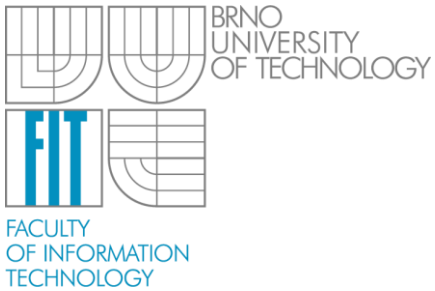
Návrh číslicových systémů (INC)

Tomáš Martínek

Vysoké učení technické v Brně
Fakulta informačních technologií
Božetěchova 2, 612 66 Brno



Od algoritmu k číslicovému obvodu

Osnova

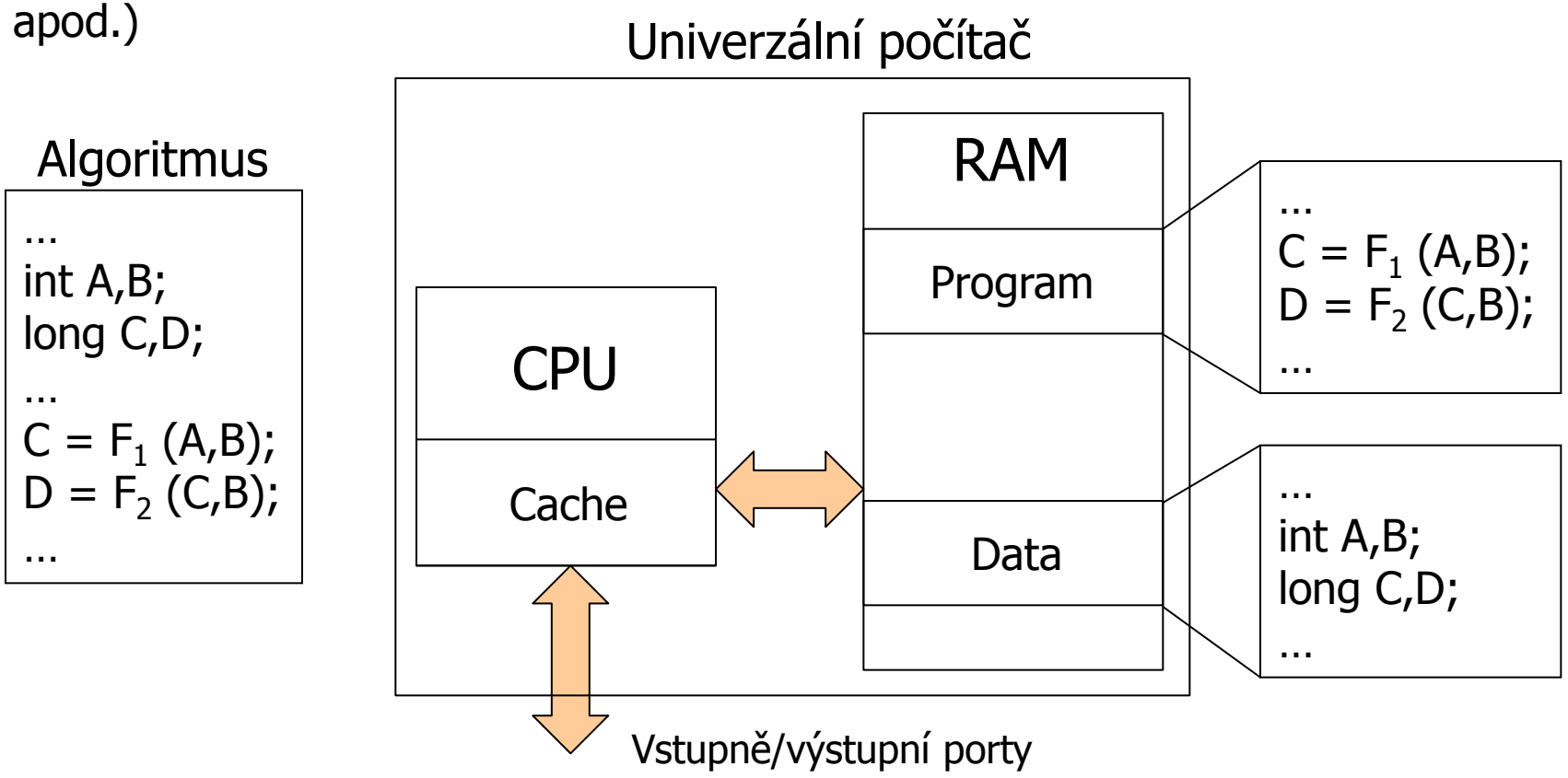
- Úvod
- Transformace základní konstrukcí
  - Sekvence
  - Selekce
  - Iterace
- Shrnutí

Co je to algoritmus?

- Intuitivně
  - Postup, který nás dovede k řešení úlohy
- Formálně
  - Přesně definovaná konečná posloupnost příkazů (kroků), jejichž prováděním pro každé přípustné vstupní hodnoty získáme po konečném počtu kroků odpovídající výstupní hodnoty [z kurzu Základy programování]
- Algoritmus je sestaven na základně
  - Datových struktur – proměnné, záznamy, pole, lin. seznamy, apod.
  - Řídicích struktur – sekvence, podmínka, iterace

Algoritmus implementovaný v software

- Výpočet běží na univerzálním procesoru (CPU)
- Datové struktury i předpis programu jsou uloženy v paměti RAM
- Na základě lokality jsou data i program přesouvány mezi pamětí RAM a interní cache procesoru
- Vstupy/Výstupy – dodávány skrze V/V zařízení (např. disk, monitor, porty, apod.)

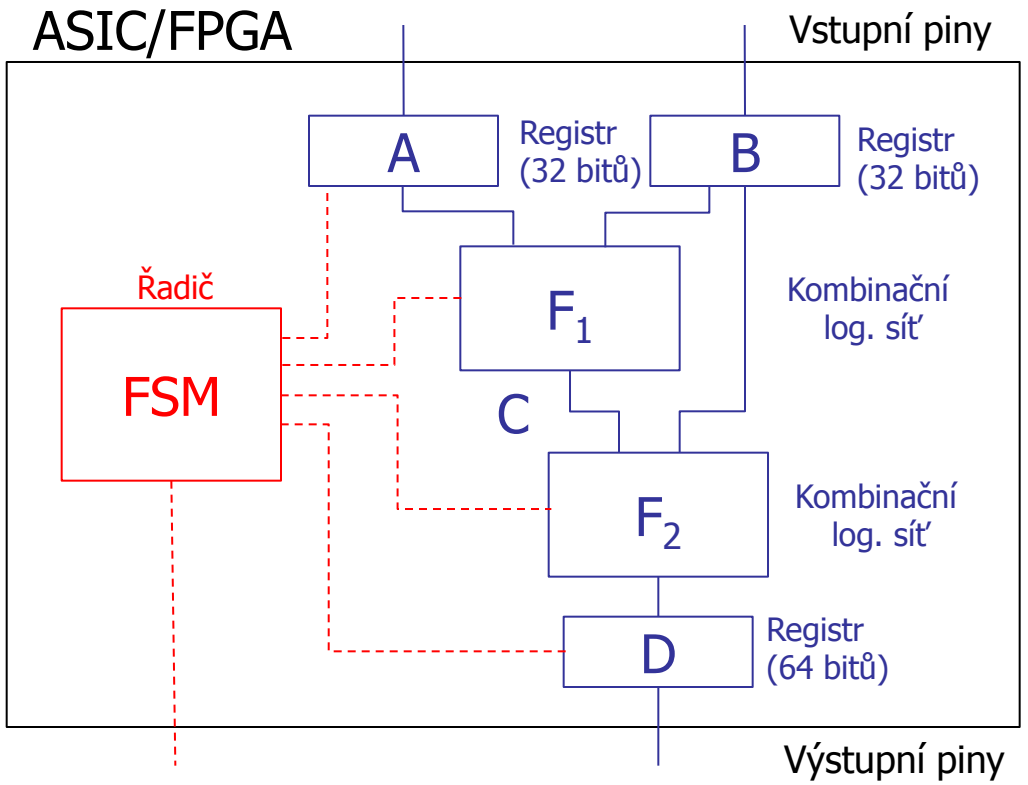


Algoritmus implementovaný v hardware

- Výpočet běží na specializovaném čipu (obvykle ASIC nebo FPGA)
- Datové struktury – uloženy v registrech nebo paměťových blocích
- Řídicí struktury – výpočet realizovat skrze datovou cestu (funkční jednotky), která je ovládána řadičem (FSM)
- Vstupy/výstupy – dostupné skrze vstupní/výstupní piny obvodu

