Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Nejc Kišek

Povezovalnik za hipotetični računalnik SIC/XE

diplomsko delo

univerzitetni študijski program
prve stopnje
računalništvo in informatika

Mentor: doc. dr. Jurij Mihelič

Ljubljana, 2016

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Povezovalnik za hipotetični računalnik SIC/XE
Linker for a hypothetical computer SIC/XE

Tematika naloge:

SIC/XE je hipotetični računalnik, ki se pogosto uporablja v izobraževanju pri predmetih s področja sistemske programske opreme. V okviru poučevanja je na Fakulteti za računalništvo in informatiko Univerze v Ljubljani nastal programski sistem SicTools, ki vsebuje zbirnik in simulator, manjka pa mu še ustrezen povezovalnik objektne kode. V okviru diplomske naloge preučite področje povezovanja s poudarkom na računalniku SIC/XE. Nato implementirajte povezovalnik in ga vgradite v sistem SicTools. Arhitektura sistema naj bo moderna in objektno-orientirano zasnovana v obliki samostojnih knjižnic izvorne kode. Poleg osnovnega povezovanja naj omogoča tudi naprednejšo uporabo preko ustreznega programskega vmesnika.

Kazalo

Seznam uporabljenih kratic

kratica	**angleško**	**slovensko**
SIC	Simplified Instructional Computer	Poenostavljen ukazni računalnik
SIC/XE	SIC/Extra Equipment	SIC z dodatno opremo
ASCII	American Standard Code	Ameriški standardni nabor
	for Information Interchange	za izmenjavo informacij
CSTAB	Control section table	Tabela kontrolnih sekcij
ESTAB	External symbol table	Tabela zunanjih simbolov

Povzetek

Naslov: Povezovalnik za hipotetični računalnik SIC/XE

Povezovalnik je del sistemske programske opreme, ki kodo programa, napisano v več datotekah združi v eno, izvedljivo datoteko. To nam omogoči uporabo knjižnic in razdeljevanje programov na ločene sekcije.

SIC/XE je hipotetični računalnik, ki se v izobraževanju pogosto uporablja za prikaz delovanja sistemskih programov – uporablja se tudi na Fakulteti za računalništvo in informatiko Univerze v Ljubljani, kjer je bila za pomoč pri poučevanju zanj razvita zbirka SicTools. Ta vsebuje napreden simulator, zbirnik in nalagalnik, manjka pa ji povezovalnik.

V sklopu naloge je bil razvit povezovalnik za računalnik SIC/XE, ki bo razširil zbirko orodij SicTools. Povezovalnik omogoča povezovanje z več različnimi načini, pa tudi urejanje posameznih sekcij, preden se povežejo. Uporaba je možna preko tekstovnega in grafičnega vmesnika, ali pa kot knjižnica v drugem programu.

Na začetku naloge je opisano delovanje SIC/XE računalnika in povezovalnikov. Sledi predstavitev uporabniškega in programskega vmesnika. V naslednjem poglavju je podrobno opisano delovanje izdelanega programa in vseh njegovih komponent. Nazadnje je predstavljen še tipičen primer povezovanja – od kode v zbirnem jeziku do končne povezane datoteke.

Ključne besede: povezovalnik, SIC, SIC/XE, sistemska programska oprema, SicTools.

Abstract

Title: Linker for SIC/XE hypothetical computer

A linker is a part of system software that combines code written in multiple files into a single, executable file which allows for use of libraries and programs divided into multiple units.

SIC/XE is a hypothetical computer often used in education for demonstrating system software – it is also in use at the Faculty of Computer and Information Science at the University of Ljubljana, where a collection of SIC/XE system software and tools named SicTools was developed. SicTools contains an advanced simulator of SIC/XE, an assembler and a loader while a linker is missing.

In this thesis, the missing linker for SIC/XE has been developed. It has multiple linking modes and allows interactive editing of the program sections before they are linked. It can be used either with a textual or graphical interface or as a library in a separate program.

We start with an overview of SIC/XE and linkers, followed by a description of linker's user interfaces and instructions for using the library. In the next chapter we have a detailed documentation for all the linker components. Finally, there is a typical example of linking a program – from an assembly code to a final linked file.

Keywords: linker, SIC, SIC/XE, system software, SicTools.

Poglavje 1

Uvod

Povezovalnik (angleško *linker*) je del sistemske programske opreme, ki prevedeno kodo iz različnih datotek poveže v eno objektno datoteko, ki se nato lahko izvede. Omogoči nam, da lahko program napišemo v več datotekah oziroma da uporabljamo zunanje knjižnice.

SIC (Simplified Instructional Computer) je hipotetični računalnik, ki je opisan v knjigi System Software avtorja Lelanda Becka [1] in ki se na Fakulteti za računalništvo in informatiko na Univerzi v Ljubljani kot primer uporablja pri predmetu Sistemska programska oprema na Univerzitetnem programu. Glavna tema knjige in omenjenega predmeta je spoznavanje s sistemsko programsko opremo – zbirnikom, nalagalnikom, povezovalnikom, delno tudi prevajalniki in osnovami operacijskih sistemov. Procesor SIC je za takšen namen zelo primeren – načrtovan je namreč tako, da ima čim več skupnega z realnimi procesorji, ne da bi se preveč približal kateremu od njih. Zaradi tega so koncepti, uporabljeni pri programiranju za SIC, uporabni pri večini ostalih procesorjev, razvoj sistemske programske opreme zanj pa ima veliko skupnega z realno sistemsko opremo. Procesorjeva razširjena verzija, imenovana SIC/XE, ima več registrov, več ukazov in več pomnilnika in se zato uporablja pogosteje kot osnovni SIC.

Za lažje delo s SIC oziroma SIC/XE je bila razvita skupina orodij SicTools [2], dostopna na http://jurem.github.io/SicTools/, ki med drugim vsebuje

zbirnik, nalagalnik in simulator z grafičnim vmesnikom, zaenkrat pa še ne vključuje povezovalnika. SicTools se med drugim uporablja na naši fakulteti pri že prej omenjenem predmetu Sistemska programska oprema za praktično demonstracijo delovanja sistemskih programov, pa tudi kot zgled študentom, ko izdelujejo svoje sistemske programe.

1.1 Cilj diplomske naloge

V diplomski nalogi bo izdelan povezovalnik za računalnik SIC/XE. Povezovalnik bo objektno orientiran, napisan v javi in izdelan čim bolj splošno, tako da bo uporaben tudi kot šolski primer. Uporaba programa bo možna na dveh nivojih: kot orodje dostopno preko ukazne vrstice in kot programski vmesnik, ki se lahko vključi v drug program, na primer v SicToolsov simulator.

Pisni del naloge bo uporaben kot pregled delovanja povezovalnika ali kot vodilo nekomu, ki bi rad izdelal svoj program. Praktični del naloge (izdelani povezovalnik) bo uporabno orodje, ki bo razširilo obstoječo zbirko SicTools.

Poglavje 2

Pregled področja

2.1 Računalnik SIC/XE

SIC vsebuje enostaven procesor, ki je bil razvit kot primer tipičnega procesorja in zato vsebuje le funkcionalnosti, ki so skupne večini pravih procesorjev. Koncepti, ki se uporabljajo pri programiranju za SIC, so zato lahko uporabni pri programiranju za večino realnih računalnikov.

SIC/XE (XE pomeni *extra equipment* oz. *extra expensive* – z dodatno opremo oz. dodatno drag) je razširjena verzija istega procesorja, ki ima dodatne ukaze, več registrov in lahko naslovi več pomnilnika.

Pomnilnik Pomnilnik je sestavljen iz 8-bitnih bajtov (byte), trije bajti pa predstavljajo besedo (word). Vsak bajt je možno nasloviti posebej, besede pa se naslavljajo z naslovom najnižjega bajta. V osnovni različici lahko SIC naslovi 2^{15} (32 768) bajtov pomnilnika, SIC/XE pa 2^{20} bajtov (1 MiB).

Registri Vsi registri so dolgi 24 bitov. Osnovni SIC vsebuje 5 registrov:

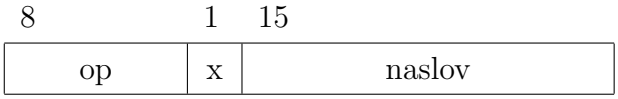
- A – akumulatorski register, uporablja se za aritmetične operacije,
- X – indeksni register, ki se uporablja za lažje naslavljanje zaporednih naslovov v pomnilniku,

- L – povezovalni register, ki shrani povratni naslov ob klicanju podprogramov,
- PC – programski števec,
- SW – statusni register.

SIC/XE ima dodatno še 4 registre:

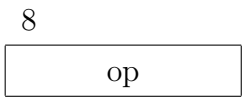
- B – bazni register, ki se uporablja za relativno naslavljanje,
- S in T – dodatna, splošno namenska registra,
- F – register za operacije s plavajočo vejico.

Ukazi Vsi ukazi za SIC so dolgi 24 bitov – 8 bitov za operacijsko kodo, 1 bit za način naslavljanja in 15 bitov za naslov:

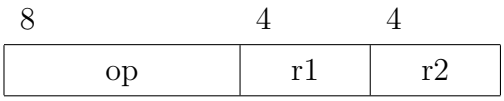


SIC/XE je kompatibilen s SIC programi in zato lahko uporablja tudi prvi format, sicer pa ima na voljo štiri različne formate ukazov:

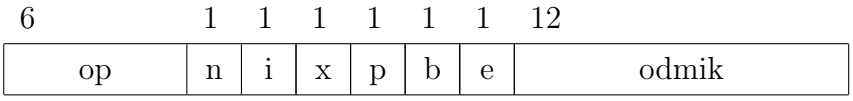
- 8-bitni ukaz, ki vsebuje samo operacijsko kodo:



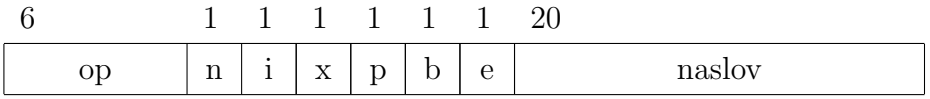
- 16-bitni ukaz, ki vsebuje 8-bitov za operacijsko kodo in dva 4 bitna operanda:



- 24-bitni ukaz, ki vsebuje 6-bitno operacijsko kodo, 6 bitov za zastavice, ki nam povejo vrsto naslavljanja, in 12 bitni odmik za posredno naslavljanje:



- 32-bitni ukaz, ki je enak 24-bitnemu, le da ima namesto odmika celoten 20-bitni absolutni naslov:



Naslavljanje Osnovni SIC podpira absolutno naslavljanje s 15 bitnimi naslovi ali pa indeksno naslavljanje, kjer naslovu prištejemo še vsebino X registra.

SIC/XE podpira neposredno naslavljanje z 20 biti, indeksno naslavljanje in relativno naslavljanje, kjer 12 bitnemu odmiku prištejemo vrednost PC registra ali B registra.

