Vývoj architektur PC

Cíl přednášky

- Prezentovat vývoj architektury PC.
- Prezentovat aktuální pojmy.

První verze Pentia

- První verze Pentia:

  kmitočet procesoru - 200 MHz (dnes vyšší jak 3 GHz)

  uvádělo se 330 MIPS, srovnání s I 486DX2 – 66: 54 MIPS

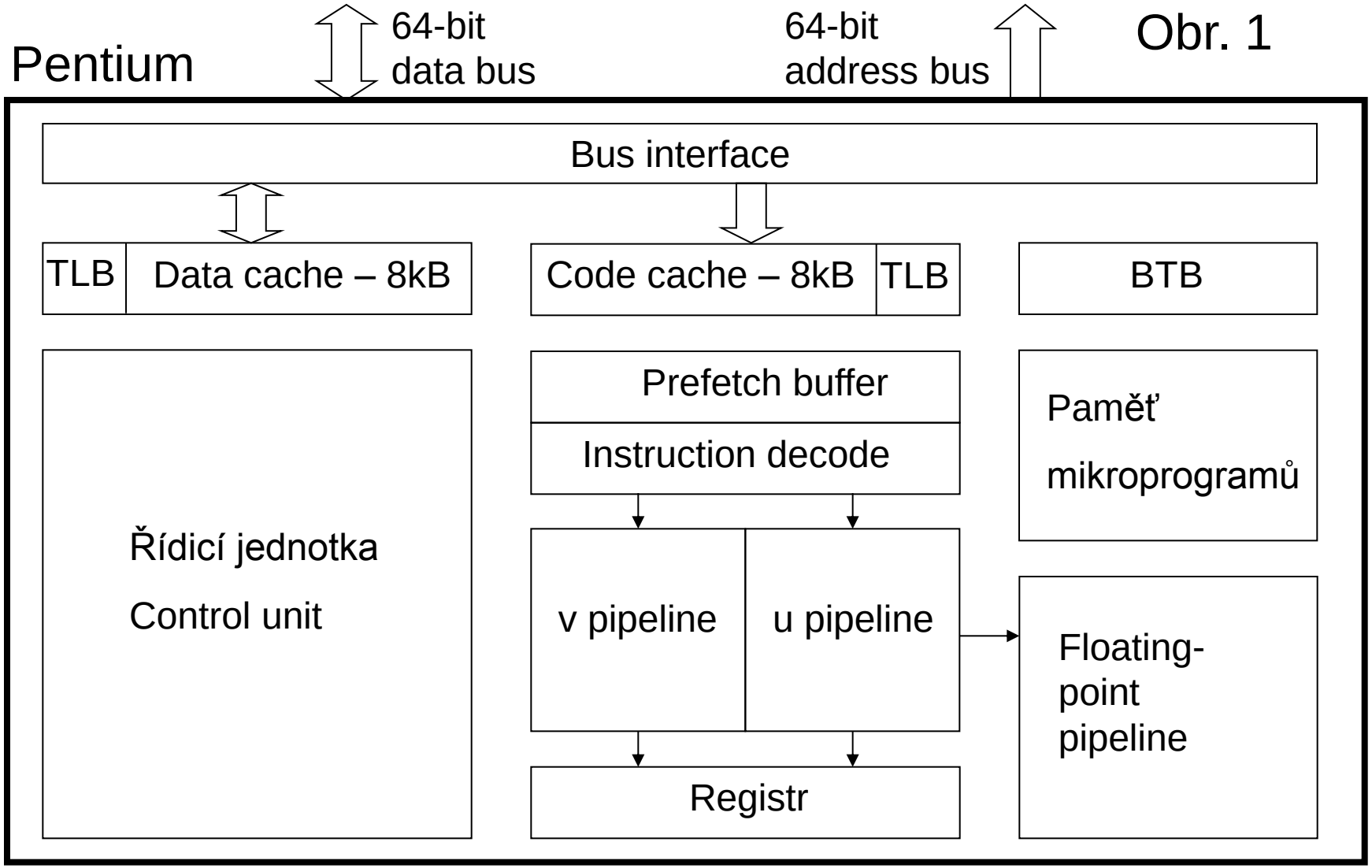
  Rozhraní: datová sběrnice - 64 bitů, adresová sběrnice: 64 bitů

  Pouzdro: PGA – Pin Grid Array – 273 vývodů, později SPGA – Staggered PGA – 296 vývodů

  První **superskalární procesor v PC** - dvě fronty instrukcí zpracovávané proudově.

První verze Pentia – blokové schéma

Obr. 1



První verze Pentia – komentář

Bus interface (řízení sběrnice) – komunikace s vnějším světem (data, adresa, řídicí signály).

- Rozdělení rychlé vyrovnávací paměti na dvě – 8 kB rychlá vyrovnávací paměť pro data (data cache), 8 kB rychlá vyrovnávací paměť pro kód (code cache) – odlišnost od předcházejících typů procesorů – sestava na bázi I80486 neměla takovou architekturu.
- Harvardská architektura – paměť dat oddělená od paměti programu.
- Každá rychlá vyrovnávací paměť má svou **TLB** (Translation Lookaside Buffer), v níž jsou uchovány posledně používané virtuální (lineární) adresy a k nim odpovídající adresy fyzické (reálné).
- Dvě fronty instrukcí – **u**, **v**.
- Technika předvídání výsledků instrukcí podmíněného skoku.

Pojem TLB

- Rychlá paměť (tzn. s krátkou vybavovací dobou) – tabulka, v níž je k dispozici část tabulky stránek.
- Je z ní zřejmý vztah mezi virtuálními a fyzickými adresami.
- Virtuální kapacita je jiná než fyzická kapacita nainstalovaná v počítači => virtuální adresa je jiná než adresa fyzické paměti.
- TLB zrychluje hledání konkrétních dat (stránek) ve fyzické paměti.
- Virtuální adresový prostor je větší než fyzický (disk – operační paměť)
- Pracuje jako asociativní paměť, tzn. adresou do TLB je virtuální adresa, na této adrese se pak získá fyzická adresa (tzv. CAM – Content Addressable Memory).
- **Jakmile se získá fyzická adresa, pak se teprve zjišťuje, zda je patřičná stránka přítomna v rychlé vyrovnávací paměti (cache) nebo jinde.**

Výpočet EAT (Effective Access Time)

- Úspěšný odkaz do TLB (TLB hit) – je zapotřebí 1 synchronizační puls (úspěšnost 99%), abychom získali fyzickou adresu

  TLB hit – stránka je v paměti
- Neúspěšný odkaz do TLB (TLB miss) – je zapotřebí 30 synchronizačních pulsů (1 % odkazů je neúspěšných)

  TLB miss – stránka není v paměti, je na disku, musí se přenést.

Výpočet EAT (Effective Access Time)

EAT = 1 x 0,99 + (1 + 30) x 0,01) = 1,3 synchronizačního pulsu

- Vysvětlení (1 + 30)

  30 – pro přenesení stránky do paměti je zapotřebí 30 synchronizačních pulsů

  1 – po přenesení do paměti se musí získat adresa fyzické stránky z TLB (je zapotřebí 1 synchronizační puls)

Co nabízí superskalární architektura?

