Escuela Politécnica Nacional
Compiladores y Lenguajes 2016-A
Definición de Proyecto – Cuarta Etapa: Generación de Código

Este trabajo consiste en el diseño e implementación de un compilador funcional para un lenguaje de programación basado en el Lenguaje C. En esta cuarta etapa del trabajo es necesario realizar la generación de código a partir del árbol sintáctico abstracto (AST).

1. Funcionalidades necesarias

Los resultados comprenden la *corrección de problemas encontrados en todas las etapas anteriores*, además de las siguientes funcionalidades:

a. Uso de registros: Para el código generado, es necesario que todas las operaciones se realicen sobre registros. Se debe implementar un módulo en el generador de código que maneje el uso y creación de registros. Los registros pueden ser vistos como variables auxiliares.

b. Uso de etiquetas: Un nuevo módulo se debe añadir al proyecto para para ofrecer etiquetas a ser utilizados en la generación de código. Las etiquetas son utilizadas para marcar los puntos de desvío en el flujo de la ejecución del programa.

c. Generación de Código: Se debe implementar una función que recorre recursivamente el árbol sintáctico. La generación de código consiste en la creación de una o más instrucciones en el lenguaje ensamblador descrito en el apartado 6 del presente documento.

La generación de código deberá traducir las siguientes construcciones:
- Declaración de variables (cálculo de dirección)
- Expresiones aritméticas
- Expresiones lógicas
- Comandos de Control de Flujo

Por cuestiones de practicidad, los programas en lenguaje de alto nivel solo implementarán una función (la función `main`).

2. Indicación Importante

En la traducción para el lenguaje ensamblador se debe considerar que el contenido de cada variable en el lenguaje de alto nivel está en una dirección de memoria. Antes de realizar cualquier operación sobre una variable, debe cargarse su contenido desde la memoria hacia un registro. Y de la misma forma, al final de cada operación el valor resultante deberá estar en un registro, de donde el valor debe ser transferido para la dirección de la variable en la memoria.

3. Recomendaciones

Como es usual, se recomienda que se ejecuten varios tests. Asimismo que se verifique la conformidad con cada una de las reglas de este documento y de la especificación de la etapa.

Verificar si el trabajo puede ser compilado y ejecutado en otro sistema aparte del utilizado para el desarrollo.

Todos los miembros del grupo deben haber realizado acciones de **commit** en el servidor **git**, por el hecho de ser un trabajo colaborativo.

4. Evaluación de las Etapas y del Proyecto

Cada etapa del proyecto será evaluada. Asimismo se evaluará la participación de los miembros del grupo por su conocimiento y capacidad para explicar el funcionamiento del proyecto.

5. Plazos de Entrega

La presentación de esta etapa del proyecto se realizará el día lunes 15 de agosto de 2016.

6. Lenguaje Ensamblador

En la siguiente tabla se detalla el lenguaje ensamblador. Los grupos podrán también especificar sentencias que consideren necesarias.

Este lenguaje ensamblador es un subconjunto del lenguaje descrito en el libro *Engineering a Compiler* de Keith D. Cooper y Linda Torczon.

Operación	**Significado**

`add r1, r2 => r3`
`sub r1, r2 => r3`
`mult r1, r2 => r3`
`div r1, r2 => r3`
	r3 = r1 + r2
	r3 = r1 − r2
	r3 = r1 ∗ r2
	r3 = r1/r2

`addI r1, c2 => r3`
`subI r1, c2 => r3`
`rsubI r1, c2 => r3`
`multI r1, c2 => r3`
`divI r1, c2 => r3`
`rdivI r1, c2 => r3`
	r3 = r1 + c2
	r3 = r1 − c2
	r3 = c2 − r1
	r3 = r1 ∗ c2
	r3 = r1/c2
	r3 = c2/r1

`lshift r1, r2 => r3`
`lshiftI r1, c2 => r3`
`rshift r1, r2 => r3`
`rshiftI r1, c2 => r3`
	r3 = r1 << r2
	r3 = r1 << c2
	r3 = r1 >> r2
	r3 = r1 >> c2

`and r1, r2 => r3`
`andI r1, c2 => r3`
`or r1, r2 => r3`
`orI r1, c2 => r3`
`xor r1, r2 => r3`
`xorI r1, c2 => r3`
	r3 = r1 ∧ r2
	r3 = r1 ∧ c2
	r3 = r1 ∨ r2
	r3 = r1 ∨ c2
	r3 = r1 xor r2
	r3 = r1 xor c2

`loadI c1 => r2`
`load r1 => r2`
`loadAI r1, c2 => r3`
`loadA0 r1, r2 => r3`
	r2 = c1
	r2 = Memoria(r1)
	r3 = Memoria(r1 + c2)
	r3 = Memoria(r1 + r2)

`store r1 => r2`
`storeAI r1 => r2, c3`
`storeA0 r1 => r2, r3`
	Memoria(r2) = r1
	Memoria(r2 + c3) = r1
	Memoria(r2 + r3) = r1