Na tem mestu je naslov samo številka, preko katere operand za ukaz lahko dobimo na različne načine. Na voljo imamo tri načine, od katerih osnovni SIC podpira le prvega, SIC/XE pa vse tri:

- enostavno naslavljanje – operand za ukaz je vsebina pomnilnika na podanem naslovu,
- takojšnje naslavljanje – operand je kar podani naslov, v zbirniku ga prepoznamo po predponi `#`,
- posredno naslavljanje – vsebina pomnilnika na podanem naslovu je naslov operanda, v zbirniku ga prepoznamo po predponi `@`.

Kateri tip naslavljanja je v uporabi, nam določi kombinacija zastavic v ukazu:

- Bit `x`, ki je prisoten v 24-bitnem, 32-bitnem in osnovnem SIC formatu nam določa indeksno naslavljanje – prištejemo vsebino registra X.

- Bita `b` in `p` v 24-bitnem in 32-bitnem formatu določata relativno naslavljanje:

b	p	vrsta
1	0	bazno
0	1	PC-relativno
0	0	neposredno
1	1	nedovoljeno

- Bita `n` in `i` v 24-bitnem in 32-bitnem formatu določata način uporabe naslova. Če sta oba bita 0, potem gre za osnovni SIC format ukaza in sta del 8-bitne operacijske kode.

n	i	vrsta
0	1	takojšnje
1	0	posredno
1	1	enostavno

- Bit `e` je nastavljen na 1, če gre za 32-bitni ukaz. Tako zbirnik ve, da sledi 20-bitni naslov.

Predstavitev podatkov Števila so 24-bitna, negativna števila so predstavljena z dvojiškim komplementom. Znaki so predstavljeni z 8-bitno ASCII kodo. SIC/XE podpira tudi 48-bitna realna števila. Prvi bit predstavlja predznak, naslednjih 11 bitov je eksponent, zadnjih 36 bitov pa mantisa. Vrednost števila se izračuna kot $(-1)^s \cdot 2^{(e-1024)} \cdot m$, kjer je s prvi bit, e eksponent in m mantisa.

Vhodno-izhodne naprave SIC in SIC/XE podpirata tudi operacije z vhodno-izhodnimi napravami. Vsaka ima določeno 8-bitno kodo in s procesorjem komunicira preko najnižjega bajta v registru A.

Kontrolne sekcije V zbirniku se nova kontrolna sekcija začne na začetku programa z ukazom `START` ali kasneje z ukazom `CSECT`. Vsaka sekcija je ločen del kode, zato je več sekcij lahko v eni ali pa v več datotekah. Med seboj komunicirajo preko zunanjih simbolov – simbol je definiran v eni sekciji, ostale pa se lahko nanj sklicujejo, če ga potrebujejo. Tako lahko v sekciji uporabljamo spremenljivke, dele pomnilnika ali rutine iz neke druge sekcije.

```
1  prog   START 0        . prva sekcija z imenom prog
2         EXTREF ref1    . reference na tri zunanje simbole
3         EXTREF ref2
4         EXTREF sect3   . ime sekcije je privzeto zunanji simbol
5         +LDA ref1      . naložimo vrednost iz reference ref1
6         +STA ref2      . sharnimo jo na naslov iz reference ref2
7         +JSUB sect3    . skočimo na naslov z reference sect3
8  halt   J halt
9
10 sect2  CSECT          . druga sekcija z imenom sect2
11        EXTDEF ref1    . definirana dva simbola
12        EXTDEF ref2
13 ref1   WORD 15        . prvi simbol - vrednost 15
14 ref2   RESW 1         . drugi simbol - rezervirano mesto za 1 besedo
15
16 sect3  CSECT          . tretja sekcija z imenom sect3
17        LDA #15
18        ADD #15
19        STA result     . result = 15 + 15
20        RSUB           . vrnemo se v prvo sekcijo
21 result RESW 1
22
23        END prog       . konec programa
```

Slika 2.1: Primer programa s tremi kontrolnimi sekcijami

Zunanji simbol definiramo z ukazom `EXTDEF ime`, kjer je *ime* labela v tej sekciji in ni daljše od 6 znakov. Paziti moramo, da v sekciji, ki bo simbol upo-

rabljala in labele z enakim imenom, in da katera od drugih sekcij ne definira enakega simbola. Imena kontrolnih sekcij so že privzeto definirani zunanji simboli, zato nam jih ni treba dodatno definirati, če jih želimo uporabiti.

Zunanji simbol uporabimo z ukazom `EXTREF ime`, kjer je *ime* simbol, definiran v neki drugi sekciji. Spet je potrebno paziti, da enako ime ni uporabljeno že kje drugje v tej kontrolni sekciji.

Objektna koda

Objektni zapisi za računalnik SIC so napisani kot običajne tekstovne datoteke z več različnimi zapisi. Vsebujejo kodo programa, navodila za prenaslavljanje relativnih naslovov in informacije o zunanjih referencah v obliki objektnih zapisov.

- H – zaglavni zapis (header) vsebuje ime, začetni naslov programa oz. sekcije in dolžino objektne kode v bajtih.

1	2–7	8–13	14–19
H	ime	naslov	dolžina

- T – programski zapis (text record) vsebuje začetni naslov zapisa, dolžino zapisa v bajtih in kodo, ki je predstavljena v šestnajstiški obliki.

1	2–7	8–9	10–69
T	naslov	dolžina	vsebina

- E – končni zapis (end record) vsebuje naslov začetka programa.

1	2–7
E	ime

- M – prilagoditveni zapis (modification record) vsebuje začetni naslov polja, ki mora biti prenaslovljeno, dolžina polja, smer popravka (`+` ali

-) in ime simbola. Enostavnješa različica tega zapisa, uporabna za enostavno prenaslavljanje, izpusti smer in simbol – privzame se, da je potrebno prišteti začetni naslov programa.

1	2–7	8–9	10	11–16
M	naslov	dolžina	+	simbol

- D – definicijski zapis (define record) vsebuje ime in naslov zunanjih simbolih, definiranih v tej kontrolni sekciji. En zapis lahko vsebuje več simbolov.

1	2–7	8–13	14–73
D	ime	naslov	ponovitev polj 2–13 za ostale simbole

- R – referenčni zapis (refer record) vsebuje imena zunanjih simbolov, ki se uporabljajo v tej kontrolni sekciji. En zapis vsebuje več simbolov.

1	2–7	8–73
R	ime	imena ostalih simbolov

```
1  Hprog 00000000000F
2  Rref1 ref2 sect3
3  T0000000F031000000F1000004B1000003F2FFD
4  M00000105+ref1
5  M00000505+ref2
6  M00000905+sect3
7  E000000
8  Hsect2 000000000006
9  Dref1 000000 ref2 000003
10 T0000000300000F
11 E000000
12 Hsect3 00000000000F
13 T0000000C01000F19000F0F20034F0000
14 E000000
```

Slika 2.2: Objektna koda za primer s tremi sekcijami

2.2 Povezovalniki

Ko imamo objektno datoteko, ki smo jo dobili kot izhod iz zbirnika, jo je potrebno naložiti v pomnilnik, kjer se bo nato program izvedel. Če smo v zbirnem jeziku uporabljali relativne naslove, se bo lahko program naložil na katerokoli mesto v pomnilniku, nalagalnik pa mora opraviti tudi prenaslavljanje – spremeniti mora relativne naslove iz objektne datoteke v absolutne naslove v pomnilniku. To se naredi z uporabo modifikacijskih zapisov v objektni kodi, ki nam povedo, na katerih mestih v kodi je potrebno prišteti začetni naslov programa, da bo program deloval na novi lokaciji.

Če smo v programu uporabljali zunanje reference, ki kažejo na spremenljivke iz nekega drugega programa, moramo na tem mestu opraviti še povezovanje. Pri povezovanju bomo zunanje reference v programu nadomestili z dejanskimi naslovi iz neke druge kontrolne sekcije. To izvedemo tako, da pri kreiranju objektne datoteke na mestih, kjer želimo uporabiti referenco, namesto naslova vstavimo ničle in med M zapise dodamo nov zapis s simbolom, ki ga referenciramo, naslovom in dolžino vstavljenih ničel. Povezovalnik bo iz R in M zapisov razbral, kaj je potrebno v kodi popraviti.

2.2.1 Vrste povezovalnikov

Povezovanje je lahko kar del nalagalnika in se izvaja vsakič, ko bomo naložili in pognali program. Tak način ne potrebuje vmesnih datotek, vendar je lahko zelo zamuden, če gre za velike programe in če program poganjamo pogosto.

Druga možnost je samostojni povezovalnik, ki nam iz več kontrolnih sekcij v več objektnih datotekah z medsebojnimi referencami vrne eno kontrolno sekcijo v eni objektni datoteki, ki vsebuje vso kodo. V izhodni datoteki praviloma ni več nobenih D in R zapisov, niti M zapisov, ki bi se nanašali na zunanje simbole. Tako objektno datoteko lahko zato naloži enostaven nalagalnik, na enak način kot bi nalagal program napisan brez uporabe kontrolnih sekcij in zunanjih simbolov. Prednost takega načina povezovanja je hitrost, saj je datoteko treba povezati samo enkrat, kasneje pa jo samo še naložimo.

2.2.2 Povezovanje

Tipično povezovanje se izvaja v dveh prehodih in uporablja dve tabeli, s katerima sledi naslovom kontrolnih sekcij in simbolov.

Prvi prehod zgradi seznam naslovov definiranih zunanjih simbolov in začetnih naslovov posameznih sekcij, ki jih povezujemo. Pri tem mora paziti na prekrivanje med sekcijami – za prenaslovljive sekcije to pomeni, da jih postavi eno za drugo, pri absolutnih pa mora preveriti morebitna prekrivanja.

