Uveďte metriky hodnocení PC, které lze získat z benchmarků. (2) (Př. 3)
Doba CPU → provedením programu; ***Hodinová frekvence***; ***Počet instrukcí*** → získává se obtížně (pomocí simulátorů nebo HW čítačů) – závisí na architektuře (pro P4 a P1 stejný pro stejný program); nejtežší je získat ***CPI*** (střední hodnota počtu cyklů na instrukci) → ovlivněno všemi atributy návrhu PC

Co jsou řídící hazardy u RISC a jak omezuje jejich vliv (5) (Př. 13 - Snímek53)
Cílová adresa je známa až na konci třetího cyklu (provádění) => pozastavení. CPU neví, jakou instrukci má načíst jako další
Řešení: Opoždění instrukcí větvení; statická a dynamická predikce - smyčky

Co jsou strukturované hazardy u RISC a jak je obejít (2) (Př. 13 - Snímek53)
Různé instrukce se snaží používat současně stejné funkční jednotky (např. paměť, registrovou sadu). Vznikají, jelikož nemáme dostatek HW pro současné provádění většího počtu instrukcí.
Řešení: duplikovat HW, úprava stupňů pipeline

Popsat datový hazard RISC (4) (Př. 13 - Snímek53)
Instrukce je závislá na výsledku předchozí instrukce, která je stále v pipeline (prostě chci počítat s něčim, co se teprve počítá)
Řešení: pozastavování vložení 3 bublin do pipeline; změna pořadí instrukcí, aby se omezilo pozastavování; využití techniky forwarding

Vyjmenujte a popište situace, za jakých se může změnit registr PC (Program Counter)
PC se zvyšuje při načtení instrukce. ; Při skoku a větvení. ; Při volání podprogramu. ; Při návratu z podprogramu.

Přímo mapované, 2 cestná, N cestná cache; výhody, nevýhody, jak se projeví při běhu programu (2) (př. 15 – Snímek15)
Dán paměťový blok B
Přímo mapovaná (1 cestná) – cache jedna k jedné → paměťové mapy (B jen v jediném konk. bloku cache)
Částečně asociativní (vícecestná) – jeden z n bloků může obsahuje B; + → klesá prohledávaný prostor; - → více míst kde je třeba hledat B, složitější HW
Plně asociativní – jakýkoliv blok může obsahovat B
Při běhu programu je potřeba prohledávat různě velké části cahce.
Počet cest = Počet Dvojic Tag-Data; Počet bloků / Počet cest = počet řádek :) (Př. 15 – Snímek64!!)

Uveďte základní charakteristiky instrukčních souboru procesoru RISC. Jakým zp. je organizován přístup do paměti. (2) (Př. 4 - Snímek7)
Instrukční soubor – lze chápat jako interface mezi SW a HW – odděluje složitost implementačních detailů od SW
Šest hl. typů – Load/Store; Aritmetické; Skoky a větvení; Floating Point operace; Memory Management; Speciální
Tři formáty instrukcí (šířka 32 bitů) – R (aritmetické), I (přesuny dat, větvení), J (skoky)
Data v paměti zarovnána a přístupná pouze pomocí lw (load) a sw (store)

Jak bude pracovat cache, když při přístupu do paměti nenajde data. (Př. 15 – Snímek21)
Je potřeba data načíst z hlavní paměti. CPU se pozastaví a čeká, dokud nejsou data načtena. Během zastevení CPU musí každý registr uchovat svoje data (snažší než zastavit pipeline). Čteme-li z datové paměti → zastavíme celé CPU, dokud nejsou data k dispozici

Co je to řadič procesoru – popsat a vysvětlit, o jaký logický obvod se jedná (Př. 11 - Snímek03)
Řídí činnost CPU; konečný automat, Stav – generování řídících signálů; Hrany – podmínky přechodu; Lze implementovat v HW (ROM) – programovatelné logické pole – obsahuje přechodovou fci. a registr, ve kterém je uložen konečný stav

Popište funkce Linkeru a jaké datové struktury používá. (4) (Př. 5 - Snímek73)
Sestavuje objektové soubory a vytváří spustitelný soubor, edituje odkazy ve skokových instrukcích, vyhodnocuje reference do paměti, umožňuje oddělenou kompilaci souborů
Postup: slučuje kódové segmenty obj. souborů → slučuje datové segmenty obj. souborů (připojeny za kódové segmenty) → vyhodnocuje reference (doplňuje absolutbí adresy)

Popište způsoby zápisu dat do paměti, pokud je v systému přítomna cache. (Př. 15 - Snímek37)
a) Write-Through – zápis dat do cache a současně do bloku hlavní paměti; *Výhoda:* případ „Miss" je jednodušší a levnější, protože není potřeba zapisovat blok zpět do nižší úrovně; snažší implementace (pouze zápisový buffer)
b) Write-Back – zápis pouze do bloku cache. Zápis do paměti pouze při výměně bloků. *Výhoda:* Zápisy omezeny pouze rychlostí zápisu cache; podporovány zápisy více slov najednou, efektivní je pouze zápis celého bloku do hlavní paměti najedou

Popište mechanismy, kterými procesor RISC potlačuje vliv relativně dlouhé doby přístupu do hlavní paměti vzhledem k taktu CPU (Př. 12 – od Snímku14)
1) Rozdělit instrukci na více kroků – každý bude trvat 1 cykl → vícecyklová implementace
2) Každá instrukce používá pouze část HW v každém kroku → využití pipeline
Pipeline – každý proces rozdělen na části – každá prováděna ve spec. funkční jednotce; Čtení instrukce → Dekódování instrukce → Provádění → zápis do paměti → zápis výsledku. Při behu běží současně instrukce a každá vykonává jinoučást svého běhu (jedna načítá, druhá počítá atd. - bacha na skoky ! → řešit vkládáním NOP)

Popište, jakým způsobem se zobrazují čísla v pohyblivé řádové čárce. Jednotlivé složky čísla jsou uloženy ve slově v určitém pořadí, uveďte důvody. (2) (Př. 8 - snímek70)
Normalizovaná notace: [ZNAMENKO]1.[MANTISA]$_2$ * 2$^{[EXPONENT]2}$

Obyčejná přesnost:	*Dvojitá přesnost:*
Znaménko S – 1 bit Exponent – 8 bitů Mantisa – 23 bitů	Znaménko S – 1 bit Exponent – 11 bitů Mantisa – 20 bitů + 32 bitů (pokračování)

Pevné pořadí uložení (Norma) – rychlá komparace → znaménko +/- → exponent větší => vetší č. → mantisa

V pseudokodu realizujte nasl. operace
a = b+c
b = a+c
d = a-d
pro zásobníkovou architekturu, Akumulator, Load store, Memory-to-memory (2) (Př. 6 - Snímek7)

a = b + c (ostatní stejně – operace + => add; operace - => sub)

