# **Школски асемблер**

Данило Лалић

0501/2017

Садржај

[**Школски асемблер** 1](#_Toc51604491)

[1. Увод 3](#_Toc51604492)

[2. Опис решења 3](#_Toc51604493)

[2.1. Класа „SyntaxAnalyzer“ 3](#_Toc51604494)

[2.2. Класа „InstructionCoder“ 3](#_Toc51604495)

[2.3. Класа „DirectiveHandler“ 3](#_Toc51604496)

[2.4. Класа „SymbolHandler“ 3](#_Toc51604497)

[2.5. Класа „FileWriter“ 3](#_Toc51604498)

[3. Синтаксна правила 4](#_Toc51604499)

[3.1. Секције 4](#_Toc51604500)

[3.2. Скокови и адресирања 4](#_Toc51604501)

[3.3. Изрази 4](#_Toc51604502)

[4. Начин коришћења 4](#_Toc51604503)

[5. Тест примери 4](#_Toc51604504)

[5.1. Први пример 4](#_Toc51604505)

[5.2. Други пример 8](#_Toc51604506)

[5.3. Трећи пример 11](#_Toc51604507)

[5.4. Четврти пример 15](#_Toc51604508)

## 1. Увод

Овим пројектом је реализован једнопролазни асемблер за школски емулатор. Улаз асемблера је текстуална датотека која садржи асемблерски код, док је излаз текстуална датотека у школском ЕЛФ формату.

Пројекат је реализован уз помоћ лексера Флекс и програмског језика Ц++.

## 2. Опис решења

У наставку је по класама дато објашњење начина на који је асемблер реализован.

### 2.1. Класа „SyntaxAnalyzer“

Функције ове класе позивају се из функције „yylex“ која је задужена за распознавање токена у улазној датотеци и њихово прослеђивање овој класи. Ова класа је задужена за проверу синтаксне исправности кода, обраду токена и њихово смештање у одговарајућу структуру и слање тих структура на даљу обраду.

Подаци неопходи за обраду инструкција и директива се смештају у структуре „Instruction“ и „Directive“. Структуре се не шаљу на даљу обраду све док се не наиђе на карактер за нови ред, крај датотеке или „.end“ директиве.

### 2.2. Класа „InstructionCoder“

Након што од класе „SyntaxAnalyzer“ прими структуру „Instruction“ врши обраду инструкције тако што генерише низ бајтова које затим шаље „FileWriter“-у.

### 2.3. Класа „DirectiveHandler“

Исто као и претходна класа само што уместо „Instruction“ структуре прима „Directive“ структуру.

### 2.4. Класа „SymbolHandler“

„SymbolHandler“ је задужен за обраду симбола и одржавање структура које су везане за њих(табела симбола, табела обраћања унапред, табела релокација...). Позивањем функција ове класе „InstructionCoder“ и „DirectiveHandler“ сазнају вредности симбола које треба да уграде у садржај.

Поред обраде симбола у овој класи је реализован и алгоритам „backpatching“.

### 2.5. Класа „FileWriter“

„FileWriter“ прихвата садржај и током „backpatching“-а прима команде за његову дораду. Након завршетка „backpatching“-а од „SymbolHandler“-а прима све потребне табеле и генерише излазну датотеку.

## 3. Синтаксна правила

Следе синтаксна правила која су поред оних датих у опису задатка додата.

### 3.1. Секције

Било коју инструкцију или директиву је могуће написати само у једној од три секције. Те секције су: .data, .text и .bss. У .bss секцији није могуће користити инструкције нити било коју директиву која управља садржајем. У .data секцији није могуће користити инструкције.

### 3.2. Скокови и адресирања

Код инструкција скока релативно адресирање у односу на програмски бројач није подразумевано. Ако се такво адресирање жели мора се експлицитно навести операнд у том облику.

### 3.3. Изрази

Између операнада и операција мора се налазити размак или хоризонтална табулација.

## 4. Начин коришћења

Програм се асемблира следећом командом: ass input.txt -o output.txt. Први аргумент је име улазног фајла док је трећи аргумент име излазног фајла. У случају неуспешног асемблирања порука о грешци биће исписана на стандардном излазу.

