Základy počítačových architektur

INP 2019

FIT VUT v Brně

Obsah

- Typy architektur instrukčních souborů
- Datové objekty
- Adresové módy
- Kódování instrukcí
- CISC vs. RISC

Terminologie

- ISA - Instruction Set Architecture
  - abstraktní model počítače (z pohledu programátora)
  - může být implementována různě – výsledné implementace se mohou lišit ve výkonnosti, ceně, příkonu apod.
  - protože jsou všechny implementace funkčně shodné, je garantována binární kompatibilita (stejná binárka běží na procesorech různých výrobců)
  - ISA definuje
    - datové typy
    - registry, paměť (vč. způsobů adresace, paměťové konzistence atd.)
    - Instrukční soubor
    - I/O model
- Mikroarchitektura (někdy nazývána computer organization)
  - obvodový způsob, kterým je implementována ISA v procesoru
  - procesory s odlišnou mikroarchitekturou mohou implementovat prakticky stejnou sadu instrukcí
    - Např. Intel Pentium i AMD Athlon realizují x86
    - Denver Nvidia, AMD K12, Apple Cyclone, Qualcomm Kryo, Samsung M1/2 realizují ARMv8
- CPU (Central Processing Unit) ~ procesor
  - Integrovaný obvod, který obsahuje ALU, datovou cestu, registry, řadič, rozhraní pro přístup do paměti a periférií a další komponenty nutné pro zpracování instrukcí.

Typy instrukcí

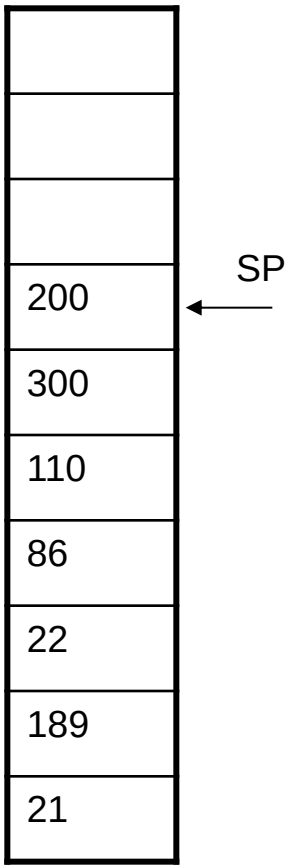
- Přesuny dat, přístup do paměti, k zásobníku a k I/O
  - R-R, R-M, M-M, vložení konstanty do R, …
- Aritmetické a logické instrukce
  - aritmetické a logické operace, posuvy, rotace, porovnávání …
- Řízení toku programu
  - ne/podmíněné skoky, volání/návrat z podprogramu, …
- Ostatní
  - Práce s koprocesorem
  - Složité instrukce
    - přesuny bloků dat, složité aritmetické operace (goniometrické funkce, odmocnina atd.), SIMD, atomické instrukce apod.

Typy architektur instrukčních souborů

- Zásobníková (stack)
- Střadačová (accumulator)
- Registrová (GPR – General Purpose Register)
- Smíšená
  - kombinace předchozích
  - dlouhodobě nejosvědčenější koncepce

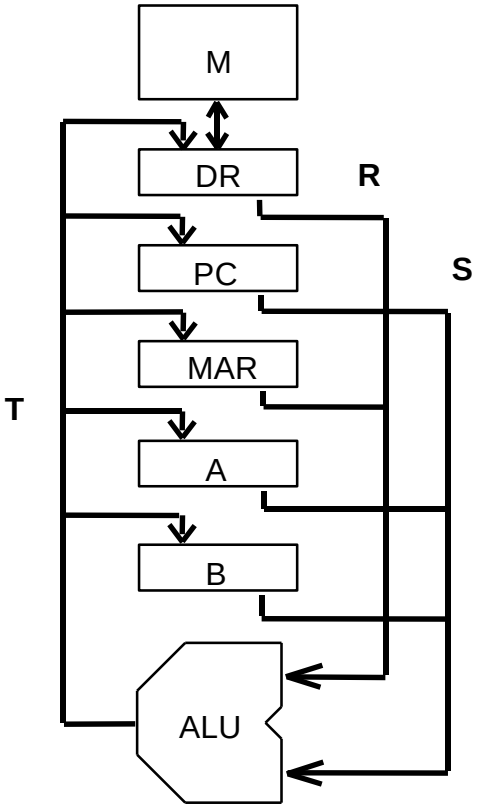
Zásobníkové počítače

- Programátor nemá k dispozici střadač ani registry – veškeré operace se provádějí se zásobníkem.
- Zásobník je implementován buď v paměti nebo pomocí registrů nebo kombinací registrů a paměti.
- Program a data mají v paměti oddělené segmenty, jejichž meze určují specializované registry.
- Stack pointer (SP)– ukazatel na vrchol zásobníku (nejdůležitější registr)
- Typické instrukce
  - PUSH m_a: přečti slovo z paměti s adresou m_a a ulož ho na vrchol zásobníku
  - POP m_a: vyber slovo z vrcholu zásobníku a ulož ho do paměti na adresu m_a
- Aritmetické instrukce (ADD, MUL apod.) nemají operandy.
  - ADD: vybere z vrcholu zásobníku dvě slova, sečte je a výsledek uloží na vrchol zásobníku
- Výhody: jednoduchá architektura, krátký kód, nízký příkon
- Nevýhody: zásobník je úzké místo (lze pracovat jen s jeho vrcholem, obtížná paralelizace výpočtu), nízká výkonnost
- Příklady
  - Minipočítač HP 3000 od r. 1972 (máme v muzeu na FIT)
  - Procesor ZPU - zabírá jen asi 80% velikosti ARM Thumb2 (jeden z nejmenších procesorů s registrovou architekturou)



