

Teorie architektury počítačů

Petr Jedlička, Cyril Klimeš, Tomáš Pitner

Předmluva

HTML · Slides · PDF | Přehled předmětu

◎ Cíle

- Především **porozumnět pojmům**, kterých není v této oblasti za ta léta vývoje málo.
- Pochopit **principy**, na nichž funguje digitální počítač, zejména reprezentace čísel ve dvojkové soustavě a operace v ní.
- Poznat nahrubo architekturu moderních počítačů a její hlavní prvky, jako jsou procesor s vnitřní strukturou a paměť, vstupy a výstupy.

Klíčová informace

🔑 Počítačový systém tvoří jeho hmatatelný technický základ (**hardware**) a programové vybavení (**software**), jehož základní část je systémová, tvořená zejména **operačním systémem**.

Počítačový systém

technické vybavení	hardware; počítač, I/O zařízení — displej, klávesnice, myš, tiskárna, …
programové vybavení	software

Programové vybavení

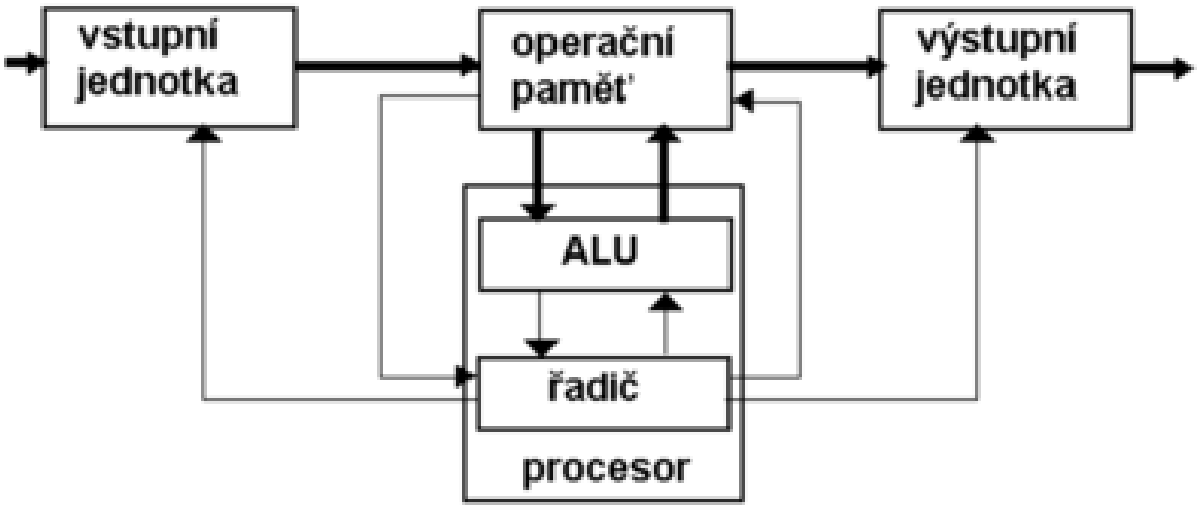
aplikační	umožňuje na počítači provádět nějakou *užitečnou činnost* (👉 zpracování textů, výpočty, kreslení technických výkresů, zpracování obrazu, zvuku nebo videa)
systémové	umožňuje efektivní používání počítače

Systémové programové vybavení

- **operační systém**
- ostatní **systémové programy**

Architektura současných počítačů

- Vychází z koncepce, která vznikla v USA v období 1940-45.
- Hlavní podíl na jejím vzniku měl John von Neumann (1903-57),
- je proto označována jako *von Neumannova architektura*.



Obr. 1. Von Neumannova architektura

Zobrazení informací v počítači

Zobrazení znaků v určené kódové soustavě, která čísla interpretuje jako znaky

Zobrazení čísel ve dvojkové soustavě, v pevné nebo pohyblivé řádové čárce, v přímém, doplňkovém, inverzním kódu

Zobrazení instrukcí

Operační (instrukční) kód říká, co se má provést, spočítat nebo kam jít dál…

Adresní část říká, s čím se to má provést (s jakou hodnotou, kterým registrem, kterou paměťovou buňkou…)

Příklad: prosté přičtení čísla 42 k registru `x1` a uložení výsledku do registru `x0` (v architektuře ARM64):

```
add x0, x1, #42
```



Nemusí jít o (paměťové) adresy v pravém slova smyslu, ale také o registry, konstanty, indexované adresy…

Podle počtu "adres" v instrukci

Zobrazení instrukcí	Operační kód	Adresní část		
jednoadresní	operační kód	A1		
dvouadresní	operační kód	A1	A2	
tříadresní	operační kód	A1	A2	A3

Číselný systém

- Číslo je *řada symbolů*, jímž se říká *číslice* (v běžné, tedy desítkové, soustavě číslice `0..9`).
- Každý symbol má definovánu *váhu* (*hodnotu*) odpovídající jeho pozici v řadě symbolů
- Neboli "více vlevo → vyšší řád → vyšší hodnota"
- Příklad: v čísle `23` má číslice `2` hodnotu `2*10=20` a číslice `3` hodnotu `3*1=3`, takže celé číslo `23` má hodnotu `20+3=23`.
- Každé číslo je tedy *součet* matematických výrazů, kde každý je dán součinem číselného symbolu a jeho váhy, přičemž *váha je mocninou báze* (základu).
- Mocnina (exponent) začíná nulou a roste po jedné (zprava doleva).
- Mějme v desítkové soustavě číslo $347,52\ (dec) = 3.10^{2} + 4.10^{1} + 7.10^{0} + 5.10^{-1} + 2.10^{-2}$.

Používané číselné soustavy

Číselná soustava	Základ	Používané cifry
Dvojková (binární)	2	0, 1
Osmičková (oktalová)	8	0, 1, 2, 3, 4, 5, 6, 7
Desítková (dekadická)	10	0, 1, 2, 3, 4, 5, 6, 7, 8, 9
Šestnáctková (hexa)	16	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F

K čemu ta osmičková?

- Osmičková (oktalová) se dnes používá spíše výjimečně jen v úplně specifických situacích, kdy se nám hodí právě jedna oktalová číslice (=3 bity).
- Napadá vás takový případ použití?
- Nápověda: hledejte v systému souborů… na UNIX/Linux-like systémech

Příklad: Hodnoty 0..16 zapsané v běžných soustavách

Table 1. Tabulka hodnot 0 až 16 v běžných číselných soustavách

DEC	BIN	OCT	HEX
0	0000	00	0
1	0001	01	1
2	0010	02	2
3	0011	03	3
4	0100	04	4
5	0101	05	5
6	0110	06	6
7	0111	07	7
8	1000	10	8
9	1001	11	9
10	1010	12	A
11	1011	13	B
12	1100	14	C
13	1101	15	D
14	1110	16	E
15	1111	17	F
16	10000	20	10

Převod DEC → HEX a DEC → BIN

Příklad: `62393` (desítkově, **dec**) ⇒ `F3B9` (šestnáctkově, **hex**) ⇒ `1111 0011 1011 1001` (dvojkově, **bin**)

- Nejjednodušší je nejdříve z desítkové do šestnáctkové a pak z ní do dvojkové, protože každá šestnáctková číslice odpovídá přesně 4 dvojkovým číslicím, například `B (hex)` = `1011 (bin)`.
- Princip je prostý: opakované dělení základem nové soustavy (tedy 16) a zapisování zbytků po dělení.
- Zbytky čteme od posledního k prvnímu, tedy zdola nahoru.

Krok 1: z DEC → HEX

Dělení 16	Podíl	Zbytek
62393 : 16	3899	9
3899 : 16	243	11 = B
243 : 16	15	3
15 : 16	0	15 = F

Zbytky čteme **zdola nahoru***:*

$$\mathbf{62393_{10} = F3B9_{16}}$$

Krok 2: z šestnáctkové do dvojkové

Table 2. Hexadecimální číslice nyní převedeme po čtveřicích bitů:

F	3	B	9
1111	0011	1011	1001

$$\mathbf{62393_{10} = F3B9_{16} = 1111\ 0011\ 1011\ 1001_2}$$



Převod HEX → BIN je jednoduchý: každá hexadecimální číslice přesně odpovídá jedné čtveřici bitů.

Shrnutí převodu DEC → HEX

1. Dělme celé dekadické číslo novým základem (:16)
2. Zbytek se stává nejnižším řádem nového čísla
3. Dělme výsledek předchozího dělení novým základem
4. Zbytek je následující číslicí nového čísla
5. Opakujme body 3. a 4., dokud neobdržíme nulový výsledek (podíl = 0)

Převod HEX → DEC

Příklad: převod čísla `F3B9`(hex) do desítkové soustavy

- Nejprve nahradíme hexadecimální číslice jejich číselnými hodnotami:
- `F3B9(hex) = (15, 3, 11, 9)`, rozložení na číslice: F → 15, 3 → 3, B → 11, 9 → 9
- Pak použijeme tzv. *Hornerovo schéma*: `(((15 * 16 + 3) * 16 + 11) * 16 + 9)`



Užitím Hornerova schématu vystačíme jen s násobením a sčítáním. Postupně přinásobujeme a přičítáme číslice zleva doprava.

Převod HEX → DEC pomocí Hornerova schématu

Table 3. Postupně počítáme:

Krok	Výsledek
15	15 (→ přeneseme)
15 * 16 + 3	243 (→ přeneseme)
243 * 16 + 11	3899 (→ přeneseme)
3899 * 16 + 9	62393 (→ výsledek)

$$\mathbf{F3B9_{16} = 62393_{10}}$$

Pravidla pro převádění HEX → DEC

1. Vynásobme číslici nejvyššího řádu starým základem.
2. K výsledku přičteme následující číslici.
3. Násobme součet starým základem.
4. Opakujeme body 2. a 3.
5. Převod je ukončen, jakmile podle bodu 2. přičteme číslici, která je nejnižším řádem.

Celá čísla se znaménkem

- **Celá kladná čísla**, nejedná-li se o BCD kód, jsou uložena přímo ve dvojkové soustavě v jednotlivých bitech.
- Celá záporná čísla jsou ukládána podle toho, co podporuje daná procesorová architektura.
- Obvykle nejvyšší bit čísla interpretujeme jako znaménko čísla:
  - 0 → kladné,
  - 1 → záporné,
- přičemž v drtivé většině případů dnes se používá **dvojkový doplňkový kód**.

Dvojkový doplňkový kód

Dvojkový doplněk získáme **inverzí všech bitů a přičtením 1**.

```
 původní číslo: 1100 1000 = -56
bitová inverze: 0011 0111
           +1: 0000 0001
               ---------
               0011 1000 = +56
```



Výhodou je, že sečtením čísla a opačného čísla bez dalších úprav získáme 0.

Přímý kód

- V některých aplikacích se doposud užívá prostého nastavení nejvyššího bitu u záporných čísel: **přímý kód**, kde opačné číslo získáme prostou *inverzí nejvyššího bitu*.

```
původní číslo: 0011 1000 = +56
    přímý kód: 1011 1000 = -56
```

Inverzní kód

- Dříve se objevoval **inverzní kód**, kde opačné číslo získáme *inverzí všech bitů*.

```
původní číslo: 0011 1000 = +56
    přímý kód: 1100 0111 = -56
```

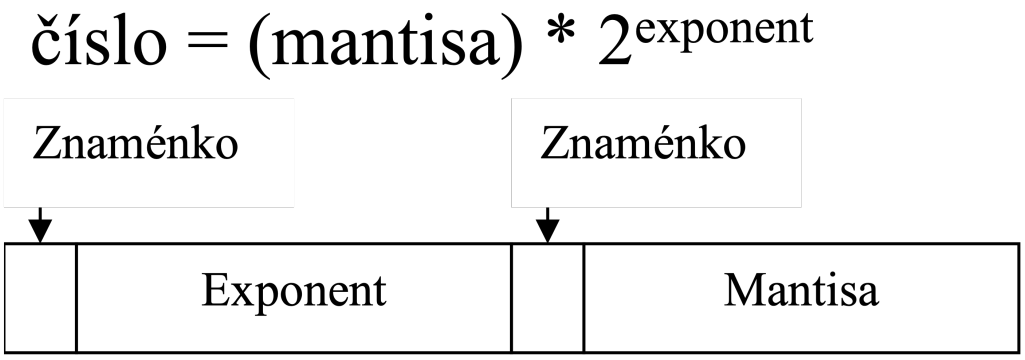


U přímého i inverzního kódu poznáme z nejvyššího bitu znaménko celého čísla.

