Trabajo Práctico Final - Pilot

Arquitectura del Computador

Gonzalo Turconi - T-2820/7
Ignacio Cattoni - C-6415/7
Redigonda Maximiliano - R-4079/7

Enero 8, 2018

Introducción

El objetivo de este proyecto es generar un compilador de lenguaje Pilot a lenguaje ensamblador ARM. **Pilot** es un lenguaje sencillo, del cual se provee una descripción precisa basada en las consignas del trabajo práctico.

Todas las sentencias en Pilot son de una línea. Cada línea comienza con una letra o símbolo que indica la operación a ejecutarse, o con el nombre de una variable para una operación de asignación.

Las variables son cadenas alfanuméricas que comienzan con una letra en minúscula, y son seguidas obligatoriamente por un entero de etiqueta[1]. De esta manera:

- a0, z24, x231, p2147483647, son nombres válidos, mientras que:
- a-1, b, romeosantos, 23z, w2147483648, r00, no son nombres válidos.

Las operaciones soportadas son las siguientes:

- Asignación: se caracteriza por comenzar con el nombre de una variable y seguir con una expresión. Su efecto es asignar el valor de la expresión en la variable.
- Entrada: comienza con un caracter R seguido por el nombre de una variable. Su efecto es leer un entero por la entrada estándar y asignárselo a la variable.
- Salida: comienza con un caracter O seguido de una expresión. Su efecto es imprimir el valor de la expresión por pantalla.
- Terminar: consiste de una única letra E, su efecto es el de terminar el programa.
- Salto: se representa por una G inicial, seguida de un entero de etiqueta. Su efecto es el de modificar el flujo de la ejecución del programa para resumirla en la etiqueta determinada por el entero.

[1] Un entero de etiqueta es un número entero no negativo representable por un entero con signo de 32 bits. Cualquier entero entre 0 y $2^{31} - 1$ inclusive, sin ceros a izquierda.

- Etiqueta: comienza con una L, y es seguida por un entero de etiqueta. Su efecto es definir la etiqueta caracterizada por el entero.
- Salto condicional: se representa por una letra I, seguida de una expresión, y termina en un entero de etiqueta. Su efecto es el de saltar a la etiqueta correspondiente si el valor de la expresión es distinto de cero.
- Comentario: se representa por un símbolo #, y tiene el efecto de ignorar todo lo que sigue en la línea.
- Definición de función: se representa por una F y sigue con el nombre de la función.
- Llamada a función: se representa por una C, seguida del nombre de una función, tiene el efecto de pasar el mando de la ejecución a la función.
- Retorno de función: se representa por la palabra RET seguida del nombre de la función de la cual se quiere retornar. Devuelve el mando de la ejecución al punto desde el cual se llamó a la función.

Un componente es una constante o una variable. Una expresión consiste de un componente, o un operador seguido de una o dos componentes (según corresponda).
No se permiten líneas en blanco, estas simbolizan el fin del programa.

Proceso

Para realizar este proyecto:

1. Se acordó una definición precisa del lenguaje Pilot.
2. Se realizó una investigación para estructurar y dividir el programa en varios archivos.
3. Se realizó una investigación para documentar todos los archivos del proyecto.
4. Se creó un sistema de lectura y discriminación de instrucciones (`instruction-decoder.c`).
5. Se diseñó una estructura de datos para mapear nombres de variables a direcciones de memoria (`storage.c`).
6. Se creó un sistema para leer y evaluar expresiones (`expressions.c`).
7. Se creó un sistema para imprimir por pantalla información útil durante el proceso de desarrollo (`debug.c`).
8. Se creó un sistema por medio del cual se imprime el programa en ARM resultante (`writer.c`).
9. Se desarrollaron 20 programas en Pilot con el fin de probar la correctitud del compilador (`tests/*`).

Nota

Desde un principio, el objetivo del proyecto ha sido desarrollar un compilador que dado un código válido en Pilot, genere un código válido en lenguaje ensamblador ARM que lo represente. Como consecuencia de esto, si se recibe un código no válido en Pilot, el comportamiento del compilador es indefinido.

Problemas

Durante el proceso de desarrollo, han surgido numerosos desafíos. Algunos de ellos se listan a continuación.

¿Cómo almacenar los valores de las variables?

Se investigó la posibilidad de guardar los valores de las variables en los registros, esto resultaría eficiente, pero si la cantidad de variables a manipular crece lo suficiente, esta opción se torna tediosa y caótica, ya que se requeriría de almacenamiento extra.

