

1. Procesor DLX byl implementován ve dvou variantách (níže). Určete teoretické zrychlení proudově pracujícího procesoru daných parametrů, je-li CPI proudově pracujícího procesoru ideální, tedy nedochází k pozastavování (stalls). [2 body]

1. Jedno cyklový (single-cycle CPU), $T_{clk}=200ns$, $CPI=1$
2. Proudově pracující (pipelined), rozdělen do 7 stupňů, jejichž zpoždění kombinační části je $t_{IF1}=30ns$, $t_{IF2}=20ns$, $t_{ID}=40ns$, $t_{EX}=30ns$, $t_{MEM1}=25ns$, $t_{MEM2}=25ns$, $t_{WB}=30ns$.

Ideální CPI = 1

$T_{clk2} = 40$ (největší)

$T_{cpu} = T_{clk} * IC * CPI = 40 * IC * 1 = 40 * IC$

$$Zrychlení = \frac{IC * CPI * TCLK1}{IC * CPI * TCLK2} = \frac{IC * 1 * 200}{IC * 1 * 40} = 5$$

2. Napište program pro počítač s akumulátorově-orientovaným souborem instrukcí (accumulator-oriented ISA), který bude vyhodnocovat následující výraz: $X=A+B*(C-D)$. (instrukce používají absolutní adresaci. V programu použijte A,B,C,D a X jako symbolické adresy proměnných v paměti.) [2 body]

```
LOAD [C]
SUB [D]
MUL [B]
ADD [A]
STORE [X]
```

předpoklad implementace SUB: store(load - arg)

3. Paměť s prokládaným cyklem a její souvislosti s vektorovými počítači. Jak tato paměť funguje, a proč ji s výhodou využíváme u vektorových počítačů? [2 body]

Paměť s prokládanými cykly (interleaved memory) se používá ve výkonných počítačích ke snížení cyklu paměti. Hlavní paměť je rozdělená do několika bloků (bank) pracujících samostatně a schopných provádět nezávisle čtecí nebo zápisový cyklus. Jestliže procesor komunikuje střídavě s různými paměťovými bloky, mohou přenosy probíhat paralelně a komunikace mezi pamětí a procesorem se celkově zrychlí. Doba potřebná k realizaci jedné komunikace mezi blokem a procesorem se nazývá cyklus bloku. Systém je vyvážený, jestliže je paměť rozdělená alespoň do tolika bloků, kolik taktů trvá jeden cyklus bloku. Znamená to, že procesor může v každém taktu provést jeden přístup do paměti. Cyklus bloku trvá ve vektorových počítačích obvykle 4 nebo 8 taktů, takže stačí, aby paměť byla rozdělená do 4 nebo 8 bloků.

Vektorové prvky mají známy formát přístupu do paměti $M \rightarrow$ prokládaná paměť M pracuje efektivně(nahrazuje cache).

4. Charakterizujte superpipeline procesor. Srovnajte ho z hlediska výkonnostní rovnice se superskalárním procesorem. [2 body]

Superpipeline procesor - paralelismus v čase - dlouho trvající operace (například F a M) se rozdělí na více částí, aby se zkrátila doba jejich vykonávání (tím pádem se zkrátí T_{clk} ve výkonnostní rovnici). Musí se však dávat větší pozor na hazardy, kterých může vznikat více. Zvýší se také Hit time a Miss penalty.

Superskalární procesor - oproti tomu zvýší počet instrukcí za takt (dělá jich několik najednou), takže se sníží CPI, ale opět hrozí více stallů.

5. Doplněte do vět MR, MP, MAPI a HT, tak, aby byly pravdivé:

- a. Za jinak stejných podmínek se zvětšením kapacity cache sníží ale zvýší
- b. Za jinak stejných podmínek se při zvětšování velikosti bloku snižuje a zvyšuje ale při růstu velikosti bloku nad určitou mez se zvyšuje i [2 body]

a) MR, HT

b) MR, MP, MR

6. Uvažujte celočíselný proudově pracující procesor DLX. Doplněte do šablony časový průběh vyhodnocování instrukcí v 7-superpipelined procesoru a zodpovězte otázky. Procesor má stejné stupně proudového zpracování jako klasický 5-stupňový DLX s tím, že přístup do instrukční a datové paměti byl rozdělen do 2 stupňů (jinak by zpoždění těchto stupňů omezovalo hodinovou frekvenci procesoru). Stupně proudového zpracování jsou následující: F1 (začátek přístupu do instrukční cache), F2 (dokončení přístupu do instrukční cache), D (dekódování a čtení operandů z regist za předpokladu nalezení položky v instrukční respektive datové cache).

- a. Definujte z jakých stupňů pipeline do jakých stupňů pipeline je vhodné vést forwarding. (Pipeline registry na rozhraní Alpha a Beta považujeme za součást stupně Beta, například registr oddělující M1 od M2 považujeme za součást stupně M2.) [2 body]

M1→E

M2→E

M1→D

M2→D

- b. Naznačte časový průběh vykonávání programu s použitím Vámi definovaného forwardingu za předpokladu ideální instrukční i datové cache. Instrukce BNEZ je normální podmíněný skok, podmínka je vyhodnocena a PC změněno ve stupni D. Obsah registru zapsaný do registrového pole v rámci stupně W může být zároveň přečten závislou instrukcí ve stupni D. Registr R3 obsahuje hodnotu 4. Všechny hazardy jsou rozpoznány ve stupni D. [2 body]

Clock cycle	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Start: LW R5, 0(R4)	F1	F2	D	E	M1	M2	W									
ADD R4, R5, R4		F1	F2	D	D	D	E	M1	M2	W						
SUBUI R3, R3, #1			F1	F2	F2	F2	D	E	M1	M2	W					
NOP				F1	F1	F1	F2	D	E	M1	M2	W				
BNEZ R3, LAB2							F1	F2	D	E	M1	M2	W			
ADD R4, R5, R4								F1	F2							
ADD R4, R5, R4									F1							
LAB2: BNEZ R3, LAB2										F1	F2	D	E	M1	M2	W

Forwarding W → E M1→E a pro BNEZ M1 → D

- c. Srovnějte výkonnost této modifikace procesoru DLX s procesorem DLX implementujícím klasickým celočíselným 5-stupňovým pipeline (Fetch, Decode, Execute, Memory, WriteBack) a forwardingem. Oba procesory jsou implementovány ve stejné technologii.
- i. Ke srovnání použijte výkonnostní rovnici procesoru (CPU performance equation) pro Tcpu. Rozeberte, jak jsou jednotlivé složky této rovnice ovlivněny přechodem z 5-stupňového na výše uvedený 7-stupňový pipeline a proč. [2 body]
 - ii. Za jakých podmínek okolností může být procesor s 5-stupňovým pipeline rychlejší než se 7-stupňovým pipeline? [2 body]
 - iii. Uveďte kolik činí u obou procesorů Use Delay a Branch Penalty (za předpokladu ideální instrukční a datové cache. [2 body]
 - 1. 5-stupňový pipeline:
 - 2. 7-stupňový pipeline:

- i) $T_{cpu} = IC * CPI * T_{clk}$
 T_{clk} klesne, protože složitější operace se rozdělí do kratších a časování záleží na tom nejdelším úseku
CPI stoupne, protože může docházet k více stallům nebo může dojít na strukturální hazardy
- ii) Když se blbě rozdělí pipeline (vždy se čeká na tu nejpomalejší fázi) takže když se rozdělí ta nejrychlejší fáze na dvě ještě 2x rychlejší, tak to bude pomalejší.
- iii) 5 stupňová pipeline - Use delay = 1, Branch penalty = 1
7 stupňová pipeline - Use delay = 2 (navíc takt u M), Branch penalty = 2 (navíc takt u F)

7. V paměťovém systému o velikosti fyzické paměti 2 GB organizované po bytech je skrytá paměť (cache) o kapacitě 128 KB s velikostí bloku BS = 128 B. Z cache je čteno zarovnané slovo W = 4 B, stupeň asociativity je S = 4, strategie výběru obětované položky je random, strategie zápisu je write-back. [10 bodů]
- a. Rozvrhněte adresu hlavní paměti do částí sloužících k adresaci skryté paměti.
 - b. Nakreslete blokové schéma skryté paměti včetně datových a řídicích cest pro čtení.
 - c. U každé cesty vyznačte její šířku v bitech.

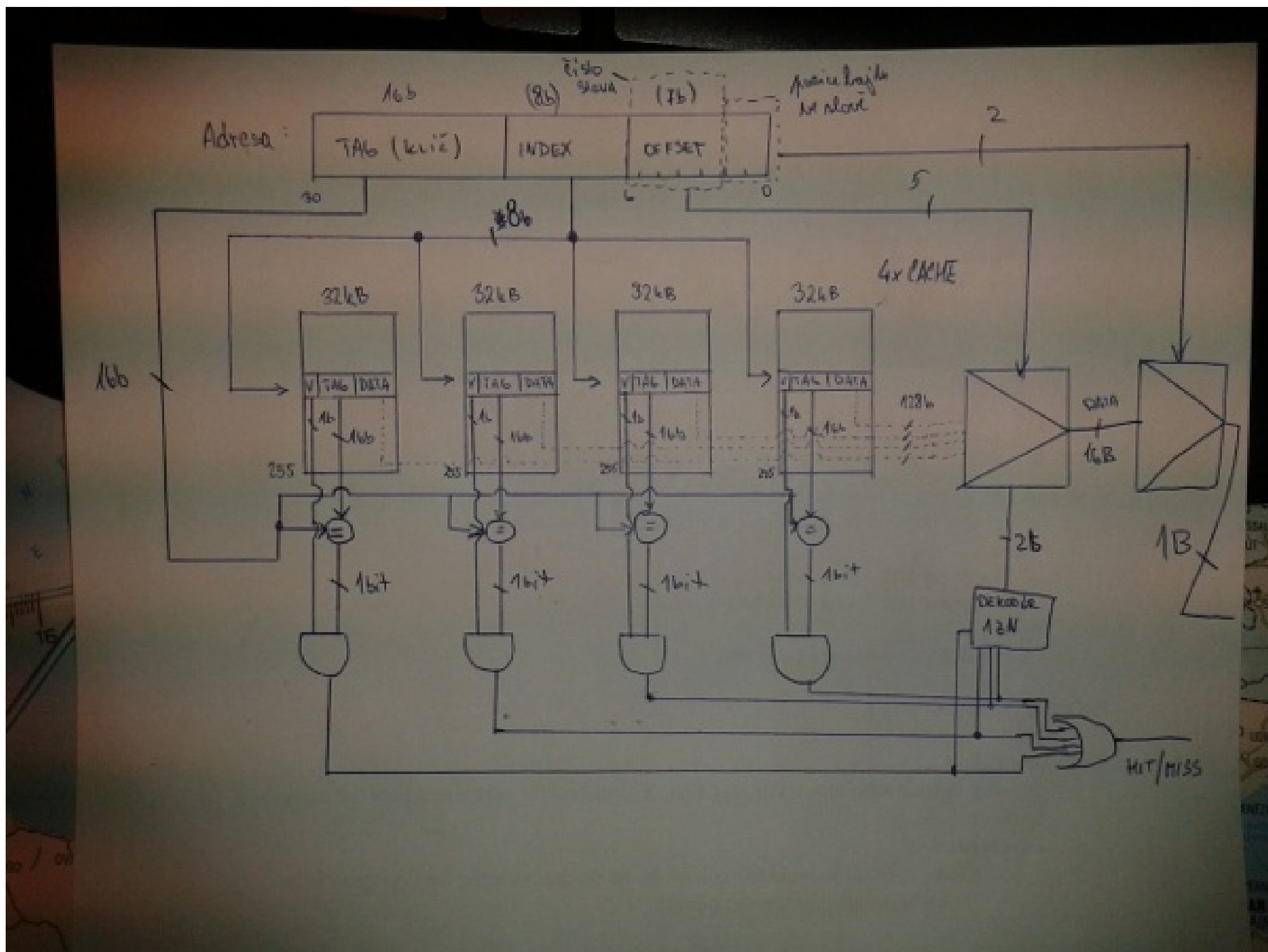
a) b) c)

Paměť = 2 GB = 2^{31}
Cache = 128 KB = 2^{17}
BS = 128 B = 2^7
W = 4 B = 2^2
S = 4 = 2^2

Adresa - 31 b (mocnitel u paměti)

Offset - 7 b x
]
- 7b (mocnitel u BS)
- 2 b (mocnitel u W)
- 5 b

$$\text{Index} - \frac{2^{17}}{2^7} = 2^{10} \Rightarrow \frac{2^{10}}{2^2} = 2^8 = 8 \text{ b}$$



UPDATE k obrázku 17.1.2013 Podľa tohto obrázku sa to úplne neučte, tie multiplexory na konci nie sú úplne v poriadku (povedal Hlaváč na ústnej). Ideálne tam dajte len jeden a neposielať von 1B, keď je $W=4B$ atď. atď.. bity aj sety sú fajne (tuším ešte, keď je Write-Back, tak musí v sete okrem V, tagu, a dat byť aj dirty bit :-* ... (ak tomu niekto rozumiete viac, tak upresnite dôvody aj správne riešenie, pomôžete spolužiakom)

