Operační systémy

Rozšíření x86

Petr Krajča

Katedra informatiky
Univerzita Palackého v Olomouci

Reprezentace čísel s plovoucí řádovou čárkou (1/3)

- standard IEEE 754
- čísla zakódovaná ve tvaru

$$hodnota = (-1)^{znamenko} \times mantisa \times 2^{exponent}$$

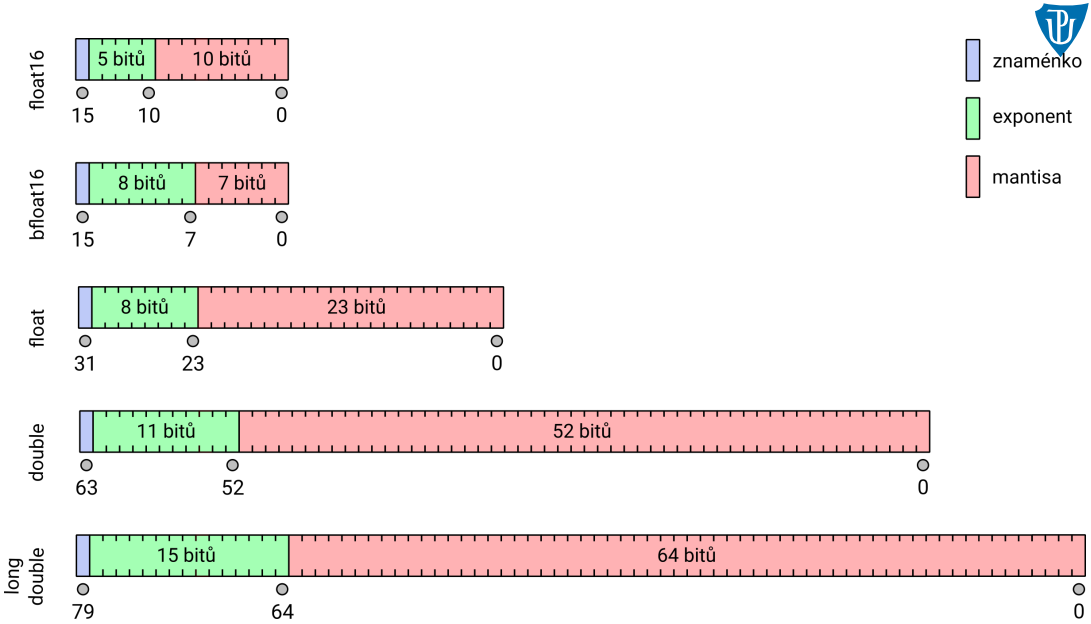
- několik variant s různou velikostí a přesností

Poznámky

- existuje záporná nula
- existují nekonečna – maximální exponent + nulová mantisa
- existuje NaN (Not a Number) – maximální exponent + nenulová mantisa

Reprezentace čísel s plovoucí řádovou čárkou (2/3)

označení	typ	velikost (bity)	exponent (bity)	mantisa (bity)
jednoduchá přesnost	float	32	8	23
dvojitá přesnost	double	64	11	52
rozšířená přesnost	long double	80	15	64
poloviční přesnost	float16, half (float)	16	5	10
brain floating point	bfloat16	16	8	7


float16
5 bitů
10 bitů
15
10
0
bfloat16
8 bitů
7 bitů
15
7
0
float
8 bitů
23 bitů
31
23
0
double
11 bitů
52 bitů
63
52
0
long double
15 bitů
64 bitů
79
64
0
znaménko
exponent
mantisa

Zásobníkové CPU

- dvě koncepce CPU – registrové vs. zásobníkové
- registrové: operandy uloženy v registrech (načtení/uložení dat z registru)
- zásobníkové
  - operandy uloženy na zásobníku
  - přidávání/odebírání hodnot přes push/load, pop/store
  - operace pracují s vrcholem zásobníku – add, sub, dup, swap
  - příklad $a^2 - 1$

```
load 1
load a
dup
mul
sub
```

  - obvykle druhý zásobník pro volání funkcí
  - výrazně jednodušší instrukční sada

Floating-Point Unit (FPU) (1/2)

- řeší výpočty s čísly s plovoucí řádovou čárkou
- pracuje s 80bitovými hodnotami (nutné převody)
- vychází z koprocesoru 80x87 (původně oddělená jednotka)
- $\Longrightarrow$ odlišná architektura + omezení
- $\Longrightarrow$ zásobníkový procesor, přenášení dat pouze přes paměť
- zásobník má kapacitu osm hodnot
- se zásobníkem jde pracovat jako s registry (označované jako ST(0)-ST(7))
- ST(0) ukazuje na vrchol zásobníku

Floating-Point Unit (FPU) (2/2)

Operace

- FLD, FST – načtení hodnot na zásobník, odebrání hodnoty ze zásobníku
- FLDZ, FLD0, FLDPI – uložení konstant na zásobník
- FADD, FSUB, . . . – numerické operace, jako jeden argument se používá vrchol zásobníku (registr ST(0)), jako druhý je možné použí kteroukoliv hodnotu ze zásobníku (registr ST(1-7)), příp. hodnotu v paměti
- větvení kódu řešeno pomocí porovnání FCOM a podmíněných přiřazení FCMOVx (FCMOVE, FCMOVB, . . .)
- další operace FSQRT, FSIN, FCOS, . . .

