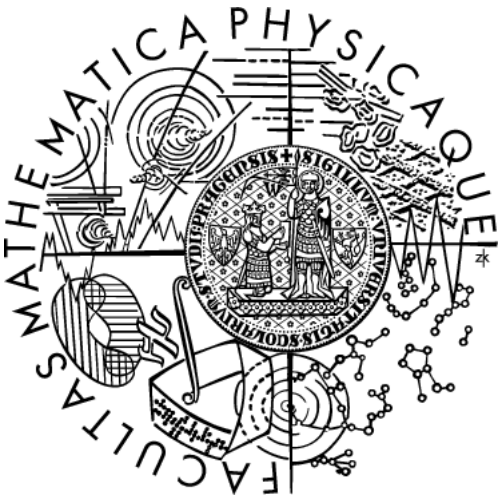
Sbírka příkladů k přednášce
Principy počítačů a operačních systémů

Matematicko-fyzikální fakulta, UK



Tomáš Martinec
fyzmat at gmail.com
2. ledna 2012

Úvodem

Tato sbírka příkladů je určená jako doplňující materiál k přednášce principů počítačů a operačních systémů, která je vyučovaná na MFF UK. Sbírka má dvě části. První část obsahuje zadání příkladů a ve druhé části je jejich řešení a v některých případech i naznačený postup řešení. Příklady, které zasahují mimo přednášené učivo, jsou označené znakem *. Autoři věří, že vypracování příkladů ve sbírce pomůže studentům v lepším porozumění této problematiky.

Spousta příkladů pochází z vlastních hlav autorů. Většina úloh ke kapitolám Mikroarchitektura a Zvyšování výkonu procesorů byla převzata z knihy [1]. V několika úlohách z oblasti operačních systémů jsme se inspirovali dílem [2]. Několik málo úloh je odvozeno ze zkouškových zadání. Snažili jsme se formulovat úlohy tak, aby jejich zadání samo o sobě nenabádalo ke správné odpovědi. Ve sbírce se také vyskytují úlohy, které jsou spíše úvahové a nemusí mít jednoznačně správnou odpověď. Autoři však v řešení píší, které věci by si měl řešitel uvědomit.

Nutno podotknout, že jsme se dbali na to, aby úlohy měly správné výsledky v řešení. Přesto se může stát, že se ve výsledcích objeví chyba. V takovém případě nás prosím kontaktujte, abychom ji opravili.

Poděkování

Děkuji Martinu Drábovi za pečlivou kontrolu zadání a řešení úloh po stylistické, faktické a pedagogické stránce. Díky přednášejícím Luboši Bulejovi a Davidu Obdržálkovi za názory a nápad vytvořit tuto sbírku.

Reference

[1] PATTERSON, David A.; HENNESSY, John L. *Computer Organization and Design : THE HARDWARE/SOFTWARE INTERFACE*. 3rd ed. San Francisco : Elsevier, 2005. 688 p. ISBN 1-55860-604-1.

- strana 229, check yourself
- strana 307, obrázek datové cesty
- strana 354, cvičení 5.1
- strana 372, example
- strana 426, check yourself
- strana 455, cvičení 6.4
- strana 499, example
- strana 527, example
- strana 555, cvičení 7.2, 7.3, 7.4
- strana 558, cvičení 7.28

[2] TANENBAUM, Andrew S.; WOODHULL, Albert S. *Operating systems : Design and implementation*. 3rd ed. Upper Saddle River (New Jersey) : Prentice Hall, 2006. 1080 p. ISBN 0-13-132938-8.

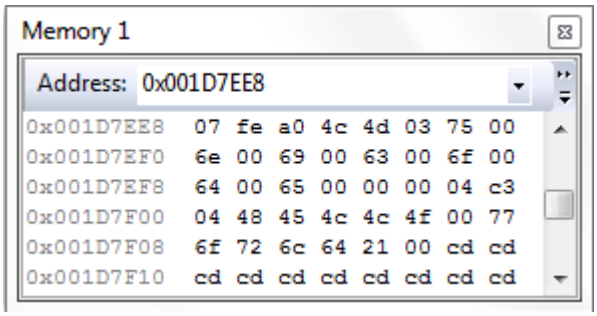
- strana 52, úloha 7
- strana 216, úloha 14
- strana 218, úloha 26

1 Zadání příkladů

1.1 Reprezentace dat

1. Převeďte z desítkové soustavy do dvojkové soustavy 8 bitová čísla 127_{10}, 65_{10} a 104_{10}.
2. Převedená čísla převeďte do osmičkové a šestnáctkové soustavy.
3. Převeďte čísla -7_{10} a 16_{10} do 8 bitového binárního kódu s posunutím nuly o 35 (tzn. $00100011_2 = 0_{10}$).
4. Převeďte čísla -70_{10}, 0_{10} a 100_{10} do 8 bitového jedničkového doplňku. Jaký rozsah čísel lze 8 bitovým jedničkovým doplňkem zakódovat?
5. Převeďte čísla -16_{10} a 50_{10} do dvojkového doplňku, poté je v něm sečtěte a výsledný součet převeďte zpět do desítkové soustavy.
6. Vyjádřete desetinné číslo $8,75_{10}$ v 8 bitovém binárním kódu s fixní řádovou čárkou. Bity 0, 1 a 2 určují desetinnou část. Jaké nejmenší rozlišení kód dává?
7. Mějme následující 4B kód s plovoucí řádovou čárkou:
   - Nejvyšší bit je znaménkový, kde nula odpovídá kladnému znaménku.
   - Následujících 8 bitů vyjadřuje exponent s posunutou nulou o $7F_{16}$.
   - Ostatní bity vyjadřují normalizovanou mantisu.

   Převeďte do daného plovoucího kódu číslo 25_{10} a $0,375_{10}$. Převedená čísla poté sečtěte a vynásobte.
8. Převeďte čísla 47_{10} a 96_{10} do BCD kódu v binárním zápisu na 8 bitech. Jaký rozsah čísel lze na 8 bitech takto zakódovat?
9. Dekadické celé číslo vyjadřujeme binárně pomocí n bitů a nejmenší vyjadřované číslo je 0. Kolik celých čísel můžeme zakódovat s n bity? Jaké je maximální číslo?
10. Podívejte se na dump paměti na obrázku 1. Architektura je little endian.



Obrázek 1: Memory view.

   (a) Zjistěte dekadické hodnoty v poli tří 32 bitových celých čísel v dvojkovém doplňku, které začíná na adrese 0x001D7EE9.
   (b) Zjistěte hodnotu nulou ukončeného ASCII řetězce (typicky používaného v C) začínajícího adresou 0x001D7F01.
   (c) Zjistěte hodnotu ASCII řetězce (typicky používaného v Pascalu) začínajícího adresou 0x001D7F00, jehož délka je určená jeho prvním bytem.
   (d) V paměti se nachází smysluplný ASCII řetězec, který je zakódován kódováním UTF-16. Jakou má hodnotu a na které adrese začíná?

1.2 Operace s daty

1. Vynásobte čísla 01101100_2 a 4_{10} (výsledek uveďte v binární soustavě).
2. Jaký efekt bude mít operace "modulo 16" použitá na 8 bitové celé číslo bez znaménka?
3. Definujme si programovací pseudojazyk, který obsahuje:
   - Dva datové typy:
     - Logický s hodnotami TRUE a FALSE. Je zaveden především pro if příkaz.
     - 8 bitový celočíselný bez znaménka. Proměnné mohou mít pouze tento typ.
   - Příkaz přiřazení ":=".
   - IF-THEN a IF-THEN-ELSE příkazy s podmínkou logického datového typu, podle které se rozhoduje, která část příkazu bude vykonána.
   - Binární logické operátory LAND, LOR a LXOR, jejichž operandy mají logický datový typ a výsledek je logická hodnota spočtená podle názvu operace (logický součin, logický součet a nonekvivalence).
   - Binární bitové operátory AND, OR, XOR, jejichž operandy mají celočíselný datový typ a výsledek je celočíselná hodnota spočtená podle názvu operace pro všechny odpovídající si dvojice bitů z operandů.
   - Celočíselné binární operátory porovnání pro rovnost "==", nerovnost "!=", větší než ">" a menší než "<".
   - Výrazy mohou obsahovat celočíselné konstanty.
   - Pořadí vyhodnocování výrazů je zleva do prava. Jinak se výraz musí patřičně ozávorkovat.

   V definovaném jazyce proveďte následující operace:

   (a) Proměnné b vynulujte bity 0 a 3, nastavte bit 7 a ostatním bitům invertujte hodnotu.

   (b) Nastavte proměnnou b tak, aby její nejvyšší bit zabezpečil ostatní bity lichou paritou.

1.3 Instrukce

1. Na RISC procesorech typicky nebývá instrukce pro odečítání. Jak byste tedy odečítali bez této instrukce?
2. *Definujeme smyšlený assembler pro smyšlenou architekturu. Architektura má nekonečně mnoho 8 bitových general purpose registrů značených R1, R2, ... a register FLAGS, který obsahuje příznakové bity ZERO a GREATER. Registr FLAGS se může změnit pouze při provádění instrukce CMP. Na architektuře je možné adresovat 256B paměti.

