Aritmetické algoritmy

Sčítání čísel

Procházka appletem

Scéna: *Sčítance*

Naším úkolem je sečíst dvě čísla napsaná v binární soustavě. Úkolem první scény je ukázat, jak se pracuje se sčítanci.

Oba sčítance mají 32 číslic; tento počet lze v rozumných mezích měnit v poli $\boxed{\text{N=}}$ na jinou mocninu 2. Knoflíkem $\boxed{\text{RND}}$ lze 1. nebo 2. sčítanci přiřadit jinou náhodnou hodnotu. V řadě případů také bude užitečné nastavit jeden sčítanec jako komplement druhého (0 proti 1 a 1 proti 0); to se provede odpovídajícími knoflíky $\boxed{\text{KMPL}}$. Nakonec lze hodnotu každé číslice změnit (z 1 na 0 a z 0 na 1) klepnutím myší.

Scéna: *Algoritmus ze základní školy*

Zdálo by se, že není třeba o sčítání podrobněji mluvit. Algoritmus se žáčci učí již na základní škole a zná jej každý gramotný člověk a způsob sám je velmi jednoduchý a na první pohled i rychlý.

Tato scéna umožňuje krokovat výpočet podle základní školy a ukazuje, že postup je ve skutečnosti pro větší čísla poměrně zdlouhavý (zkuste si jej pro 64 bitů). Jistě není třeba dodávat, že červená šipka označuje přenos do vyššího řádu a zelená tečka naopak naznačuje, že k přenosu nedošlo.

Při sčítání dvou čísel algoritmus postupuje zprava doleva a každý krok je vázán na výsledek předchozího (konkrétně na to, zda dojde k přenosu do vyššího řádu). Počet kroků je roven počtu číslic sčítanců.

Scéna: *Paralelní varianta algoritmu ze základní školy*

V moderních procesorech je k dispozici dostatečné množství logických obvodů, takže (na rozdíl od lidského počtáře) mohou provádět velké množství operací současně a tím urychlovat výpočet. Při současné šířce slova v počítačích, kdy 64 bitů se stává standardem, je takové zrychlení opravdu žádoucí. Zdá se ale, že elementární algoritmus pro sčítání nelze paralelním prováděním operací nijak urychlit.

Je zřejmé, že nejobtížnější část výpočtu je určit, ve kterých sloupcích dochází k přenosu. Pokud to víme, dá se výsledný součet určit *jediným* paralelním krokem, kdy pro každý sloupec sečtene dvě číslice sloupce a přenos z nižšího řádu modulo 2, neboli výsledek ve sloupci je 1 právě když buď 1 nebo 3 z těchto hodnot jsou rovny 1.

Při podrobnějším pohledu na algoritmus pro sčítání je zřejmé, že pro některé sloupce umíme rozhodnout, zda z nich bude vycházet přenos do vyššího řádu, aniž bychom čekali na výsledek z řádů nižších.

- Jestliže ve sloupci oba sčítance mají hodnotu 1, pak z takového sloupce bude přenos vycházet *vždy*, bez ohledu na to, zda z nižších řádů přenos přijde nebo ne. Budeme říkat, že sloupec *generuje*.
- Jestliže ve sloupci oba sčítance mají hodnotu 0, pak z takového sloupce přenos *nikdy* nevyjde, i kdyby do něho vstoupil přenos z nižšího řádu. Budeme říkat, že sloupec *absorbuje*.
- Nakonec jestliže sloupec je komplementární, tj. v jednom sčítanci je 0 a v druhém 1, pak sloupec vytváří přenos pouze pokud vytváří přenos sloupec napravo od něj. Budeme říkat, že sloupec *přenáší*.

Povšimněte si, že sloupec zcela vpravo nemá co přenášet, protože napravo od něj již další sloupec není. U praveého sloupce je proto vždy možno rozhodnout o přenosu bez znalosti hodnot v jiných sloupcích.

Pro rozdělení sloupců do typů není třeba znát informaci v jiných sloupcích a proto je snadno provedeme pro všechny sloupce současně a nezávisle v jednom paralelním kroku.

Zkuste nyní algoritmus krokovat. V prvním kroku se u sloupce vpravo a u všech sloupců, které generují nebo absorbují, určí, zda z nich vychází přenos. Zbývající (přenášející) sloupce vytvoří "ostrůvky" nerozhodnutých sloupců, oddělené rozhodnutými sloupci.

Šířka těchto ostrůvků bývá výrazně menší, než je šířka celých čísel a dá se dokázat, předpokládajíce náhodné a navzájem nezávislé hodnoty číslic sčítanců, že generujících a absorbujícíh sloupců bývá okolo jedné poloviny všech sloupců (jednoduchý důkaz) a šířka největšího ostrůvku je obvykle řádově kolem logaritmu šířky slova (značně složitý důkaz).

Můžete si zkusit opakovaně generovat náhodné sčítance a provádět první krok algoritmu a sledovat, jak ostrůvky nerozhodnutých sloupců vypadají.

Nyní budeme pokračovat krokování. V druhém kroku je možno o přenosu rozhodnout v pravých sloupcích ostrůvků a v každém dalším kroku se ostrůvky postupně zmenšují zpravo doleva, o 1 sloupec v každém kroku. Současně se přitom zpracovává tolik sloupců, kolik je dosud nerozhodnutých ostrůvků a doba paralelně prováděného výpočtu je proto dána maximální šířkou ostrůvku a je proto obvykle výrazně menší, než u základní varianty sčítacího algoritmu.

Scéna: *Paralelní varianta - nejhorší případ*

Jak bylo řečeno v předchozí scéně, doba nutná k paralelnímu provedení sčítání podle vykládaného algoritmu je dána šířkou největšího ostrůvku a je tedy silně závislá na konkrétním tvaru sčítanců.