Volání funkcí

- při volání funkcí jsou hodnoty předávány přes zásobník
- návratová hodnota přes ST(0)

Další rozšíření

- podpora „multimédií"
- SIMD (single instruction multiple data)

MMX

- 64bitové registry mm0-mm7 (shodné s ST(0)-ST(7))
- možné používat jako vektor 1-, 2-, 4-, 8bytových celých čísel
- operace se saturací

SSE

- 128bitové registry XMM0-XMM7
- kapacita pro 4 FP hodnoty s jednoduchou přesností
- základní aritmetika

SSE2

- operace pro práci s hodnotami s dvojitou přesností (CAD)
- možnost používat hodnoty v registrech XMM0-7 jako vektory celých čísel (16 8bitových hodnot, 8 16bitových, atd.); včetně saturace

AMD64 (1/2)

- 64bitové rozšíření ISA procesorů x86 (označovaná i jako EM64T, x86_64, x64)
- rozšíření velikosti registrů na 64 bitů (rax, rdx, rcx, rbx, rsi, rdi, rsp, rbp)
- nové 64bitové registry r8-r15
  - spodních 32 bitů jako registry rXd (např. r8d)
  - spodních 16 bitů jako registry rXw (např. r8w)
  - spodních 8 bitů jako registry rXb (např. r8b)
- nové 128bitové registry xmm8-xmm15
- následně přibilo rozšiření xmm0 – xmm15 na 256 bitů (registry ymm0 – ymm15); nejnověji AVX-512 rozšiřuje tyto registry na 512 bitů a přidává další (zmm0 – zmm31)
- adekvátní rozšíření operací (prefix REX); omezení délky instrukce na 15 B
- v operacích je možné používat jako konstanty maximálně 32bitové hodnoty $\Longrightarrow$ výjimkou je operace (`movabs r, i`)

AMD64 (2/2)

- rozšíření adresního prostoru
- fyzicky adresovatelných typicky 2^{36} až 2^{46} B paměti (virtuální paměť 2^{48} B)

Režimy práce

- 64bitová ISA je velice podobná 32bitové $\Longrightarrow$ minimální režie
- **Long mode**: dva submody (ve kterých jsou k dispozici 64bitové rozšíření)
  - 64-bit mode: OS i aplikace v 64bitovém režimu
  - compatibility mode: umožňuje spouštět 32bitové aplikace v 64bitovém OS
- **Legacy mode**: režimy pro zajistění zpětné kompatibility (protected mode, real mode)
- pro výpočty s čísly s plovoucí řádovou čárkou se používají operace SSE, SSE2

Volací konvence

- větší množství registrů umožňuje efektivnější volání funkcí (podobné fastcall)
- možnost zakódovat strukturovanou hodnotu do registru
- zarovnání zásobníku na 16 B
- sjednocení volácích konvencí (v rámci platformy)

AMD64: Volací konvence – Windows

- první 4 argumenty: rcx, rdx, r8, r9
- čísla s plovoucí řádovou čárkou přes: xmm0-xmm3
- na zásobníku se vytváří stínové místo pro uložení argumentů
- zbytek přes zásobník
- návratové hodnoty přes rax nebo xmm0

```
// a -> rcx, b -> xmm1, c -> r8, d -> xmm3
void foo(int a, double b, int c, float d);
```

```
sub    rsp, 0x28              ; (0x20 + 0x08 -- kvuli zarovnani po call)
movabs rcx, <addr: msg>
call   printf
add    rsp, 0x28
```

- caller-saved: rax, rcx, rdx, r8, r9, r10, r11
- callee-saved: rbx, rbp, rdi, rsi, rsp, r12, r13, r14, r15

AMD64: Volací konvence – Windows

Pozice	Obsah	Rámec
...	...	předchozí
`rbp + 56`	6. argument	
`rbp + 48`	5. argument	
...	...	
`rbp + 16`	prostor pro uložení registrů	
`rbp + 8`	návratová adresa	aktuální
`rbp`	původní `rbp`	
`rbp - 8`	lok. proměnné a nespecifikovaná data	
...	...	
`rsp`	...	

AMD64: Volací konvence – Unix (Linux)

- prvních 6 argumentů: rdi, rsi, rdx, rcx, r8, r9
- čísla s plovoucí řádovou čárkou přes: xmm0-xmm7 (počet použitých XMM registrů musí být v registrů AL)
- zbytek přes zásobník (zprava doleva)
- návratové hodnoty přes rax nebo xmm0
- pod vrcholem zásobníku oblast 128 B (červená zóna) pro libovolné použití

```
// a -> rdi, b -> xmm0, c -> rsi, d -> xmm1; 2 -> al
void foo(int a, double b, int c, float d);
```

- caller-saved: rax, rdi, rsi, rdx, rcx, r8, r9, r10, r11
- callee-saved: rbx, rsp, rbp, r12, r13, r14, r15

AMD64: Volací konvence – Unix (Linux)

Pozice	Obsah	Rámec
`rbp + 16 + 8 *` $(n-1)$	n-tý argument na zásobníku	předchozí
...	...	
`rbp + 16`	první argument na zásobníku	
`rbp + 8`	návratová adresa	aktuální
`rbp`	původní `rbp`	
`rbp - 8`	lok. proměnné a nespecifikovaná data	
...	...	
`rsp`	...	
`rsp - 128`	červená zóna	