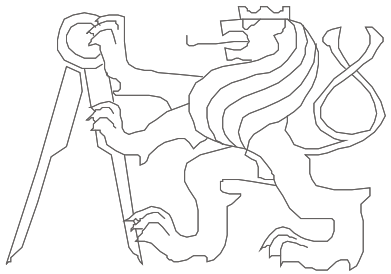
Pokročilé architektury počítačů

04

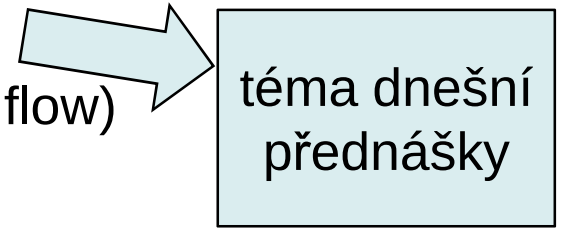
Superskalární techniky –
Předvýběr instrukcí (Predikce větvení aj.)



České vysoké učení technické, Fakulta elektrotechnická

Superskalární techniky

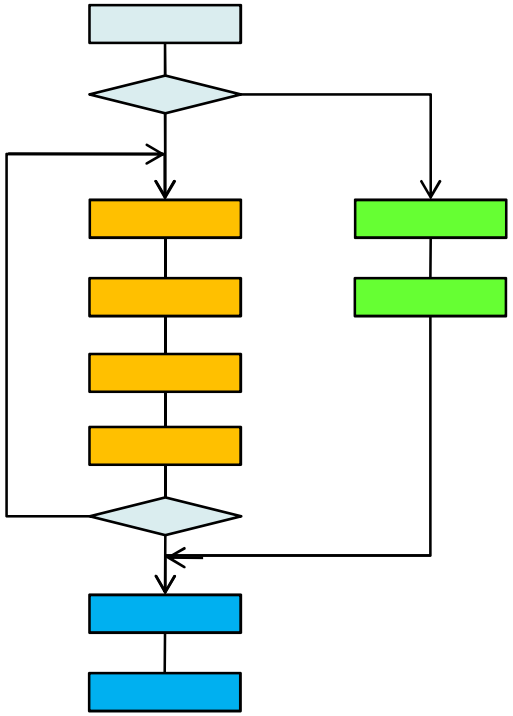
- Uvědomme si, že cílem je maximální propustnost zpracování instrukcí…
- Na **zpracování** instrukcí můžeme nahlížet jako na tok instrukcí a tok dat, přesněji:
  - **tok samotných instrukcí (instruction flow)**
  - tok dat mezi registry procesoru (register data flow)
  - a tok dat z/do paměti (memory data flow)



- To zhruba odpovídá:
  - skokové instrukce
  - aritmeticko-logické / výpočetní instrukce
  - load/strore instrukce
- Pokud tedy chceme maximalizovat celkový tok, musíme minimalizovat čas (penalizaci) těchto tří typů instrukcí

Tohle všichni známe…

Náš vlastní program vyjádřen jako Control Flow Graph (CFG):

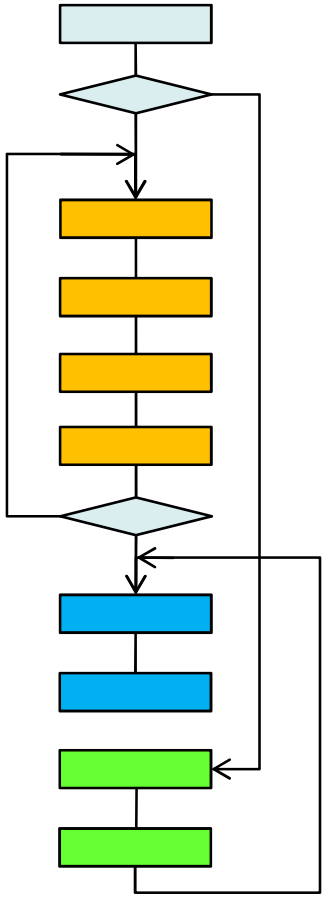


CFG musí být namapován do sekvenční paměti:

Adresa

Adr + offset

…

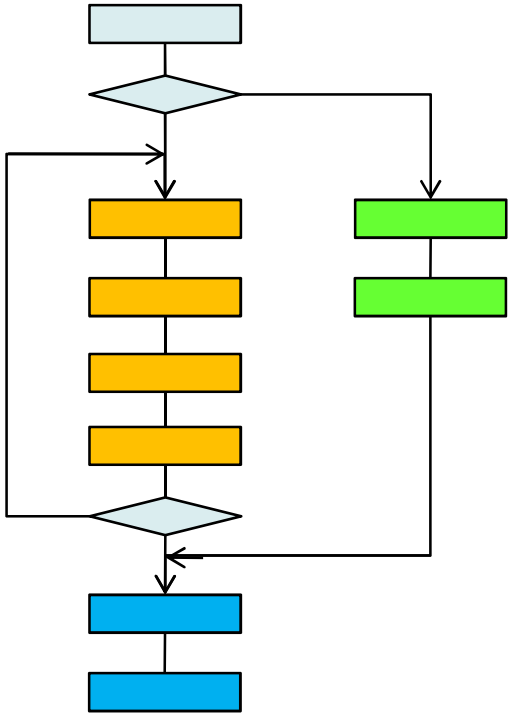


A co objektové programování ???

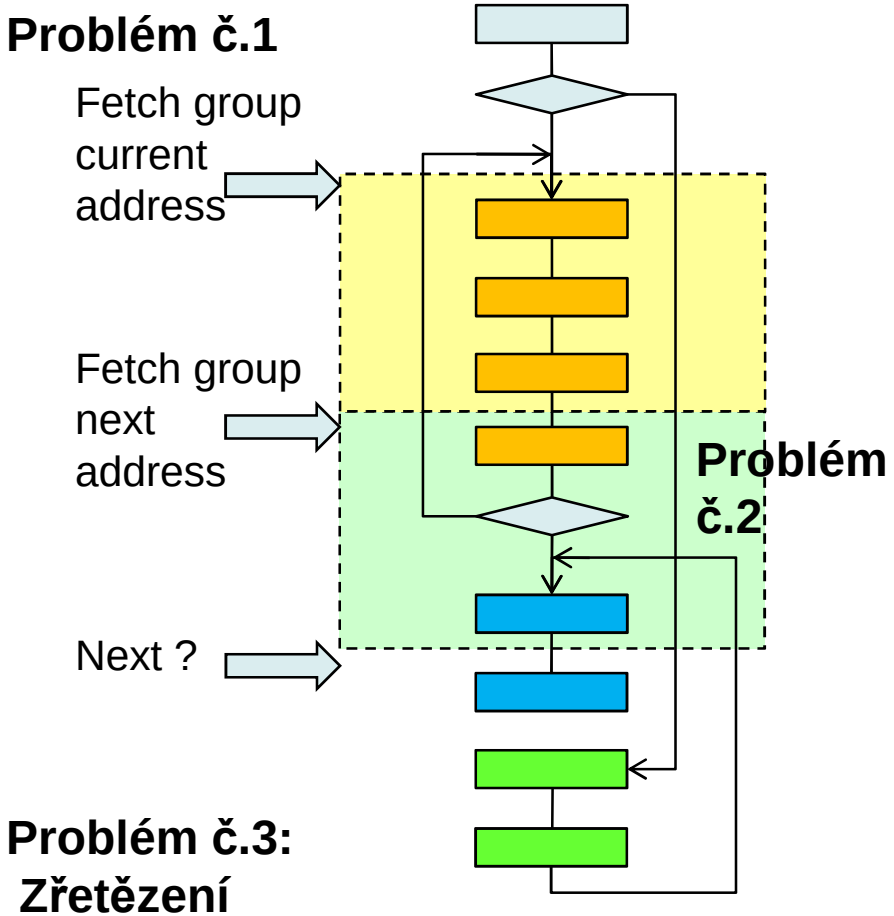
Používáte virtuální metody ???

Tohle všichni známe…

Náš vlastní program vyjádřen jako Control Flow Graph (CFG):



CFG musí být namapován do sekvenční paměti:

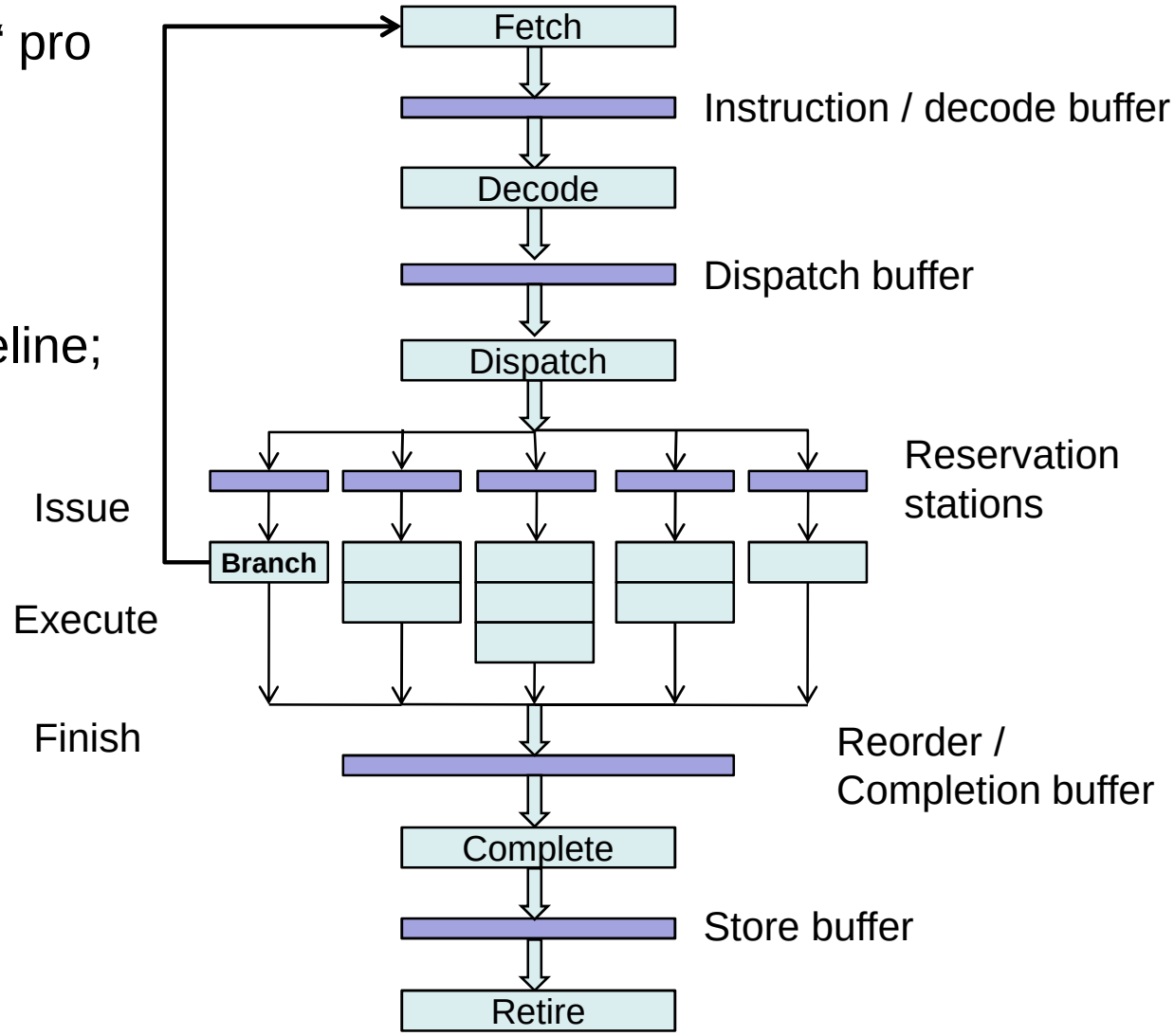


A co objektové programování ???

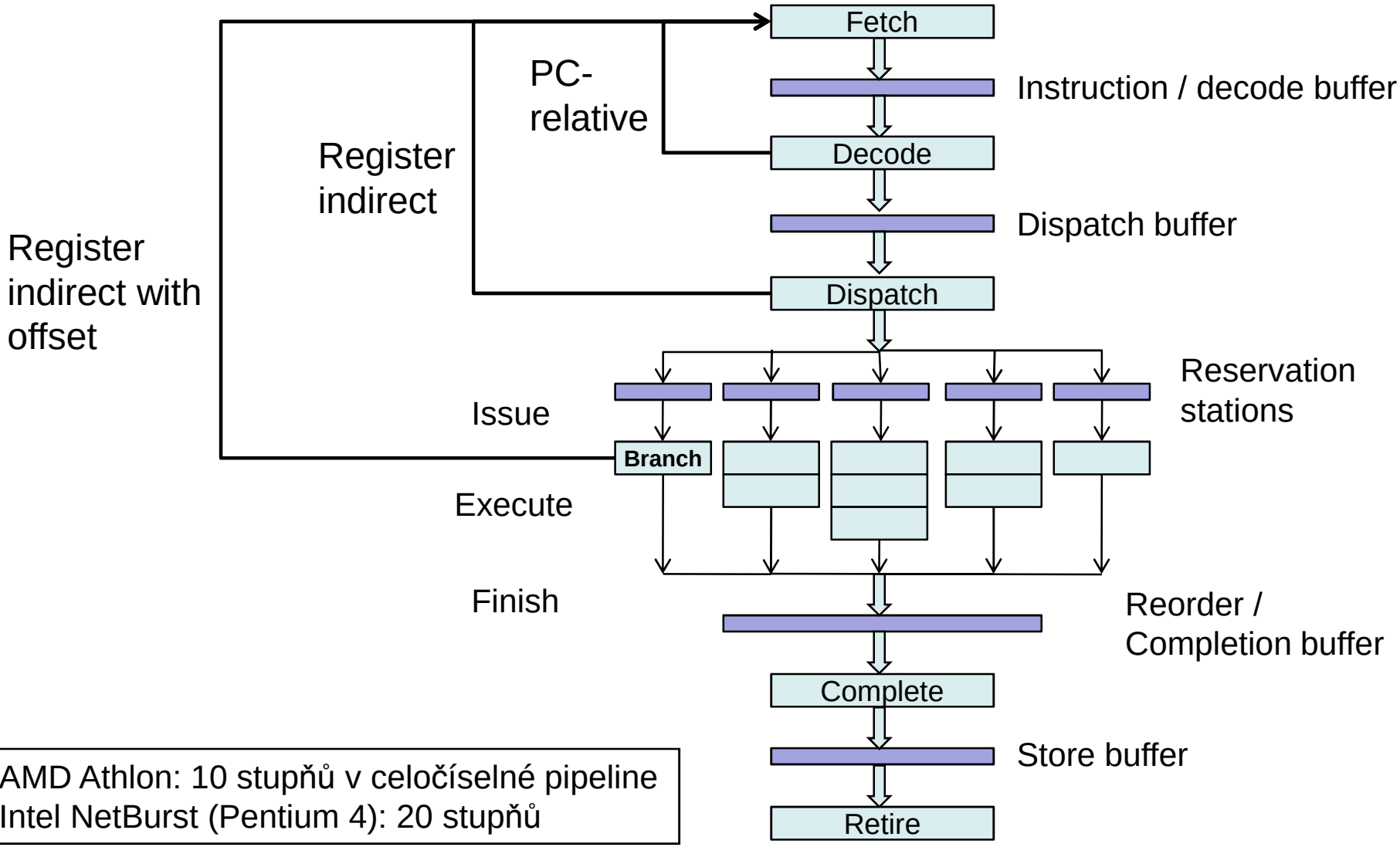
Používáte virtuální metody ???

Predikce větvení - motivace

- Penalizace „3 cykly“ pro vybrání následující instrukce;
- Počet prázdných instrukčních slotů násoben šířkou pipeline;
- Amdahlův zákon..



Predikce větvení - motivace



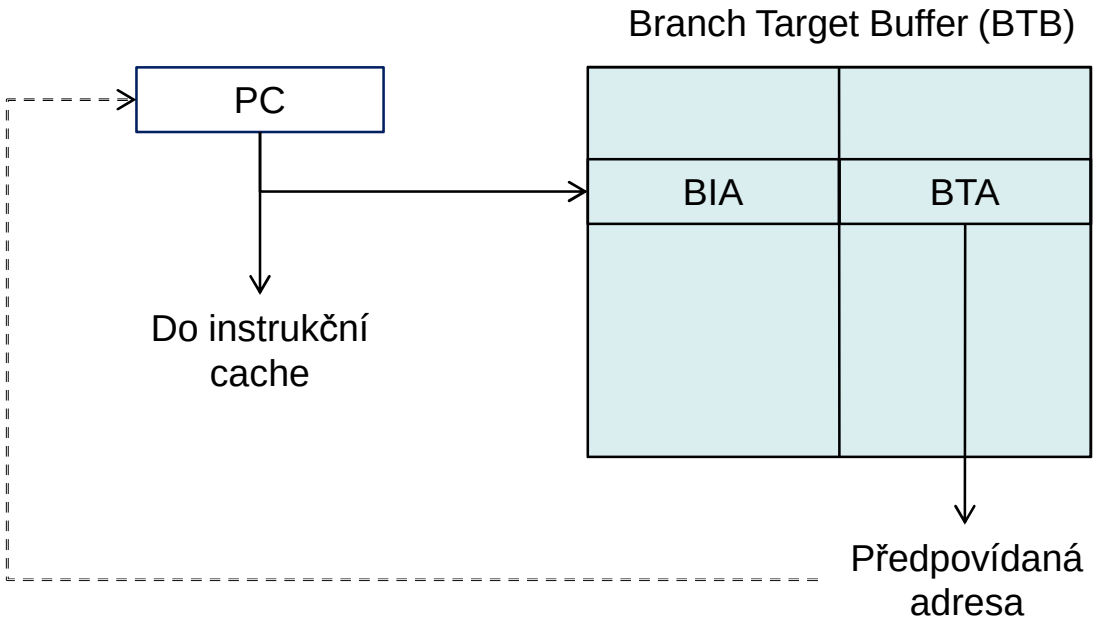
AMD Athlon: 10 stupňů v celočíselné pipeline
Intel NetBurst (Pentium 4): 20 stupňů

Predikce větvení

- Dvě fundamentální složky:
  - branch target speculation (kde),
  - branch condition speculation (zda vůbec).

