UNIVERSIDAD DE LOS ANDES
FACULTAD DE INGENIERÍA
DEPARTAMENTO DE SISTEMAS Y COMPUTACIÓN
ARQUITECTURA DEL COMPUTADOR Y PROGRAMACIÓN DEL SISTEMA - ISIS2301
Proyecto 2006-1

Trabajo en grupos de 3 personas. No se permite ninguna consulta entre grupos.

El propósito de este proyecto es construir un emulador de una máquina virtual. Para esto sderá necesario usar archivos y diversas estructuras de datos. El trabajo debe ser elaborado en grupos de 3 personas, usando MASM. No se permite ningún tipo de consulta ente grupos.

A. DESCRIPCIÓN DEL EMULADOR

Un emulador es un programa que recibe como entrada un programa hecho para otra máquina. El emulador recorre el programa de entrada ejecutando las acciones que habría efectuado la otra máquina.

En este caso, haremos el emulador de una máquina virtual, es decir, una máquina que no existe realmente, que solo está definida "en papel".

El emulador debe funcionar de la siguiente manera: empieza pidiendo el nombre de un archivo; este archivo debe tener un programa en lenguaje de máquina de la máquina virtual. El emulador abre el archivo y carga el programa en la memoria simulada. Después, recorre el programa, como haría la máquina física, decodificando y ejecutando cada una de las instrucciones.

A continuación, se describirá la máquina virtual, el formato de los archivos de entrada y se dará una idea de cómo escribir un emulador.

1. La máquina virtual

La máquina virtual tiene las siguientes características:

- Palabra de 16 bits.
- Direcciones de 16 bits (el PC también es de 16 bits).
- Memoria direccionable al byte.
- Datos en formato *big endian*.
- 16 registros de 16 bits (numerados de 0 a 15).

El formato de instrucción es el siguiente: hay instrucciones de 16 y 32 bits. El primer byte siempre es el mismo: 6 bits para el código de operación y dos bits para el modo de direccionamiento (este se explica a continuación).

Las instrucciones tienen dos operandos: el primero siempre es un registro, el segundo tiene cuatro posibilidades según el valor de los bits de modo de direccionamiento:

Bits de modo de direccionamiento	Modo de direccionamiento
00	Registro
01	Indirecto por registro
10	Inmediato
11	Directo

Así, el formato para una instrucción cuyo segundo operando es un registro es el siguiente:

`cccccc00`	`ddddffff`

donde `cccccc` es el código de operación, 00 son los bits de modo de direccionamiento, `dddd` son 4 bits que designan el primer operando (un número entre 0 y 15 que designa un registro) y `ffff` 4 bits que designan el segundo operando.

Una instrucción con direccionamiento indirecto por registro se codifica de manera similar, solo que los bits de direccionamiento son 01. La diferencia es que, en el primer caso, el operando es `ffff`, mientras que, en el segundo, `ffff` apunta al operando.

En cuanto a las instrucciones con direccionamiento inmediato, la codificación se hace con 4 bytes de la siguiente manera:

`cccccc10`	`dddd0000`	`nnnnnnnn nnnnnnnn`

Como en el caso anterior, `cccccc` es el código de operación, 10 son los bits de modo de direccionamiento y `dddd` es el primer operando (un registro). En cuanto a `nnnnnnnn nnnnnnnn` son 16 bits con la constante que interviene en la operación. El segundo registro aparece como `0000` porque no se usa en este caso (se desperdician los bits).

Una instrucción con direccionamiento directo se codifica de manera similar, solo que los bits de direccionamiento son 11. La diferencia es que, en el primer caso, el operando `nnnnnnnn nnnnnnnn` es una constante, mientras que, en el segundo, es la dirección en memoria del operando.

Si la operación es binaria, se efectúa como en el Intel: Op1 ← Op1 operación Op2.

Si es unaria: Op1 ← operación Op2

Los saltos se manejan de manera especial.

En cuanto al código de operación, existen las siguientes posibilidades:

Operación	**Código operación**	**Descripción**
Mover	000000	Op1 ← Op2
Guardar en memoria	000001	Op2 ← Op1 [*ver nota 1*]
Sumar	000010	Op1 ← Op1 + Op2
Nand	000011	Op1 ← Not (Op1 & Op2)
Shift	000100	Op1 ← Op1 shift Op2 [*ver nota 2*]
Menor que	000101	Op1 ← Op1 < Op2 [*ver nota 3*]
Salto condicional	000110	Si Op1 entonces PC ← Op2 [*ver nota 4*]
Salto incondicional	000111	Op1 ← PC; PC ← Op2 [*ver nota 5*]
Detener	001000	Detiene la ejecución del programa

Nota 1: Esta operación sirve para almacenar en memoria, por eso el primer operando (el registro) se le asigna al segundo.

