F29LP: PROGRAMMING ASSIGNMENT [20 MARKS]

OVERVIEW

Your task is to develop a compiler for the language given below, called FUNC. The compiler should produce MIPS code that can be emulated using the MARS MIPS emulator. You are expected to use the labs throughout the course to work on it, as this is where you can help with any difficulties you have.

The assignment is split into two parts:

- **Part 1** is concerned with the implementation of the front end. You have to complete this by **Thursday of week 7 (additional lab slot).** This is worth **10 marks**.
- **Part 2** is concerned with the implementation of the back end. You have to complete this by **Tuesday of week 12**. This is worth **10 marks**.

SUBMISSION DETAILS

Your solution will be **marked in during the labs**. You can get it marked any time until the deadline. In addition, **you have to submit a zip archive of your source code on Vision before each deadline.**

THE SOURCE LANGUAGE

The FUNC language consists of a collection of functions and has the following syntax:

```
<program> ::= <funcs>
<funcs> ::= <func>; [<funcs>]
<func> ::= function <name>([<args>])[returns <name>]
               [variables <args>]
               begin <commands> end function
<args> ::= <name> [,<args>]
<commands> ::= <command>; [<commands>]
<command> ::= <assign> | <if> | <while> | read <name> | write <expr>
<assign> ::= <name> := <expr>
<if> ::= if <condexpr> then <commands> [else <commands>] end if
<for> ::= while <condexpr> loop <commands> end loop
<condexpr> ::= <bop> ( <exprs> )
<bop> ::= Less | LessEq | Eq | NEq
<exprs> ::= <expr> [,<exprs>]
<expr> ::= <name>[( <exprs> )] | <number>
```

<number> is a natural number

<name> is any string starting with character followed by characters or numbers (that is disjoint from the keywords)

In addition, you can assume the following:

- Each program should have a function called **Main** with no arguments and no return value.
- All other functions should have one or two return values.
- You should support the following built-in functions, which accepts two integers and returns an integer:
  - **Plus**, which adds the arguments;
  - **Times**, which multiplies the arguments;
  - **Minus**, which subtracts the arguments;

    - **Divide**, which divides the arguments.
- All the boolean operators (<bop> ::= **Less** | **LessEq** | **Eq** | **NEq)** are binary.
- The **read** command assumes that the given variable is an integer

The following example illustrates a valid FUNC program:

```
function Inc(x) returns y
begin
  y := Plus(x,1);
end function;

function MyTimes(x, y) returns times
variables i
begin
  times := 0;
  i := 0;
  while Less(i,y) loop
    times := Plus(x,times);
    i := Inc(i);
  end loop;
  write times;
end function;

function Main()
variables a, b, x, y
begin
  read a;
  read b;
  x := Times(a,b);
  y := MyTimes(a,b);
  if Eq(x,y)
   then write 1;
   else write 0;
  end if;
end function;
```

More examples can be found at the end of this document.

PART 1: FRONT END : 10 MARKS [DEADLINE: WEEK 7]

Your task is to implement the front end. To complete this part you need to understand lexical analysis, syntax analysis and abstract syntax trees. This is covered by the lectures in weeks 4 and 5. Your task is to produce:

- A FLEX file with a suitable token representation for the FUNC language (3 marks)
- A recursive descent parser for the grammar given above (4 marks)
- An AST with a suitable representation of the nodes generated by the parser (3 marks)

For partial solutions, a suitable number of marks will be deducted. In week 7 there will be an additional lab on Thursday. This is your final chance to get this part marked. More details of how this session will be organised will be provided in due course.

THE TARGET LANGUAGE

From the AST in part 1 your task is to develop the back-end that should generate MIPS code that can be emulated in the MARS tool. In the labs you can run MARS with the command:

```
$ mars
```

If you want to run it on your own computer, then you can download it from this address:

http://courses.missouristate.edu/KenVollmar/MARS/

When the compiler is executed with a given FUNC program

```
myprogram.fun
```

then it should generate a file

```
myprogram.asm
```

which can be opened and emulated in MARS.

Similar to the SIMP language from the lectures you should use "common sense" understanding of the semantics of most constructs. Here are some further requirements and assumptions you can make beyond those already discussed in part 1:

- You can assume that there are sufficient registers, so all variables can be in the registers.
- There is no global state.
- The code in the `Main()` function should be executed (using the `main:` label in MIPS)
    - All other functions can only be reached by calling them from `Main` (possibly indirectly via other function calls)
- Use the `$s0-$s7` registers to store variables of the `Main()` function.
- Use the `$t0-$t7` registers for the other functions.
- Use `$a0-$a3` to store arguments. You can assume that there are not more than 4 arguments
- Use `$v0` to store the return value. Note that there are no return statements in the language. The variable declared in the **`returns`** field should be returned in `$v0`.
- Registers should be allocated in the order variables are introduced, starting with `$s0/$t0`
- You can use `$t8` and `$t9` for intermediate computations as we have done for SIMP.

