El lenguaje ensamblador, o assembler (assembly language en inglés), es un lenguaje de programación de bajo nivel para los computadores, microprocesadores, microcontroladores y otros circuitos integrados programables. Implementa una representación simbólica de los códigos de máquina binarios y otras constantes necesarias para programar una arquitectura dada de CPU y constituye la representación más directa del código máquina específico para cada arquitectura legible por un programador. Esta representación es usualmente definida por el fabricante de hardware, y está basada en los mnemónicos que simbolizan los pasos de procesamiento (las instrucciones), los registros del procesador, las posiciones de memoria y otras características del lenguaje. Un lenguaje ensamblador es por lo tanto específico de cierta arquitectura de computador física (o virtual). Esto está en contraste con la mayoría de los lenguajes de programación de alto nivel, que idealmente son portables.

Un programa utilitario llamado ensamblador es usado para traducir sentencias del lenguaje ensamblador al código de máquina del computador objetivo. El ensamblador realiza una traducción más o menos isomorfa (un mapeo de uno a uno) desde las sentencias mnemónicas a las instrucciones y datos de máquina. Esto está en contraste con los lenguajes de alto nivel, en los cuales una sola declaración generalmente da lugar a muchas instrucciones de máquina.

Muchos sofisticados ensambladores ofrecen mecanismos adicionales para facilitar el desarrollo del programa, controlar el proceso de ensamblaje, y la ayuda de depuración. Particularmente, la mayoría de los ensambladores modernos incluyen una facilidad de macro (descrita más abajo), y son llamados macro ensambladores.

Fue usado principalmente en los inicios del desarrollo de software, cuando aún no se contaba con potentes lenguajes de alto nivel y los recursos eran limitados. Actualmente se utiliza con frecuencia en ambientes académicos y de investigación, especialmente cuando se requiere la manipulación directa de hardware, altos rendimientos, o un uso de recursos controlado y reducido.

Muchos dispositivos programables (como los microcontroladores) aún cuentan con el

ensamblador como la única manera de ser manipulados.

Características

- El código escrito en lenguaje ensamblador posee una cierta dificultad de ser entendido ya que su estructura se acerca al lenguaje máquina, es decir, es un lenguaje de bajo nivel.
- El lenguaje ensamblador es difícilmente portable, es decir, un código escrito para un microprocesador, puede necesitar ser modificado, para poder ser usado en otra máquina distinta. Al cambiar a una máquina con arquitectura diferente, generalmente es necesario reescribirlo completamente.
- Los programas hechos por un programador experto en lenguaje ensamblador son generalmente mucho más rápidos y consumen menos recursos del sistema (memoria RAM y ROM) que el programa equivalente compilado desde un lenguaje de alto nivel. Al programar cuidadosamente en lenguaje ensamblador se pueden crear programas que se ejecutan más rápidamente y ocupan menos espacio que con lenguajes de alto nivel.
- Con el lenguaje ensamblador se tiene un control muy preciso de las tareas realizadas por un microprocesador por lo que se pueden crear segmentos de código difíciles y/o muy ineficientes de programar en un lenguaje de alto nivel, ya que, entre otras cosas, en el lenguaje ensamblador se dispone de instrucciones del CPU que generalmente no están disponibles en los lenguajes de alto nivel.
- También se puede controlar el tiempo en que tarda una rutina en ejecutarse, e impedir que se interrumpa durante su ejecución.

Típicamente, un programa ensamblador (*assembler* en inglés) moderno crea código objeto traduciendo instrucciones mnemónicas de lenguaje ensamblador en opcodes, y resolviendo los nombres simbólicos para las localizaciones de memoria y otras entidades. El uso de referencias simbólicas es una característica clave del lenguaje ensamblador, evitando tediosos cálculos y actualizaciones manuales de las direcciones después de cada modificación del programa. La mayoría de los ensambladores también incluyen facilidades

de macros para realizar sustitución textual – ej. generar cortas secuencias de instrucciones como expansión en línea en vez de llamar a subrutinas.

Los ensambladores son generalmente más simples de escribir que los compiladores para los lenguajes de alto nivel, y han estado disponibles desde los años 1950. Los ensambladores modernos, especialmente para las arquitecturas basadas en RISC, tales como MIPS, Sun SPARC, y HP PA-RISC, así como también para el x86 (-64), optimizan la planificación de instrucciones para explotar lasegmentación del CPU eficientemente.

En los compiladores para lenguajes de alto nivel, son el último paso antes de generar el código ejecutable.

Número de pasos

Hay dos tipos de ensambladores basados en cuántos pasos a través de la fuente son necesarios para producir el programa ejecutable.