- Predikce cíle větvení:
  - BTB (Branch Target Buffer) – asociativní cache obsahující dvě položky: BIA (Branch Instruction Address) a BTA (Branch Target Adress) – přistupuje se do ní současně při výběru instrukce hodnotou PC
  - pokud se BIA shoduje s PC, je vybrána BTA a v případě, že se skok predikuje mění PC

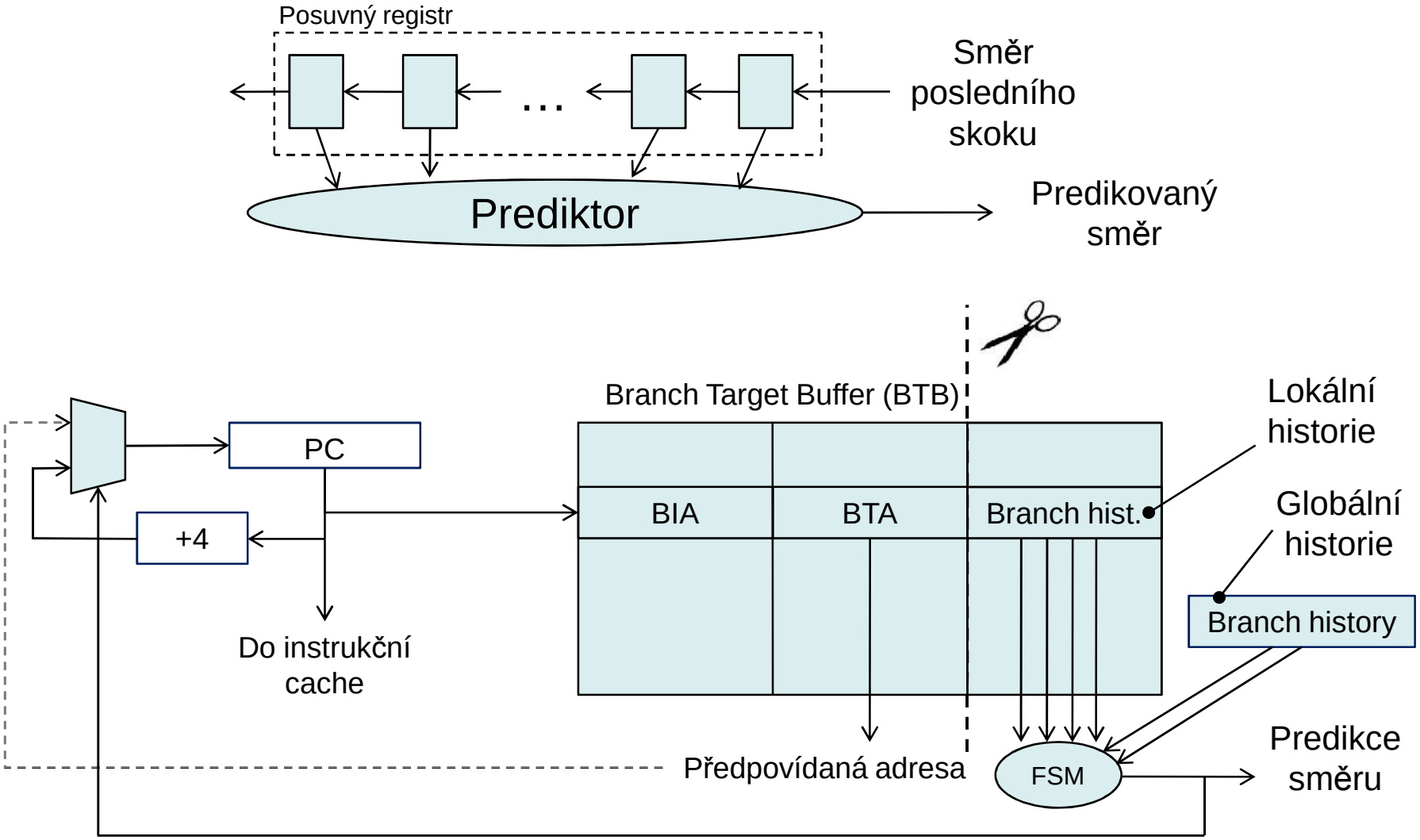
Branch Target Speculation



Predikce větvení

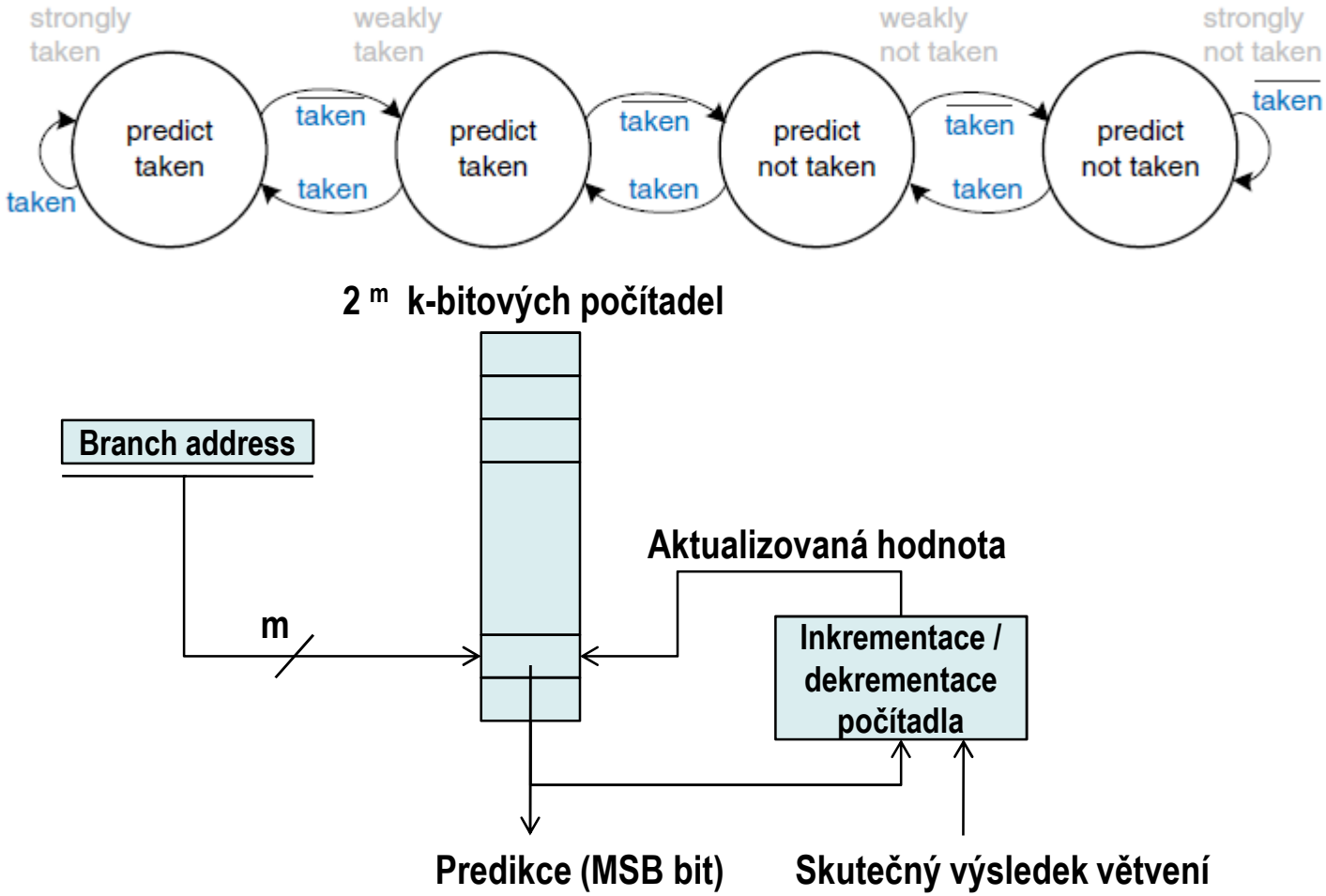
- Predikce splnění podmínky větvení:
  - statická predikce (70%-80%)
    - BTFNT (Backwards Taken / Forwards Not-Taken) – cyklus for, while, do-while,.. - relativně k PC, branch delay slot…
    - Heuristiky analyzující program (NULL pointer, porovnávání na shodu čísla, vnořené funkce…) – výsledky se předávají jako tipy (branch hints) kódované v skokových instrukcích (podpora ISA)
    - Profilace – vykonávání programu s různými vstupy - statistiky
  - dynamická predikce (80%-97%)
  - hybridní (predikuje se dynamicky, pokud však není informace o predikci dostupná, použije se statická predikce..)

Predikce větvení



Predikce splnění podmínky větvení

- Smithův algoritmus (saturující počítadlo)



Predikce splnění podmínky větvení

- Smithův algoritmus

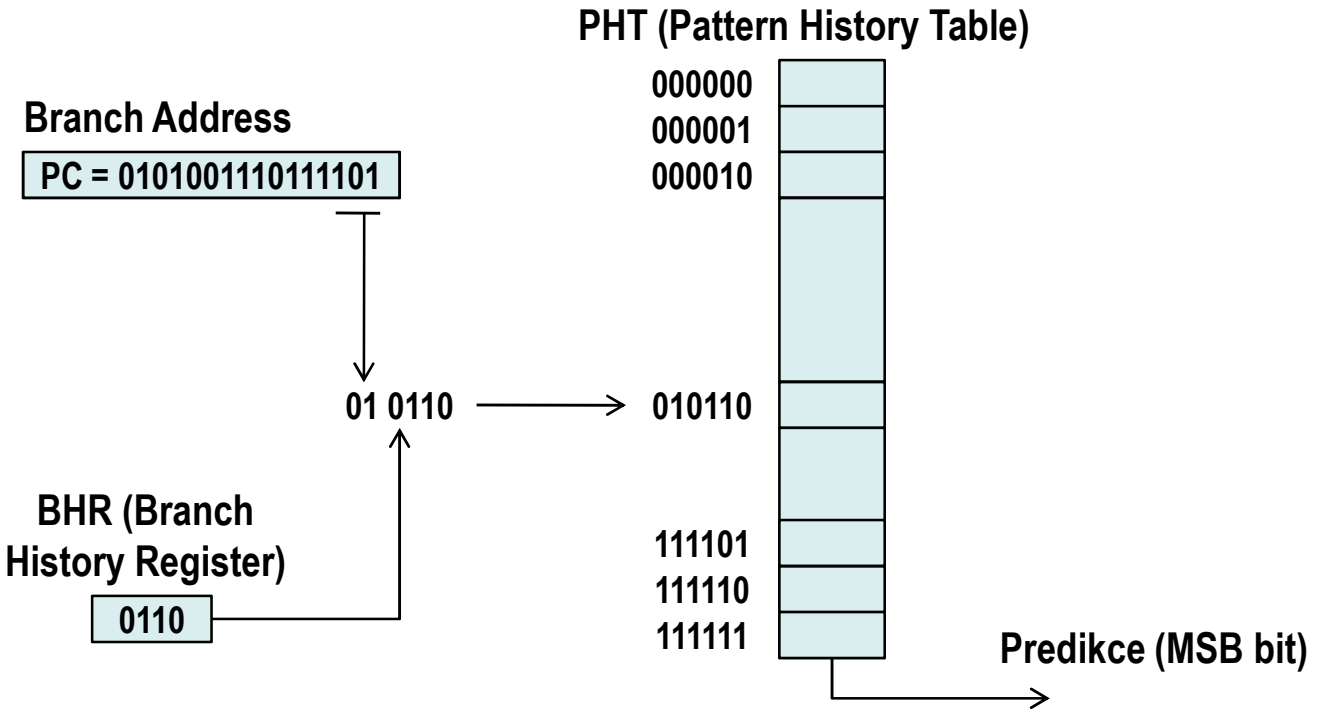
Zamysleme se…

Pokud má PC 32 bitů, pak bychom potřebovali (1/8)*k*2^32 B paměti pro uchování stavů všech počítadel (pro k=2 to představuje 1GB)

- **Conflict aliasing**:
  - neutrální interference
  - negativní interference
- ještě existuje Compulsory aliasing – uvidíme později (důsledek indexování přes adress-history)

Predikce splnění podmínky větvení

- Dvou-úrovňový prediktor s **globální** historií větvení a 4-bitovým registrem historie



Kolik bitů použijeme pro BHR a kolik pro BA?

Predikce splnění podmínky větvení

- Dvou-úrovňový prediktor s **globální** historií větvení a 4-bitovým registrem historie

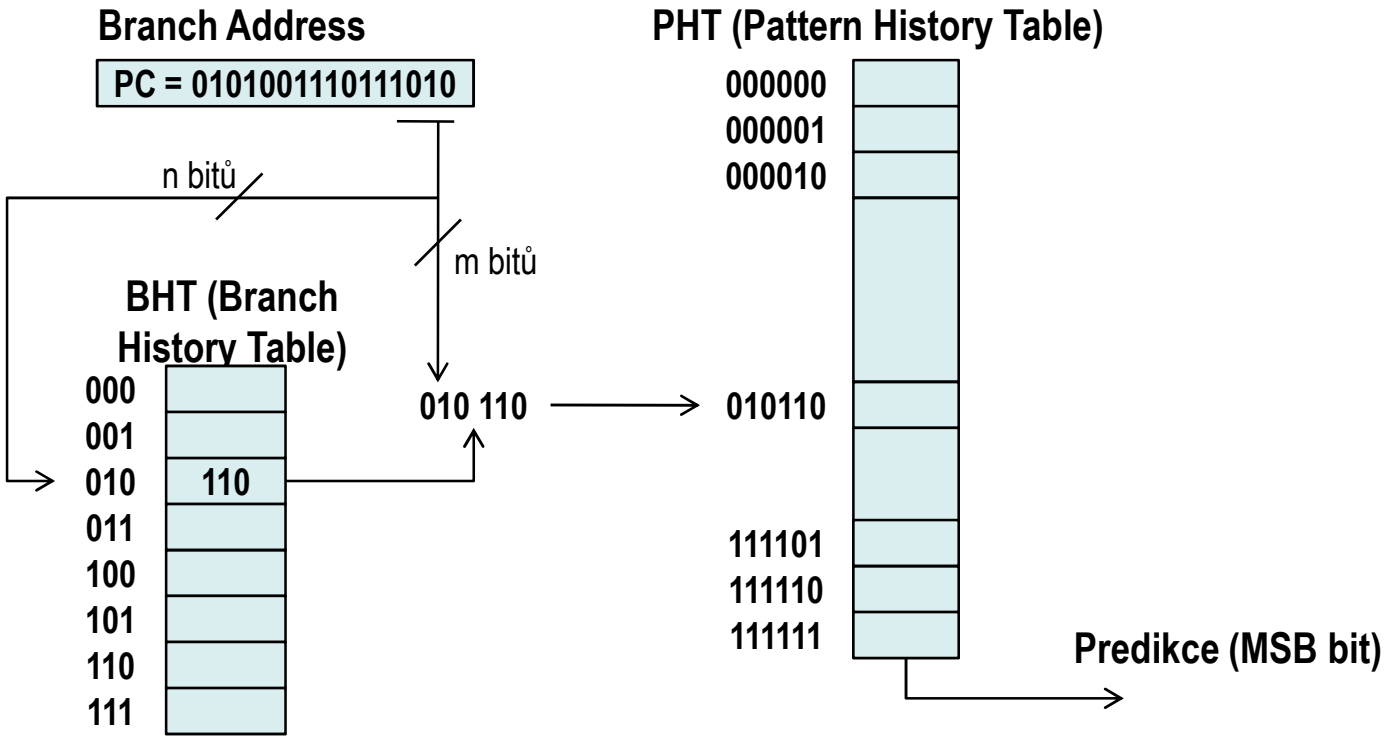
- Proč používat i globální historii pro indexaci do PHT?

```
a=0;
if(podmínka č.1)   a=3;
if(podmínka č.2)   b=10;
if(a <= 0)  F();
```

- Chování skokové instrukce může souviset (korelovat) s vykonáním jiných skokových instrukcí v minulosti…
- Na našem příkladu bude vykonání funkce F() fakticky podmíněno splněním podmínky č.1. Naopak podmínka č.2 je irelevantní. Prediktor se musí naučit tyto skokové instrukce (podmínky) odlišit.