V nejlepším případě, pokud například sčítáme dvě stejná čísla, ostrůvky vůbec nevzniknou a rozhodnutí o všech přenosech se dostane v jednom kroku a i v průměrném případě je algoritmus velmi rychlý.

V nejhorším případě však bohužel je tento algoritmus stejně rychlý (či lépe řečeno pomalý) jako sériově prováděný algoritmus ze základní školy. Nastává to v případě, kdy všechny sloupce (s případnou výjimkou sloupce vpravo) jsou komplementární. Pak totiž vznikne jen jeden ostrůvek, který má šířku o 1 menší než je šířka sčítanců a proto výpočet probíhá úplně stejně jako v základním algorimu a paralelismus je zcela nevyužit.

Tento případ generuje tato scéna; zkuste si výpočet krokovat, ale myslím, že jej '-'ani nedokončíte, protože trvá příliš dlouho a nic nového se při něm nedozvíte.

Scéna: *Bloky*

Nyní začneme vysvětlovat jiný algoritmus, obvykle označovaný jako carry look-ahead, který zaručeně i v nejhorším případě počítá velmi rychle, rychlostí srovnatelnou s rychlostí, kterou paralelizovaný elementární algoritmus počítá v obvyklém případě, a tedy podstatně rychleji než elementární algoritmus v základním provedení.

Základním prvkem pro nový algoritmus je blok. *Blok* je souvislý soubor sloupců, na displeji je označen žlutým obdélníkem. Šířku bloku je možno na displeji měnit tažením jeho levé nebo pravé strany myší. Polohu bloku lze měnit tažením bloku myší za libovolné jiné místo než je jeho pravý a levý okraj nebo okénko s číslicí. Je též možno požádat o vytvoření nového (náhodného) bloku.

U bloku nás pro dané hodnoty sčítanců bude zajímat, zda z jeho nejvyššího (tj. nejlevějšího) sloupce vychází přenos. Zda k tomu dojde však se budeme snažit určit pouze na základě těch číslic sčítanců, které jsou uvnitř bloku. To pochopitelně vždy nepůjde a proto budeme podobně jako už jsme to učinili výše, rozlišovat následující tři případy:

- blok *generuje*: některý sloupec bloku obsahuje dvě číslice 1 a všechny sloupce bloku, které leží nalevo od něho, jsou komplementární (obashují jednu 1 a jednu 0); tento případ budeme označovat symbolem '+'
- blok *absorbuje*: některý sloupec bloku obsahuje dvě číslice 0 a všechny sloupce bloku, které leží nalevo od něho, jsou komplementární (obsahují jednu 1 a jednu 0); tento případ budeme označovat symbolem '-'
- blok *přenáší*: všechny sloupce bloku jsou komplementární; tento případ budeme označovat symbolem '<' (s výjimkou případu, kdy blok zahrnuje sloupec zcela vpravo, kdy budeme používat symbol '-'.

Je jasné, že tyto tři případy se navzájem vylučují a jeden z nich vždy nastává. V prvním případě, kdy blok generuje, ze sloupce obsahujícího dvě 1 vyjde přenos a je přenášen komplementárními sloupci až k levému okraji bloku a ven z něho, proto z levého sloupce bloku přenos vychází. V druhém případě sloupec se dvěma nulami absorbuje případný přenos, který by do tohoto sloupce vstoupil z nižšího řádu a sloupce nalevo od něj samy o sobě přenos vytvořit nejsou schopny.

V nejzajímavějším třetím případě sloupce bloku samy přenos nevytvoří, ale pokud ze sloupce stojícího vpravo od bloku do bloku přenos vstoupí, sloupce bloku přenos přenesou celým blokem, až nakonec vystoupí z levého sloupce bloku dále. Jestliže však vpravo od bloku již žádný jiný sloupec neleží, nebude co přenášet, což vysvětluje výjimku v označování přenášejícího bloku.

Vyzkoušejte různé hodnoty sčítanců a šířku i polohu bloku a sledujte, která z možností '+', '-' a '<' nastává.

Scéna: *Jednosloupcové bloky*

Jak již bylo naznačeno výše, výpočet hodnoty bloku tvořeného jedním sloupcem je jednoduchý: sloupec obsahující dvě 1 generuje, pokud obsahuje dvě 0, pak absorbuje a komplementární blok zcela vpravo negeneruje a jinak přenáší.

Ověřte si tyto možnosti pro různé hodnoty sčítanců a polohu bloku, který je možno myší táhnout jako tomu bylo v předchozí scéně (ale nelze měnit jeho šířku).

Scéna: *Široké bloky*

Hodnotu bloku složeného z více než jednoho sloupce lze určit snadno podle definice: Postupujíce zleva doprava, najdeme první nekomplementární sloupec bloku. Obsahuje-li dvě 1, blok generuje, jinak absorbuje. Pokud je ale blok složen jen z kompementárních sloupců, pak přenáší nebo negeneruje podle toho, zda zasahuje až do pravého sloupce sčítanců.

Tento postup je sice jednoduchý, ale sekvenční a v nejhorším případě (komplementární operandy) by vedl ke stejně pomalému výpočtu jako předchozí dvě metody.

Budeme proto postupovat jinak:
rozdělíme blok *libovolným způsobem* na dva bloky obsahující oba alespoň jeden sloupec (stiskněte knoflík Krok v ovladači nazvaném Konstrukce),
určíme hodnoty podbloků a
spočteme hodnotu celého bloku následujícím způsobem:

- jestliže levý blok generuje, celý blok generuje také,
- jestliže levý blok absorbuje, celý blok absorbuje také,
- jestliže levý blok přenáší, celý blok má hodnotu rovnou hodnotě pravého bloku.

Pokud levý blok generuje nebo absorbuje, jeho rozhodující sloupec (nejlevější nekomplementární) je také rozhodujícím pro celý blok. Pokud levý blok přenáší, představuje pouze nedůležité přenášející sloupce pro pravý blok.

