Introducción a x86

MSc. Stefano Romero

Arquitectura interna de la familia de procesadores Intel x86 (16 bits)

- Primer chip: 8086 fue usado en una PC IBM.
- Maneja datos de 16 bits y 20 bits de direcciones.
- Tuvo 4 registros de propósito general de 16 bits cada uno:
  - AX (acumulador primario).
  - BX (Registro base para extender addressing).
  - CX (Registro contador).
  - DX (Registro de datos).
- Cada uno de estos dividido en 2 partes: **High** y **Low**.

Arquitectura interna de la familia de procesadores Intel x86 (16 bits)

- Tuvo 3 registros de puntero:
  - SP (Offset en la pila).
  - BP (Referencia de parámetros en la pila).
  - IP (contenía la dirección de la siguiente instrucción).
- 2 registros de índice:
- SI (Puntero fuente para operaciones string).
- DI (Puntero de destino para operaciones string).
- Registro de estado de bandera: Indica condiciones como overflow, paridad, acarreo interrumpido, entre otros.

Arquitectura interna de la familia de procesadores Intel x86 (16 bits)

- En 1980, Intel introdujo a 8087, el cual añade instrucciones de punto flotante a la 8086 así como una pila de 80 bits de ancho.
- A mediados de los 80, Intel introdujo el 80386 el cual fue la primera familia de 32 bits (IA-32). Los diseñadores querían que sea backward compatible para que los programas de la 80286 corran en 80386.
- Las versiones 80386 y 80486 eran de 32 bits con buses de 32 bits. Adicionalmente, la versión 80486 tuvo una memoria caché de mayor velocidad.

Arquitectura interna de la familia de procesadores Intel x86 (32 bits)

- La serie Pentium tuvo un procesador de registros de 32 bits y un bus de data de 64 bits empleando el diseño superescalar.

- La Pentium IV introdujo el concepto de hyperthreading (multithreading). Esto mejoró la eficiencia del procesador lo cual evita que esté menos tiempo sin realizar acciones.

- En el 2001, se introdujo el procesador Itanium el cual trajo el primer chip de 64 bits de Intel (IA-64). Este procesador emplea un emulador de hardware para mantener la compatibilidad hacia atrás con los conjuntos de instrucciones IA-32/ x86.

Arquitectura interna de la familia de procesadores Intel x86 (32 bits)

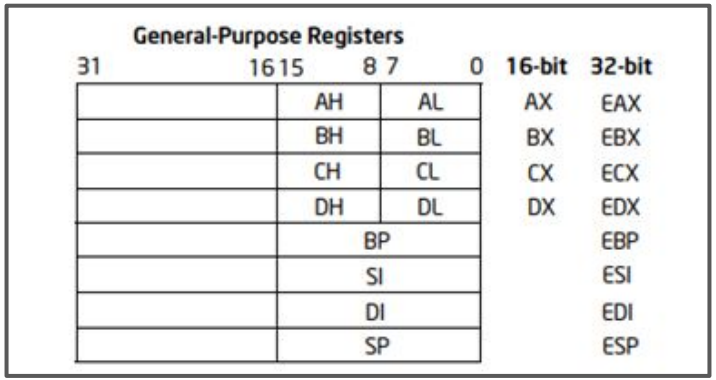

Figura 1. Registros de propósito general [3].

- Los registros de 32 bits fueron extendidos agregando el prefijo “E” (extended).
- Los registros de propósito general de 32 bits EAX, EBX, ECX, EDX, ESI, EDI, EBP y ESP se proporcionan para contener los siguientes elementos:
  - Operandos para operaciones lógicas y aritméticas.
  - Operandos para cálculos de direcciones.
  - Punteros de memoria.
- Se debe tener especial cuidado con el registro **ESP** ya que contiene el puntero de pila y como regla general, no debe utilizarse para otro propósito.

