4. listopadu 2008
11:07

Principy počítačů a operačních systémů

Instrukce

Zimní semestr 2007/2008

Instrukční sada

Z pohledu vnějšího světa, tam kde se potkává programátor a návrhář

Architektura z pohledu programátora

- styčný bod mezi návrhářem počítače a programátorem
- v abstraktním smyslu programátor nemá znát fyzický návrh – obvodové řešení
- musí ale znát logický návrh, tedy např. registrový a adresový model

Všechny ty věci z pohledu programátora
Z minulých přednášek

V základu: cO není v instrukční sadě, to programátor nemůže dělat

Obvodové řešení pr programátory není zajímavé

Instrukční sada

Nutné vlastnosti

- funkční úplnost — Neznamená úplná, ale úplná natolik, aby pomocí ní šly udělat všechny funkce, co stroj musí udělat
  - musí dovolit uživateli formulovat libovolnou úlohu zpracování jeho dat na vyšší úrovni
- účinnost — Často tento požadavek není úmyslně splněn, pak to umožnuje optimalizaci
  - často opakované funkce by měly být provedeny rychle a za pomoci malého počtu instrukcí

Ne vždy HW řešení je nejrychlejší, složitější instrukce v HW budou zabírat moc místa na ukor jiných, také potřebných, proto si nemůžeme dovolit implementovat tolik složitých

- jasně definované zdroje
  - požadavky na zdroje jasně určitelné

Předpoklad, který je automatický, pokudn ebude jasně, tak by podle toho nikdo nemohl programovat

Instrukční sada

Doporučené vlastnosti

Jaká by sada měla být

- ortogonalita — Název pro mechanismus, všechny části jsou na sobě nezávislé
  - definice instrukcí, datových typů a způsobů adresování jsou nezávislé
- kompatibilita — Pokud by nový počítač měl novou sadu, tak to moc kupovatelné nebude! Pro kompatibilitu s kompilátory
  - s existujícím softwarem i hardwarem

Pokud jsou obě podmínky splněny, takse dá dobře generovat kod a dá se dbře provádět optimalizace

Jinak pořád musím hlídat, jestli to provádím správně

synt

Obsah instrukce

Každá instrukce musí obsahovat

- operační kód — Jaká operace má být vykonána
  - co se má vykonat
- odkaz a určení zdrojových operandů
  - s čím se to má vykonat — Týká se instrukcí, které mají výsledek, tak kam ho uložit
- umístění a určení výsledku
  - co se má udělat s výsledkem
- odkaz na další instrukci

Často určen implicitně, jdeme sekvenčně podle návrhu instrukční sady

U podmíněných a nepodmíněných skoků máme v adrese uloženo kam skočit

V logickém návrhu vždycky je!

Syntaxe instrukční sady - nutná součást návrhu
Určuje částti ze kterých je istrukce složena
Operační kod (instrukční kód)
Argumenty
Pro dekodér musí být co nejjednodušší -
pokud dlouho dékódujeme, zbytečně ztrácíme čas
Někdy operační kód určuje brány, které mají být oevřeny
Nekdy to hledá až dekodér z tabulky podle výsledku operačního kódu
Některé architektury mají pevnou velikost záznamu instrukce, jiné mají různou délku, podle toho co instrukce dělá
Jednotný formát je jednodušší pro dekodér,vždy načetl pevnou délku a tu zpracoval
Jinak načte třeba kousek, zjistí typ instrukce a podle té si pak načte zbytek

Instrukční sada podle své náročnosti používá různé velikosti pro zápis (např. 16bytová)

Modalita instrukce

Několik základních typů, zde tři
Upřesnuje tu použitou instrukci

Variabilita operace a operandů

- modalita operačního kódu
  - např. směr rotace

Instrukce říká doprava
A zde upřesníme směr
Tohle řekne o kolik bude ta rotace

- modalita operandů
  - např. velikost a umístění

S jakými daty zacházíme, jak jsou velké (bit, bajt…)

- modalita ochrany (paměti)
  - např. úroveň procesu

Sem se dají schovat všechny další modality

Např.: Proces s nedostatečným oprávněním nemůže tuto instrukci vykonat

Části operačního systému jsou chráněny

Modalita operačního kódu

Upřesnění operace

- směr a vzdálenost rotace
- způsob testování podmínky v podmíněném skoku
- směr operace load/store Jestli load nebo store
- použití operandů (paměť / registry) Jestli v paměti, nebo v registru
- úprava výsledku po provedení operace

