



Napomena

Realizacija laboratorijske vježbe 9 nije obavezna. Studenti koji uspješno urade ovu vježbu će dobiti dodatne bodove u vrijednosti jedne laboratorijske vježbe.

Uvod

U ovoj laboratorijskoj vježbi ćemo se upoznati sa ARM instrukcijskim setom, te korištenje GNU asemblera za asembliranje koda za ARM arhitekturu. O programskom modelu jezgra ARM Cortex-M0, te instrukcijskom setu se može pročitati u [1].

GNU Assembler

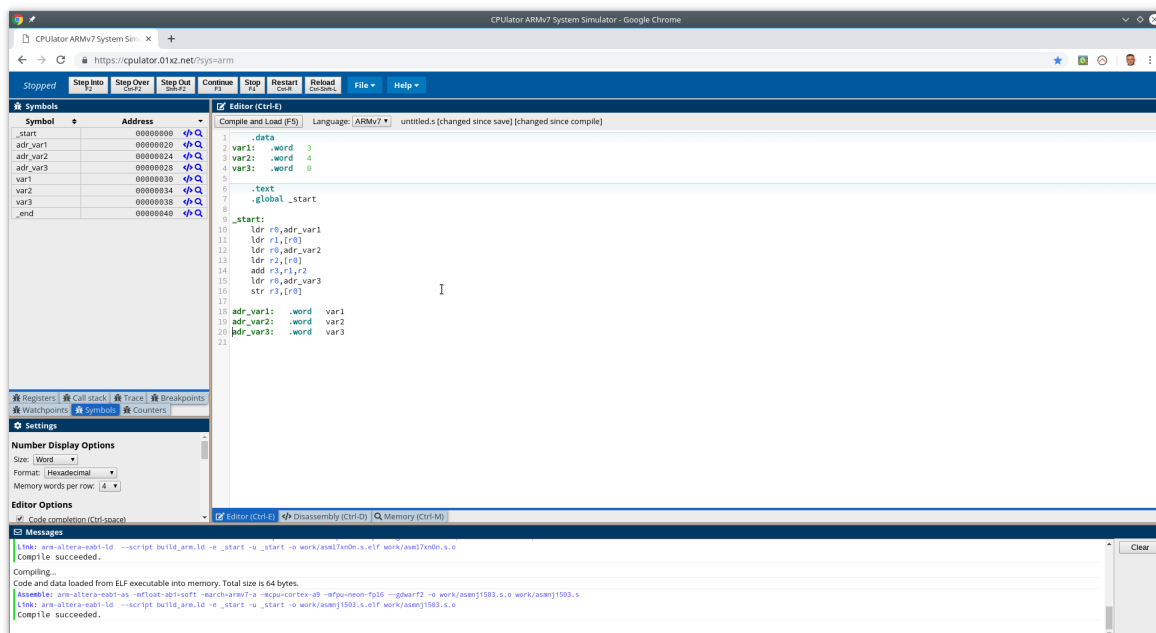
Asembler (asemblerški jezik) predstavlja najniži simbolički nivo reprezentacije aplikacije koja se izvršava na nekom računar. Programski kod pisan u asemblerškom jeziku je potrebno prevesti u mašinski kod za određenu arhitekturu, za što se koristi assembler (prevodilac). Na ARM arhitekturi, dominantna su dva asemblera:

- ARM assembler,
- GNU assembler.

Između ova dva asemblera postoje određene razlike u pogledu sintakse i direktiva, ali su razlike u programskom kodu minorne. U ovoj vježbi ćemo koristiti GNU assembler za prevođenje programskog koda i GNU debugger za debugiranje i analiziranje koda tokom izvršavanja.

Korištenje GNU asemblera u okviru online simulatora

Na linku <https://cpulator.01xz.net/?sys=arm> se nalazi online simulator ARM sistema (i drugih arhitektura) koji koristi GNU assembler za prevođenje programskog koda. Izgled ovog online simulatora je prikazan na slici 1.



Slika 1: Izgled prozora online simulatora ARM arhitekture.

