Uvod u organizaciju i arhitekturu računara 2

1 Asemblersko programiranje u Intel 64 arhitekturi

1.1 Sintaksa

Opšta sintaksa asemblera je takva da se čita linija po linija. Linije mogu biti prazne u kom slučaju se ignorišu. Linija takođe može biti direktiva (počinje simbolom .) ili instrukcija. Komentarima se smatra sav sadrzaj do kraja linije nakon simbola `#`. Linija takođe može sadržati samo komentar.

Definicija labele se sastoji od identifikatora iza koga stoji simbol :. Identifikator mora početi slovom ili `_` a može sadržati slova, brojeve i `_`. Labele zamenjuju adrese podataka i instrukcija. One se prilikom prevođenja programa prevode u memorijske adrese.

Direktive imaju posebno značenje:

- `.intel_syntax noprefix` - označava da se koristi Intel-ova sintaksa, dok se imena registara koriste bez prefiksa `%`
- `.data` - započinje sekciju inicijalizovanih podataka
- `.bss` - započinje sekciju neinicijalizovanih podataka
- `.text` - započinje sekciju koda
- `.asciz` - kreira se ASCII niska na čijem se kraju automatski navodi terminirajuća nula
- `.int` - kreira se jedan ili niz celih brojeva, članovi su razvojeni zapetom
- `.byte` - kreira se jedan ili niz bajtova
- `.word` - kreira se jedan ili niz slogova od dva bajta
- `.long` - kreira se jedan ili niz slogova dužine četiri bajta
- `.quad` - kreira se jedan ili niz slogova dužine osam bajtova
- `.global ime` - označava se labela ime kao globalna, čime se omogućava linkeru da poveže definisane simbole.

Jedna instrukcija se sastoji od koda instrukcije i operanda (jednog ili više) ukoliko instrukcija zahteva operande. Svaka instrukcija ima svoju simboličku oznaku. Opšti oblik instrukcije sa dva argumenta je

```
instr dest, src
```

Operandi mogu biti

- Registarski - navodi se registar
- Memorijski - navodi se adresa na kojoj se nalazi vrednost sa kojom radimo
- Neposredni - navodi se sama vrednost sa kojom radimo

Memorijsko adresiranje ima oblik $[B + S * I + D]$, gde je B bazna adresa, S je faktor, I je indeks, dok je D pomeraj. Izostavljanjem nekog od ovih elemenata dobijaju se različite kombinacije izračunavanja adrese operanda. Na primer, moguće je navesti samo baznu adresu $[B]$, ili baznu adresu i pomeraj $[B + D]$. Oblik $[B + S * I]$ je zgodan prilikom rada sa nizovima: u bazni registar se smesti adresa početka niza, a u indeksni registar se smesti tekući indeks (koji se ažurira u svakoj iteraciji). Za faktor se uzima veličina elementa niza.

U slučaju da instrukcija koja sadrži memorijski operand nema ni jedan registarski operand, tada se mora eksplicitno specificirati širina memorijskog operanda. Ovo se radi navođenjem jednog od prefiksa:

- byte ptr - za jednobajtni podatak
- word ptr - za dvobajtni podatak
- dword ptr - za četvorobajtni podatak
- qword ptr - za osmobajtni podatak

Ovo nije neophodno ako registarski operand postoji u instrukciji, zato što onda njegova širina implicitno određuje širinu operanada koji se koriste.

1.2 Registri opšte namene

Intel 64 arhitektura ima 16 64-bitnih registara opšte namene: RAX, RBX, RCX, RDX, RSI, RDI, RSP, RBP, kao i registre R8-R15. Nizim delovima registara se može pristupiti preko oznaka EAX, EBX, ECX, EDX, ESI, EDI, ESP, EBP, uz dodatak registara R8D-R15D.

Ovi registri su generalno opšte namene, mada u praksi neki imaju specijalne uloge. Tako, na primer, RSP se uvek koristi kao pokazivač vrha steka, RBP je pokazivač tekućeg okvira steka. Ostali registri u nekim situacijama mogu imati specijalne uloge (npr. pri množenju i deljenju se uvek implicitno koriste registri RAX i RDX, dok se pri radu sa stringovima za pokazivače uvek koriste registri RSI i RDI). Za brojačke petlje se uglavnom koristi registar RCX, pošto određene instrukcije kontrole toka implicitno koriste i menjaju ovaj registar.

1.3 Instrukcije

1.3.1 Instrukcije transfera

- MOV op1, op2 - premeštanje (op1 = op2)
- MOVZX op1, op2 - u prvi operand se smešta vrednost drugog operanda proširena nulama
- MOVSX op1, op2 - u prvi operand se smešta vrednost drugog operanda proširena u skladu sa znakom
- LEA op1, op2 - učitavanje adrese drugog operanda u prvi operand, pri čemu je drugi operand memorijski operand.

Prilikom premeštanja vrednosti, ukoliko nije navedeno drugačije, premešta se vrednost koja je veličine registra koji učestvuje u operaciji. Što znači da ako se kao operand navede memorijska lokacija, onda se radi sa veličinom koju određuje registar.

