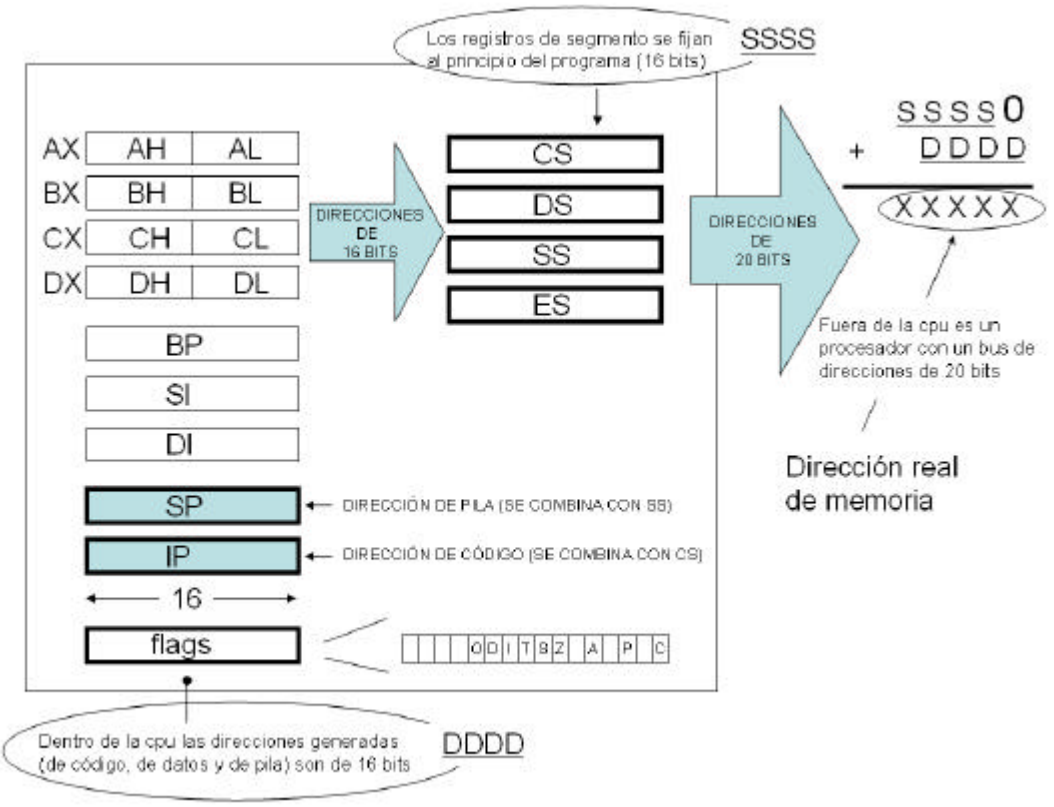
Arquitectura intel 8086

Preámbulo de OSO para alumnos formados en el procesador MIPS

Asignatura Sistemas Operativos
Murcia abril de 2005

1.- Introducción

El procesador intel 8086 es un procesador de 16 bits conectado a un bus de memoria de 2^{20} palabras de 1 byte (en terminología intel se reserva el término "word" para designar a 2 bytes consecutivos):



En la figura se observa que todos los registros son de 16 bits.

Distinguimos → R16

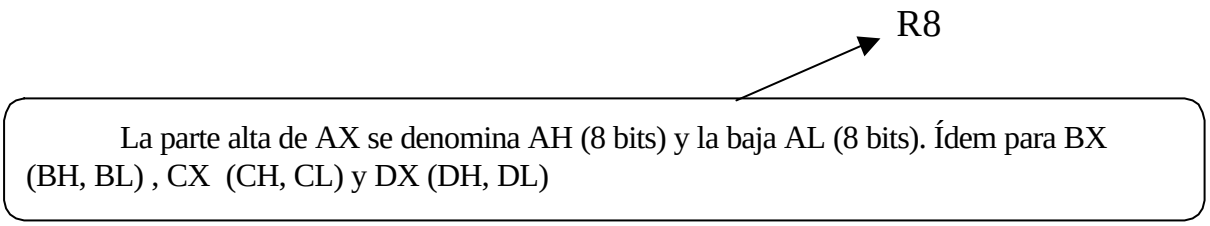
- los registros de datos: AX, BX, CX, DX , BP, SI, DI
- el registro puntero de pila (stack pointer): SP

