x86: Prácticas 1

Introducción a la arquitectura x86 y su lenguaje ensamblador

Departamento de Ingeniería en Automática, Electrónica, Arquitectura y Redes de Computadores

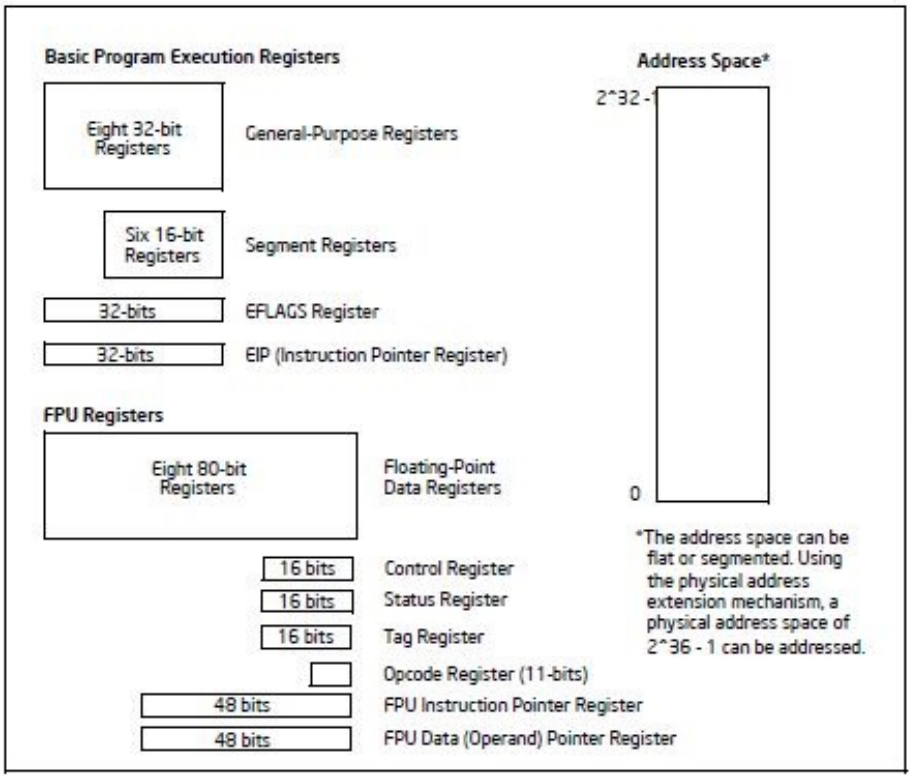
Universidad de Cádiz

Índice

El ambiente básico de ejecución

La arquitectura IA-32 otorga una serie de recursos a los programas y tareas que se ejecutan. Dichos recursos se conocen como **ambiente básico de ejecución.**

El ambiente básico de ejecución



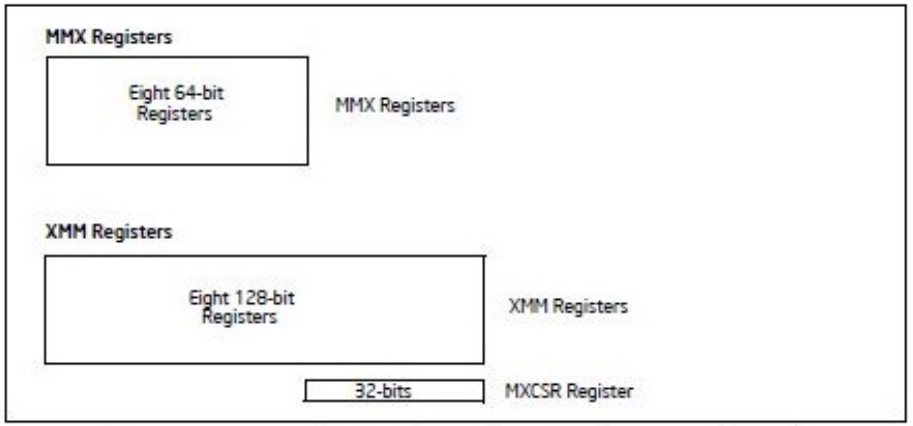


Figura: Ambiente básico de ejcución. Obtenida del manual de Intel.

