УНИВЕРЗИТЕТ У НОВОМ САДУ
ФАКУЛТЕТ ТЕХНИЧКИХ НАУКА
У НОВОМ САДУ

Кандидат: Филип Параг
Наставник: др Небојша Пјевалица

Реализација емулатора ЛПРС1 процесора

Испитни рад из предмета
Логичко пројектовање рачунарских система 1

Нови Сад, јануар 2023.

Иницијални пример покретања емулатора

Преглед

Ово је упутство за коришћење софтверског пакета „ЛПРСему“ за асемблерско превођење, емулацију и дебаговање програма написаних за ЛПРС процесор, који се користи на вежбама из предмета Логичко пројектовање рачунарских система 1.

Софтверски пакет долази са два програма: lprsemu и lprsasm. Први се користи за емулацију и верификацију исправности рада асемблерских програма, док се други користи за превођење у *VHDL* опис програмске и радне меморије за референтну имплементацију процесора. У даљем тексту је описан начин употребе ових програма са примерима.

Инсталација

Софтверски пакет „ЛПРСему“ се може преузети на два начина – у бинарном издању објављеном на платформи *GitHub*, или компилацијом изворног кôда помоћу cargo управљача пакета за програмски језик Раст.

Пример компилације на оперативном систему Убунту 22.04

```
curl https://sh.rustup.rs -sSf | sh
rustup install nightly-x86_64-unknown-linux-gnu
cargo install lprsemu
```

Адреса најновијег бинарног издања

```
https://github.com/filiparag/ftn-lprsemu/releases/latest/
```

Асемблерске директиве

Директиве увек почињу тачком, састоје се од алфанумеричких карактера и доње црте, и морају бити дефинисане у засебним редовима. Следи листа директива:

Директива за секцију података `.data`

Означава почетак одељка у коме се дефинишу вредности радне меморије. У програму може постојати само једна секција података. У сваком реду ове секције може се налазити више израза (у складу са описом података из поглавља Опште конвенције) одвојених размаком и/или запетом. Наведени изрази представљају 16-битне речи у меморији, где први израз има нулту адресу. Вредности меморије на адресама ван оних дефинисаних у овом одељку су једнаке нули.

Директива за секцију програмског кôда `.text`

Означава почетак одељка у коме се налазе инструкције програма. У програму може постојати само једна секција инструкција. Адресе инструкција у текст секцији крећу од нуле, где ће при покретању процесора програм кренути извршавање од оне која се налази на нултој адреси.

Опште конвенције

Подаци

Подаци, односно речи, у радној меморији се могу описати децималним, бинарним и хексадецималним бројевима не већим од 2^{16} ако су неозначени, или 2^{15} ако су означени. Означеност појединачних речи је произвољна и може се мешати, али је у кôду потребно водити рачуна да се не мешају у аритметичко-логичким операцијама.

Подразумевани облик броја је неозначени децимални. За хексадецимални и бинарни запис се додају префикси „0x“ и „0b“ респективно, а за аутоматску конверзију негативне вредности у комплемент двојке се на цео израз додаје предзнак минус. Као пример, дат је опис меморије која на свом почетку садржи првих пет вредности алтернативног аритметичког реда:

```
.data
     1, -2, 0x03,
     -0b100 5
```

Лабеле (симболи)

Лабеле су називи састављени од алфанумеричких карактера и доње црте праћени двотачком. Не сме постојати више лабела са истим именом. Као пример, дат је програмски кôд бесконачне петље, тј. инструкција која скаче на себе саму:

```
petlja:
     jmp petlja
```

Ову конструкцију је згодно ставити на крај програма. У емулатору се бесконачне петље, код којих инструкција скаче на саму себе, не извршавају (бесконачно), како би се емулација програма могла завршити.

Инструкције

Инструкције се деле на три врсте, зависно од броја операнада. Они се наводе након назива инструкције, међусобно раздвојени размаком и/или запетом.

Инструкције са једним операндом су инструкције условног гранања и њихов једини операнд је назив лабеле на коју скачу. Инструкције са два и три операнда очекују имена регистара, где је име дефинисано као број регистра претхођен великим латиничним словом „R“.

Коментари

Једнолинијски коментар је садржај једне линије текста који се налази након знака тарабе, тачка-запете или двоструке косе црте. Блок коментар је вишелинијски произвољан садржај започет са „/*“ и завршен са „*/“.

Пример програма

У наставку следи пример кôда програма који рачуна производ два означена броја поновљеним сабирањем и резултат складишти у радну меморију:

```
.data
     0x0000           // Производ ће бити складиштен овде
     5, -6            // Чиниоци множења
.text
priprema:
     inc   R0, R0     # Увећавање вредности регистра R0 на 1
     ld    R1, R0     # Учитавање првог чиниоца у R1
     inc   R0, R0     # Увећавање вредности регистра R0 на 2
     ld    R2, R0     # Учитавање другог чиниоца у R2
     sub   R0, R0, R0 # Нулирање регистра R0
sabiranje:
     add   R0, R0, R1 # Сабирање чиниоца са тренутном сумом
     dec   R2, R2     # Умањивање другог чиниоца (бројача)
     jmpnz sabiranje  # Условно понављање сабирања
rezultat:
     st    R0, R2     # Складиштење резултата
kraj:
     jmp kraj         # Бесконачна петља на крају
```

У даљем тексту се подразумева да је наведени пример записан у датотеци primer.asm у радном директоријуму.