RegSeg

- el registro contador de programa (instruction pointer):IP
- el registro "de flags" (estado de la cpu) que veremos más adelante
- y los registros "de segmento" que almacenan la dirección de comienzo de la zona de código, zona datos y zona de pila

  CS apunta al "segmento de código"
  DS apunta al "segmento de datos"
  SS apunta al "segmento de pila"

  El registro de segmento ES puede apuntarlo el programador a un posible espacio de memoria extra que necesitase



Todos los registros son de 16 bits y sin embargo durante el ciclo de instrucción cada vez que el procesador accede a memoria (sea para instrucción, pila o dato) lo hace mediante 20 bits. Para conseguirlo fue para lo que añadieron al viejo procesador intel 8085 los registros de segmento.

2.- Direccionamiento y Registros de Segmento

Todas las direcciones se originan en el procesador con un tamaño de 16 bits (*offset*) (dirección de instrucción –IP-, dirección de pila –SP- y dirección de datos –[XXXX] -) y al atravesar la zona de registros de segmento para conseguir llegar al bus externo estos 16 bits se combinan con el contenido de un registro de segmento, también de 16 bits, y consiguen así los 20 bits necesarios para el bus.

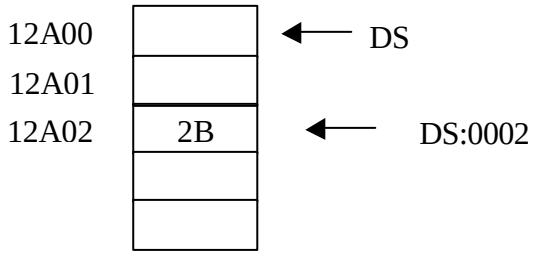
IP se combina con CS *code segment* (apuntador al segmento de código)
SP se combina con SS *stack segment* (apuntador al segmento de pila)

Las direcciones de datos (16 bits) se combinan con DS *data segment* (apuntador al segmento de datos)

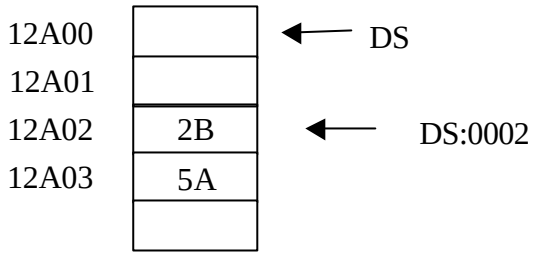
.

Aunque los registros de segmento son de 16 bits realmente hacen referencia a una dirección de 20 bits. Así si DS vale 12A0 (16 bits) la dirección a la que hace realmente referencia es a la 12A0**0** (20 bits). Se añaden 4 bits a 0 por la derecha al valor contenido en el registro de segmento para conseguir los 20 bits necesarios – se multiplica por 16_{10} - . La forma de combinarse el offset con el valor de segmento contenido en el registro de segmento se reduce por tanto a desplazar el contenido del segmento para posteriormente sumarle el offset. Es de destacar que una misma dirección física de memoria puede ser dada por más de una pareja de valores *segmento:offset*

Así (ver figura a continuación) la instrucción MOV AL , DS:[0002] trae al registro AL el contenido de la dirección 12A02 (12A0**0** + 0002) . Introduciría en el registro AL el valor 2B (movería 8 bits)



Con la instrucción MOV AX, DS:[0002] moveríamos (ver figura a continuación) 16 bits de memoria al registro AX. En AH quedaría 5A y en AL 2B



Cambiando de registro de segmento podríamos acceder a las zonas de memoria referenciadas por CS, SS o ES

```
MOV AX, CS:[0002]
MOV AX, SS:[0002]
MOV AX, ES:[0002]
```

La mayoría de las veces las instrucciones nos las encontraremos de la siguiente forma (no aparece de forma explícita el nombre de un registro de segmento):

```
MOV AX, [0002]
```

Este caso es equivalente a MOV AX, DS: [0002] (El registro de segmento que se utiliza por defecto cuando se emplea direccionamiento absoluto es el DS)

Si llamamos *juan* a la dirección DS:0002 nos podemos encontrar con la instrucción:

MOV AX, juan (instrucción *registro, memoria*)
que copia el contenido de la variable *juan* (de 2 bytes) en el registro AX.

