `1` es su argumento.

Control de ensamblado

IDNT: Identificación del programa (Identification). Debe ser la primera línea (no comentario) del programa. Tiene dos argumentos: los números de la versión y la revisión del programa.

Ej: Programa1 IDNT 1,0

ORG: Establece en qué posición de memoria absoluta se cargará el programa (ORiGin).

Ej:

ORG $2000

END: Marca el final del programa. A partir de esta directiva se ignora el resto del archivo.

Ej: END

Definición de símbolos

```
<Símbolo>  EQU <Valor>
```

Sustituye **<Símbolo>** por **<Valor>** en el resto del programa (EQUal).

Definición y reserva de memoria

```
<Símbolo>  DC.<tamaño>  <Valor>[{,Valor}]
```

DC (Define Constant) reserva una o varias posiciones de memoria, cada una de <tamaño>, inicializándola con <valor> . En el resto del programa podremos referirnos a esta posición de memoria por su <símbolo>.

```
<Símbolo>  DS.<tamaño>  <Cantidad>
```

DS (Define Storage) reserva **<Cantidad>** posiciones de memoria de tamaño **<tamaño>**, sin asignarles ningún contenido inicial. En el resto del programa podremos referirnos a la posición inicial de este espacio de memoria por su **<Símbolo>**.

Bases de numeración

Un número puede expresarse en diferentes bases de numeración, precediéndolo del carácter que indica la base:

& (o nada) decimal

$	hexadecimal
%	binario
@	octal
"	cadenas ASCII

Se recomienda trabajar siempre en hexadecimal, ya que esta es la base de numeración que emplea el simulador al mostrar las direcciones de memoria y el contenido de la memoria y de los registros.

3.2.- Debugger.

Debugger is a graphical user interface for the Erlang interpreter, which can be used for debugging and testing of Erlang programs. For example, breakpoints can be set, code can be single-stepped and variable values can be displayed and changed.

The Erlang interpreter can also be accessed through the interface module int(3)
It is assumed that the reader is familiar with the Erlang programming language. Modules to be debugged must include debug information, for example, erlc +debug_info MODULE.erl.
Getting Started
To use Debugger, the basic steps are as follows:
Step 1. Start Debugger by calling debugger:start(). The Monitor window is displayed with information about all debugged processes, interpreted modules, and selected options. Initially there are normally no debugged processes. First, it must be specified which modules that are to be debugged (also called interpreted). Proceed as follows:
Step 2. Select Module > Interpret... in the Monitor window. The Interpret Modules window is displayed.
Step 3. Select the appropriate modules from the Interpret Dialog window.
Step 4. In the Monitor window, select Module > the module to be interpreted > View. The contents of the source file is displayed in the View Module window.
Step 5. Set the breakpoints, if any.
Step 6. Start the program to be debugged. This is done the normal way from the Erlang shell.
All processes executing code in interpreted modules are displayed in the Monitor window.

Step 7. To attach to one of these processes, double-click it, or select the process and then choose Process > Attach. Attaching to a process opens an Attach Process window for this process.

Step 8. From the Attach Process window, you can control the process execution, inspect variable values, set breakpoints, and so on.

Breakpoints and Break Dialog Windows

Once the appropriate modules are interpreted, breakpoints can be set at relevant locations in the source code. Breakpoints are specified on a line basis. When a process reaches a breakpoint, it stops and waits for commands (Step, Skip, Continue ...) from the user.

Breakpoints are created and deleted using the Break menu of either the Monitor window, View Module window, or Attach Process window.

Executable Lines

To have an effect, a breakpoint must be set at an executable line, which is a line of code containing an executable expression such as a matching or a function call. A blank line or a line containing a comment, function head, or pattern in a case statement or receive statement is not executable. In the following example, lines 2, 4, 6, 8, and 11 are executable lines:

```
1: is_loaded(Module,Compiled) ->
2:   case get_file(Module,Compiled) of
3:     {ok,File} ->
4:       case code:which(Module) of
5:         ?TAG ->
6:           {loaded,File};
7:         _ ->
8:           unloaded
9:       end;
10:    false ->
11:      false
12:  end.
```

Status and Trigger Action

A breakpoint can be either active or inactive. Inactive breakpoints are ignored.

Each breakpoint has a trigger action that specifies what is to happen when a process has reached it (and stopped):

• Enable - Breakpoint is to remain active (default).

• Disable - Breakpoint is to be made inactive.

• Delete - Breakpoint is to be deleted.

Line Breakpoints

A line breakpoint is created at a certain line in a module.

Figure 2.1: Line Break Dialog Window

Right-click the Module entry to open a popup menu from which the appropriate module can be selected. A line breakpoint can also be created (and deleted) by double-clicking the line when the module is displayed in the View Module window or Attach Process window.

Conditional Breakpoints

A conditional breakpoint is created at a certain line in the module, but a process reaching the breakpoint stops only if a specified condition is true.

The condition is specified by the user as a module name CModule and a function name CFunction. When a process reaches the breakpoint, CModule:CFunction(Bindings) is evaluated. If and only if this function call returns true, the process stops. If the function call returns false, the breakpoint is silently ignored. Bindings is a list of variable bindings. To retrieve the value of Variable (given as an atom), use function int:get_binding(Variable,Bindings). The function returns unbound or {value,Value}.

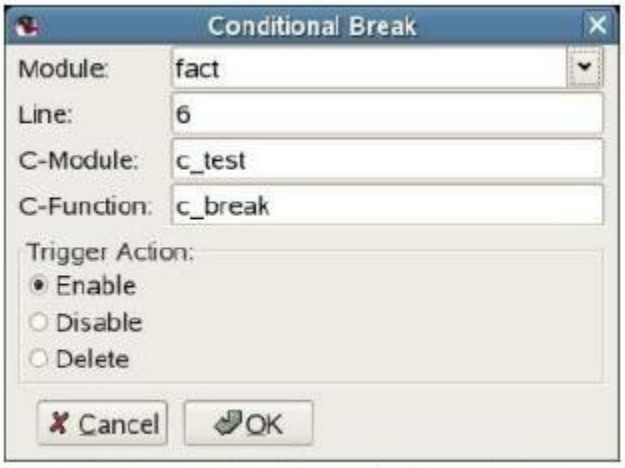


Figure 2.2: Conditional Break Dialog Window

Right-click the Module entry to open a popup menu from which the appropriate module can be selected. Example:

A conditional breakpoint calling c_test:c_break/1 is added at line 6 in module fact. Each time the breakpoint is reached, the function is called. When N is equal to 3, the function returns true and the process stops.

Extract from `fact.erl`:

```
5. fac(0) -> 1;
6. fac(N) when N > 0, is_integer(N) -> N * fac(N-1).
```

Definition of `c_test:c_break/1`:

```
-module(c_test).
-export([c_break/1]).

c_break(Bindings) ->
    case int:get_binding('N', Bindings) of
        {value, 3} ->
            true;
        _ ->
            false
    end.
```

Function Breakpoints

A function breakpoint is a set of line breakpoints, one at the first line of each clause in the specified function

Figure 2.3: Function Break Dialog Window

To open a popup menu from which the appropriate module can be selected, right-click the Module entry. To bring up all functions of the module in the listbox, click the OK button (or press the Return or Tab key) when a module name has been specified,.

Stack Trace

The Erlang emulator keeps track of a stack trace, information about recent function calls. This information is used if an error occurs, for example:

```
1> catch a+1.
{'EXIT',{badarith,[{erlang,'+',[a,1],[]},
                  {erl_eval,do_apply,6,[{file,"erl_eval.erl"},{line,573}]},
                  {erl_eval,expr,5,[{file,"erl_eval.erl"},{line,357}]},
                  {shell,exprs,7,[{file,"shell.erl"},{line,674}]},
                  {shell,eval_exprs,7,[{file,"shell.erl"},{line,629}]},
                  {shell,eval_loop,3,[{file,"shell.erl"},{line,614}]}]}}
```

For details about the stack trace, see section Errors and Error Handling in the Erlang Reference Manual. Debugger emulates the stack trace by keeping track of recently called interpreted functions. (The real stack trace cannot be used, as it shows which functions of Debugger have been called, rather than which interpreted functions.) This information can be used to traverse

the chain of function calls, using the Up and Down buttons in the Attach Process window. By default, Debugger only saves information about recursive function calls, that is, function calls that have not yet returned a value (option Stack On, No Tail). Sometimes, however, it can be useful to save all calls, even tail-recursive calls. This is done with option Stack On, Tail. Notice that this option consumes more memory and slows down execution of interpreted functions when there are many tail-recursive calls. To turn off the Debugger stack trace facility, select option Stack Off.

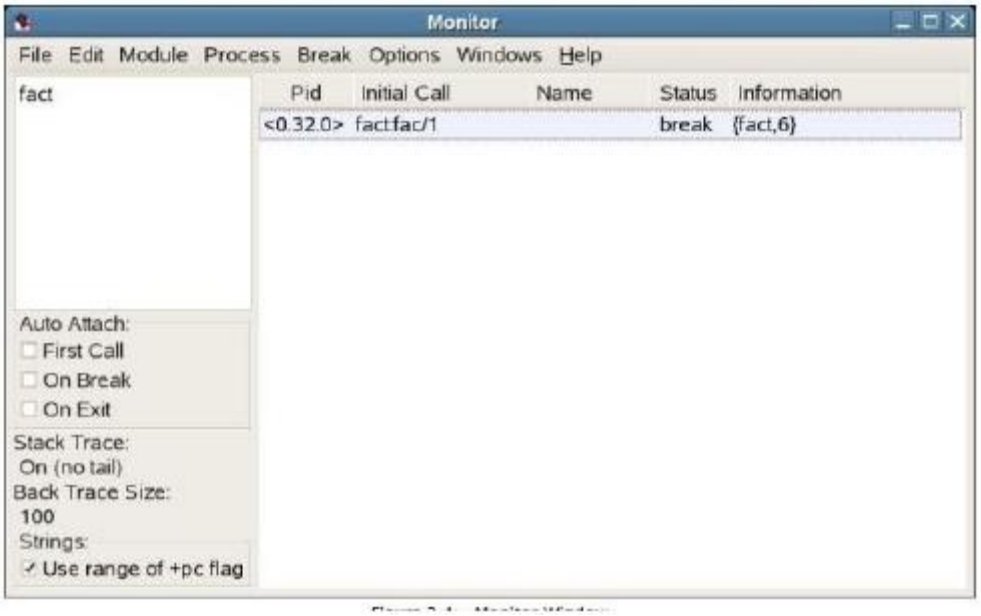
Monitor Window

The Monitor window is the main window of Debugger and displays the following:

• A listbox containing the names of all interpreted modules Double-clicking a module brings up the View Module window.

• Which options are selected

• Information about all debugged processes, that is, all processes that have been or are executing code in interpreted modules



The Auto Attach boxes, Stack Trace label, Back Trace Size label, and Strings box display some options set. For details about these options, see section Options Menu.

Process Grid

Pid

The process identifier.

Initial Call

The first call to an interpreted function by this process. (Module:Function/Arity)

Name

The registered name, if any. If a registered name is not displayed, it can be that Debugger received information about the process before the name was registered. Try selecting Edit > Refresh.

Status

The current status, one of the following:

Idle

The interpreted function call has returned a value, and the process is no longer executing interpreted code.

running

The process is running.

waiting

The process is waiting in a receive statement.

break

The process is stopped at a breakpoint.

exit

The process has terminated.

no_conn

There is no connection to the node where the process is located.

Information

More information, if any. If the process is stopped at a breakpoint, the field contains information about the location {Module,Line}. If the process has terminated, the field contains the exit reason.

File Menu

Load Settings...

Tries to load and restore Debugger settings from a file previously saved using Save Settings... (see below). Any errors are silently ignored

. Notice that settings saved by Erlang/OTP R16B01 or later cannot be read by Erlang/OTP R16B or earlier.

Save Settings...

Saves Debugger settings to a file. The settings include the set of interpreted files, breakpoints, and the selected options. The settings can be restored in a later Debugger session using Load Settings... (see above). Any errors are silently ignored.

Exit

Stops Debugger.

Edit Menu

Refresh

Updates information about debugged processes. Information about all terminated processes are removed from the window. All Attach Process windows for terminated processes are closed.

Kill All

Terminates all processes listed in the window using exit(Pid,kill).

Module Menu

Interpret...

Opens the Interpret Modules window, where new modules to be interpreted can be specified.

Delete All

Stops interpreting all modules. Processes executing in interpreted modules terminate. For each interpreted module, a corresponding entry is added to the Module menu, with the following submenu:

Delete

Stops interpreting the selected module. Processes executing in this module terminate. View

Opens a View

Module window, displaying the contents of the selected module.

Process Menu

The following menu items apply to the currently selected process, provided it is stopped at a breakpoint (for details,
see section *Attach Process window*):

Step
Next
Continue
Finish

The following menu items apply to the currently selected process:

Attach

Attaches to the process and open an *Attach Process window*.

Kill

Terminates the process using exit(Pid,kill).

Break Menu

The items in this menu are used to create and delete breakpoints. For details, see section *Breakpoints*.

Line Break...

Sets a line breakpoint.

Conditional Break...

Sets a conditional breakpoint.

Function Break...

Sets a function breakpoint.

Enable All

Enables all breakpoints.

