The SCC150 Assembler Emulator

1 Overview

The aim of this task is to develop a c emulator for a simple made-up assembler language (Let's call it SCC150 Assembler). SCC150 Assembler assumes a simple hardware platform which provides 5 registers (REGA, REGB, REGC, REGX and INSP). SCC150 Assembler supports an instruction set of 10 instructions. These are NOP, SET, AND, OR, ADD, SUB, SHL, SHR, JMP and PRT. The emulator should be able to read small programs written in SCC150 Assembler and execute these.

2 Introduction

In this task an Emulator for SCC150 Assembler should be implemented. A c program (the Emulator) should be developed that can read and execute little SCC150 Assembler programs (The Emulator will be similar to the MARS emulator used before in SCC150; however, it will obviously be a much simpler program, without a GUI and for a much simpler platform). SCC150 Assembler programs are composed of lines where each line holds one instruction. SCC150 Assembler assumes a simple hardware platform which provides 5 registers (REGA, REGB, REGC, REGX and INSP). SCC150 Assembler supports an instruction set of 10 instructions. These are NOP, SET, AND, OR, ADD, SUB, SHL, SHR, JMP and PRT. REGA, REGB, REGC are normal registers. REGX is used for JMP (if the register is 0 the instruction pointer INSP is set to the value given by JMP). The INSP register contains a number indicating which instruction is executed next. All instructions except JMP will lead to a simple increment of INSP. Here is an example SCC150 Assembler program (The full SCC150 Assembler instruction set is given at the end of this document):

```
SET REGA 20
PRT REGA
SUB REGA 1
SET REGX REGA
JMP 7
SET REGX 0
JMP 1
NOP
```

At program start INSP is set to 0 and the first line of the program (line 0; we start counting from 0) is executed. Register REGA is set to the value of 20 and INSP is set to 1. In the next line PRT is used to print out the content of register REGA (and again, INSP is incremented). Then one is subtracted from the value stored in REGA; the result is stored in REGA. Then REGX is set to the value of REGA. Next, JMP tests if REGX is 0 and if this is the case INSP is set to 7 and the next instruction is NOP. If REGX is not 0 INSP is just incremented and next REGX is set to 0 which then leads to JMP 1 which sets INSP to 1 which then leads to execution of PRT REGA again. The program implements a simple loop counting from 20 down to 1, printing out each number. The output of the program is:

```
roedig$ ./emulator
PROGRAM:
0:  SET REGA 20
1:  PRT REGA
2:  SUB REGA 1
3:  SET REGX REGA
4:  JMP 7
5:  SET REGX 0
6:  JMP 1
7:  NOP
RUNNING PROGRAM ...
```

```
REGA = 20
REGA = 19
REGA = 18
...
...
REGA = 3
REGA = 2
REGA = 1
...  DONE!
```

3 Task

3.1 Emulator Implementation

A template for the SCC150 Emulator (emulator_template.c) is provided. This program template should be used as starting point and should be completed. You should use the mystring.h library instead of string.h that you implemented in the previous practical (All functionality you require from string.h must be implemented in mystring.h). The program template provides already some functionality such as reading the program file, the execution framework and necessary variable and function definitions. The program reads an SCC150 Assembler program from a file named prog.txt and places the program into the emulator memory (variable prog[][], function load_program()). The function exec_program() takes care of executing the loaded program. Depending on the register INSP the right line of the SCC150 Assembler program is addressed and the string containing the instruction to be executed is passed to function exec_instruction(). The function exec_instruction() must be implemented. This function should tokenize the line specifying the instruction and its arguments. Then the right function for the extracted instruction should be called. An example implementation for the instructions NOP and PRT is already given in the template. Further hints on implementation can be found in the template.