   Obecný formát instrukce zapsané v assembleru je:

   navesti: jmeno_instrukce operandy,

   kde návěští je nepovinné. Operandy jsou oddělené čárkou.

   Popis jednotlivých instrukcí:

   - MOV DEST, SOURCE - Kopíruje hodnotu ze SOURCE do DESTINATION. Oboje SOURCE a DESTINATION mohou být registry, místo v paměti na dané adrese (zapsáno [#adresa]) a nebo místo v paměti, jehož adresa je určená registrem (zapsáno [RX]). SOURCE ještě může být celočíselná konstanta (zapsáno jako #konstanta).

- CMP RX, RX - Porovná hodnoty v daných registrech a nastaví podle toho registr FLAGS. Bit ZERO je nastaven právě když jsou porovnávané hodnoty stejné a bit GREATER právě když je první operand větší než druhý. Operandy mohou být pouze registry.
- ADD RX, RX - Sečte hodnoty v daných registrech a výsledek uloží do prvního operandu. Operandy mohou být pouze registry.
- JE LABEL - Skočí na instrukci označenou daným návěštím, jestliže je bit ZERO nastaven.
- JG LABEL - Skočí na instrukci označenou daným návěštím, jestliže je bit GREATER nastaven.
- JLE LABEL - Skočí na instrukci označenou daným návěštím, jestliže není bit GREATER nastaven.
- JMP LABEL - Skočí na instrukci označenou daným návěštím.
- NOP - Instrukce nic neudělá.

(a) Napište program, který načte z paměti na adrese 0x40 číslo a jestliže bude načtené číslo větší než 7, tak na adresu 0x41 zapíše číslo 1 a jinak číslo 0.

(b) Napište program, který překopíruje blok paměti o velikosti 100B, který začíná na adrese 20. Blok nakopírujte do paměti od adresy 120.

1.4 Architektura počítače

1. Nahraďte třívstupové logické hradlo NOR dvouvstupovými logickými hradly NOR.
2. *Mějme logickou funkci Y třech proměnných, která je zadaná tabulkou 1.

A	B	C	Y(A,B,C)
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

Tabulka 1: Tabulka pro funkci třech logických proměnných

(a) Realizujte Y pomocí libovolně mnoha hradel NAND, které můžou mít libovolný počet vstupů.

(b) Realizujte Y pomocí dekodéru 1 z 8 a hradla OR s libovolně mnoha vstupy.

(c) Realizujte Y pomocí multiplexoru se třemi adresovacími vstupy.

(d) Realizujte Y pomocí paměti, která má šířku slova 4 bity a uchovává 16 paměťových buněk. Paměť má datové vstupy, adresové vstupy, řídící $R/\overline{W}$ vstup a datové výstupy. Pokud je na $R/\overline{W}$ vstupu logická nula, tak si paměť zapamatuje zapisovaná data, a pokud je na $R/\overline{W}$ logická jednička, tak lze zapsaná data přečíst z výstupů paměti. Pro tuto aplikaci použijte paměť v read only režimu a napiště obsah paměti.

3. Vystačíme si s kombinační logikou, aby jsme z ní sestavili následující obvody?

(a) Obvod, který porovnává hodnotu dvou n-bitových čísel.

(b) Obvod, který snižuje hodnotu n-bitového čísla na vstupu o jedna.

(c) Konečný stavový automat.

(d) Multiplexor.

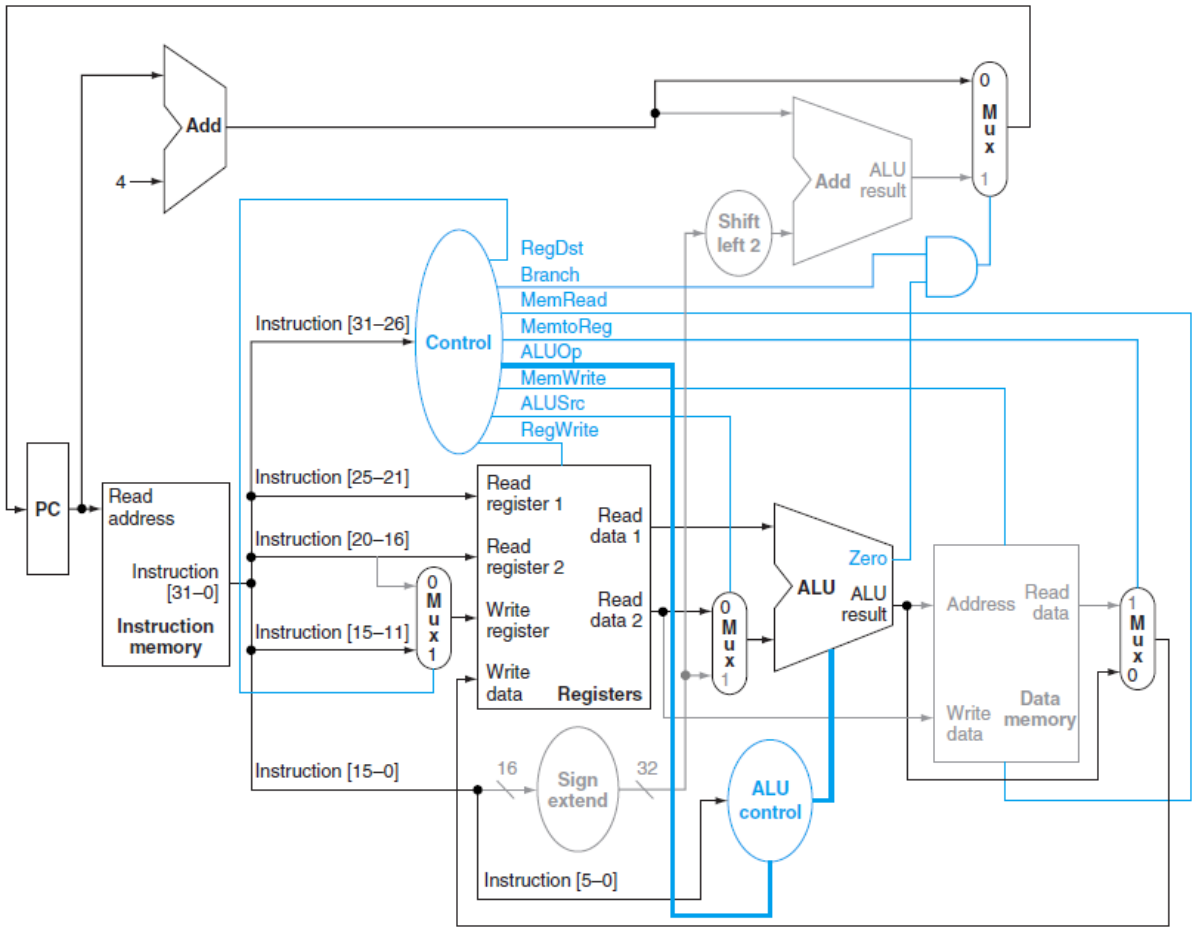
(e) Registr procesoru.

1.5 Mikroarchitektura

1) Na obrázku 2 jednocyklové datové cesty jsou dva obvody pro přístup do paměti. Jeden je pro přístup do paměti pro instrukce a druhý pro přístup do paměti pro data.

   (a) Z jakého důvodu nemůže tato datová cesta mít obvod pro přístup do paměti jen jeden?

   (b) Musí být adresové prostory paměti programu a paměti dat oddělené?



Obrázek 2: Jednocyklová datová cesta s řízením

2) Podívejte se na schéma datové cesty na obrázku 2. Uvažujme instrukci pro načítání slova z paměti do registru ve tvaru:

```
lw $r2, offset($r1)
```

Instrukce uloží obsah paměti do registru *r2*. Adresa čtené paměti se spočítá jako součet obsahu registru *r1* a číselné konstanty *offset*.

Která z následujících tvrzení jsou pravdivá při vykonávání této instrukce?

(a) Řídící signál MemtoReg musí být nastaven na logickou jedna.

(b) Řídící signál ALUSrc musí být nastaven na logickou nulu.