- Možnost dokončit až dvě instrukce zároveň (v každé frontě jednu) - první typy procesorů Pentium proto dosahovaly při stejné frekvenci vyššího výkonu než procesory 80486.
- Zásady pro superskalární zpracování:
  - Následující instrukce nesmí být závislá na instrukci předcházející (následující instrukce nesmí potřebovat výsledek instrukce předcházející).
  - Obě instrukce musí být jednoduché, tj. nejsou prováděny mikroprogramově, ale hardwarově.
- Pro tyto účely jsou vypracována pro mikroprocesory Intel tzv. **párovací pravidla** (pairing rules), která specifikují, které instrukce mohou být párovány, tzn. zařazeny do front u, v tak, že splňují podmínky pro paralelní zpracování.

Párovací pravidla

- **Pravidlo 1**
  - Obě instrukce v páru musí být tzv. jednoduché.
  - Jednoduché instrukce – jsou realizovány hardwarově (tzn. nikoliv mikroprogramově).

  Důvod:

  Je vhodné, když doba trvání instrukcí je srovnatelná (přesněji srovnatelně krátká).

  Jednoduché instrukce:

  - Aritmetické nebo logické instrukce.

  Poznámka: i instrukce operující s pamětí jsou považovány za jednoduché - dejme si to do relace s informací o architekturách RISC, kde je silná snaha o omezení komunikace s pamětí při realizaci instrukce (operandy aritmeticko-logických operací nejsou uloženy v paměti) – uplatnění párovacích pravidel v architekturách RISC – efektivnější, tzn. úspěšnější).

Aritmetické nebo logické instrukce podle Pravidla 1

MOV	reg,reg/mem/imm
MOV	mem,reg/imm
ALU	reg,reg/mem/imm
ALU	mem,reg/imm
INC	reg/mem
DEC	reg/mem
POP	reg/mem
LEA	reg,mem
NOP	

Párovací pravidla - pokračování

- **Pravidlo 2**
  - Instrukce se do obou front **u**, v zavádějí současně, do fronty **u** instrukce s nižším pořadím).
  - instrukce nepodmíněného skoku, podmíněného skoku a volání funkcí mohou být zařazovány pouze do fronty **v**.

Zdůvodnění:

Získá se tak čas na předpovědění, jak bude provádění programu ovlivněno těmito instrukcemi skoku (kdyby se skokové instrukce zaváděly do fronty **u**, pak se následující instrukce zavede do fronty **v**, a přitom se nebude možná na základě výsledku skoku provádět).

Pokud se zavede instrukce podmíněného skoku do **v**, pak instrukce za ní následující není ještě zavedena do vstupní fronty, o tom se bude teprve rozhodovat předpovězením dalšího průběhu.

=> **snaha o redukci takových činností, které jsou potenciálně zbytečné, v tomto případě načtení instrukce následující za instrukcí podmíněného skoku**.

Párovací pravidla - pokračování

- **Pravidlo 3**
  - Instrukce posuvu (SAL/SAR/SHL/SHR reg/mem,1 a SAL/SAR/SHL/SHR reg/mem,imm) a rotace (RCL/RCR/ROL/ROR reg/mem,1) mohou být zařazeny pouze do fronty u.

  Zdůvodnění (domněnka):

  pro realizaci těchto činností je vybavena pouze fronta **u**.

Párovací pravidla - pokračování

- **Pravidlo 4:**
  - Operandem instrukcí ve frontě u, v nesmí být stejný registr (explicitní nebo implicitní).

Příklad (explicitní registr):

MOV eax,0001h

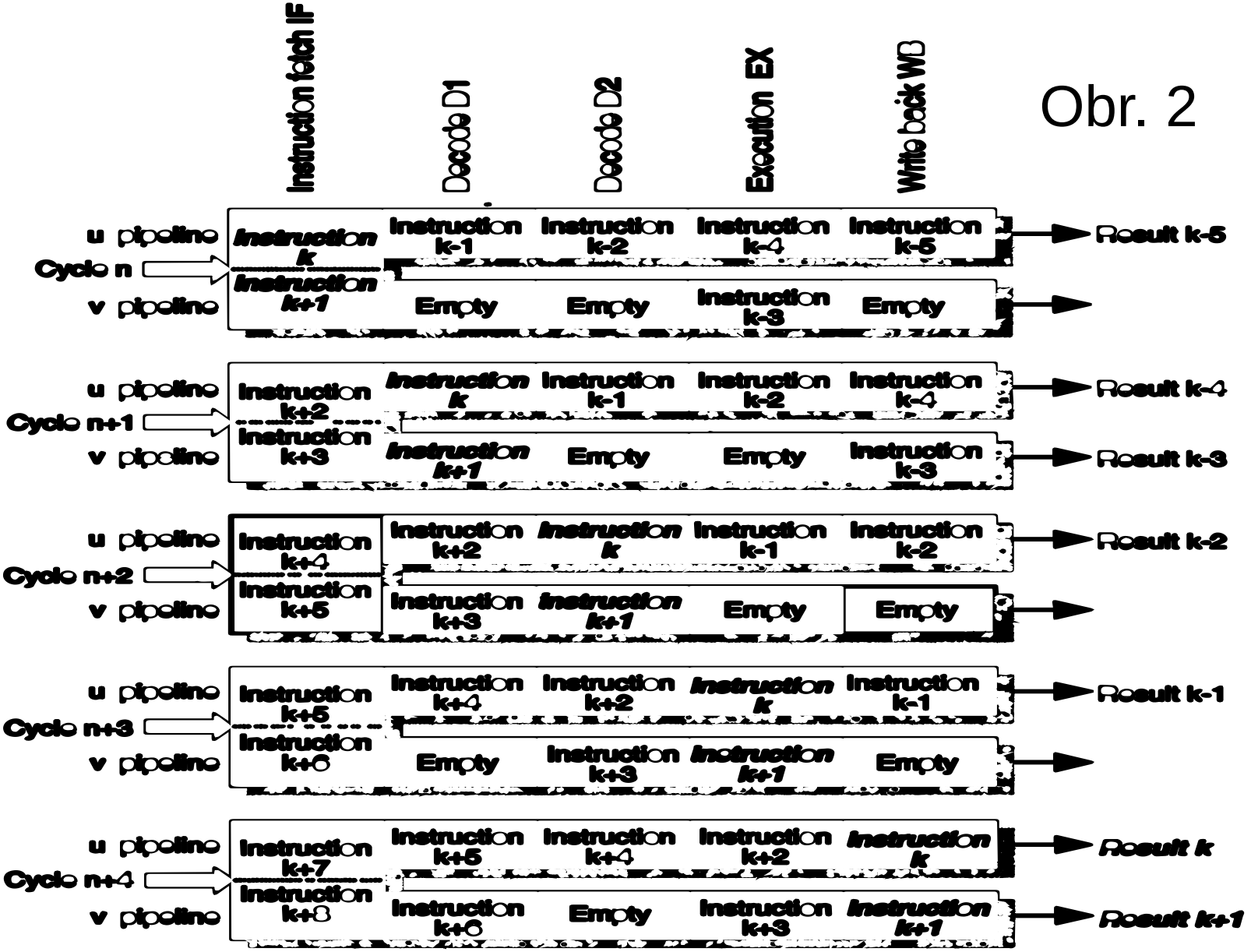
ADD ecx,eax

Důsledek: instrukce ADD ve frontě **v** se musí pozdržet, dokud není zajištěno, že instrukce MOV ve frontě **u** zapsala do eax hodnotu 0001h.

