**ORGANIZACIÓN DE REGISTROS**

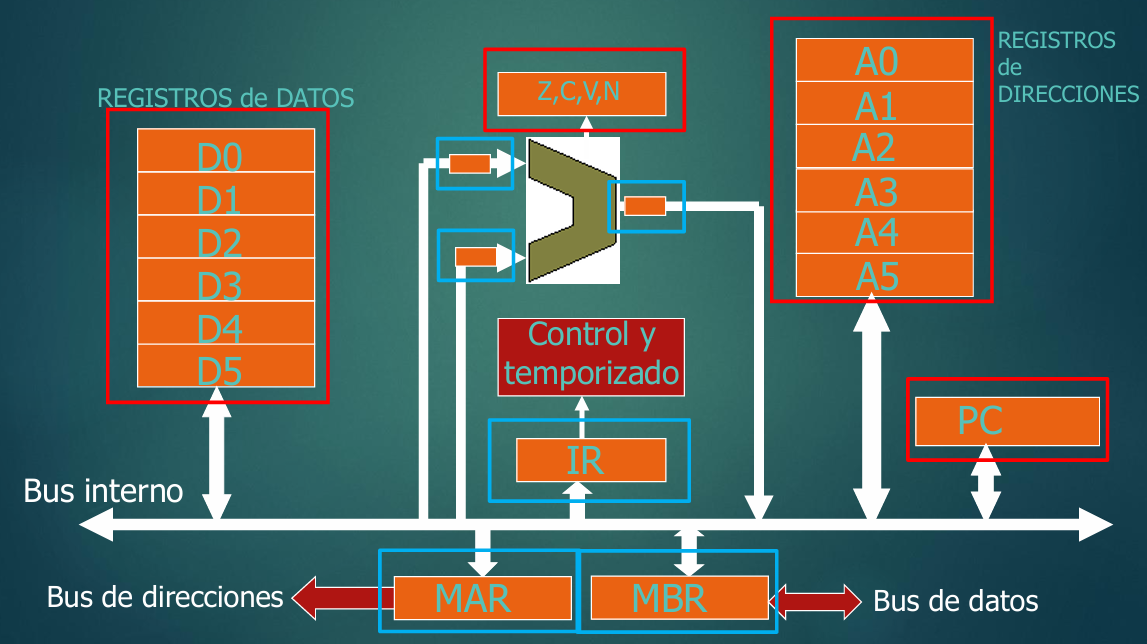
De acuerdo a la posibilidad de acceso, existen 2 tipos de registros:

Registros visibles (al programador):

* Referenciables por medio de instrucciones
* Utilizables explícitamente por el programador

Registros invisibles (al programador):

* No referenciables por instrucciones
* Principalmente utilizados por la UC
* Vinculados a control, estado, o temporales



De acuerdo al uso, los registros pueden ser:

* de Propósito general
* de Datos
* de Direcciones
* de Estado (banderas de estado o códigos de condición)

Registros de propósito general: Son de uso universal. Pueden contener indistintamente el operando en cualquier tipo de instrucción. Generalmente el uso genérico está restringido de alguna manera. Por ejemplo, dedicados a operaciones en PF o enteros, direcciones u operandos, etc.

Registros de datos: Son usados en operaciones que involucran datos (aritméticas, lógicas, etc.). Las operaciones en punto fijo usan registros distintos de las de punto flotante. Normalmente no se usan para manipular direcciones (aunque hay excepciones, como el BX en el 8086). Están relacionados al tipo de dato y tamaño de la ALU.

Registros de direcciones: Se usan para manipular direcciones. Típicamente asociados a modos de direccionamiento (ej. registro índice para el direccionamiento indexado). Incluso pueden tener tamaños distintos a los registros de datos. El tamaño está relacionado con la capacidad de direccionamiento de la CPU, el espacio de direcciones y la forma que administra dicho espacio.

Registros de estado: Almacenan bits referidos al estado operativo de la CPU. El más importante es el que contiene los bits de estado de la ALU (carry, overflow, etc.), comúnmente llamado Status Register (SR), Flags (FR), Program status word (PSW). También pueden almacenar bits referidos a la operación de la CPU (interrupciones, paros de emergencia, errores, etc.). El tamaño es muy variable. Pueden ser utilizados por las instrucciones de bifurcación condicional. Pueden o no ser alterables por el programador.

