

Arhitektura računara 1

skripta za vežbe sa zadacima

Oktober 2023

1 Uvod

U ovoj skripti akcenat će biti na 64-bitnoj varijanti x86-arhitekture, x64 i operativnom sistemu Linux.

x64 je generičko ime za 64-bitnu ekstenziju Intelovih i AMD-ovih skupova instrukcija 32-bitne x86-arhitekture. AMD je predstavio prvu verziju x64, inicijalno nazvanu x86-64, a kasnije su je preimenovali u AMD64. Intel je svoju implementaciju nazvao IA-32e, a potom EMT64. Postoje neznatne razlike između ove dve verzije, ali većina koda radi dobro u obe.

Asembler je bilo koji programski jezik niskog nivoa sa jakim vezom između instrukcija u samom jeziku i instrukcija mašinskog jezika koje odgovaraju samoj arhitekturi. Svaki assembler ima različitu sintaksu. Čak i između assemblera koji imaju "Intelovu sintaksu" odnosno koji su kompatibilni sa Intelovim procesorima, postoji razlika. Na primer, za rad sa x86 procesorima možemo da koristimo sledeće asemblere: MASM, NASM, YASM, itd.

GNU assembler, poznatiji kao gas ili as je assembler razvijan od strane GNU projekta. On je podrazumevani "deo u pozadini" GCC kompajlera. Takođe je i cross-platform, što znači da može da se izvršava na velikom broju različitih arhitektura. Na početku je podržavao samo AT&T sintaksu, a od verzije 2.10 podržava i Intelovu sintaksu. Na ovom kursu radićemo GNU assembler i Intelovu sintaksu.

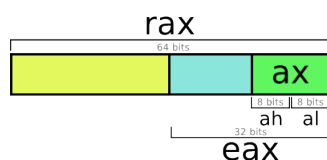
1.1 Registri

x86 ima 8 registara opšte namene (eax, ebx, ecx, edx, ebp, esp, esi, edi). Pošto je x86 32-bitna arhitektura, to znači da adrese u memoriji možemo da zapišemo u 32 bita, odnosno da ukupno možemo da zapišemo 4GB RAM-a. Takođe, veličina registara je 32 bita. U x64 ovi registri su prošireni do 64 bita. Novodobijeni registri imaju isti naziv osim što je umesto slova 'e' prvo slovo 'r' i njihova donja polovina (niži bitovi) su odgovarajući x86 registri. Npr. prošireni registar rax u x64 ima kao svoju donju polovinu eax, dok prošireni registar rbx ima kao svoju donju polovinu rbx.

U x86 nisu svi registri zaista bili registri opšte namene, jer su se neki implicitno koristili u specifične svrhe. Npr. registri ebp i esp za rad sa stekom. U x64, njima odgovarajući registri rsp i rbp zadržavaju tu namenu.

Detalnije o registrima:

- **rax** - Povratne vrednosti funkcije se skladište u njemu, niža polovina mu je eax, dok je niža polovina eax-a ax. Registar od samo 16 bitova ax, dovoljan je da se u njemu smesti npr. podatak tipa short int, i ima dve svoje polovine - višu ah i nižu al, koje su po 8 bitova i u njima može da stane podatak tipa char.