- Los ensambladores de un solo paso pasan a través del código fuente una vez y asumen que todos los símbolos serán definidos antes de cualquier instrucción que los refiera.
- Los ensambladores del dos pasos crean una tabla con todos los símbolos y sus valores en el primer paso, después usan la tabla en un segundo paso para generar código. El ensamblador debe por lo menos poder determinar la longitud de cada instrucción en el primer paso para que puedan ser calculadas las direcciones de los símbolos.

La ventaja de un ensamblador de un solo paso es la velocidad, que no es tan importante como lo fue en un momento dados los avances en velocidad y capacidades del computador. La ventaja del ensamblador de dos pasos es que los símbolos pueden ser definidos dondequiera en el código fuente del programa. Esto permite a los programas ser definidos de maneras más lógicas y más significativas, haciendo los programas de ensamblador de dos paso más fáciles leer y mantener.

Ensambladores de alto nivel

Los más sofisticados ensambladores de alto nivel proporcionan abstracciones del lenguaje tales como:

- Estructuras de control avanzadas
- Declaraciones e invocaciones de procedimientos/funciones de alto nivel
- Tipos de datos abstractos de alto nivel, incluyendo las estructuras/records, uniones, clases, y conjuntos
- Procesamiento de macros sofisticado (aunque está disponible en los ensambladores ordinarios desde finales 1960 para el IBM/360, entre otras máquinas)
- Características de programación orientada a objetos

Uso del término

Note que, en el uso profesional normal, el término ensamblador es frecuentemente usado tanto para referirse al lenguaje ensamblador como también al programa ensamblador (que convierte el código fuenteescrito en el lenguaje ensamblador a código objeto que luego será enlazado para producir lenguaje de máquina). Las dos expresiones siguientes utilizan el término «ensamblador»:

- «El CP/CMS fue escrito en ensamblador del IBM S/360»
- «El ASM-H fue un ensamblador del S/370 ampliamente usado»

La primera se refiere al lenguaje y la segundo se refiere al programa.

El lenguaje ensamblador refleja directamente la arquitectura y las instrucciones en lenguaje de máquina de la CPU, y pueden ser muy diferentes de una arquitectura de CPU a otra. Cada arquitectura de microprocesador tiene su propio lenguaje de máquina, y en consecuencia su propio lenguaje ensamblador ya que este se encuentra muy ligado al la estructura del hardware para el cual se programa. Los microprocesadores difieren en el tipo y número de operaciones que soportan; también pueden tener diferente cantidad de registros,

y distinta representación de los tipos de datos en memoria. Aunque la mayoría de los microprocesadores son capaces de cumplir esencialmente las mismas funciones, la forma en que lo hacen difiere y los respectivos lenguajes ensamblador reflejan tal diferencia.

Instrucciones de CPU

La mayoría de los CPU tienen más o menos los mismos grupos de instrucciones, aunque no necesariamente tienen todas las instrucciones de cada grupo. Las operaciones que se pueden realizar varían de un CPU a otro. Un CPU particular puede tener instrucciones que no tenga otro y viceversa. Los primeros microprocesadores de 8 bits no tenían operaciones para multiplicar o dividir números, por ejemplo, y había que hacer subrutinas para realizar esas operaciones. Otros CPU puede que no tengan operaciones de punto flotante y habría que hacer o conseguir bibliotecas que realicen esas operaciones.

Las instrucciones del CPU pueden agruparse, de acuerdo a su funcionalidad, en:

Operaciones con enteros: (de 8, 16, 32 y 64 bits dependiendo de la arquitectura del CPU)

Estas son operaciones realizadas por la Unidad aritmético lógica del CPU

- Operaciones aritméticas. Como suma, resta, multiplicación, división, módulo, cambio de signo
- Operaciones booleanas. Operaciones lógicas bit a bit como AND, OR, XOR, NOT
- Operaciones de bits. Como desplazamiento y rotaciones de bits (hacia la derecha o hacia la izquierda, a través del bit del acarreo o sin él)
- Comparaciones

Operaciones de mover datos:

Entre los registros y la memoria:

Aunque la instrucción se llama «mover», en el CPU, «mover datos» significa en realidad copiar datos, desde un origen a un destino, sin que el dato desaparezca del origen.

Se pueden mover valores:

- desde un registro a otro
- desde un registro a un lugar de la memoria
- desde un lugar de la memoria a un registro
- desde un lugar a otro de la memoria
- un valor inmediato a un registro
- un valor inmediato a un lugar de memoria

Operaciones de stack:

- PUSH (escribe datos hacia el tope del stack)
- POP (lee datos desde el tope del stack)

Operaciones de entrada/salida:

Son operaciones que mueven datos de un registro, desde y hacia un puerto; o de la memoria, desde y hacia un puerto

- INPUT Lectura desde un puerto de entrada
- OUTPUT Escritura hacia un puerto de salida

Operaciones para el control del flujo del programa:

- Llamadas y retornos de subrutinas
- Llamadas y retornos de interrupciones
- Saltos condicionales de acuerdo al resultado de la comparaciones
- Saltos incondicionales

Operaciones con números reales:

El estándar para las operaciones con números reales en los CPU está definido por el IEEE 754.

