Introduction

Ahora ya tenemos una idea de como usar ESCAPE. Al tener una cierta flexibilidad con la configuración de la ISA, en este práctico vamos a utilizar las diferentes dimensiones del ISA de la MIC I, para implementar parcialmente lo que es la IJVM presentada por Tanenbaum.

Repaso

Modelo de Memoria de la MIC I

El bus de direccionamiento de la MIC I es de 32 bits, y por ende tienen una capacidad máxima de direccionar una memoria de 2^{32} direcciones de celdas de un byte, o sea, 4,294,967,296 bytes. Esta memoria está dividida en 4 áreas:

- Constant Pool: Esta parte de la memoria que funciona como una pila, y no puede ser escrita por un programa. Es poblado cuando el programa es cargado a memoria y solo se lo puede referenciar. El registro CCP contiene un puntero al primer constante de esta pila.
- Local Variable Frame: Cuando se invoca una función (incluyendo el “main”), esta parte de la memoria es poblada con los variables locales del scope de la función. También funciona como una pila y las variables permanecen durante el ciclo de vida de la función, destruyéndose cuando la función retorna. Cada invocación anidada, genera un Local Variable Frame encima del previo. Lo primero que se encuentra en un frame es la dirección de retorno de esa función, seguido por los parámetros con que fue llamada la función, seguidos por los variables locales de la función. El registro LV contiene el puntero al comienzo de esta pila.
- Operand Stack: Inmediatamente arriba del Local Variable Frame más reciente se encuentra el Operand Stack. En términos técnicos, se lo podría considerar como parte del Local Variable Frame más reciente y es una pila utilizada por todas las instrucciones que se encuentran en la función siendo ejecutada. A diferencia de CCP, y LV, el registro SP apunta al tope de esta pila y se modifica según variables son empujados y popeados de esta pila. Hay otro registro relacionado llamado TOS, que contiene el valor que se encuentra en el tope de la pila (valor, no dirección).
- Method Area: Esta parte de la memoria contiene el código de tu programa. Es un área de solo lectura donde la primera instrucción es apuntada por el registro PC, y donde el mismo se modifica mientras se ejecuta el programa.

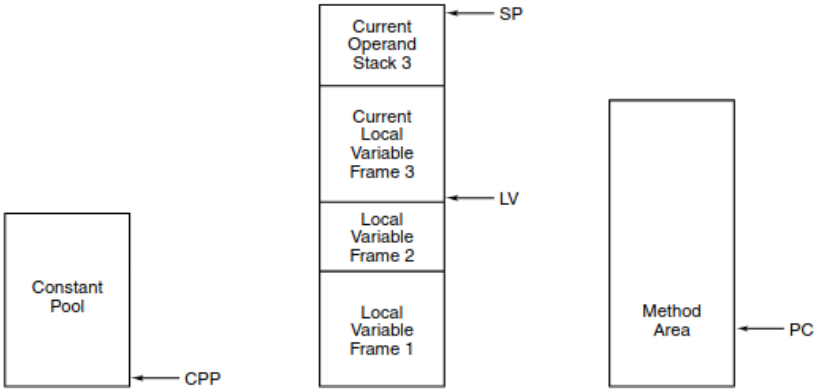

Figure 4-10. The various parts of the IJVM memory.

Figura que resume las diferentes áreas de memoria del modelo de memoria de la IJVM.

Tipos de datos

La IJVM solo trabaja con valores enteros.

Formato de Instrucción

La IJVM tiene instrucciones de tamaño variables que pueden variar entre una longitud de 1 byte a varios bytes. Por ejemplo, la instrucción BIPUSH tiene una longitud de de 2 bytes, como se ve en la siguiente figura, un byte para el código de operación (OPCODE), y otro para un operando. Como el registro MBR es de 1 bytes, esta instrucción tiene que ser leída en dos partes.

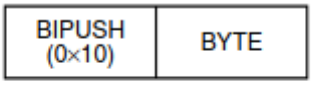

Figure 4-18. The BIPUSH instruction format.

