Arquitectura de Computadoras
Proyecto 1: Máquina Virtual

Maximiliano Monterrubio Gutiérrez

21 de octubre de 2008

1. Objetivo del Proyecto

La finalidad de este proyecto es que implementen un simulador de una arquitectura de computadora hipotética especificada en este documento en lenguage `C`.

Para poder implementar este simulador es necesario que conozcan las diferentes partes que lo componen y posteriormente lean la especificación dada de la máquina para implementarla fidedignamente. En este caso, van a implementar un intérprete del código máquina de la máquina virtual especificada. Para facilitar la programación de dicha máquina se les proporcionará un compilador de un lenguaje ensamblador similar a `MIPS` al código máquina del simulador que deben implementar.

Como se mencionó en clase, se van a dejar 3 proyectos de desarrollo en `C`, de los cuales van a necesitar finalizar este para poder elaborar el siguiente proyecto. En caso de que no puedan finalizar este proyecto, pasando la fecha de entrega les daré una versión totalmente funcional de este proyecto para que puedan continuar con los siguientes y no se queden atrás.

1.1. Componentes del simulador

La implementación de su máquina virtual debe tener los siguientes componentes:

1. **Registros**. Su máquina virtual constará de 14 registros. Más adelante viene a detalle qué deberá hacer cada uno.

2. **Memoria Primaria**. Deberán simular de algún modo la memoria primaria de la máquina (lo que sería la *RAM* en sus equipos). Recuerden, la memoria es simplemente un arreglo de bytes que almacenan datos, por lo cual deberán tratarla como tal.

3. **Unidad Aritmético-Lógica y Unidad de Control**. El núcleo principal de su máquina. Decodificará y ejecutará las instrucciones dadas en la especificación de la máquina virtual.

4. **Llamadas al sistema**. Por ahora, la implementación de esta máquina virtual sólo dispondrá de 2 tipos de *syscalls*: imprimir y leer de consola, esto para poderse comunicar con el usuario.

5. **Manejo de errores**. En caso de que haya un fallo en la máquina (causado por el programador), en caso de ser un error fatal, detener la ejecución de manera controlada. Para reportar los errores deberán detener la ejecución de su máquina virtual y devolver el control al sistema operativo devolviendo un código de error y además almacenar un volcado de la memoria (como en las máquinas reales) en un archivo. Se penalizará a las implementaciones que salgan sin códigos de error incorrectos o que devuelvan errores del runtime de `C` (*segmentation fault*, *memory corruption*, *free() error*, etc).

2. Forma de calificar

Lo primero que deberán hacer para el proyecto es realizar una presentación en laboratorio de su código y darme una idea de cómo lo desarrollaron, para ello una semana antes de la fecha de entrega les pediré que por favor asistan para ver qué personas son, y cuándo van a venir a presentarlo.

La presentación debe durar no más de 10 minutos y quiero que expliquen de manera general cómo lo implementaron, con qué retos tuvieron que enfrentarse y que muestren un ejemplo de su simulador con un programa hecho por ustedes compilado para el mismo.

Para calificar el proyecto utilizaré un paquete de prueba automatizado que probará de su código alimentándolo con diferentes programas y configuraciones posibles. Su calificación será el porcentaje de casos de prueba satisfactorios contra el número total de casos de prueba. El paquete de prueba automatizado estará disponible para ustedes desde el momento que deje el proyecto y les servirá para probar su código.

De cualquier modo inspeccionaré su proyecto y posiblemente haga algunas pruebas manuales para buscar algún tipo de anomalía. Esto con la finalidad de evitar prácticas duplicadas o personas que no hayan hecho el proyecto (*i.e.* alguien más se las haya hecho). En caso de detectar dicha situación, se anulará en su totalidad la calificación del proyecto.

3. Especificación de la máquina virtual

Su máquina virtual constará de:

1. **16** *opcodes*. 4 para arimética entera, 4 operaciones de bits, 4 operaciones de memoria, 3 operaciones de salto de instrucción y la instrucción de llamada al sistema.

2. **14** registros. 8 registros para propósito general, dos registros de argumentos para llamada a sistema, un registro de retorno de datos de llamada a sistema, un registro de contador de programa, un registro de apuntador a la pila de memoria y un registro para apuntar a direcciones de regreso (para implementar funciones).

