

B35APO: Architektury počítačů

Lekce 01. Úvod

Pavel Píša

pisa@fel.cvut.cz

Petr Štěpán

stepan@fel.cvut.cz



11. února, 2026

Obsah

1 Úvod

2 Složení počítače

3 Logické obvody - Opakování PSY

Motivace

Co můžete udělat pro zrychlení Vašeho programu:

- Použít výkonnější počítač:
 - Zvýšit výkon CPU
 - Frekvence CPU
 - Efektivita CPU - kolik operací stihne provést za 1 takt procesoru
- Změnit program:
 - Zlepšit efektivitu práce s pamětí
 - Paralelizovat program:
 - Zvýšit počet jader CPU

Motivace

Má paralelizace nějaká omezení:

- Nikdy nelze paralelizovat celý program
- Amdahlův zákon
 - α procent programu nelze paralelizovat
 - zbytek programu $1 - \alpha$ lze na p procesorech zrychlit na $\frac{1-\alpha}{p}$
 - Poměr zrychlení na p procesorech $S(p) = \frac{\alpha + 1 - \alpha}{\alpha + \frac{1-\alpha}{p}} = \frac{p}{1 + \alpha \cdot (p-1)}$
 - Limit zrychlení je pro nekonečně procesorů

$$S(p \rightarrow \infty) = \lim_{p \rightarrow \infty} \frac{p}{1 + \alpha \cdot (p-1)} = \frac{1}{\alpha}$$
 - Např pro $\alpha = 0.3$ je limit $S(p \rightarrow \infty) = 3.\bar{3}$, tedy program nelze zrychlit více než $3.\bar{3}$ krát.
- V praxi, čím více máte procesů, tím stoupá i náročnost přípravy dat pro paralelizaci.

Motivace

Proč studovat architektury počítačů:

- Naučit se jak pracuje počítač, který vykonává Váš program a kde jsou možnosti program zefektivnit
 - zjistit, v čem současný počítač zpomaluje výpočet (rychlost procesoru, velikost vyrovnávací paměti, latence hlavní paměti, počet výpočetních jader) a otestovat jiný HW
 - zjistit, zda lze program upravit, aby využíval lépe dostupné prostředky
 - upravit pořadí přístupu do paměti a tím lépe využívat vyrovnávací paměť
 - upravit program, aby požíval méně skoků a tím se vykonával rychleji
 - paralelizovat výpočet, využít specializovaný HW - GPU, externí výpočetní jednotku např. Coral USB nebo Intel Neural Compute Stick 2.
- Poptávka po absolventech kombinující umělou inteligenci a vestavné systémy (embedded system)
- Pokud je pro Vás počítač BlackBox, pak jsou Vaše programy neefektivní

Obsah přednášek

Projdeme si všechny základní součásti počítače:

- CPU
- Hierarchie paměti - Cache/RAM/Disk
- Vstupy a výstupy - I/O klávesnice, myš, obrazovka, síťová karta, ovladače HW
- Výjimky a přerušení - efektivní spolupráce CPU s HW

Motivace proč navštěvovat přednášky:

- Něco se naučíte a budete to mít lehčí při přípravě na zkoušku
- Kdo správně zodpoví kvízovou otázku v závěru přednášky, dostane bod aktivity
 - Kvízy aktivity na přednáškách budou náhodně, první bodovaný kvíz příští týden
 - Celkem 8 bodů za aktivitu na přednáškách

Náplň cvičení

- 4 menší domácí úkoly - 36 bodů
 - 2 programy v C
 - 2 formuláře
 - alespoň 3 úlohy ze 4
- Semestrální úloha - 24 bodů
 - Týmový projekt - dvojice, nebo jednotlivci
 - Speciální HW MZ APO deska
- Nepovinné úlohy nebo aktivita při cvičení/přednáškách - 10 bodů

Známka	Body
A	≥ 90
B	80 – 89.9
C	70 – 79.9
D	60 – 69.9
E	50 – 59.9
F	< 50

Zkouška:

- písemný test 30 bodů, min 15 bodů
- ústní \pm 10 bodů

Navazující předměty

Pokud Vás tento předmět zaujme, tak na něj navazují tyto předměty:

- B4M35PAP - Pokročilé architektury počítačů
- B3B38VSY - Vestavné systémy
- B4M38AVS - Aplikace vestavných systémů
- B4B35OSY - Operační systémy
- B0B35LSP - Logické systémy a procesory

Materiály k předmětu

- PATTERSON, David A. a John L. HENNESSY. Computer organization and design RISC-V edition: the hardware/software interface. Second Edition. Cambridge: Elsevier, [2021]. ISBN 978-0-12-820331-6. (12 kusů v ústřední knihovně ČVUT)
- web:
 - <https://cw.fel.cvut.cz/b192/courses/b35apo/>
 - <https://dcenet.felk.cvut.cz/apo/>
- Kurzy v angličtině:
 - MIT 6.004/6.191 – Computation Structures
 - Computation Structures | Electrical Engineering and Computer Science | MIT OpenCourseWare (2015)
 - Computer System Architecture | Electrical Engineering and Computer Science | MIT OpenCourseWare (2005)
- Kurzy v češtině:
 - <https://courses.fit.cvut.cz/BI-APS/>
 - <https://www.vut.cz/studenti/predmety/detail/218515?apid=218515>