Емулација и дебаговање

Покретање емулатора за конкретан програм се врши позивањем lprsemu програма са прослеђеном релативном или апсолутном путањом до датотеке асемблерског програма:

```
lprsemu ./primer.asm
```

При покретању је могуће добити грешку уколико програмска датотека није добро форматирана. На пример, ако је изостављен трећи операнд инструкције за сабирање, добије се следећа грешка:

```
Parsing error:   --> 12:53
   |
12 |      add   R0, R0 # Сабирање чиниоца са тренутном сумом
   |                  ^---
   = expected register
```

Ако је програм синтаксно ваљан, емулатор ће се покренути и добиће се приказ сличан следећем:

```
Registers
| R0:      0 | R1:      0 | R2:      0 | R3:      0 |
| R4:      0 | R5:      0 | R6:      0 | R7:      0 |
Flags [ zero: false ] [ sign: false ] [ carry: false ]
Program counter: 0
Runtime counter: 0
Data memory
|   0 |      1
| ··· |      0
Program memory
|     | kraj:
|   0 |     jmp   0 (kraj) <=
| ··· | nop
lprsemu >>
```

У даљем тексту ће укратко бити описано управљање емулатором кроз командну линију. Излаз из емулатора се врши слањем *SIGINT* сигнала, односно пречицом Ctrl-C.

Приказ регистара

У одељку *Registers* су приказане вредности специјалних и општенаменских регистара. Подразумевани запис бројева у регистрима опште намене је означен децимални. У одељку Интерактивна конзола је описан поступак промене приказаног записа.

Приказ меморије

У одељку *Data memory* су приказане све речи у радној меморији. Уколико се на крају налазе узастопне нуле, оне бивају приказане скраћено са три тачке. Подразумевани запис бројева у меморији је означен децимални.

У одељку *Program memory* су приказане све инструкције у програмској меморији и пропратне лабеле. Уколико меморија није попуњена до краја, на дну се налазе три тачке и специјална инструкција „*nop*“. Подразумевано извршавање програма стаје на првој појави те инструкције.

Симболи извршавања

На линији асемблерског кôда која ће следећа бити извршена се налази симбол стрелице „`<=`“.

На линијама на којима је активна тачка прекида се додатно налази симбол звездице у загради: „`(*)`“.

Приказ грешака при извршавању

Уколико при извршавању асемблерског кôда дође до грешке, рецимо покушај учитавања вредности у регистар са меморијске адресе ван опсега, емулација се зауставља и приказује се грешка:

```
Emulation error: OutOfRange
```

Емулацију је немогуће наставити даље без ресетовања процесора, повратка уназад или принудног скакања на другу инструкцију.

Интерактивна конзола

На дну стоји интерактивна конзола у коју се уписују команде наведене испод, са скраћеним обликом у загради:

- print (p)
  Исписује тренутно стање процесора.
- load-file (l)
  Учитава програм из датотеке прослеђене као први аргумент.
- radix (d)
  Мења запис приказаних вредности у регистрима и меморији на основу првог аргумента:
  - u: неозначени децимални бројеви
  - s: означени децимални бројеви
  - x: неозначени хексадецимални бројеви
  - b: неозначени бинарни бројеви
- run (r)
  Наставља извршавање програма до прве следеће тачке прекида.
- run-all (ra)
  Наставља извршавање програма од тренутне до последње иснтрукције.
- step (s)
  Извршава следећу инструкцију.
  Подразумевана команда, извршава се ако се само притисне *Return*.
- undo (u)
  Ресетује процесор и извршава све инструкције до претходне.
- breakpoint (b)
  Поставља или уклања тачку прекида на инструкцији чија је адреса први параметар команде.
- breakpoint-clear (bc)
  Уклања све тачке прекида.
- jump (j)
  Скаче на инструкцију чија је адреса први параметар команде.
- reset (x)
  Ресетује стање процесора на почетно, без уклањања тачки прекида.
- help (h)
  Исписује упутство за коришћење интерактивне конзоле.

Асемблирање

Покретање асемблера за конкретан програм се врши позивањем lprsasm програма са прослеђеном релативном или апсолутном путањом до датотеке програма:

```
lprsasm ./primer.asm
```

Слично емулатору, при покретању је могуће добити грешку уколико програмска датотека није добро форматирана.

Ако је програм синтаксно ваљан, асемблер ће у конзоли исписати *VHDL* кôд. Да се испис преусмери у датотеке, при покретању је неопходно навести произвољан назив датотека без екстензије:

```
lprsasm ./primer.asm memorija
```

У датотеке memorija.rom.vhd і memorija.ram.vhd ће бити уписан исечак *VHDL* кôда за програмску и радну меморију респективно.