Algoritmus

```
...
int A,B;
long C,D;
...
C = F1 (A,B);
D = F2 (C,B);
...
```



Od algoritmu k číslicovému obvodu

- Dijkstra dokázal, že libovolný algoritmus lze zaznamenat skrze kombinaci tří řídicích struktur:
  - Sekvence (posloupnost)
  - Selekce (podmínka)
  - Iterace (cyklus)

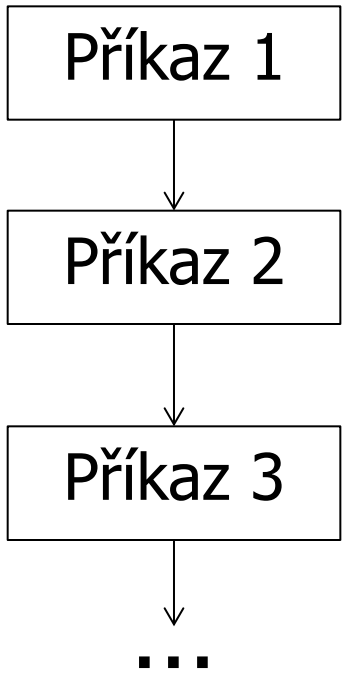
- Pojďme si ukázat, jak lze tyto struktury realizovat v hardware

Osnova

- Úvod
- Transformace základní konstrukcí
  - Sekvence
  - Selekce
  - Iterace
- Shrnutí

Sekvence příkazů

- Sekvence příkazů
  - Řada za sebou navazujících kroků (příkazů)
- Datová závislost
  - Dvojice příkazů je datově závislá, pokud vstup jednoho příkazu závisí na výstupu druhého příkazu nebo naopak
  - Jinak jsou příkazy datově nezávislé
- Příklady:
  - Datově závislé příkazy
    - Druhý příkaz potřebuje hodnotu C z prvního příkazu
  - Datově nezávislé příkazy
    - Příkazy nepotřebují výstup jiného příkazu



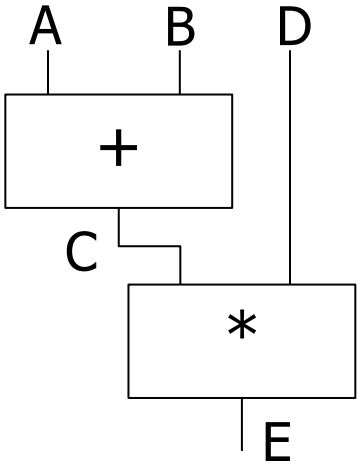

```
C = A + B
E = C * D
```

```
C = A + B
E = A * D
```

Sekvence příkazů

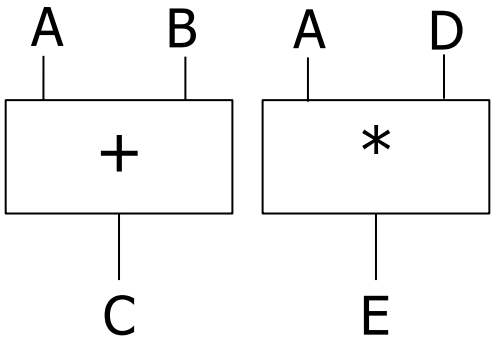
- Realizace sekvence příkazů v hardware
  - Nechť každý z příkazů je realizován určitým číslicovým obvodem/blokem (např. kombinační logickou sítí). Potom:
  - Datově závislé příkazy jsou realizovány sekvenčním zapojením bloků jednotlivých příkazy
  - Datově nezávislé příkazy mohou být realizovány paralelním zapojením bloků jednotlivých příkazů
- Příklady:

C = A + B
E = C * D



Datově závislé příkazy

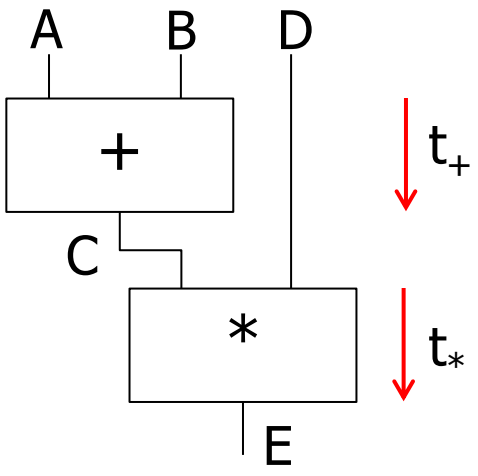
C = A + B
E = A * D



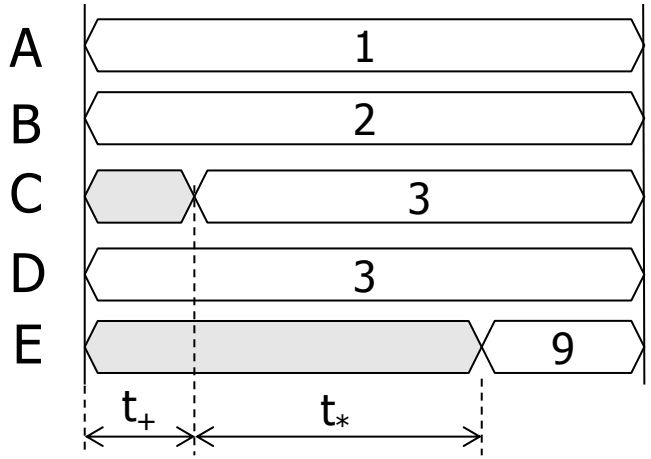
Datově nezávislé příkazy

Časové aspekty vykonávání sekvence příkazů

- Doba pro vykonání datové závislých příkazů je dána součtem dob jednotlivých příkazů (bloků)
- Příklad:



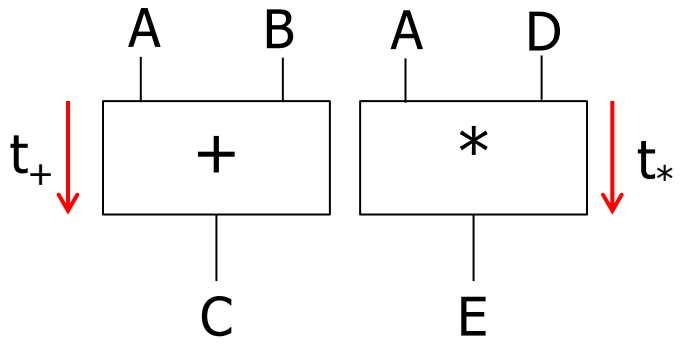
$$C = A + B$$
$$E = C * D$$

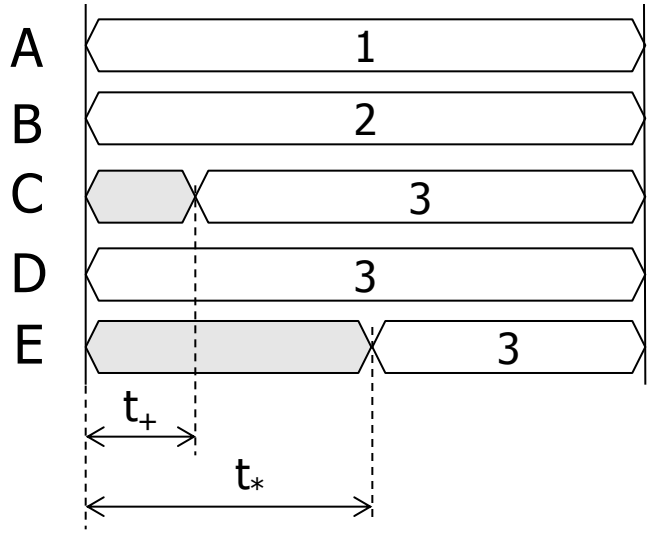


- Celková doba výpočtu $t=t_+ + t_*$
- V reálném obvodu je potřeba uvažovat i zpoždění vodičů
- Vstupní signály A, B, D musí být stabilní po celou dobu t
- Dokud se výpočet C a E neustálí, jsou na vodičích nedefinované hodnoty

