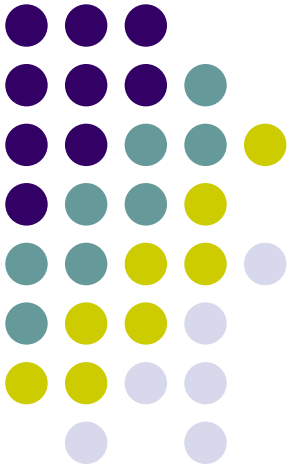
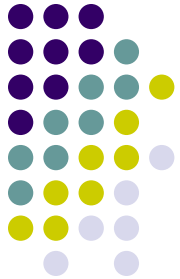
Paralelizace na úrovni instrukcí

SIMD instrukce – MMX,3DNow!, SSE

VLIW



Paralelizace



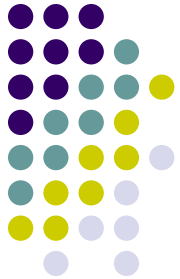
- Výpočetní výkon moderních mikroprocesorů stojí z velké části na paralelizaci
- Paralelizace = provádění více instrukcí současně

- Paralelizace
  - statická
  - dynamická

- **Statická paralelizace**
  - Pořadí a seskupení pro paralelní vykonání je **neměnné** a je přímo zapsané v programu
  - Strojový kód umožňuje zakódování instrukcí, které mají být provedeny paralelně
  - Programátor píše **paralelní program**

- **Dynamická paralelizace**
  - Seskupování instrukcí probíhá za běhu programu
  - Programátor píše **sekvenční program** a není schopen dopředu ovlivnit, které instrukce se provedou paralelně
  - Mikroprocesor si sám v programu hledá dvojice, trojice, čtveřice… instrukcí, které může vykonat paralelně
  - Algoritmus paralelizace je implementován přímo v hardware
  - Instrukce programu jsou přeskupovány tak, aby byly co nejvíc využity funkční jednotky procesoru (out-of-order)
  - Dynamická paralelizace je to, co provádí superskalární procesor

Problém paralelizace



- Paralelně proveditelné jsou pouze instrukce bez datových závislostí

- 1.Příklad

  ADD R1,R2,R3 (R1=R2+R3)
  ADD R5,R1,R4 (R5=R1+R4)

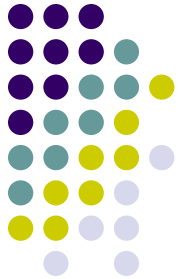
- Tyto instrukce nelze provést paralelně
- Mezi instrukcemi existuje datová závislost, registr R1 je použit v druhé instrukci jako zdrojový operand a nejprve do něj musí být uložen výsledek první instrukce

- 2.Příklad

  ADD R1,R2,R3 (R1=R2+R3)
  ADD R4,R5,R6 (R4=R5+R6)

- Toto nejsou datově závislé instrukce a lze je provést paralelně

Příklad statické paralelizace



```
load R0,100
load R1,101
load R2,102
load R3,103
add R4,R0,R2
add R5,R1,R3
div R6,R4,R5
store 104,R6
```

Sekvenční program

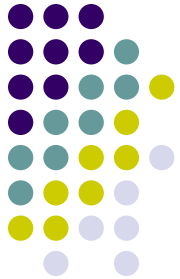
Tyto 2 instrukce zapsané **programátorem** na stejném řádku budou prováděny současně

PARALELNĚ

load R0,100	load r1,101
load R2,102	load r3,103
add R4,R0,R2	add R5,R1,R3
div R6,R4,R5	
store 104,R6	

V paralelním programu jsou ve dvou sloupcích uvedeny instrukce, které se budou provádět současně.

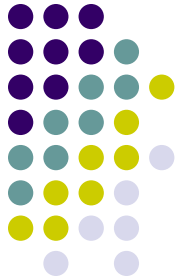
VLIW



- **VLIW** = **V**ery **L**ong **I**nstruction **W**ord
- Procesor s velmi dlouhým instrukčním slovem

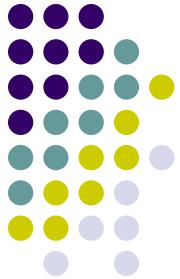
- VLIW procesory jsou typickým příkladem **statické paralelizace**
- Pracují s programem, ve kterém jsou pevně programátorem zapsané instrukce, které se mají vykonat současně
- Jedno instrukční slovo **sdružuje několik instrukcí**, které mají být provedeny naráz paralelně – proto je velmi dlouhé
- **Instrukční slovo** = slepenec instrukcí, které se provedou naráz
- Počet instrukcí sdružených v instrukčním slovu je dán počtem paralelně pracujících funkčních jednotek procesoru

VLIW



- Nevýhody
  - Složité programování – programátor musí umět psát program v paralelní podobě
  - Potřeba výkonného **kompilátoru** (při překladu z vyššího programovacího jazyka vytváří strojový kód pomocí skupin paralelních instrukcí)
  - Složitý **strojový kód**
  - Omezení vyplývající z datových závislostí při programování
  - Vysoký instrukční tok
  - Obvykle nutné realizovat více vnitřních sběrnic