3.2 An Example Program

Construct a program written in SCC150 Assembler which can perform multiplication of two integer numbers. This could be achieved using the Ethiopian Multiplication. The two numbers to be multiplied should be put in REGA and REGB. To multiply for example the numbers 17 and 34 the program should start with the following two lines:

```
SET REGA 17
SET REGB 34
...
...
```

3.3 SCC150 Assembler Testing

In addition to the SCC150 Emulator program template a compiled version of a completed emulator is provided (Obviously, the corresponding c file is not provided). This working emulator can be used to test SCC150 Assembler programs. A compiled version for Linux and OSX is provided.

4 Instruction Set

`INST ARG1 ARG2`
SCC150 Assembler instructions have the general format of INST ARG1 ARG2. Depending on the instruction type INST, ARG1 and ARG2, only ARG1 or no argument may be used.

`NOP`
This instruction has no arguments. This No Operation instruction does not modify register state except INSP which is incremented by one.

`SET ARG1 ARG2`
This instruction has two arguments. SET is used to load ARG1 with ARG2. ARG1 might be any register except INSP. ARG2 might be any register except INSP or an integer number in decimal format. INSP is incremented by one. Examples of valid instructions are:

```
    SET REGA 5 // load REGA with 5
```

```
SET REGA REGB // load content of REGB into REGA
```

AND ARG1 ARG2
This instruction has two arguments. AND is used to perform a bitwise and of ARG1 and ARG2. The result is stored in ARG1. ARG1 might be any register except INSP. ARG2 might be any register except INSP or an integer number in decimal format. INSP is incremented by one. Examples of valid instructions are:

```
AND REGA 5 // REGA = REGA & 5
AND REGA REGB // REGA = REGA & REGB
```

OR ARG1 ARG2
This instruction has two arguments. OR is used to perform a bitwise or of ARG1 and ARG2. The result is stored in ARG1. ARG1 might be any register except INSP. ARG2 might be any register except INSP or an integer number in decimal format. INSP is incremented by one. Examples of valid instructions are:

```
OR REGA 5 // REGA = REGA | 5
OR REGA REGB // REGA = REGA | REGB
```

ADD ARG1 ARG2
This instruction has two arguments. ADD is used to add ARG1 and ARG2. The result is stored in ARG1. ARG1 might be any register except INSP. ARG2 might be any register except INSP or an integer number in decimal format. INSP is incremented by one. Examples of valid instructions are:

```
ADD REGA 5 // REGA = REGA + 5
ADD REGA REGB // REGA = REGA + REGB
```

SUB ARG1 ARG2
This instruction has two arguments. SUB is used to subtract ARG2 from ARG1. The result is stored in ARG1. ARG1 might be any register except INSP. ARG2 might be any register except INSP or an integer number in decimal format. INSP is incremented by one. Examples of valid instructions are:

```
SUB REGA 5 // REGA = REGA - 5
SUB REGA REGB // REGA = REGA - REGB
```

SHL ARG1 ARG2
This instruction has two arguments. SHL is used to perform a shift left of ARG1 by the number of places given by ARG2. The result is stored in ARG1. ARG1 might be any register except INSP. ARG2 might be any register except INSP or an integer number in decimal format. INSP is incremented by one. Examples of valid instructions are:

```
SHL REGA 5 // REGA = REGA << 5
SHL REGA REGB // REGA = REGA << REGB
```

SHR ARG1 ARG2
This instruction has two arguments. SHR is used to perform a shift right of ARG1 by the number of places given by ARG2. The result is stored in ARG1. ARG1 might be any register except INSP. ARG2 might be any register except INSP or an integer number in decimal format. INSP is incremented by one. Examples of valid instructions are:

```
SHL REGA 5 // REGA = REGA >> 5
SHL REGA REGB // REGA = REGA >> REGB
```

JMP ARG1
This instruction has one argument. If the register REGX is 0 the instruction pointer INSP is set to ARG1. If REGX is not 0 the instruction pointer INSP is incremented by one. ARG1 can be an integer number in decimal format. An example is:

```
SET REGX 0 // REGX = 0
JMP 5 // INSP is set to 5 as REGX is 0
```

PRT ARG1
This instruction has one argument. ARG1 can be any register. The register content will be printed. INSP is incremented by one.

```
PRT REGX // print the content of REGX
```