Índice

Registros de propósito general

- En IA-32 son **8 registros de 32 bits**.
- Pueden utilizarse con cualquier propósito. No obstante, ciertas instrucciones **utilizan de forma predeterminada algunos determinados registros**.
- Los registros **ESP** y **EBP** se usan para controlar la pila. No deberían utilizarse para otro propósito.
- Puede utilizarse solo una parte del registro utilizando **nombres alternativos**. Esto será importante para determinar el formato de la instrucción usado.

Registros de propósito general
Usos comunes

- **Registro EAX:** Acumulador de operandos y resultados.
- **Registro EBX:** Puntero a datos en el registro de segmento DS.
- **Registro ECX:** Contador para operaciones de cadena y bucles.
- **Registro EDX:** Puntero de entrada y salida.
- **Registro ESI:** Puntero a datos del segmento apuntado por el registro de segmento DS y puntero origen para operaciones de cadena.
- **Registro EDI:** Puntero a datos del segmento apuntado por el registro de segmento ES y puntero destino para operaciones de cadena.
- **Registro ESP:** Puntero de pila.
- **Registro EBP:** Puntero a datos de la pila.

Nomenclatura de los registros de propósito general

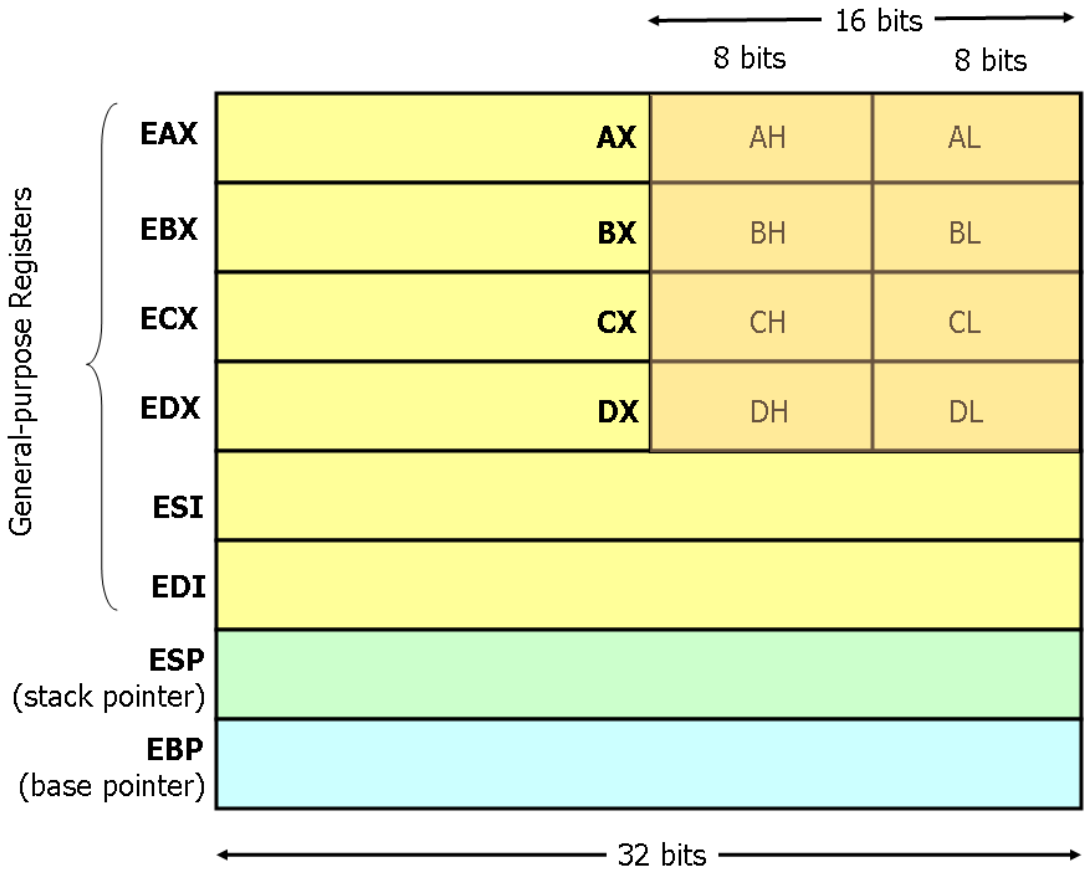


Figura: Nombres alternativos de los registros de propósito general. Imagen obtenida de la Universidad de Virginia.

Registro de banderas EFLAGS

El registro EFLAGS es un registro de 32 bits que contiene banderas, esto es, alguno de los bits del registro son utilizados como banderas por ciertas instrucciones. Algunas de ellas, por ejemplo, son utilizadas por instrucciones aritmeticológicas para **indicar si el resultado es cero, si hay acarreo, etc.**