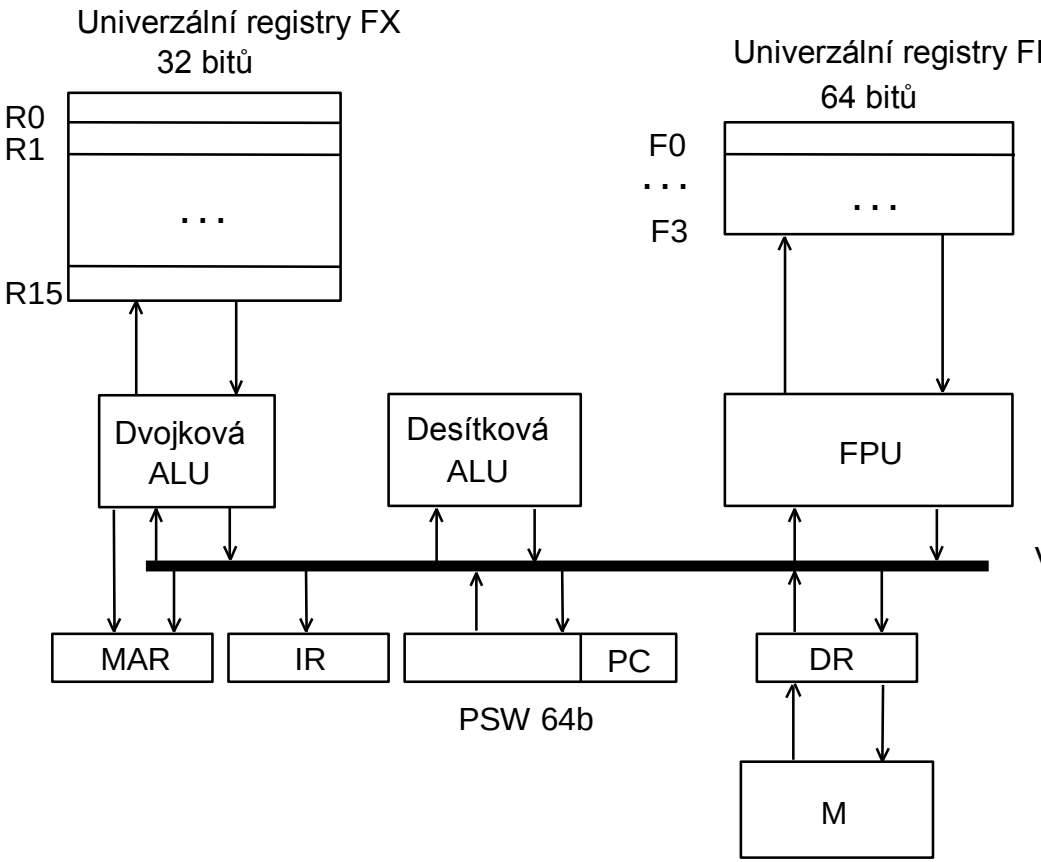
Střadačové počítače

- Existuje jeden, popř. dva, významné registry – střadače – které využívají prakticky všechny instrukce.
- Typická aritmetická instrukce má jeden operand implicitní (střadač) a výsledek je uložen opět do střadače.
  - Př. ADA a_m: k obsahu střadače A je přičten obsah paměťové buňky na adrese a_m, výsledek je uložen do střadače
- Data čtená z, popř. zapisovaná do, paměti směřují vždy do/z střadače.
  - Př. LDB a_m: Do střadače B je vložen obsah paměťové buňky na adrese a_m
- Nevýhody: střadač je úzké místo; často se komunikuje s pamětí



- Pozn.: DR - datový registr, přes který přecházejí instrukce a data čtená nebo zapisovaná do paměti. MAR - registr adresy paměti, obdoba IAR z první přednášky.

Registrové počítače



Př. IBM 360/370

- Programátor má k dispozici sadu registrů.
- Př. IBM 360/370 (1965)
  - 3 speciální ALU
    - FX
    - FP
    - Operace s proměnnou délkou, zahrnující desítkovou aritmetiku a operace s řetězci znaků.
  - dvě sady nezávisle adresovatelných registrů
    - 16 FX registrů
    - 4 FP registry
  - 64b registr pro Program Status Word (PSW) – popisuje stav procesoru (příznaky, PC, maska přerušení…)

Srovnání v úloze M[C] = M[A] + M[B]
(A, B i C jsou adresy)

- Zásobníkový stroj
- Střadačový stroj
- Registrové stroje:
  - registr – paměť (R-M): Memory Reference (v libovolné instrukci)
  - registr – registr (Load-Store): přístup k paměti pouze v instrukcích Load a Store
  - memory – memory (už jen historie – LGP, kalkulačka M3T)

Zásobníkový	Střadačový	Registrový (R-M)	Registrový (L-S)
`Push A`	`Load A`	`Load R1, A`	`Load R1, A`
`Push B`	`Add B`	`Add R1, B`	`Load R2, B`
`Add`	`Store C`	`Store C, R1`	`Add R3, R1, R2`
`Pop C`			`Store C, R3`

Výhody a nevýhody základních architektur

- Při hodnocení se používá těchto tří kritérií:
  - jak vyhovuje struktura potřebám kompilátoru,
  - jak vhodná je struktura z hlediska implementace,
  - jak dlouhý vyjde program ve srovnání s ostatními koncepcemi.

Architektura	Výhody	Nevýhody
Zásobníková	Jednoduché vyčíslování výrazů (polská notace), díky krátkým instrukcím je výsledný strojový kód hustý.	Přístup k zásobníku není libovolný, proto je obtížné vytvořit efektivní kód. Zásobník je úzké místo architektury.
Střadačová	Minimalizuje se počet vnitřních stavů počítače, instrukce jsou krátké.	Střadač je pouze dočasná paměť, zatížení paměti M je vysoké.
Registrová	Nejobecnější model pro generování kódu (tzv. tříadresový kód).	Všechny operandy musejí být pojmenovány, což vede na dlouhé instrukce.