Programski kod unesen u okviru taba *Editor* se prevodi klikom na dugme **Compile and Load**, a izvršavanjem simulacije se upravlja nizom dugmadi u gornjem lijevom uglu prozora. U lijevom dijelu prozora se može pregledati aktualni sadržaj registara, prekidne tačke, definirani simboli i sl. Na tabu *Disassembly* je prikazan kôd dobiven nakon prevođenja, pri čemu je naglašena linija kôda koja će sljedeća biti izvršena (ukoliko je izvršavanje koda zaustavljeno). U ovom tabu je moguće postavljati i uklanjati prekidne tačke. Tab *Memory* prikazuje sadržaj cjelokupne memorije.

Treba imati na umu da ćemo online simulatorom simulirati izvršavanje mašinskog koda direktno na hardveru (bez operativnog sistema).

Korištenje GNU asemblera na emuliranom Raspberry Pi

GNU assembler je već instaliran u okviru image-a distribucije Raspberry Pi OS-a koju smo koristili na QEMU, tako da se programski kod može prevoditi i u okviru emuliranog Raspberry Pi računara. Fajl sa programskim kodom se može kreirati i editirati korištenjem editora *nano*.

Da bi se kreirao izvršni fajl, potrebno je prevesti (asemblirati) i povezati (linkovati) programski kôd.

Programski kôd se prevodi pozivom GNU asemblera:

```
as ime_fajla.s -o ime_fajla.o
```

Izvorni programski kod se nalazi u fajlu `ime_fajla.s`, a nakon prevođenja će biti kreiran fajl `ime_fajla.o`, koji sadrži objektni kod. Kako bi se aplikacija mogla debugirati, potrebno je pri pozivu GNU asemblera uključiti i opciju `-g`. Osim toga, pogodno je generirati i fajl sa detaljnim listingom kôda, koji se može dobiti uključivanjem opcije `-a` pri pozivu GNU asemblera. Konačno, poziv GNU asemblera će biti:

```
as -a -g ime_fajla.s -o ime_fajla.o > ime_fajla.lst
```

Nakon što se kreira fajl sa objektnim kôdom, potrebno ga je povezati, kako bi se dobio fajl sa izvršnim kôdom. Poziv linkera (sa uključenom opcijom `-g`, koja olakšava debugiranje) će biti:

```
ld -g ime_fajla.o -o ime_fajla
```

Pogodno je kreirati *bash* skriptu (npr. `asm.sh`), koja će olakšati pozivanje GNU asemblera i linkera:

```
#!/bin/bash
as -a -g $1.s -o $1.o > $1.lst
ld -g $1.o -o $1
```

Da bi se ova skripta mogla izvršavati, potrebno je modificirati permisije fajla:

```
chmod +x asm.sh
```

