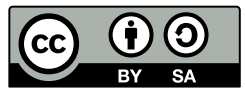
Intel x86 procesorių architektūra

Saulius Gražulis

Vilnius, 2020

Vilniaus universitetas, Matematikos ir informatikos fakultetas
Informatikos institutas

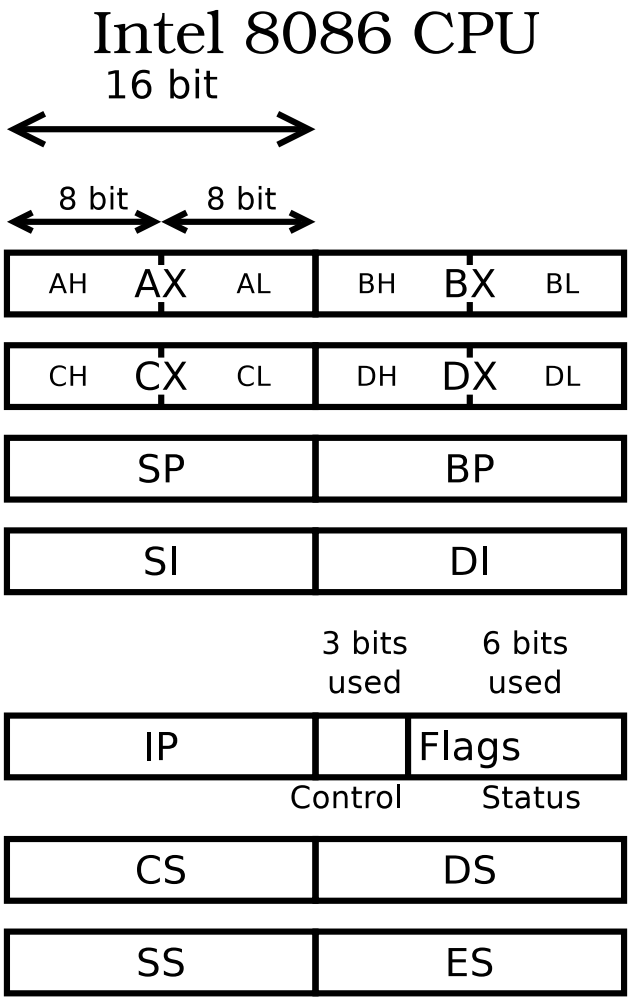


Ką reikia nurodyti apie procesoriaus architektūrą?

Programuotojui matoma architektūra:

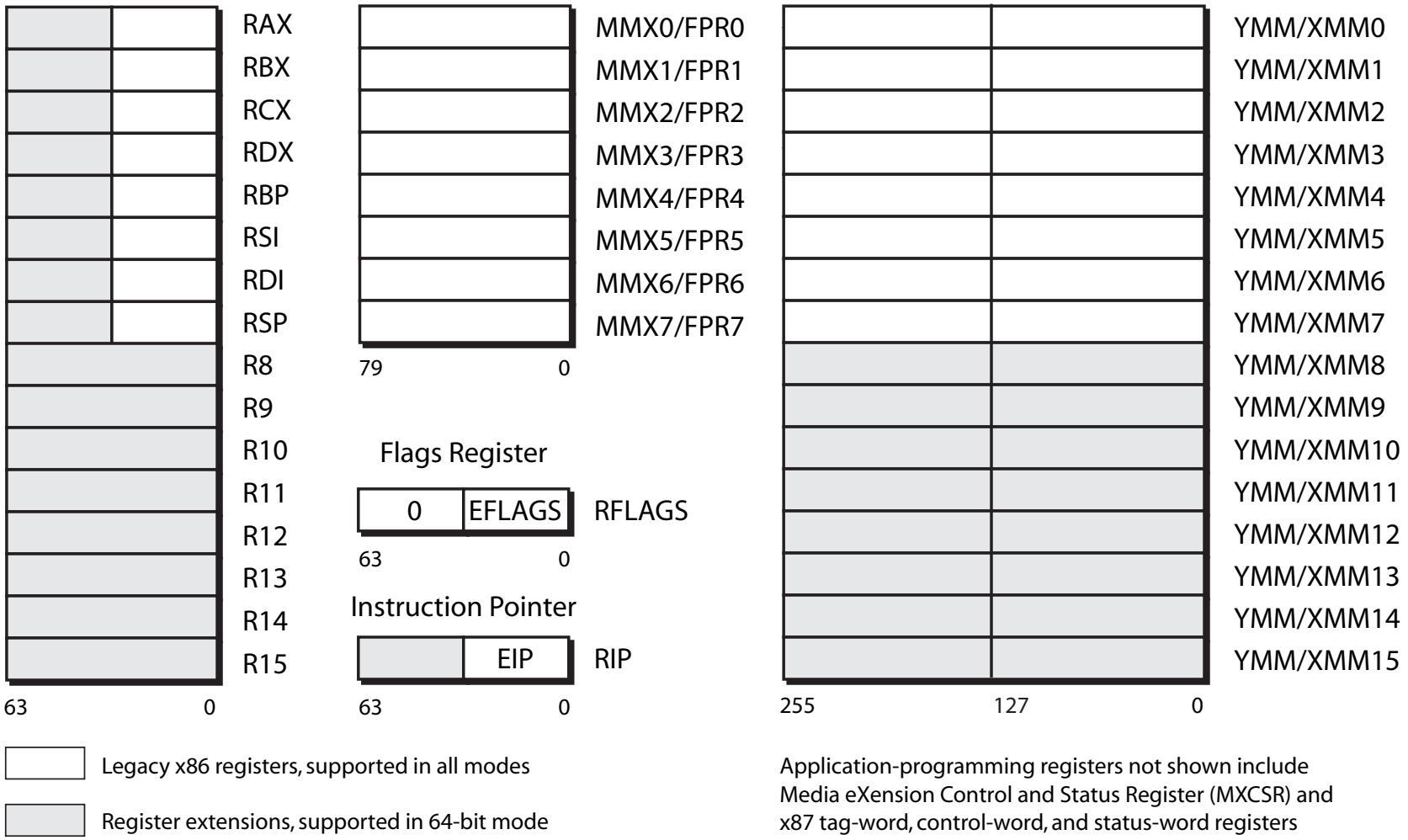
- Registrai, matomi programuotojui
- Atminties adresavimas
- Duomenų formatai
- Procesoriaus komandos
- Įvestis ir išvestis
- Pertraukimų apdorojimas