Un CPU puede tener operaciones de punto flotante con números reales mediante el coprocesador numérico (si lo hay), como las siguientes:

- Operaciones aritméticas. Suma, resta, multiplicación, división, cambio de signo, valor absoluto, parte entera
- Operaciones trascendentales
    - Operaciones trigonométricas. Seno, coseno, tangente, arcotangente
    - Operaciones con logaritmos, potencias y raíces
- Otras

El lenguaje ensamblador tiene mnemónicos para cada una de las instrucciones del CPU en adición a otros mnemónicos a ser procesados por el programa ensamblador (como por ejemplo macros y otras sentencias en tiempo de ensamblado).

ENSAMBLADO

La transformación del lenguaje ensamblador en código máquina la realiza un programa ensamblador, y la traducción inversa la puede efectuar un desensamblador. A diferencia de los lenguajes de alto nivel, aquí hay usualmente una correspondencia 1 a 1 entre las instrucciones simples del ensamblador y el lenguaje de máquina. Sin embargo, en algunos casos, un ensamblador puede proveer «pseudo instrucciones» que se expanden en un código de máquina más extenso a fin de proveer la funcionalidad necesaria y simplificar la programación. Por ejemplo, para un código máquina condicional como «si X mayor o igual que», un ensamblador puede utilizar una pseudoinstrucción al grupo «haga si menor que», y «si = 0» sobre el resultado de la condición anterior. Los Ensambladores más completos también proveen un rico lenguaje de macros que se utiliza para generar código más complejo y secuencias de datos.

Para el mismo procesador y el mismo conjunto de instrucciones de CPU, diferentes programas ensambladores pueden tener, cada uno de ellos, variaciones y diferencias en el conjunto de mnemónicos o en la sintaxis de su lenguaje ensamblador. Por ejemplo, en un lenguaje ensamblador para la arquitectura x86, se puede expresar la instrucción para mover 5 al registro `AL` de la siguiente manera: `MOV AL, 5`, mientras que para otro ensamblador para la misma arquitectura se expresaría al revés: `MOV 5, AL`. Ambos lenguajes ensambladores harían exactamente lo mismo, solo que está expresado de manera diferente. El primero usa la sintaxis de Intel, mientras que el segundo usa la sintaxis de AT&T.

El uso del ensamblador no resuelve definitivamente el problema de cómo programar un sistema basado en microprocesador de modo sencillo ya que para hacer un uso eficiente del mismo, hay que conocer a fondo el microprocesador, los registros de trabajo de que dispone, la estructura de la memoria, y muchas cosas más referentes a su estructura básica de funcionamiento.

Ejemplo

Un programa escrito en lenguaje ensamblador consiste en una serie de instrucciones que corresponden al flujo de órdenes ejecutables por un microprocesador.

Por ejemplo, en el lenguaje ensamblador para un procesador x86:

La sentencia

- `MOV AL, 061h`

Asigna el valor hexadecimal 61 (97 decimal) al registro «AL«.

El programa ensamblador lee la sentencia de arriba y produce su equivalente binario en lenguaje de máquina

- Binario: `10110000 01100001` (hexadecimal: B061)

El mnemónico MOV es un *código de operación* u «opcode». El *opcode* es seguido por una lista de argumentos o *parámetros*, completando una típica instrucción de ensamblador. En el ejemplo, AL es un registro de 8 bits del procesador, al cual se le asignará el valor hexadecimal 61 especificado.

El código de máquina generado por el ensamblador consiste de 2 bytes. El primer byte contiene empaquetado la instrucción MOV y el código del registro hacia donde se va a mover el dato:

```
10110 000   01100001
  |    |        |
  |    |        +---- Número 61h en binario
  |    |
  |    +--- Registro AL
```

```
      +-------- Instrucción MOV
```

En el segundo byte se especifica el número 61h, escrito en binario como `01100001`, que se asignará al registro AL, quedando la sentencia ejecutable como:

- `10110000 01100001`

La cual puede ser entendida y ejecutada directamente por el procesador.

Elementos básicos

Hay un grado grande de diversidad en la manera en que los autores de los ensambladores categorizan las sentencias y en la nomenclatura que usan. En particular, algunos describen cualquier cosa como pseudo-operación (pseudo-Op), con excepción del mnemónico de máquina o del mnemónico extendido.