**CARACTERÍSTICAS DE LOS REGISTROS**

Las características más importantes de los registros son:

* Tamaño
* Cantidad
* Uso

Tamaño: Los registros de direcciones, en principio, deben ser capaces de almacenar la dirección completa, o eventualmente, una fracción de la misma. Los de datos deben tener un tamaño que les permita almacenar la mayoría de los tipos de datos. Algunas máquinas permiten que 2 registros (contiguos) puedan ser utilizados separados, o como un solo registro para almacenar valores de doble longitud.

Cantidad: Es deseable tener muchos registros dentro de la CPU, porque más registros significa más información en la CPU. Pero cuantos más registros, mayor es la cantidad de bits para poder identificarlo, por lo que afecta al tamaño de la instrucción. Menos registros significa más referencias y accesos a memoria (que es mucho más lenta que la CPU). Un número típico es entre 8 y 32 registros. Más registros no resultan en una gran mejora y aumenta el tamaño de la instrucción.

Uso: Un tema muy controvertido es usar todos los registros como de propósito general o especializar su uso. Si son todos de propósito general, se facilita el uso de los registros, pero el tamaño de las instrucciones tiende a ser mayor. Si son especializados, donde está implícito en el código de operación el registro a usar (ej. Acumulador), se ahorran bits en la instrucción, pero limitan la flexibilidad del programador. No hay una receta.

**REGISTROS DE CONTROL**

Un tipo especial de registros son los Registros de Control. Empleados para controlar la operación de la CPU (es decir, forman parte de la Unidad de Control). Pueden ser visibles o invisibles al usuario. Los 4 esenciales para la ejecución de instrucciones:

* Contador de programa (PC)
* Registro de instrucción (IR)
* Registro de dirección de memoria (MAR)
* Registro buffer de memoria (MBR)

Esos 4 registros se emplean para el movimiento de información entre la CPU y Memoria. Dentro de la CPU los datos se deben presentar a la ALU para procesamiento, ésta puede acceder al MBR y a los registros visibles por el usuario. Puede haber también registros temporales adicionales para intercambiar datos con el MBR y demás registros visibles.

**ORGANIZACIÓN DE REGISTROS**

CPU x86 Intel (principales)

Como ejemplo de estudio, vamos a revisar la estructura de registros de la familia de procesadores 80x86 de Intel.

El 80x86 tiene 4 conjuntos de registros:

•De datos

•De direcciones

•De segmentos

•De control

En el caso del Pentium, los registros se extienden en tamaño y se agregan algunos más.

Procesadores 80x86 – Registros de datos (8+8)bits

* AH, AL
* BH, BL
* CH, CL
* DH, DL

De "uso general" (de Datos)

Procesadores 80x86 – Registros de direcciones (16 bits)

* SP
* BP
* SI
* DI

De "direccionamiento"

Procesadores 80x86 – Registros de segmentos (16 bits)

* CS
* SS
* DS
* ES

Segmentos

Procesadores 80x86 – Registros de control (16 bits)

* IP
* FLAGS

PC y banderas

Funciones principales de los registros de datos

• AX: acumulador, es el principal en las operaciones aritméticas

• BX: dato y puntero base (dir. de memoria)

• CX: dato y contador en instrucciones repetitivas

• DX: dato, y 2do operando en instrucciones de multiplicación y división

Funciones principales de los registros de direcciones

• SI y DI: apuntadores (registros índices) que utilizan las instrucciones que recorren arreglos o tablas

• BP: registro base, apuntador a "área de trabajo"

• SP: registro puntero de pila o stack

Funciones principales de los registros de segmentos

• CS: registro de segmento de código

• DS: registro de segmento de datos

• SS: registro de segmento de pila

• ES: registro de segmento "extra"

Los registros de segmentos determinan las direcciones reales para acceso a la memoria. Se "combinan" con las direcciones efectivas ("EA") obtenidas del modo de direccionamiento empleado en la instrucción.

