Universidad Nacional Autónoma de México

Facultad de Ingeniería

Ensamblador IA-32 (x86) de 32 Bits de Una Sola Pasada

Grupo: 5

Alumnos:

De la Fuente Flores Guadalupe Isabela
Hernández Castro Laura Isabel
Martínez Rosales Hugo Armando
Pérez Carmona José Bruno

Profesora: Ing. Ariel Adara Mercado Martínez
Semestre: 2025-2

1. Introducción

Descripción del problema

En el mundo del desarrollo de sistemas a bajo nivel, la traducción del código ensamblador a código máquina es una tarea muy importante para que el hardware ejecute instrucciones de manera eficiente., Los ensambladores han utilizado un enfoque de múltiples pasadas, lo que significa hacer varios recorridos sobre el código fuente para resolver símbolos, etiquetas y direcciones. Esto puede complicar las cosas y alargar los tiempos de procesamiento. Aunque este método funciona, puede ser poco eficiente en situaciones donde se necesita rapidez, simplicidad o cuando los recursos del sistema son limitados.

Este proyecto busca abordar este desafío mediante la creación de un ensamblador que funcione en una sola pasada, es decir, pretende llevar a cabo todo el proceso de traducción en un solo recorrido del código fuente, resolviendo de inmediato las instrucciones y referencias necesarias. Con esto, se espera mejorar el rendimiento del proceso de ensamblado, reducir el consumo de recursos y hacer que el ensamblador sea más adecuado para sistemas embebidos, entornos educativos o aplicaciones específicas donde la eficiencia es algo fundamental.

2. Desarrollo

El análisis del problema parte de la necesidad de desarrollar un ensamblador simple, funcional y eficiente para la arquitectura IA-32, con la condición particular de operar en una sola pasada sobre el código fuente.

Análisis

- Se necesita un ensamblador que procese instrucciones en lenguaje ensamblador para la arquitectura IA-32.

- El ensamblador debe trabajar en una sola pasada, es decir, recorrer el código fuente una vez y generar el código máquina directamente.

- Reservar espacio o generar instrucciones provisionales para saltos a etiquetas que aún no han sido definidas y guardar estas referencias en una tabla de referencias pendientes.

- Una vez que encuentra la definición de la etiqueta, volver atrás y completar o corregir el código generado.

- Requiere generar una tabla de símbolos con las direcciones asociadas a las etiquetas y una tabla de referencias pendientes para manejar saltos no definidos en el momento.

- La salida debe ser el código máquina generado, expresado en hexadecimal.

Diseño de la solución

Se plantea un sistema modular compuesto por:

- Un lector de línea para procesar el código fuente instrucción por instrucción.

- Un generador de tabla de símbolos que asigna direcciones a las etiquetas.

- Un manejador de referencias pendientes que registra las instrucciones con etiquetas aún no definidas.

- Un codificador de instrucciones que genera directamente los bytes del código máquina, los cuales se representan en hexadecimal para su salida.

Esta estructura modular permite facilitar el mantenimiento del código, la detección de errores y la futura ampliación del ensamblador con nuevas instrucciones o características.

3. Implementación

El ensamblador fue desarrollado en Python 3, debido a su facilidad para manipular cadenas de texto, su sintaxis clara y su manejo eficiente de estructuras de datos como lo son las listas y diccionarios. Esto permitió un desarrollo mucho más ágil y legible del código.

Se diseñó una estructura centrada en una clase principal llamada **EnsambladorIA32**, la cual coordina el proceso completo de ensamblado. Esta clase se encarga de leer el código fuente línea por línea, identificar instrucciones, etiquetas y operandos, y generar el código máquina correspondiente.

Para analizar cada instrucción, se utilizaron expresiones regulares, permitiendo extraer fácilmente los componentes de cada línea (opcode, operandos, etiquetas). Cada mnemónico se analiza en funciones específicas como generar_mov, generar_add, etc., las cuales codifican directamente los opcodes y operandos.

El proceso de generación del código máquina utiliza valores enteros para representar directamente los bytes de cada instrucción, estos bytes se muestran en formato hexadecimal como salida final. Las etiquetas se almacenan en una tabla de símbolos, donde se les asigna una dirección de manera secuencial conforme se avanza en el código.

En el caso de encontrar referencias a etiquetas aún no definidas (saltos hacia adelante), se registran en una tabla de referencias pendientes. Estas referencias son resueltas una vez que se ha identificado la definición correspondiente de la etiqueta.

Durante la implementación se presentaron algunos retos, especialmente al manejar referencias futuras sin hacer múltiples pasadas sobre el código. Resolver esto implicó diseñar una estructura eficiente para mantener y actualizar tanto la tabla de símbolos como las referencias pendientes, garantizando la coherencia del código máquina generado.

