Boğaziçi University CMPE
Software Engineering

SWE514 - Computer Systems
Simple to A86 Assembly Code Translator

Instructor: Prof. Dr. Can ÖZTURAN

Students: Cihangir ÖZMÜŞ (2017719102)

Contents

1. Introduction

A compiler is computer software that transforms computer code written in one programming language (the source language) into another programming language (the target language). Compilers are a type of translator that support digital devices, primarily computers. The name compiler is primarily used for programs that translate source code from a high-level programming language to a lower level language (e.g., assembly language, object code, or machine code) to create an executable program.(PC Magazine 2017)

The objective of this study is to build a program to translate a imaginary language called *Simple* to A86 assemly codes. Because it has powerful string functions, in this study we used Java language and İntellij IDEA to translate code strings given in syntax of Simple language to A86 assembly code. We also used Antlr parser generator to parse the code strings and **“EvalVisitor”** class to visit, define and use the parsed code strings.

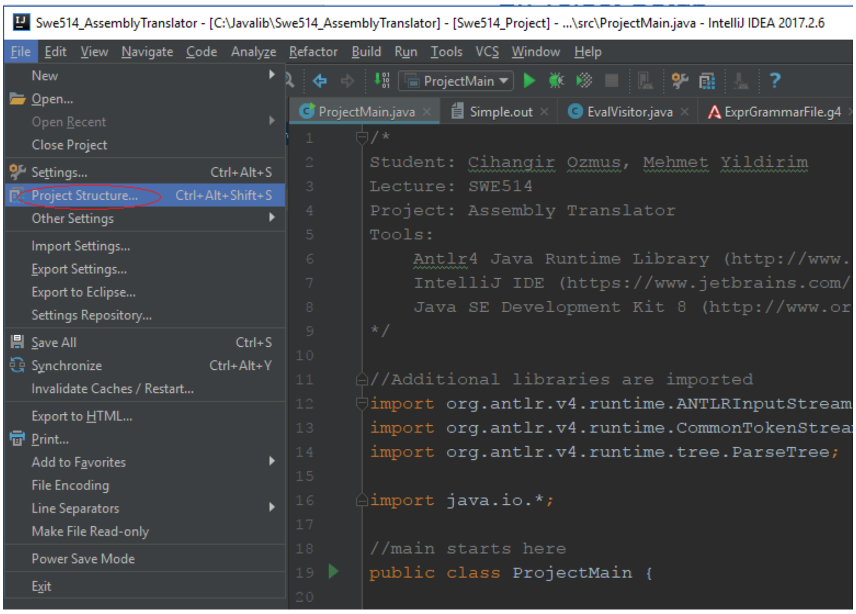
The program built in this project will translate the inputs given in Simple language and compile it to A86 assembly code ie. it will take the codes given in Simple language syntax (Simple.in) as input and give A86 assembly code as output(Simple.out).