Stack	Akumulátor	Registr-memory	Load-Store
push b	load b	load R1, b	load R1, b
push c	add c	add R1, c	load R2, c
add	store a	store a, R1	add R3, R1, R2
pop a			store a, R3

Memory-memory
Add a, b, c

Vysvětlete princip mikroprogramového řízení (mikroprogramový automat a jeho strukturu) (2) (Př. 11 - Snímek30)
Myšlenka – řízení CPU mikroprogramem; *HW implementace* – stav uložen v registru, přechodová fce. v ROM (nebo PLA); Vykonávají se jednotlivé uložené instrukce – řízení CPU; Mikroprogram není rychlejší – je snažší měnit SW než HW;

Co je to systém dynamické transformace adresy (virt. paměť). Jeden typ transformačního mechanismu. (2) (Př. 16 – Snímek11)
CPU vytváří virtuální adresu, která je pomocí MMU (Memory Management Unit) transformovaná na fyzickou adresu → ta je použita pro přístup do fyzické paměti
Převod: Adresa se rozdělí na virtuální číslo str. a offset (stranka = VA div vel. rámce, offset = VA mod ve. rámce) → v tabulce stránek najdeme pozici [stranka] = index bloku → FA = index bloku * vel. bloku + offset

U procesoru kategorie RISC na rozdíl od CISC jsou specificky řešeny podmínky pro instrukce větvení programu. Uveďte jak a vysvětlete. (Př. 6 - Snímek22)
Místo porovnání registrů se využívají speciální 1-bitové registry (tzv. „podmínkové kódy“), které tvoří vedlejší „efekt“ operací ALU. Podmínkové kódy se používají pro všechna porovnání; S = Sign Bit; Z = Zero; C = Carry out; P = Parity (na 1, je-li počet jedniček v osmi bitech výsledku operace vpravo sudý)

Jakým zp. je volán podprogram v mikroprocesoru, jaké operace je třeba provést. (2) (Př. 5 - Snímek14)
Před vyvoláním funkce volající – předá arg. ($a0-$a3; zbylé na stack) → uloží ukládané reg. volajícího ($a0-$a3; $t0-$t9) → provede instrukci *jal* (skok na proceduru a uložení return adr.)
Před záhajením výpočtu volané fce. - alokuje se paměť pro frame ($sp = $sp – fsize) → uložení ukládaných reg. volaného ($s0-$s7; $fp; $ra)
Před návratem do volajícího – uložení fční. hodnoty do reg $v0 → obnovení všech reg. bolané fce. → pop stack freme + obnova $fp → návrat na adresu uloženou v $ra

Co je to ortogonalita operačního kódu, adresy (Př. 5 – snímek 43)
Všechny instrukce umí pracovat ve všech adresních módech

Popište funkce Loaderu (Př. 5 - Snímek75)
Spustitelný program je uložen na disku → loader ho zavede do paměti a spustí; součást OS; Čte hlavičku (určení vel. programu a dat) → vytvoří adresní prostor (kód + data + stack) → kopíruje data ze souboru do paměti → inicializuje registry → skok na start rutinu → po ukončení main rutiny se program ukončuje

Co je to přerušovací vektor, o jaký typ informace se jedná a jak je v systému používán. Kde ho lze nalézt ? (2)
Je to adresa na obslužnou rutinu pro přerušení. *Nechráněný režim* → Uložen v tabulce vektorů přerušení (posl. na začátku paměti, kde jsou přerušovací vektory uloženy; struktura záleží na CPU) x *Chráněný režim* → Interrupt Description Table (IDT) (uložena ve speciálním registru)
Při každém požadavku na přerušení se uloží stav procesoru a přejde na na adresu obslužného programu. Po dokončení tohoto programu se předchozí činnost obnoví a výpočet pokračuje dál.

Jaké znáte typy organizace periferních přenosů. Pro jaké typy periferních zařízení se které metody přenosu používají a jaké vyžadují technické vybavení. (Př. 17 – Snímek09)
CPU-řadič-zařízení; Řadič s přerušením (sériové přenosy), ***I/O procesor s pamětí*** (grafika) → více info ZOS
Polled – programová kontrola stavu I/O zařízení a následné plánování činnosti volných I/O zařízení; dotazy na stav – busy, wait, idle...
Interrupt-Driven – obsluha I/O na požadavek; vyžaduje frontu na uložení čekajících I/O žádostí
DMA – přímý přenos dat mezi I/O zařízením a pamětí; velmi rychlé (HDD, video...)

Výpadky v cache (Př. 15 - Snímek46)
Povinné výpadky – při prvním přístupu k bloku se tento blok v cache nenachází; vyšší cena za výměnu bloku; redukce – zvýšení vel. bloku => ovšem pokles výkonu
Kapacitní výpadky – cache neobsáhne všechny bloky potřebné k běhu; nastane, pokud jsou bloky odklízeny do nižší úrovně a při potřebě opět načítány; zvětšení kapacity cache => ovšem vyšší přístupová doba
Výpadky kvůli konfliktu – u přímo mapované nebo část. asosiativní cache; různé bloky obsazují stejnou pozici; zvýšení asociativity cache => ovšem časové ztráty při přístupu

Ideální hradlo (Př. 2 – Snímek47)
Nekonečný zisk v přechodové oblasti; přepínací úroveň umístěna ve středu logického zdvihu; horní a dolní pásy logických úrovní na stejné a rovné polorovníně zdvihu; nulová výstupní a nekonečná vstupní impedance
Hradlo – zpracovává několik vstupů a generuje 1 výstup; repr. logickou fcí nebo pravd. tabulkou

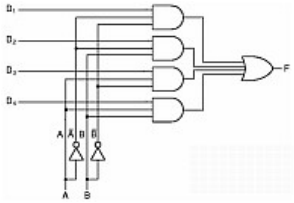
Jaké kroky vykoná procesor při větvení (Př. 9 – Snímek28)
Přístup do registru (Čtení instrukce / dat) → Vyhodnocení podmínky skoky → výpočet cílové adresy (výpočet skoku) → skok na cílovou adresu (provedení instrukce větvení)

Předpokládejme mikroprocesor s cache, který pracuje násl. způsobem: je-li třeba zavést blok, pak je zaveden, provede se výpočet a pak se vyhodí některý ze starých bloků. Rozeberte možnosti jedn. probíraných implementací cache pro takový mikroprocesor. (Př. 15 – Snímek73)
Přímo mapovaná cache – lze těžko implementovat. Nemáme informaci o tom, který blok je starý. Museli bychom vybírat náhodně.
Čistě asociativní cache – velice vhodná. Máme informaci o tom, které bloky jsou jak staré. Nevýhodou je cena.
Více-cestná částečně asociativní cache – Stejný problém jako přímomapovaná
Algoritmy nahrazování stránek
LRU – nahrazován blok, který nebyl nejdéle použit (dvoucestná cache – každý vstupní blok má přidán bit → je-li blok referencován – bit bloku nastaven a odpovídající bit druhého bloku je nulován) → výpadek → nahrazení bloku s vynulovaným bitem)
FIFO; FIFO Second-Chance; Min/OPT... → viz ZOS :)

