Trabajo Final: Arquitectura del Computador

Damian Ariel Marotte (M-3993/4)

6 de Diciembre de 2017

1. Descripción de la maquina

La arquitectura LCC_32 es una arquitectura de tamaño int (muy probablemente 32 bits). El código a ejecutar se guarda en la memoria de la maquina, lo que la convierte en una arquitectura de Von Neumann.

Posee una unidad principal, una unidad de depuración, un sistema de segmentación, una memoria de 1024 ints y 8 registros de tamaño int: cuatro de ellos de propósito general y los 4 restantes de uso especifico. Ellos son:

- Registros de propósito general
  - (3) *R0*.
  - (4) *R1*.
  - (5) *R2*.
  - (6) *R3*.
- Registros de uso especifico
  - (0) *ZERO*: Registro que siempre vale 0 y no puede modificarse.
  - (1) *PC*: Registro que guarda el numero de instrucción a ejecutar a continuación.
  - (2) *SP*: Registro que apunta al tope de la pila.
  - (7) *FLAGS*: Registro de banderas de estado.

Nótese que entre parentesis se indica el código de registro.

El registro de banderas posee las siguientes banderas, ordenadas desde el bit menos significativo en adelante:

- Segmentación: Indica si deben aplicarse los permisos de ejecución y escritura.
- Depuración: Indica si la unidad de depuración debe interrumpir el ciclo de ejecución.
- Cero: Indica si la ultima operación dio 0 como resultado.
- Igual: Indica si la ultima comparación resulto igual.
- Menor: Indica si la ultima comparación resulto menor.

2. Ciclo de ejecución

Previo a la ejecución de la maquina, el código a ejecutar es leído por un analizador sintáctico que se encargará de verificar la sintaxis y generar las instrucciones que la maquina puede ejecutar codificandolas según se detalla mas adelante.

Lo primero que ocurre cuando la maquina comienza a ejecutarse es la inicialización de sus registros y memoria en 0, con excepción de *SP* que apuntara fuera de la memoria (pila vacía) y el registro *FLAGS* que por defecto tiene la bandera de segmentación activada. Además, de haber argumentos por linea de comandos, estos se cargaran en registros/pila según corresponda de acuerdo a la convención de llamada.

A continuación se carga todo el código a ejecutar en la memoria de la maquina y se le entrega el control a la unidad principal que dará comienzo al ciclo de ejecución propiamente dicho:

- Si la bandera de depuración esta encendida se le da control a la unidad de depuramiento.
- Si la bandera de segmentación esta encendida se verifica que la instrucción indicada por *PC* pertenezca al segmento de código.
- Cuando todo lo anterior termina, se procede a ejecutar la instrucción correspondiente y luego se incrementa el registro *PC*.

La ejecución de la maquina termina cuando ocurre algún error, o se ejecuta la instrucción *HLT*.

3. Juego de instrucciones

La maquina tiene un juego de 23 instrucciones y 1 pseudo-instrucción que se detallan a continuación:

- (0) *NOP*: No realiza ninguna operación.
- (1) *MOV*: Copia un valor de/hacia registro/memoria/constante.
- (4) *SW*: Guarda el valor de *src* en la dirección de memoria *dst*.
- (3) *LW*: Carga en *dst* el valor de la memoria apuntada por *src*.
- (5) *PUSH*: Decrementa el registro *SP* y escribe el argumento *src* en el tope de la pila.
- (6) *POP*: Escribe en el argumento *dst* el tope de la pila e incrementa el registro *SP*.
- (7) *CALL*: Guarda en la pila el valor del registro *PC* y salta a la etiqueta indicada.
- (8) *RET*: Escribe en el registro *PC* lo que saca del tope de la pila.
- (9) *PRINT*: Imprime por pantalla el entero descripto en el argumento *src*.
- (10) *READ*: Lee un entero y lo guarda en el argumento destino.
- (11/12) *ADD*/SUB: Suma/resta los argumentos *src* y *dst* y lo guarda en *dst*.
- (13/14) *MUL*/DIV: Multiplica/divide los argumentos *src* y *dst* y lo guarda en *dst*.
- (2) *AND*: Efectua un AND bit a bit entre los argumentos *src* y *dst* y lo guarda en *dst*.
- (15) CMP: Compara *src* y *dst* y setea los bits correspondiente en el registro *FLAGS*.
- (16) JMP: Salta a la instrucción numero *dst* de la lista de instrucciones.

