

Универзитет у Београду
Електротехнички факултет
Системски софтвер

Асемблер и емулатор

Андрија Колић (2017/0130)

Београд, 2020.

ВЕРЗИЈЕ ДОКУМЕНТА

Број верзије	Опис измене	Датум измене
1.0	Основна верзија	09.08.2020.

САДРЖАЈ

1.	Упутство за коришћење.....	3
1.1.	Упутство за превођење	3
1.2.	Упутство за тестирање.....	3
1.2.1.	Тестирање асемблера.....	3
1.2.2.	Тестирање емулятора	4
2.	Опис имплементације.....	6
2.1.	Имплементација асемблера.....	6
2.2.	Имплементација емулятора	7

1. Упутство за коришћење

1.1. Упутство за превођење

Превођење овог пројекта је сасвим поједностављено укљученим *Makefile* фајлом. Постоје два извршна фајла која је могуће изградити:

- *assembler*
- *emulator*

Грађење ових програма се постиже коришћењем следећих команди у конзоли:

- *make* – гради оба програма
- *make bin/assembler* – гради *assembler* у *bin* директоријуму
- *make bin/emulator* – гради *emulator* у *bin* директоријуму

```
5
6  both: bin/assembler bin/emulator
7
8  bin/assembler: $(AS_OBJS)
9      g++ -Wall -o $@ $(AS_OBJS)
10
11 bin/emulator: $(EM_OBJS)
12     g++ -Wall -o $@ $(EM_OBJS)
13
```

Исечак *Makefile* фајла

1.2. Упутство за тестирање

Након успешног превођења, у *bin* директоријуму се налазе два извршна фајла: *assembler* и *emulator*. У сврху њиховог тестирања приложени су фајлови у *tests* директоријуму.

Команде за покретање тестова наведене у овом документу, као и друге команде за покретање тестова, се могу наћи у *komande.txt* фајлу у *documentation* директоријуму.

1.2.1. Тестирање асемблера

За тестирање асемблера се може користити било који фајл из *tests* директоријума, мада је тест *test.s* написан тако да тестира све функционалности асемблера – тако што садржи све инструкције, директиве и начине адресирања. Тест се покреће следећом командом:

```
bin/assembler -o out/test.o -bin out/test.bin tests/test.s
```

Као резултат овог теста, у директоријуму *out* се креирају два фајла: *test.o* и *test.bin*. Фајл *test.o* је текстуални фајл који садржи школски формат предметног

програма, док је *test.bin* бинарна репрезентација истог фајла која се користи као улаз за емулатор.

Супституцијом *test.s* ниске именом неког другог фајла се постиже асемблирање траженог фајла. Синтакса команде за покретање *assembler* фајла је следећа:

- име излазног текстуалног фајла се наводи након *-o* токена (препоручује се коришћење *out* директоријума ради боље организације)
- име излазног бинарног фајла се наводи након *-bin* токена (препоручује се коришћење *out* директоријума)
- име улазног фајла се наводи без специјалног претходног токена (приложени тестови се налазе у *tests* директоријуму)
- опционо се уноси *-local* како би у табели симбола били наведени и локални симболи

```
#symbol table
#   index      name defined section  value  vis.  size
   0      *UND      0      0      0      l
   1      data      1      1      0      l      9
   2      haha      1      2      0      l      2
   3      text      1      3      0      l     33
   4      f_var1     1      1      3      g
   5      f_var2     1      1      5      g
   6      fun       1      3      4      g
   7      asdf      0      0      0      g

#rel text:
#   offset      type  value
   31      R_386_PC16      7

#data:
02 04 05 12      00 f4 01 ee      02
#haha:
0a 00
#text:
4c 26 54 26      4c 20 4c 22      4c 24 4c 26      4c 28 4c 2a
64 6c 0e 00      20 64 6c 10      00 22 6c 22      20 64 20 6c
10 00 54 2a      54 28 54 26      54 24 54 22      54 20 14 2c
01 fe ff
```

