Pokročilé Architektury Procesorů

Superpipelinové a Superskalární Procesory

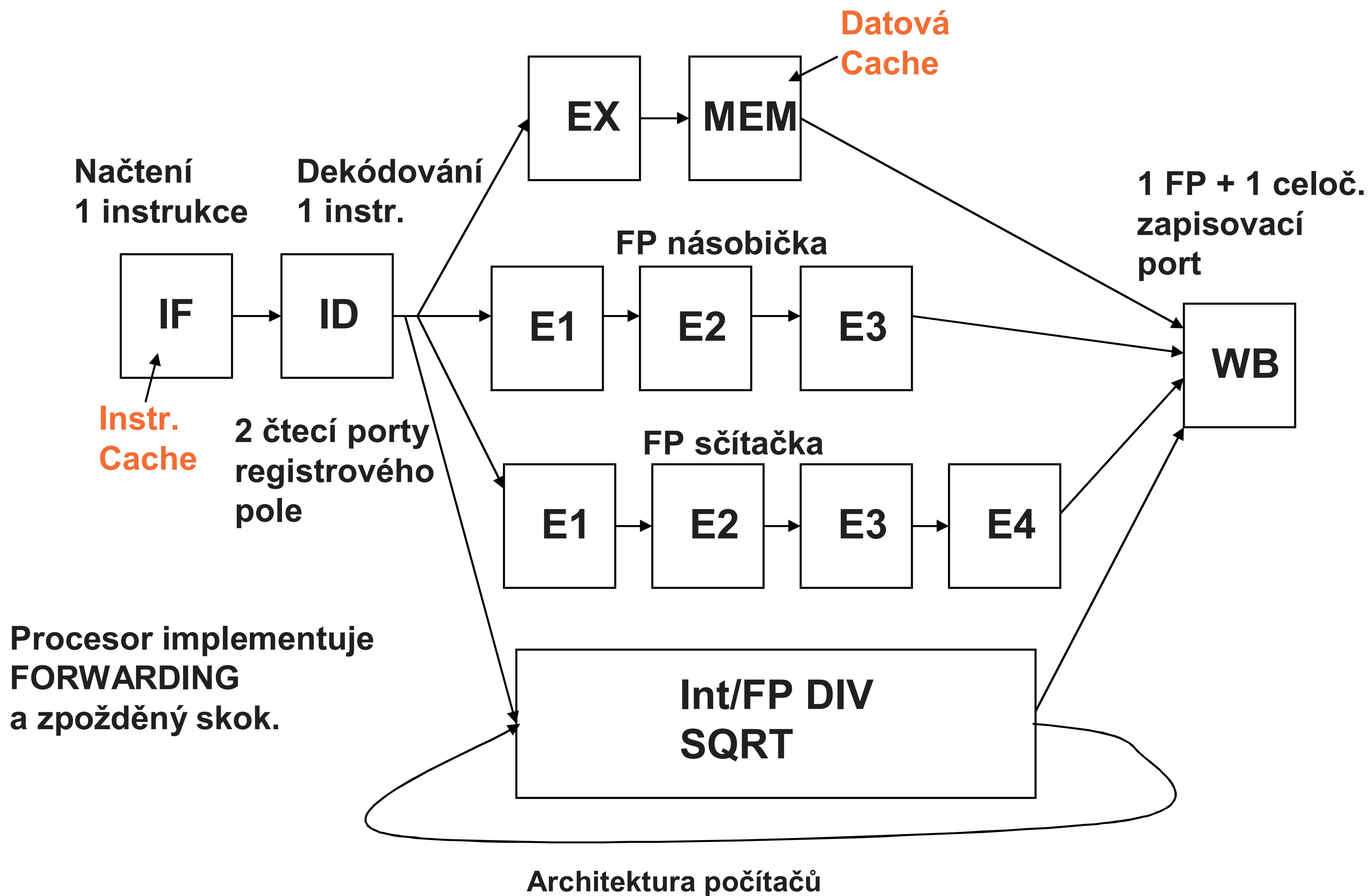
Procesory VLIW

Ing. Miloš Bečvář

Osnova přednášky

- Shrnutí vlastností skalárního proudově pracujícího procesoru
- Zvyšování výkonnosti a technologické trendy
- Pojem paralelismu na úrovni instrukcí (ILP)
- Superpipeline procesory
- Superskalární procesory
- Procesory VLIW

Proudově pracující skalární DLX



Skalární DLX – Vykonávání Programu s Ideální Cache

Takt #0

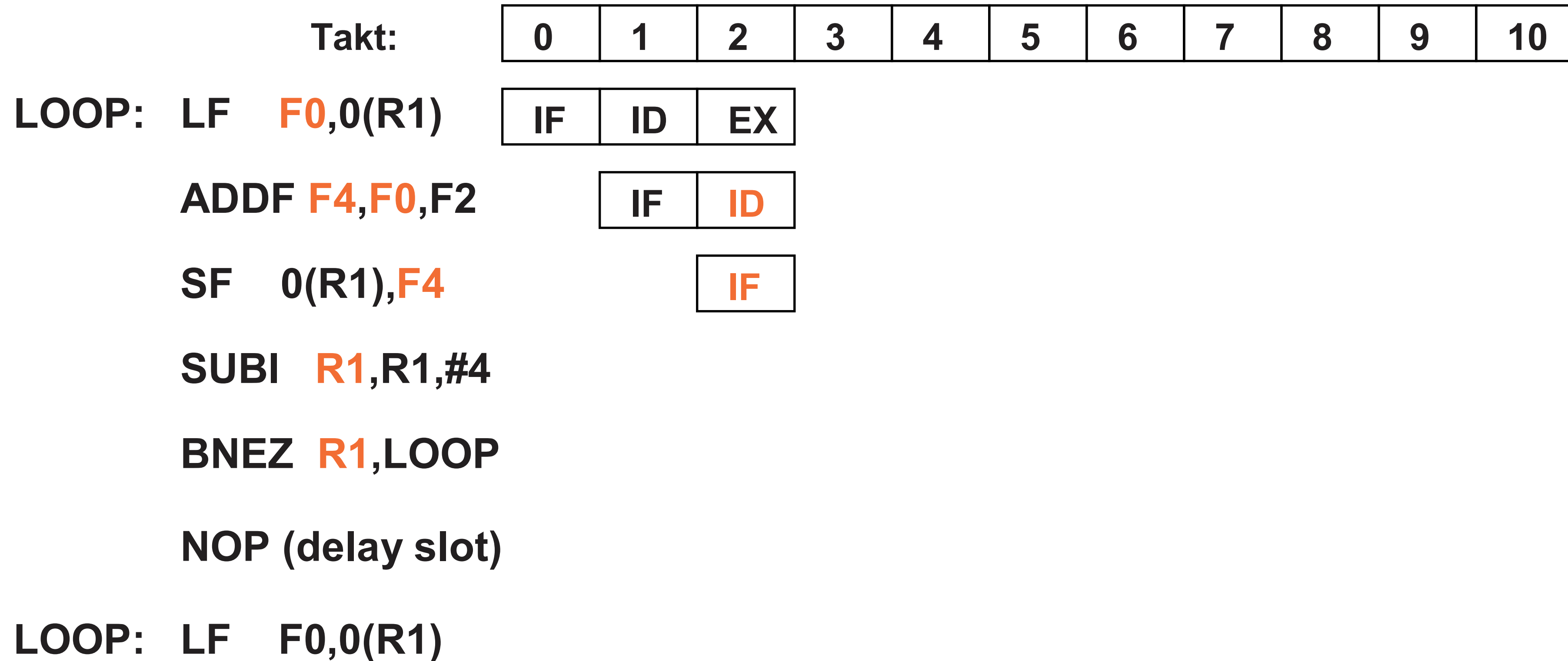
Takt:	0	1	2	3	4	5	6	7	8	9	10
LOOP: LF F0,0(R1)	IF										
ADDF F4,F0,F2											
SF 0(R1),F4											
SUBI R1,R1,#4											
BNEZ R1,LOOP											
NOP (delay slot)											
LOOP: LF F0,0(R1)											

Skalární DLX – Vykonávání Programu s Ideální Cache

Takt #1

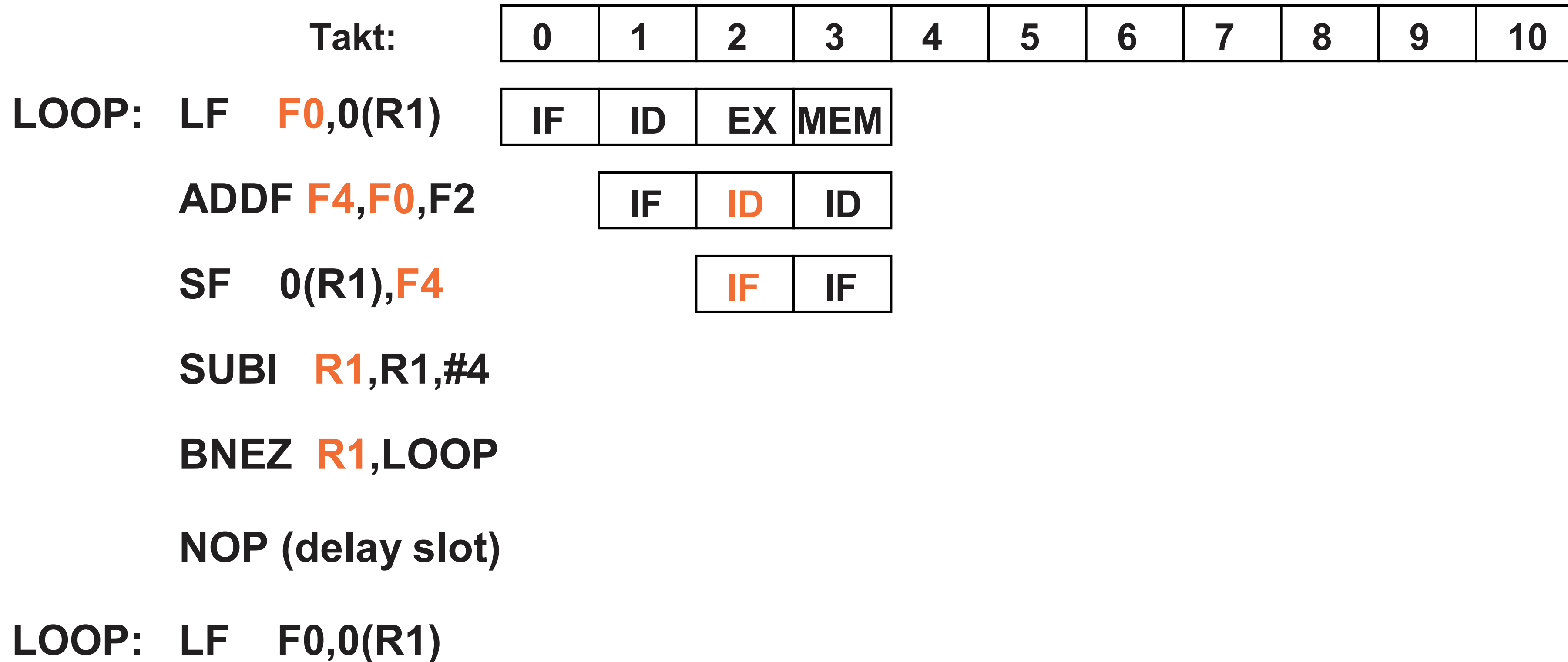
		Takt:	0	1	2	3	4	5	6	7	8	9	10
LOOP:	LF	F0,0(R1)	IF	ID									
	ADDF	F4,F0,F2		IF									
	SF	0(R1),F4											
	SUBI	R1,R1,#4											
	BNEZ	R1,LOOP											
	NOP (delay slot)												
LOOP:	LF	F0,0(R1)											

Skalární DLX – Vykonávání Programu s Ideální Cache



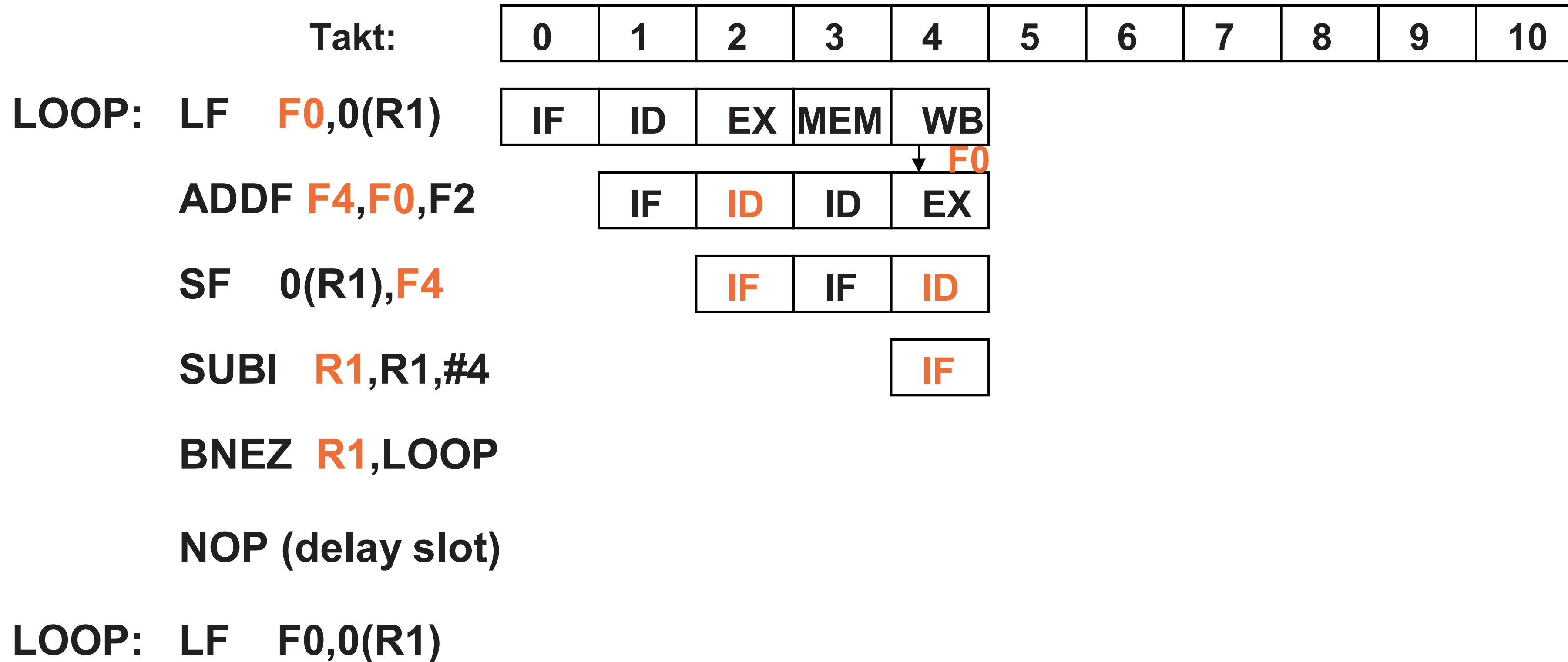
Instrukce ADDF je pozastavena v ID kvůli RAW hazardu přes F0 vůči LF
SF je také blokováno v IF.

Skalární DLX – Vykonávání Programu s Ideální Cache



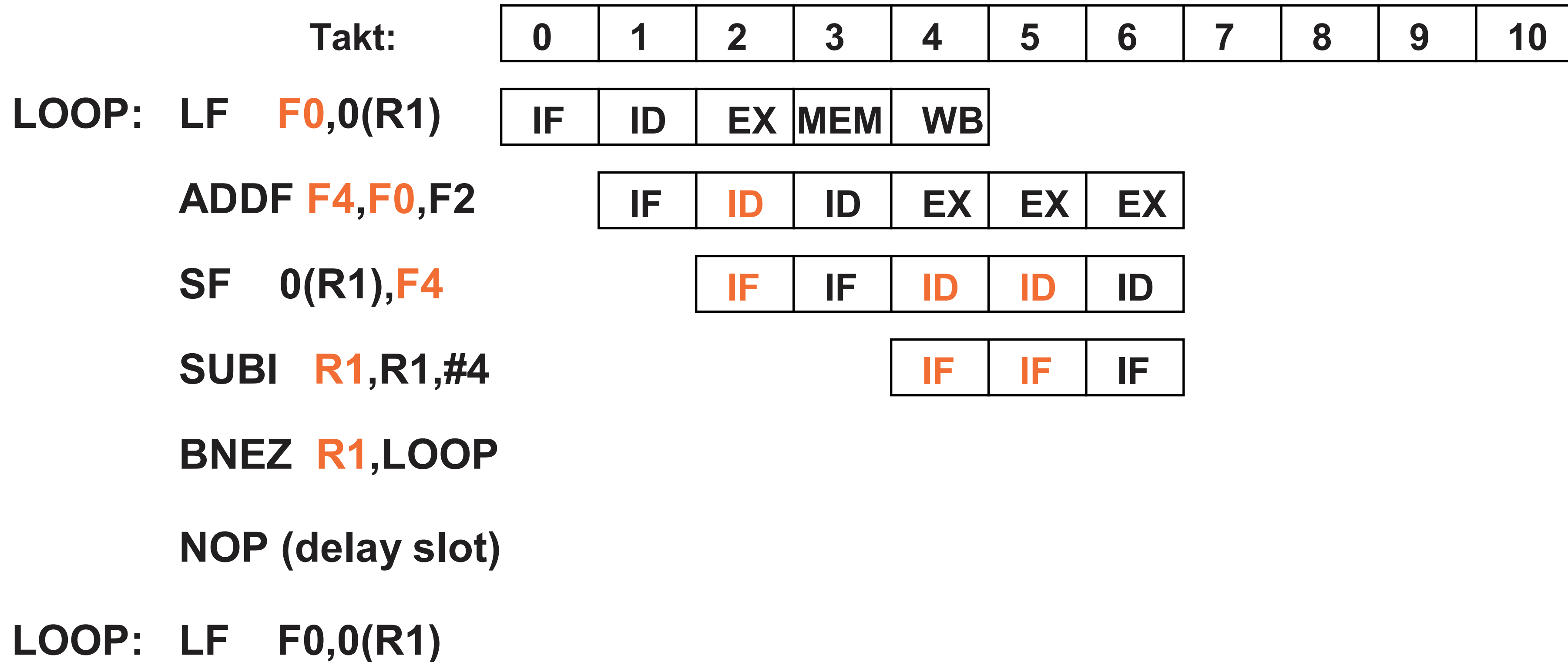
ADDF je uvolněna do EX, kde v dalším taktu získá F0 přes WB->EX forwarding.

