Operační systémy 1

Řízení výpočtu, volání podprogramů

Petr Krajča

Katedra informatiky
Univerzita Palackého v Olomouci

Intel x86: Příznaky

- jednotlivé operace nastavují hodnoty bitů v registru EF
- záleží na operaci, které příznaky nastavuje
- **příznaky pro řízení výpočtu**
  - SF (sign flag) – podle toho, jestli výsledek je nezáporný (0) nebo záporný (1)
  - ZF (zero flag) – výsledek byl nula
  - CF (carry flag) – výsledek je větší nebo menší než největší/nejmenší možné číslo
  - OF (overflow flag) – příznak přetečení znaménkové hodnoty mimo daný rozsah
- **další příznaky**
  - AF (auxiliary carry flag) – přenos ze čtvrtého do pátého bitu (BCD čísla)
  - PF (parity flag) – nastaven na jedna při sudé paritě (pouze dolních 8 bitů)
- **řídící příznaky**
  - TF (trap flag) – slouží ke krokování
  - DF (direction flag) – ovlivňuje chování instrukcí blokového přesunu
  - IOPL (I/O privilege level) – úroveň oprávnění (2 bity, pouze jádro)
  - IF (Interrupt enable flag) – možnost zablokovat některá přerušení (pouze jádro)

Intel x86: Běh programu a podmíněné skoky

- procesor zpracovává jednu instrukci za druhou (pokud není uvedeno jinak) $\Longrightarrow$ skok
- nepodmíněný skok
  - operace `JMP r/m/i` – ekvivalent GOTO (použití při implementaci smyček)
- není přítomná operace ekvivalentní `if`
- podmíněný skok je operace ve tvaru Jcc, provede skok na místo v programu, pokud jsou nastaveny příslušné příznaky
- např. `JZ i` (provede skok, pokud výsledek předchozí operace byl nula), dál `JNZ`, `JS`, `JNS`, ...

Porovnávání čísel

- srovnání čísel jako rozdíl (operace `CMP r/m, r/m/i`, je jako `SUB`, ale neprovádí přiřazení
- `JE` skok při rovnosti, `JNE`, při nerovnosti (v podstatě operace `JZ` a `JNZ`)
- a další operace

Výpočet faktoriálu: Intel x86

```
00000000 <main>:
 0:   8b 4c 24 04             mov    ecx,DWORD PTR [esp+0x4]
 4:   b8 01 00 00 00          mov    eax,0x1
 9:   83 f9 00                cmp    ecx,0x0
 c:   0f 8e 0a 00 00 00       jle    1c <main+0x1c>
12:   f7 e9                   imul   ecx
14:   83 e9 01                sub    ecx,0x1
17:   e9 ed ff ff ff          jmp    9 <main+0x9>
1c:   c3                      ret
```

Intel x86: Podmíněné skoky a porovnán

- příklad použití
- podmíněné skoky po porovnání neznaménkových hodnot

instrukce	alt. jméno	příznaky	podmínka
JA	JNBE	(CF or ZF) $= 0$	$A > B$
JAE	JNB	CF $= 0$	$A \geq B$
JB	JNAE	CF $= 1$	$A < B$
JBE	JNA	(CF or ZF) $= 1$	$A \leq B$

- podmíněné skoky po porovnání znaménkových hodnot

instrukce	alt. jméno	příznaky	podmínka
JG	JNLE	(SF $=$ OF) & ZF $= 0$	$A > B$
JGE	JNL	(SF $=$ OF)	$A \geq B$
JL	JNGE	(SF $\neq$ OF)	$A < B$
JLE	JNG	(SF $\neq$ OF) nebo ZF $= 1$	$A \leq B$

Smyčky

- pro snadnější implementaci cyklů byly zavedeny speciální operace
- `JECXZ`, `JCXZ` – provede skok, pokud registr ECX/CX je nulový (není potřeba explicitně testovat ECX)
- `LOOP` – odečte jedničku od ECX, a pokud v registru ECX není nula, provede skok