Obsah

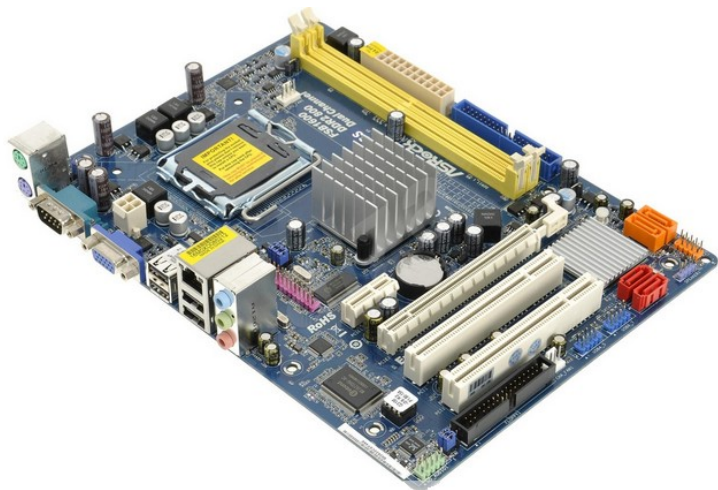
1 Úvod

2 Složení počítače

3 Logické obvody - Opakování PSY

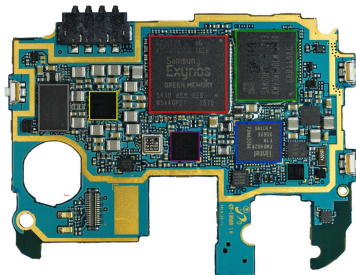
Co je uvnitř počítače

Základní deska počítače:



Co je uvnitř počítače

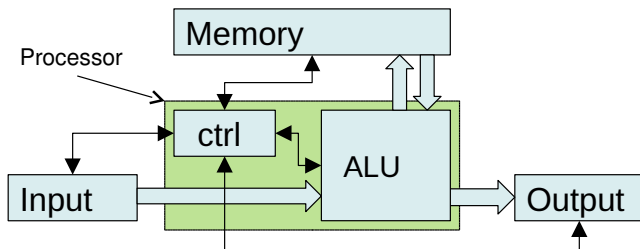
Rozebraný telefon:



von Neumannova architektura

Společný koncept navržený maďarským fyzikem Johnem von Neumannem (1903-1957) obsahuje:

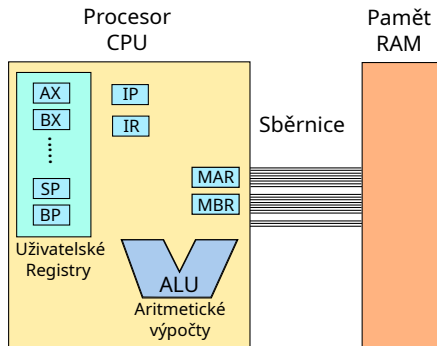
- Procesor - Central Processing Unit - CPU
- Paměť - Memory, Random-access Memory,
- Vstup/Výstup - Input/Output



Základem je jedna paměť použitá jak pro program, tak pro data.

Processor - CPU

- CPU obsahuje uživatelské registry, které obsahují 16, 32 nebo 64 bitové hodnoty podle architektury čipu
- CPU načítá z paměti instrukce tak jak jdou za sebou a každou načtenou instrukci vykoná
- Adresa právě vykonávané instrukce je ve speciálním registru PC nebo IP
- ALU část CPU která umí sčítat, odčítat, násobit, dělit a další aritmetické operace



Paměť

Paměť si pamatuje data - bajty, slova.

Pokud už znáte nějaký programovací jazyk, tak si ji můžete představit jako pole, např. v jazyce C jako:

```
unsigned char RAM[16 * 1024 * 1024 *1024]; // 16GiB RAM
```

Z paměti lze číst, například zde do 32-bitového registru R10:

```
registr R10 = RAM[adresa] | (RAM[adresa+1]<<8)
             | (RAM[adresa+2]<<16) | (RAM[adresa+3]<<24);
```

nebo zapsat registr R10 do paměti:

```
RAM[adresa] = R10 & 0xFF; RAM[adresa+1] = (R10>>8) & 0xFF;
RAM[adresa+2] = (R10>>16) & 0xFF;
RAM[adresa+3] = (R10>>24) & 0xFF;
```

Paměť

Big endian čtení z paměti probíhá následujícím způsobem:

```
registr R10 = RAM[adresa+3] | (RAM[adresa+2]<<8)
              | (RAM[adresa+1]<<16) | (RAM[adresa]<<24);
```

nebo zápis do paměti takto:

```
RAM[adresa+3] = R10 & 0xFF; RAM[adresa+2] = (R10>>8) & 0xFF;
RAM[adresa+1] = (R10>>16) & 0xFF;
RAM[adresa] = (R10>>24) & 0xFF;
```

Některé starší procesory umožňovali pouze čtení ze zarovnané adresy, tedy z adresy $\text{adresa}=4*x$.

Pokud se bude číst z adresy, která není násobkem 4 pak se přečtou dvě slova a z nich se sestaví výsledek.

