

# NASM Cheat Sheet

NASM (Netwide ASseMbler) je assembly kompajler. Kod kucan za NASM mora da se prevodi, kao i kod kucan npr. za C. Osnovna razlika je što je kod kucan za NASM više nalik nativnom kodu mašine od koda kucanog u bilo kom jeziku višeg nivoa.

## Kompajlovanje

Kompajlovanje NASM koda u izvršnu datoteku se vrši pomoću sledeće konzolne komande:

```
nasm -f bin primer.asm -o primer.com
```

Napomena: za detalje podešavanja okruženja, tako da se nasm komanda može pozvati iz bilo kog direktorijuma pogledati prvu prezentaciju sa vežbi.

Ova komanda se sastoji iz četiri razdvojena dela:

`nasm` → Naziv komande. Mora da se nalazi na početku linije. (ostala tri argumenta mogu da se daju bilo kojim redosledom)

`-f bin` → Format izlazne datoteke. Često korišćene vrednosti su `bin` (izvršni fajl), `obj` (16-bit objektni fajl za DOS), `win32` (32-bit objektni fajl za windows), `elf` (32-bit objektni fajl za linux). Pogledati NASM dokumentaciju za ostale moguće vrednosti ovog parametra.

`primer.asm` → Putanja do fajla sa izvornim kodom koji želimo da kompajlujemo.

`-o primer.com` → Putanja do fajla koji želimo da dobijemo kompajlovanjem.

## Registri i adresiranje

Da bi procesor izvršio bilo koju komandu, vrednost sa kojom treba da radi mora da se nalazi u nekom registru. DOS podržava rad isključivo sa registrima od 16 bita (u realnom režimu), dok windows i linux operativni sistemi podržavaju rad sa 32 i 64 bita, zavisno od verzije.

Postoje 4 registra opšte namene. Kažemo da su opšte namene zato što u našem kodu možemo da ih koristimo za bilo šta, i možemo po želji da im menjamo vrednost. S druge strane, neke sistemske funkcije podrazumevaju da se neki od ovih registara koriste na specifičan način.

Pored toga, postoje “nepisana” pravila za primenu ovih registara:

`ax` → Accumulator. Koristi se da drži broj na koji se dodaju drugi vrednosti pri sabiranju, množenju itd.

`bx` → Base pointer. Koristi se kao pokazivač pri referenciranju memorije.

`cx` → Counter. Koristi se kao brojač.

`dx` → Aritmetičke i I/O operacije.

Svaki od ova četiri registra je 16-bitan. Njihove 32-bit varijante imaju slovo e na početku (eax, ebx, ecx, edx). Svaki od ova četiri 16-bit registara je podeljen na dve 8-bit varijante (gornju i donju): ah, al, bh, bl, ch, cl, dh, dl.

Još neki bitni registri:

cs → Code segment. Naznačava segment u memoriji u kojem se nalazi kod.

ds → Data segment. Naznačava segment u memoriji u kojem se nalaze podaci za naš proces.

es → Extra segment. Koristi se sa pristupanje proizvoljnom segmentu u memoriji.

ss → Stack segment. Naznačava segment u kojem se nalazi stack našeg procesa.

sp → Stack pointer. Pokazivač na stek unutar stek segmenta. push i pop operacije modifikuju vrednost ovog registra.

si → Source index. Za indeksiranje nizova i pristup memoriji.

di → Destination index. Slično kao si.

bp → Base pointer. Sličan registru bx.

ip → Instruction pointer. Pokazuje na trenutnu instrukciju unutar cs.

Da bi se konstruisala prava adresa u memoriji, neophodna su četiri bajta. Na 16-bitnim mašinama, ovo su dva registra. Za konkatenciju registara koristi se zapis:

ax:bx

Gde je ax više bitan par bajtova, a bx manje bitan. Uglaste zagrade označavaju dereferenciranje, tj. ono što je unutar uglastih zagrada se tretira kao adresa, a vrednost izraza je ono što se nalazi na zadatoj adresi. Npr, da bismo upisali 0 na adresu koja se nalazi na es:bx:

```
mov [es:bx], 0
```

## ***Rad sa memorijom***

Pseudoinstrukcije za rezervisanje prostora u memoriju su oblika resx n (gde je n broj rezervisanih mesta):

resb → Rezerviše n bajtova.

resw → Rezerviše n reči (jedna reč je dva bajta, tj. 16 bita)

resd → Rezerviše n duplih reči (4 bajta)

resq → Rezerviše n četvorostrukih reči (8 bajtova)

Pre resx komande se može postaviti labela, koja se kasnije može koristiti da označi prvi bajt u skupu rezervisanih bajtova. Npr:

niz: resb 10

Rezerviš 10 bajtova, i smešta adresu prvog u labelu niz.