Registro de banderas EFLAGS

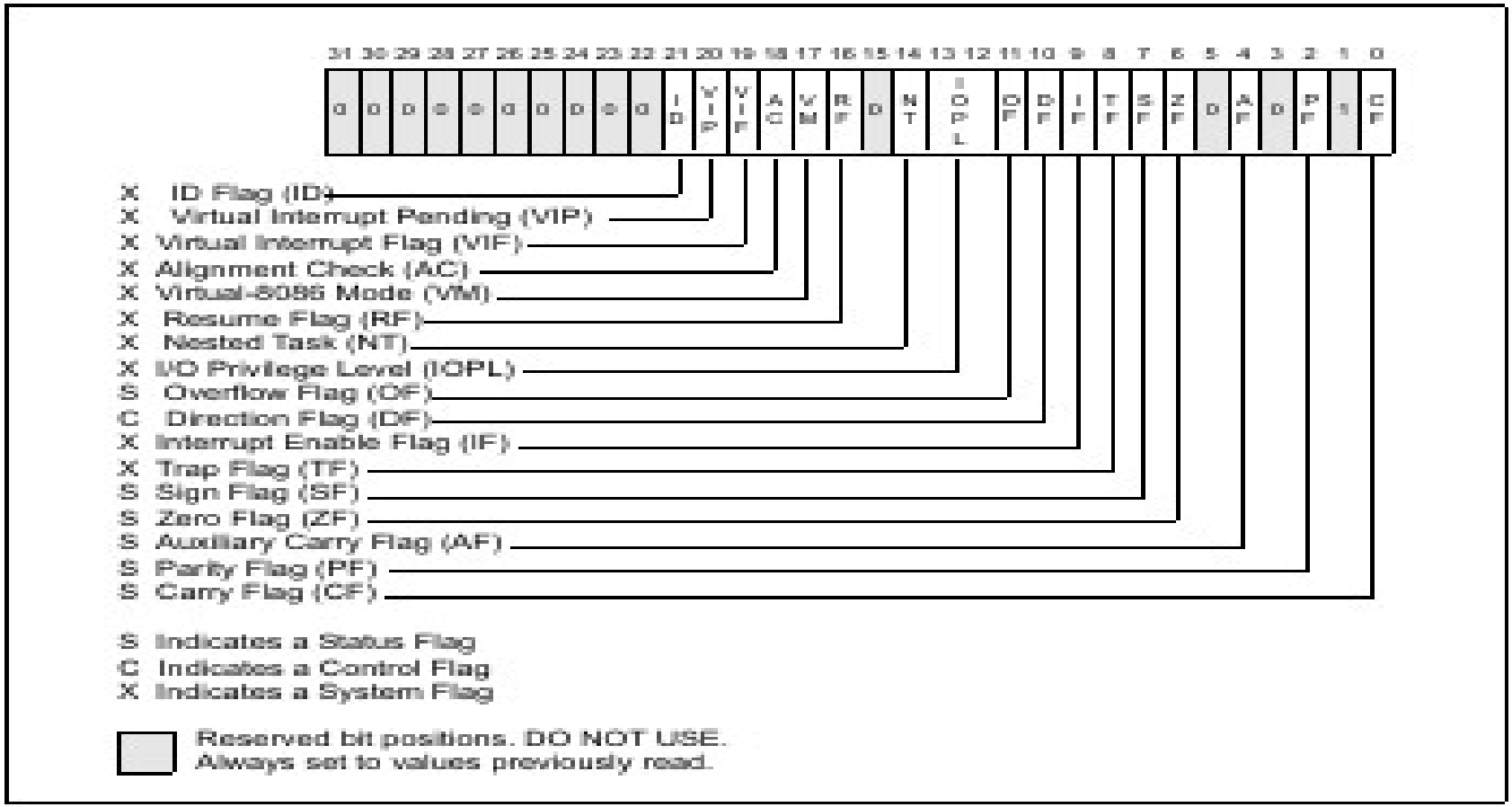


Figura: Registro de banderas. Imagen obtenida del manual de Intel.

Registro puntero de instrucción

El puntero de instrucción (EIP) contiene **el desplazamiento, con respecto al segmento de código, para obtener la siguiente instrucción**. El valor de este registro puede ser modificado (tanto avanzando como retrocediendo) por ciertas instrucciones como **JMP**, **CALL** y **RET**.

Otros registros del ambiente básico de ejecución

- **Registros de la unidad de coma flotante x87 (FPU):** Son ocho registros de 80 bits cada uno, que funcionan como una pila más otros registros de control. La **FPU** los utiliza para realizar operaciones en **coma flotante**.
- **Registros MMX:** Son **ocho registros de 64 bits** que permiten realizar instrucciones **SIMD** de la extensión **MMX**.
- **Registros XMM:** Son **ocho registros de 128 bits** que permiten realizar instrucciones **SIMD** de la extensión **SSE**. Además se incluye un registro llamado **MXCSR** de 32 bits.
- **Pila (Stack):** Es una región de memoria que **funciona como una pila**. Se utiliza en las llamadas a subprogramas, para guardar copias de ciertos valores o para almacenar variables locales.

Índice

IDE y ensamblador

Tras sopesar las ventajas y desventajas de los ensambladores existentes, se decidió utilizar para estas prácticas el **ensamblador GoAsm**, que nos permite realizar programas de 32 bits en **modo protegido** y que utilizan el **modelo de memoria plano** (el programa está contenido en un solo segmento), además dispone de una documentación amplia en su web. Para facilitar además la labor de programación se decidió **integrar dichas herramientas con el IDE EasyCode.**

Índice

Estructura del programa

El ensamblador nos permite declarar en nuestro programa tres secciones:

- **Sección de datos (.DATA):** En esta sección declararemos los datos ubicados en memoria. Para cada dato ubicado en memoria podemos asignar una etiqueta, un tamaño y un valor inicial, o dejarlo sin inicializar.
- **Sección de código (.CODE):** En esta sección escribiremos las instrucciones de nuestro programa.
- **Sección de constantes (.CONST):** En esta sección, al igual que en data, podemos declarar datos ubicados en memoria, pero estos no podrán cambiar a lo largo del programa.

¡Hola Mundo!