Disable All
Disables all breakpoints.
Delete All
Removes all breakpoints.
For each breakpoint, a corresponding entry is added to the **Break** menu, from which it is possible to disable, enable,
or delete the breakpoint, and to change its trigger action.
Options Menu
Trace Window
Sets the areas to be visible in an *Attach Process window*. Does not affect existing Attach Process windows.
Auto Attach
Sets the events a debugged process is to be attached to automatically. Affects existing debugged processes.
• **First Call** - The first time a process calls a function in an interpreted module.
• **On Exit** - At process termination.
• **On Break** - When a process reaches a breakpoint.

Stack Trace
Sets the stack trace option, see section *Stack Trace*. Does not affect existing debugged processes.

• **Stack On, Tail** - Saves information about all current calls.
• **Stack On, No Tail** - Saves information about current calls, discarding previous information when a tail
recursive call is made.
• **Stack Off** - Does not save any information about current calls.
Strings
Sets the integer lists to be printed as strings. Does not affect existing debugged processes.
• **Use range of +pc flag** - Uses the printable character range set by the erl(1) flag *+pc*.
Back Trace Size...
Sets how many call frames to be fetched when inspecting the call stack from the Attach Process window. Does
not affect existing Attach Process windows.
Windows Menu
Contains a menu item for each open Debugger window. Selecting one of the items raises the corresponding window.
Help Menu
Help
Shows the Debugger documentation. This function requires a web browser.

3.3.- Introducción a las interrupciones.

Una interrupción es una operación que suspende la ejecución de un programa de modo que el sistema pueda realizar una acción especial. La rutina de interrupción ejecuta y por lo regular regresa el control al procedimiento que fue interrumpido, el cual entonces reasume su ejecución.

Tabla de servicio de interrupción.

Cuando la computadora se enciende, el BIOS y el DOS establecen una tabla de servicios de interrupción en las localidades de memoria 000H-3FFH. La tabla permite el uso de 256 (100H) interrupciones, cada una con un desplazamiento: segmento relativo de cuatro bytes en la forma IP:CS. El operando de una instrucción de interrupción como INT 05H identifica el tipo de solicitud. Como existen 256 entradas, cada una de cuatro bytes, la tabla ocupa los primeros 1, 024 bytes de memoria, desde 000H hasta 3FFH. Cada dirección en la tabla relaciona a una ruina de BIOS o del DOS para un tipo específico de interrupción. Por lo tanto los bytes 0-3 contienen la dirección para la interrupción 0, los bytes 4-7 para la interrupción 1, y así sucesivamente:

INT 00H	INT 01H	INT 02H	INT 03H	INT 04H	INT 05H	INT 06H	. . .
IP:CS	IP:CS	IP:CS	IP:CS	IP:CS	IP:CS	IP:CS	. . .
00H	04H	08H	0CH	10H	14H	18H	. . .

Eventos de una interrupción.

Una interrupción guarda en la pila el contenido del registro de banderas, el CS, y el IP. por ejemplo, la dirección en la tabla de INT 05H (que imprime la que se encuentra en la pantalla cuando el usuario presiona Ctrl + PrtSC) es 0014H (05H x 4 = 14H). La operación extrae la dirección de cuatro bytes de la posición 0014H y almacena dos bytes en el IP y dos en el CS. La dirección CS:IP entonces apunta al inicio de la rutina en el área del BIOS, que ahora se ejecuta. La interrupción regresa vía una instrucción IRET (regreso de interrupción), que saca de la pila el IP, CS y las banderas y regresa el control a la instrucción que sigue al INT

Tipos de interrupciones.

Las interrupciones se dividen en dos tipos las cuales son: Externas y Internas. Una interrupción externa es provocada por un dispositivo externo al procesador. Las dos líneas que pueden señalar interrupciones externas son la línea de interrupción no enmascarable (NMI) y la línea de petición de interrupción (INTR). La línea NMI reporta la memoria y errores de paridad de E/S. El procesador siempre actúa sobre esta interrupción, aun si emite un CLI para limpiar la bandera de interrupción en un intento por deshabilitar las interrupciones externas. La línea INTR reporta las peticiones desde los dispositivos externos, en realidad, las interrupciones 05H a la 0FH, para cronometro, el teclado, los puertos seriales, el disco duro, las unidades de disco flexibles y los puertos paralelos. Una interrupción interna ocurre como resultado de la ejecución de una instrucción INT o una operación de división que cause desbordamiento, ejecución en modo de un paso o una petición para una interrupción externa, tal como E/S de disco. Los programas por lo

común utilizan interrupciones internas, que no son enmascarables, para acezar los procedimientos del BIOS y del DOS.

3.4.1.- Interrupciones de software.

El BIOS contiene un extenso conjunto de rutinas de entrada/salida y tablas que indican el estado de los dispositivos del sistema. El dos y los programas usuarios pueden solicitar rutinas del BIOS para la comunicación con los dispositivos conectados al sistema. El método para realizar la interfaz con el BIOS es el de las interrupciones de software. A continuación se listan algunas interrupciones del BIOS.

INT 00H: División entre cero. Llamada por un intento de dividir entre cero. Muestra un mensaje y por lo regular se cae el sistema.

INT 01H: Un solo paso. Usado por DEBUG y otros depuradores para permitir avanzar por paso a través de la ejecución de un programa.

INT 02H: Interrupción no enmascarare. Usada para condiciones graves de hardware, tal como errores de paridad, que siempre están habilitados. Por lo tanto un programa que emite una instrucción CLI (limpiar interrupciones) no afecta estas condiciones.

INT 03H: Punto de ruptura. Usado por depuración de programas para detener la ejecución.

INT 04H: Desbordamiento. Puede ser causado por una operación aritmética, aunque por lo regular no realiza acción alguna.

INT 05H: Imprime pantalla. Hace que el contenido de la pantalla se imprima. Emita la INT 05H para activar la interrupción internamente, y presione las teclas Cltr + PrtSC para activarla externamente. La operación permite interrupciones y guarda la posición del cursor.

INT 08H: Sistema del cronometro. Una interrupción de hardware que actualiza la hora del sistema y (si es necesario) la fecha. Un chip temporizador programable genera una interrupción cada 54.9254 milisegundos, casi 18.2 veces por segundo.

INT 09H: Interrupción del teclado. Provocada por presionar o soltar una tecla en el teclado.

INT OBH, INT OCH: Control de dispositivo serial. Controla los puertos COM1 y COM2, respectivamente.

INT 0DH, INT OFH: Control de dispositivo paralelo. Controla los puertos LPT1 y LPT2, respectivamente.

INT 0EH: Control de disco flexible. Señala actividad de disco flexible, como la terminación de una operación de E/S. INT 10H: Despliegue en vídeo. Acepta el número de funciones en el AH para el modo de pantalla, colocación del cursor, recorrido y despliegue.

INT 11H: Determinación del equipo. Determina los dispositivos opcionales en el sistema y regresa el valor en la localidad 40:10H del BIOS al AX. (A la hora de encender el equipo, el sistema ejecuta esta operación y almacena el AX en la localidad 40:10H).

INT 12H: Determinación del tamaño de la memoria. En el AX, regresa el tamaño de la memoria de la tarjeta del sistema, en términos de kilobytes contiguos.

INT 13H: Entrada/salida de disco. Acepta varias funciones en el AH para el estado del disco, sectores leídos, sectores escritos, verificación, formato y obtener diagnóstico.

3.4.2.- Interrupciones de hardware

Los dos módulos del DOS, IO.SYS y MSDOS.SYS, facilitan el uso del BIOS. Ya que proporcionan muchas de las pruebas adicionales necesarias, las operaciones del DOS por lo general son más fáciles de usar que sus contrapartes del BIOS y por lo común son independientes de la máquina.

IO.SYS es una interfaz de nivel bajo con el BIOS que facilita la lectura de datos desde la memoria hacia dispositivos externos.

MSDOS.SYS contiene un administrador de archivos y proporciona varios servicios. Por ejemplo, cuando un programa usuario solicita la INT 21H, la operación envía información al MSDOS.SYS por medio del contenido de los registros. Para completar la petición, MSDOS.SYS puede traducir la información a una o más llamadas a IO.SYS, el cual a su vez llama al BIOS. Las siguientes son las relaciones implícitas:

Las interrupciones desde la 20H hasta la 3FH están reservadas para operaciones del DOS. A continuación se mencionan algunas de ellas.

INT 20H: Termina programa. Finaliza la ejecución de un programa .COM, restaura las direcciones para Cltr + Break y errores críticos, limpia los bufer de registros y regresa el control al DOS. Esta función por lo regular seria colocada en el procedimiento principal y al salir del, CS contendría la dirección del PSP. La terminación preferida es por medio de la función 4CH de la INT 21H.

INT 21H: Petición de función al DOS. La principal operación del DOS necesita una función en el AH.

INT 22H: Dirección de terminación. Copia la dirección de esta interrupción en el PSP del programa (en el desplazamiento 0AH) cuando el DOS carga un programa para ejecución. A la terminación del programa, el DOS transfiere el control a la dirección de la interrupción. Sus programas no deben de emitir esta interrupción.

INT 23H: Dirección de Cltr + Break. Diseñada para transferir el control a una rutina del DOS (por medio del PSP desplazamiento 0EH) cuando usted presiona Ctlt + Break o Ctlr + c. La rutina finaliza la ejecución de un programa o de un archivo de procesamiento por lotes. Sus programas no deben de emitir esta interrupción.

INT 24H: Manejador de error crítico. Usada por el dos para transferir el control (por medio del PSP desplazamiento 12H) cuando reconoce un error critico (a veces una operación de disco o de la impresora).Sus programas no deben de emitir esta interrupción.

INT 25H: Lectura absoluta de disco. Lee el contenido de uno o más sectores de disco.

INT 26H: Escritura absoluta de disco. Escribe información desde la memoria a uno o más sectores de disco.

INT 27H: Termina pero permanece residente (reside en memoria). Hace que un programa .COM al salir permanezca residente en memoria.

INT 2FH: Interrupción de multiplexion. Implica la comunicación entre programas, como la comunicación del estado de un spooler de la impresora, la presencia de un controlador de dispositivo o un comando del DOS tal como ASSIGN o APPEND.

INT 33H: Manejador del ratón. Proporciona servicios para el manejo del ratón.

3.5.- Programación modular (MACROS).

Junto con todo lo visto anteriormente, y como se mencionó anteriormente, uno de los componentes que caracterizan los computadores personales es su sistema operativo. Una PC puede correr varios sistemas operativos: CP/M, CP/M-86, XENIX, Windows, PC-DOS, y MS-DOS. Lo que los define es la forma en que están integrados sus servicios y la forma en que se accesa a ellos. Esto es precisamente lo que el linker debe enlazar y resolver. Aquí nos enfocaremos exclusivamente en el sistema operativo MS-DOS, y lo que se mencione aquí será válido para las versiones 3.0 y superiores. Este sistema operativo está organizado de la siguiente manera:

· Comandos Internos (reconocidos y ejecutados por el COMMAND.COM) Comandos Externos (.EXEs y .COMs) Utilerías y drivers (programas de administración del sistema) Shell (Interfaz amigable, sólo versiones 4.0 o mayores) Servicios (Interrupciones)

Los servicios, más conocidos como interrupciones o vectores de interrupción, es parte medular de lo que es MS-DOS, y no son más que rutinas definidas por MS-DOS y el BIOS, ubicadas a partir de una localidad de memoria específica. La manera de accesar a estas rutinas y a los servicios que ofrecen es mediante una instrucción que permite ejecutar una interrupción.

MS-DOS proporciona dos módulos: IBMBIO.COM (provee una interfaz de bajo nivel para el BIOS) e IBMDOS.COM (contiene un manejador de archivos y servicios para manejo de registros). En equipos compatibles estos archivos tienen los nombres IO.SYS y MSDOS.SYS, respectivamente. El acceso a los servicios del computador se realiza de la siguiente manera:

Programa DOS DOS ROM EXTERNO
De usuario Alto nivel Bajo nivel
Petición de — IBMDOS.COM — IBMBIO.COM — BIOS — Dispositivo I/O

ENSAMBLADORES Y MACROENSAMBLADORES.

Existen varios ensambladores disponibles para ambiente MS-DOS: el IBM Macro Assembler, el Turbo Assembler de Borland, el Turbo Editassm de Speedware, por citar algunos. Una breve descripción de cada uno se propociona a continuación.

Macro Ensamblador IBM.- Está integrado por un ensamblador y un macroensamblador. En gran medida su funcionamiento y forma de invocarlo es sumamente similar al de Microsoft. Su forma de uso consiste en generar un archivo fuente en código ASCII, se procede a generar un programa objeto que es ligado y se genera un programa .EXE. Opcionalmente puede recurirse a la utilería EXE2BIN de MS-DOS para transformarlo a .COM. Es capaz de generar un listado con información del proceso de ensamble y referencias cruzadas.

Macro Ensamblador de Microsoft.- Dependiendo de la versión, este ensamblador es capaz de soportar el juego de instrucciones de distintos tipos de microprocesadores Intel de la serie 80xx/80x86. En su versión 4.0 este soporta desde el 8086 al 80286 y los coprocesadores 8087 y 80287. Requiere 128KB de memoria y sistema operativo MS-DOS v2.0 o superior. Trabaja con un archivo de código fuente creado a partir de un editor y grabado en formato ASCII. Este archivo es usado para el proceso de ensamble y generación de código objeto. Posteriormente, y con un ligador, es creado el código ejecutable en formato .EXE.

