Tecnológico de Monterrey Campus Santa Fe

Francisco Huelsz Prince

A01019512

Diseño de compiladores

Documentación del Proyecto 4

Índice

Generación de código

Para este proyecto, de acuerdo con las recomendaciones, se estableció como código objetivo el ensamblador **MIPS32**. Este trae consigo varias ventajas: amplia documentación, versatilidad de la arquitectura, facilidad de emulación, entre otras. El set de instrucciones de MIPS32 provee todas las instrucciones necesarias para compilar un lenguaje como C- sin ser muy complicado. Funciones como manejo de memoria dinámica y estática, entrada y salida, manejo de números enteros y flotantes y un micro-kernel hacen el desarrollo en la plataforma muy accesible y, por consiguiente, un objetivo ideal para compilar.

Manual de usuario

Requerimientos:

- Python >=3.6
- QtSpim

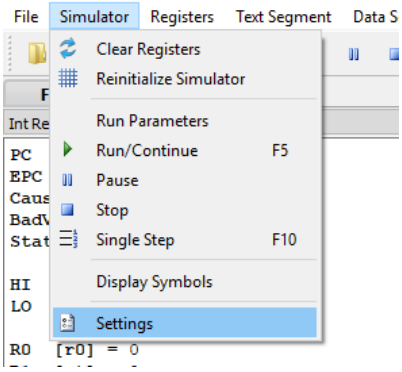
Manual del script

Archivo principal: `compile.py`

```
usage: compile.py [-h] [-o output] [--AST] [--ST] source

C- to MIPS32 compiler.

positional arguments:
  source      Source C- file.

optional arguments:
  -h, --help  show this help message and exit
  -o output   Destination asm file.
  --AST       Show AST graphically. (Requires graphviz)
  --ST        Print symbol tables.
```

Configuración de QtSpim

Para asegurarse que el programa corra sin problemas en el emulador es necesario verificar la configuración de este:



Al seleccionar esta opción se abre una ventana nueva con la configuración. En la pestaña “MIPS” se encuentra la configuración del emulador.

Se recomienda la siguiente configuración:

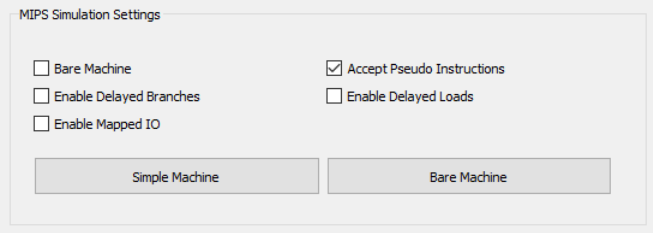


Fig. 1 Configuración recomendada

El compilador, además, toma en cuenta algunas características del hardware real, estas son las condiciones conocidas como *delay slots*. Por esto, el compilador produce código compatible con las opciones *Enable Delayed Branches* y *Enable Delayed Loads*. El compilador, sin embargo, depende del uso de pseudo instrucciones, por lo que es esencial que esta opción permanezca activa.

Ejemplo de uso

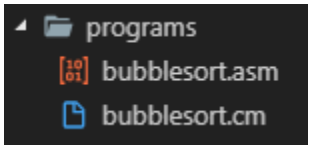
El comando básico para compilar es el siguiente, en este ejemplo se compila el archivo bubblesort.py:

```
PS D:\...\compiler> py compile.py ..\programs\bubblesort.cm
Compilation done!
```