Poznámky

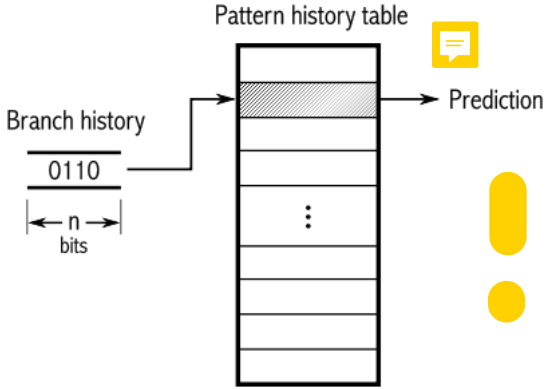
- uvádí se, že složené operace jsou pomalejší než jednotlivé kroky
- (obecně) podmíněné skoky zpomalují běh programu $\Longrightarrow$ zrušení výpočtu v pipeline
- procesory implementují různé heuristiky pro odhad, jestli daný skok bude proveden
  - statický přístup (např. u skoků zpět se předpokládá, že budou provedeny)
  - dynamický přístup (na základě historie skoků se rozhodne)
  - nápověda poskytnutá programátorem (příznak v kódu)

Odhad skoků (Branch Prediction)

- procesory používají kombinace výše zmíněných metod (hlavně dynamický odhad); různé metody
- čtyřstavové počítadlo:
- při každém průchodu procesor ukládá do Branch Prediction Buffer (2b příznak, jestli byl skok proveden, nebo ne) a postupně přechází mezi čtyřmi stavy:
  - 11 – strongly taken
  - 10 – weakly taken
  - 01 – weakly not taken
  - 00 – strongly not taken

- až na stav 00 předpokládá, že skok bude proveden
- velikost BPB a počáteční stav počítadla se mezi procesory liší
- problém: pravidelné střídání úspěšnosti $\Longrightarrow$ dvouúrovňový odhad (vzor chování)

Two-level adaptive predictor



- pro každý vzor existuje odhad založený na výše zmíněném přístupu
- velikost vzoru zavisí na procesoru
- globální vs. lokální tabulka

Zásobník

- procesor má vyčleněný úsek paměti pro zásobník (LIFO) $\Longrightarrow$ pomocné výpočty, návratové adresy, lokální proměnné, ...
- vyšší prog. jazyky obvykle neumožňují přímou manipulaci se zásobníkem (přesto má zásadní úlohu)
- procesory i386 mají jeden zásobník, který roste shora dolů
- registr ESP ukazuje na vrchol zásobníku (`mov eax, [esp]` načte hodnotu na vrcholu zásobníku)
- uložení/odebrání hodnot pomocí operací:

```
PUSH r/m/i          ;; sub esp, 4
                    ;; mov [esp], op1

POP  r/m            ;; mov op, [esp]
                    ;; add esp, 4
```

- registr ESP musí vždy obsahovat číslo, které je násobek čtyř

Operace pro volání podprogramů/funkcí

- k volání podprogramu se používá instrukce `CALL r/m/i` $\Longrightarrow$ uloží na zásobník hodnotu registru IP a provede skok

```
push eip            ;; tato operace neexistuje
jmp <addr>
```

- k návratu z funkce se používá instrukce `RET` $\Longrightarrow$ odebere hodnotu ze zásobníku a provede skok na adresu danou touto hodnotou

```
add esp, 4
jmp [esp - 4]
```

- použití zásobníku umožňuje rekurzi

Proces volání podprogramů/funkcí

- předání parametrů
- vytvoření lokálních proměnných
- provedení funkce
- odstranění informací ze zásobníku
- návrat z funkce, předání výsledku

Konvence volání funkcí (1/2)

- způsob, jakým jsou předávány argumenty funkcím, jsou jen konvence (specifické pro překladač, i když často součástí specifikace ABI OS)
- předávání pomocí registrů (dohodnou se urč. registry), příp. zbývající argumenty se uloží na zásobník
- předávání argumentů čistě přes zásobník
- kdo odstraní předané argumenty ze zásobníku? (volaná funkce nebo volající?)
- Konvence C (cdecl)
  - argumenty jsou předané čistě přes zásobník
  - zprava doleva
  - argumenty ze zásobníku odstraňuje volající
  - umožňuje funkce s proměnlivým počtem parametrů
- Konvence Pascal (pascal)
  - argumenty jsou předané čistě přes zásobník
  - zleva doprava
  - argumenty ze zásobníku odstraňuje volaný
  - neumožňuje funkce s proměnlivým počtem parametrů

Konvence volání funkcí (2/2)

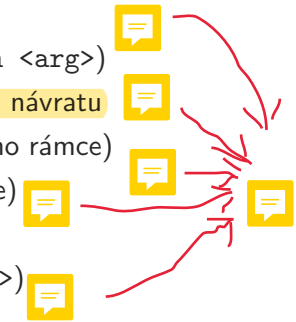
- Konvence fastcall (fastcall, msfastcall)
  - první dva parametry jsou předány pomocí ECX, EDX
  - zbylé argumenty jsou na zásobníku zprava doleva
  - argumenty ze zásobníku odstraňuje volaný
  - mírně komplikuje funkce s proměnlivým počtem parametrů
  - pod tímto jménem mohou existovat různé konvence