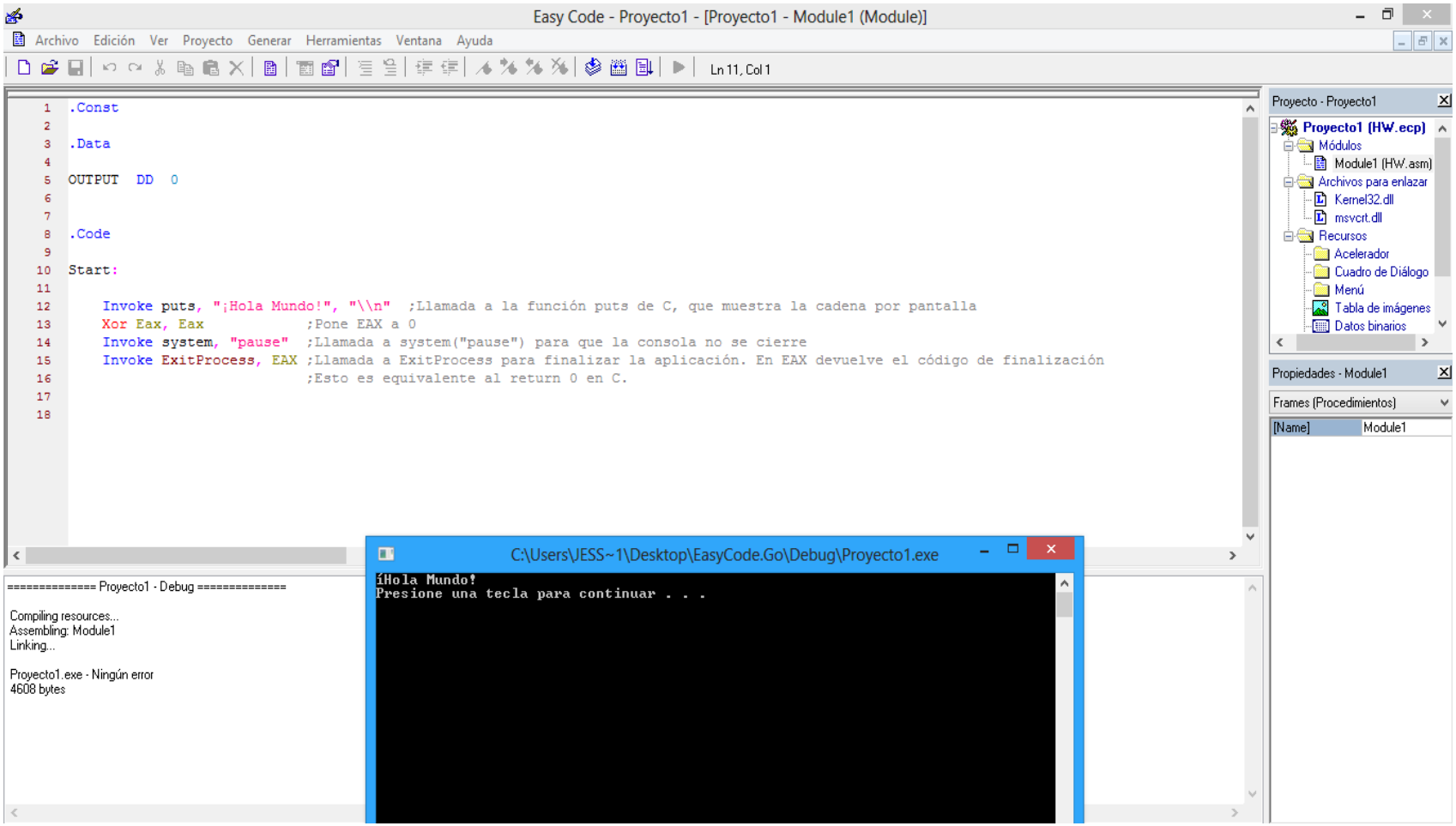


Figura: ¡Hola Mundo! en ensamblador

Índice

La sintaxis del ensamblador

- El carácter ”;” indica que el resto de la línea es **un comentario.**
- Por defecto un número se considera en **decimal**. Para que se considere hexadecimal debe usarse ”0x” al principio o ”h” al final del número.
- No se distingue entre mayúscula y minúscula, excepto en el caso de las etiquetas.
- Las etiquetas de la región de código deben terminar en ":", mientras que las de la región de datos se aconseja que no.
- Al principio de la sección de código debe haber una etiqueta **"Start:", que marca el inicio del programa.**

