Organización y arquitectura de computadoras

Profesor: Hugo Rodríguez Agosto 09, 2021
Ayudantes: Antonio Álvarez && Jorge García

Proyecto: Máquina Virtual

Objetivo del proyecto

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

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 se detalla qué deberá hacer cada uno.

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

3. Unidad Aritmética-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. Llamada al Sistema: La implementación de esta maquina virtual sólo dispondrá de 2 tipos de *syscalls*: imprimir y leer de consola, esto para comunicarse con el usuario.

5. Manejo de errores: En caso de que haya un fallo en la máquina (causado por el programador), en caso de 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 memoria en un archivo. Se penalizará a las implementaciones que salgan sin códigos de error correctos o que devuelvan errores del runtime de python.

Forma de Calificar

Tendrán que hacer una breve presentación de su proyecto de no más de 10 minutos, donde expliquen de manera general como lo implementaron, con que retos tuvieron que enfrentarse y que mostrar un ejemplo de su simulador con un programa hecho por ustedes compilado para el mismo.
Para calificar el proyecto se utilizará el paquete de pruebas incluidos en el proyecto, su calificación será el porcentaje de casos de prueba satisfactorios contra el número total de casos de prueba. De cualquier modo su proyecto será inspeccionado y posiblemente haga pruebas de forma manuales para buscar algún tipo de anomalía. Esto con la finalidad de evitar proyectos duplicados o personas que no hayan hecho el proyecto (i.e. que alguien mas se los haya hecho). En caso de detectar dicha situación, se anulará en su totalidad el proyecto.

Especificaciones de la máquina virtual

Su máquina virtual constará de:

1. **20 opcodes**: 4 de aritmética entera, 4 de aritmética de punto flotante, 4 para operaciones de bits, 4 para operaciones de memoria, 3 operaciones de salto de instrucción y una para la instrucción de llamada al sistema.

2. **14 registros**: 8 Registros de propósito general, 2 registros de argumentos para llamada al sistema, 1 registro de retorno de datos de llamada al sistema, 1 registro de contador de programa, 1 registro de apuntador a la pila de memoria y 1 registro para apuntar a la dirección de regreso(para la implementación de funciones).

3. **9 syscalls**: 4 Instrucciones para leer de la consola , 4 para escribir y 1 para terminar el programa.

Registros

La máquina constará de *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
12	Contador del programa
13	Apuntador a la pila de memoria

Códigos de operación (opcodes)

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)
fadd	0x4	4	Suma flotante
fsub	0x5	5	Diferencia flotante
fmul	0x6	9	Producto flotante
fdiv	0x7	10	Cociente flotante
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 (4 bytes)
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 a cero
syscall	0x14	50	Llamada al sistema

Llamadas al sistema (syscalls)

La forma de operar en las llamadas al sistema es igual que en **MIPS**: cargan un código de llamada del sistema en un registro, 1 argumento en otro registro y se devolverá algún dato en un tercer registro. Las convenciones de entrada y salida de datos en las llamadas al sistema serán:

1. Registro para código de llamada: **8**
2. Registro para pasar el 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 la consola:

Código	**Significado**	**Retorno (en el registro 10)**
0x0	Leer entero	Entero leído
0x1	Leer carácter	Carácter leído
0x2	Leer flotante	Número flotante leído
0x3	Leer Cadena	Número de caracteres leídos. Se debe especificar en el registro **9** la dirección de memoria donde se guardará la cadena.

- Códigos para escribir en consola:

Código	**Significado**	**Argumento (en el registro 9)**
0x4	Escribir entero	Entero a escribir
0x5	Escribir carácter	Carácter a escribir
0x6	Escribir flotante	Número flotante a escribir
0x7	Escribir cadena	Dirección de memoria donde empezará a imprimir. La cadena debe estar delimitada por el carácter nulo (al igual que C).

- Salir del programa:

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

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
Op Code	Dr	Op1	Op2

Donde *Op Code* representa al código de operación, *Dr* representa el número de registro donde se guardara el resultado, *Op1* y *Op2* representan los números de registro de los operandos. El caso de las instrucciones *BEQZ* y *BLTZ* la dirección de los saltos estarán en *Dr* y el registro a evaluar será *Op2*. Para las instrucciones que sólo tiene un operando (carga y almacenamientos de memoria, saltos de instrucciones 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 es la siguiente:

0	8	16	24	32	40
0x10	Dr	Cons de 32 bits			

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

Pila de memoria

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

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. 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 al sistema inválido

Requerimientos del simulador

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

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 interpretará su simulador.

