Uvod u GNU asembler

GNU asembler, deo GNU softverskog paketa binutils, je koristi se za prevođenje asemblerskog jezika, tj. simboličke forme mašinskog jezika u binarni objektni kod. Zvanična dokumentacija za *GNU asembler* može se naći na:

https://sourceware.org/binutils/docs/as/

Većina stvari nije x86_64 specifična, tako da važi i za druge procesorske arhitekture.

Primer programa

Primer programa koji koristi C biblioteku. Objašnjenja pojedinih elemenata programa su u sledećim poglavljima.

```
        .intel_syntax noprefix
        .text
kvadrat:
   # ulazni parametar u edi
   # rezultat u eax
        imul   edi, edi         # edi = edi * edi (signed mult.)
        mov    eax, edi             # eax = edi
        ret                   # povratak u program

        .text
        .globl main          # main je vidljiv linkeru kao ulazna tacka programa
main:  # argc u edi, argv[] u esi
        push   rbx             # sacuvaj rbx (calee saved reg.)
        mov    rdi, QWORD PTR 8[rsi] # rdi = &argv[1]
        call   atoi            # eax = atoi(rdi)
        mov    ebx, eax             # sacuvaj eax za kasnije
        mov    edi, eax            # edi = eax
        call   kvadrat          # eax = kvadrat(edi)
        mov    edx, eax             # edx = eax treci parametar za printf
        mov    esi, ebx            # esi = ebx drugi parametar za printf
        lea    rdi, .format         # rdi = adresa .format, prvi param. za prinf
        mov    eax, 0              # oznacava da nema float parametara za prinf
        call   printf
        mov    eax, 0              # povratni kod iz main
        pop    rbx                # vrati sacuvanu vrednost u rbx
        ret                   # povratak iz main, kraj programa

        .section      .rodata
.format:
        .string       "%d na kvadrat = %d\n"
.end
```

Ovaj program semantički je ekvivalentan sledećem C programu:

```
#include <stdio.h>

int kvadrat(int n){
	return n*n;
}

main(int argc, char*argv[]){
	register int n;
	n = atoi(argv[1]);
	printf("%d na kvadrat = %d\n", n, kvadrat(n));
}
```

Program sa komandne linije čita ceo broj (u vidu stringa), konvertuje ga u ceo broj pozivom atoi bibliotečke C funkcije, izračuna njegov kvadrat pozivom istoimene funkcije, a zatim ispiše polazni broj i njegov kvadrat na standarnom izlazu koristeći bibliotečku funkciju printf. Program ne vrši proveru validnosti ulaznih podataka (za samostalnu vežbu dograditi da se ispita da li je zadat bar jedan parametar na komandnoj liniji i da li se radi o celom broju i ispis odgovarajućih poruka ako provere nisu uspešne).

Pozivanje asemblera

Gnu asembler se poziva za ulazni asemblerski fajl *program.s* na dva načina: preko gcc drajvera, kada se obavlja i proces asembliranja i proces linkovanja, na primer:

```
gcc -g -no-pie program.s -o program
```

Tada se odmah dobija izvršni fajl *program* koji se može izvršavati samostalno ili pod dibagerom. Opcija –g služi da se u izvršni fajl pored programskog koda ugrade informacije za rad dibagera (simboli, brojevi linija izvornog koda). Opcija –no-pie sprečava linker da generiše poziciono nezavisni izvršni progam (odskora je default opcija u gcc-u poz. nezavisni program, ali bi u našem primeru izazvao grešku, jer neke vrste adresiranja nisu kompatibilne sa pie). Poglavlja 1 i 2, kao i sekcija 9.16, dokumentacije GNU asemblera navode druge opcije komandne linije.

Alternativno, asembler se može pozvati eksplicitno kao:

```
as program.s –g –o program.o
```

Tada se dobija objektni fajl *program.o*

Potom iskoristiti gcc fasadu za pozivanje linkera i dobijanje izvršnog fajla.

```
gcc program.o –g –no-pie –o program
```

Alternativno, ako se poziva linker direktno, komandom ld, a program koristi c biblioteku, onda se na komandnoj liniji moraju navesti svi crt fajlovi i biblioteke, kako je opisano u prezentaciji “programski alati”.

Kucanje ove komande ponovo i ponovo postaje prilično dosadno. Alat *GNU make* može da automatizuje posao oko prevođenja programa.