Formato de las instrucciones x86

- Todas las instrucciones admiten **un solo operando de memoria** a lo sumo.
- Todos los operandos de una misma instrucción deben ser del mismo tamaño. El tamaño de dichos operandos determinará el formato de la instrucción que se utilizará.
- Podemos determinar qué formato de una instrucción se utilizará mediante los nombres alternativos de los registros (por ejemplo, si utilizamos el registro **EAX** se ejecutará un formato de instrucción de 32 bits, si utilizamos **AX**, se usará un formato de 16 bits, etc.)
- Si se utiliza un operando en memoria, habrá que indicar el tamaño del mismo si existe ambigüedad en el formato de instrucción a aplicar.

Formato de las instrucciones x86

MUL—Unsigned Multiply

Opcode	Instruction	Op/ En	64-Bit Mode	Compat/ Leg Mode	Description
F6 /4	MUL *r/m8*	M	Valid	Valid	Unsigned multiply (AX ← AL ∗ *r/m8*).
REX + F6 /4	MUL *r/m8**	M	Valid	N.E.	Unsigned multiply (AX ← AL ∗ *r/m8*).
F7 /4	MUL *r/m16*	M	Valid	Valid	Unsigned multiply (DX:AX ← AX ∗ *r/m16*).
F7 /4	MUL *r/m32*	M	Valid	Valid	Unsigned multiply (EDX:EAX ← EAX ∗ *r/m32*).
REX.W + F7 /4	MUL *r/m64*	M	Valid	N.E.	Unsigned multiply (RDX:RAX ← RAX ∗ *r/m64*.

Figura: Formatos de la instrucción **MUL**. Imagen obtenida del manual de Intel.

Las directivas de memoria

- **NOM_ETIQUETA DX[1] VALOR_INICIAL**
- **NOM_ETIQUETA DX VALOR1, VALOR2, ...**
- **NOM_ETIQUETA DX NÚMERO DUP VALOR**
- **NOM_CADENA DB “Contenido de la cadena”,0**

[1] X es el tamaño. Puede ser: B: byte; W: word, 2 bytes; D: double word, 4 bytes; Q: quad word, 8 bytes T: 10 bytes

El direccionamiento de operandos

- **Direccionamiento de registro:** Nos permite utilizar un dato almacenado en un registro, como en **MOV *EAX*, *ECX***.
- **Direccionamiento inmediato:** Podemos especificar un operando en la misma instrucción, como en **MOV *EAX*, *12h***.
- **Direccionamiento implícito:** Ciertas instrucciones utilizan **de forma predeterminada ciertos registros**, sin dar la posibilidad de utilizar otros. Para más información debe consultar en la documentación de Intel los formatos de cada instruccion.

El direccionamiento de operandos

Direccionamiento a memoria: Utiliza como operando un lugar de memoria.

- Para acceder al contenido de una posición de memoria (tanto para leer como para escribir), utilizaremos []. P.E:

```
MOV [POS], EAX ;x86 → POS = EAX; //En C
```

- Si queremos obtener la dirección apuntada a la que hace referencia una etiqueta, utilizaremos ADDR. P.E:

```
MOV EAX, ADDR POS ;x86 → EAX = &POS; //En C
```

- Si el tamaño de los operandos no queda determinado, podemos decir al ensamblador de qué tamaño es la dirección. P.E:

```
 MOV [POS], 12 ;¿Con qué tamaño almacena 12?
MOV B[POS], 12 ;12 se almacena como Byte.
```