Povezovalnik pri prvem prehodu polni tabelo CSTAB, ki za vsako sekcijo vsebuje ime, nalagalni naslov in dolžino in pa ESTAB, ki za vsak zunanji simbol vsebuje ime sekcije, ime simbola in naslov simbola. Opisani algoritem bere vrstice vhodnih datotek in na podlagi prebranih podatkov zgradi obe tabeli, hkrati pa si zapomni še dodatne informacije o programu, ki jih bo povezovalnik uporabil v drugem prehodu: ime, dolžina in začetni naslov.

```
CSADDR = 0, PROGNAME = "", STARTADDR = −1
while podatki na vhodu do
begin
   read H zapis
   CSNAME = ime iz H zapisa, CSLEN = dolžina iz H zapisa
   if PROGNAME == "" then PROGNAME = CSNAME
   if CSTAB contains CSNAME then throw error
   else insert {CSNAME, CSADDR, CSLEN} into CSTAB

   read zapis
   while tip zapisa != E do
   begin
      if tip zapisa == D then
         for each simbol iz D zapisa do
         begin
            ESNAME = ime simbola, ESADDR = vrednost simbola
            if ESTAB vsebuje ESNAME then throw error
            else insert {CSNAME, ESNAME, ESADDR + CSADDR} into
                 ESTAB
         end
      read zapis
   end
   CSADDR = CSADDR + CSLEN
   if STARTADDR == −1 then STARTADDR = naslov iz E zapisa
end
PROGLEN = CSADDR
```

Drugi prehod združi kontrolne sekcije v eno in glede na seznam iz prvega prehoda nastavi naslove referenc. Če povezovanje in nalaganje izvajamo naenkrat, se objektni modul ne generira v novo datoteko, ampak se lahko naloži kar neposredno v pomnilnik. Opisani algoritem bere vhodne datoteke in istočasno izpisuje povezano objektno datoteko. Istočasno si polni seznam M zapisov, ki jih na koncu prav tako izpiše v izhodno datoteko. Odvečne M zapise pred izpisovanjem odstrani.

```
MRECORDS = [ ], CSADDR = 0
write "H", PROGNAME, "000000", PROGLEN
while podatki na vhodu do
   read H zapis
   CSNAME = ime iz H zapisa, CSLEN = dolžina iz H zapisa
   read vse T in M zapise v  sekciji

   for each M zapis do
   begin
      FADDR = naslov iz M zapisa
      FLEN = dolžina iz M zapisa
      FLAG = + ali − iz M zapisa
      ESNAME = simbol iz M zapisa
      if  ESTAB vsebuje ESNAME then
         ESVAL = naslov v ESTAB pri ESNAME
         poišči  T zapis,  ki  vsebuje  FADDR
         na FADDR v T zapisu FLEN polzlogom prištej/odštej ESVAL
         na FADDR v T zapisu FLEN polzlogom prištej CSADDR
      else
         throw error
      add {M, FADDR + CSADDR, FLEN, FLAG, PROGNAME} to MRECORDS
   end

   for each T zapis do
   begin
      TADDR = začetek T zapisa
      TLEN = dolžina T zapisa
      DATA = koda iz T zapisa
      write "T", TADDR+CSADDR, TLEN, DATA
   end

   uredi  M zapise (odstrani  pare  +/−PROGNAME na istem naslovu)
   write vse M zapise iz  MRECORDS in M zapise, ki se ne nanašajo na simbole
   write E, STARTADDR
```

2.3 Sorodna dela

2.3.1 SicTools

SicTools[2] je zbirka orodij, ki vključuje zbirnik in grafični simulator. Simulator ima grafični vmesnik in omogoča sprotno spremljanje delovanja računalnika vključno s spremljanjem vsebine registrov in celotnega pomnilnika. Prav tako ima nekatere funkcije razhroščevalnika, kot so ustavitev delovanja ob določenem ukazu, sprotno pretvarjanje objektne kode nazaj v zbirno (disassembly) in izvajanje po enega ukaza naenkrat. Vključuje tudi grafični in tekstovni zaslon. Simulatorjev nalagalnik lahko naloži relativne objektne *.obj* datoteke ali pa z uporabo zbirnika naloži program iz *.asm* datoteke v zbirnem jeziku.

Vključeni zbirnik je možno poganjati preko ukazne vrstice kot samostojen program, simulator pa ga uporablja kot knjižnico, da lahko naloži zbirniške datoteke.

Povezovalnik, predstavljen v tej nalogi, je zasnovan kot dodatek k zbirki SicTools, njegovo delovanje pa je zasnovano na podoben način kot omenjeni zbirnik – samostojno ali preko simulatorja.

Zbirka je dostopna na http://jurem.github.io/SicTools/.

2.3.2 Dela na naši fakulteti

Orodja za računalnik SIC in SIC/XE so bila tema nekaterih diplomskih del na Fakulteti za računalništvo in informatiko Univerze v Ljubljani. Izdelan je bil zbirnik[3], pretvornik med zbirniško kodo SIC/XE in Intel x86 [4], prevajalnik iz jezika C v SIC/XE zbirni jezik [5] in implementacija SIC/XE na FPGA razvojni plošči s pripadajočo sistemsko programsko opremo [6].

Pri nazadnje omenjeni nalogi je med sistemskimi orodji tudi povezovalnik. Glavne razlike med njim in povezovalnikom, predstavljenim v tej nalogi, je, da slednji podpira urejanje sekcij med povezovanjem, je bolj objektno usmerjen, ponuja več načinov uporabe in omogoča enostavno dodajanje novih upo-

rabniških vmesnikov ali vključevanje v druge programe – zato je primeren kot dopolnitev zbirke SicTools.

2.3.3 Ostala podobna orodja

Na spletu je mogoče najti še nekaj drugih implementacij povezovalnika za SIC/XE [7] [8]. Večinoma gre za naloge, ki so jih študentje izdelali v sklopu učenja o sistemski programski opremi. Noben od najdenih povezovalnikov ni bil izdelan kot orodje z uporabniku prijaznim vmesnikom, podrobnimi navodili in jasnimi pogoji uporabe. Tudi njihovi formati datotek se nekoliko razlikujejo od uporabljenih v tej nalogi, zato se z njimi npr. ne da povezati datotek, ki jih prevede SicTools zbirnik.

Poglavje 3

Predstavitev izdelanega povezovalnika

3.1 Podprte funkcionalnosti

Glavna funkcija izdelanega povezovalnika je, da poveže podane relativne *.obj* datoteke v novo skupno datoteko in razreši reference med njimi. Vhodne datoteke so lahko sestavljene iz več kontrolnih sekcij (H zapis v objekti kodi) in lahko imajo zunanje reference in definicije (R in D zapisi v objektni kodi). Izhodna datoteka ima samo eno kontrolno sekcijo ter nima referenc in definicij, saj se vse že razrešene.

Ime izhodne datoteke lahko povezovalniku podamo, sicer pa jo bo sam poimenoval z imenom prve kontrolne sekcije in pripono *_ln.obj*.

Delno povezovanje Če v vhodnih datotekah nimamo razrešenih vseh referenc, bo povezovalnik javil napako. Če želimo, pa lahko izvedemo samo delno povezovanje – razrešene reference bodo popravljene, nerazrešene reference pa ostanejo v izhodni datoteki in jih lahko razrešimo kasneje. Izhodna datoteka bo tako ohranila R in M zapise, ki nimajo pripadajočega D zapisa v nobeni izmed podanih datotek.

Take izhodne datoteke zato še ne moremo poganjati, razen če jo prej do

konca povežemo z datoteko, ki ima potrebne definicijske zapise. Ta način je uporaben, kadar nekaterih uporabljenih knjižnic še nimamo in jih bomo dobili kasneje – npr. od operacijskega sistema.

Ohranjanje zunanjih definicij Pri običajnem povezovanju bo izhodna datoteka brez D in R zapisov, saj je namenjena izvajanju. Če jo želimo kasneje povezati z neko drugo sekcijo, bomo morali uporabiti katerega od zunanjih simbolov, ki so definirani v povezani sekciji. Izdelani povezovalnik omogoča, da uporabnik zahteva, naj se vse zunanje definicije (D zapisi) ohranijo v izhodni datoteki.

Izhodna datoteka tako nima več kontrolnih sekcij, zunanjih referenc in pripadajočih prenaslovitvenih zapisov, zato se lahko naloži z običajnim nalagalnikom in požene.

Glavna sekcija Glavna sekcija je tista, kjer se bo začelo izvajanje in jo določa končni E zapis v *.obj* datoteki. Povezovalnik bo privzeto za glavno sekcijo uporabil prvo sekcijo prve podane datoteke. Če mu podamo ime sekcije, za katero bi radi, da je glavna, bo povezovalnik, preden začne z delom, preuredil vrsti red sekcij tako, da bo podana na začetku, nato pa nadaljeval kot običajno. Če podane sekcije ni v nobeni izmed vhodnih datotke, bo povezovalnik javil napako.

Urejanje sekcij Uporabnik lahko pred prevajanjem uredi sekcije, ki jih vsebujejo vhodne datoteke. Spremeni lahko imena sekcij in simbolov, katerega izmed njih odstrani iz povezovanja ali pa spremeni vrsti red sekcij.

Če preimenujemo ali izbrišemo zunanjo referenco, se skupaj z njo preimenujejo oz. izbrišejo tudi pripadajoči M zapisi. Podprto je tudi poimenovanje z enakom imenom, kot ga ima že neka druga referenca v isti sekciji – stara referenca se bo izbrisala, njeni M zapisi pa se bodo preimenovali z novim imenom.

Imena sekcij in zunanjih definicij morajo biti unikatna – sekcije ne moremo poimenovati z imenom, ki ga ima že neka druga sekcija, definicije sim-

bola ne moremo poimenovati z imenom, ki ga ima že neka druga definicija v isti sekciji.

Vsa imena sekcij in simbolov morajo biti dolga 6 ali manj znakov.

Izpisovanje podrobnosti Poleg teh možnosti izdelani povezovalnik omogoča tudi delovanje v načinu z izpisovanjem informacij (angleško *verbose* mode), kjer sproti izpisuje podrobnosti o poteku povezovanja in omogoči uporabniku, da spremlja dogajanje. Ta način je uporaben, kadar gre med povezovanjem kaj narobe in bi uporabnik rad videl, kje se je napaka pojavila.

Izdelani povezovalnik podpira samo povezovanje relativnih oz. prenaslovljivih sekcij in javi napako, če je kateri od vhodnih programov absoluten.

3.2 Uporaba samostojnega povezovalnika

Izdelani prevajalnik je mogoče uporabljati kot samostojno orodje preko ukazne vrstice ali z enostavnim grafičnim vmesnikom. Oba vmesnika ponujata enake funkcionalnosti in omogočata uporabo vseh opcij vgrajenih v povezovalnik.

3.2.1 Uporaba preko ukazne vrstice

Povezovalnik poženemo z `java sic.Link`, kjer je sic.Link prevedena *.class* datoteka. Če uporabljamo povezovalnik kot del SicTools orodij, ga poženemo z `java -cp sictools.jar sic.Link`. Temu ukazu sledi seznam zastavic za opcijske funkcionalnosti in seznam vhodnih datotek. Zastavice, ki so na voljo:

- `-o` ali `-out` in ime datoteke, ki določi izhodno datoteko. Na primer `java sic.Link -o iz.obj v1.obj v2.obj`, kjer je `iz.obj` izhodna datoteka, `v1.obj` in `v2.obj` pa vhodni datoteki.
- `-m` ali `-main` in ime, ki določi ime glavne oz. prve sekcije. Na primer `java sic.Link -m prva v1.obj v2.obj v3.obj`, kjer je `prva`

ime ene izmed sekcij, prisotnih v vhodnih datotekah.

- `-f` ali `-force` omogoči delno povezovanje – povezovanje se bo izvedlo, tudi če kakšna od referenc ni razrešena.
- `-k` ali `-keep` ohrani zunanje definicije – izhodna datoteka bo vsebovala vse D zapise iz vhodnih datotek.
- `-v` ali `-verbose` vklopi izpisovanje informacij o povezovanju na standardni izhod.
- `-e` ali `-edit` pred povezovanjem uporabniku ponudi, da uredi vrstni red ali imena sekcije in njihovih simbolov.
- `-h` ali `-help` izpiše pomoč za uporabo in ne nadaljuje s povezovanjem.
- `-g` ali `-gui` požene grafični vmesnik. Če poleg tega nastavimo še kakšno drugo zastavico ali podamo vhodne datoteke, bodo pripadajoča vnosna polja že izpolnjena.

