B35APO: Architektury počítačů

Lekce 01. Úvod

Pavel Píša
pisa@fel.cvut.cz

Petr Štěpán
stepan@fel.cvut.cz

25. února, 2024

Obsah

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 $S(p \to \infty) = lim_{p\to\infty} \frac{p}{1+\alpha\cdot(p-1)} = \frac{1}{\alpha}$
    - Např pro $\alpha = 0.3$ je limit $S(p \to \infty) = 3.\overline{3}$, tedy program nelze zrychlit více než $3.\overline{3}$ 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
  - Né každá přednáška, 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 ± 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

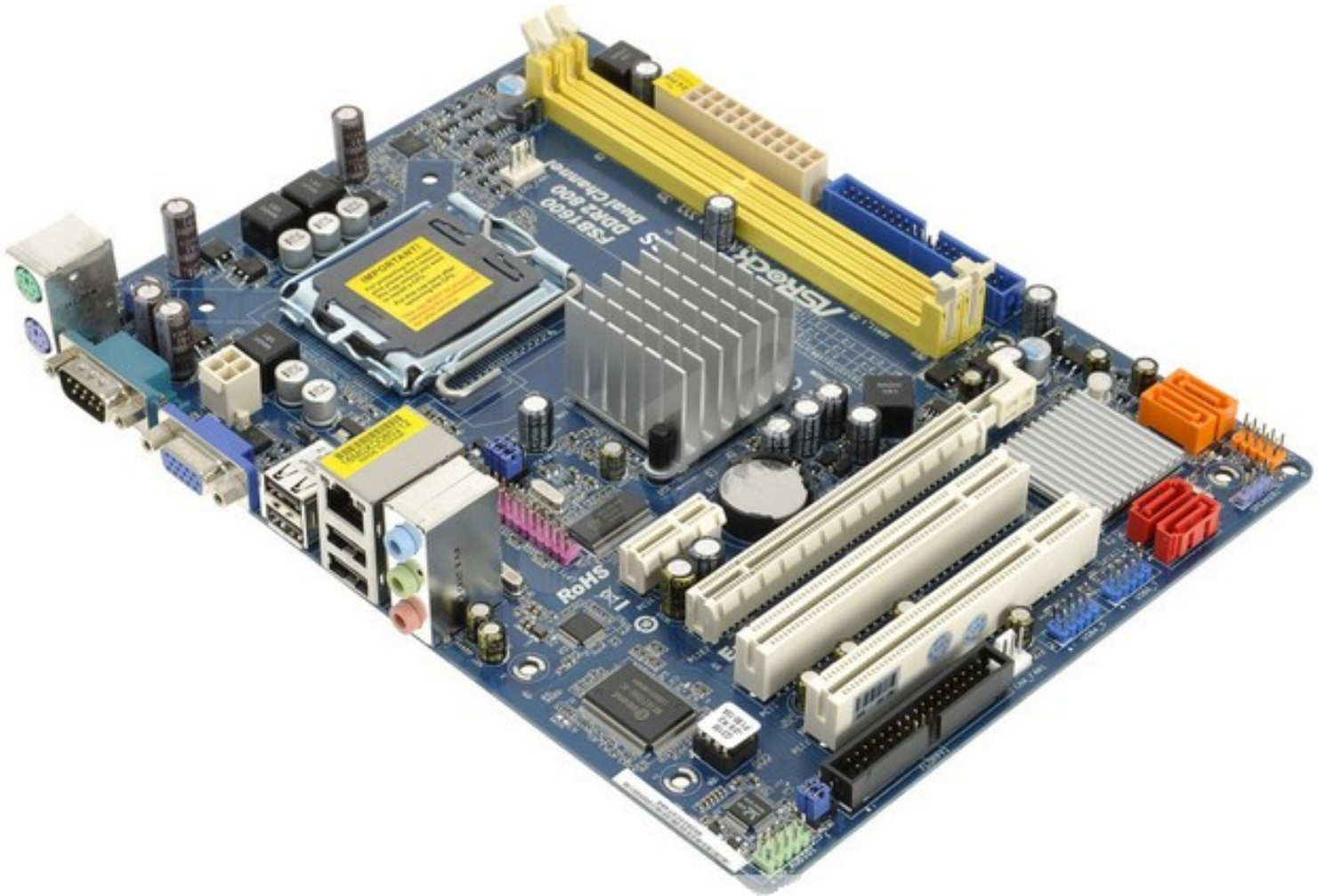
Materiály k předmětu

- 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

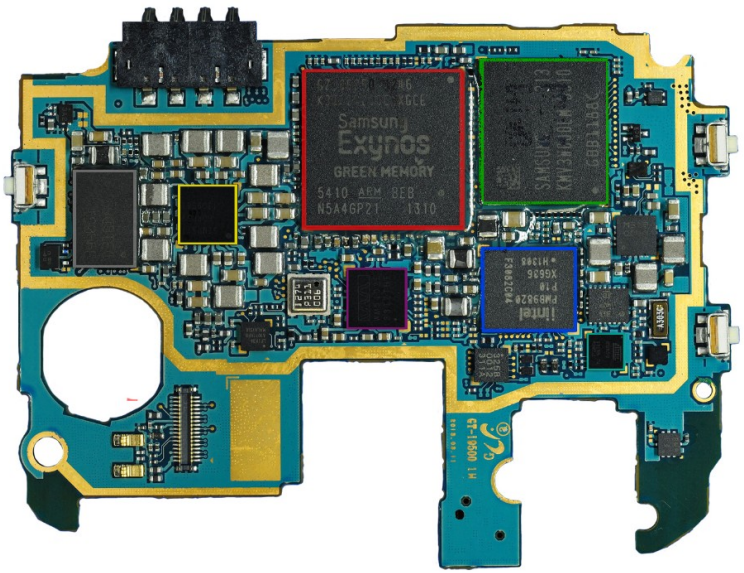
Co je uvnitř počítače

Základní deska počítače:



Co je uvnitř počítače

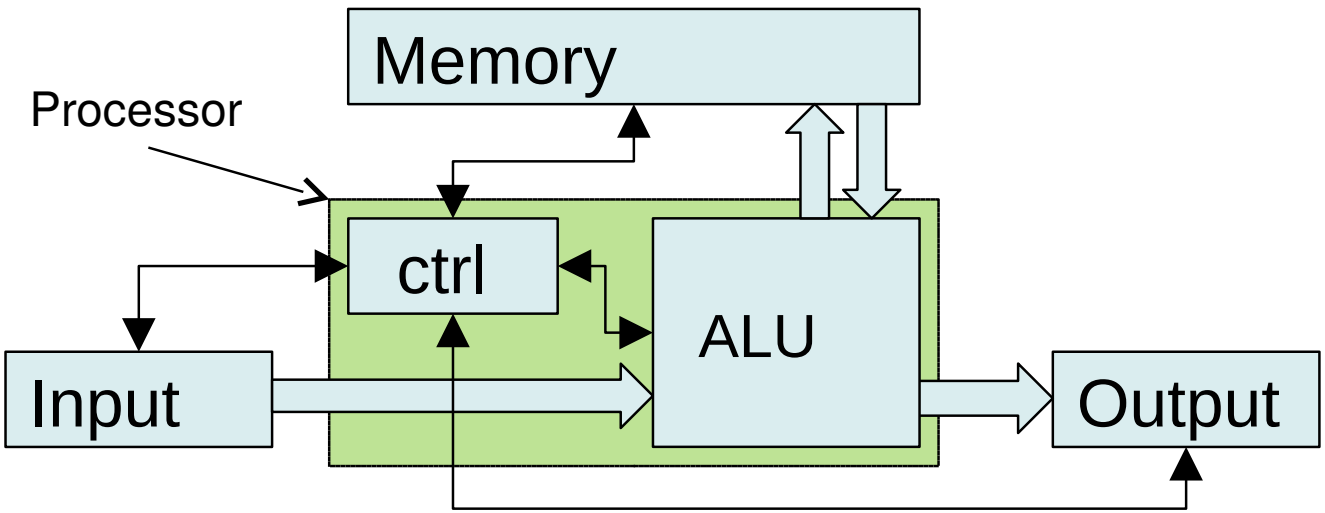
Rozebraný telefon:



von Neumann

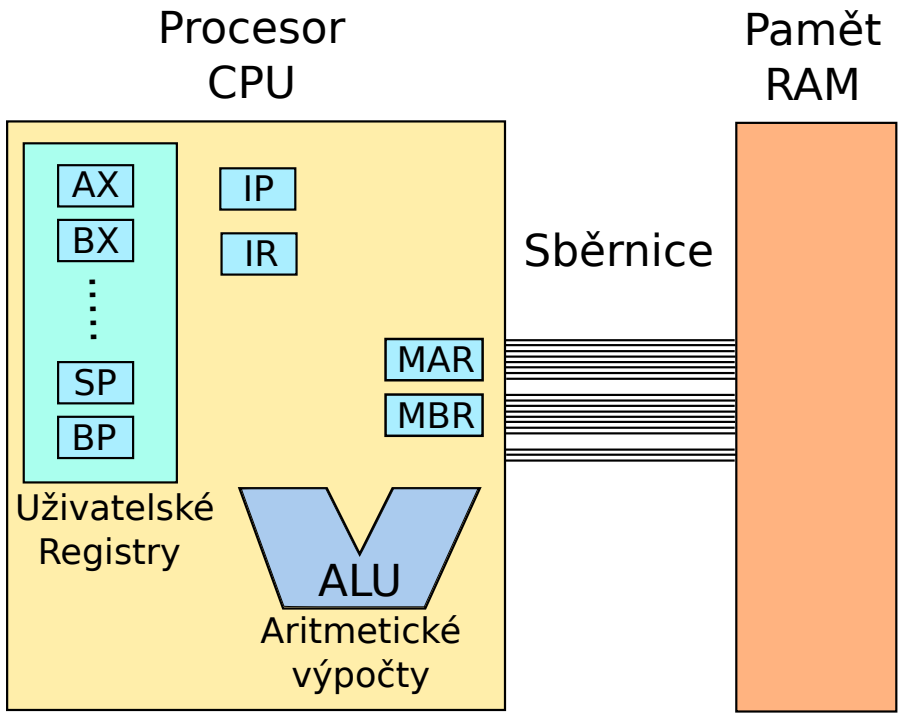
Společný koncept navržený maďarským fyzikem Johnem von Neumannem (1903-1957) obsahuje:

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



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:

```
registr R10 = RAM[adresa];
```

nebo zapisovat:

```
RAM[adresa] = R10;
```

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 dvě čísla
  - 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 složené 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)
  - množina 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

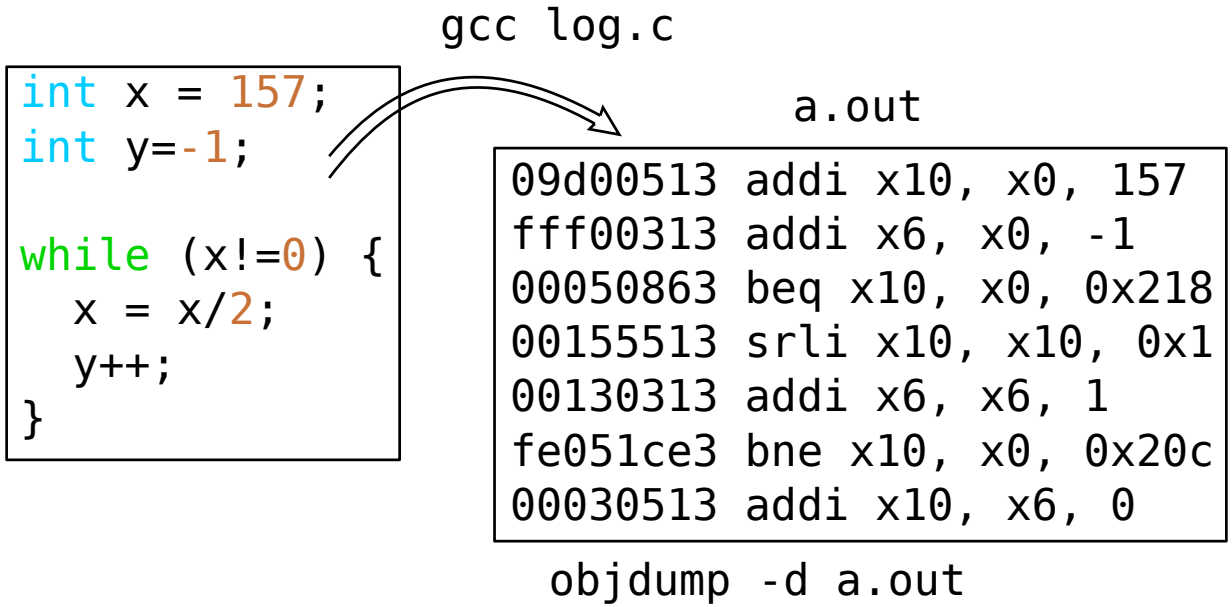
CISC

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 módu
- Matematické oprace 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.

Procesor

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

Booleova algebra

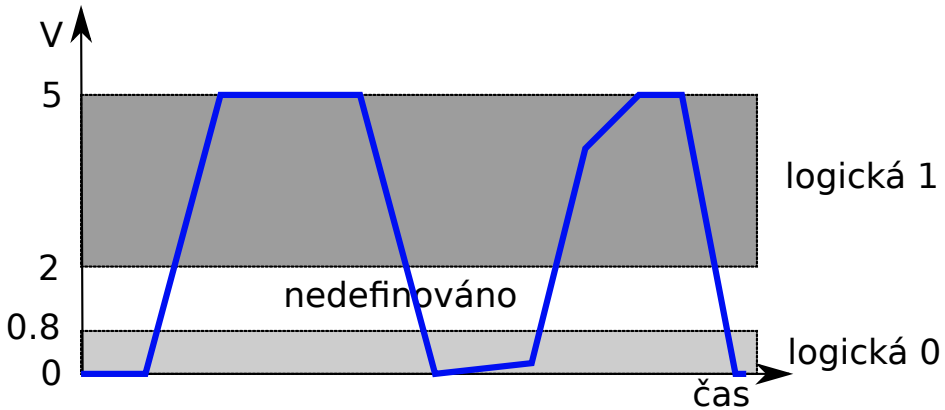
Booleova algebra je matematická struktura:

- Používá pouze dvě hodnoty 0,1
  - 0/1, nebo False/True, nebo nesvítí/svítí, nebo 0V/5V
- Operace plus (nebo, ||, or, $\vee$)
  - 0+0=0 0+1=1
  - 1+0=1 1+1=1
- Operace krát (a, &&, and, $\wedge$)
  - 0*0=0 0*1=0
  - 1*0=0 1*1=1
- Operace inverzní prvek - (negace, !, not, $\neg$)
  - -0 = 1
  - -1 = 0

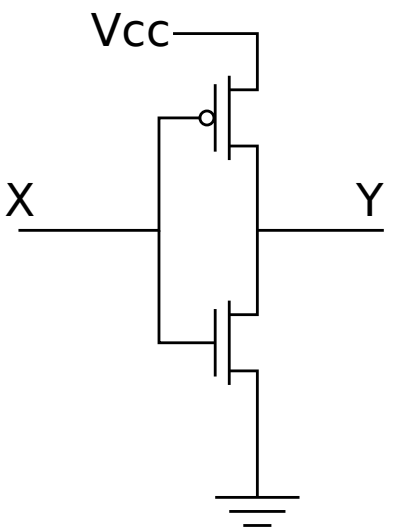
Boolova algebra

Booleova algebra se dá dobře implementovat pomocí napětí a tranzistorů (nebude u zkoušky)

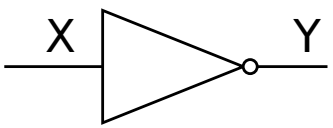
Napětí jednoho vodiče vůči zemi definuje logickou hodnotu.



Příklad: Logická operace not, jeden vstupní vodiče X, jeden výstupní vodič Y.



symbol neg



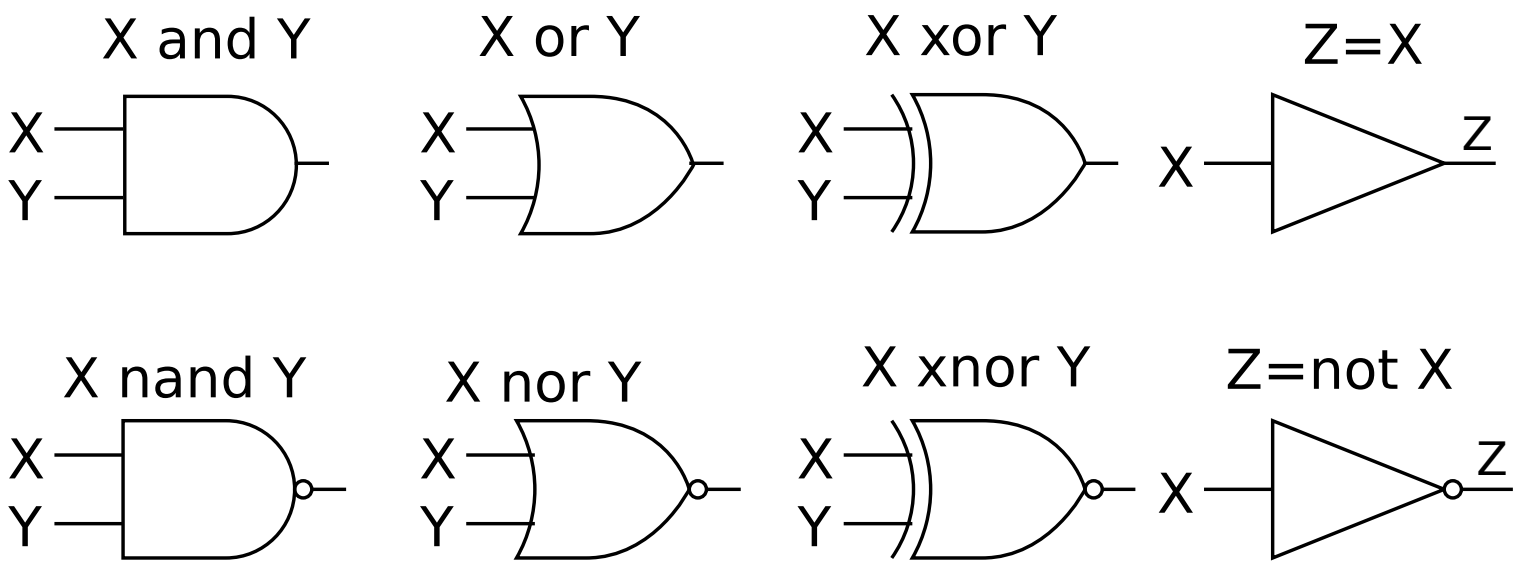
Booleova algebra

Rozšířené operace `nand`, `nor`, `xor` a libovolné logické funkce se skládají ze základních operací (`and`, `or`, `not`):

```
X nand Y  = not(X and Y)
X nor Y   = not(X or Y)
X xor Y   = (X or Y) and (not(X and Y))
          = (X or Y) and (X nand Y)
```

Booleova algebra

Přehled hradel a jejich symbolů

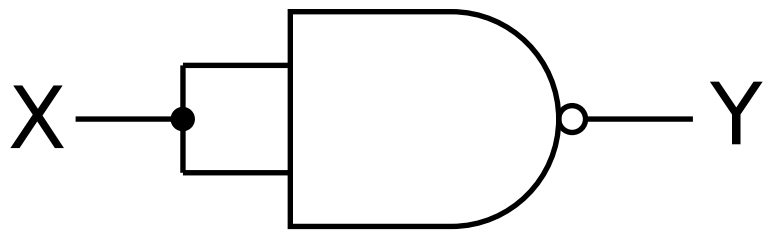


Souhrná tabulka základních logických hradel:

X	Y	X and Y	X or Y	X xor Y	X nand Y	X nor Y	X xnor Y
0	0	0	0	0	1	1	1
0	1	0	1	1	1	0	0
1	0	0	1	1	1	0	0
1	1	1	1	0	0	0	1

Booleova algebra - kvíz

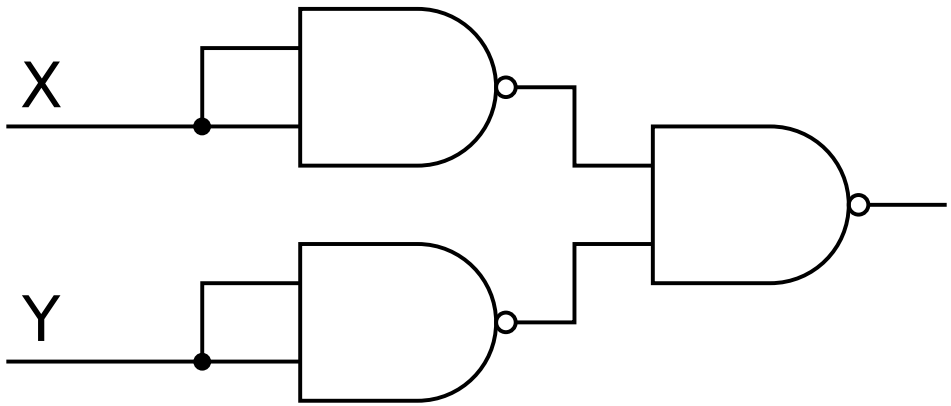
Vodiče se mohou i větvit. Co dělá následující obvod:



A) takto to nelze zapojit
B) výsledkem je X and Y
C) výsledkem je X and not Y
D) výsledkem je not X

Logické obvody

Některé funkce lze převést do hradel i efektivněji než přes základní logické funkce and, or, not.
Hradlo nand je základním hradlem a lze z něj postavit všechny ostatní hradla.