Hodnoty podbloků, které obsahují více sloupců se určí rekurzivně dalším dělením (používajíce opětovně knoflík Krok v ovladači Konstrukce).

Při paralelním výpočtu lze hodnoty levého i pravého bloku určovat *současně*.

Rekurzivní výpočet hodnoty bloku lze krokovat knoflíkem Krok v ovladači Výpočet, který se objeví v okamžiku, kdy je rekurzivní konstrukce stromu výpočtu ukončena.

Knoflíky Zpět v obou ovladačích vrací zpět akci knoflíku Krok a knoflíky Dokonči a Zruš umožňují příslušnou akci zcela dokončit nebo vrátit do původního stavu. Knoflík Široký umožňuje volit mezi dvěma způsoby kreslení vrcholů stromu rekurze. V prvním je vrchol natažen mezi krajními sloupci bloku, který reprezentuje, v druhém má šířku jednoho sloupce. Druhá metoda bude užívána ve scénách znázorňujících úplnou sčítačku, kde bude zobrazováno několik stromů najednou a široké vrcholy by daly nepřehledné schéma.

Scéna: *Nejhorší dělení*

Pro správnost výpočtu hodnoty bloku je zcela lhostejné, jak jej dělíme do podbloků. Způsob dělení je ale podstatný pro počet paralelních kroků, které je nutno provést k ukončení výpočtu hodnoty. Tato scéna ukazuje nejhorší možný způsob dělení: od celého bloku se vždy oddělí jediný sloupec vpravo nebo vlevo. Schéma výpočtu, které takto dostaneme pak má takovou hloubku, že se sotva vejde na obrazovku a vedlo by k algoritmu výpočetně srovnatelnému s algoritmem ze základní školy.

Zkuste si vytvořit široký blok a sledujte jeho strom výpočtu.

Scéna: *Optimální dělení*

Čtenář, který má alespoň základní znalosti v návrhu efektivních algoritmů, jistě ví nebo alespoň tuší, že pro rychlost paralelního výpočtu je optimální dělit blok na dva stejně velké podbloky poloviční velikosti (nebo na dva podbloky lišící se v počtu sloupců o 1 pokud blok sám má lichý počet sloupců a na přesné poloviny rozdělit nejde). Počet paralelních kroků pro výpočet hodnoty je pak $\lceil \log_2 w \rceil$, kde w je počet sloupců bloku a je to to nejlepší, co můžeme dosáhnout.

Zkuste si vytvořit široký blok a sledujte jeho strom výpočtu. Obměňujte blok i sčítance a sledujte konstrukci stromu výpočtu i odpovídající výpočet hodnot bloků.

Scéna: *Dělení podle binárních hranic*

Nyní popíšeme jiný způsob dělení, který je pro některé bloky mírně neoptimální, ale má výhody, které se stanou zřejmými později. V této scéně již je nutné předpokládat, že počet šířka operandů je mocninou dvojky (což v současné době v počítačové aritmetice nezpůsobuje prakticky žádná omezení; první mikroprocesory pracovaly se šířkou 8 bitů a záhy bily vystřídány 16-bitovými, nyní končí éra 32-bitových a tato kniha je už psána na 64-bitovém procesoru - vše jsou mocniny 2.

Blok rozdělíme nejprve podle hranice procházející polovinou šířky sčítanců (nikoli polovinou bloku), pak podél hranice procházející první nebo třetí čtvrtinou, pak 1., 3., 5. nebo 7. osminou atd. Pochopitelně jestliže některá hranice blokem neprochází (např. pokud je blok celý v dolní polovině čísel nebo ji přesahuje o jediný sloupec), pak může být odpovídající krok přeskočen.

Blok obsahující všechny sloupce je pochopitelně dělen optimálně na poloviny. Menší bloky sice mohou být děleny neoptimálně, ale počet úrovní dělení nebude nikdy větší než u všesloupcového bloku. Jelikož se všesloupcový blok v návrhu algoritmu bude vyskytovat, bude paralelní výpočetní složitost dána počtem úrovní rozkladu tohoto největšího bloku a optimalizací rozkladu bloků menších by se počet paralelních kroků výpočtu stejně nemohl zlepšit.

Dělení podle binárních hranic má ale výhodu, že všechny bloky jsou děleny v odpovídajících úrovních stejým způsobem, tedy podle stejných hranic, což se příznivě projeví na tvaru a jednoduchosti výsledného obvodu.

Scéna: *Určení přenosu v nejvyšším řádu*

V této scéně se budeme zabývat otázkou, zda z nejvyššího řádu, tedy ze sloupce úplně vlevo, vychází přenos. Obdobnou otázku budeme muset vyřešit i pro ostatní sloupce, pro levý sloupec je ale nejobtížnější, protože závisí na všech číslicích obou sčítanců.

Pro vyřešení tohoto problému stačí uvažovat blok, obsahující všechny sloupce, tedy prostírající se vlevo od sloupce, který zkoumáme, až ke sloupci, který leží úplně napravo (a tedy máme jistotu, že do něj zprava nemůže vstoupit přenos).

Tento všesloupcový blok nemá nikdy hodnotu '<' a už jsme rozebírali, že pokud generuje, pak z nejvyššího řádu vychází přenos, zatímco pokud absorbuje nebo přenáší, pak z nejvyššího řádu přenos nevychází. Naše otázka se tedy snadno zodpoví určením hodnoty bloku, kterou určujeme na základě dělení bloku podle binárních hranic jak bylo popsáno v předchozí scéně.

Knoflíky ovladače Konstrukce si nyní rozviňte obvod pro výpočet hodnoty bloku a knoflíky ovladače Výpočet si určujte hodnotu bloku i podbloků vzniklých dělením a tedy i existenci přenosu v nejvyšším řádu pro různé hodnoty sčítanců. Výpočet je krokován po jednotlivých paralelních krocích.