Further requirements and assumptions are detailed for each relevant part below, and may appear on Vision as the course progresses. A set of tests will be provided before the deadline and these will be used for marking.

PART 2: BACK END : 10 MARKS [DEADLINE: WEEK 12]

BASIC SOLUTION [4 MARKS]

The basic solution should support the `Main()` function with all its features, except the ability to call other user defined functions (you still need to support the built-in: `Plus, Minus, Times, Divide`) , For partial solutions, a suitable number of marks will be deducted. This will depend on the problem and/or which features are missing.

BASIC FUNCTION CALLS [3 MARKS]

The basic solution should be extended with function calls. You do not need to support nested and recursive calls. However, you need to implement a suitable register based protocol where `$a0-$a3` is used for arguments, and `$v0` for the result. You can assume a maximum of 4 arguments. Remember, you should use the `$t0-$t7` registers for local variables (except for `Main` where `$s0-$s7` are used).

NESTED AND RECURSIVE FUNCTIONS [3 MARKS]

Extend functions to allow recursive and nested calls. To achieve this, you need to develop a stack *frame*. As all variables are in registers, (when required) you will need to push a stack frame with all caller-save registers before the call is made and pop them back to the correct registers when control is returned to the caller.

FURTHER DETAILS

IMPLEMENTATION LANGUAGE

In the lecture notes we have been using C as the main implementation language. We have also briefly shown how to use Java. You are free to use any of these languages. You can also use other programming languages, as long as this is approved by the course leader.

However, note that you will only be given a solution in C. One requirement of the language used is that it has support for a lexical analyser comparable to FLEX.

SUBMISSION OF SOURCE CODE ON VISION

In addition to the demos during the labs, the source code needs to be submitted for each part with the same deadline specified. This should be submitted on Vision under `Assessment'. Late submissions may be penalised.

PLAGIARISM

You are being assessed on individual work. **That means that solutions that are developed in pairs/groups, or are very similar, will *NOT* be accepted and will be reported.** You should make sure you are familiar with the university plagiarism regulations

http://www.hw.ac.uk/students/doc/plagiarismguide.pdf

You should not use illegal software or copyrighted materials. You should acknowledge the source of material you have adapted, as well as help you have received from other students and staff.

LATE SUBMISSION & EXTENSION OF COURSEWORK

Students who have serious concerns about meeting submission dates for coursework should consult the Course Leader as soon as possible. Any extension to the submission deadline must be approved by the Course Leader, and the reason for the extension will be recorded. Applications for extensions made after the due submission date will not normally be approved.

For late submissions/demos the Course Leader will apply a penalty mark deduction on a case-by-case basis as an alternative time slot for the demo session needs to be found.

APPENDIX: EXAMPLES OF FUNC CODE & GENERATED MIPS CODE

BASIC SOLUTION

The following code should be supported in the basic solution:

```
function Main ()
variables inp,res
```

```
begin
  read inp;
  res := 0;
  while Less(0,inp) loop
     res := Plus(res,inp);
     inp := Minus(inp,1);
  end loop;
  write res;
end function;
```

The following MIPS code is a correct generation:

```
      .data
sinp: .asciiz
"INPUT> "

      .text
main:
      addi $sp,$sp,-4
      sw $a0,0($sp)
      addi $sp,$sp,-4
      sw $v0,0($sp)
      li  $v0, 4
      la  $a0, sinp
      syscall
      li  $v0, 5
      syscall
      move  $s0,$v0
      lw $v0,0($sp)
      addi $sp,$sp,4
      lw $a0,0($sp)
      addi $sp,$sp,4
      li $s1,0
WLOOP0:
      li $t8,0
      addi $sp,$sp,-4
      sw $t8,0($sp)
      move $t9,$s0
      lw $t8,0($sp)
      addi $sp,$sp,4
      bge $t8,$t9,WEND0
      move $t8,$s1
      addi $sp,$sp,-4
      sw $t8,0($sp)
      move $t9,$s0
      lw $t8,0($sp)
      addi $sp,$sp,4
      add $s1,$t8,$t9
      move $t8,$s0
      addi $sp,$sp,-4
      sw $t8,0($sp)
      li $t9,1
      lw $t8,0($sp)
      addi $sp,$sp,4
      sub $s0,$t8,$t9
      j WLOOP0
WEND0:
      addi $sp,$sp,-4
      sw $a0,0($sp)
      addi $sp,$sp,-4
      sw $v0,0($sp)
      move $a0,$s1
      li  $v0, 1
      syscall
      lw $v0,0($sp)
      addi $sp,$sp,4
      lw $a0,0($sp)
      addi $sp,$sp,4
      li $v0, 10
      syscall
```

It should sum up all numbers from 0 to the given number. E.g. if it reads **3** then it should print **6** (computing 3+2+1). Note that your solution does not have to be identical as long as the program produces the same answer and satisfies the specification in this document.