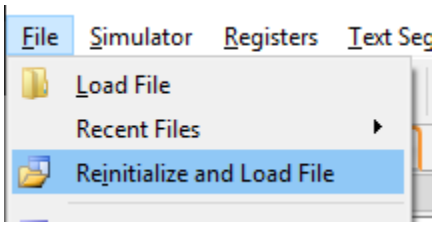
```
bubbleSort:
    addiu $sp,$sp,-32         # Start of bubbleSort. Grow stack by: 32 (8 words)
    sw $ra,-4($fp)            # Save $ra to offset -4
    li $t0,0                  # Load literal 0 to $t0
    sw $t0,-20($fp)           # Save $t0 to: j
    li $t0,0                  # Load literal 0 to $t0
    sw $t0,-16($fp)           # Save $t0 to: i
while_0:
                              # START OF COMPARISON
                              # START OF SIGM
    li $t0,1                  # Load literal 1 to $t0
    addiu $sp,$sp,-4          # Grow stack for SIGM
    sw $t0,4($sp)             # Save ADD.Right to stack
    lw $t0,-12($fp)           # Load value of: n
    sll $0,$0,0               # NOP: Wait for load delay slot
    lw $t1,4($sp)             # Load ADD.Right to $t1
    addiu $sp,$sp,4           # Shrink stack after SIGM
    subu $t0,$t0,$t1          # $t0 = $t0 - $t1
    addiu $sp,$sp,-4          # Grow stack for COMP
    sw $t0,4($sp)             # Save COMP.Right to stack
    lw $t0,-16($fp)           # Load value of: i
    sll $0,$0,0               # NOP: Wait for load delay slot
    lw $t1,4($sp)             # Load COMP.Right to $t1
    addiu $sp,$sp,4           # Shrink stack after COMP
    slt $t0,$t0,$t1           # $t0 = $t0 LT $t1
    beq $t0,$0,endwhile_0     # Evaluate while condition
    sll $0,$0,0               # NOP: Wait for delay slot
while_1:
                              # START OF COMPARISON
                              # START OF SIGM
    li $t0,1                  # Load literal 1 to $t0
    addiu $sp,$sp,-4          # Grow stack for SIGM
    sw $t0,4($sp)             # Save ADD.Right to stack
                              # START OF SIGM
    lw $t0,-16($fp)           # Load value of: i
    sll $0,$0,0               # NOP: Wait for load delay slot
```

Fig. 2 Sección del Código generado

Al finalizar, el compilador produce por defecto un archivo con el mismo nombre y en la misma dirección del archivo de entrada, solamente cambiando la extensión a `.asm`



Una vez compilado, se debe utilizar QtSpim para ejecutar el código ensamblador. Para cargar el código en QtSpim se debe cargar el archivo con la opción indicada abajo:

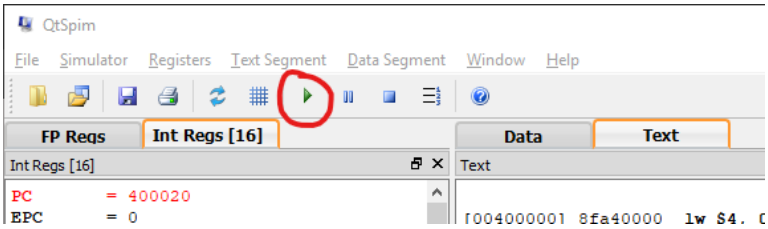


Esta opción abre una ventana del explorador de archivos para seleccionar qué archivo cargar al programa. Una vez seleccionado, QtSpim carga el código ensamblador a la pestaña *Text*.

```
                                    User Text Segment [00400000]..[00440000]
[00400000] 8fa40000  lw $4, 0($29)           ; 183: lw $a0 0($sp) # argc
[00400004] 27a50004  addiu $5, $29, 4        ; 184: addiu $a1 $sp 4 # argv
[00400008] 24a60004  addiu $6, $5, 4         ; 185: addiu $a2 $a1 4 # envp
[0040000c] 00041080  sll $2, $4, 2           ; 186: sll $v0 $a0 2
[00400010] 00c23021  addu $6, $6, $2         ; 187: addu $a2 $a2 $v0
[00400014] 0c1000b3  jal 0x004002cc [main]   ; 188: jal main
[00400018] 00000000  nop                     ; 189: nop
[0040001c] 3402000a  ori $2, $0, 10          ; 191: li $v0 10
[00400020] 0000000c  syscall                 ; 192: syscall # syscall 10 (exit)
[00400024] 34020005  ori $2, $0, 5           ; 5: li $v0,5 # Start of INPUT. Save 5 to $v0 for read int
[00400028] 0000000c  syscall                 ; 6: syscall # Read int syscall
[0040002c] 00404025  or $8, $2, $0           ; 7: or $t0,$v0,$0 # Move input to $t0
[00400030] 03e00008  jr $31                  ; 8: jr $ra # Jump back
[00400034] 00000000  nop                     ; 9: sll $0,$0,0 # NOP
[00400038] 8fc4fff8  lw $4, -8($30)          ; 11: lw $a0,-8($fp) # Start of OUTPUT. Load param from stack to $a
[0040003c] 34020001  ori $2, $0, 1           ; 12: li $v0,1 # Load 1 to $v0 for print int
[00400040] 0000000c  syscall                 ; 13: syscall # Print int syscall
[00400044] 3402000b  ori $2, $0, 11          ; 14: li $v0,11 # Load 11 to $v0 for print char
[00400048] 3404000a  ori $4, $0, 10          ; 15: li $a0,10 # Load a \n to $a0
[0040004c] 0000000c  syscall                 ; 16: syscall # Print newline syscall
[00400050] 03e00008  jr $31                  ; 17: jr $ra # Jump back
[00400054] 00000000  nop                     ; 18: sll $0,$0,0 # NOP
[00400058] 27bdffe0  addiu $29, $29, -32     ; 20: addiu $sp,$sp,-32 # Start of bubbleSort. Grow stack by: 32 (
[0040005c] afdfffc  sw $31, -4($30)          ; 21: sw $ra,-4($fp) # Save $ra to offset -4
[00400060] 34080000  ori $8, $0, 0           ; 22: li $t0,0 # Load literal 0 to $t0
[00400064] afc8ffec  sw $8, -20($30)         ; 23: sw $t0,-20($fp) # Save $t0 to: j
[00400068] 34080000  ori $8, $0, 0           ; 24: li $t0,0 # Load literal 0 to $t0
[0040006c] afc8fff0  sw $8, -16($30)         ; 25: sw $t0,-16($fp) # Save $t0 to: i
[00400070] 34080001  ori $8, $0, 1           ; 29: li $t0,1 # Load literal 1 to $t0
[00400074] 27bdfffc  addiu $29, $29, -4      ; 30: addiu $sp,$sp,-4 # Grow stack for SIGM
[00400078] afa80004  sw $8, 4($29)           ; 31: sw $t0,4($sp) # Save ADD.Right to stack
[0040007c] 8fc8fff4  lw $8, -12($30)         ; 32: lw $t0,-12($fp) # Load value of: n
[00400080] 00000000  nop                     ; 33: sll $0,$0,0 # NOP: Wait for load delay slot
```

Fig. 3 Código cargado en QtSpim

Para ejecutar el código solamente es necesario presionar el botón de inicio y el programa comenzará su ejecución:



En este ejemplo, el programa siendo compilado pide al usuario 5 números, realiza bubble sort sobre ellos e imprime los números en orden ascendente.

Al ejecutar el programa de ejemplo, la consola de QtSpim se verá similar a esto, notando que los números son entradas arbitrarias para ejecutar el ejemplo:

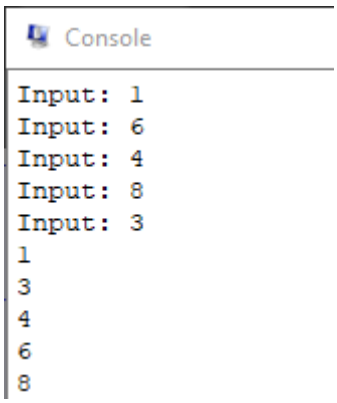


Fig. 4 Salida complete de bubblesort.cm

Opciones adicionales

Como se indica en el manual de uso del script `compile.py`, existen varios argumentos opcionales. Estos son:

- `--AST` crea una visualización gráfica del AST generado.
- `--ST` Imprime las tablas de símbolos generadas.
- `-o file` Define el archivo de salida. Debe ser un archivo con extensión .asm, no es necesario que el archivo o la ruta existan, estos se crean si es necesario.