Nyní již je tedy patrna hlavní myšlenka algoritmu: současně s prováděním početních úkonů v pravé části schématu, která zahrnuje méně významné bity, se zpracují levé poloviny sčítanců. Pokud se zjistí, že levá polovina generuje nebo absorbuje, není třeba znát výsledek z pravé části. Nové na algoritmu je ale to, že pokud levá polovina přenáší, pak případný přenos z pravé poloviny sčítanců není nutné "postrkovat" levou částí sloupec po sloupci, jako to činí elementární algoritmus ve své školní podobě vždy a v paralelizované podobě přinejmenším v nejhorším případě, ale přenos "přeskočí" z hranice v polovině sčítanců do nejlevějšího sloupce okamžitě, protože víme, že by k tomu při detailním počítání stejně došlo.

Je vidět, že výpočet je velmi rychlý - počet vrstev obvodu a tedy i počet paralelních kroků potřebných pro výpočet je $\lceil \log_2 n \rceil$, kde n je šířka sčítanců. Obzvláště důležité je to, že výpočet *nezávisí* na hodnotách sčítanců a je tedy takto rychlý i v nejhorším případě.

Scéna: *Určení obecného přenosu*

Podobně jako byl v předchozí scéně určen přenos v nejvyšším řádu, ukážeme si, jak provést určení přenosu v libovolném jiném sloupci. Výběr sloupce pro ilustraci metody provedeme kliknutím na malý bledý čtvereček nad sloupcem.

Jako v předchozí scéně stačí určit hodnotu bloku, určeného vlevo uvažovaným sloupcem a vpravo zasahující až na samé pravý okraj sčítanců. Takovýto blok nikdy nemůže dostat zprava přenos z nižšího řádu, protože vpravo od něj již nic není, a v naší klasifikaci nikdy nenabývá hodnoty '<'. Proto z jeho levého sloupce, tedy z našeho uvažovaného sloupce, vychází přenos právě když tento blok generuje.

Při konstrukci si povšimněte, že blok je dělen podle binárních hranic. Jak již bylo uvedeno, je pro nás dostačující aby počet vrstev při dělení bloku nepřekročil počet vrstev, vypočítávajících hodnotu všesloupcového bloku.

Scéna: *Překrývání obvodů*

Nyní již víme, jak zkonstruovat úplnou sčítačku: pro každý sloupec si sestrojíme obvod, určující zda z tohoto sloupce vychází přenos, způsobem popsaným v předchozích dvou scénách a pak z informace o přenosech určíme v jednom paralelním kroku hodnotu součtu. Všechny obvody pro výpočty přenosů pracují současně.

V této scéně se konečně dozvíte důvod pro dělení bloků podle binárních hranic. Na displeji je zobrazen obvod pro výpočet přenosu pro levý sloupec. Klepněte nyní na malý bledý čtvereček nad libovolným jiným sloupcem (výhodně nad sloupcem v levé části schématu). Tím se zobrazí obvod pro výpočet přenosu pro zvolený sloupec. Podle stavu konstrukce (knoflíky ovladače Konstrukce) budou obvody v různém stavu vývinu od prostého bloku po úplně dokončený obvod pro výpočet jeho hodnoty.

Nyní již určitě vidíte, proč preferujeme dělení podle binárních hranic. Při tomto způsobu se obvody pro určování přenosu v různých sloupcích podstatně *překrývají.* Není tedy nutné budovat tyto obvody zcela odděleně, což by vedlo k velkému plýtvání hardwarem, ale překrývající se část vytvoříme pouze jednou a její výsledek pak rozvádíme na vstup více než jednoho následného hradla.

Klepáním do čtverečků nad sloupci je možno zobrazovat (a nebo naopak skrývat) libovolný sloupec sledovat překrývání obvodů pro výpočet přenosů. Je vidět, že překrývání je opravdu velké, například všechny sloupce z levé části mají shodnou pravou část svých obvodů. Nakonec se zjistí, že počet hradel pro *všechny* sloupce není o moc větší než je obvod pro levý, nejsložitější, sloupec.

Je též možno zvolit nebo vypustit několik sloupců najednou: stiskněte myš v bledém čtverci nad některým sloupcem a tahněte myš do strany.

Scéna: *Kompletní sčítačka carry look-ahead*

Tato scéna již jen shrnuje, co se v předchozí scéně objevilo při volbě všech sloupců a plně rozvinuté konstrukci, umožňuje ale krokování výpočtu pro různé hodnoty sčítanců.

Teprve na úplné sčítačce je vidět, že je její konstrukce vlastně jednoduchá, je-li formulována rekurzivně: Vytvoříme dva obvody pro výpočet přenosů pro poloviční šířku slova, mírně zobecněné možností přivádět kromě dvou sčítanců ještě zprava přenos za sousedního obvodu. Pravý podobvod již dává polovinu konečných výstupů, výstupy levého podobvodu se ještě všechny zkombinují s nejvýznamnějším výstupem pravé části.

Povšimněte si, že sčítačka má $\log_2 N$ vrstev a v důsledku velkého překrývání je v každé vrstvě přesně $N/2$ hradel, které kombinují třístavové hodnoty jednotlivých bloků.

Scéna: *Layered Adder*

V kompletní sčítačce ukázané v předchozí scéně byly hradla (vrcholů stromů) uspořádána do vodorovných vrstev, ale jistá propojení mezi (žlutými) hradly přecházela přes několik vrstev. V této scéně je sčítačka doplněna dalšími hradly, vybarvenými růžově, která zaplňují "díry" v obvodu. Růžová hradla neprovádějí žádný výpočet a jsou to ve skutečnosti paměťové elementy, které si mohou zapamatovat jednu třístavovou hodnotu. V každém kroku (kliknutí knoflíku Krok) růžové hradlo přečte hodnotu vstupu a zapamatuje si jej. Výstup je vždy roven uchovávané hodnotě.