Architektury s univerzálními registry

- Vyčíslování výrazů je pružnější než při použití střadačů nebo zásobníku.
  - Např. vyčíslování výrazu (A x B) - (C x D) - (E x F) může částečně proběhnout v libovolném pořadí, kdežto u zásobníkové koncepce musí proběhnout zleva doprava.
- Jsou-li proměnné umístěny v registrech, provoz paměti se sníží a program se provede rychleji.
  - Viz sečtení pole čísel na procesoru z 1. přednášky.
- Otázkou je, kolik (nespecializovaných) registrů by mělo v procesoru být.
- Moderní procesory zásadně nekombinují aritmetické/logické instrukce a práci s pamětí
  - Specializované instrukce pro čtení/zápis (LOAD mem, reg; STORE mem reg.)
  - Aritmeticko-logické operace jen s registry: $R_1 \leftarrow R_2$ op R_3

Datové objekty

- Podle délky rozlišujeme:
  - byte B 1B
  - půlslovo HW (Halfword) 2B
  - slovo W (Word) 4B
  - dvojslovo DW (Doubleword) 8B
  - čtyřslovo QW (Quadword) 16B

Uspořádání bytů ve slově

- Adresa slova **Little Endian**

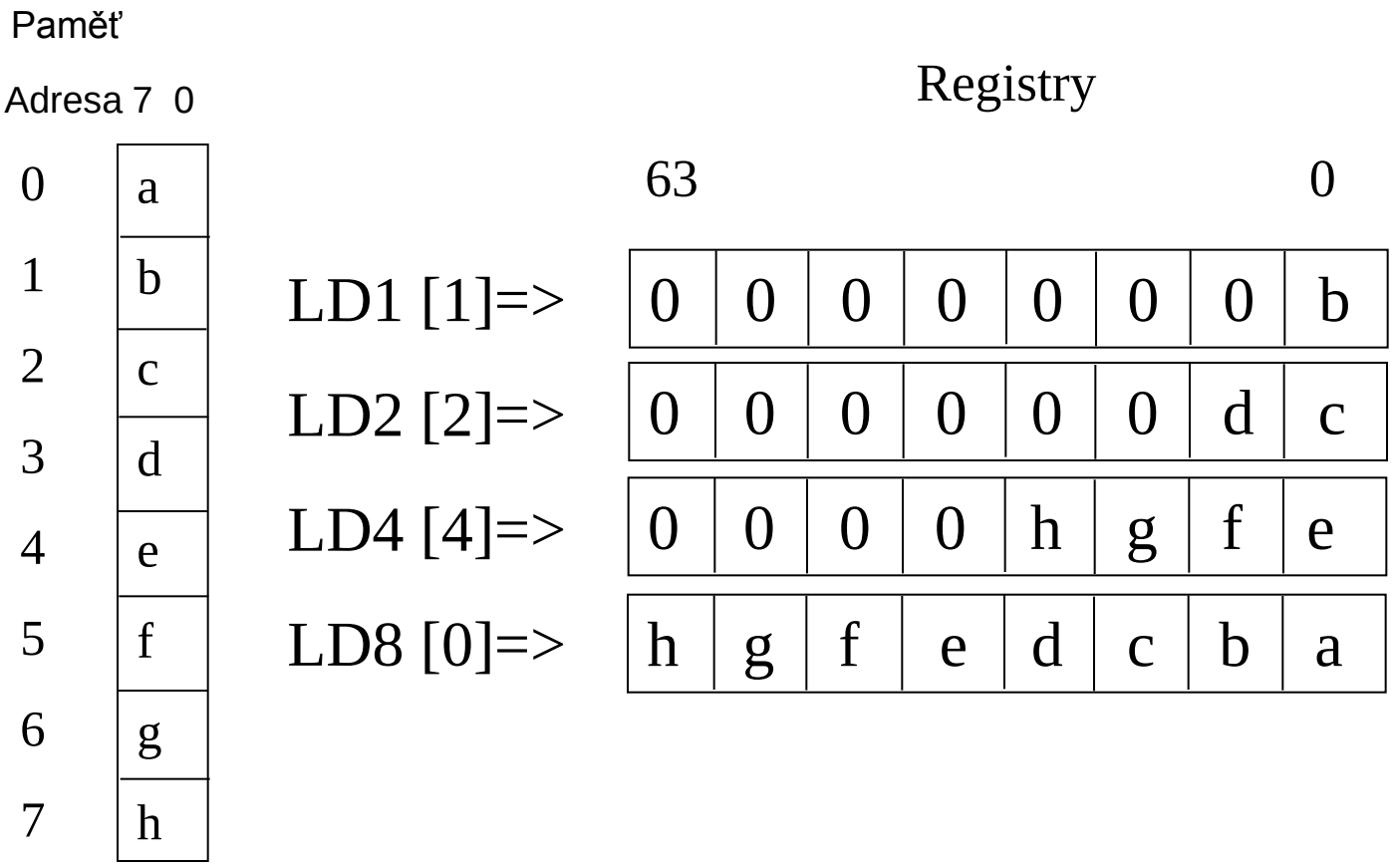
0	0	1	2	3
4	4	5	6	7

- Adresa slova **Big Endian**

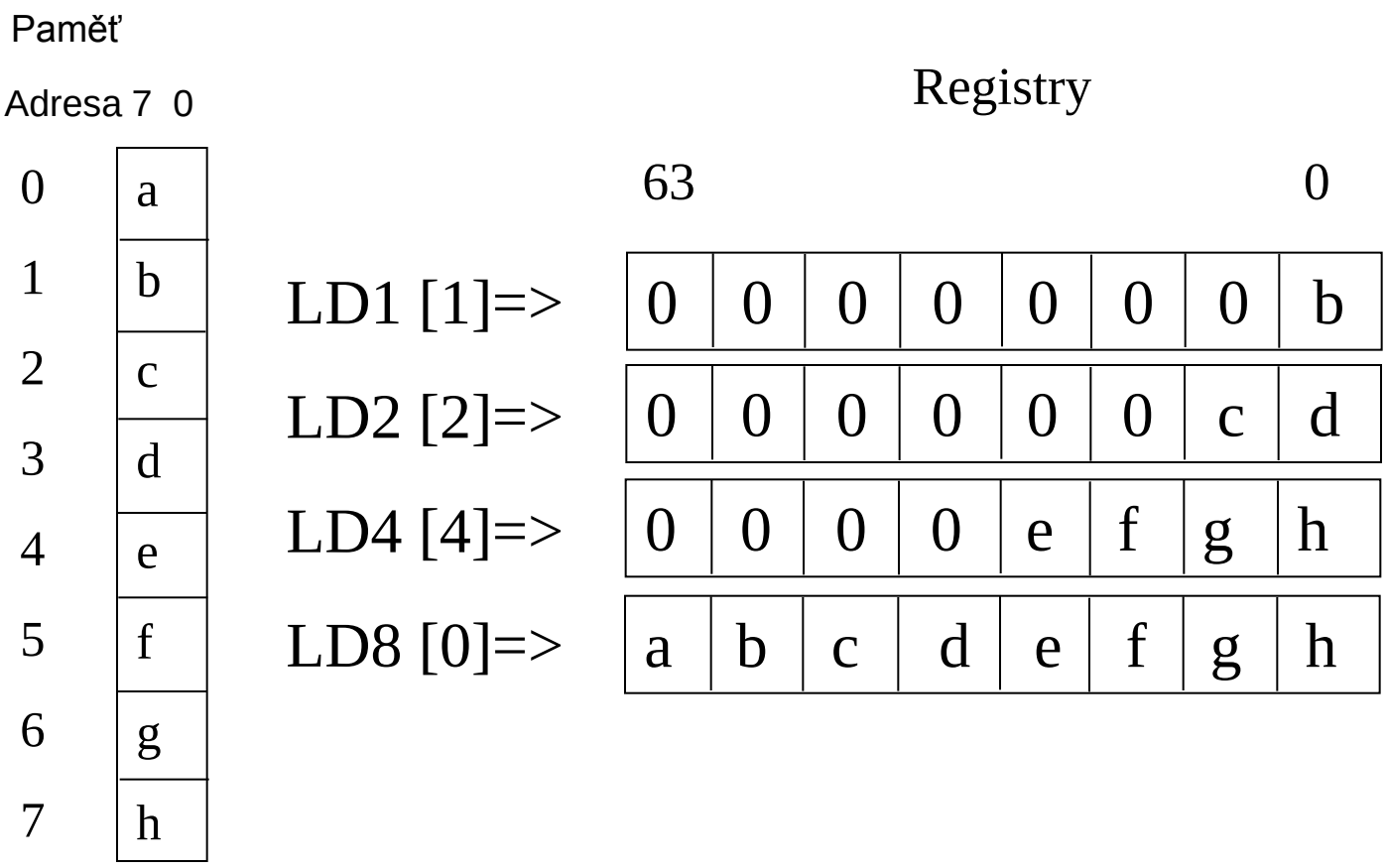
0	7	6	5	4
4	3	2	1	0

- Příklady:
  - Little Endian: DEC PDP/11, VAX, Intel 80x86, Atmel AVR, 8051
  - Big Endian: IBM 360/370, Motorola 68000
- Bi-Endian (Mixed Endian, Endianness) – podpora obou způsobů
  - ARM versions 3+, PowerPC, Alpha, SPARC V9, IA-64

Př. IA-64 Instrukce **Load** Little-endians (Intel Itanium, 64b)

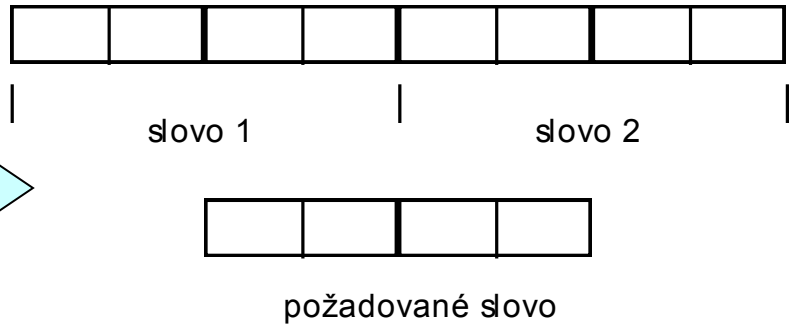


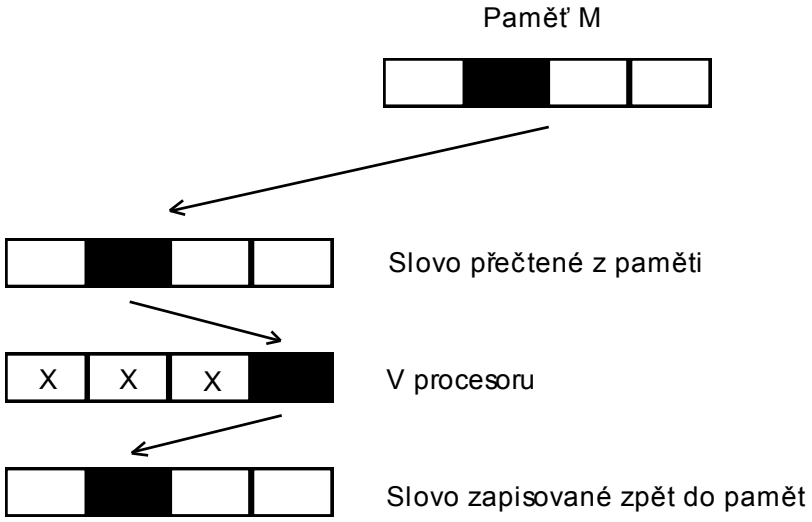
Př. IA-64 Instrukce **Load** Big-endians (Intel Itanium, 64b)