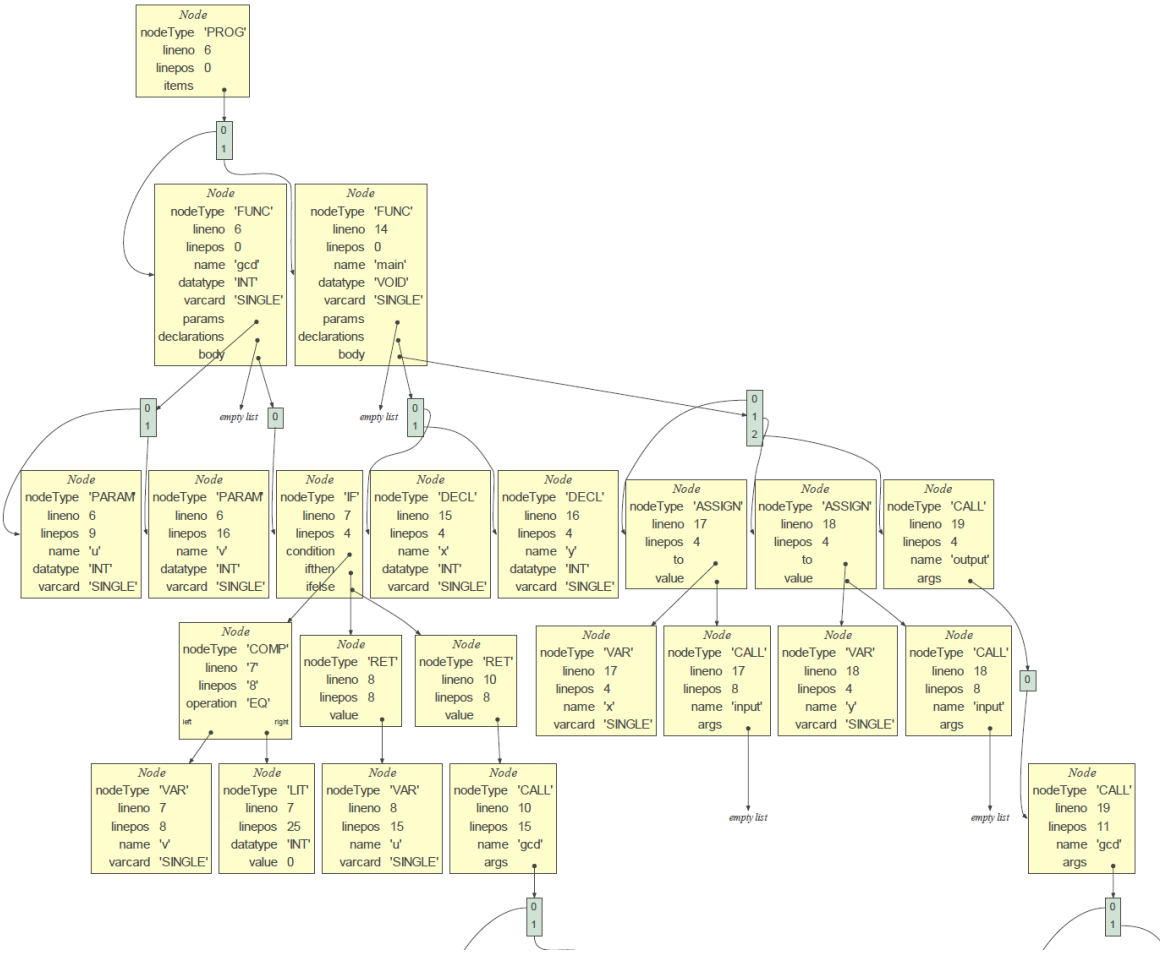


Fig. 5 Ejemplo de gráfica del AST generado para el programa de muestra que calcula el MCD de dos números.

Uso como una librería

De acuerdo con las especificaciones del proyecto, el programa puede ser utilizado como una librería. Para generar código es necesario generar el AST con el análisis semántico, luego, generar las anotaciones de símbolos necesarias con el análisis semántico. Finalmente, usando el árbol anotado se genera el código MIPS. A continuación, se presenta un ejemplo de cómo hacer esto:

```
from Parser import globales, parser
from semantica import semantica
from cgen import codeGen

programa = # string que contiene el programa


programa = programa + '$'

progLong = len(programa)
posicion = 0
globales(programa, posicion, progLong)


AST = parser(<ast_print:bool>, <ast_graph:bool>)
semantica(AST, <ST_print:bool>)
codeGen(AST, "output.asm")
```

Este Código toma como entrada un *string* que contenga el programa, imprime detalles de acuerdo con las banderas especificadas y genera el archivo `output.asm` con el programa compilado.