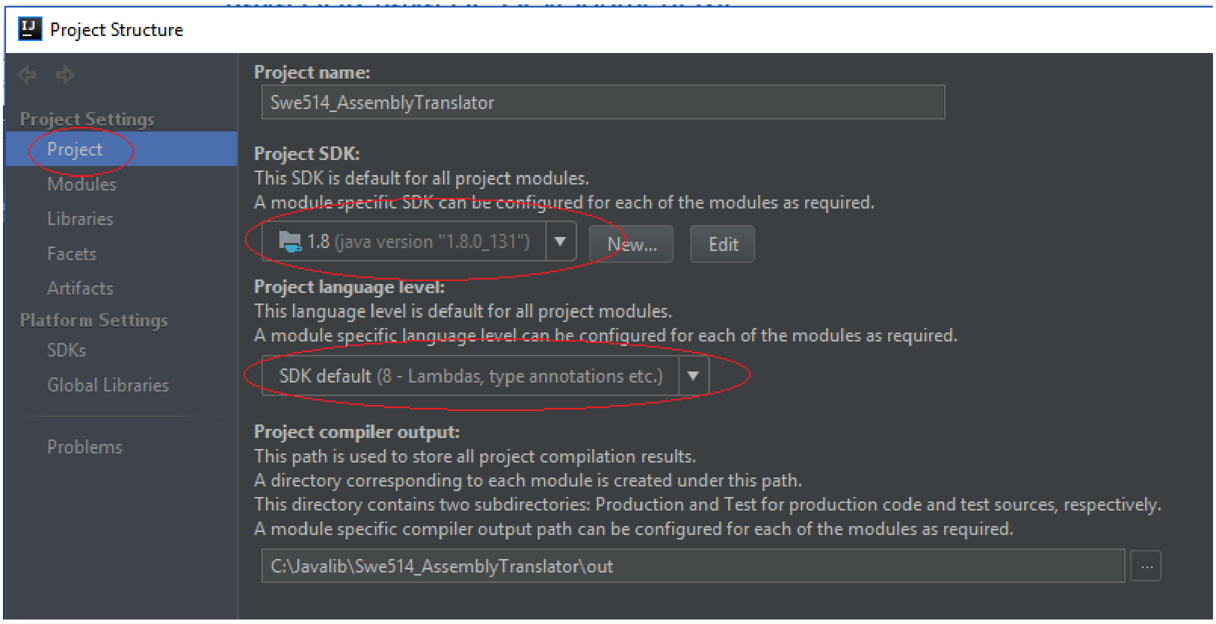
2. Project Setup

To compile the Simple to Assembly Code Translator program users must have Java SDK 1.8.XXX and Intellij IDEA installed on their computers.

[Java SDK Java SE Development Kit]
[IntelliJ IDEA - Community Edition]
[Antlr4 - Java Runtime Library] *(That runtime library is delivered in our project folder.)*

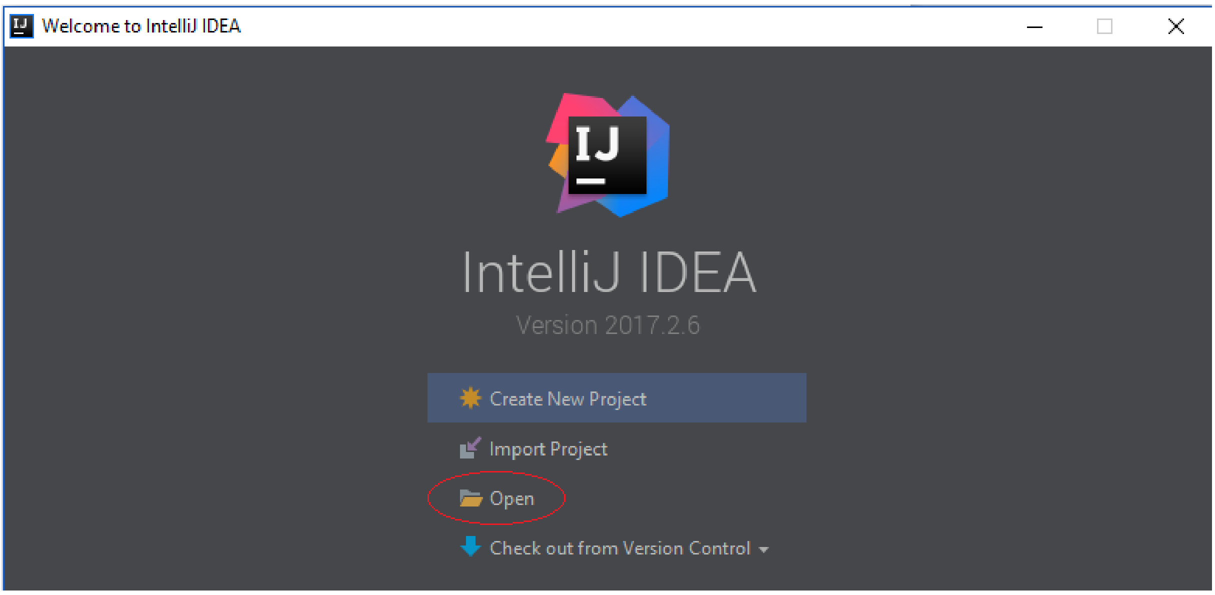
Project Languge Level should be set to 8 on the project setup menu of Intellij IDEA.