Skalární DLX – Vykonávání Programu s Ideální Cache



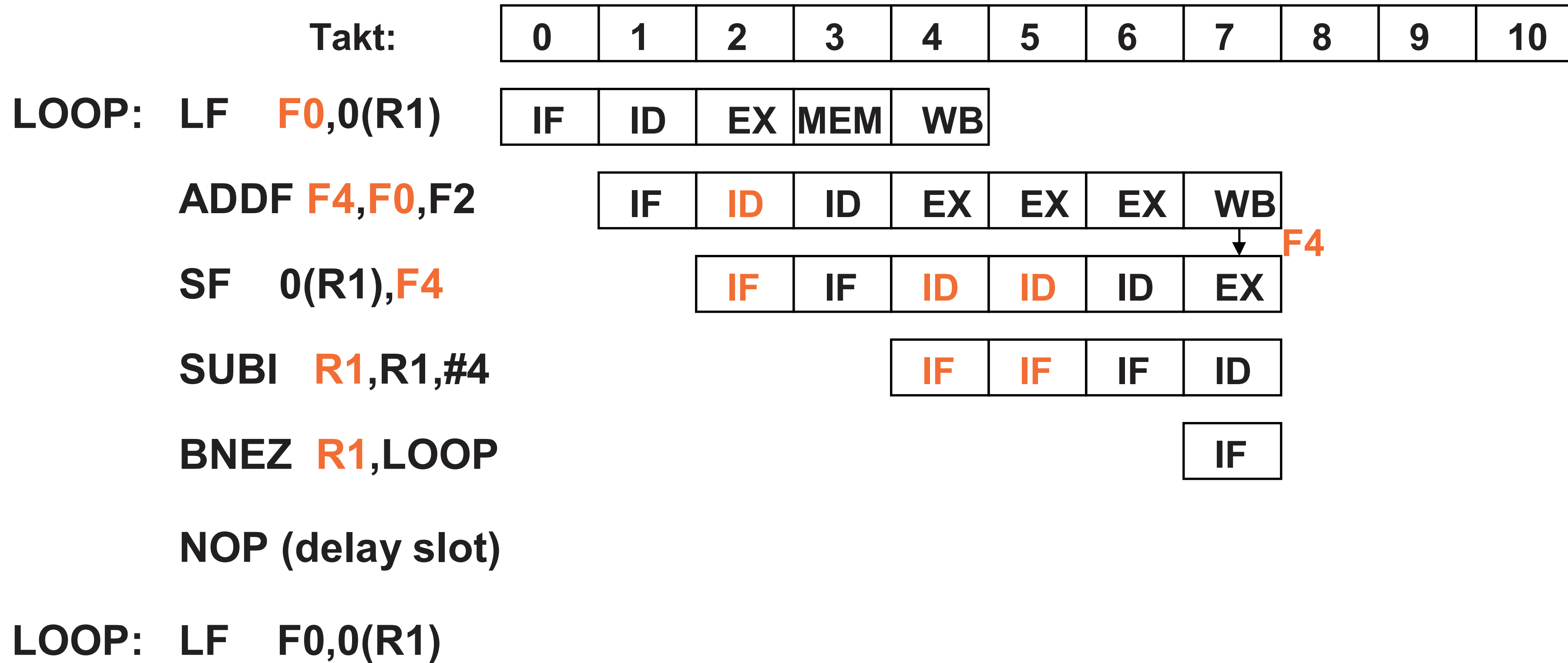
F0 je forwardováno instrukci ADDF,
SF je pozastaveno v ID kvůli RAW hazardu přes F4 vůči ADDF,
SUBI je také blokováno v IF. (NELZE PŘEDBÍHAT !)

Skalární DLX – Vykonávání Programu s Ideální Cache



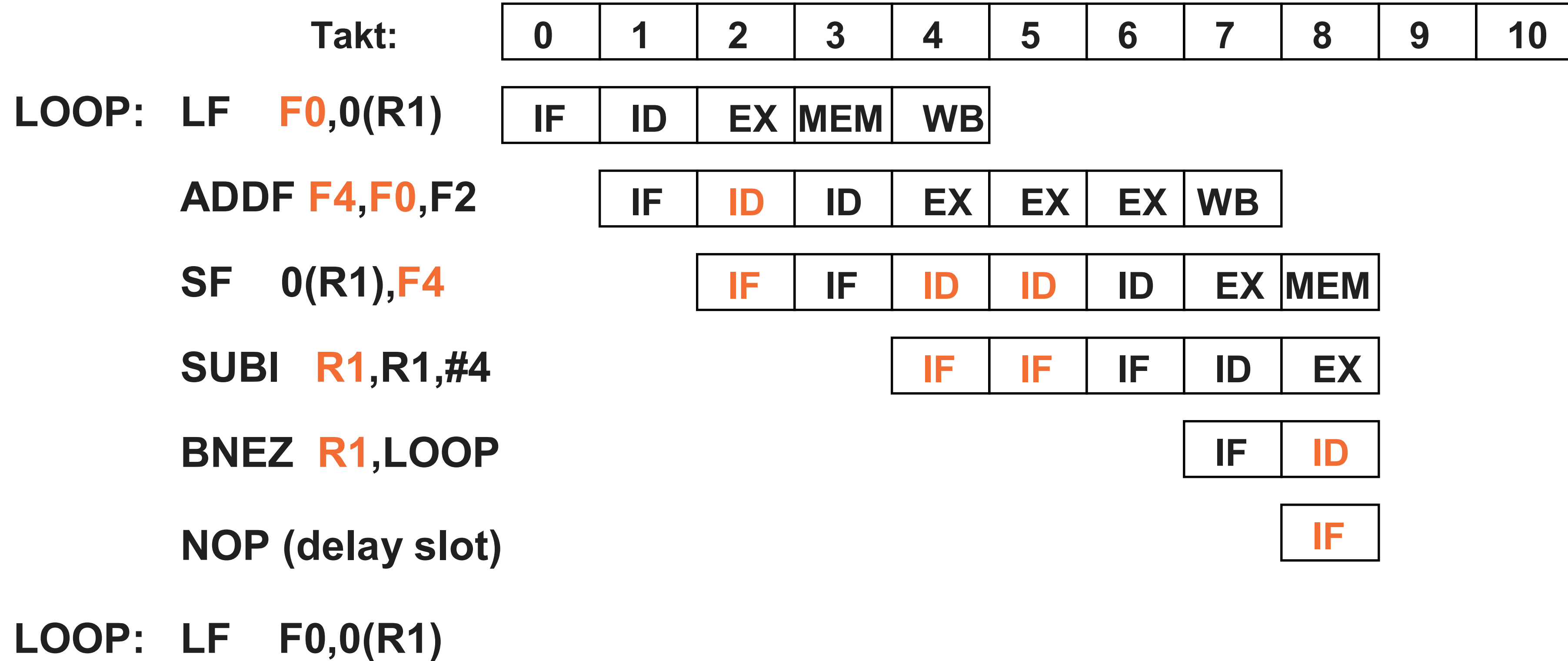
Po 2 taktech čekání, SF pokračuje a je načteno i SUBI.

Skalární DLX – Vykonávání Programu s Ideální Cache



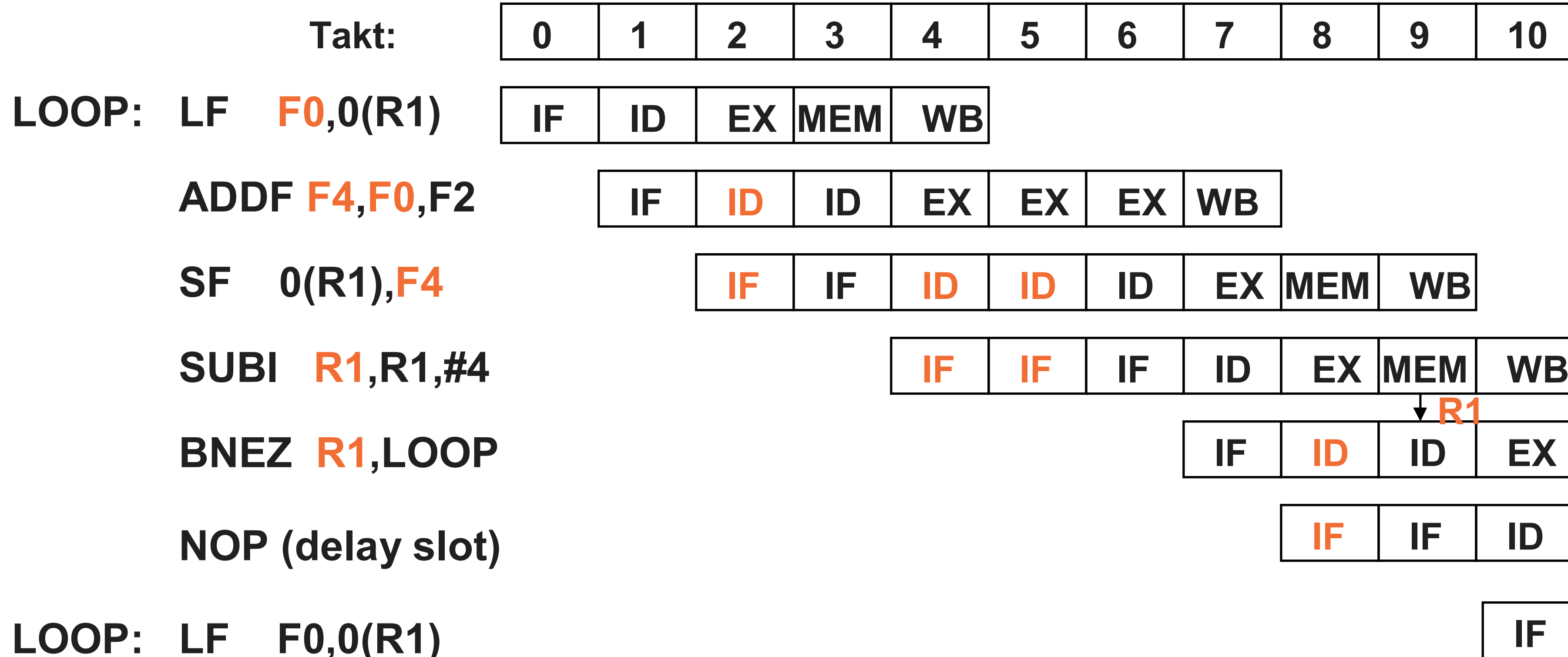
F4 je forwardováno, SUBI úspěšně načte R1 a pokračuje.

Skalární DLX – Vykonávání Programu s Ideální Cache



BNEZ blokován v ID dokud není provedena dekrementace R1

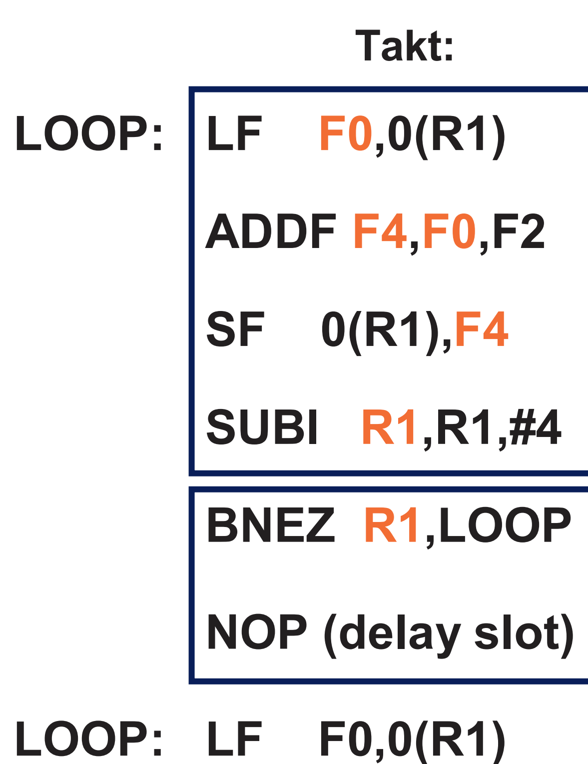
Skalární DLX – Vykonávání Programu s Ideální Cache



V 9. taktu proveden skok na základě dedikovaného MEM->ID forwardingu R1.

Celkový výpočet taktů na výpočet jedné iterace je 10, z toho 4 pozastavení (STALLS) + 1 NOP => CPI=10/5 = 2.

Jak se změní výpočet v případě reálné ICache a DCache ?



ICache blok = 16B = 4 instrukce

Uvažujme ICache a DCache bloky o velikosti 16B

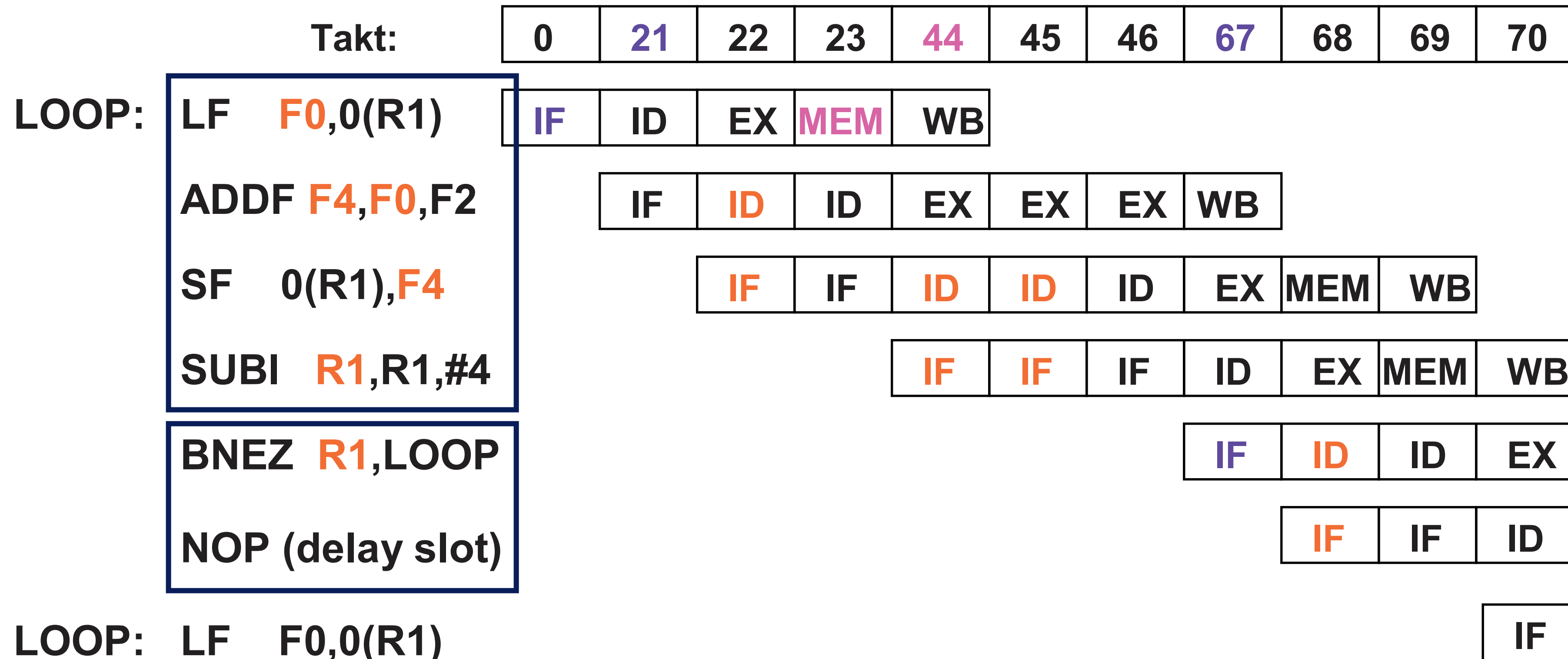
Do bloku se vejdou 4 instrukce a 4 položky pole.

Výpadek ICache – první načtení LF a první načtení BNEZ (předpoklad, že LF je zarovnán na adrese dělitelné 16) – pouze první iterace cyklu
Výpadek DCache – každá 4. iterace cyklu výpadek u instrukce LF (další 3 iterace najdou data v DCache v důsledku prostorové lokality)

Efekt výpadku ICache – pozastavení v IF o délce Miss Penalty (MP)
Efekt výpadku DCache – pozastavení v MEM o délce MP

Uvažujme MP=20 taktů (Fclk je v řádu stovek MHz)

První iterace cyklu – 2 x výpadek ICache + 1 x DCache



LOOP: LF F0,0(R1)

1.Iterace cyklu trvá10 (vnitřních) taktů + 60 paměťových taktů,
2.-4. iterace trvá 10 taktů, 5.iterace 10+20 taktů, …

100 iterací cyklu trvá 100x10 „vnitřních taktů“ + 25x20 pam. taktů (DCache) + 40 paměťových taktů (ICache) =1540 taktů

Výkonnostní Rovnice Procesoru – Včetně Paměť. Systému

$$T_{CPU} = IC * CPI * T_{clk}$$

$$T_{CPU} = IC * (CPI_{pipe_ideal} + Stalls\ Per\ Instr) * T_{clk}$$

RAW-stalls + Control-stalls + Struct.-stalls + Memory-stalls

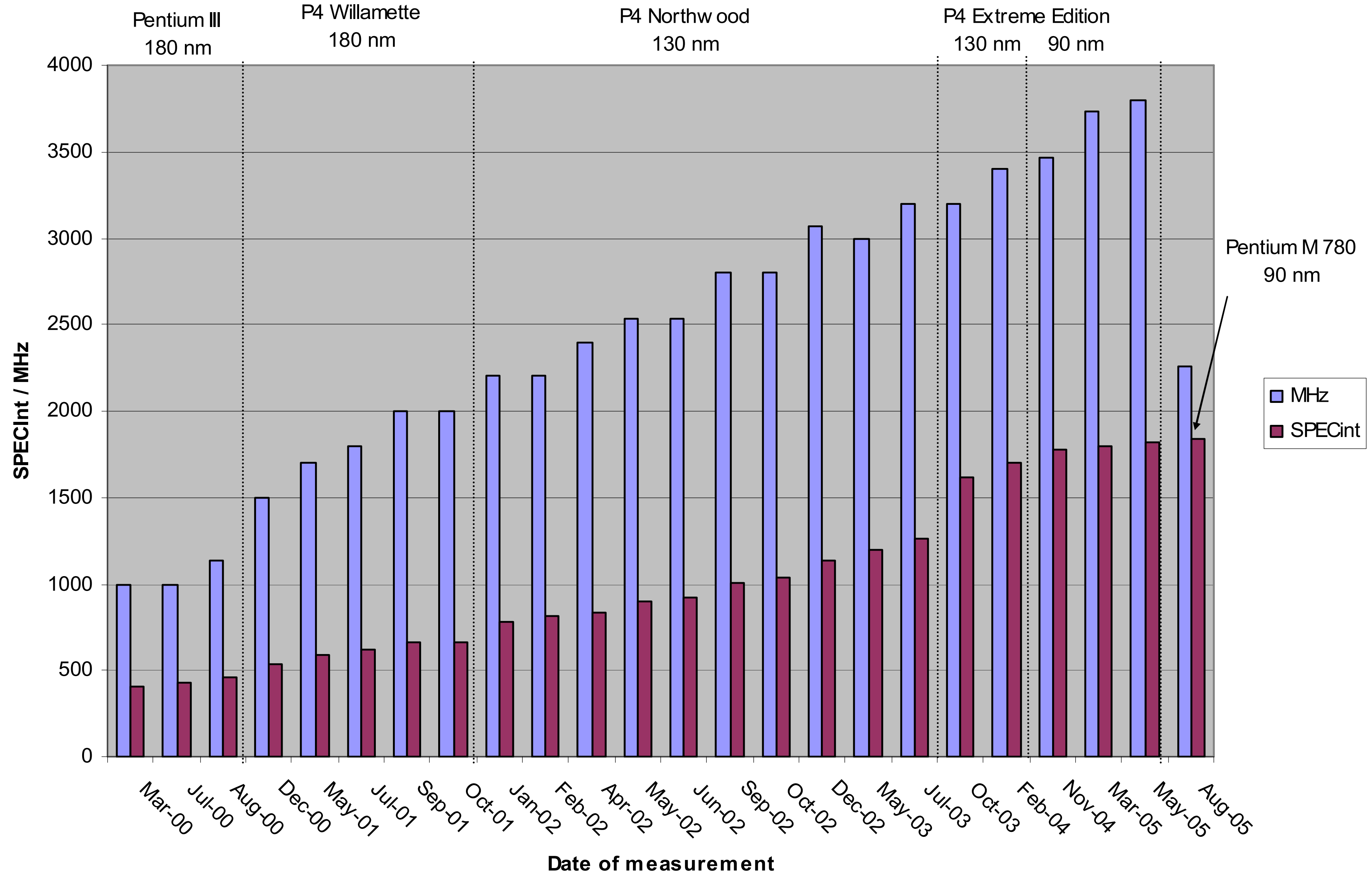
MAPI * MR * MP

Zvýšení výkonnosti = minimalizace faktorů v této rovnici.
Bohužel tyto faktory nejsou nezávislé.