RISC x CISC (Př. 1 – Snímek 58)
Instrukce prováděny přímo HW; max. průchodnost instrukcí; jednoduché instrukce; přístup do paměti jen instrukcemi load/store; velké mn. registrů; pipeling

Popište podrobně jednotlivé kroky, které vykoná procesor při provádění instrukcí podmíněného skoku. Začněte přesunem do instr. registru. (Př. 4) ???
Instrukce rozhodování → 1) skok při rovnosti – beq registr1, registr2, cil x 2) skok při nerovnosti – bne registr1, registr2, cil (bne $s3, $s4, 20). V případě použití návěští není potřeba vypočítávat cílovou adresu (if i==j goto L1)
Čtení instrukce z paměti → čtení zdrojových registrů → ALU porovnání obsahu → pozitivní výsledek - PC se naplní cílovou adresou, else – skok se neprovede – čte se další instrukce

Popište vnitřní strukturu multiplexoru se 4 datovymi vstupy. Určete celkový počet vstupů a výstupů. Nakreslete. ; ***aka. Multiplexor 1 ze 4*** (2) (Př. 2 – Snímek72)
2^n datových vstupů (=> 2^2 = 4 v našem případě); 1 datový výstup; 2 řídící vstupy



Obecný multiplexer – obr. má 4 datové vstupy, 1 výstup a 2 řídící vstupy
Základní kombinační obvod – výstupy jednoznačně určeny vstupy, bez paměťových prvků; Složený z hradel AND a u výstupu hradlo OR

Amdhalův zákon (Př. 3 - Snímek30)
doba výpočtu po zlepšení = Doba výpočtu ovlivněná zlepšením / Poměr zlepšení + doba výpočtu neovlivněná
zrychlení = doba výpočtu před zlepšením / doba výpočtu po zlepšení

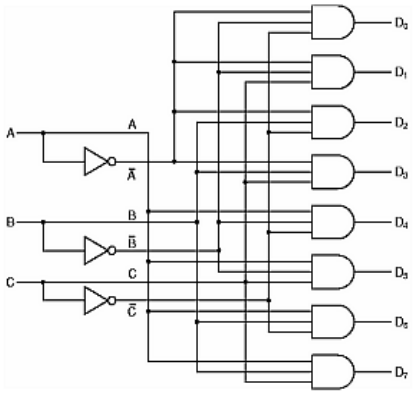
Tabulky assembleru (Př. 5 - Snímek71)
Tabulka symbolů – seznam položek souboru, který bude použit jinými soubory (návěští a data)
Relokační tabulka – seznam položek, pro které je třeba adresa (návěští na které se skáče, načítaná data)

Architektury počítačů (Př 6 - Snímek6)
Stack – žádné registry (kompaktní kódování); Akumulátor – 1 registr (drahý HW); Registr-registr (load – store), Registr-paměť (jeden z operandů může být v paměti, druhý je v registru)

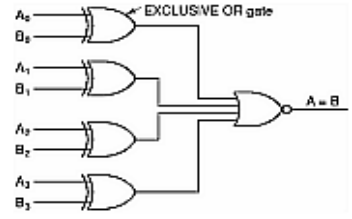
Forwarding (Př. 14 - Snímek5)
Přesun výsledku v registru pipeline do následující instrukce → zabránění datových hazardů

Vysvětlete mikroarchitekturu procesoru. V jakém vztahu je s instrukčním souborem ? (Př. 6) ??
Souhrn vlastností implementace, které uživatel na instrukční úrovni nevidí. Patří do ní vlastnosti, které se mění (technologie, výkon...)

Popište jakým zp. lze omezit „nenalezení" potřebného bloku v cache. (Př. 15) ???
Zvětšení kapacity cache → problémy
Algoritmy nahrazování stránek
LRU – nahrazován blok, který nebyl nejdéle použit (dvoucestná cache – každý vstupní blok má přidán bit → je-li blok referencován – bit bloku nastaven a odpovídající bit druhého bloku je nulován) → výpadek → nahrazení bloku s vynulovaným bitem)
FIFO; FIFO Second-Chance; Min/OPT... → viz ZOS :)

Popište Dekodér (Př. 2 - Snímek73)


n datových vstupů; 2^n datových výstupů

Logický komparátor (Př. 2 - Snímek74)


Jak se RISC „popere“ s (*p1 = *p2 – 666) a Assembler kod. (Př. 5 – kolem Snímku50)
x = *p
lw $s0, 0($a0)
*p = 200
addi $t0, $0, 200
sw $t0, 0($a0)

by Erkil :-)
lw $t0, p2
lw $t0, 0($t0)
addi $t1, $t0, -666
lw $t0, p1
sw $t1, 0($t0)

Co se děje při volání instrukce.
CPU pracuje v nekonečné smyčce – čte instrukce z paměti a provádí je → PC obsahuje adresu aktuální instrukce; Instrukce MIPS jsou dlouhé 4 byte => PC inkrementován o 4, aby ukazoval na následující instrukci

V Assembleru napište prográmky pro
a=(b+c)*c
b=a*c+c
d=(a-d)/(a+d)
a to pro Stack-machine, Accumulator machine, Load-Store, Memory-memory (2)
(viz. výše. Akorát navíc instr. sub a div + to bude delší :-/)

Popište základní vlastnosti registrové sady RISC (ne instrukčního souboru). Hlavně ty, co ovlivňují výkon procesoru, nepopisujte určení registrů ani instr. formáty. (3) (Př. 4)
Omezený počet registrů (32 32bit.); pojmenování číslem ($8) nebo jménem ($t0); $t – temp; $s - saved

Navrhněte 8-bitový posuvný registr (posun vpravo) s paralelním vstupem a sériovým výstupem. Zda se provádí vstup nebo posuv určuje řídící vstup

Navrhněte funkční blok „posuvný registr“ o délce 8 bitů. Pro jeho návrh vyberte libovolný typ klopného obvodu

Vysvětlete prinsip urychlení práce paralelní binární sčítačky. Nakreslete odpovídající schéma pro šířku n-bitu. (2)

Upravit paralelni binarni scitacku, ktera scitala v doplnkovem kodu tak, aby scitala i odcitala – použít stejnou ? (viz. př. 6 - Snímek50)
Jak RISC pracuje s pamětí (výpočet adres, přístupy...)
Přístup do pamětí jen instrukcemi load/store

Sekvenční automat, který při CLK vrací postupně 0,1,1,0,1,1 … (0100100100...) (2)

Navrhněte jednoduchý sekvenčně synchronní automat (2 bitový vratný čítač) – intuitivní a standardní

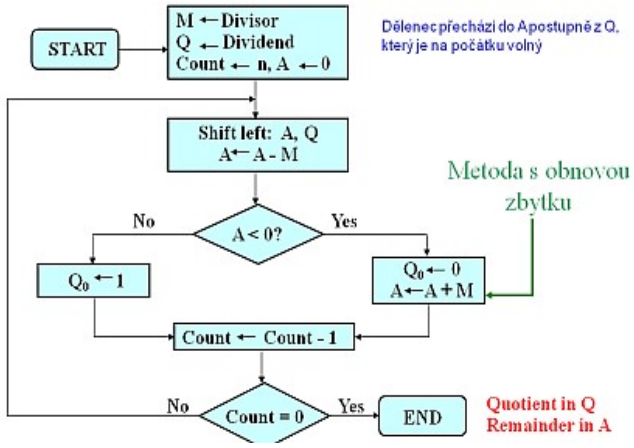
Rozdíly mezi jedno, dvou a částečne asociativní čtyřcestnou cachí.