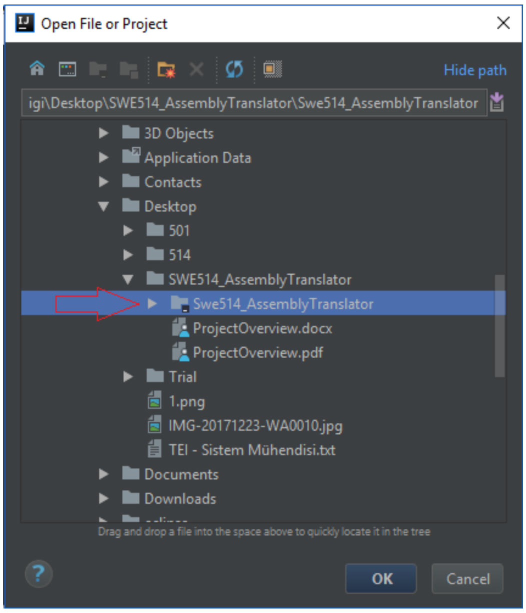


On run the program takes whatever on the Simple.in file of the project as input and prints out the translation to Simple.out file of the project.

To open the project file press open from the IntelliJ welcome screen.



Then select the project folder.



3. Overview of the Simple Language

Simple language is a imaginary programming language which has only the basic core statements such as identifying variables, some basic mathematical calculations, and if and while loops. The grammar for the Simple language is as follows:

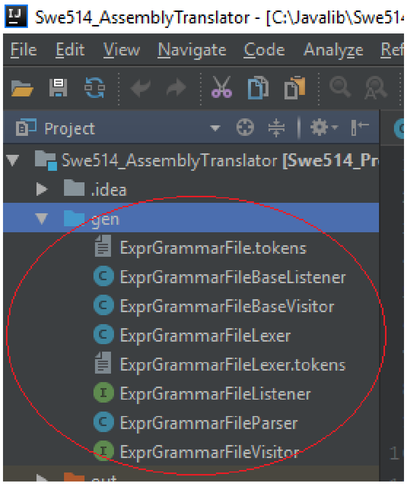
```
stm          →   id :=expr
                | print expr
                | if expr then stm
                | while expr do stm
                | begin opt_stmts end

opt_stms     →   stmt_list
                | ε

stmt_list    →   stm ; stmt_list
                | stm

expr         →   term moreterms

moreterms    →  + term moreterms
                | - term moreterms
                | ε

term         →   factor morefactors

morefactors  →   * factor morefactors
                | / factor morefactors
                | mod factor morefactors
                | ε

factor       →   ( expr )
                | id
                | num
```

We created to **“ExprGrammarFile.g4”** to teach our program how to apply grammetical rules. Grammar file can be seen below.

ProjectMain.java | Simple.out | EvalVisitor.java | ExprGrammarFile.g4 | Simple.in

```
grammar ExprGrammarFile;

//input syntax is defined, must include one of the below defined in stm
stm: 'begin' opt_stmts 'end'     #beginend
    |  ID ':=' expr              #assign
    | 'print' expr               #print
    | 'if' expr 'then' stm       #ifthen
    | 'while' expr 'do' stm      #whiledo
    ;

opt_stmts : stm_list
          ;

//endline is defined
stm_list : stm ';' stm_list
         | stm
         ;

//details of aritmetic operations are defined, #operation is used to define visitor met
expr: expr op = MUL expr         #mul
    | expr op = DIV expr         #div
    | expr op = MOD expr         #mod
    | expr op = ADD expr         #add
    | expr op = SUB expr         #sub
    | NUM                        #int
    | ID                         #id
    | '(' expr ')'               #parens
    ;

MUL : '*' ;
DIV : '/' ;
MOD : 'mod' ;
ADD : '+' ;
SUB : '-' ;

//token alternatives are define with skipping whitespaces, newline and carriage return
ID : ('a'..'z'|'A'..'Z'|'_') ('a'..'z'|'A'..'Z'|'0'..'9'|'_')*;
NUM : [0-9]+ ;
WS : (' ' | '\t' | '\r' | '\n' )+ -> skip ;
```

Then we created antlr4 classes for our project as seen below by antlr runtime library.



4.Implementation of the Simple to Assembly Code Translator

The Antlr parser simply takes the grammar as input and generates a parser using parse tree and creates **"ExprGrammerFileVisitor"** interface for further implementations of the statements of the given grammer. We basically extend ExprGrammerFileVisitor class as **"EvalVisitor"** to implement our codes. Using Antlr we can also determine the rules of defining variables and omit the whitespaces, tabs, newlines while generating the translation of the language.