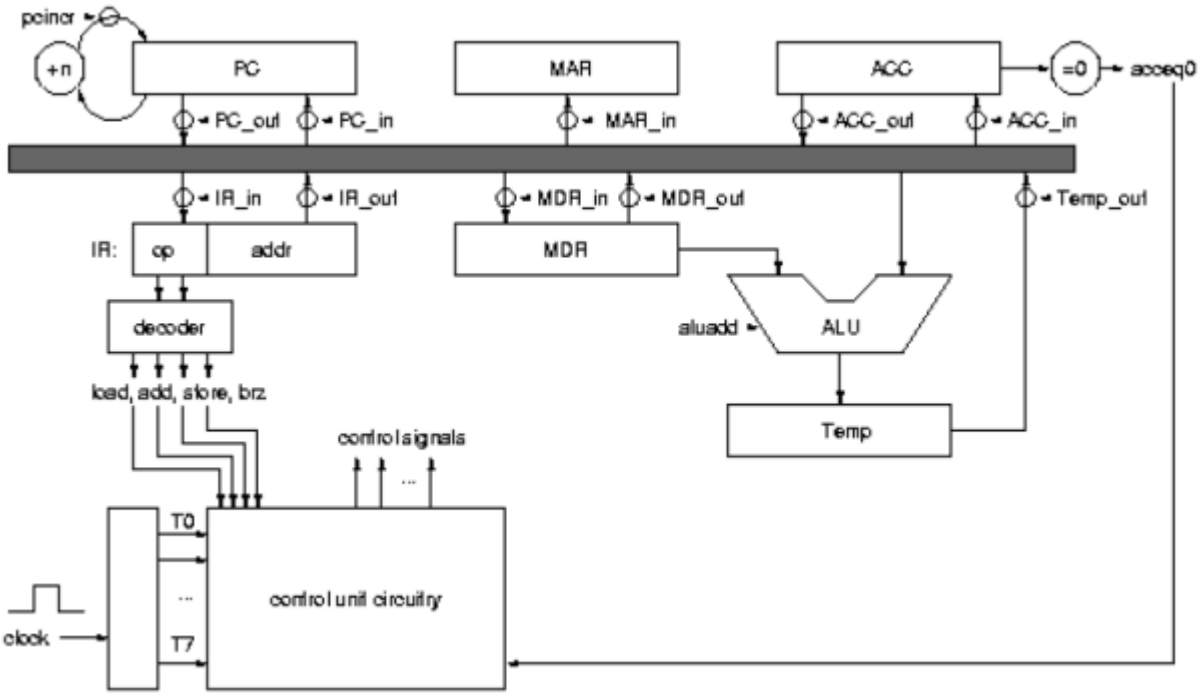
(c) Na hodnotě řídících signálů ALUOp nezáleží.

3) Z jakého důvodu jsou v řízení na obrázku 2 přítomny signály MemRead a MemWrite, když jeden je opakem druhého?

4) K čemu slouží obvod *Sign extend* na obrázku 2?

5) Uvažujme ještě datovou cestu z obrázku 2. Napište mikroprogram pro store instrukci.

6) Na obrázku 3 je schéma jednoduchého procesoru a heslovitý popis řídících signálů.



Obrázek 3: Schéma jednoduchého procesoru

Řídící signály:

- ACC_in: ACC <= CPU internal bus
- ACC_out: CPU internal bus <= ACC
- aluadd: addition is selected as the ALU operation
- IR_in: IR <= CPU internal bus
- IR_out: CPU internal bus <= address portion of IR
- MAR_in: MAR <= CPU internal bus
- MDR_in: MDR <= CPU internal bus
- MDR_out: CPU internal bus <= MDR

- PC_in: PC <= CPU internal bus
- PC_out: CPU internal bus <= PC
- pcincr: PC <= PC + 1
- read: MDR <= memory[MAR]
- TEMP_out: CPU internal bus <= TEMP
- write: memory[MAR] <= MDR
- branch_via_table: change the address of next microinstruction according to decoded instruction

Řadič je implementován metodou mikroprogramování. Mikroinstrukce mimo jiné obsahují adresu následující mikroinstrukce a bit or_address_with_acceq. Když je tento bit nastaven, tak se binární hodnota příznaku acceq0 přičtě k adrese následující mikroinstrukce.

Uvažujme, že procesor umí zpracovat pouze instrukce "DOUBLE", která zdvojnásobí obsah akumulátoru, a instrukci "ADDMEM address", která přičte na zadanou adresu v paměti obsah akumulátoru. Navrhněte obsah řídící paměti řadiče, operační kódy instrukcí a funkci dekodéru instrukcí.

1.6 Zvyšování výkonnosti

1. Obrázek 4 ukazuje, jak dlouho zpracovávají jednotlivé části procesoru instrukci. Zpoždění ostatních částí procesoru ignorujme. Jak dlouho bude trvat provedení následujících třech instrukcí v pipelině?

```
lw $r1, 0($r0)
lw $r2, 4($r0)
lw $r3, 8($r0)
```

   Instrukce lw se chová stejně jako v příkladu (2) kapitoly Mikroarchitektura. Jaká by byla odpověď, když by se délka ALU operace zkrátila na polovinu? A když by se prodloužila o polovinu?

2. Najděte v následující posloupnosti instrukcí hazardy a změňte pořadí instrukcí, tak aby se v pipelině nevyskytovala žádná bublina. Snažte se zachovat funkci programu, jestli to půjde.

```
lw   $r1, 0($r0)
lw   $r2, 4($r0)
add  $r3, $r1, $r2
sw   $r3, 12($r0)
lw   $r4, 8($r0)
add  $r5, $r1, $r4
sw   $r5, 16($r0)
lw   $r6, 20($r0)
```

Instrukce	Instruction fetch	Čtení registrů	ALU	Přístup do paměti	Zápis do registrů	Celkem
lw	200	150	200	200	150	900
sw	200	150	200	200	X	750
add	200	150	200	X	150	700

Obrázek 4: Doba zpracování jednotlivých částí instrukcí v ps

3. Vyznačte, kde se může uplatnit forwarding při vykonání instrukcí:

```
add  $r3, $r4, $r6
sub  $r5, $r3, $r2
lw   $r7, 100($r5)
add  $r8, $r7, $r2
```

Můžete vyjít ze schématu podobnému obrázku 14 z řešení.

4. Porovnejte účinnost strategií pro predikci skoku: "vždy se skočí", "vždy se neskočí" a dynamická predikce. Předpokládejte, že neúspěšná predikce zapříčiní bublinu v pipelině na dva cykly. Průměrnou úspěšnost dynamické predikce uvažujte 90%. Porovnání proveďte pro následující skoky:

   (a) Skočí se v 5% případů.

   (b) Skočí se v 95% případů.

   (c) Skočí se v 70% případů.

1.7 Paměťový subsystém

1. Popište ne nutně smysluplný program (například pseudokódem), který pracuje s daty tak, že vykazuje:

   (a) nízkou časovou a prostorovou lokalitu,

   (b) vysokou časovou lokalitu a nízkou prostorovou lokalitu,

   (c) nízkou časovou lokalitu a vysokou prostorovou lokalitu.

2. Jakými hardwarovými způsoby (jako například přepínačema na základní desce) nebo softwarovými způsoby (jako například zápisem do konfiguračního registru procesoru) se mění asociativita procesorové cache?

3. Uvažujme cache o velikosti 64kB a s bloky (tj. cachelinami) o velikosti 64B. Cache je přímo mapovaná a používá se 32-bitová adresa. Jaké číslo má blok cache, do kterého se namapuje byte na adrese 0x0F32B12A?

4. Mějme tři cache o velikosti 256B a s bloky o velikosti 64B. První cache je přímo mapovaná, druhá je dvou cestně asociativní a třetí je plně asociativní. Pro výběr, který blok se nahradí v asociativní cache se používá algoritmus LRU. Když LRU nemůže jednoznačně rozhodnout, tak se použije blok s nejmenším pořadovým číslem. LRU je implementováno čítačem pro každý blok. O čísle setu v případě dvou cestně asociativní cache rozhodují nižší bity adresy. Adresa se používá 32-bitová. Na začátku je obsah všech tří cache nevalidní.

   (a) Ke kolika cache miss dojde u jednotlivých cache, pokud program přistoupí postupně k následujícím adresám: 0xABFFF000, 0xABFFF209, 0x00000000, 0xABFFF97D, 0xABFFF000, 0xABFFF220, 0xACC00A54 a 0xABFFF209?

   (b) Kolik bitů informace musí uchovávat jednotlivé cache, když budeme uvažovat i režijní informace a nejenom prostor pro obsah operační paměti?

5. Asociativita cache obvykle zlepšuje miss ratio, ale ne vždy. Uveďte posloupnost přístupů do paměti, tak aby počet cache miss byl menší u přímo mapovené paměti než u asociativní paměti. Parametry obou cache si můžete zvolit, ale musí se lišit pouze asociativitou.

1.8 Systémová architektura, připojování a komunikace se zařízeními

1. Následuje definice smyšlené sběrnice, která slouží pro komunikaci více I/O zařízení s centrálním procesorem. Přenos dat probíhá synchronně s hodinami o frekvenci 10MHz. Procesor a všechna připojená zařízení reagují na sestupnou hranu hodinového signálu. Data jsou přenášena datovými vodiči a adresa adresovými vodiči. Sběrnice má tyto řídící signály:

   - $R/\overline{W}$ určující, zda půjde o čtení ze zařízení nebo zápis na zařízení (pro čtení je na něm logická jedna),
   - $REQUEST$ pro signalizaci platnosti adresy a $R/\overline{W}$ signálu,
   - $READY$ pro signalizaci připravenosti zařízení.