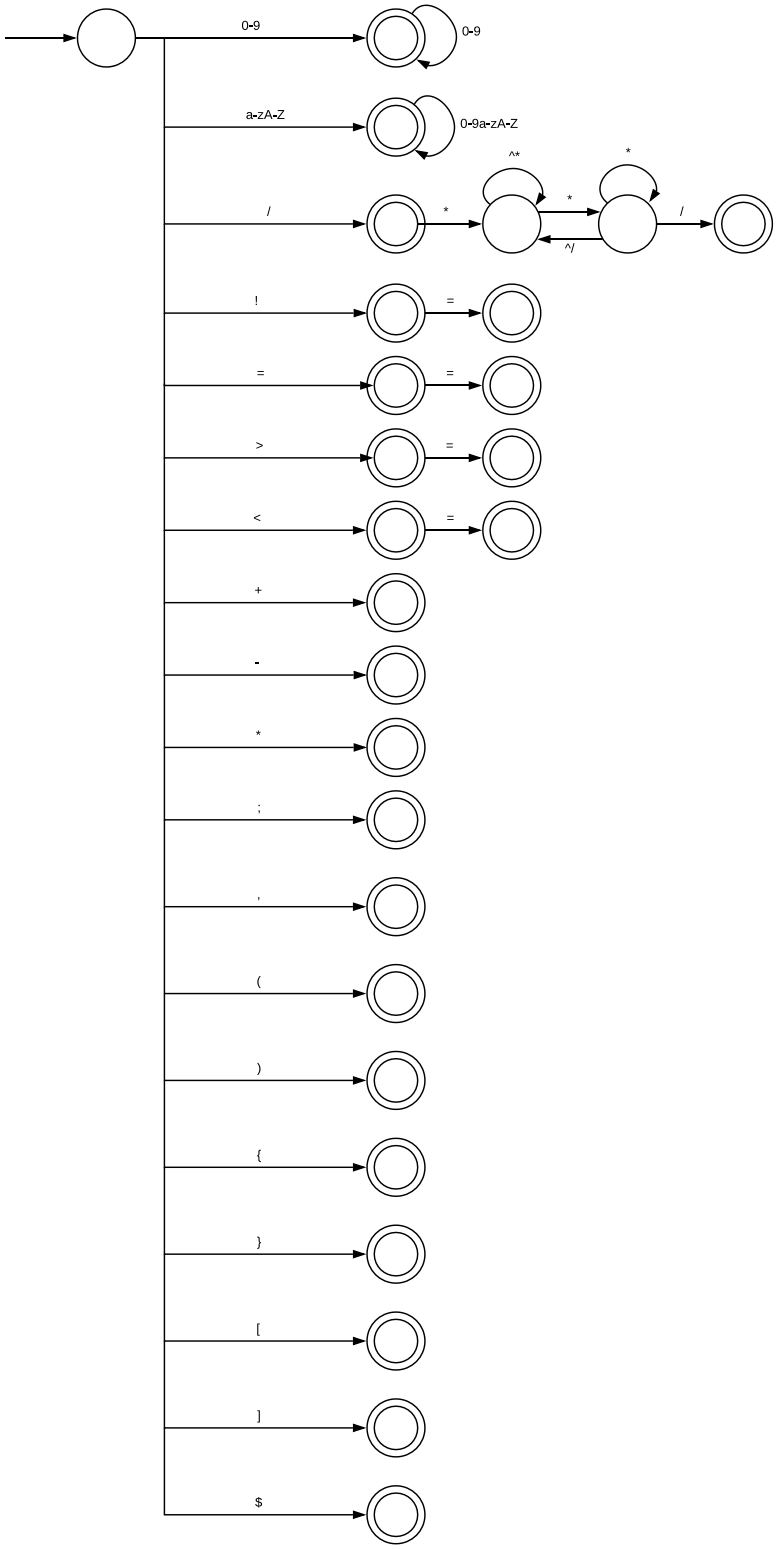
Anexo I: Expresiones Regulares

Expresiones regulares utilizadas para el tokenizador. Declaradas en formato de expresión regular de Python para escapar caracteres (uso de *backslash* '\'):

```
PLUS     \+
MINUS    -
TIMES    \*(?!/)
DIVIDE   /(?!\*)
ASSIGN   =
LT       <
LET      <=
GT       >
GET      >=
EQUALS   ==
NEQUALS  !=
COLON    ;
COMMA    ,
LPAREN   \(
RPAREN   \)
LBRACK   \[
RBRACK   \]
LCURLY   \{
RCURLY   \}
ENDFILE  \$
COMM     /\*(.|\n)*?\*/
NUM      \d+(?![0-9a-zA-Z])
ID       [a-zA-Z][a-zA-Z]*(?![0-9])
```