Predikce splnění podmínky větvení

- Dvou-úrovňový prediktor s **lokální** historií větvení a 3-bitovou tabulkou historie



Intel P6 používá 4 bity pro BHR

Predikce splnění podmínky větvení

- Dvou-úrovňový prediktor s **lokální** historií větvení a 3-bitovou tabulkou historie

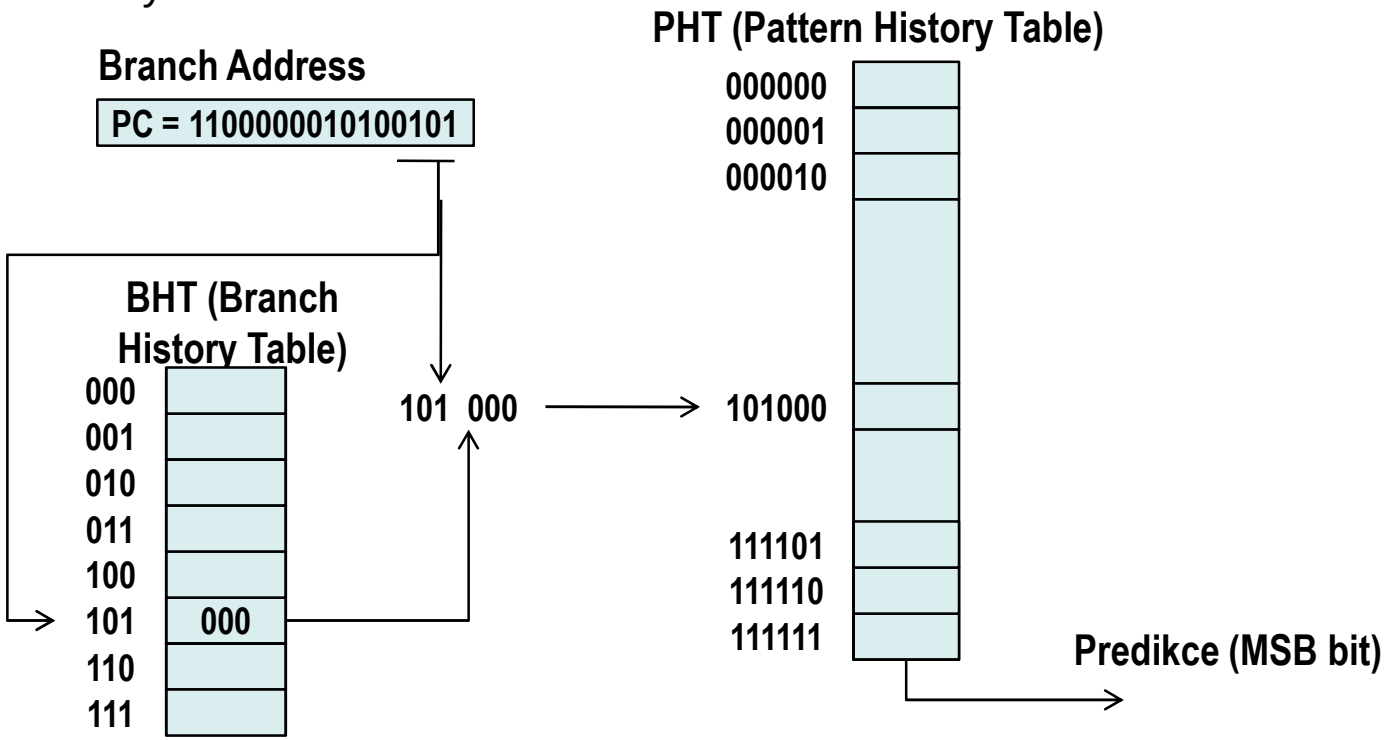
- Proč používat i lokální historii pro indexaci do PHT?

```
do{
...
}while(podmínka);
```

- Chování skokové instrukce může souviset s její vlastní minulostí…

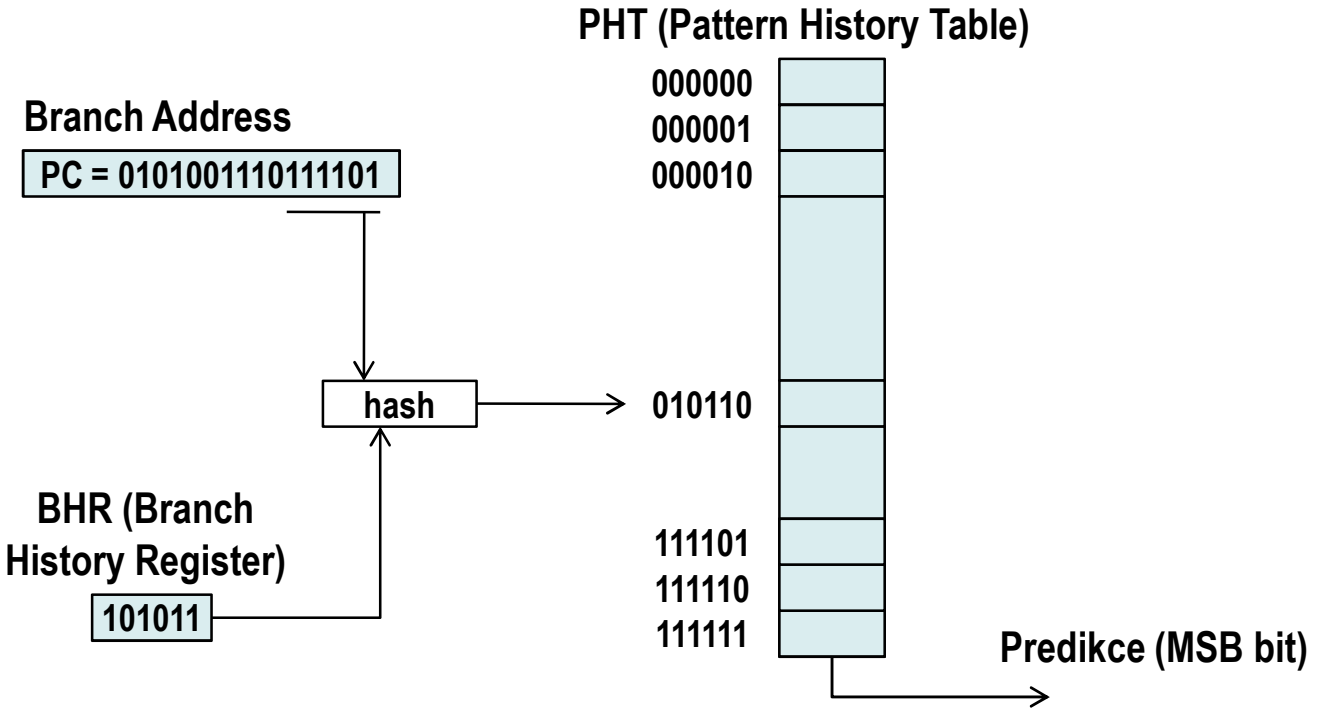
Predikce splnění podmínky větvení

- Dvou-úrovňový prediktor s **lokální** historií větvení a 3-bitovou tabulkou historie – Příklad:
- Předpokládejme, že na adrese 0xC0A5 je „loop-closing branch“ se vzorem: 1110111010111011101..., kde 1 znamená skok. Jak rychle se naučí uvedený prediktor správně predikovat skok a s jakou úspěšností? Nechť BHT a PHT jsou na začátku vynulovány.



Predikce splnění podmínky větvení

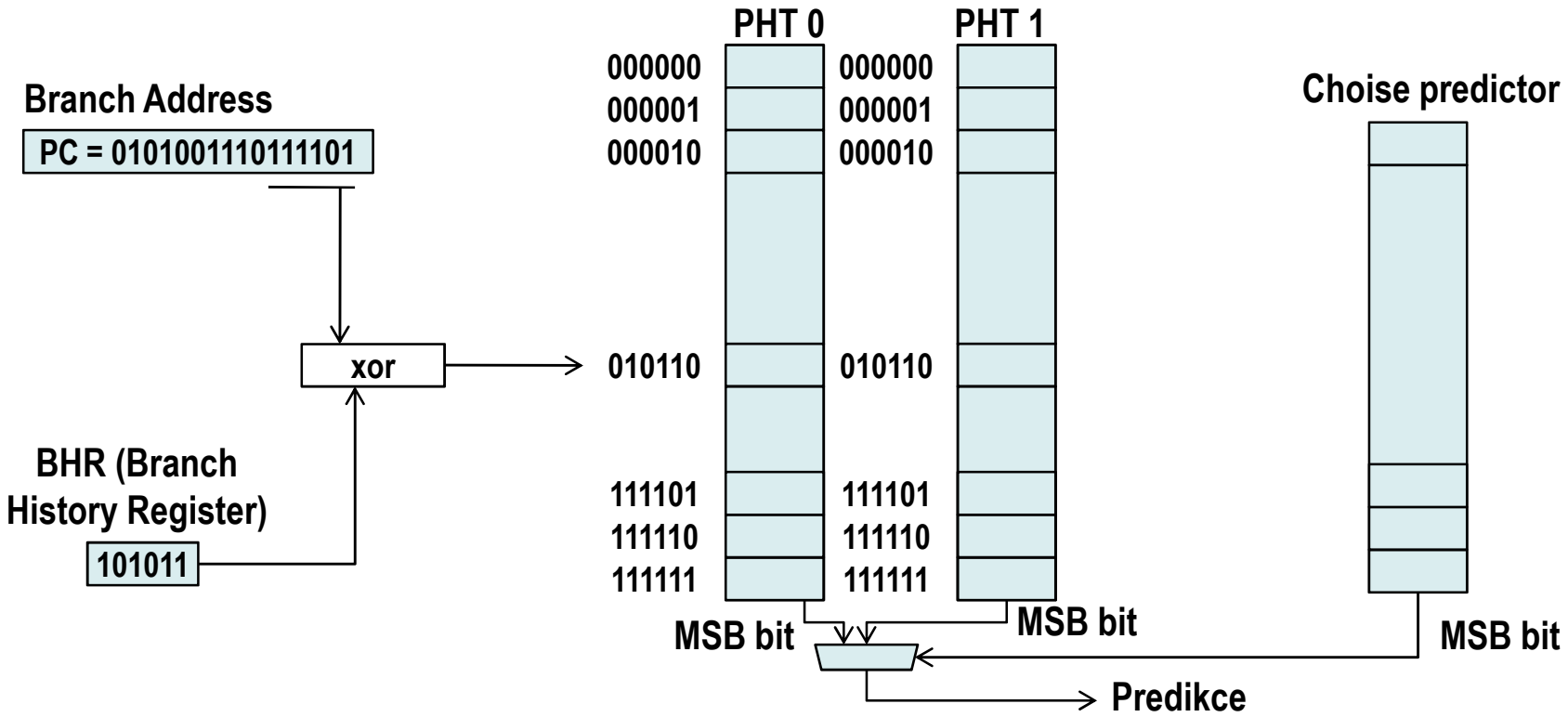
- **Index-sharing prediktory**… (hashují BHR a PC) – lepší využití bitů (větší historie..)
- Například gshare predictor:



gshare: Hash: XOR

Predikce splnění podmínky větvení

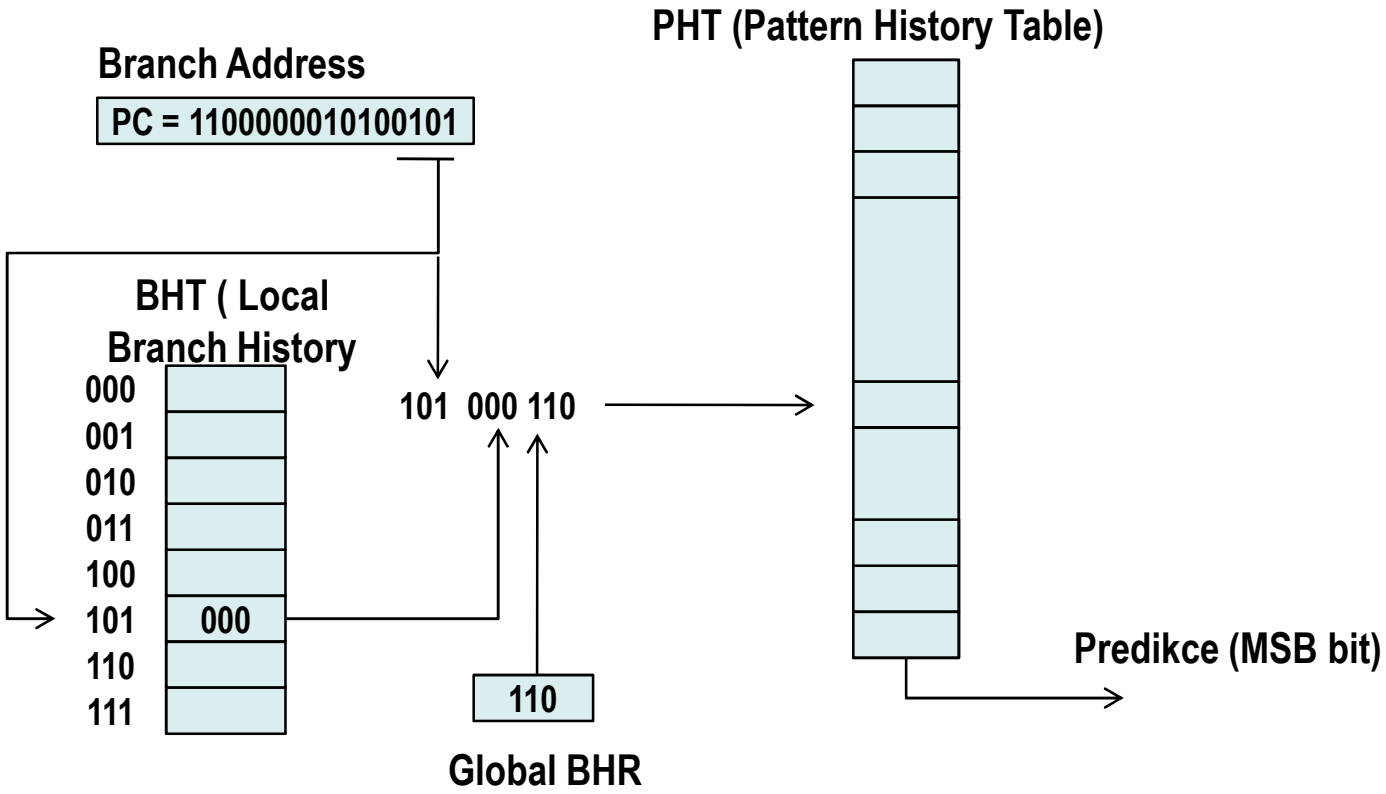
- **Index-sharing prediktory**…
- dvou-módový prediktor (bi-Mode) – dvě oddělené PHT – stejný hash:



- Choise prediktor – Smith. Důsledek: „Skákavé skoky“ se berou z jedné PHT, „neskákavé“ z té druhé PHT… (Statisticky skoky tuto vlastnost mají.)

Predikce splnění podmínky větvení

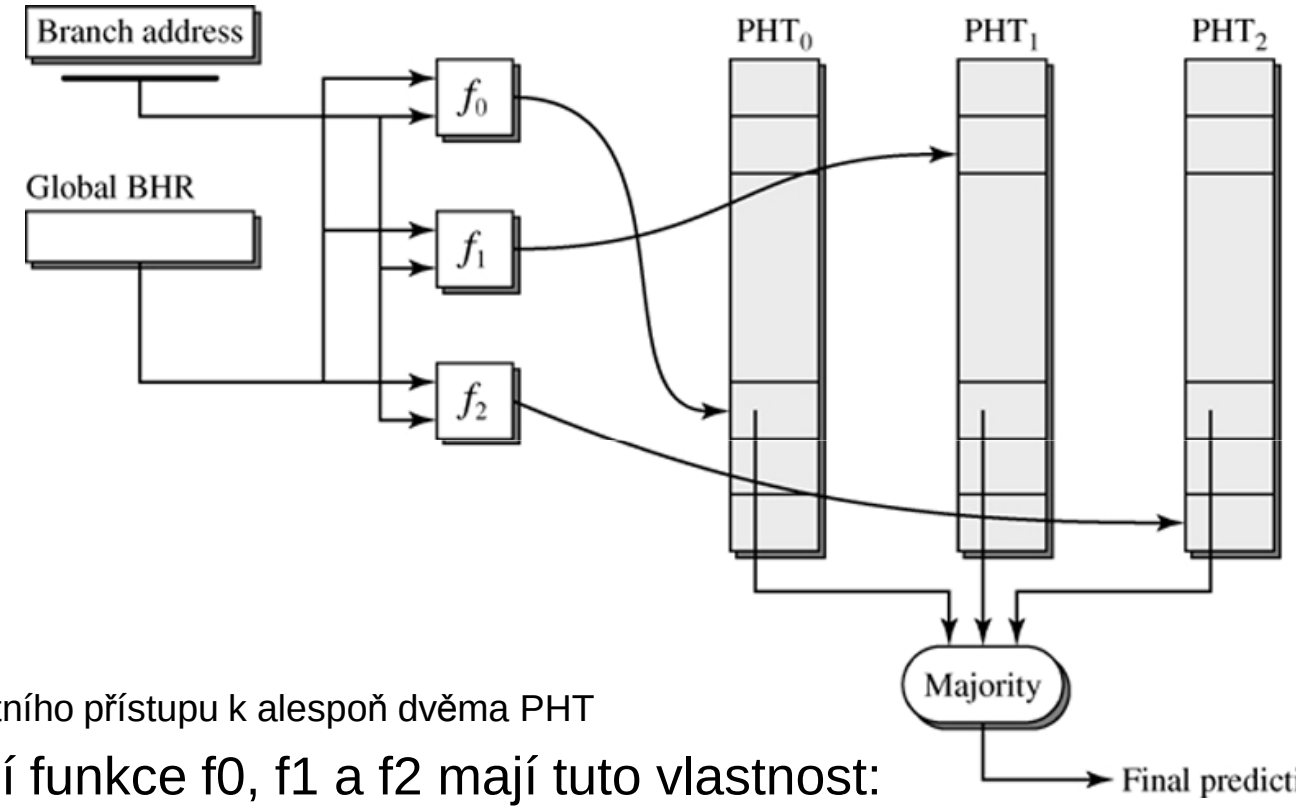
- **alloyed predictor**



- Spojuje část PC s lokální historií a globální historií pro indexaci do PHT

Predikce splnění podmínky větvení

- **gskewed predictor**

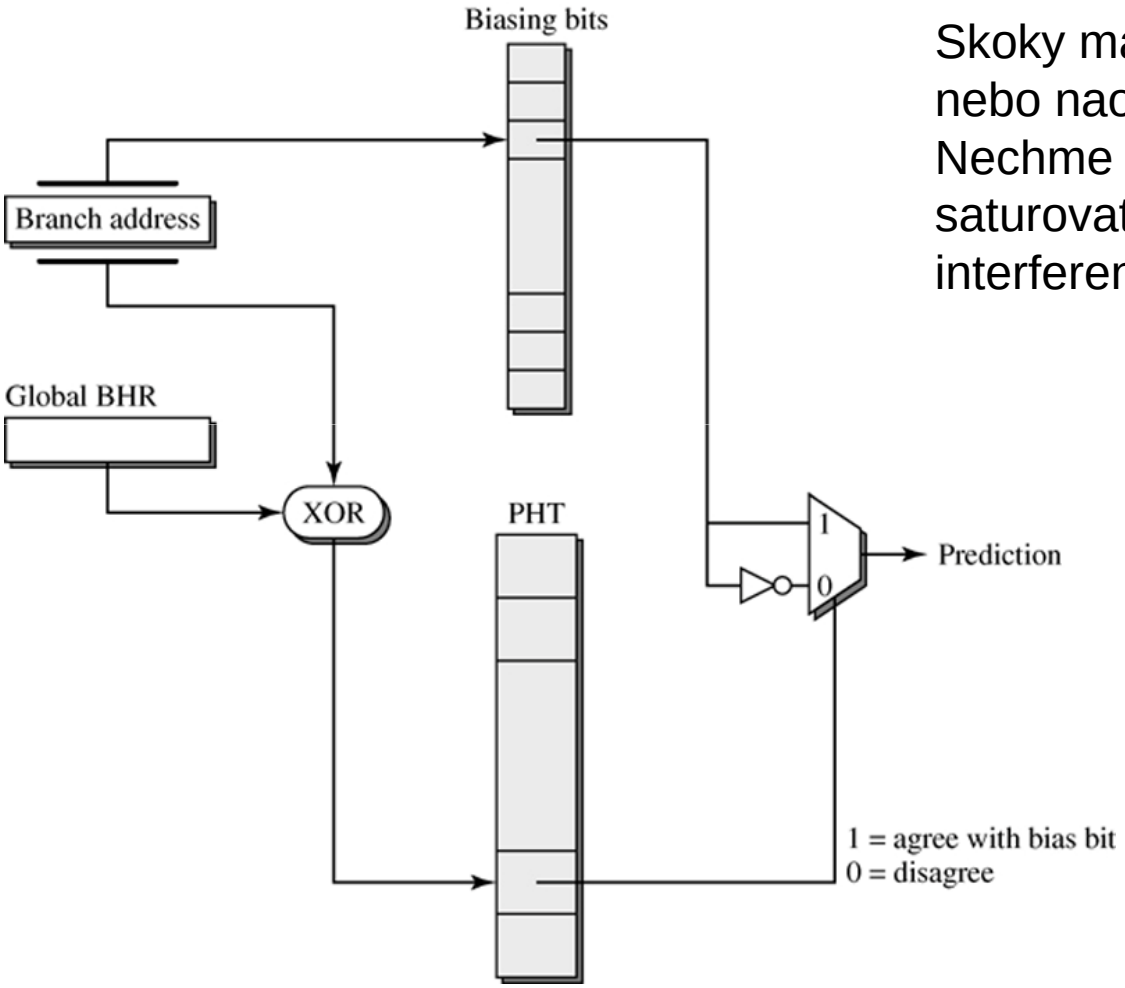


Garance nekonfliktního přístupu k alespoň dvěma PHT

- Hašovací funkce f0, f1 a f2 mají tuto vlastnost:
  Pokud f0(x1)=f0(x2) pro x1≠x2, pak f1(x1) ≠ f1(x2) a f2(x1) ≠ f2(x2)
- **Total update** (aktualizace všech bank bez rozdílu výsledkem dané skok. instrukce) a **partial update** (lepší) – neaktualizujeme danou banku i když dávala špatnou predikci, ale celková predikce byla správná