Un típico lenguaje ensamblador consiste en 3 tipos de sentencias de instrucción que son usadas para definir las operaciones del programa:

- Mnemónicos de opcode
- Secciones de datos
- Directivas de ensamblador

Mnemónicos de opcode y mnemónicos extendidos

A diferencia de las instrucciones (sentencias) de los lenguajes de alto nivel, instrucciones en el lenguaje ensamblador son generalmente muy simples. Generalmente, una mnemónico es un nombre simbólico para una sola instrucción en lenguaje de máquina ejecutable (un opcode), y hay por lo menos un mnemónico de opcode definido para cada instrucción en lenguaje de máquina.

Cada instrucción consiste típicamente en una operación u opcode más cero o más operandos. La mayoría de las instrucciones refieren a un solo valor, o a un par de valores. Los operandos pueden ser inmediatos (típicamente valores de un byte, codificados en la propia instrucción), registros especificados en la instrucción, implícitos o las direcciones de los datos localizados en otra parte de la memoria. Esto está determinado por la arquitectura subyacente del procesador, el ensamblador simplemente refleja cómo trabaja esta arquitectura. Los mnemónicos extendidos son frecuentemente usados para especificar una combinación de un opcode con un operando específico, ej, el ensamblador del System/360 usa a B como un mnemónico extendido para el BC con una máscara de 15 y NOP al BC con una máscara de 0.

Los mnemónicos extendidos son frecuentemente usados para soportar usos especializados de instrucciones, a menudo para propósitos no obvios con respecto al nombre de la instrucción. Por ejemplo, muchos CPU no tienen una instrucción explícita de NOP (No Operación), pero tienen instrucciones que puedan ser usadas para tal propósito. En el CPU 8086, la instrucción XCHG AX,AX (intercambia el registro AX consigo mismo) es usada para el NOP, con NOP siendo un pseudo-opcode para codificar la instrucción XCHG AX,AX. Algunos desensambladores reconocen esto y decodificarán la instrucciónXCHG AX,AX como NOP. Similarmente, los ensambladores de IBM para el System/360 usan los mnemónicos extendidos NOP y NOPR con las máscaras cero para BC y BCR.

Algunos ensambladores también soportan simples macroinstrucciones incorporadas que generan dos o más instrucciones de máquina. Por ejemplo, con algunos ensambladores para el Z80, la instrucción

```
LD HL, BC
```

genera las instrucciones

```
LD L, C
```

```
LD H, B.
```

LD HL, BC es un *pseudo-opcode*, que en este caso simula ser una instrucción de 16 bits, cuando se expande se producen dos instrucciones de 8 bits que equivalen a la simulada de 16 bits.

Secciones de datos

Hay instrucciones usadas para definir elementos de datos para manejar datos y variables. Definen el tipo de dato, la longitud y la alineación de los datos. Estas instrucciones también pueden definir si los datos están disponibles para programas exteriores (programas ensamblados separadamente) o solamente para el programa en el cual la sección de datos está definida. Algunos ensambladores clasifican estas instruccion

Directivas del ensamblador

Los directivas del ensamblador, también llamadas los pseudo opcodes, pseudo-operaciones o pseudo-ops, son instrucciones que son ejecutadas por un ensamblador en el tiempo de ensamblado, no por un CPU en el tiempo de ejecución. Pueden hacer al ensamblado del programa dependiente de parámetros entrados por un programador, de modo que un programa pueda ser ensamblado de diferentes maneras, quizás para diversos aplicaciones. También pueden ser usadas para manipular la presentación de un programa para hacerlo más fácil leer y mantener.

Por ejemplo, las directivas pudieran ser usadas para reservar áreas de almacenamiento y opcionalmente su para asignar su contenido inicial. Los nombres de las directivas a menudo comienzan con un punto para distinguirlas de las instrucciones de máquina.

Los ensambladores simbólicos le permiten a los programadores asociar nombres arbitrarios (etiquetas o símbolos) a posiciones de memoria. Usualmente, cada constante y variable tiene un nombre para que las instrucciones pueden referir a esas ubicaciones por nombre, así

promoviendo el código autodocumentado. En el código ejecutable, el nombre de cada subprograma es asociado a su punto de entrada, así que cualquier llamada a un subprograma puede usar su nombre. Dentro de subprogramas, a los destinos GOTO se le dan etiquetas. Algunos ensambladores soportan símbolos locales que son léxicamente distintos de los símbolos normales (ej, el uso de «10$» como un destino GOTO).

La mayoría de los ensambladores proporcionan un manejo flexible de símbolos, permitiendo a los programadores manejar diversos espacios de nombres, calcular automáticamente offsets dentro deestructuras de datos, y asignar etiquetas que refieren a valores literales o al resultado de cálculos simples realizados por el ensamblador. Las etiquetas también pueden ser usadas para inicializar constantes y variables con direcciones relocalizables.

