IA 32

Tomaž Dobravec, Sistemska programska oprema

Krastka zgodovina x86

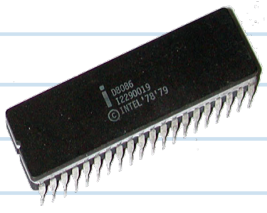
- IA 32 (Intel's 32bit computer architecture) IA32 == x86

Intel 8080 (1974)

- 8-bitni procesor

Intel 8086 (1978) ... XT

- prvi Intelov 16-bitni procesor
- Hitrost ure: 4.77MHz - 10MHz
- 29.000 tranzistorjev
- naslovni prostor: do 1MB (20-bitno vodilo)
  - "640k ought to be enough for anybody." - Bill Gates, 1981



Intel 286 (1982) ... AT

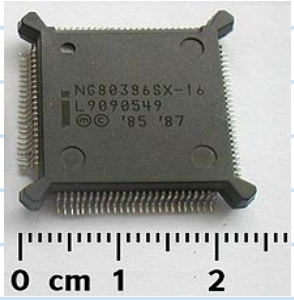
- prvi množični PC (v prvih 6 letih proizvedli 15 milionov računalnikov)
- hitrost: 6 – 12.5 MHz
- 134.000 tranzistorjev
- Pomnilnik: do 16MB (24-bitno vodilo)

- današnja tehnologija omogoča izdelavo čipov, ki so 33x manjši

Krastka zgodovina x86

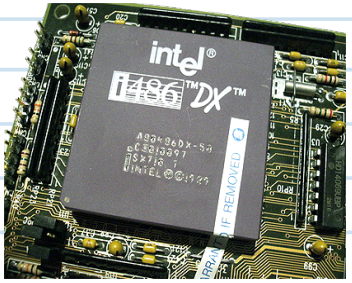
Intel 386 (1985)

- prvi Intelov 32-bitni procesor
- lahko si naslovil 4GB pomnilnika (32-bitno vodilo)
- 16-33MHz
- 275.000 tranzistorjev



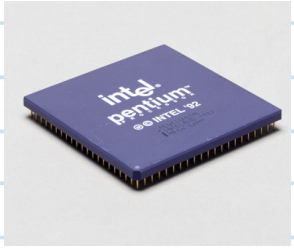
Intel 486 (1989)

- prvi z integriranim matematičnim koprocesorjem (486DX, 486SX)
- prvi, ki je imel več kot milion tranzistorjev na čipu
- hitrost: od 25-100MHz



Intel Pentium, 586 (1993)

- hitrost: 60-233MHz
- 3 - 4.5 mio tranzistorjev
- 64 bitno bus vodilo



Intel Core i7 (2008)

- hitrost 2.5 – 3Ghz
- 731 mio tranzistorjev
- 64 bitni naslovni prostor (do 128 TB)

Povzetek:

Leto	Procesor	Hitrost	Tranzistorji
1978	Intel 8086	4.77-10Mhz	29.000
1982	Intel 80286	6-12.5 MHz	134.000
1985	Intel 80386	16-33Ghz	275.000
1989	Intel 80486	25-100MHz	1mio
1993	Pentium	do 233 MHz	4.5 mio
2008	Intel Core i7	2.5GHz	731mio

Moore's law: The number of components per integrated circuit (i.e. transistors) will double every two years.

64-bitni svet

- Intel skušal prodreti z Itanium IA-64 (2000), vendar ne preveč uspešno
  - paralelizem na nivoju ukazov
  - paralelizem določi prevajalnik (in ne procesor v času izvajanja)

- AMD bolj uspešen z **AMD64** (tudi **x64**)
  - naravno nadaljevanje x86 arhitekture; 100% kompatibilno z x86