Predikce splnění podmínky větvení

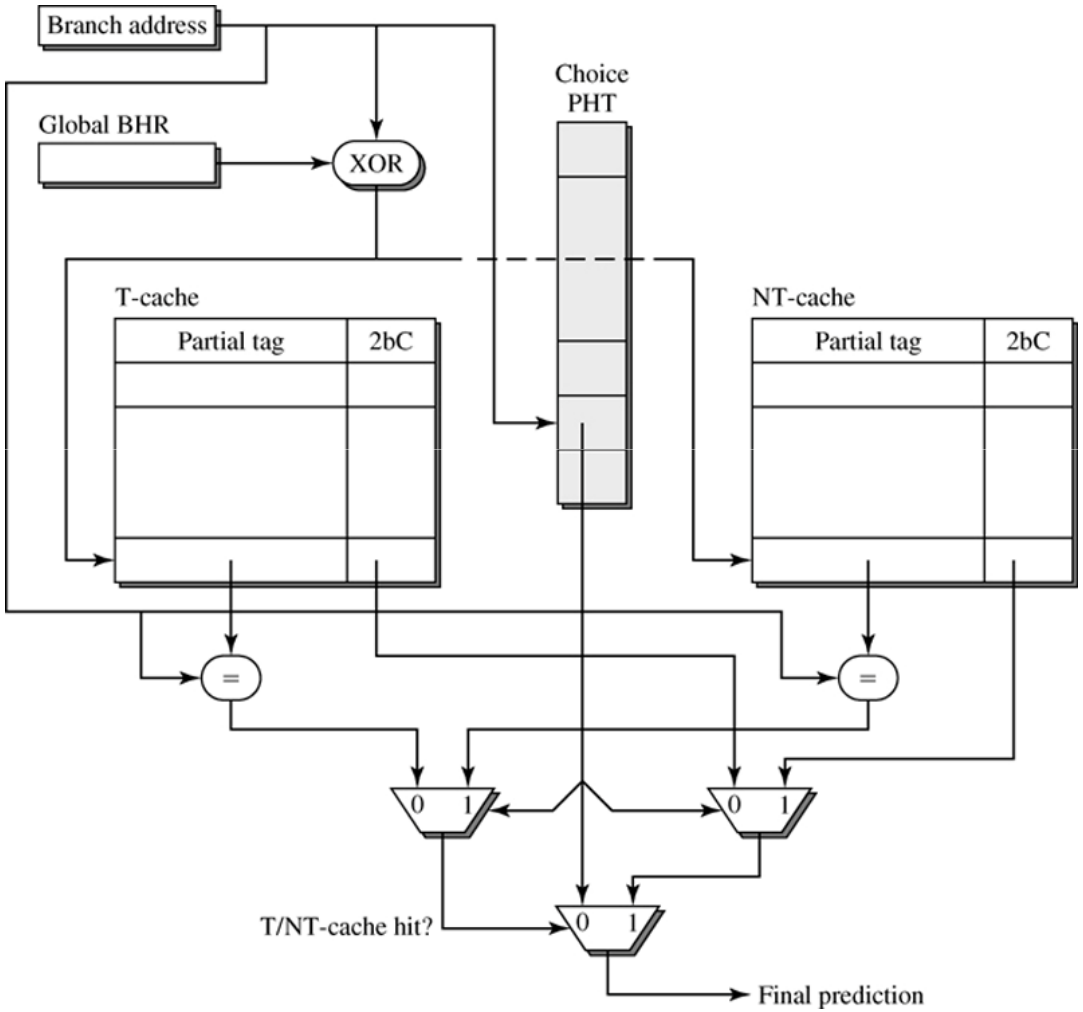
- **agree prediktor**



Skoky mají tendenci buďto skákat nebo naopak neskákat… -> bias
Nechme tedy počítadlo v PHT saturovat.. (Eliminace negativní interference)

Predikce splnění podmínky větvení

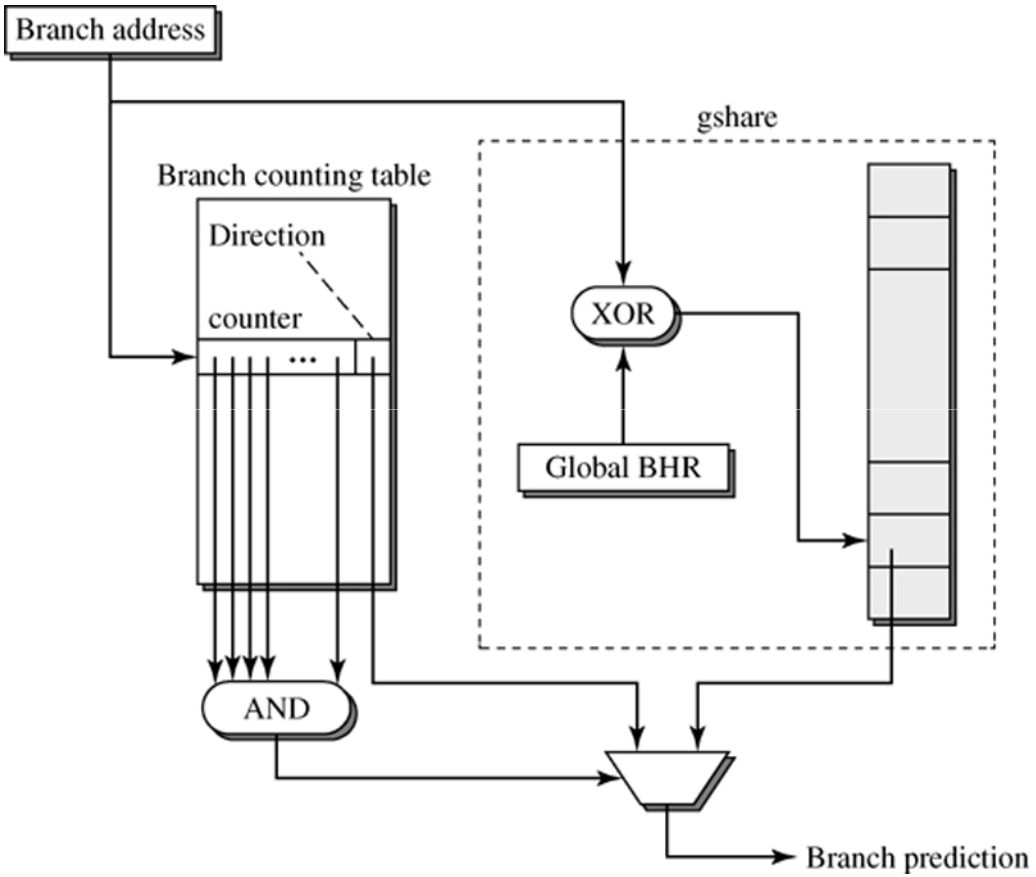
- **YAGS prediktor**



Je rozšířením Bi-mode prediktoru… Nicméně se zásadním rozdílem! Pokud PHT počítadlo indikuje „taken“, pak „NT cache“ je použita pro predikci. Obě cache si totiž pamatují pouze skoky, které nesouhlasily s PHT. PHT slouží jako pravidlo, a cache jako výjimky z tohoto pravidla. Cache miss pak znamená „souhlas“ s PHT.

Predikce splnění podmínky větvení

- **Filtering prediktor**



Víme, že skoky mají tendenci saturovat. Využijme toho a oddělme je. „Drahé“ HW struktury pak věnujme skokům se zajímavějším vzorem.

Predikce splnění podmínky větvení

Prediktory vzorce chování (Pattern-Based Predictors)

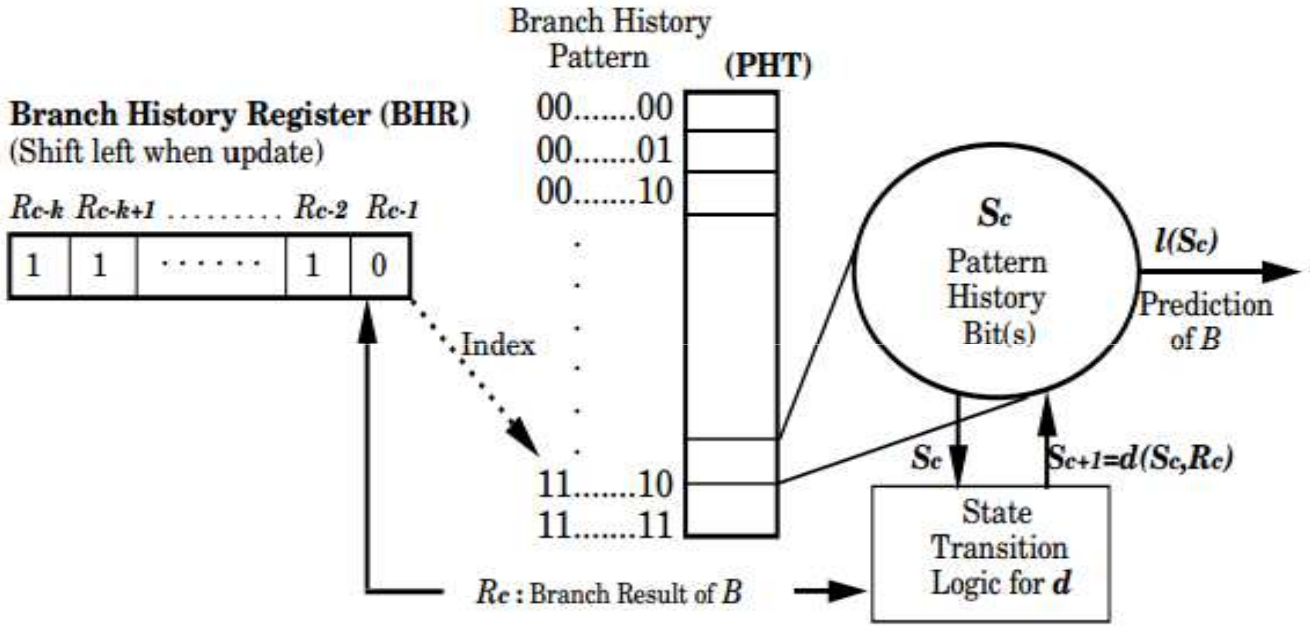
- Rozšířením myšlenky 1-bitové či vícebitové predikce je rozpoznávání vzorce v chování skoku
- Př. Vzorce chování skoku (patterns 0-neproveden, 1-proveden)
  0101010101, 110110110, 00111010011101, …
  Tyto vzorce chování jsou obtížně předpověditelné pomocí klasické 1-bitové či 2-bitové predikce, přitom jde o pravidelné chování (vzorec)

- **0101010101** – Po každé 0 přijde 1, po 1 přijde 0
- **110110110** – Po 11 přijde 0, po 10 přijde 1, po 01 přijde 1
- **00111010011101** – Po 100 přijde 1, po 001 přijde 1, po 011 přijde 1, po 111 přijde 0, po 110 přijde 1, po 101 přijde 0, po 010 přijde 0

- Rozpoznávaly předchozí prediktory vzor ???
- Pozn: Prediktory používající kombinaci lokální historie (příslušného skoku) a globální historie (předchozích skoků) se nazývají *korelační prediktory*

Predikce splnění podmínky větvení

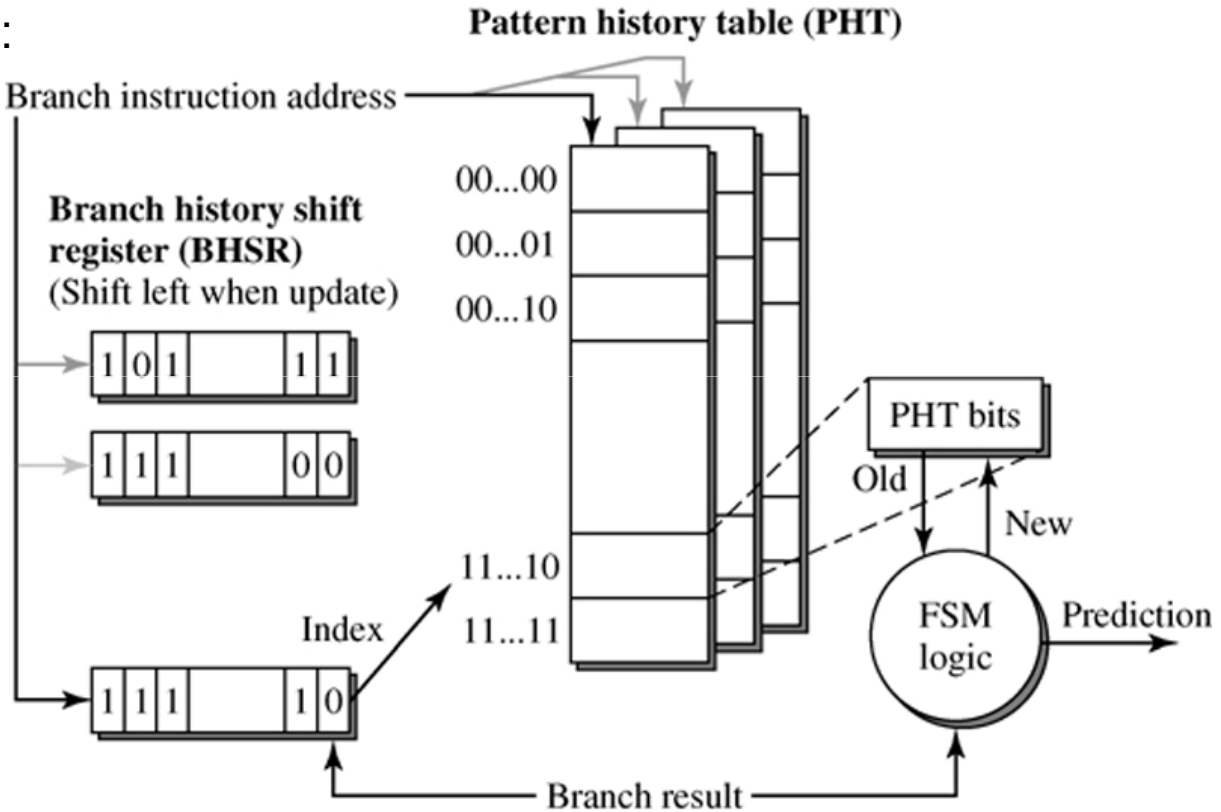
- **Two-level adaptive branch predictor** (též **Correlating/ed predictor**)
- Autoři: Yeh a Patt z University of Michigan



- První level – historie posledních K skoků (např. posledních K skokových instrukcí, nebo také K skoků téže instrukce) – vytváří vzor
- Druhý level – chování posledních N výskytů tohoto vzoru indexuje PHT
- BHR je asociován s danou skokovou instrukcí

Predikce splnění podmínky větvení

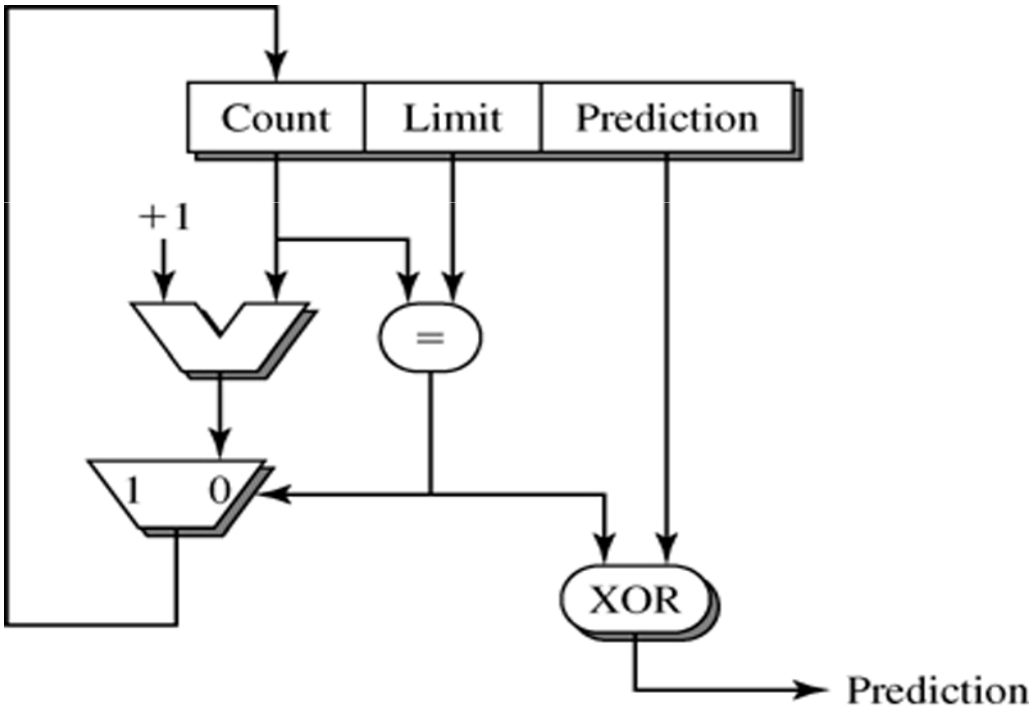
- **Two-level adaptive branch predictor** (též **Correlating/ed predictor**)
- Autoři: Yeh a Patt z University of Michigan
- Totéž, ale jiný pohled:
  (Intel P6)