Procesador Pentium - Registros de datos (16 + 8 + 8)bits

* EAX AH, AL
* EBX BH, BL
* ECX CH, CL
* EDX DH, DL

De "uso general" (de Datos)

Procesador Pentium - Registros de direcciones (16 + 16)bits

* ESP SP
* EBP BP
* ESI SI
* EDI DI

De "direccionamiento"

Procesador Pentium - Registros de segmentos

* CS
* SS
* DS
* ES
* (FS)
* (GS)

Segmentos

Procesador Pentium - Registros de control (32 bits)

* (E)IP
* (E)FLAGS

PC y banderas

**REGISTROS CPU 68000 MOTOROLA**

Otra estructura de registros con una filosofía diferente a Intel es la de la familia de procesadores 68000 de Motorola.

El 68000 tiene 3 conjuntos de registros:

• De datos

• De direcciones

• De control

En el 68000, los registros de datos y direcciones no están especializados.

En las referencias de datos, todos los registros de datos se pueden usar idénticamente.

En las referencias de direcciones, todos los registros de direcciones se pueden usar idénticamente.

Procesador 68000 – Registros de datos (32bits)

* D0
* D1
* D2
* D3
* D4
* D5
* D6
* D7

Procesador 68000 – Registros de direcciones (32 bits)

* A0
* A1
* A2
* A3
* A4
* A5
* A6
* A7 (Apuntador del stack usuario)
* A7' (Apuntador del stack supervisor)

Procesador 68000 – Registros de control

* PC (32 bits)
* Estado (16 bits)

PC y banderas

**INSTRUCCIONES EN ASSEMBLY DEL 80X86**

Como se vio anteriormente, las instrucciones de máquina del 80x86 típicamente tienen el siguiente formato:

• dest/fuente son: reg/reg, reg/mem, reg/op.inm, mem/reg, mem/op.inm.

• dest solo puede ser mem o reg.

• mem puede ser una etiqueta (dir.directo) o [BX], siendo (BX) una dirección de memoria (dir.indirecto).

Las instrucciones son de la forma:

* COP
* Destino
* Fuente

Donde:

• COP es un código de operación, por ejemplo ADD

• Fuente es una referencia al primer operando

• Destino es una referencia al resultado, e implícitamente el segundo operando.

Este formato de la instrucción en Assembly se puede vincular al formato en lenguaje de máquina siguiente:

**ADD Ref. resultado: destino Ref. Op1: fuente**

El nombre destino y fuente proviene del hecho de que si hay un movimiento de datos, es desde la derecha (fuente) hacia la izquierda (destino).

En una suma hay 2 operandos y el resultado se almacena en el lugar del operando izquierdo (destino).

Los operandos y el resultado pueden estar en un registro o en la memoria.

Si llamamos:

• mem = especificación de una dirección de memoria

• reg = especificación de un registro de la CPU

• inm = dato inmediato

Entonces las instrucciones son de la forma:

• Instrucción mem, reg

• Instrucción reg, mem

• Instrucción reg, reg

• Instrucción reg, inm

• Instrucción mem, inm

**EJEMPLOS: DIRECCIONAMIENTO A REGISTRO**

• ADD AX, BX

Significa: AX = AX + BX

La instrucción suma el contenido del registro AX con el del BX y el resultado se guarda en AX (operando de 16 bits). También puede sumar en 8 bits si los registros referenciados son de 8 bits.

• ADD AL, AH

Significa: AL = AL + AH

La misma estructura sirve para una instrucción de movimiento de datos.

• MOV AL, CH

Significa: AL = CH

**EJEMPLOS: DIRECCIONAMIENTO INMEDIATO**

• ADD AX, 35AFh

AX = AX + 35AFh

Cuando la referencia es directamente un número, y si la instrucción lo admite, el dato se toma como inmediato (16 bits). También puede tomar el operando inmediato en 8 bits, si se referencia un registro de 8 bits.

• ADD AL, 15

AL = AL + 15

La misma estructura sirve para una instrucción de movimiento de datos.

• MOV AL, 3Eh

AL = 3Eh

**EJEMPLOS: DIRECCIONAMIENTO DIRECTO**

• ADD AX, [35AFh]

AX = AX + contenido direcc. 35AFh y 35B0h

Cuando la referencia es un número entre corchetes (no válido para ASM88), el número es interpretado como directo. El tamaño del operando depende del tamaño del registro destino; si es de 16 bits el operando referenciado indirectamente es de 16 bits. También puede tomar un operando directo en 8 bits, si se referencia un registro destino de 8 bits.