Los lenguajes ensambladores, como la mayoría de los otros lenguajes de computador, permiten que comentarios sean añadidos al código fuente, que son ignorados por el programa ensamblador. El buen uso de los comentarios es aún más importante con código ensamblador que con lenguajes de alto nivel, pues el significado y el propósito de una secuencia de instrucciones es más duro de descifrar a partir del código en sí mismo.

El uso sabio de estas facilidades puede simplificar grandemente los problemas de codificar y mantener el código de bajo nivel. El código fuente de lenguaje ensamblador crudo generado por compiladores o desensambladores – código sin ningún comentario, ni símbolos con algún sentido, ni definiciones de datos – es muy difícil de leer cuando deben hacerse cambios.

Macros

Muchos ensambladores soportan macros predefinidos, y otras soportan macros definidos (y repetidamente redefinibles) por el programador que implican secuencias de líneas del texto en las cuales las variables y las constantes están empotradas. Esta secuencia de líneas de texto puede incluir opcodes o directivas. Una vez un macro ha sido definido, su nombre puede ser usado en lugar de un mnemónico. Cuando el ensamblador procesa tal sentencia, reemplaza la sentencia por las líneas del texto asociadas a ese macro, entonces las procesa

como si hubieran existido en el archivo del código fuente original (incluyendo, en algunos ensambladores, la expansión de cualquier macro que exista en el texto de reemplazo).

Puesto que las macros pueden tener nombres «cortos» pero se expanden a varias o de hecho muchas líneas de código, pueden ser usados para hacer que los programas en lenguaje ensamblador parezcan ser mucho más cortos, requiriendo menos líneas de código fuente, como sucede con los lenguajes de alto nivel. También pueden ser usados para añadir niveles de estructura más altos a los programas ensamblador, opcionalmente introducen código de depuración empotrado vía parámetros y otras características similares.

Muchos ensambladores tienen macros incorporados (o predefinidos) para las llamadas de sistema y otras secuencias especiales de código, tales como la generación y el almacenamiento de los datos realizados a través de avanzadas operaciones bitwise y booleanas usadas en juegos, software de seguridad, gestión de datos, y criptografía.

Los macro ensambladores a menudo permiten a los macros tomar parámetros. Algunos ensambladores incluyen lenguajes macro muy sofisticados, incorporando elementos de lenguajes de alto nivel tales como parámetros opcionales, variables simbólicas, condiciones, manipulaciones de strings operaciones aritméticas, todos usables durante la ejecución de un macro dado, y permitiendo a los macros guardar el contexto o intercambiar información. Así un macro puede generar un gran número de instrucciones o definiciones de datos en lenguaje ensamblador, basadas en los argumentos del macro. Esto pudiera ser usado para generar, por ejemplo, estructuras de datos de estilo de records o bucles «desenrrollados», o podría generar algoritmos enteros basados en parámetros complejos. Una organización, usando lenguaje ensamblador, que ha sido fuertemente extendido usando tal suite de macros, puede ser considerada que se está trabajando en un lenguaje de alto nivel, puesto que tales programadores no están trabajando con los elementos conceptuales de más bajo nivel del computador.

Las macros fueron usados para adaptar sistemas de software de gran escala para clientes específicos en la era del mainframe, y también fueron usados por el personal del cliente para

satisfacer las necesidades de sus patrones haciendo versiones específicas de los sistemas operativos del fabricante. Esto fue hecho, por ejemplo, por los programadores de sistema que trabajaban con el Conversational Monitor System / Virtual Machine (CMS/VM) de IBM y con los add-ons «real time transaction processing» de IBM, CICS, Customer Information Control System, y ACP/TPF, el airline/financial system que comenzó en los años 1970 y todavía corre con muchos sistemas de reservaciones computarizados (CRS) y sistemas de tarjeta de crédito de hoy.

También es posible usar solamente las habilidades de procesamiento de macros de un ensamblador para generar código escrito en lenguajes completamente diferentes, por ejemplo, para generar una versión de un programa en COBOL usando un programa macro ensamblador puro conteniendo líneas de código COBOL dentro de operadores de tiempo ensamblaje dando instrucciones al ensamblador para generar código arbitrario.

Esto era porque, como en los años 1970 fue observado, el concepto de «procesamiento de macro» es independiente del concepto de «ensamblaje», siendo el anterior, en términos modernos, más un procesamiento de textos, que una generación de código objeto. El concepto de procesamiento de macro apareció, y aparece, en el lenguaje de programación C, que soporta «instrucciones de preprocesador» de fijar variables, y hace pruebas condicionales en sus valores. Observe que a diferencia de ciertos macroprocesadores previos dentro de los ensambladores, el preprocesador de C no es Turing-completo porque carecía la capacidad de bucle o «go to», esto último permitiendo a los programas hacer bucles.