- (17) JMPE: Salta a la instrucción numero *dst* si la bandera de igualdad esta seteada.
- (18) JMPL: Salta a la instrucción numero *dst* si la bandera menor esta seteada.
- (21) DMP: Imprime por pantalla un volcado de registros y memoria.
- (22) DBG: Invierte el estado de la bandera de depuración.
- (23) SEG: Invierte el estado de la bandera de segmentación.
- (19) HLT: Termina el programa.
- LOOP: Es una pseudo instrucción equivalente a decrementar el registro *R0*, comparar 0 con el registro *R0* y saltar a la etiqueta indicada si la comparación fue menor.

Nótese que entre parentesis se indica el numero de opcode.

4. Convención de llamada

La convención de llamada para la arquitectura LCC_32 establece que todos los registros son caller saved. Además los primeros cuatro argumentos son pasados en los registros R0, R1, R2 y R3 y los restantes en la pila, dejando en el tope el quinto argumento.

5. Codificación de instrucciones

Las instrucciones se codifican en memoria como una secuencia de 10 enteros:

1. Opcode.
2. Sin uso.
3. Sin uso.
4. Sin uso.
5. Tipo del operando *src*.

6. Valor del operando *src*.
7. Sin uso.
8. Sin uso.
9. Tipo del operando *dst*.
10. Valor del operando *dst*.

donde los tipos de operandos son:

- (0) : Valor inmediato.
- (1) : Registro.
- (2) : Memoria.

De esta manera por ejemplo, la instrucción *ADD $5, %r0* se codifica como la secuencia de enteros 11 - ? - ? - ? - 0 - 5 - ? - ? - 1 - 3 y la instrucción 13 - ? - ? - ? - 0 - 2 - ? - ? - 1 - 4 corresponde a *MUL $2, %r1*.

6. Sistema de segmentación

Cuando la bandera de segmentación esta encendida, la maquina divide la memoria en dos segmentos: segmento de código y segmento de pila. De esta manera si se intenta ejecutar código en el segmento de pila o si se intenta escribir información en el segmento de código se producirá un error que interrumpirá la ejecución de la maquina.

7. Unidad de depuración

Cuando la bandera de depuración esta activada se interrumpe el flujo normal del programa y antes de cada instrucción se le da control a la correspondiente unidad. A partir de aqui podemos utilizar los siguientes comandos para controlar el comportamiento de la maquina:

- h: Muestra la lista de comandos disponibles.
- s: Devuelve el control a la unidad principal.
- n: Apaga la bandera de depuración, devuelve el control a la unidad principal y vuelve a interrumpir el flujo luego de ejecutarse una instrucción *RET*.
- j: Setea el registro *PC* en el valor indicado.
- b: Apaga la bandera de depuración, devuelve el control a la unidad principal y vuelve a interrumpir el flujo antes de ejecutarse la instrucción indicada.
- c: Modifica el valor de un registro o memoria.
- r: Muestra información sobre los registros.
- m: Muestra información sobre la memoria.
- l: Muestra el código del programa.
- w: Muestra la siguiente instrucción a ejecutar.
- x: Muestra el valor guardado en una dirección de memoria.
- q: Apaga la bandera de depuración y devuelve el control a la unidad principal.

Nótese que la unidad de depuramiento ignora las restricciones de segmentación.

8. Realización del trabajo

Para realizar el trabajo fue fundamental realizar una lectura del código provisto para poder comprender todos los detalles.

En primer lugar fue necesario leer el archivo de cabecera *<machine.h>* para familiarizarme con las diferentes estructuras que debi manejar en el programa.

Luego realice un analisis de cada función del archivo principal *<machine.c>* incluyendo comentarios donde considere conveniente.

Finalmente para terminar de comprender el comportamiento del programa, fue necesario conocer brevemente como funcionan las herramientas bison y flex que analizan el código de entrada de la máquina.

A la hora de implementar las instrucciones fue necesario utilizar funciones auxiliares que determinen valores y direcciones de memoria de acuerdo a sus operandos; decidir en que manera se iban a implementar los saltos; elegir que instrucciones modifican las banderas y decidir una convencion de llamada.

9. Extensiones

Estas son algunas de las posibles extensiones que considero para el trabajo:

- Tal como esta implementada esta máquina no existe diferencia entre el acceso a registros y el acceso a memoria. Sería interesante agregar una penalización al acceso a memoria y a partir de alli implementar un sistema de memoria cache.
- Agregar una unidad de manejo de cadenas de texto.
- Agregar una unidad de manejo de punto flotante.
- Permitir comentarios en la sintaxis del lenguaje.