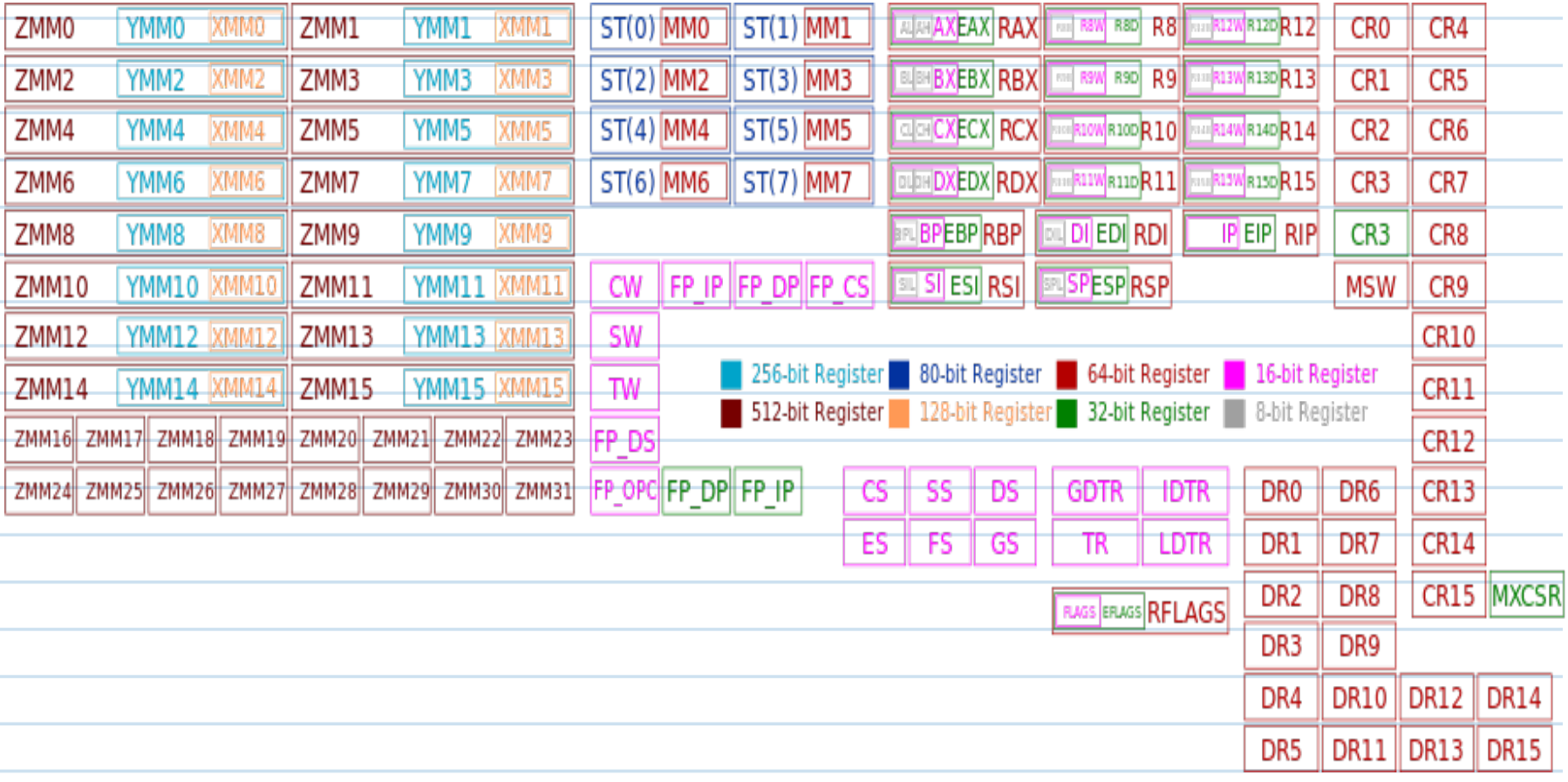
Pomnilnik

- Z 32-bitnimi registri bi lahko naslovili le 4GB pomnilnika.
- Z uporabo zaščitenega načina (”protected mode”) lahko to omejitev presežemo.

