VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

NÁVRH POKROČILÉ ARCHITEKTURY PROCESORU VE VHDL

SEMESTRÁLNÍ PROJEKT
MASTER'S THESIS

AUTOR PRÁCE Bc. Jaroslav Novotný
AUTHOR

BRNO 2007

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

NÁVRH POKROČILÉ ARCHITEKTURY PROCESORU VE VHDL

ADVANCED PROCESSOR DESIGN IN VHDL

SEMESTRÁLNÍ PROJEKT

MASTER'S THESIS

AUTOR PRÁCE Bc. Jaroslav Novotný
AUTHOR

VEDOUCÍ PRÁCE Ing. Martin Straka
SUPERVISOR

BRNO 2007

Abstrakt

Tato práce se zabývá návrhem a částečně i implementací pokročilé architektury procesoru v jazyce VHDL. Práce popisuje základní architektury procesoru a pamětí CACHE. Dále se zabývá zřetězeným zpracováním instrukcí a stručným popisem jazyka VHDL. Na závěr práce je uveden návrh jednoduchého procesoru v jazyce VHDL a jeho rozšíření o zřetězené zpracování instrukcí. Také je diskutována možnost použití datové a instrukční paměti CACHE.

Klíčová slova

Procesor, zřetězené zpracování instrukcí, cache, dvoucestná asociativní cache, VHDL

Abstract

The aim of this paper is to design and partially implement an advanced processor architecture in the VHDL language. It describes basic architectures of processors and CACHE memories. It also deals with the serial implementation of instructions and briefly describes the VHDL language. Finally, a design of a simple VHDL language processor and its upgrade by adding serial implementation of instructions is presented. A possiblity to use the data and instruction CACHE memory is discussed too.

Keywords

Processor, pipelining, cache, Two-Way Set Associative Cache, VHDL

Návrh pokročilé architektury procesoru ve VHDL

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením Ing. Martina Straky. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

……
Jméno Příjmení
Datum

Poděkování

Rád bych poděkoval svému vedoucímu Ing. Martinu Strakovi za čas věnovaný konzultování tohoto semestrálního projektu.

© Jaroslav Novotný, 2007.

Tato práce vznikla jako školní dílo na Vysokém učení technickém v Brně, Fakultě informačních technologií. Práce je chráněna autorským zákonem a její užití bez udělení oprávnění autorem je nezákonné, s výjimkou zákonem definovaných případů.

Obsah

Obsah 1
1 Úvod 2
2 Architektury procesorů a pamětí cache 3
2.1 Procesor 3
2.1.1 Von Neumann, Harvardská architektura 3
2.1.2 CISC, RISC 4
2.1.3 Zásobníková, registrová, akumulátorová architektura 6
2.1.4 Sub-skalární, Skalární, Super-skalární procesory 7
2.1.5 Zřetězené zpracování 9
2.2 Paměti cache 11
2.2.1 Plně asociativní cache 13
2.2.2 N-cestně asociativní CACHE 14
2.2.3 Přímo mapovaná CACHE 15
3 Jazyk VHDL 16
3.1 Entita, architektura 16
3.2 Strukturální popis 16
3.3 Behaviorální popis – popis chování 17
3.4 Popis na úrovni datových toků 17
4 Návrh procesoru a pipeline 18
4.1 Návrh jednoduchého procesoru 18
4.1.1 Instrukční sada 18
4.1.2 Jednotlivé bloky procesoru 20
4.1.3 Blokové schéma jednoduchého procesoru 22
4.2 Procesor se zřetězeným zpracováním instrukcí 23
5 Návrh paměti cache 24
5.1 Návrh dvoucestné asociativní paměti cache 24
6 Závěr 25
Literatura 26
Příloha A 27

1 Úvod

Málokterá jiná disciplína prošla v posledních letech tak bouřlivým vývojem jako architektura procesorů. Univerzální procesory určené nejdříve pro domácí počítače a později pro osobní počítače, podnikové síťové servery i paralelní superpočítače zaznamenaly obrovský vzestup výkonnosti. Kromě dospívající křemíkové technologie a růstu hodinových kmitočtů k tomu přispěly právě změny v architektuře procesorů. Vývoj šel od klasického modelu von Neumana se 4, 8, 16 bity k řetězenému zpracování několika instrukcí současně ve skalárních a super-skalárních procesorech se 32 a 64 bity. Vše nasvědčuje tomu, že trend zrychlování a vylepšování jader procesoru nepoleví ani v budoucnu. Proto pochopení architektur procesorů a schopnost navrhnout vlastní procesor otevírá veliké možnosti na trhu práce. Kromě toho, že jde po teoretické stránce o velmi zajímavou oblast, po stránce praktické je tvorba procesoru a jeho vylepšovaní činností velmi zábavnou.

Tato práce se zabývá návrhem a částečně i implementací pokročilé architektury procesoru v jazyce VHDL. Pokročilou architekturou je myšleno využití zřetězeného zpracování instrukcí a doplnění procesoru o datovou a instrukční paměť cache.

Dokument je členěn do několika kapitol. V první polovině práce jsou uvedeny základní informace potřebné k pochopení architektur procesorů a paměti cache. Okrajově je zde také popsán princip jazyka VHDL. V druhé polovině práce je uveden návrh jednoduchého procesoru, jeho rozšíření o zřetězené zpracování instrukcí a doplnění o paměť cache. V závěru jsou zhodnoceny dosažené cíle a diskutováno další pokračování práce.

2 Architektury procesorů a pamětí cache

2.1 Procesor

Procesor (CPU – Central Processing Unit) je ústřední výkonnou jednotkou počítače, která čte z paměti instrukce a na jejich základě vykonává program.

Součásti procesoru

- řadič nebo řídicí jednotka, jejíž jádro zajišťuje řízení činnosti procesoru v návaznosti na povely programu (generování signálů ovládajících zbývající komponenty CPU), tj. načítání instrukcí, jejich dekódování (zjištění typu instrukce), načítání operandů instrukcí z operační paměti a ukládání výsledků zpracování instrukcí,
- sada registrů (v řadiči) k uchování operandů a mezivýsledků. Registry dělíme na obecné (pracovní, univerzální) a řídící (např. registr adres instrukcí, stavové registry, indexregistry),
- jedna nebo více aritmeticko logických jednotek (ALU - Arithmetic-Logic Unit), které provádí s daty příslušné aritmetické a logické operace,
- některé procesory obsahují jednu nebo několik jednotek plovoucí čárky (FPU), které provádí operace v plovoucí řádové čárce.

2.1.1 Von Neumann, Harvardská architektura

Základním rozdílem mezi von Neumannovou a Harvardskou architekturou je, že Harvardská architektura má oddělenou paměť pro data a instrukce (každá zvlášť), kdežto u von Neumannovy architektury jsou instrukce a data v jedné paměti a mezi nimi se vybírá pomocí různé adresy.

Základní rysy Von Neumann

- společná paměť dat a instrukcí
- snadná manipulace s programy
- možnost psát sebemodifikující programy (současná CPU je ovšem nepodporují)

Základní rysy Harvardské architektury

- oddělená paměť pro data a pro instrukce
- k datům a instrukcím lze přistupovat paralelně

- paměti můžou mít různé organizace
- není třeba stejných parametrů pro data i program
- používané v malých mikrokonrolérech
- jednoduchá hardwarová implementace

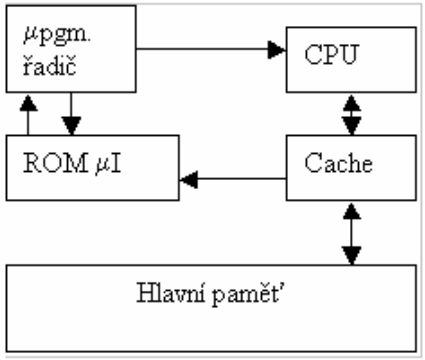
Srovnání

- paměť je slabým článkem. Neumannova architektura je nevýhodná kvůli velkému počtu přístupů do paměti - řeší se např. pomocí cache,
- rychlé moderní procesory spojují obě architektury - uvnitř procesoru je použita Harvardská architektura, kde se paměť cache dělí na paměť instrukcí a paměť pro data. Ovšem celý prostor se z venku chová jako procesor s architekturou von Neumann, protože načítá data i program z hlavní paměti najednou,
- u Harvardské architektury může procesor zároveň číst/zapisovat z/do paměti program i data.

2.1.2 CISC, RISC

CISC - Procesory s komplexním instrukčním souborem.