4. Pruebas

```
; Programa de prueba para ensamblador IA-32

inicio:
    MOV EAX, EBX      ; Mueve el valor de EBX a EAX
    ADD EAX, 10       ; Suma 10 a EAX
    SUB EAX, ECX      ; Resta ECX de EAX
    CMP EAX, ECX      ; Compara EAX con ECX
    JE iguales        ; Salta a 'iguales' si EAX == ECX
    MOV ECX, 0x20     ; Mueve el valor inmediato 0x20 a ECX
    JNE diferentes    ; Salta a 'diferentes' si EAX != ECX
iguales:
    MOV EDX, 0x1      ; Si son iguales, mueve 0x1 a EDX
    JMP fin           ; Salta al final
diferentes:
    MOV EDX, 0x2      ; Si son diferentes, mueve 0x2 a EDX

fin:
    ; Fin del programa

```

Figura 1. Programa de prueba 1 (*programa.asm*)

```
In [1]: %runfile C:/Users/sechs/OneDrive/Desktop/EyPC/Proyecto/ensamblador.py --wdir

Etiqueta 'inicio' definida en 0000
Generado MOV reg,reg: opcode 89 modrm D8
Generado ADD reg,imm: opcode 81 modrm C0 imm 10
Generado SUB reg,reg: opcode 29 modrm C8
Generado CMP reg,reg: opcode 39 modrm C8
Error: Etiqueta 'IGUALES' no definida.
Generado MOV reg,imm: opcode B9 imm 32
Error: Etiqueta 'DIFERENTES' no definida.
Etiqueta 'iguales' definida en 0011
Generado MOV reg,imm: opcode BA imm 1
Error: Etiqueta 'FIN' no definida.
Etiqueta 'diferentes' definida en 0016
Generado MOV reg,imm: opcode BA imm 2
Etiqueta 'fin' definida en 001B
Error: La etiqueta 'IGUALES' no está definida.
Error: La etiqueta 'DIFERENTES' no está definida.
Error: La etiqueta 'FIN' no está definida.

Código máquina generado:
89 D8 81 C0 0A 00 00 00 29 C8 39 C8 B9 20 00 00 00 BA 01 00 00 00 BA 02 00 00 00
```

Figura 2. Proceso y generación del código máquina correspondiente a *programa.asm*

```
; Programa de prueba para ensamblador IA-32

inicio:
    MOV EAX, EBX      ; Mueve el valor de EBX a EAX
    ADD EAX, 10       ; Suma 10 a EAX
    SUB EAX, ECX      ; Resta ECX de EAX
    CMP EAX, ECX      ; Compara EAX con ECX
    JE iguales        ; Salta a 'iguales' si EAX == ECX
    MOV ECX, 0x20     ; Mueve el valor inmediato 0x20 a ECX
    JNE diferentes    ; Salta a 'diferentes' si EAX != ECX
iguales:
    MOV EDX, 0x1      ; Si son iguales, mueve 0x1 a EDX
    JMP fin           ; Salta al final
diferentes:
    MOV EDX, 0x2      ; Si son diferentes, mueve 0x2 a EDX

fin:
    ; Fin del programa

```

Figura 3. Programa de prueba 2 (*programa1.asm*)

```
In [2]: %runfile C:/Users/sechs/OneDrive/Desktop/EyPC/Proyecto/ensamblador.py --wdir

Etiqueta 'inicio' definida en 0000
Generado MOV reg,reg: opcode 89 modrm D8
Generado ADD reg,imm: opcode 81 modrm C0 imm 10
Generado SUB reg,reg: opcode 29 modrm C8
Generado CMP reg,reg: opcode 39 modrm C8
Error: Etiqueta 'IGUALES' no definida.
Generado MOV reg,imm: opcode B9 imm 32
Error: Etiqueta 'DIFERENTES' no definida.
Etiqueta 'iguales' definida en 0011
Generado MOV reg,imm: opcode BA imm 1
Error: Etiqueta 'FIN' no definida.
Etiqueta 'diferentes' definida en 0016
Generado MOV reg,imm: opcode BA imm 2
Etiqueta 'fin' definida en 001B
Error: La etiqueta 'IGUALES' no está definida.
Error: La etiqueta 'DIFERENTES' no está definida.
Error: La etiqueta 'FIN' no está definida.

Código máquina generado:
89 D8 81 C0 0A 00 00 00 29 C8 39 C8 B9 20 00 00 00 BA 01 00 00 00 BA 02 00 00 00
```