Nezarovnaný přístup k paměti

- U některých počítačů musí být přístup k objektům větším než jeden byte tzv. zarovnán.
- Je zřejmé, že na získání nezarovnaného slova musí paměť provést dva cykly a z každého slova se bere jen jedna polovina.
- I když jsou data zarovnaná, podpora bytových a půlslovních přístupů vyžaduje zarovnávací obvod, který požadovaný byte nebo půlslovo zarovnává v registrech.
- Taková potřeba se objevila u procesorů, které z důvodů kompatibility zachovávají instrukční soubor a jeho operace, např. u 32-bitového procesoru se zachovávají bytové operace. Při práci s jedním bytem se nesmí ostatní tři byty slova změnit .





Bytová operce na 32b počítači.

Adresové módy (1)

- U registrových strojů může adresový mód určovat konstantu, registr, nebo paměťové místo. Používá-li se paměťové místo, nazývá se skutečná adresa paměti specifikovaná adresovým módem efektivní adresa. V tabulce je přehled adresových módů, které se vyskytují u hlavních typů počítačů.

Adresový mód	Příklad instrukce	Význam	Kdy se použije
Registr	Add R4,R3	R4:=R4 + R3	Když je hodnota v registru.
Bezprostřední, literál	Add R4,#3	R4:=R4 + 3	Pro konstanty, u některých procesorů jde o dva různé adresové módy.
Bázový, s posunem	Add R4,100(R1)	R4:=R4+M[100+R1]	Adresování lokálních proměnných.
Přes registr, nepřímý	Add R4,(R1)	R4:=R4+M[R1]	Přístup s ukazatelem nebo s výpočtem adresy.

Adresové módy (2)

Indexovaný	Add R3,(R1+R2)	R3:=R3+M[R1+R2]	Při adresování v poli: R1 je báze pole, R2 je index.
Přímý, absolutní	ADD R1,(1001)	R1:=R1+M[1001]	Pro přístup k pevně umístěným datům; konstanta může být značně velká.
Nepřímý přes paměť	Add R1,@(R3)	R1:=R1+M[M[R3]]	V R3 je adresa ukazatele.
Autoinkrement	Add R1,(R2)+	R1:=R1+M[R2] R2:=R2 + d	Užitečný pro průchod polem v cyklu. R2 ukazuje na začátek pole a při každém použití se zvětšuje o d.
Autodekrement	Add R1,-(R2)	R2:=R2 - d R1:=R1+M[R2]	Obdoba autoinkrementu.
Indexovaný s měřítkem	Add R1,100(R2)[R3]	R1:= R1+M[100+R2+R3*d]	K indexování pole. Velikost kroku je proměnná.

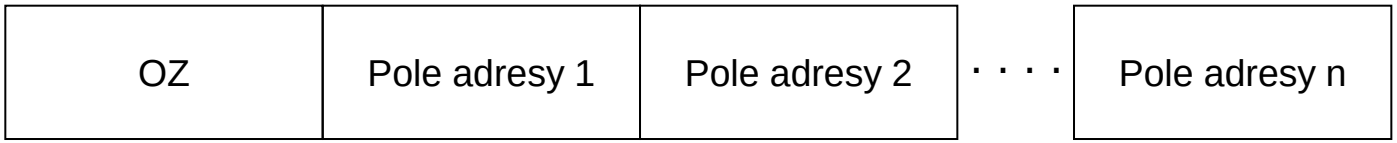
Adresové prostory

- Velikost adresového prostoru je daná vyhrazeným počtem bitů v adrese.
- Rozlišujeme
  - paměťový adresový prostor
  - V/V adresový prostor – určuje možné adresy připojených periferních zařízení
  - další adresové prostory
    - adresový řídicí prostor (přerušovací) – pro stavové informace počítače
    - registrový
    - zásobníkový
- Zopakujte si pojmy z Operačních systémů:
  - logický adresový prostor
  - fyzický adresový prostor
  - virtuální paměť
  - stránkování
  - segmentování
  - překlad adresy
  - stránka a rámec
  - TLB

Kódování instrukcí

- definuje způsob zakódování operačních znaků (OZ; opcode)
- předepisuje velikost jednotlivých polí pro adresy registrů a pro adresy paměti
- Pozor na zarovnaný přístup do paměti!

- Obecný tvar instrukce:



Kódování instrukcí – délka instrukce

- Pevná délka instrukce
  - Neúsporné, ale jednoduché
- Proměnná délka instrukce (nejčastější)
  - Obtížnější dekódování, ale dovoluje zkrátit program
  - Typické formáty instrukce s proměnnou délkou
    - OZ např. NOP, HALT
    - OZ + adresa např. LOAD adr (střadač je implicitní)
    - OZ + 2 adresy např. LOAD R1, adr
    - OZ + 3 adresy např. ADD R1, R2, R3
- Rozšiřující se kód – příklad pro 16-bit. instrukce

```
0000 a1   a2   a3
…
1110 a1   a2   a3
```
15 instrukcí s 3 adresami