Kvíz: Co je toto za funkci:

A X and Y
B X or Y
C X xor Y
D X nor Y

Logické obvody

Logické obvody jsou složitější obvody složené ze základních logických členů - hradel. Signály se mohou větvit, spojují se vždy nějakou logickou operací. Výsledkem logického obvodu jsou vždy jen logické hodnoty 0 nebo 1.

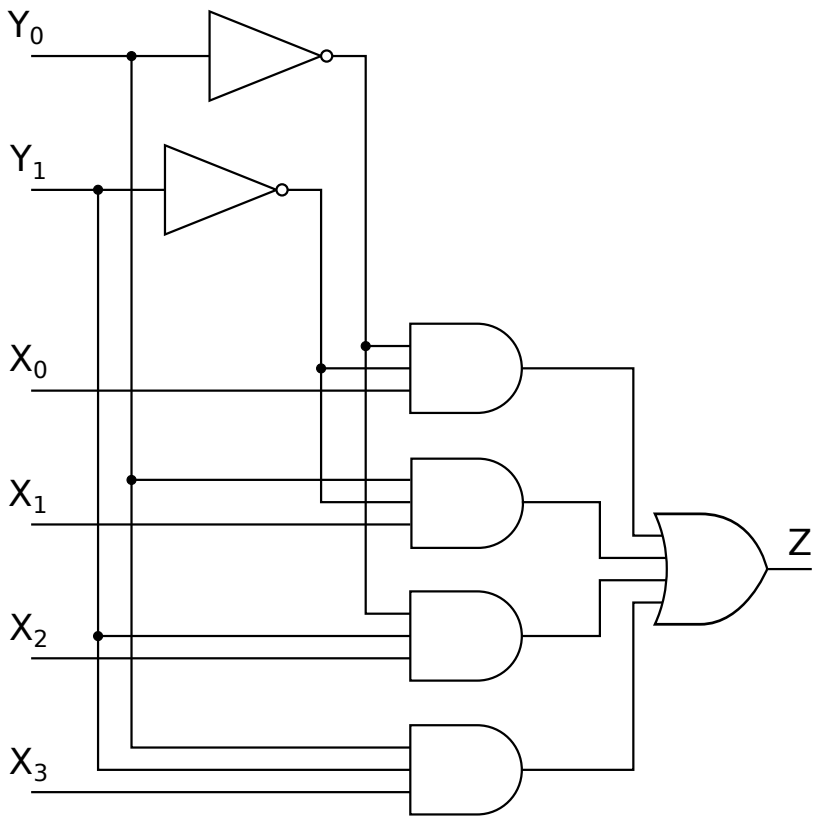
Kvíz: Co dělá tento obvod:

A Nic rozumného, je to jen spleť vodičů

B Je to multiplexor, hodnota Z je jeden ze signálů X podle hodnoty Y

C Je to dělička, hodnota Z je X/Y

D Hodnota Z je 1 pokud je $X > Y$



Binární soustava - kvíz

Kvíz: Jeden vodič reprezentuje jednu hodnotu, buď 0 nebo 1. Jak reprezentovat více hodnot, třeba všechna celá čísla od 0 do 255 (tedy jeden bajt):

A Jeden vodič má 8 rozdílných úrovní napětí

B Jeden vodič reprezentuje postupně v čase 8 různých hodnot 0/1

C Osm vodičů, každý reprezentuje v jeden čas jednu z hodnot 0/1

D 256 vodičů, pouze jeden má hodnotu 1, ostatní mají hodnotu 0

Binární soustava

Více bitová čísla - binární soustava

- více jednobitových paralelních vodičů
  - zpravidla 8,16,32,64 (mocniny 2)
  - občas potřebujeme i jen části čísel, třeba 5-bitové
- pořadí vodičů je důležité
  - každý vodič reprezentuje jednu mocninu 2
  - vodič a_i, celková hodnota $s = \sum_{i=0}^{63} a_i * 2^i$

Sčítání

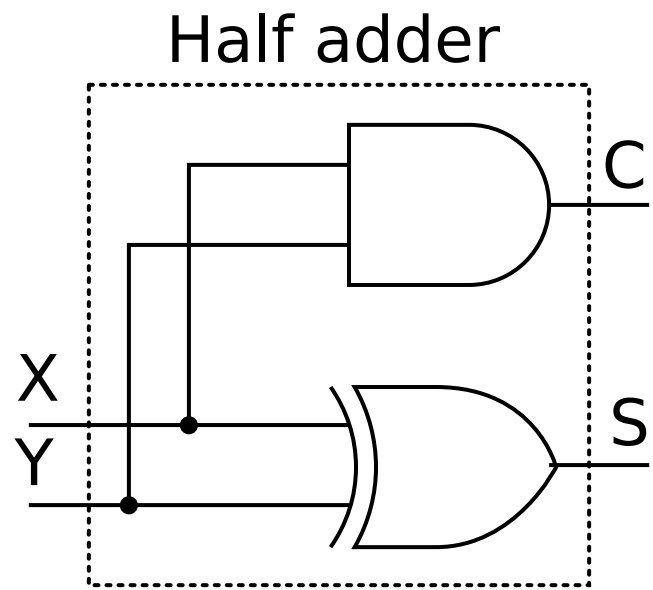
Součet dvou jednobitových čísel:

X	Y	X+Y
0	0	00
0	1	01
1	0	01
1	1	10

Výsledek součtu může být dvoubitové číslo C - přenos, S - součet.

S = X `xor` Y
C = X `and` Y

Tomuto logickému obvodu se říká poloviční sčítačka (Half adder)



Sčítání

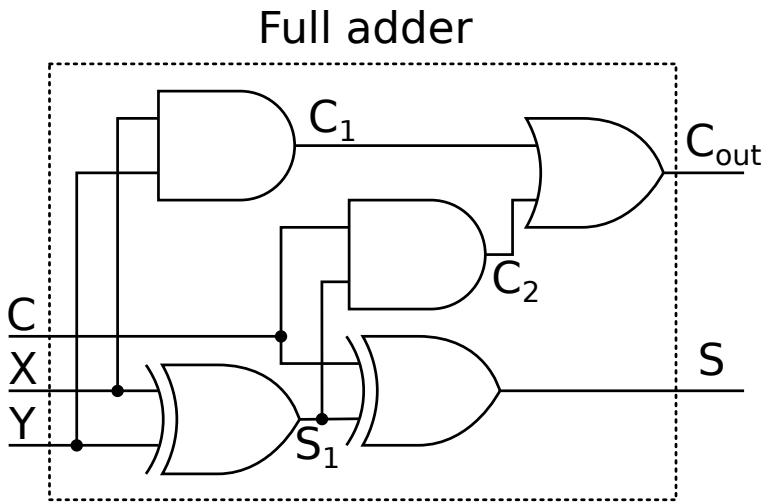
Pokud sčítáme vícebitová čísla potřebujeme později sečíst tři bitové vstupy.