Např. když sčítám dvě znaménková čísla, tak by měli zase vyjít znamenénková čísla
Kontrla jestli výsledek vyjde ve stejném typu

Modalita operandů

Specifikace operandů

- způsob adresace
  - přímé, nepřímé, ...
- velikost a typ operandů Sčítání jak s int tak float, ale instrukce to bude muset dělat trošku jinak
  - stejnou instrukci možno vykonat nad různými daty
- ovlivňuje přípravu instrukce Ovlivnuje zpracování celé instrukce
  - vícestupňové načítání Instrukce bude sčítat dva vektory, tak prvně první složky uložíme, pak druhé, atd...
  - vícenásobné načítání

Než instrukci vykonám, tak potřebuji všechna data

Modalita ochrany

Kontext běžícího vlákna

- přístupová práva k operandům
  - ochrana paměti Jestli se smí zacházet s danými daty v paměti
- právo vykonání instrukce
  - ochrana OS, procesu

Operační kod

Vyžadována jasná

Syntaxe instrukční sady

Formát zápisu instrukce v paměti

- nutná součást návrhu instrukční sady
- určuje části, ze kterých je instrukce složena
  - operační kód (instrukční kód)
  - argumenty
- pro dekodér musí být co nejjednodušší

Operační kód

Specifikace operace v instrukci

- vyžadována jasná definice, co má instrukce provést
- z návrhu počítače plyne, co znamenají jednotlivé části operačního kódu

... později

Dělení instrukcí (1)

Trošku historizující
V rámci těchto tříd by se dali najít období, kdy se která používali

Podle počtu operandů

- bezadresové
- jednoadresové
- dvouadresové
- tříadresové
- (čtyřadresové)

Spíš která instrukce převažuje, ne všechny musí byt toho tytéž typu

U Intelu jsou základní operace dvouadresové, ale najde se instrukce bez žádné adresy

Ale převažující je dvouadresové

Bezadresové instrukce

Plně bezadresový stroj asi ani neexistuje, nešlo by ani plně specifikovat argumenty

Bez operandů

- NOP, RET

Typicky NOP - No operation = nedělá nic, jen nějakou dobu trvá, často používaná operace, právě proto, že ni nedělá.. Používá se při časování, aby nšo trvalo co přesně chceme.. Nebo když nevíme při psaní kódu jak bude trvat, tak si vyhradíme nejdelší čas co sebere, pokud bude rychlejší, tak provedeme , a zbytek vyplníme NOPem

Operandy dány implicitně

- konkrétní operand dán operačním kódem
  - CLI, TBA
- zásobníková architektura
  - všechny operandy jsou na zásobníku A berou se ze zásobníku
  - instrukce ze zásobníku vyzvedne potřebné operandy
  - provede se požadovaná operace
  - případný výsledek je uložen zpět na zásobník

Nemusím říkat odkud data brát nebo kam je dávat, Je to jasně dané zásobníkem
Záleží na pořadí vložení do zásobníku, v něm se to I znej bere

RET = return - typická pro návrat z nějakého podprogramu.. Vezme ze ásobníku návratovou adresu a vrátí ji. Vrátí jen návrat, a ten kdo ji volal ví, kde si má co vzít
To se ale dá teoreticky řadit do sekce, kde je operand implicitně definován

Jednoadresové instrukce

Explicitní adresa jednoho operandu

Zbývající dvě adresy operandů implicitní

- druhý zdrojový operand a umístění výsledku
- obvyklé pro akumulátorovou architekturu

```
ADD x        ... Acc := Acc + x
```

Dvouadresové instrukce

Často u intel x86

Explicitní adresa dvou operandů

- jedna z adres je použita jak pro operand, tak pro výsledek
- velmi obvyklé

```
SUB A,4        ... A := A – 4
```

Říkám už, z jakym registrem pracuji

Typický pro typ architektur kdy se ačala zvětšovat pamět.. De facto akumulátorová ale s mnoha akumulátory

Kdyby byla sada plně ontogonální, umožnilo by to velkou možnost operací

Na obou operandech by mohl být registr, pamět, …

Zacházení s pamětí je náročnější než s registry.. Adresy paměti jsou větší než adresy registrů

Proto operace pamět pamět nejsou často definované, protože by se nevešly zapsat obě adresy

Tříadresové instrukce

Určeny jak oba operandy, tak umístění výsedku

Všechny operandy adresovány explicitně

- zdrojové operandy, umístění výsledku
- se zvyšováním rychlosti pamětí roste flexibilita počítače a dává velké možnosti dobrým kompilátorům
  - „vyhněte se problémům s udržováním dat v omezené sadě registrů a používejte jako velkou sadu registrů primární paměť.“
- pro větší délku instrukcí a výslednou délku instrukčního slova se nepoužívá často
  - prakticky použitelné především ve spojení s registry

[dost se líší od obrd] !