Sintaksa asemblerskog jezika

GNU asembler podržava veliki broj različitih procesorskih arhitektura, ne samo x86. Iz tog razloga, njegova sintaksa je malo drugačija od drugih x86 asemblera. GNU asembler koristi istu sintaksu za sve CPU arhitekture koje podržava.

Asemblerski izvorni fajlovi se sastoje od niza iskaza, jedan po liniji. Svaki iskaz ima sledeći format, svaki deo je opcioni:

oznaka: instrukcija # komentar

Oznaka (labela) vam omogućava da identifikujete lokaciju programskog brojača (tj. njegovu adresu) u određenoj tački programa. Ova oznaka se zatim može koristiti, na primer, kao cilj instrukcije skoka ili za load/store instrukcije. Oznaka može biti bilo koji validan simbol, iza koga sledi dvotačka ":". Validan simbol je onaj koji koristi samo slova alfabeta A do Z i a do z (razlikuju se velika i mala slova), cifre od 0 do 9, kao i znakove "**_**" i "**.**" "**$**". Imajte na umu, međutim, da ne možete da počnete simbol

cifrom. (Za više informacija pogledajte odeljke 3.4 i 5.3 dokumentacije GNU asemblera*).*

Komentar je nešto što sledi znak "#". Sve posle #, osim kada se # pojavi u stringu, se ignoriše do kraja tekuće linije. Komentari u C stilu (pomoću "**/ ***" i "*** /**"**)** su takođe dozvoljeni.

Polje *instrukcije* je prava srž vašeg programa: to je bilo koja važeća instrukcija x86 asemblerskog jezika koju želite da koristite. To takođe uključuje i tzv pseudo-operacije ili *asemblerske direktive:* "instrukcije" koje ne generišu direktno mašinski kod već nalažu asembleru da uradi nešto. Ove direktive biće opisane u više detalja u nastavku.

GNU asemblerske direktive

Sve Asembler direktive imaju imena koja počinju tačkom "**.**". Od svih direktiva opisanih u Poglavlju 7 dokumentacije GNU asemblera, spisak direktiva predstavljen ovde (po abecednom redosledu) su one koje ćete najviše koristiti u svojim programima.

.align *expr*

Umetanje od nula do *expr* bajtova vrednosti 0 tako da će sledeća lokacija biti na granici *expr*-bajtne reči, pri čemu vrednost *expr* mora biti stepen dvojke. Podsetimo se da uravnavanje zavisno od tipa podatka rezultuje u efikasnijem pristupu. Kao primer, sledeće tri linije će ubaciti osam bajtova u izlaz objektne datoteke, pod pretpostavkom da je prva linija već na granici reči:

```
.byte 0x55 # inserts the byte 0x55
.align 4 # inserts three alignment bytes: 0x00 0x00 0x00
.long 0xAA55EE11 # inserts the bytes 0x11 0xEE 0x55 0xAA (LSB order)
```

0x prefiks označava broj u heksadekadnom sistemu. Pogledajte odeljak o izrazima, kasnije u ovom dokumentu, za više detalja.

Direktiva .align ima opcione argumente koje nisu ovde opisani, ako morate da ih koristite, savetujemo vam da umesto toga koristite .balign direktivu.. Videti sekciju 7.3 dokumentacije GNU asemblera za više informacija.

.ascii *"niska"*...

Ubacuje string konstantu u objektni fajl na mestu navođenja, bez pratećeg NUL karaktera. Više od jednog stringa može se navesti ako su razdvojeni zarezima. Kao primer, sledeća linija ubacuje tri bajta u izlazni objektni fajl:

```
.ascii "JNZ"  # ubacuje bajtove 0x4A 0x4E 0x5A
```

.asciz *"niska"* ...

Ubacuje string konstantu u objektni fajl na mestu navođenja, praćen NUL znakom (0x00 bajt). Kao i kod .ascii, više od jednog stringa može se navesti razdvojeni zarezima. Na primer, sledeća linija ubacuje četiri bajta (ne tri) u izlazni objektni fajl:

```
.asciz "JNZ"          # Ubacuje bajtove 0x4A 0x4E 0x5A 0x00
```

.byte *izraz* ...

