

9.7.2020

1) Protože obsluha konkrétních požadavků je sekvencí, ~~toz~~ a systém je málo zatížený, tak se doba odezvy nemění.

Doba odezvy by byla rychlejší jen kdyby systém zpracovával nějakým způsobem požadavky paralelně pomocí dostupných procesorů.

Propustnost se zvýší, protože dokážu zpracovat více požadavků najednou.

2) Nasobení bude 5x rychlejší, tzn. zabere $\frac{0,6}{5} = \frac{3}{25} s$

Zbylé operace budou trvat furt stejně, tedy 0,4 s

Celkově 1 krok simulace nově zabere

$\frac{3}{25} + 0,4 = \frac{13}{25} s$, tedy zhruba půl sekundy.

3) Dekodér:



Pravdivostní tabulka: dejme tomu, že $S=0$ aktivuje AND pro X_0

x_0	x_1	S	y_s
0	0	0	0
0	1	0	0
1	0	0	1
1	1	0	1
0	0	1	0
0	1	1	1
1	0	1	0
1	1	1	1

str. 1

5) V případě jednocyklového procesoru bude 1 tch hodinového signálu muset trvat nejméně
 $200 + 150 + 120 + 190 + 140 = 800 \text{ ps}$, aby se stihla vykonat nejdelší instrukce

Zřetězená datová cesta bude mít takty dlouhé 200 ps , aby se pokaždé stihla vykonat nejdelší stage.

6) Vytřídění bude 30%. Zapisuju do registrů, když načítám z paměti (lw)

7)

add \$1, \$5, \$3

sw \$1, 0(\$2)

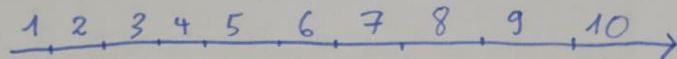
add \$2, \$2, \$3

beq \$2, \$4, Label1

sw \$1, 0(\$2) } zneplatní se

add \$2, \$2, \$3 } v taktu 6

add \$5, \$5, \$1



IF ID EX MA WB

IF ID EX MA WB

IF ID EX MA WB

IF ID EX MA WB

IF ID EX MA WB

IF ID EX MA WB

IF ID EX MA WB

V taktu 6 se zjistí, že skok nebude

2 instrukce se tedy musí zneplatnit
 další

V taktu 7 se načte instrukce add \$5, \$5, \$1

8) V 1. cyklu se trefím pouze jednou (prostřední T)

$$0 \xrightarrow{T} 1 \xrightarrow{T} 2 \xrightarrow{T} 3 \xrightarrow{NT} 2 \xrightarrow{NT} 1$$

V každém dalším cyklu trefím dvakrát T

$$1 \xrightarrow{T} 2 \xrightarrow{T} 3 \xrightarrow{T} 3 \xrightarrow{NT} 2 \xrightarrow{NT} 1$$

Tedy pokud pomínu 1. cyklus, tak přesnost predikce je $\frac{2}{5}$, což je 40% pro tento cyklus skoků.

9) Protože informace je uchována v kondenzátoru, takže se časem (~~ztr~~) ztratí. pomocí náboje

10) Znamená, že data, která byla nedávno použita, se nejspíše použijí znova.

Když si zaplním cache a mám se rozhodnout, který cache line nahradit, tak použiju LRU

1) Když mám moc malou stupeň asociativity, tak se může často stát, že musím zahodit již načtená data, protože potřebuju data, která mají stejný index, ale jiný tag. Bude tedy často docházet ke conflict miss.

Zvýšením asociativity se sníží miss rate, ale (přesněji) cache bude mít velký overhead. Potřebuju nějakou logiku, abych si vybral správná data. Zvyší se mi tedy t_{hit} .

12) Označím si moje cache C1, C2, C3

C1:

~~První~~ Průběh mapovaná (R0)

	0	4	8
	1	5	
	2	6	
	3	7	

Řekněme, že (R0) indexy paměťových bloků 0, 4, 8 spadají do první cache line

Bloky 1, 5 do druhé; 2, 6 do třetí a 3, 7 do čtvrté

První přístup:
cold miss

Druhý přístup:
cold miss
M[0] je vyhozen

Třetí přístup:
conflict miss

Čtvrtý přístup:
cold miss

Patý přístup:
conflict miss

C2:

2-cestná

		0	2	4	6	8
		1	3	5	7	

Řekněme, že první line je pro bloky 0, 2, 4, 6, 8 a druhý pro zbylé

4) Datová cesta musí porovnat obsahy 2 registrů.
(ALU je odečte od sebe a ověří, zda výsledek je 0)

Mezitím se (~~z~~) Branch Addr rozšíří, abych
měl platnou 32b adresu.

Pokud výsledek porovnání (rozdíl) je 0, tak se aktivuje
signál pro skok

(~~8~~) Řídící signály:

ALUSrc = 1 : potřebuji v ALU obsahy 2 registrů

ALUOp musí být nastavené na odečítání

Branch = 1

Jump = 0, abych použil správnou adresu

Zbylé signály mě nezajímají