Příklad (implicitní nebo nepřímo adresovaný registr):

Příklad 1: Instrukce *IMUL mem32* ukládá 64 bitový výsledek násobení eax*mem32 do dvojice implicitních registrů edx:eax => pokud by byla taková instrukce zařazena do fronty u, pak by nemohla být do fronty v zařazena jiná instrukce operující s registry edx, eax.

Příklad 2: Instrukce zařazené do fronty u, které operují s příznaky v registru EFlag – např. instrukce *CMP* - srovnej (carry, overflow, parity, sign, zero, … jsou nastavovány podle výsledku srovnání) - nemohou být párovány např. s instrukcemi *ADC* (add with carry) nebo *SBB* (subtract with borrow) ve frontě v, poněvadž bity registru EFlag jsou při těchto instrukcích modifikovány.


Instruction fetch IF
Decode D1
Decode D2
Execution EX
Write back WB
u pipeline
Cycle n
v pipeline
Instruction k
Instruction k+1
Instruction k-1
Instruction k-2
Instruction k-4
Instruction k-5
Empty
Instruction k-3
Result k-5
Cycle n+1
Instruction k+2
Instruction k+3
Instruction k
Instruction k-1
Instruction k-2
Instruction k-4
Instruction k+1
Result k-4
Result k-3
Cycle n+2
Instruction k+4
Instruction k+5
Instruction k+2
Instruction k
Instruction k-1
Instruction k-2
Instruction k+3
Instruction k+1
Result k-2
Cycle n+3
Instruction k+5
Instruction k+6
Instruction k+4
Instruction k+2
Instruction k
Instruction k-1
Instruction k+3
Instruction k+1
Result k-1
Cycle n+4
Instruction k+7
Instruction k+8
Instruction k+5
Instruction k+4
Instruction k+2
Instruction k
Instruction k+6
Instruction k+3
Instruction k+1
Result k
Result k+1

Obr. 2

Párovací pravidla - pokračování

- Na předcházejícím obrázku jsou zobrazeny situace, kdy není možné instrukce párovat (empty – prázdný)
  - Párovány mohou být instrukce k-4 s k-3, k s k+1, k+2 s k+3, k+5 s k+6.
  - Samostatně musí být prováděny instrukce k-5, k-2, k-1, k+4.

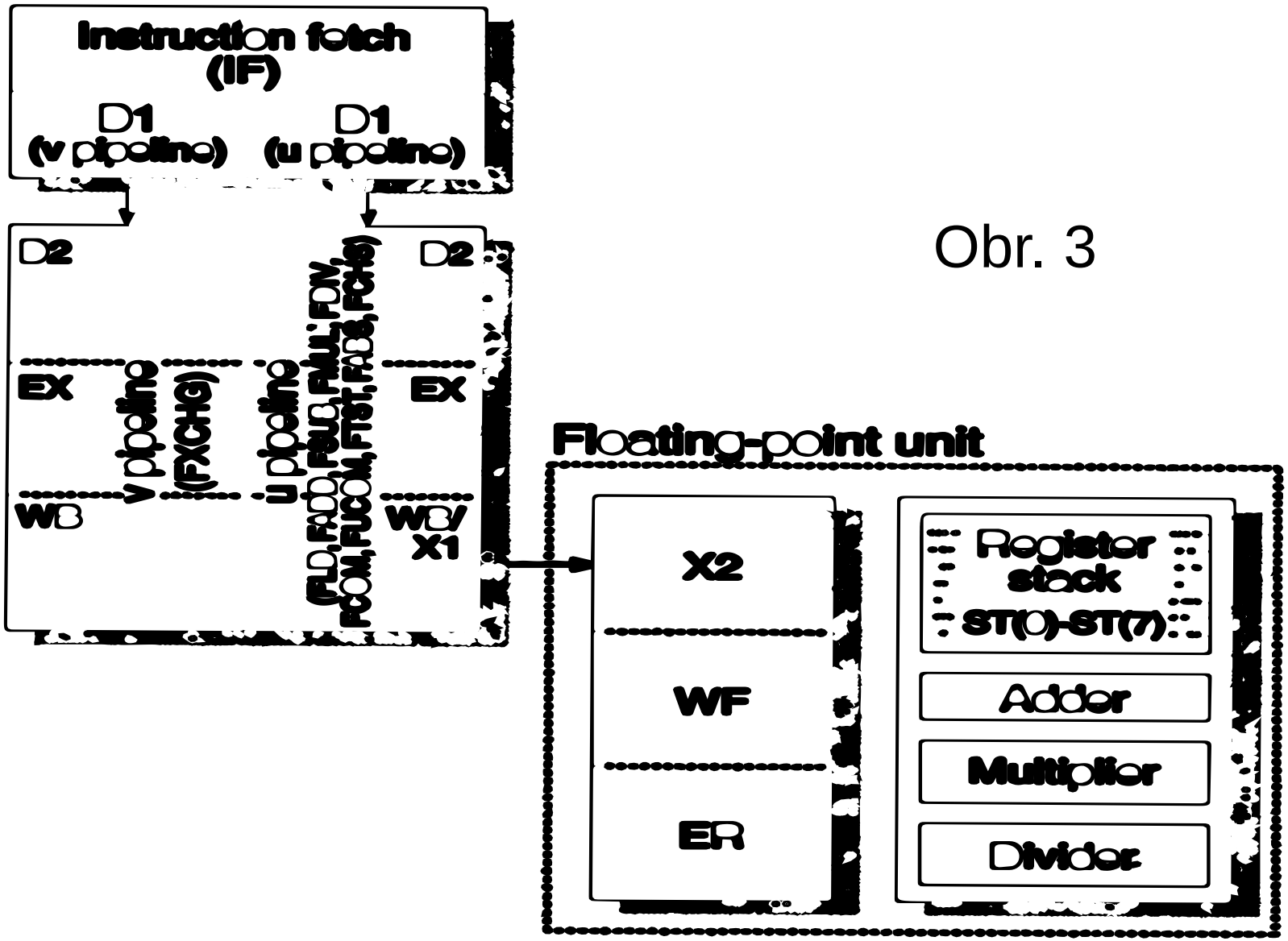
- Závěr:

  Párovací pravidla musejí být uplatněna tam, kde jsou instrukce realizovány ve více frontách.

  Zde byly prezentována některá párovací pravidla (určitě ne všechna).