Turbo Editassm.- Este es desarrollado por Speddware, Inc., y consiste de un ambiente integrado que incluye un editor y utilerías para el proceso de ensamble y depuración. Es capaz de realizar el ensamble línea a línea, conforme se introducen los mnemónicos, y permite revisar listas de referencias cruzadas y contenido de los registros. Este ensamblador trabaja con tablas en memoria, por lo que la generación del código ejecutable no implica la invocación explícita del

ligador por parte del programador. Adicionalmente permite la generación de listados de mensajes e información de cada etapa del proceso y la capacidad de creación de archivos de código objeto.

Turbo Assembler.- De Borland Intl., es muy superior al Turbo Editassm. Trabaja de la misma forma, pero proporciona una interfaz mucho más fácil de usar y un mayor conjunto de utilerías y servicios. En lo que se refiere a las presentes notas, nos enfocaremos al Microsoft Macro Assembler v4.0. Los programas ejemplo han sido desarrollados con éste y está garantizado su funcionamiento. Estos mismo programas posiblemente funcionen con otros ensambladores sin cambios o con cambios mínimos cuando utilizan directivas o p s e u d o i n s t r u c c i o n e s .

Realmente la diferencia entre los ensambladores radica en la forma de generar el código y en las directivas con que cuente, aunque estas diferencias son mínimas. El código ensamblador no cambia puesto que los microprocesadores con los que se va a trabajar son comunes. Así, todos los programas que se creen con un ensamblador en particular podrán ser ensamblados en otro, cambiando las pseudo-operaciones no reconocidas por el equivalente indicado en el manual de referencia del paquete empleado. Los programas que componen el Macro Ensamblador Microsoft v4.0 son los siguientes:

Programa Descripción
MASM.EXE Microsoft Macro Assembler
LINK.EXE Microsoft 8086 object linker
SYMDEB.EXE Microsoft Symbolic Debuger Utility
MAPSYM.EXE Microsoft Symbol File Generator
CREF.EXE Microsoft Cross-Reference Utility
LIB.EXE Microsoft Library Manager
MAKE.EXE Microsoft Program Maintenance Utility
EXEPACK.EXE Microsoft EXE File Compression Utility
EXEMOD.EXE Microsoft EXE File Header Utility
COUNT.ASM Sample source file for SYMDEB session
README.DOC Updated information obtained after the manual was printed.

El Microsoft Macro Assembler v4.0 crea código ejecutable para procesadores 8086, 8088, 80186, 80188, 80286, 8087 y 80287. Además es capaz de aprovechar las instrucciones del 80286 en la creación de código protegido y no protegido. El término macroensamblador es usado para indicar que el ensamblador en cuestión tiene la capacidad de poder ensamblar programas con facilidad de

macro. Una macro es una pseudo-instrucción que define un conjunto de instrucciones asociadas a un nombre simbólico. Por cada ocurrencia en el código de esta macro, el ensamblador se encarga de substituir esa llamada por todas las instrucciones asociadas y, en caso de existir, se dejan los parámetros con los que se estaba llamando la macro y no con los que había sido definida. Es importante señalar que no se deja una llamada, como a una subrutina o procedimiento, sino que se incorporan todas las instrucciones que definen a la macro.

UN EJEMPLO

Para comenzar veamos un pequeño ejemplo que ilustra el formato del programa fuente. Este ejemplo está completamente desarrollado en lenguaje ensamblador que usa servicios o funciones de MS-DOS (system calls) para imprimir el mensaje Hola mundo!! en pantalla.

```
; HOLA.ASM
; Programa clasico de ejemplo. Despliega una leyenda en pantalla.
STACK SEGMENT STACK ; Segmento de pila
DW 64 DUP (?) ; Define espacio en la pila
STACK ENDS
DATA SEGMENT ; Segmento de datos
SALUDO DB "Hola mundo!!",13,10,"$" ; Cadena
DATA ENDS
CODE SEGMENT ; Segmento de Codigo
ASSUME CS:CODE, DS:DATA, SS:STACK
INICIO: ; Punto de entrada al programa
MOV AX,DATA ; Pone direccion en AX
MOV DS,AX ; Pone la direccion en los registros
MOV DX,OFFSET SALUDO ; Obtiene direccion del mensaje
MOV AH,09H ; Funcion: Visualizar cadena
INT 21H ; Servicio: Funciones alto nivel DOS
MOV AH,4CH ; Funcion: Terminar
INT 21H
CODE ENDS
END INICIO ; Marca fin y define INICIO
```

La descripción del programa es como sigue:

1.- Las declaraciones SEGMENT y ENDS definen los segmentos a usar.

2.- La variable SALUDO en el segmento DATA, define la cadena a ser desplegada. El signo del dolar al final de la cadena (denominado centinela) es requerido por la función de visualización de la cadena de MS-DOS. La cadena incluye los códigos para carriage-return y line-feed.

3.- La etiqueta START en el segmento de código marca el inicio de las instrucciones del programa.

4.- La declaracion DW en el segmento de pila define el espacio para ser usado por el stack del programa.

5.- La declaración ASSUME indica que registros de segmento se asociarán con las etiquetas declaradas en las definiciones de segmentos.

6.- Las primeras dos instrucciones cargan la dirección del segmento de datos en el registro DS. Estas instrucciones no son necesarias para los segmentos de código y stack puesto que la dirección del segmento de código siempre es cargado en el registro CS y la dirección de la declaración del stack segment es automáticamente cargada en el registro SS.

7.- Las últimas dos instrucciones del segmento CODE usa la función 4CH de MS-DOS para regresar el control al sistema operativo. Existen muchas otras formas de hacer esto, pero ésta es la más recomendada.

8.- La directiva END indica el final del código fuente y específica a START como punto de arranque.

EL FORMATO DEL ENSAMBLADOR.

De acuerdo a las convenciones y notación seguidas en el manual del Microsoft Macro Assembler.

Negritas Comandos, símbolos y parámetros a ser usados como se muestra.

Itálicas Todo aquello que debe ser reemplazado por el usuario

ð ð Indican un parámetro opcional

„ Denota un parámetros que puede repetirse varias veces

¦ Separa dos valores mutuamente excluyentes

letra chica Usada para ejemplos. Código y lo que aparece en pantalla.

Cada programa en lenguaje ensamblador es creado a partir de un archivo fuente de código ensamblador. Estos son archivos de texto que contienen todas las declaraciones de datos e instrucciones que componen al programa y que se agrupan en áreas o secciones, cada una con un propósito especial. Las sentencias en ensamblador tienen la siguiente sintaxis: [nombre> mnemónico [operandos> [;comentarios>

En cuanto a la estructura, todos los archivos fuente tienen la misma forma: cero o más segmentos de programa seguidos por una directiva END. No hay una regla sobre la estructura u orden que deben seguir las diversas secciones o áreas en la creación del código fuente de un programa en ensamblador. Sin embargo la mayoría de los programas tiene un segmento de datos, un segmento de código y un segmento de stack, los cuales pueden ser puestos en cualquier lugar.

Para la definición de datos y declaración de instrucciones y operandos el MASM reconoce el conjunto de caracteres formado por letras mayúsculas, letras minúsculas (excluyendo caracteres acentuados, ñ, Ñ), números, y los símbolos: ? @ _ $: . [> ( ) { } + - / * & % ! ´ ~ ¦ \ = # ; , " `

La declaración de números requiere tener presente ciertas consideraciones. En el MASM un entero se refiere a un número entero: combinación de dígitos hexadecimales, octales, decimales o binarios, más una raíz opcional. La raíz se especifica con B, Q u O, D, o H. El ensamblador usará siempre la raíz decimal por defecto, si se omite la especificación de la raíz (la cual se puede cambiar con la directiva .RADIX). Así nosotros podemos especificar un entero de la siguiente manera: digitos, digitosB, digitosQ o digitosO, digitosD, digitosH. Si una D o B aparecen al final de un número, éstas siempre se considerarán un especificador de raíz, e.g. 11B será tratado como 112 (210), mientras que si se trata del número 11B16 debe introducirse como 11Bh.

Para los números reales tenemos al designador R, que sólo puede ser usado con números hexadecimales de 8, 16, ó 20 digitos de la forma digitosR. También puede usarse una de las directivas DD, DQ, y DT con el formato [+¦->digitos.digitos[E[+¦->digitos>. Las cadenas de carácter y constantes alfanuméricas son formadas como ´caracteres´ o "caracteres" . Para referencias simbólicas se utilizan cadenas especiales denominadas nombres. Los nombres son cadenas de caracteres que no se entrecomillan y que deben comenzar con una A..Z ¦ a..z ¦ _ ¦ $ ¦ @ los caracteres restantes pueden ser cualquiera de los permitidos, y solamente los 31 primeros caracteres son reconocidos.

DIRECTIVAS.

El MASM posee un conjunto de instrucciones que no pertenecen al lenguaje ensamblador propiamente sino que son instrucciones que únicamente son reconocidas por el ensambldor y que han sido agregadas para facilitar la tarea de ensablamblado, tanto para el programador como para el programa que lo lleva a cabo. Dichas instrucciones son denominadas directivas. En general, las directivas son usadas para especificar la organización de memoria, realizar ensamblado

condicional, definir macros, entrada, salida, control de archivos, listados, cross-reference, direcciones e información acerca de la estructura de un programa y las declaraciones de datos.

* Conjunto de instrucciones.- Dentro de las directivas más importantes, tenemos las que establecen el conjunto de instrucciones a soportar para un microprocesador en especial: .8086(default).- Activa las instrucciones para el 8086 y 8088 e inhibe las del 80186 y 8 0 2 8 6 . .8087(default).- Activa instrucciones para el 8087 y desactiva las del 80287.

.186.- Activa las instrucciones del 80186.
.286c.- Activa instrucciones del 80286 en modo no protegido.
.289p.- Activa instrucciones del 80286 en modo protegido y no protegido.
.287.- Activa las instrucciones para el 80287.

* Declaración de segmentos.- En lo que respecta a la estructura del programa tenemos las directivas SEGMENT y ENDS que marcan el inicio y final de un segmento del programa. Un segmento de programa es una colección de instrucciones y/o datos cuyas direcciones son todas relativas para el mismo registro de segmento. Su sintaxis es:

nombre SEGMENT [alineación> [combinación> [´clase´>
nombre ENDS

El nombre del segmento es dado por nombre, y debe ser único. Segmentos con el mismo nombre se tratan como un mismo segmento. Las opciones alineación, combinación, y clase proporcionan información al LINK sobre cómo ajustar los segmentos. Para alineación tenemos los siguientes valores: byte (usa cualquier byte de dirección), word (usa cualquier palabra de dirección, 2 bytes/word), para (usa direcciones de parráfos, 16 bytes/parráfo, deafult), y page (usa direcciones de página, 256 bytes/page). combinación define cómo se combinarán los segmentos con el mismo nombre. Puede asumir valores de: public (concatena todos los segmentos en uno solo), stack (igual al anterior, pero con direcciones relativas al registro SS, common (crea segmentos sobrepuestos colocando el inicio de todos en una misma dirección), memory (indica al LINK tratar los segmentos igual que MASM con public, at address (direccionamiento relativo a address). clase indica el tipo de segmento, señalados con cualquier nombre. Cabe señalar que en la definición está permitido el anidar segmentos, pero no se permite de ninguna manera el sobreponerlos.

* Fin de código fuente.- Otra directiva importante es la que indica el final de un módulo. Al alcanzarla el ensamblador ignorará cualquier otra declaración que siga a ésta. Su sintaxis es:

END [expresión> la opción expresión permite definir la dirección en la cual el programa iniciará.

* Asignación de segmentos.- La directiva ASSUME permite indicar cuáles serán los valores por default que asimirán los registros de segmento. Existen dos formas de hacer esto:

ASSUME registrosegmento:nombre,,,
ASSUME NOTHING
NOTHING cancela valores previos.

* Etiquetas.- Las etiquetas son declaradas nombre:
donde nombre constituye una cadena de caracteres. * Declaración de datos.- Estos se declaran según el tipo, mediante la regla [nombre> directiva valor,,, donde directiva puede ser DB (bytes), DW (palabras), DD (palabra doble), DQ (palabra cuádruple), DT (diez bytes). También pueden usarse las directivas LABEL (crea etiquetas de instrucciones o datos), EQU (crea símbolos de igualdad) , y el símbolo = (asigna absolutos) para declarar símbolos. Estos tienen la siguiente sintaxis:
nombre = expresión
nombre EQU expresión
nombre LABEL tipo
donde tipo puede ser BYTE, WORD, DWORD, QWORD, TBYTE, NEAR, FAR.

* Declaración de estructuras.- Para la declaración de estructuras de datos se emplea la directiva STRUC. Su sintaxis es:
nombre STRUC
campos
nombre ENDS

CONJUNTO DE INSTRUCCIONES.
El juego completo de instrucciones reconocidas por los procesadores intel 8086 a 80286, junto con los coprocesadores 8087 y 80287, se enlistan en el apendice E. Como puede verse en dicho apéndice, la mayoría de las instrucciones requieren algunos operandos o expresiones para trabajar, y lo cual es válido también para las directivas. Los operandos representan valores,

registros o localidades de memoria a ser accesadas de alguna manera. Las expresiones combinan operandos y operadores aritméticos y lógicos para calcular en valor o la dirección a acceder.

Los operandos permitidos se enlistan a continuación:

Constantes.- Pueden ser números, cadenas o expresiones que representan un valor fijo. Por ejemplo, para cargar un registro con valor constante usaríamos la instrucción MOV indicando el registro y el valor que cargaríamos dicho registro.

```
mov ax,9
mov al,´c´
mov bx,65535/3
mov cx,count
```

count sólo sera válido si este fue declarado con la directiva EQU.