3. **8** *syscalls*. 4 instrucciones para leer de consola y 4 para escribir.

3.1. Registros

La máquina contará con 14 registros de 32 bits cuyo uso se especifica a continuación:

Registros	**Descripción**
`0`,...,`7`	Registros de propósito general
`8`,`9`	Argumentos para llamada al sistema
`10`	Retorno de datos de llamada al sistema
`11`	Dirección de retorno (para implementar funciones)
`12`	Contador de programa
`13`	Apuntador de pila de memoria

3.2. Códigos de operación (*opcodes*)

Su máquina virtual deberá implementar los siguientes códigos de operación:

Operación	Código (en hex)	Duración (ciclos)	Descripción
add	0x0	3	Suma entera (con signo).
sub	0x1	4	Diferencia entera (con signo).
mul	0x2	10	Producto entero (con signo).
div	0x3	11	Cociente entero (con signo). Será fatal la división entre cero.
fadd	0x4	4	Suma flotante.
fsub	0x5	5	Diferencia flotante.
fmul	0x6	9	Producto flotante.
fdiv	0x7	10	Cociente flotante. Será fatal la división entre cero.
and	0x8	1	Operador de bits `AND`.
or	0x9	1	Operador de bits `OR`.
xor	0xA	1	Operador de bits `XOR`.
not	0xB	1	Operador de bits `NOT`.
lb	0xC	500	Cargar byte.
lw	0xD	1500	Cargar palabra (4 bytes).
sb	0xE	700	Guardar byte
sw	0xF	2100	Guardar palabra
li	0x10	1500	Cargar valor constante
b	0x11	1	Salto incondicional
beqz	0x12	4	Salto si es igual a cero
bltz	0x13	5	Salto si es menor que cero
syscall	0x14	50	Llamada al sistema

3.3. Llamadas al sistema (*syscalls*)

Como se menciona en la introducción, su máquina virtual sólo proveerá llamadas al sistema para leer y escribir en consola. La forma de operar de las llamadas al sistema es igual que en `MIPS`: cargan un código de llamada al sistema en un registro, un argumento en uno o más registros y el sistema devolverá algún dato en un tercer registro de retorno. Las convenciones de entrada y salida de datos de las llamadas al sistema serán:

1. **Registro para código de llamada:** `8`.
2. **Registro para pasar un argumento a la llamada:** `9`.
3. **Registro donde se reciben datos de la llamada:** `10`.

Los códigos de llamada al sistema serán los siguientes:

- Códigos para leer de consola.

Código	**Significado**	**Retorno** (en registro 10)
0x0	Leer entero.	Entero leído.
0x1	Leer caracter.	Caracter leído.
0x2	Leer flotante.	Número leído.
0x3	Leer cadena.	Numero de caracteres leídos. Se debe especificar como en el registro 9 la dirección en memoria donde se guarda la cadena

- Códigos para escribir en consola.

Código	**Significado**	**Argumento** (en registro 9)
`0x4`	Escribir entero.	Entero a escribir
`0x5`	Escribir caracter.	Caracter a escribir
`0x6`	Escribir flotante.	Número a escribir.
`0x7`	Escribir cadena.	Dirección de memoria donde empezar a imprimir. La cadena se delimita por el caracter nulo `0` (al igual que en `C`).

- Salir del programa.

Código	**Significado**	**Argumento** (en registro 9)
`0x8`	Salir del programa	No utilizado

3.4. Codificación de instrucciones

Las instrucciones de la máquina virtual serán de 32 bits (4 bytes) y estarán codificadas de la siguiente manera:

`0`	`8`	`16`	`24`
OpCode	*dr*	*Op1*	*Op2*

Donde *OpCode* representa el código de operación, *dr* representa el número de registro donde guardar el resultado, y *Op1* y *Op2* representan los números de registro de los operandos.

En el caso de las instrucciones `BEQZ` y `BLTZ` la dirección de salto estará en *dr* y el registro a evaluar será *op2*.

Para las instrucciones que sólo tienen un operando (carga y almacenamiento en memoria, saltos de instrucción y el operador `NOT`) sólo se toma en cuenta como operando el valor almacenado en el campo *Op2*.