- Někdy se zase PHT kreslí jako matice, kde řádky (sloupce) indexuje část PC a sloupce (řádky) indexuje BHR, případně může být použit i Global BHR (GBHR)

Predikce splnění podmínky větvení

- **Loop Counting predictor**
- Specializovaný prediktor pro zachycení mnoha iterací
- Používá se pouze v kombinaci s dalším prediktorem (viz hybridní prediktory)
- Pentium M



Predikce splnění podmínky větvení

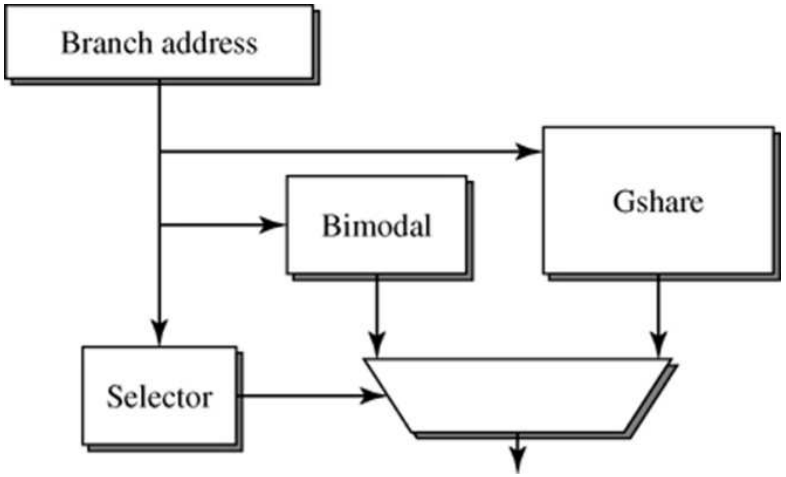
- Sofistikovanější prediktory berou v potaz negativní a neutrální interferenci (bi-Mode prediktor a další…)

Některé další dynamické prediktory:

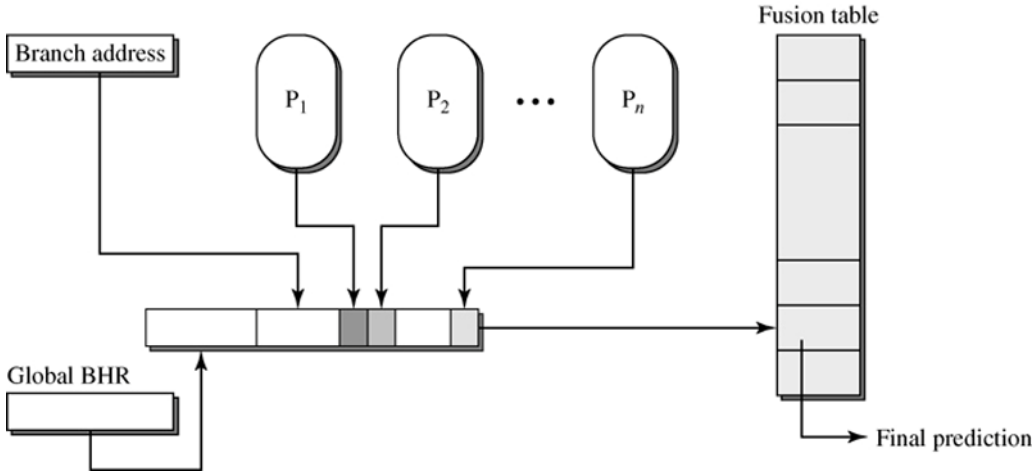
- Perceptronové prediktory (větší historie, odhalí korelované skoky)
- Data flow prediktory – explicitně sledují meziregistrové závislosti

Hybridní, multi-hybridní a fusion-based prediktory:

- Tournament prediktor – dva prediktory P0 a P1 a metaprediktor M (Smithův).



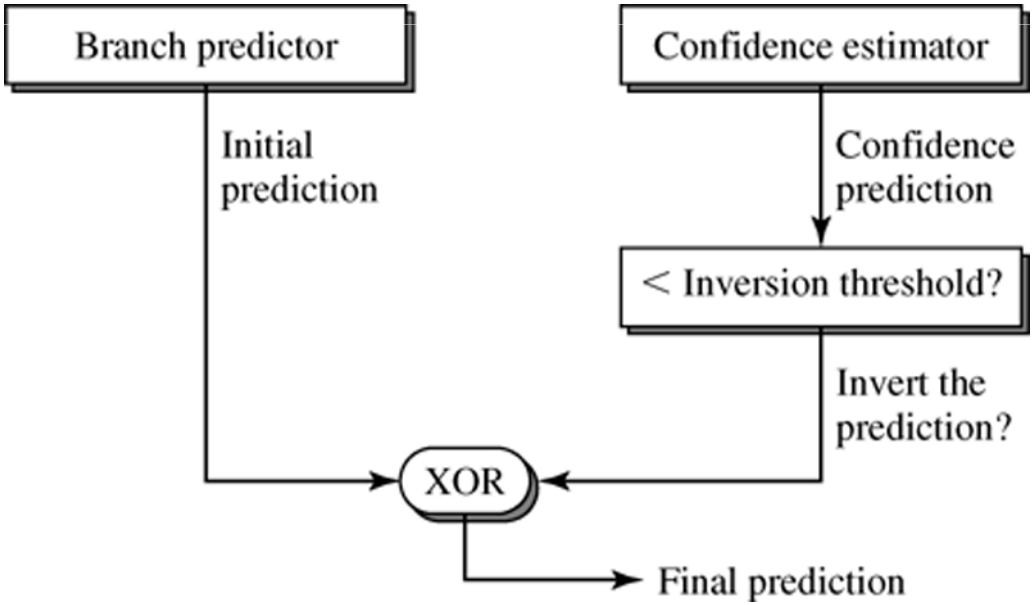
(a) Hybridní: Tournament prediktor nyní implementující Bimodalní a Gshare prediktor



(b) Fusion-Based Hybrid Predictor – odstraňuje multiplexor (který vybíral pouze jeden prediktor - Selection) za **Fusion**

Predikce splnění podmínky větvení

- **Selective branch inversion (SBI)**
- Nesoustřeďuje se primárně na redukci interference, ale spíš koriguje její dopady
- Počítá (odhaduje) spolehlivost predikce použitého prediktoru
- Například SBI Bi-Mode prediktor dává lepší výsledky než samotný Bi-Mode (zde: interference avoidance + interference correction)



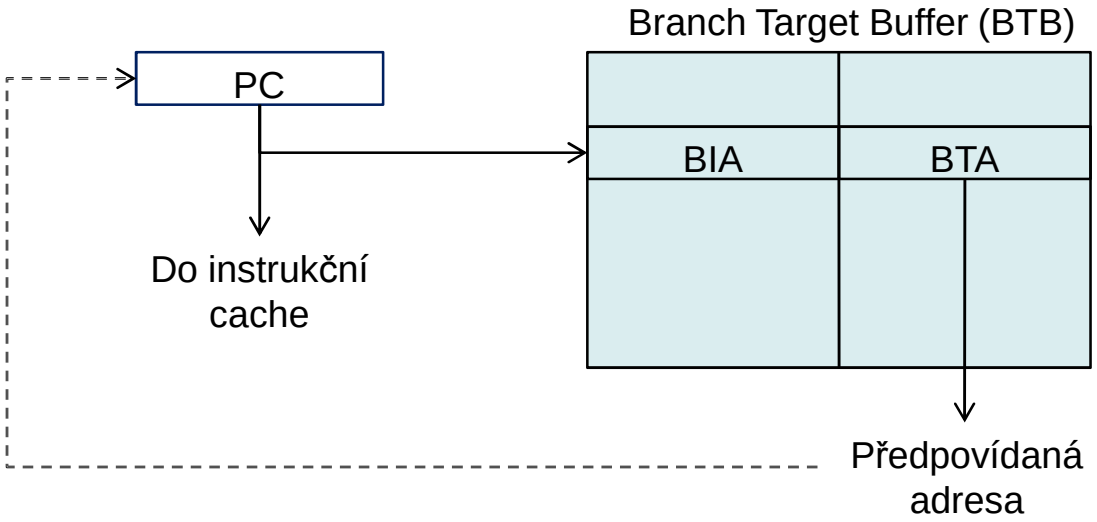
Predikce větvení

- Dvě fundamentální složky:
  - branch condition speculation (zda vůbec),
  - branch target speculation (kde).

O tom jsme se bavili doposud… (od slide 10 dál)

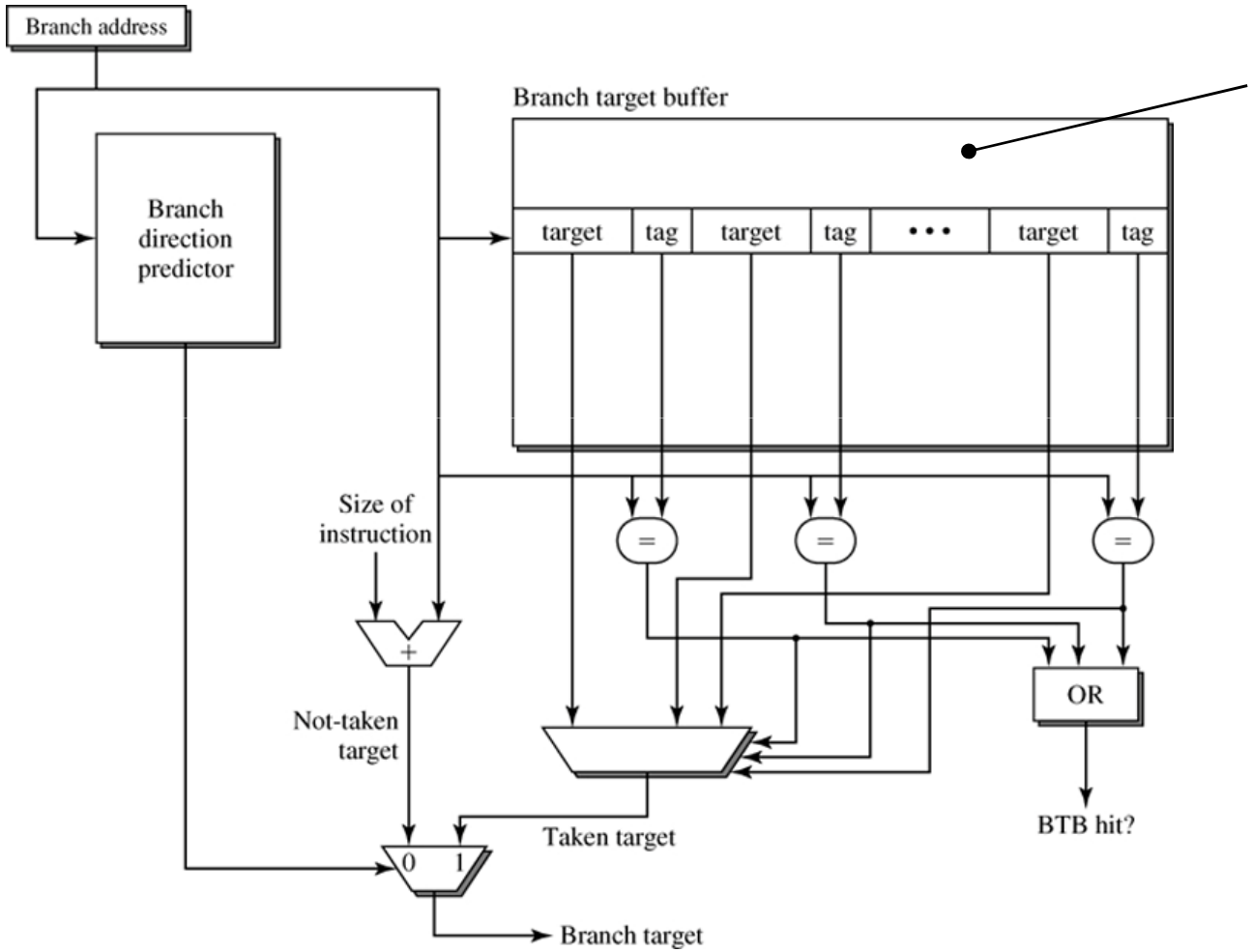
Na tohle se podíváme nyní…

- Predikce cíle větvení:
  - BTB (Branch Target Buffer) – asociativní cache obsahující dvě položky: BIA (Branch Instruction Address) a BTA (Branch Target Adress)



Predikce **cíle** větvení

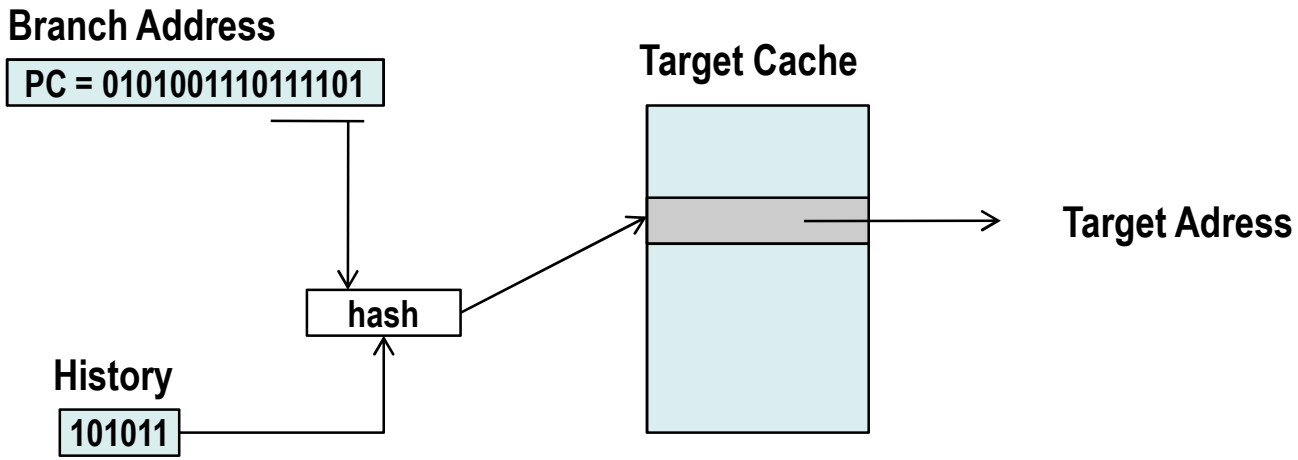
- **Cíl skoku**: PC-relativní nebo nepřímý (adresa je určena za běhu)



Co byste zde ukládali? Zamysleme se nad využitím cache…

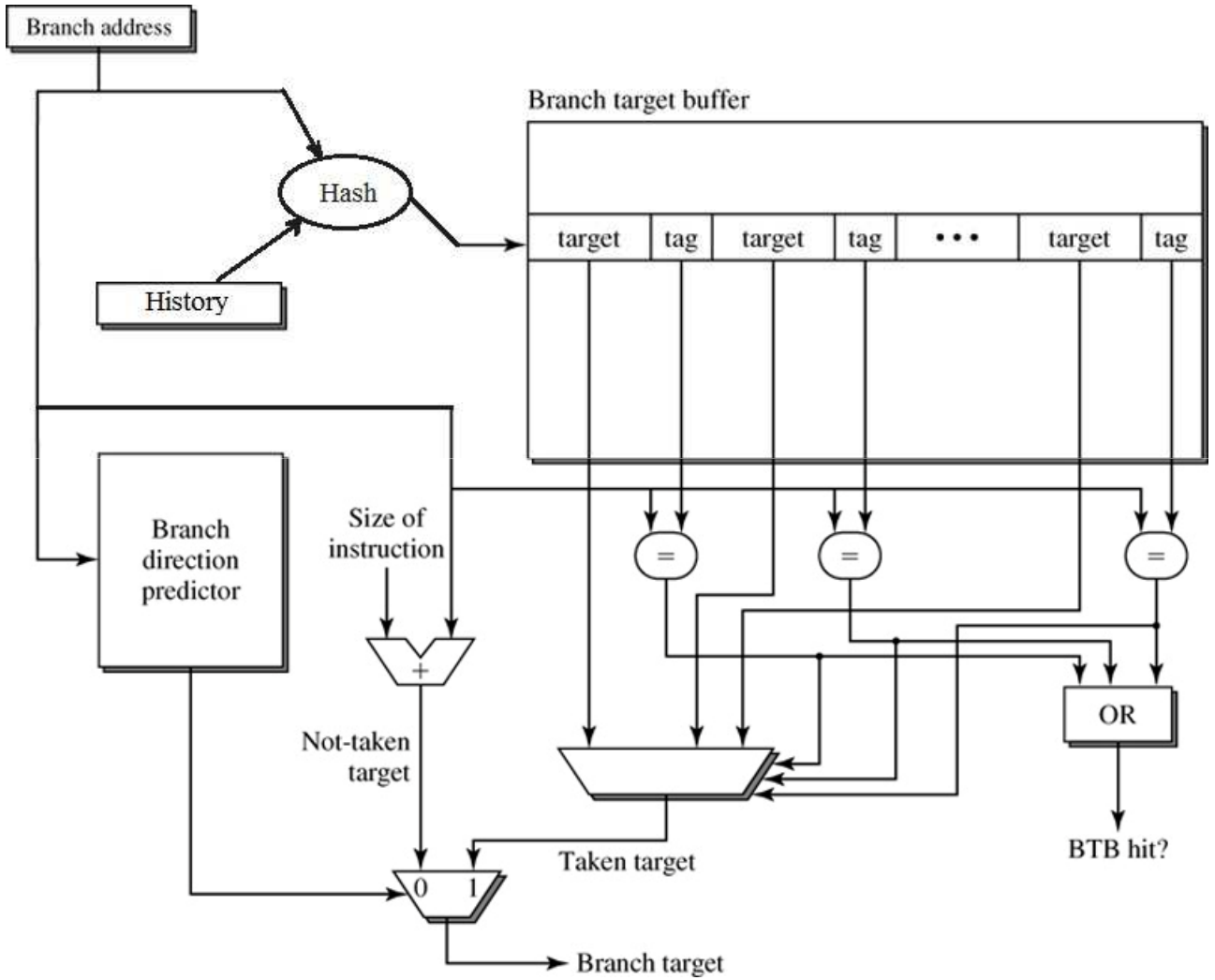
Predikce cíle větvení – A co nepřímé skoky?