Anexo II: Autómata finito determinístico para la detección de tokens

El programa para la tokenización de C- puede implementarse con el siguiente AFD:



Anexo III: Gramática

```
program                 : declaration_list ENDFILE
type_specifier          : VOID
                        | INT
declaration_list        : declaration
                        | declaration_list declaration
declaration             : var_declaration
                        | function_definition
var_declaration         : type_specifier ID COLON
                        | type_specifier ID LBRACK NUM RBRACK COLON
function_definition     : type_specifier ID LPAREN params RPAREN compound_statement
params                  : param_list
                        | VOID
param_list              : parameter_declaration
                        | param_list COMMA parameter_declaration
parameter_declaration   : INT ID
                        | INT ID LBRACK RBRACK
compound_statement      : LCURLY local_declarations statement_list RCURLY
local_declarations      : local_declarations var_declaration
                        | empty
statement_list          : statement_list statement
                        | empty
statement               : expression_statement
                        | compound_statement
                        | selection_statement
                        | iteration_statement
                        | return_statement
expression_statement    : expression COLON
                        | COLON
selection_statement     : IF LPAREN expression RPAREN statement
                        | IF LPAREN expression RPAREN statement ELSE statement
iteration_statement     : WHILE LPAREN expression RPAREN statement
return_statement        : RETURN COLON
                        | RETURN expression COLON
expression              : var ASSIGN expression
                        | simple_expression
var                     : ID
                        | ID LBRACK expression RBRACK
simple_expression       : additive_expression relop additive_expression
                        | additive_expression
relop                   : LT
                        | LET
                        | GT
                        | GET
                        | EQUALS
                        | NEQUALS
additive_expression     : additive_expression addop term
                        | term
addop                   : PLUS
                        | MINUS
term                    : term multop factor
                        | factor
multop                  : TIMES
                        | DIVIDE
factor                  : LPAREN expression RPAREN
                        | var
                        | call
                        | NUM
call                    : ID LPAREN args RPAREN
args                    : arg_list
                        | empty
arg_list                : arg_list COMMA expression
                        | expression
empty                   :
```

Anexo IV: Reglas lógicas

$$\frac{\vdash i\ es\ literal\ entera}{\vdash i: int}$$

$$\frac{O(a) = int}{O \vdash a: int}$$

$$\frac{\vdash f\ es\ una\ función\ con\ retorno\ int}{\vdash f(): int}$$

$$\frac{\vdash a: int \quad \vdash b: int}{\vdash a + b: int}$$

$$\frac{\vdash a: int \quad \vdash b: int}{\vdash a - b: int}$$

$$\frac{\vdash a: int \quad \vdash b: int}{\vdash a * b: int}$$

$$\frac{\vdash a: int \quad \vdash b: int}{\vdash a/b: int}$$

$$\frac{\vdash a: int \quad \vdash b: int}{\vdash a > b: int}$$

$$\frac{\vdash a: int \quad \vdash b: int}{\vdash a \geq b: int}$$

$$\frac{\vdash a: int \quad \vdash b: int}{\vdash a < b: int}$$

$$\frac{\vdash a: int \quad \vdash b: int}{\vdash a \leq b: int}$$

$$\frac{\vdash a: int \quad \vdash b: int}{\vdash a == b: int}$$

$$\frac{\vdash a: int \quad \vdash b: int}{\vdash a\ != b: int}$$

$$\frac{\vdash f\ es\ una\ función\ con\ retorno\ void}{\vdash f(): void}$$

$$\frac{O(a) = int[]}{\vdash a: int[]}$$

$$\frac{\vdash a: int[]}{\vdash a[n]: int}$$