Figura 4. Proceso y generación del código máquina correspondiente a *programa1.asm*

```
; Programa de prueba para ensamblador IA-32

inicio:
    MOV EAX, 0x5      ; Carga 0x5 a EAX
    MOV EBX, 0x5      ; Carga 0x5 a EBX
    MOV ECX, 0x5      ; Carga 0x5 a ECX

    CMP EAX, EBX      ; Compara el valor de EAX con el de EBX
    JNE diferentes    ; Salta a 'diferentes' si EAX != EBX

    CMP EAX, ECX      ; Compara el valor de EAX con el de ECX
    JNE diferentes    ; Salta a 'diferentes' si EAX != ECX

iguales:
    MOV EDX, 0x1      ; Si son iguales, mueve 0x1 a EDX
    JMP fin           ; Salta al fin

diferentes:
    MOV EDX, 0x2      ; Si son diferentes, mueve 0x2 a EDX

fin:
    ; Fin del programa

```

Figura 5. Programa de prueba 3 (*programa2.asm*)

```
In [3]: %runfile C:/Users/sechs/OneDrive/Desktop/EyPC/Proyecto/ensamblador.py --wdir

Etiqueta 'inicio' definida en 0000
Generado MOV reg,imm: opcode B8 imm 5
Generado MOV reg,imm: opcode BB imm 5
Generado MOV reg,imm: opcode B9 imm 5
Generado CMP reg,reg: opcode 39 modrm D8
Error: Etiqueta 'DIFERENTES' no definida.
Generado CMP reg,reg: opcode 39 modrm C8
Error: Etiqueta 'DIFERENTES' no definida.
Etiqueta 'iguales' definida en 0013
Generado MOV reg,imm: opcode BA imm 1
Error: Etiqueta 'FIN' no definida.
Etiqueta 'diferentes' definida en 0018
Generado MOV reg,imm: opcode BA imm 2
Etiqueta 'fin' definida en 001D
Error: La etiqueta 'DIFERENTES' no está definida.
Error: La etiqueta 'FIN' no está definida.

Código máquina generado:
B8 05 00 00 00 BB 05 00 00 00 B9 05 00 00 00 39 D8 39 C8 BA 01 00 00 00 BA 02 00 00 00
```

Figura 6. Proceso y generación del código máquina correspondiente a *programa2.asm*

5. Manual de Usuario

Antes de ejecutar el ensamblador, asegúrese de contar con lo siguiente:

- Python 3.x instalado en su sistema.
- Un archivo *.asm* con instrucciones válidas para este ensamblador.
- Acceso a una consola o terminal para ejecutar scripts en Python.

Ejecución:

- Crea un archivo llamado **programa.asm** en el mismo directorio donde se encuentra el script del ensamblador.
- Escribe tus instrucciones en formato ensamblador, respetando la sintaxis definida.
- En la terminal, ejecuta el ensamblador con el comando: *python nombre_del_script.py.*

Tras la ejecución, se generarán los siguientes archivos de salida:

- simbolos.txt: contiene todas las etiquetas definidas en el código, junto con su dirección en hexadecimal.
- referencias.txt: muestra las instrucciones que hacen referencia a etiquetas no definidas.
- Salida por consola: se imprime el código máquina generado en formato hexadecimal.

Errores comunes:

- Etiqueta duplicada: dos etiquetas con el mismo nombre.
- Instrucción mal formada: falta de operandos o hay errores de sintaxis.
- Operando inválido:uso de registros incorrectos o mal escritos.
- Etiqueta no definida: intento de salto a una etiqueta inexistente.

- Modo de direccionamiento no soportado: sólo registros de 32 bits (EAX, EBX, ECX, EDX) e inmediatos están permitidos.

6. Manual Técnico

Descripción general:

Este proyecto implementa un ensamblador de una pasada para la arquitectura IA-32, escrito en Python. El objetivo principal es traducir instrucciones en lenguaje ensamblador a código máquina en formato hexadecimal, manejando etiquetas y referencias a saltos.

Estructura del programa:

El ensamblador está organizado en una estructura dentro de un único archivo Python, donde la clase EnsambladorIA32 encapsula la lógica principal, Entre sus componentes clave se encuentran:

- Lector de líneas: Recorre el archivo .asm línea por línea.
- Analizador de instrucciones: Usa expresiones regulares para identificar etiquetas, opcodes y operandos.
- Tabla de símbolos: Diccionario que asocia etiquetas con direcciones en hexadecimal.
- Tabla de referencias pendientes: Lista de instrucciones que hacen uso de etiquetas aún no definidas.
- Codificador: Genera directamente los bytes de código máquina en formato hexadecimal, usando funciones por instrucción en lugar de un diccionario de opcodes.

Flujo de ejecución:

1. Se carga el archivo fuente (programa.asm).
2. Se analiza cada línea para registrar etiquetas con su respectiva dirección, codificar instrucciones válidas directamente y detectar instrucciones con etiquetas no definidas y marcarlas como pendientes.
3. Durante la única pasada, las instrucciones con referencias pendientes se almacenan temporalmente. Al encontrar la definición de la etiqueta, se actualizan sus direcciones antes de finalizar el ensamblado.