Slično funkcionišu pseudoinstrukcije dx, kod kojih mora odmah da se navede i vrednost koja će biti smeštena u rezervisanom bloku memorije, npr:

poruka: db 'Zdravo svete',0

Nakon pseudoinstrukcije dx, zarezom se odvajaju vrednosti koje su različitog tipa, a koje želimo da se nalaze jedne za drugom u memoriji. Vrednosti zadate pod apostrofima se tretiraju kao ASCII vrednosti zadatih karaktera.

Više uzastopnih db instrukcija označava konsektivne vrednosti u memoriji. Npr. primer odozgo se drugačije može napisati:

poruka:

db 'Zdravo svete'

db 0

## ***Operacije nad stekom***

Stek je memorijska struktura koja je dodeljena svakom tasku. Stek našeg taska se nalazi u ss (stek segment). Trenutna vrednost na steku se nalazi na lokaciji koja je upisana u sp (stack pointer). Jeda veoma bitna osobina steka kao strukture je da se koristi pri pozivu podprocedura. Pri izvršavanju call instrukcije, trenutna adresa IP (instruction pointer) se smešta na stek, da bi se pri izvršenju ret instrukcije ona isčitala. Ako podprocedura ne ostavi stek čist, to može da izazove nepredviđeno ponašanje. Argumenti funkcije višeg programskog jezika se tipično smeštaju na stek u obrnutom redosledu od onog u kom su dati u višem programskom jeziku, i to pre poziva procedure.

push ax → Smešta vrednost iz ax na stek. (dekrementira sp za dva)

pop ax → Uzima vrednost sa steka i smešta u ax (inkrementira sp za dva)

pusha → Smešta sve registre opšte namene na stek sledećim redom: ax, cx, dx, bx, sp, bp, si, di. Gurnuta vrednost za sp je početna, dakle pre izvršenja pusha. pusha je alias koji se kompajluje ili u pushaw (dvobajtni push) ili pushad (četvorobajtni push)

popa → Skida sa steka vrednosti i smešta u registre opšte namene: di, si, bp, ništa (odbacuje se vrednost za sp), bx, dx, cx, ax. popa je alias za popaw ili popad, slično kao pusha.

enter 2, 0 → Kreira stek koji se može koristiti kao deskriptor za funkciju iz višeg programskog jezika. Prvi argument je broj bajtova koje treba zauzeti u negativnom pravcu. Drugi argument je dubina (nivo ugnežđenja procedure) steka, koja može da ide do 31. Poziv instrukcije enter za dubinu 0 je ekvivalentan sledećem:

push bp

```
mov bp, sp
sub sp, op_1
```

Gde je op\_1 prvi operand. Primetiti da enter ima jedan implicitan push, kao što leave ima implicitan pop.

leave → Uništava stek napravljen prethodnom komandom enter. Ekvivalentno sledećem:

```
mov sp, bp
pop bp
```

### ***Aritmetičke i logičke operacije***

Logičke i aritmetičke operacije se najčešće obavljaju nad registrima opšte namene. Rezultat operacije se smešta u levi operand.

add ax, bx → Sabira vrednosti u ax i bx i smešta rezultat u ax.

sub cx, dx → Oduzima dx od cx i smešta rezultat u cx.

Kod množenja i deljenja, prvi operand se obavezno nalazi u ax (ili al). Zavisno od drugog operanda (koji mora biti registar, tj. ne može biti konstanta) rezultat se smešta u ax ili dx:ax.

mul bx → Množi ax i bx i smešta rezultat u dx i ax.

mul bl → Množi al i bl i smešta rezultat u ax.

div bx → Deli dx:ax sa bx. Celobrojni rezultat deljenja je u ax, a ostatak u dx.

div bl → Deli ax sa bl. Celobrojni rezultat deljenja je u al, a ostatak u ah.

Logičke operacije:

or al, bl → Ili operacija

and al, bl → I operacija

xor al, bl → Isključivo ili

not al → Negacija

### ***Bitovske operacije***

Postoje logičke i aritmetičke operacije pomeranja bitova. Pomeranje bitova generalno može da se tretira kao množenje i deljenje sa dva, ali postoje posebni slučajevi, tako da je bolje ne uzimati ovo kao opšte pravilo.

Logičko pomeranje translira sve bitove za jedno mesto levo ili desno. Ispražnjen bit se popunjava nulom, dok se bit koji „ispadne“ iz pomeraja gubi.

`shr ax, 1` → pomera sve bitove u `ax` jedno mesto desno.

`shl ax, 1` → pomera sve bitove u `ax` jedno mesto levo.

Aritmetički shift pri pomeraju ulevo (množenje) zadržava vrednost najznačajnijeg bita. Ovo znači da ako se koristi jednobajtni zapis negativnog broja sa 7 bitova vrednosti i jednim bitom znaka, negativni brojevi do -64 se mogu množiti sa 2 pomeranjem levo.