Registrai (x86)



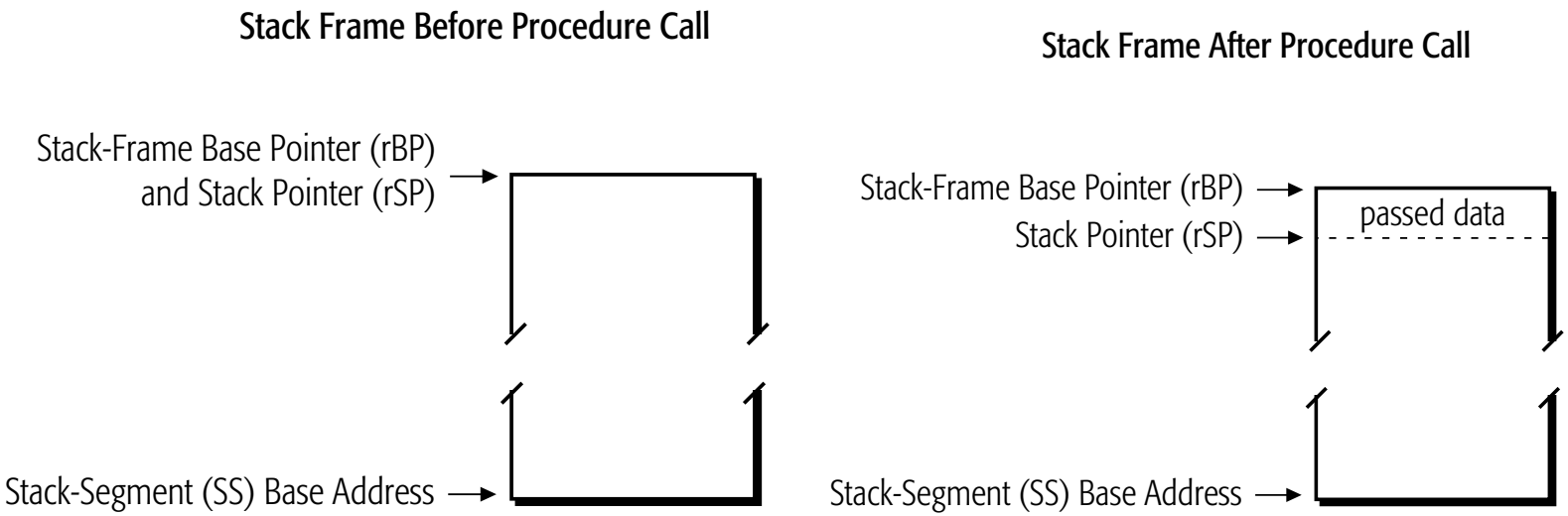
(Intel 1979)

Registrai (x86_64)



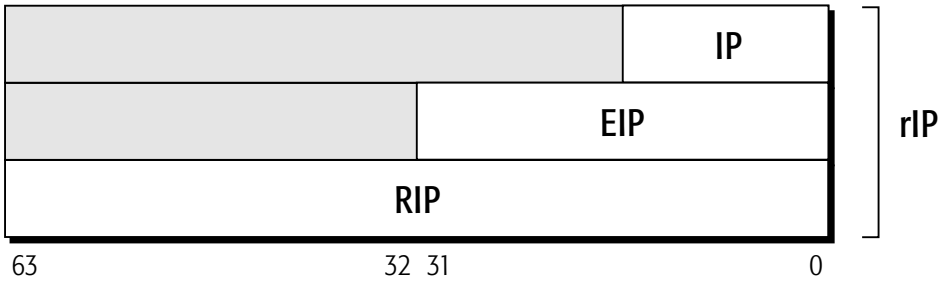
(AMD 2017)