El caso de la operación `li` es especial, ya que es una instrucción que tiene un tamaño de 6 bytes y su decodificación va como sigue:

0	8	16	24	32	48
`0x10`	*rd*	Constante de 32 bits			

Esto complicará un poco la implementación de la máquina, sin embargo, no debe ser un obstáculo muy complicado de superar.

3.5. Pila de memoria

El *stack pointer* (registro 14) **siempre** empezará apuntando al último byte de la memoria primaria. Los programas siempre se cargarán en los primeros bytes de la memoria, y el apuntador apunta del fin hacia el inicio. Esto para evitar siempre que los datos en la pila sobrescriban el código del programa en ejecución. Por lo tanto, si el programador desea realizar una operación de *push* deberá decrementar el *stack pointer* y al realizar un *pop* incrementarlo.

3.6. Códigos de error

En caso de ocurrir un error fatal, su máquina virtual deberá finalizar su ejecución devolviendo un código de error que especifica la falla ocurrida. Adicionalmente, deberán guardar un volcado de la memoria primaria en un archivo para propósitos de depuración. Es muy importante que se apeguen a esta convención ya que el paquete de prueba revisará sus códigos de error al alimentar programas que producen errores en su simulador.

Si quieren devolver mensajes de error, lo deberán hacer por medio del flujo de datos de error (*stderr*) y no en *stdout* ya que eso invalidará la prueba de su programa. Deben devolver el control al sistema operativo devolviendo un *exit code* como aparece en la siguiente tabla:

Código de Error	Significado
1	División entre cero
2	Dirección de memoria inválida
3	Memoria agotada
4	Número de registro inválido
5	Código de operación inválido
6	Código de llamada a sistema inválido

4. Requerimientos del simulador

El simulador que implementen, además de poder ejecutar programas escritos en el lenguaje de máquina especificado en la sección anterior, deberá implementar las siguientes características:

4.1. Argumentos de línea de comando

La invocación de su máquina virtual deberá ser de la forma

```
$ ./myvm -m 65536 helloworld.bin
```

Donde el argumento `-m` representa el tamaño de la memoria principal (medida en bytes), y el archivo `helloworld.bin` representa un programa escrito en el lenguaje máquina que interpreta su simulador.

4.2. Ejecución

En caso de que la ejecución del programa sea satisfactoria, su simulador debe imprimir al final un número entero positivo que representa el tiempo de ejecución del programa medido en ciclos de reloj. Ejemplo:

```
$ ./myvm -m 1048576 helloworld.bin
Hello World!
6950
```

Para ello deberán llevar la cuenta en ciclos de reloj de cada instrucción que ejecuta el programa e imprimir la suma al final de la ejecución. La duración en ciclos de reloj de cada instrucción está especificada en la tabla de operaciones que deben implementar.

Si el programa produce un error fatal en su máquina virtual, la ejecución se deberá finalizar inmediatamente y deberán guardar un volcado de la memoria en el archivo `dumpfile.bin`. **No deben** imprimir el número de ciclos de reloj en este caso, deben salir inmediatamente.

5. Compilador

Para facilitar la programación de su simulador, se les proporcionará un compilador de lenguaje ensamblador que podrán usar para producir código máquina de la máquina virtual que deben implementar. La semántica del compilador es exactamente igual que los códigos de operación de su máquina virtual, y la sintaxis es muy similar a la del intérprete de `SPIM`.

5.1. Comentarios

Para poner comentarios en su programa, se utiliza el caracter `;`. Cualquier cosa que se encuentre después de este caracter se ignora.

5.2. Palabras reservadas

5.2.1. Instrucciones

Todas las instrucciones se delimitan por el caracter de nueva línea. Las siguientes palabras reservadas representan las instrucciones de la máquina virtual:

```
add  sub  mul  div
fadd fsub fmul fdiv
or   and  xor  not
lb   sb   lw   sw
beqz bltz b    li
syscall
```

5.2.2. Instrucciones adicionales

Para facilitar el trabajo al programador, el compilador ofrece 2 instrucciones adicionales:

1. `mov` (*move*). Para asignar el valor de un registro en otro. Funciona igual que en `SPIM`.