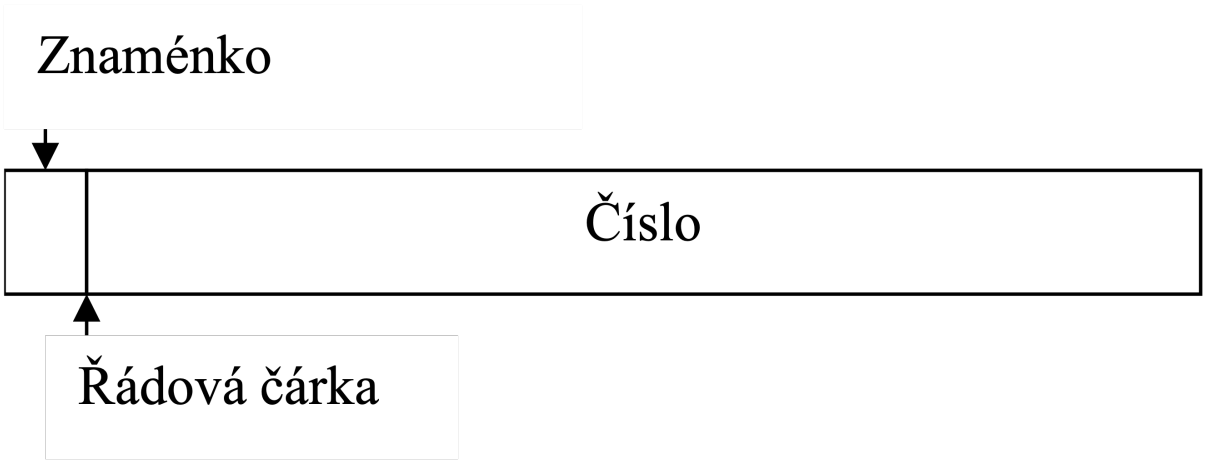
Desetinná čísla v pevné a pohyblivé řádové čárce

- v pevné řádové (desetinné) čárce
- v pohyblivé řádové (desetinné) čárce
- přímý kód
- doplňkový kód
- inverzní kód

Obecné uspořádání čísla



Obr. 2. V pohyblivé řádové čárce



Obr. 3. V pevné řádové čárce

Přímý kód

- Přímý kód používá nejvyšší bit pro znaménko:
  - `0` pro kladné číslo,
  - `1` pro záporné číslo.
- Zbývající bity vyjadřují absolutní hodnotu čísla.

```
+7/32 = 0,00111
-7/32 = 1,00111
```

Doplňkový kód

- (x) doplňkový kód = x pro x >= 0
- (x) doplňkový kód = 2 + x pro x < 0
- $+0{,}75_{10} = 0{,}110_2$
- $-0{,}75_{10} = 1{,}010_2$

```
   10,000   =(2)
   -0,110   =(+0,75) v doplňkovém
   ------
=  01,010   =(-0,75) v doplňkovém
```

Inverzní kód

- (x) inverzní kód = x pro x >= 0
- (x) inverzní kód = $2 + x - 2^{-n}$ pro x < 0
- n je počet bitů za řádovou čárkou.

Záporné číslo lze získat také prostou **inverzí všech bitů** kladného čísla.

- $+0{,}8125_{10} = 0{,}1101_2$
- $-0{,}8125_{10} = 1{,}0010_2$

Modifikované kódy

- Získáme z výše uvedených (doplňkového nebo inverzního) tak, že
- **zdvojíme znaménkový bit**.
- Zdvojení znaménkového bitu umožňuje snadnou **detekci přetečení**:
  - 00 — kladné číslo,
  - 11 — záporné číslo,
  - 01 nebo 10 — přetečení.

Modifikovaný doplňkový kód

Modifikovaný kód používá **dva znaménkové bity**.

- (x) modifikovaný doplňkový kód = x pro x >= 0

- (x) modifikovaný doplňkový kód = `4 + x` pro `x < 0`
- $+11/16_{10} = 00,1011_2$
- $-11/16_{10} = 11,0101_2$

```
   100,0000
   -00,1011
  ---------
=  011,0101
```

Modifikovaný inverzní kód

Modifikovaný kód používá **dva znaménkové bity**.

- (x) modifikovaný inverzní kód = `x` pro `x >= 0`
- (x) modifikovaný inverzní kód = $4 + x - 2^{-n}$ pro `x < 0`
- `n` je počet bitů za řádovou čárkou.

Záporné číslo opět získáme **inverzí všech bitů**:

```
+11/16 = 00,1011
         || ||||
         vv vvvv
-11/16 = 11,0100
```

Sčítání dvojkových čísel

- Význam číslic: bit přenosu z přechozího C(n), bit X, bit Y = výsledný bit Z (a přenos C(n+1))

```
Cn  X   Y   Z (přenos Cn+1)
0 + 0 + 0 = 0 (přenos 0)
0 + 1 + 0 = 1 (přenos 0)
1 + 0 + 0 = 1 (přenos 0)
1 + 1 + 0 = 0 (přenos 1)
0 + 0 + 1 = 1 (přenos 0)
0 + 1 + 1 = 0 (přenos 1)
1 + 0 + 1 = 0 (přenos 1)
1 + 1 + 1 = 1 (přenos 1)
```

Doplňkové sčítání

- Odečítáme-li číslo, provedeme to jako **přičtení doplňku** odečítaného čísla
- Při doplňkovém sčítání, dojde-li k přenosu do vyššího řádu, je výsledek v normální formě.
- Pokud nedojde k přenosu do vyššího řádu, je doplněk v doplňkové formě. Aby byl výsledek v normálním tvaru, musí počítač udělat 2 kroky:

1. vytvořit doplněk k výsledku v doplňkové formě
2. změnit znaménko výsledku

```
1110 1001          1110 1001
-0110 1011         1001 0101
                  1 0111 1110 - výsledek je v normální formě

 0111 0111         0111 0111
-1001 1101         0110 0011
                   1011 1010 - nedošlo k přenosu!

Doplněk:
      1011 1010    číslo
      0100 0101
           + 1
      ---------
      0100 0110    doplněk
Výsledek: - 0100 0110
```

Násobení

Násobení binárních čísel

Stejně jako v desítkové soustavě můžeme násobit čísla *pod sebou*. V binární soustavě je násobení jednotlivých číslic obzvlášť jednoduché: násobíme pouze `0` nebo `1`.

Příklad: 1011_2 x 1101_2

```
   1011
 x 1101
 ------
   1011    ← 1011 x 1
  0000     ← 1011 x 0, posun o 1
 1011      ← 1011 x 1, posun o 2
1011       ← 1011 x 1, posun o 3
--------
10001111
```

Kontrola v desítkové soustavě

1. 1011_2 = 11
2. 1101_2 = 13
3. 11 * 13 = 143 = 10001111_2



Každý další částečný součin je posunut o jeden bit doleva. Násobení čísla dvěma tedy v binární soustavě odpovídá posunu o jeden bit doleva.

Dělení

```
A / B = Q + Z / B
A = Q * B + Z
A - Q * B = Z
Z < B
```

A dělenec

B dělitel

Z zbytek

Q podíl

Dělení (2)

- Dělení bez návratu ke kladnému zbytku

Dělenec	A	0,10001100		Q
Dělitel	B	-0,11100000		
		-0,01010100	A-B<0	0
	A	-0,10101000		
	B	+0,11100000		
		0,00111000	A+B>0	1
	A	0,01110000		
	B	-0,11100000		
		-0,01110000	A-B<0	0
	A	-0,11100000		
	B	+0,11100000	A+B =0	1
		0,00000000		
		Q = 0,101		

- Dělení s návratem ke kladnému zbytku

Dělenec A	0,10001100		
Dělitel B	-0,11100000		Q
	-0,01010100	A-B<0	0
	0,11100000	+B	
	0,10001100	A	
	-0,01110000	posuv B doprava	
	0,00011100	A-B>0	1
	-0,00111000	posuv B doprava	
	-0,00011100	A-B<0	0
	0,00111000	+B	
	0,00011100	A	
	-0,00011100	posuv B doprava	
	0,00000000	A-B = 0	1
	-0,00001110	posuv B doprava	
	-0,00001110	A-B<0	0

Q = 0,101

Zobrazení v pohyblivé řádové čárce

`číslo = (mantisa) * 2^(exponent)`

Příklad

- Slovo má 1+32 bitů.
- Zobrazte reálné číslo: `145,625` v přímém kódu:
  - znaménko čísla ve znaménkovém bitu
  - mantisa v horních 24 bitech
  - exponent v nejnižších 8 bitech, nejvyšší z nich obsahuje znaménko exponentu

Postup

1. převedeme do dvojkové soustavy `145,625(dec) = 10010001,101(bin)`
2. posuneme řádovou čárku před nejvyšší bit a vynásobíme příslušnou mocninou základu 2: $x = 0,10010001101 * 2^{8}$
3. připojíme znaménko `-145,625(dec) = 1,10010001101000000000000000001000(bin)`

Sčítání v pohyblivé řádové čárce

- Odečtení exponentů
- Posuv mantisy čísla s menším exponentem doprava o počet bitů, který je roven rozdílu exponentů
- Sečtení mantis obou čísel
- Určení počtu nul mezi řádovou čárkou a první platnou číslicí součtu mantis
- Posunutí součtu doleva o tolik míst, kolik nul bylo nalezeno za řádovou čárkou
- Zmenšení původního exponentu o počet nalezených nul

Bitové operace posunu a rotace

- Mívají jeden až dva parametry: **co se posouvá** a o **kolik pozic** (bitů)
- Instrukce sama označuje **směr** posunu:

  doprava k nižším pozicím (od MSB k LSB)

  doleva k vyšším pozicím (od LSB k MSB)

Příklad: Posun vlevo osmibitového registru (SHL=SHift Left) o 1 pozici

před 0110 1110

po 1101 1100

- zprava = zespodu, tedy do LSB, se nasunula nula, ven vlevo z MSB se vysunula také 0

Co se provede?

- Bitový obsah posouvané hodnoty (obvykle umístěné v registru různé délky 8, 16, 32, 64 bitů) se posune o daný počet pozic vlevo nebo vpravo:

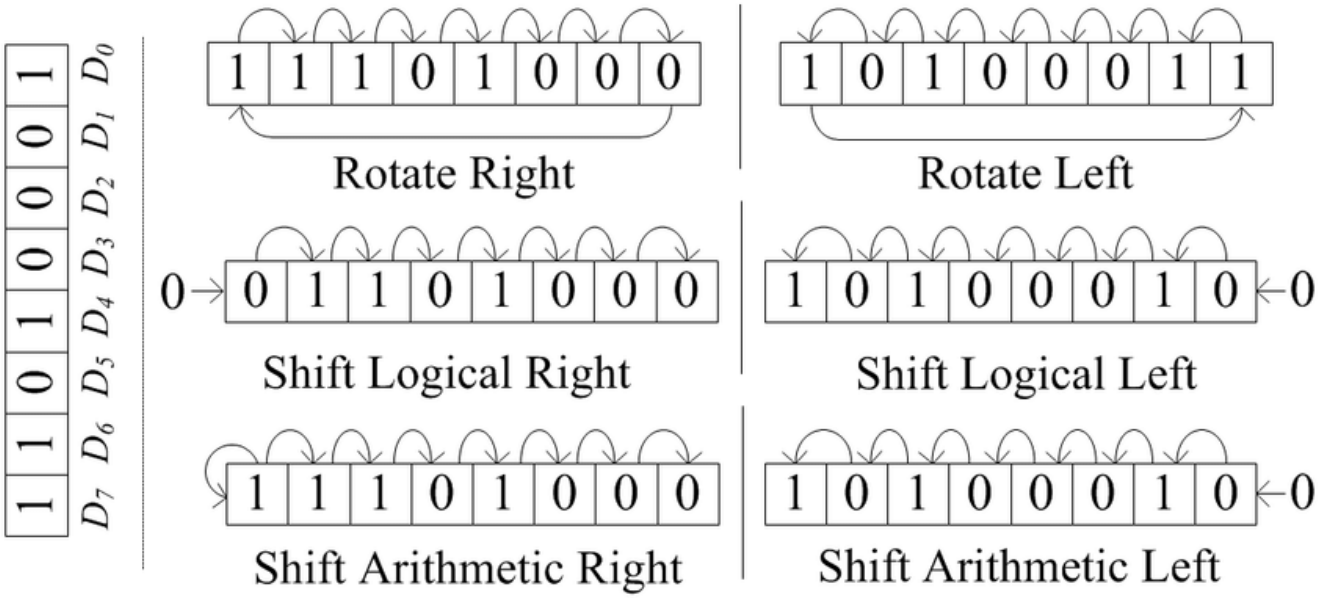
Rotace

- v případě **rotace** se vysunuté bity *nasouvají na uvolněné pozice zpět* do zdrojového registru (x64: instrukce `ROL`, `ROR`);
- v případě **rotace přes CF** se efektivně rotuje nejen rotovaná hodnota, ale i *příznak přenosu* (Carry Flag, CF) (x64: instrukce `RCL`, `RCR`);

Posun

- v případě **posunu** vysunuté bity mizí a na uvolněné pozice se přisouvají nuly.
- v případě *aritmetického posunu doprava* (arithmetic shift right) se zleva nepřisouvá nula, nýbrž v MSB (=znaménku) hodnota znaménka zduplikuje, zůstává.

Typy posunu a rotace o jeden bit


1 1 0 1 0 1 0 0
D7 D6 D5 D4 D3 D2 D1 D0
1 1 1 0 1 0 0 0
Rotate Right
1 0 1 0 0 0 1 1
Rotate Left
0
0 1 1 0 1 0 0 0
Shift Logical Right
1 0 1 0 0 0 1 0 0
Shift Logical Left
1 1 1 0 1 0 0 0
Shift Arithmetic Right
1 0 1 0 0 0 1 0 0
Shift Arithmetic Left

Kód BCD

- Zajímavý dnes spíš historicky; *Binary Coded Decimal* (BCD) je kód, který umožňuje reprezentovat desítková čísla ve dvojkové soustavě tak, že každá desítková cifra je na pevných 4 bitech,
- např. číslo `23` (desítkově) se zapíše jako `0010 0011` (BCD),
- ale i číslo `12` (desítkově) je v BCD reprezentováno jako dvě čtveřice bitů `0001 0010` (BCD) nikoli na jedné čtveřici `1100` (klasicky binárně).

Kód BCD (2)

Desítkově	BCD	Desítkově	BCD
0	0000	10	0001 0000
1	0001	11	0001 0001
2	0010	12	0001 0010
3	0011	13	0001 0011
4	0100	14	0001 0100
5	0101	15	0001 0101
6	0110	16	0001 0110
7	0111	17	0001 0111
8	1000	18	0001 1000
9	1001	19	0001 1001

Zápis desítkového čísla v kódu BCD

`79523(dec) = 0111 1001 0101 0010 0011 (BCD)`

Aikenův kód 2421

Aikenův kód je alternativní způsob kódování desítkových číslic:

- podobně jako BCD 8421 kóduje každou desítkovou číslici pomocí **4 bitů**,
- používá však váhy **2-4-2-1** místo 8-4-2-1,
- je **samodoplňkový**: inverzí všech bitů získáme **devítkový doplněk** číslice,
- např. `3 = 0011` → inverze → `1100 = 6`,
- důvod: součet vah je `2 + 4 + 2 + 1 = 9`, proto inverze převádí číslici `d` na `9 - d`.



Pro získání **desítkového doplňku** je třeba k devítkovému doplňku ještě přičíst 1.

