Asemblersko programiranje
amd64 arhitektura

Osnovne karakteristike amd64 arhitekture

- Razni nazivi: x86-64 x64, x86_64, AMD64, Intel 64
- Nadogradnja na 32-bitnu x86 arhitekturu:
  - Povećan adresni prostor
  - Povećanje veličine registara opšte namene sa 32 bita na 64 bita
  - Povećanje broja registara opšte namene sa 8 na 16
  - ...

Sintaksa

- Koristićemo Intelovu neprefiksnu sintaksu
- Opšta sintaksa asemblera – čitamo liniju po liniju
  - Prazna linija nema značenje (ignoriše se)
  - Linija može biti direktiva (počinje simbolom .)
  - Linije koje nisu direktive ili prazne smatraju se instrukcijama
  - Svaka linija može početi labelom
  - Pri nailasku na simbol #, ostatak linije se ignoriše

Sintaksa – labele

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

Sintaksa – direktive

- Imaju posebno značenje

  .intel_syntax noprefix – označava se da se koristi Intelova neprefiksna sintaksa

  .data – počinje sekciju inicijalizovanih podataka

  .text – počinje sekciju koda

Sintaksa – direktive

.asciz – kreira se ASCI nisku na čijem se kraju automatski navodi terminirajuća nula

.byte – kreira se jedan ili niz bajtova

- Članovi niza razdvojeni su zapetom

.word – kreira se jedan ili niz slogova od 2 bajta (short ukoliko koristimo označene podatke)

.long – kreira se jedan ili niz slogova dužine 4 bajta

.quad – kreira se jedan ili niz slogova dužine 8 bajtova

.globl ime (ili .global ime) označava labelu kao globalnu, omogućujući linkeru da poveže definisane simbole

Sintaksa – komentari

- Komentari pocinju simbolom # i završavaju se na kraju reda

Sintaksa – instrukcije

- Jedna instrukcija sastoji se od koda instrukcije i operan(a)da
- Svaki kod instrukcije ima svoju simboličku oznaku
- Opšti oblik instrukcije sa dva argumenta:

  instr dest, src

Sintaksa – operandi

- Načini zadavanja operanada:
  - Registarski operandi: navodi se registar
  - Neposredni operandi: navodi se vrednost
  - Memorijski operandi: navodi se adresa na kojoj se nalazi vrednost s kojom radimo
-

Memorijsko adresiranje

- Opšti oblik:

  [B + S*I + D]