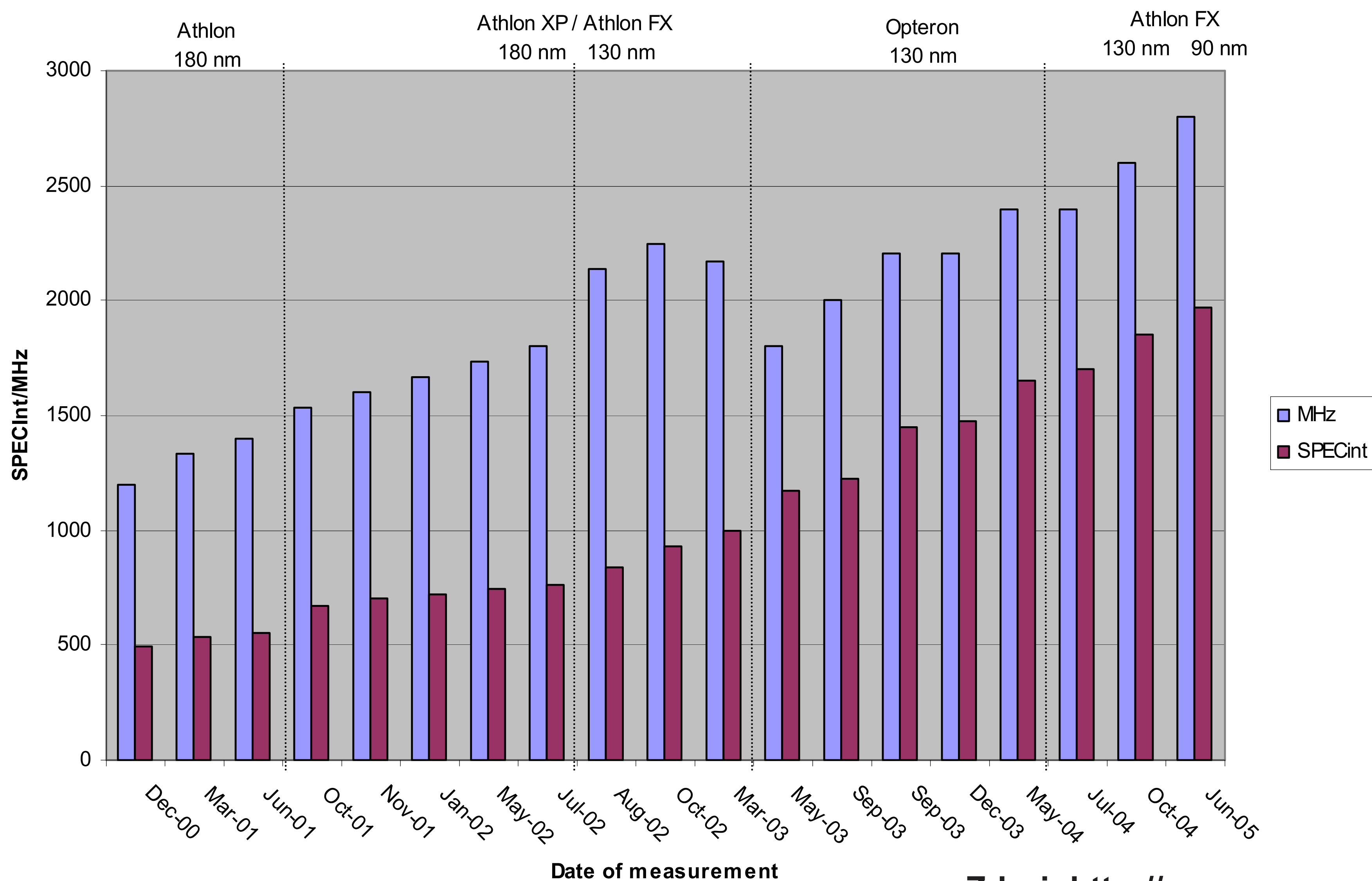
Např. zvýšení Fclk (snížení Tclk) vede na nárůst MP (při stejném pam. Systému) , cache s větší kapacitou minimalizuje MR, ale může prodloužit HT a tím i Tclk …

Osnova přednášky

- Shrnutí vlastností skalárního proudově pracujícího procesoru
- **Zvyšování výkonnosti a technologické trendy**
- Pojem paralelismu na úrovni instrukcí (ILP)
- Superpipeline procesory
- Superskalární procesory
- Procesory VLIW



Zdroj: http://spec.org



Zdroj: http://spec.org

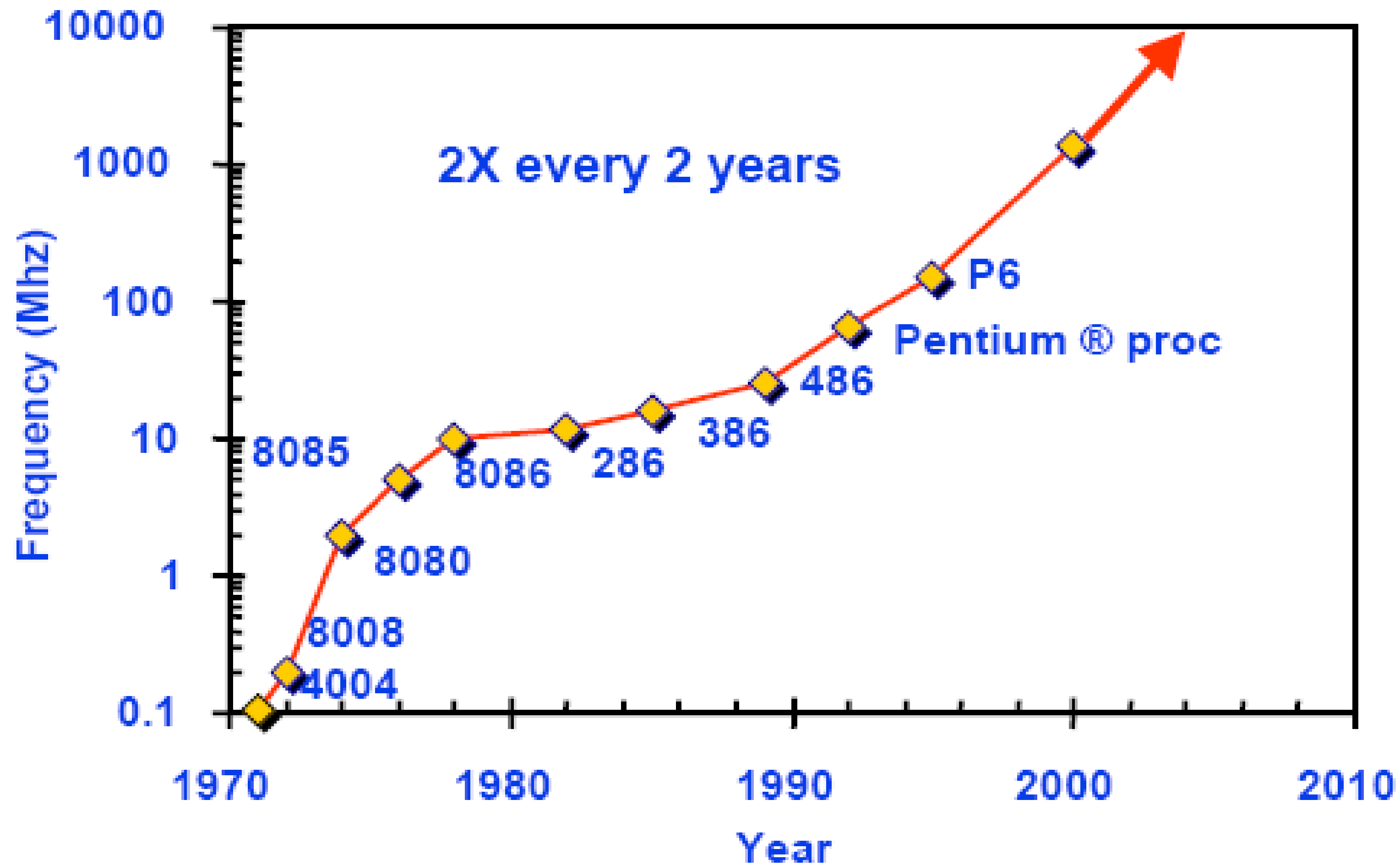
Trendy ve výkonnosti procesorů 2000-2005

	Clock Frequency Trend		SPECint performance Trend	
	Intel	AMD	Intel	AMD
2000-2002	2,47	1,87	2,17	1,88
2001-2003	1,60	1,62	2,43	1,84
2003-2005	1,26	1,56	1,51	1,68

V 90. letech platilo:
Hodinová frekvence i výkonnost procesoru se zdvojnásobí za každou technologickou generaci (jedna technologická generace = 2 roky)
Období do roku 2002 tuto tendenci potvrzuje.

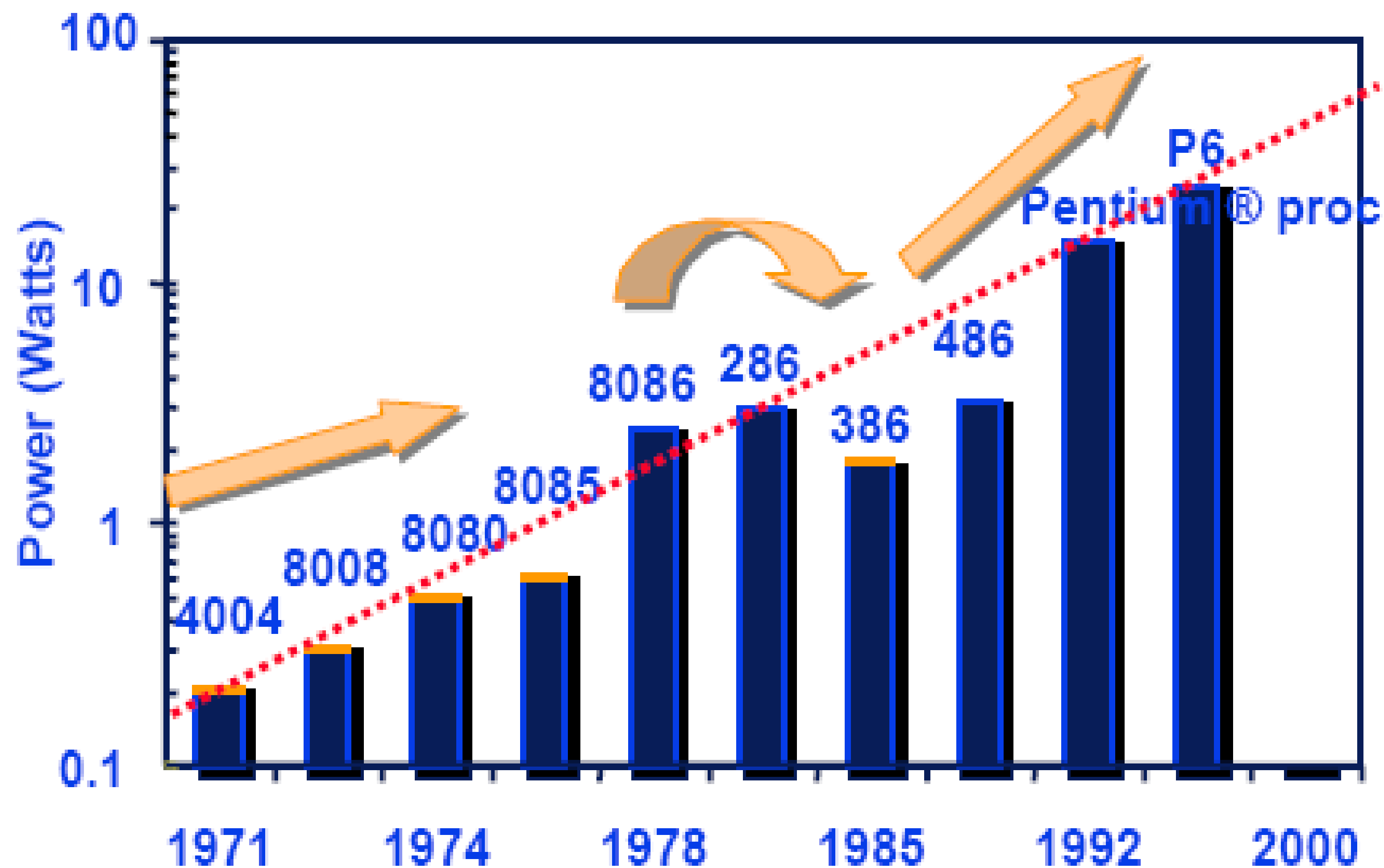
Všimněme si, že jak nárůst hodinové frekvence tak výkonnosti zpomaluje v posledních dvou technologických generacích.

Hodinová frekvence se zdvojnásobí každé 2 roky



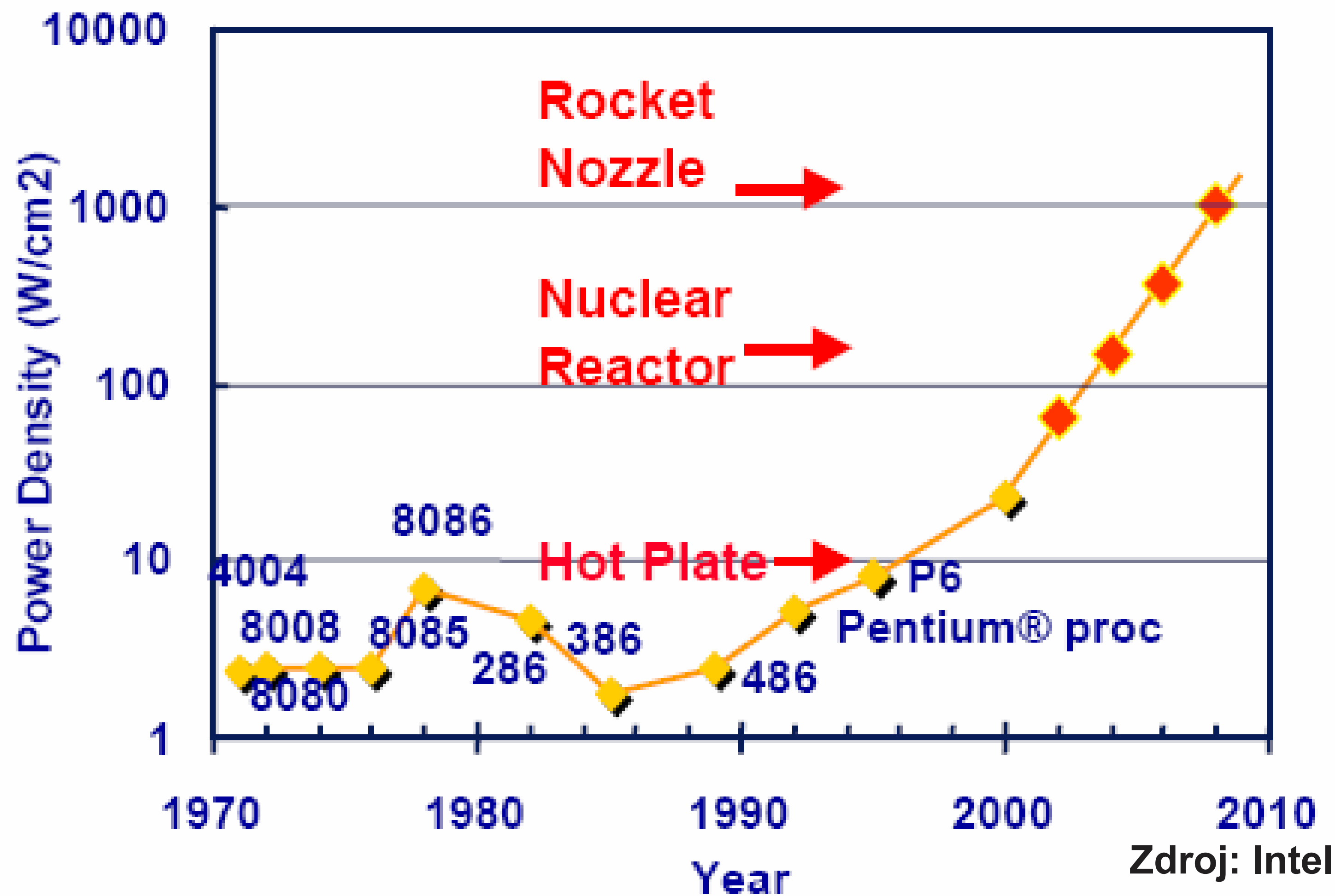
Zdroj: Intel

… ale obdobně roste i spotřeba



Zdroj: Intel

Plošná hustota příkonu také roste …



Nárůst plošné hustoty příkonu => nárůst teploty čipu => obtížnější chlazení - hlavní limit ve škálování hodinové frekvence

Možné otázky

- Proč se daří zvyšovat hodinovou frekvenci procesorů ?
- Proč nárůst frekvence ne vždy vede k odpovídajícímu nárůstu výkonnosti ?
- Jak je možné, že procesor s nižší hodinovou frekvencí je výkonnější ?
- Proč nárůst hodinové frekvence a výkonnosti v současnosti zpomaluje a jak budou vypadat budoucí procesory ?

Nárůst výkonnosti a hodinové frekvence

Dva zdroje nárůstu (první ale ovlivňuje druhý)

1. Zrychlování technologie:
   Zmenšení rozměrů v technologii faktorem F vede k odpovídajícímu zkrácení zpoždění hradla faktorem F
   Př.: Přechod ze 180 nm na 130 nm technologii vede ke zkrácení zpoždění o cca 30 %, což může vést k nárůstu hodinové frekvence o 1/0,7 = 43 %
   Dalším efektem škálování je např. možnost integrovat větší cache na čipu a tím minimalizovat čekání na paměť.

2. Vylepšení architektury – téma X36APS
   Superpipelining, superscalar, dynamic schedulling, speculative execution …

Nárůst hodinové frekvence i výkonnosti nad 1,43 je díky inovacím v architektuře procesoru vycházejícím z využití Paralelismu na Úrovni Instrukcí (ILP).