Časové aspekty vykonávání sekvence příkazů

- Doba pro vykonání datové nezávislých příkazů je dána maximem dob jednotlivých příkazů (bloků)
- Příklad:





$$C = A + B$$
$$E = A * D$$

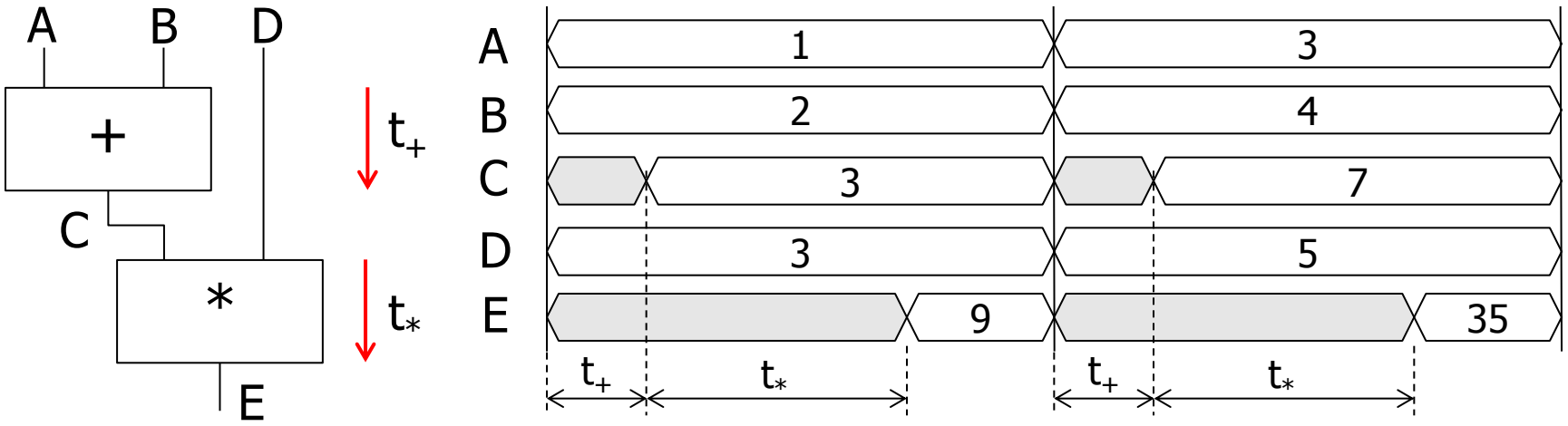
- Celková doba výpočtu $t=\max[t_+,t_*]$
- V reálném obvodu je potřeba uvažovat i zpoždění vodičů

Opakované vykonávání sekvence příkazů

- Často potřebujeme, aby byla sekvence příkazů resp. celý algoritmus vykonáván opakovaně, v některých případech i v nekonečné smyčce
- Příklady aplikací:
  - Filtrace signálů – jednotlivé vzorky vstupního signálu jsou zpracovávány v nekonečné smyčce
  - Zpracování obrazu – nad každým pixelem/snímkem obrazu je aplikována stejná operace
  - Zpracování síťového provozu – každý příchozí paket je zpracován požadovaným způsobem
- Problém:
  - Jak opakovaně spouštět číslicový obvod s novými parametry?

Opakované vykonávání sekvence příkazů

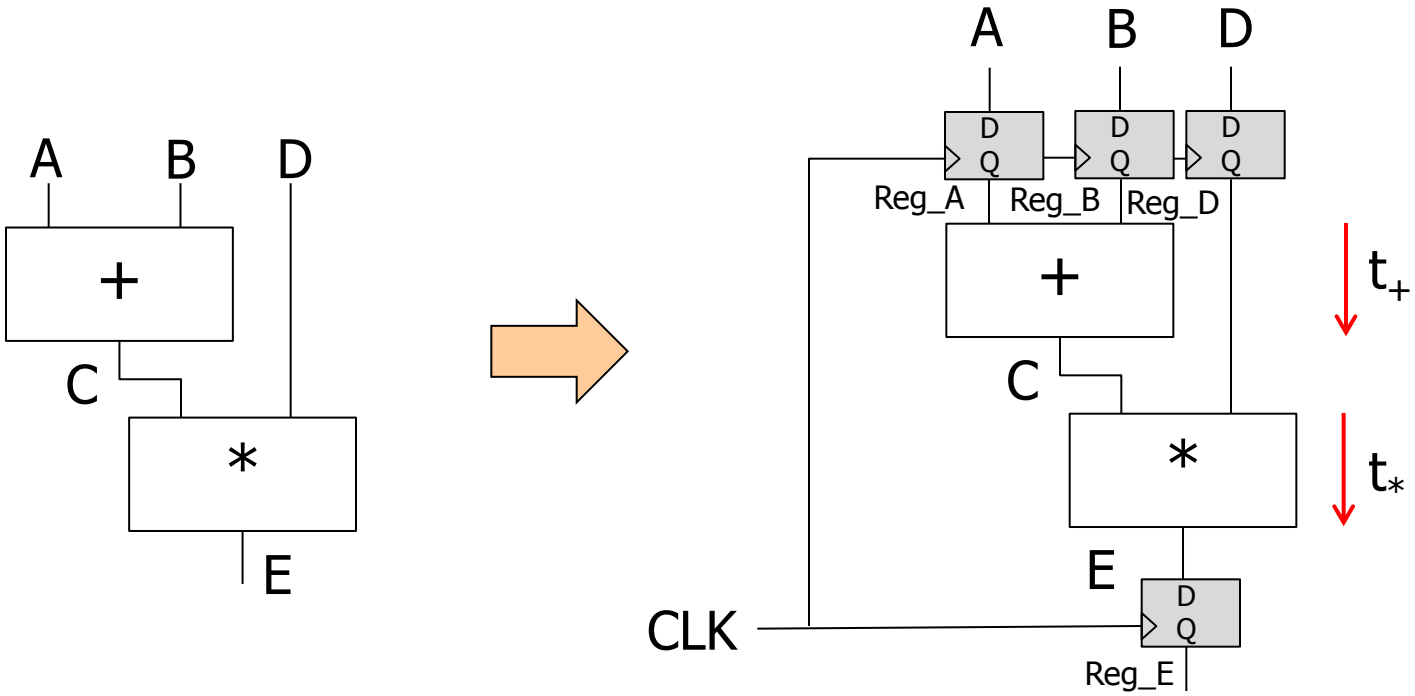
- U kombinačních obvodů je potřeba udržet vstupní parametry stabilní alespoň do té doby, dokud se výsledek neustálý
- Přiložením nových vstupních parametrů začíná opět proces výpočtu a výstupy jsou po určitou dobu nestabilní
- Příklad:



- Problémy:
  - Jak realizovat požadované zpoždění?
  - Jak udržet vstupní parametry stabilní po celou dobu výpočtu?
  - Jak ve správný okamžik uložit výsledek?

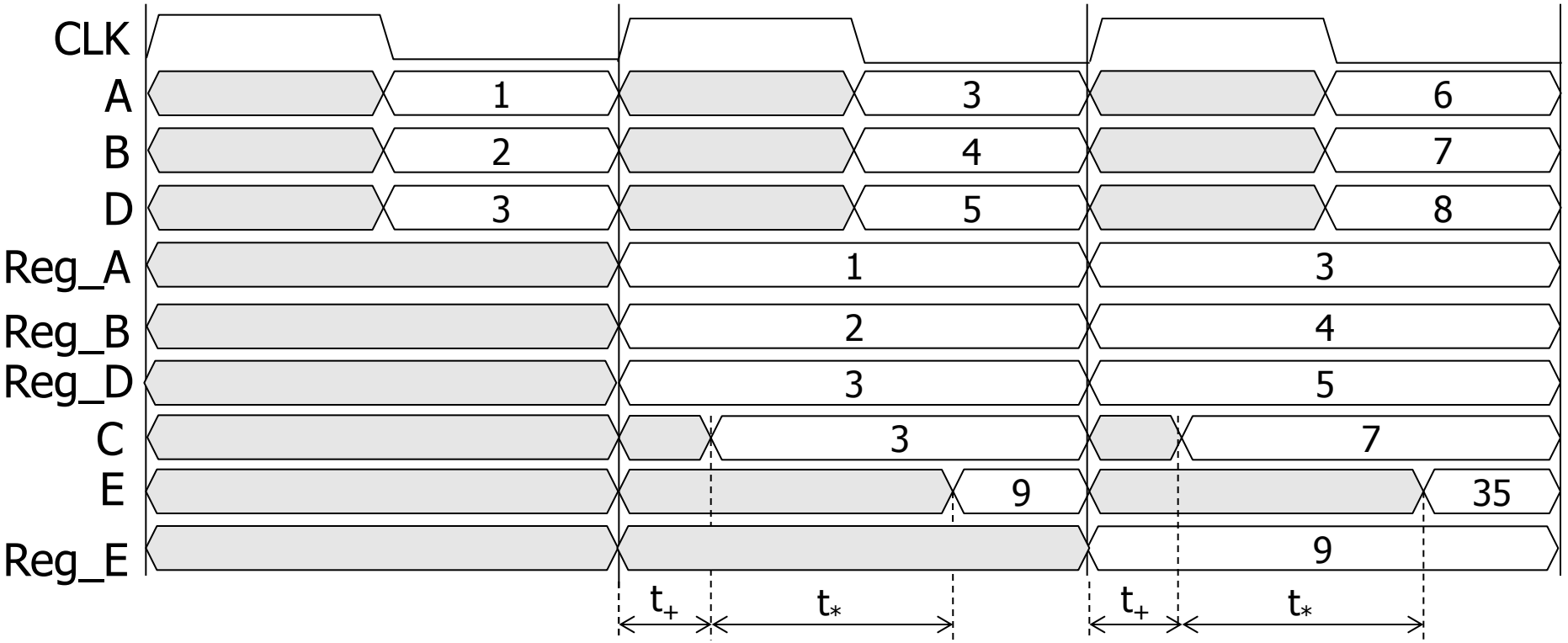
Opakované vykonávání sekvence příkazů

- Všechny uvedené problémy lze efektivně řešit převedením obvodu na synchronní obvod řízený hodinovým signálem
- Postup:
  - Stabilitu vstupních a výstupních parametrů zajistíme vložením registru
  - Periodu hodinového signálu CLK nastavíme na potřebnou délku zpoždění
- Příklad:



Opakované vykonávání sekvence příkazů

- Časový diagram

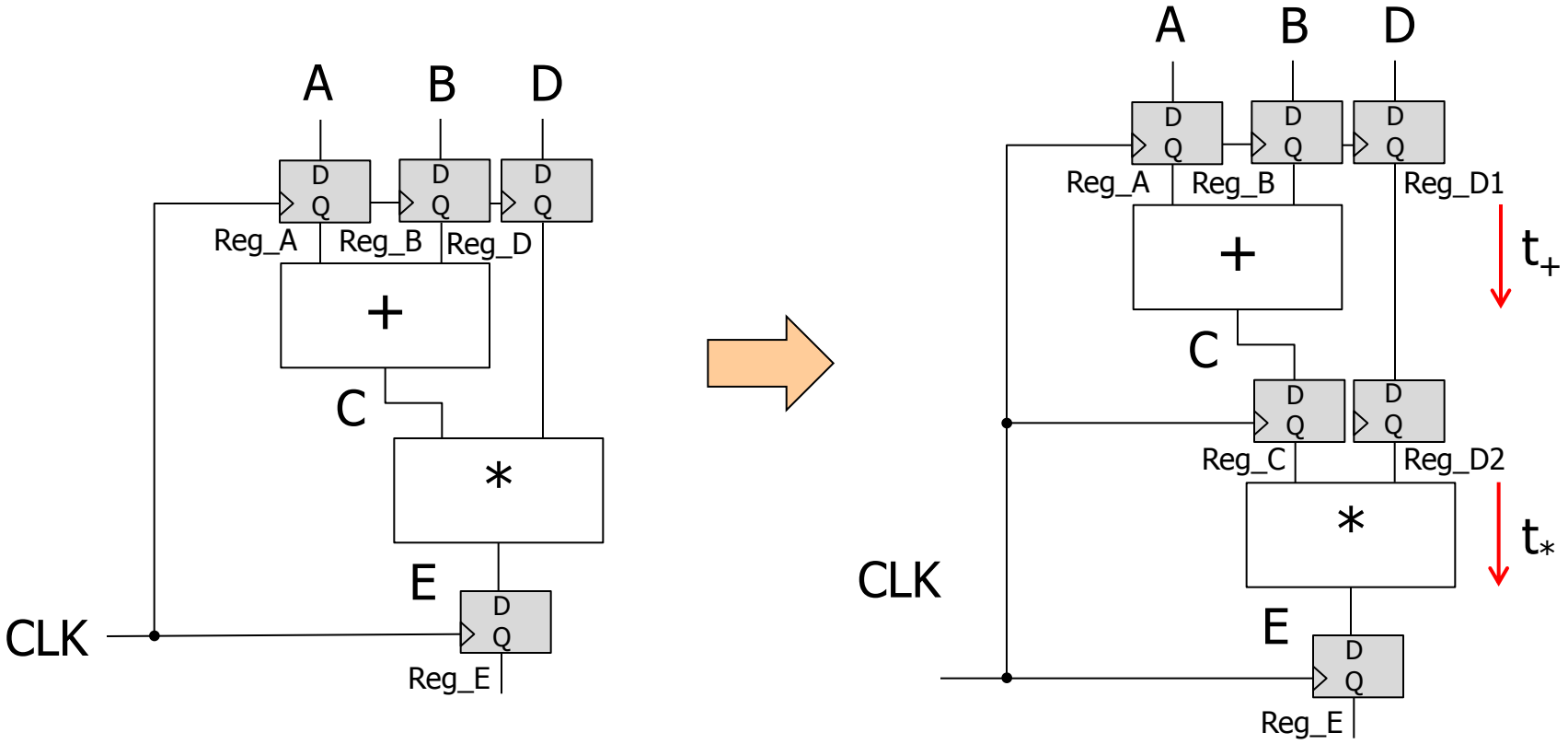


Opakované vykonávání sekvence příkazů

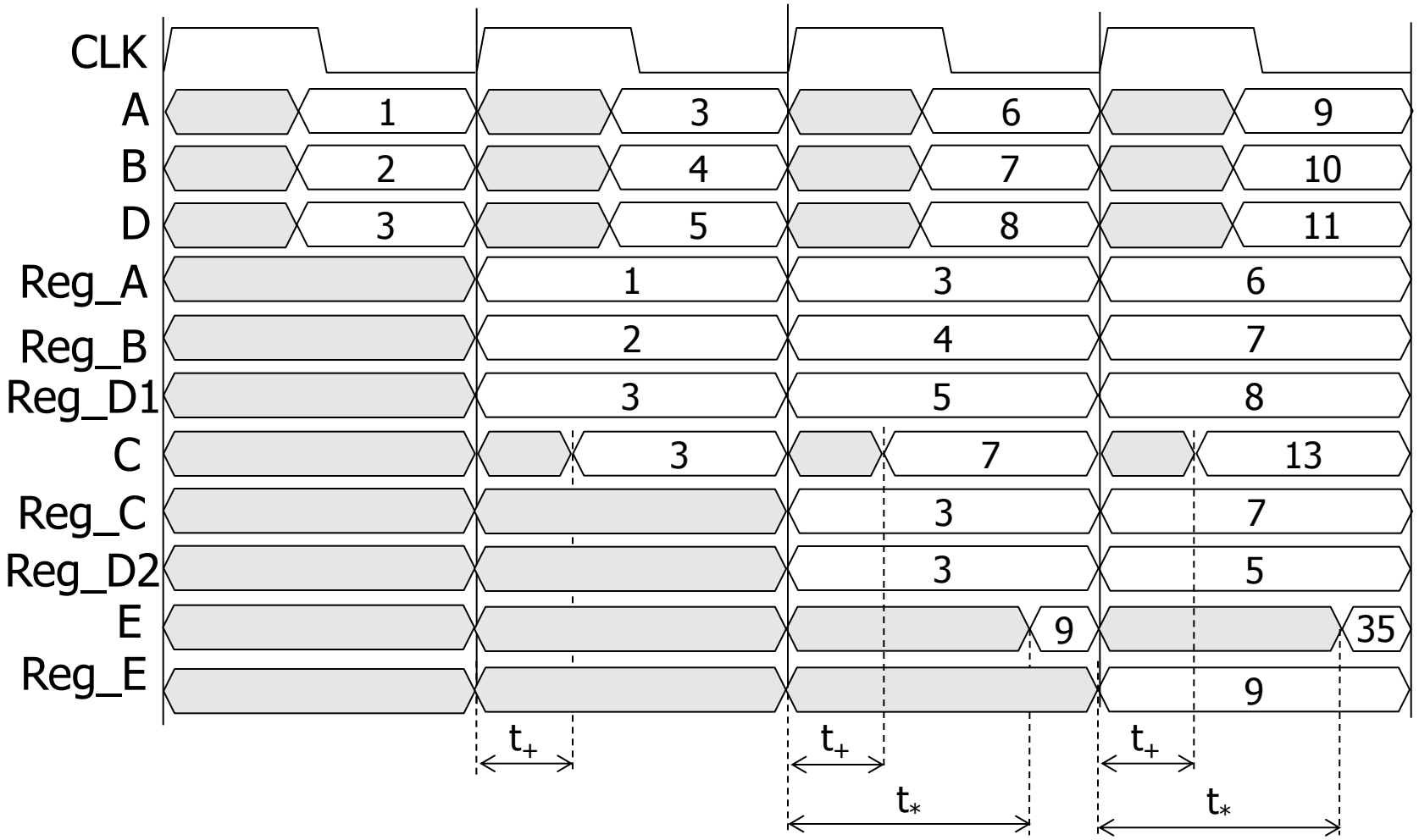
- Důsledky:
  - Výstupy registrů jsou stabilní po celou dobu periody
  - Přidáním registrů se prodlužuje latence obvodu
    - Vstupní parametry přiložené na vstup obvodu v rámci hodinového taktu i se zúčastní výpočtu až hodinovém taktu i+1
    - Výstupní hodnota vypočtené v hodinovém taktu i+1 je k dispozici na výstupu obvodu až v hodinovém taktu i+2
  - Doba periody musí pokrývat dobu výpočtu + setup time pro uložení výsledku do výstupního registru + zpoždění vodičů
- Problémy:
  - Pokud je sekvence příkazu příliš dlouhá nebo dokonce proměnlivá (např. podmíněné sekce s různou délkou), potom se nadměrně prodlužuje i délka periody hodinového signálu
  - Řešením je zřetězené zpracování