Operační paměť je mnohem pomalejší než procesor. Aby se výpočet nezpomaloval několika přístupy do paměti při vykonání jedné operace, vznikly složitější instrukce. Instrukční soubor je rozšířen o tyto složitější instrukce, aby pro celou operaci stačilo jen načtení jediné instrukce. Řadič procesoru je navrhnut na základě řídicí paměti, která uchovává mikroprogramy jednotlivých strojových instrukcí. Vykonání strojové instrukce probíhá jako posloupnost mikrooperací. Posloupnost mikroinstrukcí je dána v řídicí paměti. Řídicí paměti (většinou ROM) jsou mnohem rychlejší než operační paměť. S rozvojem mikroelektroniky se řídicí paměti integrovaly s procesorem do jednoho čipu stejně jako vyrovnávací paměť (cache) instrukcí nebo údajů. Procesory CISC začaly využívat zřetězené zpracování mikroinstrukcí a později i zřetězené zpracování, proto jsou v současnosti nejvíce rozšířeny (Pentium, PentiumPro).



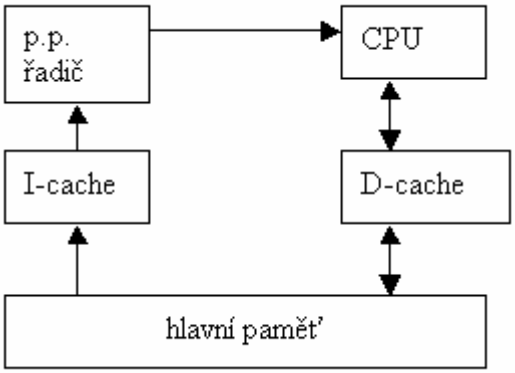
Obrázek 1 Blokové schéma procesoru CISC

RISC - Procesory s redukovaným souborem instrukcí

Instrukční soubor obsahuje pouze jednoduché a nejčastěji používané instrukce. Program zapsaný v jednoduchých instrukcích má mít sice větší počet instrukcí, ale mnohem nižší průměrný počet taktů na instrukci.

Základní rysy

- Malý počet relativně jednoduchých instrukcí.
- V každém taktu procesoru se většinou dokončí jedna instrukce. V průměru je CPI<1,5.
- Instrukce s pevnou délkou (typicky 32 bitů) s jedním nebo jen malým počtem formátů, jednotně zakódované.
- Řízení pevnou logikou – pevně propojeným řadičem – jednoduchá hardwarová implementace.
- Operace s daty pouze nad registry, 2 zdrojové registry (nedestruktivní čtení), jeden cílový.
- Přístup do paměti pouze pomocí instrukcí přesunů (proto se také hovoří o instrukční architektuře L/S), malý počet adresových módů (3-5).
- Velký počet programově dostupných registrů (32-192).
- Nezbytnost optimalizace kódu na úrovni kompilátoru – naplánování instrukcí.



Obrázek 2 Blokové schéma procesoru RISC

2.1.3 Zásobníková, registrová, akumulátorová architektura

Rozdíl mezi těmito architekturami spočívá v uložení operandů.

Akumulátorová architektura

Založena na použití speciálního registru – akumulátoru. Tento registr je trvale uvažován jako jeden ze zdrojových operandu a zároveň jako cílový. Výhodné například pro výpočty využívající předchozích výsledků.

- nižší požadavky na prostor v instrukčním slově
- je potřeba zajistit ukládání a nahrávání hodnot do/z akumulátoru

Příklad: (A+B)*(C-D)

Acc <= R2; (C)

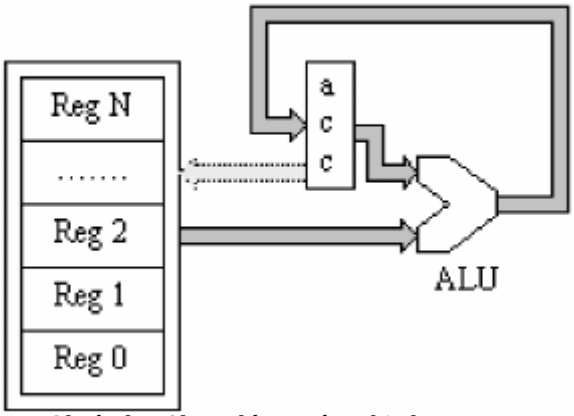
Acc <= Acc-R3; (C-D)

R4 <= Acc;

Acc <= R0; (A)

Acc <= Acc+R1; (A+B)

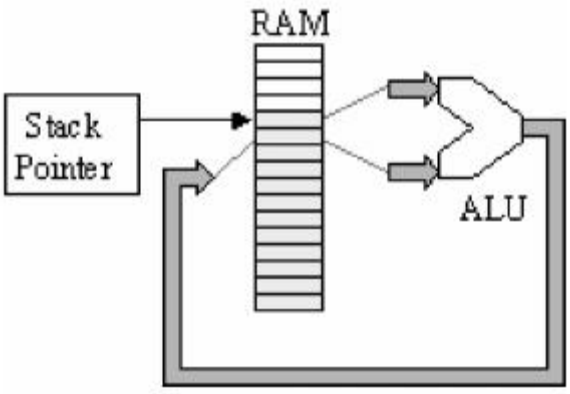
Acc <= Acc*R4



Obrázek 3 Akumulátorová architektura procesoru

Zásobníková architektura

Zásobníková architektura ukládá operandy na vrchol zásobníku, instrukce pak provádí operace nad zásobníkem, např. A+B proběhne jako PUSH A, PUSH B, ADD, tedy sečtení dvou čísel na vrcholu zásobníku, výsledek je uložen na vrcholu.



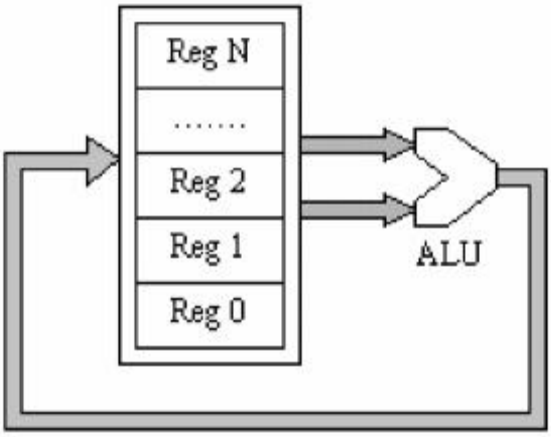
Obrázek 4 Zásobníková architektura rocesoru

Registrová architektura

Při použití této architektury je jako operand přístupný kterýkoliv registr z registrového slova. Toto vede na kratší a efektivnější programy. Nevýhodou tohoto přístupu je nutnost přítomnosti všech operandů (dva zdrojové a jeden cílový) v instrukčním slově, což vede na větší počet bitů instrukčního slova.

Pro svou jednoduchou hardwarovou realizovatelnost a zároveň pohodlnost programátora je tato metoda v praxi nejvíce rozšířena.

Příklad: (A+B)*(C-D)
R4 <= R0+R1 ; (A+B)
R5 <= R2-R3 ; (C-D)
R6 <= R4*R5



Obrázek 5 Registrová architektura procesoru

2.1.4 Sub-skalární, Skalární, Super-skalární procesory

Historický vývoj mikropočítačů zle rozdělit do třech fází. První fáze je reprezentována tradičními procesory se sekvenčním vydáváním a sekvenčním prováděním instrukcí, pro které platilo, že doba provedení programu je dána prostým aritmetickým součtem časů trvání jednotlivých instrukcí, pohybujících se od jednotek do desítek taktů (period hodinového signálu). Na vytvoření koncepce těchto procesorů v letech 1946-1952 měl hlavní podíl von Neuman, matematik maďarského původu, takže se často hovoří o von Neumanově (subskalárním) procesoru. Cestou za vyšší výkonností bylo sekvenční vykonávání instrukcí nahrazeno paralelním, a to pomocí zřetězeného zpracování nebo zavedením několika funkčních jednotek. Procesory, které v této druhé fázi vznikly, jsou označovány jako skalární procesory, poněvadž v jednom taktu byla k vykonání v některé jednotce vydávána vždy nejvýše jedna instrukce (sekvenční vydávání), vykonávání instrukce však probíhalo překrytě nebo i paralelně (např. operace v pevné a pohyblivé řádové čárce). Jelikož v některých taktech nemůže být z důvodů popsaných dále vydána žádná nová instrukce, trvá provedení programu více taktů než je počet instrukcí a jeho přesný odhad je obtížný.

V dalším vývoji byl stupeň paralelismu zvýšen ještě dále použitím několika zřetězených funkčních jednotek. Brzy se však ukázalo, že sekvenční vydávání instrukcí nestačí zásobovat tyto paralelně

pracující funkční jednotky. Vývoj tudíž dospěl do třetí fáze charakterizované paralelním vydáváním instrukcí i paralelním prováděním instrukcí u takzvaných superskalárních procesorů.