Steko veikimas



(AMD 2017)

Komandų skaitliukas



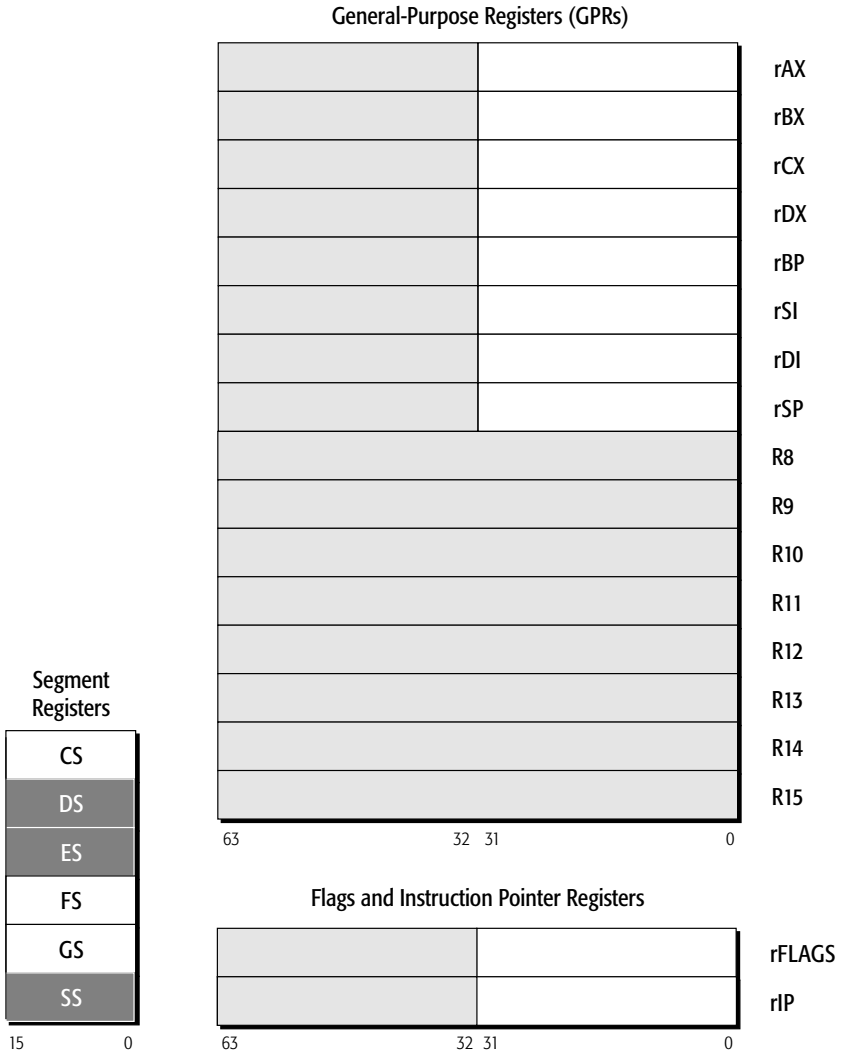
(AMD 2017)

Registrai, suderinamumo režimai

register encoding		high 8-bit	low 8-bit	16-bit	32-bit
0		AH (4)	AL	AX	EAX
3		BH (7)	BL	BX	EBX
1		CH (5)	CL	CX	ECX
2		DH (6)	DL	DX	EDX
6		SI		SI	ESI
7		DI		DI	EDI
5		BP		BP	EBP
4		SP		SP	ESP

(AMD 2017)

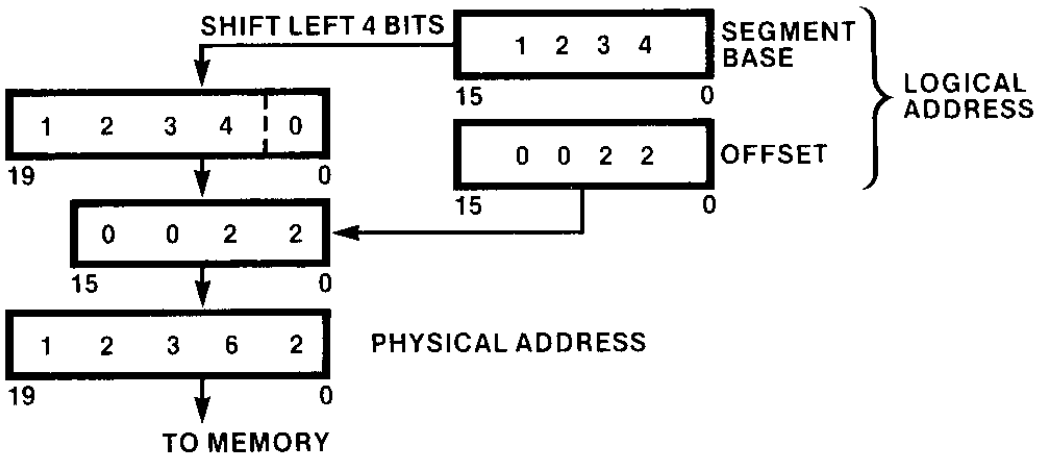
Registrai



(AMD 2017)