Instrukce procesoru

- Instrukce procesoru jsou jediné činnosti, které procesor umí vykonat
- Základní instrukce jsou:
 - uložit konstantu do registru
 - načíst data z paměti do registru
 - provést matematickou operaci s registry a výsledek uložit do registru
 - uložit data z registru do paměti
 - porovnat hodnotu dvou registrů
 - podle výsledku porovnání provádět jiné instrukce (změnit PC na jinou hodnotu než je následující instrukce = provést skok v programu)
- veškeré programy, ať už vytvořené v jazyku C, Python, programy provádějící i velmi komplexní výpočty jsou složeny pouze z těchto jednoduchých instrukcí procesoru

Instrukce procesoru

Instruction Set Architecture (ISA) Architektura souboru instrukcí

- je kompletní instrukční sada včetně adresních módů
- např. x86 (IA-32), x86-64 (AMD64, EM64T, IA-32e), ARM32, ARM64, AVR, MIPS, RISC-V
- ISA obsahuje:
 - množinu strojových instrukcí procesoru
 - podporované datové typy (celá čísla, celá čísla se znaménkem, reálná čísla)
 - množinu registrů
 - adresní módy
 - organizace paměti

Instrukce procesoru

Dva základní přístupy k návrhu ISA

RISC

Reduced Instruction Set Computer

- Obvykle menší počet instrukcí
- Všechny instrukce mají stejný počet bajtů, některé umožňují poloviční délku
- Méně jednodušších adresních módů
- Matematické operace ALU pouze s registry

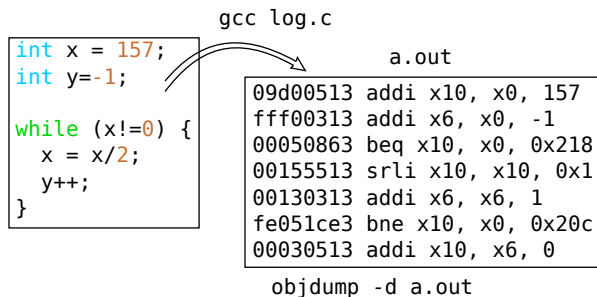
CISC

Complex Instruction Set Computer

- Obvykle větší počet instrukcí
- Délka instrukcí je od 1 bajtu do např. 12 bajtů, nejčastější instrukce jsou nejkratší
- Obvykle mnoho i složitých adresních módů
- Matematické operace ALU i s hodnotami z paměti

Překlad programu

Jak je možné, že jste o strojových instrukcích ještě neslyšeli?



- Programování v assembleru (jazyku symbolických adres) je nekomfortní.
- Překladač přeloží vyšší programovací jazyk přímo do strojových instrukcí procesoru
- Křížový překlad je, když přeložíte program pro jiný typ procesoru, než na kterém probíhá překlad.

Processor

Z čeho se vlastně procesor skládá?

- Možná jste někdy slyšeli, že procesor obsahuje třeba 16 miliard tranzistorů
- V roce 1965 spoluzakladatel společnosti Intel Gordon Moor formuloval zákon:
 - Počet tranzistorů, které mohou být umístěny na integrovaný obvod se při zachování stejné ceny zhruba každých 18 měsíců zdvojnásobí.
- Více méně platí až do dnes, i když už se dostáváme na hranice fyzikálních možností.

K čemu potřebujeme tranzistory v procesoru (CPU)?

- Abychom implementovali Booleovu algebru.

Obsah

1 Úvod

2 Složení počítače

3 Logické obvody - Opakování PSY

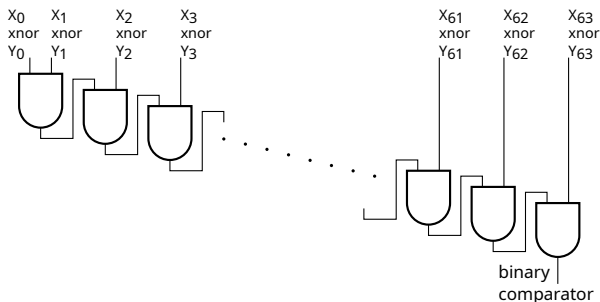
Binární komparátor

Jak udělat komparátor na rovnost?

- Máme dvě 64-bitová čísla X a Y která potřebujeme porovnat
- Porovnání dvou bitů provede operace $xnor$, která má hodnotu 1 pokud dva bity mají stejnou hodnotu.
 - Nyní potřebujeme udělat and 64 logických hodnot.
- Jak to můžeme udělat?

Lineární řešení

Řešení může být lineární:

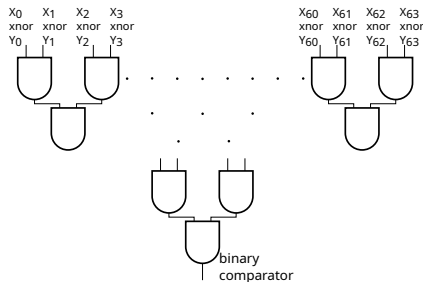


Jaké je to řešení?

- Počet and hradel se dvěma vstupy - 63 hradel
- Rychlost výpočtu?
 - Rychlost výpočtu hradla 4ps, protože na sebe výpočty navazují potřebujeme 63 výpočtů
 - Rychlost výpočtu komparátoru bude 252 ps, což odpovídá frekvenci 4GHz to je velmi pomalé

Logaritmické řešení

Rychlejší řešení využívá paralelního výpočtu všech dvojic a následné spojování do výsledku:



Jaké je to řešení?

- Počet and hradel se dvěma vstupy - 63 hradel
 - $32 + 16 + 8 + 4 + 2 + 1 = 63$
- Rychlost výpočtu?
 - Výpočty v jedné řadě jsou nezávislé a probíhají ve stejnou dobu
 - Kolik řad máme? První řada obsahuje 32 hradel and, druhá 16 and, třetí 8 and, čtvrtá 4 and, pátá 2 and, šestá 1 and
 - Rychlost výpočtu komparátoru bude $6 \cdot 4 = 24$ ps, což je 10 krát rychlejší

Rychlé mocnění

Výpočet mocnin - opakování z PSY:

```
double Exp(double a, int k) {
    if (k == 0) return 1;
    return a * Exp(a, k - 1);
}
```