Skripta se onda može pozvati kao:

```
./asm.sh ime_fajla
```

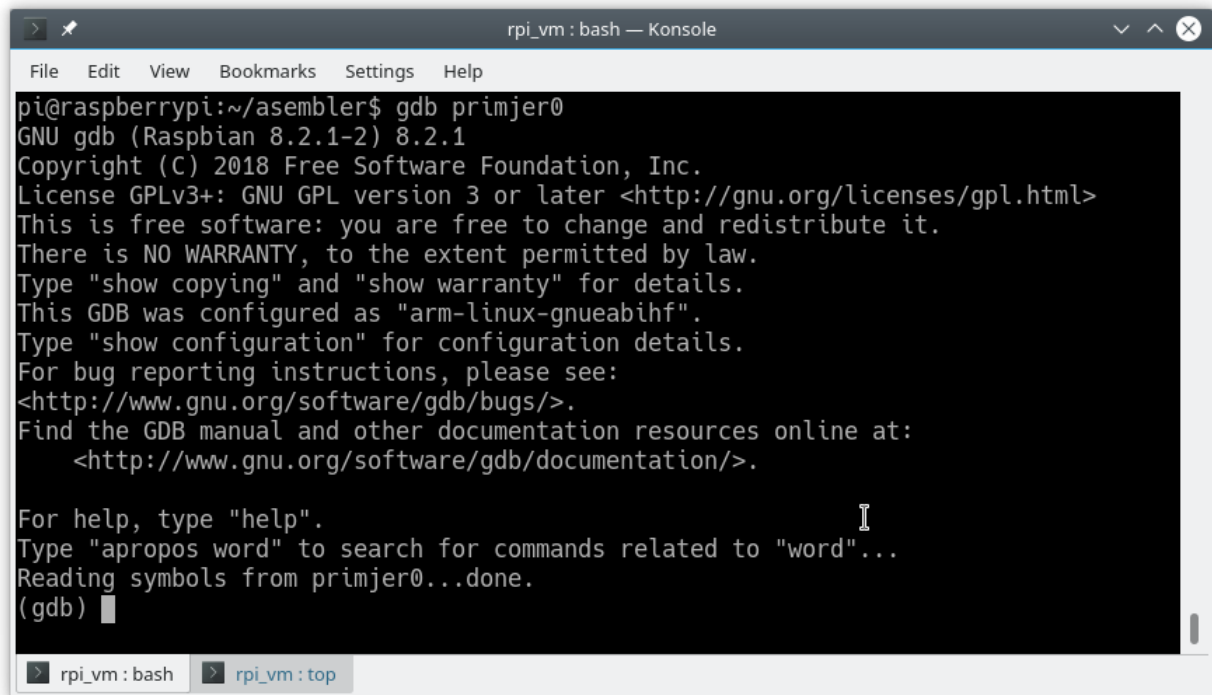
Nakon uspješno okončanog prevođenja i povezivanja, u istom folderu će se nalaziti fajl aplikacije, koji se može izvršiti pozivom `./ime_fajla`.

Korištenje GNU debuggera za praćenje izvršavanja aplikacije

GDB predstavlja sastavni dio GNU toolchain-a i omogućava debugiranje i praćenje izvršavanja aplikacija. Da bi se započelo debugiranje aplikacije `ime_fajla`, potrebno je pozvati GDB:

```
gdb ime_fajla
```

nakon čega će se pojaviti komandni prompt GDB-a, kao na slici 2.



```
rpi_vm : bash — Konsole
File Edit View Bookmarks Settings Help
pi@raspberrypi:~/assembler$ gdb primjer0
GNU gdb (Raspbian 8.2.1-2) 8.2.1
Copyright (C) 2018 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "arm-linux-gnueabi".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from primjer0...done.
(gdb) 
```

Slika 2: Komandni prompt GNU debuggera.

Pojašnjenje komandi GDB-a se može dobiti izvršavanjem komande help, a neke od najvažnijih komandi su:

- `list` - ispis listinga programa (10 linija, a može se navesti i kao `list br_linije` ili `list br_prve_linije, br_krajnje_linije`),
- `br labela` ili `br br_linije` - postavljanje prekidne tačke na adresi definiranoj labelom ili brojem linije,
- `watch $reg` - praćenje promjene vrijednosti u registru `$reg` (`$r0`, `$r1` itd.),
- `run` - izvršavanje aplikacije,
- `info registers` (kraće `info reg`) - ispis sadržaja registara,
- `info breakpoints` (kraće `info br`) - ispis postojećih prekidnih tačaka i praćenih varijabli,
- `info watch` - ispis postojećih praćenih varijabli,
- `next` ili `next br_linija` (kraće `n`) - izvršavanje jedne ili više linija koda (bez ulaska u potprogram),
- `step` (kraće `s`) - izvršavanje jedne linije koda (uz ulazak u potprogram),
- `continue` (kraće `c`) - nastavak izvršavanja programa,
- `print $reg` - ispis trenutnog sadržaja registra (npr. `print $r0`),
- `printf <format_string>, $reg` - formatirani ispis trenutnog sadržaja registra (npr. `printf "%d", $r0`),
- `kill` - prekid izvršavanja programa,

- `quit` - izlazak iz GDB-a.