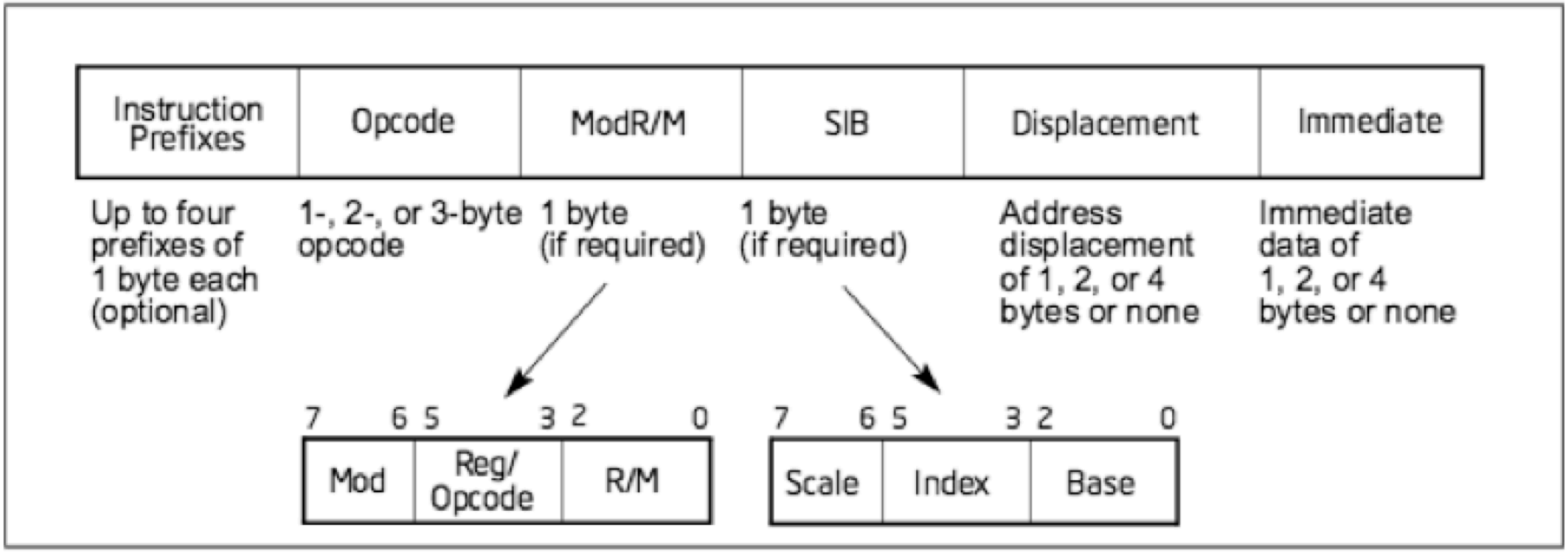
- Osnovna naslovljiva enota: 1 byte
- Beseda (**WORD**): 2 bajta
- Dvojna beseda (**DWORD**) : 4 bajti
- Podaljšana beseda (**QWORD**): 8 bajtov

Registri

Vsi registri procesorja x64



Formati ukazov



Delo s skladom

- sklad se uporablja za shranjevanje vrednosti lokalnih spremenljivk in rezultatov
- na sklad odlagamo / iz sklada pobiramo 32-bitne vrednosti
- ESP (stack pointer) kaže na vrh sklada (zadnji odloženi element)

PUSH

```
push <reg32>
push <mem>
push <con32>
```

- zmanjša ESP za 4
- nato odloži operand na lokacijo, kamor kaže ESP

POP

```
pop <reg32>
pop <mem>
```

- vrednost, na katero kaže ESP, shrani v operand
- nato ESP poveča za 4

Uporaba EBP in ESP pri klicu podprograma (Calling Convention)

- Register EBP se uporablja za shranjevanje prejšnje vrednosti ESP.
- S pomočjo vrednosti EBP dostopamo do parametrov podprograma in lokalnih spremenljivk.
- ESP vedno kaže na vrh trenutnega sklada, EBP kaže na trenutni okvir.