Příklad VLIW procesoru

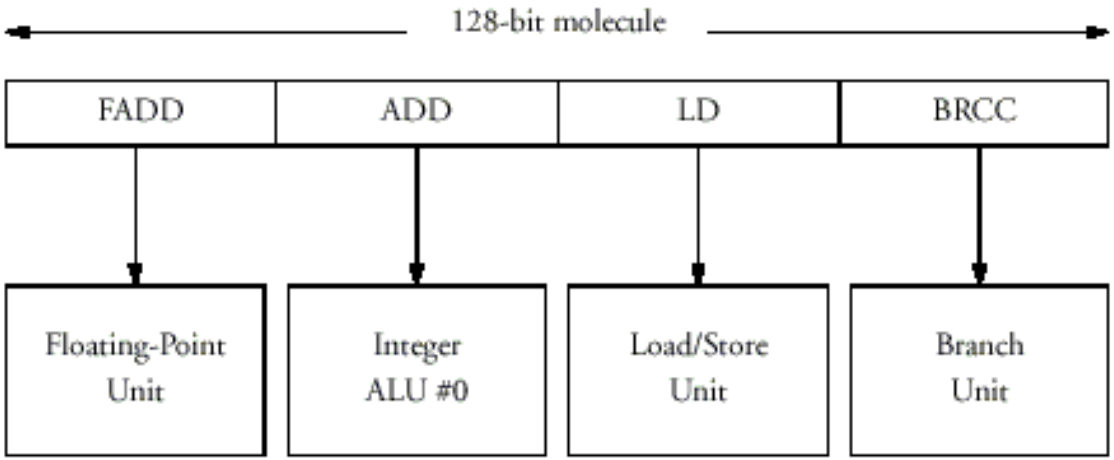


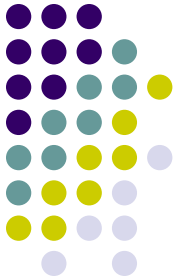
- Typickými VLIW procesory jsou digitální signálové procesory firmy Texas Instruments

- Každá instrukce je zde složena z **osmi operací**, které mají být provedeny paralelně
- **Strojový kód** jedné instrukce má šířku 256 bitů
- Jde tedy o **osm povelů** zakódovaných **32 bity**, sdružených do **256 bitů** (8x32b) dlouhého **instrukční slova**
- Někdy se také používá pojem **„instrukční paket“**
- 256 bitů velký **instrukční paket** obsahuje osm 32-bitově zakódovaných povelů pro každou z osmi paralelně pracujících jednotek procesoru

Příklad VLIW procesoru

- Dalším typickým VLIW procesorem jsou procesory **CRUSOE**
- Instrukce jsou zpracovány po tzv. **molekulách**
- Molekula má velikost **128 bitů**
- V každé **molekule** jsou uloženy **4 instrukce** (atomy)
- instrukce v molekule jsou zpracovávány **paralelně** - protože každá přísluší jiné výkonné jednotce
- Procesor obsahuje dvě jednotky pro operace s celými čísly, jednu pro čísla v plovoucí řádové čárce, dále pak jednotku pro operace s pamětí a jednotku pro zpracovávání instrukcí větvení
- Z toho vyplývá omezení, že například není možné, aby součástí molekuly by dvě instrukce výpočtu v plovoucí řádové čárce





SIMD

- **S**ingle **I**nstruction **M**ultiple **D**ata

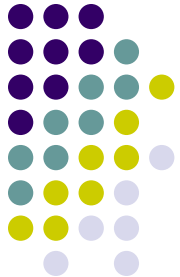
- Jedna stejná instrukce provedena paralelně s více různými daty
- Jediný povel, který je aplikován současně na více operandů

- Příklad: ***Inkrementuj*** (25, 12, 63 ,0) → Výsledek (26, 13, 64, 1)
- Stejná operace se provedla se čtyřmi čísly paralelně

- Příklad:
- ***Vynásob dvěma*** (6, 8, 2, 9, 5, 11, 3, 0)
- Výsledek = (12, 16, 4, 18, 10, 22, 6, 0)
- Stejná operace se provedla s více daty naráz

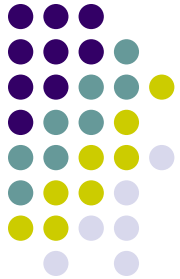
- Jedná se o statickou paralelizaci - data s kterými má být paralelně proveden výpočet si procesor nenašel sám za běhu programu, ale připravil je v programu předem programátor

MMX



- **SIMD** architektura
- 57 nových instrukcí přidáno do instrukční sady Pentia PRO
- MMX = Multimedia extension
- MMX vzniká v době, kdy byl velmi populární pojem „multimédia“ a proto bylo výhodným marketingovým tahem označit nové instrukce jako multimediální
- MMX je určeno pro aplikace s touto charakteristikou:
  - krátké celočíselné typy
  - krátké a často se opakující cykly
  - časté operace sčítání a násobení
  - provádění stejného výpočtu postupně s mnoha čísly za sebou
  - výpočetně náročné aplikace
  - paralelní výpočty