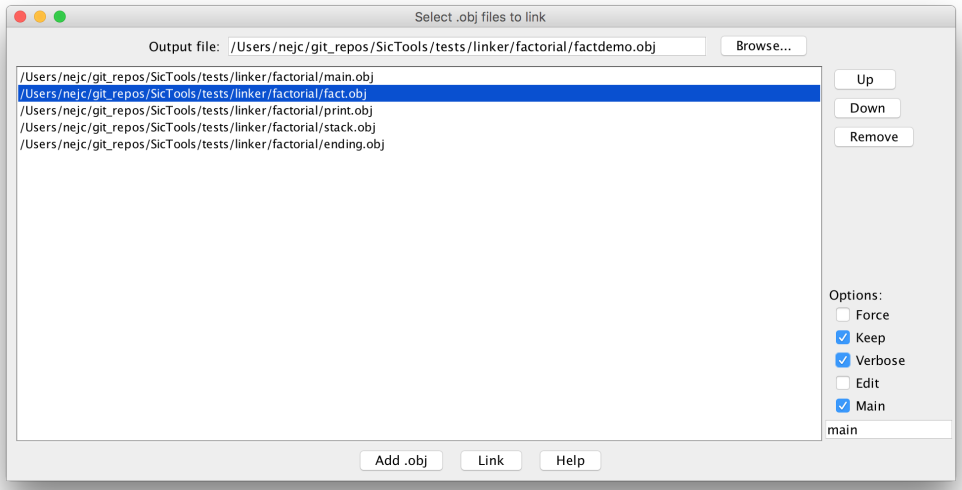
Če smo specificirali opcijo `-e`, se nam v ukazni vrstici nato odpre urejevalnik sekcij. Ukaz `help` nam izpiše vse ukaze, ki jih urejevalnik sprejme, ukaz `done` pa zapre urejevalnik in nadaljuje s povezovanjem. Ostali ukazi, ki so na voljo:

- `list` izpiše seznam sekcij ter število referenc in definicij za vsako od njih
- `info <sekcija>` izpiše reference in definicije, ki nastopajo v specificirani sekciji
- `move <sekcija> <pozicija>` premakne sekcijo na podano pozicijo
- `remove <sekcija>` izbriše podano sekcijo
- `rename <sekcija> <ime>` preimenuje podano sekcijo s podanim imenom

- `remove-ref <sekcija> <referenca>` iz sekcije izbriše referenco
- `rename-ref <sekcija> <referenca> <ime>` preimenuje referenco v podani sekciji
- `remove-def <sekcija> <definicija>` iz sekcije izbriše definicija
- `rename-def <sekcija> <definicija> <ime>` preimenuje definicijo v podani sekciji

3.2.2 Uporaba grafičnega vmesnika

Grafični vmesnik se požene z enakim ukazom, kot če bi ga uporabljali preko ukazne vrstice, le, da dodamo še zastavico `-g` ali `-gui`.



Slika 3.1: Povezovalnikov grafični vmesnik

Na vrhu okna je vnosno polje, ki določi izhodno datoteko. Pot do datoteke lahko vnesemo ročno ali pa uporabimo gumb *Browse...* in izberemo pot s sistemskim dialogom.

V osrednjem polju je seznam vhodnih datotek, na katerega vhodne datoteke dodajamo s klikom na gumb *Add .obj*. Element na seznamu lahko označimo in z gumboma *Up* in *Down* spreminjamo njegovo pozicijo. Vrstni red datotek na seznamu določa, v kakšnem vrstnem redu se bodo sekcije zapisale v izhodno datoteko. Z gumbom *Remove* lahko izbrano vhodno datoteko odstranimo s seznama.

Desno spodaj se nahajajo dodatne opcije, ki jih lahko vklopimo v povezovalniku in imajo enak pomen kot zastavice pri uporabi preko ukazne vrstice:

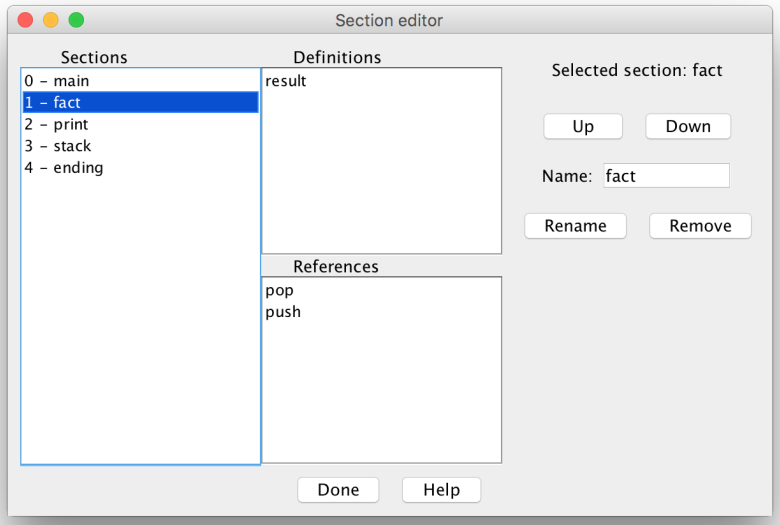
- *Force* omogoči delno povezovanje – povezovanje se bo izvedlo, tudi če kakšna od referenc ni razrešena.
- *Keep* ohrani zunanje definicije – izhodna datoteka bo vsebovala vse D zapise iz vhodnih datotek.
- *Verbose* vklopi izpisovanje informacij o povezovanju na standardni izhod.
- *Edit* omogoči urejanje sekcij preden se povezovanje izvede.
- *Main* prikaže novo vnosno polje, kamor vnesemo ime želene prve oz. glavne sekcije.

Na dnu okna imamo tudi gumb *Help*, ki izpiše pomoč za uporabo in gumb *Link*, ki požene povezovanje.

Urejevalnik sekcij Če smo v vmesniku izbrali možnost *Edit*, se nam po kliku na gumb *Link* odpre novo okno z urejevalnikom sekcij.

Na skrajni levi imamo seznam vseh sekcij, ki so prisotne v vhodnih datotekah. Ko izberemo eno od sekcij, se nam v seznama definicij in referenc, ki sta desno od seznama sekcij, izpišejo vse zunanje definicije in zunanje reference, ki so prisotne v izbrani sekciji. Na skrajni desni se nam pojavijo možnosti za urejanje:

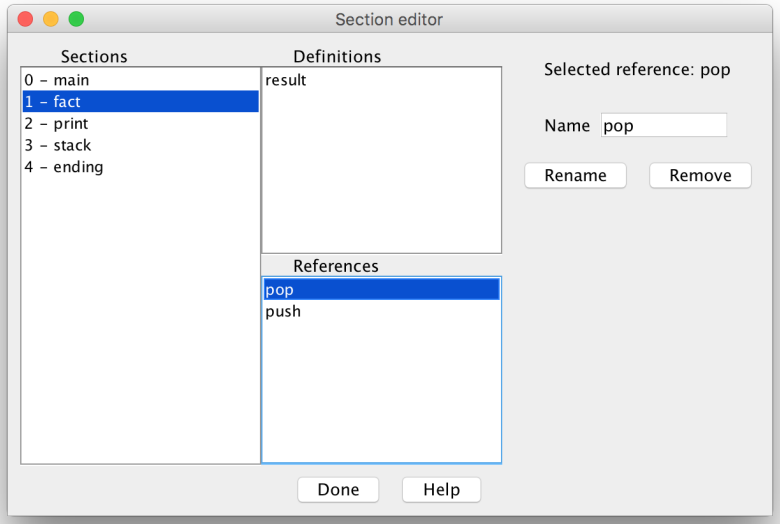
- Z gumboma *Up* in *Down* lahko izbrano sekcijo premaknemo na želeno mesto v seznamu. Vrstni red določa, kako se bodo sekcije naložile v pomnilnik, izvajanje pa se bo vedno začelo pri prvi sekciji.
- V vnosno polje *Name* lahko vnesemo novo ime za izbrano sekcijo in ga potrdimo z gumbom *Rename*. Ime mora biti dolgo 6 ali manj znakov in ne sme biti enako kot ime katere od ostalih sekcij.
- Z gumbom *Remove* sekcijo izbrišemo iz povezovanja.



Slika 3.2: Urejanje kontrolne sekcije

Če želimo urediti zunanji simbol, v levem seznamu izberemo sekcijo, v kateri se simbol nahaja, nato pa ga izberemo v seznamu definicij ali referenc.

Na desni se nam pojavi vnosno polje za preimenovanje in gumb *Remove*, ki delujeta na enak način kot pri urejanju sekcije. Pri preimenovanju zunanjih definicij ni dovoljeno uporabiti imena, ki ga ima že neka druga definicija v isti sekciji. Če zunanjo referenco poimenujemo enako kot neko drugo referenco v



Slika 3.3: Urejanje zunanjega simbola

isti sekciji, se bosta združili – M zapisi preimenovane sekcije bodo uporabili simbole sekcije, ki je že prej imela tako ime.

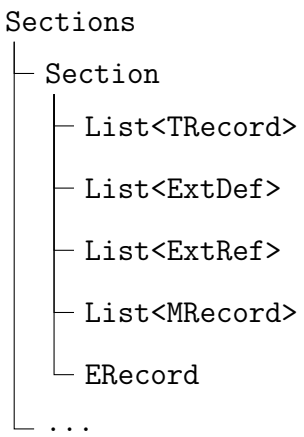
3.3 Uporaba programskega vmesnika

V izdelanem povezovalniku imamo pri povezovanju štiri faze: branje vhodnih datotek, opcijsko urejanje sekcij, povezovanje in izpis izhodne datoteke.

3.3.1 Programska predstavitev sekcij

Prva faza povezovanja je pretvarjanje kontrolnih sekcij iz *.obj* datotek v razrede, s katerimi nato izvajamo povezovanje. Ni pomembno, ali ima ena vhodna datoteka več kontrolnih sekcij ali pa samo po eno – povezovalnik bo vsebino vseh datotek združil v skupno strukturo in nadaljne operacije izvajal na tej strukturi.

Sekcije so predstavljene z drevesno strukturo. Na vrhu je razred `Sections`, ki vsebuje več razredov `Section` – vsak predstavlja eno sekcijo. Znotraj sekcije se nahaja seznam tekstovnih zapisov (razred `TRecord`), seznama zunanjih definicij (razred `ExtDef`) in referenc (razred `ExtRef`), seznam modifikacijskih zapisov (`MRecord`) in pa zaključni E zapis (razred `ERecord`).