A pesar del poder del procesamiento macro, éste dejó de usarse en muchos lenguajes de alto nivel (una importante excepción es C/C++) mientras que seguía siendo perenne para los ensambladores. Esto era porque muchos programadores estaban bastante confundidos por la sustitución de parámetros macro y no distinguían la diferencia entre procesamiento macro, el ensamblaje y la ejecución.

La sustitución de parámetros macro es estrictamente por nombre: en el tiempo de procesamiento macro, el valor de un parámetro es sustituido textualmente por su nombre. La clase más famosa de bugs resultantes era el uso de un parámetro que en sí mismo era una

expresión y no un nombre primario cuando el escritor macro esperaba un nombre. En el macro:

```
foo: macro a

load a*b
```

la intención era que la rutina que llama proporcionaría el nombre de una variable, y la variable o constante «global» b sería usada para multiplicar a «a». Si foo es llamado con el parámetro `a-c`, ocurre la expansión macro `load a-c*b`. Para evitar cualquier posible ambigüedad, los usuarios de macro procesadores pueden encerrar en paréntesis los parámetros formales dentro de las definiciones de macros, o las rutinas que llaman pueden envolver en paréntesis los parámetos de entrada. Así, el macro correcto, con los paréntesis, sería:

```
foo: macro a

load (a)*b
```

y su expansión, daría como resultado: `load (a-c)*b`

El PL/I y el C/C++ ofrecen macros, pero la esta facilidad solo puede manipular texto. Por otra parte, los lenguajes homoicónicos, tales como Lisp, Prolog, y Forth, retienen el poder de los macros de lenguaje ensamblador porque pueden manipular su propio código como datos.

Algunos ensambladores han incorporado elementos de programación estructurada para codificar el flujo de la ejecución. El ejemplo más temprano de este acercamiento estaba en el Concept-14 macro set, originalmente propuesto por el Dr. H.D. Mills (marzo de 1970), e implementado por Marvin Kessler en la Federal Systems Division de IBM, que extendió el macro ensamblador del S/360 con bloques de control de flujo IF/ELSE/ENDIF y similares. Esto

era una manera de reducir o eliminar el uso de operaciones GOTO en el código en lenguaje ensamblador, uno de los principales factores que causabancódigo espagueti en el lenguaje ensamblador. Este acercamiento fue ampliamente aceptado a principios de los años 1980 (los últimos días del uso de lenguaje ensamblador en gran escala).

Un curioso diseño fue A-natural, un ensamblador «orientado a la corriente» (stream-oriented) para los procesadores 8080/Z80 de Whitesmiths Ltd. (desarrolladores del sistema operativo Idris, similar al Unix), y lo que fue reportado como el primer compilador C comercial). El lenguaje fue clasificado como un ensamblador, porque trabajaba con elementos de máquina crudos tales como opcodes, registros, y referencias de memoria; pero incorporaba una sintaxis de expresión para indicar el orden de ejecución. Los paréntesis y otros símbolos especiales, junto con construcciones de programación estructurada orientadas a bloques, controlaban la secuencia de las instrucciones generadas. A-natural fue construido como el lenguaje objeto de un compilador C, en vez de la codificación manual, pero su sintaxis lógica ganó algunos seguidores.

Ha habido poca demanda aparente para ensambladores más sofisticados debido a la declinación del desarrollo de lenguaje ensamblador de larga escala. A pesar de eso, todavía se están desarrollando y aplicando en casos donde las limitaciones de recursos o las particularidades en la arquitectura de sistema objetivo previenen el efectivo uso de lenguajes de alto nivel.

Perspectiva histórica

Los lenguajes ensambladores fueron primero desarrollados en los años 1950, cuando fueron referidos como lenguajes de programación de segunda generación. Por ejemplo, el SOAP (Symbolic Optimal Assembly Program) era un lenguaje ensamblador de 1957 para el computador IBM 650. Los lenguajes ensambladores eliminaron mucha de la propensión a errores y del consumo de tiempo de la programación de los lenguajes de primera generación que se necesitaba con los primeros computadores, liberando a los programadores del tedio tal como recordar códigos numéricos y cálculo de direcciones. Una vez fueron ampliamente usados para todo tipo de programación. Sin embargo, por los años 1980 (1990

en los microcomputadores), su uso había sido en gran parte suplantado por los lenguajes de alto nivel, en la búsqueda de una mejorada productividad en programación. Hoy en día, aunque el lenguaje ensamblador es casi siempre manejado y generado por los compiladores, todavía se usa para la manipulación directa del hardware, acceso a instrucciones especializadas del procesador, o para resolver problemas de desempeño crítico. Los usos típicos son drivers de dispositivo, sistemas embebidos de bajo nivel, y sistemas de tiempo real.