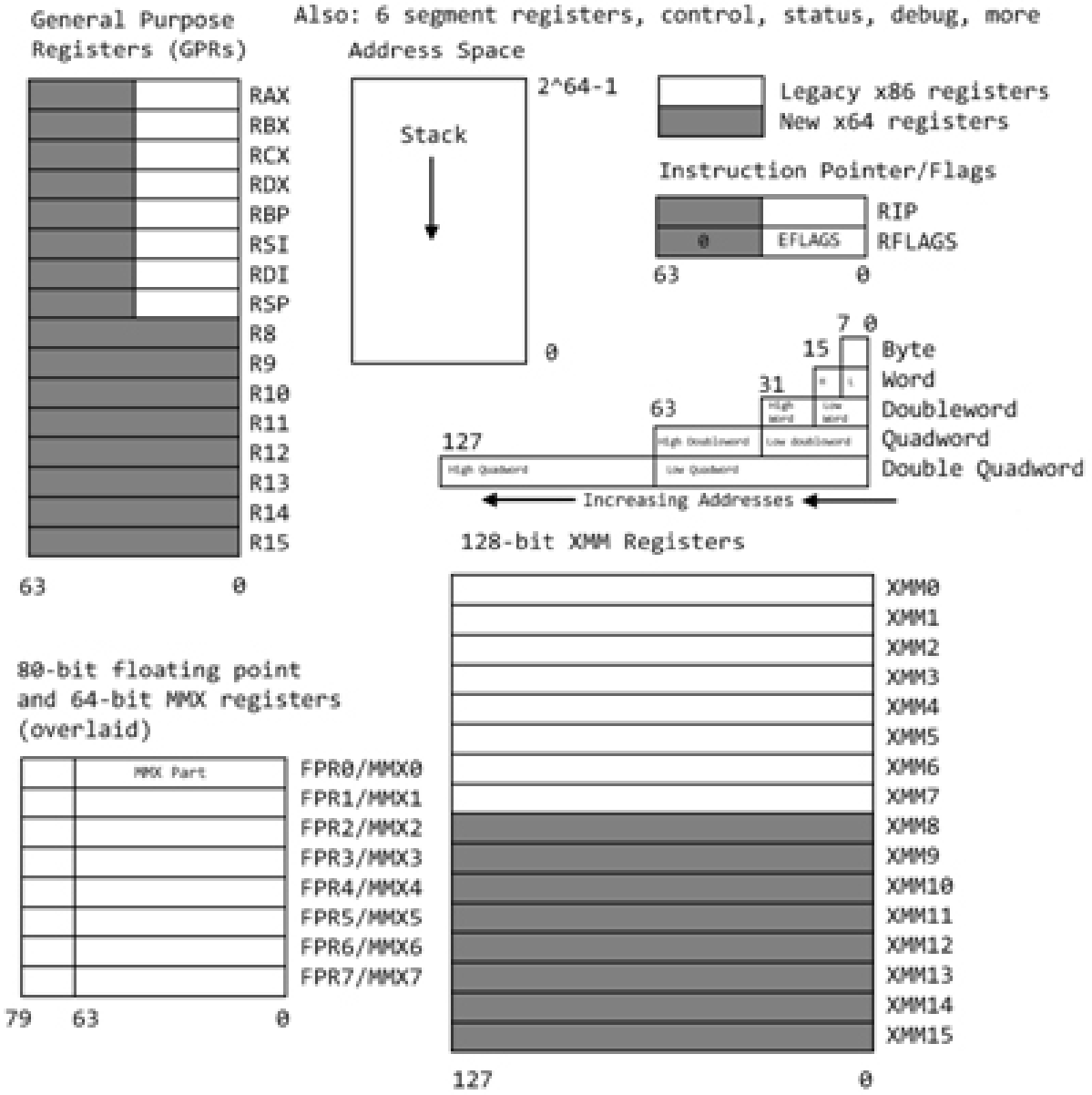
  - B je bazna adresa
  - D je pomeraj
  - I je indeks
  - S je velicina “elementa”
- Moguće je izostavljanje nekog od elemenata

  [B] – bazno (indirektno) adresiranje (nalik na dereferenciranje)

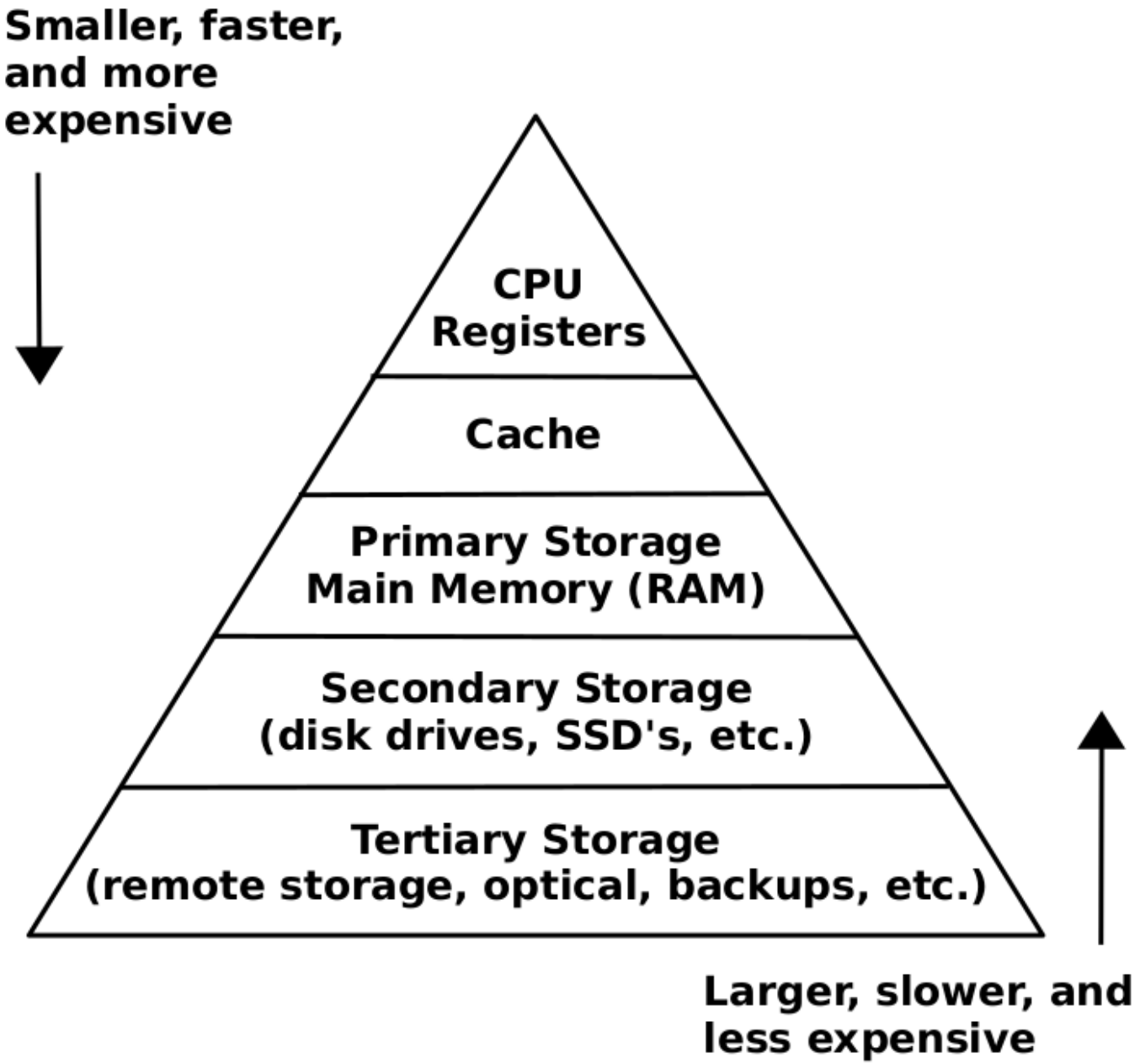
  [B + D] – bazno adresiranje sa pomerajem