El direccionamiento de operandos

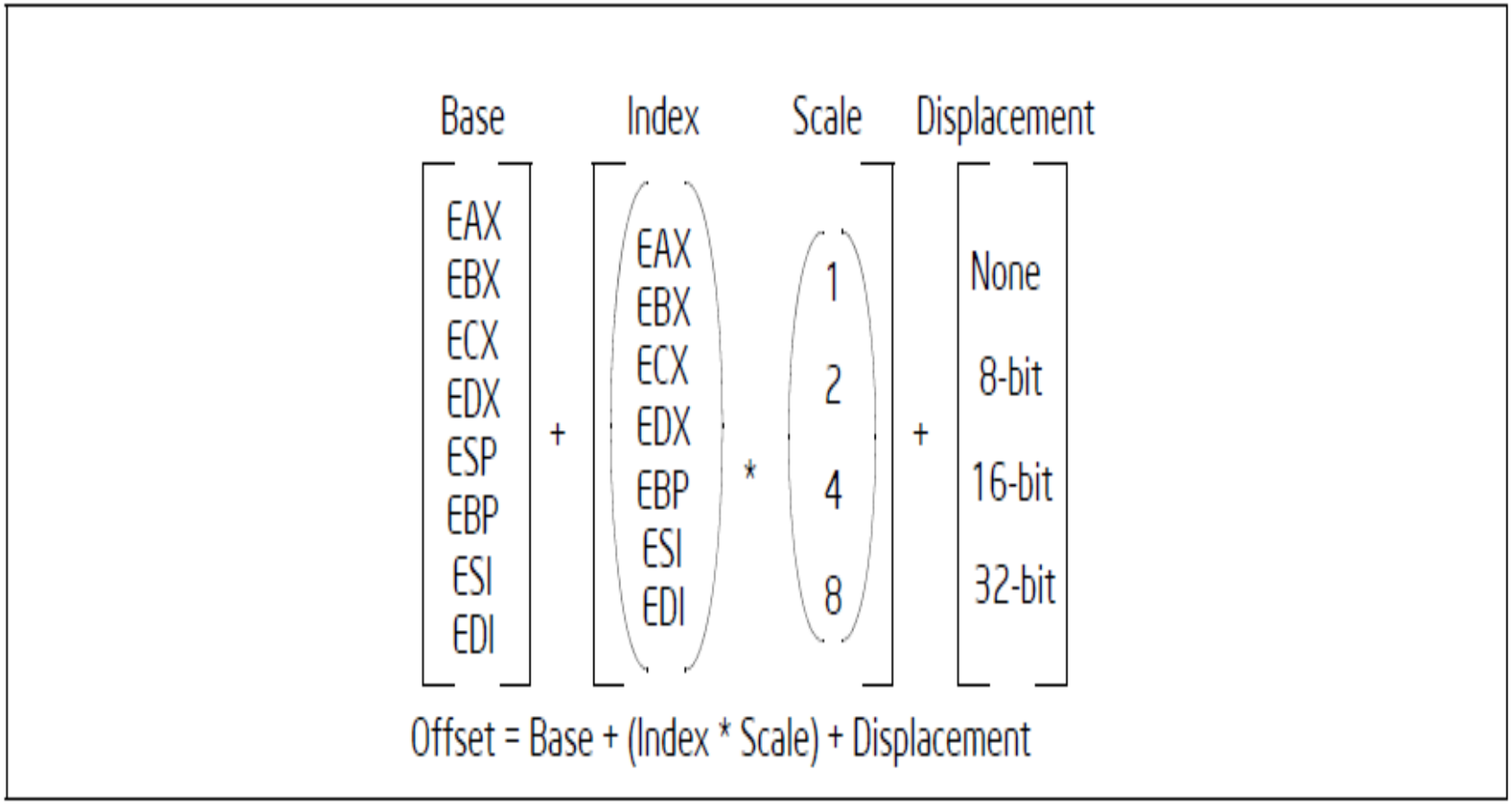


Figura: Formato genérico de direccionamiento directo. Imagen obtenida del manual de Intel.