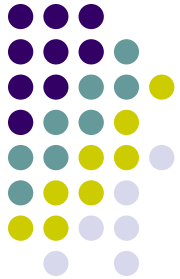
MMX



- **8 MMX** registrů o šířce 64 bitů (**MM0** až **MM7**)
- Čtyři nové datové typy
  - **packed byte** - 8 bajtů uvnitř 64-bitového MMX registru
  - **packed word** - 4 wordy (16-bitová čísla) uvnitř 64-bitového MMX registru
  - **packed dword** - dvě 32-bitová čísla uvnitř 64-bitového MMX registru
  - **qword** - registr MMX je chápán jako jedno velké 64-bitové číslo

- MMX registry jsou sdílené s FPU registry (není tedy možné, aby aplikace současně využívala FPU a MMX instrukce)

MMX – pakovaný bajt

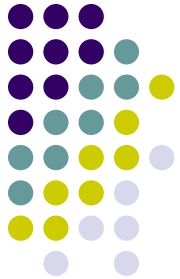


- MM2 = ABCD12347AB93D2E h

- Obsah 64-bitového registru MM2 se bere jako osm 8bitových čísel, se kterými se pracuje paralelně

- Po inkrementaci registru MM2 bychom dostali výsledek
- MM2 = ACCE13357BBA3E2F h

MMX – pakovaný WORD

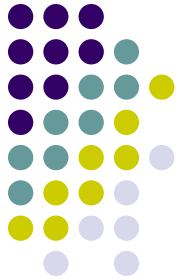


- MM2 = ABCD12347AB93D2E h

- Obsah 64-bitového registru MM2 se bere jako čtyři 16bitová čísla, se kterými se pracuje paralelně

- Po inkrementaci registru MM2 bychom dostali výsledek
- MM2 = ABCE12357ABA3D2F h

MMX – pakovaný DWORD

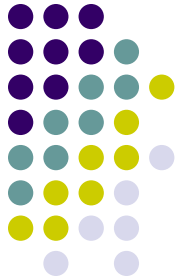


- MM2 = ABCD12347AB93D2E h

- Obsah 64-bitového registru MM2 se bere jako dvě 32bitová čísla (double word), se kterými se pracuje paralelně

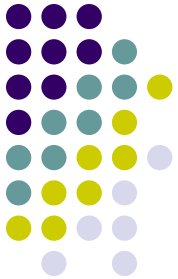
- Po inkrementaci registru MM2 bychom dostali výsledek
- MM2 = ABCD12357AB93D2F h

MMX – QWORD



- MM2 = ABCD12347AB93D2E h
- Obsah 64-bitového registru MM2 se bere jako jedno velké 64-bitové číslo
- Po inkrementaci registru MM2 bychom dostali výsledek
- MM2 = ABCD12347AB93D2F h
- Při práci s QWORD se nejedná o SIMD

MMX – pakovaný bajt, přetečení



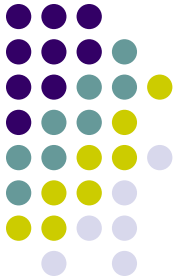
- MM2 = ABFFFF347AB93DFF h

- Obsah 64-bitového registru MM2 se bere jako osm 8bitových čísel, se kterými se pracuje paralelně

- Po inkrementaci registru MM2 bychom dostali výsledek
- MM2 = AC0000357BBA3E00 h

- Proběhlo paralelně osm nezávislých inkrementací
- Některá sčítání přetekla (FFh + 1 = 0)
- Přetečení některých součtů neovlivní sousední výsledky (nedochází k „přenosu jedničky“ do sousedních pozic)

MMX – pakovaný word, přetečení



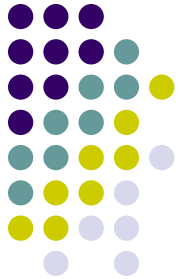
- MM2 = ABFFFF347AB93DFF h

- Obsah 64-bitového registru MM2 se bere jako čtyři 16bitová čísla, se kterými se pracuje paralelně

- Po inkrementaci registru MM2 bychom dostali výsledek
- MM2 = AC00FF357ABA3E00 h

- Proběhly 4 nezávislé inkrementace a získali jsme 4 výsledky pakované dohromady v jednom registru
- Tentokrát ani jeden z výsledků nepřetekl!
- (Přetečení by nastalo v situaci FFFFh + 1 = 0)

MMX – pakovaný word, přetečení



- MM2 = ABFFFFFF7AB9FFFF h

- Obsah 64-bitového registru MM2 se bere jako čtyři 16bitová čísla, se kterými se pracuje paralelně

- Po inkrementaci registru MM2 bychom dostali výsledek
- MM2 = AC0000007ABA0000 h

- Proběhly 4 nezávislé inkrementace a získali jsme 4 výsledky pakované dohromady v jednom registru
- Dva ze čtyř získaných výsledků přetekly
- Na rozdíl od běžné aritmetiky nedojde při přetečení v MMX výpočtu k nastavení bitu carry (CF), takže přetečení nelze nijak detekovat – dva ze čtyř výsledků nejsou platné, ale žádný příznak nás na to neupozorní