- MIPS: jal $ra - kam vlastně máme skočit ???
- Objektové programování a polymorfizmus… Výsledkem je mnohem více nepřímých skoků než při použití imperativního programování
- Základní myšlenka je zde:



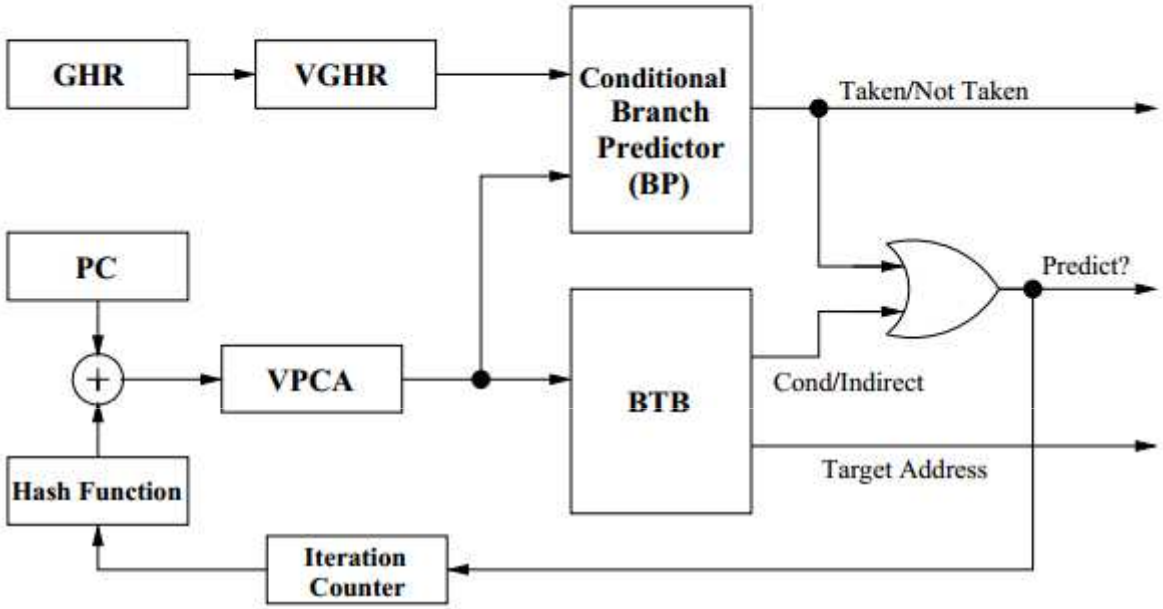
- Řešení tedy spočívá v indexaci BTB (Branch Taget Buffru)
- Hash-em může být i „pouhá“ konkatenace
- Jiný pohled: Jedná se o 2-úrovňový prediktor s tím rozdílem, že cache nyní neobsahuje záznamy o historii skoků, ale adresy návěstí kam skáčeme

Predikce cíle větvení – Řešme kolize v cache…



Virtual Program Counter (VPC) Prediction

Příklad (pro ilustraci):



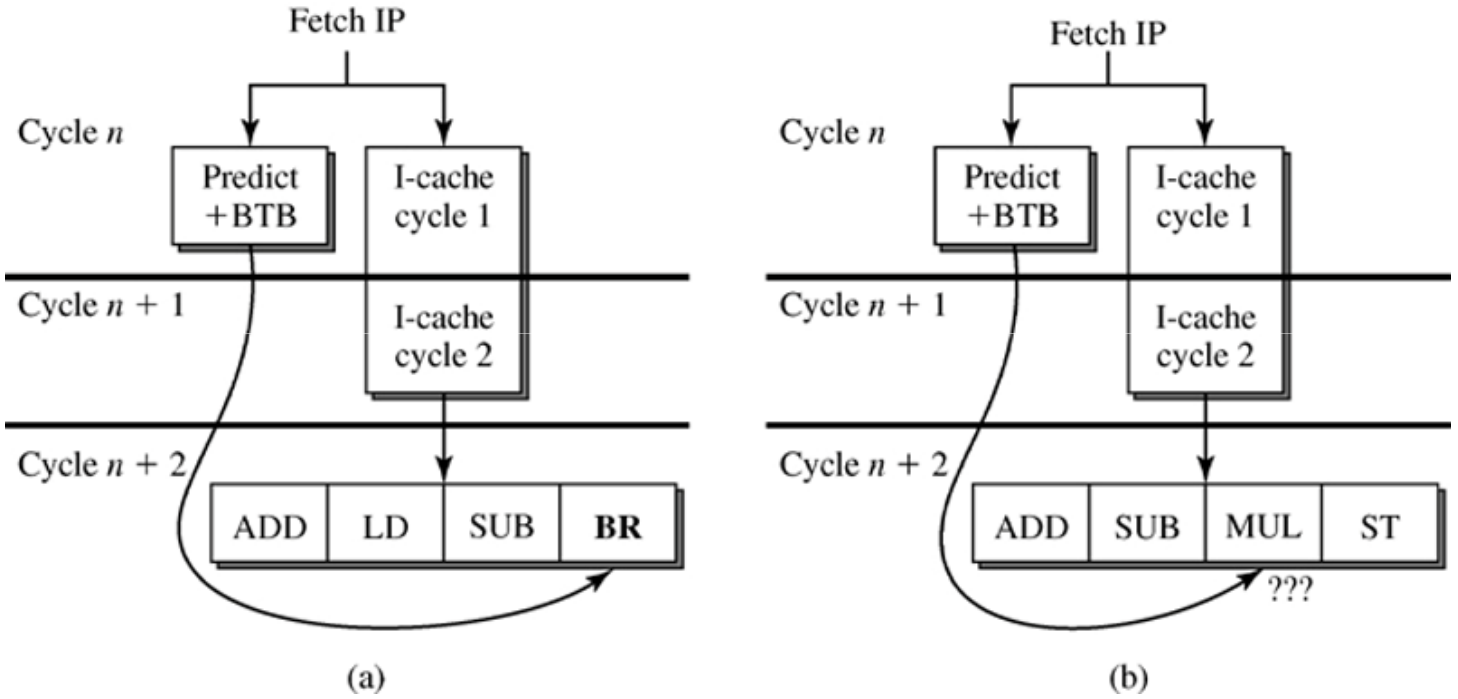
Použité zkratky:

- GHR – Global History Register, VGHR – virtual GHR
- VPCA – Virtual PC Address

- V první iteraci je VPCA = PC; a VGHR = GHR;

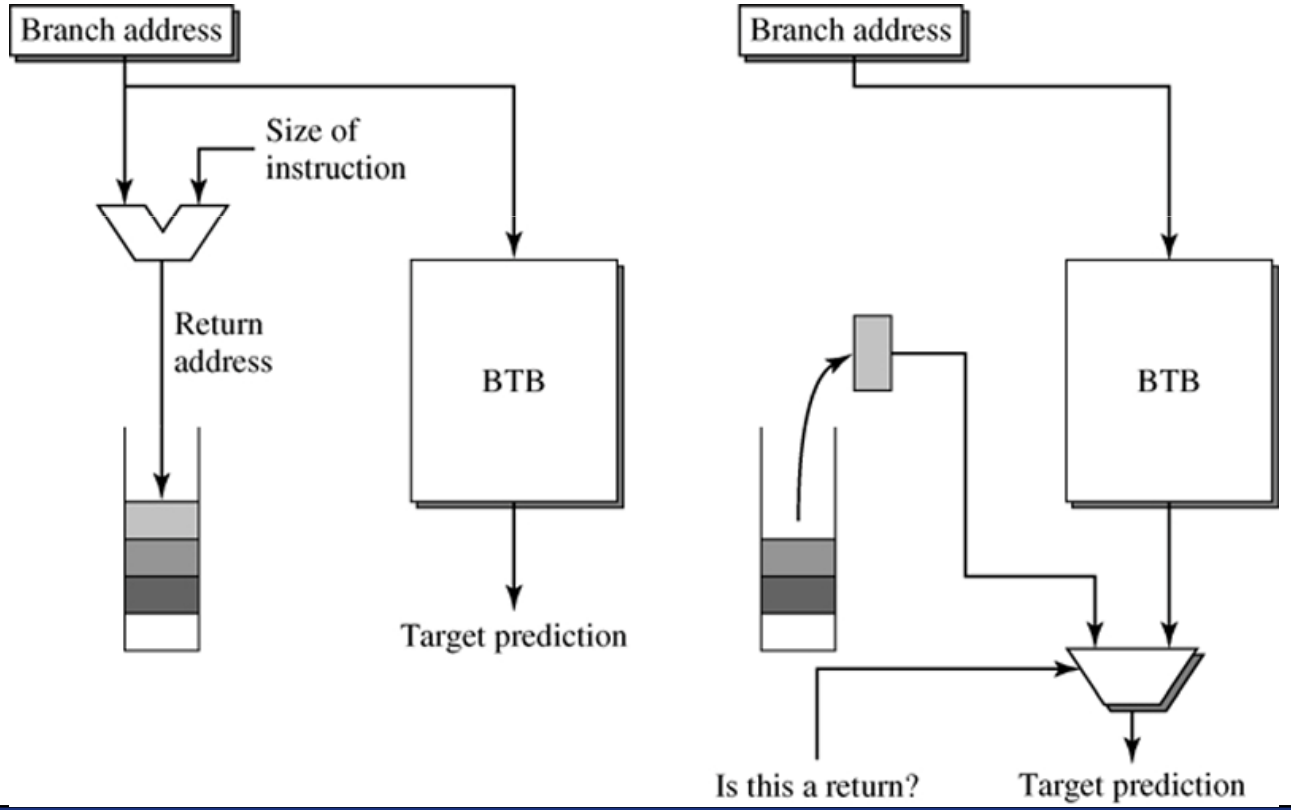
Phantom branch / bogous branch

Proč dochází k těmto falešným předpovědím?



Predikce **cíle** větvení

- **RAS prediktor** - return address stack – specializovaný prediktor
- k předpovězení návratové adresy instrukce RETURN postačí HW zásobník. Hloubka 8 poskytuje >97% úspěšnost predikce
  (MIPS R10000 hloubka 1, Alpha 21164 hloubka 12, Intel Pentium III hloubka 16)



U spekulace ale pozor

Moderní procesory užívají:

- řídicí spekulace (u podmíněných skoků - dnešní standard)
- datové spekulace (Load/Store spekulace)
  - Load se provede dříve, než je známa adresa předchozích Store (např. Itanium, Power 5, Core 2)
- Musí být splněna korektnost provádění programu
- To znamená: program splnil
  - Podmínku 1 - dosahuje správný výsledek (podle sekvenční sémantiky), tedy jako při provádění procesorem bez jakéhokoliv zřetězení,
  - Podmínku 2 - generuje stejná přerušení (výjimky) jako procesor bez jakéhokoliv zřetězení.
- Pro splnění korektnosti existují následující postačující podmínky:

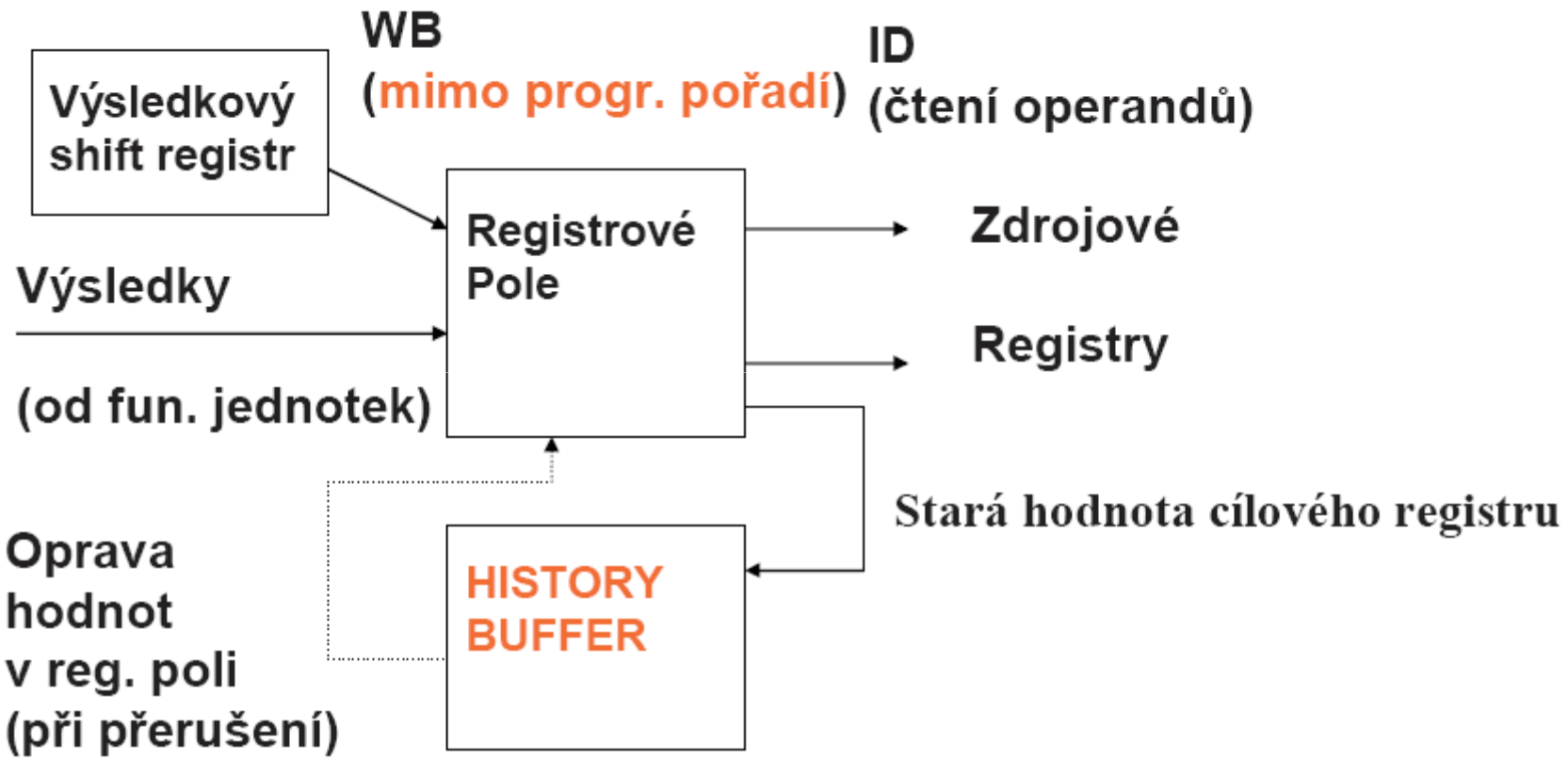
Postačující podmínky

- Podm. A – respektování datových závislostí
  - instrukce čekají na jejich vyřešení
- Podm. B – respektování řídicích závislostí
  - skok se neprovede, dokud není známa adresa příští instrukce
- Podm. C – Přerušení je přerušením přesným.
- Při spekulativním provádění se částečně nebo dočasně narušují podmínky A a B.
- Korektnost provádění programu ale splněno být musí, Podm. C trvá.

Chybná spekulace?

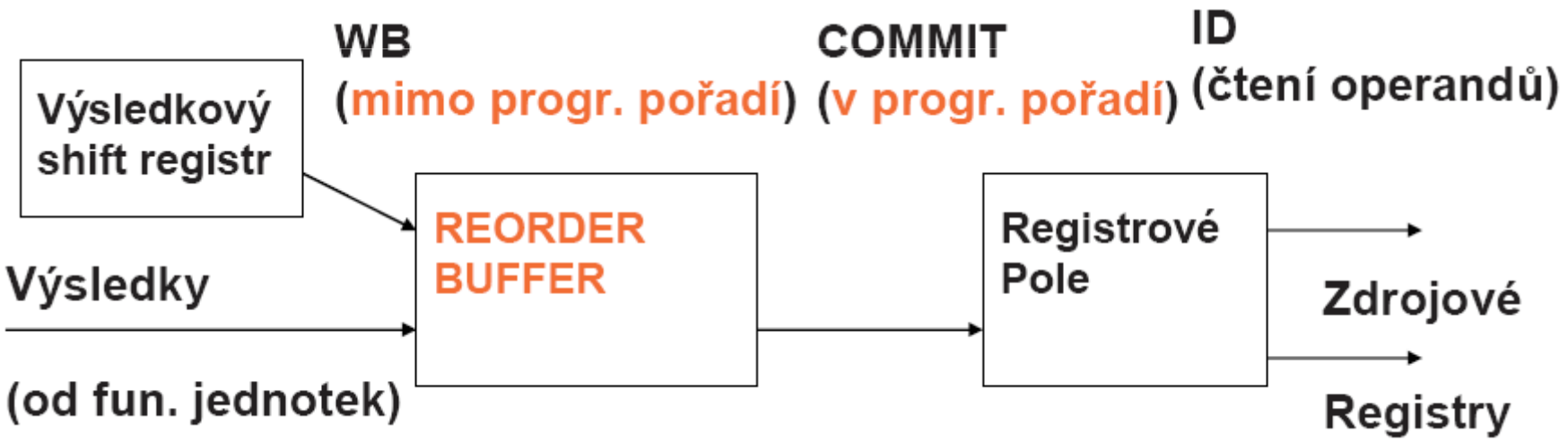
- Musí nastat: Zotavení, Restart. Obě akce jsou relativně nákladné (až desítky ztracených taktů).
- Spekulaci a zotavení podporují dvě HW fronty
  - **History Buffer** HB - M88110,…
  - **Reorder Buffer** RB – ostatní procesory. - na ten se zaměříme…

Řešení - History Buffer HB



Řešení - Reorder Buffer RB

- Výsledky instrukcí zapíšeme ve stupni WB do REORDER BUFFERU,
- z něj ve fázi COMMIT zapíšeme obsah do registrového pole až pokud
- všechny předchozí instrukce byly dokončeny.