Zřetězené zpracování

- Vložením dalších registrových stupňů dovnitř sekvence příkazu je možné výpočet rozdělit na menších části, které však mohou současně (paralelně) zpracovávat oddělené části po sobě jdoucích iterací
- Příklad:



- Časový diagram

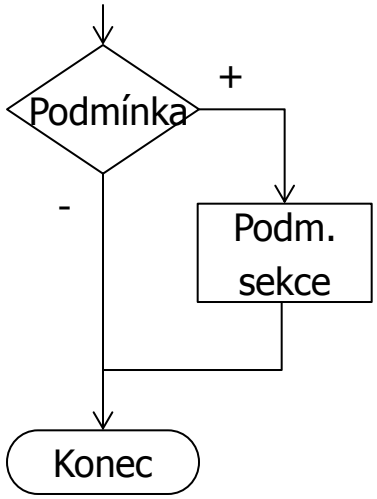


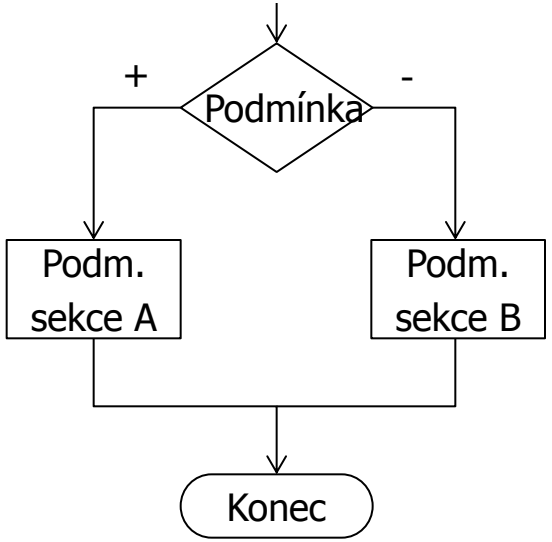
Zřetězené zpracování

- Poznámky:
  - Při vytváření stupně se musí registry vložit i doprostřed propojovacích vodičů (např. Reg_D2), jinak by výpočet nepracoval správně
- Důsledky:
  - Přidáním registrů dovnitř obvodu se dále prodlužuje jeho latence tj. výsledek výpočtu je dostupný o takt později
  - Doba periody se zkracuje na $t=\max[t_+,t_*]$ + setup time + zpoždění vodičů, neboť jednotlivé příkazy (operace) se vykonávají nezávisle v oddělených stupních
  - Technika zřetězení je nejefektivnější pokud je délka vypočtu v jednotlivých stupních vyvážená – vede na minimální délku periody tj. maximální rychlost výpočtu
- Problémy:
  - Technika zřetězení nelze aplikovat na výpočty se zpětnou vazbou

Selekce

- Skrze selekci lze ovládat průběh vykonávání jednotlivých příkazů algoritmu
- Existují různé varianty
  - Selekce s jednou podmíněnou sekcí
  - Výběr mezi dvojicí podmíněných sekcí
  - Výběr mezi několika podmíněnými sekcemi (case)
- Realizace na úrovni software
  - Nejprve je vyhodnocena podmínka a až potom vykonány příkazy některé z podmíněných sekcí
- Realizace na úrovni hardware
  - Všechny podmíněné sekce je možné vykonávat paralelně včetně vyhodnocení podmínky
  - Na závěr se skrze multiplexor vybere požadovaný výstup z příslušné podmíněné sekce
  - Alternativně lze výpočetní zdroje mezi podmíněnými sekcemi sdílet

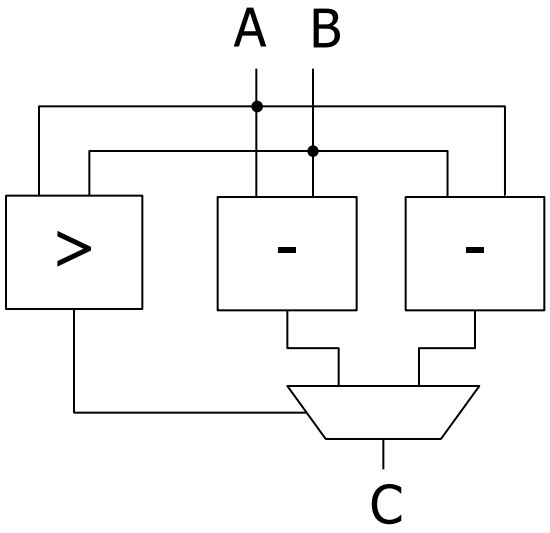




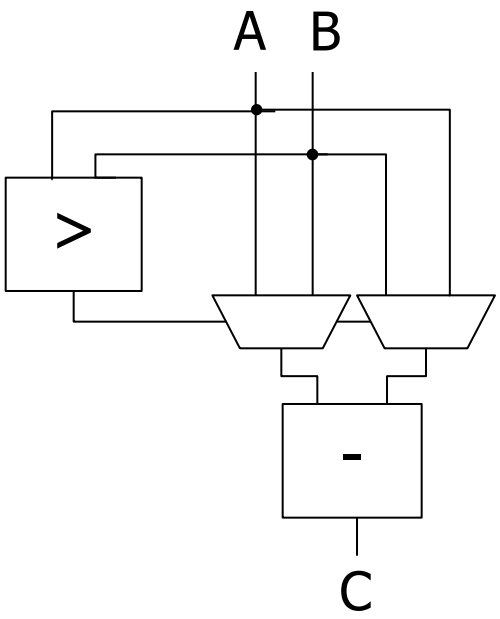
- Příklad:

```
if (A > B)
   C = A-B;
else
   C = B-A;
```

Varianta 1: Paralelní sekce



Varianta 2: Sdílení zdrojů



Varianta 1:

- Výpočet obou sekcí probíhá paralelně
- Na závěr je multiplexorem vybrán výsledek

Varianta 2:

- Nejprve jsou vybrány multiplexorem vstupní parametry
- Na závěr je proveden výpočet

Doba výpočtu:

- $t = \max[t_{>} + t_{-}] + t_{mx}$

Zdroje:

- 2x odčítačka,
- multiplexor,
- komparátor

Doba výpočtu:

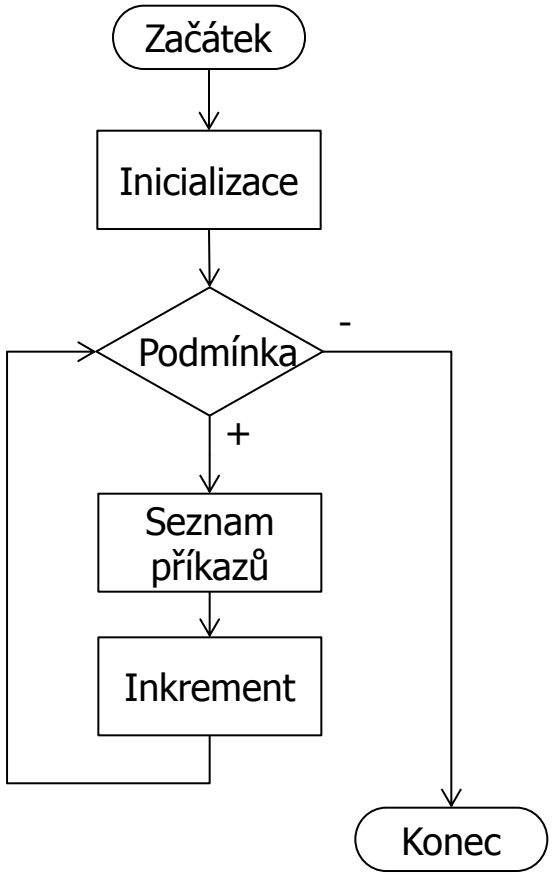
- $t = t_{>} + t_{mx} + t_{-}$

Zdroje:

- odčítačka,
- 2x multiplexor,
- komparátor

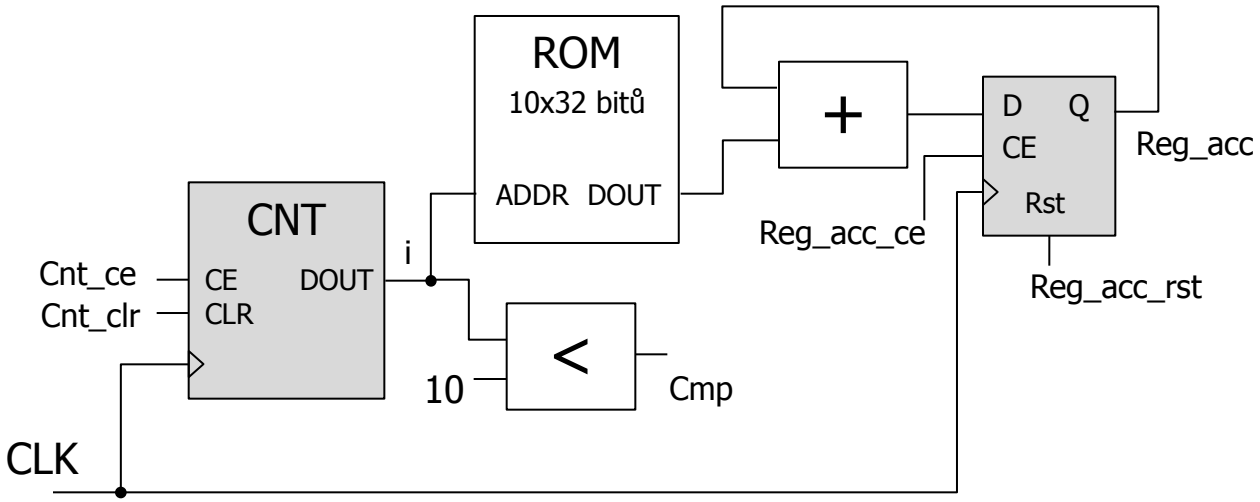
Iterace

- Umožňuje opakování určité části algoritmu
- Rozlišujeme několik typů
  - Cyklus for s pevným počtem iterací
  - Cyklus while a do while s proměnným počtem iterací
- Všechny typy cyklů zahrnují:
  1. Inicializační část
  2. Podmínku ukončení cyklu
  3. Inkrement
  4. Seznam příkazů
- Příklad:
  - for(inicializace; podmínka; inkrement) { seznam příkazů }
- Všechny tyto 4 části je potřeba realizovat i na úrovni hardware



- Příklad: Součet prvků pole
  - Vstupní pole a je uloženo v paměti typu ROM
  - V každém tiku hodinového signálu:
    - Proměnná cyklu i je inkrementována skrze čítač (Cnt_ce = 1) a současně slouží jako adresa do paměti ROM
    - Přečtené položky paměti jsou postupně akumulována v registru Reg_acc
  - Celý cyklus končí v okamžiku na základě podmínky cyklu i < 10 (Cmp = 0)
  - Před začátkem výpočtu je potřeba:
    - Vynulovat proměnnou cyklu i (Cnt_clr = 1)
    - Vynulovat obsah registru Reg_acc (Reg_acc_rst = 1)

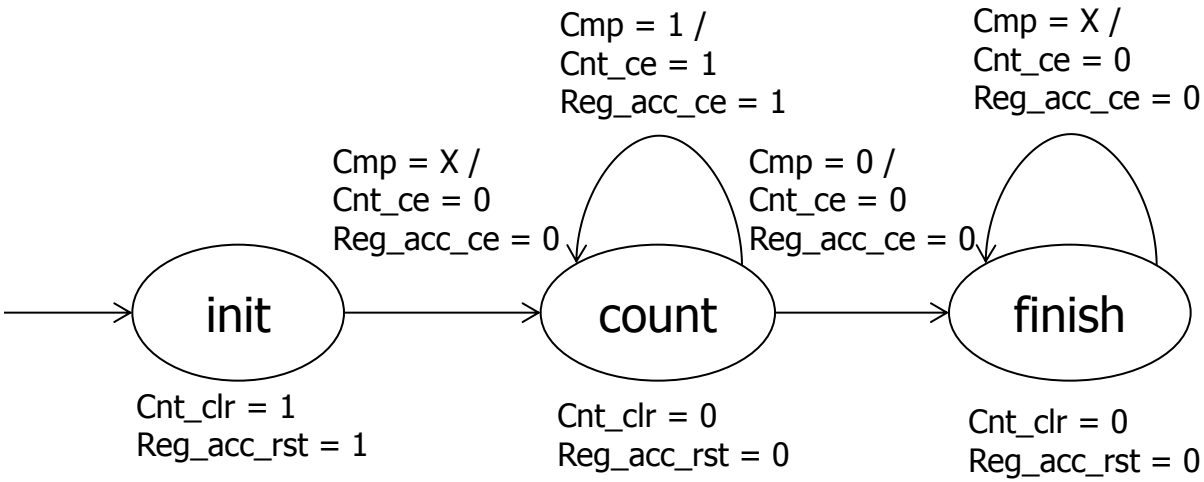
```
int a[10];
int acc = 0;

for(i=0; i<10; i++)
   acc = acc + a[i];
```



Iterace

- Činnost obvodu je řízena na základě signálů:
  - Cnt_ce, Cnt_clr, Cmp, Reg_acc_rst, Reg_acc_ce
- Postup:
  1. Inicializace: vynulování čítače (Cnt_clr=1) a registru Reg_acc (Reg_acc_rst=1)
  2. Spuštění cyklu: aktivace čítače (Cnt_ce=1) a povolení zápisu do registru Reg_acc (Reg_acc_ce=1)
  3. Ověření podmínky cyklu
     - Pokud je podmínka splněna (Cmp=1), potom návrat do bodu 2
     - Pokud není splněna (Cmp=0), potom zastav čítač (Cnt_ce=0), zápis do registru (Reg_acc_ce=0) a pokračuj na bod 4
  4. Konec
- Jaký nastavit tyto signály, aby obvod pracoval správně?
  - $\Rightarrow$ Je potřeba navrhnout automat

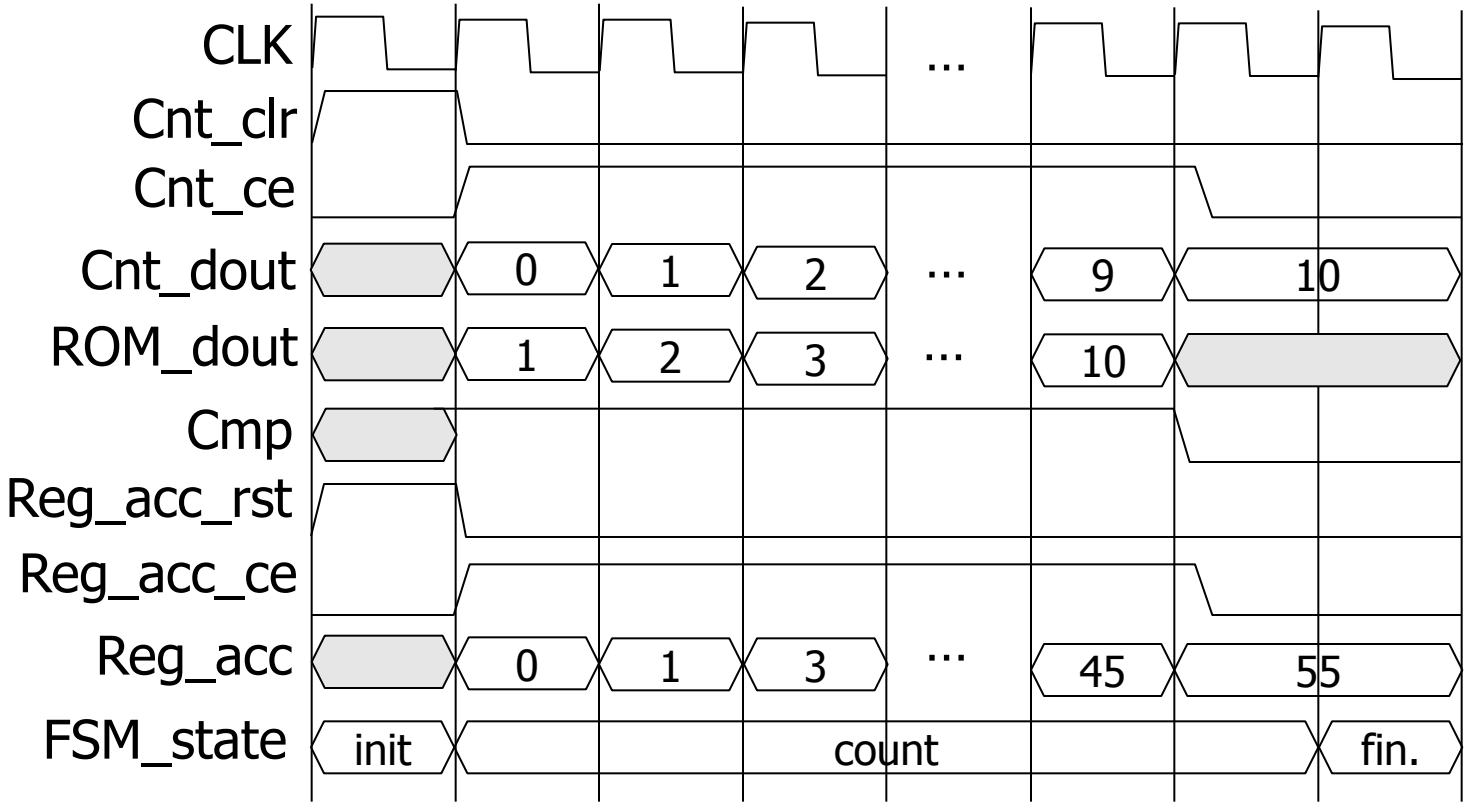
- Návrh automatu:



Mealyho výstupy:
- Cnt_ce, Reg_acc_ce

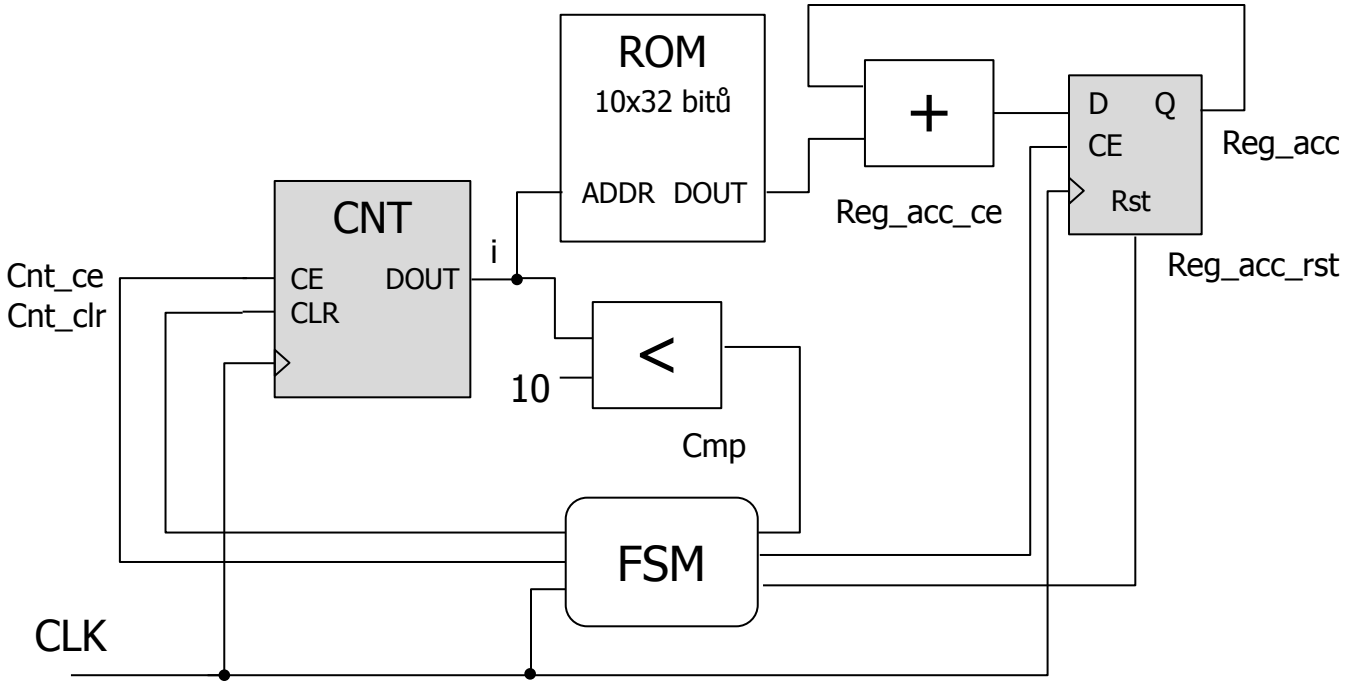
Moorovy výstupy:
- Cnt_clr, Reg_acc_rst

- Časový diagram



Délka periody: t_+ + setup time + zpoždění vodičů
Celková doba výpočtu: 13 taktů

- Výsledné schéma složeno z:
  - Datové cesty: Paměť, čítač, sčítačka, registr
  - Řídicí cesty: FSM



Iterace – rozbalení smyčky

- Smyčky se fixním počtem iterací lze rozbalovat tj. několikanásobně rozepsat tělo smyčky
- Rozbalením smyčky se zápis algoritmu stává delším, avšak snižuje se režie spojená s vyhodnocením podmínky cyklu
- Rozlišujeme:
  - Částečné rozbalení smyčky
  - Úplné rozbalení smyčky
- Příklad:

```
int a[10];
int acc = 0;

for(i=0; i<10; i++)
   acc = acc + a[i];
```

Nerozbalená smyčka

```
int a[10];
int acc = 0;

for(i=0; i<10; i+=2) {
   acc = acc + a[i];
   acc = acc + a[i+1];
}
```

Částečně rozbalená smyčka

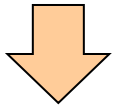
```
int a[10];
int acc = 0;

acc = acc + a[0];
acc = acc + a[1];
acc = acc + a[2];
...
acc = acc + a[9];
```

Úplně rozbalená smyčka

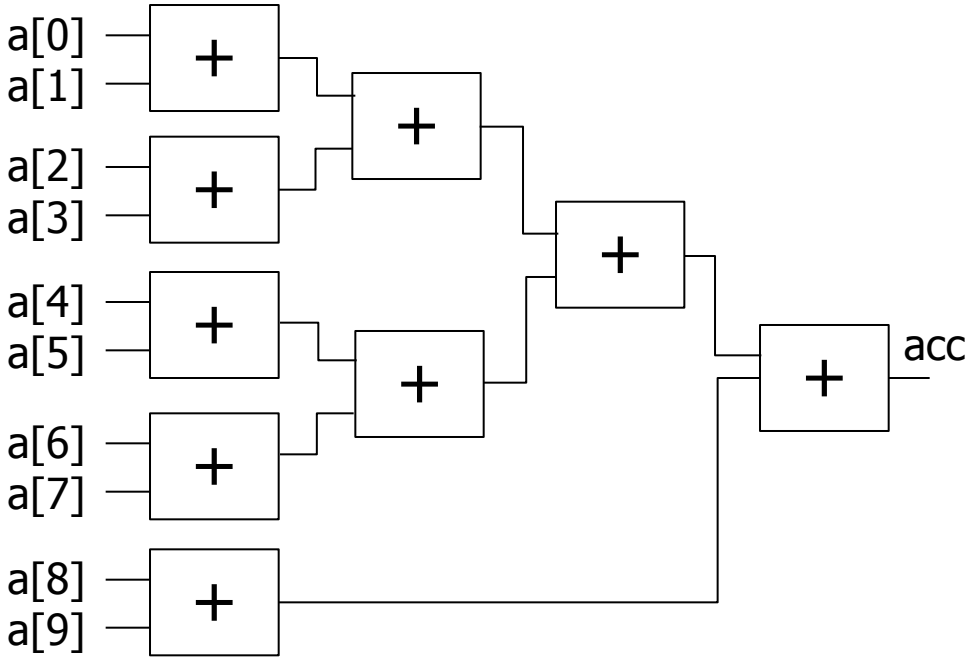
Iterace – rozbalení smyčky

- Pokud jsou jednotlivé příkazy rozbalené smyčky datově nezávislé (popř. je lze upravit na datově nezávislé), potom mohou být zpracovány v hardware paralelně
- Příklad:

```
int a[10];
int acc = 0;

acc = acc + a[0];
acc = acc + a[1];
acc = acc + a[2];
…
acc = acc + a[9];
```