Jak instrukci předat 32. bit adresu, když má na adresu vyčleněno 14 bit v instrukci ?
(Přitom adresní prostor je 2^{30} bytů. Uveďte různé způsoby, jak zajistit procesoru přístup k datům v celém rozsahu paměti.)

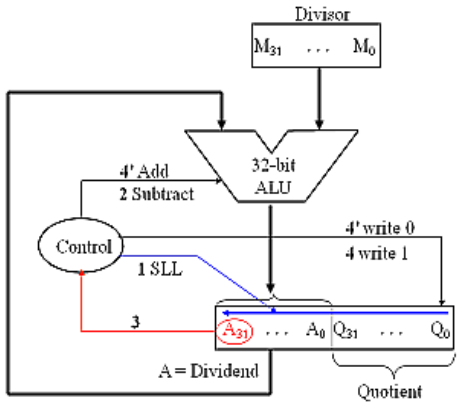
ALGORITMY

Nakreslit algoritmus dělení kladných čísel a navrhnout zapojení. (Př. 7 – snímek22)
Popiště alg. pro dělení binárních čísel, nejlépe formou vývojového diagramu + operační jednotku. (4)

Algoritmus:

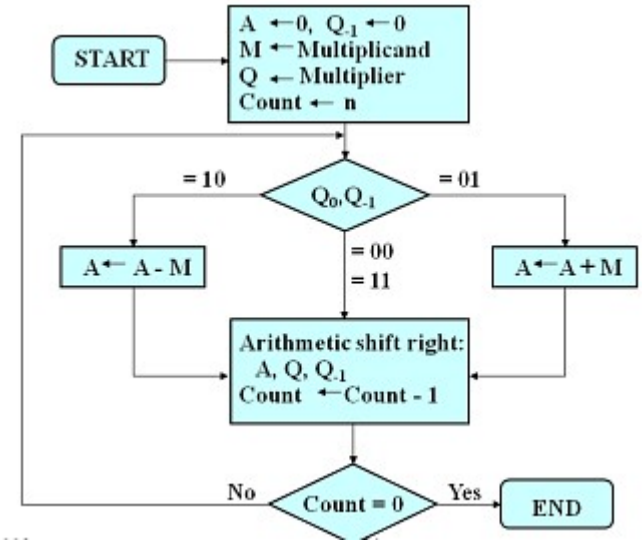


HW jednotka:

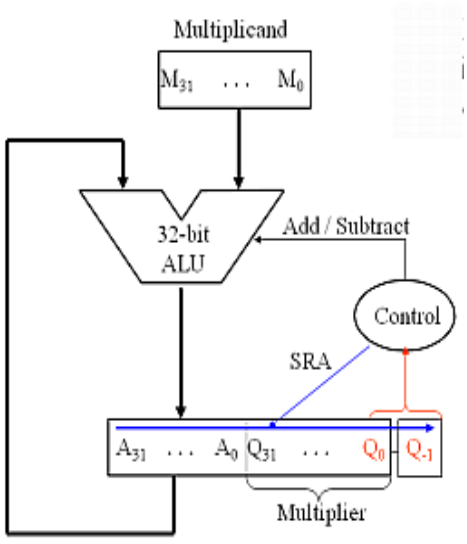


Jak funguje násobení u Boothova alg. (alg. + HW jednotka) (4) (Př. 7 – Snímek12)

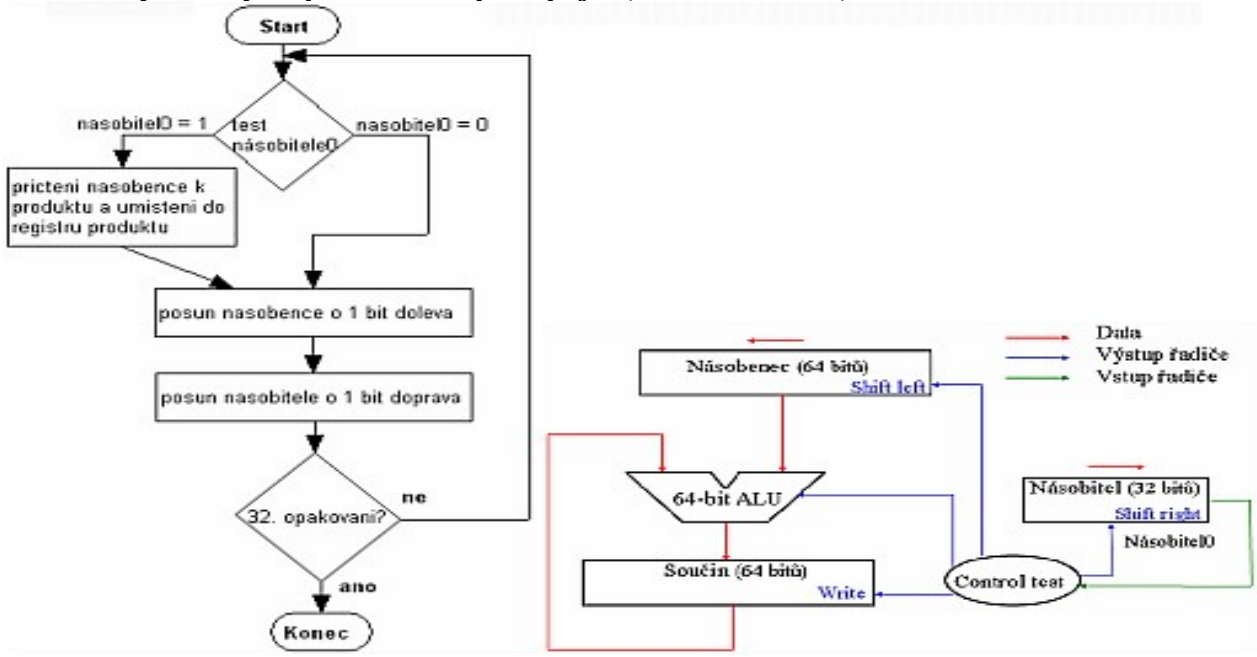
Algoritmus:



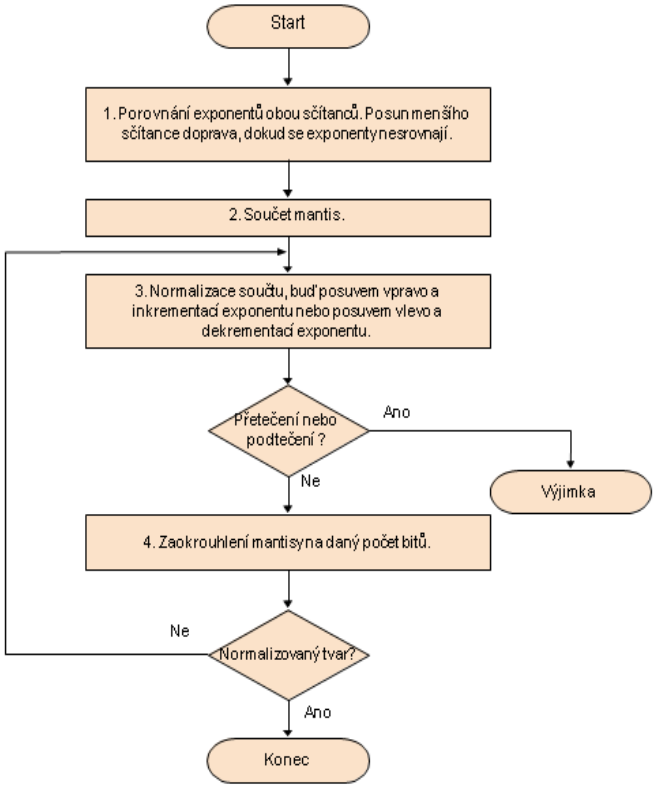
HW jednotka:



Formou vývojového diagramu vysvětlete klasický alg. násobení bin. čísel s testem 1 bitu a posuvem o 1 bit. Navrhněte operační jednotku pro provadeni tohoto alg. a pokuste se zduvodnit z které strany se obvykle při testovani postupuje. (Př. 7 – Snímek3)

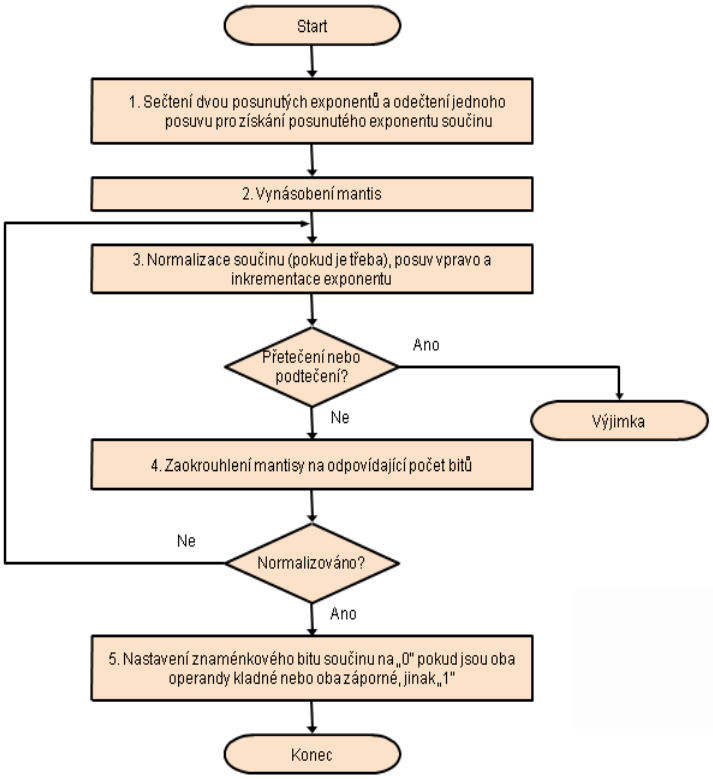


Popište alg. sčítání 2 čísel float (vývoják) (Př. 8 - Snímek35)

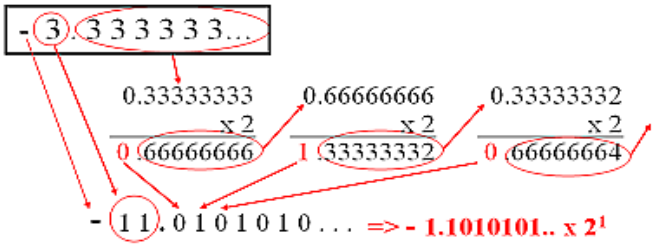


Odečítání – asi stejné, akorát krok 2 bude rozdíl mantis :-)
1 – porovnání exp. obou odečítanců
3 – normalizace rozdílu

Popište alg. násobení 2 čísel float (vývoják) (Př. 8 – Snímek44)



Popiště alg. pro převod čísel v pohyblivé řádové čárce (float) do formy celočíselné (diagram + hw jednotka) (2)



- Mantisa: 101 0101 0101 0101 0101 0101
- Znaménko: záporné => 1
- Exponent: $1+127 = 128_{10} = 1000\ 0000_2$

1	1000 0000	101 0101 0101 0101 0101 0101

Jak z toho namalovat vývoják nevim :-/ ale lepší něco, než nic

Převod z binárního tvaru do formy float

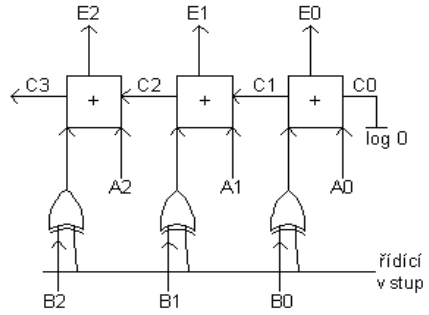
0	0110 1000	101 0101 0100 0011 0100 0010

- Znaménko: 0 => kladné
- Exponent:
  - $0110\ 1000_2 = 104_{10}$
  - Výpočet posunu: 104 - 127 = -23
- Mantisa:
  - $1 + 1x2^{-1} + 0x2^{-2} + 1x2^{-3} + 0x2^{-4} + 1x2^{-5} + \ldots$
    $= 1+2^{-1}+2^{-3}+2^{-5}+2^{-7}+2^{-9}+2^{-14}+2^{-15}+2^{-17}+2^{-22}$
    $= 1.0 + 0.666115$
- Vyjadřuje: $1.666115*2^{-23} \sim 1.986*10^{-7}$

Implicitní část mantisy

Binární sčítačka s akumulátorem (vstupuje 1 operand, druhý je součet všech předchozích)

Doplňte (navrhněte) paralelní binární sčítačku čísel v doplňkovým kódu pro operace (+,-). +,- se volí vstupním signálem. Zdůvodněte. (2) ??