## 5. Тест примери

### 5.1. Први пример

Улаз:

.section .text

.extern arr

.extern size

.global min

min: MOV $arr, %r1

MOV size, %r2

MOV (%r1), %r0

loop:SUB $1, %r2

CMP $0, %r2

JEQ fin

ADD $2, %r1

MOV (%r1), %r4

CMP %r0, %r4

JGT \*jm(%pc)

MOV (%r1), %r0

jm: JMP loop

fin: RET

.end

Излаз:

Magic: 7f 45 4c 46

Class: ELF16

Data: little-endian

Version: 1

Type: Relocatable

Machine: ETF16

Version: 1

Entry point address: 0x0

Start of program headers: 0x0

Start of section headers: 64

Flags: 0x0

Size of this header: 64

Size of program headers: 0

Number of program headers: 0

Size of section headers: 40

Number of section headers: 6

Section header string table index:0

Section headers:

Name Type Address Offset

Size EntSize Flags Link Info Align

0 NULL 0 0

0 0 0 0 0

.text PROGBITS 0 0

50 0 AX 0 0 1

.rel.text REL 0 0

64 16 I 5 1 1

.data PROGBITS 0 0

0 0 WA 0 0 1

.bss PROGBITS 0 SHT\_NOBITS

0 0 WA 0 0 1

.symtab SYMTAB 0 0

144 18 0 0 1

Sections:

.text

6400 0000 2264 8000 0024

6442 2074 0001 0024 8c00

0000 2434 0031 006c 0002

0022 6442 288c 2028 446e

0300 6442 202c 000d 0014

.data

Symbol table:

Name Addr Size T L/G Section

UND 0 0 0 0 UND

.text 0 50 0 0 .text

arr 0 0 1 2 UND

size 0 0 1 2 UND

min 0 0 1 1 .text

loop 13 0 1 0 .text

fin 49 0 1 0 .text

jm 45 0 1 0 .text

Relocations tables

Section Offset Type Symbol name

.rel.text

.text 2 0 arr

.text 7 0 size

.text 47 0 .text

.text 25 0 .text

.rel.data

Section Offset Type Symbol name Addend

.rela.text

.rela.data

### 5.2. Други пример

Улаз:

.section .text

.extern read

.extern write

.global readWrite

readWrite: CALL \*read(%pc)

ADD $4, %r0

CALL write

RET

.section .data

d1: .word d2

d2: .word b1

d3: .word d3

.section .bss

.skip 5

b1:

.end

Излаз:

Magic: 7f 45 4c 46

Class: ELF16

Data: little-endian

Version: 1

Type: Relocatable

Machine: ETF16

Version: 1

Entry point address: 0x0

Start of program headers: 0x0

Start of section headers: 64

Flags: 0x0

Size of this header: 64

Size of program headers: 0

Number of program headers: 0

Size of section headers: 40

Number of section headers: 8

Section header string table index:0

Section headers:

Name Type Address Offset

Size EntSize Flags Link Info Align

0 NULL 0 0

0 0 0 0 0

.text PROGBITS 0 0

14 0 AX 0 0 1

.rel.text REL 0 0

32 16 I 6 1 1

.data PROGBITS 0 0

6 0 WA 0 0 1

.rel.data REL 0 0

48 16 I 6 3 1

.bss PROGBITS 0 SHT\_NOBITS

5 0 WA 0 0 1

.symtab SYMTAB 0 0

198 18 0 0 1

Sections:

.text

246e feff 6c00 0400 2024

0000 0014

.data

0200 0500 0400

Symbol table:

Name Addr Size T L/G Section

UND 0 0 0 0 UND

.text 0 14 0 0 .text

.data 0 6 0 0 .data

.bss 0 5 0 0 .bss

read 0 0 1 2 UND

write 0 0 1 2 UND

readWrite 0 0 1 1 .text

d1 0 0 1 0 .data

d2 2 0 1 0 .data

b1 5 0 1 0 .bss

d3 4 0 1 0 .data

Relocations tables

Section Offset Type Symbol name

.rel.text

.text 2 1 read

.text 11 0 write

.rel.data

.data 0 0 .data

.data 2 0 .bss

.data 4 0 .data

Section Offset Type Symbol name Addend

.rela.text

.rela.data

### 5.3. Трећи пример

.section .text

.global sum

.equ size, ve - vs

sum: MOV $size, %r1

MOV vs(%pc), %r2

MOV (%r2), %r0

loop: SUB $1, %r1

CMP $0, %r1

JEQ fin

ADD $2, %r2

ADD (%r2), %r0

JMP loop

fin: MOV %r0, store

RET

.section .data

vs: .word 1, 2, 3, 4, 5

ve:

.section .bss

.skip 3

store: .skip 2

.end

Излаз:

Magic: 7f 45 4c 46

Class: ELF16

Data: little-endian

Version: 1

Type: Relocatable

Machine: ETF16

Version: 1

Entry point address: 0x0

Start of program headers: 0x0

Start of section headers: 64

Flags: 0x0

Size of this header: 64

Size of program headers: 0

Number of program headers: 0

Size of section headers: 40

Number of section headers: 6

Section header string table index:0

Section headers:

Name Type Address Offset

Size EntSize Flags Link Info Align

0 NULL 0 0

0 0 0 0 0

.text PROGBITS 0 0

45 0 AX 0 0 1

.rel.text REL 0 0

64 16 I 5 1 1

.data PROGBITS 0 0

10 0 WA 0 0 1

.bss PROGBITS 0 SHT\_NOBITS

5 0 WA 0 0 1

.symtab SYMTAB 0 0

198 18 0 0 1

Sections:

.text

6400 0a00 2264 6efe ff24

6444 2074 0001 0022 8c00

0000 2234 0027 006c 0002

0024 6c44 202c 000d 0064

2080 0300 14

.data

0100 0200 0300 0400 0500

Symbol table:

Name Addr Size T L/G Section

UND 0 0 0 0 UND

.text 0 45 0 0 .text

.data 0 10 0 0 .data

.bss 0 5 0 0 .bss

sum 0 0 1 1 .text

size 10 0 1 0 ABS

vs 0 0 1 0 .data

loop 13 0 1 0 .text

fin 39 0 1 0 .text

store 3 0 1 0 .bss

ve 10 0 1 0 .data

Relocations tables

Section Offset Type Symbol name

.rel.text

.text 7 1 .data

.text 37 0 .text

.text 25 0 .text

.text 42 0 .bss

.rel.data

Section Offset Type Symbol name Addend

.rela.text

.rela.data

### 5.4. Четврти пример

Улаз:

.section .text

.extern e

.global d, f

.equ c, b - s3 + e

.equ b, a + 1 + s4

.equ a, 5 + s2 - s1

.equ d, 5

.equ f, b

JMP \*s1(%pc)

s1: JMP \*s2(%pc)

s2: JMP \*s1(%pc)

MOV $a, %r0

MOV $b, %r0

MOV $f, %r0

MOV $d, %r0

MOV $c, %r0

MOVb $5, %r0h

MOV c(%pc), %r0

MOV f(%pc), %r0

.section .data

.word 3

s3: .byte 1

s4: .byte 2

.end

Излаз:

Magic: 7f 45 4c 46

Class: ELF16

Data: little-endian

Version: 1

Type: Relocatable

Machine: ETF16

Version: 1

Entry point address: 0x0

Start of program headers: 0x0

Start of section headers: 64

Flags: 0x0

Size of this header: 64

Size of program headers: 0

Number of program headers: 0

Size of section headers: 40

Number of section headers: 7

Section header string table index:0

Section headers:

Name Type Address Offset

Size EntSize Flags Link Info Align

0 NULL 0 0

0 0 0 0 0

.text PROGBITS 0 0

51 0 AX 0 0 1

.rel.text REL 0 0

64 16 I 6 1 1

.rela.text RELA 0 0

40 20 I 6 1 1

.data PROGBITS 0 0

4 0 WA 0 0 1

.bss PROGBITS 0 SHT\_NOBITS

0 0 WA 0 0 1

.symtab SYMTAB 0 0

234 18 0 0 1

Sections:

.text

2c6e 0000 2c6e 0000 2c6e

f8ff 6400 0900 2064 000d

0020 6400 0000 2064 0000

0020 6400 0b00 2060 0005

2164 6e09 0020 646e feff

20

.data

0300 0102

Symbol table:

Name Addr Size T L/G Section

UND 0 0 0 0 UND

.text 0 51 0 0 .text

.data 0 4 0 0 .data

e 0 0 1 2 UND

d 5 0 1 1 ABS

f 13 0 1 1 .data

s1 4 0 1 0 .text

s2 8 0 1 0 .text

a 9 0 1 0 ABS

b 13 0 1 0 .data

c 11 0 1 0 UND

s3 2 0 1 0 .data

s4 3 0 1 0 .data

Relocations tables

Section Offset Type Symbol name

.rel.text

.text 43 1 e

.text 34 0 e

.text 29 0 d

.text 19 0 .data

.rel.data

Section Offset Type Symbol name Addend

.rela.text

.text 48 1 .data 13

.text 24 0 .data 13

.rela.data