BASIC FUNCTION CALLS

The following program illustrates an example that should work with basic function calls. It does the same as the previous one, but with the computation that sums up the result in a separate function:

```
function Sum (inp) returns res
begin
  res := 0;
  while Less(0,inp) loop
     res := Plus(res,inp);
     inp := Minus(inp,1);
  end loop;
end function;

function Main ()
variables inp,res
```

```
begin
  read inp;
  res := Sum(inp);
  write res;
end function;
```

The following MIPS code is a correct generation:

```
      .data
sinp: .asciiz  "INPUT> "

      .text
main:
      addi $sp,$sp,-4
      sw $a0,0($sp)
      addi $sp,$sp,-4
      sw $v0,0($sp)
      li  $v0, 4
      la  $a0, sinp
      syscall
      li  $v0, 5
      syscall
      move  $s0,$v0
      lw $v0,0($sp)
      addi $sp,$sp,4
      lw $a0,0($sp)
      addi $sp,$sp,4
      move $a0,$s0
      jal Sum
      move $s1,$v0
      addi $sp,$sp,-4
      sw $a0,0($sp)
      addi $sp,$sp,-4
      sw $v0,0($sp)
      move $a0,$s1
      li  $v0, 1
      syscall
      lw $v0,0($sp)
      addi $sp,$sp,4
      lw $a0,0($sp)
      addi $sp,$sp,4
      li $v0, 10
      syscall

Sum:
      li $v0,0
WLOOP0:
      li $t8,0
      addi $sp,$sp,-4
      sw $t8,0($sp)
      move $t9,$a0
      lw $t8,0($sp)
      addi $sp,$sp,4
      bge $t8,$t9,WEND0
      move $t8,$v0
      addi $sp,$sp,-4
      sw $t8,0($sp)
      move $t9,$a0
      lw $t8,0($sp)
      addi $sp,$sp,4
      add $v0,$t8,$t9
      move $t8,$a0
      addi $sp,$sp,-4
      sw $t8,0($sp)
      li $t9,1
      lw $t8,0($sp)
      addi $sp,$sp,4
      sub $a0,$t8,$t9
      j WLOOP0
WEND0:
      jr $ra
```

NESTED/RECURSIVE CALLS

The following program illustrates support for nested and function acalls. It does the same as the previous one, but with the here the computation is achieved recursively:

```
function Sum (inp) returns res
variables tmp
begin
  if Eq(inp,0) then
    res := inp;
  else
    tmp := Sum(Minus(inp,1));
    res := Plus(tmp,inp);
  end if;
end function;

function Main ()
variables inp,res
begin
  read inp;
  res := Sum(inp);
```

```
  write res;
end function;
```

The following MIPS code is a correct generation:

```
      .data
sinp: .asciiz  "INPUT> "

      .text
main:
      addi $sp,$sp,-4
      sw $a0,0($sp)
      addi $sp,$sp,-4
      sw $v0,0($sp)
      li  $v0, 4
      la  $a0, sinp
      syscall
      li  $v0, 5
      syscall
      move  $s0,$v0
      lw $v0,0($sp)
      addi $sp,$sp,4
      lw $a0,0($sp)
      addi $sp,$sp,4
      move $a0,$s0
      jal Sum
      move $s1,$v0
      addi $sp,$sp,-4
      sw $a0,0($sp)
      addi $sp,$sp,-4
      sw $v0,0($sp)
      move $a0,$s1
      li  $v0, 1
      syscall
      lw $v0,0($sp)
      addi $sp,$sp,4
      lw $a0,0($sp)
      addi $sp,$sp,4
      li $v0, 10
      syscall

Sum:
      move $t8,$a0
      addi $sp,$sp,-4
      sw $t8,0($sp)
      li $t9,0
      lw $t8,0($sp)
      addi $sp,$sp,4
      bne $t8,$t9,IFALSE0
      move $v0,$a0
      j IFEND0
IFALSE0:
      addi $sp,$sp,-16
      sw $t0,0($sp)
      sw $ra,4($sp)
      sw $v0,8($sp)
      sw $a0,12($sp)
      move $t8,$a0
      addi $sp,$sp,-4
      sw $t8,0($sp)
      li $t9,1
      lw $t8,0($sp)
      addi $sp,$sp,4
      sub $a0,$t8,$t9
      jal Sum
      move $t0,$v0
      lw $ra,4($sp)
      lw $v0,8($sp)
      lw $a0,12($sp)
      addi $sp,$sp,16
      move $t8,$t0
      addi $sp,$sp,-4
      sw $t8,0($sp)
      move $t9,$a0
      lw $t8,0($sp)
      addi $sp,$sp,4
      add $v0,$t8,$t9
IFEND0:
      jr $ra
```