Sub-skalární procesory

U těchto procesorů je vždy rozpracována jen jedna instrukce. CPI (počet cyklů na instrukci) je větší jedné.

Čas →	1	2	3	4	5	6	7	8	9	10	11	12
i_1	S_1	S_2	S_3									
i_2				S_1	S_2	S_3	S_4	S_5				
i_3									S_1	S_2	S_3	S_4

Obrázek 6 klasické nezřetězené zpracování instrukcí. i-instrukce, S doba provádění instrukce

Skalární procesory

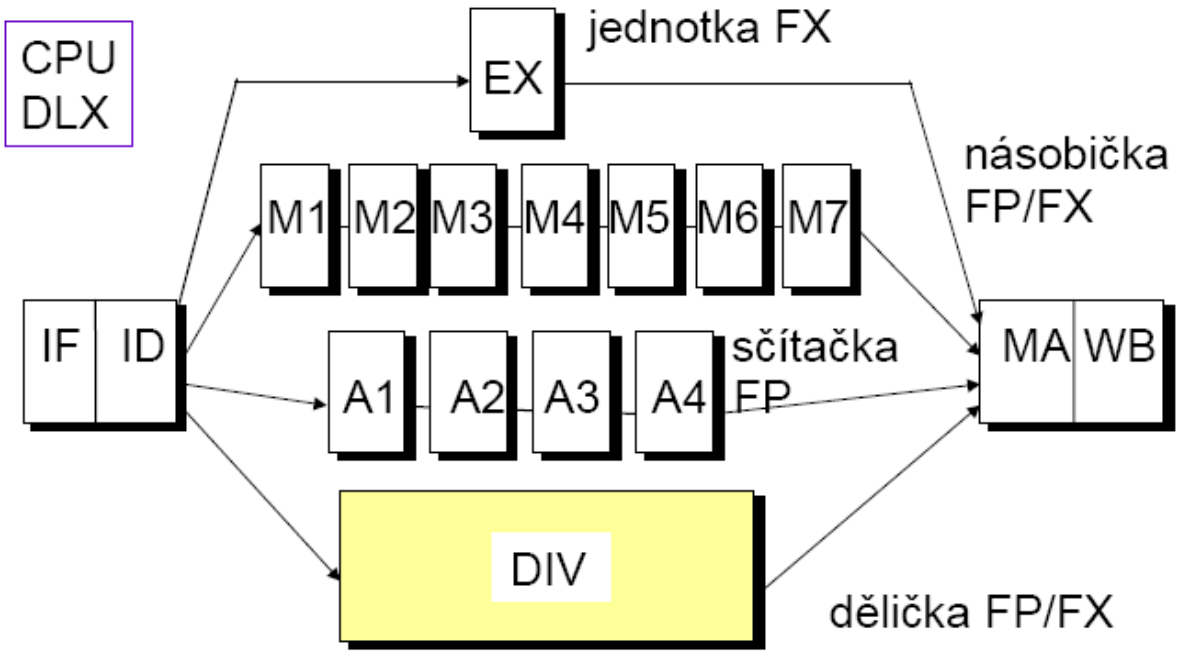
Tyto procesory využívají zřetězeného zpracování instrukcí (tzv. pipelining, viz. dále) popřípadě paralelních zpracování instrukcí ve více funkčních jednotkách.

Pokud linku procesoru rozdělíme na dva stupně schopné pracovat samostatně např. stupeň F (preFetch, předvýběr) a E (Execute, provedení), může se činnost těchto stupňů překrývat

Čas →	1	2	3	4	5	6	7	8	9	10
i_1	F	E	E							
i_2			F	E	E	E	E			
i_3							F	E	E	E

Obrázek 7 Časový diagram zřetězené linky se 2 stupni, CPI = 3,33

Pokud použijeme v lince procesoru více funkčních jednotek, je možné provádět časově náročné operace (typicky FP operace, dělení…) paralelně a tím dosáhnout zrychlení. Dokonce je možné rozdělit výkonnou jednotku na více stupňů (př. násobičku, sčítačku).



Obrázek 8 Zřetězené zpracování instrukcí FX, B, L/S, FP

Super-skalární procesory

Super-skalární procesory využívají paralelní vydávání instrukcí i paralelní provádění instrukcí.

Rysy super-skalárních procesorů

- paralelní řetězené linky: paralelní dekódování, více instrukcí vydáváno současně do řady FJ, paralelní provádění a dokončování instrukcí.
- Dynamické plánování instrukcí: přeskládání instrukcí v HW za běhu programu: po dekódování instrukce čekají na operandy (zpracování stylem data-flow). Instrukce jsou prováděny a dokončují se v jiném pořadí než jsou uvedeny v programu (out-of-order, OOO).
- Přejmenování registrů v HW.
- Spekulativní zpracování instrukcí na základě dynamické predikce skoků v HW, spekulativní načítání.
- Přístupy do paměti v jiném pořadí než v programu.

2.1.5 Zřetězené zpracování

Zřetězené zpracování je založené na tom, že se vykonání nějaké operace rozdělí na sekvenční kroky nejlépe stejného trvání, přičemž v každém kroku se používají samostatné technické prostředky. Tak je možné, že v jednom okamžiku je rozpracováno několik operací, každá je ovšem v jiném stavu rozpracování a využívá jiný stupeň řetězu technických prostředků.

V praxi se zřetězené zpracování (pipelining) realizuje tak, že se vetší úsek kombinační logiky rozdělí na menší části, mezi které je vložen oddělovací registr. Při vložení k registrových stupňů lze dosáhnout ideálně až k+1 násobné zrychlení výpočtu – v každém taktu hodinového signálu je k dispozici jeden výsledek.

Pro dosáhnutí co nejlepších výsledků je dobré dodržet následující doporučení:

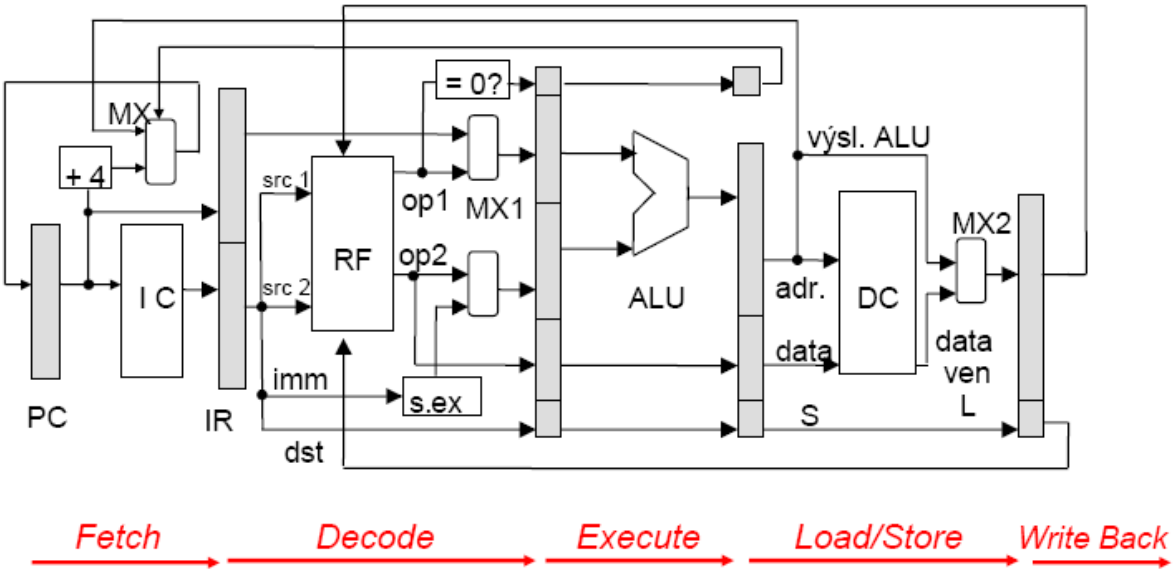
- velikost logiky v jednotlivých stupních by měla být vyvážená,
- zajistit nepřetržitý přísun dat ke zpracování.

Na celkové dosažené zrychlení má dále vliv:

- nezbytné zastavování linky (v reálné situaci není většinou možné zpracování jedné operace za jeden hodinový cyklus),
- náběh a doběh řetězeného zpracování při konečném počtu N zpracovaných položek,
- zpoždění na oddělovacích registrech.