Históricamente, un gran número de programas han sido escritos enteramente en lenguaje ensamblador. Los sistemas operativos fueron casi exclusivamente escritos en lenguaje ensamblador hasta la aceptación ámplia del lenguaje de programación C en los años 1970 y principios de los 1980. También, muchas aplicaciones comerciales fueron escritas en lenguaje ensamblador, incluyendo una gran cantidad del software escrito por grandes corporaciones para mainframes de IBM. Los lenguajes COBOL y FORTRAN eventualmente desplazaron mucho de este trabajo, aunque un número de organizaciones grandes conservaran las infraestructuras de aplicaciones en lenguaje ensamblador hasta bien entrados los años 1990.

La mayoría de los primeros microcomputadores confiaron en el lenguaje ensamblador codificado a mano, incluyendo la mayoría de los sistemas operativos y de las aplicaciones grandes. Esto era porque estos sistemas tenían limitaciones severas de recursos, impusieron idiosincráticas arquitecturas de memoria y de pantalla, y proporcionaron servicios de sistema limitados y con errores. Quizás más importante era la falta de compiladores de primera clase de lenguajes de alto nivel adecuados para el uso en el microcomputador. Un factor psicológico también pudo haber jugado un papel: la primera generación de programadores de los microcomputadores conservó una actitud de aficionado de «alambres y alicates».

En un contexto más comercial, las más grandes razones para usar el lenguaje ensamblador era hacer programas con mínimo tamaño, mínima sobrecarga, mayor velocidad y confiabilidad.

Los típicos ejemplos de programas grandes en lenguaje ensamblador de ese tiempo son los

sistemas operativos IBM PC DOS y aplicaciones tempranas tales como la hoja de cálculo Lotus 1-2-3, y casi todos los juegos populares para la familia Atari 800 de computadores personales. Incluso en los años 1990, la mayoría de los videojuegos de cónsola fueron escritos en ensamblador, incluyendo la mayoría de los juegos para la Mega Drive/Genesis y el Super Nintendo Entertainment System. Según algunos insiders de la industria, el lenguaje ensamblador era el mejor lenguaje de programación a usar para obtener el mejor desempeño del Sega Saturn, una cónsola para la cual era notoriamente desafiante desarrollar y programar juegos. El popular juego de arcade NBA Jam (1993) es otro ejemplo. El ensamblador ha sido por largo trecho, el lenguaje de desarrollo primario en los computadores hogareños Commodore 64, Atari ST, así como el ZX Spectrum. Esto fue así en gran parte porque los dialectos delBASIC en estos sistemas ofrecieron insuficiente velocidad de ejecución, así como insuficientes características para aprovechar completamente el hardware disponible. Algunos sistemas, más notablemente el Amiga, incluso tienen IDEs con características de depuración y macros altamente avanzados, tales como el freeware ASM-One assembler, comparable a las del Microsoft Visual Studio (el ASM-Uno precede al Microsoft Visual Studio).

El ensamblador para el VIC-20 fue escrito por Don French y publicado por French Silk. Con 1639 bytes de longitud, su autor cree que es el más pequeño ensamblador simbólico jamás escrito. El ensamblador soportaba el direccionamiento simbólico usual y la definición de cadenas de caracteres o cadenas hexadecimales. También permitía expresiones de direcciones que podían combinarse con las operaciones de adición, substracción, multiplicación, división, AND lógico, OR lógico, y exponenciación.

Uso actual

Siempre ha habido debates sobre la utilidad y el desempeño del lenguaje ensamblador relativo a lenguajes de alto nivel. El lenguaje ensamblador tiene nichos específicos donde es importante (ver abajo). Pero, en general, los modernos compiladores de optimización para traducir lenguajes de alto nivel en código que puede correr tan rápidamente como el lenguaje ensamblador escrito a mano, a pesar de los contraejemplos que pueden ser encontrados. La complejidad de los procesadores modernos y del subsistema de memoria

hace la optimización efectiva cada vez más difícil para los compiladores, así como para los programadores en ensamblador. Adicionalmente, y para la consternación de los amantes de la eficiencia, el desempeño cada vez mayor del procesador ha significado que la mayoría de los CPU estén desocupados la mayor parte del tiempo, con retardos causados por embotellamientos predecibles tales como operaciones de entrada/salida y paginación de memoria. Esto ha hecho que la velocidad de ejecución cruda del código no sea un problema para muchos programadores.

