x86-64

- 64-bitna CISC arhitektura.
- 16 opštenamenskih registara (za svaki od njih postoji mogućnost pristupa nižm 32, 16 i 8 bita).
- `little-endian` (podaci širi od 1 bajt, kontinualno se zapisuju, tako da je na nižoj adresi zapisan bajt nižeg značaja).
- Dužina instrukcije je između 1 i 15 bajta.
- Stek raste ka nižim adresama.
- Registar `rsp` sadrži adresu poslednje zauzete lokacije ne steku.
- Vrednost `(rsp+8)` mora biti deljiva sa 16 u trenutku poziva procedure.

Opštenamenski registri

64-bit	32-bit	16-bit	8-bit
`rax`	`eax`	`ax`	`al`
`rbx`	`ebx`	`bx`	`bl`
`rcx`	`ecx`	`cx`	`cl`
`rdx`	`edx`	`dx`	`dl`
`rsi`	`esi`	`si`	`sil`
`rdi`	`edi`	`di`	`dil`
`rbp`	`ebp`	`bp`	`bpl`
`rsp`	`esp`	`sp`	`spl`
`r8`	`r8d`	`r8w`	`r8b`
`r9`	`r9d`	`r9w`	`r9b`
`r10`	`r10d`	`r10w`	`r10b`
`r11`	`r11d`	`r11w`	`r11b`
`r12`	`r12d`	`r12w`	`r12b`
`r13`	`r13d`	`r13w`	`r13b`
`r14`	`r14d`	`r14w`	`r14b`
`r15`	`r15d`	`r15w`	`r15b`

Vrste operanada

Tip	Zapis	Vrednost
Neposredni	`Imm`	`Imm`
Registarski	`Rx`	`R[Rx]`
Memorijski	`[Rx]`	`M[R[Rx]]`
Memorijski	`[Rx + Imm]`	`M[R[Rx] + Imm]`
Memorijski	`[Rx + Ry]`	`M[R[Rx] + R[Ry]]`
Memorijski	`[Rx + Ry + Imm]`	`M[R[Rx] + R[Ry]+ Imm]`
Memorijski	`[Rx + s*Ry]`	`M[R[Rx] + s*R[Ry]]`
Memorijski	`[Rx + s*Ry + Imm]`	`M[R[Rx] + s*R[Ry] + Imm]`

- `s` (od `scale`) se neposredno navodi, i mora imati neku od vrednosti: 1, 2, 4 ili 8.
- Nekada iz konteksta instrukcije nije moguće zaključiti koja je širina memorijskog operanda, na primer kod instrukcije `inc [Rx]`. U tim slučajevima neophodno je navesti širinu pre operanda, na primer ako bi inkrementirali podatak širine dva bajta `inc DWORD PTR [Rx]`.

Naziv	Širina	Oznaka
byte	8-bit	`BYTE PTR`
word	16-bit	`WORD PTR`
double word	32-bit	`DWORD PTR`
quad word	64-bit	`QWORD PTR`

Instrukcije za prenos podataka

Instrukcija	Efekat	Opis
`mov D, S`	`D := S`	move
`movsx D, S`	`D := signExtend(S)`	move sign extend
`movzx D, S`	`D := zeroExtend(S)`	move zero extend
`movabs R, Imm`	`R := Imm`	mov 64-bit immediate to a register

- Kod `mov` instrukcija najviše jedan operand može biti memorijski.
- Kod instrukcije `movabs` destinacion operand je registar, a izovrisni neposredna 64-bitna vrednost. Kod ostalih `mov` instrukcija neposredna vrednost je 32-bitna.

Aritmetičke i logičke operacije

Instrukcija	Efekat	Opis
`inc D`	`D := D+1`	Increment
`dec D`	`D := D-1`	Decrement
`neg D`	`D := -D`	Negate
`not D`	`D := ~D`	Complement
`add D, S`	`D := D+S`	Add
`sub D, S`	`D := D-S`	Subtract
`imul D, S`	`D := D*S`	Multiply
`xor D, S`	`D := D^S`	Exclusive-or
`or D, S`	`D := D|S`	Or
`and D, S`	`D := D&S`	And
`sal D, k`	$D \ll_A k$	Arithmetic left shift
`shr D, k`	$D \ll_L k$	Logic left shift
`sar D, k`	$D \gg_A k$	Arithmetic right shift
`shr D, k`	$D \gg_L k$	Logic right shift

