Formal Languages and Compilers
Prof. Breveglieri, Morzenti, Agosta
Written exam: laboratory question 30/06/2023[1]

Time: 60 minutes. Textbooks and notes can be used. Pencil writing is allowed.
Important: Write your name on any additional sheet.

SURNAME (Cognome): ....

NAME (Nome): ....

Matricola: .... or Person Code: ....

Instructor: ☐ Prof. Breveglieri ☐ Prof. Morzenti ☐ Prof. Agosta

The laboratory question must be answered taking into account the implementation of the `Acse` compiler given with the exam text.

Modify the specification of the lexical analyser (`flex` input) and the syntactic analyser (`bison` input) and any other source file required to extend the `Lance` language with support for a **software-emulated division operator**.

The software-emulated division operator shall have the same functional behavior of regular division for positive dividend and divisors. However, *no DIV instructions* shall be emitted by ACSE to translate it to assembly language. The operation may have undefined behavior if either or both dividend and divisor are negative — i.e. the implementation does not need to handle that case. Division by zero must be handled by computing a quotient of $2^{31} - 1$. The proposed implementation should perform proper constant folding. The code executed at compile time is not restricted in its use of division instructions. Therefore the C division operator may be freely used in the implementation, while the following functions are not allowed:

- `genDIV()`,
- `genDIVI()`,

There are no constraints imposed regarding the run-time complexity of the software-emulated division.

The symbol associated to this new operator is a bracketed slash (**`[/]`**) instead of the familiar slash (`/`). Its precedence and associativity are the same as non-emulated standard division, which is retained and coexists with the new software-emulated one.

The following code snippet exemplifies the use of the new operator. As long as both a and b are positive and $b \neq 0$, the snippet will always print `1` when executed.

```
int a, b;

read(a);
read(b);
if (a / b == a [/] b) {
  write(1);
} else {
  write(0);
}
```

Tips: The simplest algorithm for computing division is called *repeated subtraction.* To compute a/b, it subtracts b from a until a becomes smaller than b. The quotient is given by the number of subtractions performed.

In your solution you can write `INT_MAX` to refer to the constant $2^{31} - 1$.

[1]The text and solution to this exam have been adapted to ACSE 2.0 from its initial formulation.

1. Define the tokens (and the related declarations in **scanner.l** and **parser.y**). (1 points)
2. Define the syntactic rules or the modifications required to the existing ones. (2 points)
3. Define the semantic actions needed to implement the required functionality. (22 points)

Solution

The solution is shown below in *diff* format. All lines that begin with '+' were added, while the lines that begin with '−' were removed. **It is not required and it is not encouraged to provide a solution in *diff* format to get the maximum grade.**