Directos.- Aquí se debe especificar la dirección de memoria a accesar en la forma segmento:offset.

```
mov ax,ss:0031h
mov al,data:0
mov bx,DGROUP:block
```

Relocalizables.- Por medio de un símbolo asociado a una dirección de memoria y que puede ser usado también para llamados.

```
mov ax, value
call main
mov al,OFFSET dgroup:tabla
mov bx, count
```

count sólo será válido si fue declarado con la directiva DW.

Contador de localización.- Usado para indicar la actual localización en el actual segmento durante el ensamblado. Representado con el símbolo $ y también conocido como centinela.

```
help DB ´OPCIONES´,13,10
F1 DB ´ F1 salva pantalla´,13,10
.
.
.
F10 DB ´ F10 exit´,13,10,´$
DISTANCIA = $-help
```

Registros.-

Cuando se hace referencia a cualquiera de los registros de propósito general, apuntadores, índices, o de segmento.

Basados.- Un operador basado representa una dirección de memoria relativa a uno de los registros de base (BP o BX). Su sintaxis es:

desplazamiento[BP>

desplazamiento[BX>

[desplazamiento>[BP>

[BP+desplazamiento>

[BP>.desplazamiento

[BP>+desplazamiento en cada caso la dirección efectiva es la suma del desplazamiento y el contenido del registro.

mov ax,[BP>

mov al,[bx>

mov bx,12[bx>

mov bx,fred[bp>

Indexado.- Un operador indexado representa una dirección de memoria relativa a uno de los registros índice (SI o DI). Su sintaxis es:

desplazamiento[DI>

desplazamiento[SI>

[desplazamiento>[DI>

[DI+desplazamiento>

[DI>.desplazamiento

[DI>+desplazamiento en cada caso la dirección efectiva es la suma del desplazamiento y el contenido del registro.

mov ax,[si>

mov al,[di>

mov bx,12[di>

mov bx,fred[si>

Base-indexados.- Un operador base-indexado representa una dirección de memoria relativa a la combinación de los registros de base e índice. Su sintaxis es:

desplazamiento[BP>[SI>

desplazamiento[BX>[DI>

desplazamiento[BX>[SI>

desplazamiento[BP>[DI>

[desplazamiento>[BP>[DI>

[BP+DI+desplazamiento>

[BP+DI>.desplazamiento [DI>+desplazamiento+[BP> en cada caso la dirección efectiva es la suma del desplazamiento y el contenido del registro.

mov ax,[BP>[si>

mov al,[bx+di>

mov bx,12[bp+di>

mov bx,fred[bx>[si>

Estructuras.- Su sintaxis es variable.campo. variable es el nombre con que se declaró la estructura, y campo es el nombre del campo dentro de la estructura.

date STRUC mes DW ?

dia DW ?

aa DW ?

date ENDS

actual date ´ja´,´01´,´84´

mov ax,actual.dia

mov actual.aa, ´85´

Operadores y expresiones.- Se cuenta con los siguientes operadores:

-aritméticos

expresión1 * expresión2

expresión1 / expresión2

expresión1 MOD expresión2

expresión1 + expresión2

expresión1 - expresión2

+ expresión

- expresión

-de corrimiento

expresión1 SHR contador

expresión1 SHL contador

-relacionales

expresión1 EQ expresión2

expresión1 NE expresión2

expresión1 LT expresión2

expresión1 LE expresión2

expresión1 GT expresión2

expresión1 GE expresión2

- de bit

NOT expresión

expresión1 AND expresión2

expresión1 OR expresión2

expresión1 XOR expresión2

-de índice

[expresión1> [expresión2>

ejemplos:

mov al, string[3>

mov string[last>,al

mov cx,dgroup:[1> ; igual a mov cx,dgroup:1

-de apuntador

tipo PTR expresión

tipo puede ser BYTE ó 1, WORD ó 2, DWORD ó 4, QWORD ó 8, TBYTE ó 10, NEAR ó 0FFFFh, FAR ó 0FFFEh. Ejemplos:

call FAR PTR subrout3

mov BYTE ptr [array>, 1

add al, BYTE ptr [full_word>

-de nombre de campo

estructura.campo

ejemplos:

inc month.day

mov time.min,0

mov [bx>.dest

-de propósito especial.

OFFSET expresion.- Regresa el desplazamiento del operando

mov bx, OFFSET dgroup:array

mov bx, offset subrout3

SHORT etiqueta.- Para un salto de menos de 128 bytes

jmp SHORT loop

LENGTH variable.- Regresa el número de elementos de variable según su tipo mov cx,length array SIZE variable.- Regresa el tamaño en bytes alojados para variable mov cx,size array

SEG expresión.- Regresa el valor del segmento para expresión mov ax, SEG saludo

MACROS Y PROCEDIMIENTOS.

La manera más fácil de modularizar un programa es dividirlo en dos o más partes. Para esto, es necesario que datos, símbolos, y demás valores de un módulo sean reconocidos por el otro u otros módulos. Para este tipo de declaraciones globales existen dos directivas:

PUBLIC nombre,,, que hace la variable, etiqueta o símbolo absoluto disponible para todos los programas.

EXTRN nombre: tipo,,, que especifica una variable, etiqueta o símbolo externos identificados por nombre y tipo (que puede ser BYTE, WORD, DWORD, QWORD, TBYTE, NEAR, FAR, o ABS, éste último para números absolutos). El siguiente ejemplo ilustra el uso de las directivas. El primer listado corresponde al módulo principal, mientras que el segundo al módulo que contiene una rutina. Ambos módulos son archivos que se editan por separado, se ensamblan por separado, pero se ligan juntos.

Unidad 4

Señales en los pines del microprocesador 80x86

4.1.- Generador de reloj.

Al igual que los más recientes microprocesadores, el 8086 requiere una única señal de reloj. Este microprocesador no genera su propia señal de reloj siendo necesaria la utilización del generador de reloj 8284, que usa un cristal oscilador para determinar la frecuencia de señal. Intercambiando este cristal, se puede seleccionar diferentes velocidades de operación. Intel tiene una versión de 5 MHz y otra de 8 Mhz para el 8086. Estas versiones representan las velocidades más altas, recomendables para este chip.

Para un rendimiento óptimo, el 8086 requiere una señal de reloj que se mantenga a tensión alta una tercera parte del tiempo total de ciclo. Esto significa que el reloj está activo una tercera parte del tiempo y desactivado las dos terceras partes del tiempo.

En la búsqueda de una instrucción, su dirección se obtiene sumando el desplazamiento, contenido en el IP, al valor del registro de segmento CS, multiplicado por 16.

Normalmente al terminar la ejecución de una instrucción, el IP se incrementa, con lo que se pasa a direccionar la siguiente instrucción. Las instrucciones de salto y salto a subrutinas pueden modificar el contenido de IP de tres formas diferentes:

· Por direccionamiento relativo. Al contenido del IP se suma, de forma inmediata, un desplazamiento de 8 a 16 bits, con signo, proporcionado por la misma instrucción.

· Por direccionamiento directo. Se carga, en el IP, una nueva dirección presente en la instrucción.

· Por direccionamiento indirecto. El dato, obtenido por cualquiera de las formas de direccionado de la memoria de datos, es interpretado por las instrucciones de salto, como la dirección a la que se debe saltar.

Modos de direccionamiento de la memoria de datos.

· Modo inmediato. El operando se proporciona en el byte o bytes que siguen al código de operación de la instrucción.

Ejemplo: ADD CX,385Fh

· Modo de direccionado por registro. Un registro, definido por la instrucción, contiene el operando.

Ejemplo: ADD CX,AX

· Modo directo. El byte o par de bytes que siguen al código OP de la instrucción dan el desplazamiento de 8 ó 16 bits, que, sumado al contenido del registro DS, determina la dirección efectiva en la que se encuentra el dato a transferir.

Ejemplo: ADD CL,TABLA

· Modo directo indexado. El byte o par de bytes que siguen al código OP representan un desplazamiento que se suma al contenido de uno de los registros índice (DI o SI). El contenido de DS se añade al resultado de la suma, con lo que se obtiene la dirección del operando.

Incrementando o decrementando los registros índice, se puede acceder a posiciones de memoria consecutivas.

Ejemplo: ADD CX, [SI+4]

· Modo indirecto. La dirección del operando es el contenido de uno de los siguientes registros: BP, BX, DI o SI.

Ejemplo: ADD CX, [BX]

· Por registro base indexado. El desplazamiento que ha de sumarse a un registro segmento se halla sumando el contenido de un registro índice y un desplazamiento de 8 ó 16 bits, contenido en la instrucción, al contenido de un registro base.

Ejemplo: MOV AX,TABLA[BX][SI]

· Modo relativo a base: El byte o par de bytes que siguen al código OP representan un desplazamiento que se suma al contenido de uno de los registros base (BX o BP). El contenido de DS se añade al resultado de la suma, con lo que se obtiene la dirección del operando.

Ejemplo: MOV AX, [BP]+4

Formato de las instrucciones.

A semejanza con otros microprocesadores, las instrucciones del 8086 pueden clasificarse en tres grupos, según la definición que se utilice para los operandos:

· Sin operando. Por ejemplo, CLI, STI, DAA, WAIT, etc.

· Con un sólo operando. Por ejemplo, JMP, PUSH, CALL, etc.

· Con dos operandos. Por ejemplo, MOV.

Las instrucciones varían de 1 a 6 bytes en longitud. Los desplazamientos y los datos inmediatos pueden tener 8 ó 16 bits dependiendo de la instrucción. El código de operación y el modo de direccionamiento se encuentran en los primeros uno o dos bytes de la instrucción. Estos pueden ir seguidos por:

· Ningún byte adicional.

· Una EA de 2 bytes (sólo para direccionamiento directo).

· Un desplazamiento de 1 ó 2 bytes.

· Un operando inmediato de 1 ó 2 bytes.

· Un desplazamiento de 1 ó 2 bytes seguidos de un operando inmediato de 1 ó 2 bytes.

· Un desplazamiento de 2 bytes y una dirección de un segmento de 2 bytes (sólo para direccionamientos fuera del segmento).

El código de operación y el modo de direccionamiento determinan cual de estas posibilidades es la utilizada. Si un desplazamiento o un operando inmediato son de 2 bytes, siempre aparece primero el byte de menor orden.

El código de operación, que generalmente es el primer byte de la instrucción (aunque hay algunas veces en las que este byte designa un registro y otras en las que 3 bits del código de operación están en el segundo byte), es el encargado de determinar la operación, informando a su vez del tamaño de los operandos. En la mayoría de los códigos de operación hay indicadores especiales de 1 bit. Estos indicadores son:

Bit W. Nos informa del tamaño de los operandos, es decir, si se trata de byte o palabra. Con W = 0 estamos accediendo a un byte y con W = 1 a una palabra.

Bit D. Este bit es utilizado con instrucciones de dos operandos, salvo en los casos en que uno de los operandos debe ser un registro especificado en el campo REG.

Este bit nos informa si el registro especificado en el campo REG se trata del operando fuente u operando destino de instrucción especificada por el código de operación.

Para D = 0, el registro indicado por el campo REG es el operando fuente y para D = 1, el registro indicado por el campo REG es el operando destino.

Bit S. El bit S aparece junto con el bit W en instrucciones de suma inmediata registro / memoria, resta y comparación. El bit S indica el tamaño del dato u operando inmediato, el bit W nos indica el tamaño del dato u operando destino. Si SW es 00, tanto la fuente como destino tienen 8 bits, es decir, se trata de una operación de 8 bits; Si SW es 11 se trata de una operación de 16 bits con un operando inmediato de 16 bits con extensión de signo; Si SW es 01 indica una operación de 16 bits con un operando inmediato de 8 bits con extensión de signo.

Bit V. Utilizado por instrucciones de desplazamiento y rotación para determinar el número de desplazamientos.

Bit Z. Utilizado para instrucciones REP.

A continuación se muestra la combinación de bits para asignación de registros (de segmento o cualquier otro tipo).

Dirección de registro	Registros W = 1	Registros W = 0
000	AX	AL
001	CX	CL
010	DX	DL
011	BX	BL
100	SP	AH
101	BP	CH
110	SI	DH
111	DI	BH

Dirección de registro	Registro de segmento
00	ES
01	CS
10	SS
11	DS

Como se puede apreciar en estas tablas se podría producir ambigüedad al representar, por ejemplo, 00 para indicar el registro de segmento ES, 000 para indicar AX y a su byte de menor orden AL , pero esto no sucede ya que el código de operación implica el tipo de registro que indica.

Cuando el código de operación y el modo de direccionamiento ocupan 2 bytes pueden verse representado de las dos formas siguientes:

MOD	COD OP	R/M

MOD	REG	R/M

El primero de los casos es para instrucciones de un sólo operando o bien instrucciones de dos operandos pero uno de ellos implícito en el código de operación.

El segundo de los casos se trata de instrucciones de dos operandos, en cuyo caso uno de ellos viene especificado por el campo REG y el otro operando viene especificado por los campos MOD y R/M que pueden indicar un registro o bien una posición de memoria.

A continuación se especifica una tabla donde quedan reflejados los modos de direccionamiento y registros de segmento por defecto para varias combinaciones de los campos MOD y R/M.