Kód 2421 má dnes především historický a didaktický význam.

Tabulka dvojkových kódů pro desítkové číslice

Desítkově	8421 (BCD)	2421 (Aiken)	(8421)+3 (Excess-3)	5421	3n+2	5043210 (biquinární)
0	0000	0000	0011	0000	00010	0100001
1	0001	0001	0100	0001	00101	0100010
2	0010	0010	0101	0010	01000	0100100
3	0011	0011	0110	0011	01011	0101000
4	0100	0100	0111	0100	01110	0110000
5	0101	1011	1000	1000	10001	1000001
6	0110	1100	1001	1001	10100	1000010
7	0111	1101	1010	1010	10111	1000100
8	1000	1110	1011	1011	11010	1001000
9	1001	1111	1100	1100	11101	1010000

Vysvětlení hlaviček (váhy jednotlivých bitů zleva doprava)

- **8421 (BCD)** – váhy 8-4-2-1, základní přímý BCD kód,
- **2421 (Aikenův)** – váhy 2-4-2-1, samodoplňkový (inverzí bitů vznikne doplněk do 9),
- **(8421)+3 (Excess-3)** – kód s přebytkem 3, hodnota = BCD + 3, samodoplňkový,
- **5421** – váhy 5-4-2-1, samodoplňkový,
- **3n+2** – binární zápis čísla 3·n + 2 (n = desítková číslice), samodoplňkový, neváhový,
- **5043210** – biquinární kód s váhami 5-0-4-3-2-1-0 (dva bity určují pětku, pět bitů zbytek).

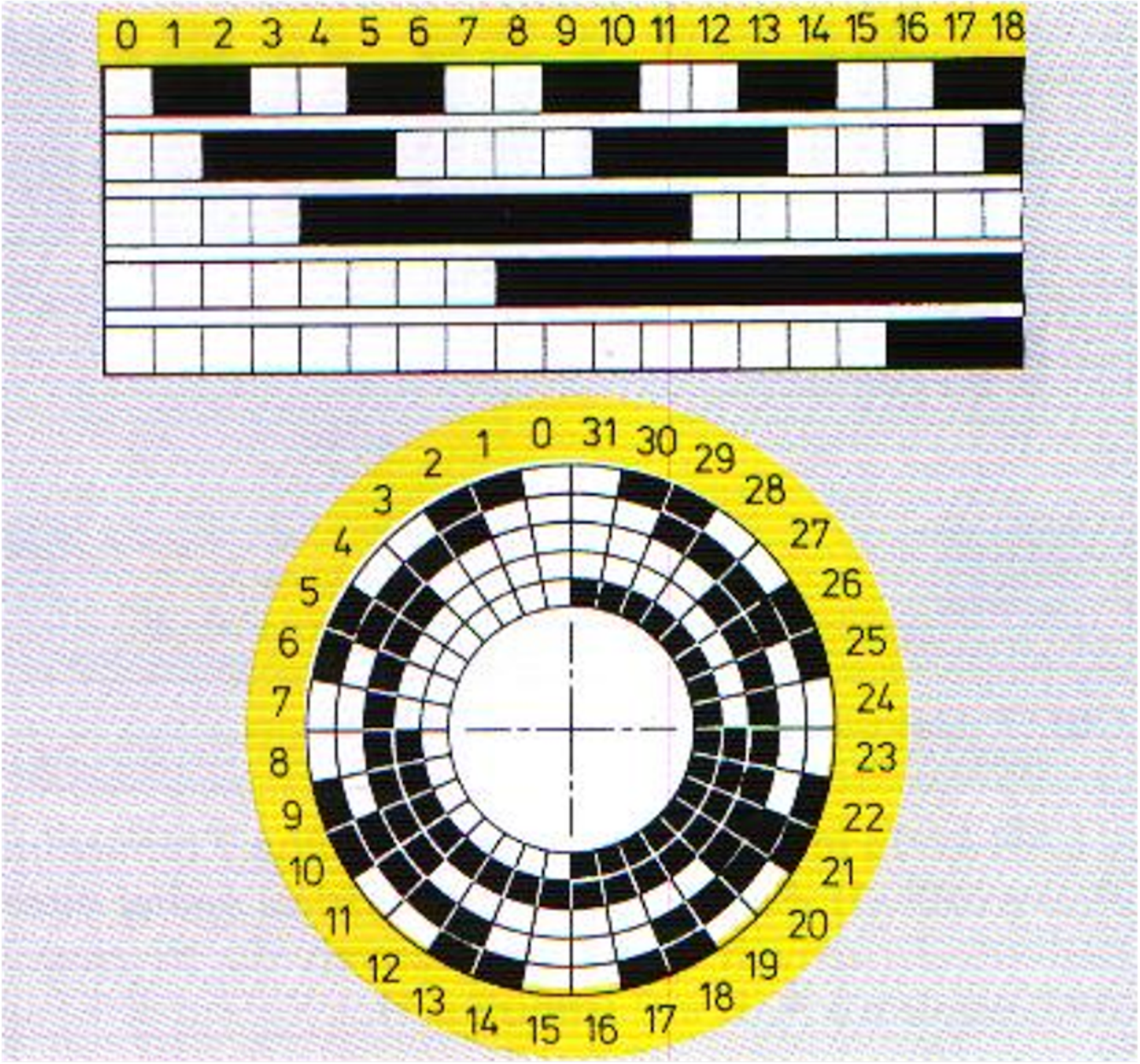
Zabezpečení informace (kódů) paritou

Bity čísla	Sudá parita
10110	1
01001	0
01111	0
01011	1
01001	0



Sudá parita tedy "dorovnává" počet jedniček na sudý.

Kódové pravítko a kotouč v Grayově kódu



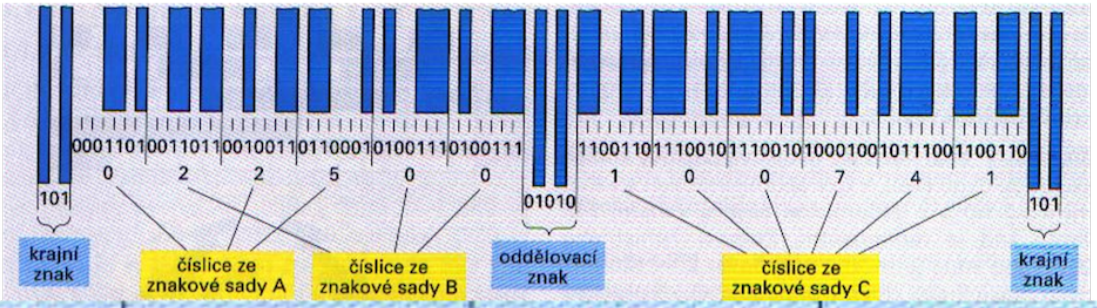
- V Grayově kódu se při přechodu +/- 1 mění jen jeden bit.
- Změna jen jednoho bitu při +/-1 má u některých systémů příznivý dopad na spolehlivost — například je nižší rušení a nekolísá odběr proudu.

Čárové kódy

název kódu	znaková zásoba	– znaků v modulu – počet šířek proužků (SB) – počet šířek mezer (LB)
lineární kód (poštovní)	číslice 0, ..., 9	Z = 4 3 z 5 plus 1 startovací
EAN-kód	číslice 0, ..., 9 5 pom. znaků	Z = 7 SB = 4 LB = 4
kód 2/5 (překryvný)	číslice 0, ..., 9 start, stop	Z = 3 SB = 2 LB = 2
Codabar	číslice 0, ..., 9 6 pom. znaků	Z = 4 SB = 2 LB = 2
Code 39	číslice 0, ..., 9 26 alfanum. znaků 7 pomocných znaků	Z = 4 SB = 2 LB = 2
Code 128	ASCII kód 128 znaků	Z = 3 SB = 4 LB = 4
Code 2/5	číslice 0, ..., 9 start, stop	Z = 5 SB = 2



Dekódování čárového EAN kódu



znak	znaková sada A	znaková sada B	znaková sada C
0	0001101	0100111	1110010
1	0011001	0110011	1100110
2	0010011	0011011	1101100
3	0111101	0100001	1000010
4	0100011	0011101	1011100
5	0110001	0111001	1001110
6	0101111	0000101	1010000
7	0111011	0010001	1000100
8	0110111	0001001	1001000
9	0001011	0010111	1110100

desítkový exponent	předpona	desítkový exponent	předpona	dvojkový exponent	desítková hodnota
10^{-3}	mili	10^{3}	Kilo	2^{10}	1024
10^{-6}	micro	10^{6}	Mega	2^{20}	1024 K
10^{-9}	nano	10^{9}	Giga	2^{30}	1024 M
10^{-12}	pico	10^{12}	Tera	2^{40}	1024 G
10^{-15}	femto	10^{15}	Peta	2^{50}	1024 T
10^{-18}	atto	10^{18}	Exa	2^{60}	1024 P
10^{-21}	zepto	10^{21}	Zetta	2^{70}	1024 E
10^{-24}	yocto	10^{24}	Yotta	2^{80}	1024 Z

Znaky na počítači

- Počítače dávno nepočítají jen s čísly.
- Častěji s *textem*, který je třeba nějak kódovat.
- 👉 Obvykle tak, že co *znak* z nějaké *sady* (*character set*), to jeho číslo (*codepoint*).
- Historicky byla zásadním milníkem sada ASCII (1963)
- Nicméně znakových sad vzniklo enormní množství,
- téměř pro každou národní abecedu.

Kódování znaků ASCII

ASCII

American Standard Code for Information Interchange (1963)

- 7bitové kódování znaků
- 128 znaků (kódy 0-127)
- Zahrnuje *písmena* anglické abecedy (velká, malá), *číslice*, *interpunkci* a *řídicí znaky*.
- Na spoustu účelů jako programování, popisovače, konfigurační soubory… funguje a stačí.
- Na znaky většiny světových abeced už *nestačí*.

Znaky 7bitového ASCII

Dec	Hex	Symbol	Dec	Hex	Symbol
0	00	NUL (null)	32	20	(space)
1	01	SOH (start of heading)	33	21	!
2	02	STX (start of text)	34	22	"
3	03	ETX (end of text)	35	23	#
4	04	EOT (end of transmission)	36	24	$
5	05	ENQ (enquiry)	37	25	%
6	06	ACK (acknowledge)	38	26	&
7	07	BEL (bell)	39	27	'
8	08	BS (backspace)	40	28	(
9	09	HT (horizontal tab)	41	29	)
10	0A	LF (line feed)	42	2A	*
11	0B	VT (vertical tab)	43	2B	+
12	0C	FF (form feed)	44	2C	,
13	0D	CR (carriage return)	45	2D	-
14	0E	SO (shift out)	46	2E	.
15	0F	SI (shift in)	47	2F	/
16	10	DLE (data link escape)	48	30	0
17	11	DC1 (device control 1)	49	31	1
18	12	DC2 (device control 2)	50	32	2
19	13	DC3 (device control 3)	51	33	3
20	14	DC4 (device control 4)	52	34	4

Dec	Hex	Symbol	Dec	Hex	Symbol
21	15	NAK (negative ack.)	53	35	5
22	16	SYN (synchronous idle)	54	36	6
23	17	ETB (end of block)	55	37	7
24	18	CAN (cancel)	56	38	8
25	19	EM (end of medium)	57	39	9
26	1A	SUB (substitute)	58	3A	:
27	1B	ESC (escape)	59	3B	;
28	1C	FS (file separator)	60	3C	<
29	1D	GS (group separator)	61	3D	=
30	1E	RS (record separator)	62	3E	>
31	1F	US (unit separator)	63	3F	?

Horní polovina 7bitového ASCII

Dec	Hex	Symbol	Dec	Hex	Symbol
64	40	@	96	60	`
65	41	A	97	61	a
66	42	B	98	62	b
67	43	C	99	63	c
68	44	D	100	64	d
69	45	E	101	65	e
70	46	F	102	66	f
71	47	G	103	67	g
72	48	H	104	68	h
73	49	I	105	69	i
74	4A	J	106	6A	j
75	4B	K	107	6B	k
76	4C	L	108	6C	l
77	4D	M	109	6D	m
78	4E	N	110	6E	n
79	4F	O	111	6F	o
80	50	P	112	70	p
81	51	Q	113	71	q
82	52	R	114	72	r

Dec	Hex	Symbol	Dec	Hex	Symbol
83	53	S	115	73	s
84	54	T	116	74	t
85	55	U	117	75	u
86	56	V	118	76	v
87	57	W	119	77	w
88	58	X	120	78	x
89	59	Y	121	79	y
90	5A	Z	122	7A	z
91	5B	[	123	7B	{
92	5C	\	124	7C	|
93	5D	]	125	7D	}
94	5E	^	126	7E	~
95	5F	_	127	7F	DEL

Řídicí znaky ASCII

Řídicí znaky ASCII jsou určeny pro ovládání zařízení, jako jsou tiskárny nebo terminály:

`NUL` (null) znamená žádný znak (kód 0)

`CR` (carriage return) vrátí kurzor na začátek řádku (kód 13)

`LF` (line feed) posune kurzor na další řádek (kód 10)

`ESC` (escape) slouží k zahájení speciálních sekvencí (kód 27)

`BEL` (bell) terminál pípne (kód 7)

`HT` (horizontal tab) posune kurzor na další tabulátor (kód 9)

Ukončování řádků

V počítačových systémech se používají různé způsoby ukončování řádků:

Unix/Linux jen `LF` (Line Feed, kód 10)

Windows `CR+LF` (Carriage Return + Line Feed, kódy 13 a 10)

Mac OS (před OS X) jen `CR` (Carriage Return, kód 13), nyní jako Unix/Linux (`LF`)

Unicode

- **ASCII** dávno nestačí
- **Unicode**: standard pro kódování znaků, který zahrnuje většinu světových písem
- https://home.unicode.org/

> universal (addressing the needs of world languages) uniform (fixed-width codes for efficient access), and unique (bit sequence has only one interpretation into character codes)
>
> — Unicode Consortium

Unicode: co v něm je?

- Spousta znaků z *různých abeced* a symbolů, https://www.unicode.org/charts/
- Zahrnuje i *ikonky*, *symboly*, *emoji*
- Úsporný způsob kódování nejen znaků
- Namísto bitmapy obrázku stačí kód emoji

Unicode: zajímavosti

- Unicode zahrnuje t.č. více než *1,1 miliónu kódových pozic*.
- Unicode podporuje různé jazyky a písma, včetně *latinky*, *azbuky*, *čínských znaků*, *arabštiny* a dalších.
- Unicode umožňuje reprezentovat znaky z různých jazyků a kultur *v jednom standardu*,
- tedy v jednom textu může být současně libovolný počet libovolných různých znaků libovolných abeced světa.

Unicode: varianty znaků

- Unicode zahrnuje i historické znaky, matematické symboly, hudební notace a další
- Pozoruhodné je, že některé znaky mohou být zapsány různými způsoby:
- é může být zapsáno jako jeden znak (U+00E9) nebo jako dva znaky (U+0065 + U+0301)
- Některé znaky (třeba písmeno Z má v Unicode několik codepointů podle účelu toho písmena):

U+2124 ℤ *double-struck capital z* (často používané pro množinu celých čísel)

U+2128 ℨ *black-letter capital z*

Unicode: definuje tabulku znaků

Každý znak má přiřazen(o):

číslo (kód, *codepoint*)	`U+005A`
název znaku	`LATIN CAPITAL LETTER Z`
obvyklé zobrazení znaku	`Z`

Unicode codepoint

Každý znak má unikátní kódovou pozici (*codepoint*, ordinální číslo), například:

- `A` je U+0041 (= 0x41, 65 v desítkové soustavě)
- `a` je U+0061
- `č` je U+010D
- `` je U+1F60A (emoji)



Codepoint se zapisuje U + 00ED (tj. U+hexa číslice).

Unicode: definuje pravidla

- například převod malá písmena na velká
- pravidla řazení znaků
- například zobrazení znaků v různých písmech

Unicode: poskytuje nástroje

- knihovny (*International Components for Unicode*, ICU)
- vzorová zobrazení znaků a jejich vlastností

Unicode: kódování znaků

Unicode definuje více způsobů kódování znaků, nejběžnější jsou:

UTF-8 variabilní délka, zpětně kompatibilní s ASCII

UTF-16 většinou 2 bajty na znak, ale některé znaky mohou zabrat 4 bajty

UTF-32 pevná délka, každý znak zabírá 4 bajty

UTF-8 i 16 mají výhody i nevýhody

- *proměnný počet* bajtů na znak
- menší zábor paměti
- složitější zpracování: *n*-tý znak není na pozici *4n* ani *2n* ...

UTF-8 schéma kódování

Rozsah Unicode	Hex rozsah	Binární UTF-8 vzor	Délka (B)	Poznámka
U+0000 – U+007F	0x00 – 0x7F	`0xxxxxxx`	1	ASCII kompatibilní, čistá angličtina, čísla,...
U+0080 – U+07FF	0x80 – 0x7FF	`110xxxxx 10xxxxxx`	2	Evropské znaky, latinka s diakritikou
U+0800 – U+FFFF	0x800 – 0xFFFF	`1110xxxx 10xxxxxx 10xxxxxx`	3	Základní multijazyčné znaky, např. řečtina, hebrejština
U+10000 – U+10FFFF	0x10000 – 0x10FFFF	`11110xxx 10xxxxxx 10xxxxxx 10xxxxxx`	4	Emojis, historická písma, zřídka používané znaky



- `x` označuje bity, do kterých se zakóduje samotný Unicode point.
- UTF-8 je **variabilně dlouhé kódování**: čím vyšší znak, tím více bajtů.

UTF-8 reprezentace slova "žlutý"

žlutý (5 znaků) → `C5 BE 6C 75 74 C3 BD` (7 bajtů = 2+1+1+1+2)

Znak	Unicode kód (U+xxxx)	UTF-8 bajty (hex)	Poznámka
ž	U+017E	C5 BE	Dvoubytový znak (písmeno s háčkem)
l	U+006C	6C	ASCII (jednobytový)
u	U+0075	75	ASCII (jednobytový)
t	U+0074	74	ASCII (jednobytový)

Znak	Unicode kód (U+xxxx)	UTF-8 bajty (hex)	Poznámka
ý	U+00FD	C3 BD	Dvoubytový znak (písmeno s čárkou)

Starší způsoby kódování UCS (Universal Character Set)

UCS-2 2 bajty na znak, používané v Javě a C# - prvních `2^16 = 65536` znaků se zapsalo přímo jako jejich codepoint, bez problémů stačilo na tzv. *Basic Multilingual Plane* (BMP), běžné "malé" abecedy

UCS-4 4 bajty na znak



UCS-2 bylo omezeno na 2^16 = 65536 znaků, což nestačilo pro všechny znaky Unicode a musely se používat (třeba v Javě) *surrogate páry* pro znaky mimo tento rozsah.

Endianita u Unicode kódování

- Endianita (Little- a Big-) znamená způsob, jak ukládat vícebajtová slova do posloupnosti bajtů — třeba do paměti nebo do (běžného sekvenčního) souboru

Little-Endian nižší bajty na nižších pozicích

Big-Endian vyšší bajty na nižších pozicích

- Endianita je podstatná u:
  1. ukládání vícebajtových instrukcí do paměti; třeba 64bitové na ARM: big-endian a Intel/AMD: little-endian
  2. textových souborů v *kódováních Unicode*: UTF-16, UTF-32, UCS-2, UCS-4

Označení endianity: *Byte-Order Mark*

- Některé systémy používají na začátku souboru tzv. *Byte-Order Mark* (BOM):

Kódování	BOM (hex)	Význam
UTF-8	EF BB BF	Označuje soubor jako UTF-8 (není povinné)
UTF-16 BE	FE FF	UTF-16 big endian (nejvýznamnější bajt první)
UTF-16 LE	FF FE	UTF-16 little endian (nejméně významný bajt první)
UTF-32 BE	00 00 FE FF	UTF-32 big endian
UTF-32 LE	FF FE 00 00	UTF-32 little endian

Deklarace (sdělení) zvoleného kódování znaků

- V *elektronické poště*, *HTTP* a dalších protokolech:
- pomocí `charset` v *MIME type* (*Multipurpose Internet Mail Extensions*), RFC 2045 a následujících.

```
Content-Type: text/plain; charset=us-ascii
Content-Type: text/plain; charset=iso-8859-2
Content-Type: text/plain; charset=windows-1250
Content-Type: text/plain; charset=UTF-8
```

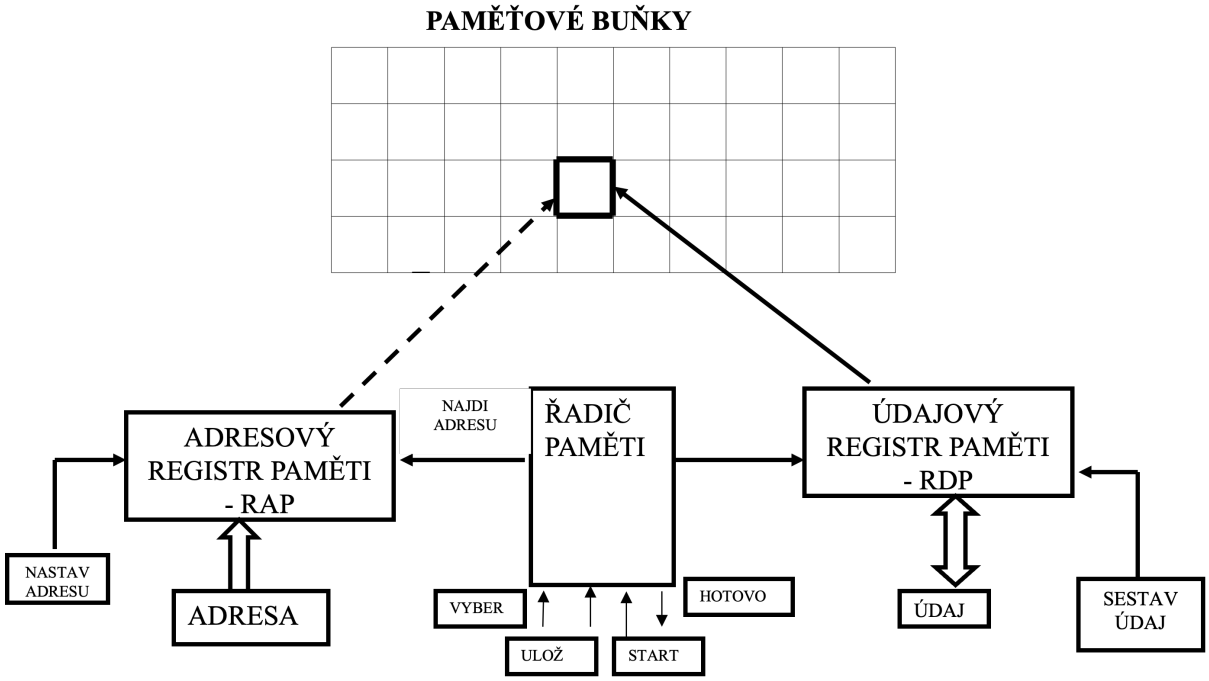
Popis částí počítače

Dříve, než si v dalším výkladu každou zdánlivou samozřejmost rozebereme pořádně a do detailu, projdeme tu na jednom místě jen zběžně základní rozdělení počítače — paměť, procesor a V/V zařízení --, abychom od začátku měli představu, za co která část zodpovídá a jak spolu souvisejí.

- paměť
- procesor
- V/V zařízení

Operační paměť

Operační paměť (RAM) je *rychlý, ale ztrátový* pracovní prostor, v němž procesor drží instrukce a data, s nimiž právě pracuje, a právě proto — na rozdíl od trvalých, ale výrazně pomalejších pamětí (SSD, HDD, páska) — přímo podmiňuje, jak svižně počítač běží; bez ní by nebyl schopen provádět žádné výpočty.

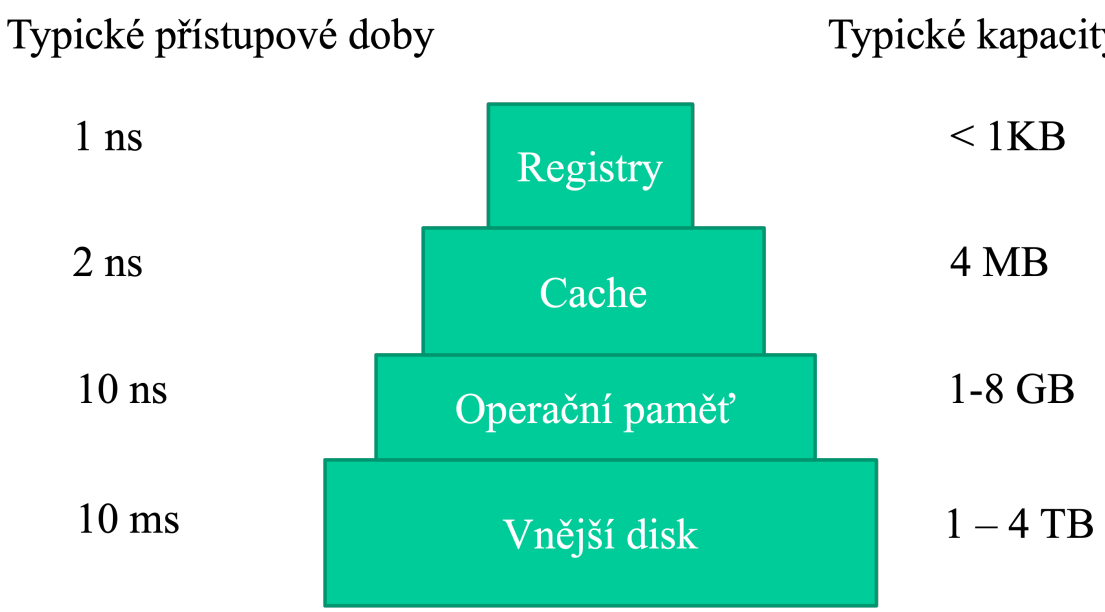


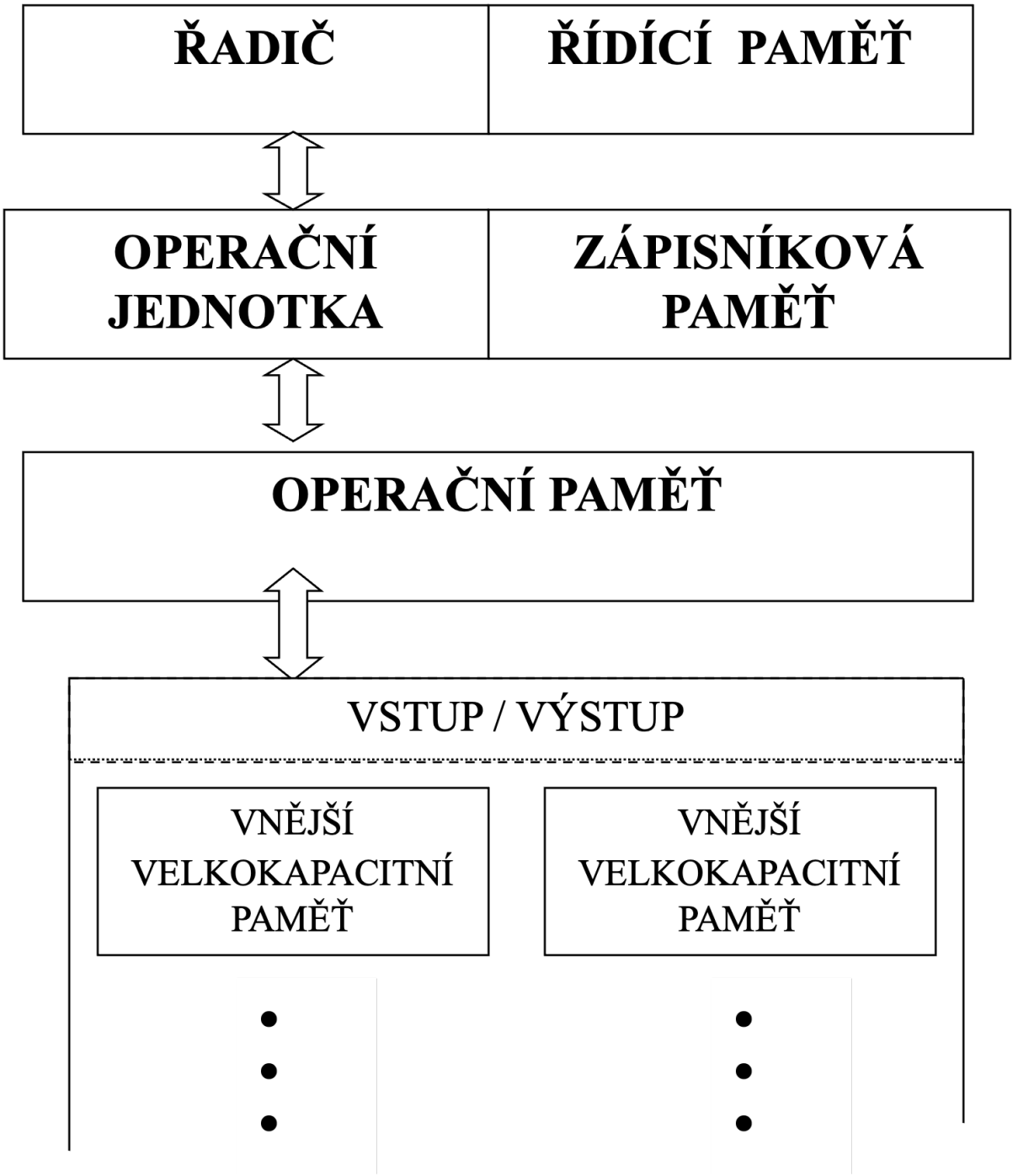
- Každá adresovatelná buňka obsahuje jednu slabiku = byte (bajt)
- Při získání údaje čtením se obsah neporuší
- Lze do ní zapsat novou informaci, čímž se zruší existující informace
- Má jednoznačně přidělenou adresu

Hierarchie pamětí

Úroveň	Paměť	Přístupová doba	Typ / určení
1	reg. CPU	~0,2–0,5 ns	v aritmeticko-logické jednotce, jen stovky bytů
2	cache L1	~0,5–1 ns	na čipu CPU, desítky KiB
3	cache L2	~2–5 ns	na čipu CPU, stovky KiB až jednotky MiB
4	cache L3	~10–15 ns	sdílená napříč jádry, jednotky až desítky MiB
5	operační paměť (RAM)	~50–100 ns	DRAM, *ztrátová* paměť, jednotky až stovky GiB
6	SSD (SATA/NVMe)	~20–100 µs	trvalá paměť (flash), stovky GiB až desítky TiB

Úroveň	Paměť	Přístupová doba	Typ / určení
7	HDD	~5–10 ms	trvalá magnetická paměť, mechanické rameno
8	magnetická páska	sekundy až desítky sekund	nejlevnější na bajt, jen pro zálohy



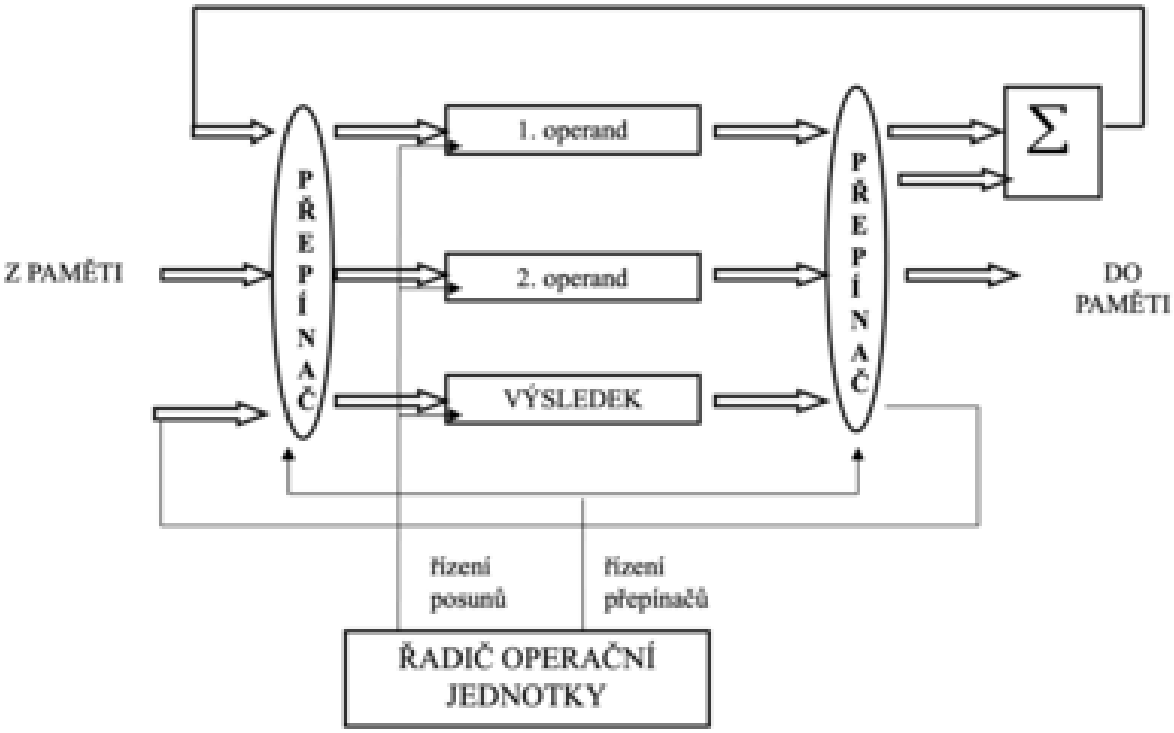

ŘADIČ
ŘÍDÍCÍ PAMĚŤ
OPERAČNÍ JEDNOTKA
ZÁPISNÍKOVÁ PAMĚŤ
OPERAČNÍ PAMĚŤ
VSTUP / VÝSTUP
VNĚJŠÍ VELKOKAPACITNÍ PAMĚŤ
VNĚJŠÍ VELKOKAPACITNÍ PAMĚŤ

Procesor

Procesor (CPU) je *řídicí a výpočetní jádro* počítače: čte z operační paměti instrukce, dekóduje je a provádí, počítá v aritmeticko-logické jednotce a řídí činnost všech ostatních částí počítače. Od paměti se liší tím, že informace nejen uchovává, ale *aktivně zpracovává*, a od V/V zařízení tím, že data nepředává ven do okolního světa, ale je jejich vnitřním "šéfem".

- aritmeticko-logická jednotka
- registry
- řadič

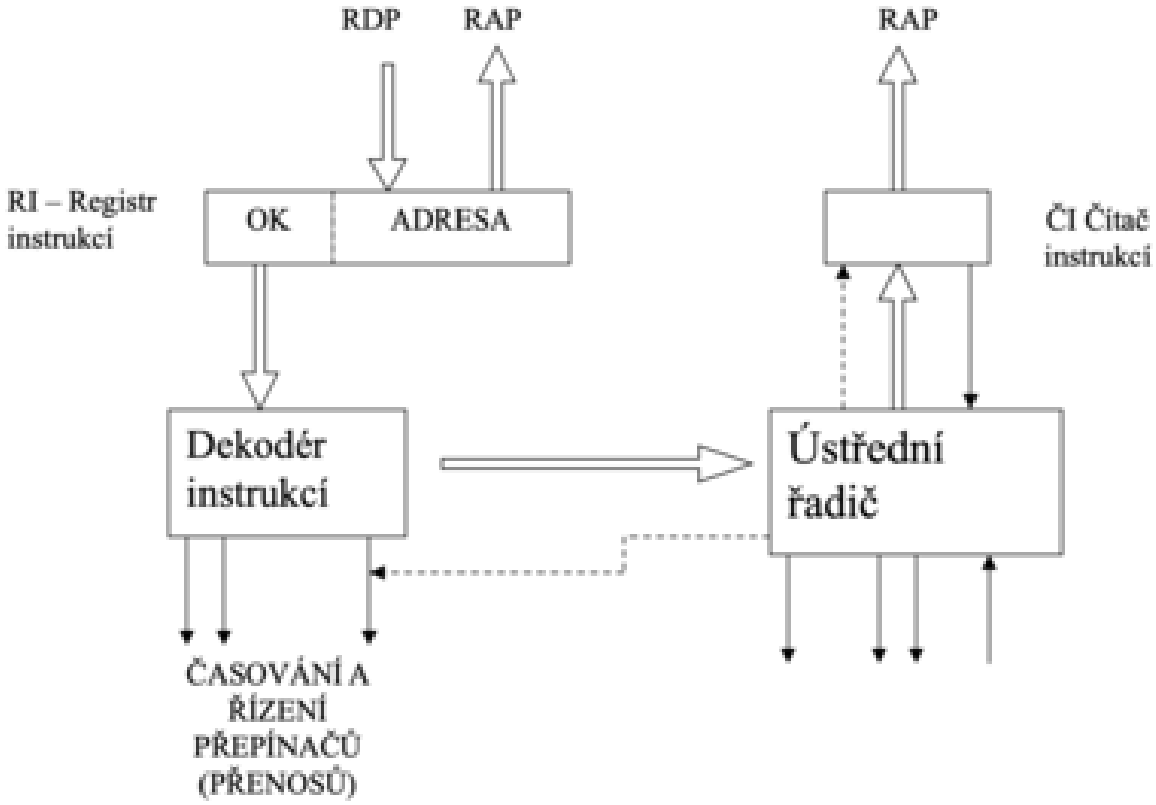
Aritmeticko-logická jednotka (ALU)

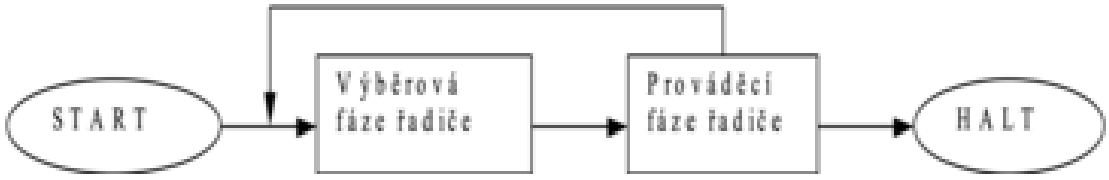


Tradiční možnosti ALU

Sečítání	v přímém kódu
Odčítání	v doplňkovém nebo inverzním kódu
Násobení	postupným sečítáním
Dělení	postupným odčítáním
Složitější operace	metodami numerické matematiky

Řadič a základní instrukční cyklus





Instrukční cyklus řadiče (1)

Výběrová fáze

1. počáteční nastavení čítače instrukcí
2. načtení instrukce do RI
   - ČI → adresa RAP
   - Načtení instrukce do RDP
   - RDP → RI

Vysvětlivky zkratek

RI

registr instrukce

RAP

registr adresy paměti

RDP

registr dat paměti

ČI

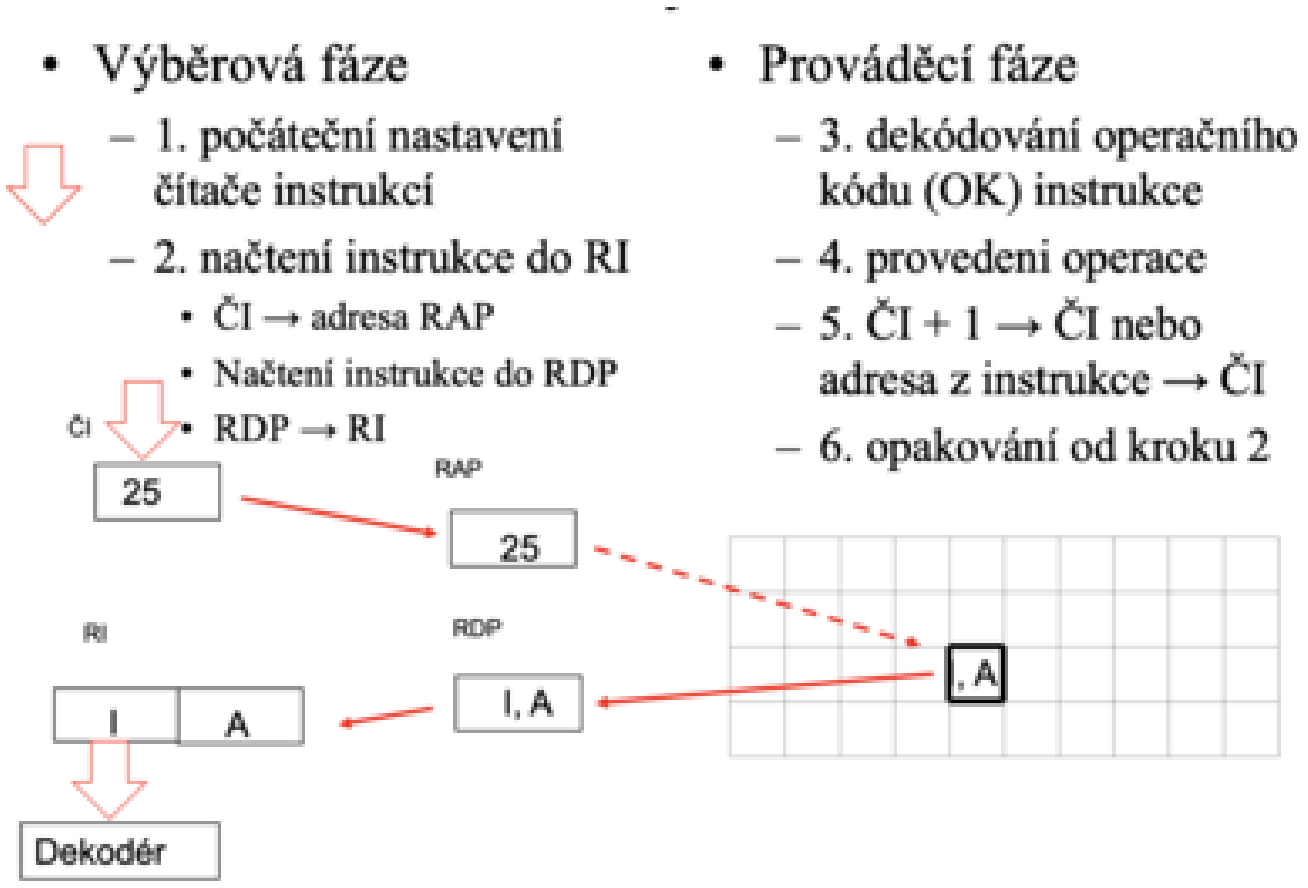
čítač instrukcí (program counter, PC nebo instruction pointer, IP)

Instrukční cyklus řadiče (2)

Prováděcí fáze

1. dekódování operačního kódu (OK) instrukce
2. provedeni operace
3. ČI + 1 → ČI nebo adresa z instrukce → ČI
4. opakování od kroku 2

Instrukční cyklus (2)



Metody určování (adresování) operandů (1)

Přímé adresování

`mov eax, [0x12345678]` načte do `eax` hodnotu z paměťové buňky s adresou `0x12345678`

Přímý operand

`mov eax, 0x12345678` načte do `eax` přímo hodnotu `0x12345678`

Nepřímé adresování

`mov eax, [ebx]` načte do `eax` hodnotu z paměťové buňky, jejíž adresa je v registru `ebx`

Implicitní adresování

`push eax` uloží obsah `eax` na vrchol zásobníku, jehož adresa je implicitně, bez přímého uvedení,dána v registru `esp`



Uvedené módy jsou ukázány na příkladu AMD64 (Intel x64) architektury.

Metody určování (adresování) operandů (2)

Implicitní operand

`inc eax` zvýší obsah registru `eax` o 1, tj. operand `1` je implicitně určen operačním kódem

Modifikované adresování

`mov eax, [ebx+4]` adresa paměťové buňky je dána součtem obsahu registru `ebx` a konstanty `4`

Relativní adresování

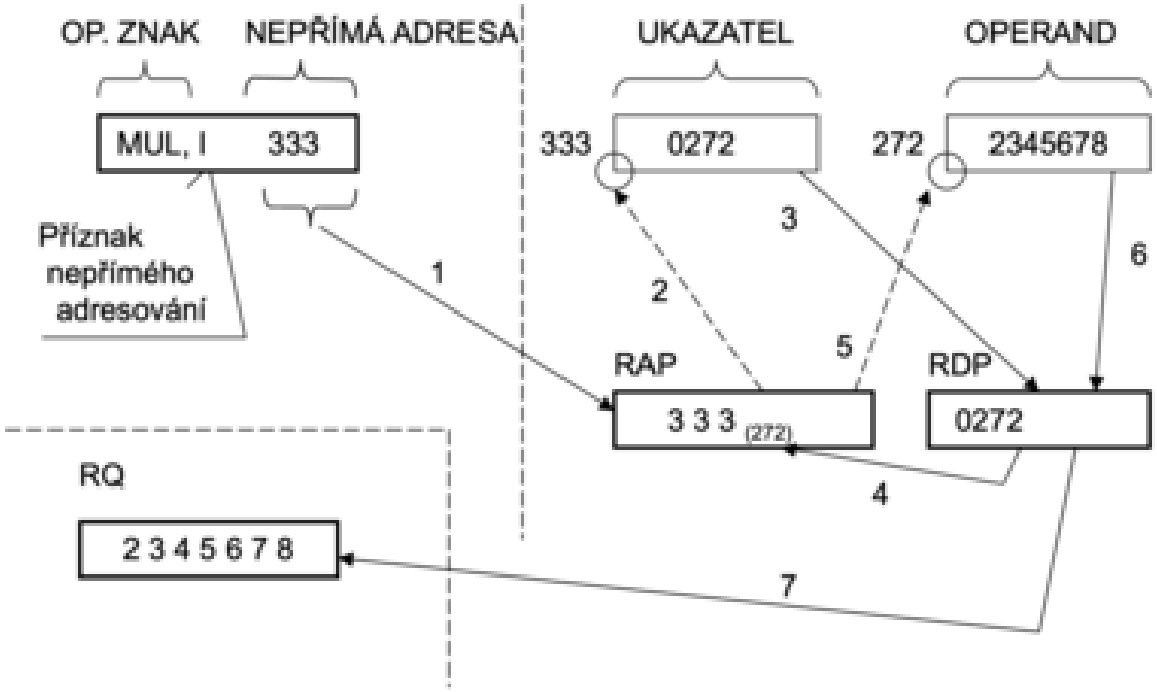
`jmp short +0x1E` skok na adresu, která je o `0x1E` vyšší než adresa následující instrukce

Asociativní adresování

používá se v pamětech typu CAM (Content Addressable Memory), kdy se vyhledává podle obsahu, nikoli podle adresy. Instrukce na to v AMD64 nejsou, vyhledává se přímo hardwarem.

Získání operandu

- Nepřímé adresování



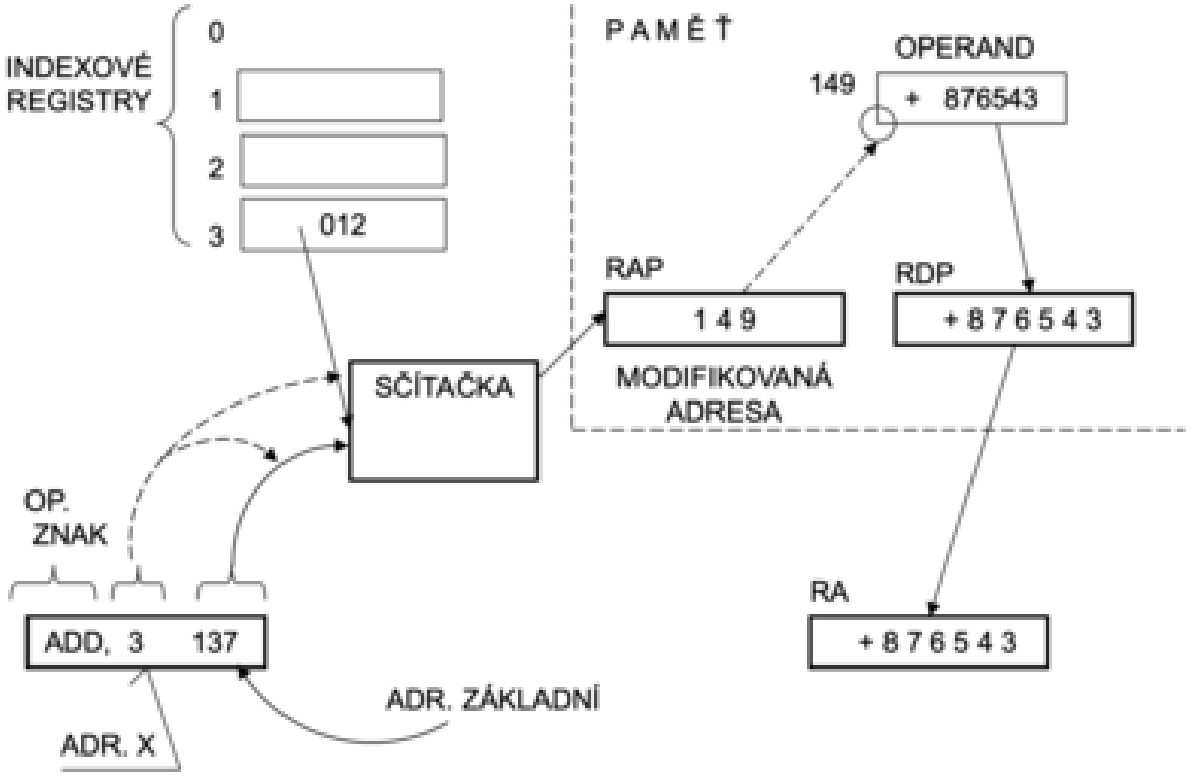
- Přímé adresování

REGISTR OPERAČNÍ JEDNOTKY URČEN IMPLICITNĚ OPERAČNÍM ZNAKEM

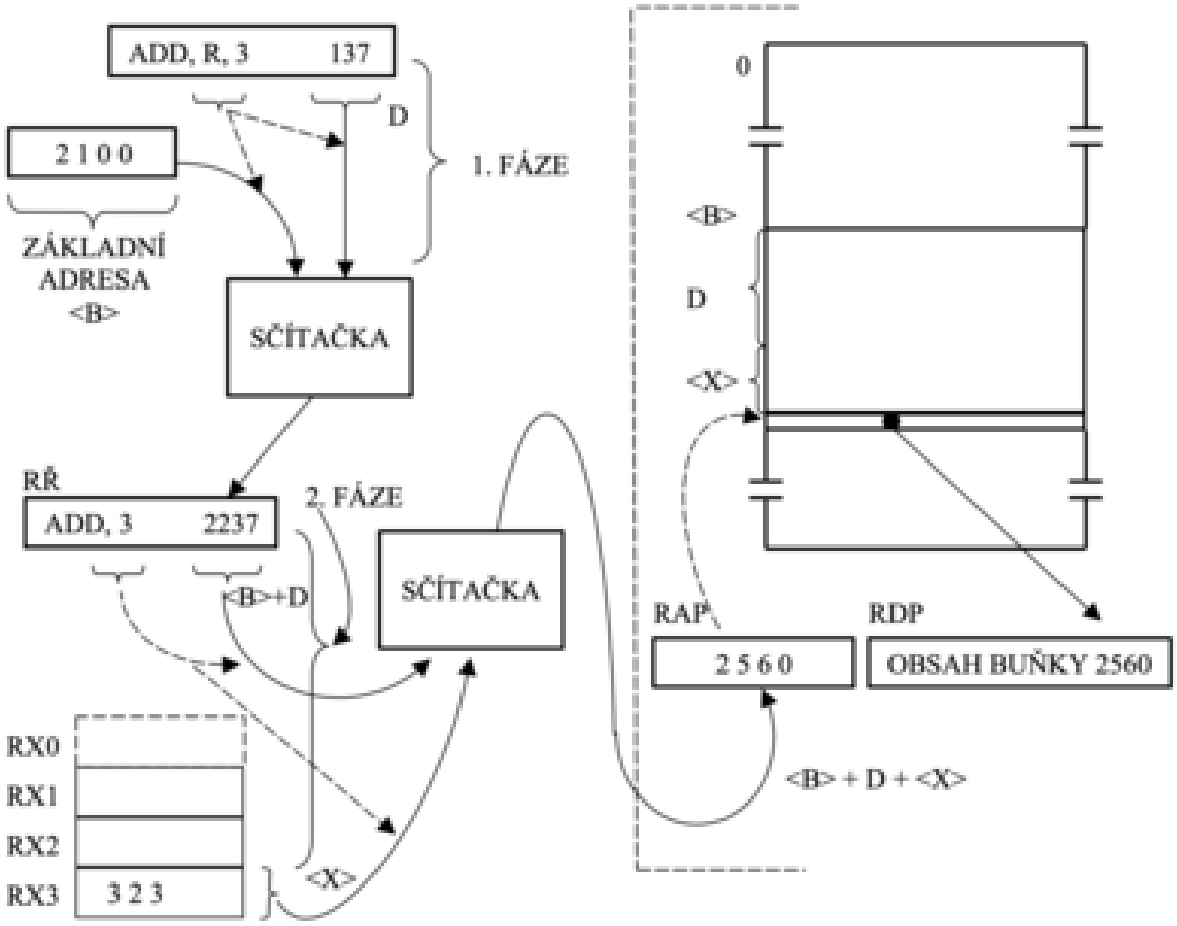
Implicitní adresování a implicitní operand

- Práce s operandy, které jsou vždy na stejném místě — např. přesun mezi registry
- Implicitní operand, kdy operand je určen operačním kódem — např. násobení konstantou 2, zaokrouhlování …

Modifikace adresy



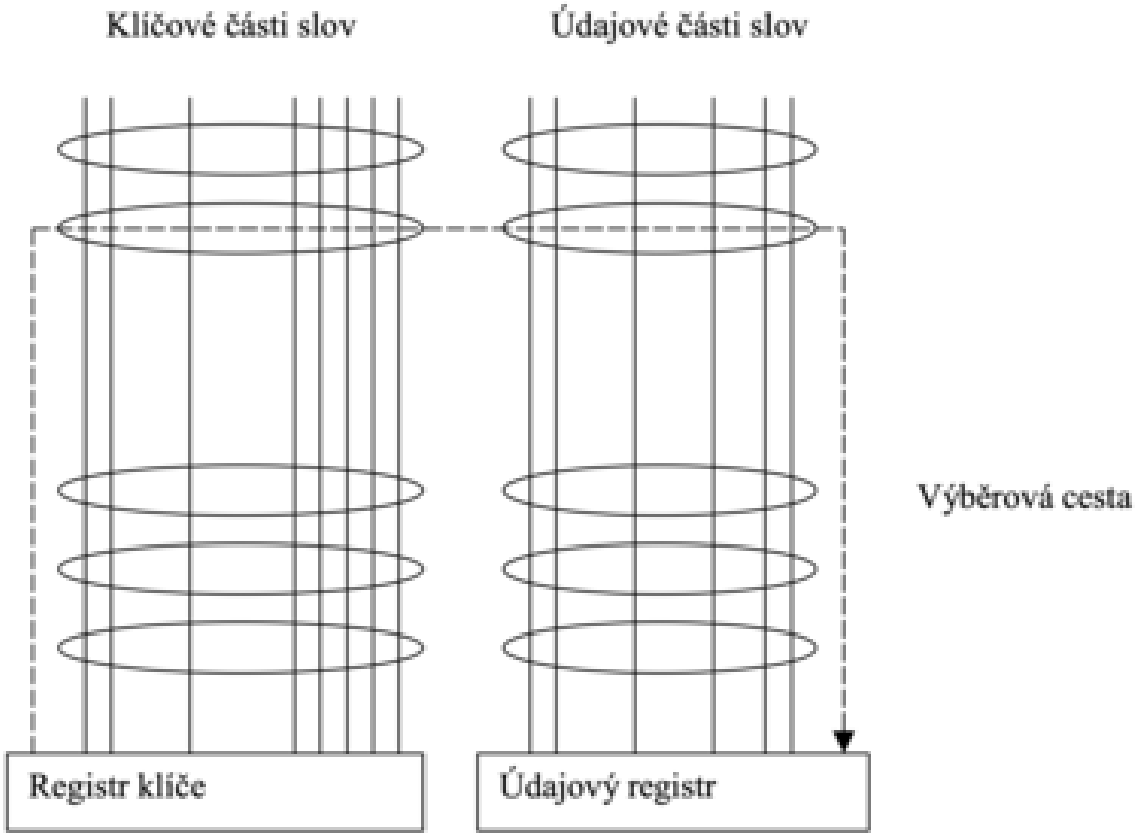
Obr. 4. Modifikace základní adresy



Obr. 5. Modifikace relativní adresy

Adresování obsahem (asociativní)

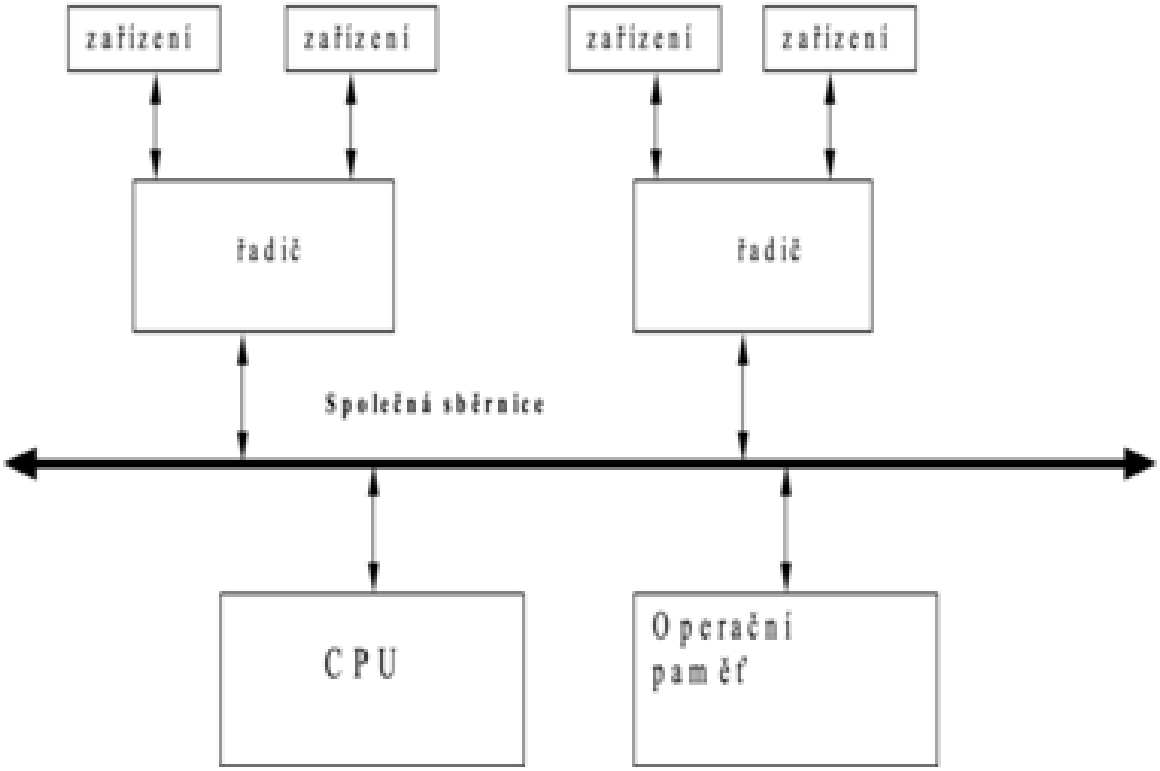
- Ukládání a výběr informace na základě obsahu
- Označuje se jako *paralelní vyhledávání*, tj. porovnání všech paměťových míst



Obr. 6. Princip asociativní paměti

Vstupné a výstupní zařízení

Komunikace V/V zařízení

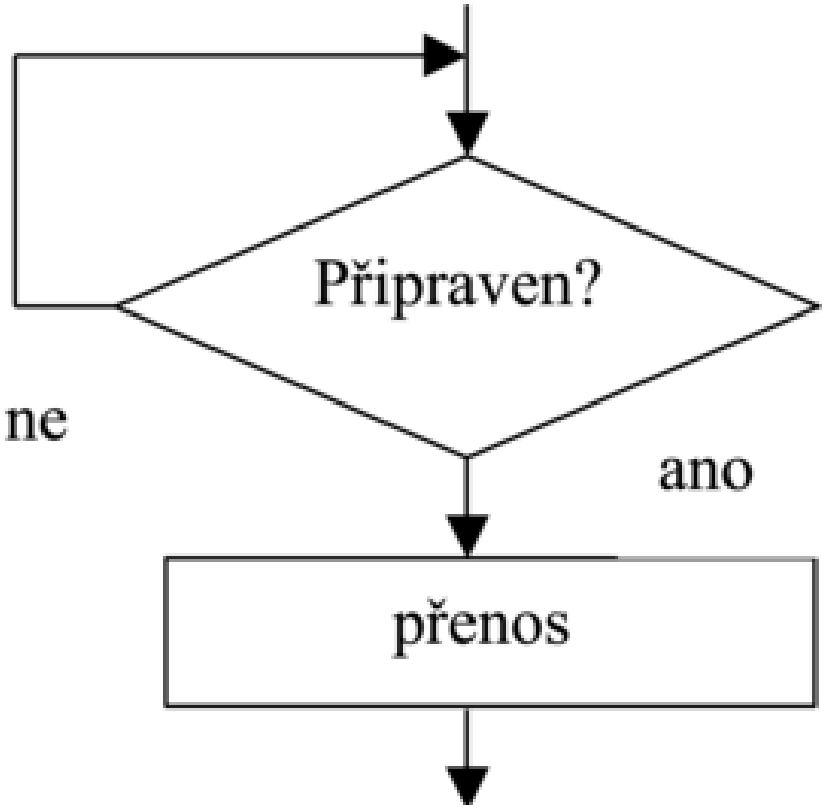


Způsoby komunikace procesoru s okolím

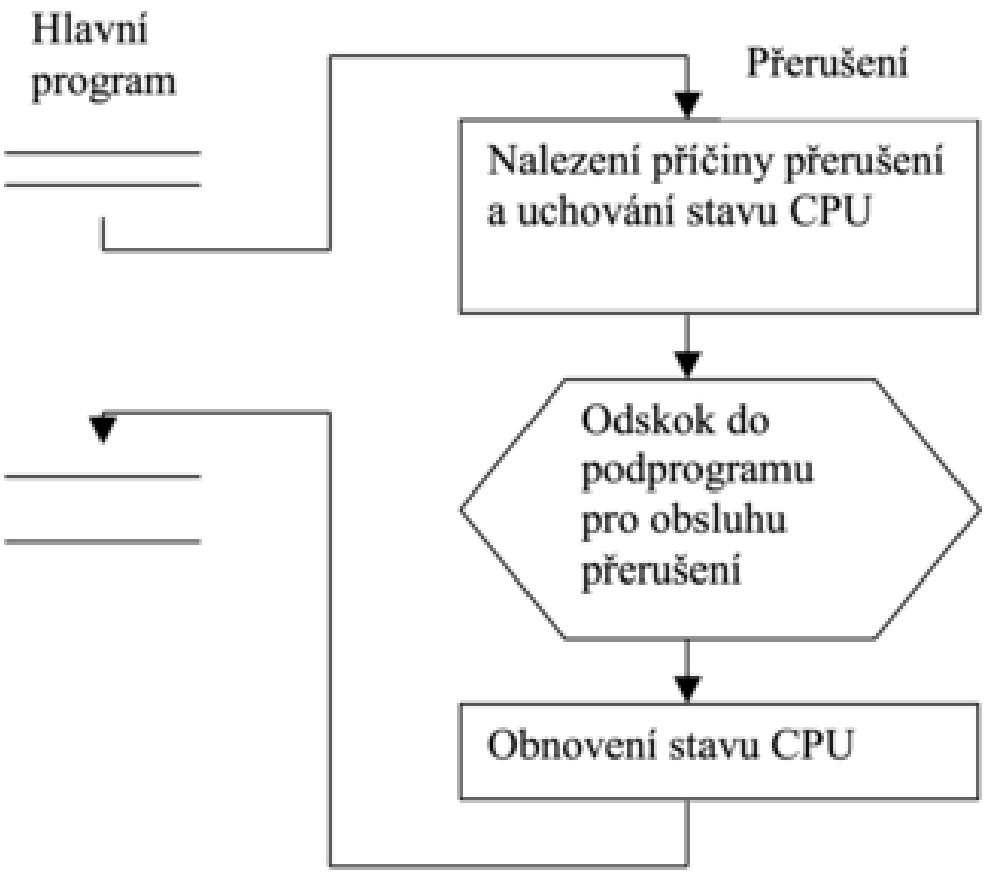
- Programovou smyčkou
- Přerušením
- DMA

Přenos dat

Podmíněný přenos dat



Přenos dat systémem přerušení



Kanály

DMA kanály

- Pro kopírování bloků dat mezi pamětí a I/O zařízením.
- Je třeba je naprogramovat zápisem do hardwarových registrů.

Specializované I/O procesory

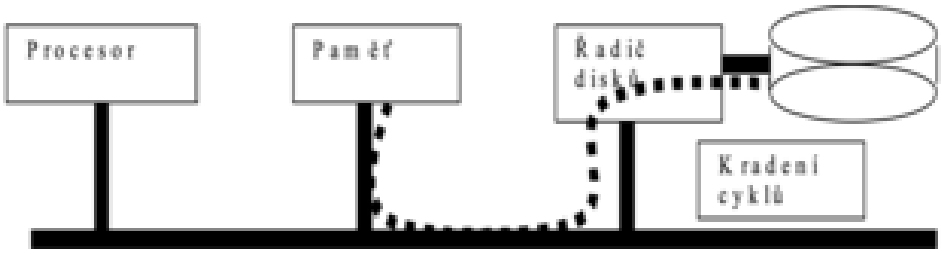
- Jsou řízeny posloupností vlastních instrukcí (tzv. kanálovým programem).

selektorové
: obsluhuje 1 rychlé zařízení (mg. pásky)

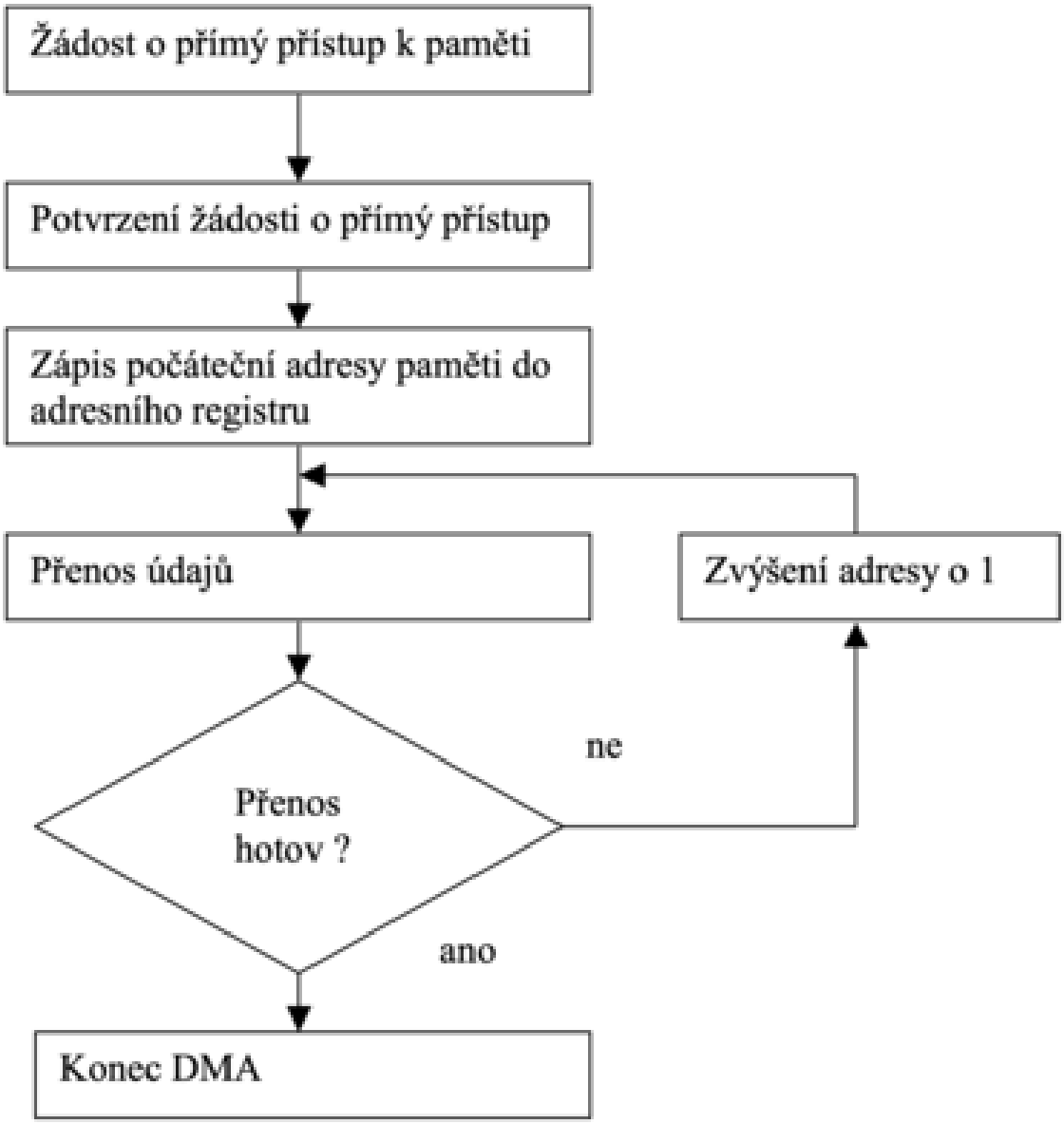
multiplexní
: mohou obsluhovat několik pomalých zařízení (tiskárny, někt. terminály, děr. pásky)

Direct Memory Access

Direct Memory Access



Vývojový diagram DMA


Žádost o přímý přístup k paměti
Potvrzení žádosti o přímý přístup
Zápis počáteční adresy paměti do adresního registru
Přenos údajů
Zvýšení adresy o 1
Přenos hotov ?
ne
ano
Konec DMA

Odlišná schémata počítačů

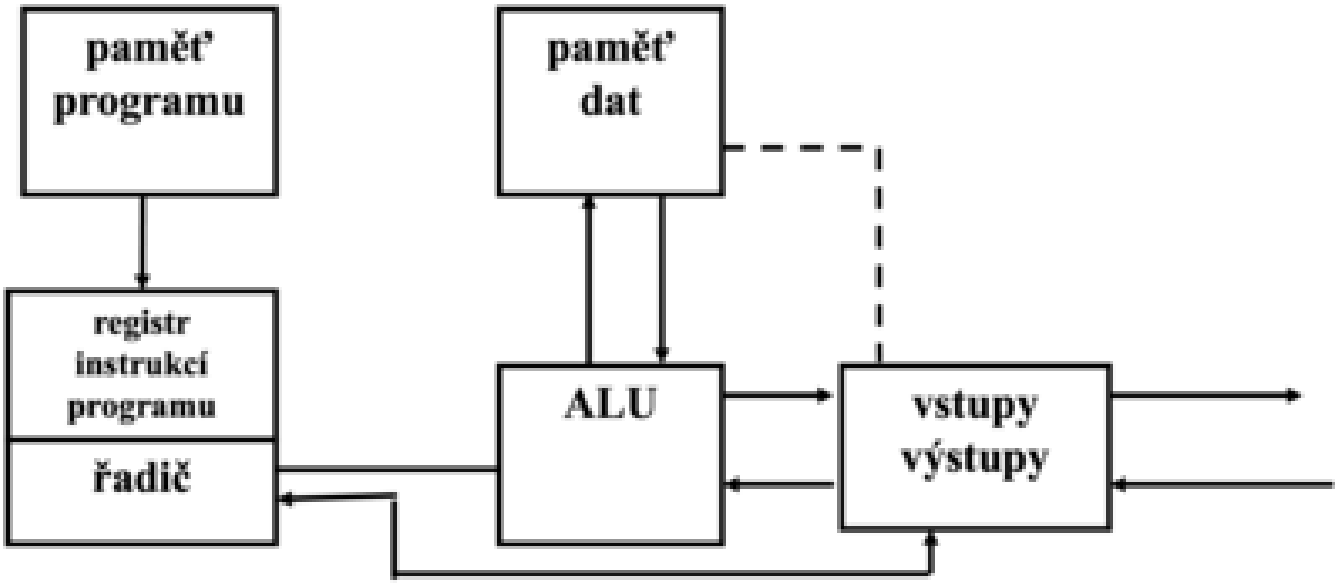
Hardvardské schéma počítače

- oddělená paměť pro program a pro data (některé jednočipové mikropočítače)

Víceprocesorové počítače

- počítače s několika CPU. Dělí se podle toho, zda mají sdílenou paměť:
  - multiprocessors (multiprocesory) mají sdílenou paměť
  - multicomputers (multipočítače) nemají sdílenou paměť, procesory komunikují například pomocí mechanismu zasílání zpráv

Harvardská koncepce



Flynnova klasifikace víceprocesorových systémů

Rozdělení počítačů podle počtu instrukčních a datových proudů:

SISD *(single instruction, single data)* - běžné jednoprocesorové počítače

SIMD *(single instruction, multiple data)* jedna instrukce se provádí na větší množství dat (například sčítání dvou vektorů) - tzv. vektorové počítače, některé superpočítače jsou SIMD

MSIMD *(Multiple Single Instruction stream, Multiple Data stream)* – několik nezávislých podsystémů, každý provádí jiný program (řízení analogické MIMD)

Flynnova klasifikace (2)

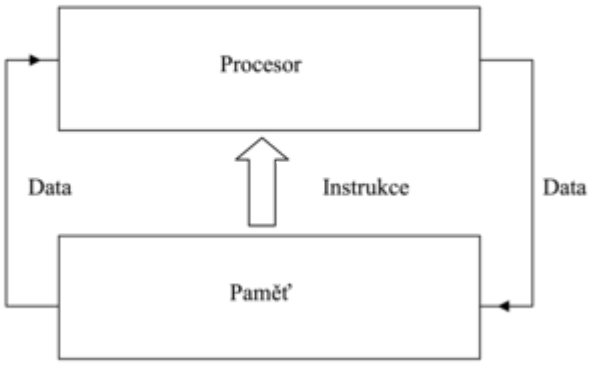
SPMD Same Program, Multiple Data stream — všechny procesory provádějí stejný program, ale nezávisle na sobě — bez synchronizace (proto každý procesor má svůj řadič)

MISD (multiple instruction, single data) — nerealizovatelné

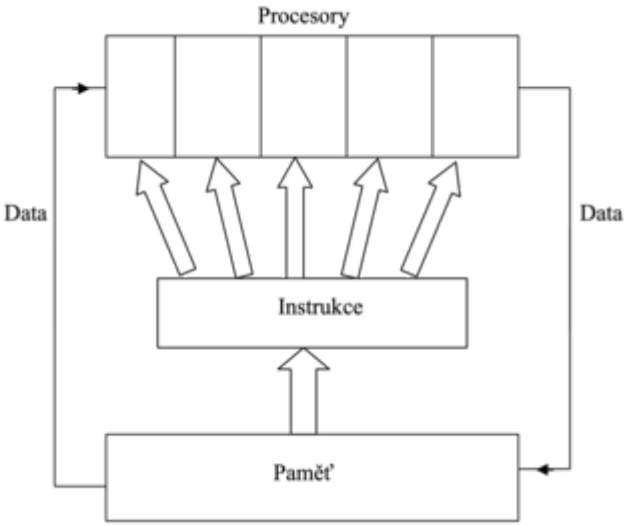
MIMD (multiple instruction, multiple data) — víceprocesorové systémy; podle toho zda mají nebo nemají sdílenou paměť se rozdělují na multiprocesory (se sdílenou pamětí) a multipočítače (multicomputers - spolupracující počítače propojené sítí)

Procesory se single data (SISD a MISD)

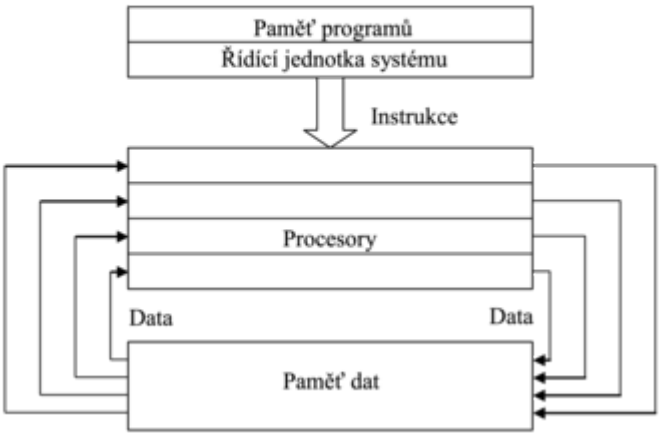
Procesor se single instruction, single data (SISD)



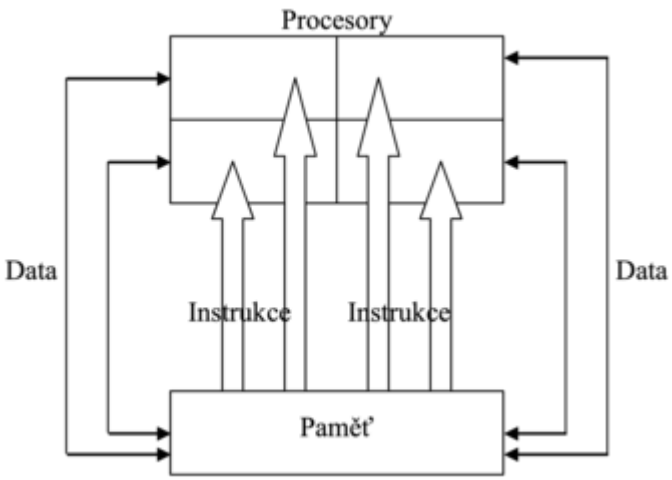
Procesor s multiple instruction, single data (MISD)



Maticový procesor se single instruction, multiple data (SIMD)

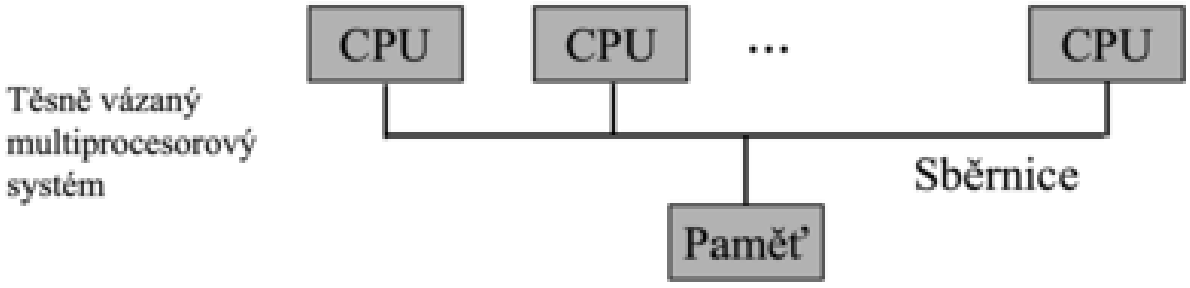


Multiprocesor s multiple instruction, multiple data (MIMD)



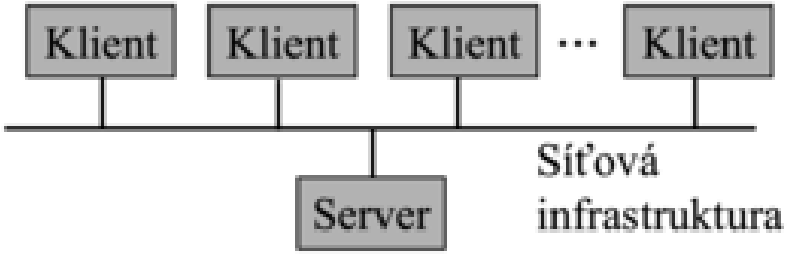
Paralelní a distribuované systémy

Paralelní a distribuované systémy



Distribuovaný systém klient-server

Distribuovaný systém typu klient-server



📖 Rozšiřující téma

- Není součástí zkoušky, ale asi patří k dobrému tónu to znát…

Co říká počet bitů u procesoru: 8, 32, 64, ...?

- v zásadě šířka (počet bitů) *datové sběrnice*
- kolik bitů lze *najednou přenášet* z/do procesoru
- obvykle i základní bitová velikost zpracovávaných čísel

64-bitové: počítače

- většina soudobých počítačů je již 64bitová
- instrukční sady **ARM64** (Apple Silicon), **AMD64** a.k.a. **x64** nebo také **x86_64** (Intel a AMD)
- základní bitová velikost zpracovávaných čísel je 64 bitů

Charakteristické parametry:

- operace čtení a zápisu mezi procesorem a pamětí jsou fyzicky alespoň 64bitové
- bitová šířka datové sběrnice je aspoň 64 bitů
- z hlediska rychlosti je tedy lhostejné, zda sčítáme 8bitová nebo 64bitová čísla

32-bitové: kontroléry

- vestavěná zařízení ("chytré krabičky")
- Např. méně výkonná Raspberry Pi Pico 2 a *Pico 2 W* (2024)
  - Mikrokontrolér: RP2350
  - CPU: Dvoujádrový *32bitový* Arm Cortex-M33 nebo RISC-V
- výkonnější Raspberry Pi jsou už *64bitové*
- 32 bitů je *minimum* pro Javu a většinu dnešních operačních systémů

8-bitové: "hračky"

- Dnes: Arduino Uno, Nano s *8bitovými* procesory ATmega328P
- Stále používané 8bitové CPU jako Zilog Z80, MOS 6502 a kompatibilní
- Stály na nich extrémně populární domácí počítače ZX Spectrum, Sharp MZ-800, Commodore C64 a Atari 800 XL
- "Legacy" *průmyslové řídicí systémy*
- herní *retro konzole*

128-bitové?

Příliš mnoho!

- jako samostatné "komoditní" komponenty je moc nemá smysl vyrábět:
- vnější obvodové řešení by bylo příliš složité

- uvažme jen nutný počet datových pinů takového čipu
- uvnitř čipů Apple Silicon jsou šířky sběrnic mezi CPU a GPU či pamětí i větší: 256 u variant *M2 Pro*
- větší šířka datového toku stejně má smysl u typově jiných zařízení, než univerzálních CPU: třeba u **GPU**

Proč ne méně než 8 bitů?

- osmibitové procesory uměly pohánět jednoduché počítače na hry, hudbu, psaní textů…
- 8 bitů je už jeden znak
- čtyřbitové tak maximálně ovládat kávovar, nejvýš kalkulačku
- 4 bity je jen jedina cifra (0-9)

Na další kapitolu: Logické obvody, kombinační logika

HTML · Slides · PDF | Přehled předmětu