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 1048578 helloworld.bin
Hello World!
6950
```

Para ello deberán de llevar la cuenta de los ciclos de reloj de cada instrucción que ejecuta el programa e imprimir la suma al final de la ejecución. La duración de 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 de 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.

Compilador

Para facilitar la programación de su simulador, se les proporciona 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**.

Comentarios

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

Palabras reservadas

Instrucciones

Todas las instrucciones se delimitan por el carácter 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**

Instrucciones adicionales

Para facilitar el trabajo del programador, el compilador ofrece 1 instrucción adicional:

- **mov (move)**: Para asignar el valor de un registro en otro.

Registros

Los identificadores de registro, al igual que en **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,...,7	Registros de propósito general
$a0,$a1	8,9	Registros de argumentos en llamada al sistema
$s0	10	Registro de retorno para llamada al sistema
$ra	11	Registro para almacenar direcciones de retorno
$pc	12	Contador del programa
$sp	13	Apuntador al tope de la pila de memoria

Definiciones

Los **macros** para la definición de datos son:

- **.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 su arreglo por **4**, la sintaxis es:
  .ascii ID INT
  Donde **ID** representa un identificador del arreglo e **INT** una literal entera que representa el tamaño en **bytes**.

- **.asciiz**
  Representa cadenas de caracteres. A las cadenas definidas con *.asciiz* se les agrega automáticamente el carácter nulo (\0) 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 $\backslash[bnft]$ para *Backspace, Newline, Line Feed y Tab.*

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**.

Etiquetas

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

Funciones no soportadas del intérprete SPIM

- No se soporta el manejo de direcciones a la memoria con la sintaxis *Offset* ($registro) por lo cual, toda la aritmética de direcciones la deberán de hacer manual.
- No se soporta el uso del *Frame pointer*.
- No se soportan las instrucciones **jump** y **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 función de instrucción que no sea **li**. Por lo tanto, siempre que se quiera un valor de variable (o identificador) se debe primero usar **li** par cargar en un registro.

Programas de ejemplo

En el comprimido incluido con este *PDF*, se encuentra una carpeta llamada *test*, donde todos los archivos finalizados con *.s* son programas que se pueden tomar como ejemplo.

Instalación

Para poder instalar el compilador, deberán desempacar el comprimido donde viene el código fuente y compilarlo ingresando en el sub-directorio **src** y ejecutando:

```
$ make
```

Es importante remarcar que deben tener instalado **flex** en su computadora. Para instalarlo ejecute el siguiente comando (o su equivalente dependiendo la distribución **Linux** que ocupe):

```
$ sudo apt-get install flex
```

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

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

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

Paquete de pruebas

En el sub-directorio *test* se encuentra los archivos de prueba, tanto los programas escritos en ensamblador con terminación *.s* y los que tiene el código máquina generado de dichos programas con terminación *.bin*.
Los programas de prueba llamados *err1, err2, err3, err6* como su nombre lo indican deben detenerse con el tipo de error correspondiente. Los programas de prueba **branch** (prueba los saltos), **flt** (prueba operaciones con punto flotante), **int** (prueba operaciones con enteros), **log** (prueba operaciones lógicas) y **mem** (prueba sw, lw, etc) tienen aparte un archivo con el nombre respectivo y terminación *.res*, dentro de este archivo esta lo que su simulador debe imprimir en la consola para que este sea correcto. Por último el programa de prueba **syscall**, pide al usuario los datos mismo que deben ser impresos inmediatamente después para que esta prueba sea correcta.
Para poder obtener **10 en el proyecto** su implementación debe poder realizar lo comentado anteriormente.

Dudas y preguntas

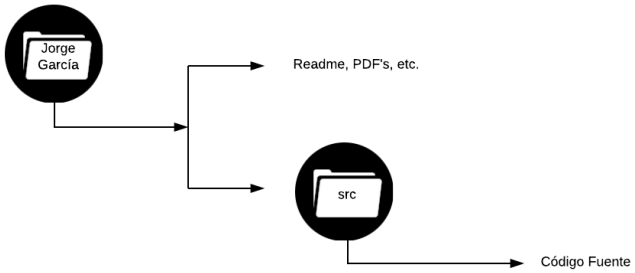
Para dudas, nos pueden contactar por medio del **discord** del grupo o por medio del correo electrónico.
Se recomienda que sea por **Discord** para que así haya una mejor interacción con todo el grupo para poder aclarar las dudas y que así también puedan compartir sus ideas para facilitar la implementación de su proyecto entre todos.

Entrega obligatoria

Como ya se había definido con anterioridad en los lineamientos y reglas del curso, es necesaria la entrega del proyecto, además de **obtener una calificación mayor o igual a 6** en éste, para así poder obtener una calificación aprobatoria en el curso. Es decir, si no se entrega un proyecto que tenga una calificación mínima de 6, **no se podrá aprobar el curso**.

Entregas

Para la entrega del proyecto deberán crear una carpeta con su nombre y apellido en el cual guardarán los archivos readme (especificaciones sobre su programa) y una sub-carpeta llamada src el cual tendrá todos los códigos fuente de su proyecto.
Deberán comprimir la carpeta en un .zip y enviarla a los correos:
hrodriguez@ciencias.unam.mx, *jorgel_garciaf@ciencias.unam.mx*
y *antonio_wata@ciencias.unam.mx* con asunto *Proyecto*.



La fecha de entrega para el proyecto es para el Viernes 3 de Septiembre, antes de las 23:59.
No se recibirá el proyecto pasada la fecha de entrega.
Si sus códigos no compilan, en automático tendrán 0 en su proyecto.
No se responderán dudas dos días antes de la entrega.
Si se descubre que alguien copio en el proyecto, todos los involucrados en automático tendrán cero en el proyecto, lo cuál los hará acreedores a *reprobar el curso*.