5.2.3. Registros

Los identificadores de registro, al igual que `SPIM` utilizarán el símbolo $. No se utilizará el número de registro como en el código máquina, sino un nombre más descriptivo que se muestra en la tabla a continuación:

Nombre	Número de Registro	Descripción
`$r0,...,$r7`	$0, \ldots, 7$	Registros de propósito general
`$a0, $a1`	$8, 9$	Registros para argumentos en llamada al sistema
`$s0`	10	Registro de retorno para llamada a sistema
`$ra`	11	Registro para almacenar direcciones de retorno
`$pc`	12	Contador de programa
`$sp`	13	Apuntador al tope de la pila de memoria

5.2.4. Definiciones

Macros para definiciones de datos.

```
.text
.asciiz .ascii
```

`.text` Delimita el inicio del código de su programa.

`.ascii` Sirve para representar arreglos de bytes, por lo tanto, si quieren reservar un arreglo de enteros, recuerden multiplicar el tamaño de arreglo que quieren por 4. Su sintaxis es:

```
.ascii ID INT
```

Donde ID representa un identificador del arreglo e INT una literal entera que representa el tamaño en bytes.

`.asciiz` Representan cadenas de caracteres. A las cadenas definidas con `.asciiz` se les agrega automáticamente el caracter nulo al final. Su sintaxis es:

```
.asciiz STR ID
```

Donde STR representa una literal de cadena e ID representa un identificador para la cadena. Las literales de la cadena son del estilo de `C`. Se soportan sólo las secuencias de control `\[bnft]` para `Backspace`, `Newline`, `Line Feed` y `Tab`. respectivamente.

El delimitador `.text` es obligatorio en cualquier programa para especificar en dónde inicia el código del mismo. Todos los demás macros son opcionales. Todas las definiciones de variables deberán hacerse *antes* del token `.text`.

5.2.5. Etiquetas

Para definir subrutinas se puede hacer uso de etiquetado al igual que en `SPIM` utilizando el caracter `:` para especificar una subrutina. Todo programa debe tener una etiqueta `main` la cual representa el punto de arranque del programa.

5.2.6. Funciones no soportadas del intérprete de `SPIM`

- No se soporta manejo de direcciones de memoria con la sintaxis `Offset($registro)` por lo cual, toda la aritmética de direcciones la deberán hacer manual.
- No se soporta el uso del *frame pointer*.
- No se soporta la instrucción *jump and link*. Deberán guardar la pista de la dirección de retorno en la pila de memoria de manera manual en caso de implementar funciones.
- No se permite usar identificadores para ninguna instrucción que no sea `li`. Por lo tanto, siempre que se quiera usar un valor de variable (o identificador) se *debe primero* usar `li` para cargar en un registro.

5.2.7. Programas de ejemplo

El famoso *Hello World*:

```
; Programa que imprime
; la cadena de texto "Hello World"

.asciiz "Hello World!" hwstr   ; La cadena a imprimir

.text                          ; Inicia el codigo del programa
main:   li $a0, 12             ; Guardamos en a0 el codigo de servicio de
                               ; impresion en consola.
        li $a1, hwstr          ; Pasamos la direccion de memoria de la cadena
                               ; a imprimir.
        syscall                ; Llamada al sistema.
        li $a0, 13             ; Salir del programa
        syscall
```

5.3. Instalación

Para poder instalar el simulador, deberán desempacar el *tarball* donde viene el código fuente y compilarlo ingresando en el subdirectorio `src` y tecleando lo siguiente:

```
$ make
```

Al final de este proceso deberán obtener un archivo ejecutable llamado `sasm`. La sintaxis del ensamblador es:

```
$ sasm <archivo fuente> <binario destino>
```

El compilador está totalmente contenido en dicho ejecutable y lo pueden mover a donde necesiten para poder compilar sus programas.

6. Paquete de pruebas

El paquete de pruebas estará contenido en el subdirectorio `test`. Para probar su proyecto, compilen su máquina virtual y llamen al ejecutable del simulador `svm`. Posteriormente colóquenlo en `test` y estando en dicho subdirectorio ejecuten:

```
$ ./test
```

7. Dudas y preguntas

Cualquier duda que tengan respecto al proyecto por favor no duden en preguntar en el laboratorio o en la lista de correo, entre más oportunos sean menos probabilidad habrá de que se lleven sorpresas o que no puedan implementar el proyecto.