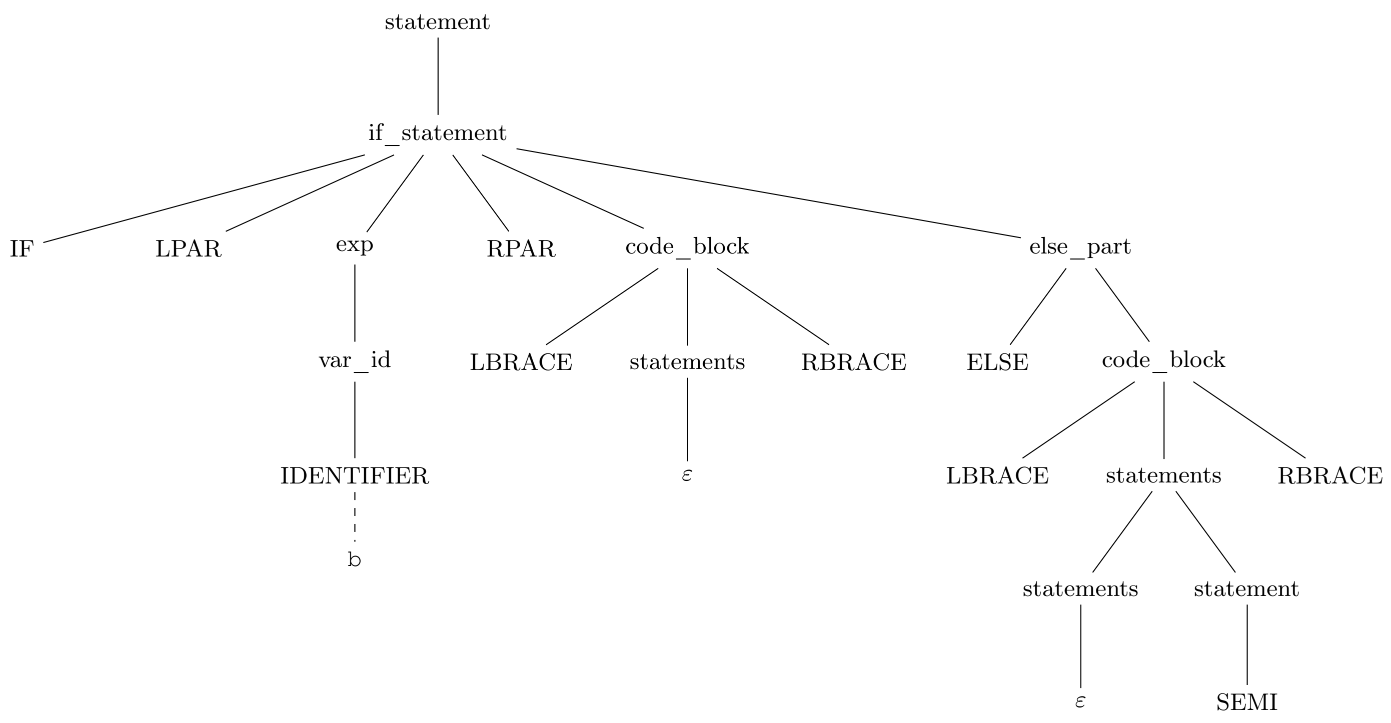
```
diff --git a/acse/parser.y b/acse/parser.y
--- a/acse/parser.y
+++ b/acse/parser.y
@@ -70,6 +70,7 @@ void yyerror(const char *msg)
 %token TYPE
 %token RETURN
 %token READ WRITE ELSE
+%token SW_DIV

 // These are the tokens with a semantic value.
 %token <ifStmt> IF
@@ -105,7 +106,7 @@ void yyerror(const char *msg)
 %left LT GT LTEQ GTEQ
 %left SHL_OP SHR_OP
 %left PLUS MINUS
~~-%left MUL_OP DIV_OP MOD_OP~~
+%left MUL_OP DIV_OP MOD_OP SW_DIV
 %right NOT_OP

 /*
@@ -438,6 +439,38 @@ exp
     $$ = getNewRegister(program);
     genOR(program, $$, rNormalizedOp1, rNormalizedOp2);
   }
+  | exp SW_DIV exp
+  {
+    // Reserve the register that will hold the result (quotient).
+    $$ = getNewRegister(program);
+
+    // Generate code that checks if the divisor is not zero, and in that case
+    // jumps to the actual computation
+    t_label *lNotZero = createLabel(program);
+    genBNE(program, REG_0, $3, lNotZero);
+    // Otherwise set the result to INT_MAX and we are done
+    genLI(program, $$, INT_MAX);
+    t_label *lExit = createLabel(program);
+    genJ(program, lExit);
+    assignLabel(program, lNotZero);
+
+    // Generate the loop code.
+    // First generate the initialization of the result to zero
+    genLI(program, $$, 0);
+    // Generate the label for the back-edge
+    t_label *lLoop = createLabel(program);
+    assignLabel(program, lLoop);
+    // Generate code to exit the loop if what remains of the dividend < divisor
+    genBLT(program, $1, $3, lExit);
+    // Generate code to subtract divisor from dividend and count one subtraction
+    genSUB(program, $1, $1, $3);
+    genADDI(program, $$, $$, 1);
+    // Generate a jump back to the beginning of the loop
+    genJ(program, lLoop);
+
+    // Assign the label used to skip the loop altogether
+    assignLabel(program, lExit);
+  }
 ;

 var_id
diff --git a/acse/scanner.l b/acse/scanner.l
--- a/acse/scanner.l
+++ b/acse/scanner.l
@@ -72,6 +72,7 @@ ID                       [a-zA-Z_][a-zA-Z0-9_]*
 "&&"                      { return ANDAND; }
 "||"                      { return OROR; }
 ","                       { return COMMA; }
+"[/]"                     { return SW_DIV; }

 "do"                      { return DO; }
 "else"                    { return ELSE; }
diff --git a/tests/sw_div/sw_div.src b/tests/sw_div/sw_div.src
--- /dev/null
+++ b/tests/sw_div/sw_div.src
@@ -0,0 +1,25 @@
+int a, b;
+a = 1234;
+b = 33;
+
+write(1234 / 33);
+write(1234 [/] 33);
+
+write(1234 / b);
+write(1234 [/] b);
+
+write(a / 33);
+write(a [/] 33);
+
+write(a / b);
+write(a [/] b);
+
+b = 0;
+write(a [/] 0);
+write(1234 [/] 0);
+write(a [/] b);
+
+read(a);
+read(b);
+write(a / b);
+write(a [/] b);
```

4. Given the following `Lance` code snippet:

```
if (b) {} else { ; }
```

write down the syntactic tree generated during the parsing with the Bison grammar described in Acse.y *starting from the* `statement` *nonterminal.* (5 points)

Solution



5. (**Bonus**) Briefly explain in plain English words how to modify the solution given to the questions about the software-emulated division in order to add support for negative dividends and negative divisors.