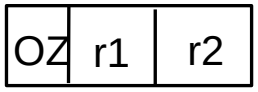
```
1111 0000 a1   a2
…
1111 1101 a1   a2
```
14 instrukcí s 2 adresami

```
1111 1110 0000 a1
…
1111 1111 1110 a1
```
31 instrukcí s 1 adresou

```
1111 1111 1111 0000
…
1111 1111 1111 1111
```
16 instrukcí bez adresového pole

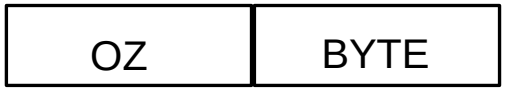
Př. Formáty instrukcí Intel 8080

1B



r1, r2 jsou tříbitové adresy registru. Např. jednobytová instrukce s hexadecimálním kódem 78 znamená MOV A, B, což je přesun obsahu registru B do A

2B



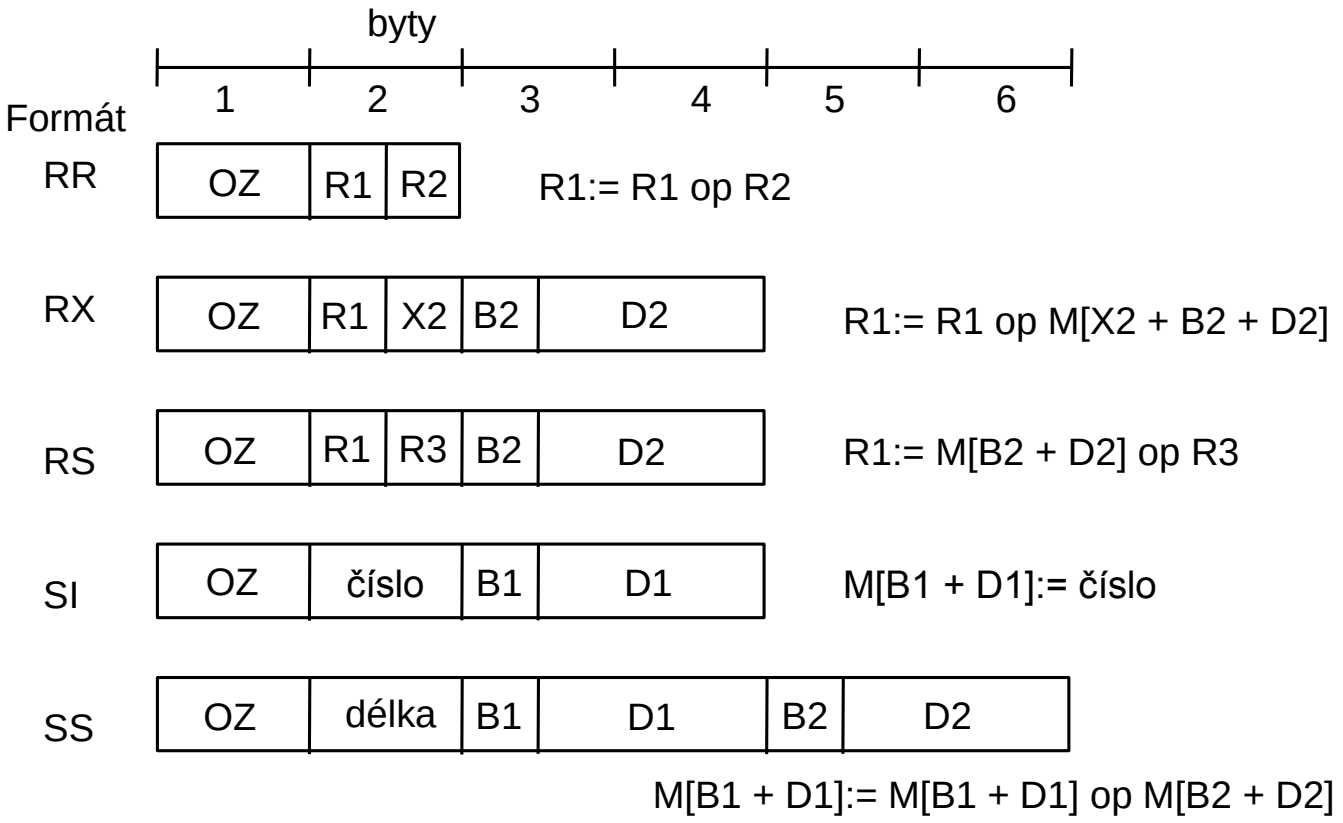
Např. zápis 3E BYTE znamená instrukci MVI, tedy přesun bezprostředně následujícího operandu BYTE do registru A

3B

Např. zápis C3 ADRL ADR znamená instrukci JMP ADR ADRL, tedy nepodmíněný skok na adresu ADR ADRL