  [B + S*I] – indeksno adresiranje

Osnovne karakteristike amd64 arhitekture



Memorija


Smaller, faster,
and more
expensive
CPU
Registers
Cache
Primary Storage
Main Memory (RAM)
Secondary Storage
(disk drives, SSD's, etc.)
Tertiary Storage
(remote storage, optical, backups, etc.)
Larger, slower, and
less expensive

Osnovne karakteristike amd64 arhitekture – registri

- **Registri opšte namene: rax, rbx, rcx, rdx, rsi, rdi, rsp, rbp**; veličina: 64 bita
  - Registri rsp i rbp koriste se za rad sa stekom
- Moguć je pristup nižih 32 bita: eax, ebx, ecx, edx, esi, edi, esp, ebp
- **Postoje i registri r8, r9, …, r15**
  - Pristup nižih 32 bita: r8d - r15d

Osnovne karakteristike amd64 arhitekture – registri sa posebnim značenjem

- rbp (frame/base pointer): sadrži pokazivač na okvir steka tekuće funkcije
- rsp (stack pointer): sadrži pokazivač na vrh steka u svakom trenutku
  - Alokacija memorije za lokalne promenljive vrši se umanjivanjem vrednosti rsp registra!
- rflags

Konvencije za pozive funkcija

- Instrukcija za poziv:
  - call ime
- Prenos parametara:
  - Celobrojni podaci (uključujući adrese) prenose se, redom, preko registara: rdi, rsi, rdx, rcx, r8, r9
  - Ukoliko ima više parametara nego što možemo da smestimo, smeštaju se na stek
    - Argumenti koji se prenose preko steka na stek se stavljaju u obrnutom poretku: s desna na levo
- Povratna vrednost nalazi se u rax registru