Ubacuje bajt vrednosti izraza u objektni fajl. Više od jednog izraza se može pojaviti, ako su razdvojeni zarezima. Kao primer, sledeći redovi ubacuju 5 bajtova u izlazni objektni fajl:

```
.byte 64, 'A'         # inserts the bytes 0x40 0x41
.byte 0x42  # inserts the byte 0x42
.byte 0b1000011, 0104 # inserts the bytes 0x43 0x44
```

Imajte na umu da su brojevi koji počinju sa **0x** ili **0X** u heksadekadnom, brojevi koji počinju sa **0b** ili **0B** u binarnom, a brojevi koji počinju sa **0** (nula) u oktalnom sistemu. Videti odeljak izraza, kasnije u ovom dokumentu, za više informacija.

Pogledajte takođe .word, .long i .quad direktive.

.data

Prebacivanje odredišta sledećih iskaza u sekciju podataka izvršnog programa. Svi izvršni programi imaju najmanje jednu sekciju zvanu .text. Podaci se mogu smestiti u .data sekciju. Istoimene direktive omogućavaju vam da naizmenično prebacujete generisanje izlaza između te dve sekcije.

U .text sekciji trebalo bi da se pojavi samo izvršni kod a podaci (za čitanje/upis) trebalo bi da se pojavljuju u .data sekciji. Konstante (samo za čitanje) pojavljuju se u .rodata sekciji. Neinicijalizovani statički podaci pojavljuju se u .bss sekciji (Block Started By Symbol, ime vuče korene od nekog starog IBM mainframe računara). Sve ovo se, međutim, ne forsira od strane GNU asemblera. Poglavlje 4 dokumentacije GNU asemblera detaljnije se bavi ovom temom.

.end

Označava kraj datoteke izvornog koda; *sve* nakon ove direktive se ignoriše od strane asemblera. Nije obavezna, ali se preporučuje.

.equ *simbol*, *izraz*

Dodeluje vrednost izraza simbolu. Ova direktiva takođe se može napisati kao .set ili kao **"="**. Naredne tri linije su potpuno identične i dodeljuju vrednost 42 simbolu var:

```
.equ var, (5 * 8) + 2
.set var, 0x2A
var = 0b00101010
```

.extern *simbol*

Specifikuje da je *simbol* definisan u nekom drugom fajlu izvornog koda (tj. modulu). Ova direktiva je opciona jer asembler tretira sve simbole koji nisu definisani kao eksterne. Upotreba se, ipak, preporučuje kao deo dokumentovanja izvornog koda.

.global *simbol*

Specifikuje da *simbol* treba da bude globalno vidljiv svim ostalim modulima (objektnim fajlovima) koji su deo izvršnog fajla, i vidljiv za GNU linker. Simbol **_start,** koji je neophodan GNU Linkeru da odredi prvu instrukciju za izvršavanje u programu, mora uvek da bude globalni (i definisan samo u jednom modulu).

.word *izraz* ...

.2 byte *izraz* ...

Ubacuje (16-bitnu) vrednost izraza (polurеč) u objektni fajl. Više od jednog izraza se može pojaviti, ako su razdvojeni zarezima. Direktiva .2byte može se koristiti kao sinonim. Kao primer, sledeći redovi ubacuju 8 bajtova u izlazni objektni fajl:

```
.word 0xAA55, 12345 # inserts the bytes 0x55 0xAA 0x39 0x30
.2byte 0x55AA, -1       # inserts the bytes 0xAA 0x55 0xFF 0xFF
                    # Little Endian ordering assumed
```

.include *"fajl"*

Ubacuje sadržaj *fajla* na mestu ove direktive u izvornoj datoteci, kao da taj fajl bio unet u tekućoj datoteci direktno. Ponaša se isto kao Covski #include, a koristi se iz istog razloga: omogućava vam da uključite datoteke zaglavlja pune raznih definicija. Uzgred, čuvajte se korišćenja direktive .end u uključenom fajlu: GNU asembler će prestati da čita sve posle te direktive!

.set *simbol*, *izraz*

Ovo je sinonim za .equ direktivu.

.skip *izraz*

Preskoči onoliko bajtova u izlaznom objektnom kodu koliko iznosi vrednost *izraza*. Tako preskočene bajtove treba

tretirati kao nepredvidive vrednosti, iako se često postavljaju na nulu. Ova direktiva se koristi za deklarisanje neinicijalizovane promenljive određene veličine. Na primer, sledeće tri linije deklarišu tri promenljive, dve koje su inicijalizovane i jednu (Bafer) koja nije:

```
head_ptr:    .word 0       # Head pointer to within buffer (initially zero)
```

```
tail_ptr:    .word 0      # Tail pointer to within buffer (initially zero)
buffer:      .skip 512    # Buffer of 512 bytes, uninitialised
```

Pogledati takođe .ascii, .asciz, .byte, .hword i .word direktive za inicijalizovanu dodelu prostora.