The program does the following:

1st Step: Takes the Simple.in file written in Simle grammer rule as input,

2nd Step: Parse it into a parse tree using the parser generated by Antler,

3rd Step: Visit the parse tree using EvalVisitor and define the statements,

4th Step: Execute the statements i.e take the given codes and put the corresponding A86 assembly code to an array,

5th Step: Finally, print out the array to the Simple.out output file.

5. Test

You can find the attached A86 codes in the following pages for the given instructions.
*** Loop termination condition is added to instructions.

	Generated A86 Code and attached it	Runs correctly
add.s	yes	yes
cramer.s	yes	yes
dbloop.s	yes	yes
fibonacci.s	yes	yes
gcd.s	yes	yes
lcm.s	yes	yes
primes.s	yes	yes
swap.s	yes	yes

<table>
<tr><th colspan="2">ADD.S</th></tr>
<tr><td>

```
begin
  a := 3   ;
  b := 4   ;
  n := a + b ;
  print n
end
```

</td><td>

```
code segment
    PUSH 3
    POP AX
    MOV a, AX
    PUSH 4
    POP AX
    MOV b, AX
    PUSH a
    PUSH b
    POP CX
    POP AX
    ADD AX, CX
    PUSH AX
    MOV DX, 0
    POP AX
    MOV n, AX
    PUSH n
    POP AX
    CALL print
EXIT:
    INT 20
print:
    TEST AX, AX
    JNS positive
    PUSH AX
    MOV DX, '-'
    MOV AH, 02H
    INT 21H
    POP AX
    NEG AX
positive:
    MOV SI, 10d
    XOR DX, DX
    MOV CX, 0
nonzero:
    DIV SI
    ADD DX, 48d
    PUSH DX
    INC CX
    XOR DX, DX
    CMP AX, 0h
    JNE nonzero
printloop:
    POP DX
    MOV AH, 02h
    INT 21h
    DEC CX
    JNZ printloop
    MOV DX, ' '
    MOV AH, 02h
    MOV DX, 0
    INT 21h
    ret
DATA:
a DW 0
b DW 0
n DW 0
```

</td></tr>
</table>

CRAMER.S	

begin
  a := 1 ;
  b := 2 ;
  c := 3 ;
  d := 4 ;
  e := 1 ;
  f := 1 ;
 	
  x := (e * d - b * f) / (a * d - b * c) ;
  y := (a * f - e * c) / (a * d - b * c) ;
 	
  print x ;
  print y
end
	code segment
	    PUSH 1
	    POP AX
	    MOV a, AX
	    PUSH 2
	    POP AX
	    MOV b, AX
	    PUSH 3
	    POP AX
	    MOV c, AX
	    PUSH 4
	    POP AX
	    MOV d, AX
	    PUSH 1
	    POP AX
	    MOV e, AX
	    PUSH 1
	    POP AX
	    MOV f, AX
	    PUSH e
	    PUSH d
	    POP CX
	    POP AX
	    TEST AX, AX
	    JNS pehtg
	    NEG AX
	    NEG CX
	pehtg:
	    IMUL CX
	    PUSH AX
	    MOV DX, 0
	    PUSH b
	    PUSH f
	    POP CX
	    POP AX
	    TEST AX, AX
	    JNS tmkuw
	    NEG AX
	    NEG CX
	tmkuw:
	    IMUL CX
	    PUSH AX
	    MOV DX, 0
	    POP CX
	    POP AX
	    SUB AX, CX
	    PUSH AX
	    MOV DX, 0
	    PUSH a
	    PUSH d
	    POP CX
	    POP AX
	    TEST AX, AX
	    JNS xmxsm

	<pre> NEG AX
	    NEG CX
	xmxsm:
	    IMUL CX
	    PUSH AX
	    MOV DX, 0
	    PUSH b
	    PUSH c
	    POP CX
	    POP AX
	    TEST AX, AX
	    JNS uabyw
	    NEG AX
	    NEG CX
	uabyw:
	    IMUL CX
	    PUSH AX
	    MOV DX, 0
	    POP CX
	    POP AX
	    SUB AX, CX
	    PUSH AX
	    MOV DX, 0
	    POP CX
	    POP AX
	    TEST AX, AX
	    JNS lzykr
	    NEG AX
	    NEG CX
	lzykr:
	    IDIV CX
	    PUSH AX
	    MOV DX, 0
	    POP AX
	    MOV x, AX
	    PUSH a
	    PUSH f
	    POP CX
	    POP AX
	    TEST AX, AX
	    JNS uqvms
	    NEG AX
	    NEG CX
	uqvms:
	    IMUL CX
	    PUSH AX
	    MOV DX, 0
	    PUSH e
	    PUSH c
	    POP CX
	    POP AX
	    TEST AX, AX
	    JNS opmry
	    NEG AX
	    NEG CX</pre>