Jednotka FPU

- Kromě prostředků pro zpracování instrukcí pro práci s operandy typu integer, existovala už v prvních typech Pentií jednotka FPU (nahrazující koprocesor předcházejících typů mikroprocesorů), v níž jsou realizovány instrukce zpracovávající operandy v pohyblivé řádové čárce.
- Zařazení jednotky FPU – provádění instrukce pak sestává z více stupňů.
- Před jednotkou FPU jsou instrukce řazeny do dvou front u, v, tyto fronty jsou totožné s frontami pro provádění instrukcí s operandy typu *integer*.
- **Na jednotku FPU je možné pohlížet jako na komponentu integrovanou do procesoru** (na rozdíl od architektury procesor-koprocesor v I80486) – výrazná inovace.
- Další novum – **sčítačka, dělička a násobička jsou realizovány hardwarově.**


Instruction fetch
(IF)
D1
(v pipeline)
D1
(u pipeline)
D2
D2
EX
v pipeline
(FXCHG)
u pipeline
(FLD, FADD, FSUB, FMUL, FDIV,
FCOM, FUCOM, FTST, FABS, FCHS)
EX
WB
WB/
X1
Floating-point unit
X2
WF
ER
Register
stack
ST(0)-ST(7)
Adder
Multiplier
Divider

Obr. 3

Zpracování instrukce - pokračování

- Obě fronty instrukcí **u**, **v** sestávaly z pěti stupňů podílejících se na realizaci instrukce:

  Instruction Fetch (IF),

  Decoding 1 (D1),

  Decoding 2 (D2),

  Execution (EX), obsahuje ALU,

  Write Back (WB) – zápis výsledku, přístupy do rychlé vyrovnávací paměti
- Párovací pravidlo – souvislost s FPU:

  Instrukce operující s operandy typu integer nesmí být párována s instrukcí operující s operandy v pohyblivé řádové čárce.

Zpracování instrukce - pokračování

- **Jednotka Instruction Fetch (IF)**

  Načítá z rychlé vyrovnávací paměti (cache hit) nebo z operační paměti (cache miss) 2 instrukce, každou z nich do vyrovnávací paměti velikosti 32 B.

  Pokud přečtenou instrukcí je instrukce jiná než skoková (podmíněný nebo nepodmíněný skok), předá jednotka **IF** instrukci do dekodéru **D1**.

  Pokud je právě přečtenou instrukcí instrukce skoková, pak jednotka IF zahájí spolupráci s jednotkou **BTB** (**Branch Target Buffer**) – výsledkem této komunikace je předpověď, jestli nastane skok či nikoliv.

  Stav, kdy je předpovězeno, že nastane skok, se označuje jako **taken branch** (pozor: jde pouze o předpověď, definitivně se o skoku – přechodu na jinou instrukci rozhodne, až se bude instrukce provádět).

  Stav **taken branch** - do instrukční fronty se začne načítat z místa, kam ukazuje skoková instrukce.

  Pokud se později zjistí (v jednotce **EX**), že predikce nebyla správná, pokračuje se v provádění instrukcí z fronty **u**.

  Znamená to, že technika **BTB** předpokládá predikci a **na základě predikce čtení instrukcí z místa**, kam ukazuje skok.

  Jsou i jiná řešení problému skoků ve zřetězených architekturách.

Zpracování instrukce - pokračování

- **Jednotka D1 (Decoding 1)**

  Obsahuje dva paralelně pracující dekodéry.

  Rozhoduje se zde, zda je možné instrukce k a k+1 označit za párové a zda je možné je provádět paralelně ve frontách u a v.

  Pokud ano, pak se instrukce k zapíše do fronty **u** a instrukce k+1 do fronty **v**.

  Obě instrukce pak procházejí frontami paralelně.

  Důležité**: jednotka Decoding 1 pracuje podle pravidel pro párování.**

  Pokud není možné instrukce k a k+1 zpracovávat paralelně (není splněno některé z párovacích pravidel), **zařadí se do fronty u instrukce k, načte se instrukce k+2 a posoudí se, zda je možné paralelně zpracovávat instrukce k+1 a k+2.**

Zpracování instrukce - pokračování

- **Jednotka D2 (Decoding 2)**

  Dekóduje adresy operandů a čte je.

- **Jednotka WB (Write Back)**

  Zápis výsledku (týká se pouze instrukce s operandy typu integer).

- **Jednotka X1**

  Je to první stupeň jednotky FPU.

  Jednotka provádí konverzi operandů.

  Identifikuje, zda prováděné instrukce jsou tzv. bezpečné (safe instructions) – kontroluje, zda nedojde např. k přetečení.

Zpracování instrukce - pokračování

- **Jednotka X2**

  Provádí vlastní operaci s čísly zobrazenými v pohyblivé řádové čárce.

  Provedení této operace je časově výrazně náročnější než pro celá čísla (integer).

- **Jednotka WF (FP register write stage)**

  Provádí zaokrouhlení výsledku.

- **Jednotka ER**

  Analyzuje možné chyby.

Řešení problému skoků

- 5 technik:
  Multiple streams (více toků instrukcí)
  Prefetch branch target
  Loop buffer
  Branch prediction
  Delayed branch

- Tyto techniky byly využívány v různých etapách rozvoje využívání výpočetní techniky - počínaje sálovými počítači až po dnešní výkonné počítače.

Multiple streams (více toků instrukcí)

- Využito v IBM 370/168 a IBM 3033 – 70. – 80. léta.
- Situace, kdy existuje pouze jedna fronta instrukcí – výběr jedné z možností výsledků skoku – bude se možná skákat jinam.
- Řešení – zdvojení počátečních stupňů fronty – budou k dispozici instrukce z obou adres – po rozhodnutí o pokračování programu (skok/nebude se skákat) budou k dispozici oba toky instrukcí.
- Pokud se v jednom z těchto dvou toků instrukcí objeví instrukce podmíněného skoku, vytvoří se pro tyto instrukce další tok (stream).
- Tato technika znamenala zvýšení výkonu.

Prefetch Branch Target

- Využito v počítači IBM 360/91.
- Prefetch Branch Target – přečti cíl skoku.
- Jakmile se objeví instrukce podmíněného skoku, přečte se instrukce, na niž tato instrukce ukazuje.
- V případě, že skok nastane, je tato instrukce k dispozici – přečtena z paměti, příp. částečně rozdekódována.
- Jednoduchá metoda, která má jisté (nevelké) požadavky na technické vybavení (hardware).

Loop buffer

- Loop buffer (loop –smyčka, buffer – vyrovnávací paměť) – malá a rychlá paměť řízena stupněm „instruction fetch“ (jeden ze stupňů zřetězeného zpracování instrukcí) .
- Obsahuje *n* posledních instrukcí přečtených do fronty.
- Pokud se objeví instrukce skoku, zjišťuje se, zda je instrukce, na niž ukazuje skok, přečtena do vyrovnávací paměti.
- Čím větší kapacita vyrovnávací paměti, tím větší šance, že se to stane (instrukce, na niž ukazuje skok, je přečtena).
- Výhody:
  - Instrukce, na niž se bude skákat, se nebude číst z pomalé paměti, ale bude k dispozici ve vyrovnávací paměti.
  - Je výhodné v situaci, kdy se objevují kombinace IF-THEN a IF-THEN-ELSE – v závislosti na délce vyrovnávací paměti může být vše načteno do ní – pak se instrukce načítají pouze jednou, pak se implementují z vyrovnávací paměti.

Branch prediction

- Různé techniky:

  Predict never taken

  Predict always taken

  Predict by opcode

  Taken/not taken branch

  Branch history table
- První tři techniky – statické (nezávisejí na předcházející historii).
- Poslední dvě techniky – dynamické, analyzují historii.

Statické techniky

- Predict never taken

  Předpoklad, že skok nikdy nenastane, proto se čte instrukce následující za instrukcí skoku.
- Predict always taken

  Předpoklad, že skok vždy nastane, proto se čte instrukce, na niž ukazuje instrukce skoku.
- Predict by opcode

  Předpoklad, že některé typy instrukcí podmíněného skoku způsobí skok, jiné nikoliv.

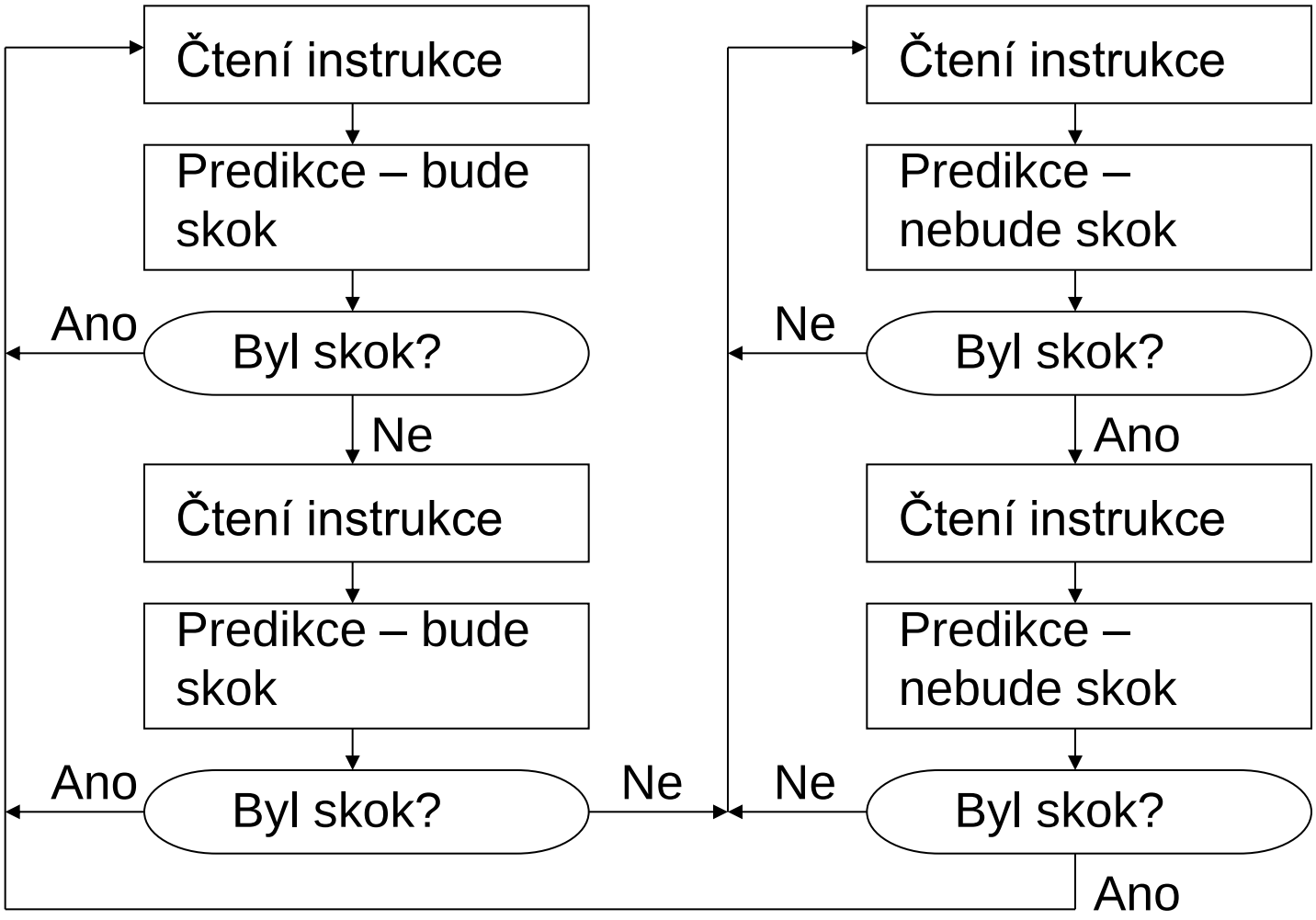
Statické techniky

- Výsledky experimentů: skoky nastávají ve více jak 50% instrukcí podmíněného skoku → pokud „cena“ obou situací skok nastal/skok nenastal je stejná → z hlediska výkonu je důležitější číst instrukce z místa, na niž ukazuje adresa v instrukci podmíněného skoku (technika „predict always taken“).
- Problém, pokud jsou informace organizovány po stránkách: může častěji nastávat „výpadek stránky“, nepříjemné z hlediska výkonu.

Dynamické techniky

- Základní myšlenka: **zvýšení přesnosti predikce pomocí záznamu historie výsledku instrukcí podmíněného skoku**.
- Příklad: každé instrukci podmíněného skoku jsou přiděleny jeden nebo dva bity (příp. více bitů u moderních technik) – tyto dva bity reflektují historii výsledků provádění této instrukce (přepínač **taken/not taken**).
- Typická situace: bity jsou přiřazeny instrukcím podmíněného skoku, které jsou umístěny v RVP.
- Odstranění instrukce z RVP, její historie je zrušena.
- Jiná možnost: informace o historii se uchovává pro instrukce, které byly prováděny za jistý časový interval předcházející aktuálnímu času.

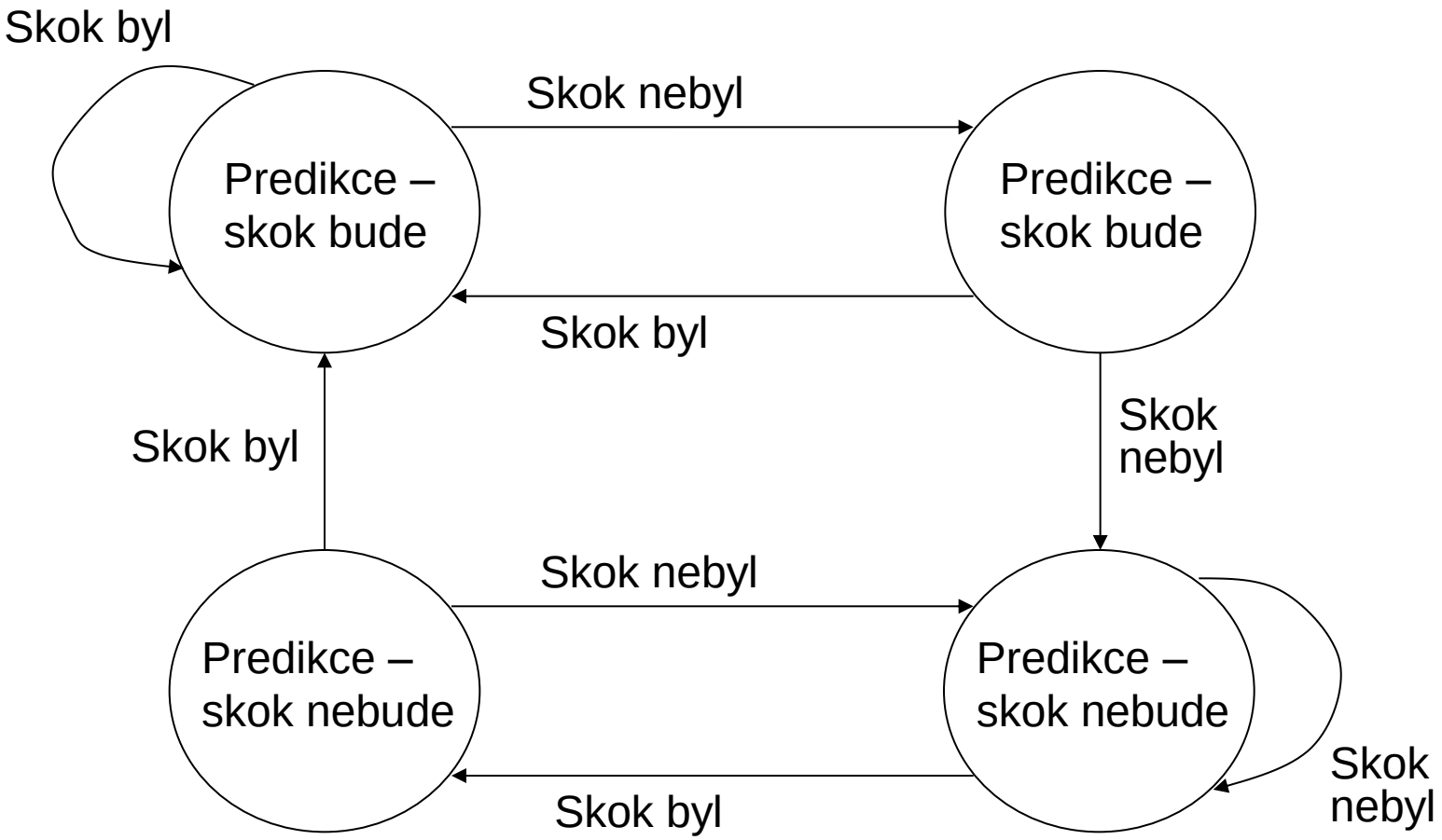
Dynamické techniky – vývojový diagram



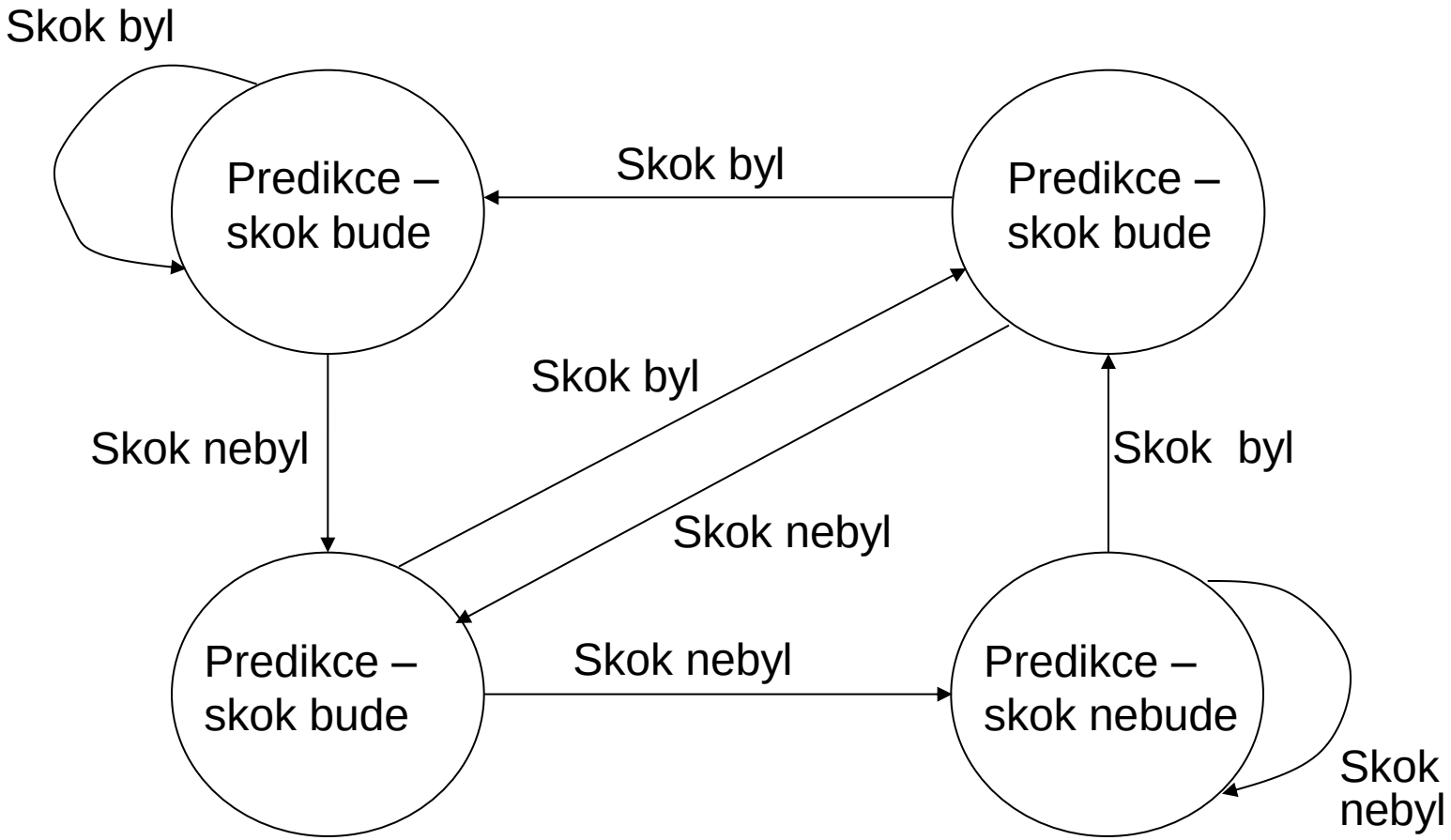
Dynamické techniky – vývojový diagram

- Začátek algoritmu – levý horní roh.
- První předpověď: skok bude – pokud ano, jde se na začátek. Pokud ne, pokračuje se druhým stupněm – při výskytu instrukce opět predikce, že skok bude.
- Pokud ano, vracíme se na začátek. Pokud ne, přechod do druhé větve, kde se bude předpovídat, že skok nebude.
- Návrat do větve „predikce bude“, pokud bude predikce „skok nebude“ dvakrát neúspěšná – přechod na začátek algoritmu (levý horní roh).
- Závěr: pokud se predikce dvakrát nepodaří, přechod do větve s opačnou predikcí.

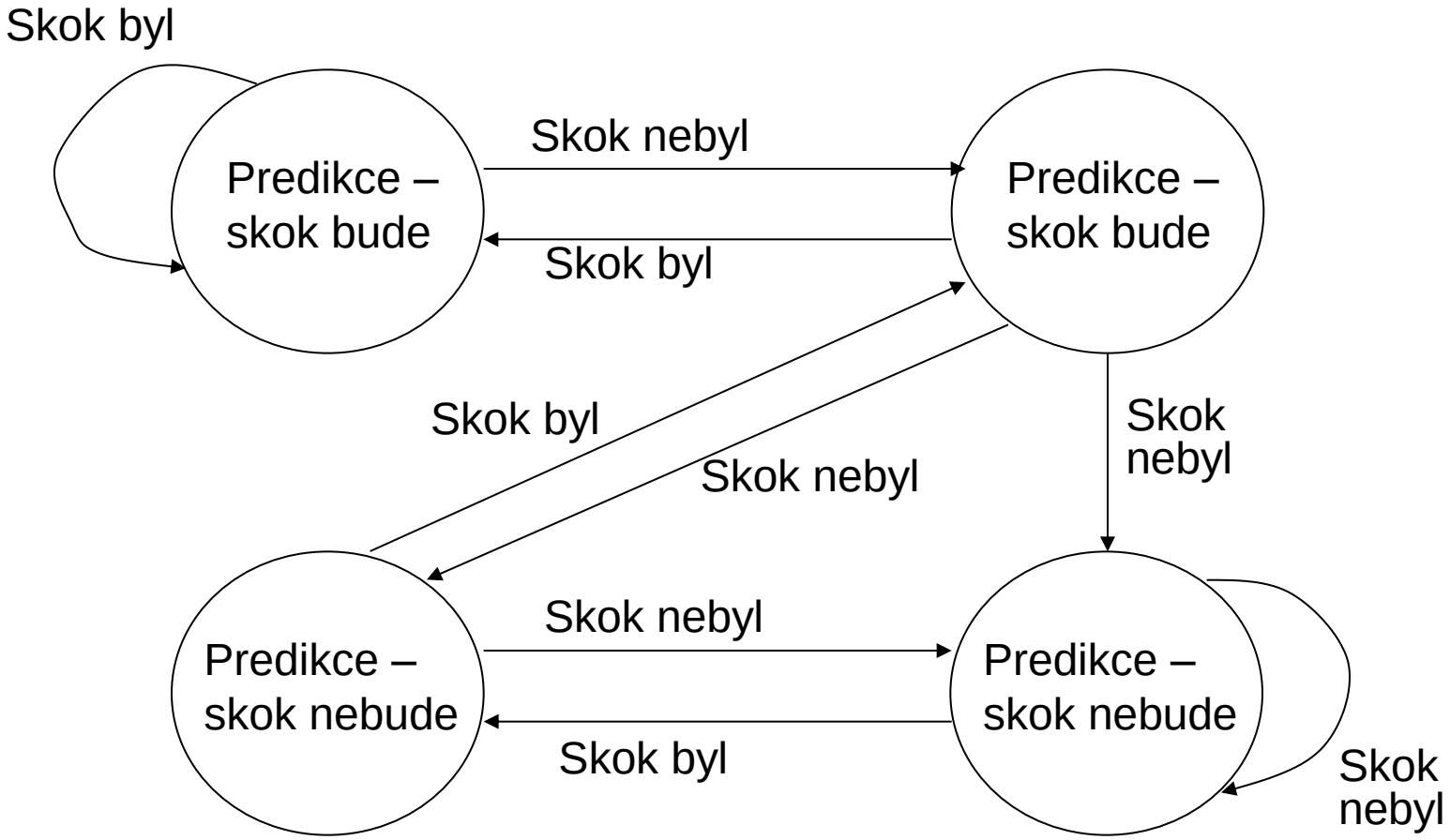
Dynamické techniky - automat



Dynamické techniky – jiná verze



Dynamické techniky – jiná verze



Dynamické techniky

- Nevýhody metod založených na „historii“:

  Provede se predikce „bude skok“ (v okamžiku načtení instrukce podmíněného skoku do fronty), pak se musí čekat, až se instrukce podmíněného skoku ve fázi realizace instrukce rozdekóduje – do té doby není možné přečíst instrukci, na niž se bude skákat.
- Řešení: je nutné uložit více informace – využití BTB (Branch Target Buffer).
- Princip - každá položka v BTB obsahuje tři pole:

  adresu instrukce podmíněného skoku,
  bity informující o výsledcích předcházejících instrukcí skoku,
  adresu instrukce, na niž se bude skákat.
- Jiná možnost, jak by mělo vypadat třetí pole: celá instrukce.

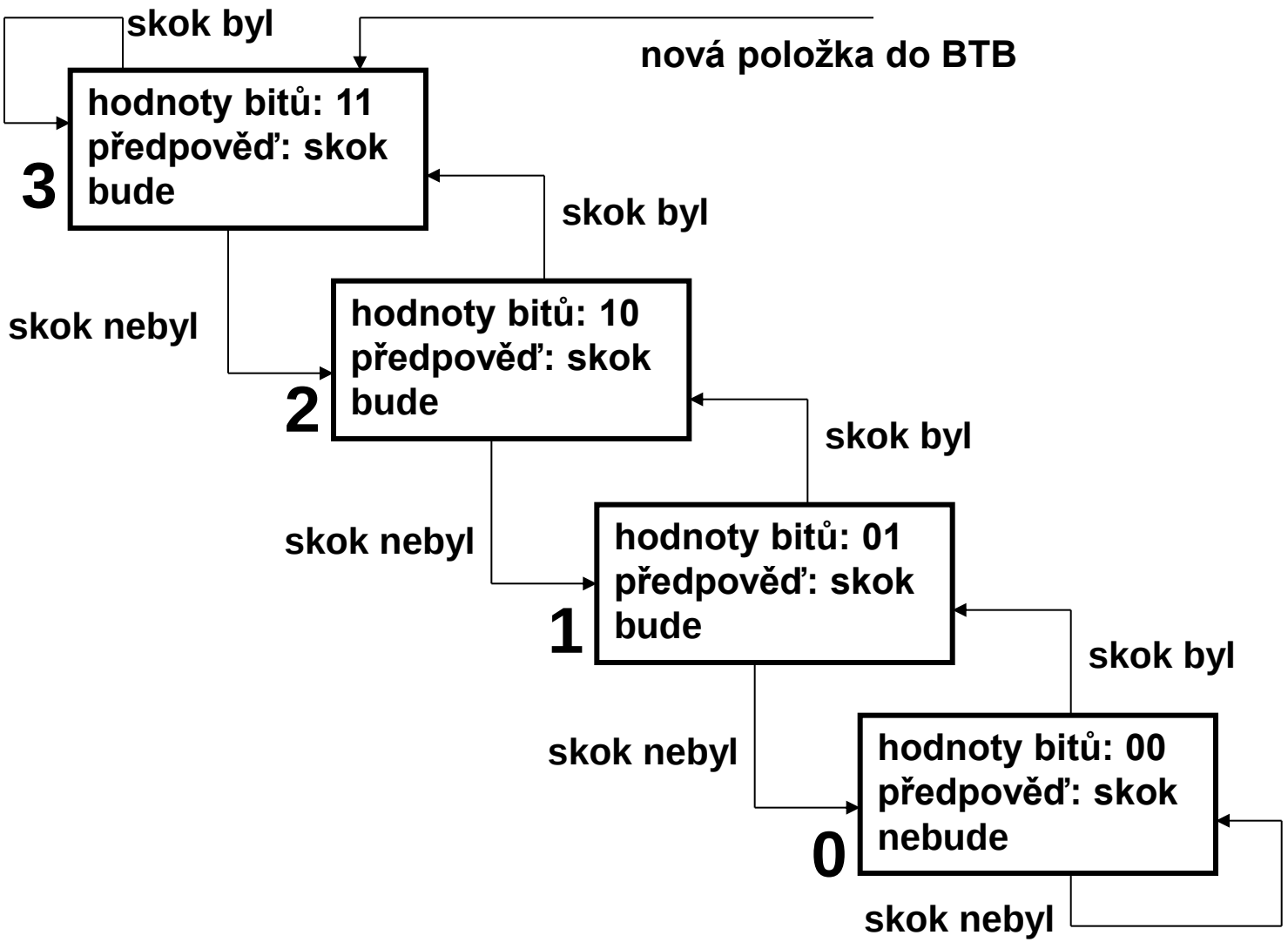
  Výhoda: instrukce je nachystána na dekódování a realizaci, nikoliv pouze její adresa.