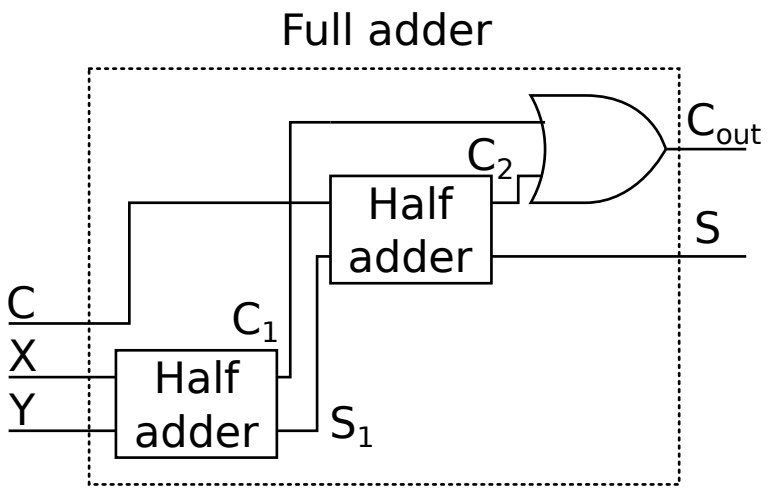
Výsledek součtu je opět dvoubitové číslo:
C_{out} - přenos, S - součet.

C	X	Y	C+X+Y
0	0	0	00
0	0	1	01
0	1	0	01
0	1	1	10
1	0	0	01
1	0	1	10
1	1	0	10
1	1	1	11

$S_1 = (X \texttt{ xor } Y)$
$S = (S_1 \texttt{ xor } C)$
$C_1 = (X \texttt{ and } Y)$
$C_2 = (S_1 \texttt{ and } C)$
$C_{out} = C_1 \texttt{ or } C_2$

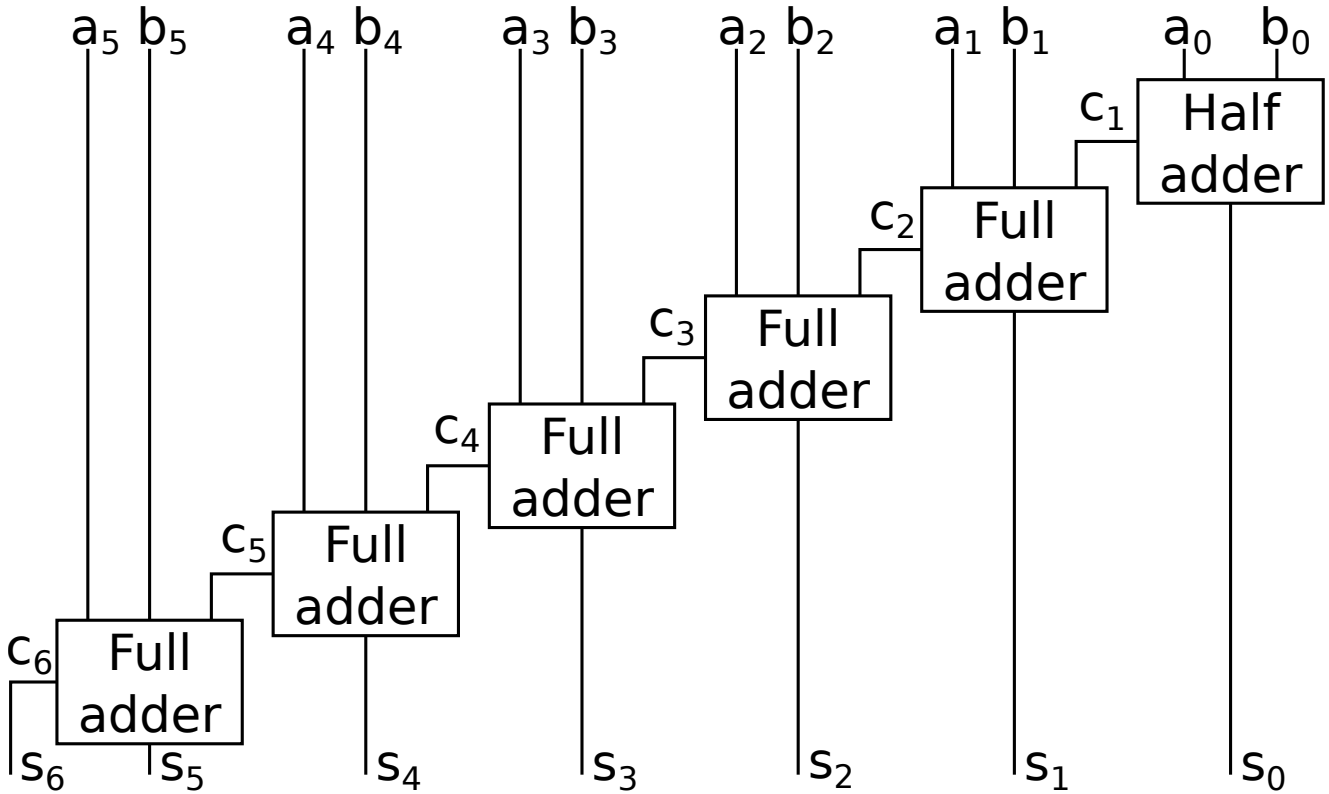
Tomuto logickému obvodu se říká úplná sčítačka (Full adder)





Sčítání - Ripple Carry Adder

Nejjednodušší sčítačka vícebitových čísel prostě propojí jednu poloviční sčítačku a plné sčítačky.
Tato sčítačka se jmenuje Ripple Carry Adder (Řetězová sčítačka).



$a_5 a_4 a_3 a_2 a_1 a_0 + b_5 b_4 b_3 b_2 b_1 b_0 = s_6 s_5 s_4 s_3 s_2 s_1 s_0$ platí, že $s_6 = c_6$

Sčítání - Ripple Carry Adder

Jak rychlá je Ripple Carry Adder?