Slika 3.4: Struktura programa v povezovalniku

Sections Največja enota v povezovalniku je seznam sekcij – tega predstavlja razred `Sections`, ki vsebuje seznam razredov `Section` in metode za

njihovo urejanje ter spreminjanje. Pomembnejše metode tega razreda:

- `void addSection(Section section)` doda sekcijo na konec seznama,
- `List<Section> getSections()` vrne seznam vseh sekcij,
- `Section getSection(String name)` vrne sekcijo s podanim imenom,
- `void remove(String sectionName)` odstrani sekcijo s podanim imenom,
- `void move(String name, int position)` premakne sekcijo s podanim imenom na želeno mesto v seznamu,
- `void setName(String name)` nastavi ime izhodne sekcije,
- `Section combine(boolean keep)` združi vse sekcije, ki jih razred vsebuje v eno, `boolean keep` pa nam pove, ali naj bodo v izhodni datoteki prisotne zunanje definicije.
- `void clean(boolean force)` odstrani M in R zapise iz sekcij. Če je `force == true`, neporabljene M in R zapise obdrži.
- `void renameDef(String sectionName, String oldName, String newName)` v podani sekciji poišče zunanjo definicijo `oldName` in jo preimenuje v `newName`. Za reference obstaja ekvivalentna metoda `renameRef`.
- `void removeDef(String sectionName, String symbolName)` iz podane sekcije odstrani zunanjo definicijo `symbolName`. Za reference obstaja ekvivalentna metoda `removeRef`.

Section Informacije o sekciji in seznami zapisov, ki so v njej. Razred med drugim vsebuje:

- `String name` – ime sekcije,
- `long start` – začetni naslov,
- `long length` – dolžina sekcije,

- `List<TRecord> tRecords` – seznam tekstovnih zapisov,
- `List<Mrecord> mRecords` – seznam modifikacijskih zapisov,
- `List<ExtDef> extDefs` – seznam zunanjih definicij,
- `List<ExtRef> extRefs` – seznam zunanjih referenc,
- `ERecord eRecord` – končni zapis.

in njihove pripadajoče get in set metode.

TRecord Predstavlja tekstovni zapis in vsebuje samo začetni naslov (`long startAddr`), dolžino (`long length`) in vsebino zapisa (`String text`), ter pripadajoče get in set metode.

ExtDef Predstavlja zunanjo definicijo in vsebuje ime simbola, ki ga definira (`String name`), naslov znotraj svoje sekcije (`long address`) in začetni naslov sekcije, v kateri se nahaja (`long csAddress`).

ExtRef Predstavlja zunanjo referenco in vsebuje le ime simbola, ki ga referencira (`String name`).

MRecord Predstavlja modifikacijski zapis. Vsebuje začetni naslov bajta, na katerega se nanaša (`long start`) in število polzlogov, ki jih je potrebno spremeniti (`int length`), smer popravka (`boolean positive`) in ime simbola, katerega naslov je potrebno prišteti (`String symbol`). Če je `symbol == null`, to pomeni, da je treba prišteti začetni naslov programa – gre za običajni M zapis, ki se uporablja pri prenaslavljanju.

3.3.2 Razred Options

Povezovalnikove opcije se nastavijo preko razreda `Options`, in jih lahko nastavimo na več načinov.

Za nastavitev vseh opcij naenkrat lahko uporabimo naslednji konstruktor:

```
public Options(String output, boolean force, String main, boolean
    verbose, boolean keep) throws LinkerError
```

- `String output` je pot do izhodne datoteke – če je neveljavna, bo konstruktor prožil `LinkerError` napako.
- `boolean force` omogoči prevajanje, tudi če vse reference niso razrešene.
- `String main` je ime glavne oz. prve sekcije. Če je ne želimo specificirati, jo nastavimo na `null`.
- `boolean verbose` omogoči izpisovanje delovanja povezovalnika na standardni izhod.
- `boolean keep` omogoči ohranjanje D zapisov v izhodni datoteki.

Posamezne opcije lahko nastavimo tudi preko get in set metod. Spremenljivke `force`, `main`, `verbose`, `keep`, ki so opisane v prejšnjem odstavku, imajo običajne get in set metode, izhodno datoteko pa je potrebno nastaviti preko dveh različnih – prva določa samo ime datoteke, druga pa celotno pot do nje:

```
public void setOutputName(String outputName)
public String getOutputName()

public void setOutputPath(String outputPath)
public String getOutputPath()
```

V razredu `Options` je tudi metoda, ki iz zastavic nastavljenih v klicu povezovalnika z ukazne vrstice prebere specificirane opcije

```
public int processFlags(String[] args)
```

Sprejme seznam argumentov in vrne število argumentov, ki so bili zaznani kot zastavice. Ostali niso bili prepoznani in so najverjetneje vhodne datoteke. Opcije, ki jih podpira, so opisane v poglavju 3.2.1

3.3.3 Razred Linker

Vse povezovanje se dogaja preko razreda `Linker`, ki mu moramo v konstruktorju podati seznam vhodnih datotek in vključene povezovalnikove opcije:

```
public Linker(List<String> inputs, Options options)
```

Vhodne datoteke so podane v obliki seznama poti do njih (`List<String> inputs`), nastavitve prevajanja pa v razredu `Options`. Poti do vhodnih datotek so lahko absolutne ali relativne glede na direktorij, v katerem se program izvaja.

Enostavno povezovanje Povezovanje lahko izvedemo naenkrat z metodo `link()`, ki iz vhodnih datotek prebere sekcije, jih preuredi glede na nastavljene opcije, izvede povezovanje in vrne izhodno sekcijo v obliki razreda `Section`.

Povezovanje z vmesnim urejanjem sekcij Če želimo urejati razred `Sections`, ki se prebere iz vhodnih datotek, lahko povezovanje izvedemo v dveh delih. Najprej pokličemo Linkerjevo metodo `parse()`, ki prebere vhodne datoteke in nam vrne razred `Sections`, na katerem lahko izvajamo želene spremembe, npr. spremenimo ime kakšnega simbola, spremenimo vrstni red sekcij...

Ko želimo nadaljevati s povezovanjem, uporabimo Linkerjevo metodo `passAndCombine(Sections sections)`, ki ji podamo spremenjen razred `Sections`, vrne pa nam končno povezano sekcijo v obliki razreda `Section`.

Isti dve metodi sta uporabljeni znotraj metode `link()`, le da sta uporabljeni ena za drugo, brez vmesnega urejanja sekcij.

Poglavje 4

Pregled kode in delovanja

Povezovalnik je zasnovan tako, da ga je mogoče uporabljati kot del zbirke SicTools. Njegova razredna struktura je zato podobna kot v omenjeni zbirki, še vedno pa ne uporablja nobene njene kode. Zaradi tega je povezovalnikove razrede mogoče uporabiti povsem samostojno ali pa jih vključiti v nek drug program.

Vsa koda se nahaja v paketu *sic*, tako kot je to narejeno v SicTools. Na tem nivoju se nahaja razred `Link`, preko katerega se prevajalnik lahko požene. Poleg njega imamo na tem nivoju še paket *link*, ki vsebuje vso kodo, potrebno za delovanje povezovalnika.

Paket *sic.link* vsebuje tri razrede: `Linker`, njegova uporaba je opisana v poglavju 3.3.3, `Options`, ki je opisan v poglavju 3.3.2 in pa razred `LinkerError` – napaka, ki se proži, ko gre v povezovalniku kaj narobe. Poleg teh razredov vsebuje še pakete *section*, *ui*, *utils* in *visitors*.

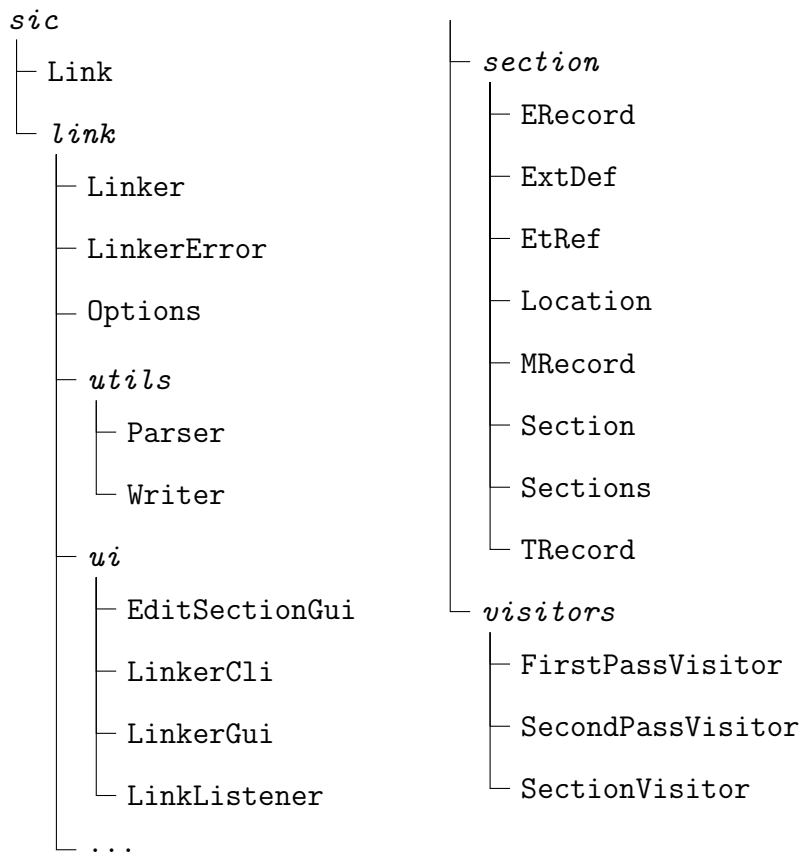
Paket *sic.link.section* vsebuje vse razrede za predstavitev sekcij, ki so podrobneje opisani v poglavju 3.3.1. Poleg njih vsebuje še razred `Location`, ki predstavlja položaj v vhodni datoteki – uporablja se zato, da ob težavah pri povezovanju uporabniku pomaga najti napako v objektni kodi.

Paket *sic.link.ui* vsebuje vse razrede za interakcijo z uporabnikom, tako za grafični vmesnik, kot za vmesnik preko ukazne vrstice. To so razredi: `EditSectionGui`, ki skrbi za grafični urejevalnik sekcij; `LinkerCli`, ki skrbi za

tekstovni urejevalnik sekcij; `LinkerGui`, ki vsebuje kodo za grafični vmesnik povezovalnika; ter `LinkListener` in `SectionEditListener` vmesnika, ki se pri grafičnem delu uporabljata za obveščanje med komponentami.

Paket *sic.link.utils* vsebuje dva razreda: `Parser` in `Writer`, ki skrbita za pretvarjanje objektnih datotek v kodo in nazaj.

Paket *sic.link.visitors* vsebuje kodo za oba prehoda povezovalnika. Implementirana sta z obiskovalcem, ki je definiran v abstraktnem razredu `SectionVisitor`. Implementaciji prvega in drugega prehoda sta v razredih `FirstPassVisitor` in `SecondPassVisitor`.



Slika 4.1: Povezovalnikovi razredi

4.1 Razred Link

Razred `Link` skrbi za zagon samostojnega povezovalnika in procesiranje zastavic pri poganjanju preko ukazne vrstice. Vsebuje samo metodo `main(String[] args)`, znotraj katere najprej iz ukazovih argumentov kreira razred `Options` (preko njegove metode `processFlags()`), in sestavi seznam vhodnih datotek. Glede na izbrane možnosti nato požene grafični ali tesktovni vmesnik preko razreda `LinkerGui` ali `LinkerCli`. Če se znotraj metode `main()` proži kakšna `LinkerError` izjema, njeno vsebino izpiše na sistemski izhod za napake (`System.err`).