V historických strojích založených na feritových pamětích
Snaha zjednodušit zápis do méne instrukcí
Používá se doted u supervýkonných počítačů pro optimalizaci, spíš okrajové použití

Čtyřadresové už vymřeli skoro… a více adresové už téměř také

Příklad zápis výrazu pomocí různých instrukcí

Výpočet hodnoty

$Y = (A–B) \div (C+D\times E)$ Běžný lidský zápis

Y = AB - CDE*+/ … postfixový zápis záznam, znaménka nakonci

3–adresové

```
SUB   Y,A,B
MUL   T,D,E
ADD   T,T,C
DIV   Y,Y,T
```

První argument, kam uložit

T .. Pomocný registr

Nakonec do Y uložím Y / T

2–adresové

```
MOV   Y,A
SUB   Y,B
MOV   T,D
MUL   T,E
ADD   T,C
DIV   Y,T
```

Jeden z argumentů je vsledek

1) Do pracovního uložím A
2) Odečtu
3) Uložim D do T
4) .
5) .
6) .

1–adresové

```
LOAD  D
MUL   E
ADD   C
STORE Y
LOAD  A
SUB   B
DIV   Y
STORE Y
```

Jediný střadač, ve kterém se počítá, proto ho nikde neuvádím
Prohodil jsem výpočet levé a pravé strany, protože chci v registru prvně naopak, abych nemusel vícekrát načítat

0–adresové

```
PUSH  A
PUSH  B
SUB
PUSH  C
PUSH  D
PUSH  E
MUL
ADD
DIV
POP   Y
```

Pracují se zásobníkem,
Na zásobník pošlu A pak B
A nakonec je odečtu, neřkám co kam, protože to je v zásobníku
Výsledek této operace čeká stále v zásobníku
Proto poslední operace je DIV

Pak tam dám zbytek a zavolám operace v přílušném pořadí

Proto je výhodný postfixový zápis

Zápisu výrazu

Infixová notace

$(A–B) \div (C+D\times E)$

Z matematiky, zápis výrazů a znát priority operátorů

Postfixová notace

$AB–CDE\times+\div$

Závorky nepotřebujeme, priority jsou dány jasně zápisem výrazu v jakém pořadá jsou za sebou , potíž by byla rolišit unární nebo binírní mínus

- Reverzní polská notace
- Užitečná protože se snadno strojově zpracovává, když příjde číslo, uložím do zásobníku, když přijde operace, načtu čísla, potřebná pro danou operaci

Prefixová notace

$\div–AB+C\times DE$

Celej výraz je podíl rozdílu A a B a součtu C a součinu D a E

- Někdy se jí říká polská notace

Převody mezi notacemi

- pomocí gramatiky výrazu

Gramatika aritmetického výrazu

```
EXPR :=   EXPR '+' TERM
      |   EXPR '-' TERM
      |   TERM ;

TERM :=   TERM '×' FACT
      |   TERM '÷' FACT
      |   FACT ;

FACT :=   '(' EXP ')'
      |   identifier ;
```

Nebude vyžadována při zkoušce ;)

Reprezentace výrazu

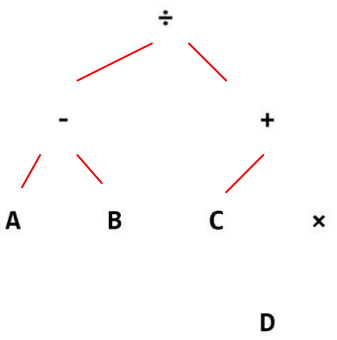
Syntaktický strom

- uzly ~ operace
- listy ~ operandy

Průchod stromem

- in-order
  (A–B) ÷ (C+D×E)
- post-order
  AB–CDE×+÷
- pre-order
  ÷–AB+C×DE

Strom výrazu



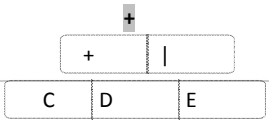
Zpracování inflixového výrazu:
Inflixovej výraz začínám od kořene,
Pak levý podstrom,
nakonec pravý podstrom
dodržuji závorkování

Postfixová prvně zpracuji levý podstrom, pak pravý a nakonec uzel

Asociativita kdyby polední bylo sčítání a né nasobení

Pokud je více stejných operací, jdeme od leva asociativně matematicky to je sice jedno, ale logicky jde jen to jedno