~ Pokiaľ si to dobre pamätajú z ústní: buď pôjde do prvého multiplexoru len drát z dekodéru a do druhého číslo slova v bloku, alebo tam bude len jeden s dráty z dekodéru a z čísla slova v bloku, pak je ale potreba pripísať, že sa využije špeciálny multiplexor, ktorý si správne sklobí tyto dva vstupy. Z té pozice bajtu ve slově nic věst nemá. A ano, šířka výstupu na datovém drátu je ta velikost slova W ze zadání.

d. Napište výkonnostní rovnici (T_{cpu}) zohledňující vliv paměťového subsystému. Jak jsou jednotlivé složky této rovnice ovlivněny parametry skryté paměti (kapacita, vel. bloku, stupeň asociativity,...)

$$T_{cpu} = IC * (CPI + MAPI * MR * MP) * T_{clk}$$



Menší kapacita

- menší HT
- větší MR

Větší velikost bloku

- menší MR

Větší stupeň asociativity

- menší MR-

e. Vysvětlete pojmy povinný výpadek (compulsory miss), kapacitní výpadek (capacity miss) a kolizní výpadek (collision miss).

povinný výpadek (compulsory miss) - nebo také „studený výpadek“, při startu PC nebo po spuštění nového programu je cache prázdná a musí se tam data nahrát.

kapacitní výpadek (capacity miss) - způsoben omezenou velikostí cache (už se do ní nic nevejde a musí se něco vyměnit), u plně asoc. cache

kolizní výpadek (collision miss)/konfliktní výpadky(conflict missed)- 2 a více rozdílných adres jsou mapovány na stejné místo v cache. (stejný index)

Oprava

8. Popište implementaci zámku (Lock & Unlock routines) pomocí atomické instrukce C&S (porovnání registru s hodnotou v paměti a případné prohození obsahu jiného registru a tohoto paměťového místa). Proveďte analýzu počtu transakcí na sběrnici (bus transactions) v systému SMP s p=4 procesory se sdílenou sběrnici a MESI koherenčním protokolem za předpokladů uvedených níže. V analýze uveďte přesný počet sběrnicových transakcí daného typu pro systém s p=4 procesory od okamžiku odemčení zámku procesorem P1, dokud poslední uspokojený procesor P2-P4 zámek neodemkne. Vysvětlete funkci každého typu sběrnicové transakce (uvedení jména resp. zkratky nestačí).
- a. Na začátku procesor P1 odemkne zámek, který je před touto operací ve všech cache ve stavu S (shared) jako zamčený.
 - b. Každý z procesorů P2-P4 usiluje o vstup do kritické sekce, dokud není uspokojen, po opuštění kritické sekce odemkne zámek a dále již se o jeho hodnotu nezajímá.
 - c. Když je jeden z procesorů v kritické sekci, každý z ostatních čekajících procesoru stihne jeden neúspěšný pokus o vstup, než je tato sekce opuštěna (odemčena).

[10 bodů]/[15 bodů]

C&S MESI	\$P1		\$P2		\$P3		\$P4		M	BUS
	slav	daba	slav	daba	slav	daba	slav	daba		
P1: Unlock ST	S	1	S	1	S	1	S	1	1	
P1: Unlock ST	M	0	I		I		I		1	P1: Bus Upgr S=0
P2: C&S [Pr Rd Pr Wr]	S	0	S	0	I		I		0	P1: Cache flush; P2: Bus Rd S=1
	I		M	1	I		I		0	P2: Bus Upgr
P3: C&S Pr Rd	I		S	1	S	1	I		1	P3: Bus Rd S=1; P2: Cache flush
P4: C&S Pr Rd	I		S	1	S	1	S	1	1	P4: Bus Rd S=1; M: Mem Sup
P2: Unlock ST	I		M	0	I		I		1	P2: Bus Upgr
P3: C&S [Pr Rd Pr Wr]	I		S	0	S	0	I		0	P3: Bus Rd S=1; P2: Cache flush
	I		I		M	1	I		0	P3: Bus Upgr
P4: C&S Pr Rd	I		I		S	1	S	1	1	P4: Bus Rd S=1; P3: Cache flush
P3: Unlock ST	I		I		M	0	I		1	P3: Bus Upgr
P4: C&S [Pr Rd Pr Wr]	I		I		S	0	S	0	0	P4: Bus Rd S=1; P3: Cache flush
	I		I		I		M	1	0	P4: Bus Upgr
P4: Unlock ST	I		I		I		M	0	0	P4: Bus Upgr

Compare and swap - porovná obsah specifikované paměťové adresy addr s očekávanou hodnotou exp a v případě rovnosti nahradí obsah paměťové adresy novou hodnotou val. O úspěchu či neúspěchu informuje uživatele návratovou hodnotou. Celá procedura proběhne atomicky

Postup při přístupu ke sdíleným datům

- Přečtu stávající hodnotu sdíleného objektu
- Připravím novou hodnotu sdíleného objektu

- Aplikují instrukci C&S

Navratová hodnota

- **True** - Objekt nebyl v mezičase modifikován, nově vypočtena hodnota je platná a je uložena ve sdíleném objektu.
- **False** - Objekt byl v mezičase modifikován (z jiného vlákna), instrukce CAS neměla žádný efekt a je nutné celý postup opakovat.

```
int compare_and_swap(int *reg, int oldval, int newval) {
    ATOMIC();
    int old_reg_val = *reg;
    if (old_reg_val == oldval)
        *reg = newval;
    END_ATOMIC();
    return old_reg_val;
}
```

9. RAW, WAR, WAW hazardy, příklady.

RAW - mezi operandy je datová závislost

```
LW  R1, (R2)
ADD R3, R4, R1
```

WAR - druhá instrukce chce zapsat dříve, než ji první přečte

U klasického DLX nemůže nastat
Instrukce 1 čte data, instrukce 2 zapisuje.
Hazard nastává, když instr 2 zapíše dřív, než instr 1 data dočte.

WAW - druhá instrukce chce zapsat dříve než první

```
MULTF F1, F2, F3
ADDF F1, F2, F3
```

MULTF zapíše do F1 až po ADDF, protože trvá víc taktů.