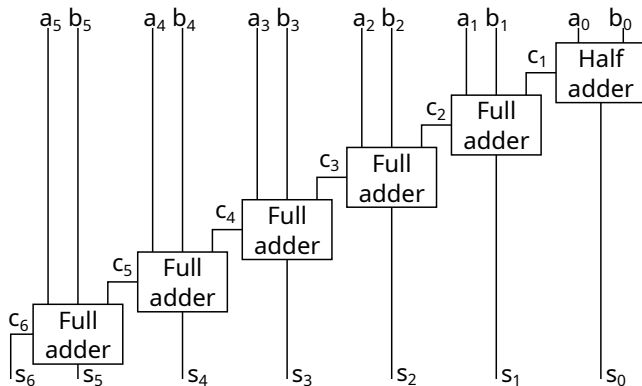
Lineární mocnění - v kryptografii je potřeba vypočítat i výrazy $X^{1000000}$
K tomu je potřeba provést milión násobení.

```
double FastExp(double a, int k) {
    if (k == 0) return 1;
    if (k % 2 == 0) {
        double i = FastExp(a, k / 2);
        return i * i;
    } else {
        return a * FastExp(a, k - 1);
    }
}
```

Logaritmické mocnění - nahradí milión násobení, desítkami násobení.
Logaritmické mocnění je 100000 krát rychlejší.

Sčítání - Ripple Carry Adder - opakování

Lineární sčítačka se jmenuje Ripple Carry Adder (Řetězová sčítačka).

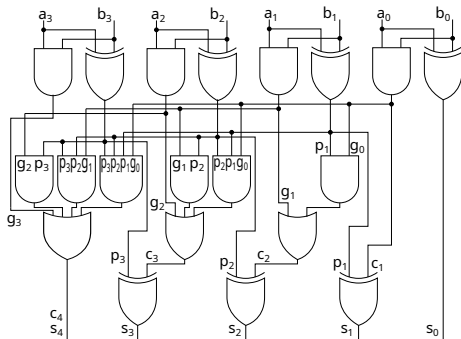


Je obdobně pomalá jako lineární komparátor - vlastně 2-krát tak pomalá než lineární komparátor

Sčítání - Carry Lookahead Adder

Přímý výpočet přenosů – carry na základě hodnot bitů sčítanců, je nejrychlejší sčítačka pro 4-bitová čísla (opakování PSY).

Dvě čísla sečte na 4 zpoždění hradla - teoreticky konstantní doba vzhledem k počtu bitů čísel.



Pro 64 bitová čísla by potřebovala sčítačka kolem 10^{20} tranzistorů, což je moc.

Sčítání - Carry Lookahead Adder

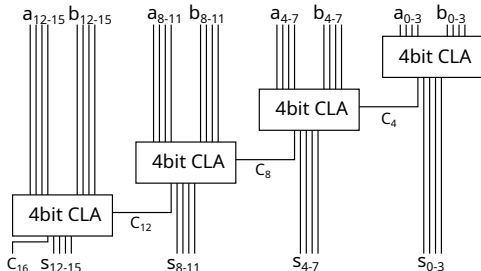
Co tedy s většími čísly, třeba 64-bitovými?

- Upravíme 4-bitovou sčítačku tak, aby mohla přijmout přenos C z předchozích výpočtů.

- POZOR budou se muset přidat hradla pro zpracování carry od sčítaček nižších řádů.

- Takto upravené sčítačky můžeme řetězit.

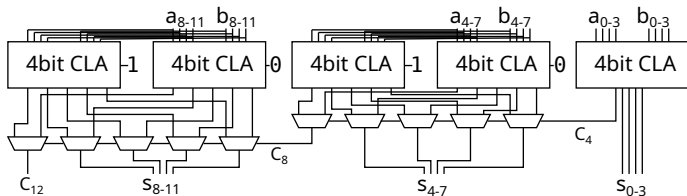
- Zrychlení je pouze o konstantu, závislost na počtu bitů sčítanců je pořád lineární.
 - Rychlost 16-ti bitové sčítačky z obrázku bude 16 zpoždění hradel
 - Rychlost 64-ti bitové sčítačky bude 64 zpoždění hradel, což je 2 krát lepší, než 127 zpoždění u jednoduché Ripple carry adder.



Sčítání - Carry Select Adder

Jiné řešení?

- Víme, že Carry je buď 0 nebo 1
- 4bit CLA zdvojíme a spočteme výsledek pokud vstupní Carry je 0 i 1
- Nakonec řetězíme pouze multiplex, zda skutečně Carry bylo 0 nebo 1



- Sčítačka je rychlejší opět o konstantu krát, místo zpoždění 4 hradel bude řetězit pouze zpoždění 2 hradel pro multiplexor
- Rychlost 16-ti bitové sčítačky z obrázku bude 10 zpoždění hradel
- Rychlost 64-ti bitové sčítačky bude 34 zpoždění hradel, což je 2 krát lepší, než 64 zpoždění u řetězení CLA.

Propagace a generování carry

V PSY jsme si ukazovali, že pokud se zaměříme na jeden sloupeček výpočtu součtu, mohou nastat pouze 3 rozdílné situace:

$$\begin{array}{r}
 C_{i+1} C_i \\
 X_i \\
 Y_i \\
 \hline
 S_i
 \end{array}$$

- Pokud je $X_i = 0$ a $Y_i = 0$, pak musí být $C_{i+1} = 0$
- Pokud je $X_i = 1$ a $Y_i = 0$, nebo $X_i = 0$ a $Y_i = 1$, pak musí být $C_{i+1} = C_i$.
 - Tento případ naveme propagování carry, protože C_{i+1} propaguje hodnotu C_i
 - $P_i = X_i \oplus Y_i$
- Pokud je $X_i = 1$ a $Y_i = 1$, pak musí být $C_{i+1} = 1$.
 - Tento případ naveme generování carry, protože C_{i+1} nezávisí na C_i a má vždy hodnotu 1
 - $G_i = X_i \wedge Y_i$

