Compilers

Lab 5

Piotr Błaszyński

23rd March 2022

Tasks (explained later in the document):

- if not already there, add compile option -std=c++11 to Makefile (where we use g++),
- prepare a rule for the non-terminal symbol representing the assignment (if we have not done so previously),
- create a symbol array (store all identifiers in it),
- generate temporary variables to store the results of triples,
- for each triplet generate 4 lines of assembly code and store them in a vector,
- write the code that writes the lines from the vector to the file *yyout*, call this code after *yyparse*,
- after *yyparse* write the symbol array to the file *symbols.txt*,
- save the symbols from the symbol array before the code in the data block.

The symbol array contains all identifiers, including identifiers for temporary variables generated by the compiler. It should be a hash table (in C++ *std* :: *map*). The symbol array allows you to store and search for variable information based on the identifier: variable type, storage location, array size if any. It should be saved to the file *symbols.txt* at the end of the compiler run.

Temporary variables will be used to store intermediate calculation results in memory. Subsequent variables should be numbered. It should not be possible to use a variable with the same name as a temporary variable in normal code.

The example code generated for three is of the form:

```
li $t0, 27
lw $t1, x
sub $t0, $t0, $t1
sw $t0, result15
```

The registers $\$t0 - \$t7$ are so-called temporary registers. You can insert a value into such a register using the (for now) *li* or *lw* instruction. The *li* instruction inserts a numeric (direct) value. The *lw* instruction inserts a value from the address called by the variable name. The *sub* instruction performs subtraction, the *add* instruction performs addition, the *mul* instruction performs multiplication, the *div* instruction performs division, and the *sw* instruction inserts a value from a register into a memory cell. For arithmetic operations, the operation is performed on the last two registers and the result is stored in the first register.
The generalized code generated for three is of the form (underscores indicate spaces to be filled):

```
l_ $t0, __
l_ $t1, __
___ $t0, $t0, $t1
sw $t0, ____
```

The code for the assignment can be shortened (I leave the shortened form to you to invent).

The header part (the data block) will be used to reserve space for variables (in the future, also constants). The data block starts with the *.data* directive
Data block format:

```
name: type value
```

Przykłady:

```
        x:                .word    0
        arr:              .space   40
        caption:          .asciiz  "Screen␣caption"
        f:                .float 3.14
```

The code begins with the *.text* directive. A comment in assembler code is denoted by '#'.

Example code for the expression ($x = 3; z = 5 + x * 2;$):

```
.data
        x:       .word   0
        z:       .word   0
  result1:       .word   0
  result2:       .word   0
.text
        li $t0, 3
        sw $t0, x
        lw $t0, x
        li $t1, 2
        mul $t0, $t0, $t1
        sw $t0, result1

        li $t0, 5
        lw $t1, result1
        add $t0, $t0, $t1
        sw $t0, result2

        lw $t0, result2
        sw $t0, z
```