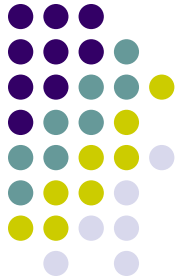
Saturace

- Všechny MMX výpočty lze volitelně provádět se saturací
- Při výpočtu se saturací **nedojde k přetečení** – hodnota čísla se zastaví na maximální nebo minimální možné hodnotě (podle toho, jestli se zvyšuje nebo snižuje)

- Pro bajt je maximální možná hodnota 255 (FF h)
- Pro word je maximální možná hodnota 65535 (FFFF h)
- Pro double word je maximální možná hodnota 4294967295 (FFFFFFFFh)

- Saturace je **nepovinná** – každá MMX instrukce existuje ve dvou variantách – se saturací a bez saturace
- Pokud programátor zvolí variantu bez saturace, musí počítat s tím, že některé výsledky mohou přetéct a nelze zjistit, které to jsou, protože jednotlivé paralelně prováděné výpočty nenastaví **Carry flag**

Saturace



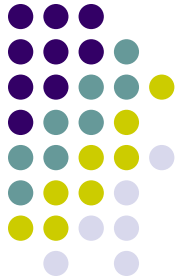
Výpočty s bajtem

Bez saturace, s přetečením	Se saturací
255+1=0	255+1=255
255+2=1	255+2=255
200+100=44	200+100=255
0-1=255	0-1=0
0-2=254	0-2=0
5-10=251	5-10=0

Výpočty s Wordem (16 b)

Bez saturace, s přetečením	Se saturací
255+1=256	255+1=256
200+100=300	200+100=300
65535+1=0	65535+1=65535
65535+2=1	65535+2=65535
50000+20000=4464	50000+20000=65535
0-1=65535	0-1=0
0-2=65534	0-2=0
5-10=65531	5-10=0

MMX – pakovaný word, výpočet se saturací



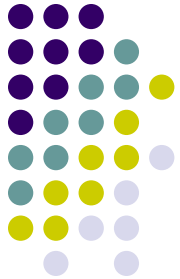
- MM2 = ABFFFFFF7AB9FFFF h

- Obsah 64-bitového registru MM2 se bere jako čtyři 16bitová čísla, se kterými se pracuje paralelně

- Po inkrementaci registru MM2 se saturací bychom dostali výsledek
- MM2 = AC00FFFF7ABAFFFF h

- Proběhly 4 nezávislé inkrementace a získali jsme 4 výsledky pakované dohromady v jednom registru
- Dva ze čtyř získaných výsledků jsou saturované
- Při provádění výpočtu se saturací nebylo možné hodnotu čísel FFFFh dále zvýšit

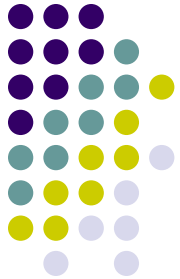
MMX



- Nové instrukce
  - aritmetické operace s pakovanými datovými typy (sčítání, násobení, odečítání, aritmetický posuv a instrukce multiply-add)
  - aritmetické operace se **saturací** (bez přetečení nebo podtečení výsledku)
  - porovnávání
  - konverze datových typů (pack, unpack)
  - přesuny mezi MMX registry

- Písmeno za instrukcí označuje datový typ
  - **PADDW** - instrukce **ADD** provedená s dvěma MMX registry, jejichž obsah bude chápán jako **packed word**
  - **PADDB** instrukce **ADD** provedená s dvěma MMX registry, jejichž obsah bude chápán jako **packed byte**

MMX Sčítání



- Sčítání pakovaných bajtů se provádí povelem **PADDB**
- Sčítání pakovaných wordů se provádí povelem **PADDW**
- Sčítání pakovaných double wordů se provádí povelem **PADDD**
- Sčítání 64-bitových qword čísel v MMX registru se provádí povelem **PADDQ**

- Zápis všech **MMX** povelů začíná písmenem **P**

- **PADDB**
- **P** = pakovaný výpočet
- **ADD** = součet
- **B** = s bajty

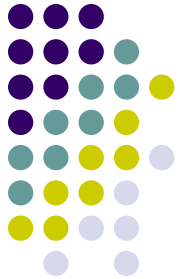
PADDW

P = pakovaný výpočet

ADD = součet

W = s wordy (16 bit)

Operace se saturací



- Pokud má být výpočet proveden se saturací, vkládá se navíc do zápisu povelu písmeno S

- **PADDSB**
- **P** = pakovaný výpočet
- **ADD** = součet
- **S** = se saturací
- **B** = s bajty

PADDSW

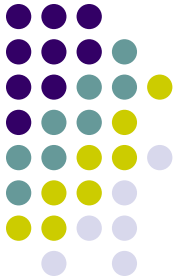
P = pakovaný výpočet

ADD = součet

S = se saturací

W = s wordy (16 bit)

Příklady MMX operací

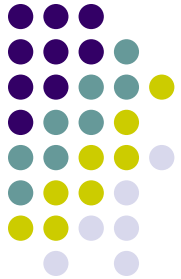


- Maximálně lze provést jednou MMX instrukcí **osm** paralelních operací, pokud se pracuje s datovým typem **packed byte**
- Instrukce PADDB provede součet osmi dvojic 8-bitových čísel (jedno sčítané číslo z dvojice vždy leží v prvním a druhé v druhém uvedeném MMX registru) a získáme 8 výsledků