MOD R/M	00	01	10	11 W=0	 W=1
000 Reg. de segmento	(BX) + (SI) DS	(BX)+(SI)+D8 DS	(BX)+(SI)+D16 DS	AL	AX
001 Reg. de segmento	(BX) + (DI) DS	(BX)+(DI)+D8 DS	(BX)+(DI)+D16 DS	CL	CX
010 Reg. de segmento	(BP) + (SI) SS	(BP)+(SI)+D8 SS	(BP)+(SI)+D16 SS	DL	DX
011 Reg. de segmento	(BP) + (DI) SS	(BP)+(DI)+D8 SS	(BP)+(DI)+D16 SS	BL	BX
100 Reg. de segmento	(SI) DS	(SI) + D8 DS	(SI) + D16 DS	AH	SP
101 Reg. de segmento	(DI) DS	(DI) + D8 DS	(DI) + D16 DS	CH	BP
110 Reg. de segmento	D16 DS	(BP) + D8 SS	(BP) + D16 SS	DH	SI
111 Reg. de segmento	(BX) DS	(BX) + D8 DS	(BX) + D16 DS	BH	DI

Viendo esta tabla podemos apreciar que según la combinación de MOD y R/M tendremos diferentes posibilidades para el segundo operando. Como anteriormente se ha indicado este segundo operando puede ser un registro o una posición de memoria, que principalmente va a estar en función del campo MOD. Si MOD = 11 nos indica que se trata de un registro, pudiendo ver de qué tipo de registro se trata según la combinación de R/M. Si MOD es distinto de 11 nos indica una posición de memoria y la dirección efectiva se calcula según las tablas. Si nos fijamos en MOD = 00 esto significa que no hay desplazamiento a menos que R/M = 110, lo que implicaría direccionamiento directo. Si MOD = 01 indica que el tercer byte de la instrucción contiene un desplazamiento de 8 bits que se extiende automáticamente a 16 (extensión de signo) antes de ser usado para calcular la dirección efectiva, y MOD = 10 indica que los bytes tercero y cuarto contienen un desplazamiento de 16 bits.

Podemos observar en la tabla el registro de segmento que es utiliza para cada una de las combinaciones de MOD y R/M. Aunque la dirección efectiva de un operando en memoria está determinada por los campos MOD y R/M, la dirección física de 20 bits se obtiene, como ya vimos, sumando la dirección efectiva y el contenido de un registro
de segmento multiplicado por 16.

Existe un byte especial de prefijo para permitir excepciones a la utilización del registro de segmento dado por la tabla anterior, es decir, nos permite ignorar algunas de estas asignaciones pero no todas, este byte se denomina prefijo de sustitución de segmento. En particular, si la CPU usa la instrucción puntero para ayudarse a apuntar a memoria, es decir, para localizar parte de una instrucción, entonces el registro de segmento de código siempre se usa. Y si la CPU usa el puntero de pila para ayudarse a apuntar a memoria, tanto para introducir como para extraer de la pila, entonces se usa

Siempre el registro de segmento de pila. El caso del destino en operaciones con cadena es el único en el cual hay una restricción. La combinación del índice de destino (DI) y el segmento extra (ES) se usa siempre para calcular la dirección del destino en cualquier operación con cadenas. El byte de prefijo de sustitución de segmento puede usarse, sin embargo, para obligar a utilizar alguno de los cuatro registros de segmento en el cálculo de la dirección puente de una operación de cadenas, resumiendo, los casos específicos en los que no puede haber sustituciones son:

· El CS se utiliza siempre como registro de segmento para el cálculo de la próxima instrucción que se va a ejecutar.

· Cuando se utiliza SP siempre se utiliza SS como registro de segmento.

· Para operaciones con cadenas siempre se utiliza ES como registro de segmento del operando destino.

Los 24 modos de referenciar la dirección, usados para el acceso de datos, pueden aceptar un byte de prefijo de sustitución de segmento e ignorar sus asignaciones por defecto al segmento, usando cualquier registro de segmento. La asignación por defecto como se puede ver en la tabla es o el segmento de datos (DS) o el de pila (SS) y, de hecho, se usa siempre el segmento de datos a menos que el modo de direccionamiento use el puntero base (BP), en cuyo caso se usa el registro de pila. Intel aconseja utilizar el puntero base para acceder a datos en la pila del sistema y normalmente con llamadas a subrutinas.

Para examinar con más claridad las instrucciones máquina del 8086, vamos a ver unos ejemplos con la instrucción suma (ADD). Una suma (ADD) provoca que el contenido de la posición indicada para el operando fuente sea sumado al contenido de la posición indicada para el operando destino, y que la suma reemplace a ésta última. Esta instrucción puede adoptar uno de

los tres formatos que se muestran en la siguiente figura, dependiendo del modo de direccionamiento.

a) Suma de un registro con un registro o con memoria, y almacenamiento del resultado en un registro o en una memoria.

000000DW	MOD	REG	R/M	DESP menor orden	DESP mayor orden

* El campo DESP puede ocupar uno o dos bytes, dependiendo del campo MOD.

b) Suma inmediata con registro (memoria) y colocación del resultado en un registro (memoria).

100000SW	MOD	000	R/M	DESP bajo	DESP alto	DAT bajo	DAT alto

* El campo DESP puede ocupar uno o dos bytes, dependiendo del campo MOD.
* El byte alto del campo DAT es opcional, y estará presente si S:W = 01.

c) Suma inmediata con AX(AL) y almacenamiento del resultado en AX(AL). Caso especial para el acumulador.

0000010W	DATO menor orden	DATO mayor orden

La siguiente figura muestra el código en el lenguaje máquina para dos instrucciones ADD, las cuales suman los contenidos de los registros BH y CL, y colocan el resultado en CL. En la primera, D = 1 indica que REG = 001 = CL es donde será almacenada la suma. MOD = 11, por lo que R/M designa un registro, el registro 111 = BH. En la segunda instrucción, D = 0, lo que hace que REG = !!! = BH sea la fuente. De nuevo MOD = 11, y R/M designa a un registro que en este caso es el 001 = CL.

	D	W	MOD	REG	R/M
000000	1	0	11	001	111

D = 1 Indica que el destino es un registro.
W = 0 Operandos de byte (suma en 8 bits).
MOD = 11 La fuente será un registro que quedará determinado por R/M.
REG = 001 Indica el registro CL.
R/M = 111 Indica el registro BH.

a) REG indica destino.

	D	W	MOD	REG	R/M
000000	0	0	11	111	001

D = 0 Indica que la fuente es un registro.

W = 0 Operandos de byte.

MOD = 11 La fuente será un registro que quedará determinado por R/M.

REG = 111 Indica el registro BH.

R/M = 001 Indica el registro CL.

b) REG indica fuente.

El siguiente ejemplo indica una suma de una posición de memoria con un registro. La dirección efectiva se obtiene sumando los contenidos de BX y DI al desplazamiento de 16 bits (W = 1), que es 2345. Si (BX) = 0892 y (DI) = 59A3, entonces EA = 0892 + 59A3 +2345 = 857A.

	D	W	MOD	REG	R/M	DESPLAZAMIENTO
000000	0	1	10	010	001	01000101 00100011

D = 0 Indica que la fuente es un registro.

W = 1 Operandos de palabra.

MOD = 10 La fuente que será memoria quedará determinada por el campo R/M.

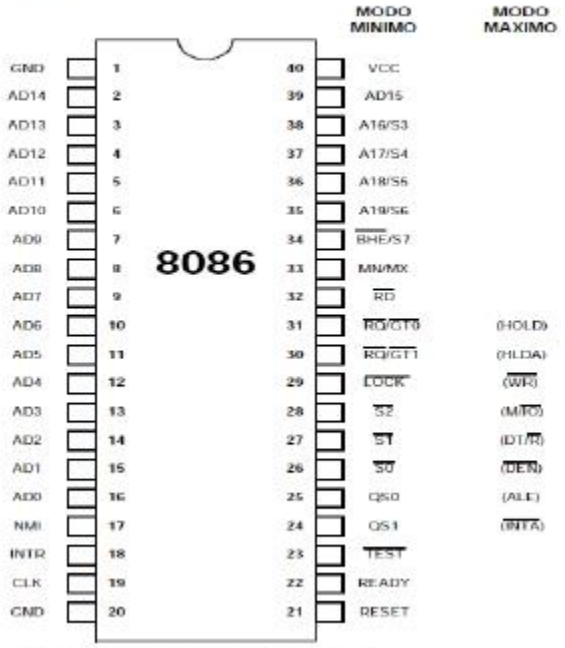
REG = 010 Indica el registro DX.

R/M = 001 Indica una posición de memoria que será:

EA = (BX) + (DI) + desplazamiento de 16 bits.

c) Suma de un registro con memoria.

DIAGRAMA DE CONEXIONADO DEL 8086. SEÑALES Y TERMINALES.



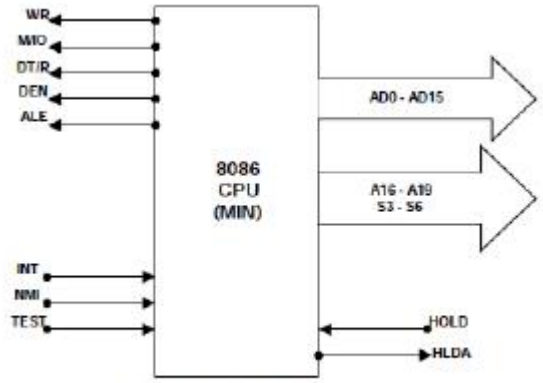
MICROPROCESADOR 8086

El 8086 puede configurarse de dos formas distintas: el modo máximo y el modo mínimo. El modo queda determinado al colocar el terminal MN/MX a tierra o a la tensión de alimentación.

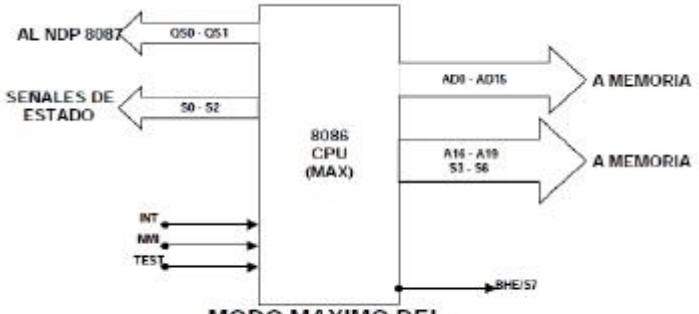
En modo mínimo no admite la multitarea, mientras que en modo máximo es capaz de soportar un bus local, para ampliar directamente el 8086, y un bus de sistema MULTIBUS, que permite configuraciones con varios procesadores, más concretamente el 8086 debe estar en modo máximo si quiere trabajar en colaboración con el procesador de datos 8087 y el procesador de entrada / salida 8089.

En el modo máximo, el 8086 depende de otros chips adicionales como es el controlador de bus 8288 para generar el conjunto completo de señales de control de bus. El modo mínimo permite al 8086 trabajar de una forma más autónoma.

La figura siguiente muestra un esquema de ambos modos.



MODO MINIMO DEL 8086



MODO MAXIMO DEL 8086

En ambos modos, las señales del 8086 se pueden agrupar de la siguiente manera:

· Alimentación.

· Reloj.

· Control y estado.

· Direcciones.

· Datos.

Hay tres terminales para la alimentación: tierra (GND) en los terminales 1 y 20, y una tensión de entrada de 5 voltios (Vcc) en el terminal 40. El terminal de tierra es tierra a la vez para la alimentación y para las señales.

Cuenta con una entrada de la señal de reloj (CLK) en el terminal 19.

El 8086 cuenta con 20 bits de dirección. Los 4 bits más significativos de la dirección comparten terminales con algunas de las señales de estado. Los 16 bits menos significativos son multiplexados tanto para las direcciones como para los datos, es decir, en ciertos instantes tales terminales conducen parte de una dirección, y en otros son transmitidos los datos.

Estos terminales pueden llevar información de una dirección e información sobre el estado y los datos. El latch 8282 está diseñado para seleccionar la información sobre la dirección de dichos terminales en el instante preciso e ignorar lo referente al estado y los datos.

Hay varios grupos de control y señales de estado.

El terminal **MN / MX** controla si el procesador está en modo mínimo o máximo, conectándolo a tierra o a una tensión de 5 voltios.

Del **S0 al S7** son señales de estado en los terminales 26, 27, 28, 38, 37, 36, 35, 34 respectivamente. En ciertos momentos son salidas del procesador. En otros momentos aparecen otras señales distintas en los mismos terminales. Mirando el estado pueden decirse cosas tales como el tipo de acceso al bus (lectura o escritura, memoria o E/S), el registro de segmento en uso y el estado del sistema de interrupciones. **S0, S1 y S2** son sólo accesibles en modo máximo, en cuyo caso se introducen en los chips controladores de bus 8288. Estas señales decodifican el estado del procesador, de acuerdo con la siguiente tabla.

S2	S1	S0	ESTADO DE LA CPU
0	0	0	RECONOCIMIENTO DE INTERRUPCION
0	0	1	LECTURA PUERTA I/O
0	1	0	ESCRITURA PUERTA I/O
0	1	1	PARO (HALT)
1	0	0	CODIGO DE ACCESO A INSTRUCCION
1	0	1	LECTURA MEMORIA
1	1	0	ESCRITURA MEMORIA
1	1	1	PASIVO. NO OPERA.

El controlador de bus 8288 genera a partir de estas, otras importantes señales de control.

