Pontificia Universidad Católica de Chile
Escuela de Ingeniería
Departamento de Ciencia de la Computación

IIC2343 – Arquitectura de Computadores (II/2017)

Enunciado Tarea 02

Assembler y Disassembler, 30 % de la entrega 02

Fecha de entrega: 25 de Septiembre del 2017, 17:00 hrs.

1. Motivación

Al tener un computador programable, existe la necesidad de armar un lenguaje que sea más sencillo de comprender que instrucciones binarias. De esta necesidad surge el lenguaje Assembly. Sin embargo, dado que el lenguaje de la máquina sigue siendo binario, es razonable querer crear una herramienta que, dado un programa en Assembly, sea capáz de generar el mismo código pero en lenguaje del computador básico.

Al mismo tiempo, para comprender mejor este proceso, se presenta el desafío de crear un Disassembler. Un Disassembler es una herramienta que hace lo opuesto a un Assembler: a partir de un código en lenguaje de máquina (binario), una tabla de opcodes y la definición del orden opcode-literal puede generar código Assembly.

2. Descripción

2.1. Parte 1: Assembler (40 %)

El Assembler es una herramienta que deberán desarrollar a lo largo de todo el proyecto. Desde esta entrega en adelante, tendrán un avance sobre esto. Para obtener más detalles sobre el mismo deben revisar dentro del repositorio *syllabus* en la carpeta *Assembler*. **Para esta entrega deben poder soportar todas las instrucciones de la entrega 2 que aparecen en el enunciado del Assembler.**

2.2. Parte 2: Disassembler (40 %)

Un Disassembler es un programa de computador que traduce el lenguaje de máquina a lenguaje Assembly, la operación inversa de la que hace el Assembler. Para esta tarea tendrán que programar en Python un Disassembler que funcione del siguiente modo:

1. Debe recibir un archivo .csv con la tabla de opcodes. Una tabla de opcodes contiene el opcode binario asociado a cada operación de lenguaje de máquina. Por ejemplo, una de las líneas sería de la forma: “00010; MOV A,B”.

2. Debe recibir un archivo .txt con la secuencias de código binario a transformar a código Assembly.

3. Escribir un nuevo archivo con las instrucciones en código Assembly.

2.2.1. Codificación

Su programa debe tener en las primeras líneas del código una variable llamada "CSV_FILE' la cual almacenará el *path* de un archivo .csv que posee la codificación del opcode en binario y su traducción en código Assembly.

```
Ejemplo
CSV_FIle = "Escritorio\Arquitectura\Tareas\T02\codificador.csv"
```

Este archivo tendrá 2 líneas: La primera corresponde al opcode del comando y la segunda línea corresponde al nombre del código en lenguaje Assembly.
Supuestos:

- Los únicos registros existentes son A y B (ambos con mayúscula).
- Los comando serán los mismos utilizados por el Assembler de la parte 1.
- "dir" y "lit" siempre serán escritas en minúscula y hacen referencia a una dirección de la RAM y al literal que viene junto al opcode, respectivamente.
- No existirán dos líneas que posean el mismo opcode y un comando asociado distinto, ni vice-versa.
- Todos los opcode tendrán el mismo largo de bits.
- No existirán para esta tarea instrucciones de dos ciclos.
- los literales serán números enteros positivos representables en 16 bits.

Por ejemplo, un archivo csv se vería de la siguiente forma:

```
000001010; ADD A,B
000010010; CMP A,B
000100000; SUB A,lit
```

2.2.2. Código binario

Su programa debe tener otra variable llamada "TXT_FILE' la cual almacenará el *path* de un archivo .txt que posee el código binario a leer.

```
Ejemplo
TXT_FIle = "Escritorio\Arquitectura\Tareas\T02\ejemplo_1.txt"
```

Las líneas siempre tendrán el mismo formato: Los primeros 16 bits[1] son los literales y los siguientes N bits serán el opcode. El tamaño de N será definido por la cantidad de bits de un opcode del archivo .csv.
Supuestos:

- Pueden haber opcodes en el archivo txt que no estén en el csv. En ese caso **deben** dejar en esa línea un NOP[2].
- El código binario puede tener comentarios, ya sea en un una línea propia o después de la palabra binaria. Estos siempre parten con doble guión y son comentarios de una sola línea.