Jedna z hlavních oblastí využití zřetězeného zpracování je právě konstrukce rychlých linek zpracovávajících instrukce v procesoru. Za jakýsi standard v této oblasti je považována pipeline s pěti stupni, implementována například v prvních procesorech MIPS. Jedná se o následující stupně:

1. Instruction fetch - vyzvednutí instrukce
2. Decode - dekódování instrukce, zároveň se načítají registry
3. Execute - provedení instrukce
4. Load/Store – čtení/zápis z paměti
5. Writeback - zápis výsledku do registrů



Obrázek 9 Příklad zřetězené linky (šedou barvou jsou znázorněny oddělovací registry)

Bohužel při rozdělení linky oddělovacími registry vznikají konflikty způsobené snahou využívat některé části linky dvěma rozpracovanými instrukcemi současně. Řešení těchto konfliktů je širší problematika, kterou se tato práce nezabývá (viz závěr).

2.2 Paměti cache

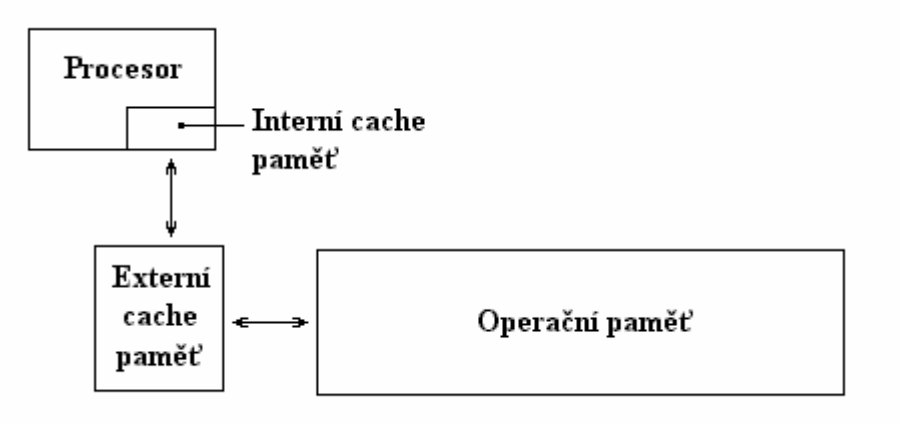
Cache paměť je rychlá vyrovnávací paměť mezi rychlým zařízením (např. procesor) a pomalejším zařízením (např. operační paměť). V dnešních počítačích se běžně používají dva druhy cache pamětí:

Externí (sekundární, L2) cache

Externí cache paměť je paměť, která je umístěna mezi pomalejší operační pamětí a rychlým procesorem. Tato paměť je vyrobena jako rychlá paměť SRAM a slouží jako vyrovnávací paměť u počítačů s výkonným procesorem, které by byly bez ní operační pamětí velmi zpomalovány. První externí cache paměti se objevují u počítačů s procesorem 80386. Jejich kapacita je 32 kB popř. 64 kB. S výkonnějšími procesory se postupně zvyšuje i kapacita externích cache pamětí na 128 kB, 256 kB, 512 kB. Externí cache paměť je osazena na základní desce počítače (výjimku tvoří procesory Pentium Pro a Pentium II, které mají externí cache paměť integrovánu v pouzdře procesoru). Její činnost je řízena řadičem cache paměti.

Interní (primární, L1) cache

Interní cache paměť je paměť, která slouží k vyrovnání rychlosti velmi výkonných procesorů a pomalejších pamětí. Tento typ cache paměti je integrován přímo na čipu procesoru a je také realizován pomocí paměti SRAM. Interní cache paměť se objevuje poprvé u procesoru 80486 s kapacitou 8 kB. Takovýto procesor musí mít v sobě integrován také řadič interní cache paměti pro řízení její činnosti.



Obrázek 10 Umístění paměti cache mezi procesor a operační paměť

Práce cache paměti vychází ze skutečnosti, že program má tendenci se při své práci určitou dobu zdržovat na určitém místě paměti, a to jak při zpracování instrukcí, tak při načítání (zapisování) dat z (do) paměti. Je-li požadována nějaká informace z paměti, je nejdříve hledána v cache paměti (interní, pokud existuje, a následně v externí). Pokud požadovaná informace není přítomna v žádné z cache pamětí, je zavedena přímo z operační paměti. Kromě momentálně požadované informace se však do cache paměti zavede celý blok paměti, takže je velká pravděpodobnost, že následně požadované informace již budou v cache paměti přítomny. Pokud dojde k zaplnění cache paměti a je potřeba zavést další blok, je nutné, aby některý z bloků cache paměť opustil. Nejčastěji se k tomuto používá LRU (Least Recently Used) algoritmu, tj. algoritmu, který vyřadí nejdéle nepoužívaný blok.

Cache paměti bývají organizovány jako tzv. asociativní paměti. Asociativní paměti jsou tvořeny tabulkou (tabulkami), která obsahuje vždy sloupec, v němž jsou umístěny tzv. tagy (klíče), podle kterých se v asociativní paměti vyhledává. Dále jsou v tabulce umístěna data, která paměť uchovává, a popř. další informace nutné k zajištění správné funkce paměti. Např.:

- informace o platnosti (neplatnosti) uložených dat,
- informace pro realizaci LRU algoritmu.

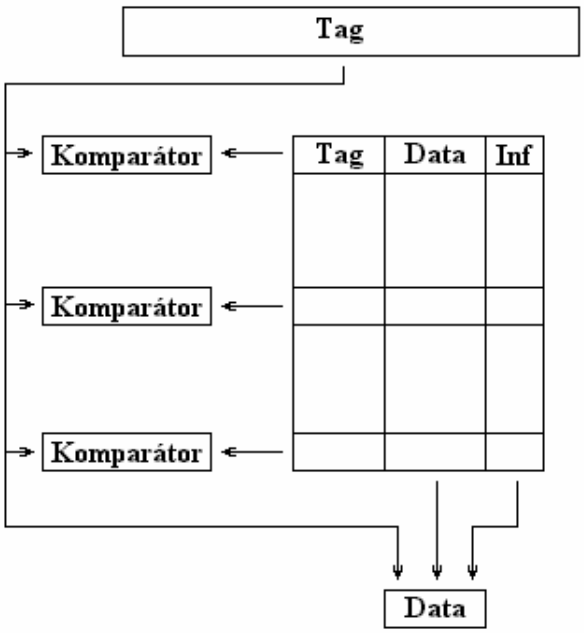
Při přístupu do cache paměti je nutné zadat adresu, z níž data požadujeme. Tato adresa je buď celá, nebo její část považovaná za tag, který se porovnává s tagy v cache paměti.

Cache paměti jsou konstruovány jedním ze tří způsobů – plně asociativní cache, n-cestně asociativní cache, přímo mapovaná CACHE.

Podle způsobu práce při zapisování dat lze cache paměti ještě rozdělit do dvou skupin:

1. **Write-through**: cache paměti, u kterých v případě zápisu procesoru do cache paměti dochází okamžitě i k zápisu do operační paměti. Procesor tak obsluhuje jen zápis a o další osud dat se stará cache paměť.
2. **Write-back**: cache paměti, u nichž jsou data zapisována do operační paměti až ve chvíli, kdy je to třeba, a nikoliv okamžitě při jejich změně. K zápisu dat do operační paměti tedy dochází např. v okamžiku, kdy je cache zcela zaplněna a je třeba do ní umístit nová data. Tento způsob práce cache paměti vykazuje oproti předešlému způsobu vyšší výkon.

2.2.1 Plně asociativní cache



Obrázek 11 plně asociativní cache

U plně asociativní cache paměti je celá adresa, ze které se budou číst data (popř. na kterou se budou data zapisovat), brána jako tag. Tento tag je přiveden na vstup komparátorů (zařízení

realizující porovnání dvou hodnot) společně s tagem v daném řádku tabulky. Pokud některý z tagů v tabulce je shodný se zadaným tagem na vstupu, ohlásí odpovídající komparátor shodu a znamená to, že požadovaná informace je v cache paměti přítomna a je možné ji použít. Pokud všechny komparátory signalizují neshodu, je to známka toho, že požadovaná informace v cache paměti není a je nutné ji zavést odjinud (externí cache paměť, operační paměť).

Tento způsob cache paměti má své nevýhody:

- Je nutné velké množství komparátorů.
- Vzhledem k tomu, že se musí v každém řádku tabulky uchovávat celý tag, musí mít cache paměť velkou kapacitu, do které se tyto tagy ukládají a kterou není možné využít k uchovávání dat.

Z těchto důvodů se plně asociativní paměti prakticky nepoužívají.