- návratová hodnota se na i386 obvykle předává pomocí registru EAX, příp. EDX:EAX
- větší hodnoty předávané odkazem

Rámec funkce (stack frame)

- při volání funkcí se na zásobníku vytváří tzv. rámec (stack frame)
- obsahuje předané argumenty, adresu návratu, příp. lokální proměnné
- k přístupu k tomuto rámci se používá registr EBP

Volání funkce s konvecím cdecl (1/4)



Volání funkce

1. na zásobník jsou uloženy parametry funkce zprava doleva (`push <arg>`)
2. zavolá se funkce (`call <adresa>`), na zásobník se uloží adresa návratu
3. funkce uloží obsah registru EBP na zásobník (adresa předchozího rámce)
4. funkce uloží do registru EBP obsah ESP (začátek nového rámce)
5. vytvoří se na zásobníku místo pro lokální proměnné
6. na zásobník se uloží registry, které se budou měnit (`push <reg>`)

Návrat z funkce

1. obnovíme hodnoty registrů (které byly umístěny na zásobník `pop <reg>`)
2. odstraníme lokální proměnné (lze k tomu použít obsah EBP)
3. obnovíme hodnotu EBP
4. provedeme návrat z funkce `ret`
5. odstraníme argumenty ze zásobníku (lze použít přičtení k ESP)

Obsah zásobníku

```
                    ...
                    argument n
                    ...
EBP + 12    -->     argument 2
EBP + 8     -->     argument 1
                    návratová adresa
                    původní EBP
EBP - 4     -->     první lokální proměnná
EBP - 8     -->     druhá lokální proměnná
                    ...
ESP         -->     poslední lokální proměnná
```

Volání funkce s konvecím cdecl (3/4)

Volání funkce

```
push arg2          ;; druhy argument
push arg1          ;; prvni argument
call func
add  esp, 8        ;; odstrani oba argumenty ze zasobniku
```

Tělo funkce

```
push ebp
mov  ebp, esp
sub  esp, n        ;; vytvori misto pro lokalni promenne
push ...           ;; ulozi obsah pouzivanych registru
...                ;; telo funkce
pop ...            ;; vrati hodnoty registru do puvodniho stavu
mov  esp, ebp      ;; odstrani lokalni promenne
pop  ebp
ret
```

Volání funkce s konvecím cdecl (4/4)

- první argument leží na adrese [ebp + 8], druhý na [ebp + 12], atd.
- první lokální proměnná na [ebp - 4], druhá na [ebp - 8], atd.

Uchovávání registrů

- uchovávání všech použitých registrů na začátku každé funkce nemusí být efektivní
- používá se konvence, kdy se registry dělí na
  - *callee-saved* – o uchování hodnot se stará volaný (EBX, ESI, EDI)
  - *caller-saved* – o uchování hodnot se stará volající (EAX, ECX, EDX)

- po návratu z funkce mohou registry EAX, ECX a EDX obsahovat cokoliv

Přerušení (1/2)

- mechanismus umožňující reagovat na asynchronní události
- nejčastěji vyvolané vnějším zařízením (např. stisk klávesnice, příchod síťového paketu), které vyžaduje CPU
- pokud vznikne přerušení (Interrupt Request – IRQ; testuje se po provedení instrukce), činnost procesoru je zastavena a je vyvolána *obsluha přerušení*
- po skončení obsluhy přerušení program pokračuje tam, kde byl přerušen
- obslužné rutiny – velice podobné běžným funkcím
- procesor ví, kde jsou uloženy obslužné rutiny přerušení $\Longrightarrow$ číslo přerušení $\Longrightarrow$ vektor přerušení (pole adres)
- souběh více přerušení $\Longrightarrow$ řadič přerušení
  - přerušení je možné přerušit
  - přerušení nelze přerušit (řazení přerušení)
  - systém priorit (přerušení s nižší prioritou nemůže přerušit, pokud již běží přerušení s vyšší a musí počkat)
- maskovatelné a nemaskovatelné přerušení (lze/nelze blokovat)

Přerušení (2/2)

- na x86 256 přerušení (prvních 32 speciální určení pro výjimky)
- adresa tabulky přerušení (IDT – Interrupt Descriptor Table) uložena v registru IDTR
- při přerušení se na zásobník uloží aktuální adresa (CS+EIP) + EFLAGS
- obslužná rutina obvykle ukládá i ostatní registry
- provede se obsluha přerušení
- návrat z obsluhy přerušení je realizovaný operací `IRET`