- **rbx** - Registar koji mora da se sačuva pre upotrebe. Nakon što završimo rad sa njim, mora da ponovo ima vrednost koju je imao pre nego što smo ga koristili. Niža polovina mu je ebx, čija je niža polovina bx. Registar bx ima višu polovinu bh i nižu bl.
- **rcx** - Tipični registar opšte namene. Često se koristi kao brojač. Niža polovina mu je ecx, čija je niža polovina cx. Registar cx ima višu polovinu ch i nižu cl.
- **rdx** - Registar opšte namene. Niža polovina mu je edx, čija je niža polovina dx. Registar dx ima višu polovinu dh i nižu dl.
- **rsp** - Registar koji pokazuje na vrh steka (detalji u nastavku). Niža polovina mu je esp, čija je niža polovina sp. Niža polovina registra sp je spl. Kada koristimo ovaj registar moramo da budemo veoma pažljivi. Nije preporučljivo koristiti ga za druge namene, jer onda dolazi do grešaka u radu sa stekom.
- **rbp** - Ponekad se koristi da čuva staru vrednost steka ili "bazu". Preciznije, kada smo u pozivu funkcije (što podrazumeva da je na stek stavljen okvir funkcije i da je rsp promenjen), moramo da zapamtimo gde smo stali na steku kada se funkcija završi i ukloni sa steka. Za to nam služi rbp. Niža polovina mu je ebp, čija je niža polovina bp. Niža polovina registra bp je bpl. Nije preporučljivo koristiti ga za druge namene, jer onda dolazi do grešaka u radu sa stekom.
- **rsi** - Registar opšte namene. Koristi se za prenos drugog argumenta funkcije. Niža polovina mu je esi, čija je niža polovina si. Niža polovina registra esi je sil.

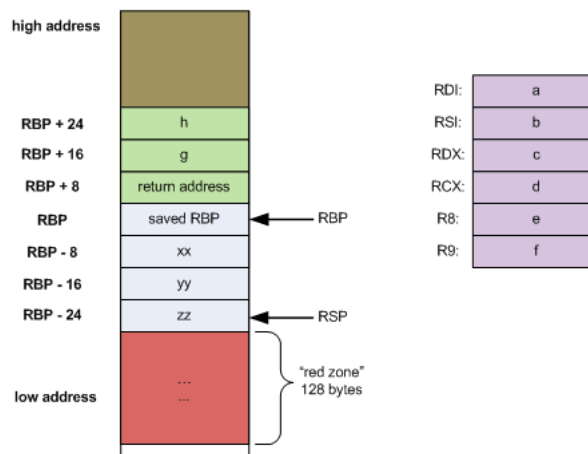
- **rdi** - Registar opšte namene. Koristi se za prenos prvog argumenta funkcije. Niža polovina mu je edi, čija je niža polovina di. Niža polovina registra edi je dil.
- **r8, r9, r10, r11** - Registri opšte namene. Niža polovina r8 je r8d. Niža polovina r8d je r8w. Niža polovina r8w je r8b. Sufiksi d,w,b su skraćenice od dword, word, byte i opisuju koliko memorije zauzimaju. Slično se određuju delovi registara r9, r10 i r11.
- **r12, r13, r14, r15** - Registri koje bi trebalo sačuvati, odnosno ako radimo sa njima, trebalo bi ih nakon što završimo vratiti na vrednosti koje su imali pre nego što smo počeli rad sa njima. Njihovi delovi se određuju slično kao za registar r8.

1.2 Prosledjivanje argumenata

Prvih šest celobrojnih argumenata funkcije (od kojih su neki potencijalno pokazivači, jer pokazivači čuvaju adrese (koje su celi brojevi) i zauzimaju isti prostor kao neki celobrojni podatak) prenose se redom u registrima rdi, rsi, rdx, rcx, r8 i r9. Ako imamo više od šest ovakvih argumenata onda se oni prenose preko steka. Najbolje je da pogledamo na primeru. Za ovu funkciju stek izgleda na sledeći

```
long myfunc(long a, long b, long c, long d,
            long e, long f, long g, long h)
{
    long xx = a * b * c * d * e * f * g * h;
    long yy = a + b + c + d + e + f + g + h;
    long zz = utilfunc(xx, yy, xx % yy);
    return zz + 20;
}
```