En el modo mínimo no es preciso el controlador de bus 8288 puesto que el propio procesador genera por si sólo algunas de estas señales de control.

La señal **RD** es una señal de estado generada por el procesador sobre el terminal 32. Indica un ciclo de lectura de memoria o entradas y salidas.

La señal **READY** que se encuentra en el terminal 22, es una entrada de los dispositivos externos (memoria o controladores E/S) y su función es adaptar las velocidades de memoria y periféricos a la CPU. Esta señal pasa a través del generador de pulsos 8284 para sincronizarse con la señal de reloj.

La señal **READY** trabaja de la siguiente forma: Si se ha seleccionado un dispositivo externo para lectura o escritura y todavía no está preparado para completar la transferencia de datos, pone a cero la línea de señal READY. El procesador ve esta señal y añade ciclos extras de << espera >> hasta que dicha señal se pone a su nivel normal, es decir, a 1 indicando que el dispositivo externo está preparado para realizar la transferencia. Acabada la transferencia las actividades del procesador continúan normalmente.
La señal **RESET** que se encuentra en el terminal 21, es otra de las entradas que también pasa por el generador de pulsos 8284 para sincronizarse con la señal de reloj. Se utiliza para inicializar el procesador borrando la cola de instrucciones y ciertos registros tales como los indicadores, segmento de datos (DS), segmento de pila (SS), segmento extra (ES) poniéndolos a cero. El puntero de instrucciones (IP) y el segmento de código (CS) los carga con FFFFH.

Los terminales **NMI** (**N**on-Maskable **I**nterrup : Interrupción no enmascarable), terminal 17, e **INTR** (**INT**errupt **R**equest : petición de interrupción), terminal 18, son parte del sistema de interrupciones del 8086. Un pulso en el terminal NMI provoca una interrupción especial, llamada *interrupción tipo 2*. Una señal en el terminal INTR causará una respuesta de interrupción de tipo general. El término << no enmascarables>> se refiere al hecho de que la interrupción generada por el terminal NMI no se puede activar o desactivar vía un software a la CPU. Las interrupciones generales por INTR pueden desactivarse vía software.

El terminal **BHE / S7** se utiliza como ayuda en la interfaz de los dispositivos de 8 bits con el bus de datos de 16 bits. Su funcionamiento es el siguiente: Si la línea de dirección 0 es 0 (indicando una dirección par), la señal BHE especifica si se está direccionando una palabra entera o un sólo byte. BHE igual a 0 significa que se trata de una palabra, BHE igual a 1 significa que es un byte. Si la

línea de dirección 0 es 1 (indicando una dirección impar), el 8086 direcciona siempre un byte y no una palabra. En este caso BHE es 0.

La señal **M / IO** (terminal 28) informa al sistema cuando el microprocesador requiere acceso a la memoria o al espacio de E/S, es decir, indica la realización de una operación sobre memoria ó sobre entrada / salida.

El terminal **WR** de selección de escritura (terminal 29) indica que los datos están disponibles en las líneas de datos, es decir, se trata de la señal que indica un ciclo de escritura de la CPU.

El terminal **DT / R** (terminal 27) cuya misión es la recepción y transmisión de datos, es decir, para controlar la transferencia de datos, el 8086 precisa de la colaboración del circuito auxiliar 8286 (8287). Este se gobierna por la señal DT / R, que indica el sentido del movimiento de la información (transmisión o recepción).

DEN (terminal 26) igual que la anterior sirve para controlar la transferencia de datos y mas concretamente confirma la validación de los datos.

ALE (terminal 25). Esta señal activa el latch 8282 cuando viene una dirección por las líneas AD0-AD15, es decir, sirve para controlar el multiplexado de datos y direcciones.

INTA (terminal 24). La función de esta señal es el reconocimiento de interrupciones. (Se hablará más en el apartado de interrupciones).

Las señales **HOLD** y **HLDA** son parte del propio sistema de control de bus del 8086. **HOLD** (terminal 31) indica la petición del bus por un periférico exterior. Cuando otro procesador o un aparato como un controlador DMA quieren acceder al control del bus, manda una señal al 8086 a través de la línea HOLD. Cuando está preparado para hacerlo, el 8086 pone sus líneas de datos / direcciones y la mayoría de las líneas de control en el estado de alta impedancia. Al mismo tiempo, envía la señal **HLDA** (terminal 30) para indicar que el bus está libre. El otro aparato puede usar ahora el bus. Cuando finaliza con el bus, envía una señal a la línea HOLD. Inmediatamente después de recibir la señal, el 8086 reanuda el uso del bus.
Las señales **RQ / GT1** y **RQ / GT0** (terminales 30 y 31 respectivamente) se corresponden con las señales HOLD y HLDA del modo mínimo. Se utiliza para liberar el bus y reconocer la acción.

Las señales **QS1** y **QS0** (terminales 24 y 25 respectivamente) son señales de estado de las colas de instrucciones. Sólo son utilizables en modo máximo. El procesador de datos 8087 utiliza estas señales para coordinarse con el 8086.

La señal **TEST** (terminal 23) se utiliza para enlazar el 8086 con un procesador paralelo, sincronizando el procesador principal con los otros.

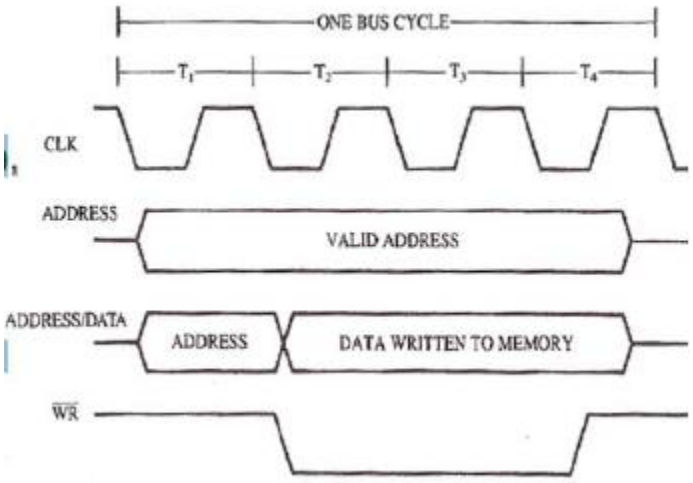
4.2.- Temporización del canal.

Temporización en General

• Las operaciones de transferencia de datos hacia o desde el 8086 ocupan al menos un bus cycle

• Cada bus cycle consiste en 4 períodos de reloj del sistema (T), T1, T2,T3, T4.

• Si el reloj funciona a 5 MHz: - T = 1/5 MHz = 0.2 ms - Bus cycle = 4 T = 0.8 ms=800ns - Velocidad máxima de lect. o esc. de datos de la memoria o del espacio de E/S = 1/0.8 = 1.25 millones de operaciones por seg. - El 8086 puede ejecutar 2.5 millones de inst. por seg.(MIPS) debido a que posee una cola interna



Temporización en General

• T1:

- La dirección de la memoria o del puerto de E/S es enviada por el micro

-Se proporcionan las señales de control ALE, DT/#R y M/#IO que indica si el canal de direcciones contiene una dir. de la memoria o el número de un puerto para un disp. de E/S.

• T2:

- El 8086 proporciona las señales #DEN, #RD para lectura o #WR para escritura * En el caso de escritura, los datos que se van a escribir aparecen en el canal de datos * En el caso de lectura, el canal multiplexado pasa a alta impedancia, para que se puedan colocar los datos a leer.

• T3:

-Este período se produce para dar tiempo a la memoria para acceder a los datos. -Si el ciclo es de Lectura el bus de datos se muestrea al final de T3.

T4:

-Se desactivan todas las señales de canal en preparación para el siguiente ciclo -En operaciones de Escritura, el 8086 muestrea los terminales del canal de datos que se leen de la memoria o de E/S. Además, en este momento, el flanco de subida de WR transfiere datos a la memoria o a E/S, los cuales se escriben cuando la señal WR retorna al nivel al nivel de 1 lógico.

• Wait States: la entrada Ready (Lista) produce estados de espera para componentes de memoria y de E/S más lentos. Un estado de espera (TW) es un período adicional de reloj introducido para alargar el ciclo del canal.

4.3.- Interfaz de memoria.

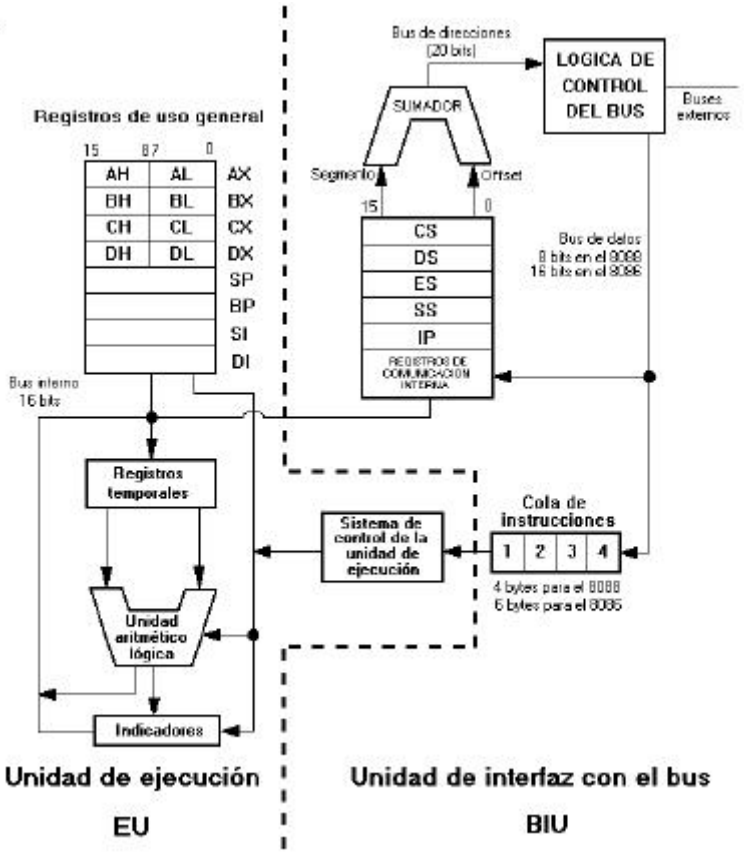
Como ya sabemos, los microprocesadores 8086 y 8088 poseen registros de un tamaño máximo de 16 bits. Con estos 16 bits podemos direccionar como máximo un total de 64 KB de memoria. Pero este microprocesador puede direccionar hasta 1 MB. Si hacemos cuentas, para poder direccionar 1 MB, tendríamos que tener registros de 20 bits. Como este no es el caso, hay que recurrir a algún mecanismo para poder direccionar toda la memoria. Dicho mecanismo consiste en la segmentación que divide la memoria en segmentos de 64 KB. Cada segmento se asocia con un registro de segmento, el desplazamiento (u *offset*) dentro de ese segmento lo proporciona otro registro de 16 bits. La dirección absoluta se calcula multiplicando por 16 (que es lo mismo que desplazar sus bits hacia la izquierda 4 posiciones) el valor del registro de segmento y sumando el desplazamiento. Esto equivale a generar la dirección absoluta como si los registros de segmento tuvieran 4 bits a 0 a la derecha antes de sumarles el desplazamiento:

Dirección = segmento * 16 + desplazamiento

Los registro básicos que posee el 8088 son el **CS,DS,ES y SS**, son registros de 16 bits como el resto de los registros de microprocesador pero su uso interno a la hora de componer direcciones

es un tanto especial. Estos registros no se emplean para acceder a una dirección de memoria directamente, sino que definen una dirección base o de segmento sobre la que aplicar el desplazamiento de 16 bits

- **Registro CS:** Almacena la dirección inicial del segmento de código de un programa. Esta dirección de segmento, más un valor de desplazamiento en el registro apuntador de instrucción (IP), indica la dirección de una instrucción que es buscada para su ejecución.
- **Registro DS:** La dirección inicial de un segmento de datos de programa es almacenada en el registro DS. Esta dirección, más un valor de desplazamiento en una instrucción, genera una referencia a la localidad de un byte específico en el segmento de datos.
- **Registro SS:** El registro SS permite la colocación en memoria de una pila, para almacenamiento temporal de direcciones y datos. El DOS almacena la dirección de inicio del segmento de pila de un programa en el registro SS. Esta dirección de segmento, más un valor de desplazamiento en el registro del apuntador de la pila (SP), indica la palabra actual en la pila que está siendo direccionada.
- **Registro ES:** Algunas operaciones con cadenas de caracteres utilizan el registro extra de segmento para manejar el direccionamiento de memoria.



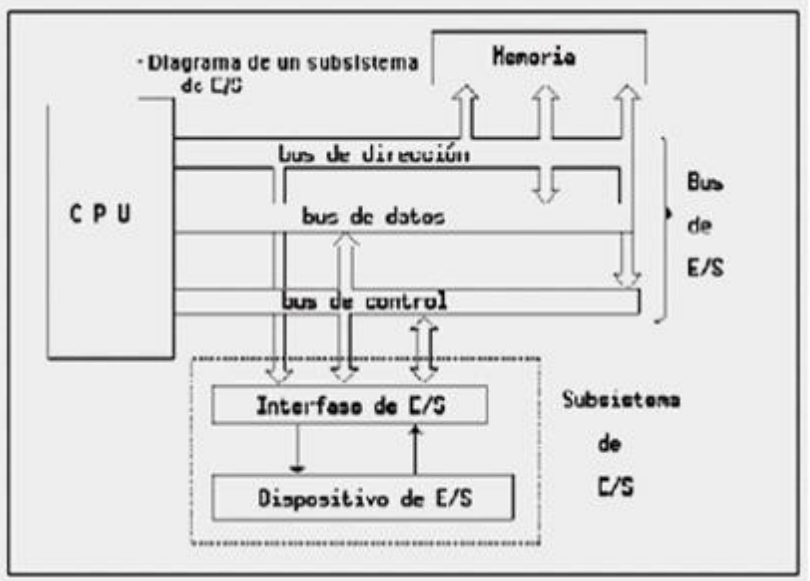
4.4.- Interface de entrada/salida.

