Diseño de Compiladores I
Trabajo Práctico Nº 4
Fecha de Entrega: 20/11/2023

Objetivo

Generar código Assembler, a partir del código intermedio generado en el Trabajo Práctico Nº 3.
El mecanismo para generar código será el de variables auxiliares.

El código Assembler generado debe poder ser ensamblado y ejecutado sin errores.

Los grupos podrán optar entre la generación de:

- Assembler para Pentium de 32 bits. El ensamblador a utilizar se puede descargar desde la sección Herramientas del Aula Virtual.
    - Para las operaciones entre datos de tipo entero se deberá generar código que utilice los registros del procesador (EAX, EBX, ECX Y EDX o AX, BX, CX y DX.
    - Para las operaciones entre datos de punto flotante se deberá utilizar el co-procesador 80X87.
- WebAssembly. https://webassembly.org/

Controles en Tiempo de Ejecución

Incorporar los chequeos en tiempo de ejecución que correspondan al tema particular asignado al grupo**.** Cada grupo deberá efectuar los chequeos indicados para situaciones de error que pueden producirse en tiempo de ejecución. El código generado por el compilador deberá, cada vez que se produzca la situación de error correspondiente, emitir un mensaje de error, y finalizar la ejecución.

a) División por cero para datos enteros y de punto flotante:
El código Assembler deberá chequear que el divisor sea diferente de cero antes de efectuar una división. Este chequeo deberá efectuarse para los dos tipos de datos asignados al grupo.

b) Overflow en sumas de enteros:
El código Assembler deberá controlar el resultado de la operación indicada, para los tipos de datos enteros asignados al grupo. Si el mismo excede el rango del tipo del resultado, deberá emitir un mensaje de error y terminar.

c) Overflow en sumas de datos de punto flotante
El código Assembler deberá controlar el resultado de la operación indicada, para el tipo de datos de punto flotante asignado al grupo. Si el mismo excede el rango del tipo del resultado, deberá emitir un mensaje de error y terminar.

d) Overflow en productos de enteros:
El código Assembler deberá controlar el resultado de la operación indicada, para los tipos de datos enteros asignados al grupo. Si el mismo excede el rango del tipo del resultado, deberá emitir un mensaje de error y terminar.

e) Overflow en productos de datos de punto flotante:
El código Assembler deberá controlar el resultado de la operación indicada, para el tipo de datos de punto flotante asignado al grupo. Si el mismo excede el rango del tipo del resultado, deberá emitir un mensaje de error y terminar.

f) Resultados negativos en restas de enteros sin signo:
El código Assembler deberá controlar el resultado de la operación indicada. Este control se aplicará a operaciones entre enteros sin signo. En caso que una resta entre datos de este tipo arroje un resultado negativo, deberá emitir un mensaje de error y terminar.

g) Recursión en invocaciones de funciones
El código Assembler deberá controlar que una función no pueda invocarse a sí misma.

GRUPO	CHEQUEOS
1	a b e
2	a c d
3	f b e
4	f c d
5	g b e
6	g c d
7	a b e
8	a c d
9	f b e
10	f c d
11	g b e
12	g c d
13	a b e
14	a c d
15	f b e

GRUPO	CHEQUEOS
16	f c d
17	g b e
18	g c d
19	a b e
20	a c d
21	f b e
22	f c d
23	g b e
24	g c d
25	a b e
26	a c d
27	f b e
28	f c d
30	g b e

Chequeo de Tipos y Conversiones

Tema 29: *Conversiones Explícitas*

Los grupos cuyos lenguajes incluyen conversiones explícitas, deberán traducir el código de las conversiones a código Assembler.

Tema 30: *Conversiones Implícitas*

Para los grupos que deben considerar conversiones implícitas, el código Assembler deberá efectuar dichas conversiones cuando los tipos de los datos involucrados en una expresión lo requieran, y siempre que el lenguaje lo permita.

