Desarrollo de un Ensamblador Hack en Python

Equipo Nucleo Digital

1 Introducción

El presente trabajo se enfoca en la construcción de un ensamblador Hack utilizando Python, como parte del Proyecto 6 del curso **Nand2Tetris**. El objetivo del ensamblador es convertir archivos de ensamblado Hack con extensión `.asm` a archivos binarios `.hack`, los cuales pueden ser ejecutados en la máquina Hack diseñada en proyectos previos.

El propósito principal de este proyecto es aplicar conocimientos de ensamblador, traducción de lenguajes y manejo de estructuras de datos en la implementación de un ensamblador funcional. Este ensamblador toma instrucciones en lenguaje ensamblador Hack y las traduce a su correspondiente representación binaria, que luego puede ser ejecutada en el simulador de CPU proporcionado por Nand2Tetris.

2 Metodología

La construcción del ensamblador se realizó en varias fases clave, con un enfoque modular para garantizar la claridad y escalabilidad del código.

2.1 Fase 1: Procesamiento de Archivos de Ensamblador

El ensamblador necesita leer y procesar el archivo de ensamblador Hack, eliminando espacios en blanco y comentarios, y preparando las instrucciones para su traducción. Para esto, se desarrolló la clase `HackFileProcessor`, que cumple con las siguientes funciones:

- **Lectura del archivo**: Se lee el archivo línea por línea, eliminando comentarios y espacios en blanco.
- **Preparación de las instrucciones**: Las instrucciones se almacenan en una lista para ser procesadas posteriormente.
- **Identificación de comandos**: Se implementaron métodos para identificar si una instrucción es del tipo A, C o L (etiquetas), lo que resulta crucial para su correcta traducción.

2.2 Fase 2: Traducción a Binario

Una vez que las instrucciones están limpias y organizadas, el siguiente paso es su traducción a código binario. Para ello, se desarrolló la clase `BinaryTranslator`, que se encarga de convertir los mnemónicos del ensamblador (como 'D', 'M', 'D+M', 'JMP', etc.) en su representación binaria correspondiente. Esta clase contiene tres tablas de traducción fundamentales:

- **Tabla de destino (dest)**: Traduce los destinos de una instrucción C (por ejemplo, 'D', 'MD', 'A').
- **Tabla de cómputo (comp)**: Traduce las operaciones aritméticas y lógicas (por ejemplo, 'D+M', 'A-1', '!D').
- **Tabla de salto (jump)**: Traduce las instrucciones de salto (por ejemplo, 'JGT', 'JEQ', 'JMP').

Este paso es fundamental ya que convierte las instrucciones simbólicas en código binario que puede ser interpretado directamente por la máquina Hack.

2.3 Fase 3: Manejo de Símbolos

El lenguaje ensamblador Hack permite el uso de etiquetas y variables simbólicas, las cuales deben ser gestionadas mediante una tabla de símbolos. Para esto, se desarrolló la clase `SymbolTable`, que se encarga de:

- **Gestión de símbolos predefinidos**: La tabla de símbolos contiene entradas predefinidas, como 'R0' a 'R15', 'SCREEN', 'KBD', etc., que tienen direcciones de memoria fijas.
- **Añadir etiquetas**: Las etiquetas definidas en el código ensamblador se añaden a la tabla de símbolos durante la primera pasada.
- **Asignación de direcciones a variables**: En la segunda pasada, las variables son asignadas a direcciones de memoria a partir de la dirección 16.

2.4 Fase 4: Pasadas del Ensamblador

El ensamblador realiza dos pasadas sobre el archivo `.asm`:

1. **Primera pasada**: Se registran las etiquetas en la tabla de símbolos, ya que estas indican la dirección de memoria de una instrucción en particular.
2. **Segunda pasada**: Se traduce cada instrucción A y C a código binario. Las variables simbólicas no predefinidas son asignadas a direcciones de memoria a partir de la dirección 16.

3 Código

El ensamblador se ha diseñado de manera modular, con tres clases principales que cumplen roles específicos:

- `HackFileProcessor`: Procesa y limpia el archivo de entrada, y organiza las instrucciones para su traducción.
- `BinaryTranslator`: Se encarga de convertir los mnemónicos del ensamblador Hack en binario utilizando tablas predefinidas.
- `SymbolTable`: Administra la tabla de símbolos, manejando las direcciones de memoria de variables y etiquetas.

Cada una de estas clases colabora para transformar las instrucciones en código binario que pueda ser ejecutado en el simulador de la CPU Hack.

4 Proceso de Verificación

Para verificar el funcionamiento correcto del ensamblador, se realizaron las siguientes pruebas:

1. **Prueba en el IDE Online**: El ensamblador fue probado utilizando el IDE Online de Nand2Tetris. El archivo `Prog.asm` fue traducido a un archivo `Prog.hack` utilizando el ensamblador desarrollado en Python.
2. **Comparación de archivos**: El archivo `Prog.hack` generado fue comparado con el archivo binario de referencia proporcionado por el curso en el IDE Online. La herramienta de comparación del ensamblador del IDE fue utilizada para asegurarse de que los archivos fueran idénticos.
3. **Simulación en CPU Emulator**: El archivo `Prog.hack` fue cargado en el simulador de CPU proporcionado por Nand2Tetris para verificar que el comportamiento del programa fuera el esperado.

5 Resultados

Los archivos generados por el ensamblador fueron comparados con éxito utilizando el IDE Online, y todos los resultados. Esto indica que el ensamblador fue implementado correctamente y que las instrucciones fueron traducidas de manera precisa.