Uso del depurador

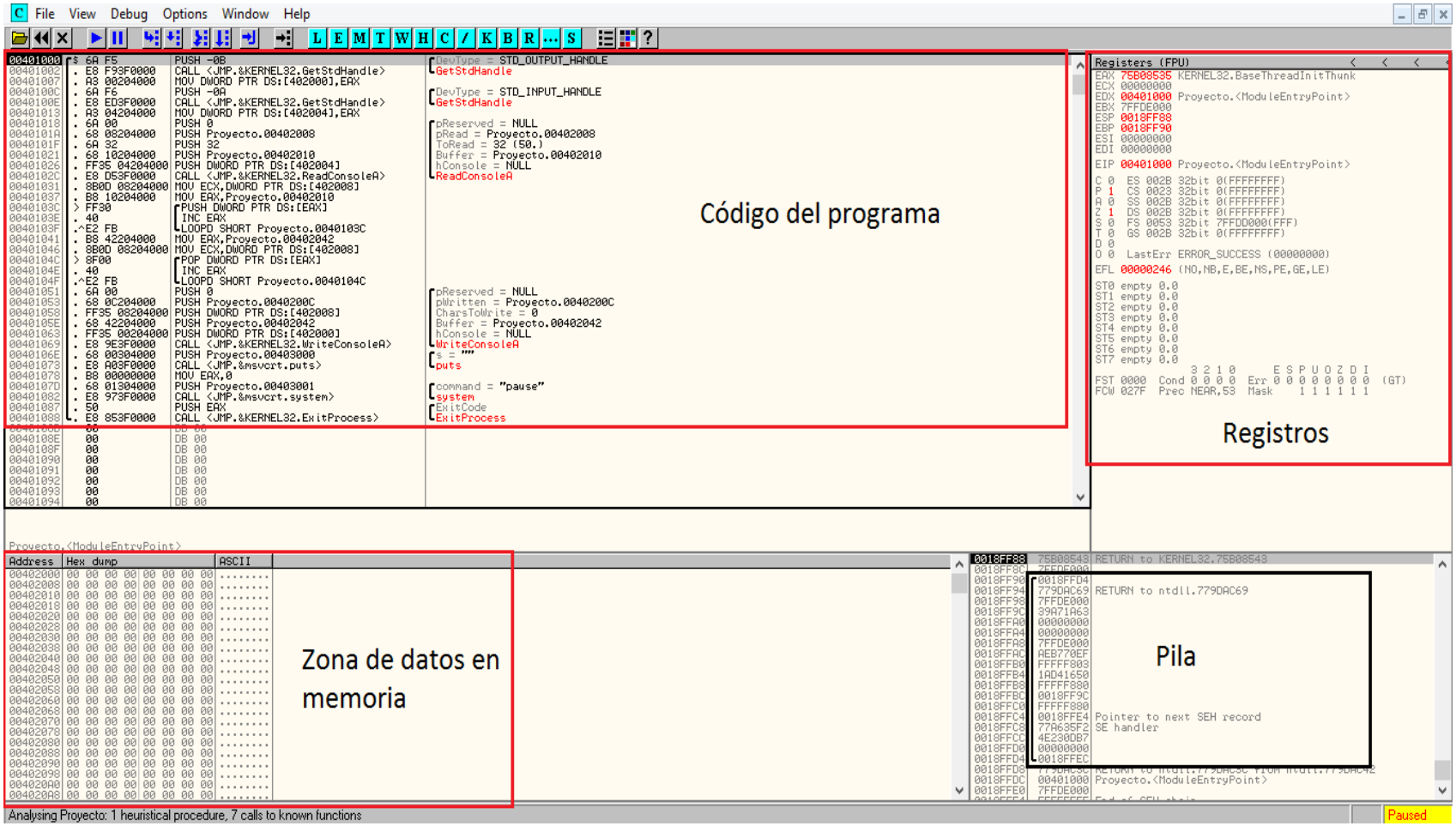


Figura: Ejemplo del depurador en funcionamiento.