Růžové prvky jsou velmi podobné tomu, co se v elektronice nazývá klopný obvod typu D; jediný rozdíl je, že si klopný obvod pamatuje jediný bit. Třístavovou veličinu je ale možno uchovávat ve dvou bitech a proto se naše růžové prvky obvykle implementují dvojicí klopných obvodů typu D.

Zkuste si znovu počítat hodnoty přenosů; výpočet probíhá stejně jako v předchozí scéně, ale vstupní hodnoty přicházejí "just in time" a nemusí čekat jako se to stávalo v minulé scéně.

Je vidět, že postup mezivýsledků obvodem je velmi pravidelný, v každém kroku jedna nová vrstva určí své hodnoty. V následujících scénách uvidíme, proč je to užitečné.

Scéna: *Zapomínající sčítačka*

Jak bylo uvedeno v minulé scéně, tok informace v obvodu je velmi pravidelný. V každém kroku je aktivní jedna vrstva, která vypočte své hodnoty z hodnot v *předchozí* vrstvě. Protože nás meyivýsledky ve skutečnosti nezajímají a címel je určit hodnoty poslední vrstvy, je možné mezivýsledky zapomenout, jakmile byly použity následující vrstvou.

Zkuste znovy provádět výpočet a uvidíte, že je dostatečné si uchovávat pouze mezivýsledky jediné vrstvy.

Scéna: *Zřetězená sčítačka*

V předchozí scéně jsme viděli, že stačí si pamatovat vždy pouze výsledky jediné vrstvy. Ty vrstvy, jejichž jejichž výsledky již byly použity, "zahálí" a je je možno využít k sčítání jiných dvojic sčítanců. Jakmile hodnoty pro jeden pár postoupí o jednu vrstvu dolů, je možné do obvodu vložit nový pár sčítanců, který postupuje obvodem se zpožděním jedné vrstvy.

V této scéně sčítáme plynulý proud dvojic sčítanců a každý pár má svoji barvu, která označuje jemu odpovídající mezivýsledky. Prvky obvodu mají svoji původní barvu pouze dokud k nim nedorazí první mezivýsledky, pak jsou již označovány barvou páru, který právě zpracovávají. Při krokování je jasně vidět, jak informace obvodem postupuje.

Sčítačka typu carry look-ahead je tedy velmi silný nástroj, kromě toho, že jeden pár v obvodu stráví pouze čas úměrný logaritmu počtu bitů a doba trvání výpočtu je zcela nezávislá na jejich hodnotách, mohou do obvodu sčítance vstupovat bezprostředně za sebou s odstupem, který na velikosti sčítanců nezávisí a je dán pouze výpočetním zpožděním použitých výpočetních prvků.

Scéna: *Animovaná sčítačka*

Tato scéna přináší inženýrský náhled na implementaci zřetězené sčítačky z předcházející scény. Růžové paměťové prvky jsou nezměněny, pouze jsou nakresleny poněkud menší. Na druhé straně každý žluté výpočetní hradlo je doplněno o jeden růžový paměťová prvek stejného typu jako růžové prvky popsané v předchozích scénách.

Žlutá výpočetní hradla pracují *asynchronně*: jakmile se změní vstupní hodnoty, odpovídajícím způsobem se ihned změní i výstup. Přesněji řečeno, změna se objeví na výstupu po jistém časovém okamžiku, zpoždění, které může být *různé* pro různá hradla.

Pokud bychom zpracovávali pouze jeden pár sčítanců, růžové paměťové prvky by nebyly třeba, ale rychlost postupu informace by byla v různých částech obvodu jiná. To by ale v případě zřetězeného zpracování více párů sčítanců vedlo k tomu, že by nebylo možné hodnoty odpovídající různým párům na výstupu obvodu odlišit.

Z tohoto důvodu musíme používat *synchronní* obvod a synchronizace je dosahována růžovými paměťovými prvky ve všech pozicích obvodu. Výstup paměťového obvodu je roven uchovávané hodnotě; po většinu doby prvek ignoruje vstupní hodnotu. Paměťové prvky však dostávají hodinový signál (jeho přivádění k prvkům není znázorněno). Jakmile paměťový prvek dostane hodinový signál (v reálných systémech například vzestupná hrana hodinového signálu, u našeho appletu kliknutí knoflíku Krok), prvek přečte vstupní hodnotu a nahradí jí dříve uchovávanou hodnotu. Po této mžikové události se zapamatovaná hodnota dále nemění až do dalšího hodinového impulzu.

Zkuste si výpočet s použitím knoflíku Krok; v okamžiku hodinového impulzu (kliknutí) všechny paměťové prvky najednou nahradí původní uchovávanou hodnotu hodnotou, která byla na vstupu. Animace ukazuje, jak informace postupuje podél vodičů, připojujících výstupy paměťových prvků se vstupy následujících hradel, kterými mohou být výpočetní hradla nebo další paměťové prvky.

Jakmile informace dorazí na výpočetní hradlo, je zpracována a s jistým zpožděním se výsledek objeví na jeho výstupu (zpoždění jsou různá pro různá hradla - rozdíly jsou úmyslně zvýrazněny), což se projeví m.j. změnou barvy hradla.

Barvy hradel a signálů reflektují barvu odpovídajícího páru sčítanců, ke kterému se vztahují. Je tedy snadné sledovat postup informace obvodem.

Jistě je vám zřejmé, že hodinový kmitočet musí být takový, aby doba mezi dvěma po sobě následujícími hodinovými pulzy byla dostatečná pro přenos signálu mezi hradly a jeho zpracování. Náš applet je silně synchronizován a nepřijímá kliknutí, dokud neprovede všechny nutné operace, ale v případě reálného obvodu by příliš rychlé hodiny vedly k chybným výpočtům.