PADDB MM0, MM1

MM0	a7	a6	a5	a4	a3	a2	a1	a0
MM1	b7	b6	b5	b4	b3	b2	b1	b0
výsl.	a7+b7	a6+b6	a5+b5	a4+b4	a3+b3	a2+b2	a1+b1	a0+b0

PADDB

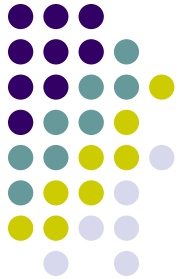


Příklad

```
PADDB MM2,MM3
MM2=12FF7E342DBCE689 h
MM3=9C7AD4B9FC271D11 h
                     h
```

- Provedlo se paralelně 8 součtů
- 89h + 11h = 9Ah
- E6h + 1Dh = 103h výsledek přetekl a uložilo se 03h
- BCh + 27h = E3h
- 2Dh + FCh = 129h výsledek přetekl a uložilo se 29h
- 34h + B9h = EDh
- 7Eh + D4h = 152h výsledek přetekl a uložilo se 52h
- FFh + 7Ah = 179h výsledek přetekl a uložilo se 79h
- 12h + 9Ch = AEh

PADDSB



Příklad

```
PADDSB MM2,MM3
MM2=12FF7E342DBCE689 h
MM3=9C7AD4B9FC271D11 h
    AEFFFFEDFFE3FF9A h
```

- Provedlo se paralelně 8 součtů **se saturací**
- 89h + 11h = 9Ah
- E6h + 1Dh = 103h výsledek přetekl a uložilo se FFh
- BCh + 27h = E3h
- 2Dh + FCh = 129h výsledek přetekl a uložilo se FFh
- 34h + B9h = EDh
- 7Eh + D4h = 152h výsledek přetekl a uložilo se FFh
- FFh + 7Ah = 179h výsledek přetekl a uložilo se FFh
- 12h + 9Ch = AEh

Příklady MMX operací

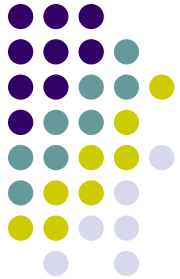
PADDW MM0, MM1

MM0	a3	a2	a1	FFFFh
MM1	b3	b2	b1	8000h
výsledek	a3+b3	a2+b2	a1+b1	7FFFh

- PADDW sečte dva MMX registry jako by v každém z nich byla uložena čtyři 16-bitová čísla
- Jde o SIMD instrukci - stejná operace (sčítání) se provádí současně na čtyřech dvojicích různých sčítanců

- **Přetečení** některého ze sčítání **není detekováno** žádným příznakovým bitem
- Přetečení kteréhokoliv z paralelních sčítání **neovlivní** výsledek jiného sčítání (například v tomto případě došlo k přetečení při sčítání dvou čísel uložených v nejnižší části MMX registrů)

Příklady MMX operací

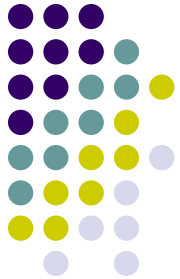


PADD**S**W MM0, MM1

MM0	a3	a2	a1	FFFFh
MM1	b3	b2	b1	8000h
výsledek	a3+b3	a2+b2	a1+b1	FFFFh

- PADDSW sečte se **saturací** dva MMX registry, jakoby v každém z nich byla uložena čtyři 16-bitová čísla
- Sčítání se saturací nemůže přetéct – maximálně může vyjít nejvyšší možný výsledek (např. FFFF+1=FFFF nebo FFFF+1234 = FFFF)

PADDW

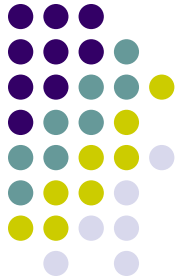


Příklad

```
PADDW MM2,MM3
MM2=12FF7E342DBCE689 h
MM3=9C7AD4B9FC271D11 h
                     h
```

- Provedly se paralelně 4 součty
- E689h + 1D11h = 1039Ah, výsledek přetekl a uložilo se 039Ah
- 2DBCh + FC27h = 129E3h, výsledek přetekl a uložilo se 29E3h
- 7E34h + D4B9h = 152EDh, výsledek přetekl a uložilo se 52EDh
- 12FFh + 9C7Ah = AF79h

PADDSW

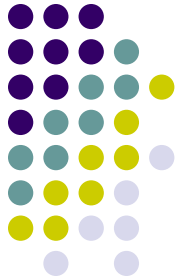


Příklad

```
PADDSW MM2,MM3
MM2=12FF7E342DBCE689 h
MM3=9C7AD4B9FC271D11 h
    AF79FFFFFFFFFFFF h
```

- Provedly se paralelně 4 součty **se saturací**
- E689h + 1D11h = 1039Ah, výsledek přetekl a uložilo se FFFFh
- 2DBCh + FC27h = 129E3h, výsledek přetekl a uložilo se FFFFh
- 7E34h + D4B9h = 152EDh, výsledek přetekl a uložilo se FFFFh
- 12FFh + 9C7Ah = AF79h