2.2.2 N-cestně asociativní CACHE

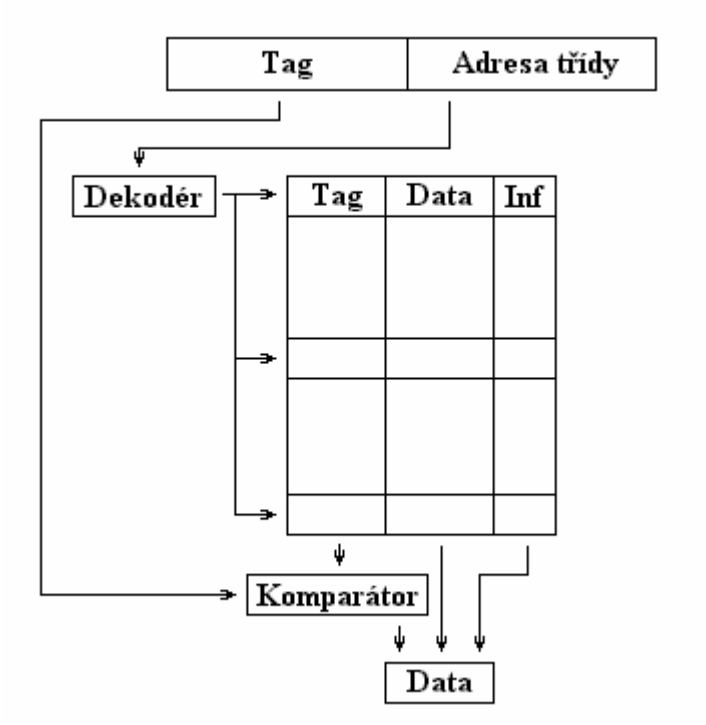
N-cestně asociativní paměti pracují tak, že zadaná adresa se rozdělí na dvě části:

1. tag
2. adresa třídy

Adresa třídy je přivedena na n dekodérů (zařízení, které na základě vstupní hodnoty vybere jeden ze svých výstupů, na který umístí hodnotu log. 1, a na ostatní výstupy umístí hodnotu log. 0), které v každé tabulce vyberou jeden řádek. Z těchto řádků se potom vezmou příslušné tagy a komparátorem se porovnají se zadaným tagem. Podobně jako u plně asociativních cache pamětí pokud jeden z komparátorů signalizuje shodu, je informace v cache paměti přítomna. V opačném případě je nezbytné informaci hledat jinde.

N-cestně asociativní paměti částečně eliminují nevýhody plně asociativních cache pamětí a v současnosti jsou nejpoužívanějším typem cache pamětí.

2.2.3 Přímo mapovaná CACHE



Obrázek 12 přímo mapovaná cache

Přímo mapovaná cache paměť je speciální případ n-cestně asociativní cache paměti pro n=1. Zadaná adresa je opět rozdělena na tag a adresu třídy. Adresa třídy je přivedena na vstup dekodéru, který podle ní vybere jeden řádek v tabulce. Tag na tomto řádku je následně porovnán se zadaným tagem, čímž se rozhodne o přítomnosti resp. nepřítomnosti informace v cache paměti.

Přímo mapovaná cache ve srovnání s n-cestně asociativní cache pamětí vykazuje nižší výkon, a proto její použití není dnes příliš časté.

3 Jazyk VHDL

VHDL je programovací jazyk sloužící pro popis hardware. Používá se pro návrh a simulaci digitálních integrovaných obvodů, například programovatelných hradlových polí (CPLD, FPGA, …), nebo různých zákaznických obvodů (ASIC).

Jazyk VHDL byl původně vyvinut pro vojenské účely. V roce 1987 se stal standardem IEEE, v roce 1997 byl revidován tak, že použitelný i pro návrh analogových obvodů. Jedná se o typovaný programovací jazyk disponující prostředky pro popis paralelismu, konektivity a explicitní vyjádření času. Jazyk se používá jak pro simulaci obvodů, tak i pro popis integrovaných obvodů, které se mají vyrábět.

Zkratka VHDL znamená VHSIC Hardware Description Language (česky jazyk pro popis hardware), kde VHSIC je zkratka z Very-High-Speed Integrated Circuit (česky velmi rychlé integrované obvody).

3.1 Entita, architektura

Z těchto dvou součásti se skládá každý obvod popsaný jazykem VHDL. Entita definuje rozhraní obvodu, tím se rozumí jeho vstupy a výstupy což odpovídá schématické značce obvodu. Architektura definuje, co má daný obvod dělat. Každý obvod nebo systém popsaný v jazyce VHDL obsahuje minimálně jeden pár klíčových slov „entity - architecture“. Tato kombinace se obvykle nazývá „design entity“ - „návrhová entita“. Rozsahem větší návrhy se skládají z několika návrhových entit, které jsou spojeny do výsledného obvodu. Spojování návrhových entit může být hierarchické: jedna návrhová entita může být složena z jiných, které se tak nalézají na nižší hierarchické úrovni.
Při popisu chování obvodu je možné pracovat na různých úrovních abstrakce – strukturální, na úrovni datových toků a na behaviorální úrovni abstrakce.

3.2 Strukturální popis

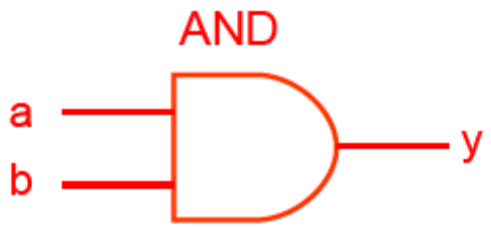
Architektura obsahuje pouze instance komponent. V podstatě říkáme, z čeho je výsledné zařízení skládá - použijeme hotové funkční bloky a propojíme je. Lze využít hierarchického přístupu, kdy se dílčí součásti mohou opět skládat z více částí. Komponenty na nejnižší úrovni hierarchie jsou vždy popsány behaviorálně.

3.3 Behaviorální popis – popis chování

Architektura je při popisu na úrovni chování obvodu složena z jednoho nebo více procesů. Cílem na této úrovni abstrakce je pouze popsat, jak se mění výstupy v závislosti na změnách vstupních signálů, přičemž nemusí být zřejmá hardwarová realizace. Toto je výhodné pro popis složitějších HW obvodů. Nevýhodou tohoto přístupu může být problém se syntézou do cílové technologie.

Příklad : popis hradla AND

```
Entity and_gate is
port( a, b : in std_logic; y : out std_logic);
end and_gate;
Architecture behv of and_gate is
begin
    process (a,b)
    begin
        y <= a AND b;
    end process;
end behv;
```



Obrázek 13 Schématická značka navržené součástky

3.4 Popis na úrovni datových toků

Úroveň popisu datových toků (data flow) je známá mnoha návrhářům číslicových obvodů. Na této úrovni je nejsnadnější popis pohybu dat v navrhovaném systému. Uvnitř většiny číslicových systémů jsou registry, takže úroveň toku dat popisuje, jak jsou informace předávány mezi registry v obvodu.

4 Návrh procesoru a pipeline

4.1 Návrh jednoduchého procesoru

Při navrhování jednoduchého procesoru jsem zvolil kombinaci RISC, Harvardská architektura, registrová architektura. Vše pro jednoduchou implementovatelnost v hardware. Datová šířka procesoru je 8 bitů, dálka instrukčního slova 16 bitů. Znaménková čísla ani čísla s plovoucí desetinnou čárkou se neuvažují.

4.1.1 Instrukční sada

Kódování instrukcí je klasické pro procesory RISC. Délka instrukčního slova je 16 bitů. Uvažují se instrukce s jedním nebo dvěma parametry. Situace, kde by bylo teoreticky třeba zadat tři parametry např. vyčíslení součtu dvou registrů do jednoho (R3 = R1+R2), je řešena standardně tak, že první zadaný operand je zároveň brán jako cílový.

Seznam instrukcí jednoduchého procesoru

Aritmeticko-logické operace
– add, sub, and, or, xor, sl, sr, mul, div

Instrukce pro přesun dat
– mov přenos mezi registry
– ld, st load/store práce s daty v DC
– in, out vstupně/výstupní porty

Instrukce pro porovnání
– cmp

Instrukce pro řízení toku programu
– jmp nepodmíněný skok
– jz, jnz skok na základě příznaku Zero
– jc, jnc skok na základě příznaku Carry