	opmry:
	    IMUL CX
	    PUSH AX
	    MOV DX, 0
	    POP CX
	    POP AX
	    SUB AX, CX
	    PUSH AX
	    MOV DX, 0
	    PUSH a
	    PUSH d
	    POP CX
	    POP AX
	    TEST AX, AX
	    JNS offkl
	    NEG AX
	    NEG CX
	offkl:
	    IMUL CX
	    PUSH AX
	    MOV DX, 0
	    PUSH b
	    PUSH c
	    POP CX
	    POP AX
	    TEST AX, AX
	    JNS atpnf
	    NEG AX
	    NEG CX
	atpnf:
	    IMUL CX
	    PUSH AX
	    MOV DX, 0
	    POP CX
	    POP AX
	    SUB AX, CX
	    PUSH AX
	    MOV DX, 0
	    POP CX
	    POP AX
	    TEST AX, AX
	    JNS lxqfo
	    NEG AX
	    NEG CX
	lxqfo:
	    IDIV CX
	    PUSH AX
	    MOV DX, 0
	    POP AX
	    MOV y, AX
	    PUSH x
	    POP AX
	    CALL print
	    PUSH y
	    POP AX

	<pre> CALL print
	EXIT:
	    INT 20
	print:
	    TEST AX, AX
	    JNS positive
	    PUSH AX
	    MOV DX, '-'
	    MOV AH, 02H
	    INT 21H
	    POP AX
	    NEG AX
	positive:
	    MOV SI, 10d
	    XOR DX, DX
	    MOV CX, 0
	nonzero:
	    DIV SI
	    ADD DX, 48d
	    PUSH DX
	    INC CX
	    XOR DX, DX
	    CMP AX, 0h
	    JNE nonzero
	printloop:
	    POP DX
	    MOV AH, 02h
	    INT 21h
	    DEC CX
	    JNZ printloop
	    MOV DX, ' '
	    MOV AH, 02h
	    MOV DX, 0
	    INT 21h
	    ret
	DATA:
	a DW 0
	b DW 0
	c DW 0
	d DW 0
	e DW 0
	f DW 0
	x DW 0
	y DW 0</pre>

DBLOOPS.S	

begin
  n := 3 ;
  m := 4 ;
  sum := 0 ;
  while(n) do begin
    while(m) do begin
      sum := sum + n + m;
      m := m - 1
    end;
    n := n - 1
  end;
  print sum
end
	code segment
	    PUSH 3
	    POP AX
	    MOV n, AX
	    PUSH 4
	    POP AX
	    MOV m, AX
	    PUSH 0
	    POP AX
	    MOV sum, AX
	kljjv:
	    PUSH n
	    POP AX
	    CMP AX, 0
	    JZ jdhlf
	uwlkx:
	    PUSH m
	    POP AX
	    CMP AX, 0
	    JZ jxfqr
	    PUSH sum
	    PUSH n
	    POP CX
	    POP AX
	    ADD AX, CX
	    PUSH AX
	    MOV DX, 0
	    PUSH m
	    POP CX
	    POP AX
	    ADD AX, CX
	    PUSH AX
	    MOV DX, 0
	    POP AX
	    MOV sum, AX
	    PUSH m
	    PUSH 1
	    POP CX
	    POP AX
	    SUB AX, CX
	    PUSH AX
	    MOV DX, 0
	    POP AX
	    MOV m, AX
	    JMP uwlkx
	jxfqr:
	    PUSH n
	    PUSH 1
	    POP CX