Tak se strom změní takto



Postfixovej zápis
CD+E+
Prefix: ++CDE

	D	E

Postfix:
CDE++

Prefi:
+C+DE

Reverzní polská notace

Zápis výpočtu pomocí zásobníku

- vstup: výraz v RPN
- výstup: číslo na vrcholu zásobníku

Zpracování výrazu

- číslo
  - uložit na zásobník
- operace
  - vyzvednout potřebné operandy
  - provést operaci
  - výsledek uložit na zásobník

Dělení instrukcí (2)

Podle typu operace

- aritmetické operace
- logické operace
- operace pro přesun dat uvnitř počítače
- operace pro vstup/výstup dat do/z počítače
- řídící operace

Aritmetické instrukce

U floatu, když nebyly běžně implementováni, tak se se přidávala jednotka, která uměla jen floaty, a měla i své vlstní registry
Dnes už jinak...

Základní výpočetní operace

- + – * /
- abs, neg, inc, dec

Negace může a nemusí být unární mínus Inc (načtu, přidám 1, uložím zpět)

Speciální aritmetické operace

- práce v plovoucí řádové čárce
- logaritmy, odmocniny, ...

Konverze, překlady

Např konverze délek, překlady podle tabulky, číslo jako index v tabulce a podle toho najdu výsledek

Logické instrukce

Booleovské operace

- AND, OR, NOT, XOR

Porovnávání, test aritmetické operace a výsledky se neukládáají

- pouze nastavení příznaků

Např. ZERO_FLAG ...

Posuny a rotace

- pozor na znaménko

Pozor na směr a znaménko.. Záporná čísla

Přesuny dat a vstup/výstup

Load/store architektury

- načíst obsah registru z paměti
- uložit obsah registru do paměti
- ALU realizuje operace pouze mezi registry
  - přesuny mezi registry
  - operace s obsahem registrů

Jiné operace

- přesuny v paměti, řetězcové operace
- vstup/výstup na periferní zařízení

Vstup/výstup
- Přesuny
- Obsluha

Řídící instrukce

Obecné instrukce

Skoky

- nepodmíněný, podmíněný
- volání podprogramu, návrat
  - vstupní parametry
  - návratová adresa

Řízení

- halt, wait, nop
- cli, sti

Execute - vezme argument, a ten vykoná jako instrukci (na virtuálním stroji)

Skip - často použito s nějakou modalitou, říkající kdy se to má přeskočit
Např. skip if bit is set …
Snadno se implementují podmíněné skoky

Určení argumentů instrukce

Implicitní

Už použitím instrukce, vim jaké parametry použít

- parametry dány použitou instrukcí

Explicitní

- součástí zápisu instrukce je odkaz na parametry
- nutno jasně definovat při návrhu instrukční sady

Jinak by výkonná jednotka nevěděla co má dělat

Způsoby adresace

- immediate – bezprostřední zápis v instrukci
- direct – v instrukci zapsána adresa operandu
- indirect – odkaz do paměti, kde je adresa oper.
- indexed – k adrese je přičten index
- based – adresa tvoří posunutí vzhledem k bázi
- relative – vzhledem k adresovému čítači

Immediate Mode

Operand je obsažen v instrukci

- data jsou za běhu kódu konstantní
- po načtení instrukce není třeba přistupovat do paměti
- velikost operandu je omezená

```
. . .
opcode   data
. . .
```

V programové paměti je část operační kód a data

Chceme-li změnit data, musíme přepsat program celý

Výhoda, přečtemeli instrukci z paměti, máme rovnou i ta data

Direct Mode

V instrukci je zapsána adresa operandu

- k vykonání instrukce je třeba navíc 1 přístup k paměti
- rozsah adres limitován velikostí instrukce
- adresa operandu je konstatní, data se mohou měnit

V paměti

Nejzákladnější způsob, podle Von neumannovy koncepce,

Abych načetl celou instrukci, tak načtu instrukci, adresu a pak teprve data, může způsobit zpoždění
Rozsah adres je dán velikostí instrukce nebo naopak rozsah instrukce dá jak jsou velké adresy

V registru je

Indirect Mode

V instrukci je adresa s adresou operandu

- k vykonání instrukce jsou třeba 2 přístupy navíc:
  - načtení adresy operandu
  - přístup k operandu

Často implementovaný, v záznamu instrukce, je adresa, kde najdu adresu operandu
Načtu instrukci, načtu adresu, a pak načíst další adresu kde jsou data a nakonec data