Příklady MMX operací

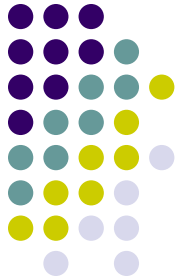


PADDSD MM0, MM1

MM0	a1	a0
MM1	b1	b0
výsledek	a1+b1	a0+b0

- PADDSDW sečte se saturací dva MMX registry jako by v každém z nich byla uložena **dvě 32-bitová čísla**

PADDD

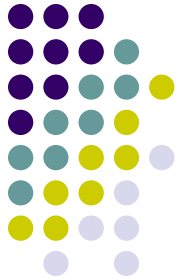


Příklad

```
PADDD MM2,MM3
MM2=12FF7E342DBCE689 h
MM3=9C7AD4B9FC271D11 h
    ---------------- ---
                     h
```

- Provedly se paralelně 2 součty
- 2DBCE689h + FC271D11h = 129E4039Ah, výsledek přetekl a uložilo se 29E4039Ah
- 12FF7E34h + 9C7AD4B9h = AF7A52EDh

PADDSD

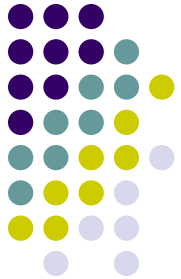


Příklad

```
PADDSD MM2,MM3
MM2=12FF7E342DBCE689 h
MM3=9C7AD4B9FC271D11 h
    AF7A52EDFFFFFFFF h
```

- Provedly se paralelně 2 součty
- 2DBCE689h + FC271D11h = 129E4039Ah, výsledek přetekl a uložilo se FFFFFFFFh
- 12FF7E34h + 9C7AD4B9h = AF7A52EDh

Příklady MMX operací

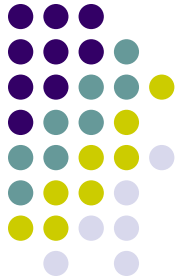


PMULB MM0,MM1

MM0	0	a3	0	a2	0	a1	0	a0
MM1	0	b3	0	b2	0	b1	0	b0
výsledek	a3*b3		a2*b2		a1*b1		a0*b0	

- MUL = multiplication
- Instrukce PMULB **vynásobí** čtyři dvojice 8-bitových čísel a čtyři 16-bitové součiny jsou uloženy jako typ packed word
- Násobením dvou 8-bitových čísel vzniká 16-bitový výsledek
- Násobení nemůže přetéct – pro výsledek je rezervováno 16 bitů
- Proto tato instrukce neexistuje ve variantě se saturací

PMULB

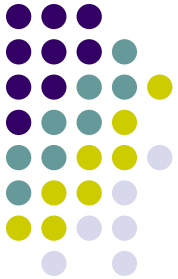


Příklad

```
PMULB MM2,MM3
MM2=00FF003400BC0089 h
MM3=00FF00B900270011 h
    FE0125941CA40919 h
```

- Provedly se paralelně 4 součiny
- FFh * FFh = FE01h (to je vlastně nejvyšší možný výsledek násobení dvou 8-bitových čísel)
- 34h * B9h = 2594h
- BCh * 27h = 1CA4h
- 89h * 11h = 919h

Příklady MMX operací

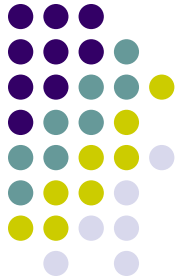


PMULW MM0,MM1

MM0	0	a1	0	a2
MM1	0	b1	0	b2
výsledek	a1*b1		a2*b2	

- Instrukce PMULW **vynásobí** dvě dvojice 16-bitových číslel dva 32-bitové součiny jsou uloženy jako typ packed dword
- Násobením dvou 16-bitových čísel vzniká 32-bitový výsledek
- Násobení nemůže přetéct – pro výsledek je rezervováno 32 bitů
- Proto tato instrukce neexistuje ve variantě se saturací

PMULW

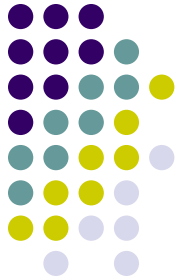


Příklad

```
PMULW MM2,MM3
MM2=0000FFFF00001234 h
MM3=0000FFFF0000ABCD h
    FFFE00010C374FA4h
```

- Provedly se paralelně 2 součiny
- FFFFh * FFFFh = FFFE0001h (to je vlastně nejvyšší možný výsledek násobení dvou 16-bitových čísel)
- 1234h * ABCDh = C374FA4h

Příklady MMX instrukcí

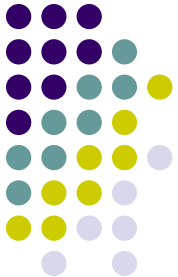


- PMADD (multiply-add)

MM0	a3	a2	a1	a0
MM1	b3	b2	b1	b0
výsledek	a3*b3+a2*b2		a1*b1+a0*b0	

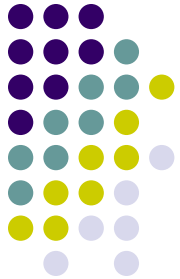
- Paralelně jsou vypočteny čtyři součiny a ty jsou pak po dvou sečteny
- Tato instrukce je vhodná pro DSP (digital signal processing) výpočty (například realizace FIR a IIR filtrů, harmonická analýza, komprese obrazu a zvuku)

3DNow!