10. Cache coherence, snooping, formální vysvětlení.

S nástupem systémů s více procesory a společnou hlavní pamětí (SMP) (**Symmetric multiprocessing**) se vyskytl problém coherence cache. Aby se předešlo problémům s konzistencí, musí SMP splňovat následující podmínky

- Paměťové operace každého procesu musí být vykonávány ve stejném pořadí, v jakém byly daným procesem spuštěny
- Hodnoty získané při všech operacích čtení musí odpovídat posledním hodnotám operace zápisu na dané místo v paměti

Snooping

Jedná se o metodu, kdy každá cache monitoruje adresní sběrnici, zda není přistupováno na adresu v paměti, jejíž kopie je v cache uložena. Pokud je zjištěna operace zápisu na adresu, jejíž kopie je uložena v cache, dojde k zneplatnění těchto dat.

Protokol WTWNA (**Write Through Write Non Allocate**)

Jedná se o protokol založený na snoopingu. Cache sleduje adresní sběrnici, a pokud zjistí zápis na adresu, z níž má v sobě uloženu kopii dat, zneplatní ji. Při pokusu o čtení neplatných dat jsou data znovu načtena z hlavní paměti. Jelikož se jedná o Write-Through protokol, jsou již znovu načítaná data vždy platná.

MESI

Používá metodu zápisu Write-Back. U tohoto protokolu jsou na každé řádce v paměti přítomny dva bity navíc. Tyto bity reprezentují jeden ze čtyř stavů, ve kterých se může jedna řádka v cache nacházet při použití protokolu MESI.

M – Modified – znamená, že daný blok se nachází pouze v dané cache a že byl změněn oproti hodnotě v hlavní paměti a že jej bude potřeba v budoucnu uložit do hlavní paměti

E – Exclusive – znamená, že tento řádek obsahuje pouze daná cache a že je nezměněn oproti hodnotě v hlavní paměti

S – Shared – znamená, že daný řádek je uložen ve více caches

I – Invalid – určuje, že daný řádek není platný

Množina stavů a přechodů. http://upload.wikimedia.org/wikipedia/commons/c/c1/Diagrama_MESI.GIF :

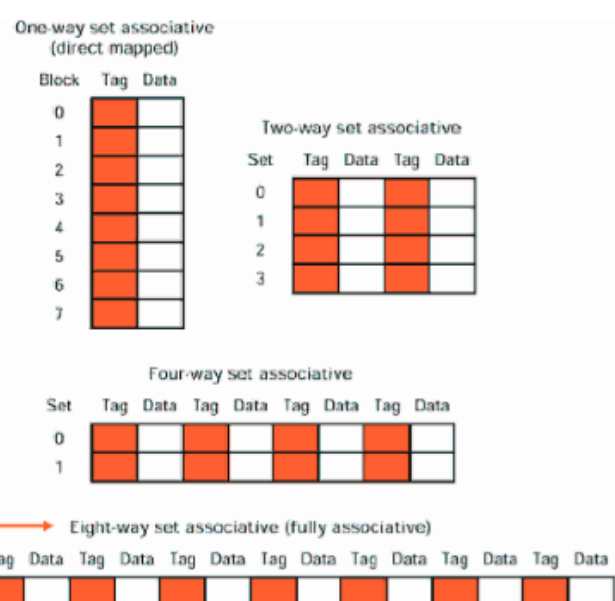
PUSH A
SUB
POP X

předpoklad implementace SUB: push(pop1 - pop2)

12. Cache velikosti 128 KB a BS = 128 B (přibližně) [2 body]
- Jaká bude asociativita u přímo mapované cache?
 - Jaká bude asociativita u plně asociativní cache?
 - Co to je asociativita?

- a) z definície priamo mapovanej cache, asociativita $A = 1$
- b)
- kolik bloků je v setu, teda $A = \text{CacheSize} / \text{BlockSize}$
 $CS = \text{CacheSize} = 128\text{KB} = 2^{17}\text{B}$
 $BS = \text{BlockSize} = 2^7\text{B}$
 $A = \text{Asociativita} = CS / BS = 2^{17} / 2^7 = 2^{10} = 1024$
- c) Vyjadruje na kolko roznych miest moze byt jedna adresa namapovana.

Příklad organizace cache 8 bloková cache



13. Co je to interní a externí fragmentace paměti, plus příklady, kde se vyskytuje. [2 body]

Interní fragmentace

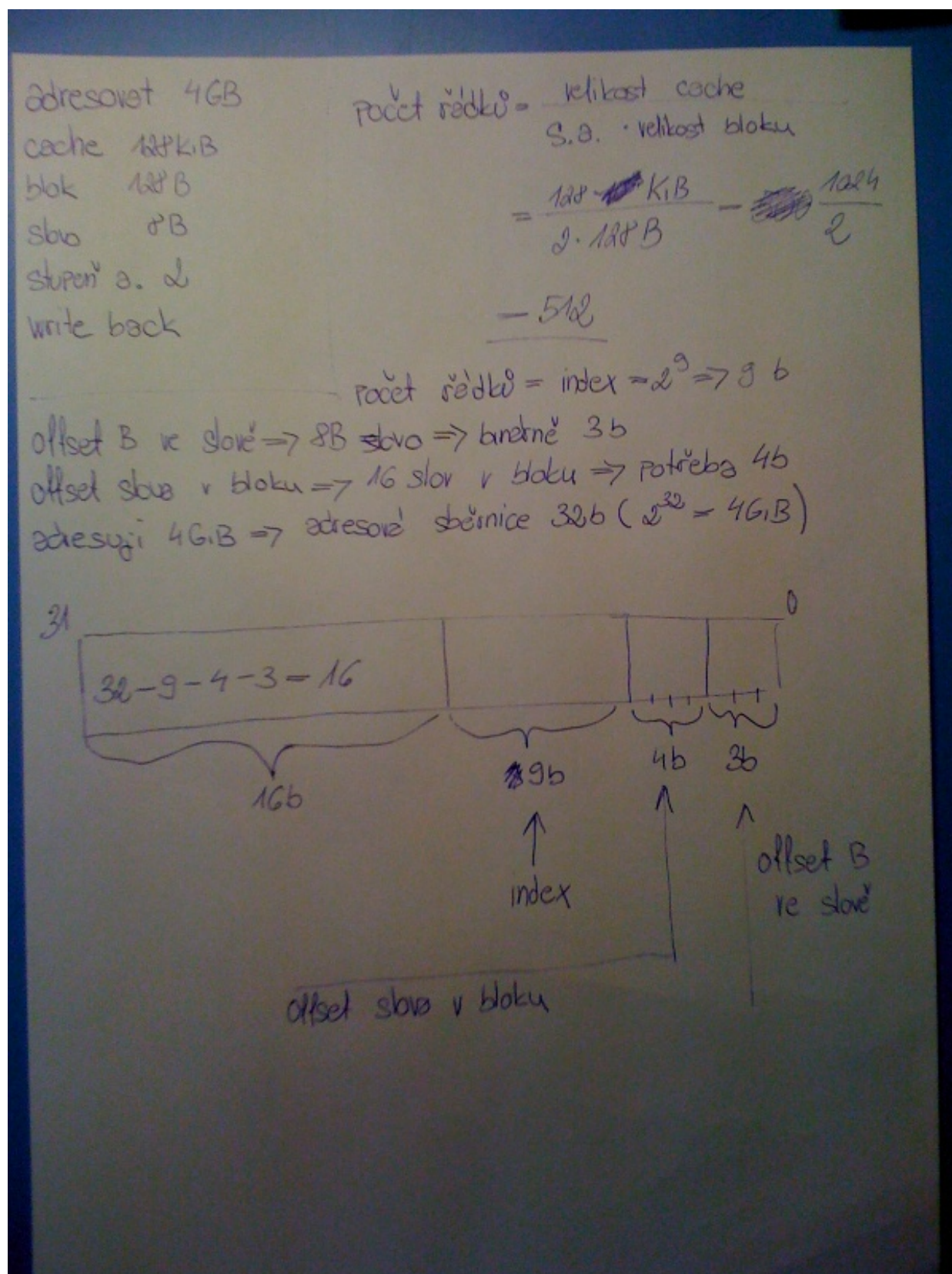
- typická pro stránkování
- alokovaná část paměti může být větší než je požadováno