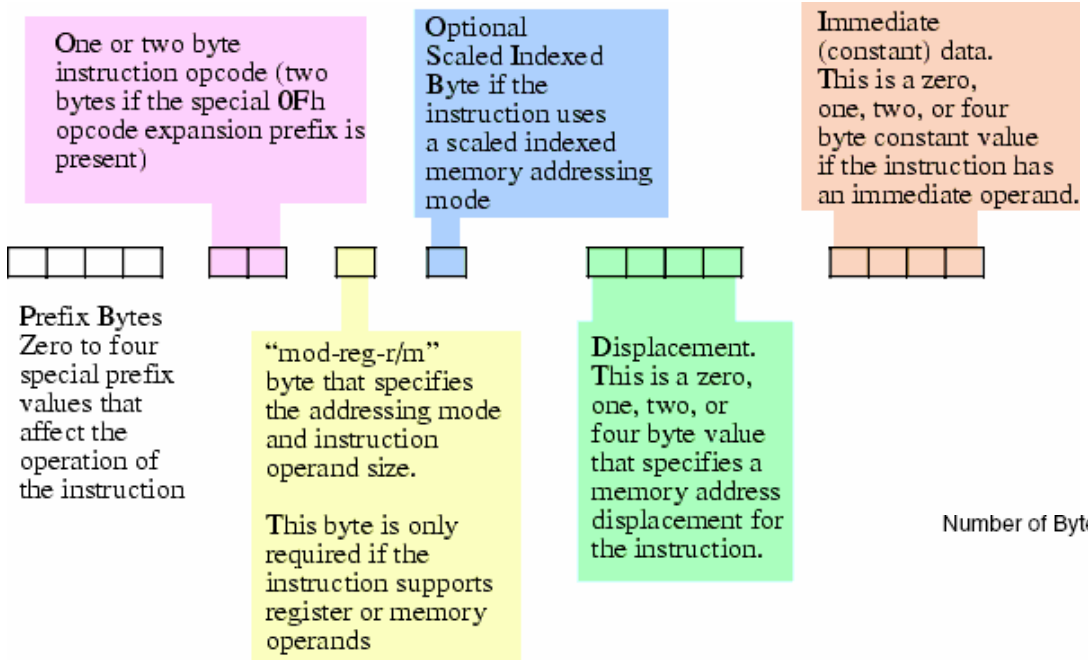
- Adresa paměti je 16-bitová, což znamená, že velikost adresového paměťového prostoru je 2^{16}= 64 K, adresované místo má šířku 1 B.
- Vstup-výstupní instrukce mají formát 2B, v části BYTE je číslo V/V jednotky, kterých tedy může být 256. Říkáme, že adresový prostor V/V má 256 míst.

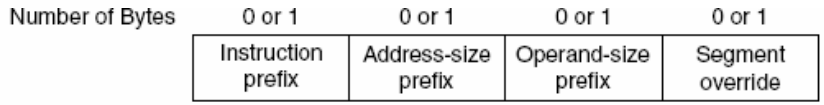
Př. Formáty instrukcí IBM/360



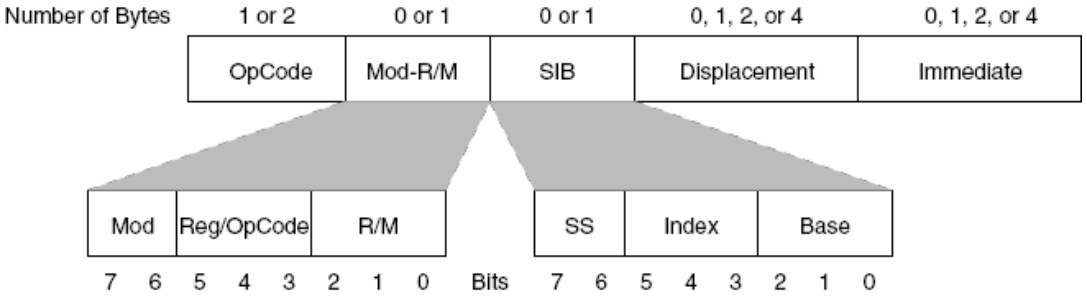
Pozn.: RR: Registr - Registr
RX: Registr - Indexovaná adresa
RS: Registr - Paměť (Storage)
SI: Paměť - Bezprostřední (Immediate) operand
SS: Paměť - Paměť