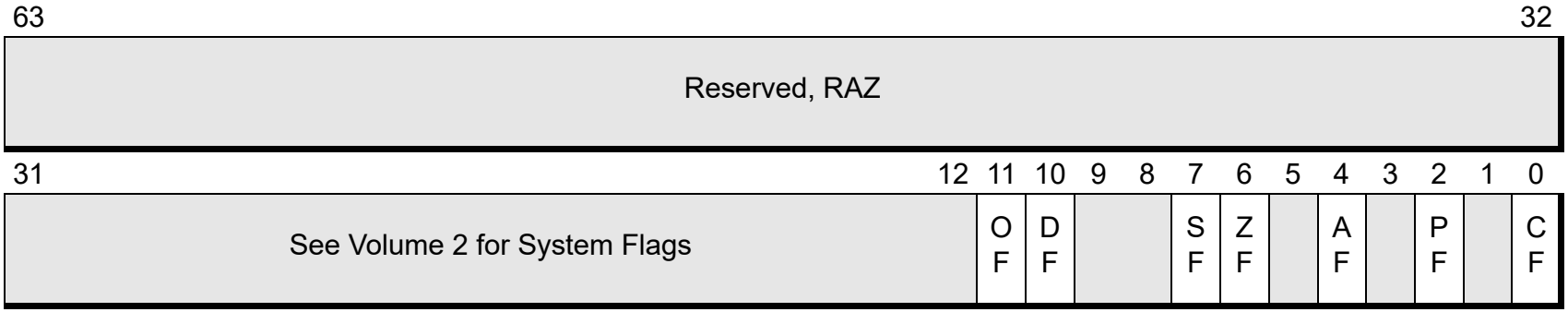
Segmentinė adresacija

Segmento adresas pastumiamas 4-iais bitais ir sudedamas su postūmiu:



(Intel 1979)

Požymių registras



Bits	Mnemonic	Description	R/W
11	OF	Overflow Flag	R/W
10	DF	Direction Flag	R/W
7	SF	Sign Flag	R/W
6	ZF	Zero Flag	R/W
4	AF	Auxiliary Carry Flag	R/W
2	PF	Parity Flag	R/W
0	CF	Carry Flag	R/W

(AMD 2017)

Požymiai

- **Perkėlimo požymis (CF)**
- **Perpildymo požymis (OF)**
- **Pagalbinio perpildymo požymis (AF)**
- **Lyginimo požymis (PF)**
- **Nulio požymis (ZF)**
- **Ženklo (neigiamo) požymis (SF)**
- **Krypties požymis (DF)**

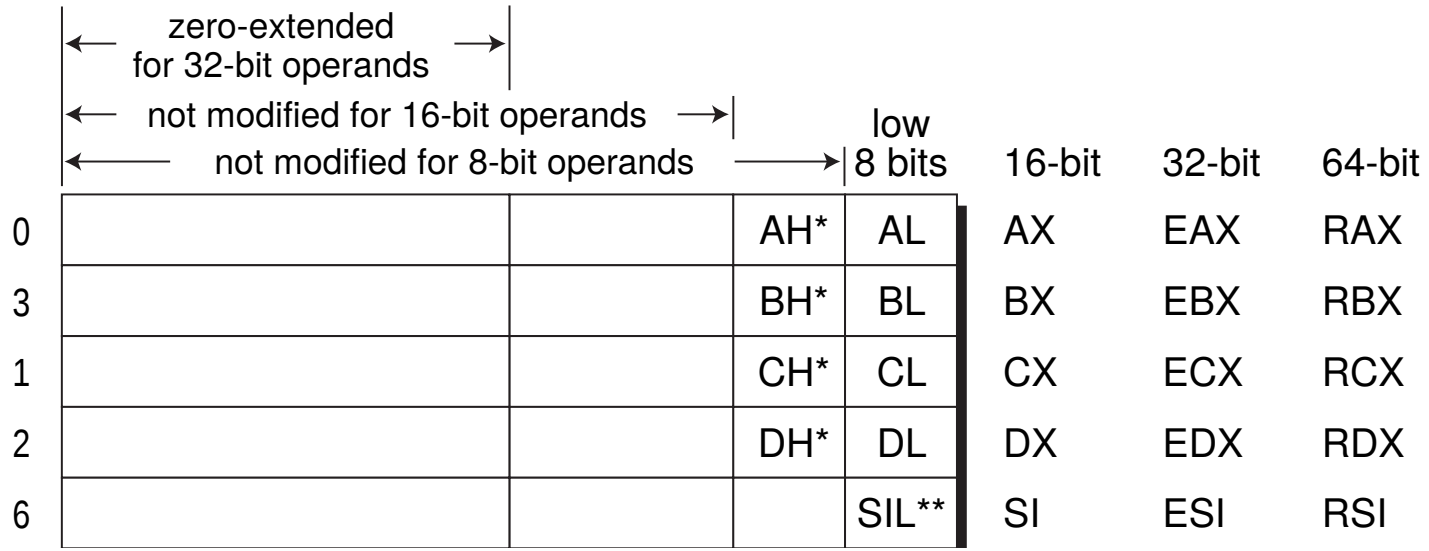
Komandų tipai

- Duomenų persiuntimo
- Duomenų transformacijos komandos
- Aritmetinės komandos
- Loginės komandos
- Valdymo komandos