	<pre> POP AX
	    SUB AX, CX
	    PUSH AX
	    MOV DX, 0
	    POP AX
	    MOV n, AX
	    JMP kljjv
	jdhlf:
	    PUSH sum
	    POP AX
	    CALL print
	EXIT:
	    INT 20
	print:
	    TEST AX, AX
	    JNS positive
	    PUSH AX
	    MOV DX, '-'
	    MOV AH, 02H
	    INT 21H
	    POP AX
	    NEG AX
	positive:
	    MOV SI, 10d
	    XOR DX, DX
	    MOV CX, 0
	nonzero:
	    DIV SI
	    ADD DX, 48d
	    PUSH DX
	    INC CX
	    XOR DX, DX
	    CMP AX, 0h
	    JNE nonzero
	printloop:
	    POP DX
	    MOV AH, 02h
	    INT 21h
	    DEC CX
	    JNZ printloop
	    MOV DX, ' '
	    MOV AH, 02h
	    MOV DX, 0
	    INT 21h
	    ret
	DATA:
	sum DW 0
	m DW 0
	n DW 0</pre>

FIBONACCI.S	

begin
  n := 15 ;
  f0 := 0 ;
  print f0 ;
  f1 := 1 ;
  print f1 ;
  while (n) do begin
    fnew := f0 + f1 ;
    print fnew ;
    f0 := f1 ;
    f1 := fnew ;
    n := n - 1
   end
end
	code segment
	    PUSH 15
	    POP AX
	    MOV n, AX
	    PUSH 0
	    POP AX
	    MOV f0, AX
	    PUSH f0
	    POP AX
	    CALL print
	    PUSH 1
	    POP AX
	    MOV f1, AX
	    PUSH f1
	    POP AX
	    CALL print
	plrjm:
	    PUSH n
	    POP AX
	    CMP AX, 0
	    JZ yfzwq
	    PUSH f0
	    PUSH f1
	    POP CX
	    POP AX
	    ADD AX, CX
	    PUSH AX
	    MOV DX, 0
	    POP AX
	    MOV fnew, AX
	    PUSH fnew
	    POP AX
	    CALL print
	    PUSH f1
	    POP AX
	    MOV f0, AX
	    PUSH fnew
	    POP AX
	    MOV f1, AX
	    PUSH n
	    PUSH 1
	    POP CX
	    POP AX
	    SUB AX, CX
	    PUSH AX
	    MOV DX, 0
	    POP AX
	    MOV n, AX
	    JMP plrjm

```
yfzwq:
EXIT:
    INT 20
print:
    TEST AX, AX
    JNS positive
    PUSH AX
    MOV DX, '-'
    MOV AH, 02H
    INT 21H
    POP AX
    NEG AX
positive:
    MOV SI, 10d
    XOR DX, DX
    MOV CX, 0
nonzero:
    DIV SI
    ADD DX, 48d
    PUSH DX
    INC CX
    XOR DX, DX
    CMP AX, 0h
    JNE nonzero
printloop:
    POP DX
    MOV AH, 02h
    INT 21h
    DEC CX
    JNZ printloop
    MOV DX, ' '
    MOV AH, 02h
    MOV DX, 0
    INT 21h
    ret
DATA:
f0 DW 0
fnew DW 0
f1 DW 0
n DW 0
```

GCD.S

```
begin
  a := 555 ;
  b := 115 ;
  while (b) do begin
       t := b ;
       b := a mod b ;
       a := t
  end ;
  print a
end
```

```
code segment
    PUSH 555
    POP AX
    MOV a, AX
    PUSH 115
    POP AX
    MOV b, AX
qehrp:
    PUSH b
    POP AX
    CMP AX, 0
    JZ jbsgf
    PUSH b
    POP AX
    MOV t, AX
    PUSH a
    PUSH b
    POP CX
    POP AX
    DIV CX
    PUSH DX
    MOV DX, 0
    POP AX
    MOV b, AX
    PUSH t
    POP AX
    MOV a, AX
    JMP qehrp
jbsgf:
    PUSH a
    POP AX
    CALL print
EXIT:
    INT 20
print:
    TEST AX, AX
    JNS positive
    PUSH AX
    MOV DX, '-'
    MOV AH, 02H
    INT 21H
    POP AX
    NEG AX
positive:
    MOV SI, 10d
    XOR DX, DX
    MOV CX, 0
nonzero:
    DIV SI
```

	ADD DX, 48d
	PUSH DX
	INC CX
	XOR DX, DX
	CMP AX, 0h
	JNE nonzero
	printloop:
	POP DX
	MOV AH, 02h
	INT 21h
	DEC CX
	JNZ printloop
	MOV DX, ' '
	MOV AH, 02h
	MOV DX, 0
	INT 21h
	ret
	DATA:
	a DW 0
	b DW 0
	t DW 0