Osnova přednášky

- Shrnutí vlastností skalárního proudově pracujícího procesoru
- Zvyšování výkonnosti a technologické trendy
- **Pojem paralelismu na úrovni instrukcí (ILP)**
- Superpipeline procesory
- Superskalární procesory
- Procesory VLIW

Pojem paralelismu na úrovni instrukcí (ILP)

Clock cycle	IF stage	ID stage	EX stage	MEM stage	WB stage
1	Instr1	Instr0	Instr-1	Instr-2	Instr-3
2	Instr2	Instr1	Instr0	Instr-1	Instr-2
3	Instr3	Instr2	Instr1	Instr0	Instr-1
4	Instr4	Instr3	Instr2	Instr1	Instr0
5	Instr5	Instr4	Instr3	Instr2	Instr1

- **Všechny moderní procesory se snaží nějakou formou využít paralelismu mezi instrukcemi**
- **ILP = provádění instrukcí se může plně nebo částečně překrývat**
- **ILP je omezeno datovými, jmennými a řídícími závislostmi které jsou v programech inherentní**

Zvýšení výkonnosti – ILP techniky

Snížení IC – VLIW
(více paralelních operací
v jedné instrukci)

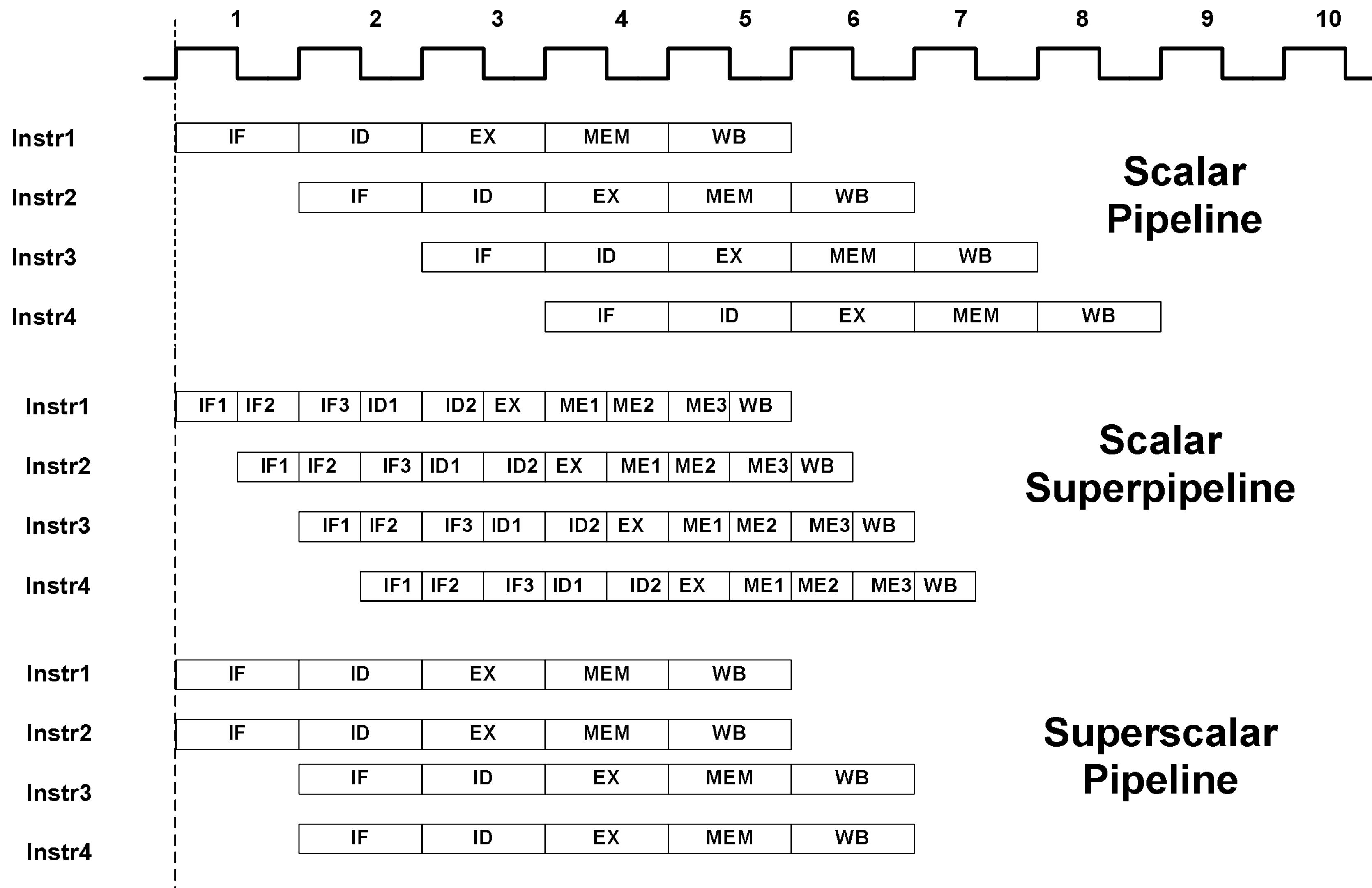
Snížení T_{clk} – superpipelining
(více stupňů pipeline)
a rychlejší technologie

$$T_{CPU} = IC * (CPI_{pipe_ideal} + \text{Stalls Per Instr}) * T_{clk}$$

Snížení CPI_{pipe_ideal} pomocí paralelním
prováděním více instrukcí – superskalární proc.

Překrytí pozastavování (stallů) užitečným výpočtem
– dynamické plánování, dynamická predikce skoku,
spekulativní provádění instrukcí …

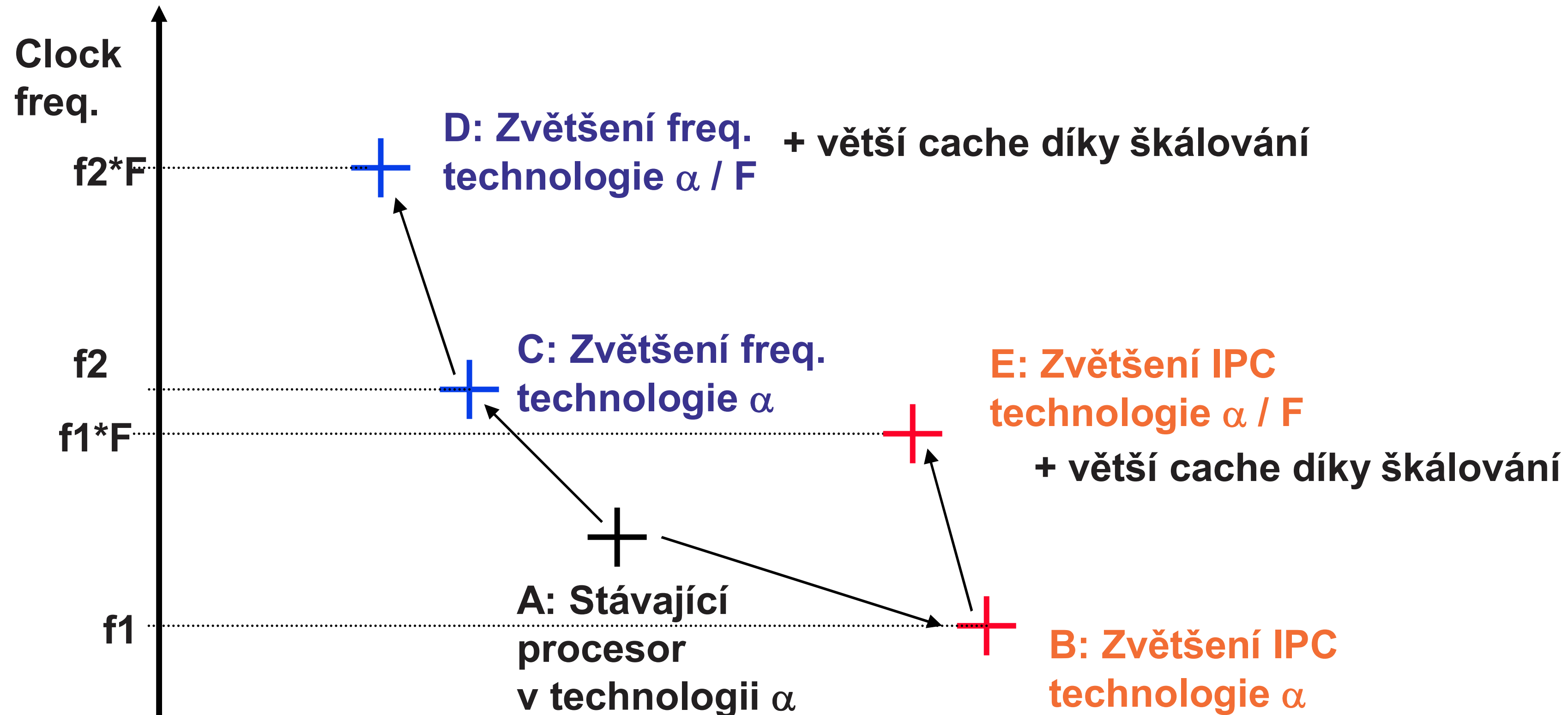
Zvýšení paralelismu v *pipeline* – dva přístupy



Superskalární proc. vs superpipeline proc.

- Tyto přístupy jsou vzájemně ortogonální
- Superpipeline využívá paralelismu v čase (mezi po sobě jdoucími stupni pipeline)
- Superscalar využívá paralelismu v prostoru (mezi paralelními stupni pipeline)
- Superpipeline zvyšuje hodinovou frekvenci (snižuje T_{clk})
- Superscalar zvyšuje Instruction Per Clock - IPC (snižuje CPI)
- HW složitost je podobná, ale je snazší přechod od skalární pipeline ke superpipeline než od skalární pipeline k superskalární pipeline
- Definice superskalární pipeline je jasná – $IPC_{ideal} > 1$ ($CPI_{ideal} < 1$)
- Definice skalární superpipeline není tak zřejmá
  – typicky S > 5 - 6 u celočíselné pipeline znamenalo superpipeline
- Moderní procesory jsou často kombinace – nazývané *SUPER – SUPER*
- *Pro nejnovější procesory se S>30 se zavádí pojem hyperpipelining*

Superskalár vs Superpipeline: Kam se vydat ?



Zvětšení hodinové frekvence zhoršuje IPC (za jinak stejných podmínek).
Zvětšení IPC zhoršuje hodinovou frekvenci (za jinak stejných podmínek).

Zvýšit hodinovou frekvenci procesoru lze v dané technologii i zvýšením napájecího napětí => to ovšem vede ke kubickému nárůstu spotřeby čipu…

Osnova přednášky

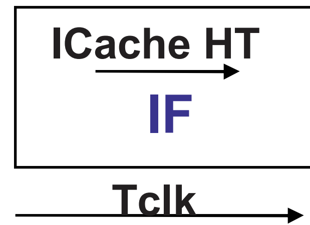
- Shrnutí vlastností skalárního proudově pracujícího procesoru
- Zvyšování výkonnosti a technologické trendy
- Pojem paralelismu na úrovni instrukcí (ILP)
- **Superpipeline procesory**
- Superskalární procesory
- Procesory VLIW

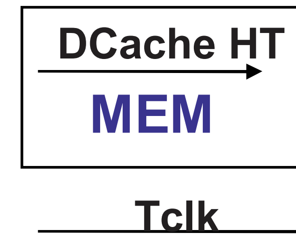
Superpipelining = Více Stupňů Pipeline – IF a MEM

IF stupeň a MEM stupeň

- vhodné rozdělit do více stupňů pipeline což umožní přístup do větší cache (menší MR, větší HT) bez vlivu na Tclk

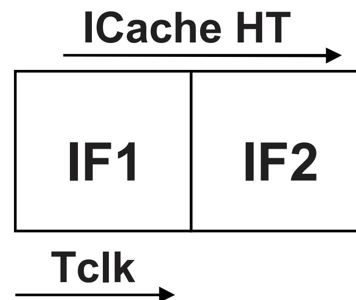
DLX
5-stage
pipeline

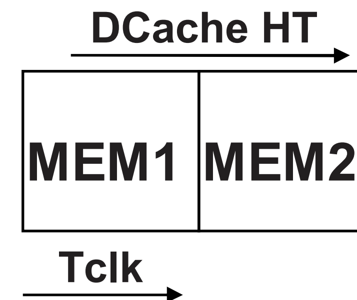




Tclk>HT

DLX
7-stage
superpipeline





2*Tclk>HT

Přístup do cache (a TLB) nyní trvá 2 takty, cache však umožňuje začít čtení každý takt – přístup do cache je pipelineován.

Superpipelining = Více Stupňů Pipeline – ID a EX

ID stupeň

- pokud je dekódování složité je možné rozdělit ho typicky do 2 stupňů:
- dekódování (ID) a čtení registrů (RF)

EX stupeň (výkonné jednotky)

- je-li některá operace příliš složitá na provedení v 1 taktu je vhodné ji rozdělit do více stupňů aby se nestala omezením Tclk

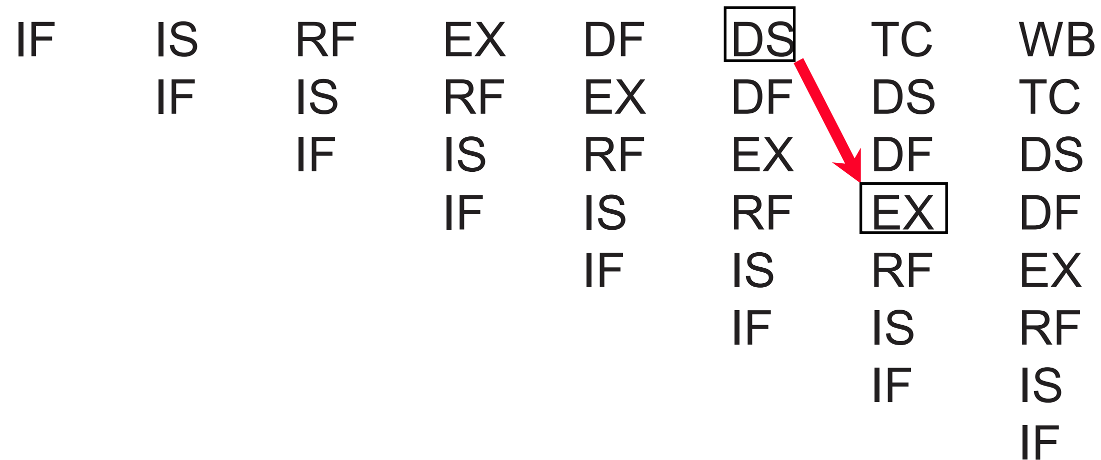
Příklad: MIPS R4000 (1995, 200 MHz) – superpipeline proc.

- 8 stupňová pipeline:
  - IF– první polovina načtení instrukce; výběr PC a počátek přístupu do instrukční skryté paměti (cache)
  - IS–druhá polovina přístupu do instrukční cache.
  - RF–dekódování instrukce a čtení registrů, kontrola hazardů a also detekce úspěšnosti načtení z cache (instruction cache hit)
  - EX–provádění, které zahrnuje výpočet efektivní adresy, ALU operace a výpočet cílové adresy skoku a vyhodnocení podmínky.
  - DF–data fetch, první polovina přístupu do datové cache
  - DS–druhá polovina přístupu k datové cache.
  - TC–tag check, detekce úspěšnosti načtení z cache
  - WB–write back (pro instrukce load a registr-registr operace).
- 8 Stupňů: Jak se změní „Load delay“? „Branch delay“ ?

© David Patterson, CS252 UCB 2000

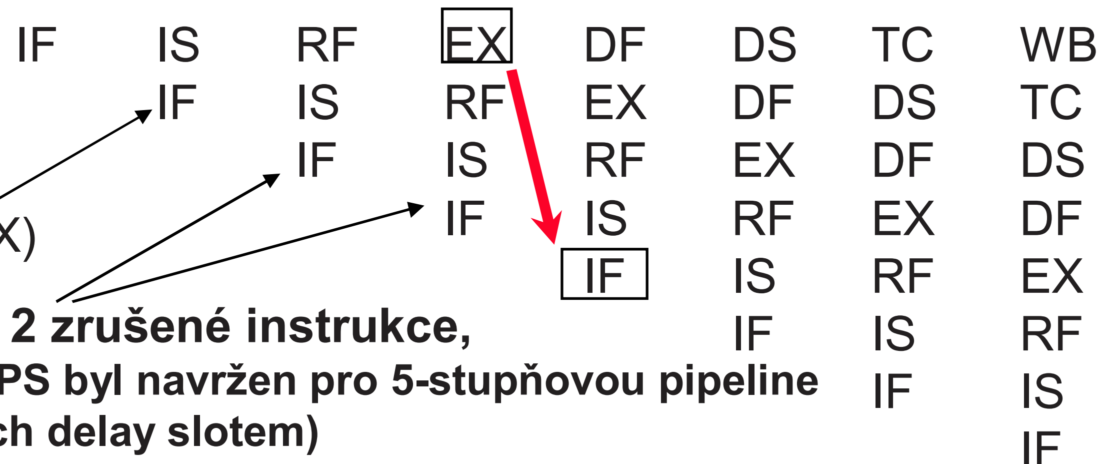
Příklad: MIPS R4000

2 taktový „Load Delay Slot“



3 taktový „Branch Delay Slot“

(podmínka a adresa vypočtena ve stupni EX)



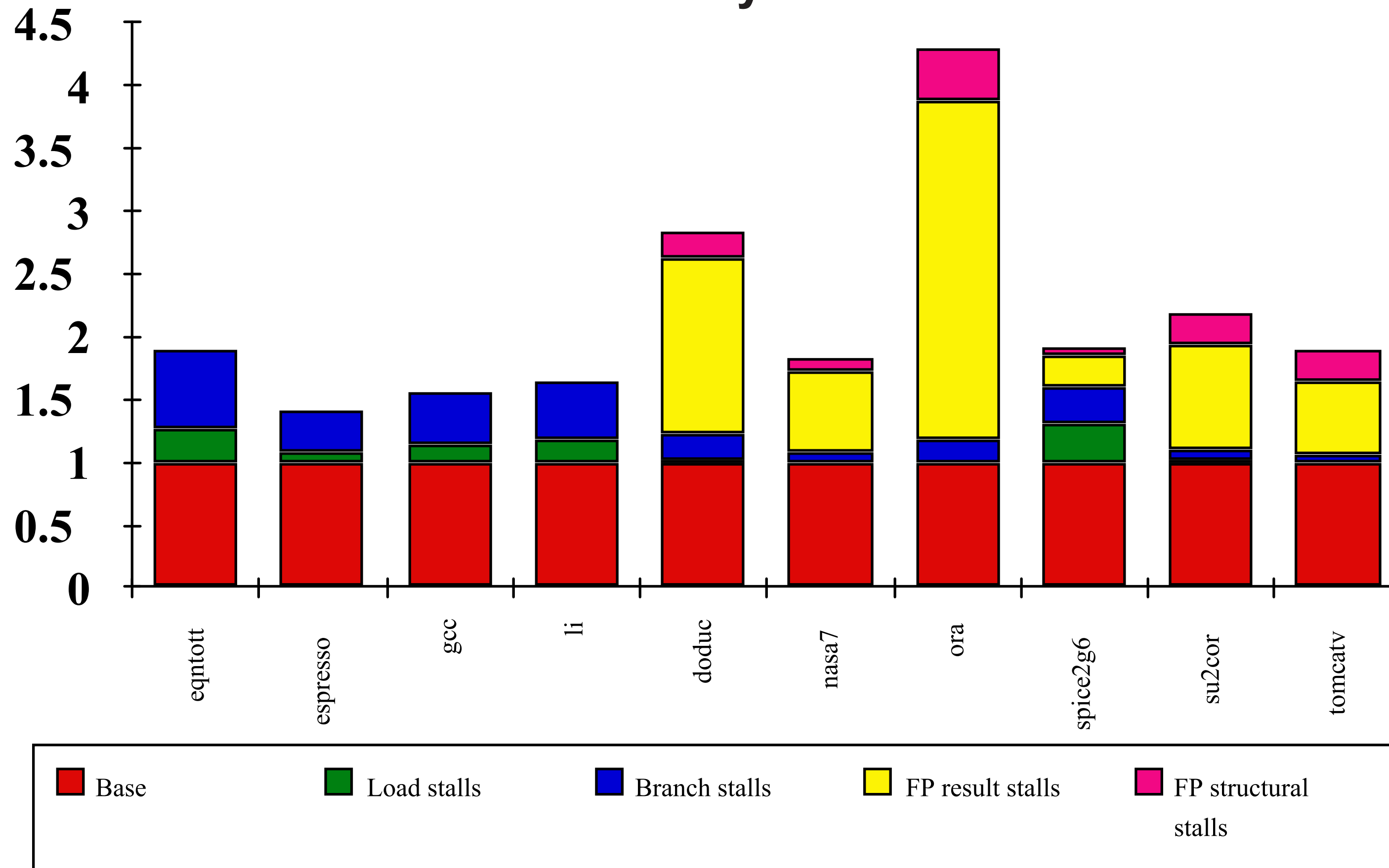
Původní delay slot + 2 zrušené instrukce,