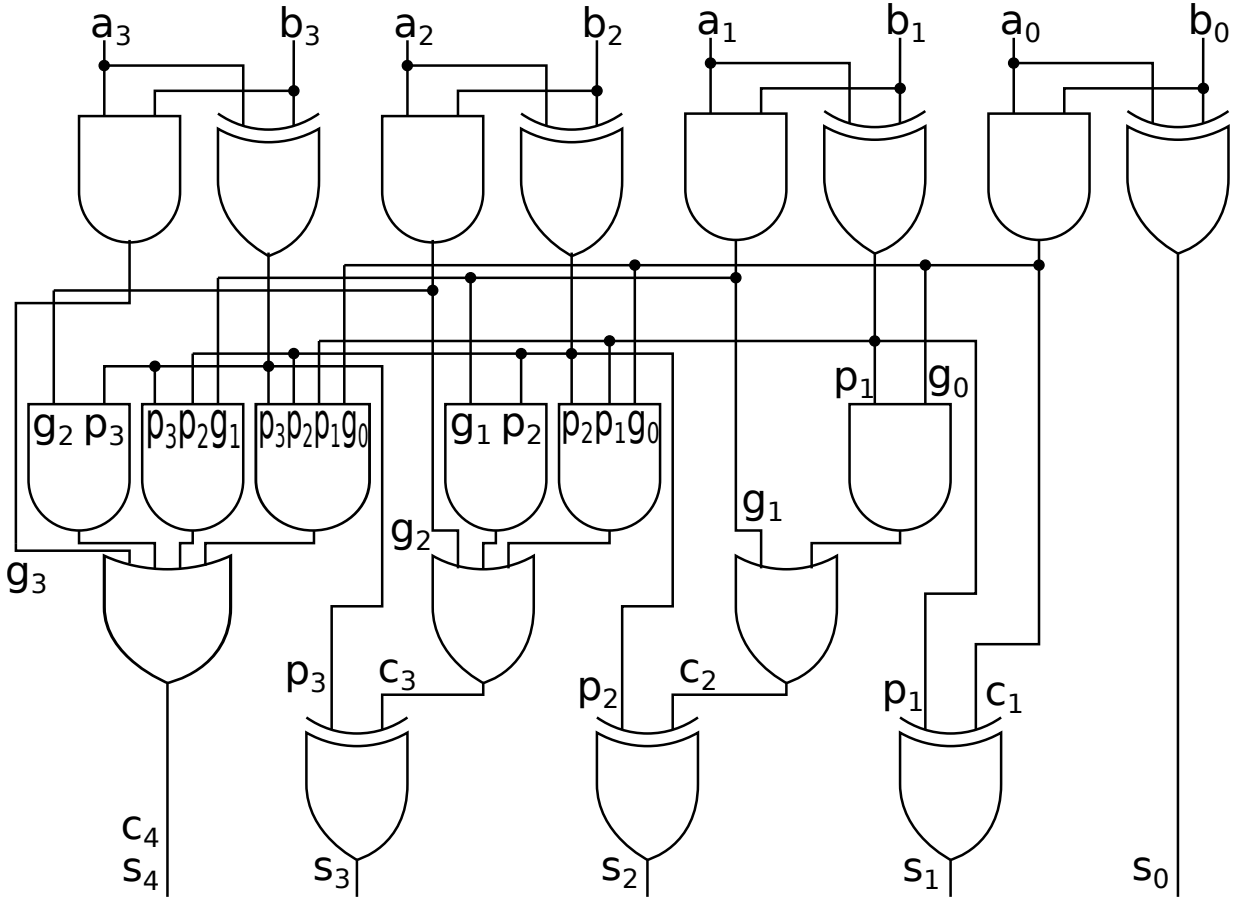
- Pokud je zpoždění jednoho hradla N pak zpoždění Ripple Carry Adder pro dvě 64-bitové hodnoty 63*2*N+N
- Je to důležité?
  - Ano, při taktu CPU 4GHz trvá jeden takt 250 ps (pikosekund)
  - Omezení je i rychlost světla 0.3 mm/ps, to je maximální rychlost šíření informace
  - Při zpoždění hradla kolem 10ps, pak Ripple Carray Adder by sečetla 2 čísla za 1270 ps (to je více než 5 taktů procesoru).
- Lze to rychleji?
  - Ano, Carry Lookahead adder

Sčítání - Carry Lookahead adder

- Lze spočítat C_1, C_2, ... , rovnou ze sčítaných čísel?
  - Ano, ale pro hodně bitová čísla to je drahé, potřebujeme hodně hradel.
  - Ukážeme si to pro 4-bitová čísla
  - Nadefinujeme si dvě základní hodnoty:
    - Generování carry – pokud $A_i = 1$ a $B_i = 1$ pak určitě bude $C_{i+1} = 1$ tedy carry se vygeneruje $G_i = A_i \wedge B_i$
    - Propagování carry – pokud $A_i = 1$ nebo $B_i = 1$ pak bude $C_{i+1} = C_i$ tedy carry se bude propagovat, pokud bylo nižší carry; $P_i = A_i$ xor B_i
  - Pro 4-bitové číslo:
    - $C_1 = G_0$
    - $C_2 = G_1 \vee (C_1 \wedge P_1) = G_1 \vee (G_0 \wedge P_1)$
    - $C_3 = G_2 \vee (C_2 \wedge P_2) = G_2 \vee (G_1 \wedge P_2) \vee (G_0 \wedge P_1 \wedge P_2)$
    - $C_4 = G_3 \vee (C_3 \wedge P_3) = G_3 \vee (G_2 \wedge P_3) \vee (G_1 \wedge P_2 \wedge P_3) \vee (G_0 \wedge P_1 \wedge P_2 \wedge P_3)$
    - takto bychom mohli pokračovat, ale výraz by byl složitější a složitější

Sčítání - Carry Lookahead Adder

Takto bude vypadat nejrychlejší sčítačka pro 4-bitová čísla.
Dvě čísla sečte na 4 zpoždění hradla.

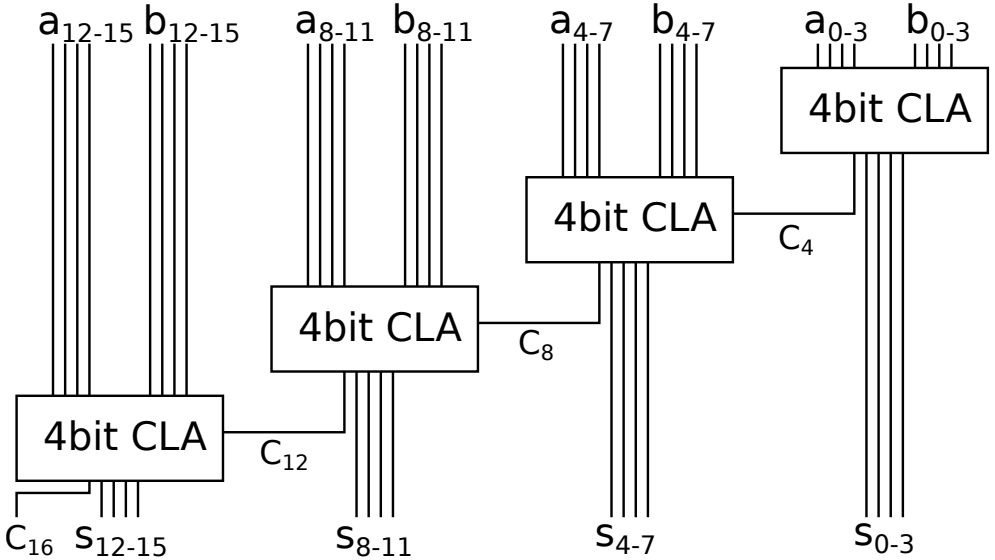


Pro 64 bitová čísla by potřebovala sčítačka kolem 10^{20} hradel, což je moc.

Sčítání - Carry Lookahead Adder

Co tedy s většími čísly, třeba 64-bitovými?

- Upravíme 4-bitovou sčítačku tak, aby mohla přijmout C z předchozích výpočtů
  - POZOR budou se muset přidat hradla
- Takto upravené sčítačky můžeme řetězit.