način: Ne smemo da pokvarimo rbp zato što nam rbp određuje poziciju na



steku iznad koje ne smemo da diramo podatke. Preciznije, mi znamo da se na

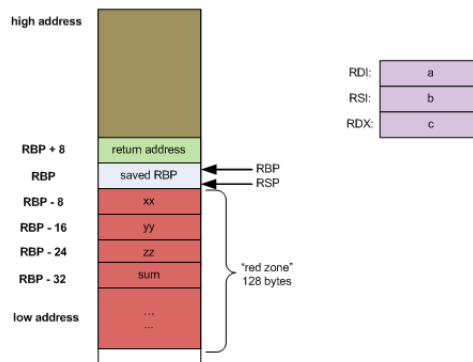
adresi `rbp+8` nalazi adresa povratka iz funkcije. Ako to "pokvarimo" nećemo znati gde treba da se vratimo iz funkcije. Takodje, pošto smo imali osam argumenata funkcije, a ne šest, dva su morala da budu preneti preko steka. Oni se redom nalaze iznad adrese povratka, na adresama `rbp+16`, `rbp+24`, itd. (**Ovde treba biti pažljiv. U pitanju su bili podaci tipa `long` i oni zauzimaju 8 bajtova, da je bio u pitanju podatak tipa `int` zauzeo bi samo 4 bajta, i ne bi išao na adresu `rbp+16`, nego na `rbp+12`, jer bi bio 4 bajta udaljen od adrese povratka koja je na `rbp+8`).**)

Ostaje da kažemo nešto i o crvenoj zoni koja je jedan vid optimizacije. Možemo da pretpostavimo da 128 bajtova ispod `rsp` neće biti narušene raznim signalima i prekidima, te možemo da koristimo taj prostor za međjuračunanja, bez da eksplicitno pomeramo vrh steka. Treba imati na umu da ova zona može da bude pogodjena pozivima drugih funkcija, te je najbolje da je koristimo u funkcijama koje ne sadrže pozive drugih funkcija u sebi. Pogledajmo kako "radi" crvena zona na primeru funkcije koja u sebi nema pozive drugih funkcija: Izgled

```
long utilfunc(long a, long b, long c)
{
    long xx = a + 2;
    long yy = b + 3;
    long zz = c + 4;
    long sum = xx + yy + zz;

    return xx * yy * zz + sum;
}
```

steka za ovu funkciju dat je na slici ispod. U funkciji `utilfunc`, nema prenosa



promenljivih po steku, jer argumenata funkcije ima manje od šest. Takodje, zbog nedostatka poziva ka drugim funkcijama, moguće je da se rezultati nekih izračunavanja (ovde sve promenljive za međjukorake zauzimaju manje jednako od 128 bajtova) sačuvaju u crvenoj zoni.

1.3 GDB debugger

Ako smo napisali naš program 1.s koristeći assembler Intel64 arhitekture (x64), taj program ćemo prevesti sledećim dvema naredbama:

```
as -gstabs -o 1.o 1.s
```

```
gcc -g -Wall -pedantic -no-pie -o 1 1.o
```

Opcije -g i -gstabs odnose se na formate za debugovanje. Opcija -g podrazumeva naivni format za debugovanje koji pruža operativni sistem i ovakav format za debugovanje podržan je od strane GDB debagera koji ćemo koristiti. Što se tiče -gstabs opcije, na nekim sistemima GNU assembleri zahtevaju ovu opciju. Opcija -pedantic uključuje dodatne ekstenzije i generiše detaljnija upozorenja. Opcija -no-pie omogućava da se ne generiše poziciono nezavisan izvršni fajl. Pie je bezbednosni preduslov da svaki put kernel učitava binarni fajl i zavisnosti u proizvoljnu adresu virtuelne memorije. Često zahteva dodatno programiranje.

GDB debugovanje pokrećemo u terminalu sa: `gdb ./1` gde je 1 generisani izvršni program. Ako želimo da pokrenemo program potrebno je da unesemo komandu `start`. Ako želimo da se pomerimo na sledeću mašinsku instrukciju koristimo komandu `stepi`. Ako želimo da udjemo u poziv funkcije i proverimo šta se u njoj dešava koristimo instrukciju `step`. Za štampanje vrednosti registara npr. registra `rax` kucamo `info registers $rax`.