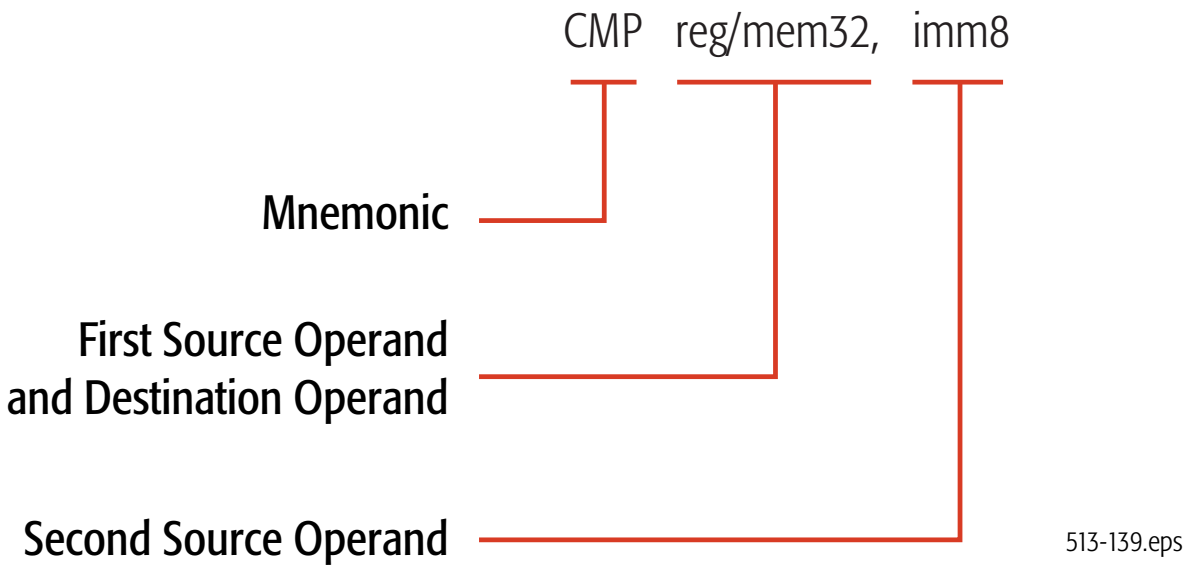
(Jorgensen 2020)

Išplėtimas nuliais įkeliant 32-jų bitų operandus

Kai 32-jų bitų operandai įkeliami į bendros paskirties registrus 64-ių bitų režime, skaičius išplečiamas nuliais (pav. 3-3, 3-4).



Komandų sintaksė



(AMD 2017)

https://en.wikipedia.org/wiki/X86_instruction_listings

Komandų skaičius

- Apie 100 skirtingų instrukcijų (asemblerio lygyje) (Intel 1979)
- Bet: pvz., viena `MOV` instrukcija atitinka 28 skirtingas mašinos komandas (operacijų kodus)...
  - ... `MOV` yra pilna pagal Tiuringą! (Dolan 2013)
- Procesoriaus x86_64 architektūra turi, grubiai tariant, apie 1000 (ar 3000) komandų, priklausomai nuo to, kaip skaičiuosime... [1]
- Procesoriaus x86_64 architektūros *komandų erdvė* (1-15 baitų) yra $1.3 \cdot 10^{36}$ galimų komandų. Šį skaičių galima sumažinti iki aprėpiamo $\approx 10^8$ naudojant gudrią paiešką į gylį (Domas 2017)

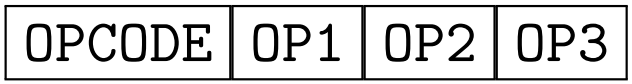
[1] https://stefanheule.com/blog/how-many-x86-64-instructions-are-there-anyway

Operandų skaičius

- 0 adresų mašinos: 

- 1 adreso mašinos: 