Ostale komande se mogu pronaći u dokumentaciji.

Tipično sesija započinje postavljanjem prekidne tačke na labelu `_start`, a zatim pokretanjem izvršavanja programa. Nakon što se izvršavanje zaustavi na prvoj liniji koda, moguće je koristiti komande za upravljanje izvršavanjem koda i za prikaz sadržaja registara i memorijskih lokacija.

ZADACI

Napomena

Prilikom razvoja kôdova za zadatke 2.-4. koristiti GNU debugger.

Zadatak 1

Online simulator

Potrebno je napisati programski kôd u assembleru, koji omogućava da se generira prvih N Fibonaccijevih brojeva, pri čemu se ovi brojevi zapisuju u sukcesivne memorijske lokacije. Omogućiti da se konstanta N definira na početku kôda, te testirati aplikaciju za $N=47$ i $N=48$.

Zadatak 2

GNU assembler - QEMU

Potrebno je napisati programski kôd u assembleru, koji omogućava da se preko *stdin* unesu elementi niza kao ASCII karakteri. Broj elemenata niza je definiran konstantom u okviru programa. Nakon unosa elemenata, program treba sortirati elemente niza po rastućem redoslijedu i ispisati sortirani niz na *stdout*.

GNU assembler - Raspberry Pi

Kôd razvijen korištenjem QEMU izvršiti na stvarnom Raspberry Pi u laboratoriji.

Zadatak 3

GNU assembler - QEMU

Potrebno je napisati programski kôd u assembleru, koji omogućava da se preko *stdin* unese cijeli broj, koji će biti upisan u memorijsku lokaciju.

GNU assembler - Raspberry Pi

Kôd razvijen korištenjem QEMU izvršiti na stvarnom Raspberry Pi u laboratoriji.

Zadatak 4

GNU assembler - QEMU

Potrebno je napisati programski kôd u assembleru, koji omogućava da se sa *stdin* unesu elementi niza cijelih brojeva. Broj elemenata niza se unosi na početku, pri čemu je maksimalni broj elemenata definiran konstantom u okviru programa (tako da nije moguće unijeti veći broj elemenata od maksimalnog, nego se u tom slučaju traži ponovni unos broja elemenata niza). Nakon unosa elemenata program treba sortirati elemente niza po rastućem redoslijedu, te

korištenjem poziva C funkcije `printf` ispisati najmanji cijeli broj, najveći cijeli broj, opseg (razliku između najvećeg i najmanjeg cijelog broja) i median. Da bi se fajl sa objektnim kodom povezo sa bibliotekom C funkcija, linker je potrebno pozvati na sljedeći način:

```
ld -g ime_fajla.o -o ime_fajla -dynamic-linker /lib/ld-linux-armhf.so.3 -lc
```

GNU assembler - Raspberry Pi

Kôd razvijen korištenjem QEMU izvršiti na stvarnom Raspberry Pi u laboratoriji.

Napomena

Linux sistemski pozivi omogućavaju pozivanje funkcija Linux kernela. O Linux sistemskim pozivima se može pročitati na linku <http://man7.org/linux/man-pages/man2/syscalls.2.html>. U assembleru se sistemski pozivi koriste tako što se parametri pohrane u odgovarajuće registre, nakon čega se izvrši mašinska instrukcija `swi` (odnosno `svc` - radi se o istoj mašinskoj instrukciji). Rezultat izvršavanja systemske funkcije (ukoliko postoji) se vraća u nekom od registara. Konkretni registri koji se koriste za prosljeđivanje parametara i vraćanje rezultata ovise o arhitekturi, a za ARM arhitekturu se detaljan pregled korištenih registara može naći na linkovima:

- https://syscalls.w3challs.com/?arch=arm_thumb
- https://syscalls.w3challs.com/?arch=arm_strong.

Literatura

- [1] S. Konjicija (2022) *Predavanje Ugradbeni sistemi: Instrukcijski set ARM procesora*
- [2] GNU Toolchain *Dodatni materijali za GNU Toolchain i primjeri*