3. Architektura procesoru - registry, typy operandů, formát instrukcí, adresování paměti, přerušení

Procesory Intel do roku 2000

CPU	rok	frekv. [MHz]	tranzistory		délka registru	paměť	FPU
			počet	velikost			
4004	1971	0,108	2,3 k	10 µm	GP: 4 b	4 KiB + 640 B	
8008	1972	0,5 – 0,8	3,5 k	10 µm	GP: 8 b	4 KiB + 16 KiB	
8080	1974	2	4,5 k	6 µm	GP: 8 b + 16 b	64 KiB	
8086/88	1978/79	5 – 10	29 k	3 µm	GP: 16 b	1 MiB	8087
80286	1982	6 – 25	134 k	1,5 µm	GP: 16 b	16 MiB	80287
80386	1985	12 – 40	275 k	1,5 µm	GP: 32 b	4 GiB	80387
80486	1989	16 – 150	1,2 M	1 µm	+FPU: 80 b		integrována
Pentium (P5)	1993	60 – 100	3,1 M	0,8 µm			
Pentium MMX	1996	120 – 233	4,5 M	0,35 µm	+MMX: 64 b		
Pentium Pro	1995	150 – 200	5,5 M	0,6 µm	-MMX !!!	64 GiB	
Pentium II	1997	233 – 450	7,5 M	0,25 µm	+MMX: 64 b		
Pentium III	1999	400 – 1 400	9,5 M	0,18 µm	+XMM: 128 b		
Pentium 4	2000	1 300 – 3 800	42+ M	0,18- µm			

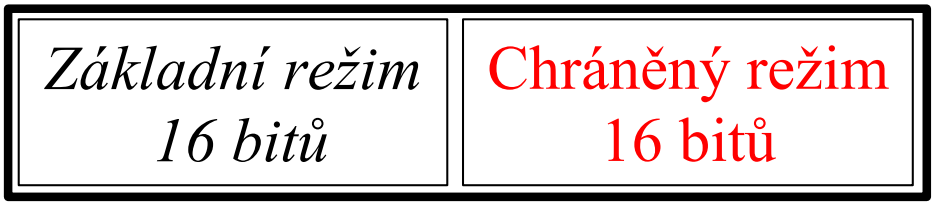
Procesory Intel po roce 2000

CPU	rok	frekv. [GHz]	tranzistory		délka registru	paměť	počet jader
			počet	velikost			
Pentium D	2005	2,66 – 3,73	228+ M	90 – 65 nm	GP: 64 b MMX: 64 b XMM: 128 b	až 64 GiB	2
Core 2	2006	1,06 – 3,33	291 M	65 – 45 nm			1 – 8
Core iX		1 – 4,5			+iGPU		
- Nehalem	2008		731 M	45 nm		až 256 TiB (platí pro serverové procesory, pro desktop je maximum 64 GiB)	
- Westmere	2010		382 M	32 nm			
- Sandy Bridge	2011		504 M	32 nm	+YMM: 256 b		
- Ivy bridge	2012		až 2,1 G	22 nm			2 – 12
- Haswell	2013		až 5,56 G	22 nm			2 – 18
- Broadwell	2014		?	14 nm			2 – 24
- Skylake	2015		?	14 nm	+YMM: 512 b		2 – 28
- Kabylake	2016		?	14 nm	?		?
- Cannonlake	2017	?	?	10 nm (?)	?		?

Režimy procesorů Intel (x86)

8086 (80186)

80286



80386, 80486, Pentium

Registry a operandy

Registry

31 … 16	15 … 8	7 … 0
A … Accumulator	AH	AL
	AX	
EAX		
B … Base	BH	BL
	BX	
EBX		
C … Counter	CH	CL
	CX	
ECX		
D … Data	DH	DL
	DX	
EDX		
Code Segment	CS	
Data Segment	DS	
Extra Data Seg.	ES	
Stack Segment	SS	

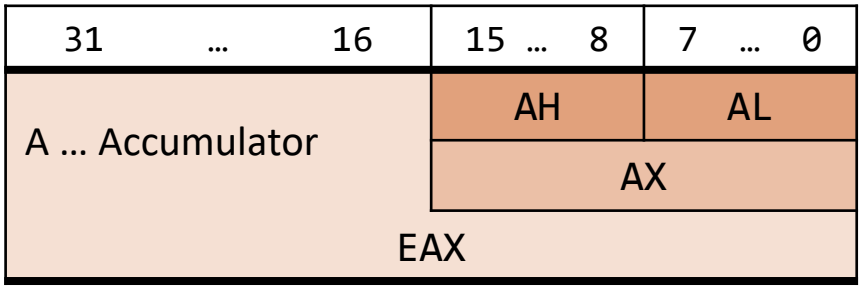
31 … 16	15 … 0
SP … Stack Pointer	SP
ESP	
BP … Base Pointer	BP
EBP	
SI … Source Index	SI
ESI	
DI … Destination Index	DI
EDI	
IP … Instruction Pointer	IP
EIP	
registr příznaků	FLAGS
EFLAGS	