Konvencije za pozive funkcija

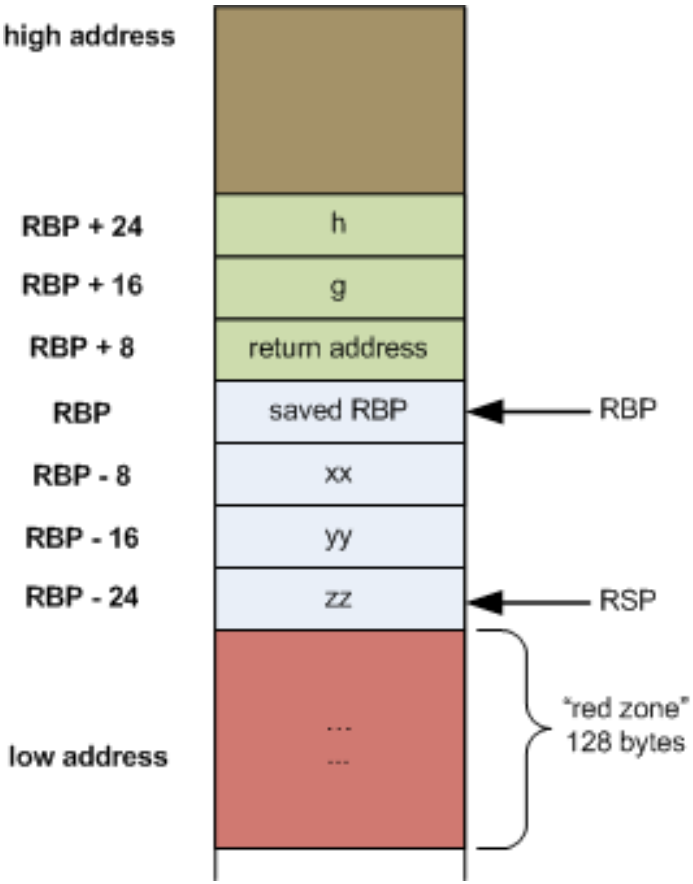
- Prilikom poziva funkcije, moguće je da će ona izmeniti neke registre
- Registri koji pripadaju pozivajućoj funkciji:
  - rbx, rbp, r12-r15
  - Date registre moramo sačuvati ukoliko ih menjamo u funkciji, ostale možemo slobodno koristiti
- Registri koji pripadaju pozvanoj funkciji:
  - rax, rdi, rsi, rdx, rcx, r8-r11
- Ukoliko je korišćen stek prilikom poziva funkcije, neophodno je da vrh steka bude poravnat sa adresom deljivom sa 16

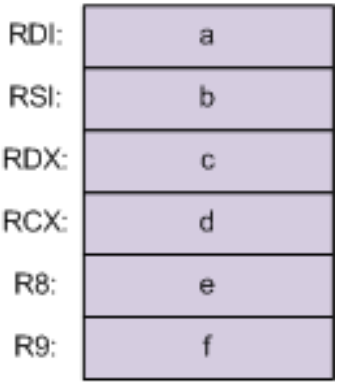
Konvencije za pisanje funkcija

- Na početku svake funkcije nalazi se prolog:
  - Čuvamo vrednosti rbp registra na steku i u rbp registar upisujemo pokazivač na trenutni vrh steka – rsp
  - Alternativa: enter n, 0
    - Argument n označava broj bajtova koji odvajamo za lokalne promenljive
- Na kraju svake funkcije nalazi se epilog:
  - Sklanjamo sa steka sve do početka funkcije i u vrednost rbp registra skidamo sa steka
  - Alternativa: leave
- Svaku funkciju završavamo instrukcijom ret

Primer izgleda steka

```
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;
}
```





Osnovne karakteristike amd64 arhitekture – rflags registar

- rflags: sadrži informacije o specijalnim dogadjajima koji su se desili nakon izvršene instrukcije u procesoru
- Nije mu moguće direktno pristupiti iz programa već se koriste razne instrukcije

Ime	**Oznaka**	**Bit**	**Značenje**
Carry	CF	0	Prenos
Zero	ZF	6	Rezltat operacije je nula
Sign	SF	7	1 na najvišem mestu u rezultatu
Overflow	OF	11	Prekoračenje

Instrukcije za rad sa stekom

- Stek raste ka nižim memorijskim adresama!
- push – operand se stavlja na vrh steka
- pop – u operand se upisuje vrednost sa vrha steka dok se stek smanjuje za veličinu operanda
  - Na stek se mogu staviti samo dvobajtne ili osmobajtne vrednosti!