Výhoda je flexibilita, pokud je to šikovně navrhnuté

Indexed Mode, Base Mode

výsledná

Adresa složena ze dvou částí

- počáteční/bázová adresa
- posunutí vzhledem k počátku
- velmi podobné, někdy se nerozlišuje

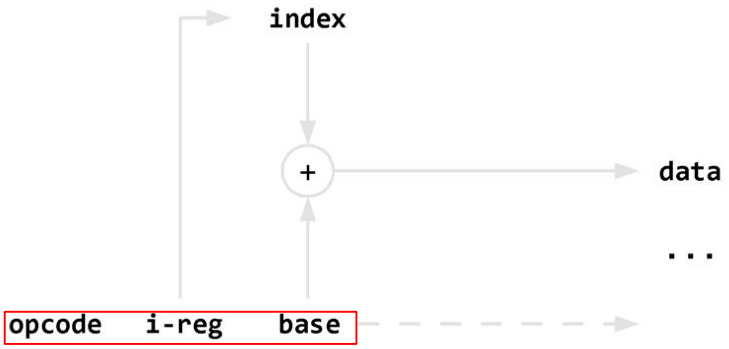
Pokud se rozlišuje...

- způsob realizace a použití
  - indexed: v programu pro přístup k datům
  - based: v OS pro implementaci ochrany/segmentace

Indexed Mode

Přístup k datům program

- báze je statická, index/offset se mění



Vůči čemu posunuju

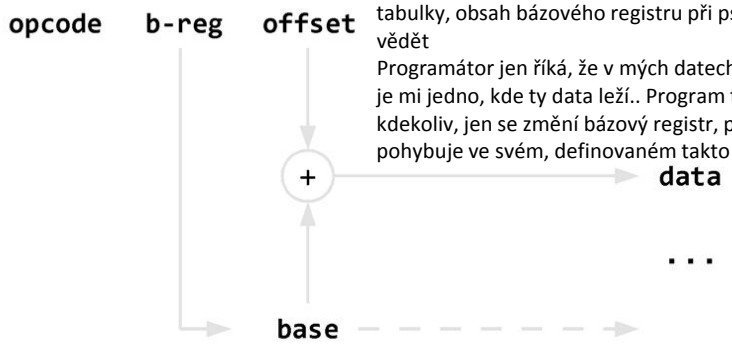
Např. Slouží k průchodu tabulkou

Indexem měním kde se v tabulce pohybuju

Based Mode

Relokace, ochrana paměti

- báze je součást kontextu, offset je statický



Posunutí a odkaz na bázový registr, který obsahuje začátek tabulky, obsah bázového registru při psaní program nemusím vědět
Programátor jen říká, že v mých datech to je takto posunuto, ale je mi jedno, kde ty data leží.. Program tedy může běžet v paměti kdekoliv, jen se změní bázový registr, proces se tedy jen pohybuje ve svém, definovaném takto

Relative Mode

Operand je určen relativně vzhledem k programovému čítači

Záznam instrukce obsahuje jen posunutí kam se mám přehodit pro dat

Např u skoků

Vzhledem k programovému čítači

- relativní posun

Adresace s použitím registrů

Adresy v instrukcích z domény registrů

- register addressing
  - jako direct, ale adresová část určuje registr s operandem
- register indirect
  - jako indirect, ale adresová část určuje registr, který obsahuje adresu operandu

Některá se zúčastněných adres je z domény registrů nikoliv hlavní paměti

Automatické změny argumentů

a. a .b nesouvisí s 1. a 2.

Např. 1. pre a. increment
2. post b. decrement

- automatický posun na další zpracovávaná data je součástí logiky instrukce
- typické pro přesuny dat a konverze
  - Intel IA-32: movs, lods, stos, ins, outs, ...
- typické v kombinaci s nepřímým adresováním

Je třeba už rozlišovat na co provedu inkrement, jestli na adresu nebo až předadresovaná data a kdy, jestli pre nebo post

Načtu data zprauju a automaticky se potom navýší

Kombinované režimy

Kombinace různých způsobů

- base + index * scale + displacement
  - base, index ... registry
  - scale ... { 1, 2, 4, 8 }
  - displacement ... konstanta v instrukci
- Intel IA-32
  - segment + base + index * scale + displacement

Můžu při jednom adresování kombinobat

Literatura

W. Stallings

- Computer Organization & Architecture

A. Tanenbaum

- Structured Computer Organization

J. Bayer et al.

- Počítače pro řízení

N. J. Davis

- Computer Organisation