LCM.S	

begin
  a := 5 ;
  b := 17 ;
  aa := a ;
  bb := b ;
  while (b) do begin
    t := b ;
    b := a mod b ;
    a := t
  end ;
  gcd := a ;
  lcm := (aa*bb) / gcd ;
  print lcm
end
	code segment
	  PUSH 5
	  POP AX
	  MOV a, AX
	  PUSH 17
	  POP AX
	  MOV b, AX
	  PUSH a
	  POP AX
	  MOV aa, AX
	  PUSH b
	  POP AX
	  MOV bb, AX
	jpkiw:
	  PUSH b
	  POP AX
	  CMP AX, 0
	  JZ qhkln
	  PUSH b
	  POP AX
	  MOV t, AX
	  PUSH a
	  PUSH b
	  POP CX
	  POP AX
	  DIV CX
	  PUSH DX
	  MOV DX, 0
	  POP AX
	  MOV b, AX
	  PUSH t
	  POP AX
	  MOV a, AX
	  JMP jpkiw
	qhkln:
	  PUSH a
	  POP AX
	  MOV gcd, AX
	  PUSH aa
	  PUSH bb
	  POP CX
	  POP AX
	  TEST AX, AX
	  JNS dcjmh
	  NEG AX
	  NEG CX
	dcjmh:
	  IMUL CX
	  PUSH AX
	  MOV DX, 0
	  PUSH gcd
	  POP CX
	  POP AX
	  TEST AX, AX

	<pre> JNS ihbom
	    NEG AX
	    NEG CX
	ihbom:
	    IDIV CX
	    PUSH AX
	    MOV DX, 0
	    POP AX
	    MOV lcm, AX
	    PUSH lcm
	    POP AX
	    CALL print
	EXIT:
	    INT 20
	print:
	    TEST AX, AX
	    JNS positive
	    PUSH AX
	    MOV DX, '-'
	    MOV AH, 02H
	    INT 21H
	    POP AX
	    NEG AX
	positive:
	    MOV SI, 10d
	    XOR DX, DX
	    MOV CX, 0
	nonzero:
	    DIV SI
	    ADD DX, 48d
	    PUSH DX
	    INC CX
	    XOR DX, DX
	    CMP AX, 0h
	    JNE nonzero
	printloop:
	    POP DX
	    MOV AH, 02h
	    INT 21h
	    DEC CX
	    JNZ printloop
	    MOV DX, ' '
	    MOV AH, 02h
	    MOV DX, 0
	    INT 21h
	    ret
	DATA:
	aa DW 0
	bb DW 0
	a DW 0
	b DW 0
	t DW 0
	lcm DW 0
	gcd DW 0</pre>

PRIMES.S	

begin
  i := 10 ;
 	
  aprev := 7 ;
  n := 1 ;
  while (i) do begin
    n := n + 1 ;
    k := n ;
    m := aprev ;
    while (m) do begin
      t := m ;
      m := k mod m ;
      k := t
    end ;
    anew := aprev + k ;
    i := i - 1 ;
    if (anew - aprev - 1) then
        print (anew-aprev) ;
    aprev := anew
  end
end
	code segment
	    PUSH 10
	    POP AX
	    MOV i, AX
	    PUSH 7
	    POP AX
	    MOV aprev, AX
	    PUSH 1
	    POP AX
	    MOV n, AX
	bnaut:
	    PUSH i
	    POP AX
	    CMP AX, 0
	    JZ sxgev
	    PUSH n
	    PUSH 1
	    POP CX
	    POP AX
	    ADD AX, CX
	    PUSH AX
	    MOV DX, 0
	    POP AX
	    MOV n, AX
	    PUSH n
	    POP AX
	    MOV k, AX
	    PUSH aprev
	    POP AX
	    MOV m, AX
	tuuru:
	    PUSH m
	    POP AX
	    CMP AX, 0
	    JZ pppmn
	    PUSH m
	    POP AX
	    MOV t, AX
	    PUSH k
	    PUSH m
	    POP CX
	    POP AX
	    DIV CX
	    PUSH DX
	    MOV DX, 0
	    POP AX
	    MOV m, AX
	    PUSH t