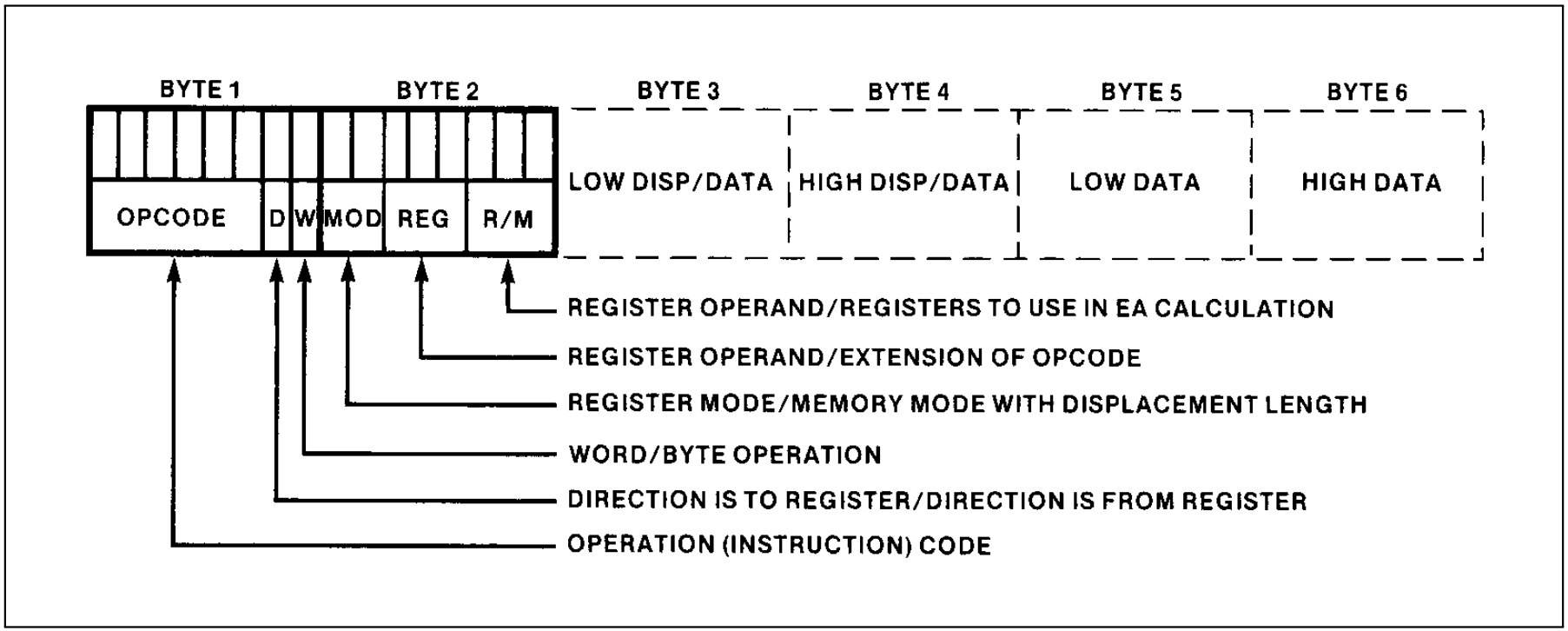
- 2 adresų mašinos: 

- 3 adresų mašinos: 

- 4 adresų mašinos: 

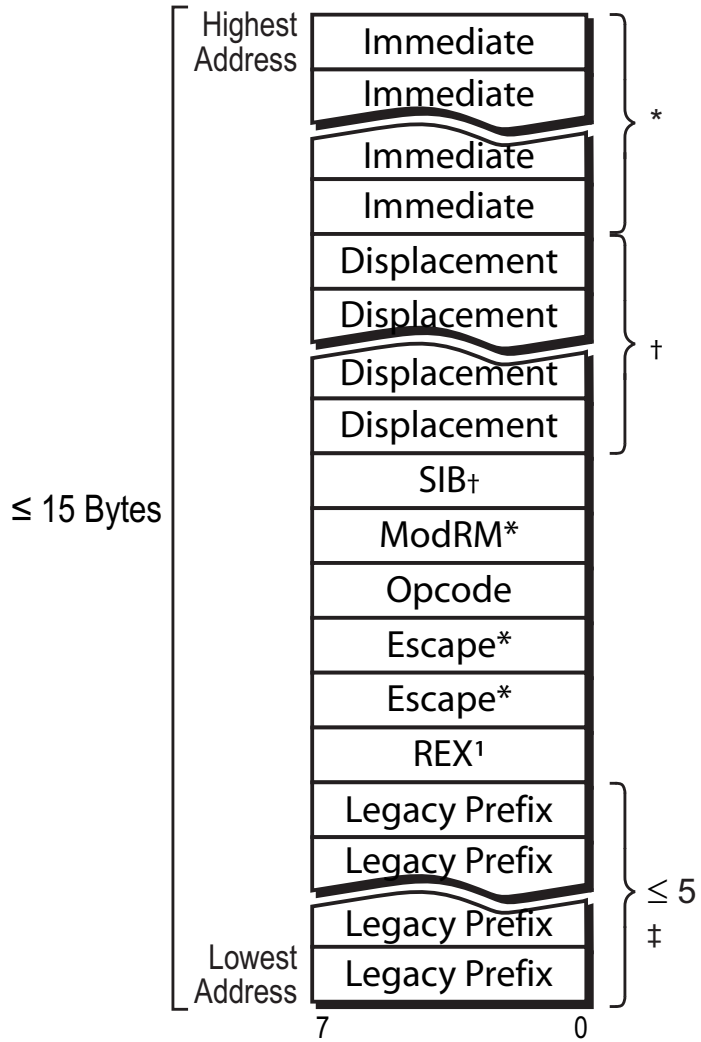
Komandų kodavimas (formatas)

Tik vienas operandas gali būti atmintyje!



