Homework – LL(1) Parser to Assembly

Goal:

Past assignments have implemented an LL(1) grammar processing many types of expressions. This assignment's goal is to map the internal representation (IR) into useful assembly.

Purposes:

- Understand the more mechanical process of turning code into assembly, rather than just providing a hand-rolled solution.
- Further utilizing your symbol table.
- Tracking actual registers in use.

Assignment:

Fully compile the file assembly.txt into a working x64 executable.

The steps will proceed as follows:

1) Add to your IR new productions
2) Parse and store all information into IR. Also store all variables and procedures names into a symbol table
3) Generate all x64 assembly code. I recommend writing all procedures first, and then all "main()" code afterward.

Console input and output will utilize calling printf and scanf directly, so you don't need to implement your own signed/unsigned/floating pointer displaying logic. You also do not need to implement floating point arithmetic in x64.

The grammar has been updated below for calling procedures. New productions are preceded with an asterisk. (You can ignore the if/else productions.)

```
Goal                      LineFull
LineFull                  VarType  VarTypeAfter
LineFull                  LineVarName
* LineFull                ExprWithoutName
LineFull                  return  GTerm
* LineFull                if  (  Condition  )  {
* LineFull                else  {
LineFull                  }
* LineFull                printNum name
* LineFull                printIsh name
* LineFull                readNum name
* LineFull                readIsh name
* LineFull                printString "any text in here"
VarTypeAfter              LineVarName
VarTypeAfter              procedure  name  ProcedureParams  {
LineVarName               name  LineVarNameRemaining
LineVarNameRemaining      =  Expr
LineVarNameRemaining      PowerAndRightOp  MultDiv'  AddSub'
LineVarNameRemaining      MultAndRightOp  AddSub'
LineVarNameRemaining      DivAndRightOp  AddSub'
LineVarNameRemaining      AddSub'
```

```
    * ExprWithoutName        num_value  Power'  MultDiv'  AddSub'
    * ExprWithoutName        negnum_value  Power'  MultDiv'  AddSub'
    * ExprWithoutName        Parens  Power'  MultDiv'  AddSub'
    * Condition              Expr  ==  Expr
    * Condition              Expr  !=  Expr
    ProcedureParams          (  Params  )
    Params                   VarType  name  MoreParams
    Params                   ϵ
    MoreParams               ,  VarType  name  MoreParams
    MoreParams               ϵ
    VarType                  num
    VarType                  ish
    Expr                     LTermAddSub  AddSub'
    LTermAddSub              LTermMultDiv  MultDiv'
    LTermMultDiv             LTermPower  Power'
    RTermAddSub              RTermMultDiv  MultDiv'
    RTermMultDiv             RTermPower  Power'
    AddSub'                  +  RTermAddSub  AddSub'
    AddSub'                  -  RTermAddSub  AddSub'
    AddSub'                  ϵ
    MultDiv'                 MultAndRightOp
    MultDiv'                 DivAndRightOp
    MultDiv'                 ϵ
    MultAndRightOp           *  RTermMultDiv  MultDiv'
    DivAndRightOp            /  RTermMultDiv  MultDiv'
    Power'                   PowerAndRightOp
    Power'                   ϵ
    PowerAndRightOp          ^  RTermPower  Power'
    LTermPower               GTerm
    LTermPower               negnum_value
    LTermPower               negish_value
    LTermPower               negname
    RTermPower               GTerm
    * GTerm                  NameOrProcedure
    GTerm                    Parens
    * GTerm                  num_value
    * GTerm                  ish_value
    GTerm                    SpaceNegVal
    * NameOrProcedure        name  Arguments
    * Arguments              (  Expr  MoreArguments  )
    * Arguments              ϵ
    * MoreArguments          ,  Expr  MoreArguments
    * MoreArguments          ϵ
    Parens                   (  Expr  )
    SpaceNegVal              spacenegnum_value
    SpaceNegVal              spacenegish_value
    SpaceNegVal              spacenegname
```

Console input/output:

This is completed with printf/scanf. See the nasm file below:

```
            section .data
msg:        db "Hello, this is my string", 0
```

```
fmtstr:     db "%s", 10, 0
fmtuint:    db "%d", 10, 0
fmtuintin:  db "%d", 0
fmtfloatin: db "%f", 0
float1:     dd     0.0

            section .bss
var1:       resd   1

            section .text

            extern printf
            extern scanf
            global main
main:
            push rbp ; Push base pointer onto stack to save it

            ; print a string
            mov rsi, msg
            mov rdi, fmtstr
            mov rax, 0
            call printf

            ; print an unsigned integer
            mov rsi, 42
            mov rdi, fmtuint
            mov rax, 0
            call printf

            ; read an integer
            lea rdi, [fmtuintin]
            lea rsi, [var1]
            mov rax, 0
            call scanf

            ; print integer back out
            lea rdi, [fmtuint]
            mov rsi, [var1]
            xor rax, rax
            call printf

            ; read float
            lea rdi, [fmtfloatin]  ; read string format input
            lea rsi, [float1]      ; read float storage location
            call scanf

            ; print float
            lea rdi, [fmtfloat]    ; read string format output
            movss xmm0, [float1]   ; read float storage location into xmm0
            cvtss2sd  xmm0, xmm0   ; convert float to double
            mov rax, 1             ; tell printf there is 1 double parameter
            call printf
            call printf


            pop rbp  ; restore stack base pointer
            mov rbx, 0
            mov rax, 0
            int 0x80
```

Instructions:

Implement the file irassembly.txt
Note, printing floating point in assembly is ***optional***.

Verification:

- Verify by console output.