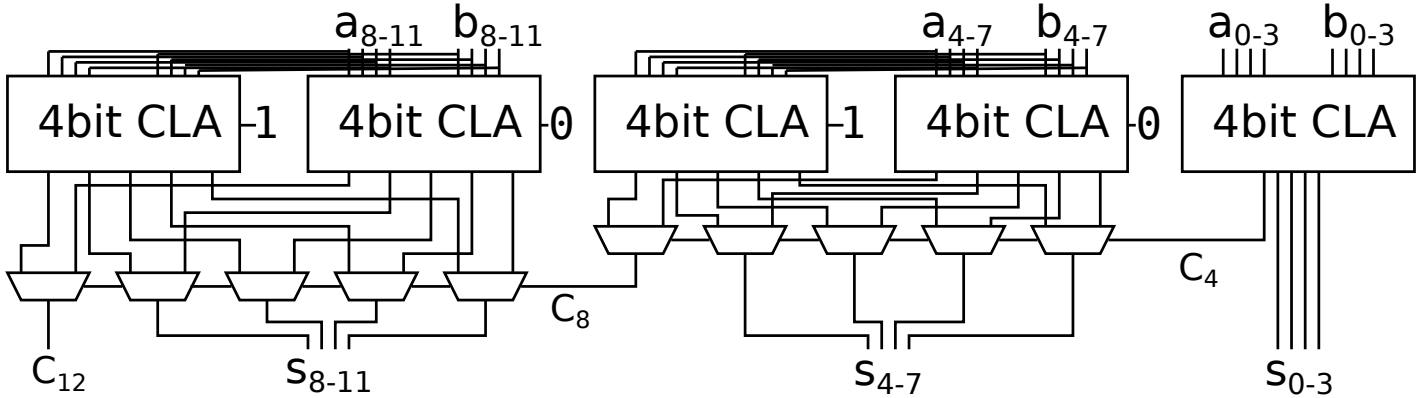


- Rychlost 16-ti bitové sčítačky z obrázku bude 16 zpoždění hradel
- Rychlost 64-ti bitové sčítačky bude 64 zpoždění hradel, což je daleko lepší, než 127 zpoždění u jednoduché Ripple carry adder.

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, 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 o něco lepší, než 64 zpoždění u řetězení CLA.

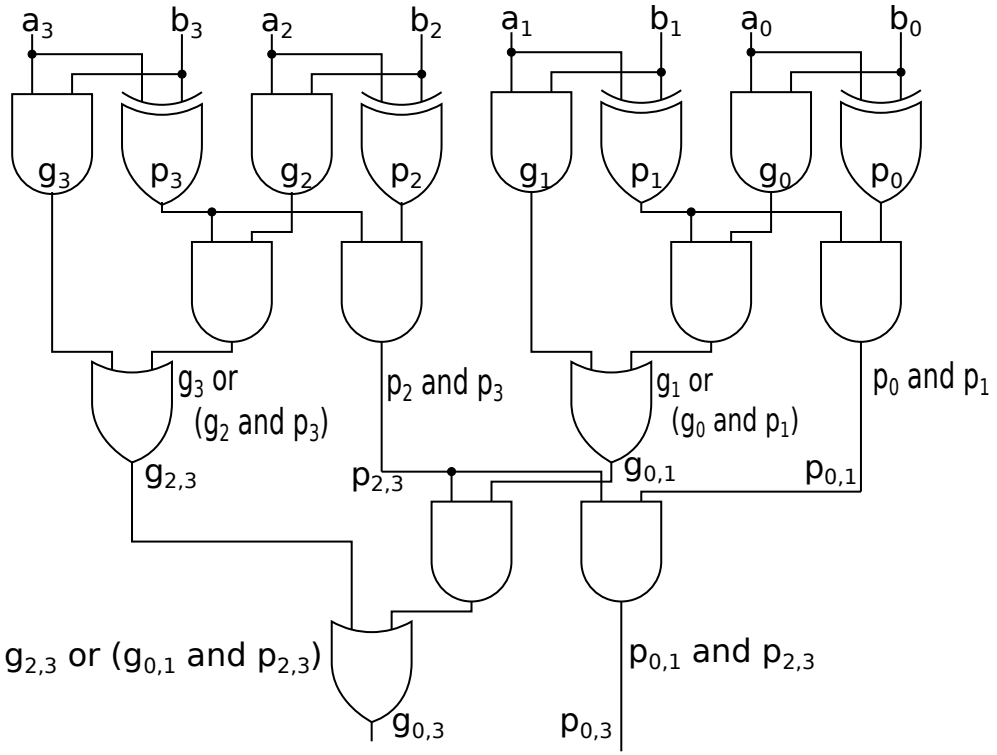
Sčítání - Carry Lookahead Adder

Můžeme ještě něco vylepšit?

- Můžeme zavést proměnné generuj a propaguj pro skupinu bitů:
  - Pokud budou bity i až j generovat přenos (carry), pak $G_{i,j} = 1$
  - Pokud budou bity i až j propagovat přenos (carry), pak $P_{i,j} = 1$
- Nyní si definujeme pravidla, jak $G_{i,k}$ a $P_{i,k}$ spočítat z $G_{i,j}$, $G_{j+1,k}$,$P_{i,j}$, $P_{j+1,k}$
  - $G_{i,k} = G_{j+1,k} \vee (G_{i,j} \wedge P_{j+1,k})$
  - $P_{i,k} = P_{i,j} \wedge P_{j+1,k}$
- Počáteční hodnoty jsou staré známé G_i a P_i.
- Tedy $G_{i,i} = G_i = a_i \wedge b_i$ a $P_{i,i} = P_i = a_i$ xor b_i.

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 - 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 and b_i	0	0	1	0	0	0	0	1
p = a_i xor b_i	1	0	0	1	1	1	1	0
g = g_h or (g_l and p_h)	0		1		0		1	
p = p_h and p_l	0		0		1		0	
g = g_h or (g_l and p_h)	0				1			
p = p_h and p_l	0				0			
g = g_h or (g_l and p_h)	0							
p = p_h and p_l	0							

