Универзитет у Београду

Електротехнички факултет



Интерпретативни емулатор

Системски софтвер

Страхиња Стефановић

2016/0130

Београд, јун 2019.

# Опис проблема

У овом делу пројекта задатак је направити линкер/пунилац који ће објектне фајлове настале као производ асемблирања спојити у један извршни фајл и тај фајл проследити емулатору. Са друге стране, треба направити и емулатор чији је задатак да тај програм емулира на замишљеном процесору дефинисаном у прилогу пројекта. Осим извршавања програма, емулатор треба да обезбеди и прекиде и интеракцију са тајмером и терминалом.

# Упутство за превођење и покретање

За превођење програма користи се makefile скрипта:

make ./bin/emulator

која генерише извршни фајл у одговарајућем директоријуму.

Сам програм потребно је покренути на следећи начин:

./bin/emulator –place=<section>@<address> [<input\_file>]+

Аргументе је могуће наводити произвољним редоследом, не морају прво ићи –place опције. Уколико се не последи бар један објектни фајл (фајл са екстензијом .о) или се проследе не одговарајући аргументи линкер ће пријавити грешку. По покретању програма, кориснику је дата могућност да интерагује са процесором путем терминала.

За извршавање датог тест програма потребно је покренути скрипту test.sh.

# Опис решења

По покретању програма позива се линкер чији је задатак да из датих објектних фајлова извуче потребне информације и, потом, (1) разреши симболе, односно сваки симбол повеже са једном дефиницијом и (2) изврши релокацију секција по упуству дато позивом програма и затим модификује све референце на симболе дате релокационим записима и упише њихову праву вредност. Смештање табеле симбола, табеле секција и релокационих записа и њихово касније читање извршено је методама серијализације, односно десеријализације. Те методе је лакше било имплементирати кроз бинарне датотеке па су зато и оне генерисане као излаз асемблера. Линкер прво пролази кроз све симболе и гледа само секције. Додаје их у своју интерну табелу секција тако да истоимене секције бивају спојене у једну јединствену секцију са тим именом. Након тога, разрешава недефинисане симболе и ако се деси да су дефинисана два глобална симбола са истим именом или остаје неки симбол који није нигде дефинисан, пријављује фаталну грешку. Такође, доћи ће до грешке и ако није дефинисан глобални симбол \_start који представља улазну тачку програма. Задатак самог линкера је само да ажурира релокације у односу на померене секције. Међутим, у овој реализацији он извршава и улогу пуниоца тако да ће он и премесити секције на крајње локације и коначно уписати праве вредности симбола на свим местима где се реферишу. Ако је успео да угради све секције у меморију без преклапања, као резултат добија се садржај меморије коју емулатор треба да интерпретира.

Уколико се не дефинише iv\_table секција, линкер ће пријавити грешку, а уколико се не наведе њена стартна адреса, подразумевано почиње на адреси 0x0000 (као и у самој поставци).

Емулатор своје извршавање почиње из прекидне рутине за иницијализацију (улаз 0 IV табеле) где се систем поставља у коректно почетно стање и на крају рутине се у pc регистар уписује вредност симбола \_start.

Даље се извршавање наствља стандардном процедуром: дохватање инструкције, читање операнада и њено извршавање и, на крају, обрада прекида. Тајмер и терминал су реализовани као засебне нити које у зависности од ситуације шаљу процесору захтев за прекид.

# Тест програм

Идеја програма је сортирање низа бројева са локације array у меморији растуће по збиру њихових цифара:

* array.s – дефиниција низа који треба сортирати
* print.s – потпрограм за испис низа бројева на екран терминала
* digit\_sum.s – потпрограм за рачунање збира цифара
* sort.s – потпрограм за сортирање низа
* main.s – главни програм који прво исписује дати низ на екран терминала, а онда користећи ове функције налази решење и резултат такође исписује на екран терминала
* setup.s – иницијализација система и прекидне рутине.

Улаз:

# array.s

.global size

.global array

.data

.equ size, 10

array:

.word 274, 338, 883, 225, 743

.word 542, 808, 694, 395, 221

.end

# print.s

.global print

# arguments:

# r0 = array's starting address

# r1 = array's size

#

# return:

# void

.text

print:

push r2

push r3

push r4

push r5

mov r5, r0 # r5 = array

mov r4, r1 # r4 = size

mov r3, r4 # r3 = iterator

loop:

jeq exit

mov r2, r4

sub r2, r3 # r2 = current index

shl r2, 1

add r2, r5

mov r0, [r2] # r0 = current element

call printn

mov data\_out, 32

sub r3, 1

jmp loop

exit:

mov data\_out, 10

pop r5

pop r4

pop r3

pop r2

ret

# helper recursive function

# that prints just one number

#

# arguments:

# r0 = number to print

#

# return:

# void

.equ ascii\_zero, 48

.equ data\_out, 0xFF00

printn:

push r2

push r3

push r4

push r5

mov r4, r0 # r4 = number

jeq return

div r4, 10 # remainder is in r5 = digit to print

mov r0, r4

call printn

add r5, &ascii\_zero

mov data\_out, r5

return:

pop r5

pop r4

pop r3

pop r2

ret

.end

# digit\_sum.s

.global digit\_sum

# arguments:

# r0 = target number

#

# return:

# r0 = sum of his digits

.text

digit\_sum:

push r2

push r3

push r4

push r5

mov r4, r0 # r4 = number

mov r0, 0 # r0 = digit sum

mov r4, r4

loop:

jeq exit

div r4, 10 # remainder is in r5 = digit to add

add r0, r5

mov r4, r4

jmp loop

exit:

pop r5

pop r4

pop r3

pop r2

ret

.end

# sort.s

.global sort

.extern digit\_sum

# arguments:

# r0 = array's starting address

# r1 = array's size

#

# return:

# void

.text

sort:

push r2

push r3

push r4

push r5

mov limiti, r1

sub limiti, 1 # limiti = size - 1

mov limitj, r1 # limitj = size

mov r5, r0 # r5 = array

mov r2, 0 # r2 = i

loopi:

cmp r2, limiti

jeq exit

jgt exit

mov index, r2 # index = i

mov r3, r2

add r3, 1 # r3 = j = i + 1

loopj:

cmp r3, limitj

jeq break

jgt break

mov r4, r3

shl r4, 1

add r4, r5

mov r0, [r4] # r0 = array[j]

call digit\_sum

mov temp, r0 # temp = first digit sum

mov r4, index

shl r4, 1

add r4, r5

mov r0, [r4] # r0 = array[index]

call digit\_sum

cmp temp, r0

jeq next

jgt next

mov index, r3 # index = j

next:

add r3, 1

jmp loopj

break:

mov r3, index

shl r3, 1

add r3, r5 # r3 = array + index

mov r4, r2

shl r4, 1

add r4, r5 # r4 = array + i

mov temp, [r3] # temp = array[index]

mov [r3], [r4] # array[index] = array[i]

mov [r4], temp # array[i] = temp

add r2, 1

jmp loopi

exit:

pop r5

pop r4

pop r3

pop r2

ret

.data

limiti: # limit for i loop

.word 0

limitj: # limit for j loop

.word 0

index: # minimum index

.word 0

temp: # helper location

.word 0

.end

# main.s

.global \_start

# this is not necessary

# but recommended

.extern size

.extern array

.extern print

.extern sort

.text

\_start:

mov r0, &array

mov r1, &size

call print

mov r0, &array

mov r1, &size

call sort

mov r0, &array

mov r1, &size

call print

halt

.end

# setup.s

.section iv\_table, "a"

.word ivt\_entry0

.word ivt\_entry1

.word ivt\_entry2

.word ivt\_entry3

.skip 8

.section routines, "ax"

.global ivt\_entry0, ivt\_entry1

.global ivt\_entry2, ivt\_entry3

.extern \_start

# processor initialization

ivt\_entry0:

mov r0, 0

mov r1, 0

mov r2, 0

mov r3, 0

mov r4, 0

mov r5, 0

mov sp, 0xFF00 # stack start

mov \*0xFF10, 0 # reset timer\_cfg

mov psw, 0xE000 # allow all interrupts

mov pc, &\_start # jump to address of the first instruction

# invalid instruction

ivt\_entry1:

mov \*0xFF00, 73 # I

mov \*0xFF00, 110 # n

mov \*0xFF00, 118 # v

mov \*0xFF00, 97 # a

mov \*0xFF00, 108 # l

mov \*0xFF00, 105 # i

mov \*0xFF00, 100 # d

mov \*0xFF00, 32 #

mov \*0xFF00, 105 # i

mov \*0xFF00, 110 # n

mov \*0xFF00, 115 # s

mov \*0xFF00, 116 # t

mov \*0xFF00, 114 # r

mov \*0xFF00, 117 # u

mov \*0xFF00, 99 # c

mov \*0xFF00, 116 # t

mov \*0xFF00, 105 # i

mov \*0xFF00, 111 # o

mov \*0xFF00, 110 # n

mov \*0xFF00, 33 # !

mov \*0xFF00, 10 # \n

halt

# timer interrupt routine

ivt\_entry2:

mov \*0xFF10, 5 # make timer period longer = 10sec

iret

# terminal interrupt routine

ivt\_entry3:

mov \*0xFF00, \*0xFF02

mov \*0xFF02, 0

iret

.end

Излаз:

array.s

Assembling finished successfully!

print.s

Assembling finished successfully!

digit\_sum.s

Assembling finished successfully!

sort.s

Assembling finished successfully!

main.s

Assembling finished successfully!

setup.s

Assembling finished successfully!

Loading program...

Starting emulator...

274 338 883 225 743 542 808 694 395 221

221 225 542 274 743 338 808 395 694 883

Test program ended.