Intel 8086 Family User's guide

Komandų formatas – 64-ių bitų režimas



(AMD 2017), Vol. 3

Registrų kodavimas

Table 4-9. REG (Register) Field Encoding

REG	W = 0	W = 1
000	AL	AX
001	CL	CX
010	DL	DX
011	BL	BX
100	AH	SP
101	CH	BP
110	DH	SI
111	BH	DI

Intel 8086 Family User's guide, p. 162

Antrojo registro ir režimo kodavimas

Table 4-10. R/M (Register/Memory) Field Encoding

MOD = 11			EFFECTIVE ADDRESS CALCULATION			
R/M	W = 0	W = 1	R/M	MOD = 00	MOD = 01	MOD = 10
000	AL	AX	000	(BX) + (SI)	(BX) + (SI) + D8	(BX) + (SI) + D16
001	CL	CX	001	(BX) + (DI)	(BX) + (DI) + D8	(BX) + (DI) + D16
010	DL	DX	010	(BP) + (SI)	(BP) + (SI) + D8	(BP) + (SI) + D16
011	BL	BX	011	(BP) + (DI)	(BP) + (DI) + D8	(BP) + (DI) + D16
100	AH	SP	100	(SI)	(SI) + D8	(SI) + D16
101	CH	BP	101	(DI)	(DI) + D8	(DI) + D16
110	DH	SI	110	DIRECT ADDRESS	(BP) + D8	(BP) + D16
111	BH	DI	111	(BX)	(BX) + D8	(BX) + D16

Intel 8086 Family User's guide, p. 162

MOV komanda

nasm:

```
 5 00000000 A0[2D00]                    mov al, [b1]
 6 00000003 8A26[2E00]                  mov ah, [b2]
 7 00000007 A1[2F00]                    mov ax, [w1]
 8
 9 0000000A 8A1E[2D00]                  mov bl, [b1]
10 0000000E 8A3E[2E00]                  mov bh, [b2]
11 00000012 8B1E[2F00]                  mov bx, [w1]
12
13 00000016 8B18                        mov bx, [bx + si]
14 00000018 8B98[2F00]                  mov bx, [bx + si + w1]
15
16 0000001C 8B00                        mov ax, [bx + si]
17 0000001E 8B803412                    mov ax, [bx + si + 1234h]
18
20 00000024 89C8                        mov ax, cx
21 00000026 88C4                        mov ah, al
22
```

LEA komanda

nasm:

```
 5 00000000 8D1A                        lea bx, [bp + si]
 6 00000002 8D9A[1300]                  lea bx, [bp + si + w1]
 7
 8 00000006 8D02                        lea ax, [bp + si]
 9 00000008 8D82[1300]                  lea ax, [bp + si + w1]
10
11                                      ;;  exit:
12 0000000C B8004C                      mov ax, 4c00h
13 0000000F CD21                        int 21h
14
15 00000011 55                  b1:     DB 055h
```

XCHG komanda

```
 5 00000000 8706[1000]                      xchg ax, [w1]
 6 00000004 8606[1200]                      xchg al, [b1]
 7
 8 00000008 92                              xchg ax, dx
 9 00000009 86D8                            xchg bl, al
10
11                                          ;;  exit:
15 00000010 2301                    w1:     DW 0123h
16 00000012 0100                    b1:     DW 01h
```

If a memory operand is referenced, the processor's locking protocol is automatically implemented for the duration of the exchange operation, regardless of the presence or absence of the LOCK prefix or of the value of the IOPL. (See the LOCK prefix description in this chapter for more information on the locking protocol.) This instruction is useful for implementing semaphores or similar data structures for process synchronization.

https://c9x.me/x86/html/file_module_x86_id_328.html

2020-03-23

ADD, ADC komandos

nasm:

```
 5 00000000 01D8                        add ax, bx
 6 00000002 11D1                        adc cx, dx
 7
 8 00000004 01F2                        add dx, si
 9 00000006 0314                        add dx, [si]
10
11 00000008 0316[1E00]                  add dx, [w1]
12 0000000C 0394[1E00]                  add dx, [si+w1]
13 00000010 03547F                      add dx, [si+07Fh]
14 00000013 03948000                    add dx, [si+080h]
15
20 0000001C 55                  b1:     DB 055h
21 0000001D AA                  b2:     DB 0AAh
22 0000001E 2301                w1:     DW 0123h
```

INC, DEC komandos

```
 5 00000000 40                                  inc ax
 6 00000001 49                                  dec cx
 7
 8 00000002 FE04                                inc byte [si]
 9 00000004 FF447F                              inc word [si+07Fh]
10 00000007 FF888000                            dec word [si+bx+080h]
```