4.2 Razred Linker

Uporaba razreda `Linker` je opisana v poglavju 3.3.3, na tem mestu pa je podrobneje opisano delovanje njegovih metod `parse` in `passAndCombine`.

Metoda parse je odgovorna za pretvorbo sekcij iz vhodnih datotek v tako obliko, da so pripravljene na povezovanje.

Najprej ustvari novo instanco razreda `Sections`, nato pa iz vsake vhodne datoteke pridobi seznam njenih sekcij in jih doda vanj. Seznam kontrolnih sekcij v vhodni datoteki dobi preko razreda `Parser` oz. njegove metode `parse()`, ki je opisana nekaj odstavkov nižje. Če v nobeni od vhodnih datotek ni uspela dobiti sekcij, javi napako.

Zatem preuredi sekcije tako, da je glavna sekcija (`options.getMain()`) na prvem mestu. To naredi z uporabo metode `move` v razredu `Sections`.

Če je bilo v nastavitvah povezovalnika specificirano ime izhodne datoteke, se na tem mestu vzame prvih 6 znakov imena (brez končnice *.obj*) in se ga nastavi kot ime izhodne sekcije z metodo `sections.setName`)).

Napolnjen, preurejen in poimenovan razred `Sections` se nato vrne kot rezultat metode.

Metoda passAndCombine z uporabo obiskovalcev iz paketa *sic.link.visitors* in razreda `Sections` podane kontrolne sekcije poveže v eno.

Metoda kot argument sprejme `Sections section`, ki je izhod iz prejšnje metode. Na začetku ustvari tabelo zunanjih simbolov kot `Map<String, ExtDef>`. To tabelo poda v konstruktor obiskovalca `FirstPassVisitor`, na katerem zatem kliče metodo `visit(sections)`. Ta opravi prvi prehod in pri tem tabelo napolni z zunanjimi simboli.

Napolnjeno tabelo, skupaj z imenom izhodne sekcije in nastavitvami povezovalnika, poda v konstruktor obiskovalca `SecondPassVisitor`, na katerem nato prav tako pokliče `visit(sections)`. Ta sekcije poveže v eno in razreši njihove medsebojne reference.

Povezanim sekcijam prečisti M in R zapise, tako da pokliče `sections.clean()`. Ta metoda bo poskrbela, da se razrešene reference odstranijo iz razreda, prav tako pa pregleda M zapise in odstrani pare, ki si med seboj nasprotujejo – eden prišteje začetni naslov, drugi pa ga odšteje.

Zatem lahko pokliče metodo `combine(boolean keep)` na povezanem razredu `Sections`. Ta vrne eno sekcijo v obliki razreda `Section`, ki jo dobi tako, da vse zapise iz vseh sekcij po vrsti zloži v novo sekcijo. Njeno dolžino izračuna kot seštevek dolžin vseh sekcij, njeno ime pa dobi iz razreda `Sections` – to je bilo prebrano iz uporabnikovih ukazov, sicer pa je kar ime prve sekcije.

Združeno sekcijo vrne kot rezultat metode.

Oba prehoda sta podrobneje opisana na koncu tega poglavja.

4.3 Razred LinkerError

Razred `LinkerError` razširja vgrajeni razred `Throwable` in se uporablja za vse napake, ki se zgodijo znotraj povezovalnika. Na voljo so trije različni konstruktorji, ki na različne načine zgradijo sporočilo v razredu `Throwable`:

```
public LinkerError() { super(""); }
```

```
public LinkerError(String phase, String msg) {
   super(phase + ": " + msg);
}

public LinkerError(String phase, String msg, Location loc) {
   super(phase + ": " + msg + " - " + loc.toString());
 }
```

Pri prvem se vsebine sploh ne določi, pri drugem mu podamo, v kateri fazi povezovanja smo in kakšno je sporočilo, pri tretjem pa poleg tega dodamo še razred `Location`, ki predstavlja položaj v eni izmed vhodnih datotek. Če sporočila o napakah gradimo na ta način, lahko uporabnik približno ve, v katerem delu povezovalnika se je napaka zgodila ali kateri del vhodne datoteke je napako povzročil. Več informacij je v samem besedilu (`String msg`).

4.4 Paket *sic.link.utils*

V tem paketu se nahajata dva razreda, ki sta odgovorna za pretvarjanje med objektnimi datotekami in razredi za predstavitev kontrolnih sekcij.

4.4.1 Razred Parser

Ustvari se ga preko konstruktorja, ki mu je potrebno podati pot do vhodne *.obj* datoteke (kot `String`) in razred `Options` z nastavitvami.

```
public Parser(String input, Options options) {
   this.input = input;
   this.options = options;
}
```

Vsebuje le eno metodo `parse()`, ki vrne seznam sekcij, prebranih v vhodni datoteki (`List<Section>`), v primeru napačno oblikovane objektne datoteke pa proži `LinkerError`. Metoda po vrsti gradi seznam razredov `Section`, tako da po vrsti bere vrstice objektne datoteke in gradi razrede za predstavitev

sekcij, ki so opisani v poglavju 3.3.1. Vsakemu od njih doda tudi razred `Location`, ki vsebuje ime vhodne datoteke in vrstico – z njegovo pomočjo lahko uporabniku v primeru težav sporoči, na katerem delu vhodne datoteke naj išče napako.

4.4.2 Razred Writer

Ustvari se ga preko konstruktorja, ki mu podamo razred `Section`, z izhodno sekcijo in razred `Options` s prevajalnikovimi nastavitvami.

```
public Writer (Section section, Options options) {
   this.options = options;
   this.section = section;
}
```

Vsebuje metodo `write()`, ki izpiše vsebino podane sekcije v objektno datoteko in jo vrne v obliki razreda `File`. Če ime izhodne datoteke ni podano v razredu `Options`, se datoteko poimenuje z imenom sekcije in pripono *_ln.obj*.

4.5 Paket *sic.link.ui*

V tem paketu so vsi razredi, ki skrbijo za interakcijo z uporabnikom in pa dva vmesnika (javanski `interface`), ki se uporabljata pri grafičnem povezovalniku.

4.5.1 Razred LinkerCli

Ta razred vsebuje dve statični metodi, ki skrbita za upravljanje z razredom `Linker` in za urejenje sekcij, kadar povezovalnik poženemo v ukazni vrstici.

Metoda link

Prva metoda, `File link(Options options, List<String> inputs)`, sprejme razred `Options` z nastavitvami in seznam vhodnih datotek, vrne pa razred

`File`, ki predstavlja izhodno datoteko. Iz podanih parametrov metoda najprej ustvari razred `Linker`. Če v nastavitvah ni zahtevano vmesno urejanje sekcij, se na razredu `Linker` pokliče metoda `link()`, ki vrne povezano sekcijo v obliki razreda `Section`. Nato se kreira razred `Writer`, ki mu podamo povezano sekcijo in razred z nastavitvami, na njem pa se pokliče metoda `write()`, ki nam vrne povezano datoteko. Povezana datoteka je izhod iz metode `link()`.

Če zahtevamo tudi vmesno urejanje, se na razredu `Linker` najprej pokliče metoda `parse()`, ki nam vrne prebrane kontrolne sekcije v obliki razreda `Sections`. Te nato podamo metodi `sectionEdit(Sections sections)`, ki požene tekstovni urejevalnik sekcij in je opisana v naslednjem odstavku. Izhod iz nje je prav tako razred `Sections`, ki pa vsebuje že urejene in preimenovane kontrolne sekcije in simbole in lahko na njem izvedemo povezovanje. To storimo tako, da ga podamo kot argument Linkerjevi metodi `passAndCombine(Sections sections)`, iz katere nato dobimo končno povezano sekcijo. To nato, tako kot pri prejšnji opciji brez vmesnega urejanja, preko razreda `Writer` izpišemo v datoteko in pripadajoči `File` vrnemo iz metode `Link`.

Metoda sectionEdit

Druga metoda, `Sections sectionEdit(Sections sections)`, sprejme prebrane kontrolne sekcije, ki jih uporabnik lahko preko tekstovnega vmesnika uredi, metoda pa jih urejene vrne kot rezultat. Vmesnik uporablja `BufferedReader`, ki s standardnega vhoda bere uporabnikove ukaze – po eno vrstico naenkrat. Vrstico nato razdeli na besede in preko `switch` stavka na prvi vtipkani besedi razbere uporabnikov ukaz. Če besede, ki sledijo, niso v pravi obliki ali pa ne predstavljajo smiselnega ukaza, bo urejevalnik zaznano napako izpisal na standardni izhod.

Ukazi `help`, `list`, `info <sekcija>` in `done` so implementirani kar znotraj te metode, saj je pri njih potreben le izpis na zaslon. Ostali ukazi: `move`, `remove`, `rename`, `remove-ref`, `remove-def`, `rename-ref`, `rename-def` pa so

implementirani s klici ustreznih metod v razredu `Sections`. Preden vmesnik pokliče ustrezno metodo, samo preveri, ali je bilo podano pravilno število argumentov – ostale napake javi klicana metoda kot `LinkerError`, vmesnik pa jih ujame in izpiše na zaslon. Omenjene metode iz razreda `Sections` so predstavljene v poglavju 3.3.1.

4.6 Paket *sic.link.visitors*

Obiskovalec `SectionVisitor` je napisan tako, da obišče vse zapise v sekcijah, vsebovanih v razredu `Sections`. To naredi tako, da na vsakem od njegovih elementov kliče metodo `accept(SectionVisitor visitor)`, ki je vedno enaka:

```
public void accept(SectionVisitor visitor) throws LinkerError {
   visitor.visit(this);
}
```

Abstraktni razred `SectionVisitor` obišče vse sekcije znotraj razreda `Sections` in nato vse zapise v posamezni sekciji.

```
public abstract class SectionVisitor {
   public void visit(Sections sections) throws LinkerError {
      for (Section section : sections.getSections())
         section.accept(this);
   }

   public void visit(Section section) throws LinkerError {
      if (section.getExtDefs() != null)
         for (ExtDef extDef : section.getExtDefs())
            extDef.accept(this);

      if (section.getExtRefs() != null)
         for (ExtRef extRef : section.getExtRefs())
            extRef.accept(this);
```

```
      if (section.getTRecords() != null)
         for (TRecord tRecord : section.getTRecords())
            tRecord.accept(this);

      if (section.getMRecords() != null)
         for (MRecord mRecord : section.getMRecords())
            mRecord.accept(this);
   }
   public void visit(ExtDef extDef) throws LinkerError {}
   public void visit(ExtRef extRef) throws LinkerError {}
   public void visit(TRecord tRecord) throws LinkerError {}
   public void visit(MRecord mRecord) throws LinkerError {}
   public void visit(ERecord eRecord) throws LinkerError {}
}
```

Tega obiskovalca nato razširjata razreda `FirstPassVisitor` in `SecondPassVisitor`, ki na novo definirata tiste njegove metode, ki jih potrebujeta. Oba prehoda se kličeta iz metode `Section passAndCombine(Sections sections)` v razredu `Linker`.