.text

Usmerava odredište sledećih iskaza u sekciju teksta u izlaznom objektnom fajlu. To je sekcija u kojoj se smešta mašinski kod i read-only podaci. Pogledati i .data direktivu.

.long *izraz* ...

.4byte *izraz* ...

Ubacuje (32-bitnu) vrednost *izraza* u objektni kod. Više od jednog izraza se može pojaviti, ako su razdvojeni zarezima. Direktiva .4byte može da se koristi kao sinonim. Na primer, sledeći redovi ubacuju 8 bajtova u izlaznu objektnu datoteku:

```
.long 0xDEADBEEF      # inserts the bytes 0xEF 0xBE 0xAD 0xDE
.4byte -42            # inserts the bytes 0xD6 0xFF 0xFF 0xFF
                      # Least Significant Byte (LSB) ordering assumed
```

Podatak tipa .long bi zbog efikasnosti pristupa trebalo smestiti na adresu deljivu sa 4 (najniža dva bita te adrese da budu nule). Pogledajte .align direktivu za rešenje.

Pogledajte takođe .byte, .word i .quad direktive.

.quad *izraz* ...

.8byte *izraz* ...

Ubacuje (64-bitnu) vrednost *izraza* u objektni kod. Više od jednog izraza se može pojaviti, ako su razdvojeni zarezima. Direktiva .8byte može da se koristi kao sinonim za ,quad.

Izrazi

Mnogi asemblerski iskazi, kao i direktive, zahtevaju nekakav numerički operand. Na primer, "mov r0, #1" zahteva

broj "1". Izraz se može pojaviti na bilo kojem mestu gse se očekuje broj.

Na najosnovnijem nivou, izraz može da bude jednostavan ceo broj. Ovaj broj može se izraziti u dekadnom (sa uobičajenom notacijom), u heksadekadnom (sa **0x** ili **0X** prefiksom), u oktalnom (sa nulom) ili u binarnom (sa **0b** ili **0B** prefiksom) sistemu. Takođe se izražava kao znakovna konstanta, okružen jednostrukim navodnicima ili sa jednim navpdnikom ispred znaka.

Stoga, sledećih šest linija sve pune registar rax istom vrednošću, 74:

```
mov rax,74          # decimal number 74
mov rax,0x4A # hexadecimal number 0x4A (0X4A and 0x4a are also
OK)
mov rax,0112 # octal number 0112
```

```
        mov rax,0b1001010 # binary number 0b1001010 (0B1001010 is also
OK)
        mov rax,'J'   # character constant "J" (preferred syntax)
        mov rax,'J    # character constant "J" (alternative syntax)
```

Naravno, dobar programer će definisati simbol sa odgovarajućom vrednošću i koristiti ga umesto konstante:

```
        .set letter_J, 'J'
        mov rax, letter_J
```

Znakovne konstante mogu da sadrže određene escape sekvence koje se pišu sa \, slično kao u Cu. Videti odeljak 3.6.1 dokumentacije GNU asemblera za detalje.

Pored jednostavnih celim brojevima, izrazi mogu sadržati standardne matematičke i logičke operaratore definisane u jeziku C. Oni su detaljno opisani u Poglavlju 6 dokumentacije GNU asemblera. Neki primeri su:

```
        .set    ROM_size, 128 * 1024    # 131072 bytes (128KB)
        .set    start_ROM, 0xE0000000
        .set    end_ROM, start_ROM + ROMsize # 0xE0020000
        .set    bm1, 0b11001101             # Binary bit-mask (hex 0xCD)
        .set    val1, -2 * 4 + (45 / (5 << 2))  # -6 (in two's complement)
        .set    val2, ROM_size >> 10        # 128 (131072 shifted right by 10)
        .set    val3, bm1 | 0b11110000      # bm1 OR 0b11110000 =
0b11111101
```

Izrazi mogu da sadrže prethodno definisane labele, kao što je prikazano iznad. Takvi izrazi mogu dati *apsolutne* vrednosti ili *relativne* vrednosti. *Apsolutna vrednost* ne zavisi od svog položaja u završnom izvršnom kodu (tj. ona je poziciono nezavisna); Primer aps. vrednosti je jednostavna numerička konstanta. *Relativna vrednost*, sa druge strane, izražava se u odnosu na neku adresu, kao što je početak sekcije podataka ili programa. Ove vrednosti se fiksiraju (tj. dodeljuje im se konačna vrednost) od strane linkera kada se pravi finalni izvršni program, a ne od strane asemblera.