Arquitectura interna de la familia de procesadores Intel x86 (32 bits)

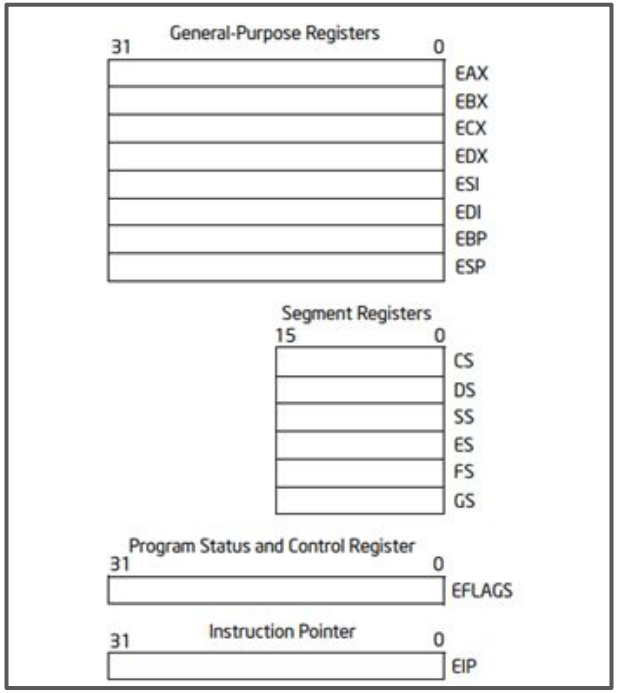

Figura 5. Registros de propósito general [3].

Arquitectura interna de la familia de procesadores Intel x86 (32 bits)

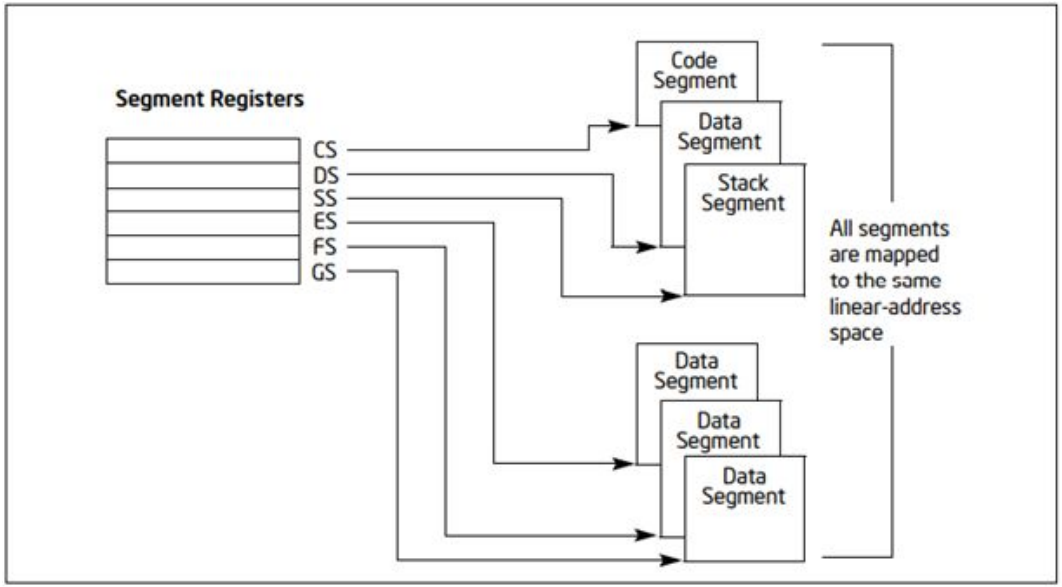

Figura 6. Registros de segmentos [3].

Cada uno de los registros de segmento está asociado con uno de los tres tipos de almacenamiento: código, datos o pila.

¿Por qué aprender lenguaje en ensamblador?

Ventajas	Desventajas
Propósitos educacionales	Tiempo de desarrollo
Depuración y verificación	Depuración y verificación
Realizar compiladores	Fiabilidad y seguridad
Instrucciones no accesibles por alto nivel	Compiladores han mejorado en los últimos años
Optimización por tamaño y velocidad	Códigos de sistemas usan funciones intrínsecas (C++)

Programación en lenguaje ensamblador

- Generalmente, cuando se escribe alguna aplicación esta se subdivide en 3 secciones:
  - **Sección data:** Declaración y definición de data inicial. Se declaran db, dw, dd y dq para 8, 16, 32 y 64 bits.
    seccion .data
  - **Sección bss:** Declaración y definición de data sin valor inicial. Se declaran espacios como resb,resw,resd,resq.
    seccion .bss
  - **Sección text:** Código de operación
    seccion .text
- Este código se complementa con las interrupciones propias del sistema.

Programación en lenguaje ensamblador

- Dependiendo de la arquitectura que se utilice, es posible fijar la sintaxis que se utilizará. Históricamente, para la programación de ensamblador en sistemas de cómputo se han utilizado las sintaxis de Intel y de AT&T.