Archivos generados:

- simbolos.txt: Guarda las etiquetas con su dirección hexadecimal.

- referencias.txt: Muestra referencias a etiquetas que no pudieron resolverse.
- Salida estándar: Se imprime el código máquina en hexadecimal.

Limitaciones:

- Solo se permiten instrucciones básicas (MOV, ADD, SUB, CMP, JMP, JE, JNE).
- No se manejan secciones .data, .text, ni pseudoinstrucciones.
- Los registros soportados son de 32 bits: EAX, EBX, ECX, EDX.
- No se incluye validación de direccionamientos indirectos, instrucciones con múltiples operandos complejos, ni soporte para macros.

7. Conclusiones

Individuales:

- De la Fuente Flores Guadalupe Isabela: En conclusión, este proyecto fue una valiosa oportunidad para reforzar nuestros conocimientos en programación de bajo nivel y arquitectura de computadoras. La implementación de un ensamblador de una sola pasada no solo nos permitió aplicar los conceptos teóricos vistos en clase pero aplicados a un contexto práctico, sino que también nos enfrentó a desafíos un poco más técnicos que a su vez impulsaron nuestro crecimiento en el diseño algorítmico y estructuración de código. El resultado fue una herramienta funcional y en un futuro (como propuesta de mejora) extensible, con bases sólidas para futuras mejoras y ampliaciones.

- Hernández Castro Laura Isabel: La elaboración de este proyecto, tanto la codificación como el documento, me permitió identificar las áreas de oportunidad existentes en mis conocimientos adquiridos de la materia, y de programación. Cada una de las partes que conforman el resultado final del mismo representó, en su momento, una oportunidad de analizar y comprender detalladamente el funcionamiento de un ensamblador de una sola pasada y su fundamental importancia en la programación de bajo nivel, además de un reto que supuso la creación e implementación de soluciones eficientes y adecuadas al codificar.

- Martínez Rosales Hugo Armando: Crear este ensamblador de una sola pasada fue una experiencia muy educativa, ya que me permitió poner en práctica de forma tangible ideas que hasta ahora solo había aprendido de manera teórica. Afrontar el desafío de convertir instrucciones de ensamblador a código máquina en un único recorrido del código fuente me brindó una mejor comprensión de la arquitectura IA-32 y la necesidad de optimizar recursos en ambientes de bajo nivel. La implementación de tablas de símbolos y referencias

pendientes me pareció especialmente interesante, ya que fueron fundamentales para resolver cuestiones relacionadas con saltos y etiquetas sin definir. Este proyecto no solo mejoró mis competencias en programación con Python, sino que también me hizo apreciar la lógica detrás de compiladores y ensambladores, y despertó en mí un mayor interés por la creación de herramientas que se comunican directamente con el hardware. Considero que es un buen punto de partida para el desarrollo de proyectos futuros más sofisticados y completos.

- Pérez Carmona José Bruno: Este proyecto me permitió comprender de forma más clara cómo opera un ensamblador, en especial para la arquitectura IA-32. Al desarrollar un generador de código máquina en python, reforcé mis conocimientos sobre el funcionamiento de instrucciones, etiquetas, saltos condicionales y la estructura básica de un programa en ensamblador. Además, profundicé en temas clave como el manejo de memoria, el análisis de instrucciones y la representación binaria y hexadecimal. Aunque el proyecto es una versión simplificada, representa una base sólida sobre la cual puedo construir ensambladores más completos o integrarlos a compiladores más avanzados.

Generales:

Este proyecto fue una experiencia muy enriquecedora la cual nos permitió acercarnos de manera práctica al desarrollo a bajo nivel, un área clave en la programación de computadoras. A través de la implementación de un ensamblador, profundizamos en el entendimiento del lenguaje ensamblador y del proceso de traducción a código máquina, lo cual reforzó nuestros conocimientos sobre la arquitectura IA-32.

El ensamblador cumple con los objetivos establecidos: procesa el código en una sola pasada, maneja etiquetas y referencias futuras mediante el uso de tablas auxiliares y genera como salida el código máquina en formato hexadecimal. Aunque actualmente soporta un subconjunto de instrucciones ya definidas, su diseño modular permite que se amplíe fácilmente para incluir más instrucciones o características.

Uno de los principales retos fue implementar el ensamblador en una sola pasada, ya que esto implicó desarrollar estrategias como el uso de referencias pendientes, pero al superar estos desafíos fortaleció nuestras habilidades en diseño algorítmico y lógica de programación.

Como mejora futura, consideramos agregar soporte para macros, instrucciones extendidas y una mejor gestión de errores, con el fin de hacer el ensamblador más funcional.