H … High byte, **L** … Low byte

E … Extended (= 32bitový režim)

další segmentové registry – FS, GS

Registry – pojmenování částí registru a jejich vztah



31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	1	1
																								AL = 1 (3)							
																AH = 1 (1)															
																AX = 257 (259)															
EAX = 65 793 (65 795)																															

- Změna obsahu registru AL způsobí změnu obsahu registru AX i EAX
- Změna obsahu registru AX způsobí změnu obsahu registrů EAX, AL i AH
- Změna obsahu registru EAX způsobí změnu obsahu registrů AX, AL i AH

Využití registrů

- Obecné registry
  - (E)AX (AH,AL): střádač (akumulátor), matematické operace, obecné využití
  - (E)BX (BH,BL): základ efektivní adresy – ukazatel do paměti (v 16bitovém režimu)
  - (E)CX (CH,CL): čítač
  - (E)DX (DH,DL): data, nejvyšší významové slovo násobení a zbytek celočíselného dělení
- Indexové registry
  - (E)SP: ofset (ukazatel) vrcholu zásobníku
  - (E)BP: bázový ofset (ukazatel) zásobníku (parametry funkcí)
  - (E)SI: ofset (ukazatel) zdroje dat v paměti (řetězcové operace), indexování polí
  - (E)DI: ofset (ukazatel) cíle dat v paměti (řetězcové operace), indexování polí
- Segmentové registry
  - CS: kódový segment – ve spojení s instrukčním ukazatelem (E)IP ukazuje na následující instrukci (skok = změna CS:(E)IP)
  - DS: implicitní datový segment – ve spojení s (E)SI a (E)DI ukazuje na proměnné
  - SS: zásobníkový segment – ve spojení s (E)SP ukazuje na vrchol zásobníku a ve spojení s (E)BP ukazuje na jiné místo v zásobníku (např. parametry funkce)
  - ES: extra segmentový registr (pro řetězcové operace a jiné využití)
  - FS, GS: další extra segmentové registry (mají význam především v 16bitovém režimu)

Příznakový registr (E)FLAGS

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	NT	IOPL		OF	DF	IF	TF	SF	ZF	0	AF	0	PF	1	CF

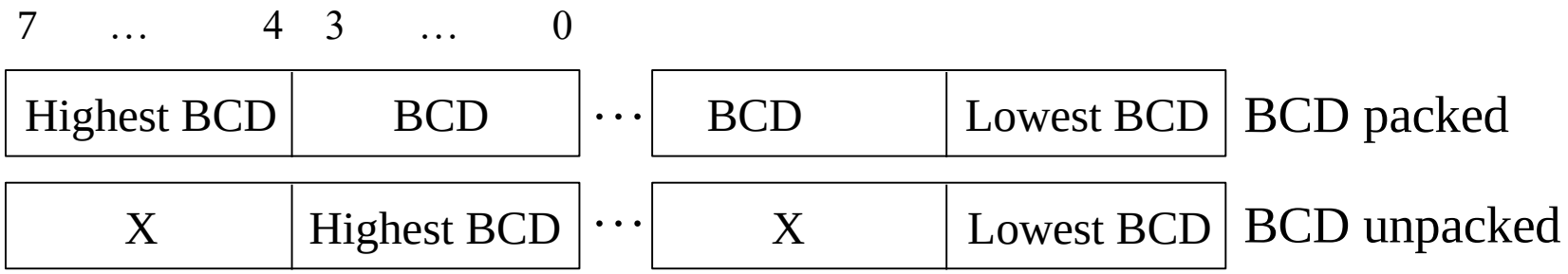
- **OF** (*Overflow Flag*) ~ příznak *overflow* (O, NO)
  - aritmetika se znaménkem (indikace přetečení: OF = 1)
- **CF** (*Carry Flag*) ~ příznak *carry* (C, NC)
  - aritmetika bez znaménka (indikace přetečení: CF = 1), rotace, posuny
- **SF** (*Sign Flag*) ~ příznak *sign* (S, NS)
  - znaménko výsledku (SF = 1: záporný, SF = 0: kladný)
- **ZF** (*Zero Flag*) ~ příznak *zero* (Z, NZ)
  - výsledek = 0 (ZF = 1: výsledek = 0, ZF = 0: výsledek != 0)
- **DF** (*Direction Flag*)
  - směr řetězových instrukcí (DF = 1: zvýšení, DF = 0: snížení)
- PF (*Parity Flag*) ~ příznak *parity* (PE, PO)
  - součet (počet) jedničkových bitů nejnižších 8 bitů výsledku je sudý (P = 1, PE)/lichý (P = 0, PO)
- AF (*Auxiliary Carry Flag*) = přenos z bitu 3 do bitu 4