Scéna: *Číslování*

Tato scéna již není součástí procházky appletem, ale bude sloužit pro ilustraci definic a pojmů, které budou zavedeny v laboratoři a ve škole. Nechte tuto scénu na obrazovce a pokračujte v čtení této kapitoly.

Laboratoř

Připomeňme, že booleovské funkce $\vee$, $\wedge$ a $\oplus$ jsou definovány takto

x	y	$x \vee y$	$x \wedge y$	$x \oplus y$
0	0	0	0	0
0	1	1	0	1
1	0	1	0	1
1	1	1	1	0

Bit je proměnná, která má dvě možné hodnoty 0 a 1. Nezáporné celé číslo (unsigned integer) je reprezentováno v binární poziční soustavě posloupností $b_{n-1} \ldots b_1 b_0$ bitů, která představuje číslo

$$b_{n-1} \cdot 2^{n-1} + \cdots + b_1 \cdot 2 + b_0.$$

Cílem je nalézt součet $s_n \ldots s_1 s_0$ dvou čísel $a_{n-1} \ldots a_1 a_0$ a $b_{n-1} \ldots b_1 b_0$, který je definován takto

$$s_0 = a_0 \oplus b_0, \qquad s_i = a_i \oplus b_i \oplus c_{i-1} \quad \text{pro } i = 1, \ldots, n-1, \qquad s_n = c_{n-1},$$

kde

$$c_0 = a_0 \wedge b_0 \qquad \text{a} \qquad c_i = (a_i \wedge b_i) \vee (a_i \wedge c_{i-1}) \vee (a_i \wedge c_{i-1}) \quad \text{pro } i = 1, \ldots, n-1.$$

Ve zbývající části kapitoly budeme předpokládat, že n je mocnina 2 a označíme $k = \log_2 n$, t.j. $n = 2^k$.

Jelikož digitální obvody používají prakticky výhradně dvouhodnotové veličiny, hodnoty $+$, $-$ a $<$, které byly používány v procházce appletem se nahradí dvojicí (g, p) bitů následujícím způsobem:
$(1, 0)$ a $(1, 1)$ oba představují $+$ (generuje),
$(0, 1)$ představuje $<$ (přenáší),
$(0, 0)$ představuje $-$ (absorbuje),

Páry bitů se kombinují použitím následující operace, která odpovídá funkci výpočetních hradel, které byly používány v procházce appletem:

$$(g_1, p_1) \bullet (g_2, p_2) = (g_1 \vee (p_1 \wedge g_2), p_1 \wedge p_2)$$

Nechť $\mathcal{P}$ je množina všech dvojic (s, j), kde čísla s, j jsou celá čísla $0 \le s < k$ a $0 \le j < n$. Dvojice označují polohy žlutých hradel počítajících operaci $\bullet$ a růžových zpožďovacích paměťových bloků v sčítačce z procházky appletem. $\mathcal{G}_{s,j}$ je hradlo v s-té řádce a j-tém sloupci sčítačky při číslování, jak je uvedeno v poslední scéně procházky. Vstupy hradel z i-tého řádku jsou výstupy z řádku $i-1$ a proto je přidána ještě řádka označená -1, představující kombinace číslic v jednotlivých sloupcích operandů.

Je-li $(s, j) \in \mathcal{P}$, označíme jako $\xi(s, j)$ s-tou číslici binární reprezentace čísla j (kde 0 je nejméně významá číslice). Hodnoty $\xi(s, j)$ jsou nakresleny v hradlech v obvodu v poslední scéně procházky. Povšimněte si, že pro každý sloupec je pořadové číslo sloupce dáno binárním číslem tvořeným hodnotami ξ nakreslenými ve sloupci v řádcích označených $k-1, \ldots, 0$ (tedy namísto zleva doprava se číslo čte zdola nahoru).

Povšimněte si, že $\xi(s, j) = 1$ právě tehdy, když $\mathcal{G}_{s,j}$ je hradlo vypočítávající funkci $\bullet$, pokud $\xi(s, j) = 0$, pak $\mathcal{G}_{s,j}$ je paměťový prvek.

Pokud je $(s, j) \in \mathcal{P}$, definujme $\kappa(s, j)$ jako největší číslo dělitelné 2^s, které je menší nebo rovné j. Každému programátorovi je jasné, že za použití syntaxe jazyka C je $\kappa(s, j)$ rovné $j - j\%S$ a také $S * (j/S)$, kde $S = 2^s$, % je funkce zbytku při celočíselném dělení a / představuje celočíselné dělení. Jinými slovy, $\kappa(s, j)$ se dostane tak, že se s nejméně významných bitů čísla nahradí nulami. Jelikož je každé číslo dělitelné číslem $2^0 = 1$, je $\kappa(0, j) = j$ pro každé j. Pro $j, n = 2^k$ je celočíselný podíl j/n je roven 0 a proto je $\kappa(k, j) = 0$. Zřejmě také $s_1 < s_2$ implikuje $\kappa(s_1, j) > \kappa(s_2, j)$.

Navíc hradla v j-tém řádku, která jsou vpravo od $\mathcal{G}_{s,j}$, mají stejnou barvu jako $\mathcal{G}_{s,j}$ až do sloupce $\kappa(s, j)$.

Stiskněte myš v s-tém řádku v j-tém sloupci poslední scény procházky; všechna hradla, která přispívají k hodnotě zvoleného hradla $\mathcal{G}_{s,j}$ zčervenají. Začervenĕná hradla řádku -1 tvoří souvislý pás od sloupce j do sloupce $\kappa(s + 1, j)$.