   Přenos dat probíhá v následujících krocích:

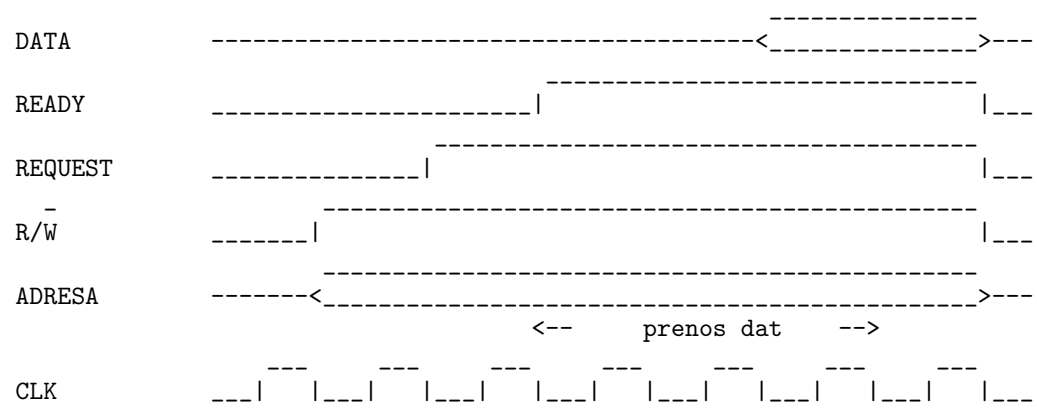
   (a) Procesor nastaví zároveň adresu zařízení, na které chce přistupovat, a $R/\overline{W}$ signál.

   (b) V následujícím taktu hodin procesor nastaví $REQUEST$ signál. Všechna připojená zařízení při náběžné hraně $REQUEST$ signálu kontrolují, jestli jejich adresa odpovídá adrese na sběrnici. Když odpovídá, tak má zařízení jeden takt hodin na to, aby nastavilo $READY$ signál. Tím je spojení navázáno. Pokud procesor nedetekuje $READY$ signál včas, tak vyvolá přerušení, že zařízení neodpovídá.

   (c) Po navázání spojení má ten, kdo zapisuje na sběrnici, tři hodinové takty na zapsání dat na datovou sběrnici. Při třetím taktu se považuje obsah datové sběrnice za platný.

   (d) Při čtvrtém taktu po nastavení $READY$ signálu se všechny řídící signály nulují. Adresa a data jsou v nespecifikovaném stavu. Přenos dat tímto skončil.

   Obrázky 5 a 6 ukazují časové diagramy pro čtení a pro zápis.



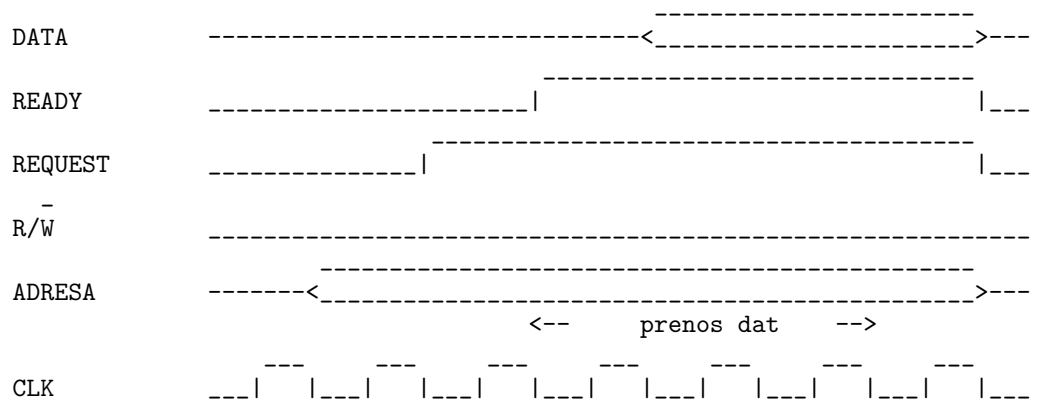
Obrázek 5: Časový diagram pro čtení

   (a) Kolik je třeba adresových vodičů, aby bylo možné komunikovat s devíti zařízeními?

   (b) Kolik je třeba datových vodičů, aby propustnost této sběrnice pro čtení i pro zápis byla alespoň 25MB/s?

   (c) Předpokládejme, že sběrnice podporuje zařízení s velmi pomalou odezvou. Umožňuje to tak, že čeká vysoký počet taktů na nastavení $READY$ signálu. Kolik má zařízení času na nastavení $READY$ signálu, aby přenosová rychlost byla v nejhorším případě alespoň 500kB/s? Uvažujme sběrnici s 16 datovými vodiči.

   Tato sběrnice s čekáním na pomalé zařízení by nebyla vhodná jako systémová sběrnice procesoru. Z jakého důvodu?



Obrázek 6: Časový diagram pro zápis

1.9 Operační systémy - úvod

1. Jakým způsobem lze při programování uživatelských aplikací zakázat vyvolání obsluhy přerušení?
2. U kterých instrukcích se dá očekávat, že jsou privilegované (tj. můžou být vykonané pouze v kernelu operačního systému)?
   (a) HALT, která pozastaví činnost procesoru do příchodu přerušení.
   (b) MOV, která kopíruje obsah paměti a kde jsou oba operandy registry, které obsahují zdrojovou a cílovou adresu.
   (c) EXEC, po které bude procesor vykonávat program od zadané adresy. Adresa může být i v paměti, která není označená jako executable.
   (d) SYSCALL.
   (e) RDTM, která do fixně stanoveného registru nahraje počet taktů hodin od zapnutí počítače.

1.10 Operační systémy - procesy, plánování, synchronizace

1. Uvažme zvláštní systém, kde všechny procesy počítají po dobu T a potom se zablokují na I/O volání. Systém běží pouze na jednom procesoru a používá round robin plánování s kvantem Q. Přeplánování procesu trvá čas S a tato doba se považuje jako režije operačního systému. Plánovač operačního systému nemá nikdy prázdnou frontu. Pro následující předpoklady vyjádřete, kolik procent času je vykonáván kód procesů.
   (a) $Q = \infty$
   (b) $Q > T$
   (c) $S << Q < T$
   (d) $Q = S$
   (e) Q se blíží nule
2. Programátora operačního systému napadlo, že by pozdržel obsluhu přerušení, čímž by zamezil vstupu více vláken do synchronizované kritické sekce. K tomu použije před kritickou sekcí instrukci pro zakázání vyvolání přerušení a následně za kritickou sekcí umístí instrukci pro povolení vyvolání přerušení. Tyto instrukce zakáží/povolí přerušení na procesoru, který je vykoná. Programátor si myslí, že tento postup zamezí přeplánování běžícího vlákna, které by mohlo nastat kvůli vypršenému kvantu nebo nastalé I/O události.

(a) Jak dlouhé kritické sekce by bylo vhodné tímto způsobem synchronizovat?

(b) Jak dobře by navrhovaný postup fungoval na počítači s jedním procesorem?

(c) Jak dobře by navrhovaný postup fungoval na počítači s více procesory?

(d) Jak dobře by navrhovaný postup fungoval na počítači s více procesory, kdyby popsané instrukce zakázaly/povolily přerušení na všech procesorech najednou?

3. Uvažujme vícevláknovou aplikaci, která v hlavním vlákně vytvoří 100 vláken. Každé vytvořené vlákno jednou zvýší globální proměnnou *prom* o 100. Proměnná *prom* je zpočátku nastavená na hodnotu 1000. Hlavní vlákno počká až všechna ostatní vlákna doběhnou a potom vypíše hodnotu proměnné *prom*.

   Jakou hodnotu proměnné *prom* vypíše program?

   Pro upřesnění je program podrobněji popsán následujícím C kódem:

```
// includovani nezbytnych souboru s deklaracemi, ktere neuvadime
#include <...>

// Globalni promenna
int prom = 1000;

// Pocet vlaken
#define THREAD_COUNT 100

// V tomto poli si budeme pamatovat identifikace vlaken,
// abychom na ne potom mohli cekat
pthread_t id[THREAD_COUNT];

/*
 * Hlavni vlakno
 */
int main(int argc, char *argv[])
{
  // tady vytvorime a spustime vsechna vlakna;
  // jako startovaci funkci vytvorenych vlaken stanovujeme
  // funkci funkce_pro_vytvorene_vlakna, ktera inkrementuje
  // promennou prom

  // funkce pthread_create nam taky vraci identifikator
  // vytvoreneho vlakna, ktery si ukladame do pole id
  // na prislusnou pozici
  for (int i = 0; i < THREAD_COUNT; i++) {
    pthread_create(&id[i], NULL, funkce_pro_vytvorene_vlakna, NULL);
  }

  // tady postupne cekame az se vlakna dokonci
  for (int i = 0; i < THREAD_COUNT; i++) {
    pthread_join(thread_info[i].id, NULL);
  }

  // nakonec vypiseme hodnotu promenne prom
  printf("%d\n", prom);

  return 0;
}
```

Fyzická adresa tabulky		0x44000000
Index	Číslo rámce	Present bit
0	6	1
1	0	0
2	20	0
3	24	1
4	0	0
5	1	0
6	`0	1
7	0	0

Fyzická adresa tabulky		0x00000000
Index	Číslo rámce	Present bit
0	1	1
1	13	0
2	15	0
3	57	1
4	31	0
5	40	0
6	28	1
7	0	0

Fyzická adresa tabulky		0x18000000
Index	Číslo rámce	Present bit
0	3	1
1	63	0
2	57	0
3	0	0
4	0	0
5	43	1
6	17	1
7	48	1

Fyzikální adresa tabulky		0x60000000
Index	Číslo rámce	Present bit
0	35	0
1	14	1
2	67	0
3	51	0
4	61	0
5	0	0
6	0	0
7	11	0

Obrázek 7: Stránkovací tabulky. Číslo rámce je v **dekadické soustavě.**

```
/*
 * Tuto funkci provadi vlakna, ktera jsou vytvorena
 * ve funkci main. Kazde vytvorene vlakno zvysi hodnotu
 * prom o 100 a potom se ukonci.
 */
void funkce_pro_vytvorene_vlakna()
{
  prom = prom + 100;
}
```

4. Předpokládejme, že procesor umí vykonat instrukci, která atomicky vymění zadané místo v paměti s obsahem zadaného registru. Dala by se taková instrukce použít pro synchronizaci kritické sekce?

1.11 Operační systémy - správa paměti, virtuální paměť

1. Kolik bitů potřebujeme na zachycení adresového prostoru o velikosti 0,5KB, 1KB, 4KB, 1MB a 4GB?

2. V této úloze budeme uvažovat ne příliš praktickou implementaci dvou úrovňového stránkování s netypickou velikostí stránek. Na obrázku 7 je vyobrazen obsah dvou úrovňových stránkovacích tabulek. Počítač má 32-bitový adresový prostor a tabulka první úrovně je na fyzické adrese 0x44000000. Záznam tabulky obsahuje mimo čísla fyzického rámce také pět příznakových bitů. Procesor nepodporuje nastavení executable nebo read-only práv pro jednotlivé stránky paměti. Je zobrazen bit *present* udávající, jestli je stránka přítomna v operační paměti. Mechanismus stránkování je navrhnut tak, aby pro každou stránkovací tabulku byl vyhrazen celý rámec fyzické paměti (i přestože je rámec větší).

   (a) Jak je veliký rámec?

(b) Kam do fyzické paměti se přistoupí, když uživatelký proces zapíše na adresu 0x7D8F00AB.

(c) Kam do fyzické paměti se přistoupí, když uživatelský proces přečte adresu 0x19AB4309?

(d) Co může běžně očekávát programátor uživatelského procesu, když přečte paměť z adresy 0x00000010?

(e) Tabulka na fyzické adrese 0x60000000 svědčí o tom, že v implementaci stránkovacích tabulek je chyba. Co je konkrétně špatně?

(f) Obsah stránkovacích tabulek také svědčí o tom, že v návrhu mechanismu stránkování je závažná bezpečnostní chyba. Nalezněte, o co se jedná.

3. Operační systém má k dispozici 3 zpočátku nepoužité rámce fyzické paměti. Uživatelské programy používají stránky v tomto pořadí: 1 2 3 2 5 6 2 4 2 3 1. Jakým algoritmem výměny stránek z FIFO, LRU a One handed clock docílíme v daném případě nejméně výpadků stránek?

4. Smyšlená procesorová architektura má 32-bitový adresový prostor a 64-bitové registry. Instruční sada je uzpůsobená tak, aby během provedení jedné instrukce mohl nastat nejvýše jeden přístup do paměti. Hardware podporuje použití virtuální paměti. Dolní 4 bity virtuální adresy tvoří offset do stránky paměti. Překlad virtuální adresy probíhá tak, že procesor umístí překládanou adresu do zvláštního registru a vyvolá přerušení. Následně operační systém začne hledat příslušnou fyzickou adresu ve stránkovacích tabulkách. Pokud se překlad nenalezne ve stránkovacích tabulkách, tak se bude hledat ještě ve swap prostoru na disku, a pokud se nenalezne ani tam, tak operační systém zabije běžící proces za přístup do nenamapované paměti. Po nalezení překladu operační systém uloží získanou fyzickou adresu zpět do registru, ve kterém měl na počátku virtuální adresu, a ukončí se přerušení. Procesor poté pokračuje ve vykonávání programu a při adresaci paměti použije získanou fyzickou adresou. V horní polovině virtuálního adresového prostoru jsou adresy mapovány přímo na první 2GB fyzické paměti a tyto adresy mohou být použity jen v privilegovaném režimu procesoru (tj. zpravidla pouze v jádře operačního systému). Překlad těchto adres probíhá přímo v procesoru.

   Co je na návrhu této architektury nevhodné?

5. Uvažte všechny kombinace výpadků "cache miss", "TLB miss" a "page miss" (překlad není ve stránkovacích tabulkách), které nastanou během provedení jedné instrukce. Za jakých okolností mohou jednotlivé kombinace nastat a případně, které kombinace nemohou nastat?

2 Řešení příkladů

2.1 Reprezentace dat

1. $127_{10} = 01111111_2$, $65_{10} = 01000001_2$ a $104_{10} = 01101000_2$
2. $01111111_2 = 177_8 = 7F_{16}$, $01000001_2 = 101_8 = 41_{16}$ a $01101000_2 = 150_8 = 68_{16}$
3. $-7_{10} = 00011100$, $16_{10} = 00110011$
4. $-70_{10} = 11000110$, $0_{10} = 00000000$ nebo $0_{10} = 10000000$, $100_{10} = 01100100$
5. $-16_{10} = 11110000$, $50_{10} = 00110010$
6. $8,75_{10} = 01000110$
7. $25_{10} = 0100000111001000\ldots$, $0,375_{10} = 00111110110000\ldots$

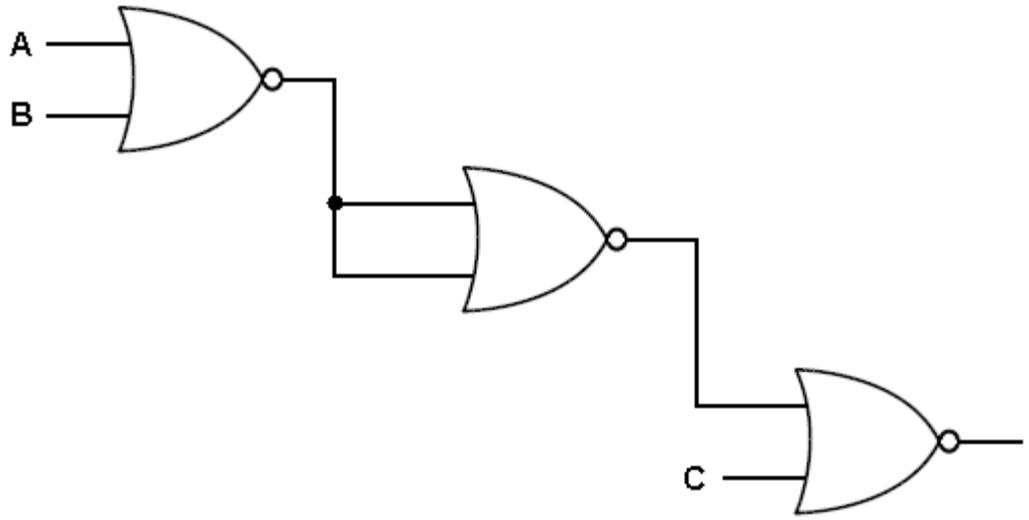
   Pro sčítání je třeba číslo 0,375 vyjádřit v nenormalizovaném tvaru $0100000110000011000\ldots$. Součet je $0100000111001011000\ldots$ a násobek je $01000001000101100\ldots$.
8. $47_{10} = 01000111_{BCD}$, $96_{10} = 10010110_{BCD}$. Na 8 bitech lze v BCD kódování zakódovat čísla 0 až 99.
9. S n bity můžeme zakódovat 2^n čísel. Maximální číslo je $2^n - 1$.
10. (a) -24322, 19788, 29995
    (b) HELLO
    (c) HELL
    (d) unicode; začíná na adrese 0x001D7EEE

2.2 Operace s daty

1. Násobení se dá provést rychle posunem bitů. 10110000_2
2. Operace vynuluje nejvyšší 4 bity.
3. (a)
```
b := b AND 0xF6
b := b OR 0x80
b := b XOR 0x76
```
   (b)
```
IF ( ((b AND 0x01) == 0)) LXOR ((b AND 0x02) == 0)) LXOR
   ((b AND 0x04) == 0)) LXOR ((b AND 0x08) == 0)) LXOR
   ((b AND 0x10) == 0)) LXOR ((b AND 0x20) == 0)) LXOR
   ((b AND 0x40) == 0)) )
THEN
   b := b AND 0x7F
ELSE
   b := b OR 0x80
```

2.3 Instrukce

1. Instrukce pro sčítání celých čísel typicky pracuje s operandy v dvojkovém doplňku. Tedy by se neodečítalo kladné číslo, ale přičítalo opačné záporné číslo.

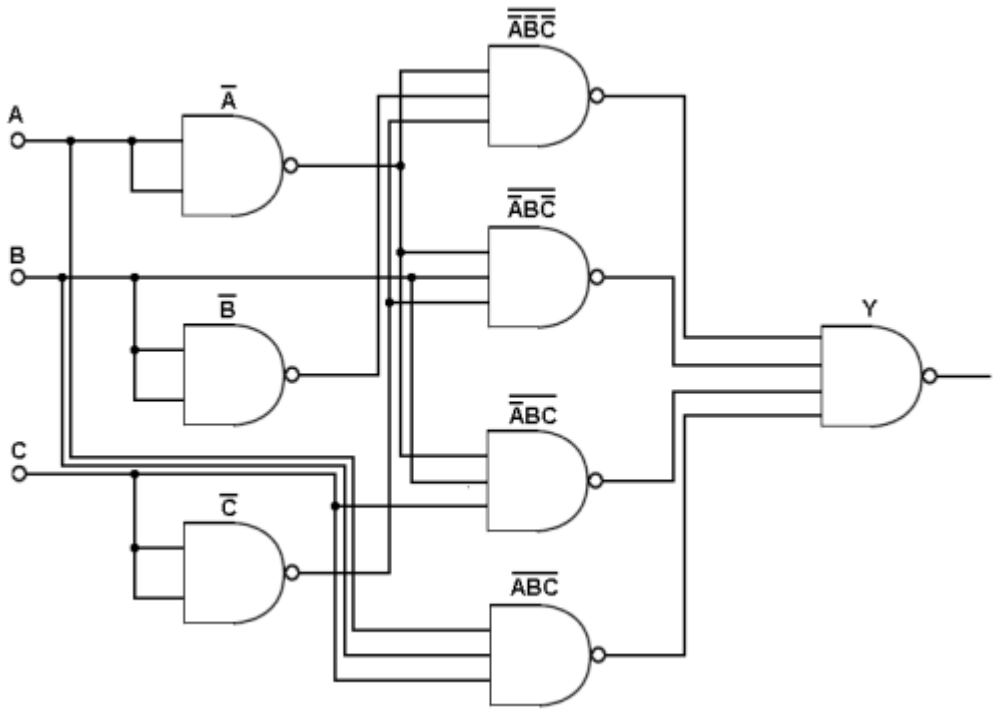


Obrázek 8: Nahrazení třívstupového hradla NOR dvouvstupovým hradlem NOR

2. (a)
```
                MOV R1,[#64]
                MOV R2, #7
                CMP R1,R2
                JG vetsi
                MOV [#65], #0
                JMP konec_zvonec
vetsi:          MOV [#65], #1
konec_zvonec:   NOP
```
(b)
```
                MOV R1, #20
                MOV R2, #0
                MOV R3, #100
                MOV R4, #1
                MOV R5, #120
for_cyklus:     CMP R2,R3
                JLE konec_zvonec
                MOV [R5], [R1]
                ADD R1, R4
                ADD R2, R4
                ADD R5, R4
                JMP for_cyklus
konec_zvonec:   NOP
```

2.4 Architektura počítače

1. Jedna z možností zapojení je na obrázku 8:
2. (a) Zde je uveden postup:
   i. Všímáme si řádků, kde Y platí. Například první řádek lze přepsat do implikace $\overline{A} \cdot \overline{B} \cdot \overline{C} \Rightarrow Y$.
   ii. Y platí právě když platí předpoklad alespoň jedné takové implikace. $Y = (\overline{ABC}) + (\overline{A}B\overline{C}) + (\overline{A}BC) + (ABC)$



Obrázek 9: Realizace logické funkce pomocí NAND hradel

iii. Abychom mohli všude použít NAND, tak na získaný předpis funkce aplikujeme dvojí negaci a De Morganovy zákony. $Y = \overline{\overline{(\overline{A}B\overline{C}) + (\overline{A}B\overline{C}) + (\overline{A}BC) + (ABC)}} = \overline{\overline{(\overline{A}B\overline{C})} \cdot \overline{(\overline{A}B\overline{C})} \cdot \overline{(\overline{A}BC)} \cdot \overline{(ABC)}}$

iv. Získaný výraz překreslíme do hradlové sítě na obrázku 9.

Je také možné snažit se funkci minimalizovat, čímž by se pravděpodobně dospělo k jednoduššímu řešení.

(b) Schéma řešení je na obrázku 10.

(c) Schéma řešení je na obrázku 11.

(d) Na obrázcích 12 je schéma a obsah paměti.

3. (a) Ano.

(b) Ano.

(c) Ne.

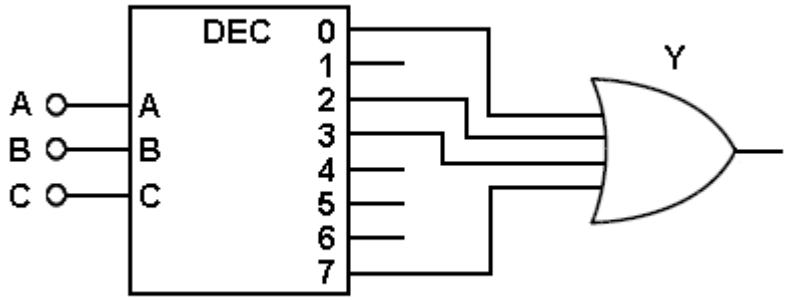
(d) Ano.

(e) Ne.

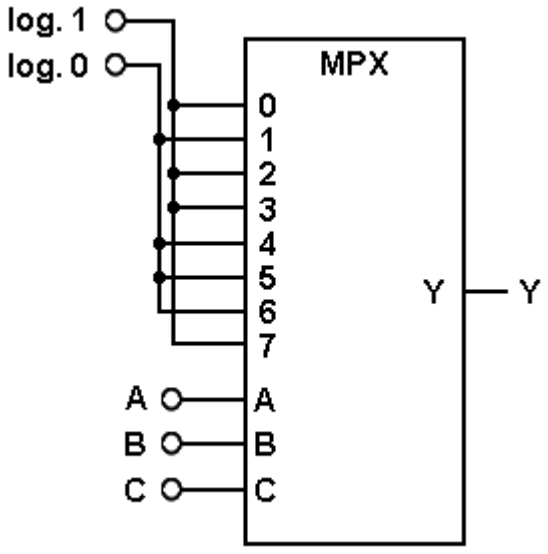
2.5 Mikroarchitektura

1. Každý synchronní obvod dělá v jednom taktu právě jednu operaci, pro kterou je konstruován. Když je instrukce provedena v jednom taktu, tak jeden obvod pro přístup do paměti nestačí.

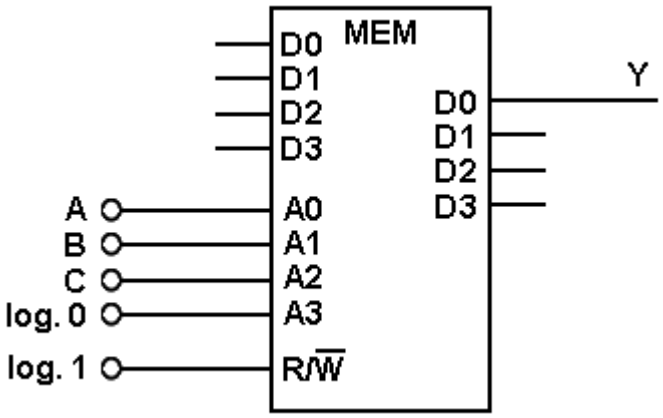
2. Paměť pro program a pro data může být sdílená, ale jak by se to dělalo je mimo rozsah učiva. Čtenář by si měl uvědomit, proč oba obvody pro přístup do paměti nemůžou používat stejnou sběrnici k paměti.



Obrázek 10: Realizace logické funkce pomocí dekodéru



Obrázek 11: Realizace logické funkce pomocí multiplexoru



(a) Schéma paměti

Adresa	Data
0	9
1	8
2	9
3	9
4	8
5	8
6	8
7	9
8	A
9	B
A	C
B	D
C	E
D	F
E	A
F	0

(b) Obsah paměti

Obrázek 12: Realizace logické funkce pomocí paměti

Nemožnost sdílení sběrnice musí být ošetřena nějakým mechanismem, aby bylo možné sdílet paměť. Například je možné mít sběrnice dvě k jedné paměti. Nebo by šlo propojit obvody pro přístup do paměti s cache namísto přímo s pamětí. Obvody cache by si potom zajistily synchronizovaný přístup do paměti samy.

3. (a) MemtoReg musí být nastaven na logickou jedna.
   (b) ALUSrc musí být nastaven na logickou jedna.
   (c) Na jejich hodnotě záleží. Musí být taková, aby ALU sčítala.
4. U instrukcí, které nepracují s pamětí by se přistupovalo do paměti zbytečně.
5. Obvod *Sign extend* umožňuje používat relativní adresování i se zápornou adresou.
6. Uvádíme jednu možnost návrhu:

   Operační kód je jednobitový. Instrukce DOUBLE má operační kód 0 a instrukce ADDMEM má operační kód 1.

   Obsah řídící paměti je na obrazku 13

   Výkonný mikrokód instrukce DOUBLE začíná na adrese 0100 a instrukce ADDMEM na adrese 0111.

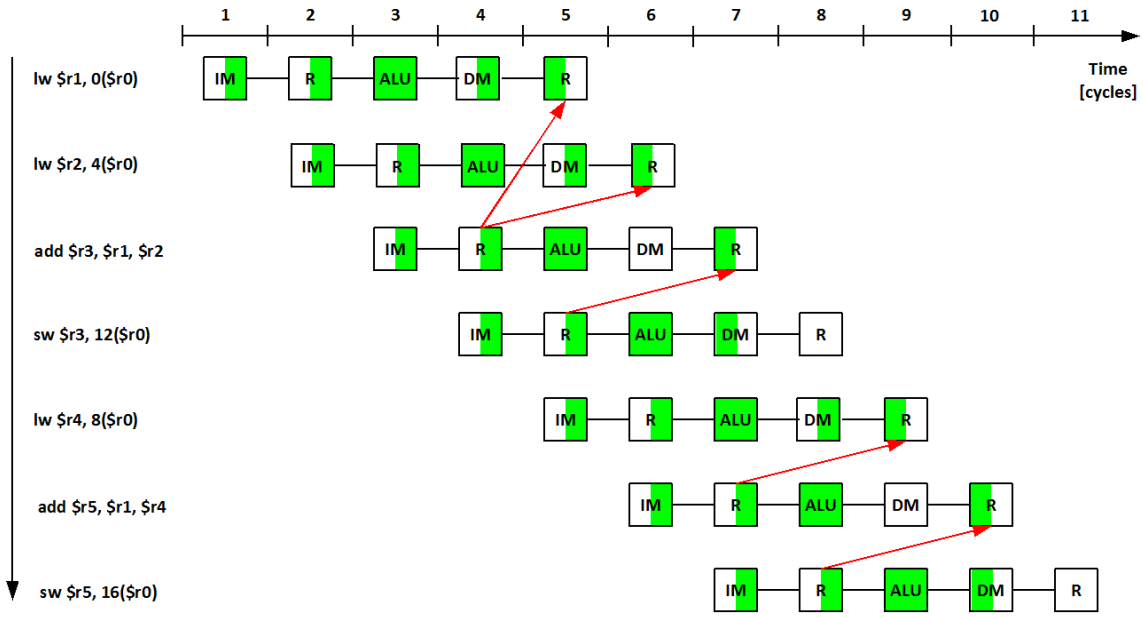
2.6 Zvyšování výkonnosti

1. 1400ps. Při zrychlení ALU 1400ps a při zpomalení ALU 2100ps.
2. Hazardy jsou nakreslené v obrázku 14.

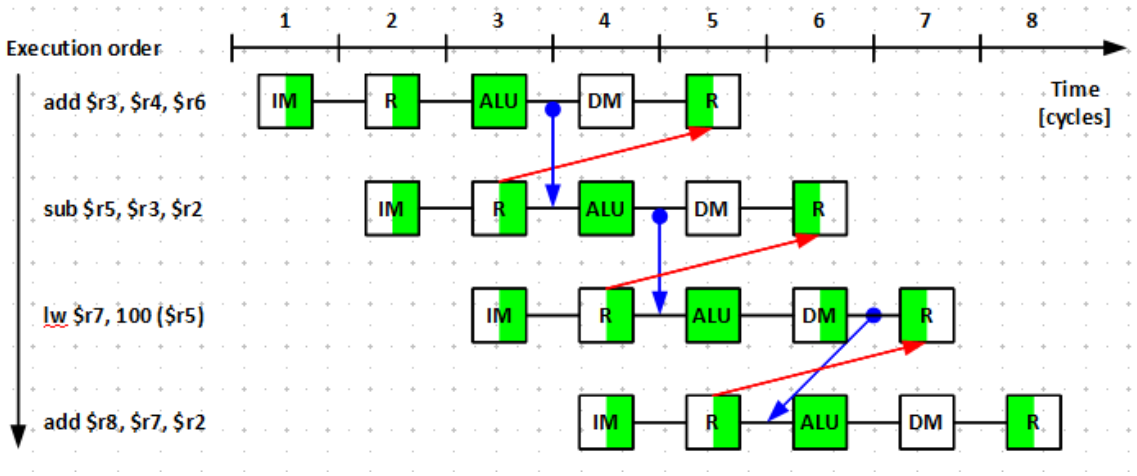
   Původní funkci programu nejde přerovnáním zachovat, když ještě nemá dojít ke stallu v pipeline. Toto je jedna z variant řešení:

	ACC_in	ACC_out	aluadd	IR_in	IR_out	MAR_in	MDR_in	MDR_out	PC_in	PC_out	pcincr	read	TEMP_out	write	branch-via-table	next CS address	or-address-with-accecq
0000						1				1						0001	
0001											1	1				0010	
0010				1				1								0011	
0011															1	xxxx	
0100		1					1									0101	
0101		1	1													0110	
0110	1												1			0000	
0111					1	1										1000	
1000												1				1001	
1001		1	1													1010	
1010							1						1			1011	
1011														1		0000	

Obrázek 13: Obsah řídící paměti



Obrázek 14: Datové hazardy



Obrázek 15: Forwarding

```
lw   $r1, 0($r0)
lw   $r2, 4($r0)
lw   $r4, 8($r0)
lw   $r6, 20($r0)
sw   $r3, 12($r0)
sw   $r5, 16($r0)
add  $r3, $r1, $r2
add  $r5, $r1, $r4
```

3. Řešení je na obrázku 15.

4. Když se skočí v 5% případů, tak je nejlepší predikce "vždy se neskočí" s průměrným stallem pipeliny 0,1 cyklu na skok. V případě instrukce větvení s 95% pravděpodobností skoku je to strategie "vždy se skočí" se stallem 0,1 cyklu. V případě 70% to je dynamická predikce s průměrným stallem 0,2 cyklu.

2.7 Paměťový subsystém

1. Následuje pseudokód popsaných programů. Při reálné implementaci těchto programů je třeba hlouběji rozumět chování překladače. V komentářích jsou pro zajímavost zmíněny problematické místa, ale nečeká se, že by si jich čtenář měl být vědom.

   (a)
```
// naalokujeme dostatečně velké pole bytů
byte pole[VELIKOST_CACHE];

delej do nekonecna {
        // vybereme si náhodně jeden byte a zapíšeme do něj
        // náhodnou hodnotu
        int index = random(0, VELIKOST_CACHE);
        byte hodnota = random(0, 256);

        // poznamenejme, že čtení by v tomto případě nemuselo stačit, protože
        // by ho mohl vypustit překladač; je taky potřeba dát pozor na další
        // optimalizace překladače, což zde nemá smysl podrobněji rozebírat
        pole[index] = hodnota;
}
```

číslo přístupu	adresa přístupu	hit/miss	Obsah jednotlivých bloků (určen začínající adresou)			
			0	1	2	3
1	0xABFFF000	miss	0xABFFF000			
2	0xABFFF209	miss	0xABFFF200			
3	0x00000000	miss	0x00000000			
4	0xABFFF97D	miss	0x00000000	0xABFFF940		
5	0xABFFF000	miss	0xABFFF000			
6	0xABFFF220	miss	0xABFFF200			
7	0xACC00A54	miss	0xABFFF200	0xACC00A40		
8	0xABFFF209	hit	0xABFFF200	0xACC00A40		

Obrázek 16: Přímo mapovaná cache

číslo přístupu	adresa přístupu	hit/miss	Obsah jednotlivých bloků (určen začínající adresou)/čítač			
			0	1	2	3
1	0xABFFF000	miss	0xABFFF000/0			
2	0xABFFF209	miss	0xABFFF000/1		0xABFFF200/0	
3	0x00000000	miss	0x00000000/0		0xABFFF200/1	
4	0xABFFF97D	miss	0x00000000/0	0xABFFF940/0	0xABFFF200/1	
5	0xABFFF000	miss	0x00000000/1	0xABFFF940/0	0xABFFF000/0	
6	0xABFFF220	miss	0xABFFF200/0	0xABFFF940/0	0xABFFF000/1	
7	0xACC00A54	miss	0xABFFF200/0	0xABFFF940/1	0xABFFF000/1	0xACC00A40/0
8	0xABFFF209	hit	0xABFFF200/0	0xABFFF940/1	0xABFFF000/1	0xACC00A40/0
	Set 0					
	Set 1					

Obrázek 17: Dvou cestně asociativní cache

(b)
```
int cislo = 0;

for i = 0 to 1000000 do {
        // zde si ale musime od prekladace explicitne vynutit zápis do
        // paměti, jinak by mohl inkrementovat přímo v registru
        cislo = cislo + 1;
}
```

(c)
```
byte pole[VELIKOST_POLE];

for i = 0 to (VELIKOST_POLE - 1) do {
        pole[i] = pole[i] + 1;
}
```

2. Asociativita procesorové cache se nedá změnit. Měnitelná asociativita by nejspíše nepřinesla příliš užitku. Navíc velikost cache v procesoru je limitovaná počtem hradel, který je výrobce schopen integrovat do čipu. A další zesložitění obvodů pro cache by ještě více omezovalo její velikost.

3. Byte se namapuje do bloku číslo 708 (dekadicky).

4. (a) Odpověď je na obrázcích 16, 17 a 18.

   (b) V případě přímo mapované cache 2148 bitů. V případě dvou cestně asociativní 2156 bitů a v případě plně asociativní 2164 bitů.

číslo přístupu	adresa přístupu	hit/miss	Obsah jednotlivých bloků (určen začínající adresou)/čítač			
			0	1	2	3
1	0xABFFF000	miss	0xABFFF000/0			
2	0xABFFF209	miss	0xABFFF000/1	0xABFFF200/0		
3	0x00000000	miss	0xABFFF000/2	0xABFFF200/1	0x00000000/0	
4	0xABFFF97D	miss	0xABFFF000/3	0xABFFF200/2	0x00000000/1	0xABFFF940/0
5	0xABFFF000	hit	0xABFFF000/0	0xABFFF200/3	0x00000000/2	0xABFFF940/1
6	0xABFFF220	hit	0xABFFF000/1	0xABFFF200/0	0x00000000/3	0xABFFF940/2
7	0xACC00A54	miss	0xABFFF000/2	0xABFFF200/1	0xACC00A40/0	0xABFFF940/3
8	0xABFFF209	hit	0xABFFF000/2	0xABFFF200/0	0xACC00A40/1	0xABFFF940/3

Obrázek 18: Plně asociativní cache

Plně asociativní cache

číslo přístupu	adresa přístupu	hit/miss	Obsah jednotlivých bloků (určen začínající adresou)/čítač	
			0	1
1	0	miss	0/0	
2	1	miss	0/1	1/0
3	3	miss	3/0	1/1
4	0	miss	3/1	0/1

Přímo mapovaná cache

číslo přístupu	adresa přístupu	hit/miss	Obsah jednotlivých bloků (určen začínající adresou)	
			0	1
1	0	miss	0	
2	1	miss	0	1
3	3	miss	0	3
4	0	hit	0	3

Obrázek 19: Kdy je přímé mapování lepší než asociativita

5. Jev ukážeme na plně asociativní cache s dvěma bloky. Každý blok má velikost 1B. Obrázek 19 ukazuje, na jaké adresy a v jakém pořadí se přistupovalo.

2.8 Systémová architektura, připojování a komunikace se zařízeními

1. (a) Je potřeba mít 4 adresové vodiče.
   (b) Je potřeba mít 147 datových vodičů. Po jednom datovém vodiči se přenese přibližně 1428571 bitů za sekundu. Je potřeba přenést $25 * 1024 * 1024 * 8$ bitů za sekundu.
   (c) Zařízení musí nastavit $READY$ signál do 3,3 mikrosekundy po detekci signálu $REQUEST$. Jako jeden důvod proč není sběrnice vhodná lze považovat, že procesor ztratí příliš mnoho času prováděním I/O operace. Přístup k pomalým I/O zařízením by bylo lepší řešit softwarově než aktivním čekáním procesoru. Například s podporou mechanismu přerušení nebo metodou pollování.

2.9 Operační systémy - úvod

1. Zakázat obsluhu přerušení při programování uživatelské aplikace nelze. Je to tak navrženo, aby se aplikace nemusely zatěžovat složitými mechanismy pro práci s hardware. Když aplikace potřebuje funkcionalitu založenou na systému přerušení (nebo jiném hardwarovém prostředku), tak využívá k tomu určené rozhraní operačního systému. Navíc případné špatné zacházení se systémem přerušení by vedlo ke kolapsu systému.

2. (a) HALT privilegovaná

(b) MOV neprivilegovaná

(c) EXEC neprivilegovaná

(d) SYSCALL neprivilegovaná

(e) RDTM neprivilegovaná

2.10 Operační systémy - procesy, plánování, synchronizace

1. (a) $\frac{T}{T+S}$

   (b) $\frac{T}{T+S}$

   (c) $\frac{T}{T+\lceil T/Q \rceil * S}$

   (d) $\frac{T}{T+\lceil T/Q \rceil * S}$, pokud se kvantum začíná odpočítávat až od konce přeplánování. Jinak by se muselo uvažovat kvantum zmenšené o čas mezi začátkem odpočítávání a koncem přeplánování.

   (e) Doba, po kterou se vykonávají procesy se blíží k nule. Hned po přeplánování se přeplánuje znovu.

2. (a) Kritické sekce by měli být velmi krátké - ideálně pouze několik instrukcí. Některé zařízení očekávají velmi rychlou odezvu od operačního systému a operační systém nemůže reagovat se zakázanými přerušeními.

   (b) Na jednoprocesorovém stroji by popsaný postup fungoval, pokud by provedení kritické sekce nezpůsobilo přeplánování (například voláním funkce sleep). Když se během provádění kritické sekce nemůže přeplánovat, tak ji provede jen jedno vlákno současně.

   (c) Na víceprocesorovém stroji by popsaný postup nefungoval, protože na ostatních procesorech se přeplánovat může.

   (d) Postup by nefungoval i v tomto případě. Na ostatních procesorech totiž nějaké vlákna také mohou běžet. Sice by nedošlo k přeplánování, ale vlákna z ostatních procesorů by také mohla vstoupit do kritické sekce. Pomohlo by zakázání přerušení a zároveň pozastavení ostatních procesorů. To by ale bylo neefektivní a tak se o synchronizování pomocí zákazu přerušení na víceprocesorových strojích neuvažuje.

3. Program vypíše nějaké číslo z množiny {1100, 1200, 1300, ..., 11000}.

4. Ano. Například by v paměti byla hodnota 1 a před zamknutím kritické sekce by procesor vynuloval nějaký registr. Procesor by se snažil vyměnovat hodnotu paměti s vynulovaným registrem, dokud by v registru měl nulu. Potom by měl kritickou sekci zamčenou. Pro odemčení stačí nastavit hodnotu v příslušné paměti zpět na 1.

2.11 Operační systémy - správa paměti, virtuální paměť

1. Potřebujeme po řadě 9, 10, 12, 20 a 32 bitů.

2. (a) Každý rámec je veliký 64MB.

   (b) Stránka příslušná k virtuální adrese 0x7D8F00AB není načtená v operační paměti. Pokud si ji proces vůbec namapoval, tak záleží na operačním systému, který rámec použije. Jinak operační systém ukončí proces za přístup do nenamapované paměti.

   (c) Bude přečtená fyzická adresa 0x45AB4309.

Cache miss	TLB miss	Page miss	Možné?
Ne	Ne	Ne	Možné; ideál
Ne	Ne	Ano	Nemožné; co je v TLB, to musí být ve stránkovacích tabulkách
Ne	Ano	Ne	Možné, operační systém doplnil do TLB informace o překladu
Ne	Ano	Ano	Možné; operační systém musel hledat vyswapovanou stránku
Ano	*	*	Cache miss je možný vždy. Adresa použitá pro cache je známá až po překladu adres, takže závislost TLB miss a page miss je stejná jako v předchozích řádcích.

Tabulka 2: Vztahy výpadků v paměťovém subsystému

(d) Programátor by měl očekávat, že program vždy spadne. Neplatné ukazatele mají často nulovou hodnotu. Je rozumné stránku příslušející k adrese 0x00000000 (a tedy i k 0x00000010) vůbec nemapovat. Díky tomu je přístup přes nulové ukazatele (například *NULL* v C nebo *nil* v Pascalu) okamžitě detekován operačním systémem, který program vzápětí ukončí.

V našem příkladu je ale stránka 0x00000000 namapovaná.

(e) Maximální smyslupná hodnota čísla rámce je 63.

(f) Stránkovací tabulka první úrovně je namapovaná. Uživatelský proces tak může měnit její obsah a tak si může zpřístupnit obsah celé operační paměti.

3. Při FIFO nastane 9 výpadků, při LRU 8 výpadků a při One handed clock 9 výpadků.

4. (a) Při každém použití virtuální paměti je volán operační systém, aby provedl překlad. To výrazně zhorší výkonnost architektury. Běžně je v procesoru používána asociativní (tzn. v důsledku rychle prohledávaná) paměť, která obsahuje omezené množství překladů. Tato paměť se nazývá TLB (Translation Lookaside Buffer) a díky ní může překlad adres často probíhat bez asistence operačního systému.

(b) Velikost stránek je příliš malá. Toto by mělo za důsledek, že by výrazně častěji docházelo k výpadkům stránek než s běžnými 4kB stránkami. Navíc by byl problematický návrh stránkovacích tabulek, protože pro každých namapovaných 16B virtuální paměti musí být veden záznam o velikosti 4B (při typickém způsobu implementace stránkovacích tabulek).

(c) Registry, které pracují s adresami (například program counter) využívají pouze 32 bitů ze 64 bitů.

5. Odpovědi jsou v tabulce 2.