Načini naslavljanja

Registrsko naslavljanje

Takojšnje naslavljanje (drugi operand je takojšnja konstanta)

Neposredno naslavljanje (naloži neposredno iz pomnilnika preko podanega naslova)

Neposredno naslavljanje z odmikom

Registrsko posredno naslavljanje (dostop do pomnilnika preko naslova, ki je shranjen v registru)

✧ **mov** — Move (Opcodes: 88, 89, 8A, 8B, 8C, 8E, ...)

```
mov <reg>,<reg>
mov <reg>,<mem>
mov <mem>,<reg>
mov <reg>,<const>
mov <mem>,<const>
```

✧ **lea** — Load effective address

```
lea <reg32>,<mem>
```

✧ **add** — Integer Addition

```
add <reg>,<reg>
add <reg>,<mem>
add <mem>,<reg>
add <reg>,<con>
add <mem>,<con>
```

✧ **sub** — Integer Subtraction

```
sub <reg>,<reg>
sub <reg>,<mem>
sub <mem>,<reg>
sub <reg>,<con>
sub <mem>,<con>
```

✧ **inc, dec** — Increment, Decrement

```
inc <reg>
inc <mem>
dec <reg>
dec <mem>
```

imul — Integer Multiplication

```
imul <reg32>,<reg32>
imul <reg32>,<mem>
imul <reg32>,<reg32>,<con>
imul <reg32>,<mem>,<con>
```

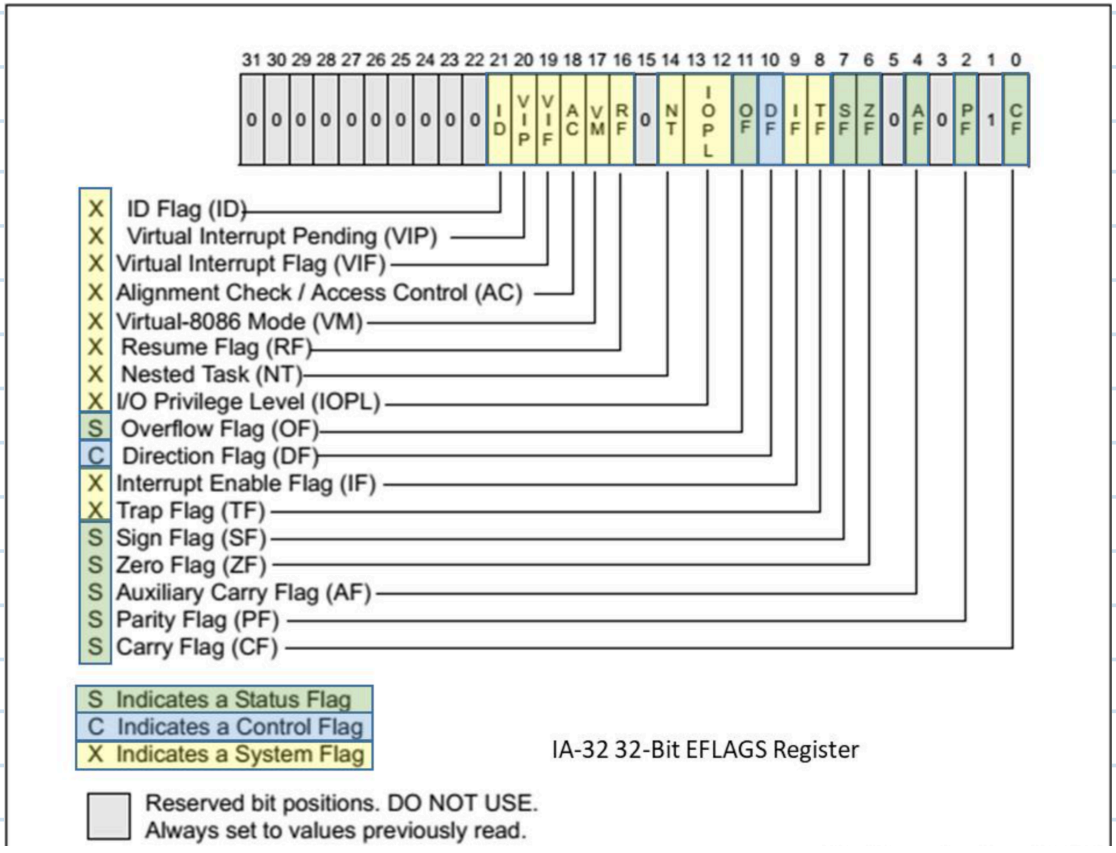
✧ **idiv** — Integer Division

```
idiv <reg32>
idiv <mem>
```