Př. Formáty instrukcí x86





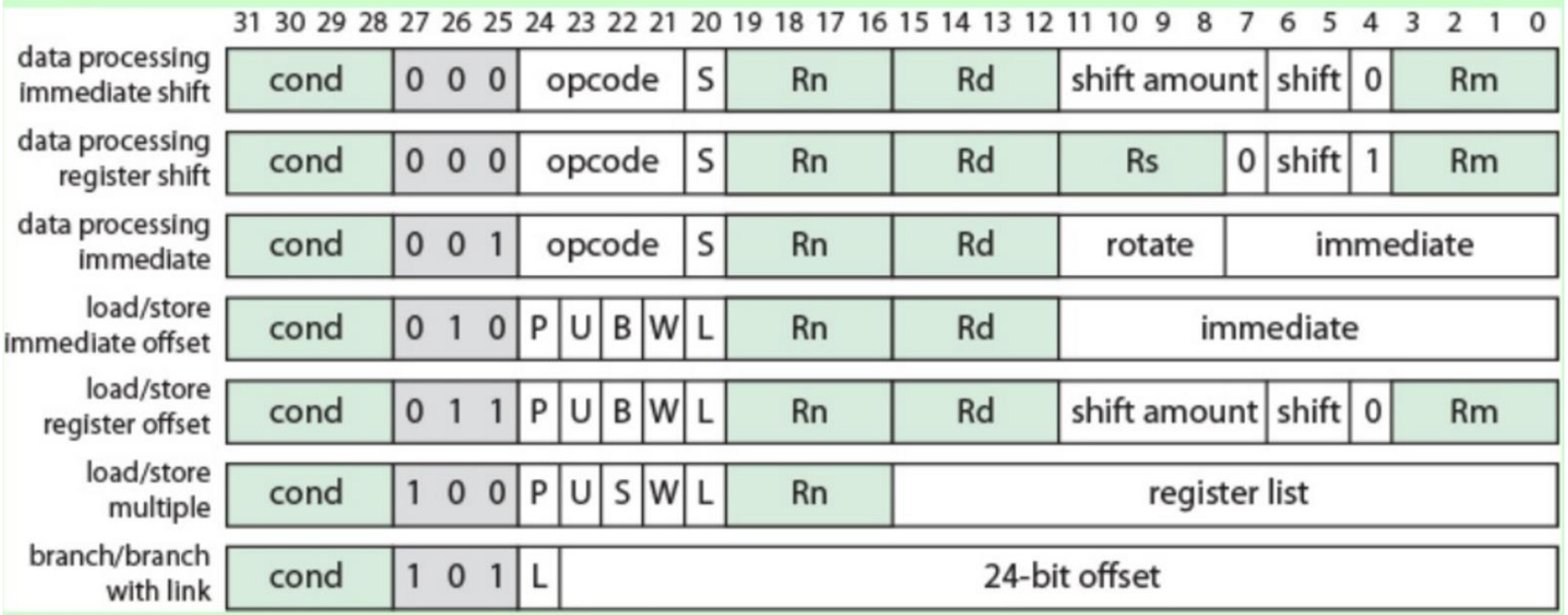
(a) Optional instruction prefixes



(b) General instruction format

http://www.c-jump.com/CIS77/CPU/x86/X77_0020_encoding_overview.htm

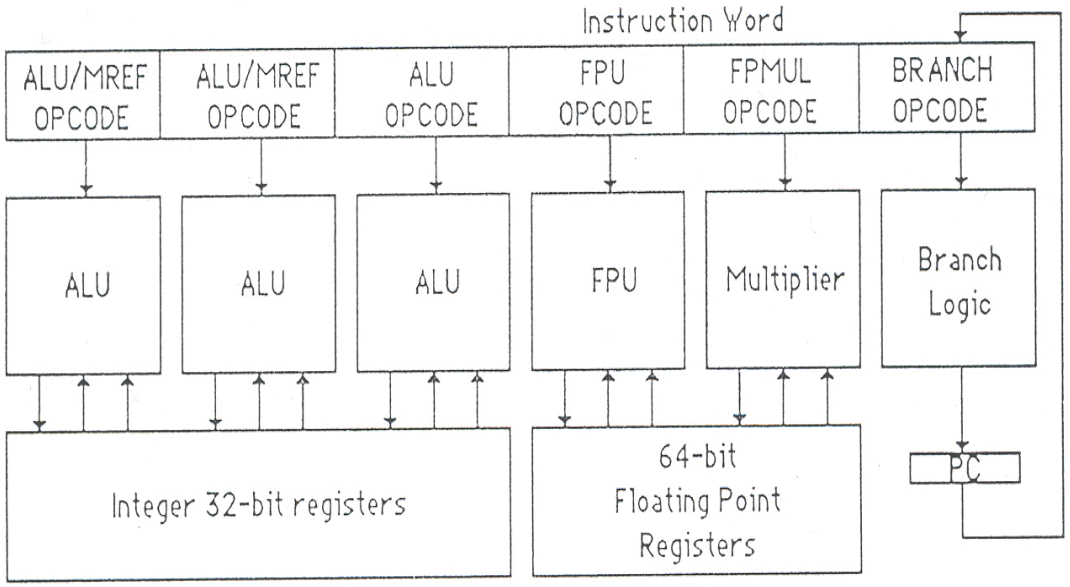
Př. Formáty instrukcí ARM



- S = For data processing instructions, updates condition codes
- S = For load/store multiple instructions, execution restricted to supervisor mode
- P, U, W = distinguish between different types of addressing_mode
- B = Unsigned byte (B==1) or word (B==0) access
- L = For load/store instructions, Load (L==1) or Store (L==0)
- L = For branch instructions, is return address stored in link register

VLIW – Very Long Instruction Word

- Architektura procesoru s paralelně pracujícími vícenásobnými jednotkami.
- V jedné instrukci může být zakódováno několik operací s pevnou řádovou čárkou, s pohyblivou řádovou čárkou i čtení nebo zápisu do paměti
- Kompilátor se snaží generovat kód tak, aby bylo současně využito co nejvíc jednotek.
- Př.



Složitost ISA

- Podle složitosti ISA dělíme procesory na
  - CISC - Complex Instruction Set Computers, tedy procesory se složitým instrukčním souborem a
  - RISC - Reduced Instruction Set Computers, tedy procesory s jednoduchým instrukčním souborem.
- Nejstarší procesory měly málo instrukcí, které byly značně jednoduché.
- Kolem roku 1960 už měly procesory kolem 100 typů instrukcí, které se dále modifikovaly použitým adresovým módem a datovým typem operandů – vzniká CISC.
- Architektura CISC se vyvíjela postupně, přidáváním dalších a dalších, stále složitějších instrukcí, které podporovaly vyšší programovací jazyky a principiálně vyplňovaly tzv. sémantickou mezeru mezi strojovým kódem počítače a příkazy vyšších programovacích jazyků.
- Ukázalo se však, že tyto složité instrukce jsou využívány jen zřídka.
- Vznikla otázka, zda není možné využít plochu na čipu, kterou zabírá jejich obvodová podpora, užitečněji.

RISC – Reduced Instruction Set Computer

- IBM 801 (1973) – jeden z prvních počítačů, který je koncepcí RISC
  - John Cocke – Turingova cena 1987
- Další RISC: MIPS (SPARC), PowerPC, ARM, …v mobilech i superpočítačích
- Relativně málo typů instrukcí a adresových módů (složité instrukce nahrazeny podprogramy sestavenými z jednoduchých instrukcí).
- Pevné a snadno dekódovatelné formáty instrukcí.
- Snaha o CPI = 1 (Cycles per Instruction)
- Řadič procesoru je kvůli rychlosti obvodový, tedy není mikroprogramový.
- Přístup k paměti je pouze v jednoduchých instrukcích Load a Store.
- Pro generování cílového kódu se využívá optimalizujících kompilátorů.
- Jde tedy o koncepci procesoru R-R, přičemž registrů je vyšší počet (deset a více).
- Pro jednoduché úlohy, např. v pevné řádové čárce FX, poběží program „RISC“ rychleji. Bude-li však vysoké procento operací FP, poběží rychleji program „CISC“ (toto nastává u vědeckotechnických výpočtů).
- Nejúspěšnější procesory pragmaticky kombinují principy RISC i CISC.

*** Konec kapitoly ***