  Nevýhoda: je potřeba větší kapacita BTB.

Branch Target Buffer a jeho využití

- **Účel:** řešení problémů souvisejících s instrukcemi skoku v procesorech založených na zřetězeném zpracování instrukcí.
- Řešení: již první procesory Intel Pentium měly zabudováno tzv. dynamické předvídání skoků (Dynamic Branch Prediction) - technika pro odhadnutí, zda při dalším průchodu skoková instrukce skok způsobí nebo ne.
- Dynamické předvídání se řídí tím, jak se program při předcházejících průchodech konkrétním bodem choval.
- BTB (Branch Target/Trace Buffer) - v ní jsou uchovány instrukce, které způsobily skok, spolu s dvoubitovou informací, jež určuje dosavadní chování těchto instrukcí.
- O každé instrukci skoku je uložena informace o tom, jak se tato instrukce skoku „chovala“ (tzn. skok nastal/nenastal).
- Podle hodnot těchto bitů je také stanovena předpověď, zda výsledkem provedení instrukce bude skok či ne.

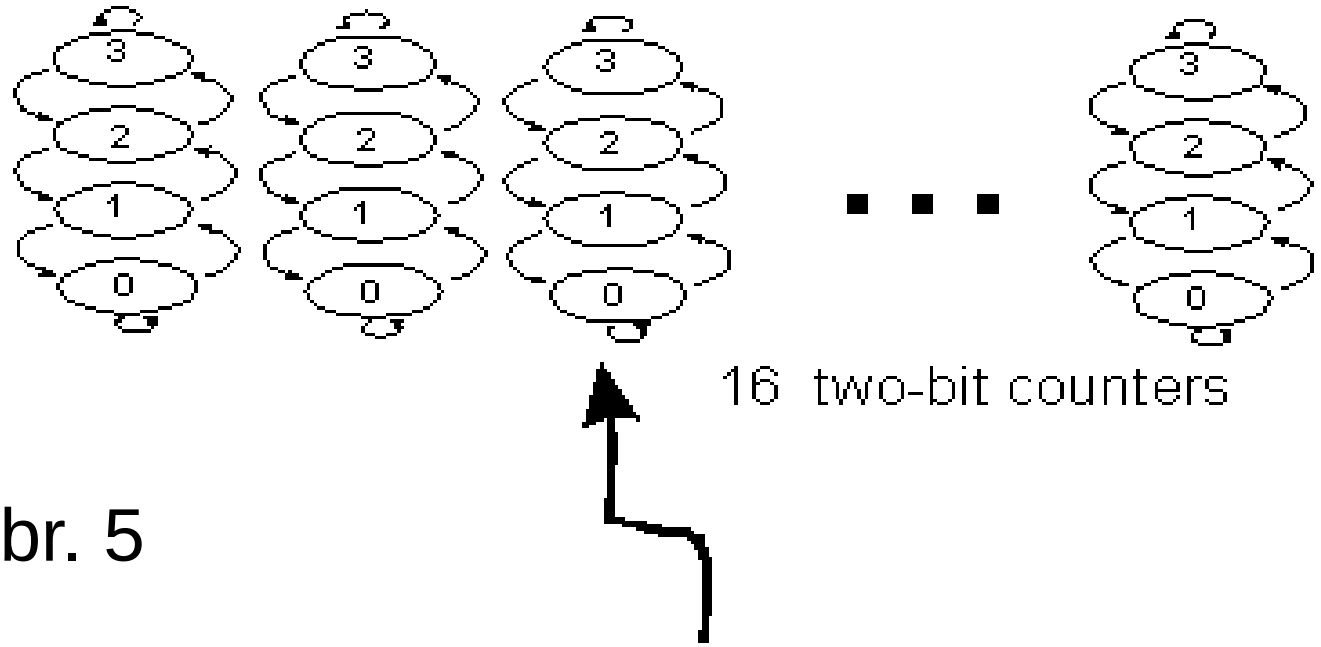
Branch Target Buffer (BTB) - schema



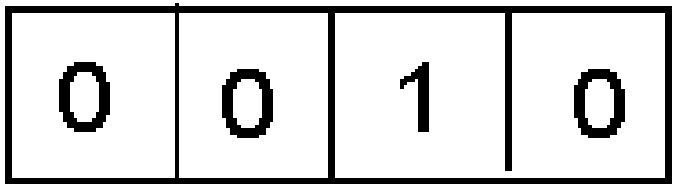
Branch Target Buffer

- Princip funkce:
  1. Instrukce, která způsobila skok, je uložena do BTB spolu se dvěma bity, jejichž hodnoty jsou rovny 1 (stav 3).
  2. Tyto hodnoty při příštím průchodu programu přes tuto instrukci signalizují předpověď, že skok bude.
  3. Pokud skok skutečně byl, hodnoty bitů zůstanou nezmodifikovány.
  4. Pokud byla předpověď mylná a skok nebyl, jsou bity nastaveny na hodnotu 10 (stav 2), která opět signalizuje, že skok bude.
  5. Podle toho, zda skok skutečně následuje nebo ne, jsou pak příslušným způsobem bity modifikovány (viz obrázek) a jejich hodnota signalizuje předpověď skoku.
  6. Pokud při prvním výskytu instrukce skok nenastane, nastaví se hodnoty bitů na 00 (stav 0).
  7. Při dalším výskytu této instrukce se předpokládá, že skok nebude – pokud se to splní, zůstane nastaven současný stav 0.
  8. Pokud skok nastane, hodnota bitů se změní na 01 (stav 1) a předpověď se změní na „skok bude“.
- Je to vlastně samoučicí se mechanismus.

Dvoúrovňové předvídání výsledků skoků



Obr. 5



History in 4 bit shift register

Dvoúrovňové předvídání výsledků skoků

- Úroveň 2 – šestnáct dvoubitových čítačů podle obr. 5.
- Úroveň 1 – čtyřbitový posuvný registr, v něm je uložena historie posledních čtyř událostí (instrukcí skoku). Tyto čtyři bity jsou ukazatelem na jeden ze 16 dvoubitových čítačů.
- Příklad: v čítači je hodnota 00, pokud při předcházejících čtyřech instrukcích skoku nedošlo ke skoku, naopak 11, pokud se čtyřikrát skákalo.
- Dvoúrovňové předvídání výsledků – jemnější informace o předcházejících výsledcích skoku.
- Tento princip je využíván dnes v procesorech Intel.