Vlastnosti uvedené v předchozích odstavcích vysvětlují důvod, proč jinak divně vypadající funkci $\kappa(s, j)$ definujeme.

Je-li dáno s a j takové, že $-1 \leq s < k$ a $0 \leq j < n$, definujme pár bitů $\mathcal{V}_{s,j}$ následujícím způsobem:

$$\mathcal{V}_{-1,j} = (a_j \wedge b_j, a_j \vee b_j),$$

$$\mathcal{V}_{s,j} = \mathcal{V}_{s-1,j} \quad \text{pro každé } s \geq 0 \text{ a } \xi(s, j) = 0,$$

$$\mathcal{V}_{s,j} = \mathcal{V}_{s-1,j} \bullet \mathcal{V}_{s-1,\kappa(s,j)-1} \quad \text{pro každé } s \geq 0 \text{ a } \xi(s, j) = 1.$$

Povšimněte si, že $\mathcal{V}_{s,j}$ je hodnota hradla $\mathcal{G}_{s,j}$, protože definice popisuje jak je carry look-ahead sčítačka konstruována. Ověřte si, že v případě $\xi(s, j) = 1$ je pravý vstup hradla $\mathcal{G}_{s,j}$ skutečně napojen do sloupce $\kappa(s, j) - 1$.

Bude dokázáno, že j-tý sloupec vytváří přenos, to jest $c_j = 1$ ve výše uvedeném označení, právě tehdy, když první bit $\mathcal{V}_{k,j}$ je 1. Jestliže tedy první (levý) bit g páru bitů $A = (g, p)$ se označí jako $A^{(L)}$, pak číslice hledaného součtu se vypočtou takto

$$s_0 = a_0 \oplus b_0, \quad s_j = a_j \oplus b_j \oplus \mathcal{V}^{(L)}_{k,j-1} \quad \text{for } j = 1, \ldots, n.$$

Škola

Naším cílem je dokázat, že sčítačka, která byla neformálně popsána v procházce a velmi formálně a matematicky přesně definována v laboratoři, funguje správně. Budou používány všechny definice a pojmy z laboratoře.

Lemma 1 *Operace $\bullet$ je asociativní.*

Důkaz: S využitím asociativity a distributivity operací $\wedge$ a $\vee$ dostaneme

$$(g_1, p_1) \bullet ((g_2, p_2) \bullet (g_3, p_3)) = (g_1, p_1) \bullet (g_2 \vee p_2 \wedge g_3, p_2 \wedge p_3) =$$

$$= (g_1 \vee p_1 \wedge (g_2 \vee p_2 \wedge g_3), p_1 \wedge (p_2 \wedge p_3)) = (g_1 \vee (p_1 \wedge g_2 \vee p_1 \wedge (p_2 \wedge g_3))), (p_1 \wedge p_2) \wedge p_3) =$$

$$= ((g_1 \vee p_1 \wedge g_2) \vee p_1 \wedge (p_2 \wedge g_3)), (p_1 \wedge p_2) \wedge p_3) = ((g_1 \vee p_1 \wedge g_2) \vee (p_1 \wedge p_2) \wedge g_3), (p_1 \wedge p_2) \wedge p_3) =$$

$$= (g_1 \vee p_1 \wedge g_2, p_1 \wedge p_2) \bullet (g_3, p_3) = ((g_1, p_1) \bullet (g_2, p_2)) \bullet (g_3, p_3).$$

♣

Definujme nyní

$$\mathcal{C}_{i,i} = (a_i \wedge b_i, a_i \vee b_i) \quad \text{for } 0 \le i < n,$$

$$\mathcal{C}_{i,j} = \mathcal{C}_{i,i} \bullet \cdots \bullet \mathcal{C}_{j,j} \quad \text{for } 0 \le j < i < n.$$

Vzhledem k asociativitě $\bullet$ není nutné popisovat, jak jsou výrazy $\mathcal{C}_{*,*}$ kombinovány v definici $\mathcal{C}_{i,j}$ pro $j < i$.

Kromě toho nechť levý a pravý bit dvojice $\mathcal{C}_{i,j}$ jsou označeny $g_{i,j}$ a $p_{i,j}$, to znamená $\mathcal{C}_{i,j} = (g_{i,j}, p_{i,j})$.

Lemma 2 $c_i = g_{i,0}$ *pro* $j = 0, \ldots, n-1$

Důkaz: Indukcí podle i. Pro $i = 0$,

$$c_0 = a_0 \wedge b_0 = g_{0,0}$$

na základě definicí c_0 a $g_{i,i}$ for $i = 0$.

Jestliže $i > 0$ a lemma platí pro $i - 1$, to znamená $\mathcal{C}_{i-1,0} = (c_{i-1}, p_{i,0})$, pak

$$\mathcal{C}_{i,0} = \mathcal{C}_{i,i} \bullet \mathcal{C}_{i-1,0} = (a_i \wedge b_i, a_i \vee b_i) \bullet (c_{i-1}, p_{i,0}) =$$

$$= (a_i \wedge b_i \vee (a_i \vee b_i) \wedge c_{i-1}), (a_i \vee b_i) \wedge p_{i,0}),$$

a proto

$$g_{i,0} = a_i \wedge b_i \vee (a_i \vee b_i) \wedge c_{i-1}) =$$

$$= a_i \wedge b_i \vee a_i \wedge c_{i-1} \vee b_i \wedge c_{i-1} = c_i.$$

♣

Celočíselné dělení čísla j číslem 2^s není nic jiného, než odstranění s nejméně významných bitů z binární reprezentace čísla j. Násobení číslem 2^s znamená přidání s nul na konec binární reprezentace čísla. Na druhé straně je číslo dělitelné 2^s právě když jeho s nejméně významných bitů jsou nuly a proto $\kappa(s,j)$, největší číslo menší nebo rovné j a dělitelné číslem 2^s,se získá z j nahrazením jeho s nejméně významných bitů nulami. Tatáž operace se dostane celočíselným dělením 2^s (odebrání s nejméně významných bitů) a pak násobením 2^s (přidáním s nul):