Další užití systému přerušení

- ošetření výjimek (dělení nulou, neplatná operace)
- debugování (krokování, breakpointy)
- explicitní vyvolání přerušení operace `INT` $\Longrightarrow$ systémové volání

Vektor přerušení (Linux)

Vector	Description
0	Division by zero
6	Invalid instruction
7	No coprocessor
8	Double fault
14	Page fault
32	IRQ0: Timer
33	IRQ1: Keyboard
34	IRQ2: PIC cascading
38	IRQ6: Floppy
46	IRQ14: Disk controller
128 (0x80)	System call (Linux defined)
129-238	External inputs
239	Local APIC timer interrupt
251-253	Interprocessor interrupts

I/O zařízení (1/2)

Aktivní čekání

- procesor pracuje se zařízením přímo (instrukce `in`, `out` – zápis/čtení hodnoty z portu)
- výpočetně náročné (obzvlasť přenosy velkých dat); omezené na speciální operace (jen zápis/čtení)

DMA

- řadič DMA dostane požadavek: čtení/zápis + adresu v paměti 1
- předá požadavek řadiči zařízení (např. disku) 2
- zapisuje/čte data z/do paměti 3
- dokončení je oznámeno řadiči DMA
- DMAC vyvolá přerušení 4
- př. Tan p.277

Sdílení paměťového prostoru

- zařízení mají přímý přístup k operační paměti

Režimy práce CPU

- od operačního systému očekáváme:
  - správu a sdílení procesoru (možnost spouštět více procesů současně)
  - správu paměti (procesy jsou v paměti odděleny)
  - komunikaci mezi procesy (IPC)
  - obsluhu zařízení a organizaci dat (souborový sýstém, síťové rozhraní, uživatelské rozhraní)
- není žádoucí, aby:
  - každý proces implementoval tuto funkcionalitu po svém
  - každý proces měl přístup ke všem možnostem hardwaru
- $\Longrightarrow$ jádro operačního systému $\Longrightarrow$ sdílení funkcionality, zajištění bezpečnosti/konzistence systému
- CPU různé režimy práce:
  - privilegovaný (kernel mode) – běží v něm jádro OS (umožňuje vše)
  - neprivilegovaný (user mode) – běží v něm aplikace (některé funkce jsou omezeny)
- existují i další módy, moc se nepoužívají; x86 má čtyři módy označované jako ring 0-3; (OS/2 používá tři úrovně oprávnění, VMS čtyři – kernel, executive, supervisor a user)

Systémová volání (1/2)

- přepnutí do režimu jádra přes výjimku, přerušení nebo systémové volání
- **systémové volání:** komunikace aplikace s jádrem OS pomocí přesně definovaného rozhraní
- přepnutí do režimu jádra by mělo být co nejrychlejší
- různé metody

SW přerušení

- OS má definované číslo přerušení obsluhující systémová volání (Linux: 0x80, Windows NT: 0x2e, MS-DOS: 0x21)
- je zvolen jeden registr (na i386 typicky EAX), který udává číslo požadavku (např. otevření souboru, atd.)
- ostatní registry slouží k předání argumentů (příp. se použije zásobník)
- je vyvoláno SW přerušení

Systémová volání (2/2)

Speciální instrukce

- pro zrychlení systémových volání bývají do ISA začleněny speciální instrukce
- i386: SYSENTER/SYSCALL, SYSEXIT/SYSRET

Volací brány (call gates)

- zvláštnost x86
- volá se specifická funkce, která se postará o přechod z jednoho módu do druhého
- využívá se mechanizmus spojený se segmentací
- možnost přecházet mezi různými úrovněmi oprávnění
- používaly jej Windows NT (přesun ke specializovaným instrukcím)

Historické záležitosti

Poznámka

- procesory x86 mají možnost běžet v několika režimech
- pro jednoduchost uvažujeme pouze *chráněný mód* *(protected mode)*, kde je výše zmíněná funkcionalita k dispozici
- ve starším *realném módu* není možné od sebe oddělit jádro a aplikace
- podobně i u dalších jednodušších procesorů

MS-DOS

- poskytoval své služby pod přerušením 0x21
- aplikace i OS ve stejném režimu $\Longrightarrow$ vše povoleno

BIOS

- zajišťuje základní operace počítače (rodina PC)
- obslužné rutiny BIOSu navázany na přerušení (0x10 – obrazovka, 0x13 – práce s diskem, 0x16 klávesnice)
- současné OS jej převážně ignorují