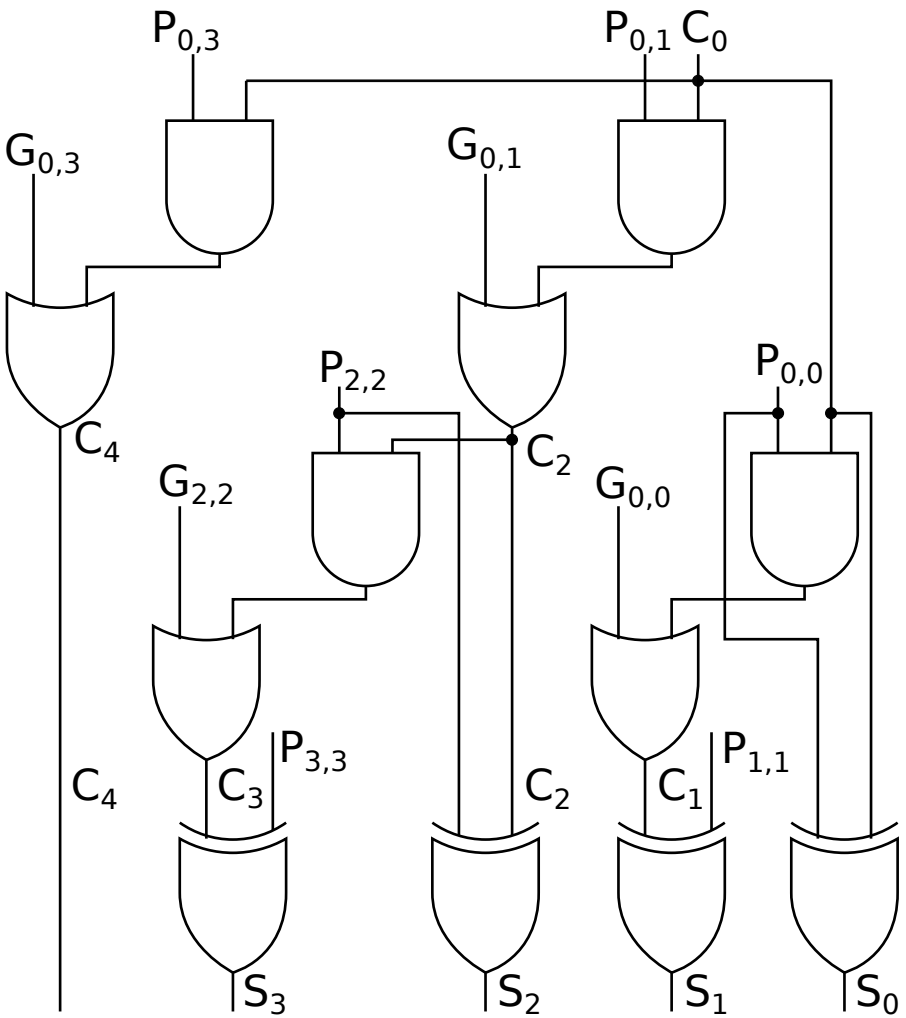
Sčítání - Carry Lookahead Adder

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, $G_{i,j}$ a $P_{i,j}$
  - pak $C_{j+1} = G_{i,j} \vee (C_i \wedge P_{i,j})$
- 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

Výpočet C_i pak vypadá následovně:



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

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

Sčítání - Carry Lookahead Adder - Příklad

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

- $C_i = g$ or $(p$ 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 $(p$ 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 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 $C_0) = 0$							

Sčítání - Carry Lookahead Adder - Příklad

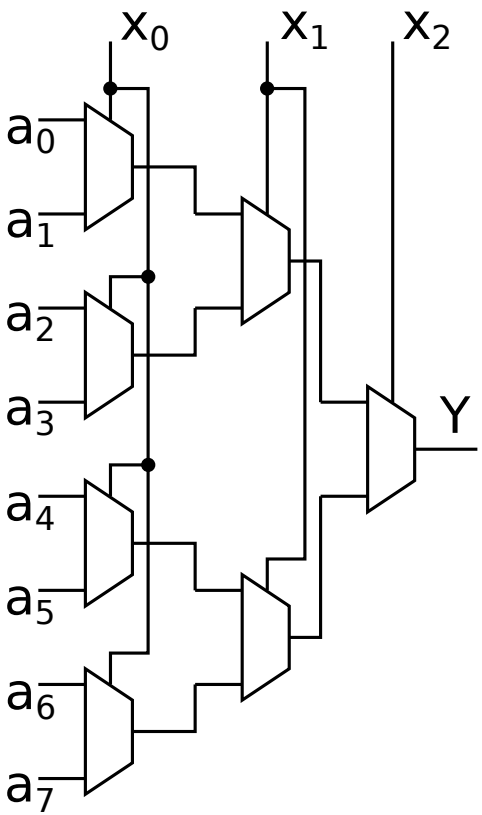
Výpočet carry na základě carry nižších řádů a příznaků g,p:

- $C_i = g$ or $(p$ 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 $C_0) = 0$							

Multiplexor

Podobný přístup (rozděl a panuj - vytvoř stromovou strukturu) lze využít třeba i při konstrukci multiplexu:



- Podle tříbitového čísla x se vybere odpovídající vstup a_x na výstup Y
- Není to nejrychlejší implementace, ale je přehledná
- Můžeme využít i 4 vstupový multiplex a snížit počet úrovní stromu.

Operace posun – shift

Oprace posun – shift:

- Odpovídá to operaci jazyka C >> a <<
  - Operace lze mimo jiné použít pro násobení mocninou 2 – oprace <<; pro dělení mocninou 2 – oprace >>
- Jak implementovat posun o k-bitů?

k-krát rotovat o 1 bit

Porovnej algoritmus mocnění:

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

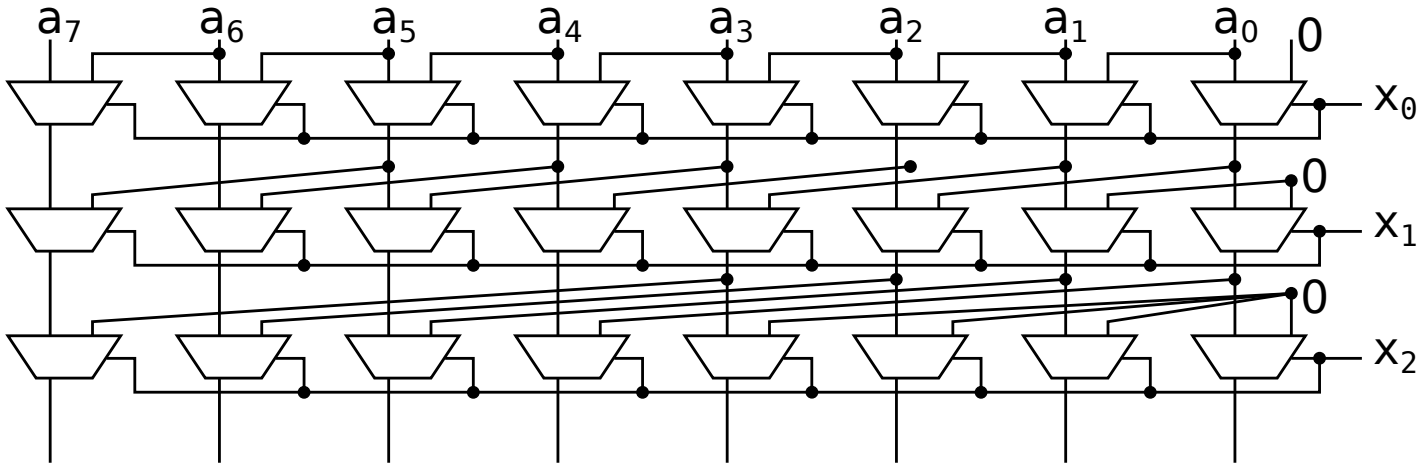
rotovat o 1, 2, 4, 8, 16 bitů

Porovnej algoritmus rychlého mocnění:

```
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);
  }
}
```

Barrel shifter

Rotaci o 0-7bitů lze složit z rotací o 1, 2, 4:



- Multiplexory v každé řadě vybírají buď nedělej nic nebo posuň o daný počet bitů.
- Bitové vstupy x_2, x_1, x_0 reprezentující binární číslo o kolik bitů rotovat
- Příklad:
  - posunutí o 5 bitů dostaneme složením posunutí o 1 bit a posunutí o 4 bity ($x_2 = 1$, $x_1 = 0$, $x_0 = 1$)
  - posunutí o 3 bity dostaneme složením posunutí o 1 bit a posunutí o 2 bity ($x_2 = 0$, $x_1 = 1$, $x_0 = 1$)

Kvíz

Kolik vrstev (řádek) Barrel shifter musí mít pro 64 bitové číslo, tedy pro rotace od 0 do 63? (Pro rotaci od 0 do 7 to byly 3 vrstvy)

A 4

B 6

C 16

D 64

Testovací kvíz

Kolik jste toho zatím pochopili?

A Vše bez problému.

B Téměř vše.

C Téměř nic.

D Vůbec nic.