REP STOS komandos

```
 5 00000000 FC                                  cld
 6 00000001 B0AA                                mov al, 0AAh
 7 00000003 BF[1000]                            mov di, str
 8 00000006 B91200                              mov cx, strend - str
 9
10 00000009 F3AA                                rep stosb
11
16 00000010 412076657279206C6F-         str:    DB 'A very long srting'
16 00000019 6E6720737274696E67
17 00000022 00                           strend: DB 0
```

LOOP komanda

```
 5 00000000 B90500                      mov cx, 5
 6 00000003 B80300                      mov ax, 3
 7 00000006 89C3                        mov bx, ax
 8 00000008 89C6                        mov si, ax
 9
10 0000000A F7EB                begin:  imul bx
11 0000000C 0FAFF3                      imul si, bx
12 0000000F E2F9                        loop begin
```

JMP,Jcc komandos

```
 5 00000000 E410                        in   al, 010h
 6
 7 00000002 3C00                        cmp al, 0
 8
 9 00000004 7A04                        jpe even
10 00000006 B401                        mov ah, 1
11 00000008 EB02                        jmp finish
12 0000000A B402                  even: mov ah, 2
13                                finish:
```

PUSH/POP komandos

```
 5 00000000 50                  push ax
 6 00000001 56                  push si
 7 00000002 1E                  push ds
 8 00000003 6A0A                push 10
 9
10 00000005 58                  pop ax
11 00000006 5B                  pop bx
12 00000007 59                  pop cx
13 00000008 5A                  pop dx
```

CALL, RET komandos

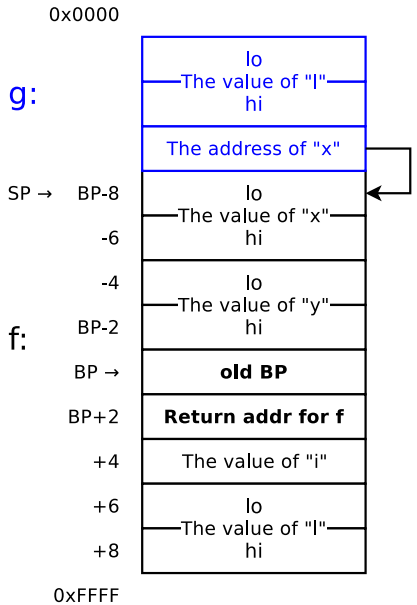
```
 5 00000000 E80500                          call subr
 6
11 00000008 B83412                  subr:  mov ax, 01234h
12 0000000B C3                             ret
```

Steko kadrai

```
int main( )
{
  static
  int a; long b;

  /* ... */
  a = f( a, b );
  /* ... */
  return 0;
}
```

```
mov ax, [b+2]
push ax
mov ax, [b]
push ax
mov ax, [a]
push ax
call f
add sp, 6
mov [a], ax
```




```
int f( int i, long l )
{
  float x, y;

  /* ... */
  i = g( l, &x );
  /* ... */
  return i;
}
```

```
push bp
mov bp, sp
sub sp, 8
; ...
lea ax, [bp-8]
push ax ; place &x onto the stack
mov ax, [bp+8] ; msw of "l"
push ax ; msw of "l" → stack
mov ax, [bp+6] ; lsw of "l"
push ax ; lsw of "l" → stack
call g
; ...
add sp, 8
pop bp
ret
```

SYSCALL/INT, SYSRET/IRET

16-bit (nasm):

```
 7                                          ;;  exit:
 8 00000003 B8004C                          mov ax, 4c00h
 9 00000006 CD21                            int 21h
```

64-bit (yasm):

```
 8                                  EXIT_SUCCESS equ 0
 9                                  SYS_exit equ 60
10                                  SYS_write equ 1
11
26 0000001F 48C7C03C000000          mov rax, SYS_exit
27 00000026 48C7C700000000          mov rdi, EXIT_SUCCESS
28 0000002D 0F05                    syscall
```

Šaltiniai

AMD (Dec. 2017). *AMD64 Architecture Programmer's Manual, Volume 1: Application Programming, revision 3.22*. AMD. URL: https://www.amd.com/system/files/TechDocs/24592.pdf.

Dolan, Stephen (2013). *mov is Turing-complete*. URL: https://drwho.virtadpt.net/files/mov.pdf.

Domas, Christopher (2017). *Breaking the x86 ISA*. URL: https://github.com/xoreaxeaxeax/sandsifter/blob/master/references/domas_breaking_the_x86_isa_wp.pdf.

Intel (Oct. 1979). *Intel 8086 Family User's Manual*. Intel Corporation. URL: https://edge.edx.org/c4x/BITSPilani/EEE231/asset/8086_family_Users_Manual_1_.pdf.

Jorgensen, Ed (Jan. 2020). *x86-64 Assembly Language Programming with Ubuntu*. Vol. 1. URL: http://www.egr.unlv.edu/ēd/assembly64.pdf. Ed Jorgensen.