4.6.1 Prvi prehod

Prvi prehod je implementiran z razredom `FirstPassVisitor`. Pri svojem delovanju uporablja tabelo kontrolnih sekcij, tabelo zunanjih simbolov in spremenljivko z naslovom trenutne sekcije. Izdelan je po psevdokodi iz poglavju 2.2.2 – z nekaj razlikami, saj v psevdokodi prvi prehod bere neposredno iz vhodnih datotek, obiskovalec pa dela z objektnimi predstavitvami sekcij. Naslovi sekcij, simbolov in tekstovnih zapisov se popravijo že kar v prvem prehodu, ker že imamo vse potrebne informacije – v psevdokodi se to naredi šele tik pred izpisom podatkov v datoteko.

Tabela kontrolnih sekcij je implementirana kot `Map<String, Section> csTable`, ki preslika ime sekcije na pripadajoč razred `Section`, tabela zunanjih simbolov je implementirana kot `Map<String, ExtDef> esTable`. Kon-

struktorju se poda tabela `esTable`, ki jo je potrebno kreirati že prej (zunaj obiskovalca). V konstruktorju se tudi kreira tabela `csTable`, spremenljivka `csAddr` pa se nastavi na 0.

```
public FirstPassVisitor(Map<String, ExtDef> esTable) {
   this.esTable = esTable;
   csTable = new HashMap<>();
   csAddr = 0;
}
```

Na novo definirana metoda `visit(Section section)` opravi naslednje korake:

1. Nastavi naslov obiskovane sekcije na vrednost `csAddr`.
2. V `csTable` vstavi obiskano sekcijo – če je isto ime že v tabeli, javi napako.
3. Ime sekcije doda tudi v `esTable`, saj je to tudi zunanji simbol.
4. Obišče vse zunanje definicije, tako da pokliče `accept` na vseh elementih seznama `extDef`, ki ga dobi z klicanjem metode `getExtDefs()` na trenutno obiskani sekciji.
5. Uredi T zapise v sekciji glede na začetni naslov – T zapise dobi z metodo `getTRecords()` na obiskani sekciji.
6. Kliče metodo `accept` na vseh T zapisih iz prejšnjega koraka.
7. Poveča spremenljivo `csAddr` za dolžino sekcije.

Na ostalih elementih sekcije – M zapisih, R zapisih in končnem zapisu se metoda `accept` ne pokliče, saj jih pri tem prehodu ne potrebujemo.

Na novo definifana metoda `visit(ExtDef extDef)` nastavi naslov kontrolne sekcije za obiskani zapis (`extDef.setCsAddress(csAddr)`) in ga nato zapiše v `esTable`. Če simbol s tem imenom v tabeli že obstaja, se proži napaka.

Na novo definirana metoda `visit(TRecord tRecord)` samo poveča začetni naslov T zapisa za `csAddr`.

Ko je prvi prehod končan, imamo napolnjeno tabelo kontrolnih sekcij in tabelo zunanjih simbolov. Tabele kontrolnih sekcij kasneje ne potrebujemo več, saj so potrebne informacije shranjene že v razredih `Section`, `ExtDef` in `TRecord` – uporabna je bila predvsem za zagotavljanje unikatnih imen sekcij. Tabela zunanjih simbolov je bila kreirana zunaj obiskovalca, zato lahko do nje dostopamo in jo podamo kot argument še v drugi prehod.

4.6.2 Drugi prehod

Drugi prehod je implementiran z razredom `SecondPassVisitor` in je odgovoren za popravljanje T zapisov glede na informacije v M zapisih. Za izvedbo prehoda potrebuje ime programa (`String progname`), tabelo zunanjih simbolov (`Map <String, ExtDef> esTable`) in nastavitve povezovalnika (`Options options`), ki mu jih podamo v konstruktorju.

```
public SecondPassVisitor(String progname, Map<String, ExtDef>
    esTable, Options options) {
        this.progname = progname;
        this.esTable = esTable;
        this.options = options;
    }
```

Na novo definirana metoda `visit(Section section)` nastavi globalno spremenljivko `currSection` na trenutno obiskano sekcijo, nato pa na vseh njenih M zapisih, ki jih dobi z `section.getMRecords()`, kliče metodo `accept`.

Na novo definirana metoda `visit(MRecord mRecord)` opravi vse ostale operacije v tem prehodu.

1. Če M zapis ne vsebuje nobenega simbola ali pa vsebuje ime programa, ga izpusti – gre za običajni prenaslovitveni M zapis, ki ga potrebuje nalagalnik.

2. V tabeli poišče simbol, ki ga referencira obiskani M zapis, in ga shrani v spremenljivko `ExtDef symbol`. Če simbol ne obstaja vrne napako, oziroma ga le izpusti, če je vklopljena možnost delnega povezovanja (`-force` zastavica).

3. Ugotovi, kateri naslov je potrebno popraviti, tako da sešteje začetni naslov M zapisa (`mRecord.getStart()`) in začetni naslov trenutne sekcije (`currSection.getStart()`)

4. V seznamu T zapisov (`currSection.getTRecords()`) poišče zapis, ki vsebuje ta naslov. To naredi preko metode `contains(long addr)` v razredu `TRecord`, ki z začetnim naslovom in dolžino ugotovi, ali je naslov vsebovan v T zapisu. Če naslov ni vsebovan v nobenem od T zapisov, javi napako.

5. Poišče znake v vsebini T zapisa, ki jih je potrebno popraviti. Začetek in konec se izračunata kot:

   ```
   int start = (int)(fixAddress - fixRecord.getStartAddr()) * 2;
   start = start + 6 - mRecord.getLength();
   int end = start + mRecord.getLength();
   ```

   kjer je `fixAddress` začetni naslov besede (24 bitov oz. 6 znakov), na katero kaže M zapis, `mRecord.getLength()` pa število znakov, ki jih je potrebno popraviti. Število znakov se šteje od konca besede naprej.

6. Najdeni del T zapisa pretvori v spremenljivko tipa `long` in mu glede na smer popravka v M zapisu (`mRecord.isPositive()`) prišteje ali odšteje naslov simbola.

7. Popravljeni naslov pretvori nazaj v šestnajstiške znake in z njimi nadomesti prej izračunano mesto v T zapisu,

8. Za konec spremeni simbol v M zapisu v ime programa, začetek M zapisa pa poveča za začetni naslov trenutne sekcije.

Po koncu drugega prehoda so v kodi popravljeni vsi naslovi, vsi M zapisi pa se nanašajo le še na začetek programa. Na tem mestu je potrebno le še odstraniti odvečne M zapise in združiti sekcije v eno, za kar poskrbi razred `Linker`.

Poglavje 5

Primer povezovanja programa

Kot primer je opisan enostaven program, ki računa fakultete za prvih 10 naravnih števil in jih izpisuje na standardni izhod. Fakulteta se računa rekurzivno z uporabo sklada, ki je implementiran v ločeni datoteki *stack.asm*, prav tako pa je v ločeni datoteki *print.asm* implementirano izpisovanje na zaslon. Sama rutina za računanje fakultete je napisana v datoteki *fact.asm*, zanka, ki jo desetkrat pokliče, pa je implementirana v datoteki *main.asm*. Na koncu vsake sekcije je rezerviranih 64 besed, ki vedno ostanejo prazne – zaradi tega lahko, ko program naložimo v simulator, takoj vidimo, kje se začne nova sekcija. V realnem programu tega seveda ne bi dodali. Na zadnjem mestu je dodana še datoteka *ending.asm*, ki nam pove, kje se konča koda in se lahko začne graditi sklad.

main.asm:

```
1  main   START 0
2         EXTREF result
3         EXTREF fact
4         EXTREF print
5         EXTREF stinit
6         EXTREF end
7
8         +LDA #end
9         +JSUB stinit   . postavi sklad na naslovu iz end
10 loop   LDA #1
11        +STA result    . result = 1
12        LDA i
```

```
13          ADD #1          . i++
14          STA i
15          COMP #10
16          JEQ halt        . if i == 10 then halt
17          +JSUB fact      . poklici fact(i)
18          +LDA result
19          +JSUB print     . izpisi rezultat
20          J loop
21  halt    J halt
22
23  i       WORD 0
24  gap     RESW 64
```

fact.asm:

```
1   fact    START 0
2           EXTREF push
3           EXTREF pop
4           EXTDEF result
5
6           COMP #1
7           JEQ exit    . if A == 1 then exit
8           STA tmpA    . shrani A na tmpA
9           STL tmpL    . shrani L na tmpL
10
11          +JSUB push  . push A
12          LDA tmpL
13          +JSUB push  . push L
14          LDA tmpA    . A = tmpA
15          SUB #1      . A--
16          JSUB fact   . rekurzivni klic
17
18          +JSUB pop   . pop L
19          STA tmpL    . shrani na tmpL
20          +JSUB pop   . pop A
21
22          MUL result
23          STA result  . result = result * A
24          LDL tmpL    . obnovi L
25          RSUB
26  exit    RSUB
27
28  result  WORD 1
29  tmpA    RESW 1
30  tmpL    RESW 1
31  gap     RESW 64
```

print.asm:

```
print   START 0
        STA buffer

prtbuf  LDA buffer
        SUB max         . najdi prvo potenco 10, vecjo od buffer
        COMP #0
        JLT found
        LDA max
        MUL #10
        STA max
        J prtbuf
found   LDA max         . deli z 10 in izpisi buffer/max
        DIV #10
        STA max
        COMP #0
        JEQ exit
        LDA buffer
        DIV max
        ADD #48         . ASCII 0
        WD #1
        SUB #48
        MUL max
        STA tmp
        LDA buffer
        SUB tmp
        STA buffer
        J found
exit    LDA #1
        STA max         . max = 1
        LDA #10         . ASCII newline
        WD #1
        RSUB
max     WORD 1
tmp     RESW 1
buffer  RESW 1
gap     RESW 64
```

stack.asm:

```
stack   START 0
        EXTDEF stinit
        EXTDEF push
        EXTDEF pop
stinit  STA stackptr    . inicializira sklad na naslovu iz A
        RSUB
```

```
push    STA @stackptr  . spravi vrednost iz A na sklad
        LDA stackptr
        ADD #3
        STA stackptr
        RSUB
pop     LDA stackptr   . spravi vrednost s sklada v A
        SUB #3
        STA stackptr
        LDA @stackptr
        RSUB

stackptr RESW 1        . kazalec na sklad
```

ending.asm:

```
ending START 0
       EXTDEF end
end WORD 17
       END ending
```

Zaradi razdelitve po datotekah so posamezni deli programa med seboj neodvisni - če bi želeli uporabiti različne implementicije sklada, bi zamenjali samo njegovo datoteko in poskrbeli, da nova definira enake zunanje simbole (`stinit`, `push` in `pop`). Na podoben način bi lahko zamenjali rutino za izpisovanje števil s tako, ki bi namesto na standardni izhod izpisovala na tekstovni zaslon. Primer uporabe tega v realnem svetu bi bil, da rutino za izpisovanje števil priskrbi nekdo drug, npr. operacijski sistem.