(Zpožděný skok pro MIPS byl navržen pro 5-stupňovou pipeline s jednotaktovým branch delay slotem)

© David Patterson, CS252 UCB 2000

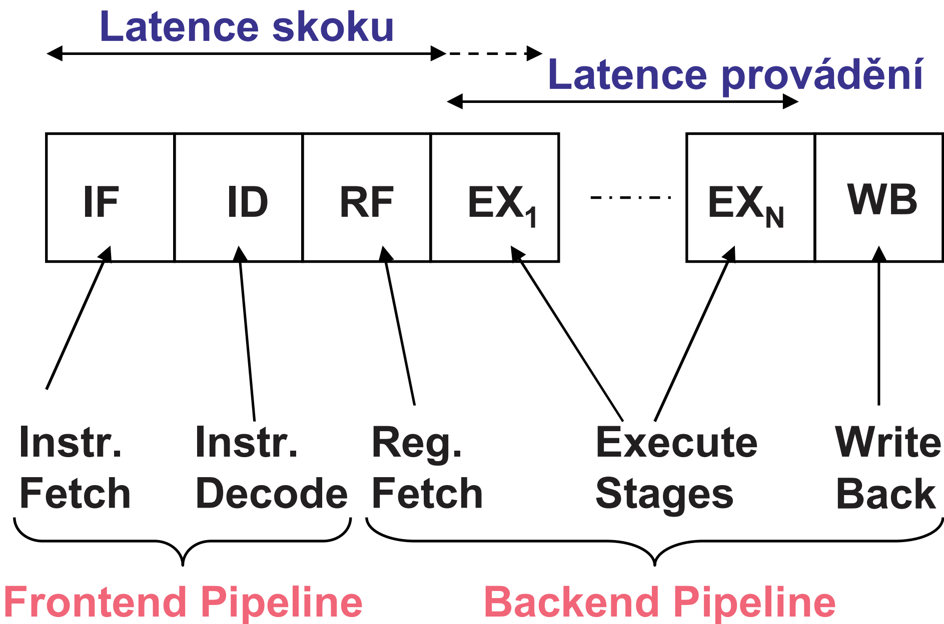
R4000 Výkonnost

- CPI není zdaleka 1:
  - Load stalls (1 nebo 2 takty)
  - Branch stalls (2 takty + nezaplněné br. delay sloty)
  - FP result stalls: RAW datové hazardy (latence)
  - FP structural stalls: Konflikty u FP hardware



© David Patterson, CS252 UCB 2000

Zvýšení Výkonnosti Superpipeliningem - Problémy



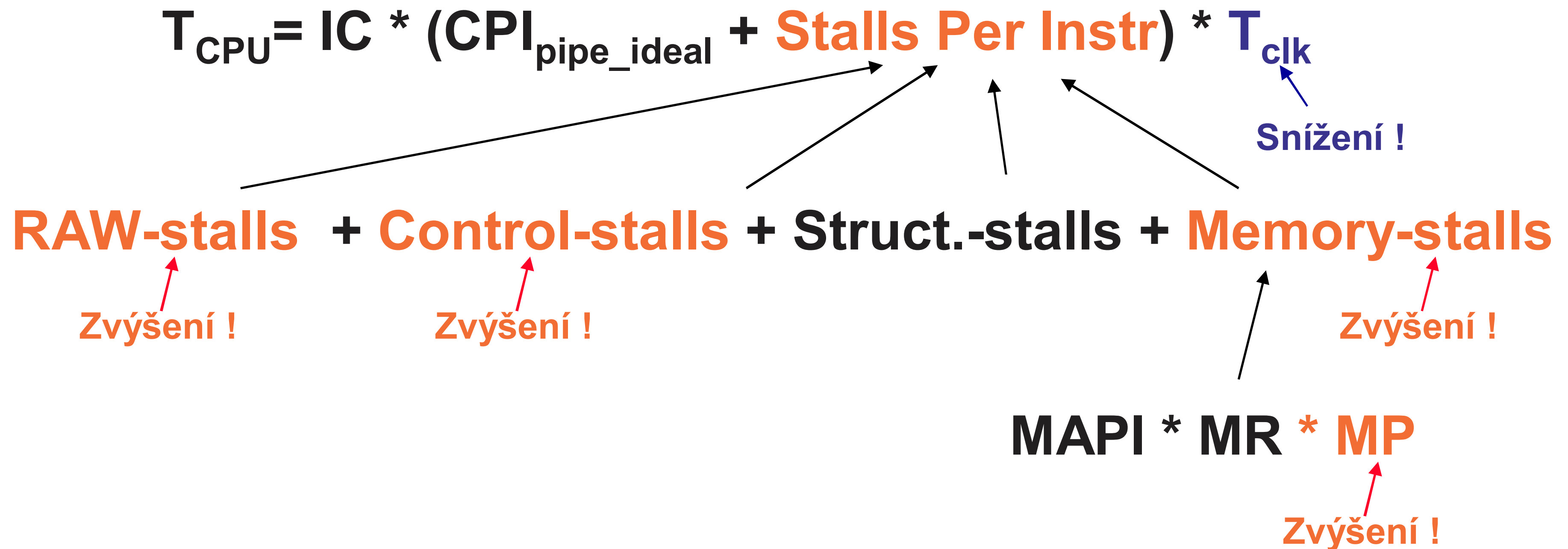
- Zvyšování počtu stupňů v přední části (frontend) zvyšuje latenci skoku
- Zvyšování počtu stupňů v zadní části (backend) zvyšuje latenci provádění

Vliv zkrácení T_{clk} na paměťový subsystém:

Vybavovací doba nižších úrovní paměti = MP * T_{clk}

- Zkrácení T_{clk} vede k relativnímu nárůstu doby výpadku MP

Zvýšení výkonnosti superpipeliningem - problémy

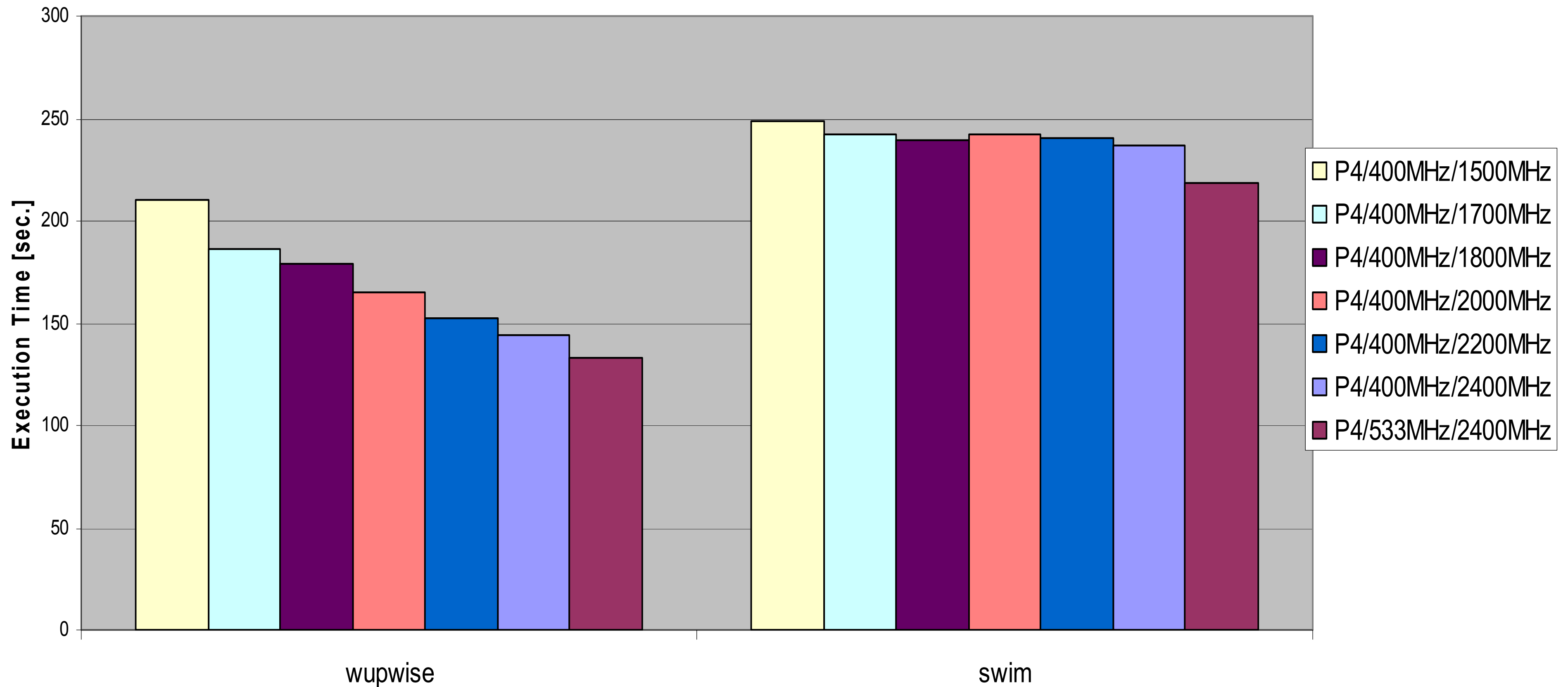


Výhodou superpipeliningu je, že snižování T_{clk} je dále posíleno díky škálování technologie a naivní uživatel ztotožňuje f_{clk} s výkonností počítače.

T_{clk} však neklesá lineárně s počtem stupňů pipeline (kvůli T_{skew}, T_{dq} a dalším limitům) a vede k nárůstu SPI a tím i celkového CPI.

Doba výpočtu dvou benchmarků jako $f(f_{clk})$

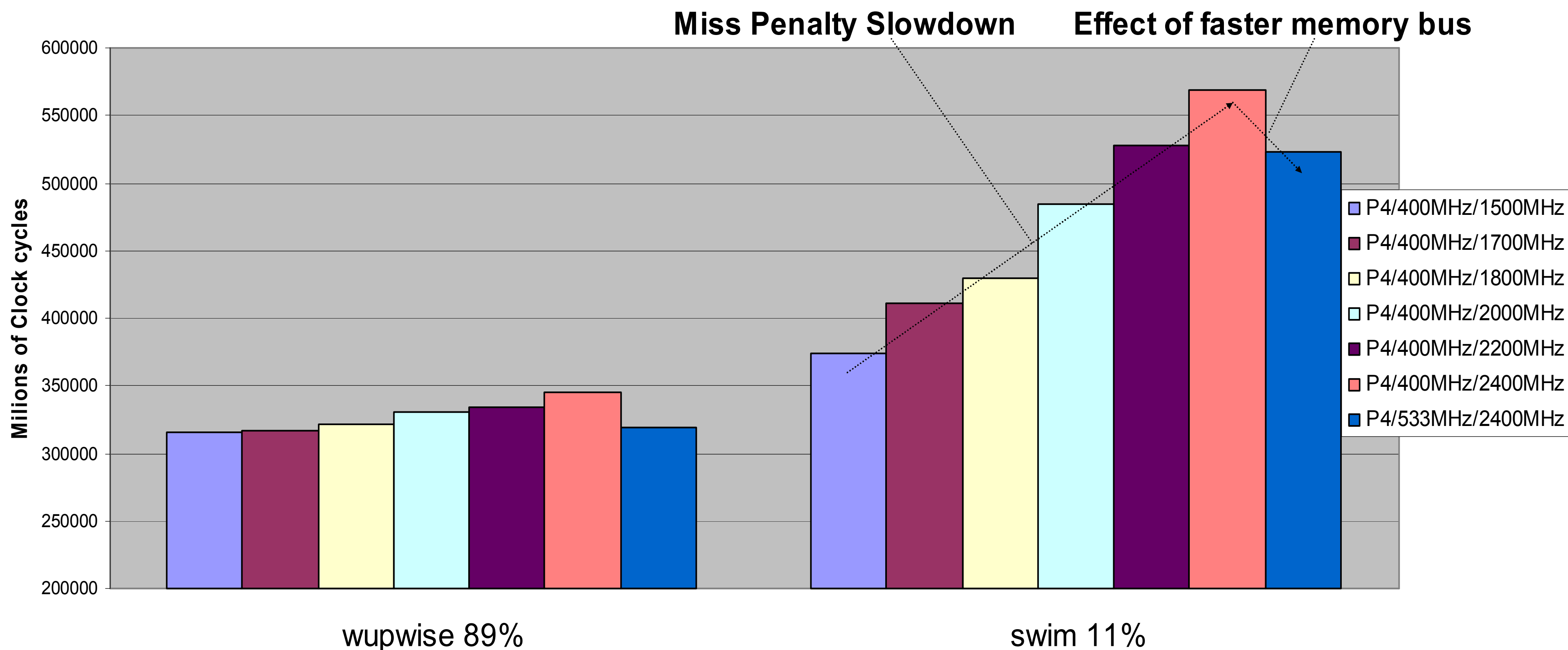
Proč je *wupwise* urychlován a *swim* není ?



Zdroj:http://spec.org

Počet hodinových taktů jako f(f_{clk})

Wupwise **se vejde do L3 cache,** ***Swim*** **nikoli…**



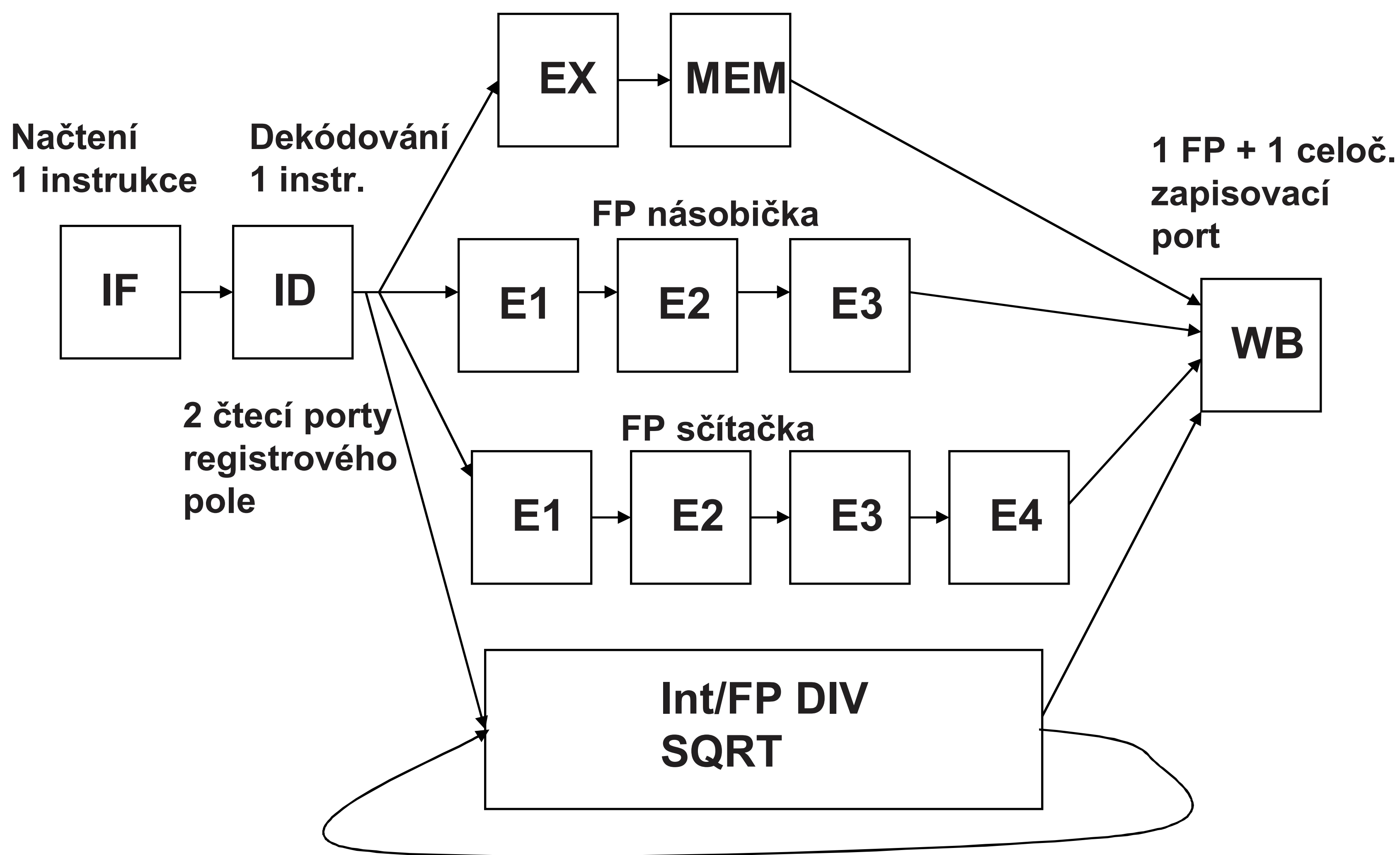
Zdroj:http://spec.org

S rostoucí hodinovou frekvencí roste i počet taktů na daném procesoru v důsledku nárůstu vlivu MP na celkové CPI.