- Instrukcije za sabiranje, oduzimanje i mnozenje su iste za označene i neozančene brojeve.
- Drugi argument `shift` instrukcija je neposredni ili registar `cl` (nižih osam bita registra `rcx`).
- Ako je prvi operand neke od `shift` instrukcija širine `w` bita, vrednost za koju se šiftuje se određuje ne osnovu najnižih $\texttt{log}_2\texttt{(w)}$ bita drugog operanda posmatranih kao neoznačen ceo broj. Na primer, ako je vrednost koja se šiftuje širine 64-bita, vrednost za koje ima smisla šiftovati su `0, 1, ..., 63`, za zapis ovih vrednosti je dovoljno $\texttt{log}_2\texttt{(64)=6}$ bita.

Posebne aritmetičke operacije

Instrukcija	Efekat	Opis
`imul S`	`R[rdx]:R[rax] := S*R[rax]`	Signed full multiply
`mul S`	`R[rdx]:R[rax] := S*R[rax]`	Unsigned full multiply
`idiv S`	`R[rax] := R[rdx]:R[rax] div S` `R[rdx] := R[rdx]:R[rax] mod S`	Signed divide
`idiv S`	`R[rax] := R[rdx]:R[rax] div S` `R[rdx] := R[rdx]:R[rax] mod S`	Unigned divide
`cqo`	`R[rdx]:R[rax] := signeExtend(R[rax])`	Convert to oct word

Instrukcija `lea`

Instrukcija	Efekat	Opis
`lea D, S`	`D := &S`	Load effective address

- Instrukcija `lea` se često koristi za izračunavanje jednostavnih aritmetičkih izraza. Na primer, potrebno izračunati vrednost `x+4*y+12` i rezultat smestiti u registar `rcx`, vrednost `x` je u registru `rax`, a `y` u `rdx`. Ovo se može postici sa dve instrukcije sabiranje i jednom instrukcijom množenja, ili korišćenjem jedne `lea` instrukcije, `lea rcx, [rax + 4*rdx + 12]`.

Flegovi

Oznaka	Naziv	Značenje
`CF`	`Carry flag`	Poslednja operacija je generisala prenos na bitu najveće težine (detekcija `overflow`-a kod neoznačenih brojeva).
`ZF`	`Zero flag`	Poslednja operacija je za rezultat imala nulu.
`SF`	`Sign flag`	Poslednja operacija je imala negativan rezultat.
`OF`	`Overflow flag`	Poslednja operacija je izazvala `overflow` u komplementu dvojke (detekcija `overflow`-a kod označenih brojeva).

- Aritmetičke instrukcije postaljavju sve flegove u skladu se njihovim definicijama.
- Logičke instrukcije, postavljaju flegove `SF` i `ZF` u skladu sa njihovim definicijama, dok flagove `CF` i `OF` postavljaju na `0`.
- Instrukcije šiftovanja postavljaju flegove `SF` i `ZF` u skladu sa njihovim definicijama, `CF` postavljaju na poslednji bit koji je "iz-šiftovan", dok `OF` postavljaju na 0.
- Ne postoji način za direktan pristup vrednostim flegova.

Instrukcije `cmp` i `test`

Instrukcija	Opis
`cmp S1, S2`	Računa vrednost `S1-S2`, ne beleži rezultat, ali menja flegove.
`test S1, S2`	Računa vrednost `S1&S2`, ne beleži rezultat, ali menja flegove.

`set` instrukcije

- Postavljaju najniži bajt operanda na `0x00` ili `0x01`, u zavisnosti od vrednosti flegova.

Instrukcija	Sinonim	Efekat	Opis
`sete D`	`setz`	`D := ZF`	Equal / zero
`setne D`	`setnz`	`D := ~ZF`	Not equal / not zero
`sets D`		`D := SF`	Negative
`setns D`		`D := ~SF`	Nonnegative
`setg D`	`setnle`	`D := ~(SF^OF)&~ZF`	Greater (signed `>`)
`setge D`	`setnl`	`D := ~(SF^OF)`	Greater or equal (signed `>=`)
`setl D`	`setnge`	`D := SF^OF`	Less (signed `<`)
`setle D`	`setng`	`D := (SF^OF)|ZF`	Less or equal (signed `<=`)
`seta D`	`setnbe`	`D := ~CF&~ZF`	Above (unsigned `>`)
`setae D`	`setnb`	`D := ~CF`	Above or equal (unsigned `>=`)
`setb D`	`setnae`	`D := CF`	Below (unsigned `<`)
`setbe D`	`setna`	`D := CF|ZF`	Below or equal (unsigned `<=`)