Ko zbirnik *.asm* datoteke prevede v objektno kodo, iz njih dobimo naslednje *.obj* datoteke:

main.obj:

```
Hmain  0000000000F3
Rend   fact  print resultstinit
T0000001E011000004B1000000100010F10000003201E1900010F201829000A33200F
T00001E154B100000031000004B1000003F2FDB3F2FFD000000
M00000105+end
M00000505+stinit
M00000C05+result
M00001F05+fact
M00002305+result
M00002705+print
E000000
```

fact.obj:

```
Hfact 000000000103
Dresult00003A
Rpop  push
T0000001D2900013320310F20341720344B10000003202D4B1000000320231D0001
T00001D1D4B2FE04B1000000F20194B10000023200C0F20090B200C4F00004F0000
T00003A03000001
M00000D05+push
M00001405+push
M00002105+pop
M00002805+pop
E000000
```

print.obj:

```
Hprint 000000000123
T0000001E0F205D03205A1F20512900003B200C03204821000A0F20423F2FE803203C
T00001E1E25000A0F203629000033202103203327202A190030DD00011D003023201E
T00003C1E0F201E03201E1F20180F20183F2FD00100010F200901000ADD00014F0000
T00005A03000001
E000000
```

stack.obj:

```
Hstack 000000000027
Dpop  000015 push 000006 stinit000000
T0000001E0F20214F00000E201B0320181900030F20124F000003200C1D00030F2006
T00001E060220034F0000
E000000
```

ending.obj:

```
Hending000000000003
Dend  000000
T00000003000011
E000000
```

Iz D in R zapisov lahko vidimo, katere zunanje simbole sekcije definirajo in katere uporabljajo. Vsako ime sekcije je tudi zunanji simbol, zato v datoteki *print.obj* ni nobenega R zapisa, *main.obj* pa se vseeno lahko sklicuje na simbol `print`. Če si pozorno ogledamo T zapise v *main.obj* in *fact.obj*, lahko vidimo, da imajo na določenih mestih nize ničel – te so na mestih, kjer se uporabljajo referencirani simboli in bodo med povezovanjem nadomeščeni z dejanskimi naslovi. Za to bodo poskrbeli M zapisi, ki povedo, kateri simbol

je potrebno vstaviti kam.

Datoteke v eno povežemo tako, da poženemo povezovalnik z argumenti

```
-o linked.obj main.obj fact.obj print.obj stack.obj ending.obj
```

Zgornji argumenti pomenijo, da se bo izhodna datoteka imenovala *linked.obj*, vhodne datoteke pa so v podanem zaporedju.

linked.obj:

```
Hlinked000000000343
T0000001E011003404B1003190100010F10012D03201E1900010F201829000A33200F
T00001E154B1000F30310012D4B1001F63F2FDB3F2FFD000000
T0000F31D2900013320310F20341720344B10031F03202D4B10031F0320231D0001
T0001101D4B2FE04B10032E0F20194B10032E23200C0F20090B200C4F00004F0000
T00012D03000001
T0001F61E0F205D03205A1F20512900003B200C03204821000A0F20423F2FE803203C
T0002141E25000A0F203629000033202103203327202A190030DD00011D003023201E
T0002321E0F201E03201E1F20180F20183F2FD00100010F200901000ADD00014F0000
T00025003000001
T0003191E0F20214F00000E201B0320181900030F20124F000003200C1D00030F2006
T000337060220034F0000
T00034003000011
M00000105
M00000505
M00000C05
M00001F05
M00002305
M00002705
M00010005
M00010705
M00011405
M00011B05
E000000
```

Vidimo lahko, da so po povezovanju T zapisi ostali zelo podobni, le da so združeni v eni datoteki in da imajo drugačne naslove. Prav tako so nizi ničel v njih nadomeščeni s pravimi naslovi, v izhodni datoteki pa ni več D in R zapisov. M zapisi, ki so ostali, se zdaj ne nanašajo več na posamezne simbole, ampak na začetni naslov programa in se uporabijo le za prenaslavljanje.

Takšno povezano datoteko lahko naložimo in poženemo, poleg tega pa je še vedno prenaslovljiva. Če bi želeli povezati vse datoteke, razen *print.obj*, bi uporabili argumente

```
-f -o delno.obj main.obj fact.obj stack.obj
```

Za razliko od prej je uporabljena opcija `-f`, ki pomeni, da je dovoljeno delno povezovanje, *print.asm* pa ni med vhodnimi datotekami. Pri tem programu bi izpustiti tudi datoteko *ending*, saj ta označuje konec programa.

print.asm:

```
1  Hdelno 00000000021D
2  T0000001E011000004B1001F60100010F10012D03201E1900010F201829000A33200F
3  T00001E154B1000F30310012D4B1000003F2FDB3F2FFD000000
4  T0000F31D2900013320310F20341720344B1001FC03202D4B1001FC0320231D0001
5  T0001101D4B2FE04B10020B0F20194B10020B23200C0F20090B200C4F00004F0000
6  T00012D03000001
7  T0001F61E0F20214F00000E201B0320181900030F20124F000003200C1D00030F2006
8  T000214060220034F0000
9  M00000105+end
10 M00000505
11 M00000C05
12 M00001F05
13 M00002305
14 M00002705+print
15 M00010005
16 M00010705
17 M00011405
18 M00011B05
19 Rend
20 Rprint
21 E000000
```

Za razliko od prejšnjega primera sta v datoteki še vedno zunanji referenca `print` in `end`, pripadajoča M zapisa sta ostala nespremenjena, v prvem in drugem T zapisu pa sta še vedno niza ničel, na katere se M zapisa nanašata. Seveda manjkajo tudi T zapisi iz datotek *print.obj* in *ending.obj*– vrstice od 7 do 10 in pa vrstica 13 v prejšnjem primeru.

Če bi hoteli tako datoteko naložiti in pognati, bi jo morali najprej do konca povezati tako, da bi uporabili argumente

```
-o celota.obj delno.obj print.obj ending.obj
```

Poglavje 6

Zaključek

6.1 Doseženi cilji naloge

Osnovni cilj naloge je bil izdelati povezovalnik, ki ga bo možno uporabiti samostojno ali vključenega v drugih programih. Te cilje nam je z implementiranim povezovalnikom uspelo doseči – povezovalnik je izdelan tako, da se da zanj enostavno napisati nov uporabniški vmesnik (implementirana sta dva primera – grafični in tekstovni), ali pa ga pognati iz drugega javanskega programa. Z nekaj vrsticami kode se ga je dalo vključiti v meni obstoječega SicToolsovega simulatorja, ki tako poleg nalaganja ene objektne datoteke omogoča tudi njihovo povezovanje in nato izvajanje, pri tem pa uporabniku ni treba zapustiti grafičnega vmesnika.

Glede dodatnih funkcionalnosti: povezovalnik omogoča veliko različnih nastavitev, kot so delno povezovanje, določanje prve sekcije in ohranjanje definiranih zapisov. Te so sicer precej enostavne, vendar v določenih situacijah pridejo zelo prav.

Poleg samega povezovanja je v povezovalniku omogočeno tudi urejanje sekcij. Tega se da sicer relativno enostavno izvajati ročno z urejanjem objektnih datotek, vendar sta tako tekstovni kot grafični interaktivni urejevalnik veliko bolj prijazna do uporabnika – sploh če ta ni podrobno seznanjen s formatom objektnih datotek.

6.2 Pomankljivosti in možne izboljšave

Ena od možnih izboljšav, bi bila podpora za absolutne sekcije in sekcije razdeljene z ORG direktivami, ki prestavijo del sekcije na drug naslov v pomnilniku. Pri enostavnih absolutnih sekcijah implementacija ne bi smela biti prezahtevna, za sekcije, razdeljene z ORG direktivami, pa se začnejo pojavljati težave. Že pri razporejanju sekcij v pomnilnik je težko izračunati, kje so v pomnilniku prosta mesta in kako dolge so posamezne sekcije. Glede na to, da take vrste programov povzročajo nevšečnosti, ne ponujajo pa bistvenih prednosti, jih implementirani povezovalnik zaenkrat ne podpira. Takšni programi danes tudi niso pogosti v realnem svetu, oziroma v modernih operacijskih praktično ni potrebe po njih.

Še ena možna izboljšava bi bil lepši grafični vmesnik. Izdelani vmesnik omogoča vse funkcije, ki jih povezovalnik podpira, vendar bi bili lahko položaji elementov in poravnave boljše. Grafični vmesnik je bil izdelan deloma kot demonstracija povezovalnikove prilagodljivosti, deloma pa, da je omogočil lepšo vključitev v SicToolsov grafični simulator. Dobra stran te pomankljivosti je, da je zaradi načina implementacije relativno lahko dodati nov vmesnik ali pa obstoječega nadomestiti – potrebno je spremeniti samo en razred (*sic.link.ui.LinkerGui* oz. *sic.link.ui.EditSectionGui* za urejevalnik sekcij), ali pa ga nadomestiti z novim.

Literatura

[1] Leland L. Beck. *System Software (3rd Ed.): An Introduction to Systems Programming*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1997.

[2] J. Mihelič and T. Dobravec. Sicsim: A simulator of the educational SIC/XE computer for a system-software course. *Computer Applications in Engineering Education*, 23(1):137–146, 2015.

[3] Urša Levičnik. Zbirnik za hipotetični računalnik SIC/XE. Fakulteta za računalništvo in informatiko, Univerza v Ljubljani, Marec 2011. `http://eprints.fri.uni-lj.si/1310/`.

[4] Benjamin Kastelic. Pretvornik med SIC/XE in intel pentium x86 zbirno kodo. Fakulteta za računalništvo in informatiko, Univerza v Ljubljani, September 2012. `http://eprints.fri.uni-lj.si/1845/`.

[5] Klemen Košir. Prevajalnik za programski jezik C za računalnik SIC/XE. Fakulteta za računalništvo in informatiko, Univerza v Ljubljani, September 2015. `http://eprints.fri.uni-lj.si/3055/`.

[6] Klemen Kloboves. Implementacija procesorja SIC/XE na FPGA in podporna sistemska programska oprema. Fakulteta za računalništvo in informatiko, Univerza v Ljubljani, Julij 2014. `http://eprints.fri.uni-lj.si/2610/`.

[7] Dmitry Brant. Assembler and linker for SIC/XE. http://dmitrybrant.com/2003/11/02/assembler-and-linker-for-sicxe, November 2003. Dostopano: Avgust 2016.

[8] Johnny Philavanh. SIC/XE linker. https://github.com/statcode/assembler-linker. Dostopano: Avgust 2016.

[9] Oracle Corporation. Java Platform Standard Edition 8 Documentation. http://http://docs.oracle.com/javase/8/docs/. Dostopano: Avgust 2016.