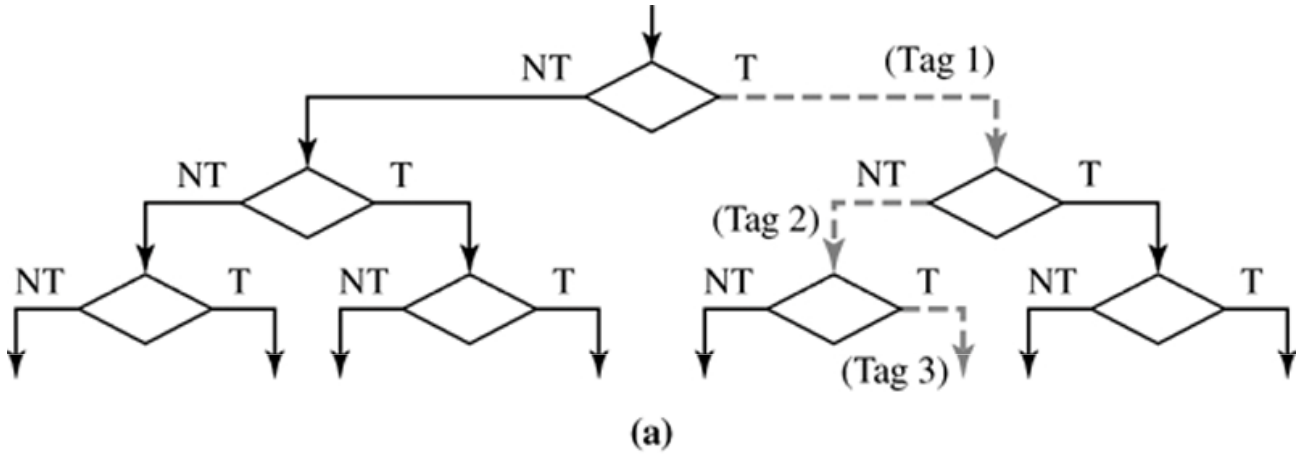


History Buffer vs Reorder Buffer

- Výhody HB
  - Hodnoty z history bufferu se nemusejí forwardovat (není asociativní prohledávání HB) => jednodušší forwarding.
- Nevýhody HB
  - Nutnost rekonstruovat stav registrů sekvenčním procházením HB,
  - Dodatečný čtecí port pro registrové pole (problém se škálováním pro superskalární procesory).
- Výkonnost (za běhu) obou metod je srovnatelná, při přerušení však má HB značnou nevýhodu. Opět zde je problém s instrukcí STORE.

Zotavení po chybné předpovědi…

- Branch speculation – o tom jsme se bavili doposud
- Branch validation/recovery – kdy se dozvíme skutečný směr skoku? a co pak?



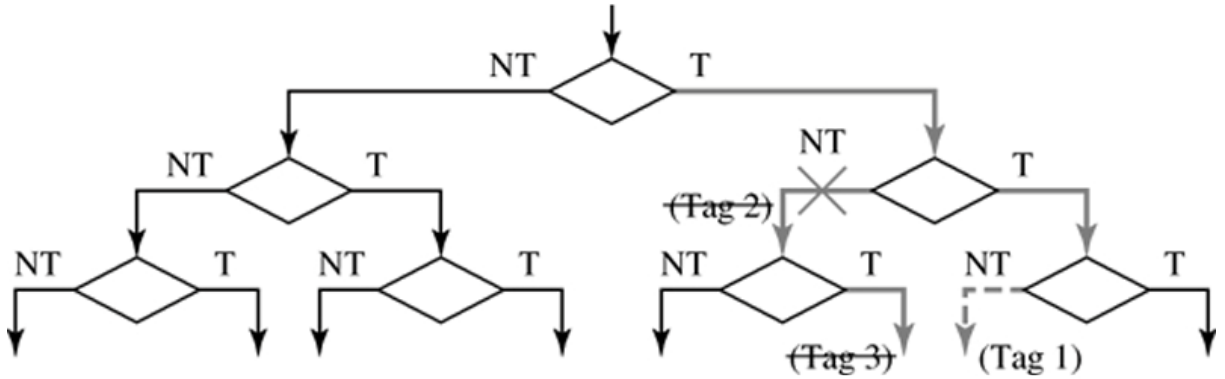
(a)

- Během Branch speculation jsou označovány všechny instrukce v prováděném bloku unikátním Tag-em. Takto označená instrukce indikuje zda je spekulativní a její Tag identifikuje spekulativní blok, kterému patří.

- Během Branch speculation jsou ukládány (do buffru) adresy všech skokových instrukcí (příp. adresy následující) – nutné pro Branch recovery

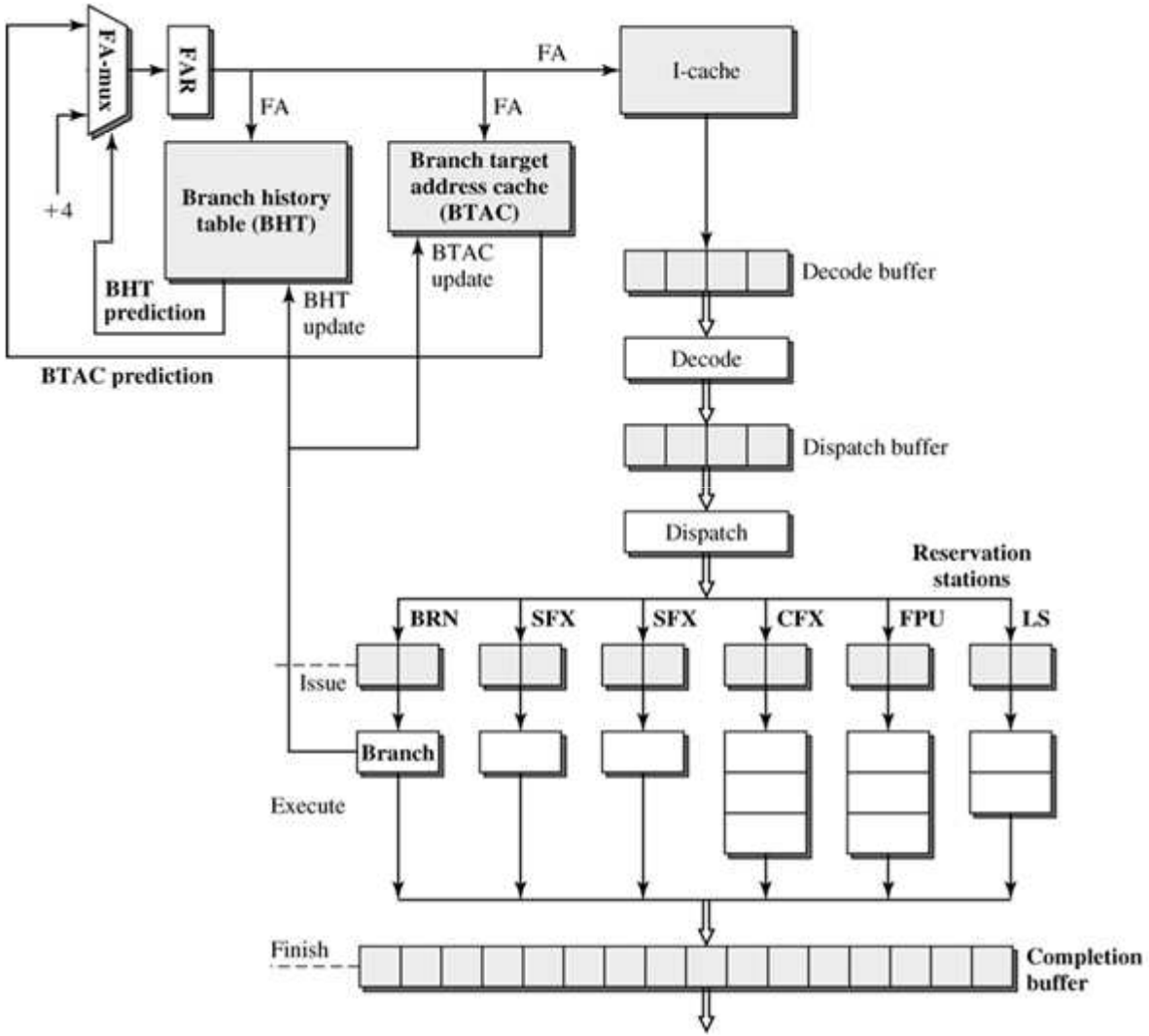
Zotavení po chybné předpovědi…

- Branch validation/recovery – nastává v okamžiku kdy již známe skutečný směr skoku (zda jsme měli skočit nebo ne)



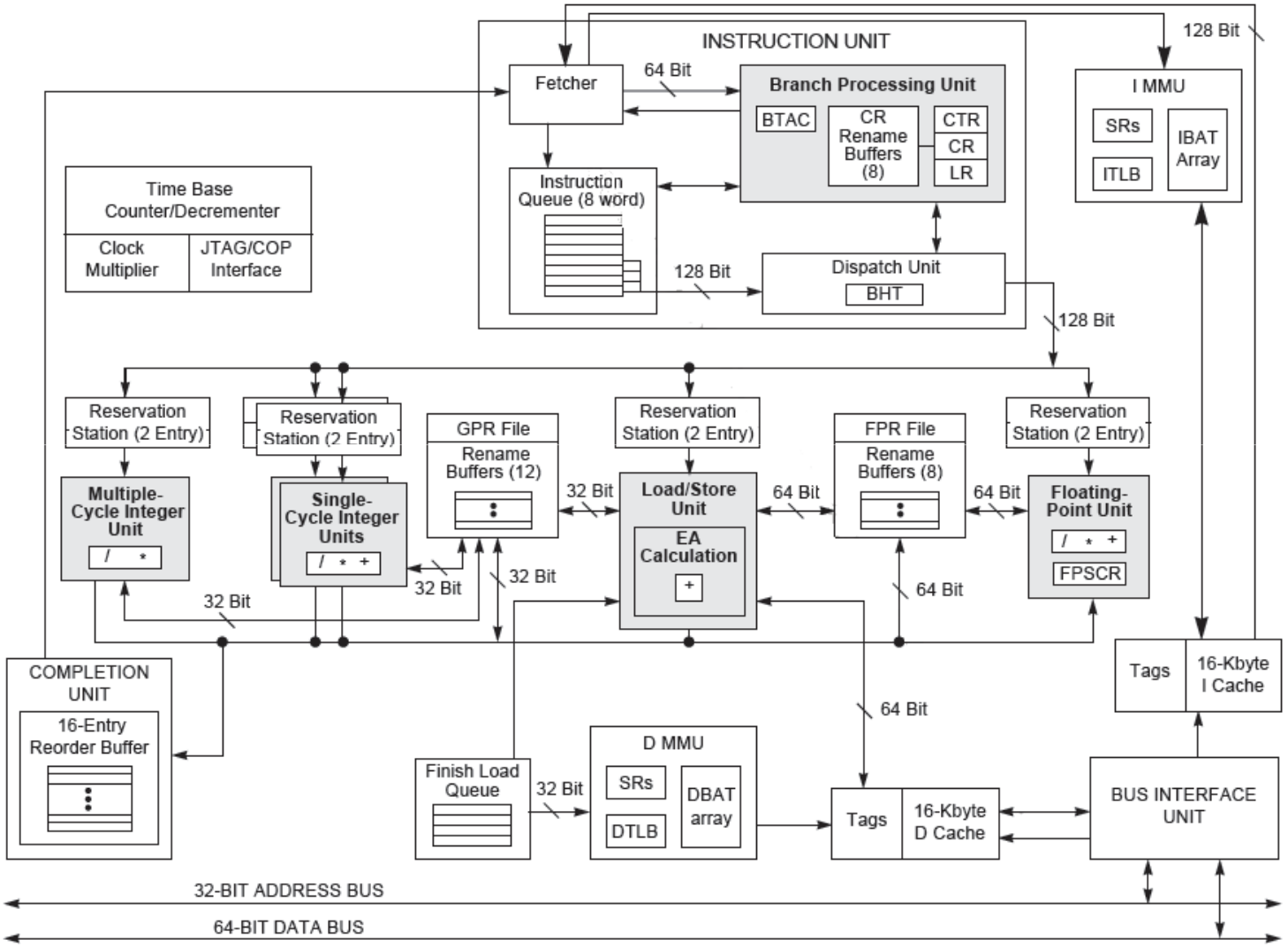
- **Správná predikce**:
  Tag patřící danému bloku je dealokován a všechny instrukce označené tímto tag-em se stávají nespekulativní (je jim umožněno dokončení).
- **Nesprávná predikce**:
  Dvě akce: Ukončení celé nesprávné větvě a zahájení vykonávání té správné.
  Ukončení větve znamená: Identifikovat všechny nesprávné instrukce (tagy všech bloků nesprávné větve) – zneplatnění instrukcí v decode a dispatch bufrech, reservačních stanicích, dealokace prostoru v reorder buffru…
  Zahájení vykonávání správné větve: Aktualizace PC – buďto nově vypočtenou/získanou adresou nebo pomocí adresy uložené během Branch speculation

Predikce skoku - PowerPC 604



- Superskalární procesor šířky 4 (schopen přinést, dekódovat nebo dokončit 4 instrukce/cyklus)
- Schopen vykonat 6 instrukcí/cyklus
- BTAC – Branch Target address cache (plně asociativní, 64 položek)
- BHT – Branch History Table (přímo mapovaná, 512 položek)
- BTAC – jeden cyklus, BHT – dva cykly
- Organizace FA-mux je komlexnější než ukazuje obrázek.

PowerPC 604 – trochu jinak…



Předvýběr instrukcí…

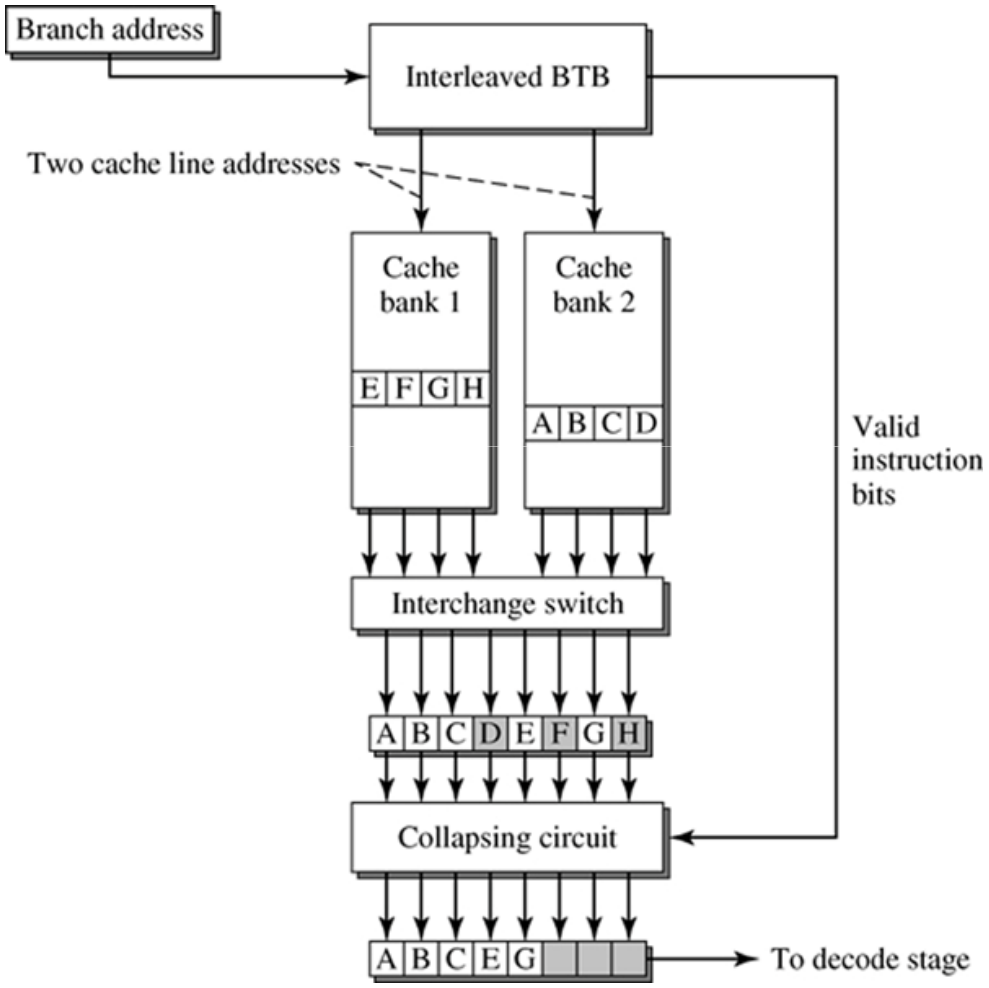
- Pro superskalární procesor je nesmírně důležité správně předpovídat směr skoku a cíl skoku !!!
- V jednom cyklu může být nutno zpracovat několik skokových instrukcí – například pokud fetch grupa má délku 4, pak všechny 4 instrukce mohou být skokové. Ideálním případem je pak použití adres všech skokových instrukcí… Někdy první skoková..
- Dalším mechanizmem spekulace je tzv „**predikace**“ (angl. predication / eager execution) – což je vykonávání obou větví programu
  - Predikace poskytuje dvě jistoty – první: garance, že procesor bude vykonávat „užitečnou“ práci; druhá jistota: garance, že procesor bude vykonávat „neužitečnou“ práci – a to i v případě vysoce predikovatelných skoků. Další faktor: power consumtion

High-Bandwidth Fetch Mechanism

- Výběr instrukcí z i-cache je typicky limitován na výběr z jedné cache line v daném časovém okamžiku
- Typicky, fetch grupa obsahuje skokovou instrukci (obvykle 20% instrukcí jsou skokové instrukce; vzdálenost mezi místem skoku a příští skokovou instrukcí je 5-6 instrukcí pro běžné programy)
- Jenom, pokud je „skákající“ skoková instrukce umístěna ve fetch grupě / cache line jako poslední instrukce, celá fetch grupa obsahuje užitečné instrukce. V opačném případě fetch grupa/i-cache není schopna dodat N instrukcí (N-šířka grupy). Jaký by byl důsledek???
- Podobný problém vzniká pokud je požadovaný blok instrukcí rozložen ve dvou cache lines.
- Řešením je :
  - **Collapsing buffer** – který zarovnává nesousedící instrukce do bloku
  - **Trace cache –** která trasuje (pro daný skok) sekvenci následných instrukcí – a ty pak vykonává – pokud je hit v trace cache, nevybíráme z instrukční cache…