Пример излазног текстуалног фајла асемблера

1.2.2. Тестирање емулатора

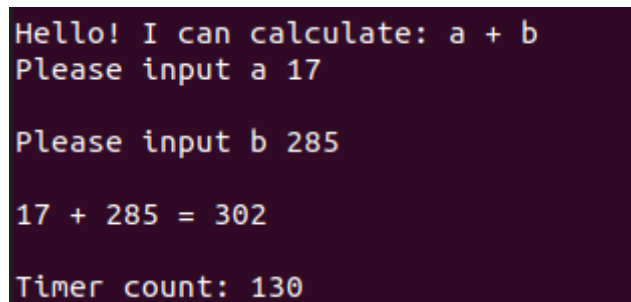
Како би се емулатор тестирао, неопходно је претходно покретање асемблера, јер се излазни фајлови асемблера користе као улаз за емулатор. Могуће је коришћење било које комбинације бинарних излазних фајлова асемблера, мада ће само неке дати смислене резултате (тестови писани за асемблер не представљају неки смислени програм, већ су само низ насумичних инструкција).

Припремљена су два програма за тестирање емулатора. Први програм исписује енглески алфабет и покреће се помоћу следеће две команде:

```
bin/assembler -o out/emu01.o -bin out/emu01.bin tests/emu01.s
bin/emulator out/emu01.bin -place=ivt@0x0000 -place=mmr@0xff00 -
place=text@0x0010 -place=stack@0xfe00
```

Други програм рачуна тражи од корисника да унесе два ненегативна цела броја, и потом рачуна њихов збир.

```
bin/assembler -o out/ivt.o -bin out/ivt.bin tests/ivt.s
bin/assembler -o out/mmr.o -bin out/mmr.bin tests/mmr.s
bin/assembler -o out/routines.o -bin out/routines.bin tests/routines.s
bin/assembler -o out/main.o -bin out/main.bin tests/main.s
bin/emulator out/ivt.bin out/mmr.bin out/routines.bin out/main.bin -place=ivt@0x0000
-place=mmr@0xff00 -place=stack@0xfe00 -place=text@0x0100 -
place=data@0xfd00
```



```
Hello! I can calculate: a + b
Please input a 17

Please input b 285

17 + 285 = 302

Timer count: 130
```

Пример покретања другог програма

Синтакса за позивање емулятора је следећа:

- наводи се име једног или више улазних фајлова (бинарни излазни фајлови асемблера, вероватно се налазе у *out* директоријуму)
- наводе се адресе секција у формату *-place=<ime_sekcije>@<adresa>* што говори емулятору где у меморији да смести агрегирану секцију
- опционо се наводи *-table* како би се иштампала агрегирана табела симбола

2. Опис имплементације

2.1. Имплементација асемблера

Први корак асемблера јесте пролаз кроз улазни фајл. У овом пролазу, асемблер чита реч по реч из улазног фајла, проверава да ли тренутна реч поштује синтаксу, и затим обрађује ту реч на основу њене категорије – генеришући садржај у тренутној секцији, додавањем симбола у табелу симбола, додавањем израза у табелу израза (када наиђе на *.equ* директиву), додавањем симбола у табелу глобалних симбола (када наиђе на *.global* или *.extern* директиву), додавањем референци на симбол у табелу референци, или одговарајућом комбинацијом поменутих акција. За читање и анализу речи из улазног фајла се користи *Parser* класа. Ова класа чита реч по реч и категоризује их помоћу регекс израза, или скупова дозвољених вредности (нпр. *Parser* поседује скуп свих инструкција помоћу ког може проверити да ли је нека реч инструкција). Поред категорије речи, *Parser* такође издваја битне податке из речи (из речи **a(%r7)* издваја симбол *a* и регистар *r7*), и поставља одређене индикаторе – индикатор да се ради о новом реду, индикатор о начину адресирања, индикатор о присуству запете на крају речи, итд.

По завршетку проласка, прво се разрешавају изрази. Изрази се памте као целобројна основа добијена рачунањем литералних делова израза, и као низ симбола са предзнаком. Да би се израз израчунао, потребно је да су сви симболи у поменутом низу познати. Пролази се кроз табелу неизрачуњивих израза и покушава се рачунање сваког. Ако се у пролазу израчуна макар један, поново се пролази. Ако на крају још увек има неизрачуњивих, баца се грешка.

Након тога се пролази кроз табелу глобалних симбола и у табели симбола се означавају симболи као глобални. Ако се симбол извози, провера се да је дефинисан, ако се увози, проверава се да је недефинисан. Проверава се и да ли су локални симболи дефинисани.