[1] De izquierda a derecha o en otras palabras, los 16 bits más significativos

[2] El comando NOP suele ser utilizado en Assembly para indicar que en dicha instrucción no se realiza nada. Dado que la ROM tiene un número fijo de instrucciones, el comando NOP suele usarse para rellenar las lineas no usadas de la ROM

- Los “0” y “1” del código binario vendrán todos juntos, es decir, no hay espacio entre un bit y otro.

Por ejemplo, un .txt basado en el csv mostrado antes se vería así:

```
000000000000011111000001010  -- 16 bit de literal y 9 bit de opcode.
000000000000000000000101010  -- Este es un comentario.
000000000000000001000101010  -- El literal de esta instruccion es 1.
000000000000000011000001010  -- Esta es la ultima linea
```

2.2.3. Resultados

Su programa debe ser capaz de leer el archivo csv y el txt para generar uno nuevo llamado “Resultados.txt”, en la misma carpeta de su código con la traducción de cada línea. El archivo debe poseer dos secciones. Una llamada “DATA:” donde están las variables declaradas y otra llamada “CODE:” donde estarán todas las acciones que debe ejecutar el computador básico. A continuación, un ejemplo de cómo se vería un archivo:

```
DATA:

CODE:
MOV A,B
CMP A,B
JMP 0
```

2.2.4. Un Disassembler no puede

- Rescatar los nombres de las variables. Por lo tanto, la sección DATA estará vacía.
- Recuperar los comentarios.
- Rescatar el nombre de las subrutinas o labels. Por lo tanto los saltos indicarán el número de la linea a saltar.

2.2.5. Ejemplo de funcionamiento

El siguiente ejemplo solo muestra cómo es el funcionamiento de un Disassembler.

```
Archivo.csv                    Archivo.txt
00000; MOV A,lit               00000000000111100000
00010; ADD A,lit               00000000000000000011
00011; MOV (dir),A             00000000000000111111
                               00000000000000100011
```

Si analizamos cada línea:

1. 00000000000111100000.
   Literal = 0000000000001111 = 15
   Opcode = 00000 = MOV A,lit
   Entonces esta línea es MOV A,15
2. 00000000000000000011.
   Literal = 0000000000000000 = 0
   Opcode = 00011 = MOV (dir),A
   Entonces esta línea es MOV (0),A

3. 00000000000000111111.
   Literal = 0000000000000001 = 1
   Opcode = 11111 = No está en el .csv
   Entonces esta línea es NOP

4. 00000000000000100011.
   Literal = 0000000000000001 = 1
   Opcode = 00011 = MOV dir,A
   Entonces esta línea es MOV (1),A

El archivo que resulta de ese .csv y .txt es:

```
DATA:

CODE:
MOV A,15
MOV (0),A
NOP
MOV (1),A
```

2.3. Informe (20 %)

Para este informe deberán incluir una descripción de cómo funciona su Assembler y su Disassembler. Deben explicar los pasos que sigue para llevar el archivo que recibe a binario (en el caso del Assembler) o a código Assembly (en el caso del Desassembler).

2.4. Bonus T01 (0.8 pts)

Por primera vez en el curso daremos un bonus que subirá la nota de la T01[3]. Si desean optar al bonus, deben indicarlo en su informe.

Como se mencionó antes, un Disassembler tiene la sección "DATA" vacía puesto que no puede rescatar variables. El bonus consistirá en lograr rescatar las variables pero con otro nombre.

Para contextualizar, cuando tenemos en "DATA" unas líneas de la forma.

```
DATA:
numero 2
otra_variable 10
```

Lo que hace al final el Assembler es traducir la sección "Data" en las siguientes líneas[4]:

```
MOV A,2
MOV (0),A
MOV A,10
MOV (1),A
```

Podemos apreciar que el Assembler seguirá un patrón:

[3] Sí, es una forma de pedir disculpas por las molestias causadas con lo poco claro del enunciado de la T01

[4] Puede usar el registro A o B pero solo usará 1 de ellos, no ambos al mismo tiempo

1. Guarda en un registro (A o B) el literal de la sección "Data". MOV A,lit o MOV B,lit

2. Guarda en la memoria RAM el valor que tenga el registro. Parte guardando en la posición 0 y va en aumento, en este caso: MOV (0),A y luego MOV (1),A.

El objetivo de este bonus es que su programa sea capaz de encontrar el patrón (con el registro A o B) y en vez de ponerlo en la sección "CODE", lo ponga en "DATA" con el nombre que ustedes quieran. Por ejemplo: "var_1", "var_2", etc.

Deben tener en cuenta que las variables también son usadas dentro de la sección "CODE" y ahí deben ser encontradas para tener el bonus. A continuación, se muestran 3 resultados de un Disassembler. El primero está sin bonus, el segundo con bonus incompleto mientras que el tercero es el completo. El tercer ejemplo es el esperado si es que implementan esta bonificación.

Archivo assembler original

```
DATA:
numero 2
otro_numero 10

CODE:
MOV A,(numero)
ADD A,1
MOV (otro_numero),A
```

Resultado del Disassembler

Sin bonus

```
DATA:

CODE:
MOV A,2
MOV (0),A
MOV A,10
MOV (1),A
MOV A,0
MOV A,(0)
ADD A,1
MOV (1),A
```

Bonus incompleto

```
DATA:
var_1 2
var_2 10

CODE:
MOV A,(0)
ADD A,1
MOV (1),A
```

Bonus completo

```
DATA:
var_1 2
var_2 10

CODE:
MOV A,(var_1)
ADD A,1
MOV (var_2),A
```

Para aclarar, el MOV A,0 de la quinta línea del resultado sin bonus lo pone el Assembler cuando lee el código original en Assembly y lo transforma a binario. Esto es porque el Assembler convierte cada línea a código binario, pero cuando pasa de la sección "DATA" a "CODE" debe asegurarse que todos los registros estén en 0. Si el Assembler usó el registro A para llevar los literales a la RAM, entonces debe dejarlo en 0 al final de la sección "DATA".

3. Contacto

- Francesca Lucchini: flucchini@uc.cl

- Felipe Pezoa: fipezoa@uc.cl
- Hernán Valdivieso: hfvaldivieso@uc.cl
- Luis Leiva: lileiva@uc.cl

4. Integridad académica

Los alumnos de la Escuela de Ingeniería de la Pontificia Universidad Católica de Chile deben mantener un comportamiento acorde a la Declaración de Principios de la Universidad. En particular, se espera que mantengan altos estándares de honestidad académica. Cualquier acto deshonesto o fraude académico está prohibido; los alumnos que incurran en este tipo de acciones se exponen a un Procedimiento Sumario. Es responsabilidad de cada alumno conocer y respetar el documento sobre Integridad Académica publicado por la Dirección de Docencia de la Escuela de Ingeniería.

Específicamente, para los cursos del Departamento de Ciencia de la Computación, rige obligatoriamente la siguiente política de integridad académica. Todo trabajo presentado por un alumno para los efectos de la evaluación de un curso debe ser hecho individualmente por el alumno, sin apoyo en material de terceros. Por "trabajo" se entiende en general las interrogaciones escritas, las tareas de programación u otras, los trabajos de laboratorio, los proyectos, el examen, entre otros. Si un alumno copia un trabajo, obtendrá nota final 1,1 en el curso y se solicitará a la Dirección de Docencia de la Escuela de Ingeniería que no le permita retirar el curso de la carga académica semestral. Por "copia" se entiende incluir en el trabajo presentado como propio partes hechas por otra persona.

Obviamente, está permitido usar material disponible públicamente, por ejemplo, libros o contenidos tomados de Internet, siempre y cuando se incluya la referencia correspondiente.

Lo anterior se entiende como complemento al Reglamento del Alumno de la Pontificia Universidad Católica de Chile. Por ello, es posible pedir a la Universidad la aplicación de sanciones adicionales especificadas en dicho reglamento.