• ADD AL, DATO

AL = AL + variable contenida en DATO (8 bits)

La misma estructura sirve para una instrucción de movimiento de datos:

• MOV CH, NUM1

CH = variable contenida en NUM1 (8 bits)

**EJEMPLOS: DIRECCIONAMIENTO INDIRECTO POR REGISTRO**

• ADD AX, [BX]

AX = AX + dato almacenado en dirección contenida en BX y siguiente.

Cuando la referencia es un registro entre corchetes, el registro es interpretado como registro base y su contenido es la dirección donde reside el operando (indirecto vía registro). El tamaño del operando depende del tamaño del registro destino; si es de 16 bits el operando referenciado indirectamente es de 16 bits. También puede tomar un operando indirecto en 8 bits, si se referencia un registro de 8 bits.

• MOV [BX], AL

Mueve el dato contenido en AL a la dirección contenida en BX.

**EJEMPLOS: DIRECCIONAMIENTO BASE + ÍNDICE**

• MOV CX, [BX + SI]

En CX se carga el dato almacenado en la direcc. BX + SI y la siguiente.

Cuando la referencia es un registro base sumado a un registro índice, entre corchetes, la suma de ambos registros es la dirección donde reside el operando (base indexado). El tamaño del operando depende del tamaño del registro destino; si es de 16 bits el operando referenciado indirectamente es de 16 bits. También puede tomar un operando base indexado en 8 bits, si se referencia un registro de 8 bits.

• MOV [BX + DI], AL

En la direcc. BX + DI se almacena el dato cargado en AL.

**EJEMPLOS: DIRECCIONAMIENTO RELATIVO POR REGISTRO (BASE CON OFFSET)**

• MOV AL, [BX + 2]

AL = dato almacenado en dir BX + 2

Cuando la referencia es un registro base sumado a un número, entre corchetes, la suma de ambos términos es la dirección donde reside el operando (base con offset). El tamaño del operando depende del tamaño del registro destino; si es de 16 bits el operando referenciado indirectamente es de 16 bits, y si es de 8 el operando es de 8 bits.

• MOV [BX + 2Ah], AX

Direc. BX + 2Ah y la que sigue <= dato en AX (16 bits)

**EJEMPLOS: DIRECCIONAMIENTO RELATIVO BASE + ÍNDICE**

• MOV AL, [BX + SI + 2]

AL = dato almacenado en la dir BX + SI + 2

• MOV [BX + DI + 2Ah], AX

Dato almacenado en la dir BX + DI + 2Ah y la que sigue = AX (16 bits)

**FORMATOS DE INSTRUCCIÓN - CRITERIOS DE DISEÑO**

Cuando se diseña el repertorio de instrucciones, es necesario definir si se utilizarán instrucciones cortas, largas o combinadas. En general, se prefiere hacer las instrucciones (por lo menos, las más usadas) cortas porque la velocidad de ejecución de las instrucciones depende de la velocidad de transferencia (“ancho de banda”) de la memoria, es decir, la cantidad de bits que puede transferir por segundo. En general, la velocidad del procesador es varias veces mayor que la de la memoria. Instrucciones más cortas se leen más rápido, y esto hace que el procesador “parezca” más rápido.

De todas maneras, las instrucciones deben tener suficientes bits para expresar todas las operaciones deseadas. La experiencia demuestra que es útil dejar bits libres en los diferentes campos que componen la instrucción para permitir cambios o extensiones en el futuro de la familia de procesadores. Otro aspecto importante es el tamaño del campo de datos, es decir, cuántos bits se asignan a un campo que es una referencia numérica (de dato o dirección).

Ejemplo para MSX88

El proceso de desarrollo de un programa en el MSX88 (y en general en el lenguaje Assembly) consta de tres pasos:

1. Editar el programa: Utilizar un editor de texto para crear el archivo fuente, que típicamente tiene la extensión .asm (por ejemplo, prueba.asm).
2. Ensamblar el programa fuente: Utilizar un programa ensamblador, en el caso del MSX88, el Asm88. La salida del ensamblador son dos archivos: el módulo objeto (prueba.o) y un archivo con formato imprimible (prueba.lst).
3. Enlazar el módulo objeto: Usar un programa enlazador, como el Link88, para crear un módulo ejecutable con la extensión .exe (por ejemplo, prueba.exe).