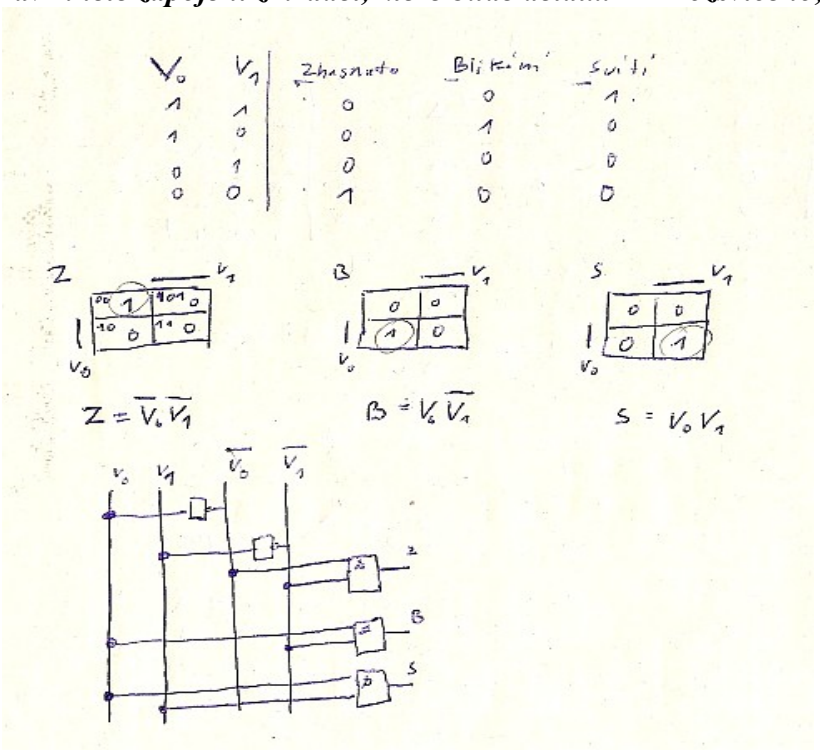


Řídící vstup – 0 = sčítání; 1 = odčítání

VIZ. PŘÍKLADY ZE CVIČENÍ

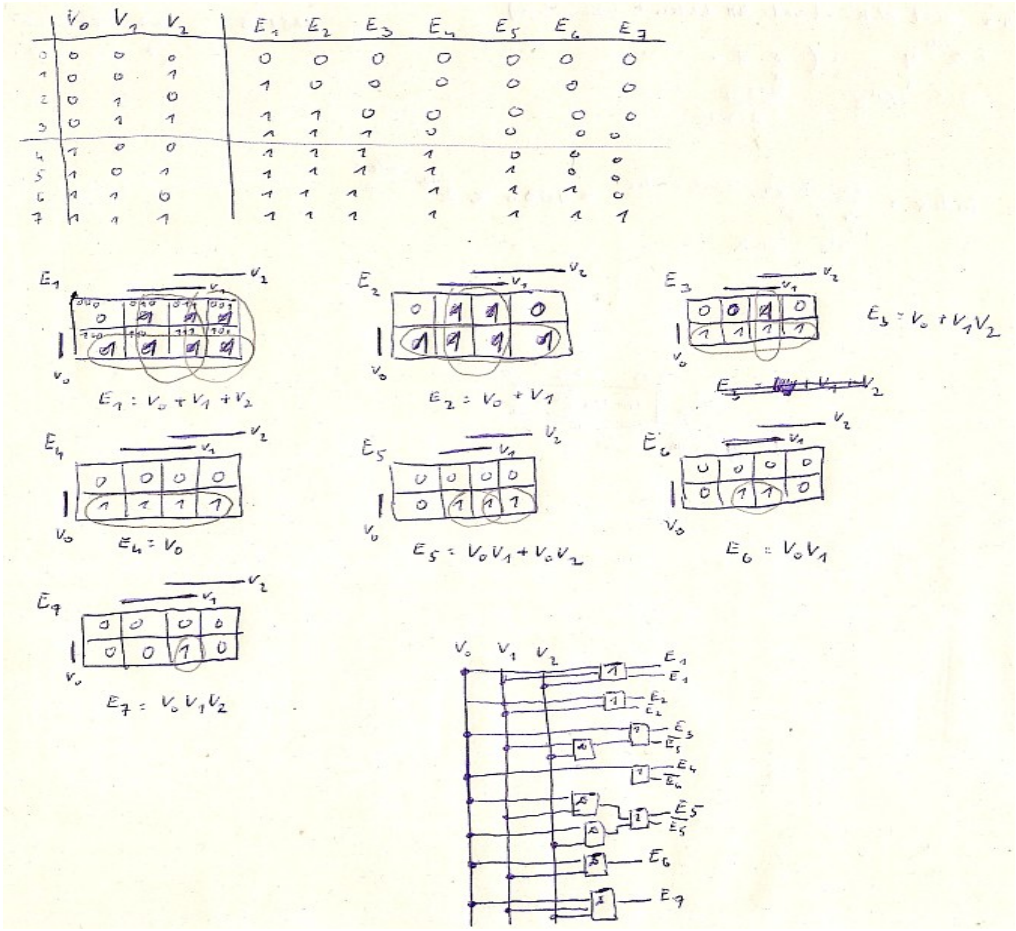
1 bitová sčítačka, která si pamatuje předchozí součet. Vstup carry in, carry out, součet. Pro pamatování stavu použít paměťový prvek (např. D)

Navrhněte zapojení z hradel, které bude dělat... 11 - rozsvíceno, 10 - blikání, 00 - zhasnuto