Bitne operacije – `and, or, xor, not, neg, shl, shr`

Zastavice

- zastavice opisujejo stanje procesorja; od njihovih vrednosti je odvisen potek nadaljnjega izvajanja
- zastavice se prižigajo/ugašajo kot posledica operacij, lahko pa jih nastavljamo tudi ”ročno”
- tri vrste zastavic: statusne / kontrolne / sistemske
- zastavice procesorja x86 so shranjene v registru `FLAGS (EFLAGS, RFLAGS)`



Zastavice - podrobneje

Kontrolna zastavica

- DF (direction flag)
- vpliva na operacije, ki delujejo v "auto-increment" načinu; na primer: MOVS … kopira večji del (1/2/4/8 bajtov) pomnilnika iz enega na drugo mesto;
- če je DF=0, operacija pomnilnik prekopira v vrstnem redu od spodnjega do zgornjega naslova; če je DF=1 pa od zgodnjega proti spodnjemu;
- zastavica je pomembna, če se izvorna in ciljna lokacija prekrivata.

Statusne zastavice

- se nastavijo kot rezultat operacije
- na primer: `ADD EAX, EBX` … ta operacija vrne rezultat (`EAX+EBX`); odvisno od tega rezultata se lahko prižge/ugasne katera od zastavic

ZF (zero flag)

- ZF=1, če je rezultat enak 0, ZF=0 sicer

SF (sign flag)

- SF=1, če je rezultat negativen, SF=0, če je rezultat pozitiven;
- SF = zgornjemu bitu.

Zastavice - podrobneje

PF (Parity flag)

- Gledamo spodnjih 8 bitov rezultata (najmanj pomemben bajt); če ta vsebuje sodo število prižganih bitov → `PF=1`, sicer `PF=0`;
- včasih uporabno (kot preprost checksum test), danes manj;
- zastavica je ohranjena zaradi kompatibilnosti.

CF (Carry flag) in **OF (Overflow flag)**

- Ugotavljanje pravilnosti aritmetičnih operacij;
- če je zastavica prižgana → rezultat prejšnje operacije je bil napačen;
- OF se uporablja pri prednačeni, CF pri nepredznačeni aritmetiki.

Zastavice - podrobneje

CF podrobneje

- nepredznačena aritmetika
- zastavica se prižge v dveh primerih:
  - če pri seštevanju najpomembnejši bit prenesemo za eno mesto
  - če si pri odštevanju sposodimo (borrow) preko najpomembnejšega biti

✧ OF podrobneje

- predznačena aritmetika
- zastavica se prižge le v dveh primerih:
  - če da vsota dveh pozitivnih števil (sign bit == 0) negativen rezultat
  - če da vsota dveh negativnih števil (sign bit == 1) pozitiven rezultat
  - pogledati je treba le zgornji biti treh števil, da ugotoviš vrednost OF

Zastavice - podrobneje

Preveri vrednost zastavic po naslednjih operacijah

- `3+7 → PF = 1`
- `3+8 → PF = 0`
- `0x7fffffff + 1 → OF=1, CF=0`
- `0xffffffff + 1 → OF=0, CF=1`