Lemma 3 *Jestliže* $(s,j) \in \mathcal{P}$, *pak* $\kappa(s,j) = 2^s(j \ \mathbf{div}\ 2^s)$, *kde* **div** *znamená celočíselné dělení.*

Důkaz: Pokud j **div** 2^s se označí jako q, pak $j = 2^s q + r$, kde r je celé číslo takové, že $0 \leq r < 2^s$. Největší číslo menší nebo rovné j, které je dělitelné číslem 2^s, je $2^s q = 2^s(j \textbf{ div } 2^s)$. ♣

Náhrada s nejméně významných bitů nulami může být získána nahrazením $s-1$ nejméně významných bitů nulami a pak odečtením 2^{s-1} pokud $(s-1)$-tý nejméně významný bit (tedy $\xi(s-1,j)$) je 1:

Lemma 4 *Nechť* $(s,j) \in \mathcal{P}$ *a* $s > 0$*;*
je-li $\xi(s-1,j) = 1$, *pak* $\kappa(s,j) = \kappa(s-1,j) - 2^{s-1}$, *a*
je-li $\xi(s-1,j) = 0$, *pak* $\kappa(s,j) = \kappa(s-1,j)$.

Důkaz: Pro jistá celá čísla r, q taková, že $r \geq 0$ a $2^{s-1}\ qge0$, je $j = r2^s + \xi(s-1,j)2^{s-1} + q$, a proto

$$j \textbf{ div } 2^s = r \quad \text{a} \quad \kappa(s,j) = r2^s,$$

$$j \textbf{ div } 2^{s-1} = 2r + \xi(s-1,j),$$

$$\kappa(s-1,j) = 2^{s-1}(2r + \xi(s-1,j)) = r2^s + 2^{s-1}\xi(s-1,j).$$

♣

Lemma 5 $\kappa(s, \kappa(s,j) - 1) = \kappa(s,j) - 2^s$ *pro každé* $(s,j) \in \mathcal{P}$.

Důkaz: Nechť $i = \kappa(s,j)$. Pak i je dělitelné 2^s na základě definice $\kappa(s,j)$, ale $i-1$ je menší a proto největší celé číslo menší nebo rovné $i-1$ a dělitelné 2^s je nejbližší nižší násobek 2^s, tedy $i - 2^s$, a proto

$$\kappa(s, \kappa(s,j) - 1) = \kappa(s, i-1) = i - 2^s = \kappa(s,j) - 2^s.$$

♣

Lemma 6 $\mathcal{V}_{s-1,j} = \mathcal{C}_{j,\kappa(s,j)}$ *pro* $0 \leq s \leq k$ *a* $0 \leq j < n$.

Důkaz: Indukcí podle s. Jelikož $2^0 = 1$ dělí libovolné celé číslo, je $\kappa(0,j) = 1$ a proto

$$V_{-1,j} = (a_j \wedge b_j, a_j \vee b_j) = \mathcal{C}_{j,j} = \mathcal{C}_{j,\kappa(0,j)}.$$

Nechť nyní $0 < s \leq k$ a lemma platí pro $s-1$ a libovolné j.
Pokud $\xi(s-1,j) = 0$, pak

$$\mathcal{V}_{s,j} = \mathcal{V}_{s-1,j} = \mathcal{C}_{j,\kappa(s,j)} = \mathcal{C}_{j,\kappa(s+1,j)},$$

pokud $\xi(s-1,j)=1$, pak

$$\kappa(s,\kappa(s,j)-1)=\kappa(s,j)-2^s=\kappa(s+1,j)$$

podle Lemma 5 a Lemma 4, a

$$\mathcal{V}_{s,j}=\mathcal{V}_{s-1,j}\bullet\mathcal{V}_{s-1,\kappa(s,j)-1}=\mathcal{C}_{j,\kappa(s,j)}\bullet\mathcal{C}_{\kappa(s,j)-1,\kappa(s,\kappa(s,j)-1)}=$$

$$=\mathcal{C}_{j,\kappa(s,j)}\bullet\mathcal{C}_{\kappa(s,j)-1,\kappa(s+1,j)}=\mathcal{C}_{j,\kappa(s+1,j)}.$$

♣

Jako důsledek dostáváme, že za předpokladu, že $n=2^k$ pro některé celé číslo k dostáváme

Věta 7 *Pro každé $i=0,\ldots,n-1$, je přenos c_i levým bitem dvojice $\mathcal{V}_{k-1,i}$.*

Důkaz: Jelikož $i<2^k$, největší celé číslo menčí nebo rovné i a dělitelné číslem 2^k je 0 a proto $\kappa(k,i)=0$ a z Lemma 6 plyne, že

$$\mathcal{V}_{k-1,i}=\mathcal{C}_{i,\kappa(k,i)}=\mathcal{C}_{i,0}$$

a v důsledku Lemma 2 je přenos c_i roven levému bitu dvojice $\mathcal{V}_{k-1,i}$. ♣

Dokázali jsme, že informace o přenosu může být získána výpočtem všech hodnot $\mathcal{V}_{s,j}$ a extrakcí přenosů z hodnot $\mathcal{V}_{k-1,*}$. Výpočet může být proveden ve fázích $0,1,\ldots,k$; ve fázi $s\geq 0$ se hodnoty $\mathcal{V}_{s,j}$, $j=0,\ldots,n-1$ vypočtou v jednom paralelním kroku z hodnot $\mathcal{V}_{s-1,i}$, $i=0,\ldots,n-1$, které jsou již určeny. Dostáváme tedy

Věta 8 *Dvě N-ciferná binární čísla mohou být sečtena v čase $O(\log N)$.*