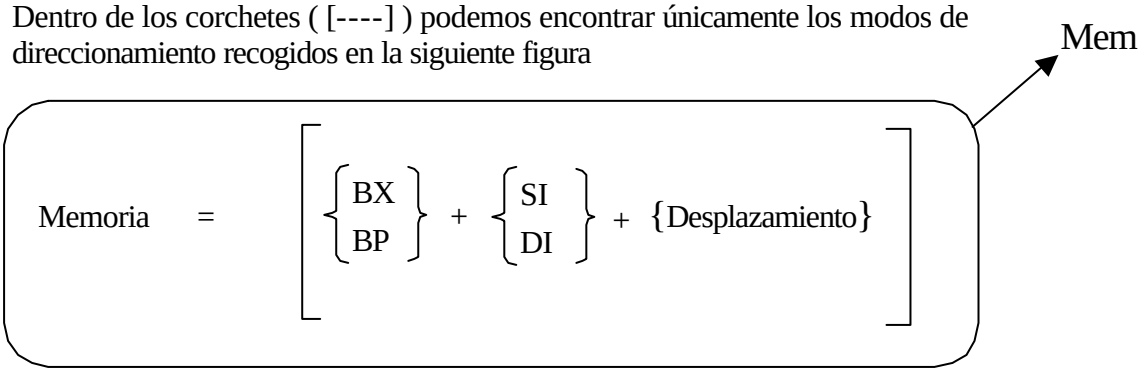
Nos podemos encontrar también con las instrucciones

MOV juan, AX (instrucción *memoria, registro*)
MOV CX, AX (instrucción *registro, registro*)
MOV CX, 012B (instrucción *registro, inmediato*)

Sin embargo en el procesador intel 8086 no existen las instrucciones *memoria, memoria*

MOV ~~juan, pedro~~

Dentro de los corchetes ([----]) podemos encontrar únicamente los modos de direccionamiento recogidos en la siguiente figura



Puede aparecer un primer componente que puede ser bien BX o bien BP, un posible segundo componente que puede ser bien SI o bien DI y un posible tercer componente que siempre es un valor de 16 bits. De los tres componentes pueden aparecer sólo uno, dos o los tres.

Ejemplos

MOV AX, [123A] que corresponde con el modo de direccionamiento absoluto ya visto

MOV AX, [SI+123A] en este caso 123A adoptaría el papel de desplazamiento a sumar al contenido del registro SI para así obtener la dirección final de 16 bits . Esta dirección se combinará con el registro de segmento DS para así conseguir la dirección final de 20 bits.

Los direccionamientos [BX+SI+123A] , [BX+DI], [BP] serían válidos

Direccionamientos no válidos:

[BP+BX] no sería válido puesto que el primer componente bien no está o es BP o es BX. No puede emplearse al mismo tiempo BX y BP

[DX+BX] no sería válido puesto que DX no queda recogido en la fórmula anterior

Nota: En todos los casos el registro de segmento utilizado por defecto es DS salvo que aparezca el registro BP entre los corchetes en cuyo caso el registro por defecto es SS. Si quisiéramos utilizar el DS debería de aparecer DS de forma explícita:

MOV AX, [BP + 0025] sería equivalente a MOV AX, SS:[BP+0025]

si quisiéramos usar el DS deberíamos de poner

MOV AX, DS:[BP + 0025]

En el caso en que se utilice el direccionamiento *Memoria – inmediato* es necesario indicar la longitud del dato

Ejemplo

MOV BYTE PTR [2000],6
modificaría la dirección DS:2000

MOV WORD PTR [2000],6
modificaría las direcciones DS:2000 y DS:2001

En la tabla recopilatoria de instrucciones final (ver tabla final) cuando aparece " " se debe de interpretar como cualquier modo de direccionamiento: registro (R16, R8) , memoria (Mem) , 8 bits inmediato (##) o 16 bits inmediato (####)

Así MOV , significa que tiene 2 operandos que pueden utilizar cualquier tipo de direccionamiento (excepto *Mem* , *Mem* como vimos anteriormente y siempre que ambos operandos tengan el mismo tamaño)

Nota: para introducir un dato inmediato en un registro de segmento previamente hay que introducirlo en un registro R16 (no existe MOV *RegSeg, inmediato*)

Ejemplo

```
MOV AX , 123A
MOV  DS , AX
```

3.- La pila (instrucciones PUSH y POP)

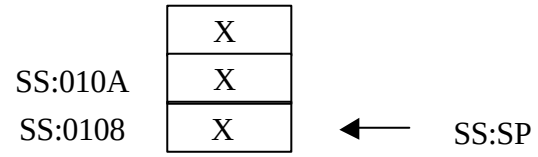
La pila en el procesador intel 8086 se implementa mediante un puntero a memoria (20 bits – SS:SP -) y la zona de memoria referenciada por el mismo. Los elementos que se apilan y desapilan son siempre de 16 bits (word). La pila crece en direcciones decrecientes.