Zakódování instrukcí

	Bity instrukčního slova			
Instrukce	15..12	11..8	7..4	3..0
add	0000	xxxx	kkkk	kkkk
	1111	xxxx	yyyy	0000
mul	0001	xxxx	kkkk	kkkk
	1111	xxxx	yyyy	0001
sub	0010	xxxx	kkkk	kkkk
	1111	xxxx	yyyy	0010
div	0011	xxxx	kkkk	kkkk
	1111	xxxx	yyyy	0011
and	0100	xxxx	kkkk	kkkk
	1111	xxxx	yyyy	0100
or	0101	xxxx	kkkk	kkkk
	1111	xxxx	yyyy	0101
xor	0110	xxxx	kkkk	kkkk
	1111	xxxx	yyyy	0110
mov	0111	xxxx	kkkk	kkkk
	1111	xxxx	yyyy	0111
cmp	1000	xxxx	kkkk	kkkk
	1111	xxxx	yyyy	1000
ld	1001	xxxx	kkkk	kkkk
	1111	xxxx	yyyy	1001
st	1010	xxxx	kkkk	kkkk
	1111	xxxx	yyyy	1010
in	1011	xxxx	kkkk	kkkk
	1111	xxxx	yyyy	1011
out	1100	xxxx	kkkk	kkkk
	1111	xxxx	yyyy	1100
sl0	1101	xxxx	----	0000
sl1	1101	xxxx	----	0001
sla	1101	xxxx	----	0010
slx	1101	xxxx	----	0100
rol	1101	xxxx	----	0110
sr0	1101	xxxx	----	1000
sr1	1101	xxxx	----	1001
sra	1101	xxxx	----	1010
srx	1101	xxxx	----	1100
ror	1101	xxxx	----	1110
jmp	1110	0000	aaaa	aaaa

jc	1110	1011	aaaa	aaaa
jnc	1110	1010	aaaa	aaaa
jz	1110	1001	aaaa	aaaa
jnz	1110	1000	aaaa	aaaa

tabulka 1 Tabulka zakódování instrukcí

Legenda

k - přímý operand, resp. konstanta

a - adresa skoku

x - adresa prvního zdrojového operandu v registrovém poli/resp. adresa zápisu

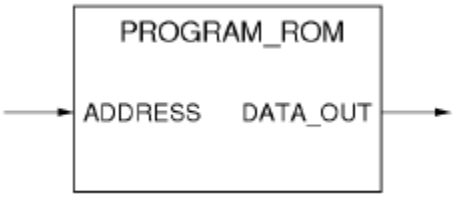
y - adresa druhého zdrojového operandu v registrovém poli

4.1.2 Jednotlivé bloky procesoru

Paměť instrukcí

Synchronní blok paměti RAM 256x16 bitů.

```
entity program_rom is
port(
  clk : in std_logic;
  address : in std_logic_vector(7 downto 0);
  data : out std_logic_vector(15 downto 0)
);
end entity program_rom;
```



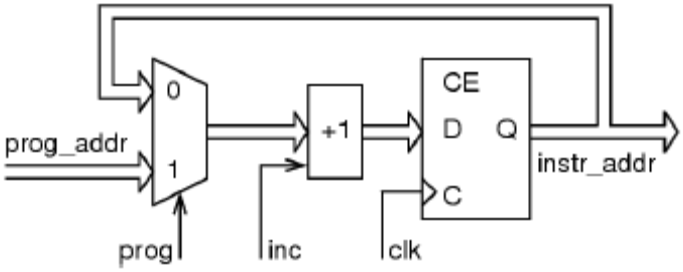
Obrázek 14 paměť instrukcí

Programový čítač

Sekvenčně načítá jednotlivé instrukce z paměti instrukcí a posílá je k dekódování a dalšímu zpracování. Procesor podporuje instrukci skoku, tzn. je nutné programový čítač vybavit vstupem pro načtení cílové adresy skoku. Tento vstup je připojený přes multiplexor.

```
mx_pc <= prog_addr when (prog='1')
else cnt_pc;

process(CLK)
begin
  if (CLK'event AND CLK='1') then
  cnt_pc <= mx_pc + inc;
  end if;
end;
```

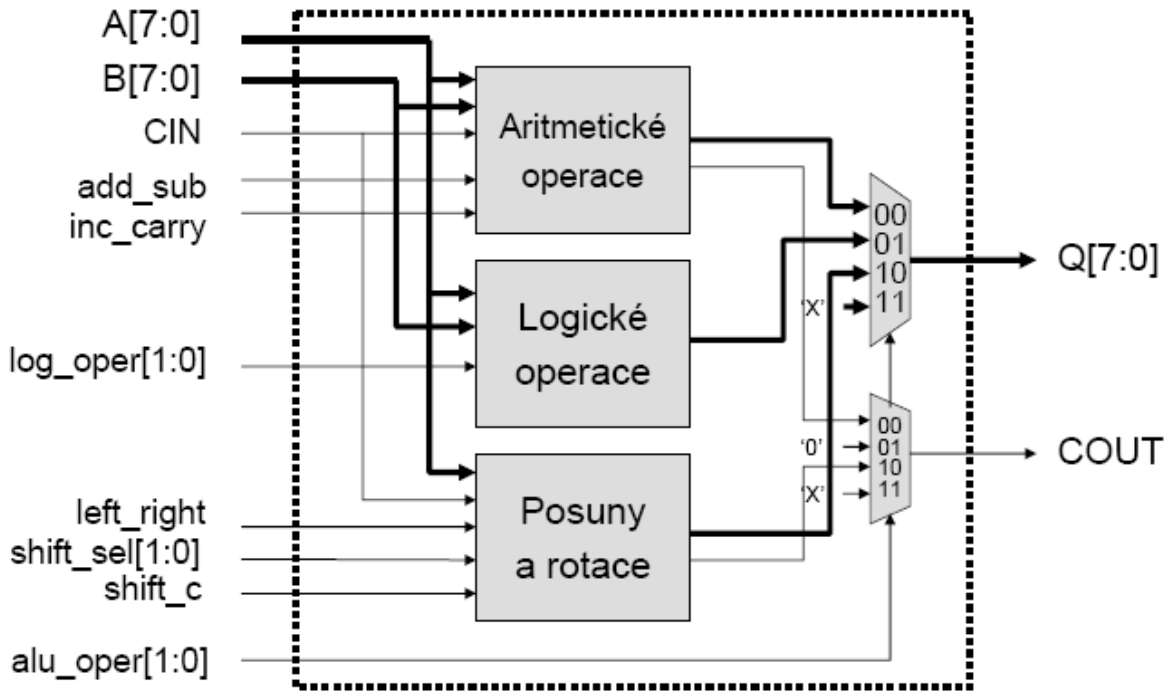


Obrázek 15 programový čítač

Aritmeticko-logická jednotka

ALU se skládá ze tří hlavních bloků. Každý blok řeší jeden typ operací – Aritmetické operace, logické operace, posuny a rotace. Na vstup bloků jsou přivedeny osmibitové operandy A, B, přenos

z vyšších řádů, a typ instrukce která se má provést. Na výstupu z ALU se vybere požadovaný výsledek (z požadované jednotky) a výsledek přenosu.

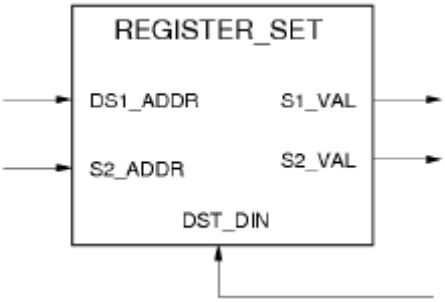


Obrázek 16 Aritmeticko logická jednotka

Registrové pole

Navrženo jako dvouportová paměť (adresa zdrojového operandu 1 je zároveň brána jako adresa cílová) 16x8 bitů.

```
entity register_set is
port(
  we : in std_logic;
  wclk : in std_logic;
  ds1_addr : in std_logic_vector(3 downto 0);
  dst_din : in std_logic_vector(7 downto 0);
  s2_addr : in std_logic_vector(3 downto 0);
  s1_val : out std_logic_vector(7 downto 0);
  s2_val : out std_logic_vector(7 downto 0)
);
end entity register_set;
```

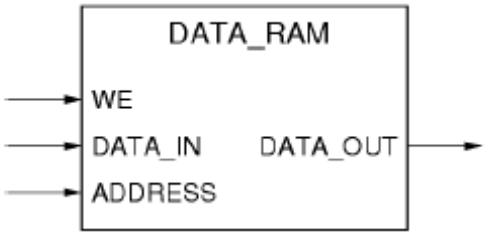


Obrázek 17 Registrové pole

Datová paměť

Blok synchronní paměti RAM 256x8 bitů

```
entity data_ram is
port(
clk : in std_logic;
we : in std_logic;
  address : in std_logic_vector(7 downto 0);
  data_in : in std_logic_vector(7 downto 0);
  data_out : out std_logic_vector(7 downto 0)
);
end entity data_ram;
```