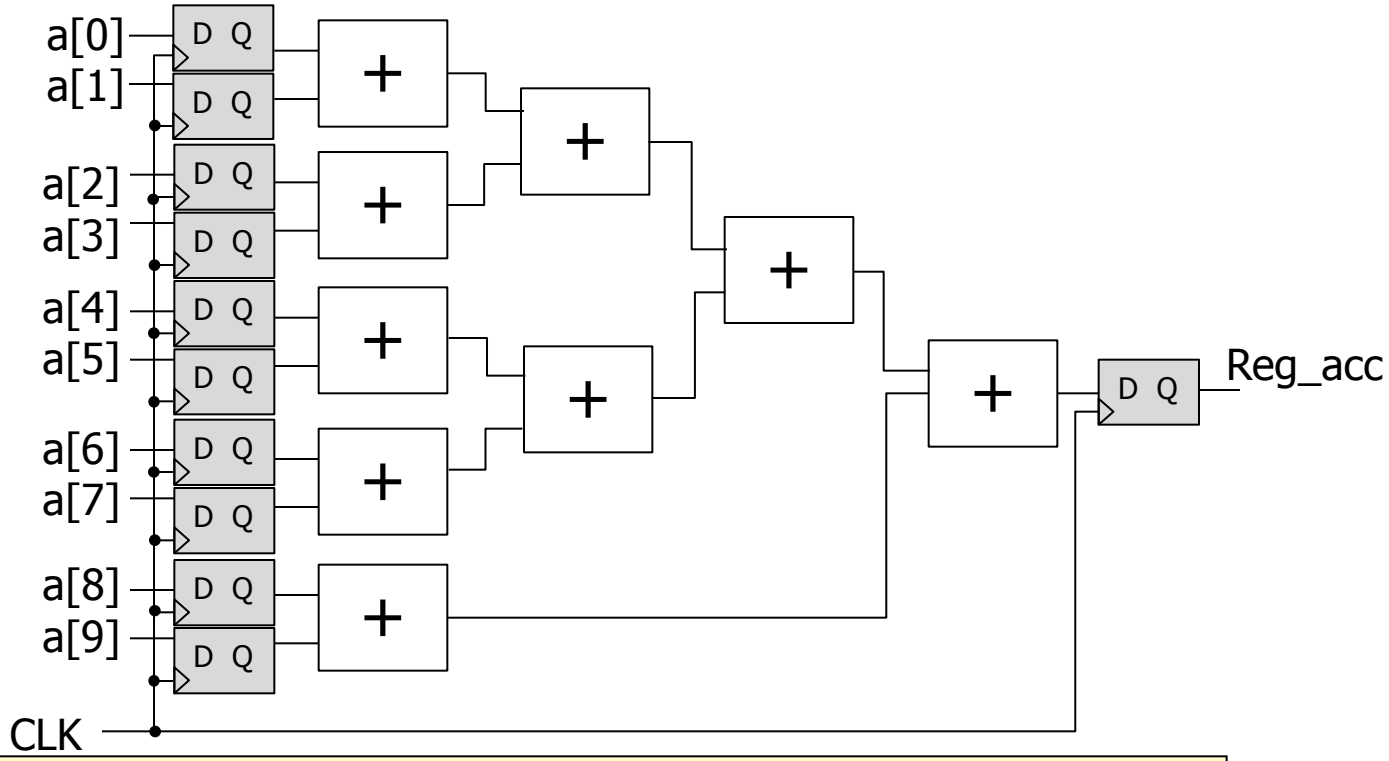


```
int a[10];

acc = a[0] + a[1] + … + a[9]
```



Iterace – rozbalení smyčky

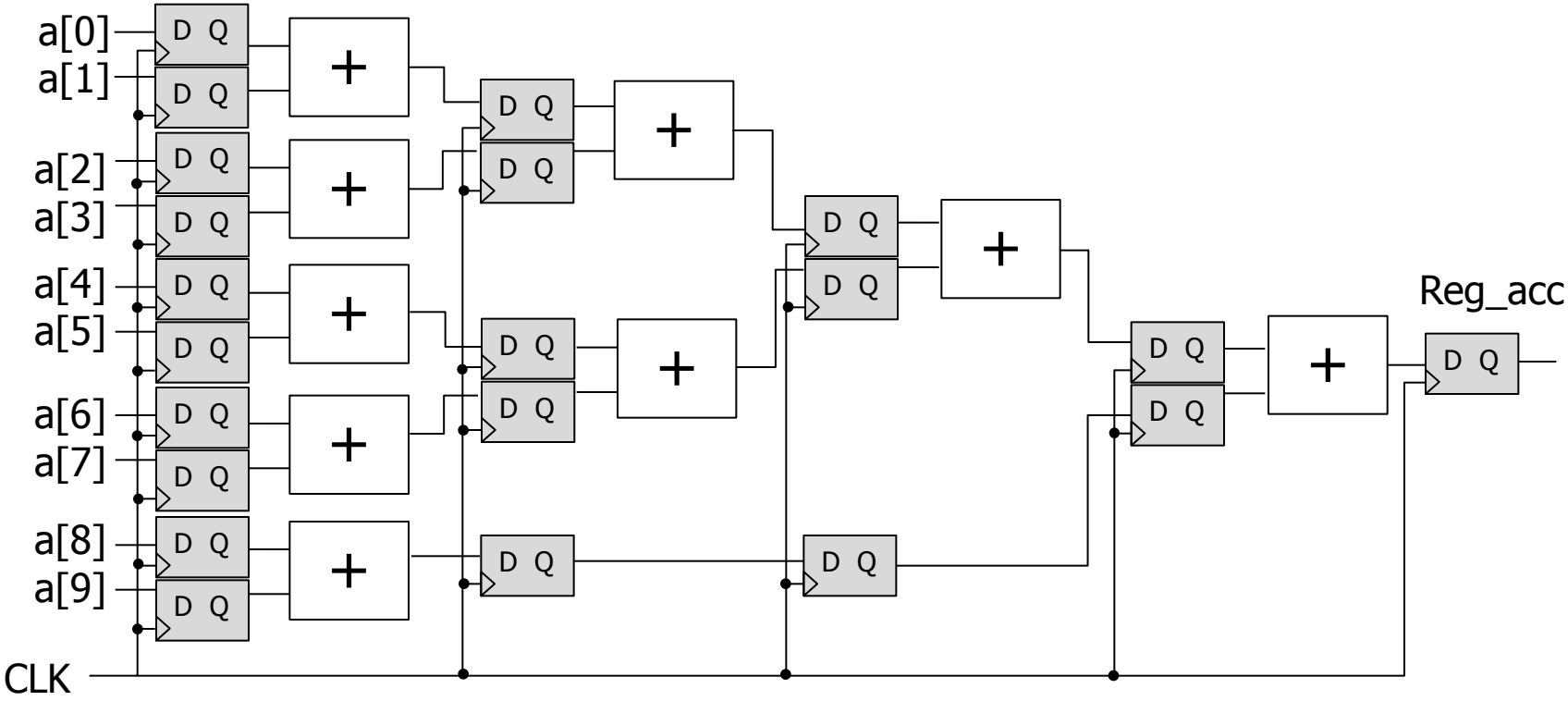
- Důsledky:
  - Při rozbalení smyčky využívající (na vstupu nebo výstupu) pole hodnot je obvykle nutné přesunout tato pole z paměťových bloků do registrů – paměťové bloky nejsou schopny zajistit paralelní přístup



Délka periody: (4 * t_+) + setup time + zpoždění vodičů
Celková doba výpočtu: 1 takt

Iterace – zřetězení rozbalené smyčky

- Rozbalenou smyčku lze dále zřetězit a zvýšit tak její výkonnost
- Příklad:



Délka periody: t_+ + setup time + zpoždění vodičů
První výsledek dostupný za: 4 takty
Každý další výsledek dostupný za: 1 takt

Osnova

- Úvod
- Transformace základní konstrukcí
  - Sekvence
  - Selekce
  - Iterace
- Shrnutí

Shrnutí

- Libovolný algoritmus vytvořený na základě sekvence, selekce a iterace lze rovněž realizovat na úrovni hardware
- Výsledný obvod zahrnuje:
  - Datovou cestu: výpočetní elementy a jejich propojení
  - Řídicí cestu: řadič ve formě FSM, který ovládá průběh výpočtu
- Oproti implementaci algoritmu v software
  - Nabízí HW realizace vyšší potenciál k paralelizaci výpočtu
  - Při návrhu obvodu lze vyžít různých technik např. zřetězení, rozbalení smyček nebo jejich kombinaci – vede na velké množství realizací s různou dobou výpočtu a množstvím požadovaných výpočetních zdrojů
  - Návrhář aplikace však musí mnohem více uvažovat časové aspekty obvodu, což výrazně zvyšuje složitost návrhu

Konec

Děkuji za pozornost