- Ejemplo Intel: MOV EAX,4

- Ejemplo AT&T: MOVL $4, %EAX

Intel		AT&T	
1	section .data	1	.section .data
2	var1 dd 40	2	var1: .int 40
3	var2 dd 20	3	var2: .int 20
4	var3 dd 30	4	var3: .int 30
5	section .text	5	.section .text
6	global _start	6	.globl _start
7	_start:	7	_start:
8	mov ecx, [var1]	8	movl (var1), %ecx
9	cmp ecx, [var2]	9	cmpl (var2), %ecx
10	jg check_third_var	10	jg check_third_var
11	mov ecx, [var2]	11	movl (var2), %ecx
12	check_third_var:	12	check_third_var:
13	cmp ecx, [var3]	13	cmpl (var3), %ecx
14	jg _exit	14	jg _exit
15	mov ecx, [var3]	15	movl (var3), %ecx
16	_exit:	16	_exit:
17	mov eax, 1	17	movl $1, %eax
18	mov ebx, ecx	18	movl %ecx, %ebx
19	int 80h	19	int $0x80

Toolchain

- Es la cadena completa de programas que se usan para convertir el código fuente, en código de máquina, vincular entre sí los módulos de código ensamblados/compilados, desensamblar los binarios y convertir sus formatos.
- Inicializa su composición de la siguiente forma:

  a. El **ensamblador** convierte los programas en lenguaje ensamblador en código binario. Esto lo hace generando los archivos de objeto (extensión .o).

  b. El **enlazador (linker)** combina varios archivos objeto resolviendo sus referencias de símbolos externos y reubicando sus secciones de datos, generando un único archivo ejecutable.

Toolchain

- Para escoger el ensamblador adecuado, se debe observar qué tipo de sintaxis se está utilizando:
  - Si se utiliza sintaxis Intel, se debe utilizar el ensamblador **NASM**.
  - Si se utiliza sintaxis AT&T, se debe utilizar el ensamblador **GAS.**
  - Ambos ensambladores están en la capacidad de generar un archivo objeto.
- Independientemente al ensamblador, el enlazador que se utilizará es **ld.**
- Si los objetos se deben enlazar con otros lenguajes, dependerá del tipo de lenguaje que se use para compilar. Por ejemplo, gcc también funciona como linkeador para objetos de ASM y códigos en C.

Arquitectura interna de la familia de procesadores Intel x86 (64 bits)

	64 Bits	32 Bits	16 Bits	8 Bits
Acumulador	RAX	EAX	AX	AH - AL
Base	RBX	EBX	BX	BH - BL
Contador	RCX	ECX	CX	CH - CL
Datos	RDX	EDX	DX	DH - DL
Source Index	RSI	ESI	SI	SIH - SIL
Destination Index	RDI	EDI	DI	DIH - DIL
Base Pointer	RBP	EBP	BP	BPH - BPL
Stack Pointer	RSP	ESP	SP	BPH - SPL
Propósito general	R8-R15	R8D-R15D	D8W	D8B

Arquitectura interna de la familia de procesadores Intel x86 (64 bits)

Bandera	Descripción
CF	Acarreo
PF	Paridad
ZF	Zero
SF	Signo
OF	Desborde
AF	Ajuste
IF	Interrupciones

- RFLAGS es el registro que almacena las banderas usadas para resultados de operaciones y como controlador del procesador. Estas banderas están formadas de las EFLAGS de los registros del x86 de 32-bit. Adicionalmente, se añaden los 32 bits superiores que normalmente se encontraban reservadas y en aquellos tiempos inutilizadas.

Arquitectura interna de la familia de procesadores Intel x86 (64 bits)

- Existen una serie de registros específicos que se usar para operaciones de coma flotante de 32 y 64 bits e instrucciones de Datos Múltiples de Instrucción Única (SIMD).
- Las instrucciones SIMD permiten que una sola instrucción se aplique simultáneamente a múltiples elementos de datos (mayor rendimiento).
- Las aplicaciones típicas incluyen procesamiento de gráficos y procesamiento de señales digitales.
- Algunos procesadores X86-64 más recientes admiten registros XMM de 256 bits.
- Los registros XMM se utilizan para admitir las Extensiones SIMD de transmisión por secuencias (SSE).

Registros de 128 bits	XMM0	XMM1	XMM2	XMM3	**...**	XMM12	XMM13	XMM14	XMM15

Referencias

[1] Guide, Part. "Intel® 64 and ia-32 architectures software developer’s manual." Volume 3B: System programming Guide, Part 2 (2011).

Q&A