- Technologie pro urychlení operací s čísly s **plovoucí řádovou čárkou**
- Připomeňme si, že MMX je použitelné pouze pro **celočíselné** výpočty
- odpověď firmy AMD na instrukční sadu MMX
- 3DNow! je implementováno v procesorech firmy **AMD** (počínaje AMD K6-2)
- Nejtypičtější aplikační oblast 3DNow! jsou výpočty grafických dat ve třech rozměrech (prostorová grafika) – proto je v názvu 3D
- 3DNow! Obsahuje všechny původní MMX instrukce a navíc přináší nové instrukce pro výpočty s pakovanými reálnými čísly
- 3DNow! Instrukce umí provést výpočet se dvěma reálnými čísly uloženými pakovaně v jednom MMX registru
- Instrukce pracují s původními MMX registry (ty jsou nyní použity pro uložení reálných čísel, ale nadále v nich může být i pakovaný bajt apod.)

3DNow!

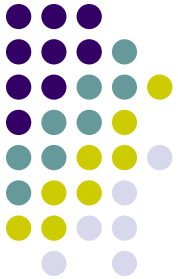


- Instrukce 3DNow! pracují s **32-bitovými FP** čísly, která jsou zakódována uvedeným způsobem

1bit	8 bitů	23 bitů
+/-	EXPONENT	MANTISA

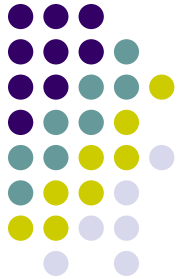
- Dvě FP čísla jsou uložena v jednom **MMX** registru

3DNow! instrukce



- Některé zajímavé instrukce
  - PFRCP – výpočet 1/x (paralelně pro obě čísla uložena v MMX registru)
  - PFRSQRT – výpočet odmocniny
  - PAVGUSB – osmibitové průměrování
  - PFADD, PFSUBB – paralelní sčítání, odčítání dvou FP čísel

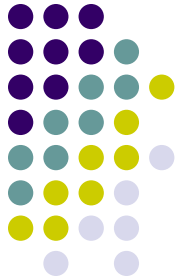
3DNow!



- Příklad

MM6 ← 4 ; 0.01	Do registru MM6 se uloží dvě reálná čísla
PFRCP MM6	Výpočet 1/x
MM6 = 0.25 ; 100	V registru MM6 jsou uloženy dva paralelně vypočítané výsledky

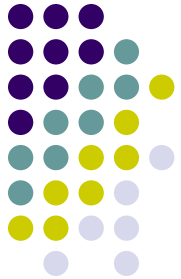
SSE



- **S**treaming **S**IMD **E**xtensions
- Zavedeno firmou Intel u procesoru **Pentium III**
- **SSE** je nástupcem technologie **MMX a 3DNow!**
- SSE registry jsou **128-bitové** a jmenují se **XMM0** až **XMM7**
- SSE přináší 70 nových instrukcí
- SSE instrukce pracují se 128-bitovými registry jako by v nich byla uložena **čtyři 32-bitová FP čísla**, ale existují i celočíselné SSE instrukce, které jsou rozšířením MMX pracujícím s typy packed byte, word, dword...

- U konkrétních typů mikroprocesorů se později seznámíme s dalším rozšířením SSE2, SSE3…

PADDB v SSE

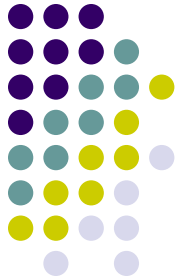


Příklad

```
PADDB XMM2,XMM3
XMM2=12FF7E342DBCE6896598AABB35120EF7 h
XMM3=9C7AD4B9FC271D112DE9C4E2B9D32AAC h
     AE7952ED29E3039A92816E9DEEE538A3 h
```

- Provedlo se paralelně 16 součtů
- F7h + ACh = 1A3h, výsledek přetekl a uloží se A3h
- 0Eh + 2Ah = 38h
- 12h + D3h = E5h
- 51h + 9Dh = Eeh
- atd…

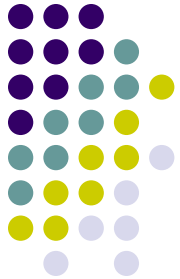
SSE



- Příklad

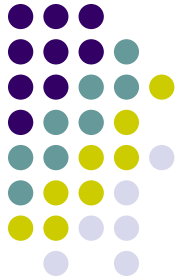
XMM6 ← 4 ; 0.01 ; 2 ; 5	Do registru XMM6 se uloží 4 reálná čísla
PFRCP XMM6	Výpočet 1/x
XMM6 = 0.25 ; 100 ; 0.5 ; 0.2	V registru XMM6 jsou uloženy čtyři paralelně vypočítané výsledky