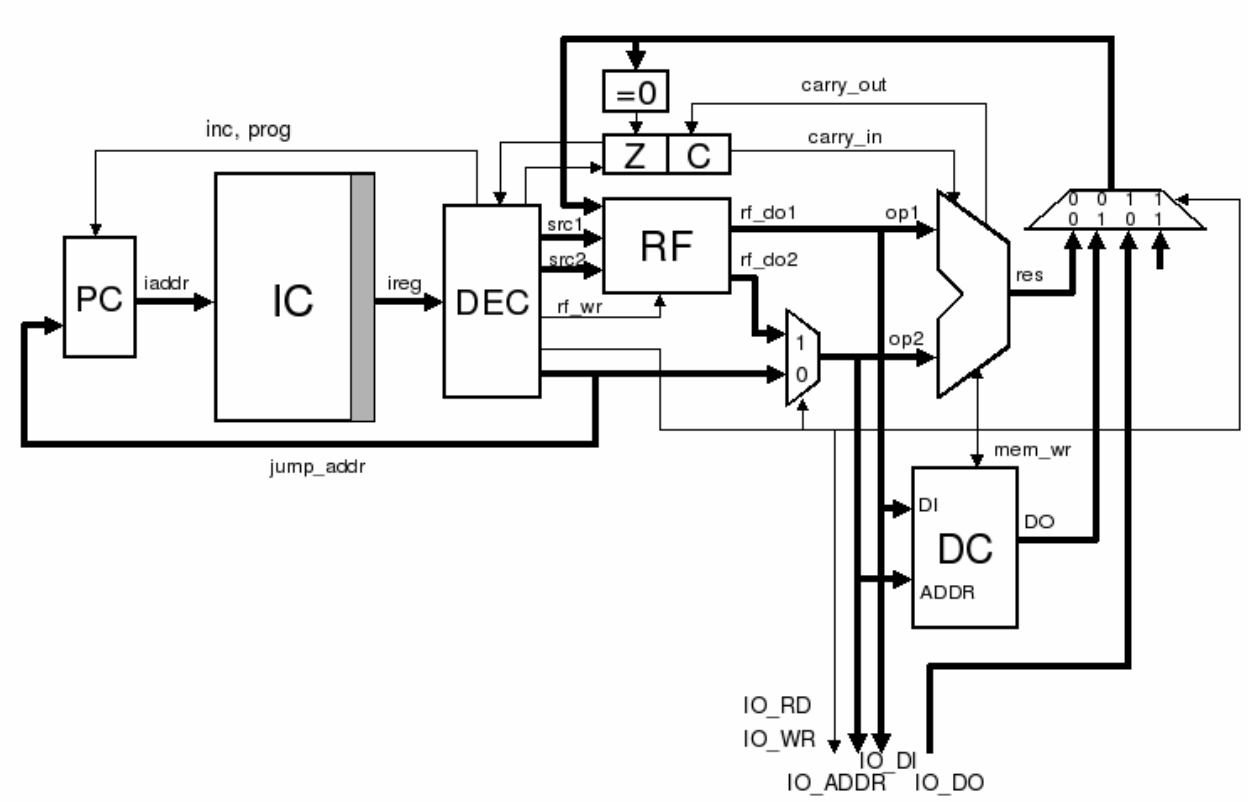
Obrázek 18 Datová paměť

Dekodér instrukcí

Realizuje dva základní úkoly.

1) Dekódování instrukčního slova – vytáhnout z něj adresy operandů popř. konstantu nebo adresu skoku + zjištění o jakou instrukci se jedná
2) Vygenerování řídících signálů ovládajících ostatní bloky procesoru.

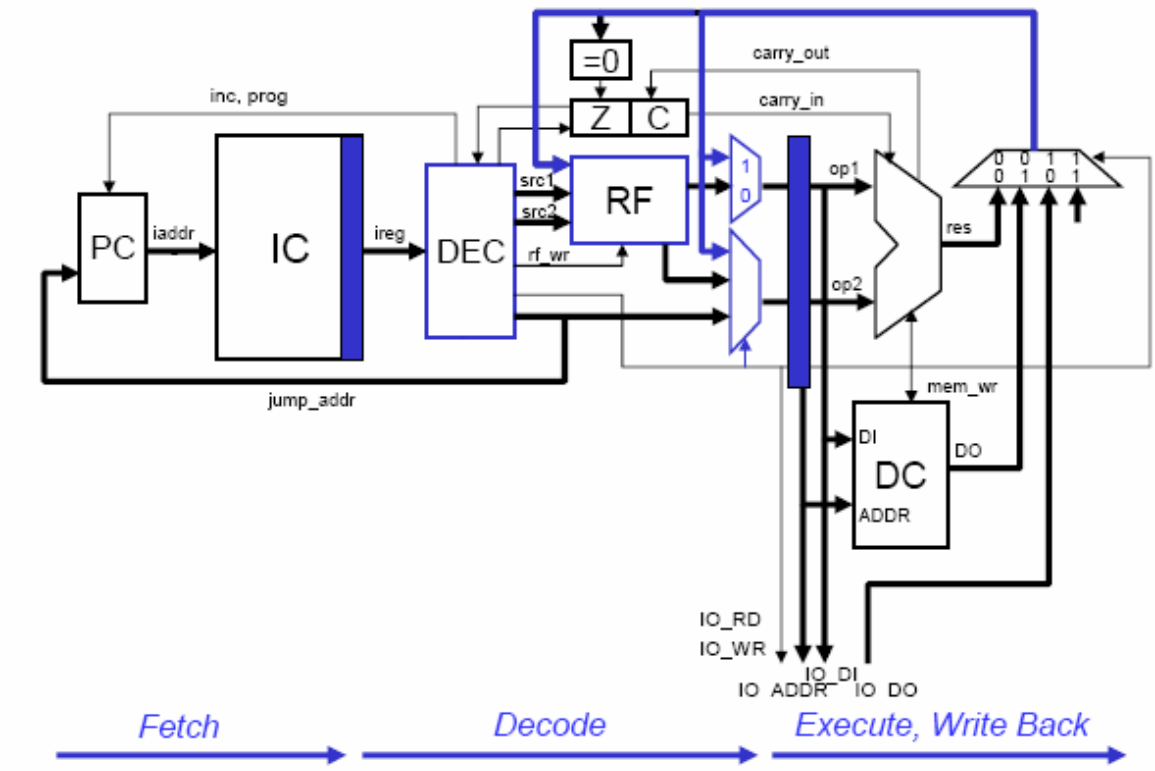
4.1.3 Blokové schéma jednoduchého procesoru



Obrázek 19 Blokové schéma jednoduchého procesoru

4.2 Procesor se zřetězeným zpracováním instrukcí

Procesor se sřetězeným zpracováním instrukcí vychází z procesoru sekvenčního, uvedeného výše. Tento jednoduchý procesor je doplněný o tří stupňovou pipeline. Architektura, instrukční sada i datová šířky zůstávají stejné.