Základní typy operandů

- Celá čísla

31 … 16	15 … 8	7 … 0			bez znaménka	se znaménkem
			byte	db	<0; 255>	<-127; 127>
	High byte	**L**ow byte	word	dw	<0; 65 535>	<-32 768; 32 767>
high word	low word		double word	dd	<0; 4 294 967 295>	<-2 147 483 648; 2 147 483 647>

- BCD čísla



- Znaky (ordinální čísla znaků – čísla bez znaménka – většinou byte)
- Řetězy slabik, slov, dvouslov (= pole 8-, 16-, 32-bitových hodnot)
- Paměťové operandy (= odkazy do paměti přes ukazatel – adresu)

Paměť, paměťové operandy

Paměť

- Hlavní paměť (*main memory*)
  - fyzická paměť
  - pole bytů indexované od 0
  - v 16bitovém režimu – kapacita 1 MiB (= 2^{20})
  - ve 32bitovém režimu – kapacita 4 GiB (= 2^{32}) i více
  - v jazyce C lze teoreticky definovat například jako pole bezznaménkových 8bitových hodnot `mem` o velikosti KAPACITA

```
unsigned char mem[KAPACITA];
```

- Fyzická adresa = ukazatel do fyzické paměti (*pointer*) = index do pole `mem`
  - 20bitová hodnota bez znaménka v 16bitovém režimu
  - 32bitová hodnota bez znaménka v 32bitovém režimu

`unsigned char mem[0x100000];`		
ADR	`mem[ADR]`	význam hodnoty
`0xFFFFF`	`0x00`	0
`adresa + 3`	`0x0C`	12
`adresa + 2`	`0x45`	‘A’
`adresa + 1`	`0x09`	00001001
`adresa`	`0x78`	`0x78`
`0x00001`	`0xFF`	-1
`0x00000`	`0x03`	1,5

Uložení čísla v paměti

- Dvě konvence
  - *Big Endian*: MSB (*Most Significant Byte*) je na nejnižší adrese
  - *Little Endian*: LSB (*Least Significant Byte*) je na nejnižší adrese
- Příklad uložení hodnoty 0x12345678 na adrese `adresa` v 16bitovém režimu znázorňuje tabulka →

Procesory Intel Pentium používají uložení typu **LITTLE ENDIAN**

	OBSAH PAMĚTI	
FYZICKÁ ADRESA	LITTLE ENDIAN	BIG ENDIAN
0xFFFFF	?	?
adresa + 3	0x12	0x78
adresa + 2	0x34	0x56
adresa + 1	0x56	0x34
adresa	0x78	0x12
0x00001	?	?
0x00000	?	?

Segmentová adresa (základní, 16bitový režim)

- Fyzická adresa – 20 bitů … Registry – 16 bitů => chybí 4 bity
- Segmentace
  - rozčlenění paměti na bloky o velikosti 64 KiB = **segment**
  - každý segment začíná na fyzické adrese, která je násobkem 16
  - posun relativně vůči počátku segmentu definuje 16bitový **ofset**
- Segment = 16bitová hodnota bez znaménka, udává se v **segmentovém registru** (`sreg`)
- Ofset = 16bitová hodnota bez znaménka, udává se v hranatých závorkách, například: `[0x012F]`
  - **přímá** hodnota (číslo – konstanta – *immediate operand*)
  - **nepřímá** hodnota (obsah některých registrů)
  - => přímé x nepřímé adresování
- Segmentová adresa = vzdálená adresa (far)
  - dvojice **segment:ofset**, zapisujeme **sreg:[ofset]**, například `DS:[0x012F]`
- Efektivní adresa (EA)
  - konkrétní hodnota ofsetu vyjádřená ve chvíli provádění kódu s paměťovým operandem, např. efektivní adresa operandu [BX + 0x0015] závisí na hodnotě BX

Implicitní segmentové registry

- použití určitého typu ofsetu při adresování = použití implicitně definovaného segmentového registru:

Typ odkazu paměti	implicitní sreg	alternativní sreg	ofset
Instrukce	CS	---	(E)IP
Implicitně zásobník	SS	---	(E)SP
Zdrojová řetězová data	DS	CS, ES, FS, GS, SS	(E)SI
Cílová řetězová data	ES	---	(E)DI
Data adresovaná (E)BP, (E)SP	SS	CS, DS, ES, FS, GS	EA
Jiná (standardní) data	DS	CS, ES, FS, GS, SS	EA

- Například:

```
[0x0115] = DS:[0x0115], [BP] = SS:[BP], [EIP] = CS:[EIP]
```

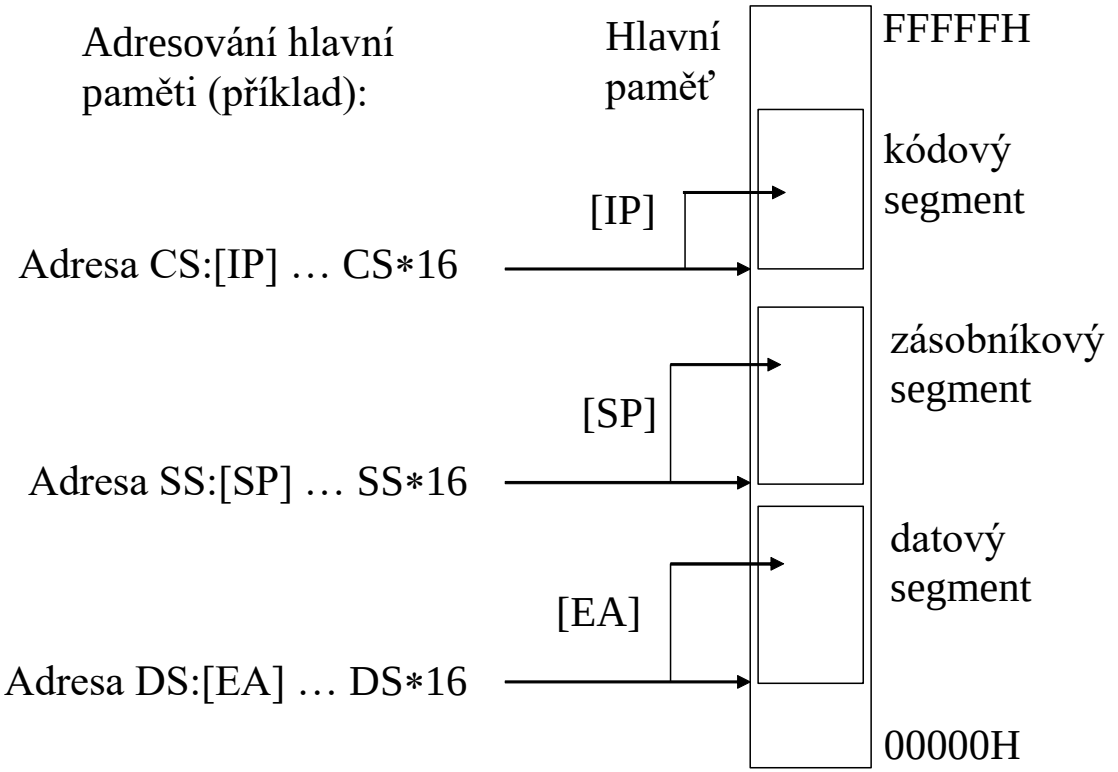
- Vynucení použití jiného segmentového registru pomocí prefixu instrukce (*override segment prefix*):
  - například CS = prefix 0x2E, SS = prefix 0x36, DS = prefix 0x3E, atd.

Výpočet fyzické adresy (základní 16bitový režim)

bit	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
segment*16																		0	0	0	0
+ ofset (efektivní adresa)	0	0	0	0																	
= fyzická adresa																					

Maximální adresovatelný paměťový prostor:

2^{20} = 1 MiB



Příklady výpočtu fyzických adres (základní 16bitový režim)

Předpokládejme následující obsahy dále použitých registrů:

```
CS = 0x45F4, SS = 0xAB00, DS = 0x5541, ES = 0x1111
IP = 0x2211, SP = 0x1FE0, BX = 0x3344, BP = 0x2244
SI = 0x1AD5
```