Externí fragmentace

- typická pro segmentaci
- absolutní paměťový prostor je k dispozici pro splnění požadavků, ale není souvisle obsazován

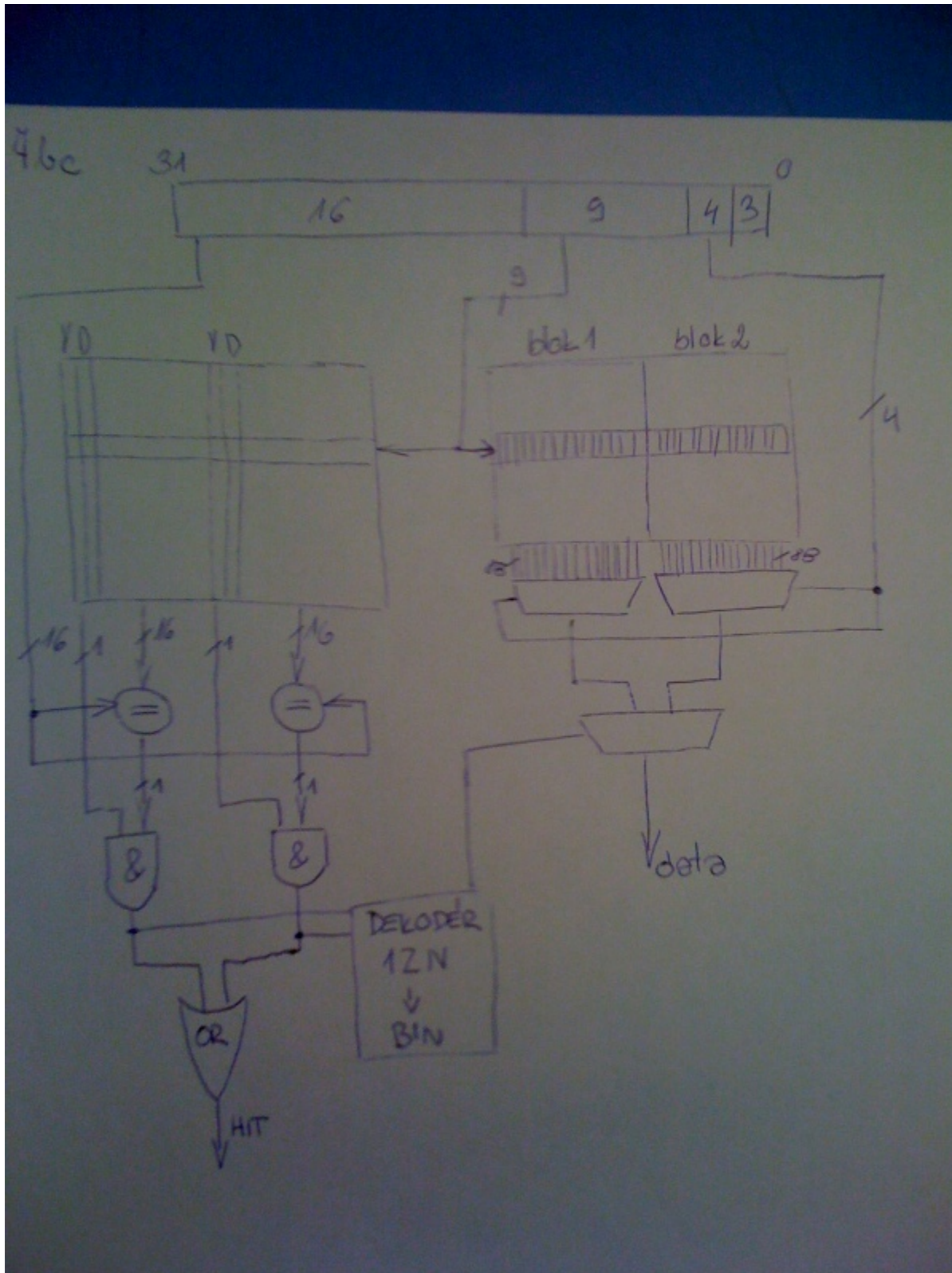
16. V paměťovém systému o velikosti fyzické paměti 4GB organizované po bytech je skrytá paměť (cache) o kapacitě 128KB s velikostí bloku BS=128B. Z cache je čteno zarovnané slovo W=8B, stupeň asociativity je s=2, strategie výběru obětované položky je random, strategie zápisu je write-back. [10 bodů]

- Rozvrhněte adresu hlavní paměti do částí sloužících k adresaci skryté paměti.



- Nakreslete blokové schéma skryté paměti včetně datových a řídicích cest pro čtení.
- U každé cesty vyznačte její šířku v bitech.

b)c)



- d. Napište výkonnostní rovnici (T_{cpu}) zohledňující vliv paměťového subsystému. Jak jsou jednotlivé složky této rovnice ovlivněny parametry skryté paměti (kapacita, vel. bloku, stupeň asociativity,...)
- e. Vysvětlete pojem povinný výpadek (compulsory miss), kapacitní výpadek (capacity miss) a kolizní výpadek (collision miss).