Hay algunas situaciones en las cuales los profesionales pudieran elegir utilizar el lenguaje ensamblador. Por ejemplo cuando:

- es requerido un ejecutable binario independiente (stand-alone), es decir uno que deba ejecutarse sin recursos a componentes de tiempo de ejecución o a bibliotecas asociadas con un lenguaje de alto nivel; ésta es quizás la situación más común. Son programas empotrados que solo almacenan una pequeña cantidad de memoria y el dispositivo está dirigido para hacer tareas para un simple propósito. Ejemplos consisten en teléfonos, sistemas de combustible e ignición para automóviles, sistemas de control del aire acondicionado, sistemas de seguridad, y sensores
- interactuando directamente con el hardware, por ejemplo en drivers de dispositivo y manejadores de interrupción
- usando instrucciones específicas del procesador no explotadas o disponibles por el compilador. Un ejemplo común es la instrucción de rotación bitwise en el núcleo de muchos algoritmos de cifrado
- creando funciones vectorizadas para programas en lenguajes de alto nivel como C. En el lenguaje de alto nivel esto es a veces ayudado por funciones intrínsecas del compilador que mapean directamente a los mnemónicos del SIMD, pero sin embargo resulta en una conversión de ensamblador de uno a uno para un procesador de vector asociado
- es requerida la optimización extrema, ej, en un bucle interno en un algoritmo intensivo en el uso del procesador. Los programadores de juegos toman ventaja de las habilidades de las características del hardware en los sistemas, permitiendo a los juegos correr más rápidamente. También las grandes simulaciones científicas

requieren algoritmos altamente optimizados, ej, álgebra lineal con BLAS o la transformada de coseno discreta (ej, la versión SIMD en ensamblador del x264, (una biblioteca para codificar streams de video)

- un sistema con severas limitaciones de recursos (ej, un sistema empotrado) debe ser codificado a mano para maximizar el uso de los limitados recursos; pero esto está llegando a ser menos común a medida que el precio del procesador decrece y el desempeño mejora
- no existe ningún lenguaje de alto nivel, en un procesador nuevo o especializado, por ejemplo
- escribiendo programas de tiempo real que necesitan sincronización y respuestas precisas, tales como sistemas de navegación de vuelo, y equipo médico. Por ejemplo, en un sistema fly-by-wire (vuelo por mandos eléctricos), la telemetría debe ser interpretada y hay que actuar dentro de limitaciones estrictas de tiempo. Tales sistemas deben eliminar fuentes de retrasos impredecibles, que pueden ser creados por (algunos) lenguajes interpretados, recolección de basura automática, operaciones de paginación, o multitarea preventiva. Sin embargo, algunos lenguajes de alto nivel incorporan componentes de tiempo de ejecución e interfaces de sistema operativo que pueden introducir tales retrasos. Elegir el ensamblador o lenguajes de bajo nivel para tales sistemas da a los programadores mayor visibilidad y control sobre el proceso de los detalles
- es requerido control total sobre el ambiente, en situaciones de seguridad extremadamente alta donde nada puede darse por sentado.
- se escriben virus de computadora, bootloaders, ciertos drivers de dispositivo, u otros elementos muy cerca del hardware o al sistema operativo de bajo nivel
- se escriben simuladores del conjunto de instrucciones para monitoreo, trazado y depuración de errores donde la sobrecarga adicional es mantenida al mínimo
- se hace ingeniería inversa en binarios existentes que pueden o no haber sido escritos originalmente en un lenguaje de alto nivel, por ejemplo al crackear la protección anticopia del software propietario.
- se hace ingeniería inversa y modificación de video juegos (también denominado ROM hacking), que es posible por medio de varios métodos. El más ampliamente implementado es alterando el código del programa a nivel de lenguaje ensamblador

- se escribe código automodificable, algo para lo que el lenguaje ensamblador se presta bien
- se escriben juegos y otros softwares para calculadoras gráficas
- se escribe software compilador que genera código ensamblador, y por lo tanto los desarrolladores deben ser programadores de lenguaje ensamblador
- se escriben algoritmos criptográficos que siempre deben tomar estrictamente el mismo tiempo para ejecutar, previniendo ataques de tiempo

Sin embargo, el lenguaje ensamblador es todavía enseñado en la mayoría de los programas de ciencias de la computación e ingeniería electrónica. Aunque hoy en día, pocos programadores trabajan regularmente con el lenguaje ensamblador como una herramienta, los conceptos fundamentales continúan siendo muy importantes.

Tales tópicos fundamentales, como aritmética binaria, asignación de memoria, procesamiento del stack, codificación de conjunto de caracteres, procesamiento de interrupciones, y diseño de compiladores, serían duros de estudiar en detalle sin la comprensión de cómo el computador opera a nivel del hardware. Puesto que el comportamiento del computador es fundamentalmente definido por su conjunto de instrucciones, la manera lógica de aprender tales conceptos es estudiar un lenguaje ensamblador. La mayoría de los computadores modernos tienen un conjunto de instrucciones similares.

Por lo tanto, estudiar un solo lenguaje ensamblador es suficiente para aprender: i) los conceptos básicos; ii) reconocer situaciones donde el uso de lenguaje ensamblador puede ser apropiado; y iii) ver cómo el código ejecutable eficiente puede ser creado por los lenguajes de alto nivel.