Nota 2: Op1 se corre el número de veces que indique Op2. Si Op1 > 0, se corre hacia la izquierda; Si Op1 < 0, se corre hacia la derecha.

Nota 3: si Op1 < Op2, a Op1 le asigna 1; si no, le asigna cero.

Nota 4: salta si el bit menos significativo de Op1 vale 1 (es decir, le asigna Op2 al PC); si no, continúa con la siguiente instrucción.

Nota 5: a Op1 le asigna el PC, pero el PC debe estar apuntando a la siguiente instrucción.

2. Formato del archivo de entrada

El archivo de entrada debe ser un archivo binario (**no es** un archivo de texto). Los dos primeros bytes son un número que indica en qué dirección de memoria (de la máquina virtual) se debe cargar el programa. Después viene una secuencia de bytes que se deben poner en memoria a partir de la dirección indicada. El PC debe quedar apuntando a la dirección donde se cargó el programa.

3. Escritura de un emulador

Un programa emulador debe tener un conjunto de variables y estructuras de datos que representan los elementos físicos del procesador. Por ejemplo, el PC se puede mantener en una variable de tipo WORD. Los registros podrían ser un vector de tipo WORD de 16 posiciones (cada posición representa un registro). La memoria podría ser un vector de bytes de 64K posiciones.

El programa emulador debe realizar sobre las entidades simuladas las mismas acciones que haría la máquina física: traer la instrucción apuntada por el PC e incrementarlo según el tamaño de la instrucción, decodificarla (determinar su código de operación, su modo de direccionamiento y sus operandos) y ejecutarla. Después debe continuar con la siguiente instrucción, y así hasta encontrarse con la instrucción "Detener". Por supuesto, las direcciones (en particular el PC) deben interpretarse como posiciones en el vector que simula a la memoria.

B. UTILIDADES PARA MANEJO DE ARCHIVOS

Para manejar los archivos, dispone de las siguientes utilidades (se incluyen "utilidades.inc"); se llaman con `invoke`:

openFile: Abre un archivo con el nombre dado como parámetro. Retorna en eax la manija del archivo abierto; si falla, retorna INVALID_HANDLE_VALUE (-1). Invocación:

```
invoke openFile,  lpFileName,      DesiredAccess, CreationDistribution
```

- lpFileName: apuntador al nombre del archivo (cadena de caracteres con convenciones C).
- DesiredAccess: GENERIC_WRITE, para abrir el archivo en escritura.
- CreationDistribution: CREATE_ALWAYS, para crearlo siempre (si ya existe, destruye el que hay).

Todos los parámetros son de tipo DWORD.

WriteFile: escribe una secuencia de bytes en el archivo indicado. Invocacion:

```
invoke WriteFile, hFile, pMemory, numberBytes, lpSizeReadWrite, NULL
```

- hFile: manija del archivo.
- pMemory: apuntador a la zona de memoria donde está lo que se quiere escribir.
- numberBytes: número de bytes que se desea escribir
- lpSizeReadWrite: apuntador a una variable donde deja el número de bytes escritos.

Todos los parámetros son de tipo DWORD.

CloseHandle: cierra el archivo indicado. Invocacion:

```
invoke CloseHandle, hFile
```

El parámetro es de tipo DWORD.

SetFilePointer: posiciona el apuntador al archivo (el sitio donde se hará la siguiente lectura o escritura). Invocación:

```
invoke SetFilePointer, hFile, lDistanceToMove, NULL, dwMoveMethod
```

- hFile: manija del archivo.
- `lDistanceToMove`: número de bytes que se desplazará el apuntador.
- `dwMoveMethod`: con respecto a qué se desplaza. el apuntador: FILE_BEGIN, FILE_CURRENT, FILE_END

Todos los parámetros son de tipo DWORD.