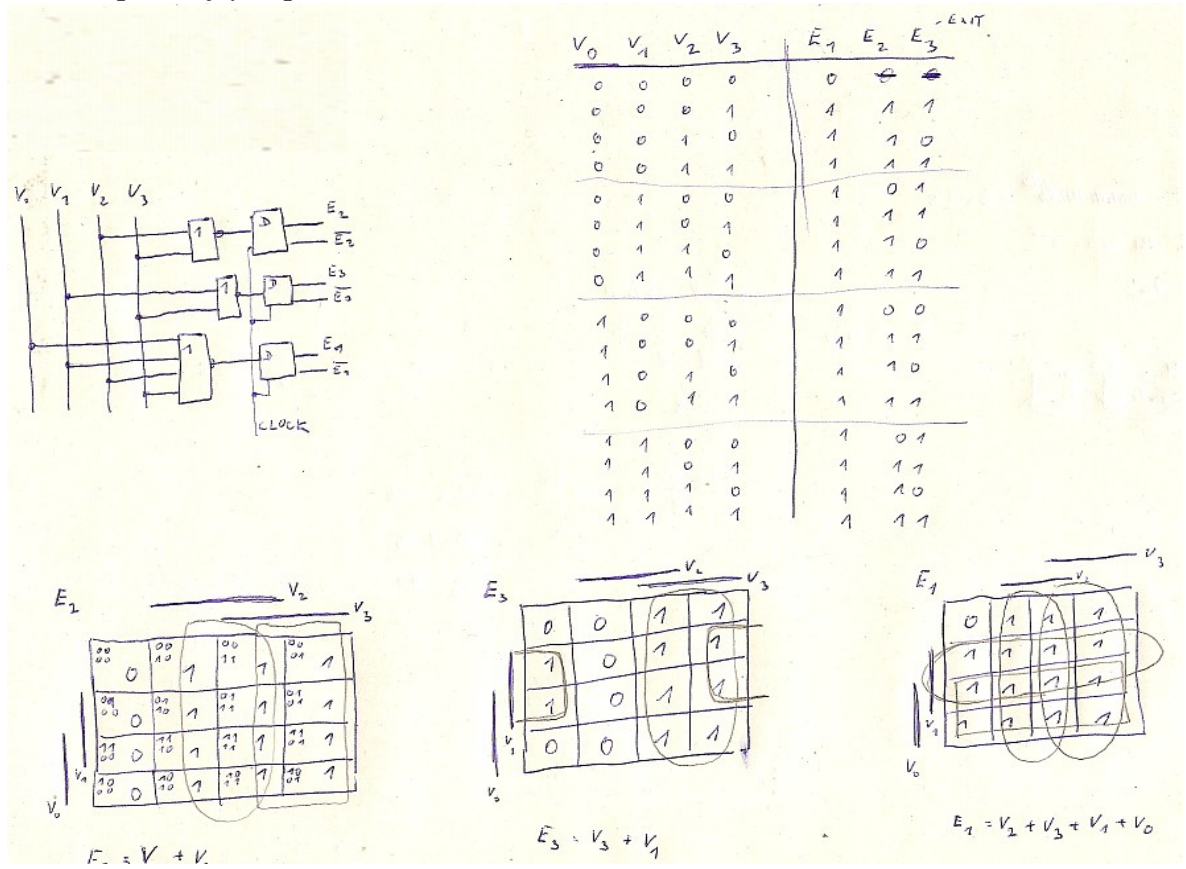


Navrhnout synchronní 3bitový čítač - „Gandalf“ … ale jak na to, to je otazka :-D

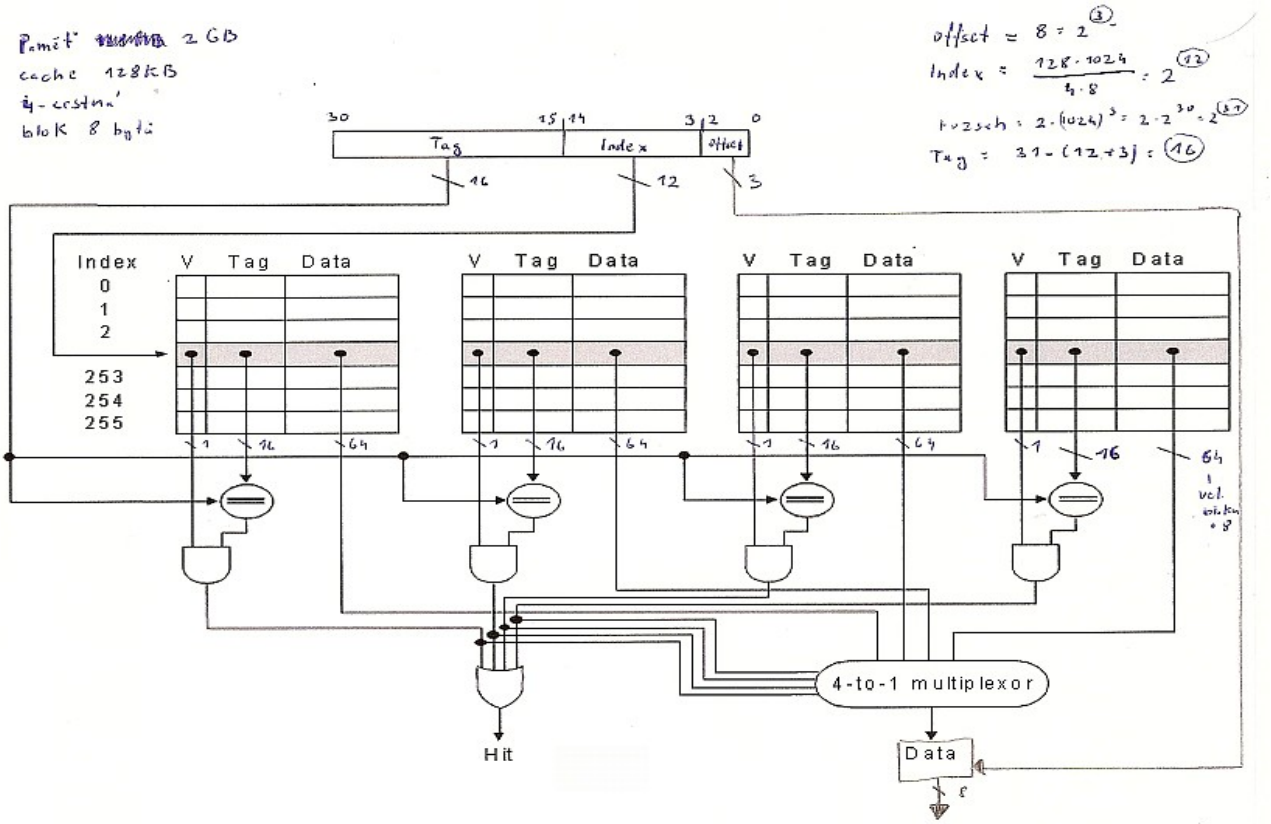
Navrhněte logický obvod s použitím hradel, který má 3 vstupy a 7 výstupů a je charakterizován funkcí: "Počet aktivních výstupů je roven binárnímu číslu na vstupu". Nakreslete schéma. (2)



Navrhněte kombinační logický obvod "priorotní funkce". Obvod má 4 vstupy (číslované od 0 do 3) a 3 výstupy. První výstup je v 1, pokud byla alespoň na jeden vstup přivedena 1. Na dalších dvou výstupech se objeví číslo vstupu na němž je přivedena 1. Pokud je 1 na více vstupech, pak na výstupu číslo vstupu s nejvyšší prioritou.