Ejemplo

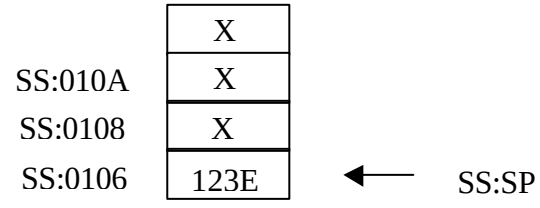
PUSH AX. Almacena en la pila el contenido de AX

Suponemos AX=123E ; SP= 0108

Antes de la ejecución de la instrucción el estado de la pila es



Tras la ejecución de PUSH AX la pila queda como sigue



SP es decrementado en 2 unidades (1 elemento de pila) y en ese lugar de la memoria queda copiado el valor contenido en AX

Con la instrucción POP BX retiraríamos de la cima de la pila el dato 123E que quedaría en el registro BX. SP volvería a su antiguo valor 0108.

4.- El registro de flags (ver instrucciones de salto condicional a continuación)

El registro de flags (PSW en inglés – palabra de estado -) reúne en una entidad de 16 bits (word) el conjunto de flags o indicadores binarios de estado y de control. De esa manera se puede almacenar en la pila el estado del procesador mediante la instrucción PUSHF y recuperarlo mediante POPF

Estos indicadores (o flags) se clasifican en dos grupos:

Flags de estado (C, A, O, Z, P y S):

Muestran el estado del procesador (su contenido es modificado de forma indirecta por las instrucciones aritmético–lógicas aportando información adicional sobre el resultado) a la altura de la última instrucción aritmético lógica ejecutada. El resto de instrucciones no alteran los flags de estado

C: bit de carry, se activa a1 si se produce acarreo en una operación aritmética, se activa a 0 si no.

P: bit de paridad, se activa a 1 si el resultado de una operación tiene paridad par, es decir, si el resultado tiene un número par de unos. Se activa a 0 en caso contrario

A: bit carry auxiliar, se activa a 1 si una operación aritmética produce acarreo de peso 16. Se utiliza en las operaciones aritméticas decimales

Z: bit cero, se activa a 1 si una operación produce 0 como resultado. Se activa a 0 en caso contrario

S: bit de signo, se activa a 1 si el bit más significativo de un resultado es 1. Se activa a 0 en caso contrario

O: bit overflow, indica un desbordamiento en una operación aritmética

Flags de control (D, I, T):

Determinan cómo el procesador responderá a determinadas situaciones. El programador manipula estos bits para controlar el modo de ejecución de las instrucciones que siguen

I: El más importante es el flag I que condiciona a la cpu a atender o no las interrupciones pedidas por los periféricos. El programador pone este bit a 1 mediante la instrucción “sti” y a cero mediante la instrucción “cli”. Cuando este bit está a 0 la cpu hace caso omiso a las interruciones (interrupciones inhabilitadas)

5.- Instrucciones de salto condicional

En el procesador intel 8086 las instrucciones de salto condicional evalúan el contenido de los flags y en función del estado realizan o no el salto (éste es un cambio importante frente a MIPS)

Ejemplo

```
aqui:   MOV AX,0001
        ADD AX,0002     <------  la única de las tres que afecta a los flags
        MOV AX,0000
        JZ  aqui
```

La instrucción JZ realiza el salto si el indicador Z está activado a 1 (si el resultado de la última operación aritmética o lógica ha sido 0). En nuestro ejemplo el indicador Z lo dejó activado a 0 la instrucción ADD AX,0002 y como la instrucción MOV AX,0 no altera los flags (por no ser instrucción aritmético-lógica) cuando la cpu alcanza JZ sigue estando activado a 0 y por tanto no se realizará el salto.