У овом тренутку су познати сви симболи, па се могу решити референце. Пролази се кроз све референце и зависно од типа референце и типа симбола, генерише се садржај у секцији, креира се релокациони запис, или оба.

У овом тренутку би се могао вршити испис табеле симбола, релокационих записа и самих секција. Међутим, редослед симбола у табели симбола је хаотичан, јер се симболи појављују у табели редом којим су дефинисани или референцирани у улазном фајлу. Такође, у табели се налазе и локални симболи који више нису потребни. Из ових разлога се табели симбола мења редослед, ремапирањем сваког симбола на нови индекс. У релокационим записима се памти референцирани симбол помоћу његовог индекса у табели симбола, што значи да како би премештање било потпуно, неопходно је изменити и одређена поља у табели симбола (број секције) као и релокационе записе. Након што је извршено ремапирање, секције се налазе на почетку табеле симбола, потом се налазе глобални симболи, а на крају долазе локални симболи. Локални симболи се уклањају из табеле.

Врши се испис табеле симбола, релокационих запис и садржаја секција у текстуалном и бинарном формату.

Ако у се у било ком кораку асемблирања наиђе на неисправност, јавља се грешка кориснику и прекида се асемблирање.

2.2. Имплементација емулятора

Емулятор се извршава у две фазе. Прво се извршава линкер фаза у којој се повезују улазни фајлови, а затим се извршава емулација.

Фаза линковања почиње проласком кроз табеле симбола свих улазних фајлова. Тренутно се обрађују само секције и оне се додају у агрегирану табелу симбола. Рачуна се дужина сваке агрегиране секције, и рачуна се померај сваке од секција улазних фајлова у односу на почетак агрегиране секције.

Затим се пролази кроз *-place* конзолне аргументе и покушава се постављање секција на тражене адресе. Секције за које није наведена почетна адреса се постављају секвенцијално иза последње секције за коју јесте наведена почетна адреса. Још се не копира садржај секција, само се проверава да ли све стају у меморијски простор поштујући наведена правила, без преклапања.

Затим се копирају сви преостали симболи из улазних табела симбола у агрегирану табелу симбола. За сваку улазну табелу симбола се памте индекси за сваки симбол, који указују на тај симбол у агрегираној табели симбола. У овом кораку би требало да се повежу сви увезени и извезени симболи, па се проверава да ли су сви симболи дефинисани.

Копирају се секције у меморију, поштујући почетне адресе израчунате у једном од претходних корака. Фаза линковања се завршава обрађивањем релокационих записа из свих улазних фајлова. Релокациони записи се решавају уписом директно у меморију.

Фаза емулације може да почне. У претходној фази је припремљена меморија, тако да се сада само иницијализује контекст процесора тако да се креће од локације на коју указује меморија на адреси 0. Такође се бележи почетно време емулације, како би било познато када се треба позвати прекидна рутина тајмера.

Главна петља емулятора се састоји из следећих корака:

- провера да процесор није заустављен (процесор се зауставља *halt* инструкцијом, и тада се емулација завршава)
- извршење следеће инструкције
- провера да ли је дошло до грешке у инструкцији (лош операциони код, или лош метод адресирања), и у случају грешке, скок на прекидну рутину за обраду грешке
- провера колико је времена прошло од почетка емулације/претходног позива тајмера. У случају да је прошло онолико времена колико је постављено за циклус тајмера, као и да прекид није маскиран – позива се прекидна рутина тајмера

- провера да ли је неки тастер притиснут у току ове инструкције. У случају да јесте притиснут тастер, као и да прекид није маскиран – позива се прекидна рутина терминала

Позивање прекидне рутине подразумева памћење тренутне локације и статусног регистра на стеку, и уписивање адресе прекидне рутине у *PC* регистар. Петља се наставља нормално, и у следећој итерацији петље биће извршена прва инструкција прекидне рутине јер *PC* указује на ту локацију.

Извршење инструкције се емулира тако што се чита бајт који представља дескриптор инструкције, тај бајт се декодује и добија се инструкција, и онда се позива одговарајућа метода за ту инструкцију. У оквиру те методе биће прочитани сви потребни операнди, биће извршена одговарајућа операција над њима, и резултат ће по потреби бити уписан на одговарајућу адресу.

Ако у се у било ком кораку у фази линковања наиђе на неисправност, јавља се грешка кориснику и прекида се емулатор.