SSE2



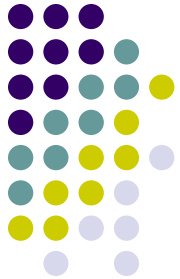
- Představil Intel v roce 2001 na novém procesoru Pentium IV
- 144 nových instrukcí
- Počet XMM registrů se rozšířil na 16 (XMM0 až XMM15)
- Nově umí pracovat i s reálnými čísly s **dvojnásobnou přesností**
- **Double precision** – reálné číslo je zakódované pomocí **64 bitů**
- Do jednoho 128-bitového XMM registru lze uložit pakovaná 4 reálná čísla s běžnou přesností nebo 2 reálná čísla s dvojnásobnou přesností

SSE3



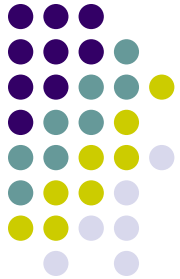
- Objevuje se v roce 2004 na procesorech Pentium 4 a Pentium D
- Nové instrukce pro počítání s pakovanými komplexními čísly (číslo má reálnou a imaginární část)
- Nové instrukce využitelné pro urychlení komprese a dekomprese videa

SSE4



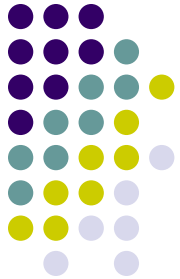
- Přichází v roce 2008 na procesorech Intel Core
- Podporuje více způsobů kódování reálných čísel
- Podporuje výpočet CRC32 - Cyklický redundantní součet (hashovací funkce)
- Nové instrukce pro spočítání počtu bitů 0 a 1 v registru (POPCNT)
- Instrukce pro porovnávání textových řetězců a hledání vzoru v řetězci
- Instrukce pro výpočet skalárního součinu – pakovaná čísla jsou chápána jako vektor

AVX



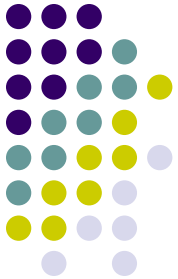
- Advanced Vector Extension
- Nástupce instrukční sady SSE
- Objevuje se poprvé na mikroprocesorech Intel Core – Sandy Bridge
- Registry se rozšířily na 256 bitů a přejmenovaly na YMM0 – YMM15
- Jako pakovaný bajt se dá do AVX registru uložit 32 čísel naráz
- Pakované číslo je vlastně z matematického hlediska vektor
- Nové instrukce umožňují pracovat s pakovanými čísly jako s vektory nebo řádky či sloupci matice
- Od roku 2013 existuje AVX-512, kde registry dále rozšířily na 512 bitů
- Do registru AVX-512 se nyní jako pakovaný bajt vejde 64 čísel naráz
- Do jednoho registru umí AVX-512 uložit 16 reálných čísel současně

Kontrolní otázky



- Vysvětlete rozdíl mezi statickou a dynamickou paralelizací – **statická - píše programátor, pořadí neměnné; dynamické – generuje se za běhu**
- Uveďte příklad datově závislých instrukcí, které nelze vykonat paralelně – **ADD R1, R2, R3; ADD R4, R1, R5**
- Vysvětlete význam zkratky VLIW – **very long instruction word – slepenec instrukcí**
- Co je instrukční paket procesoru VLIW ? - **až osm povelů zakódovaných 32 bity**
- Vysvětlete význam zkratky SIMD ? – **single instruction multiple data – jedna instrukce provede naráz operaci s více daty**
- VLIW je příkladem statické nebo dynamické paralelizace ? - **statické**
- SIMD je příkladem statické nebo dynamické paralelizace ? - **statické**
- U kterého mikroprocesoru firmy Intel se poprvé objevuje SIMD ? – **Pentium MMX**
- MMX je příkladem statické nebo dynamické paralelizace ? - **statické**
- Co je to packed byte ? **8x8 bitů uložených v 64 bitovém registru**
- Co je to packed word ? **16x4 bitů uložených v 64 bitovém registru**
- Jak se liší sčítání se saturací od běžného sčítání ? – **saturace nemůže překročit maximální hodnotu či minimální hodnotu**
- Pro jaké typy výpočtů je vhodná aritmetika se saturací ? – **například nastavení barvy – jedna hodnota nabývá rozsah 0-255, je proto žádoucí, aby byla v tomto rozsahu**
- Jakou šířku mají MMX registry a jak se jmenují ? – **64 bitů, MM0-MM7**
- Co nového přináší 3DNow! oproti MMX ? – **práci s čísly s plovoucí desetinnou čárku**
- Co nového přináší SSE oproti MMX ? – **128 bitové registry, umí pracovat s reálnými čísly**
- Jak se jmenují registry používané SSE operacemi ? – **XMM0-XMM7**

Kontrolní otázky



- MM0=11FF 00FF FF01 FFFE h
- MM1=AB23 FFFF 2734 0003 h

- Vypočítejte výsledek operací
  - PADDB MM0,MM1
  - PADDSB MM0,MM1
  - PADDW MM0,MM1
  - PADDSW MM0,MM1
  - PADDQ MM0,MM1

Výsledky: BC22FFFE2635FF01h, BCFFFFFFFF35FFFFh, BD2200FE26350001h, BD22FFFFFFFFFFFFh, BD2300FF 26360001h