Además de JZ ####, tenemos

JC #### salto si C=1
JO #### salto si O=1
JP #### salto si P=1
JS #### salto si S=1

y las contrarias

JNC #### salto si C=0
JNZ #### salto si Z=0
JNO #### salto si O=0
JNP #### salto si P=0
JNS #### salto si S=0

también tenemos JA, JL y JG junto con sus contrarias JNA, JNL y JNG (ver tabla recopilatoria final)

Es de notar que las anteriores instrucciones sólo modifican IP (CS quedaría con el mismo valor). En el argot intel esto se conoce como saltos cercanos (NEAR)

Otro detalle a resaltar es el hecho de que estas instrucciones utilizan direccionamiento relativo al IP y para ello gastan un solo byte – salto de -128 a +127 – (aunque cuando se utilizan de forma simbólica mediante lenguaje ensamblador parece que utilicen el direccionamiento absoluto – JZ aqui –)

6.- Instrucciones de salto incondicional (JMP)

Saltos lejanos (FAR) – modifican CS e IP –

JMP ####:####
Ejemplo: JMP 123A:569A

JMP FAR Mem

Ejemplo: JMP FAR [2000] Salto indirecto hacia la dirección apuntada por el puntero almacenado en DS:2000 (en esa dirección se encuentra la parte de desplazamiento y 2 bytes más adelante se encuentra la parte de segmento del puntero)

Saltos cercanos (NEAR) – modifican IP –

JMP ####

Ejemplo: JMP aqui Utiliza direccionamiento relativo al IP y para ello gasta dos bytes – salto de -32768 a +32767 –

JMP R16

Ejemplo: JMP AX Copia AX en IP

JMP Mem

Ejemplo JMP [BX+0200] Copia el contenido de DS:[BX+0200] en IP

7.- Instrucciones de llamada y retorno de procedimiento (CALL y RET)

Llamadas NEAR

CALL ####, CALL R16, CALL Mem antes de saltar almacenan en la pila la dirección de retorno (sólo IP)

RET retira la dirección de retorno de la pila y la introduce en IP

RET #### después de hacer ret le suma #### a SP

Ejemplo RET 0002 después de RET le sumaría 2 a SP

Llamadas FAR

CALL ####:####, CALL FAR Mem, antes de saltar almacenan en la pila la dirección de retorno (apilan primero CS y después IP)

RETF retira la dirección de retorno de la pila (2 elementos pila) y la introduce en IP y CS

RETF #### después de hacer retf le suma #### a SP

Ejemplo RETF 0002 después de RETF le sumaría 2 a SP

8.- Instrucciones LDS R16, Mem y **LES R16, Mem**

Sirven para cargar un puntero FAR en un par de registros

Ejemplo: LDS CX, [2000] modifica los registos DS y CX con los bytes de memoria DS:2000 a DS:2003

LES CX, [2000] modificaría ES y CX

9.- Instrucción LEA R16, Mem

A diferencia de la instrucción "*MOV R16, Mem*" que carga en el R16 el contenido de la dirección Mem, "*LEA R16, Mem*" carga la parte de offset (16 bits) de la dirección Mem

Ejemplo

MOV AX, DS:[0200] carga el contenido de la dirección DS:[0200] en AX
LEA AX, DS:[0200] carga 0200 en AX

Esta instrucción tiene sentido cuando utilizamos en lugar de direcciones de memoria etiquetas

Ejemplo

LEA AX, juan Si juan ocupa las direcciones DS:0200 y DS:0201 la instrucción dejaría el valor 0200 en AX

10.- Instrucciones de E/S

El procesador intel 8086 tiene un bus de E/S (8bits de datos, 16 bits dirección de E/S) y por él puede acceder a 2^{16} direcciones de E/S mediante las instrucciones "IN AL, DX" y "OUT DX,AL". Este espacio de direcciones se reparte entre todos los periféricos conectados por este bus al procesador

Ejemplos

```
MOV DX, 0080
IN AL, DX          copiaría en el registro AL el contenido del puerto 0080

MOV   DX, 0100
MOV  AL,25
OUT  DX,AL         copiaría en el puerto 0100 el valor 25
```

11.- Memoria de video

El ibm pc consume 4000 bytes de su memoria para manejar el contenido de la pantalla en modo texto (25 líneas de 80 caracteres). Cada carácter gasta 2 bytes, en uno (el primer byte) el programador almacena el código ascii del carácter que quiera hacer aparecer y en el otro (el segundo) puede introducir un nuevo valor para así cambiar los colores del carácter o el fondo del mismo (byte de atributo). La memoria de video comienza en la dirección B800:0000 , dirección que corresponde con el carácter situado en la esquina superior izquierda de la pantalla. Escribir en pantalla es tan sencillo como modificar el primer byte (código ascii) de cada una de las posiciones de pantalla que queremos modificar.