High-Bandwidth Fetch Mechanism

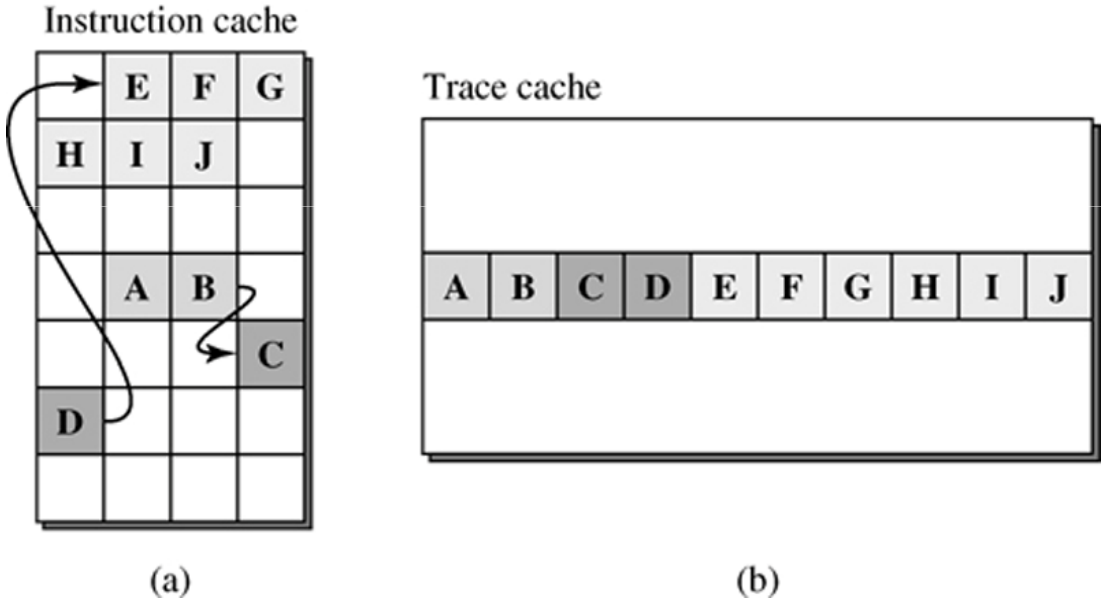
- **Collapsing buffer** – který zarovnává nesousedící instrukce do bloku



- Vyžaduje cache rozdělenou po bankách – banked instruction cache – poskytující víc než jednu cache line paralelně
- a interleaved BTB.
- Představme si, že chceme vykonat sekvenci instrukcí:
  A, B, C, E, G – tzn. C je skoková a přeskakuje D; instrukce E skáče na G.
- Konvenční cache by dodala: A,B,C,D (jedna neužitečná instrukce) a v dalším cyklu pak E,F,G,H (dvě neužitečné)
- BTB musí poskytovat informace (valid bits) specifikující položky v cache line platné pro predikovanou cestu
- **Organizace Collapsing buffru není snadné škálovat na víc než 2 cache lines / cyklus**

High-Bandwidth Fetch Mechanism

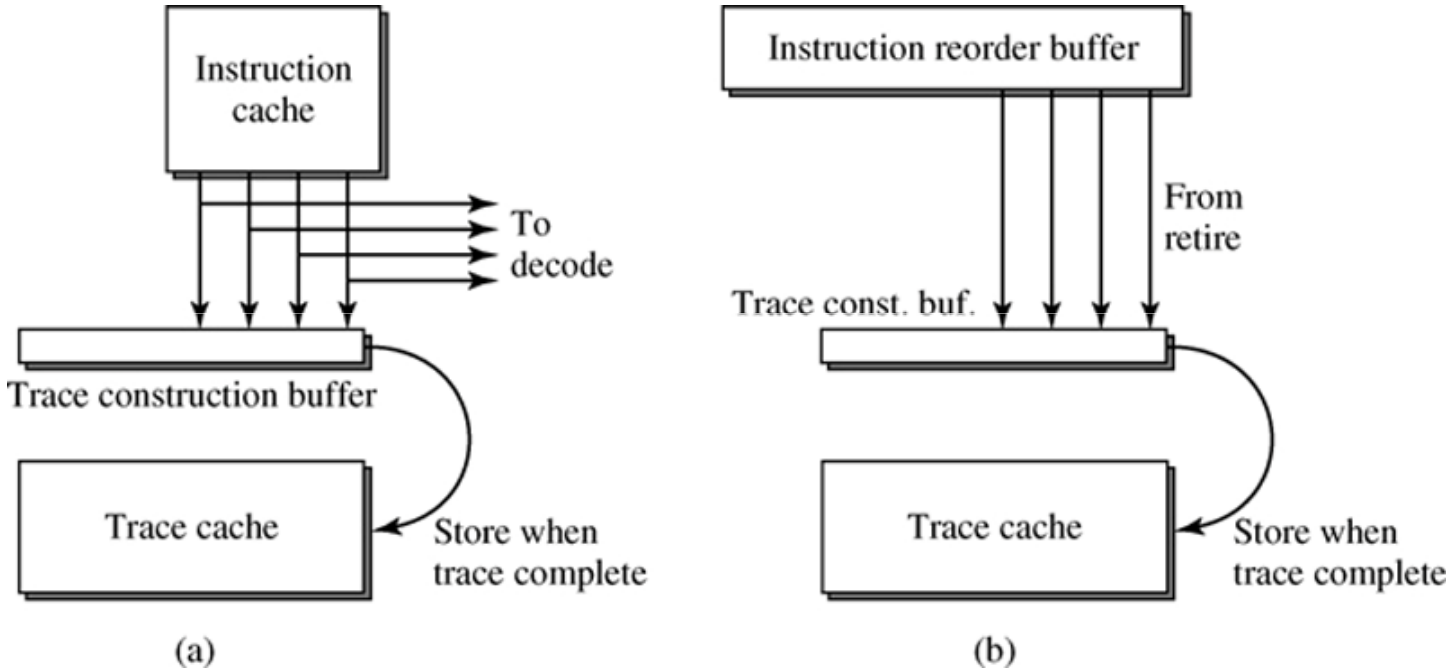
- **Trace cache –** která trasuje (pro daný skok) sekvenci následných instrukcí – a ty pak vykonává – pokud je hit v trace cache, nevybíráme z instrukční cache…
- Princip (viz obrázek):



- Základní problém: Jak tuto cache vůbec naplnit?

High-Bandwidth Fetch Mechanism – Trace cache

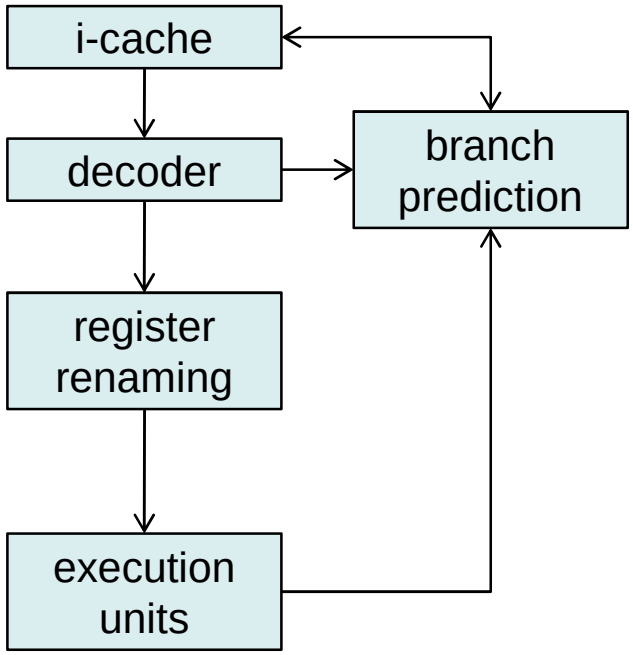
- Plnit *Trace cache* můžeme v zásedě na dvou místech (a) a (b)
- Když je Trace cnstruction buffer naplněn (což může být dáno nejenom počtem instrukcí, ale i počtem skoků v buffru – zejména případ (a)), přesováme *trace* do cache

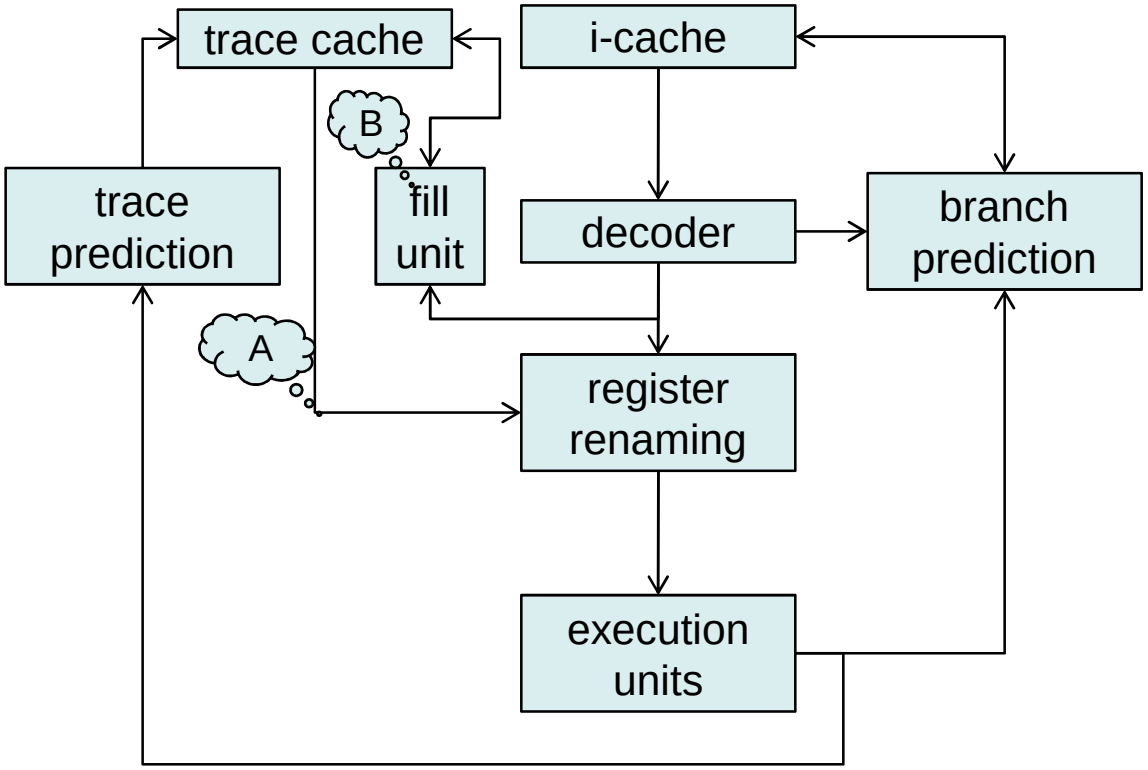


- Každý Trace v Trace cache musí rovněž obsahovat i počáteční adresy jednotlivých bloků instrukcí. Jednotka výběru instrukcí musí pak poskytovat predikované adresy několika budoucích skoků a v případě shody v Trace cache -> cache hit. (Je možná i částečná shoda, pak Trace cache poskytne část Trace)

Trace cache

Princip konvenční Instruction Fetch Unit bez použití Trace Cache:





- A: Všimněte si, že z Trace cache jdeme přímo do Register renaminig (Nemusí to být vždy tak – pokud instrukce v Trace cache nebyli plně dekódovány)
- B: Sekvence instrukcí, která plní Trace cache mohou přicházet i z reorder buffru – po Commitu – viz předchozí slide

Trace cahce - Pentium 4 processor, NetBurst microarchitecture

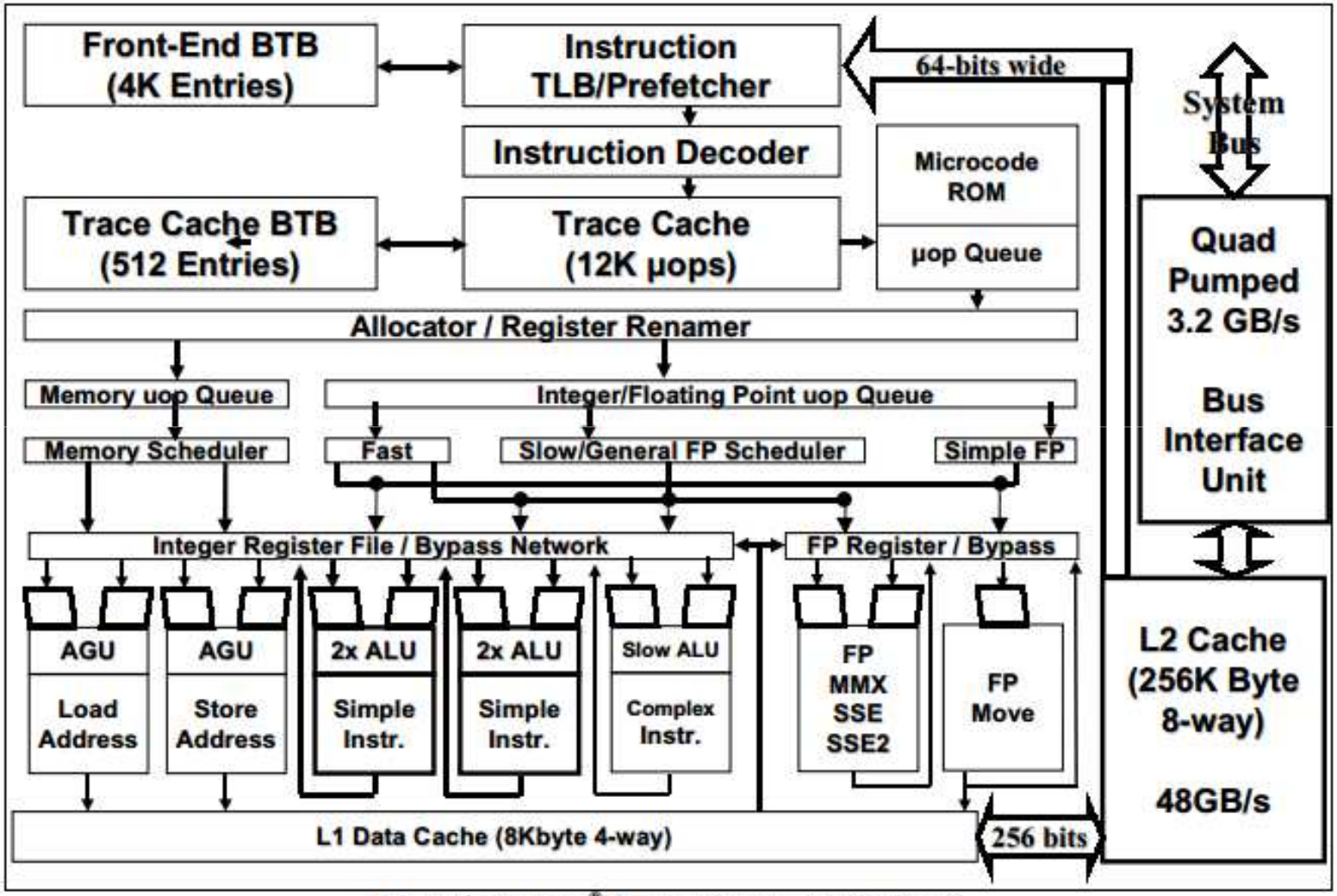


Figure 4: Pentium® 4 processor microarchitecture

Závěrem k předvýběru instrukcí…

- Zkuste si změřit:

```
for (int i = 0; i < max; i++)
    if (podmínka)
        sum++;
```

podmínka	vzor	čas
(i & 0×80000000) == 0	vždy T	
(i & 0xffffffff) == 0	vždy F	
(i & 1) == 0	střídavě TF	
(i & 3) == 0	TFFF	
(i & 2) == 0	TTFF	
(i & 4) == 0	TTTTFFFF	
(i & 8) == 0	8T 8F	
(i & 32) == 0	32T 32F	

Závěrem k předvýběru instrukcí…

- Zkuste si změřit:

```
for (int i = 0; i < max; i++){
  a=0;
  if(podmínka č.1)   a=3;
  if((i & 2) == 0)   b=10;
  if(a <= 0)  sum++;   //if(podmínka č.1)  sum++;
}
```

podmínka č.1	vzor	čas A	čas B
(i & 0×80000000) == 0	vždy T		
(i & 0xffffffff) == 0	vždy F		
(i & 1) == 0	střídavě TF		
(i & 3) == 0	TFFF		
(i & 4) == 0	4T 4F		

Typické hodnoty dnešních procesorů

	AMD FX-8150	Intel i7 2600
Instruction Decode Width	4-wide	4-wide
Single Core Peak Decode	4 instructions	4 instructions
Instruction Decode Queue	16 entry	18+ entry
Buffers	40-entry load queue 24-entry store queue	48 load and 32 store buffers
pipeline depth	18+ stages	14 stages
branch misprediction penalty	20 clock cycles for conditional and indirect branches 15 clock cycles for unconditional jumps and returns	17 cycles
reservation station	40-entry unified integer, memory scheduler; 60-entry unified floating point scheduler	36-entry centralized reservation station shared by 6 functional unit
reorder buffer	128-entry retirement queue	128-entry reorder buffer

Literatura:

- PowerPC604 RISC Microprocessor Technical Summary: http://www.freescale.com/files/32bit/doc/data_sheet/MPC604.pdf
- Karel Driesen: Accurate Indirect Branch Prediction http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.89.4874&rep=rep1&type=pdf
- Hyesoon Kim et al.: VPC Prediction: Reducing the Cost of Indirect Branches via Hardware-Based Dynamic Devirtualization http://users.ece.cmu.edu/~omutlu/pub/kim_isca07.pdf
- TseYu Yeh and Yale N Patt: Alternative Implementations of TwoLevel Adaptive Branch Prediction http://www.eecg.toronto.edu/~moshovos/ACA05/read/isca-92.2-level-adaptive.pdf
- **Shen, J.P., Lipasti, M.H.: Modern Processor Design : Fundamentals of Superscalar Processors, First Edition, New York, McGraw-Hill Inc., 2005**
- **Glenn Hinton et al.: The Microarchitecture of the Pentium 4 Processor. http://www.ecs.umass.edu/ece/koren/ece568/papers/Pentium4.pdf**