1.3.2 Aritmetičke instrukcije

- ADD op1, op2 - sabiranje (op1 = op1 + op2)
- SUB op1, op2 - oduzimanje (op1 = op1 - op2)
- CDQE - označeno proširivanje eax na rax
- CDQ - označeno proširivanje eax na edx:eax
- CQO - označeno proširivanje rax na rdx:rax
- MUL op - množenje neoznačenih celih brojeva (rdx:rax = rax * op)
- IMUL op - množenje označenih celih brojeva (rdx:rax = rax * op)
- DIV op - deljenje neoznačenih celih brojeva (rax = rdx:rax / op, rdx = rdx:rax % op)
- IDIV op - deljenje označenih celih brojeva (rax = rdx:rax / op, rdx = rdx:rax % op)
- NEG op - promena znaka (op = -op)
- INC op - uvećanje (op = op + 1)
- DEC op - umanjenje (op = op - 1)

Oznaka rdx:rax znači 128-bitni ceo broj čiji su viši bitovi u rdx a niži u rax.

1.3.3 Logičke instrukcije

- AND op1, op2 - bitovska logička konjukcija (op1 = op1 `&` op2)
- OR op1, op2 - bitovska logička disjunkcija (op1 = op1 `|` op2)
- XOR op1, op2 - bitovska logička ekskluzivna disjunkcija (op1 = op1 `^` op2)
- NOT op - bitovska negacija (op = `~`op)
- SHL op1, op2 - shift-ovanje ulevo (op1 = op1 `<<` op2). op2 je konstanta.
- SHR op1, op2 - shift-ovanje udesno (logičko) (op1 = op1 `>>` op2). op2 je konstanta.
- SAR op1, op2 - shift-ovanje udesno (aritmetičko) (op1 = op1 `>>` op2). op2 je konstanta.

1.3.4 Instrukcije poređenja

- CMP op1, op2 - upoređivanje (oduzimanje bez upisivanja rezultata)
- TEST op1, op2 - testiranje bitova (bitovska konjukcija bez upisivanja rezultata)

1.3.5 Instrukcije za rad sa stekom

- PUSH op - postavljanje na stek
- POP op - skidanje sa steka

1.3.6 Instrukcije kontrole toka

- JMP op - bezuslovni skok na adresu op (memorijski operand)
- CALL op - bezuslovni skok uz pamćenje povratne adrese na steku.
- RET - skida sa steka adresu i skače na tu adresu.
- JZ op - skače ako je rezultat prethodne instrukcije nula.
- JE op - skače ako je rezultat prethodnog poređenja jednakost (ekvivalentno sa JZ)
- JNZ op - skače ako je rezultat prethodne operacije različit od nule
- JNE op - skače ako je rezultat prethodnog poređenja različitost (ekvivalentno sa JNZ)
- JA op - skače ako je rezultat prethodnog poređenja veće (neoznačeni brojevi)
- JB op - skače ako je rezultat prethodnog poređenja manje (neoznačeni brojevi)
- JAE op - skače ako je rezultat prethodnog poređenja vece ili jednako (neoznačeni brojevi)
- JBE op - skače ako je rezultat prethodnog poređenja manje ili jednako (neoznačeni brojevi)
- JG op - skače ako je rezultat prethodnog poređenja veće (označeni brojevi)
- JL op - skače ako je rezultat prethodnog poređenja manje (označeni brojevi)
- JGE op - skače ako je rezultat prethodnog poređenja veće ili jednako (označeni brojevi)
- JLE op - skače ako je rezultat prethodnog poređenja manje ili jednako (označeni brojevi)

Slično, postoje i negacije gornjih instrukcija uslovnog skoka: JNA, JNB, JNAE, JNBE, JNG, JNL, JNGE, JNLE.

1.4 Konvencije za pozivanje funkcija

Instrukcija kojom se poziva funkcija je

```
call naziv
```

Celobrojni parametri funkcija, uključujući i adrese, prenose se, redom, preko registara rdi, rsi, rdx, rcx, r8, r9. Ukoliko ima više od šest parametara, ostali se smeštaju na stek u obrnutom redosledu - s desna na levo. Povratna vrednost funkcije se nalazi u rax registru.

Prilikom poziva funkcije, moguće je da će ona izmeniti neke registre. Registri koji pripadaju pozivajućoj funkciji su rbx, rbp, r12-15. Sadržaj ovih registara se mora sačuvati ukoliko se menja u funkciji. Ostali registri se mogu menjati bez čuvanja. Registri koji pripadaju pozvanoj funkciji su rax, rdi, rsi, rdx, rcx, r8-r11. Na sadržaje ovih registara ne može računati pozivajuća funkcija. Ukoliko je korišćen stek prilikom poziva funkcije, neophodno je da vrh steka bude poravnat sa adresom deljivom sa 16.

Na početku svake funkcije se nalazi prolog u kome je potrebno izvršiti sledeće instrukcije

```
push rbp
mov rbp, rsp
sub rsp, N
```

Alternativa je

```
ENTER N, 0
```

gde N označava broj bajtova koji odvajamo za lokalne promenljive.

Na kraju svake funkcije se nalazi epilog u kome se izvršavaju instrukcije

```
mov rsp, rbp
pop rbp
```

Alternativa je

```
LEAVE
```

Povratak iz funkcije se izvršava instrukcijom

```
RET
```

Ova instrukcija skida povratnu adresu sa steka i prelazi na izvršavanje instrukcije sa te adrese.

1.5 Prevođenje

Izvorni kod sa asemblerskim funkcijama 1.s se prevodi na sledeći način:

- gcc 1.s

Moguće je prevoditi kod iz više izvornih datoteka navodeći ih u jednoj komandi:

- gcc 1.c 1.s