`sar ax, 1` → aritmetički pomeraj desno.

`sah ax, 1` → aritmetički pomeraj levo.

Rotacija bitova je slična kao pomeranje. Jedina razlika je da se poslednji bit, koji „ispadne“ iz pomeranja, prepisuje u prvi bit.

`ror ax, 4` → vrši rotaciju za 4 bita desno.

`rol ax, 4` → vrši rotaciju za 4 bita levo.

## **Skokovi**

Bezuslovni skok se vrši instrukcijom `jmp`. Argument je adresa na koju se skače, obično labela:

`jmp pocetak` → Nastavlja sa izvršavanjem od labele „pocetak“.

Pre uslovnog skoka se obično vrši ili neka aritmetička operacija ili poređenje pomoću insturcije `cmp`.

`cmp al, bl` → Poredi sadržaj `al` i `bl` i podešava flegove koji se koriste pri uslovnim skokovima

`jne nejednako` → Nastavlja sa izvršavanjem od labele „nejednako“, ako operandi u prethodnom poređenju nisu bili jednaki.

`je jednako` → Nastavlja sa izvršavanjem od labele „jednako“, ako su operandi u prethodnom poređenju bili jednaki.

`jg vece` → Nastavlja sa izvršavanjem od labele „vece“, ako je levi operand u prethodnom poređenju veći od desnog.

`jl manje` → Nastavlja sa izvršavanjem od labele „manje“, ako je levi operand u prethodnom poređenju manji od desnog.

`jge veceili jednako` → Nastavlja sa izvršavanjem od labele „veceilijednako“, ako je levi operand u prethodnom poređenju veći ili jednak desnom.

`jle manjeilijednako` → Nastavlja sa izvršavanjem od labele „manjeilijednako“, ako je levi operand u prethodnom poređenju manji ili jednak desnom.

## ***Pozivi i labele***

Poziv podprograma se vrši komandom `call`. Podprogram se završava komandom `ret`, nakon koje se tok programa nastavlja od poziva podprograma.

`call print` → Poziva `print` kao podproceduru.

Ako labela počinje tačkom, ona je lokalna. Lokalna labela se može pozvati svojim imenom samo u okviru iste labele koja nema tačku, npr.

```
main:
add ax,bx
.sub:
sub ax,bx
jz .sub
```

Ovde je labela `.sub` podlabela labele `main`. Nakon naredne labele koja nema tačku na početku, `.sub` neće označavati ništa, ali će moći da se koristi pun naziv labele: `main.sub`.

## ***Brojačke petlje***

Postoje dva često korišćena načina da se skup instrukcija ponovi zadat broj puta:

```
mov cx, 10
petlja:
;kod koji treba da se ponavlja
loop petlja
```

Komanda `loop` dekrementira vrednost u `cx` dok ne dođe do nule. Ako `cx` još uvek nije nula, vrši se skok na navedenu labelu.

```
times 10 db 0
```

Komanda `times` asemblira instrukciju zadat broj puta.

## ***Neki prekid***

Prekid	Opis
10h	BIOS prekid za rad sa ekranom. Zavisno od vrednosti smeštene u <code>ah</code> , prekidna rutina obavlja različite operacije. Neke vrednosti za <code>ah</code> : <ul style="list-style-type: none"><li>• 02h → Postavljanje kursora. <code>d1</code> je kolona, <code>dh</code> je red.</li><li>• 0eh → Ispis karaktera čiji ASCII kod je smešten u <code>a1</code>.</li></ul>

16h	<p>BIOS prekid za rad sa tastaturom. . Zavisno od vrednosti smeštene u ah, prekidna rutina obavlja različite operacije. Neke vrednosti za ah:</p> <ul style="list-style-type: none"> <li>• 0h → Čitanje sa tastature. ASCII kod učitanoj karaktera se smešta u a1.</li> </ul>
21h	<p>DOS sistemski poziv. Zavisno od vrednosti smeštene u ah, prekidna rutina obavlja različite operacije. Neke vrednosti za ah:</p> <ul style="list-style-type: none"> <li>• 9h → Ispis stringa koji se nalazi na adresi zadatoj sa dx. String je terminiran znakom '\$'.</li> </ul>
9h	<p>Rad sa tastaturom. Prekidna rutina se poziva od strane hardvera pri svakom pritisku i otpuštanju dugmeta na tastaturi.</p>
1CH	<p>Korisnički tajmerski prekid. Hardverski sat na svakih 55ms poziva prekidnu rutinu 8h, koja obavlja svoju logiku, i između ostalog poziva 1CH. 1CH tipično ima samo instrukciju <code>iret</code>, tako da je savršeno bezbedno zameniti hendler.</p>