Předchozí příklad.

17. Program běžící na 2 jádrovém CPU běží 1.6x rychleji než 1 jádrovém. Jaké je největší teoretické zrychlení programu máme-li neomezený počet cpu?

$$P = \frac{\frac{1}{1.6} - 1}{\frac{1}{2} - 1}$$

$$S = \frac{1}{1-P} = 4$$

stejně řešení, ale možná snadněji pochopitelné:

$$\text{část, kterou lze urychlit} = P = \frac{\text{čas, který se ušetřil zrychlením}}{\text{čas, který šel maximálně ušetřit}} = \frac{1 - \frac{1}{1.6}}{1 - \frac{1}{2}} = 0.75$$

$$\text{zrychlení} = S = \frac{1}{1-P} = \frac{1}{1-0.75} = \frac{1}{0.25} = 4$$

a co takhle?:

$$S = \frac{1}{\frac{2}{1.6} - 1}$$

a co takhle?

$$S_{\text{overall}} = 1.6 \quad \dots \text{celkové zrychlení}$$

$$S_E = 2 \quad \dots \text{zrychlení vylepšené části}$$

$$F_E = ? \quad \dots \text{velikost zrychlené části } (<= 1)$$

$$S_{overall} = \frac{1}{1 - F_E + \frac{F_E}{S_E}}$$

...

$$F_E = \frac{S_E - S_E \times S_{overall}}{S_{overall} - S_E \times S_{overall}} = \frac{2 - 2 \times 1,6}{1,6 - 2 \times 1,6} = 0,75$$

$$\text{Nekonečně mnoho vláken} \Rightarrow S_E \rightarrow \infty \Rightarrow \frac{F_E}{S_E} \rightarrow 0$$

$$S_{overall} = \frac{1}{1 - F_E + \frac{F_E}{S_E}} = \frac{1}{1 - 0,75 + 0} = 4$$

18. Výhody a nevýhody registrově orientovaná ISA a střadačově orientovanou ISA.

Střadačově orientovaná ISA

- + Jednoduchý HW
- + Rychlé přepínání kontextu
- + Krátké instrukce
- + Jednoduché dekódování instrukcí

- Častý přístup do paměti
- omezený paralelizmus

Zásobníkově orientovaná ISA

- + Jednoduché a rychlé instrukce
- + Programy zabírají málo místa v paměti
- + využívá se pro virtuální ISA (java)

- Chybí možnost náhodného přístupu do paměti

GPR ISA

- + Registry jsou rychlejší než paměť
- + Náhodný přístup k datům
- + Méně častý přístup do paměti

- Složitější překladač
- Omezený počet registrů

19. Statická predikce skoku a jak ji implementujeme do ISA. Doplněte ji do programu a proveďte analýzu úspěšnosti skoků. V kolika procentech se skok provede?

Statická predikce - v době překladu je zakódována v instrukci

- pipeline se chová tak, jako by skok měl vždy proběhnout

```

    ADDUI R1, R0, 4
L1: ADDUI R2, R0, 100
L2: SW    0(R4), R0
    ADDUI R4, R4, 4
    SUBUI R2, R2, 1
    BNEZ  R2, @L2
    SUBUI R1, R1, 1
    BNEZ  R1, @L1

```

vnitřní cyklus: 100x BNEZ, 99x se provede

vnější cyklus: 4x BNEZ, 3x se provede

=> 99*4 + 3 provedení, 100*4 + 4 celkem

procento úspěšnosti = 399/404 = 98.8 %

20. Proč je u proudově pracujícího CPU $CPI_{IDEAL}=1$?

Proudově pracující (pipeline)

Protože ideálně je 1 takt = 1 instrukce => CPI = 1

V pipeline skončí každá instrukce za 1 takt.

21. Co je strukturální hazard. Doplněte instrukce, při kterých nastane.

Kolize sdílením prostředků. Instrukce Load a Instr 3 současně přistupují do paměti.

Instr	1	2	3	4	5	6	7	8
Load	IF	ID	EX	MEM	WB			
Instr 1		IF	ID	EX	MEM	WB		
Instr 2			IF	ID	EX	MEM	WB	
Instr 3				IF	ID	EX	MEM	WB

22. V paměťovém systému o velikosti fyzické paměti 4 GB je plně asociativní skrytá cache o velikosti 128 KB s velikosti bloku BS = 128 B. Z cache je čteno zarovnané slovo W = 4 B, strategie výběru obětavé položky je random, strategie při zápisu je write-back.

- a. Rozvrhněte adresu hlavní paměti do části sloužící k adresaci skryté paměti.
 - b. Nakreslete blokové schéma skryté paměti včetně datových a řídicích cest pro čtení.
 - c. U každé cesty vyznačte její šířku v bitech.
 - d. Napište výkonnostní rovnice (T_{CPU}) zohledňující vliv paměťového subsystému. Jako jsou jednotlivé složky této rovnice ovlivněny parametry skryté paměti (kapacita, velikost bloku, stupeň asociativity, ...).
- $T_{cpu} = IC * (CPI + MAPI * MR * MP) * T_{clk}$

Menší kapacita
- menší HT
- větší MR

Větší velikost bloku
- menší MR

Větší stupeň asociativity
- menší MR

- e. Vystvěte pojmy povinný výpadek, kapacitní výpadek a kolizní výpadek.
Povinný výpadek - při startu počítače, cache je prázdná
Kapacitní výpadek - u plně asociativní cache, data byla odstraněna, protože nová data lze nahrát kamkoliv, při zaplnění cache je tedy nic nechrání
Kolizní výpadek - nastává u přímo mapované cache nebo s nízkými stupni asociativity, data musela být odstraněna, protože na jejich místo byla nahrána nová data, která jinak nelze uložit (kolize s daty, která patří na stejnou adresu)

23. Uvažujte symetrický multiprocessorový systém (SMP) s WTWNA koherenčním protokolem

- a. Formálně definujte koherenční protokol skryté paměti

Protokol WTWNA
Jedná se o protokol založený na snoopingu. Cache sleduje adresní sběrnici, a pokud zjistí zápis na adresu, z níž má v sobě uloženu kopii dat, zneplatní ji. Při pokusu o čtení neplatných dat, jsou data znovu načtena z hlavní paměti. Jelikož se jedná o Write-Through protokol, jsou již znovu načítaná data vždy platná

- b. Dále uvažujme, že v paměti je uložen blok na adresách x, x+1, x+2, x+3. Počáteční hodnota tohoto bloku je {a, b, c, d}.

Do následující tabulky doplňte jak se mění stav bloku v jednotlivých skrytých pamětech v systému s 3 procesory a jaké instrukce se posílají na sběrnici.

BusRd: Po lokálním Read miss, použít S/S – signalizuje, jestli existují sdílené kopie
BusRdX: po lokálním Write miss, žádost o nový blok + zneplatnění
BusUpgr: po lokálním Write hit nad blokem se stavem S (zneplatnění)
MemSup: hlavní paměť dodává do cache požadovaný blok
CacheFlush: zdrojem dat je cache, kde je blok ve stavu. Data jsou vypláchnuta na sběrnici a uložena do hl. paměti i předána žádající cache.

POZOR je to MESI	\$ (P1) Stav	Hodnota	\$ (P2) Stav	Hodnota	\$ (P3) Stav	Hodnota	Hodnota v hl.paměti	Transakce na sběrnici (zdroj:typ transakce)

Počáteční stav	I	-	I	-	I	-	{a,b,c,d}	
P2: PrRd M[x]	I	-	E	{a,b,c,d}	I	-	{a,b,c,d}	P2: BusRd(S=0)
P2: PrWr M[x], e	I	-	M	{e,b,c,d}	I	-	{a,b,c,d}	-
P1: PrRd M[x+1]	S	{e,b,c,d,}	S	{e,b,c,d}	I	-	{e,b,c,d}	P1: BusRd (S=1), P2: CacheFlush
P2: PrRd M[x]	S	{e,b,c,d}	S	{e,b,c,d}	I	-	{e,b,c,d}	-
P1: PrWr M[x+1], f	M	{e,f,c,d}	I	-	I	-	{e,b,c,d}	P1: BusUpgr
P3: PrWr M[x+2], g	I	-	I	-	M	{e,f,g,d}	{e,f,c,d}	P3: BusRdX, P1: CacheFlush

27. Paměťová hierarchie má tuto podobu: Mezi procesorem a hlavní pamětí je umístěna skrytá paměť (L1 cache), která má následující parametry:

HT = 2 ns
MR = 10 %
MP = 1000 ns

a. Vypočtete střední přístupovou dobu AMAT pro uvedený systém s L1 cache

$$AMAT = HT + MP * MR$$

$$AMAT = 2 + 1000 * 0.1$$

$$AMAT = 102\text{ ns}$$

b. Vypočtete střední přístupovou dobu do paměti, přidáme-li mezi L1 cache a hlavní paměť další vrstvu skryté paměti (L2 cache) s parametry:

HT = 5 ns
MR = 1 %
MP = 1000 ns

$$AMAT_{L1} = HT_{L1} + MP_{L1} * MR_{L1}$$

$$MP_{L1} = AMAT_{L2} = HT_{L2} + MP_{L2} * MR_L$$

$$AMAT_{L1\&L2} = HT_{L1} + (HT_{L2} + MP_{L2} * MR_{L2}) * MR_{L1}$$

$$AMAT = 2 + (5 + 0.01 * 1000) * 0.1$$

$$AMAT = 3.5\text{ ns}$$

28. Vysvětlete pojmy inicializační interval a strip mining u vektorových počítačů.

Inicializační interval - Čas potřebný k nastartování dané jednotky. Na konci iniciačního intervalu je již k dispozici jeden výsledek.

Strip mining - Délka vektoru je větší než maximální délka vektoru
- Generuje kód tak, aby délka odpovídala S

29. Proč výkonnost počítače neměříme v MIPS (miliony vykonaných instrukcí za sekundu)? Zdůvodněte na základě výkonnostní rovnice.
7iiiiiiiiii

MIPS závisí na:
- ISA a programu, neboť CPI závisí na programu
- mixu instrukcí
- CISC / RISC procesor

$$MIPS = \frac{f_{CLK}}{CPI*10^6}$$

Důsledek:
- výkonnost vyjádřena v jednotkách MIPS je závislá na programu, i když se tato závislost často neuvádí

30. Pipeline

Pipelining neboli zřetěžené zpracování, či překrývání strojových instrukcí. Základní myšlenkou je rozdělení zpracování jedné instrukce mezi různé části procesoru a tím i dosažení možnosti zpracovávat více instrukcí najednou.

- 1. **Instruction fetch - IF** - čtení instrukce
- 2. **Decode - ID** - dekodování instrukce, zároveň se načítají registry
- 3. **Execute - EX** - provedení instrukce
- 4. **Access - MEM** - přístup do paměti (ukládání do paměti při store, čtení při load)
- 5. **Writeback - WB** - zápis výsledku do registrového pole (při reg->reg operacích nebo při loadu)

32. Celočíselný proudově pracující DLX. Časový průběh 5-stupňové pipeline.
a. Časový průběh. BNEZ normální podmíněný skok, podmínka je vyhodnocena a PC změněno již ve stupni D. Registr zapsaný v W může být přečten v D. Hazardy jsou rozpoznány v D. R3 má hodnotu 4. (forwarding definován dle vlastního uvážení - W->E, M->E, M->D)

Clock cycle	1	2	3	4	5	6	7	8	9	10	11	12
Start: LW R4, 0(R1)	F	D	E	M	W							
LW R5, 0(R2)		F	D	E	M	W↓						
ADD R4, R5, R4			F	Ds	D	E	M↓	W				
SUB R6, R4, R7				Fs	F	D	E	M	W			
SUBUI R3, R3, #1						F	D	E	M↓	W		
BNEZ R3, LAB2							F	Ds	D	E	M	W
SW 4(R29), R1								Fs	F			
SW (R29), R2												
LAB2: ADD R30, R29, R0										F	..	

33. Výkonostní rovnice 5 a 7 stupňová pipeline.

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

T_{CLK} - klesne protože složitější operace rozloží na více stupňů.
CPI - stoupne protože může docházet k více pozastavováním.

34. 5 a 7 stupňová pipeline, F1, F2, D, EX, M1, M2, W. Load use delay, branch penalty.

5 stupňová pipeline - Use delay = 1, Branch penalty = 1
7 stupňová pipeline - Use delay = 2 (navíc takt u M), Branch penalty = 2 (navíc takt u F)

35. 32 b DLXV s jednobránovou pamětí, která je rozdělena do 8 buňek. Šířka banky je 64 b. Latence banky je 6 taktů. Adresové bity A(5:3) určují číslo banky, A(31:6) jsou přivedeny na adresní vstupy každé banky (banky = 512 MB v organizaci 64M x 64 b).(64 bitová čísla jsou nahrazena symbolickými jmény).

A(31:6)	000	001	010	011	100	101	110	111
0.....00000000	a	b	c	D	e	f	g	H
0.....00000001	i	j	k	L	m	n	o	p
0.....00000010	q	r	s	T	u	v	w	x
0.....00000011	y	z	aa	Ab	ac	ad	ae	af
0.....00000100	ag	ah	ai	Aj	ak	al	am	an
0.....00000101	ao	ap	aq	Ar	as	at	au	av
0.....00000110	aw	ax	ay	Az	ba	bb	bc	bd

Dále je dán stav DLXV: R2 = 0x140, R3=0x48, R4=0x20, VLR=0x8

	A(31:6) = adresa v bance	A(5:3) = banka	A(2:0)
R2 = 0x140	1 01	00 0	000
R3 = 0x48	0 01	00 1	000

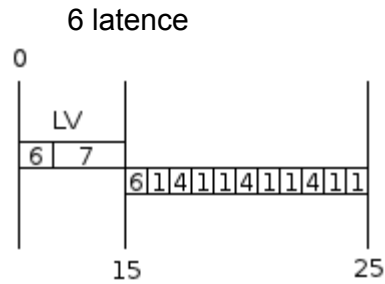
R4 = 0x20	0 00	10 0	000
-----------	------	------	-----

| - oddělovač čtveřic bitů

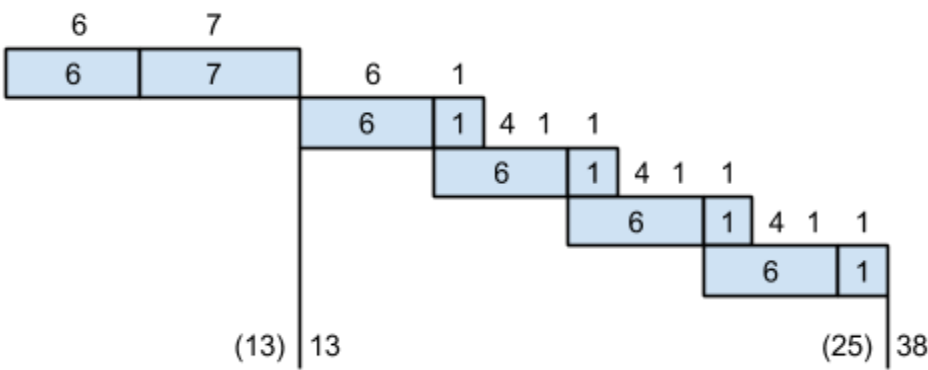
a. Graaficky znázorníte čsový průběh následujících instrukcí a určete délku jejich provádění (v taktech) pro danou konfiguraci paměti a procesoru:

```
LV    V0, ( R2)
LVWS V1, (R3), R4
```

Střída (stride) u instrukce LVWS udána v bytech je v registru R4.



6 latence, pak ihned +1 z jiné banky další položka, ale střídající se banky 001 a 101 (jen dvě), proto se pak musí vždy 4 čekat než je možné je opět použít



b. Určete obsah registrů V0 a V1 po provedení těchto instrukcí

v0	ao	ap	aq	Ar	as	at	au	av
v1	j	n	r	v	z	ad	ah	al

c. Uved'te způsob, jakým může vektorový procesor číst a zapisovat data do řídkých matic

Data řídkých matic lze číst/zapisovat pomocí operací scatter & gather — tedy pomocí instrukcí lvws a svws.

d. Uved'te způsob, jakým může vektorový procesor provádět operace s vybranou složkou vektoru .

Nad vybranými složkami lze pracovat pomocí maskování (instrukce lvi a svi)

e. Uved'te jakým způsobem může vektorový procesor pracovat s vektory delšími než je MVL.

Strip mining

36. Zpožděný skok

Zpožděný skok značí skok, při němž se vždy provede následující instrukce. Je tedy jedno zda se skáče či ne, instrukce po skoku se prostě provede pokaždé. Nepatří mezi predikce skoku.

37. Co jsou to registrová okna?

http://en.wikipedia.org/wiki/Register_window

Části programu je přiřazeno tzv. registrové okno, tj. každá taková část vidí jen registry uvnitř svého okna (registry mimo toto okno nejsou viditelné), aby nedocházelo ke kolizím s jinými částmi programu.

38. Za jakých podmínek je procesor klasicky virtualizovatelný?

Podpora dvoch režimov inštrukcii: privilegované a nepriviligované. Medzi privilegované patria tie, ktoré pracujú s HW, ako maskovanie prerušení, info o HW a tak - v nepriviligovanom režime hodia výnimku.

39. Co označuje zkratka GPGPU?

General Purpose Graphics Processing Unit
- provádění obecných výpočtů pomocí grafických procesorů.

40. Bariéry
<http://msdn.microsoft.com/cs-cz/library/dd537615%28v=vs.110%29.aspx>

Vlákná narazí na bariéru a tu „protlačí“ pouze, pokud je jich nějaký konkrétní počet. Pokud je například na bariéře očekáváno 6 vláken, pak 5 a méně zde čeká, než mohou pokračovat v programu dál.

41. LL SC, pri prvom Locku z P2 sa ma este zmenit hodnota v pamati na 0 lebo je CacheFlush
Navic v LAR je jenom adresa, napr. 0x1ac, nikoli M[...]

Simulujte zámek s LL a SC na MESI.																
Function	Instruction	\$P1				\$P2				\$P3				M[0x1ac]	Bus Transaction	
		State	Data	LAR	LF	State	Data	LAR	LF	State	Data	LAR	LF			
		S	1		0	S	1		0	S	1		0	1	-	
P1:Unlock(0x1ac)	Store M[0x1ac],#0	M	0			I				I					P1:BusUpgrade	
P2:Lock(0x1ac)	LL	S	0			S	0	M[...]	1						P2:BusRd s=1, P1:CacheFlush	
P3:Lock(0x1ac)	LL									S	0	M[...]	1		P3:BusRd s=1, MemSup	
P2:Lock(0x1ac)	SC	I				M	1			I			0		P2: BusUpgrade	
P3:Lock(0x1ac)	SC														---	
P1:Lock(0x1ac)	LL	S	1	M[...]	1	S	1							1	P1:BusRd S=1, P2:Cacheflush	
P3:Lock(0x1ac)	LL									S	1	M[...]	1		P3:BusRd S=1, MemSup	
P1:Lock(0x1ac)	LL			M[...]	1										nic, P1 ma CacheHit	
P3:Lock(0x1ac)	LL											M[...]	1		nic, P3 ma CacheHit	