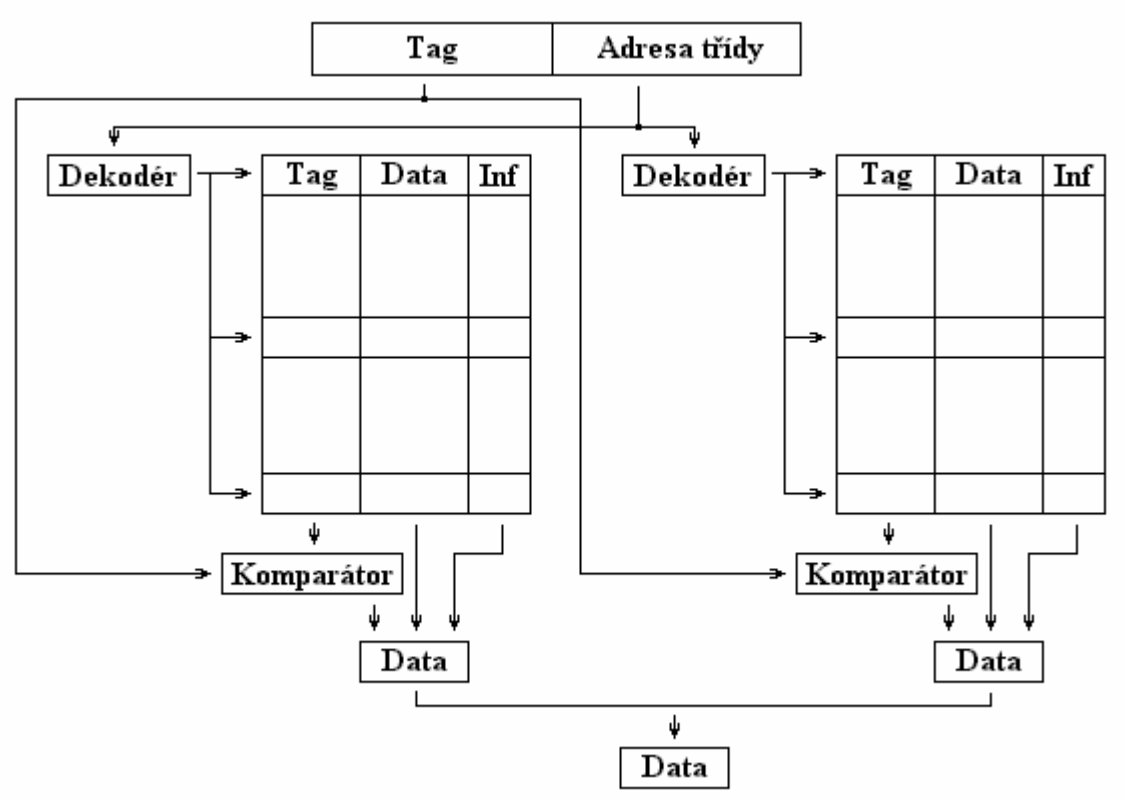


Obrázek 20 Jenoduchý procesor doplněný o pipeline

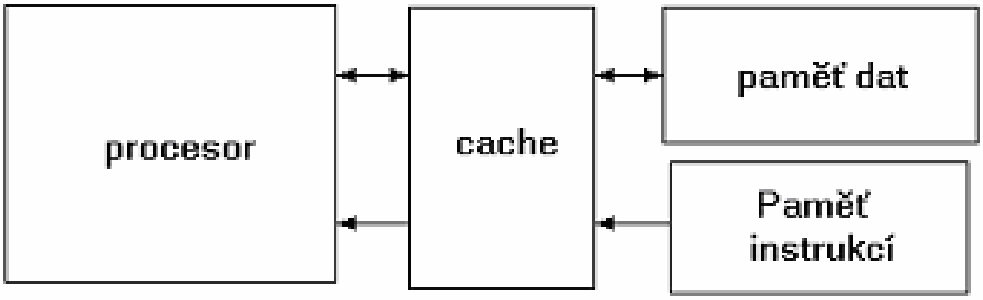
5 Návrh paměti cache

5.1 Návrh dvoucestné asociativní paměti cache

Pro návrh paměti cache byla vybrána architektura dvoucestně asociativní paměti cache. Jde o konkrétní prípad paměti cache, uvedené v kapitole 2.3.2



Obrázek 21 Dvoucestně asociativní paměť cache



Obrázek 22 Umístění paměti cache mezi procesor a datovou a instrukční paměť.

Závěr

Cílem této práce bylo seznámit se s architekturami soudobých procesorů a rychlých vyrovnávacích pamětí cache, prostudovat jazyk VHDL, promyslet a navrhnout jednoduchý procesor a následně se pokusit o jeho zřetězení. Tyto úkoly byly splněny.

Studium literatury týkající se architektur procesoru [1] bylo uplatněno při výběru vhodné architektury. Dále byla nastudována literatura popisující zřetězené zpracování instrukcí [1], [2]. Problematika týkající se pamětí cache byla nastudována z [3]. Základní informace o návrzích procesorů v jazyce VHDL byly získány z materiálu předmětu pokročilé číslicové systémy [5]. Na základě všech těchto nově nabytých informací byl vytvořen návrh jednoduchého a zřetězeného procesoru. Jednoduchý procesor byl nad rámec zadání implementován. Příklad činnosti tohoto procesoru je uveden v příloze.

Práce bude násadně (v letním semestru) pokračovat řešením konfliktů vznikajících při zřetězeném zpracování instrukcí. Poté budou všechny součásti systému implementovány v jazyce VHDL a budou případně i syntetizovány. Na závěr práce bude uvedeno srovnání sekvenčního procesoru s procesorem zřetězeným.

Literatura

[1] Dvořák V, Drábek V., *Architektura procesorů*. Brno, Vysoké učení technické v Brně 1999.

[2] Hennessy, J.L., Patterson, D. A. Copmuter Architecture- A Quantitative Approach. 4. vydání, Morgan Kaufman Publishers, Inc., 2007

[3] Studijní materiál Masarykovy university, *Cache paměti.* Dostupné z http://www.fi.muni.cz/usr/pelikan/ARCHIT/TEXTY/CACHE.HTML (leden 2007)

[4] Kořenek J., *HDL a VHDL jazyky*, Dostupné z (leden 2007) https://www.fit.vutbr.cz/study/courses/PCS/private/prednasky/01_uvod/01_uvod.pdf

[5] Martínek T., *Návrh mikroprocesoru,*Dostupné z (leden 2007) https://www.fit.vutbr.cz/study/courses/PCS/private/prednasky/06_navrh_procesoru/microprocessor.pdf

Příloha A

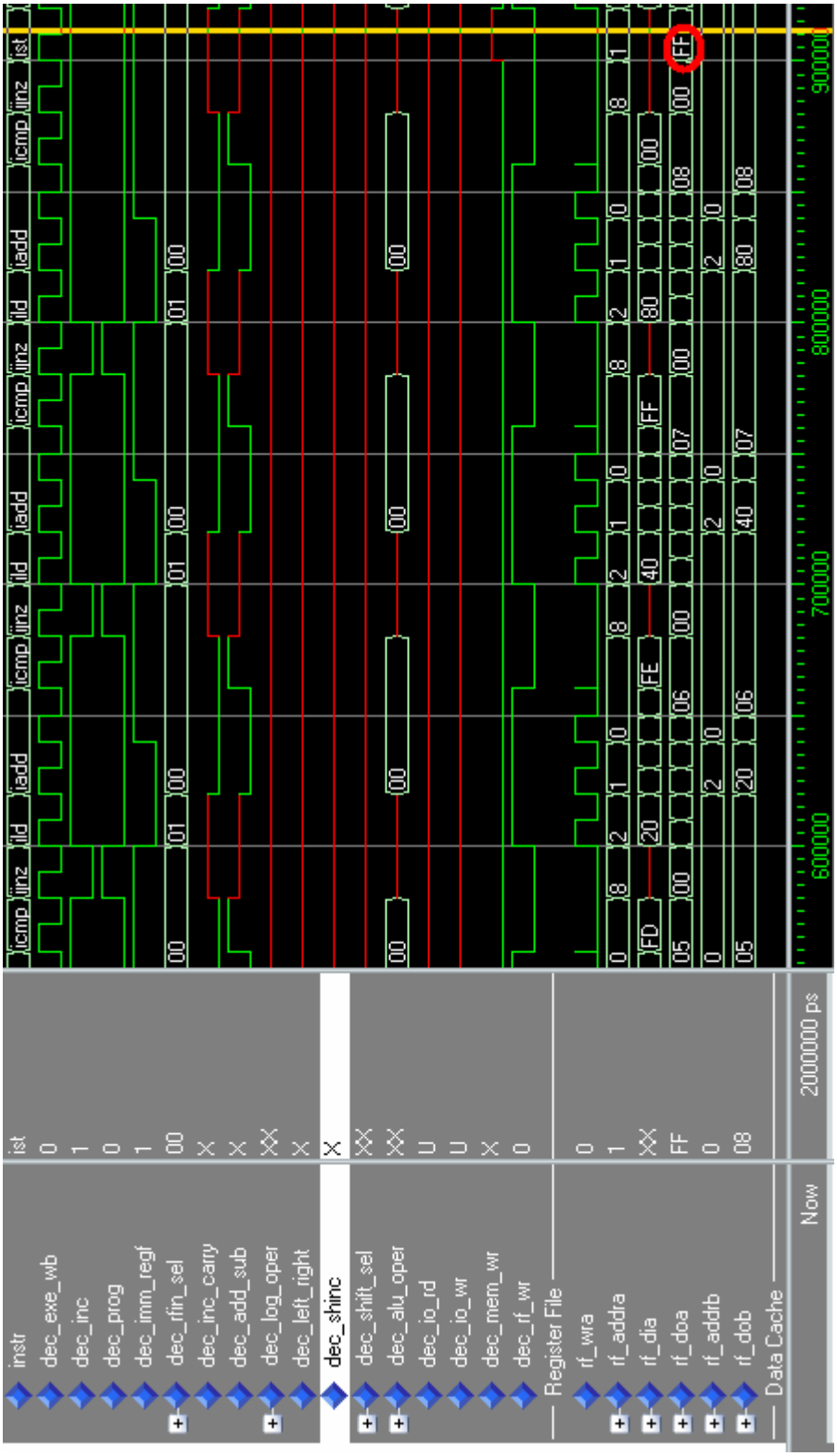
Ukázka práce navrženého a implementovaného jednoduchého procesoru. Procesor spočítal prvních osm hodnot nacházející se na začátku datové paměti a výsledek uložil na pozici adresy 9.

Obsah paměti

```
"00000001",
"00000010",
"00000100",
"00001000",
"00010000",
"00100000",
"01000000",
"10000000",
-------------
"11111111"  -- součet = FF hex.
```

Algoritmus realizující výpočet

```
mov R0, 0
mov R1, 0
cyklus: ld R2, R0
add R1, R2
add R0, 1
cmp R0, 8
jnz cyklus
st R1, 9
```



Obrázek 23 výstup simulace