Fyzická adresa instrukce: `CS:IP` = $(45F40 + 02211 = 48151)_{16}$

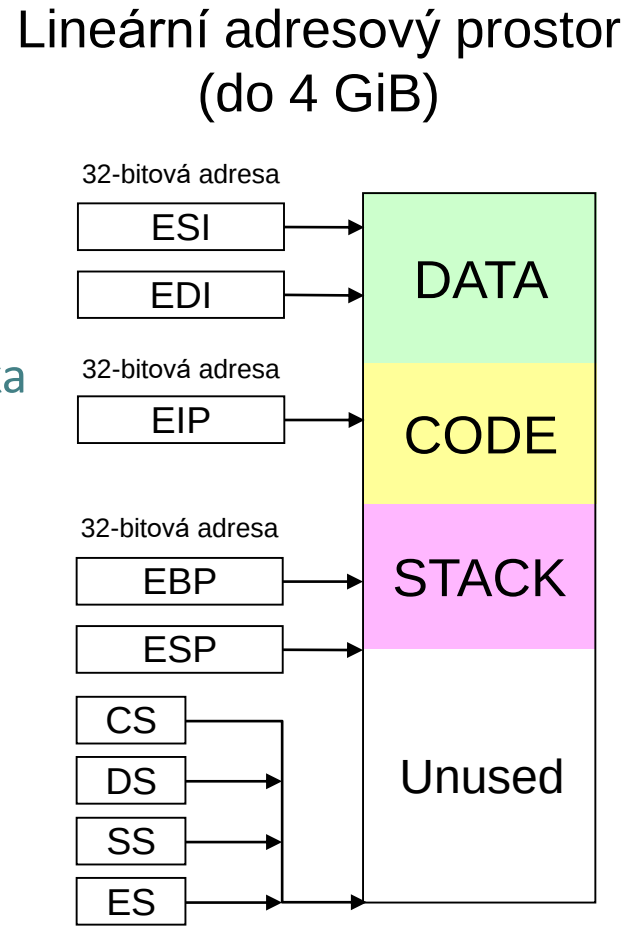
Fyzická adresa zásobníku: `SS:SP` = $(AB000 + 01FE0 = ACFE0)_{16}$

Fyzické adresy operandů:

- `[BX]` ⇒ `DS:[BX]` = $(55410 + 03344 = 58754)_{16}$
- `ES:[BX]` ⇒ `ES:[BX]` = $(11110 + 03344 = 14454)_{16}$
- `[BP+4]` ⇒ `SS:[BP+4]` = $(AB000 + 02244 + 00004 = AD248)_{16}$
- `[BX+SI+1]` ⇒ `DS:[BX+SI+1]` = $(55410 + 03344 + 01AD5 + 1 = 5A22A)_{16}$

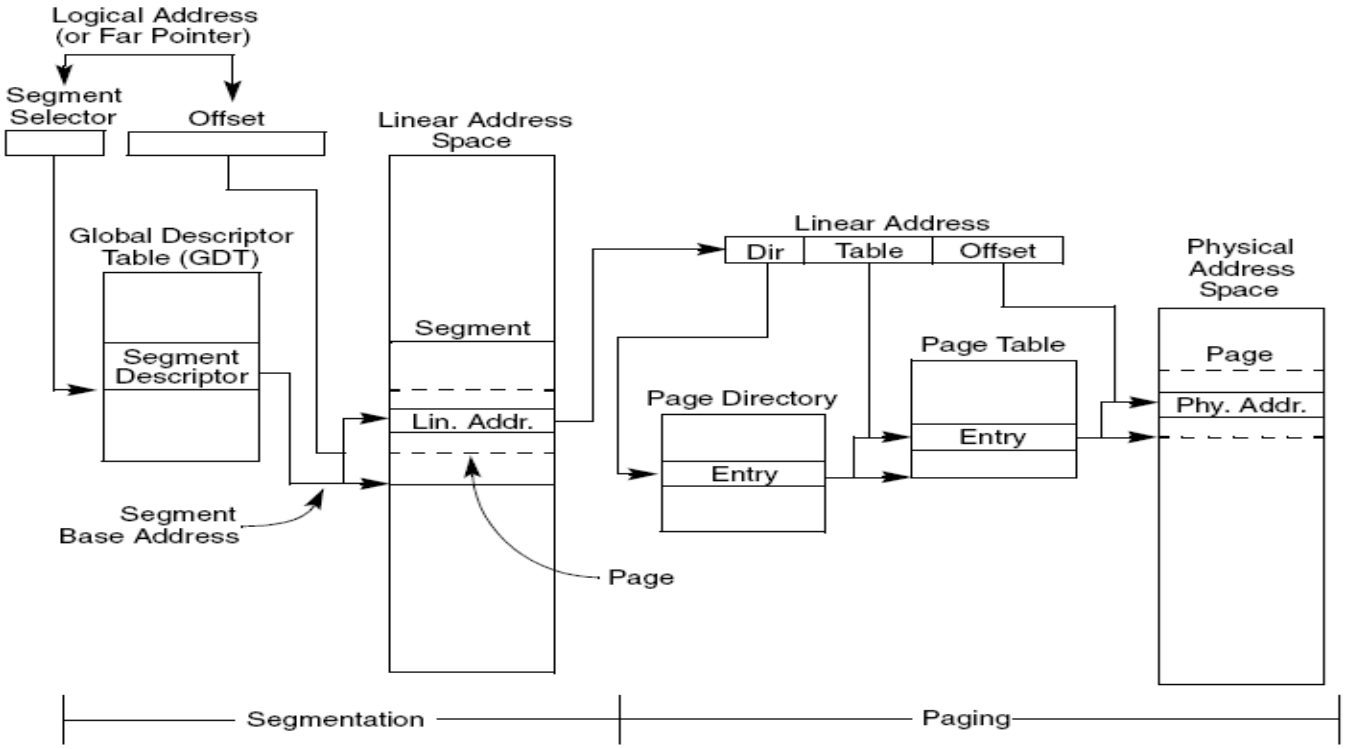
Plochý paměťový model (chráněný 32bitový režim)