C. ENTREGABLES

Se realizarán dos entregas:

Primera entrega:

Debe entregar un programa con los siguientes procedimientos:

- `leerPrograma`: este procedimiento recibe como parámetro por la pila un apuntador al nombre del archivo (una cadena de caracteres con formato C) y un apuntador al vector donde se simula la memoria. Se encarga de abrir el archivo, y poner su contenido en el vector de memoria a partir de la posición indicada. Retorna en `eax` el tamaño del programa cargado.
- `escribirPrograma`: este procedimiento recibe como parámetros por la pila un apuntador al vector que simula a la memoria, la posición donde se cargó programa y el tamaño del mismo. El procedimiento recorre el programa instrucción por instrucción, y, para cada una de ellas, reporta en pantalla: de qué instrucción se trata, el modo de direccionamiento, la dirección del primer operando (el número del registro) y la dirección o el valor del segundo operando (según si es un registro o un operando en memoria, o una constante).[1]
- Programa principal: realiza las acciones iniciales del caso, pide el nombre del archivo de entrada, llama a `leerPrograma`, y después a `escribirPrograma`.

Segunda entrega:

Debe entregar un programa que, como en la entrega uno, cargue un programa de la máquina virtual en la memoria simulada, y, después, lo ejecute. Al terminar, el emulador debe escribir en pantalla el valor en que quedaron el PC y todos los registros de la máquina virtual.

Por supuesto, en las dos entregas, puede (incluso debe) desarrollar otros procedimientos que le ayuden a estructurar el problema. En estos, el paso de parámetros y demás convenciones los puede hacer a su gusto, pero debe ser consistente con el método que use y debe explicarlo en el documento adjunto a la entrega.

Cada entrega debe incluir: el programa fuente en ensamblador, el ejecutable y un documento explicando la forma de uso y el diseño del programa.

D. CRITERIOS DE CALIFICACIÓN

1. **Diseño (30%) (Funcionamiento, estructuras de datos y estructura del programa)**

Nota	Criterio
5	Solución bien pensada
3	Solución parcialmente planeada
1	Solución "pensada en el teclado"

2. **Ejecución (35%)**

Nota	Criterio
5	El programa corre correctamente
4	El programa corre bien la mayoría de veces
2	El programa corre mal la mayoría de veces
0	El programa no corre

[1] Note que este es un análisis estático; no implica ejecutar el programa sino recorrerlo para ver de qué instrucciones está compuesto.

3. **Estilo de codificación (20%)**

Nota	Criterio
5	Buen formato. Legible. Uso apropiado del lenguaje.
3	Código difícil de leer. Uso pobre del lenguaje.
1	Código incomprensible. No se usaron capacidades del lenguaje.

4. **Comentarios (15%)**

Nota	Criterio
5	Concisos, dicientes.
3	Parciales, pobres.
1	Verbosos, innecesarios, incorrectos.
0	No hay comentarios.

E. CONDICIONES DE ENTREGA

- El trabajo se realiza en grupos de máximo 3 personas, usando MASM y no debe haber consultas entre grupos.
- Se debe anotar, como comentario en el archivo fuente, los nombres y códigos de los integrantes del grupo.
- La entrega es por Sicua. Solo debe entregar uno de los integrantes del grupo.

Se realizarán dos entregas en las siguientes fechas:

- **Entrega 1**: por Sicua el miércoles 3 de mayo a las 12:00 del día. Vale un 10% de la nota del curso.
- **Entrega 2**: por Sicua el lunes 22 de mayo a las 12:00 del día. Vale un 20% de la nota del curso.

Recomendaciones:

- Sangrar y comentar el programa.
- Usar nombres dicientes para las variables.
- Los trabajos en grupo son en grupo; es decir, todo el grupo responde solidariamente por el contenido de todo el trabajo, y lo elabora conjuntamente (no es trabajo en grupo repartirse puntos o trabajos diferentes).
- Para los trabajos en grupo, se puede solicitar una sustentación a cualquier miembro del grupo sobre cualquier parte del trabajo. El resultado de la sustentación podrá afectar la nota de todos los miembros del grupo.
- Trabajos en grupo entregados individualmente (o en grupos diferentes al original) son una forma de fraude académico.
- No dejen el proyecto para el último momento. Ustedes no están acostumbrados a programar en ensamblador. Descubrirán que la programación en ensamblador tiene características especiales a las cuales no están habituados: la depuración toma bastante tiempo; los "errores tontos" son frecuentes, difíciles de detectar y se pagan caro; los errores más frecuentes no son los mismos que en alto nivel (así que no están acostumbrados a detectarlos).