- Los grupos que debieron generar las conversiones en la representación intermedia (Tercetos y Árbol Sintáctico), deberán traducir el código de las conversiones a código Assembler.
- Los grupos que deben considerar el chequeo de tipos y/o las conversiones en esta etapa (Polaca Inversa), deberán agregar código para efectuar las conversiones cuando las mismas sean requeridas, siempre que el lenguaje lo permita. Si hubiera incompatibilidad de tipos, esto deberá ser informado por el compilador como un error.

Tema 31: Sin Conversiones

El compilador debe prohibir cualquier operación (expresión, asignación, comparación, etc.) entre operandos de tipos diferentes, informando en cada caso, cuál es la combinación de tipos que está provocando la incompatibilidad.

- Los grupos que debieron efectuar el chequeo de tipos durante la etapa 3 (Tercetos y Árbol Sintáctico), ya tienen asegurado que las operaciones a generar se efectuarán con operando compatibles.
- Los grupos que deben considerar el chequeo de tipos en esta etapa (Polaca Inversa), deberán chequear que los operandos sean de tipos compatibles durante la traducción de Polaca Inversa a Assembler, informando el error de incompatibilidad de tipos cuando esto no ocurra.

Salidas del compilador

1) Los errores generados en tiempo de compilación (Errores Léxicos, Sintácticos y Semánticos) se deberán mostrar todos juntos, indicando la descripción de cada error y la línea en la que fue detectado.
2) Representación intermedia, según indicaciones del Trabajo Práctico 3.
3) Contenidos de la Tabla de Símbolos.
4) Archivo conteniendo el código Assembler.

Nota: No se deberán mostrar los tokens ni las estructuras sintácticas que se pidieron como salida de los Trabajos Prácticos 1 y 2, a menos que exista una reentrega pendiente que así lo requiera.

Forma de entrega

Se deberá entregar:

- **Código fuente completo y ejecutable, incluyendo librerías del lenguaje y todo otro componente que fuera necesario para la ejecución.**

- **Casos de Prueba:**
  - Casos válidos para cada estructura sintáctica solicitada
  - Casos semánticamente válidos
  - Casos con errores por Tipos incompatibles, y/o que requieran conversiones para los temas 29, 30 y 31
  - Casos con y sin errores semánticos por redeclaración o no declaración de variables / funciones / métodos, atributos, etc.
  - Casos que violen las restricciones de alcance del lenguaje
  - Casos de prueba para cada tema particular asignado:
    - Operadores especiales (temas 9 al 12)
    - Sentencias de control (temas 13 a 16) Incluir en las pruebas, el anidamiento de este tipo de sentencias, así como con sentencias de Selección.
    - Objetos (temas 17 a 24)
    - Comprobaciones de uso de variables (temas 25 al 28)
  - Casos que generen errores en tiempo de ejecución (para cada control asignado al grupo)

- **Informe conteniendo:**
  - Introducción
  - Descripción de la Generación de Código Intermedio, indicando:
    - estructura utilizada para el almacenamiento del código intermedio,
    - uso de notación posicional de Yacc ($$, $n), indicando en qué casos se usó, y con qué fin,
    - descripción tipo pseudocódigo del algoritmo usado para la generación de las bifurcaciones en sentencias de control,
    - nuevos errores considerados,
    - y todo otro aspecto que se considere relevante.
  - Descripción del proceso de Generación de Código Assembler, indicando:
    - Mecanismo utilizado para la generación del código Assembler
    - Mecanismo utilizado para efectuar cada una de las operaciones aritméticas
    - Mecanismo utilizado para la generación de las etiquetas destino de las bifurcaciones.
    - Todo otro aspecto que se considere relevante
  - Modificaciones a las etapas anteriores, si hubieran existido.
  - Para los **temas 17 al 28**, incluir una descripción del modo en que fue resuelto el tema correspondiente en cada etapa de desarrollo del compilador.
  - Conclusiones