ILLUSTRATIVE EXAMPLE FROM PAGE 2

The following MIPS code is a correct generation of the example of page 2:

```
      .data
sinp: .asciiz     "INPUT> "

      .text
main:
      addi $sp,$sp,-4
      sw $a0,0($sp)
      addi $sp,$sp,-4
      sw $v0,0($sp)
      li  $v0, 4
      la  $a0, sinp
      syscall
      li  $v0, 5
      syscall
      move  $s0,$v0
      lw $v0,0($sp)
      addi $sp,$sp,4
      lw $a0,0($sp)
      addi $sp,$sp,4
      addi $sp,$sp,-4
      sw $a0,0($sp)
      addi $sp,$sp,-4
      sw $v0,0($sp)
      li  $v0, 4
      la  $a0, sinp
      syscall
      li  $v0, 5
      syscall
      move  $s1,$v0
      lw $v0,0($sp)
      addi $sp,$sp,4
      lw $a0,0($sp)
      addi $sp,$sp,4
      move $t8,$s0
      addi $sp,$sp,-4
      sw $t8,0($sp)
      move $t9,$s1
      lw $t8,0($sp)
      addi $sp,$sp,4
      mul $s2,$t8,$t9
      move $a0,$s0
      move $a1,$s1
      jal MyTimes
      move $s3,$v0
      move $t8,$s2
      addi $sp,$sp,-4
      sw $t8,0($sp)
      move $t9,$s3
      lw $t8,0($sp)
      addi $sp,$sp,4
      bne $t8,$t9,IFALSE0
      addi $sp,$sp,-4
      sw $a0,0($sp)
      addi $sp,$sp,-4
      sw $v0,0($sp)
      li $a0,1
      li  $v0, 1
      syscall
      lw $v0,0($sp)
      addi $sp,$sp,4
      lw $a0,0($sp)
      addi $sp,$sp,4
      j IFEND0
IFALSE0:
      addi $sp,$sp,-4
      sw $a0,0($sp)
      addi $sp,$sp,-4
      sw $v0,0($sp)
      li $a0,0
```

```
      li  $v0, 1
      syscall
      lw $v0,0($sp)
      addi $sp,$sp,4
      lw $a0,0($sp)
      addi $sp,$sp,4
IFEND0:
      li $v0, 10
      syscall

Inc:
      move $t8,$a0
      addi $sp,$sp,-4
      sw $t8,0($sp)
      li $t9,1
      lw $t8,0($sp)
      addi $sp,$sp,4
      add $v0,$t8,$t9
      jr $ra


MyTimes:
      li $v0,0
      li $t0,0
WLOOP1:
      move $t8,$t0
      addi $sp,$sp,-4
      sw $t8,0($sp)
      move $t9,$a1
      lw $t8,0($sp)
      addi $sp,$sp,4
      bge $t8,$t9,WEND1
      move $t8,$a0
      addi $sp,$sp,-4
      sw $t8,0($sp)
      move $t9,$v0
      lw $t8,0($sp)
      addi $sp,$sp,4
      add $v0,$t8,$t9
      addi $sp,$sp,-20
      sw $t0,0($sp)
      sw $ra,4($sp)
      sw $v0,8($sp)
      sw $a0,12($sp)
      sw $a1,16($sp)
      move $a0,$t0
      jal Inc
      move $t0,$v0
      lw $ra,4($sp)
      lw $v0,8($sp)
      lw $a0,12($sp)
      lw $a1,16($sp)
      addi $sp,$sp,20
      j WLOOP1
WEND1:
      addi $sp,$sp,-4
      sw $a0,0($sp)
      addi $sp,$sp,-4
      sw $v0,0($sp)
      move $a0,$v0
      li  $v0, 1
      syscall
      lw $v0,0($sp)
      addi $sp,$sp,4
      lw $a0,0($sp)
      addi $sp,$sp,4
      jr                $ra
```