Instrukcije kontrole toka

Instrukcija	Sinonim	Uslov	Opis
`jmp label`		`1`	Uncoditional jump
`jmp *Operand`		`1`	Unconditional jump
`je label`	`jz`	`ZF`	Jump if equal / zero
`jne label`	`jnz`	`~ZF`	Jump if not equal / not zero
`js label`		`SF`	Jump if negative
`jns label`		`~SF`	Jump if nonnegative
`jg label`	`jnle`	`~(SF^OF)&~ZF`	Jump if greater (signed `>`)
`jge label`	`jnl`	`~(SF^OF)`	Jump if greater or equal (signed `>=`)
`jl label`	`jnge`	`SF^OF`	Jump if less (signed `<`)
`jle label`	`jng`	`(SF^OF)|ZF`	Jump if less or equal (signed `<=`)
`ja label`	`jnbe`	`~CF&~ZF`	Jump if above (unsigned `>`)
`jae label`	`jnb`	`~CF`	Jump if above or equal (unsigned `>=`)
`jb label`	`jnae`	`CF`	Jump if below (unsigned `<`)
`jbe label`	`jna`	`CF|ZF`	Jump if below or equal (unsigned `<=`)

Instrukcije uslovnog prenosa podataka

- Efekat ovih instrukcija je `R := S` ukoliko je ispunjen uslov, u suprotnom ništa.
- Destinacioni operand mora biti registar.

Instrukcija	Sinonim	Uslov	Opis
`cmove R, S`	`cmovz`	`ZF`	Move if equal / zero
`cmovne R, S`	`cmovz`	`~ZF`	Move if not equal / not zero
`cmovs R, S`		`SF`	Move if negative
`cmovns R, S`		`~SF`	Move if nonnegative
`cmovg R, S`	`jnle`	`~(SF^OF)&~ZF`	Move if greater (signed `>`)
`cmovge R, S`	`cmovnl`	`~(SF^OF)`	Move if greater or equal (signed `>=`)
`cmovl R, S`	`cmovnge`	`SF^OF`	Move if less (signed `<`)
`cmovle R, S`	`cmovng`	`(SF^OF)|ZF`	Move if less or equal (signed `<=`)
`cmova R, S`	`cmovnbe`	`~CF&~ZF`	Move if above (unsigned `>`)
`cmovae R, S`	`cmovnb`	`~CF`	Move if above or equal (unsigned `>=`)
`cmovb R, S`	`cmovnae`	`CF`	Move if below (unsigned `<`)
`cmovbe R, S`	`cmovna`	`CF|ZF`	Move if below or equal (unsigned `<=`)

Instrukcije za rad sa stekom

- `ISA` definiše registar `rsp` kao `stack-pointer`, a ne `ABI`.

Instrukcija	Efekat	Opis
`push S`	`R[rsp] := R[rsp] - 8; M[R[rsp]] := S`	Push on stack
`pop D`	`D := M[R[rsp]]; R[rsp] := R[rsp] + 8`	Pop from stack

Instrukcije za rad sa procedurama

Instrukcija	Opis
`call labela`	Procedure call
`call *Operand`	Procedure call
`ret`	Return from call

- Instrukcija `call` na stek postavlja povratnu adresu (adresu instrukcije koja se nalazi nakon `call` instrukcije), a u registar `rip` upisuje vrednost operanda.
- Instrukcija `ret` sa vrha steka skida povratnu adresu i upisuje je u registar `rip`.

System V ABI

Registar	Namena	Vrednost očuvana između poziva
`rax`	1st return register	`NO`
`rbx`	general purpose	`YES`
`rcx`	4th argument register	`NO`
`rdx`	3rd argument register, 2nd return register	`NO`
`rsi`	2nd argument register	`NO`
`rdi`	1st argument register	`NO`
`rbp`	frame pointer	`YES`
`rsp`	stack pointer	`YES`
`r8`	5th argument register	`NO`
`r9`	6th argument register	`NO`
`r10`	general purpose	`NO`
`r11`	general purpose	`NO`
`r12`	general purpose	`YES`
`r13`	general purpose	`YES`
`r14`	general purpose	`YES`
`r15`	general purpose	`YES`