- Nevyužívá se segmentace
  - = stejná segmentová adresa pro všechny segmenty
  - = stejný obsah segmentových registrů
- Program používá **lineární adresový prostor**
  - lineární adresový prostor není fyzický adresový prostor – může být virtuální (stránkování)
  - 32bitový => lze adresovat 2^{32} = 4 GiB paměti
  - **lineární adresa** = 32bitová hodnota bez znaménka = ofset = ukazatel do lineárního adresového prostoru
  - segmentové registry
    - stále 16bitové
    - spravuje operační systém
    - ukazují do tabulek deskriptorů segmentů
  - všechny segmenty jsou nastaveny do stejného lineárního adresového prostoru



Výpočet fyzické adresy (chráněný 32bitový režim)

- **Logická adresa** se skládá z
  - 16bitového **selektoru segmentu** (CS, SS, DS, ES, FS, GS)
  - 32bitového ofsetu (EIP, ESP, EBP, ESI ,EDI, EAX, EBX, ECX, EDX, disp)
- Segmentování převádí logickou adresu na lineární adresu
- Lineární adresa může být přímo fyzickou adresou (pokud je zapnuto stránkování, pak je proces složitější – ukázku znázorňuje obrázek)



Formát instrukce

Formát instrukce (základní 16bitový režim)

prefixes	opcode	ModR/M	displacement	immediate operand
0 – 4 B	1 – 2 B	0 – 1 B	0, 1 B (d8), 2 B (d16)	0 – 2 B

7	6	5	4	3	2	1	0
mod		reg/opcode			r/m ~ reg/mem		

Instrukce pracuje:

- ***bez operandů***
- s ***jedním operandem***:
  - v registru: ID registru je buď v operačním kódu instrukce, nebo v poli *r/m*
  - v paměti: efektivní adresa operandu je určena obsahy polí *mod* a *r/m* *(mem)*
- se ***dvěma operandy***:
  - v registrech: *mod* = 3 = $(11)_2$, ID registrů jsou v polích *r/m* (*reg*) a *reg/opcode*
  - v registru a paměti: ID registru je v poli *reg/opcode* a efektivní adresa operandu je určena obsahy polí *mod* a *r/m* *(mem)*

mod=11	000	001	010	011	100	101	110	111
8 b	AL	CL	DL	BL	AH	CH	DH	BH
16 b	AX	CX	DX	BX	SP	BP	SI	DI

mem	mod=00	mod=01	mod=10
000	[BX+SI]	[BX+SI+d8]	[BX+SI+d16]
001	[BX+DI]	[BX+DI+d8]	[BX+DI+d16]
010	[BP+SI]	[BP+SI+d8]	[BP+SI+d16]
011	[BP+DI]	[BP+DI+d8]	[BP+DI+d16]
100	[SI]	[SI+d8]	[SI+d16]
101	[DI]	[DI+d8]	[DI+d16]
110	[d16]	[BP+d8]	[BP+d16]
111	[BX]	[BX+d8]	[BX+d16]

Formát instrukce (32bitový režim)

prefixes	opcode	ModR/M	SIB	displacement	immediate operand
0 – 4 B	1 – 3 B	0 – 1 B	0 – 1 B	0 B 1 B (d8) 2 B (d16) 4 B (d32)	0, 1, 2, 4 B