Propagace a generování carry

Nyní si zkusíme vypočítat hodnotu propagace a generování carry pro skupinu bitů od j do i :

výstupní
carry

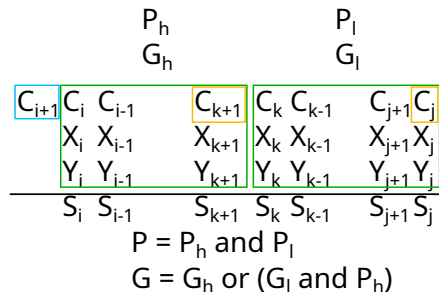
vstupní
carry

C_{i+1}	C_i	C_{i-1}	C_{j+1}	C_j
X_i	X_{i-1}	X_{j+1}	X_j	
Y_i	Y_{i-1}	Y_{j+1}	Y_j	
S_i	S_{i-1}	S_{j+1}	S_j	

- Propagování carry nastane, pokud kombinace hodnot $X_j \dots X_i$ a $Y_j \dots Y_i$ způsobí, že $C_{i+1} = C_j$.
 - Jak toto $P_{j,i}$ spočítat si ukážeme za chvíli
- Generování carry nastane, pokud kombinace hodnot $X_j \dots X_i$ a $Y_j \dots Y_i$ způsobí, že $C_{i+1} = 1$.
 - Tento případ si označíme $G_{j,i}$

Propagace a generování carry

Jak tedy $P_{j,i}$ a $G_{j,i}$ spočítat? Logarithmicky - rozděl a panuj.

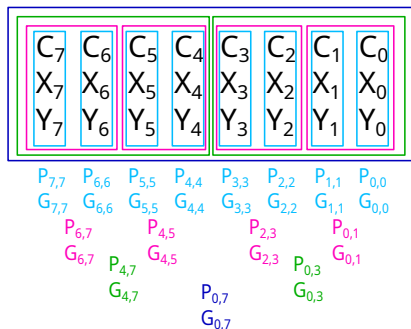


- Hodnoty $G_{i,i} = G_i = X_i \wedge Y_i$ a $P_{i,i} = P_i = X_i \oplus Y_i$
- Nyní si musíme uvědomit, kdy dojde k propagaci carry?
 - Pokud dojde k propagaci carry mezi bity j a k a zároveň dojde k propagaci carry mezi bity $k+1$ a i
 - $P_{j,i} = P_{j,k} \wedge P_{k+1,i}$ tedy $P_{j,i} = P_h \wedge P_l$

- Kdy dojde ke generování carry?
 - Pokud vyšší bity vygenerují carry nebo nižší bity vygenerují carry a vyšší bity ho budou propagovat
 - $G_{j,i} = G_{k+1,i} \vee (G_{j,k} \wedge P_{k+1,i})$ tedy $G_{j,i} = G_h \vee (G_l \wedge P_h)$

Propagace a generování carry

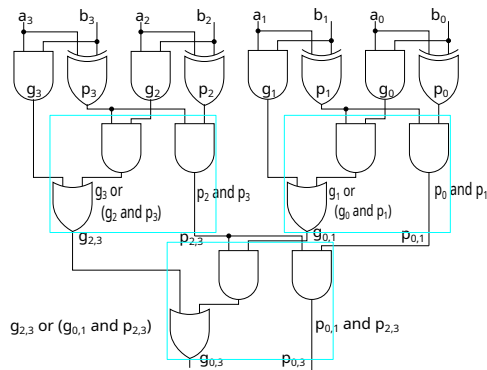
Plán výpočtu hodnot propagace a generování carry.



- V prvním kroku spočteme základní hodnoty propagace a generování pro dvojice X_i, Y_i
- V dalším kroku spočteme hodnoty propagace a generování pro čtveřice $X_{2*i}, X_{2*i+1}, Y_{2*i}, Y_{2*i+1}$
- V každém dalším kroku se vždy spojí dva bloky do jednoho

Sčítání - Carry Lookahead Adder

Takto lze realizovat výše uvedený výpočet:



Doba výpočtu dvojic $G_{i,k}$ a $P_{i,k}$:

- Rychlost 4 bitových sčítanců
5 zpoždění hradel
- Rychlost 8 bitových sčítanců
7 zpoždění hradel
- Rychlost 16 bitových
sčítanců 9 zpoždění hradel
- Rychlost 32 bitových
sčítanců 11 zpoždění hradel
- Rychlost 64 bitových
sčítanců 13 zpoždění hradel

Sčítání - Carry Lookahead Adder

Nyní nám zbývá vypočítat všechny Carry C_i , pokud bychom předpokládali řetězení, tak známe C_0 jinak by :

- Opět použijeme strom, tentokrát v opačném pořadí než výpočet $P_{i,j}$ a $G_{i,j}$:
 - Pokud známe C_i , $G_{j,i}$ a $P_{j,i}$
 - pak $C_{i+1} = G_{j,i} \vee (C_j \wedge P_{j,i})$
- Výpočet pro 4-bitové sčítance bude probíhat v následujícím pořadí:
 - $C_4 = G_{0,3} \vee (C_0 \wedge P_{0,3})$, $C_2 = G_{0,1} \vee (C_0 \wedge P_{0,1})$
 - $C_3 = G_{2,2} \vee (C_2 \wedge P_{2,2})$, $C_1 = G_{0,0} \vee (C_0 \wedge P_{0,0})$
- Opět platí, že pro $2 \times$ větší sčítance se doba prodlouží o 2 zpoždění.
- Tedy pro 64 bitový je výpočet všech C_i s 12 zpožděním hradel.
- Celý součet pro 64 bitový sčítance trvá 26 zpožděním hradel.