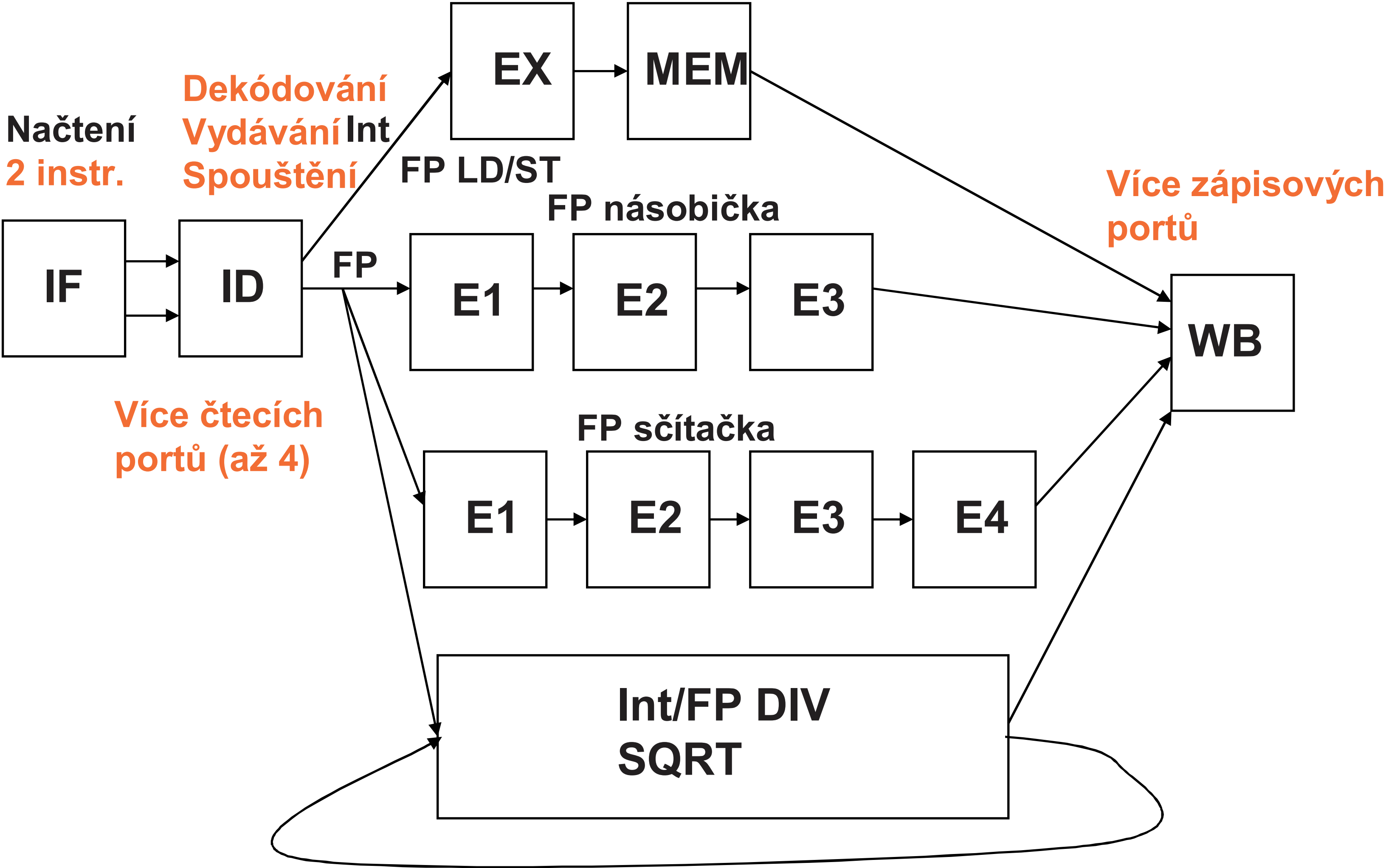
První Superskalární Procesory = Statické Superskaláry

- První superskalární procesory byly staticky plánované => pouze instrukce přesně za sebou v programu mohly být spuštěny paralelně
- Statické plánování instrukcí = provádění instrukcí v programovém pořadí
- Paralelní provádění může zahájit pouze limitovaná kombinace typů instrukcí
- Příklad – IBM Power1 – může spustit v každém taktu:
  - 1 celočíselnou instrukci
  - 1 FP instrukci
  - 1 podmíněný skok
  - 1 instrukci s podmínkovým registrem
- Procesor je jen o něco složitější než konvenční skalární procesor
- Nejtypičtější pro první generaci superskalárních procesorů bylo paralelní provádění celočíselné a FP instrukce (např. Intel 860)
- Superskalární Pentium (první superskalární x86) také patří mezi tuto kategorii procesorů (limitované kombinace "*párovatelných instrukcí*" v pipelinách U a V)

Proudově pracující skalární DLX



Superskalární DLX – FP a Int



Superskalární DLX - příklad statického superskaláru

- Superskalární DLX: 2 instrukce, 1 FP a 1 jiná
  – Načtení 2 instr. - 64 bitů/takt; Int vlevo, FP vpravo (příp. i opačně)
  – Spustí druhou instrukci pokud první instrukce je spuštěna (spouštění v programovém pořadí) – *in order issue*
  – Více zápisových portů na FP reg. poli pro párování FP load & FP ops

Typ	*Stupně pipeline*						
Int. instruction	IF	ID	EX	MEM	WB		
FP instruction	IF	ID	EX	MEM	WB		
Int. instruction		IF	ID	EX	MEM	WB	
FP instruction		IF	ID	EX	MEM	WB	
Int. instruction			IF	ID	EX	MEM	WB
FP instruction			IF	ID	EX	MEM	WB

- 1 taktová load delay znamená 3 instrukce v Superskalárním DLX
  - Instrukce in vpravo nemůže použít výsledek load, ani žádná ze 2 instrukcí v následujícím spoštěcím slotu
  - podobný vliv se uplatní u podmíněného skoku

© David Patterson, CS252 UCB 2000

Superskalární DLX – příklad na optimalizaci cyklu

° Uvažujme následující cyklus (k prvkům pole v paměti přičítáme F2)

```
LOOP:   LF     F0,0(R1)
        ADDF   F4,F0,F2
        SF     0(R1),F4
        SUBI   R1,R1,#4
        BNEZ   R1,LOOP
```

Kolik taktů bude trvat provádění tohoto cyklu na superskalárním DLX ?

Uvažujme, že ADDF bude mít latenci 3 takty (3 stupně EX).

V IF načítáme 2 libovolné instrukce, stupeň ID a další je rozdělen pro celočíselné a FP instrukce (ID_{INT} a ID_{FP}) V ID tedy může v daném taktu být pouze 1 celočíselná a 1 FP instrukce. IF je blokováno dokud obě instrukce nepostoupí do ID.

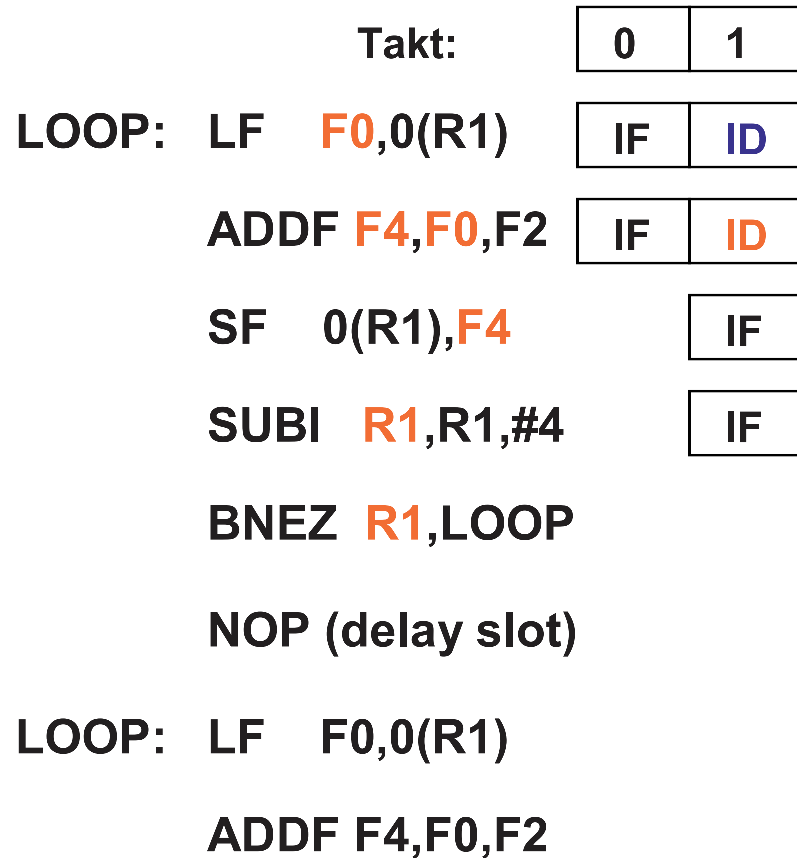
© David Patterson, CS252 UCB 2000

Superskalární DLX – příklad – takt #0

```
           Takt:          0
LOOP:  LF   F0,0(R1)      IF
       ADDF F4,F0,F2      IF
       SF   0(R1),F4
       SUBI R1,R1,#4
       BNEZ R1,LOOP
       NOP (delay slot)
LOOP:  LF   F0,0(R1)
       ADDF F4,F0,F2
```

Načtení dvou instrukcí, jedné celočíselné (INT) a jedné v pohyblivé ř.č. (FP)

Superskalární DLX – příklad – takt #1



LF je spuštěna, ADDF bude čekat v ID_{FP} na F0, další 2 instrukce načteny, SF bude pokračovat do ID_{INT}, SUBI musí čekat v IF na uvolnění ID_{INT} .

Superskalární DLX – příklad - takt #2

Instrukce	Takt: 0	1	2
LOOP: LF F0,0(R1)	IF	ID	EX
ADDF F4,F0,F2	IF	ID	ID
SF 0(R1),F4		IF	ID
SUBI R1,R1,#4		IF	IF
BNEZ R1,LOOP			
NOP (delay slot)			
LOOP: LF F0,0(R1)			
ADDF F4,F0,F2			

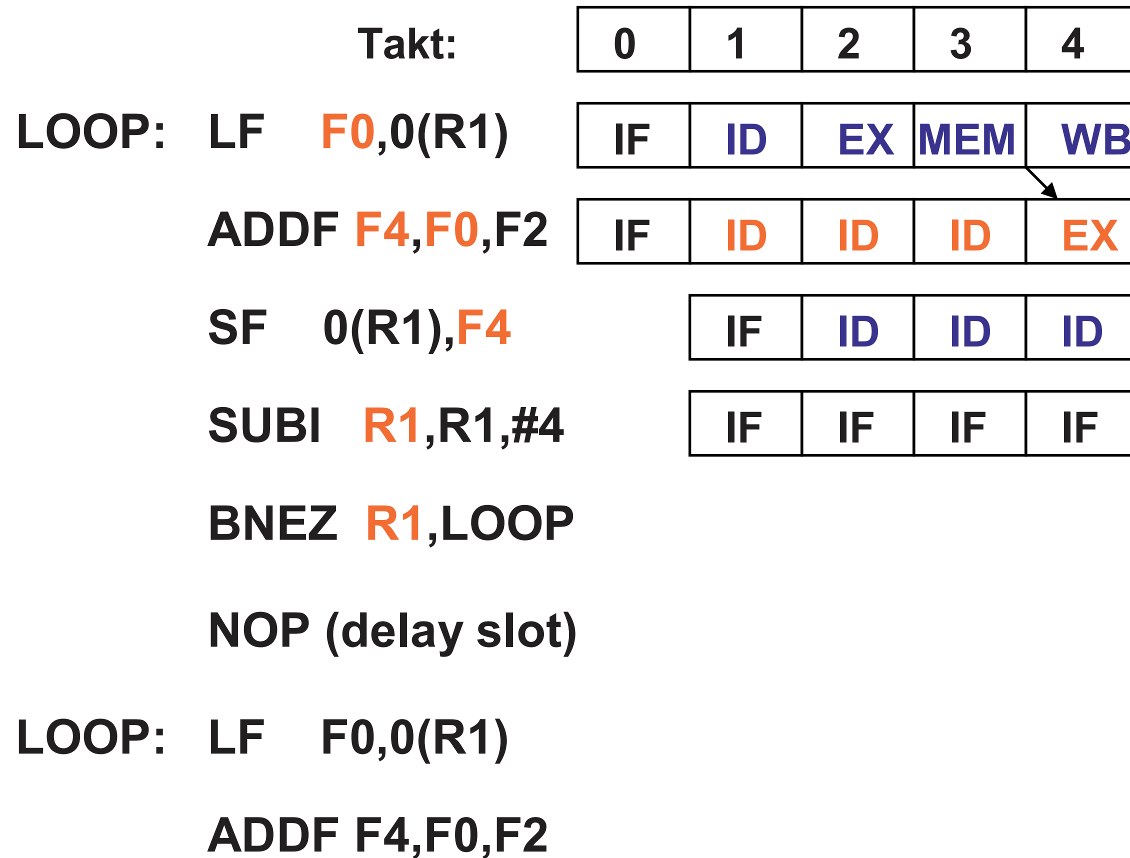
LF vypočítává adresu, ADDF čeká v ID_{FP} na F0, SF musí čekat v ID_{INT} na F4 a spuštění ADDF, SUBI musí čekat v IF na uvolnění ID_{INT} čímž je IF blokováno.

Superskalární DLX – příklad - takt #3

	Takt:	0	1	2	3
LOOP:	LF F0,0(R1)	IF	ID	EX	MEM
	ADDF F4,F0,F2	IF	ID	ID	ID
	SF 0(R1),F4		IF	ID	ID
	SUBI R1,R1,#4		IF	IF	IF
	BNEZ R1,LOOP				
	NOP (delay slot)				
LOOP:	LF F0,0(R1)				
	ADDF F4,F0,F2				

LF přistupuje do cache, ADDF opustí ID_{FP} (v dalším taktu FW F0),
SF musí čekat v ID_{INT} na F4 a spuštění ADDF,
SUBI musí čekat v IF na uvolnění ID_{INT} čímž je IF blokováno.

Superskalární DLX – příklad - takt #4



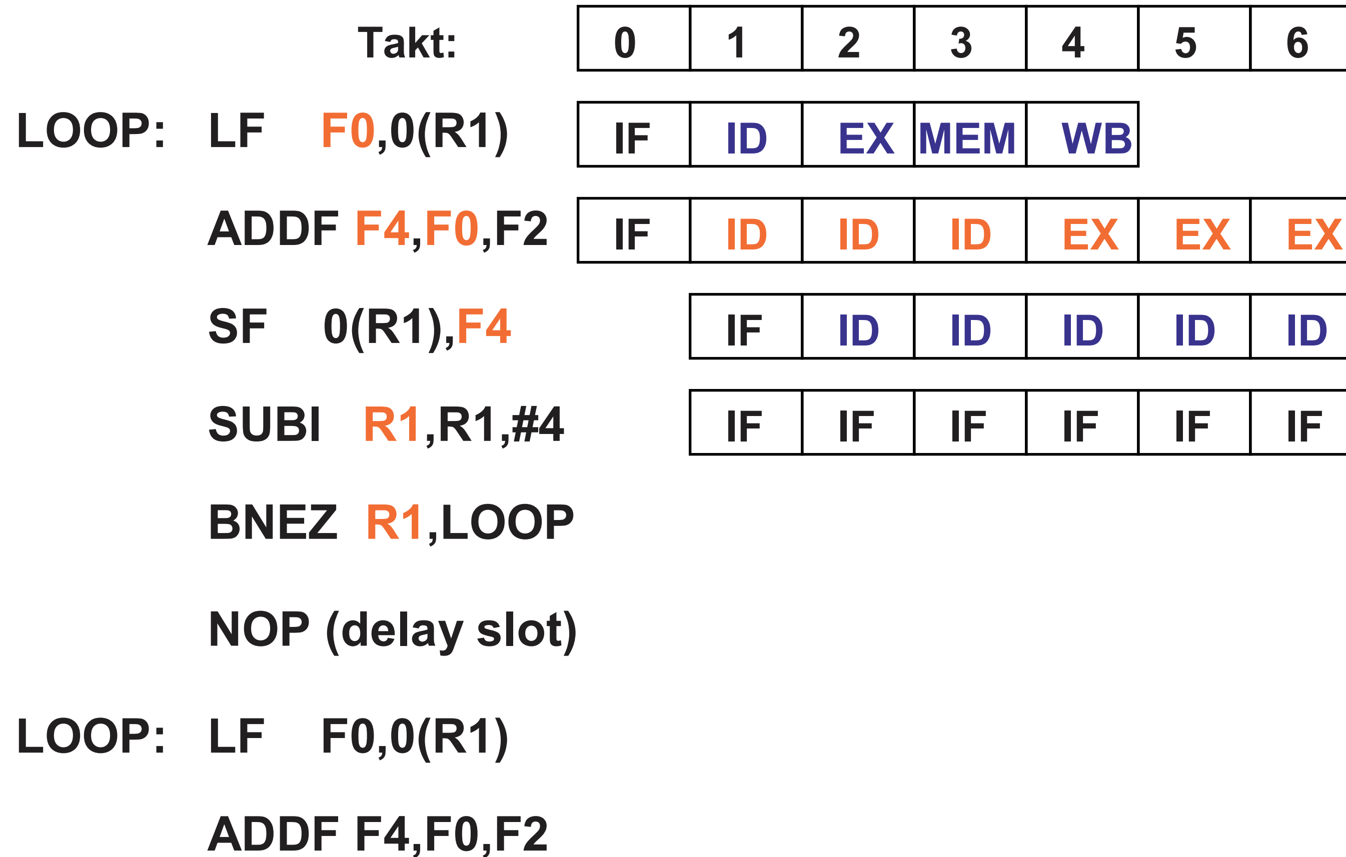
Forwarding F0 od LF k ADDF, SF musí čekat v ID_{INT} na F4, SUBI musí čekat v IF na uvolnění ID_{INT} čímž je IF blokováno.

Superskalární DLX – příklad - takt #5

Takt:	0	1	2	3	4	5
LOOP: LF F0,0(R1)	IF	ID	EX	MEM	WB	
ADDF F4,F0,F2	IF	ID	ID	ID	EX	EX
SF 0(R1),F4		IF	ID	ID	ID	ID
SUBI R1,R1,#4		IF	IF	IF	IF	IF
BNEZ R1,LOOP						
NOP (delay slot)						
LOOP: LF F0,0(R1)						
ADDF F4,F0,F2						

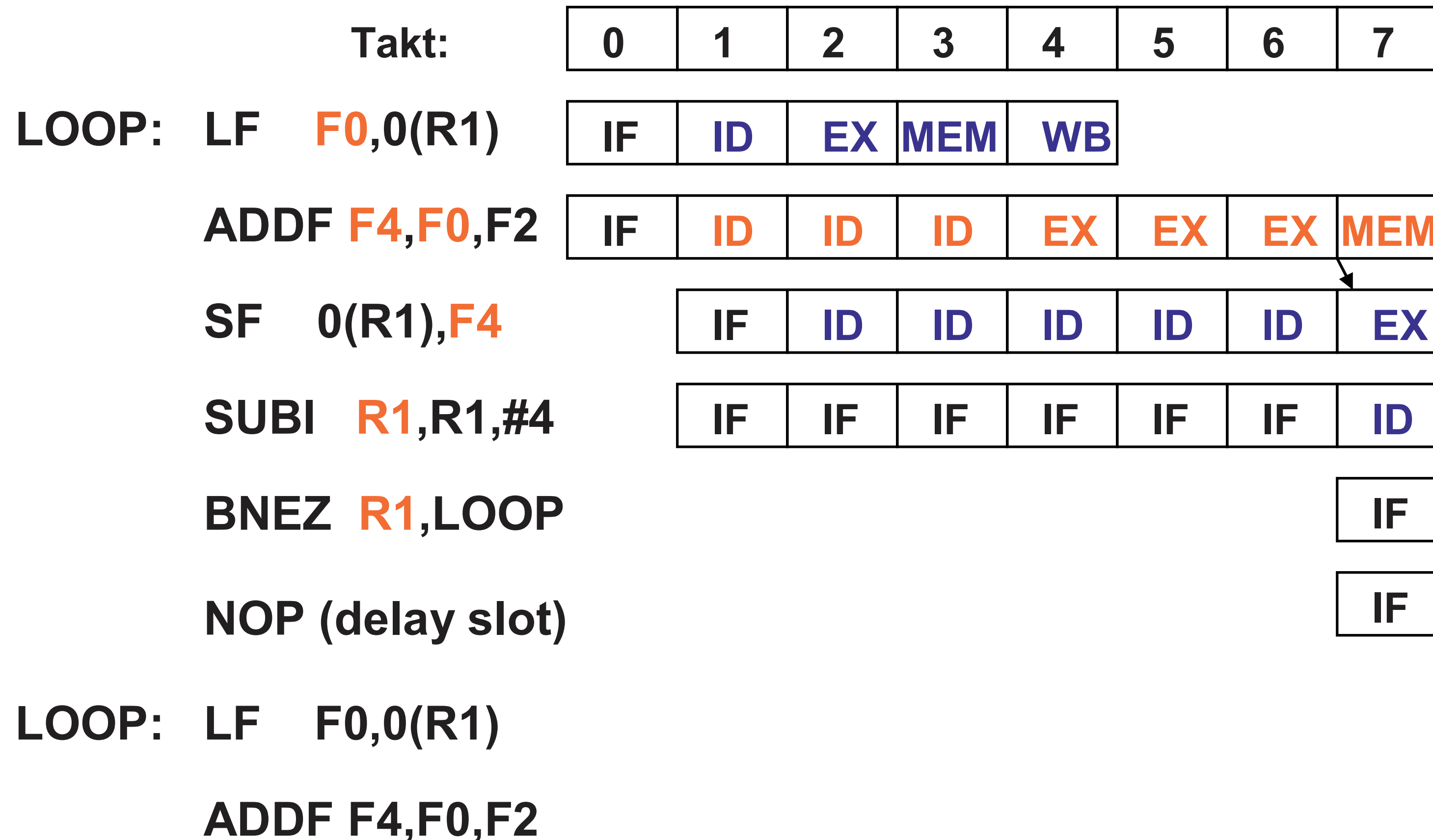
ADDF se provádí, SF musí čekat v ID_{INT} na F4,
SUBI musí čekat v IF na uvolnění ID_{INT} čímž je IF blokováno.

Superskalární DLX – příklad - takt #6



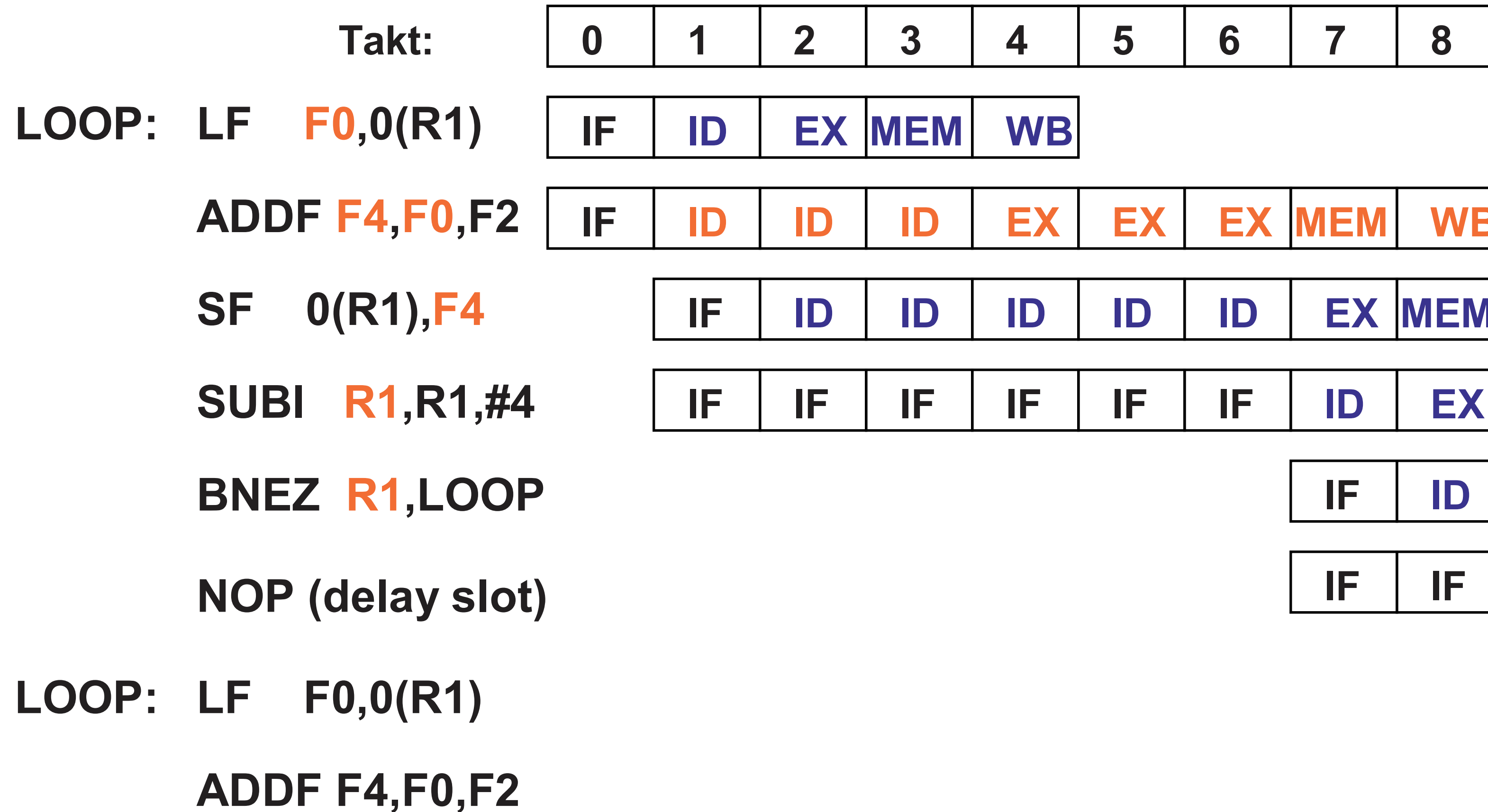
ADDF dokončuje výpočet, SF opustí ID_{INT} (forwarding F4 v dalším taktu), SUBI postoupí do ID_{INT} čímž se uvolní IF.

Superskalární DLX – příklad - takt #7



Forwarding F4 ADDF k SF, SF počítá adresu, SUBI načítá registry v ID_{INT} , BNEZ a NOP načteny, obě jsou celočíselné.

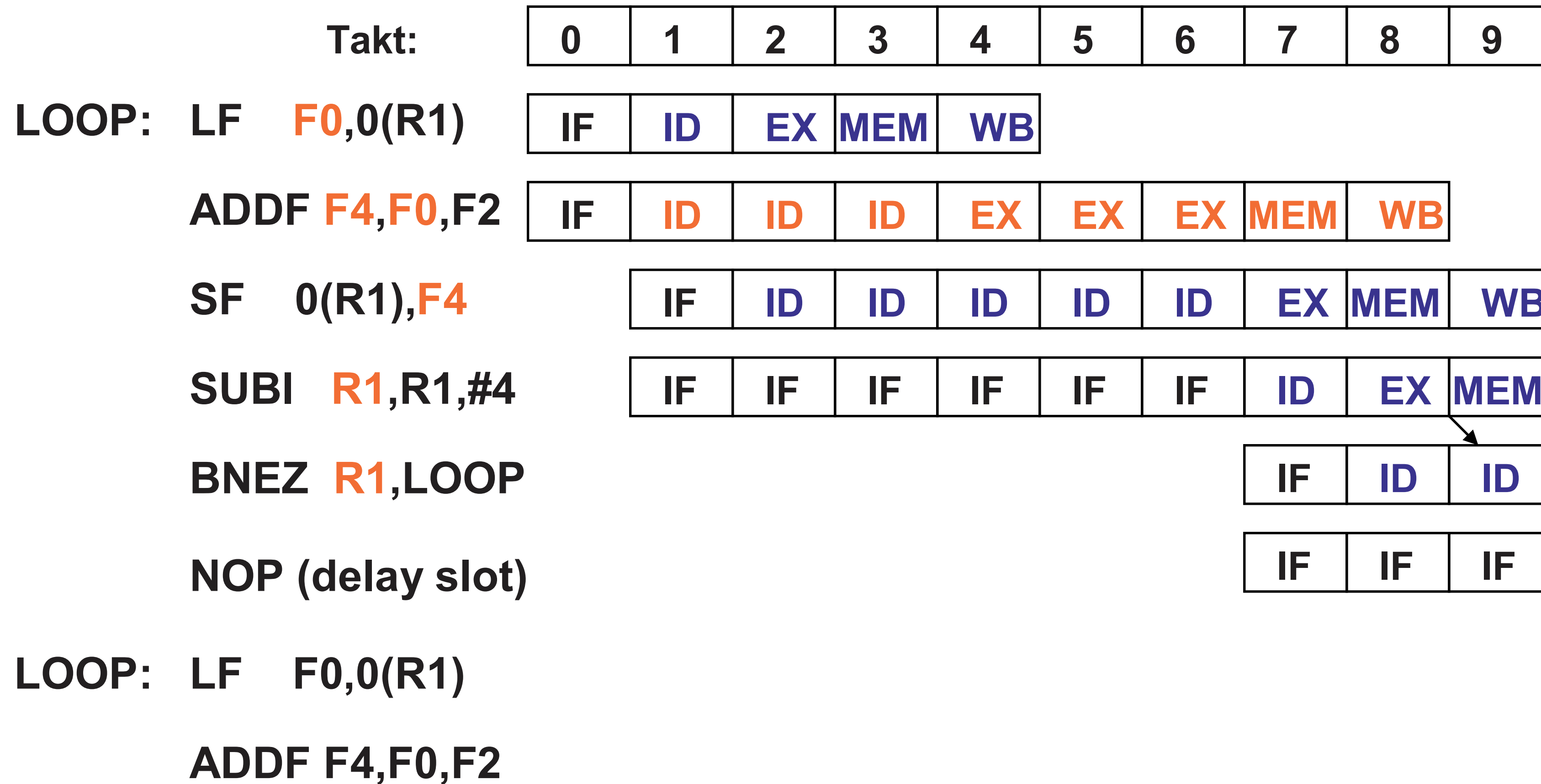
Superskalární DLX – příklad - takt #8



Takt:	0	1	2	3	4	5	6	7	8
LOOP: LF F0,0(R1)	IF	ID	EX	MEM	WB				
ADDF F4,F0,F2	IF	ID	ID	ID	EX	EX	EX	MEM	WB
SF 0(R1),F4		IF	ID	ID	ID	ID	ID	EX	MEM
SUBI R1,R1,#4		IF	IF	IF	IF	IF	IF	ID	EX
BNEZ R1,LOOP								IF	ID
NOP (delay slot)								IF	IF
LOOP: LF F0,0(R1)									
ADDF F4,F0,F2									

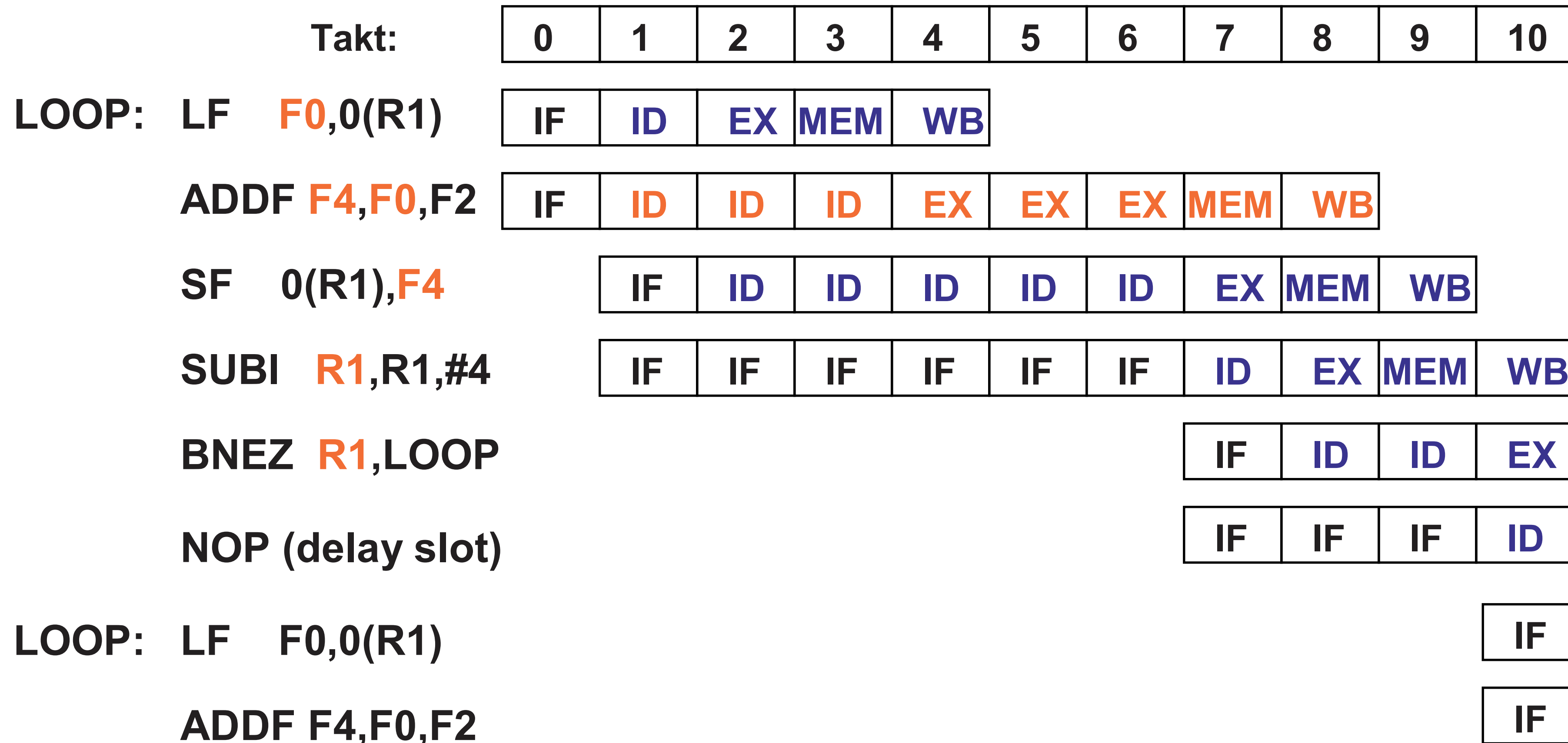
SF přistupuje do paměti, SUBI provádí odečtení, BNEZ musí čekat v ID_{INT} na hodnotu R1, NOP čeká na uvolnění ID_{INT}

Superskalární DLX – příklad - takt #9



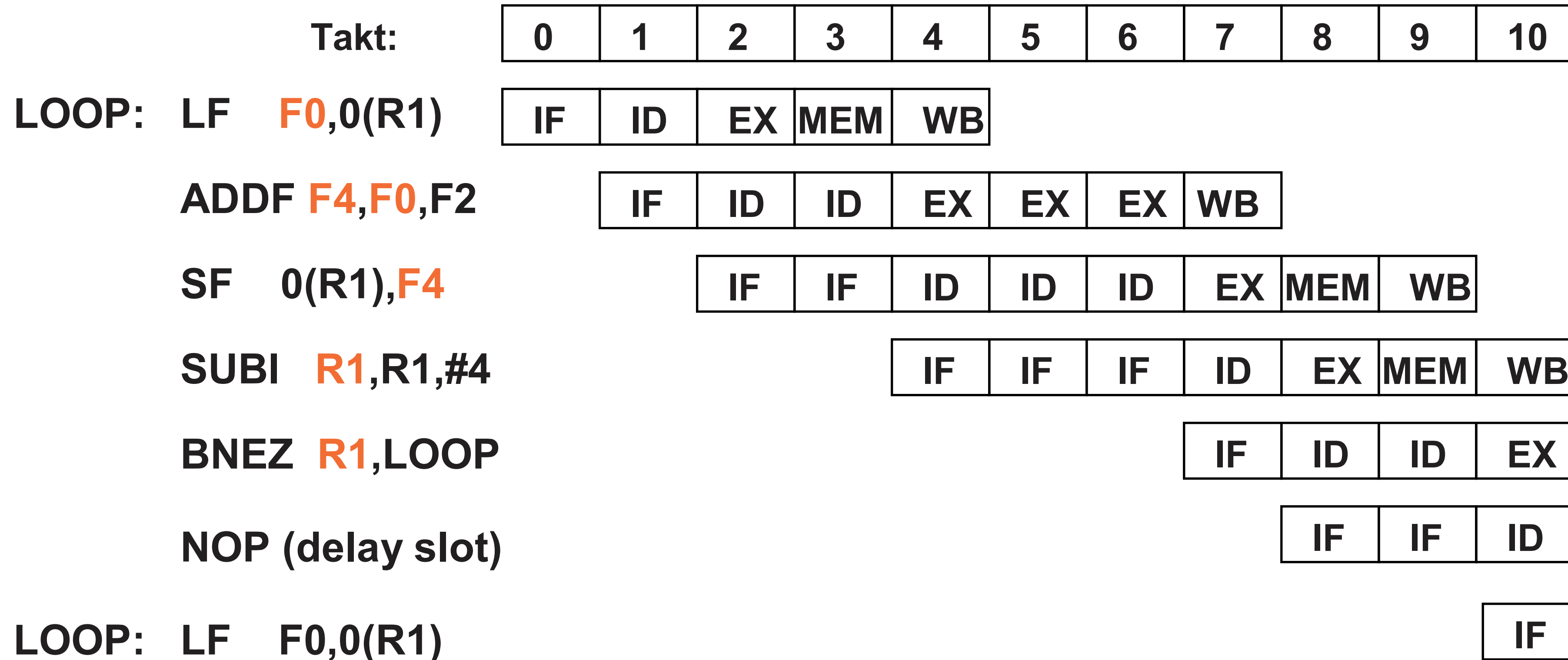
Forwarding R1 pro BNEZ do ID_{INT} , změna PC na návěstí LOOP, NOP postoupí v dalším taktu do ID_{INT}

Superskalární DLX – příklad - takt #10



Načítám opět LF a ADDF …

Skalární DLX – příklad - takt #10



Výpočet na skalárním DLX by trval úplně stejně dlouho …

Superskalární DLX – příklad na optimalizaci cyklu

	Celočíselná pipeline	FP pipeline	Takt (opuštění ID)
LOOP:	LF F0,0(R1)	Prázdný slot	1
	Prázdný slot	Prázdný slot	2
	Prázdný slot	ADDF F4,F0,F2	3
	Prázdný slot	Prázdný slot	4
	Prázdný slot	Prázdný slot	5
	SF 0(R1),F4	Prázdný slot	6
	SUBI R1,R1,#4	Prázdný slot	7
	Prázdný slot	Prázdný slot	8
	BNEZ R1,LOOP	Prázdný slot	9
	NOP (Branch Delay Slot)	Prázdný slot	10

RAW hazard

RAW hazard

RAW hazard

Jedna iterace trvá 10 taktů (75 % prázných slotů).

 © David Patterson, CS252 UCB 2000

Rozbalený a naplánovaný cyklus pro superskalární DLX

	Celočíselná instr.	FP instrukce	Takt
Loop:	LF F0,0(R1)	Prázdný slot	1
	LF F6,-4(R1)	Prázdný slot	2
	LF F10,-8(R1)	ADDF F4,F0,F2	3
	LF F14,-12(R1)	ADDF F8,F6,F2	4
	LF F18, -16(R1)	ADDF F12,F10,F2	5
	SF 0(R1),F4	ADDF F16,F14,F2	6
	SF -4(R1),F8	ADDF F20,F18,F2	7
	SF - 8(R1),F12	Prázdný slot	8
	SF -12(R1),F16	Prázdný slot	9
	SUBI R1,R1,#20	Prázdný slot	10
	BNEZ R1,LOOP (delayed br.)	Prázdný slot	11
	SF 4(R1),F20	Prázdný slot	12

12 taktů na 5 iterací, neboli 2.4 taktu na iteraci (30 % prázdných slotů)

© David Patterson, CS252 UCB 2000

Limity Staticky Plánovaných Superskalárních Procesorů

- Ačkoli spouštění celočíselné a FP instrukce paralelně je jednoduché pro HW, CPI = 0.5 pouze pro program kde je:
  - přesně 50% FP instrukcí ☺
  - žádné hazardy ☺

- Závisí silně na kvalitním překladači (relativně slabá výkonnost na neoptimalizovaných starších programech)
  – pro řadu aplikací to však je vyhovující a ze staticky plánovaného superskalárního procesoru dostaneme obdobnou výkonnost jako z dynamicky plánovaného superskalárního procesoru pokud máme kvalitní překladač

- Komerční statické superskaláry dnes
  - SUN UltraSPARC
  - PowerPC jádro v Cell (Sony PS3), PowerPC v XENON (Microsoft XBOX360), použit i v Nintendo hrací konzoli …

Problémy ve Statických Superskalárech

- Chceme-li spouštět více instrukcí paralelně, stupeň dekódování a spouštění (decode/dispatch/issue) se komplikuje :
  - I pro 2-cestný superskalár => dekódovat 2 oper. znaky, 6 čísel registrů a rozhodnout zdali bude spuštěna 0, 1 nebo 2 instrukce + detekce strukturních hazardů, RAW, WAW hazardů řízení forwardingu ...

- Superskalární CPU může mít horší T_{clk} kvůli komplexnímu stupni ID

$$T_{CPU} = IC * (CPI_{pipe_ideal} + \text{Stalls Per Instr}) * T_{clk}$$

sníženo (CPI) zvýšeno (Stalls Per Instr) zvýšeno ? (T_{clk})

Několik poznámek:

- Superskalární procesor často kombinován se superpipeliningem (širší pipeline je efektivnější rozdělit i do více stupňů)
- Mezi frontend a backend pipeline je vhodné vložit frontu,která zabrání šíření pozastavení pipeline z backendu do frontendu a poskytne backendu instrukce, když frontend stojí…

Very Long Instruction Word – VLIW – Alternativa ?

- Výkonnost statického superskaláru silně závisí na kvalitním překladači, hardware je relativně jednoduchý s výjimkou logiky detekce a řešení hazardů
  => Proč tedy nepřesunout tuto zodpovědnost na SW (překladač)

- Toto je přístup (Very) Long Instruction Word - (V)LIW

Int operation	Int operation	FP operation	FP operation	Branch
32b	32b	32b	32b	32b

Dlouhá instrukce = sém. jednotka pro přerušení

Původní (krátká) instrukce

- Pevný formát instrukce obsahuje kód několika operací které mohou být provedeny paralelně
- HW nekontroluje hazardy => je jednoduchý, rychlý a škálovatelnější (zůstává problém škálování forwardingu a vícebránového reg. pole)

$$T_{CPU} = IC * (CPI_{pipe_ideal} + \text{Stalls Per Instr}) * T_{clk}$$

Snížení (IC)

Zodpovědnost překladače (Stalls Per Instr)

Program pro superskalární DLX vs (V)LIW DLX

```
LF    F0,0(R1)
LF    F6,-4(R1)
LF    F10,-8(R1)
ADDF  F4,F0,F2
LF    F14,-12(R1)
ADDF  F8,F6,F2
LF    F18, -16(R1)
ADDF  F12,F10,F2
SF    0(R1),F4
ADDF  F16,F14,F2
SF    -4(R1),F8
ADDF  F20,F18,F2
SF    - 8(R1),F12
SF    -12(R1),F16
SUBI  R1,R1,#20
BNEZ  R1,LOOP
SF    4(R1),F20
```

17 instructions
x 4B each
= 68B

```
LF    F0,0(R1)         NOP
LF    F6,-4(R1)        NOP
LF    F10,-8(R1)       ADDF F4,F0,F2
LF    F14,-12(R1)      ADDF F8,F6,F2
LF    F18, -16(R1)     ADDF F12,F10,F2
SF    0(R1),F4         ADDF F16,F14,F2
SF    -4(R1),F8        ADDF F20,F18,F2
SF    - 8(R1),F12      NOP
SF    -12(R1),F16      NOP
SUBI  R1,R1,#20        NOP
BNEZ  R1,LOOP          NOP
SF    4(R1),F20        NOP
```

12 (long) instructions
x 8B each
= 96B

VLIW - Problémy

- Velikost programu – synchronizační NOPy jsou u VLIW navíc
  => nějaký mechanismus komprese / dekomprese VLIW instrukcí
  je užitečný

- Některé závislosti nejsou staticky rozpoznatelné v době překladu (např. aliasing problém ukazatelů, oddělený překlad, dynamické linkování) => překladač musí být konzervativní a výslená výkonnost je nižší než při schopnosti rozpoznat všechny závislosti

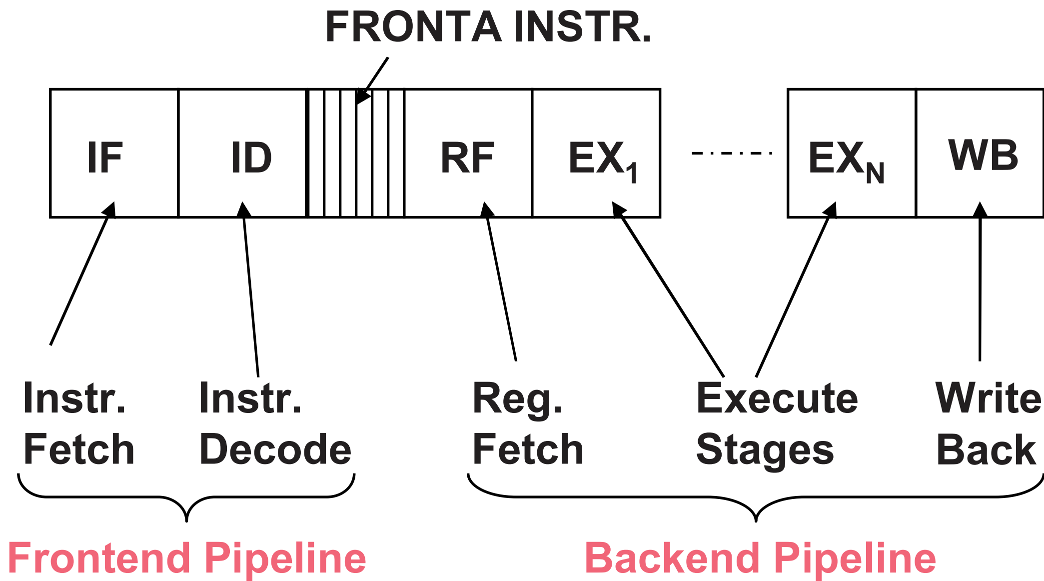
- Některé latence nejsou odhadnutelné v době překladu
  např. latence load/store (cache hit/miss)
  => Některé VLIWy používají softwarově řízenou lokální paměť místo datové cache, jinde překladač předpokládá 100% hit-rate …

- VLIWy nejsou zpětně softwarově kompatibilní (ale existují cesty …)

VLIWy jsou používány jako specializované procesory ve vestavných systémech (např. DSP TI320C6xx, media procesory – Philips TM3200)

VLIWy v univerzálních počítačích nebyly dosud tak úspěšné (Transmeta Crusoe, Intel IA-64 EPIC - Itanium)

Predikce Skoku



U SUPER-SUPER procesorů je problém s latencí skoku. Nelze to vyřešit rozšířením *delay slotu* (to je častým řešením u VLIW) a proto se používá PREDIKCE skoku.
Frontend načítá instrukce podle predikce a ukládá je do fronty. Backend provádí instrukce a verifikuje predikci. V případě chybné predikce vypláchneme všechny chybně načtené instrukce z pipeline (včetně zmíněné fronty)
Penalizace za chybnou predikci skoku = délka pipeline

Dynamická predikce skoku

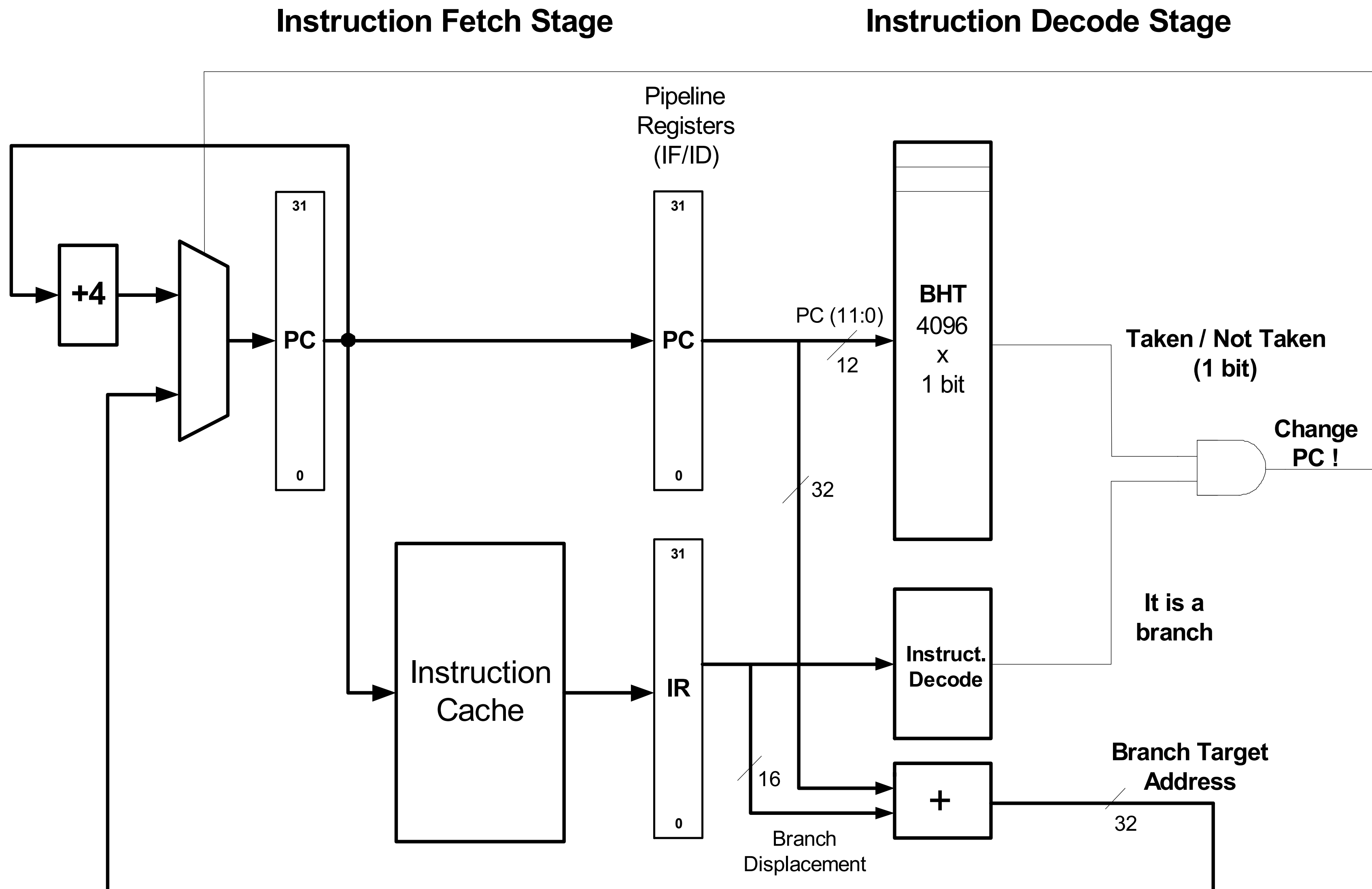
- Predikce může být "*Statická*" (v době překladu, zakódována v instrukci) nebo "*Dynamická*" (za běhu v HW)

- Predikce je užitečná pouze pokud *cílová adresa skoku* je známa dříve než výsledek podmínky
(to platí pro nejčastější PC-relativní skoky)

- Je dynamická predikce skoku lepší než statická ?
  - Zdá se, že ano. Nejlepší statická predikce skoku na bázi profilování kódu dosahuje přesnosti kolem 90 % pro FP programy a 70 - 90 % for celočíselné programy (měření SPEC). To není dostatečné pro superskalární procesory.
  - Dynamické prediktory často vyžaduje „*zahřívací*" dobu než se stabilizuje na správné predikci. Statická predikce (je-li správná) tuto nevýhodu nemá.

Dynamická predikce je jedinou možností pokud nemůžeme změnit ISA a chceme vyšší výkonnost ze starých programů, které nelze rekompilovat.

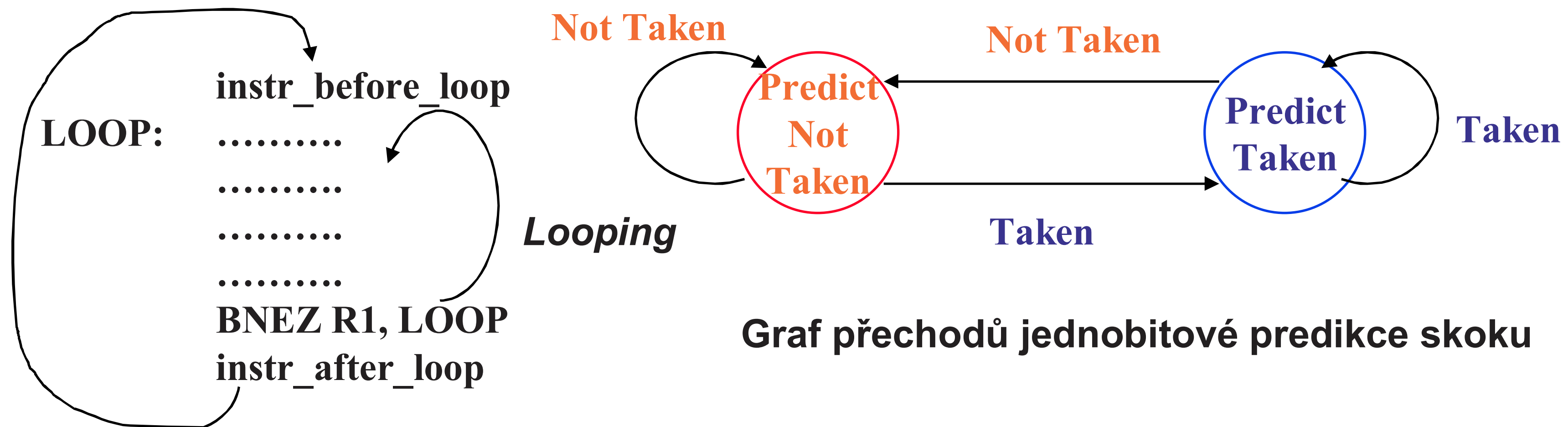
1-bitová Branch History Table (BHT) v Pipeline



1-bitová Branch History Table

- Nižší bity PC adresují tabulku 1-bitových hodnot
  - BHT indikuje zdali skok naposledy byl nebo nebyl proveden
  - Kompletní adresa PC není kontrolována u BHT (s chybnou predikcí se počítá)

- Problém: v cyklu, 1-bitová BHT bude mít dvě chybné predikce (průměrný cyklus má 10 iterací => 20 % neúspěšnost predikce):
  - V poslední iteraci (BHT predikuje pokračování cyklu místo ukončení)
  - V první iteraci když cyklus provádíme znovu (BHT predikuje ukončení cyklu místo pokračování)



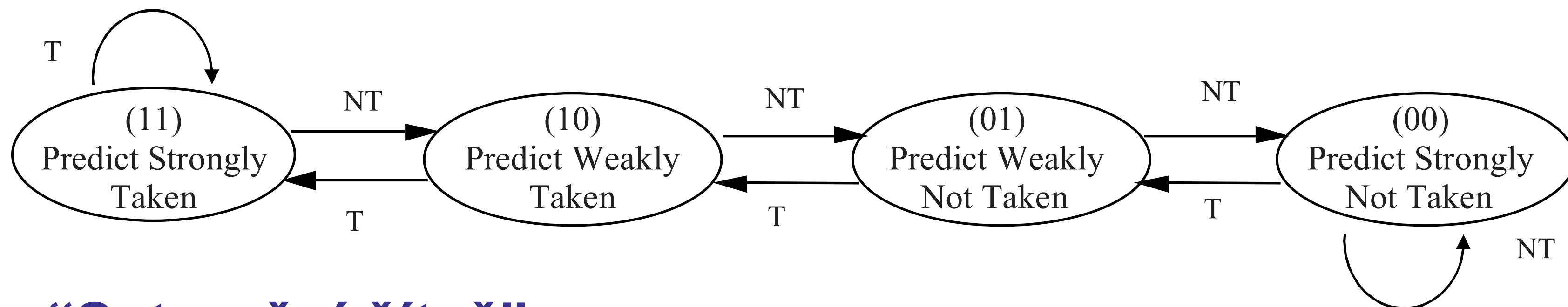
Graf přechodů jednobitové predikce skoku

Next time executing this code including loop

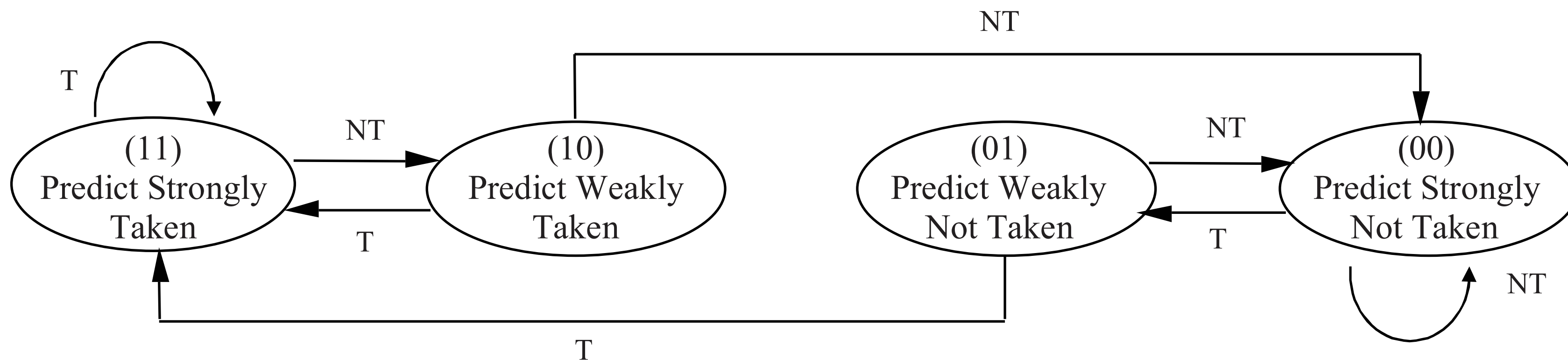
2-bitová Branch History Table

- **Řešení: 2-bitová schemata kde se predikce mění až po *dvou* chybných predikcích**

- **Dvě implementační varianty (výkonnost podobná).**



"Saturační čítač"



"Hystereze"

Nevýhoda 2-bitového prediktoru vůči 1-bitovému - delší doba „zahřívání"

Shrnutí dynamické predikce skoků

- 2-bitový prediktor je základem složitějších technik predikce využívající lokální a globální historii skoků
- Nevýhoda predikce skoku je nutné zpoždění kvůli dekódování instrukce a výpočet cílové adresy skoku (ztrácíme i při úspěšné predikci nejméně jeden takt a neumíme předpovídat *počítané skoky* – instrukce JR)
- Agresivnější technikou je predikce cílové adresy skoku založené na Branch Target Address Caches (BTAC), Branch Target Instruction Caches (BTIC) a Return Address Stack (RAS) – umožňuje implementovat tzv. *zero cycle branches*
- Pokročilé procesory kombinují několik technik predikce k docílení více než 90 % úspěšnosti predikce skoku

=> Vysvětlení BTAC, BTIC a RAS je v doplňkovém výukovém materiálu (pro zájemce).