7	6	5	4	3	2	1	0
mod		reg/opcode			r/m ~ reg/mem		

7	6	5	4	3	2	1	0
scale		index			base		

mod =11 reg	8 b	16 b	32 b
000	AL	AX	EAX
001	CL	CX	ECX
010	DL	DX	EDX
011	BL	BX	EBX
100	AH	SP	ESP
101	CH	BP	EBP
110	DH	SI	ESI
111	BH	DI	EDI

mem	mod=00	mod=01	mod=10
000	[EAX]	[EAX+d8]	[EAX+d32]
001	[ECX]	[ECX+d8]	[ECX+d32]
010	[EDX]	[EDX+d8]	[EDX+d32]
011	[EBX]	[EBX+d8]	[EBX+d32]
100	= následuje slabika SIB		
101	[d32]	[EBP+d8]	[EBP+d32]
110	[ESI]	[ESI+d8]	[ESI+d32]
111	[EDI]	[EDI+d8]	[EDI+d32]

Instrukce pracuje podobně jako v 16bitovém režimu, rozdíl je jen v případě, že je operandem paměť.

- místo 16bitových registrů jsou používány 32bitové (v hranatých závorkách)
- pokud je mem=100, pak se k výpočtu efektivní adresy použije slabika SIB

Formát instrukce (32bitový režim)

prefixes	opcode	ModR/M	SIB	displacement	immediate operand
0 – 4 B	1 – 2 B	0 – 1 B	0 – 1 B	0 B 1 B (d8) 2 B (d16) 4 B (d32)	0, 1, 2, 4 B

7	6	5	4	3	2	1	0
mod		reg/opcode			r/m		

7	6	5	4	3	2	1	0
scale		index			base		

index	32 b
000	EAX
001	ECX
010	EDX
011	EBX
100	---
101	EBP
110	ESI
111	EDi

	$s = 2^{scale}$		
base	mod=00	mod=01	mod=10
000	[EAX+index*s]	[EAX+index*s+d8]	[EAX+index*s+d32]
001	[ECX+index*s]	[ECX+index*s+d8]	[ECX+index*s+d32]
010	[EDX+index*s]	[EDX+index*s+d8]	[EDX+index*s+d32]
011	[EBX+index*s]	[EBX+index*s+d8]	[EBX+index*s+d32]
100	[ESP+index*s]	[ESP+index*s+d8]	[ESP+index*s+d32]
101	[d32+index*s]	[EBP+index*s+d8]	[EBP+index*s+d32]
110	[ESI+index*s]	[ESI+index*s+d8]	[ESI+index*s+d32]
111	[EDI+index*s]	[EDI+index*s+d8]	[EDI+index*s+d32]

Efektivní adresa (základní, 16bitovým režim)

EA = base + index + displacement

base = {BX, BP}

index = {SI, DI}

displacement = {d8, d16}

Efektivní adresa (32bitovým režim)

EA = base + index * 2^{scale} + displacement

base = {EAX, EBX, ECX, EDX, ESP, EBP, ESI, EDI}

index = {EAX, EBX, ECX, EDX, EBP, ESI, EDI} **(chybí zde ESP!!!)**

displacement = {d8, d32}

scale = {0, 1, 2, 3}

Využití registrů pro adresování (16bitový vs. 32bitový režim)

	16bitový režim	32bitový režim
Adresování v datovém segmentu – implicitní segmentový registr DS	`BX, SI, DI`	`EAX, EBX, ECX, EDX` `ESI, EDI`
Adresování v zásobníkovém segmentu – implicitní segmentový registr SS	`BP, (SP)`	`EBP, (ESP)`
	operand `[BX]` – ANO operand `[AX]` – NE operand `[CX]` – NE	operand `[EBX]` – ANO operand `[EAX]` – ANO operand `[ECX]` – ANO v 32bitovém režimu lze použít všechny 32bitové registry

- Opakování: lze vynutit použití jiného segmentového registru pomocí prefixu instrukce.

Instrukce a operandy

Instrukce a jejich základní vlastnosti

- Instrukce zapisujeme ve tvaru:

```
INSTRUKCE
INSTRUKCE operand1
INSTRUKCE operand1,operand2
```

- V případě instrukce s jedním nebo se dvěma operandy bývá první operand cílový. Zapíšeme-li operaci, kterou instrukce provádí, jako funkci *operace*, pak platí:

```
operand1 = operace(operand1)
operand1 = operace(operand1, operand2)
```

- Operandem je buď: registr, paměť nebo konstanta (*immediate operand*)