Sčítání - Carry Lookahead Adder

Když máme spočtené G a P pro zadané rozsahy, pak začneme počítat vlastní carry:

$$\begin{array}{cccccccc}
 P_{7,7} & P_{6,6} & P_{5,5} & P_{4,4} & P_{3,3} & P_{2,2} & P_{1,1} & P_{0,0} \\
 G_{7,7} & G_{6,6} & G_{5,5} & G_{4,4} & G_{3,3} & G_{2,2} & G_{1,1} & G_{0,0} \\
 P_{6,7} & & P_{4,5} & & P_{2,3} & & P_{0,1} & \\
 G_{6,7} & & G_{4,5} & & G_{2,3} & & G_{0,1} & \\
 & P_{4,7} & & & & P_{0,3} & & \\
 & G_{4,7} & & & & G_{0,3} & & \\
 & & & P_{0,7} & & & & \\
 & & & G_{0,7} & & & & \\
 C_8 = G_{0,7} \text{ or } (C_0 \text{ and } P_{0,7}) & & & & & & & \\
 & & C_4 = G_{0,3} \text{ or } (C_0 \text{ and } P_{0,3}) & & & & & \\
 C_6 = G_{4,5} \text{ or } (C_4 \text{ and } P_{4,5}) & C_2 = G_{0,1} \text{ or } (C_0 \text{ and } P_{0,1}) & & & & & & \\
 C_7 = G_{6,6} \text{ or } (C_6 \text{ and } P_{6,6}) & C_3 = G_{2,2} \text{ or } (C_2 \text{ and } P_{2,2}) & & & & & & \\
 C_5 = G_{4,4} \text{ or } (C_4 \text{ and } P_{4,4}) & C_1 = G_{0,0} \text{ or } (C_0 \text{ and } P_{0,0}) & & & & & &
 \end{array}$$

Obecně lze napsat:

$$C_{i+1} = G_{j,i} \text{ or } (C_j \text{ and } P_{j,i})$$

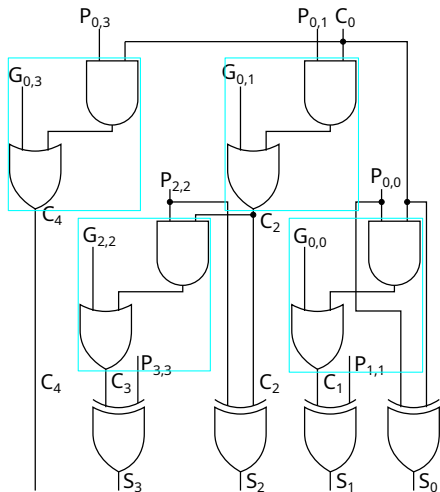
Výpočet C proběhne ve 4

krocích, barevně odlišených:

- Výpočet C_8
- Výpočet C_4
- Výpočet C_6, C_2
- Výpočet C_7, C_5, C_3, C_1

Sčítání - Carry Lookahead Adder

Výpočet C_i a pak i S_i vypadá následovně:



Doba výpočtu všech carry C_i a S_i :

- Rychlost 4-ti bitových sčítanců 5 zpoždění hradel
- Rychlost 8-ti bitových sčítanců 7 zpoždění hradel
- Rychlost 64-ti bitových sčítanců 13 zpoždění hradel

Celkem doba výpočtu pro 64-bitové sčítance je 24 zpoždění tedy asi 96ps.

Sčítání - Carry Lookahead Adder - Příklad

Výpočet generování a propagace carry bloku pomocí:

- g_h, p_h - generování a propagace carry ve vyšším podbloku
- g_l, p_l - generování a propagace carry v nižším podbloku

a	0	0	1	0	1	0	0	1
b	1	0	1	1	0	1	1	1
$g = a_i \text{ and } b_i$	0	0	1	0	0	0	0	1
$p = a_i \text{ xor } b_i$	1	0	0	1	1	1	1	0
$g = g_h \text{ or } (g_l \text{ and } p_h)$	0	1		0		1		
$p = p_h \text{ and } p_l$	0	0		1		0		
$g = g_h \text{ or } (g_l \text{ and } p_h)$	0				1			
$p = p_h \text{ and } p_l$	0				0			
$g = g_h \text{ or } (g_l \text{ and } p_h)$	0							
$p = p_h \text{ and } p_l$	0							

Sčítání - Carry Lookahead Adder - Příklad

Výpočet carry na základě carry nižších řádů a příznaků g,p:

- $C_i = g$ or $(p \text{ and } C_{i-k})$ - buď se carry vygeneruje a nebo se propaguje z nižších řádů

a	0	0	1	0	1	0	0	1
b	1	0	1	1	0	1	1	1
g	0	0	1	0	0	0	0	1
p	1	0	0	1	1	1	1	0
C								
g		0		1		0		1
p		0		0		1		0
C								
g			0				1	
p			0				0	
C					C ₄ = 1 or (0 and C ₀) = 1			
g					0			
p					0			
C								C ₈ = 0 or (0 and C ₀) = 0

Sčítání - Carry Lookahead Adder - Příklad

Výpočet carry na základě carry nižších řádů a příznaků g,p:

- $C_i = g$ or $(p \text{ and } C_{i-k})$ - buď se carry vygeneruje a nebo se propaguje z nižších řádů

a	0	0	1	0	1	0	0	1
b	1	0	1	1	0	1	1	1
g	0	0	1	0	0	0	0	1
p	1	0	0	1	1	1	1	0
C								
g		0		1		0		1
p		0		0		1		0
C		$C_6 = 1$ or $(0 \text{ and } C_4) = 1$				$C_2 = 1$ or $(0 \text{ and } C_0) = 1$		
g		0				1		
p		0				0		
C						$C_4 = 1$ or $(0 \text{ and } C_0) = 1$		
g						0		
p						0		
C		$C_8 = 0$ or $(0 \text{ and } C_0) = 0$						

Sčítání - Carry Lookahead Adder - Příklad

Výpočet carry na základě carry nižších řádů a příznaků g,p:

- $C_i = g$ or $(p \text{ and } C_{i-k})$ - buď se carry vygeneruje a nebo se propaguje z nižších řádů

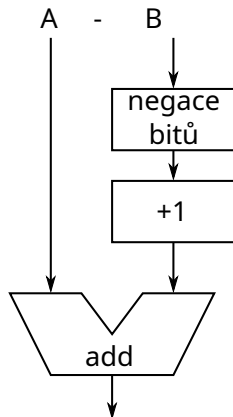
a	0	0	1	0	1	0	0	1
b	1	0	1	1	0	1	1	1
g	0	0	1	0	0	0	0	1
p	1	0	0	1	1	1	1	0
C	$C_7 = 0$ or $(0$ and $C_6) = 0$		$C_5 = 0$ or $(1$ and $C_4) = 1$		$C_3 = 0$ or $(1$ and $C_2) = 1$		$C_1 = 1$ or $(0$ and $C_0) = 1$	
g	0		1		0		1	
p	0		0		1		0	
C	$C_6 = 1$ or $(0$ and $C_4) = 1$				$C_2 = 1$ or $(0$ and $C_0) = 1$			
g	0				1			
p	0				0			
C					$C_4 = 1$ or $(0$ and $C_0) = 1$			
g					0			
p					0			
C	$C_8 = 0$ or $(0$ and $C_0) = 0$							

Odčítání

Nyní jsme si ukázali optimální sčítačku, pracující i pro čísla o velikost 128 bitů.

Odčítání lze řešit:

- speciálním obvodem podobným sčítačce se všemi možnostmi zrychlení jako byly u sčítání
- nebo z druhého čísla vytvoříme číslo opačné a to pak jednoduše sečteme s tím prvním



V PSY jsme si ukázali obvod pro přičtení 1, jedná se vlastně o výpočet and pro všechny kombinace bitů od nejnižšího bitu.

Násobení celých čísel

Stejný princip násobení, jak jste se ho naučili na základní škole pro desítkovou soustavu:

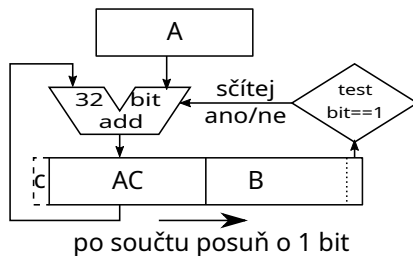
$$\begin{array}{r}
 153 \\
 *45 \\
 \hline
 765 \\
 612 \\
 \hline
 6885
 \end{array}$$

$$\begin{array}{r}
 10011001 \\
 *101101 \\
 \hline
 10011001 \\
 00000000 \\
 10011001 \\
 10011001 \\
 00000000 \\
 10011001 \\
 \hline
 1101011100101
 \end{array}$$

Násobení celých čísel

Podle uvedeného algoritmu můžeme vytvořit následující násobičku s posuvným registrem:

(A,B 32 bitů, výsledek 64 bitů)



- Výsledek je ve dvou registrech AC a B
- Pomalé, už sčítání je náročné, nyní 32 sčítání, tedy přibližně $32 \cdot 20 = 640$ zpoždění, tedy asi 2560ps, tedy 10 taktů procesoru.
- Pro násobení 64-bitových čísel přibližně $64 \cdot 24 = 1536$ zpoždění, tedy asi 25 taktů procesoru.

Rychlé násobení - Motivace Wallace tree

Jak to zrychlit - odložené carry (Carry Save Adder).

Carry Save Adder je vlastně Full Adder s tím, že oddělíme součet a přenos do rozdílných čísel.

Jak nejrychleji spočítat součet čtyř 32-bitových čísel:

$$\begin{array}{r}
 \boxed{w_{31}} \dots \boxed{w_4} \boxed{w_3} \boxed{w_2} \boxed{w_1} \boxed{w_0} \\
 + x_{31} \dots x_4 x_3 x_2 x_1 x_0 \\
 + y_{31} \dots y_4 y_3 y_2 y_1 y_0 \\
 + z_{31} \dots z_4 z_3 z_2 z_1 z_0 \\
 \hline
 p_{31} \dots p_4 p_3 p_2 p_1 p_0 \\
 \leftarrow \\
 c'_{31} c'_{30} \dots c'_3 c'_2 c'_1 c'_0 \\
 \leftarrow \\
 z_{31} \dots z_4 z_3 z_2 z_1 z_0 \\
 \leftarrow \\
 c_{31} q_{31} \dots q_4 q_3 q_2 q_1 q_0 \\
 \leftarrow \\
 c_{31} c_{30} \dots c_3 c_2 c_1 c_0 \\
 \hline
 s_{33} s_{32} s_{31} \dots s_4 s_3 s_2 s_1 s_0
 \end{array}$$

- Sečteme paralelně $p=w+x$ a $q=y+z$ a potom $p+q$ – trvá dlouho, nejméně doby dvou plných součtů
- Odložíme carry (nebudeme carry propagovat, jen v posledním kroku):
 - 1. krok použijeme Full adder a sečteme vždy bity $w_i + x_i + y_i = c'_i p_i$
 - 2. krok použijeme Full adder a sečteme vždy bity $p_i + c'_{i-1} + z_i = c_i q_i$
 - 3. krok použijeme normální sčítačku 32-bitových čísel ($s_0 = q_0$, c'_{31} připojíme ke q)

Rychlé násobení - Motivace Wallace tree

Jakého zrychlení tedy dosáhneme?

Sčítání šesti čísel:

$$\begin{array}{r}
 a_{31} \dots a_4 a_3 a_2 a_1 a_0 \\
 + b_{31} \dots b_4 b_3 b_2 b_1 b_0 \\
 + c_{31} \dots c_4 c_3 c_2 c_1 c_0 \\
 + d_{31} \dots d_4 d_3 d_2 d_1 d_0 \\
 + e_{31} \dots e_4 e_3 e_2 e_1 e_0 \\
 + f_{31} \dots f_4 f_3 f_2 f_1 f_0 \\
 \hline
 w_{31} \dots w_4 w_3 w_2 w_1 w_0 \\
 + x_{31} x_{30} \dots x_3 x_2 x_1 x_0 \\
 + y_{31} \dots y_4 y_3 y_2 y_1 y_0 \\
 + z_{31} z_{30} \dots z_3 z_2 z_1 z_0 \\
 \hline
 + x_{31} p_{31} \dots p_4 p_3 p_2 p_1 p_0 \\
 + c_{31} c'_{30} \dots c'_3 c'_2 c'_1 c'_0 \\
 + z_{31} z_{30} \dots z_3 z_2 z_1 z_0 \\
 \hline
 q_{32} q_{31} \dots q_4 q_3 q_2 q_1 p_0 \\
 \hline
 c_{32} c_{31} c_{30} \dots c_3 c_2 c_1 c_0 \\
 \hline
 s_{34} s_{33} s_{32} s_{31} \dots s_4 s_3 s_2 s_1 s_0
 \end{array}$$

- Každý krok zredukuje velikost problému na $\lceil \frac{2}{3} \rceil$
- Kolik kroků potřebuje na součet 6 čísel:
 - 1. krok redukce na součet 4 čísel - 2 zpoždění
 - 2. krok redukce na součet 3 čísel - 2 zpoždění
 - 3. krok redukce na součet 2 čísel - 2 zpoždění
 - 4. krok normální sčítačka 2 čísel - 24 zpoždění
- Součet 6 čísel - 30 zpoždění
- Rychlost součtu N čísel, počet kroků na redukcii součtu dvou čísel $\log_{\frac{3}{2}}(N) = \frac{1}{\log(\frac{3}{2})} \cdot \log(N) = 5.678 \cdot \log(N) = O(\log(N))$

Rychlé násobení - Wallace tree

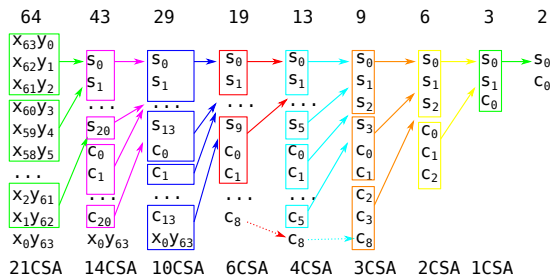
Použijeme předchozí princip na co nejrychlejší součet 32 nebo 64 různých čísel:

				x_{63}	\dots	x_2	x_1	x_0	
				*	y_{63}	\dots	y_2	y_1	y_0
0	0	0	\cdot	$x_{63}y_0$	\cdot	x_2y_0	x_1y_0	x_0y_0	
0	0	0	\cdot	$x_{62}y_1$	\cdot	x_1y_1	x_0y_1	0	
0	0	0	\cdot	$x_{61}y_2$	\cdot	x_0y_2	0	0	
				\cdot	\dots	\cdot	\dots		
0	0	$x_{63}y_{61}$	\cdot	x_2y_{61}	\cdot	0	0	0	
0	$x_{63}y_{62}$	$x_{62}y_{62}$	\cdot	x_1y_{62}	\cdot	0	0	0	
$x_{63}y_{63}$	$x_{62}y_{63}$	$x_{61}y_{63}$	\cdot	x_0y_{63}	\cdot	0	0	0	
q_{127}	q_{126}	q_{125}	q_{124}	q_{63}	q_2	q_1	q_0		

- Součiny jsou jednoduché:
 $x_i \cdot y_j = x_i \text{ and } y_j$
- Nejtěžší je sečíst
prostřední sloupec 64
jednobitových čísel
- Pustíme na všechny bity,
co můžeme, paralelně
sčítačky a carry budeme
přičítat v dalších krocích
- V první fázi to bude
1323 sčítaček

Rychlé násobení - Wallace tree

Když se podíváme jen na prostřední nejdelší sloupec:



- Vidíme, že za 8 kroků sčítaček, tedy 16 zpoždění hradel nám zbývá sečíst dva bity
- Vpravo od tohoto sloupce je již něco sečteno a již se i propagovali carry posledních 8 sčítanců
- Zbývá sečíst dvě 120-bitová čísla (součty a carry), což se dá stihnout asi za 30 zpoždění hradel
- Výsledek - vynásobíme dvě čísla za cenu doby odpovídající přibližně dvěma součtům 64-bitových čísel
- V praxi to trvá trochu déle, je použito méně sčítaček CSA.

Testovací kvíz

Kolik jste toho zatím pochopili?

- A Vše bez problému.
- B Téměř vše.
- C Téměř nic.
- D Vůbec nic.