La interfaces de entrada y de salida proporciona un método para transferir información entre dispositivos de (E/S) de almacenamiento interno y de (E/S) externas. Los periféricos conectados a una computadora necesitan enlacé de comunicación especial para funcionar como una interfaces con la unidad de procesamiento central.

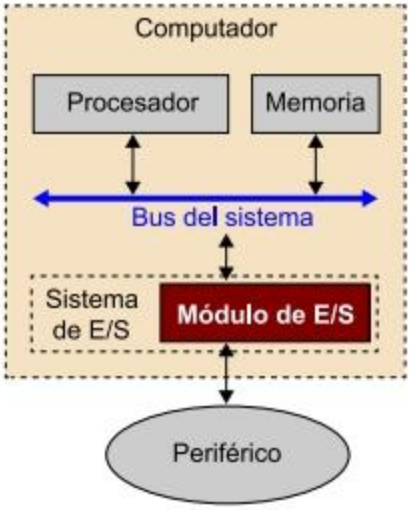
El propósito del enlace de comunicación es resolver las diferencias que existen en la computadora central y de cada periférico.

Se llama interfaces porque se comunica tanto con el canal del procesador como con el dispositivo periférico.

Las funciones de la interfase son almacenar los datos y realizar las conversiones que se le requieran. También detecta errores en la transmisión y es capaz de reiniciar la transacción en casos de error. Más aún, la interfase puede testear, arrancar y detener el dispositivo según las directivas impartidas por la CPU. En algunos casos la interfase puede consultar a la CPU si algún dispositivo está requiriendo atención urgente.



De manera más concreta, toda operación de E/S que se lleva a cabo entre el computador y un periférico es solicitada y gobernada desde el procesador, es decir, es el procesador quien determina en qué momento se debe hacer y con qué periférico, si la operación es de lectura o escritura, qué datos se han de transferir, y también quién da la operación por acabada. Para llevar a cabo la operación de E/S, hemos de conectar el periférico al computador. Para hacerlo, es necesario que el computador disponga de unos dispositivos intermedios por donde ha de pasar toda la información que intercambia el computador con el periférico y que nos permite hacer una gestión y un control correctos de la transferencia. Estos dispositivos los llamamos de manera genérica módulo de E/S.



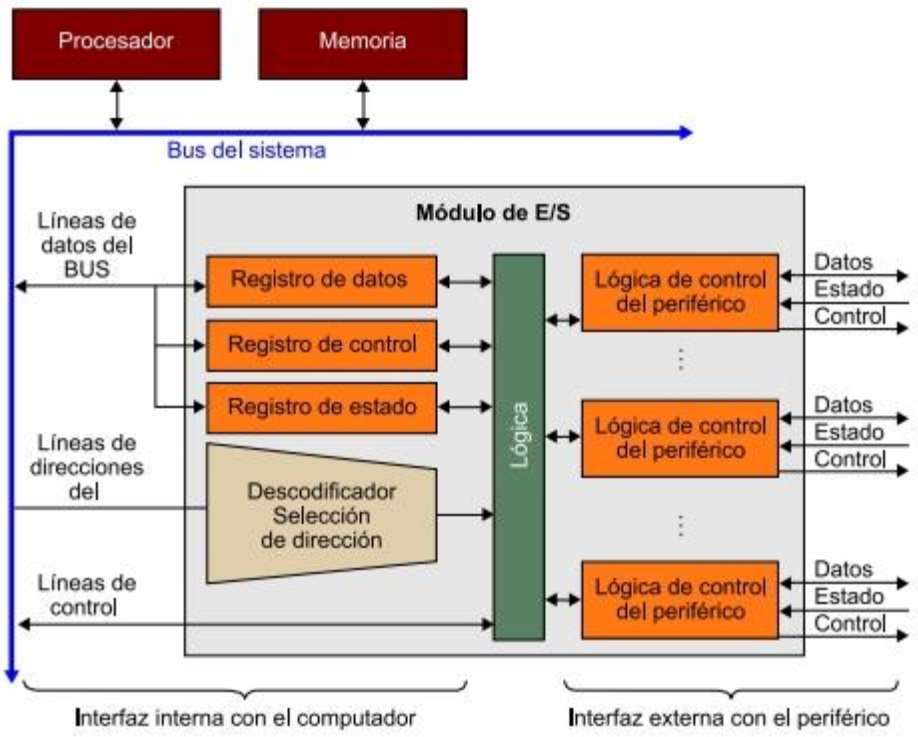
Puede parecer lógico conectar el periférico directamente al bus del sistema del computador, pero esta opción no es factible básicamente por dos razones:

- La necesidad de gestionar una gran variedad de periféricos con unas características muy específicas y diferenciadas. Esto hace muy complejo añadir la lógica necesaria dentro del procesador para gestionar esta gran diversidad de dispositivos.
- La diferencia de velocidad entre sí, en la que, salvo casos excepcionales, el procesador es mucho más rápido que el periférico. Por un lado, hay que asegurar que no se pierdan datos y, por otro, garantizar principalmente la máxima eficiencia del procesador, pero también de los otros elementos del computador.

Cuando queremos hacer la operación de E/S, hemos de diferenciar el caso de una transferencia individual, en la que se transmite un solo dato y el control de la transferencia es muy simple (leer una tecla, mirar si se ha hecho un clic en el ratón), y la transferencia de bloques, que se basa en una serie de transferencias individuales y en la que se necesita un control mayor de todo el proceso (leer un fichero, actualizar el contenido de la pantalla).

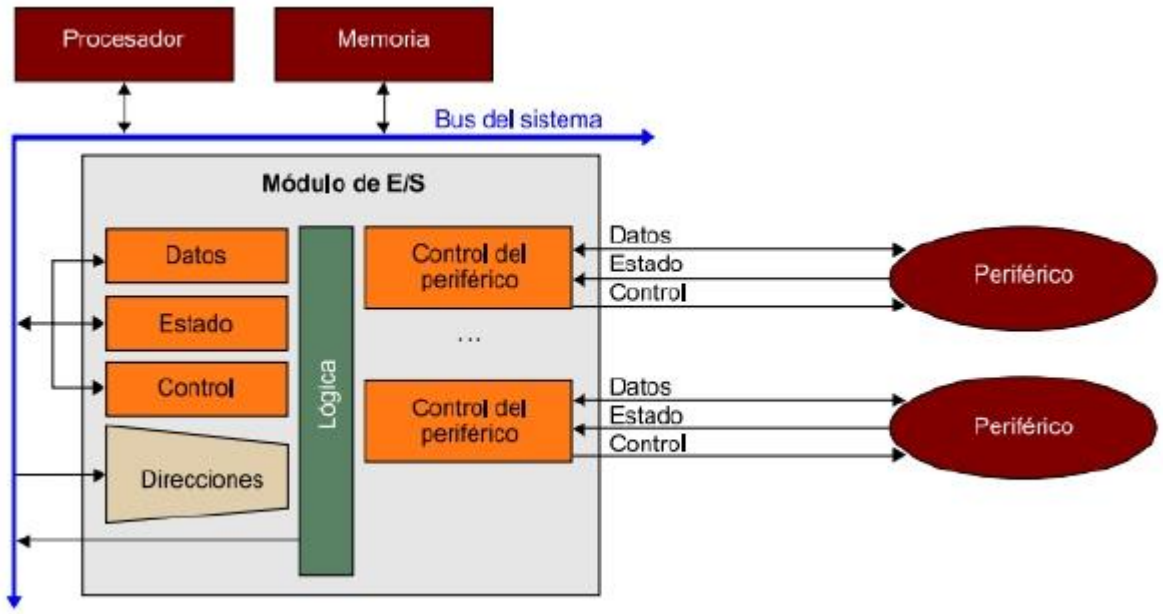
Módulos de E/S

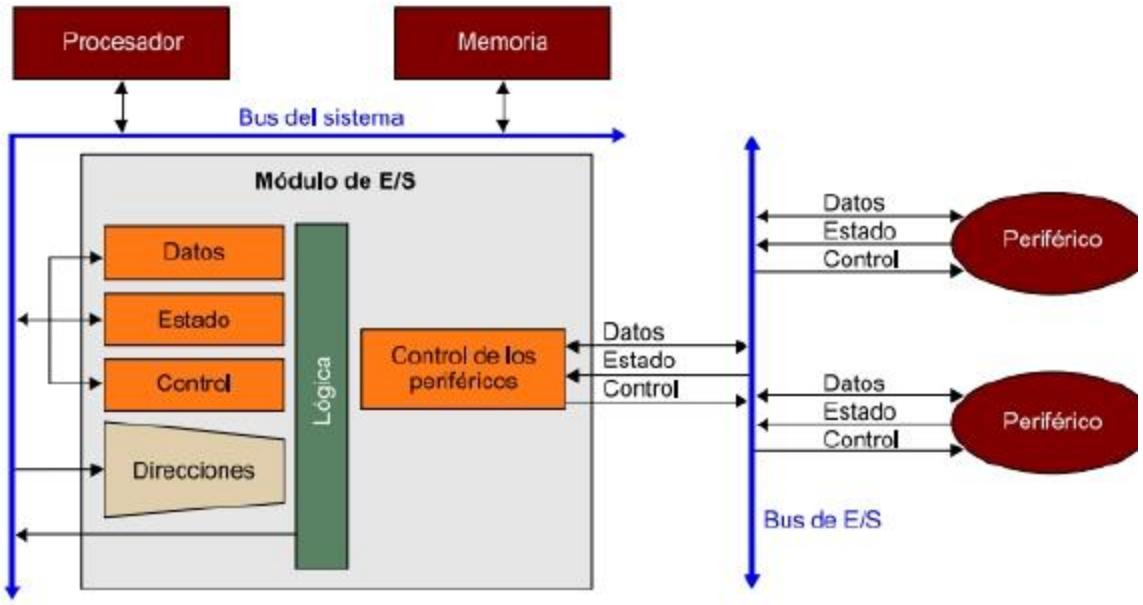
- Del módulo de E/S distinguimos tres partes básicas: 1) Una interfaz interna normalizada con el resto del computador mediante el bus de sistema que nos da acceso al banco de registros del módulo de E/S.
- Una interfaz externa específica para el periférico que controla. Habitualmente la conexión con el periférico se realiza mediante un sistema de interconexión normalizado de E/S.
- La lógica necesaria para gestionar el módulo de E/S. Es responsable del paso de información entre la interfaz interna y externa. En la siguiente figura podéis ver el esquema general de un módulo de E/S.



La forma de comunicación entre el módulo de E/S y el periférico es específica para cada periférico. Lógicamente, depende de las características del periférico que queremos controlar, pero también del sistema de interconexión utilizado para comunicarse. Esta conexión tiene habitualmente unas especificaciones normalizadas y adaptadas al tipo de transferencia que se debe realizar y lo denominamos sistema de interconexión de E/S. Esto hace que la interfaz externa tenga unas características propias que difícilmente se pueden generalizar.

Cuando un módulo de E/S gestiona más de un periférico, hay dos configuraciones básicas, la conexión punto a punto y el multipunto, aunque las configuraciones que encontramos en máquinas reales son muy variadas. En la conexión punto a punto el módulo de E/S gestiona la comunicación con cada periférico individualmente; no es un bus de E/S, pero sí que tiene unas especificaciones normalizadas de la conexión, de manera parecida a las de un bus normalizado de E/S. En la conexión multipunto el módulo de E/S gestiona la comunicación con los periféricos mediante un bus normalizado de E/S y hay que añadir la lógica para acceder al bus.





La comunicación entre los módulos de E/S y el computador es siempre la misma para todos los módulos. Esta comunicación se establece mediante el bus del sistema, de modo que el procesador ve el módulo de E/S como un espacio de memoria, pero estas direcciones, físicamente, corresponden (están mapeadas) a cada uno de los registros que tiene el módulo de E/S del computador y se denominan habitualmente puertos de E/S. De esta manera conseguimos que la comunicación entre el computador y el módulo de E/S se lleve a cabo mediante instrucciones de transferencia para leer y escribir en sus registros, de una manera muy parecida a cómo hacemos para acceder a la memoria.

Estos registros se pueden agrupar según el tipo de señales o el tipo de información que necesitamos para hacer una gestión correcta del periférico:

- Registros de control.
- Registros de estado.
- Registros de datos.

Para gestionar la comunicación entre el procesador y el módulo de E/S son necesarios diferentes tipos de señales.

Las señales de control las utilizamos generalmente para dar órdenes al módulo de E/S, como empezar o parar una transferencia, seleccionar modos de operación del periférico o indicar acciones concretas que debe hacer el periférico, como comprobar si está disponible. Estas señales se pueden recibir directamente de las líneas de control del bus del sistema o de las líneas de datos del bus del sistema y se almacenan en el registro de control.