	<pre> POP AX
	    MOV k, AX
	    JMP tuuru
	pppmn:
	    PUSH aprev
	    PUSH k
	    POP CX
	    POP AX
	    ADD AX, CX
	    PUSH AX
	    MOV DX, 0
	    POP AX
	    MOV anew, AX
	    PUSH i
	    PUSH 1
	    POP CX
	    POP AX
	    SUB AX, CX
	    PUSH AX
	    MOV DX, 0
	    POP AX
	    MOV i, AX
	    PUSH anew
	    PUSH aprev
	    POP CX
	    POP AX
	    SUB AX, CX
	    PUSH AX
	    MOV DX, 0
	    PUSH 1
	    POP CX
	    POP AX
	    SUB AX, CX
	    PUSH AX
	    MOV DX, 0
	    POP AX
	    CMP AX, 0
	    JZ yhrvz
	    PUSH anew
	    PUSH aprev
	    POP CX
	    POP AX
	    SUB AX, CX
	    PUSH AX
	    MOV DX, 0
	    POP AX
	    CALL print
	yhrvz:
	    PUSH anew
	    POP AX</pre>

	(code below)

```
    MOV aprev, AX
    JMP bnaut
sxgev:
EXIT:
    INT 20
print:
    TEST AX, AX
    JNS positive
    PUSH AX
    MOV DX, '-'
    MOV AH, 02H
    INT 21H
    POP AX
    NEG AX
positive:
    MOV SI, 10d
    XOR DX, DX
    MOV CX, 0
nonzero:
    DIV SI
    ADD DX, 48d
    PUSH DX
    INC CX
    XOR DX, DX
    CMP AX, 0h
    JNE nonzero
printloop:
    POP DX
    MOV AH, 02h
    INT 21h
    DEC CX
    JNZ printloop
    MOV DX, ' '
    MOV AH, 02h
    MOV DX, 0
    INT 21h
    ret
DATA:
anew DW 0
t DW 0
i DW 0
k DW 0
m DW 0
n DW 0
aprev DW 0
```

<table>
<tr><th colspan="2">SWAP.S</th></tr>
<tr><td>

```
begin
  a := 3  ;
  b := 4  ;

  a := a + b ;
  b := a - b ;
  a := a - b ;
  print a ;
  print b
end
```

</td><td>

```
code segment
    PUSH 3
    POP AX
    MOV a, AX
    PUSH 4
    POP AX
    MOV b, AX
    PUSH a
    PUSH b
    POP CX
    POP AX
    ADD AX, CX
    PUSH AX
    MOV DX, 0
    POP AX
    MOV a, AX
    PUSH a
    PUSH b
    POP CX
    POP AX
    SUB AX, CX
    PUSH AX
    MOV DX, 0
    POP AX
    MOV b, AX
    PUSH a
    PUSH b
    POP CX
    POP AX
    SUB AX, CX
    PUSH AX
    MOV DX, 0
    POP AX
    MOV a, AX
    PUSH a
    POP AX
    CALL print
    PUSH b
    POP AX
    CALL print
EXIT:
    INT 20
print:
    TEST AX, AX
    JNS positive
    PUSH AX
    MOV DX, '-'
    MOV AH, 02H
    INT 21H
```

</td></tr>
</table>

	<pre> POP AX
	    NEG AX
	positive:
	    MOV SI, 10d
	    XOR DX, DX
	    MOV CX, 0
	nonzero:
	    DIV SI
	    ADD DX, 48d
	    PUSH DX
	    INC CX
	    XOR DX, DX
	    CMP AX, 0h
	    JNE nonzero
	printloop:
	    POP DX
	    MOV AH, 02h
	    INT 21h
	    DEC CX
	    JNZ printloop
	    MOV DX, ' '
	    MOV AH, 02h
	    MOV DX, 0
	    INT 21h
	    ret
	DATA:
	a DW 0
	b DW 0</pre>

6. Conclusion

In this study we reach our desired goal to accurutely translate the imaginary simple language to A86 Assembly language. Altough the Simple language has very few core statements of a standart programming language, the project enlightens the way of building compilers for further assignments.

For further studies the researchers can be challenged to implement some other core statements such as select…case statements, defining and using functions with and/or without parameters.

7. References

1. *PC Mag Staff (28 February 2017). "Encyclopedia: Definition of Compiler". PCMag.com. Retrieved 28 February 2017.*