Skočni ukazi

Instruction	Description	signed-ness	Flags	short jump opcodes	near jump opcodes
JO	Jump if overflow		OF = 1	70	0F 80
JNO	Jump if not overflow		OF = 0	71	0F 81
JS	Jump if sign		SF = 1	78	0F 88
JNS	Jump if not sign		SF = 0	79	0F 89
JE **JZ**	Jump if equal Jump if zero		ZF = 1	74	0F 84
JNE **JNZ**	Jump if not equal Jump if not zero		ZF = 0	75	0F 85
JB **JNAE** **JC**	Jump if below Jump if not above or equal Jump if carry	unsigned	CF = 1	72	0F 82
JNB **JAE** **JNC**	Jump if not below Jump if above or equal Jump if not carry	unsigned	CF = 0	73	0F 83
JBE **JNA**	Jump if below or equal Jump if not above	unsigned	CF = 1 or ZF = 1	76	0F 86
JA **JNBE**	Jump if above Jump if not below or equal	unsigned	CF = 0 and ZF = 0	77	0F 87
JL **JNGE**	Jump if less Jump if not greater or equal	signed	SF <> OF	7C	0F 8C
JGE **JNL**	Jump if greater or equal Jump if not less	signed	SF = OF	7D	0F 8D
JLE **JNG**	Jump if less or equal Jump if not greater	signed	ZF = 1 or SF <> OF	7E	0F 8E
JG **JNLE**	Jump if greater Jump if not less or equal	signed	ZF = 0 and SF = OF	7F	0F 8F
JP **JPE**	Jump if parity Jump if parity even		PF = 1	7A	0F 8A
JNP **JPO**	Jump if not parity Jump if parity odd		PF = 0	7B	0F 8B
JCXZ **JECXZ**	Jump if %CX register is 0 Jump if %ECX register is 0		%CX = 0 %ECX = 0	E3	

Direktive za deklaracijo in inicializacijo

- tri direktive `DB`, `DW` in `DD` ... za deklaracijo (in inicializacijo) enodimenzionalnih tabel podatkov (zaporedje spominskih celic)

Primeri:

Sistemski klici

- `int 0x80` ... sistemski klici (le na Linux arhitekturi)
- Številka sistemkega klica je v `eax`, ostali registri vsebujejo morebitne dodatne parametre

Seznam sistemskih klicev:

`exit`	1
`fork`	2
`read`	3
`write`	4
`open`	5
`close`	6
`waitpid`	7
`creat`	8
`link`	9
`unlink`	10
`execve`	11
`chdir`	12
`time`	13
`mknod`	14
`chmod`	15
`lchown`	16

Sintaksa Intel v.s. AT&T

Asembler `as` podpira obe – Intel in AT&T sintaksi

- `.intel_syntax, .att_syntax`
- `prefix, noprefix`

Glavne razlike

Pred operandi v AT&T so zapisani posebni znaki ($, %, ...)

- `pushl $4` `push 4`

o Obratni vrstni red operandov

- `addl $4, %eax` `add eax, 4`

o Dodatni znak za vsakim ukazom v AT&T sintaksi

- `movb foo, %al` `mov al, byte ptr foo`

o Razlika pri dolgih skokih

- `lcall/ljmp $section, $offset` `call/jmp far section:offset`

Nekaj primerov

- Program v jeziku C
  - prevedem z `gcc -S test.c` → dobim prevedeno asm kodo v `test.s` datoteki (AT&T sintaksa)
  - dodam stikalo `-masm=intel` (za Intel sintakso) in `-m32` za prevod v IA32
  - stikala `-fno-asynchronous-unwind-tables -fno-dwarf2-cfi-asm` za lepšo in bolj razumljivo asm kodo

- Program hello.asm za izpis besedila Hello world v asm.
  - prevedeš z
```
nasm -f elf main.asm
ld -m elf_i386 -s -o demo *.o
```

- Program za izpis abecede

Pisanje zbirne kode v asm bloku jezika C

- ASM blok

```
asm ( assembler template
      : output operands                    (optional)
      : input operands                     (optional)
      : clobbered registers list           (optional)
);
```

- Primer:

```
1  #include <stdio.h>
2
3  int
4  get_random(void)
5  {
6      asm(".intel_syntax noprefix\n"
7          "mov eax, 42           \n");
8  }
9  int
10 main(void)
11 {
12     return printf("The answer is %d.\n", get_random());
13 }
```

Prevajanje: `gcc prog.c -m32 -masm=intel`

Primeri programov v asm bloku

- Funkcija `zamenjaj()`, ki zamenja vrednosti lokalnih spremenljivk
- Produkt dveh števil brez operatorja * (seštevanje v zanki)
- Program, ki sešteje vrednost podanih argumentov in na zaslon izpiše stanje zastavic po operaciji
- Množenje vektorjev po komponentah