También se evaluó utilizar la pila, y asignarle a cada variable un offset desde el tope de la pila. Esta opción también tiene limitaciones cuando el número de variables es elevado, además de que es probable que no sean sólo variables los valores que se guarden en la pila.

Finalmente, se determinó que declarar un arreglo con la cantidad mínima requerida de memoria para guardar el valor de todas las variables era la mejor opción. Este método resulta fácil y cómodo de manejar, sólo asignamos a cada variable una posición en el arreglo, en el cual sabemos que sólo va a haber variables guardadas, y además resulta óptimo en el uso de la memoria.

Una vez que se tiene guardada una expresión, ¿cómo obtener su valor? ¿dónde guardarlo?

Debido a que una expresión posee a lo más 2 componentes, se realizó un evaluador de expresiones, que utiliza los registros `r2` y `r3` (esto es configurable) para almacenar valores auxiliares y guarda el resultado en el registro `r1`.

Se determinó guardar el valor en el registro `r1` ya que resulta conveniente para las llamadas posteriores a `printf`, ahorrando una instrucción `mov`.

¿Cómo soportar la operación de división entera? (/)

Uno de los inconvenientes se presentó al agregar el cociente a la lista de operadores. El conjunto de instrucciones de ARM contiene a `sdiv` y `udiv`, división con signo y sin signo, respectivamente.

Al ejecutar estas instrucciones se ha obtenido, en ambos casos, el siguiente error: `«Error: selected processor does not support ‘sdiv r1,r2,r3’ in ARM mode»` con la instrucción correspondiente.

Dichas instrucciones fueron las únicas con las que el compilador arrojó tal error. Luego de consultar foros informáticos y manuales de ARM en busca de alternativas, se pudo observar

que las operaciones de cociente han sido agregadas en ARMv8, y todos los emuladores de dicha arquitectura ejecutan, por defecto, ARMv6.

Esto se pudo solucionar agregando `-march=armv8-a` como opción en la compilación, dando por finalizado el problema.

¿Cómo seguir?

El compilador soporta todas las características adicionales sugeridas en las consignas del trabajo. También soporta comentarios, y posee una documentación realizada con `Doxygen`. Además de estas extensiones, el proyecto puede continuar por medio de:

- Agregar más tests

  Un compilador es un programa particularmente difícil de probar. Agregar más tests es una de las mejores opciones para dejar en evidencia fallas, y volver el compilador más robusto a cambios que puedan provocar problemas en el futuro.

- Agregarle un `for` a Pilot

  El código actualmente permite la integración de nuevas instrucciones con relativa facilidad. Debido a esto, implementar un `for` es una buena idea para continuar el proyecto.

  Una posible sintaxis puede ser `FOR var comp1 comp2` donde `var` es la variable que itera, mientras que `comp1` y `comp2` son componentes. Para terminarlo, se podría usar `END FOR`.

  Debido a que deberían poder anidarse, es necesario implementar una pila que recuerde la información de cada `for`, de manera que un `END FOR` cierre el último `for` que se abrió.

- Determinar si el programa ingresado en Pilot es válido.

  Se determinó que esta característica no es crucial para el compilador, debido a que no afecta a la correctitud de éste, sino a su facilidad de uso.

  Para realizarlo, se requiere de al menos:

  1. Guardar las etiquetas definidas, para verificar que no haya dos definiciones de la misma etiqueta.
  2. Hacer verificaciones de la cantidad de palabras de cada instruccion, dependiendo del tipo de instruccion.
  3. Evitar la declaración de una función dentro de otra, y verificar que el `RET` al final de una función corresponde a esa función.

- Optimizar la traducción

  Al inspeccionar un código `ARM` generado por el compilador, resulta evidente que el impacto de eficiencia debido al proceso de compilación es elevado.

Por medio de mantener el valor de los registros guardados en el compilador, se puede evitar que el compilador ingrese ciertas operaciones redundantes, haciendo que el programa generado sea más eficiente.

De todos modos, se debe ser cuidadoso, debido a los saltos condicionales, etiquetas y funciones que pueden hacer difícil mantener el valor exacto de los registros.

Resultados

Se han creado 20 programas en Pilot para probar la correctitud del compilador, con los que se han obtenido resultados satisfactorios.
Entre los programas en Pilot más destacados creados se encuentran:

1. `test7.txt`: Calcula el n-ésimo número en la sucesión de Fibonacci.
2. `test8.txt`: Invierte los dígitos de un número.
3. `test16.txt`: Recibe un rango $[a, b]$ e imprime todos los números primos en ese rango.
4. `test17.txt`: Multiplica dos números con el famoso algoritmo del campesino ruso.
5. `test19.txt`: Implementa un juego de frio-caliente.