Las señales de estado nos dan información del estado del módulo de E/S, como saber si el módulo está disponible o está ocupado, si hay un dato preparado, si se ha acabado una operación, si el periférico está puesto en marcha o parado, qué operación está haciendo, o si se ha producido algún error y qué tipo de error. Estas señales se actualizan generalmente mediante la lógica del módulo de E/S y se almacenan en el registro de estado.

Los datos son la información que queremos intercambiar entre el módulo de E/S y el procesador mediante las líneas de datos del bus del sistema y se almacenan en el registro de datos.

Las direcciones las pone el procesador en el bus de direcciones y el módulo de E/S debe ser capaz de reconocer estas direcciones (direcciones de los puertos de E/S) correspondientes a los registros de este módulo. Para saber si la dirección corresponde a uno de los registros del módulo utilizamos un descodificador. Este descodificador puede formar parte del módulo de E/S o de la misma lógica del bus del sistema.

Hay que tener presente que un computador puede tener definidos diferentes tipos de conexiones normalizadas entre el módulo de E/S y el resto del computador. Tanto el módulo de E/S como el computador se deben adaptar a estos tipos de conexión, de modo que tenemos módulos de E/S adaptados a las diferentes normas, y eso tiene implicaciones con respecto al hardware y a la manera de gestionar las operaciones de E/S, como veremos más adelante cuando analicemos las técnicas básicas de E/S.

4.4.1.- Interface programable.

Uno de los integrados universalmente usados en sistemas basados en microprocesadores es sin duda el 8255. Este circuito fue inicialmente diseñado por Intel Corporation como parte del juego de integrados de apoyo a sus primeros sistemas de 16 bits (8086 y 8088). El chipset incluía numerosos dispositivos tales como controladores serie, controlador de CRT, gestores de acceso directo a memoria, controladores de unidades de disco, etc. La fuerte evolución en el diseño de computadoras ha convertido a gran parte del chipset del 8086 en piezas de museo debido a que muchas de las funciones no tienen hoy día utilidad alguna (carece de sentido emplear viejos controladores de CRT o disco).

Sin embargo, existen una serie de componentes que conservan todavía hoy, después de veinte años, toda su utilidad. En concreto nos estamos refiriendo a la UART 8251 y al controlador de interfaz paralelo PPI 8255. En este artículo trataremos el 8255, un versátil y económico integrado de fácil conexión a cualquier sistema basado en microprocesador o microcontrolador, que proporciona de un modo elegante y sencillo puertos E/S disponibles.

La opción más correcta sería emplear estos dispositivos en un sistema basado en un 8086/8088 (actualmente manufacturados en versiones CMOS de bajo consumo y alta velocidad por OKI Semiconductor Corp.), si bien dada su versatilidad pueden ser empleados por cualquier otro sistema.

Entre las aplicaciones actuales podemos comprobar como el 8255 se encuentra con facilidad en tarjetas de expansión de puertos para el bus ISA del PC, ya que simplifica enormemente la elaboración de la placa de circuito impreso aportando suficiente potencia de control. Es también ideal para expansión de puertos E/S en monoplacas o gestión de periféricos como conversores analógicos/digital y otros.

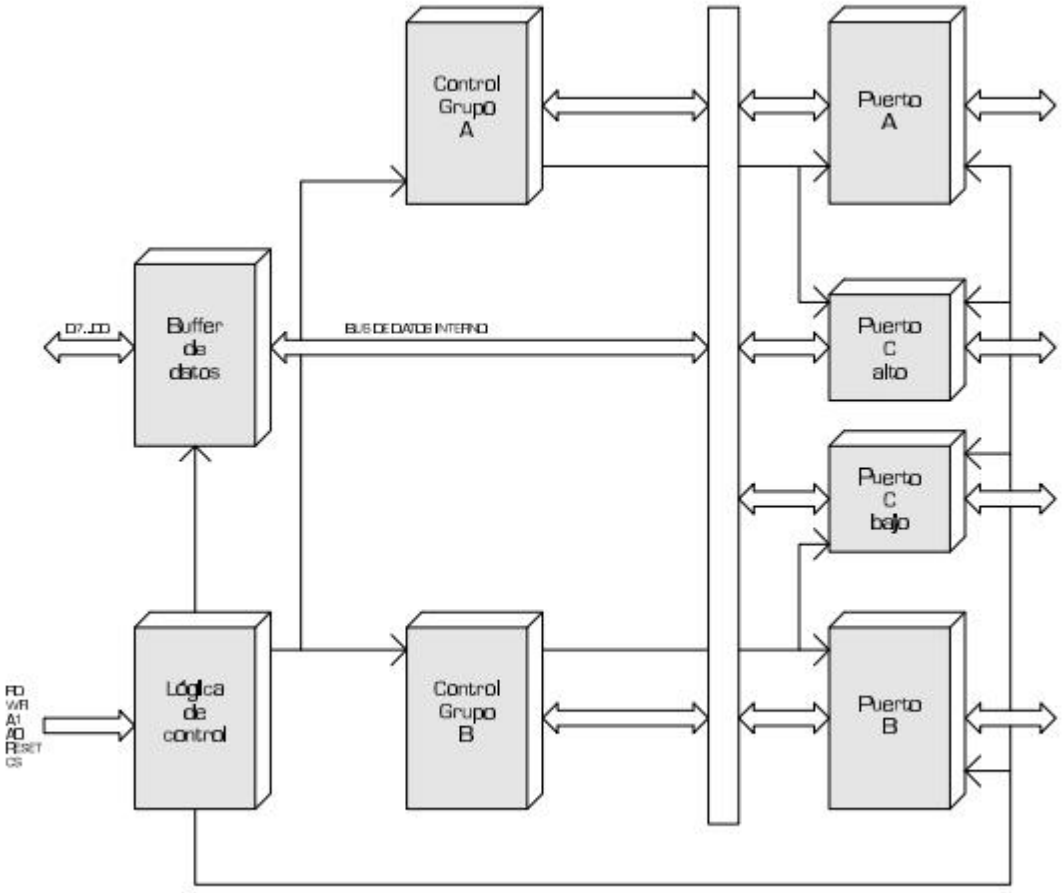


Figura 1

Básicamente se trata de tres puertos (A,B,C) de ocho bits, encontrándose el puerto C dividido en dos puertos de cuatro bits. Estos cuatro puertos formados (dos de ocho bits y dos de cuatro) se organizan en dos grupos de 12 bits. Cada grupo contiene un puerto de ocho bits y otro de cuatro.

El primer grupo comprende el puerto A y la parte alta del puerto C mientras que el segundo comprende el puerto B y la parte baja del puerto C.

- Grupo A: PA0..PA7 + PA4..PA7
- Grupo B: PB0..PB7 + PA0..PA3

Estos grupos pueden operar en tres modos de funcionamiento diferentes:

- Modo 0: Tres puertos de cuatro bits programables como entrada o salida.

- Modo 1: Un puerto de ocho bits programable como entrada o salida y un puerto de cuatro bits de control.
- Modo 2: Este modo configura al puerto A como un puerto bidireccional dejando los cinco bits más significativos del puerto C (PC3...PC7 nótese que emplea por tanto más de un grupo) como líneas de control del puerto.

Estos modos de operación se definen mediante un octeto de estado del que hablaremos a continuación.

Para acceder tanto al octeto de estado como a los puertos en sí debemos hacer uso de las señales de control del 8255. Estas señales son las habituales en el chipset del 8086, y pasamos a continuación a describirlas:

- Dx: Bus de datos bidireccional con estado de alta impedancia. Conecta el dispositivo con el bus de datos del microprocesador.
- CS: Señal chip select. Cualquier operación sobre el chip requiere un nivel bajo en esta señal. Usualmente se compara la dirección emitida por el microprocesador mediante lógica externa y si dicha dirección corresponde con el rango asignado por el diseñador al periférico la lógica comparadora de dirección pondrá a cero esta línea. Mientras esta línea se encuentre alta el bus de datos D7...D0 se encuentra en un estado de alta impedancia para permitir que otros periféricos hagan uso del bus del microprocesador.
- RD: Señal de lectura. Un valor bajo en esta señal dispondrá en el bus de datos bidireccional D7..D0 el valor del puerto indicado por las dos líneas de dirección A1 y A0.
- WR: Señal de escritura. Un valor bajo escribirá el octeto presente en el bus de datos en el registro/puerto indicado por A1 y A0.

A1 y A0: Selección de registro:

00 puerto A

01 puerto B

10 puerto C

11 octeto de control

Acceso al registro de control

Hemos visto hasta ahora los diferentes puertos que existen y los modos de configurarlos. Sabemos también que esto se realiza por medio del octeto de control (para ello realizamos una

operación de escritura sobre él, no siendo posible su lectura). Veremos a continuación la descripción completa de dicho octeto:

- Bit 0: Puerto C bajo (1=Entrada, 0=Salida)
- Bit 1: Puerto B (1=Entrada, 0=Salida)
- Bit 2: Selección modo grupo uno (0=Modo 0, 1=Modo 1)
- Bit 3: Puerto C alto (1=Entrada, 0=Salida)
- Bit 4: Puerto A (1=Entrada, 0=Salida)
- Bit 5,6: Selección modo grupo dos (00=Modo 0, 01=1, 1x= 2)
- Bit 7: Flag de modo a uno (1=Activo)

4.5.- Temporizador programable.

El 8086 y 8088 fue usado como generador de reloj en el IBM PC. Se usaba para generar la frecuencia de 4.77 MHz para el microprocesador del IBM PC, otra de 3.58 MHz para la tarjeta de video y una de 1.19 MHz para los tres temporizadores, de los cuales salían otras frecuencias (ver abajo).

Aunque el microprocesador 8088 del computador soportaba una frecuencia de hasta 5 MHz, necesitando un cristal de 15 MHz para el 8284 (3 veces la frecuencia del procesador), IBM decidió usar una frecuencia de cristal ligeramente menor, a 14.31818 MHz, que al dividirla entre 3 resultaba en el microprocesador del IBM PC corriendo a 4.7727267 MHz. Este pequeño cambio permitía dividir la frecuencia base del cristal entre cuatro y generar una frecuencia de 3.579545 MHz usada por la tarjeta de video Color Graphics Adapter (CGA) para generar el burst de color en el estándar de televisión NTSC, ahorrándose así algunas piezas que hubieran sido necesarias para generar esa frecuencia independientemente.

La frecuencia PCLK de 2.3863633 que generaba el 8284 (la mitad de los 4.77 MHz para el procesador), era dividida entre 2 para generar una señal de reloj de 1.1931817 MHz para los tres contadores del temporizador programable de intervalos Intel 8253.

- El contador 0 del 8253 dividía esa frecuencia en 65536 y generaba la señal de 18,2065 ticks (por segundo). La salida del contador estaba conectada al IRQ 0 del controlador de interrupciones 8259. Al finalizar el intervalo de tiempo (18.2065 veces por segundo) se

disparaba una interrupción que era procesada por el IBM PC ROM BIOS para mantener un contador del tiempo transcurrido desde el encendido del computador, que podía usarse para calcular la hora del día.

- El contador 1 del 8253 dividía la frecuencia en 18 y enviaba una señal de 66287.87 Hz al canal 0 del controlador DMA Intel 8237 para el refrescamiento de la memoria RAM.
- El contador 2 del 8253 podía usarse con cualquier divisor entre 1 y 65536 y generar una frecuencia desde 18,2 Hz hasta 1.1931817 MHz que podía usarse para generar tonos por el altavoz del computador y también para generar los tonos que representaban el 1 y el 0 para almacenar datos y archivos en el grabador de cassettes.

Bibliografía básica y complementaria:

- Departamento de arquitectura y tecnología de computadores. Universidad de Sevilla.
- Erlang debugger Ericsson AB. All Rights Reserved.: Debugger
- Depto. de Automatica Juana María López Area de Arquitectura de Computadores
- Cuaderno de Prácticas Laboratorio de Fundamentos de Computadores.

  Facultad de Informática Universidad Complutense de Madrid.

- Teoría de computadores.
  http://www.computacion.geozona.net/teoria.html
- Dispositivos lógicos microprogramables,
  http://perso.wanadoo.es/pictob/indicemicroprg.htm
- Curso de Microcontroladores Motorola,
  http://www.geocities.com/moto_hc08/index.html
- Abel, P.; Lenguaje Ensamblador para IBM PC y Compatibles; Ed. Prentice Hall; 3ª Edición; 1996.
- Brey, B.; Los microprocesadores de Intel: Arquitectura, Programación e Interfaces; Ed. Prentice Hall; 3ª Edición; 1995.
- Caballar, J.; El libro de las comunicaciones del PC: técnica, programación y aplicaciones; Ed.

Rama-Computec; 1ª Edición; 1997.

- Morgan y Waite; Introducción al microprocesador 8086/8088; Ed. Byte Books/Mc Graw Hill; 1ª Edición; 1992.
- Pawelczak; Pass32 32 bit Assembler V 2.5 Instruction Manual; 1997.
- Rojas, A.; Ensamblador Básico; Ed. Computec; 2ª Edición; 1995.
- Socha y Norton; Assembly Language for the PC; Ed. Brady Publishing; 3ª Edición; 1992.
- Tannenbaum, A.; Organización de Computadoras un enfoque estructurado; Ed. Prentice Hall; 3ª Edición; 1992.
- http://www.cryogen.com/Nasm
- http://www.geocities.com/SiliconValley/Bay/3437/index.html