Instrukcije transfera

- mov op1, op2 – premeštanje (nalik na op1=op2); očekuje operande iste sirine
- movzx op1, op2 – u prvi operand smešta se vrednost drugog operanda, proširena nulama
- movsx op1, op2 – u prvi operand smešta se vrednost drugog operanda, proširena u skladu sa znakom
- Kod instrukcija movzx i movsx, drugi operand ne može biti dat neposredno i očekivano je da je drugi operand manje širine od prvog.
- lea op1, op2 – učitavanje adrese; drugi argument je memorijski operand

Aritmetičke instrukcije

- Sabiranje:
  - add arg1, arg2 (arg1 = arg1 + arg2)
- Oduzimanje:
  - sub arg1, arg2 (arg1 = arg1 - arg2)
- Proširivanje znaka:
  - cdqe – označeno proširuje eax na rax (32 na 64 bita)
  - cdq – označeno proširuje eax na edx:eax (32 na 64 bita)
  - cqo – označeno proširuje rax na rdx:rax (64 na 128 bitova)

Aritmetičke instrukcije

- Množenje neoznačenih celih brojeva:
  - mul arg (rdx:rax = rax * arg)
- Množenje označenih celih brojeva:
  - imul arg (rdx:rax = rax * arg)

Aritmetičke instrukcije

- Deljenje neoznačenih celih brojeva:
  - div arg (rax = rdx:rax / arg, rdx = rdx:rax % arg)
- Deljenje označenih celih brojeva:
  - idiv arg (rax = rdx:rax / op, rdx = rdx:rax % op)
- Negiranje argumenta:
  - neg arg (arg = -arg)

Aritmetičke instrukcije

- Uvećanje argumenta za 1:
  - inc arg (arg = arg + 1)
- Umanjenje argumenta za 1:
  - dec arg (arg = arg - 1)
-

Logičke instrukcije

- Konjunkcija
  - and arg1, arg2 (arg1 = arg1 & arg2)
- Disjunkcija
  - or arg1, arg2 (arg1 = arg1 | arg2)
- Ekskluzivna disjunkcija:
  - xor arg1, arg2 (arg1 = arg1 ^ arg2)
- Negacija:
  - not arg (arg = ~arg)

Logičke instrukcije

- Šiftovanje (pomeranje) u levo:
  - shl arg1, arg2 (arg1 = arg1 << arg2); arg2 je konstanta
- Šiftovanje (pomeranje) u desno:
  - Logičko: shr arg1, arg2 (arg1 = arg1 >> arg2); arg2 je konstanta
  - Aritmetičko: sar arg1, arg2 (arg1 = arg1 >> arg2); arg2 je konstanta

Instrukcije poređenja

- cmp – upoređivanje (oduzimanje bez upisivanja rezultata)
- test – testiranje bitova (bitovska konjukcija bez upisivanja rezultata)

Instrukcije kontrole toka

- jmp arg – bezuslovni skok na adresu op (memorijski operand)
- call arg – bezuslovni skok uz pamćenje povratne adrese na steku.
- ret – skida sa steka adresu i skače na tu adresu.
- jz arg – skače ako je rezultat prethodne instrukcije nula.
- je arg – skače ako je rezultat prethodnog poređenja jednakost (ekvivalentno sa JZ)
- jnz arg – skače ako je rezultat prethodne operacije različit od nule
- jne arg – skače ako je rezultat prethodnog poređenja različitost (ekvivalentno sa JNZ)

Instrukcije kontrole toka

- ja arg – skače ako je rezultat prethodnog poređenja veće (neoznačeni brojevi)
- jb arg – skače ako je rezultat prethodnog poređenja manje (neoznačeni brojevi)
- jae arg – skače ako je rezultat prethodnog poređenja vece ili jednako (neoznačeni brojevi)
- jbe arg – skače ako je rezultat prethodnog poređenja manje ili jednako (neoznačeni brojevi)
- jg arg – skače ako je rezultat prethodnog poređenja veće (označeni brojevi)
- jl arg – skače ako je rezultat prethodnog poređenja manje (označeni brojevi)
- jge arg – skače ako je rezultat prethodnog poređenja veće ili jednako (označeni brojevi)
- jle arg – 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.

Širina operanada

- Ukoliko je jedan od operanada u instrukciji registar, onda je implicitno određena širina drugog operanda.
- Ukoliko nemamo nijedan registarski operand, neophodno je naglasiti širine memorijskih operanada:
  - byte ptr – za jednobajtni podatak
  - word ptr – za dvobajtni podatak
  - dword ptr – za četvorobajtni podatak
  - qword ptr – za osmobajtni podata

Povezivanje sa C kodom

- gcc kompajler dopušta prevođenje asemblerskog koda:
  - gcc 1.s
- Moguće je prevoditi kod iz više izvornih datoteka:
  - gcc 1.c 1.s

U nastavku

- Rad sa pokazivačima
- Rad sa nizovima
- Rekurzija