El Set de Instrucciones de la IJVM

Hex	Mnemonic	Meaning
0x10	BIPUSH *byte*	Push byte onto stack
0x59	DUP	Copy top word on stack and push onto stack
0xA7	GOTO *offset*	Unconditional branch
0x60	IADD	Pop two words from stack; push their sum
0x7E	IAND	Pop two words from stack; push Boolean AND
0x99	IFEQ *offset*	Pop word from stack and branch if it is zero
0x9B	IFLT *offset*	Pop word from stack and branch if it is less than zero
0x9F	IF_ICMPEQ *offset*	Pop two words from stack; branch if equal
0x84	IINC *varnum const*	Add a constant to a local variable
0x15	ILOAD *varnum*	Push local variable onto stack
0xB6	INVOKEVIRTUAL *disp*	Invoke a method
0x80	IOR	Pop two words from stack; push Boolean OR
0xAC	IRETURN	Return from method with integer value
0x36	ISTORE *varnum*	Pop word from stack and store in local variable
0x64	ISUB	Pop two words from stack; push their difference
0x13	LDC_W *index*	Push constant from constant pool onto stack
0x00	NOP	Do nothing
0x57	POP	Delete word on top of stack
0x5F	SWAP	Swap the two top words on the stack
0xC4	WIDE	Prefix instruction; next instruction has a 16-bit index

Figure 4-11. The IJVM instruction set. The operands *byte*, *const*, and *varnum* are 1 byte. The operands *disp*, *index*, and *offset* are 2 bytes.

Figura lista todas las instrucciones de la IJVM que se pueden implementar en la MIC I

Ejercicio

1. Desafortunadamente, en ESCAPE no se pueden implementar instrucciones de longitud variable. Entonces vamos a implementar las siguientes de la siguiente forma:

Mnemónica	Descripción
NOP	Hacer nada
BIPUSH i	Empujar un valor al tope de la pila
POP	Borrar un valor del tope de la pila
DUP	Duplicar un valor en la pila
IADD	Hacer pop de los dos valores tope de la pila, sumarlos y empujar el resultado a la pila.
ISUB	Hacer pop de los dos valores tope de la pila, restarlos y empujar el resultado a la pila.
IAND	Hacer pop de los dos valores tope de la pila, hacer AND entre ellos y empujar el resultado a la pila.
IOR	Hacer pop de los dos valores tope de la pila, hacer OR entre ellos y empujar el resultado a la pila.
IFEQ j	Hacer pop del tope de la pila, ramificar si ese valor es 0.
IFLT j	Hacer pop del tope de la pila, ramificar si ese valor es menor que 0.
ILOAD i	Empujar un variable local al tope de la pila.
ISTORE i	Hacer pop del tope de la pila y guardar en un variable local.
GOTO j	Salto incondicional.

(Mientras implementen, tengan en cuenta que TOS retienen el valor del tope de la pila, vayan viendo cuáles instrucciones aquí afectaron el TOS.)

2. Configuren ESCAPE para implementar lo siguiente:
    a. Instrucciones de Kernel: Un programa escrito en IJVM supone un JVM (u otro sistema operativo) subyacente que prepara los registros CCP, SP, TOS, LV y PC antes de la ejecución del programa. Nosotros no tenemos esto entonces vamos a implementar unas extra instrucciones que simularán la preparación del sistema a nivel kernel. Estas son:

Instrucción	**Descripcion**
SETCCP	Establece el CCP
SETSP	Establece el SP
SETLV	Establece el LV
LV	Agrega un variable local

    b. El resto de la configuración puede quedar igual al práctico anterior.

3. Implementen el siguiente código:
    i. Establecer el CCP
    ii. Establecer el SP
    iii. Establecer el LV
    iv. Agregar tres variables locales de valor 10, 20 y 30.
    v. Saltar a la etiqueta “main” (que sea unas instrucciones más adelantes que la última en paso iv)
    vi. Empujar 5 a la pila.
    vii. Cargar variable local 1 a la pila.
    viii. Sumar los dos.
    ix. Guardar el resultado en variable local 2.
    x. Cargar variables locales 2 y 3 a la pila.
    xi. Hacer un IOR entre ellos.
    xii. Guardar el resultado en variable local 2.
    xiii. Cargar variable local 3 a la pila.
    xiv. Duplicar.
    xv. Hacer un IAND entre ellos.
    xvi. Guardar el resultado en variable local 1.
    xvii. Cargar variables locales 1 y 3 a la pila.
    xviii. Restar los dos.
    xix. Ramificar a “main” si el resultado es 0.

Bonus: Implementar la instrucción INVOKEVIRTUAL de la IJVM.