Опис проблема

Потребно је направити двопролазни асемблер, који се ослања на принципима GNU асемблера. Формат улазних и излазних фајлова се базира на структури ,, школског ” ELF формата.

Овај асемблер се ослања на 16-битни двоадресни процесор са Von-Neuman архитектуром. Структура и формат инстрикција и опис архитектуре процесора су описани у тексту пројектног задатка.

Асемблер ради под оперативним системом Linux.

Решење проблема

Решење за имлементацију оваквог асемблера се састоји из два пролаза, читајући са улаза ред по ред, јер је по једна инструкција у сваком реду.

У **првом пролазу** се пролази кроз код, попуњава се **табела симбола** (локалним симболима), са секцијама којима припадају и вредностима – на којој адресу у секцији се налазе. Попуњава се истовремено и **табела сегмената** са подацима о секцијама - где почињу и колика је њихова величина.

У **другом пролазу** се разрешавају симболи у директивама и инструкцијама, решава увожење и извожење симбола, праве релокациони записи за линкер и кодују инструкције машинским кодом. Попуњавају се: **табеле реколација** и **садржаји секција**.

Дат је опис имплементације по целинама.

1. Секције

Секције почињу тачком и у једној линији може стајати само назив секције. Постоје следеће секције:

.text,

.data,

.rodata,

.bss

Операције иду само у .text секцији, а подаци у осталим секцијама. У супротом асемблер пријављује грешку.

1. Директиве

Постоје следеће директиве:

.align

.skip

.char

.word

.long

Директива .global мора стајати на почетку програма. Означава увожење или извожење симбола, зависно од тога да ли је дати симбол дефинисан у програму.

1. Лабеле

Лабеле обично стоје на почетку новог реда. Могу да садрже мала и велика слова, бројеве и \_ . На крају назива лабеле стоји двотачка (:). Када асемблер наиђе на лабелу, дода је у табели симбола и настави обраду са остатком реда. Грешка је ако се наведе лабела испред почетка секције.

1. Инструкције и адрeсирања

Скуп инструкција и начини адресирања су описани у тексту пројекта, као и условна извршавања инструкција. Зависно од начина адресира, величина инструкције може бити 2B или 4B.

Подражани начини адресирања су непосредно, регистарско, меморијско и регистарско индиректно са померајем.

Код jmp инструкције могућа адресирања су:

* мемориjско: jmp &labela
* PC релативно: jmp $labela

Бар један од операнада мора бити регистар код инструкција са 2 аргумената.

1. Коментари

Коментари садрже # на почетку реда. Грешка је ако је # после инструкције или директиве.

1. Грешке

Уколико дође до грешке приликом превођења, грешка ће се исписати на терминалу, као и линија на којој је дошло до грешке, али и пролаз у ком је дошло до грешке.

Уколико нема грешака приликом превођења, на терминалу ће се исписати порука:

First pass...

First pass done.

Second pass...

Second pass done.

Превођење и покретање програма

Превођење се обавља GCC компајлером комадом у терминалу:

g++ \*.cpp –o naziv\_programa

Покретање програма се обавља командом:

./naziv\_programa input.txt output.txt 1000

Где је input.txt назив улазног фајла, output.txt назив улазног фајла и 1000 адреса на којој смештен програм.

Команде за инсталацију GCC компајлера верзије 6 су:

sudo apt-get update && \

sudo apt-get install build-essential software-properties-common -y && \

sudo add-apt-repository ppa:ubuntu-toolchain-r/test -y && \

sudo apt-get update && \

sudo apt-get install gcc-snapshot -y && \

sudo apt-get update && \

sudo apt-get install gcc-6 g++-6 -y && \

sudo update-alternatives --install /usr/bin/gcc gcc /usr/bin/gcc-6 60 --slave /usr/bin/g++ g++ /usr/bin/g++-6 && \

sudo apt-get install gcc-4.8 g++-4.8 -y && \

sudo update-alternatives --install /usr/bin/gcc gcc /usr/bin/gcc-4.8 60 --slave /usr/bin/g++ g++ /usr/bin/g++-4.8;

или

sudo apt-get install g++-6

Примери превођења

Пример 1

Улазни фајл:

.global loop, printc

.data

array: .char loop, 1, -1, 2

array\_end:

.word end, 5, array

.text

\_start:

mov r3, &array

mov r4, &array\_end

loop:

cmp r4, r3

jmpeq &end

mov r5, r3[0]

call &loop

add r3, 1

jmp $loop

end:

ret

.rodata

.skip 7

.align 3

.long loop, 25, 20

.end

Излазни фајл:

# Simbols' table

Name Section Value Vis Num

.data DATA 0 local 0 1

array DATA 0 local 1 1

array\_end DATA 4 local 2 1

.text TEXT 0 local 3 1

\_start TEXT 0 local 4 1

loop TEXT 8 global 5 1

end TEXT 30 local 6 1

.rodata RODATA 0 local 7 1

printc UND 0 global 8 0

# Segments' table

Name Base Size

.data 1000 10

.text 1010 32

.rodata 1042 20

# .rel .text

Offset Type Ref

2 PC\_ABS 0

6 PC\_ABS 0

# .rel .data

Offset Type Ref

0 PC\_ABS 3

4 PC\_ABS 3

# .rel .rodata

Offset Type Ref

8 PC\_ABS 3

# .text section

f5 60 00 00 f5 80 04 00

d1 8b 35 e0 1e 00 f5 bb

00 00 ec 00 08 00 c1 60

01 00 c1 f0 08 00 e9 e0

# .data section

08 01 ff 02 1e 00 05 00

00 00

# .rodata section

00 00 00 00 00 00 00 00

08 00 00 00 19 00 00 00

14 00 00 00

Пример 2

Улазни фајл:

.global \_start, niz, temp1, temp2, printc, printi

.data

temp: .word 3, 2,temp1, 5, temp2, tem

tem:

.long -12, temp2, loop, niz

.bss

niz:

.word 1,-2,3,-4,5

.skip 10

.text

\_start:

add r1, r2

jmp $loop

jmpne &loop

popne r1

subgt r1, r2

mov r0, 65

movgt r3, loop

call &printc

push r1

popgt r1[5]

loop: cmp r2, r1

moveq r7, &cont

mov r3, r2[niz]

addgt r0, r3

incr: add r2, 2

jmp &loop

cont: call &printi

ret

.end

Излазни фајл:

# Simbols' table

Name Section Value Vis Num

.data DATA 0 local 0 1

temp DATA 0 local 1 1

tem DATA 12 local 2 1

.bss BSS 0 local 3 1

niz BSS 0 global 4 1

.text TEXT 0 local 5 1

\_start TEXT 0 global 6 1

loop TEXT 32 local 7 1

incr TEXT 44 local 8 1

cont TEXT 52 local 9 1

temp1 UND 0 global 10 0

temp2 UND 0 global 11 0

printc UND 0 global 12 0

printi UND 0 global 13 0

# Segments' table

Name Base Size

.data 1000 28

.bss 1028 20

.text 1048 58

# .rel .text

Offset Type Ref

18 PC\_ABS 12

28 PC\_ABS 3

36 PC\_ABS 13

# .rel .data

Offset Type Ref

4 PC\_ABS 10

8 PC\_ABS 11

10 PC\_ABS 11

14 PC\_ABS 5

18 PC\_ABS 3

# .text section

c1 2a c1 f0 20 00 75 e0

20 00 69 20 85 2a f5 00

41 00 b5 70 20 00 ec 00

fe ff e5 20 ab 20 05 00

d1 49 35 e0 34 00 f5 7a

00 00 81 0b c1 40 02 00

f5 e0 20 00 ec 00 fe ff

e9 e0

# .data section

03 00 02 00 00 00 05 00

00 00 0c 00 f4 ff ff ff

00 00 00 00 20 00 00 00

00 00 00 00

# .bss section

00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00

00 00 00

Пример 3

Улазни фајл:

.global \_start, printi

.data

head:

.word 12

.word node1

node1:

.word 17

.word node2

node2:

.word -2

.word head

.text

\_start:

mov r1, &head

loop: cmp r1, 0

mov r0, r1[0]

mov r1, r1[2]

call &printi

jmp $loop

kraj:

ret

.end

Излазни фајл:

# Simbols' table

Name Section Value Vis Num

.data DATA 0 local 0 1

head DATA 0 local 1 1

node1 DATA 4 local 2 1

node2 DATA 8 local 3 1

.text TEXT 0 local 4 1

\_start TEXT 0 global 5 1

loop TEXT 4 local 6 1

kraj TEXT 24 local 7 1

printi UND 0 global 8 0

# Segments' table

Name Base Size

.data 1000 12

.text 1012 26

# .rel .text

Offset Type Ref

2 PC\_ABS 0

12 PC\_ABS 8

# .text section

f5 20 00 00 d1 20 00 00

f5 19 00 00 f5 39 02 00

ec 00 fe ff c1 f0 04 00

e9 e0

# .data section

0c 00 04 00 11 00 08 00

fe ff 00 00