TP Final Arquitectura - Pilot

Gonzalo Turconi, Ignacio Cattoni, Redigonda Maximiliano

Enero 8, 2018

Introducción

El objetivo de este projecto es generar un compilador para el lenguaje Pilot a lenguaje ensamblador ARM. **Pilot** es un lenguaje sencillo, del cual nosotros proveemos una descripción precisa basada en las consignas del trabajo práctico.

Todas las sentencias en Pilot son de una linea. Cada linea 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, 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 de 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 variable o una expresión. Su efecto es imprimir el valor de la variable o 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 interrumpir la ejecución del programa para resumirla en la etiqueta caracterizada por el entero.
- Etiqueta: comienza con una L, y es seguida por un entero de etiqueta. Su efecto es definir la etiqueta caracterizada por el entero. Sólo puede haber una definición de la etiqueta por cada 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.

- 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 o variable es distinto de cero.
- Comentario: se representa por un símbolo #, y tiene el efecto de ignorar todo lo que sigue en la linea.
- 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 y dos componentes.
No se permiten lineas en blanco, estas simbolizan el fin del programa.

Proceso

Para realizar este proyecto:

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

Nota

Desde un principio, el objetivo del proyecto fue 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

En el proceso de desarrollo del proyecto, nos hemos enfrentado a numerosos desafíos, los cuales se indican a continuación:

¿Cómo recordar los valores de las variables?

Pensamos primero en 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 un lugar de almacenamiento extra.

Por otro lado, pensamos en utilizar la pila, y asignárle 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 guardan en la pila.

Finalmente, decidimos declarar un arreglo con la cantidad mínima requerida de memoria para guardar el valor de todas nuestras variables. Esto 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 (a diferencia de con la pila), y además resulta óptimo en el uso de la memoria.

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

Una vez que creamos la funcionalidad para leer las expresiones, el siguiente desafío era el de obtener el valor de la expresión luego de evaluarla.

Debido a que una expresión tiene a lo más 2 componentes, hicimos un evaluador de expresiones, que utiliza los registros `r2` y `r3` (esto es configurable) y guarda el resultado en el registro `r1`.

Decidimos guardar el valor en el registro `r1` ya que resultaba conveniente para las llamadas posteriores a `printf`, ahorrándo 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.

¿Podría retornar de una función en la que no estoy?

Por cómo está implementado, lo siguiente es posible:

```
F funcion1
O 15
RET funcion2
```

Nuestro compilador entenderá estas instrucciones como válidas, pero el programa no tendrá sentido.

Esto es debido a que a pesar de que nuestro compilador entienda las instrucciones como válidas, el programa en Pilot no es válido, resultando en comportamiento indefinido.

¿Cómo seguir?

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

- Agregar más tests

  Un programa nunca está bien probado, y un compilador es un programa particularmente difícil de probar. Agregar más tests puede probablemente dejar en evidencia fallas, y hacer el compilador más robusto a cambios que puedan provocar problemas en el futuro.

- Agregarle un `for` a Pilot

  El código actualmente permite una 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 terminar el for, se podría usar `END FOR`.

  Debido a que los `fors` 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.

  Esto se omitió debido a que 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.

  Lo cual hubiera tomado mucho más tiempo.

- Optimizar la traducción

  Al inspeccionar un código `ARM` generado por nuestro compilador, resulta evidente que el impacto de eficiencia 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

Hemos creado 20 programas en Pilot para probar la correctitud del compilador, y hemos obtenido resultados satisfactorios.
Entre los programas en Pilot más destacados que creamos se encuentran:

1. `test7.txt`: Calcula n-ésimo número 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.