Relativne vrednosti se uglavnom koriste za sračunavanja pomeraja, i mogu samo da koriste "+" i "-" operatore. Na primer:

```
        .text
code_start:
        . . . .         # Many assembly language instructions...
code_end:
        .set instr_size, 4
        .set code_length, code_end - code_start
        .set first_instr, code_start
        .set instr_11th, code_start + 10 * instr_size
        .set instr_10th, instr_11th - instr_size
```

Simbol code_length je postavljen na apsolutni broj bajtova (razlika između code_end i code_start).

Ostali simboli, first_instr, instr_11th i instr_10th, svi su u odnosu na početak .text sekcije. Njihova

tačna vrednost podestavlja tek se formira konačni izvršni program.

Generalno, sledeća pravila se primenjuju na relativne izraze:

relative **+** *absolute* → *relative*
absolute **+** *relative* → *relative*
relative **–** *absolute* → *relative*
relative **–** *relative* → *absolute*

Svi ostali izrazi koji uključuju relativne vrednosti nisu dozvoljeni (osim, naravno, za jednostavan slučaj jedinstvenog simbola). Relativni simboli u izrazu moraju biti u istoj sekciji programa: ne može se naći razlika između vrednosti labele u .text sekciji i neke druge labele u .data sekciji, na primer.

Više informacija o relativnim izrazima može se naći u odeljku 4.1 dokumentacije GNU asemblera.

Drugi primer asemblerskih programa

Finalna varijanta primera sastoji se iz dva fajla pr2main.s i pr2kvadrat.s i ilustruje odvojeno prevođenje modula:

```
as –g pr2main.s –o pr2main.o
as –g pr2kvadrat.s –o pr2kvadrat.o
```

Potom linker povezuje međusobno ove module u program i povezuje sve sa C bibliotekom:

```
gcc –g –no-pie pr2main.o pr2kvadrat.o –o pr2
```

Primetiti da je u ovoj varijanti simbol kvadrat proglašen za globalni, a u pr2main naznačeno da se on uvozi (neobaveznom) deklaracijom .extern kvadrat. Globalni simboli su oni koji su definisani u jednom modulu, a koriste se u drugim modulima. Takve simbole razrešava linker (ugrađuje njihove vrednosti na mestu korišćenja). Direktiva .end koja označava kraj asemblerskog izvornog koda u datom fajlu je neobavezna, ali je dobra praksa navoditi je.

```
#### pr2main.s ####
        .intel_syntax noprefix
        .text
           .extern kvadrat
        .global       main
main:  # argc u edi, argv[] u esi
        push  rbx             # sacuvaj rbx (calee saved reg.)
        mov   rdi, QWORD PTR 8[rsi] # rdi = &argv[1]
        call    atoi          # eax = atoi(rdi)
        mov   ebx, eax            # sacuvaj eax za kasnije
        mov   edi, eax           # edi = eax
        call    kvadrat         # eax = kvadrat(edi)
        mov   edx, eax            # edx = eax treci parametar za printf
        mov   esi, ebx           # esi = ebx drugi parametar za printf
```

```
        lea     rdi, .format          # rdi = adresa .format, prvi param. za prinf
        mov     eax, 0               # oznacava da nema float parametara za prinf
        call    printf
        mov     eax, 0               # povratni kod iz main
        pop     rbx                 # vrati sacuvanu vrednost u rbx
        ret                     # povratak iz main, kraj programa

        .section      .rodata
.format:
        .string       "%d na kvadrat = %d\n"
.end
```

```
#### pr2kvadrat.s #####
        .intel_syntax noprefix
        .text
   .global kvadrat
kvadrat:
   # ulazni parametar u edi
   # rezultat u eax
        imul    edi, edi          # edi = edi * edi (signed mult.)
        mov     eax, edi              # eax = edi
        ret                     # povratak u program
  .end
```