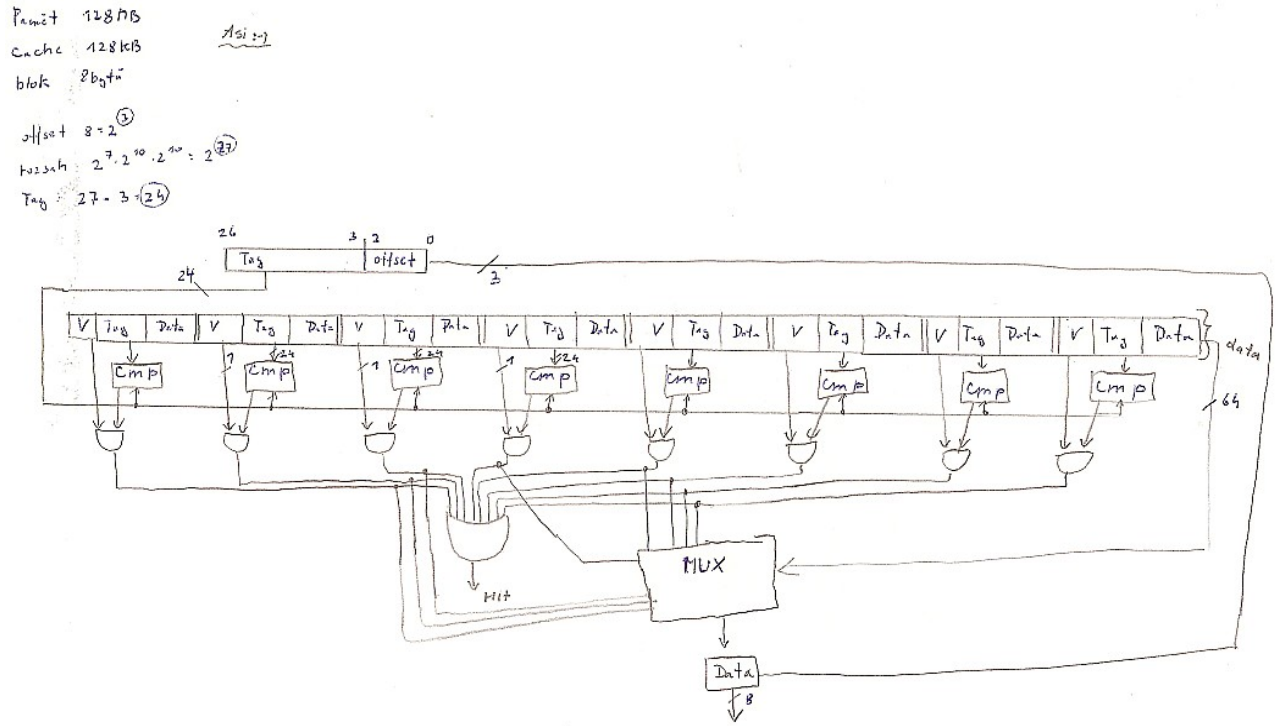


Hlavní paměť = 2GB, cache = 128KB je organizována jako 4-cestná „částečně“ asociativní cache s velikostí bloku 8 byte. Nakreslete a popiste výběrový mechanismus při operaci čtení (rozpoznat zda jsou data v paměti nebo ne) (2)



Blok **Data** dole se skládá zase z nějakých MUXů (asi 4) a vybírá data podle offsetu
V je valid bit – jsou-li data, je nastaven na 1, nejsou-li je na 0
Divná součástka s „rovná se“ je komparátor
Na cestě od Index by měl být ještě asi **Dekodér**, ale v přednášce ho neměl

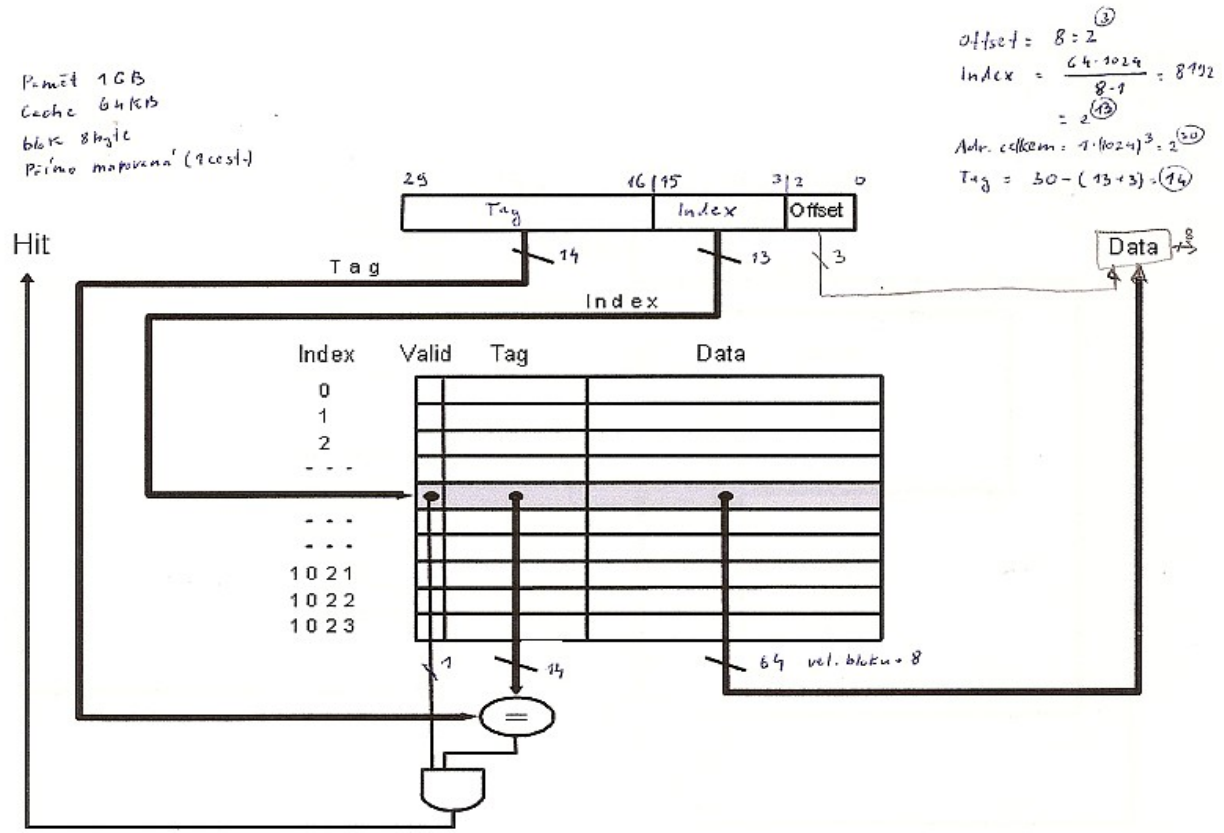
Plně asociativni CACHE, paměť 128MB, cache 128KB, blok 8 bytů



Do Multiplexeru by měla jít z každého bloku Data jedna cara s „64 draty“.. ale nevešlo se to tam :-)
Tak je to tam jenom 1x a je u toho ta svorka :-)

Hlavní paměť = 1GB, cache = 64KB je organizována jako 2-cestná „částečně“ asociativní cache s velikostí bloku 16 byte. Nakreslete a popiste výběrový mechanismus při operaci čtení (rozpoznat zda jsou data v paměti nebo ne)

PC má hlavní paměť 1GB a Cache má mít velikost 64KB a je organizována jako přímo mapovaná cache s velikostí bloku 8byte. Nakreslete a stručně popište výběrový mechanismus při operaci čtení



Vysvetlete vyznam a fci bloku, ktery je na obrazku vyznacen. jakym zpusobem je pouzivan. [4]
(oznacen byl ctverec ve fazi decode (druhy ctverec zleva)) (2)