Každá (normální) instrukce má nejvýše jeden paměťový operand.	~~`MOV [VAR1],[VAR2]`~~	`MOV [VAR1],EAX`
Pro instrukce se dvěma operandy musí souhlasit velikost operandů.	~~`CMP AH,AX`~~	`CMP AH,AL`
Pokud nelze určit velikost operandu například z názvu registru, musí se uvést explicitně.	~~`MOV [VAR1],10`~~	`MOV [VAR1],byte 10`
Cílový operand (obvykle první) musí být registr nebo místo v paměti.	~~`MOV 1234,AX`~~	`MOV AX,1234`

Adresování paměťových operandů

Typy adresování

- Přímé: `MOV AL,[adr]`
  … přečte z adresy adr slabiku a uloží ji do registru AL
- Nepřímé: `MOV BL,[EBX]`
  … přečte z adresy v EBX slabiku a uloží ji do registru BL
- Nepřímé indexované: `MOV AX,[adr + EAX*2]`
  … přečte z adresy adr+EAX*2 slovo a uloží ho do registru AX
- Nepřímé indexované s bází: `MOV EAX,[EBX + EDI*4]`
  … přečte z adresy EBX+EDI*4 dvojslovo a uloží ho do registru EAX
- Nepřímé indexované s bází a posunem: `MOV CL,[EBP + EDI + adr]`
  … přečte z adresy EBP+EDI+adr slabiku a uloží ji do registru CL

Příklady

```
MOV AL,[0x1023]
MOV [0xFF00FF00],ECX
MOV [EAX],byte 5
MOV DH,[EBX+EDI*2]
MOV word [ESP],-150
```

```
ADD BL,CL
ADD EAX,[0x10F]
ADD [EBX],CX
ADD DL,[EDI+ESI*2+1]
ADD AL,'A'
```

```
SUB AH,10
SUB [EBP-8], dword 10
SUB ECX,[ESP-12]
SUB AL,[100]
SUB [EAX+EBX],AL
```

Základní struktura programu v asembleru

```
; toto je poznámka (uvozeno středníkem) – vše za středníkem neexistuje
; vložíme soubor s funkcemi pro usnadnění práce
%include „rw32-2017.inc“

; datový segment = místo pro data (proměnné)
segment .data

  ; „ptrData“ ukazuje na hodnotu (byte) 6
  ptrData DB 6

; kódový segment – místo pro kód
segment .text

; vstupní místo programu
main:
  ; instrukce s immediate operandem
  MOV AL,5
  ; instrukce s paměťovým operandem
  ADD BL,[ptrData]
  ; instrukce s paměťovým operandem
  SUB [ptrData],CL
  RET  ; konec programu
```

```
#define KAPACITA 0x1000
#define ptrData 0x0100

static unsigned char mem[KAPACITA];
static unsigned char AL, BL, CL;

void main() {
    mem[ptrData] = 6;

    AL = 5;
    BL += mem[ptrData];
    mem[ptrData] -= CL;
    return;
}
```

```
static char Data = 6;
static void *ptrData = (void*)&Data;

void main() {
    AL = 5;
    BL += *(char *)ptrData;
    *(char *)ptrData -= CL;
    return;
}
```

Výjimky a přerušení

Výjimky a přerušení

- výjimky způsobí výjimečné situace (např. dělení nulou)
- přerušení může vyvolat vnější zařízení (např. žádá o obsluhu).

Typický průběh přerušení:

1. Vykonává se program.
2. Zařízení požaduje přerušení.
3. Dokončí se instrukce. Příznaky (FLAGS) a CS:IP se uloží na zásobník.
4. Z tabulky přerušení se přečte adresa ISR (*Interrupt Service Routine*).
5. Adresa ISR se uloží do CS:IP.
6. Provádí se ISR.
7. Po ukončení ISR se vyjme CS:IP a příznaky ze zásobníku.
8. Pokračuje původní program (od místa, ve kterém bylo jeho vykonávání přerušeno).

Výjimky a přerušení základního režimu

číslo	popis	zdroj
0	Chyba dělení	DIV, IDIV
1	Ladění	každý odkaz na paměť
2	NMI	externí nemaskovatelné přerušení
3	Breakpoint	INT 3
4	Přetečení	INTO
5	Překročení hranice	BOUND
6	Chybný op. kód	UD2, rezervovaný op. kód
7	Nedostupný koprocesor	FPU instrukce, FWAIT/WAIT
8	Dvojitá chyba	každá instrukce, která může generovat výjimku
12	Chyba zásobníkového segm.	zásobníkové instrukce
16	Chyba koprocesoru	FPU instrukce, FWAIT/Wait
32-255	Maskovatelná přerušení	INT n nebo vnější přerušení INTR

Zpracování výjimek a přerušení: