ZÁPADOČESKÁ UNIVERZITA V PLZNI
FAKULTA ELEKTROTECHNICKÁ

Katedra aplikované elektroniky a telekomunikací

DIPLOMOVÁ PRÁCE

Návrh jádra RISC procesoru pro výukové účely

Bc. Lukáš Paločko **2012**

Original zadania alebo kopia

Anotácia

Diplomová práca prezentuje návrh jadra RISC procesoru pre výukové účely. Cieľom bolo navrhnúť a implementovať 8-bitový mikroprocesor RISC do jazyka VHDL. Realizácia kládla vysoký dôraz na možnosti následného zapracovania do výukových procesov. Navrhnutá inštrukčnú sada je založená na inštrukčnej sade THUMB a optimalizovaná pre 5-stupňovú zreťazenú linku. Práca taktiež predkladá analýzu 3 a 5-stupňovej zreťazenej linky v náväznosti na pokročilejšie architektúry zreťazených liniek. Dôležitá časť práce pozostávala z navrhnutia techniky pre vizualizáciu vnútorných signálov, ktorá je podporov pre hardwarový prípravok.

Kľučová slova

RISC, mikroprocesor, zreťazená linka, superzreťazená linka, superskalárna zreťazená linka, inštrukčná sada

Abstract

The master thesis presents the design of a RISC-processor core for teaching purposes. The aim of this project was to design and implement an 8-bit RISC microprocessor in the VHDL language. The implementation has been developed with regard to the teaching purposes. The developed instruction set is based on a THUMB Instruction Set and optimized for a 5-stage pipeline. The evolution of the pipeline from a 3-stage pipeline to the Superpipeline and Superscalar technique is discussed. An important part of the task - visualization of internal signals – has been tailored to the hardware support available on the Development and Education Board which has been selected for this purpose.

Key words

RISC, Micro-processor, Pipeline, Superpipeline, Superscalar, Instruction Set

Prehlásenie

Predkladám týmto k posúdeniu a obhajobe diplomovú prácu vypracovanú na záver štúdia na Fakulte elektrotechnickej Západočeskej univerzity v Plzni. Prehlasujem, že som túto diplomovú prácu vypracoval samostatne, s použitím odbornej literatúry a prameňov uvedených v zozname, ktorý je súčasťou tejto diplomovej práce. Ďalej prehlasujem, že všetok software, použitý pri riešení tejto diplomovej práce, je legálny.

V Plzni dňa 11.5.2012

Meno a Priezvisko

……..

Poďakovanie

Týmto by som chcel poďakovať vedúcemu diplomovej práce prof. Ing. Jiřimu Pinkerovi, CSc. za cenné profesionálne rady, prístup, pripomienky a metodické vedenie práce. Na záver, ale nie ako posledným, chcem poďakovať všetkým, ktorí mi akokoľvek pomohli, za ich podporu, trpezlivosť a dôveru.

Zoznam použitých skratiek a symbolov

ALU	*Arithmetic logic unit* – aritmeticko-logická jednotka
BCLA	*Block Carry-Lookahead adder* – sčítačka s blokovým prenosom
BP	*Branch Prediction* – jednotka vykonávajúca špekulatívne predvídanie skokov
CISC	Complex instruction set computer – architektúra inštrukčnej sady
CLA	*Carry- Lookahead adder* – sčítačka s predikciou prenosu
CPI	*Cycles per instruction* – počet taktov na inštrukciu
CPU	*Central processing unit* – centrálna procesorová jednotka
CRA	*Carry -Ripple adder* – sčítačka s propagáciou prenosu
CSA	*Carry -Select adder* – sčítačka s výberom prenosu
DA	*Data Access* – označuje blok zreťazenej linky v ktorom sa pristupuje k dátovej pamäti
DC	*Data Cache Access* – označuje blok zreťazenej linky v ktorom sa pristupuje k dátovej pamäti cache
DE2-115	Označenie vývojovej dosky od firmy Terasic
DEC	*Decode* – označuje dekódovací stupeň zreťazenej linky
DM	*Data Memory* – označenie pamäťového segmentu v architektúre procesoru
DMA	*Direct Memory Access* – priamy prístup do pamäti, bez účasti procesora
DPS	Doska plošných spojov
DT	*Data Translate* – označuje blok zreťazenej linky, ktorý ma za úlohu preložiť virtuálnu adresu na fyzickú
EX	*Execution Block* – označuje časť zreťazenej linky, ktorá je určená pre vykonávanie inštrukcií
EXr	Označenie pre register oddeľujúci dekódovací stupeň a stupeň pre výkon inštrukcie
FLAGr	Označenie pre register príznakových bitov
FPGA	*Field Programmable Gate Array*- číslicový integrovaný obvod obsahujúci programovateĺne bloky
FX	*Fixed point* – reprezentácia číselných hodnôt v pevnej radovej čiarke
HA	*Half Adder* – polovičná sčítačka
HiT	Indikačný signál potvrdzujúci platnosť dát v pamäti cache
IA	*Instruction address* – označenie pre časť (stupeň) zreťazenej linky, ktorá sa stará o výpočet nasledujúcej adresy
IC, ICACHE	Označenie pre inštrukčnú pamäť cache
IDEC	*Instruction Decode* - označenie pre dekódovací stupeň zreťazenej linky
IF	*Instruction address* – označenie pre časť (stupeň) zreťazenej linky, ktorá sa stará o výpočet nasledujúcej adresy
IM	*Instruction Memory* - pamäť inštrukcií
IR	*Instruction Register* – inštrukčný register
IT, ITBL	Transformácia virtuálnej adresy na fyzickú a kontrola správnosti tagu
k	oznacuje v rovnici (2) počet stupňov zreťazenej linky
LED	*Light-Emitting Diode* – dióda emitujúca svetlo
LX 4180	Implementácia konkrétnej architektúry mikroprocesora
MA	*Memory Access – označenie bloku (stupňa) pre prístup k pamäti*
MIPS	*Microprocessor without Interlocked Pipeline Stages- architektúra procesorov*
MOV	Inštrukcia presunu dát medzi registrami
MR	*Memory read – doba čítania z pamäte*

MP	*Memory penalty* - doba, ktorá súvisí s výpadkom dát alebo inštrukcií z pamäte cache
MS	*Memory stals* – doba pozastavenia pamäte
MUX	Multiplexor
NOP	*No Operation* – typ inštrukcie
OP	Operačný kód
OV, OVF	Príznakový bit pretečenia
PC	*Program Counter* – programový čítač
RAM	*Random access memory* - pamäť s ľubovoľným prístupom
RAR	*Read after Read* - závislosť typu čítanie po čítaní, nevyskytuje sa
RAW	*Read after Write* – závislosť typu čítanie po zápise
RCLA	*Ripple block Carry-Lookahead adder* – sčítačka s blokovým prenosom
Rd	Cieľový register
RF	*Read register File* - označenie pre stupeň zreťazenej linky, ktorá prevádza načítanie operandov z registrového poľa
RISC	*Reduced Instruction Set Computer* - označenie typu architektúry inštrukčnej sady
SP	*Stack Pointer* – ukazovateľ na zásobník
SRAM	Statická pamäť RAM
TC	*Tag Check* - označenie pre stupeň zreťazenej linky, ktorá uskutočňuje kontrolu tagu
THUMB	Instrukčná sada procesorov ARM
VHDL	Programovací jazyk slúžiaci pre popis hardware
VLIW	*Very Long Instruction Word* – architektúra procesov
WAW	*Write after Write* – závislosť typu zápis po zápise
WAR	*Write after Read* - závislosť typu zápis po čítaní
WB	*Writte Back* - označenie pre stupeň zreťazenej linky, ktorá uskutočňuje spätný zápis do registrového poľa
WBr	označenie pre register oddeľujúci stupen MA a WB
Z	*Zero* – príznakový bit indikujúci nulový výsledok operácie
ZL	Zreťazená linka

Zoznam **obrázkov**

Zoznam tabuliek

Obsah

Úvod

Od doby, kedy Marcian "Ted" Hof a Stan Mazor myšlienkou integrovania dvanástich obvodov na jeden čip odštartovali éru mikroprocesorov, prešlo mnoho času a koncepčné návrhy štruktúr sa výrazne zmenili. V dnešnej dobe (rok 2012) sú cenovou politikou výrazne atakované 8-bitové architektúry užívane v mikrokontroléroch, a to modernejšími 32-bitovými RISC architektúrami. Pomer cena výkon u niektorých výrobcov je výrazne v prospech 32-bitových architektúr. Tento aspekt sa bude pravdepodobne aj naďalej prehlbovať. Inštrukčný paralelizmus v týchto sofistikovanejších systémoch je na inej úrovni ako spomínane jadrá založené na 8-bitových architektúrach.

Predkladaná práca vznikla na základe potreby začleniť architektúry do výukového procesu. Ukázať, ako zreťazená linka pristupuje k inštrukciám, aké konflikty vznikajú pri zreťazenom spracovaní a ako je možné sa s nimi vysporiadať.

V prvej kapitole je v krátkosti predložená problematika aritmetiky sčítačiek a jej náväznosti na zreťazené spracovanie. Je vytvorený prehľad sčítačiek a diskutovaný problém latencie na dvoch konkrétnych typoch. Cieľom je poukázať na štrukturálne problémy sčítačiek, nutnosť vyvodzovať dôsledky z ich latencie, spotreby a potrebnej plochy na čipe. V tejto kapitole je tiež venovaný priestor pre návrh sčítačky implementovanej do navrhovaného jadra procesoru.

Druhá kapitola je zameraná na problematiku zreťazeného spracovania. Pozostáva z analýzy skalárnych zreťazených liniek a vyvodzuje z nej výhody, nevýhody, aplikačné zameranie a náväznosť na pokročilé architektúry. Veľmi podstatná vlastnosť zreťazenej linky je náchylnosť k hazardom a závislostiam. Z tohto dôvodu táto kapitola zahŕňa aj rozpravu o konfliktoch vznikajúcich pri zreťazenom spracovaní inštrukcií. Všetky uvedené pasáže tejto kapitoly sú výsledkom veľkého množstva informácií pochádzajúcich z kvalitných zahraničných knižných publikácií, vedeckých a odborných článkov, aplikačných poznámok a katalógových listov. Ďalej je treba poznamenať, že vyššie zmieňované teoretické pasáže nenesú povahu len samoúčelnej kapitoly obsahujúcej základný obecný popis, ale slúžia ako základ pre následnú praktickú realizáciu zreťazenej linky a v niektorých častiach poukazujú na špecifické detaily.

Tretia kapitola je venovaná návrhu samotného inštrukčného setu, ktorý pre účel diplomovej práce vychádza z inštrukčného setu THUMB (ARM).

Štvrtá kapitola si kladie za cieľ implementáciu navrhnutého inštrukčného setu do zreťazeného spracovania. Na základe vlastností, vzhľadom k ucelenému rozdeleniu vykonávania inštrukcie do piatich fáz (IF, DEC, EX, MA, WB), bola vybraná k implementácii 5-stupňová zreťazená linka. Táto kapitola je rozdelená do dvoch podkapitol. Zatiaľ čo záležitosťou prvého celku je návrh základnej štruktúry zreťazeného spracovania a mikroarchitektúr jednotlivých stupňov, druhá časť sa venuje metódam pre indikáciu a eliminácii konfliktov a závislosti.

Implementácii navrhnutej mikroarchitektúry do jazyka VHDL sa venuje piata kapitola. Tá zároveň popisuje návrh vizualizačných častí, obsluhy a celkovej koncepcie hardwarového prípravku, ktorý je realizovaný na vývojovej doske DE2-115 od firmy Terasic.

1 Aritmetika, jej princípy a funkčné bloky

Aritmeticko-logická jednotka (ALU) je väčší (nie vždy čisto kombinačný) funkčný blok, ktorý vykonáva aritmeticko-logické operácie s vstupnými operandami. To, aký rozsah operácií jednotka vykonáva, sa líši vzhľadom k čomu je daný procesor určený. Pre hlbšiu analýzu je vhodné rozdeliť ALU na menšie funkčné prvky. Prvým krokom je rozdelenie operácii na aritmetické a logické.

Najčastejšie implementované operácie v ALU jednotke sú sčítanie a odčítanie v pevnej radovej čiarke. Zložitejšie aritmeticko-logické jednotky obsahujú aj násobičky a deličky. Dnes je bežnou súčasťou vyspelejších architektúr používanie ALU jednotky pracujúcej v pohyblivej radovej čiarke. Obe ALU jednotky pracujú paralelne s vlastnými zbernicami a vyčleneným registrovým poľom.

1.1 Sčítačky

Každú aritmetickú operáciu je možné vykonať za pomoci dvoch funkčných blokov, ktorými sú sčítačka a register. Preto budú v ďalšom texte predostreté niektoré typy sčítačiek, ich princíp, výhody a nevýhody. Ako pri každom inom funkčnom bloku, tak aj pri sčítačkách aplikujeme dve protichodné požiadavky. Tými sú rýchlosť a potrebná oblasť na čipe, ktorá sa prejavuje hlavne pri zvyšovaní počtu bitov sčítancov. I keď niektoré typy majú lepšie pomery rýchlosť/potrebná oblasť na čipe, vždy je to do istej miery kompromis medzi týmito požiadavkami a záleží na aplikácii, v ktorej to plánujeme použiť.

Prehľad niektorých typov sčítačiek:

- Sčítačka s propagáciou prenosu, CRA (Carry-Ripple Adder)
- Sčítačka s predikciou prenosu, CLA (Carry- Lookahead Adder)
- Sčítačky s blokovým prenosom
  - Block carry look-ahead adder (BCLA)
  - Ripple block carry look-ahead adder (RCLA)
- Sčítačka s výberom prenosu, CSA (Carry- Select Adder)
- Lingové sčítačky, (Ling adder)
- Sčítačka s podmieneným prenosom, (Conditional sum adder)
- Sčítačka s tranzitívnym prenosom, (Carry-Skip Adder)
- Asynchrónna sčítačka
- Sčítačka s uchovaným prenosom, (Carry- Save Adder)
- Paralelné prefixové sčítačky, (Parallel prefix adders)
  - Ladner-Fisher adder
  - Brent-Kung adder

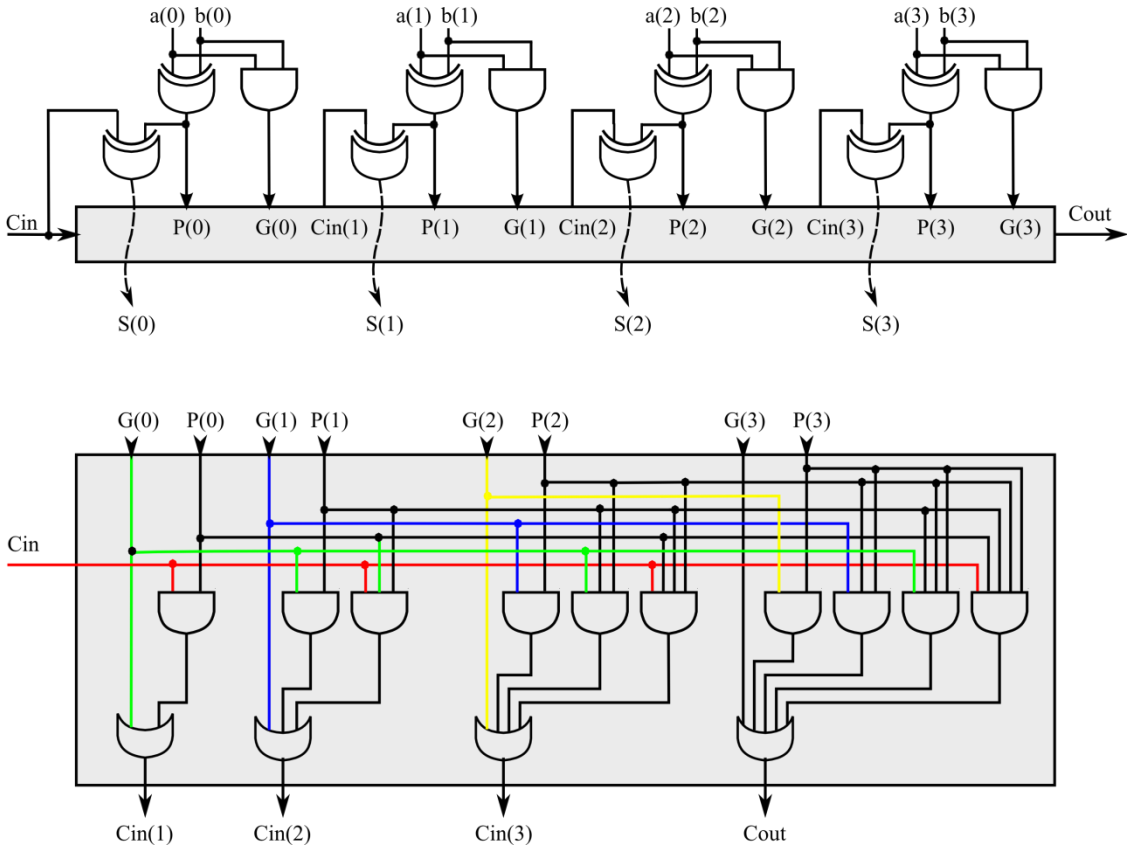
- Kogge-Stone adder
- Han-Carlson adder

V nasledujúcom texte budú v krátkosti popísane dve princípy sčítačiek, ktoré sú implementované do návrhu aritmeticko-logickej jednotky. A to konkrétne:

- Sčítačka s predikciou prenosu
- Sčítačka s výberom prenosu

1.1.1 Sčítačka s predikciou prenosu (Carry- Lookahead Adder)

Sčítačka tohto typu znižuje latenciu vytvorenou propagáciou prenosu skrátením kritickej cesty, ktorá je nedostatkom sčítačky CRA. Základnou myšlienkou aplikovaného princípu je zistiť, či v nižších stupňoch nastane prenos alebo nie. Táto myšlienka je veľmi elegantná, ale prináša celú radu problémov. Jedným z najvýraznejších je nárast vstupov hradla AND rozširovaním vstupných operandov. Túto situáciu prezentuje aj obr. č. 1.1. Pri 4-bitových operandoch je potrebné AND hradlo s piatimi vstupmi. S nárastom vstupov hradla rastie kapacita a tým sa zvyšuje jeho latencia.



Obr. č. 1.1 *4-bitová sčítačka s predikciou prenosu*

+

Ďalším problémom je narastajúci počet potrebných AND hradiel s každým pridaným stupňom. S tým je spojený aj nárast vstupov hradla OR. V štvrtom stupni sú potrebné až štyri hradlá typu AND. Práve z týchto dôvodov sa pre operandy s vyšším počtom bitov požívajú hierarchické zapojenia a viacúrovňové zapojenia CLA.

Kritická cesta sčítačky typu CLA pozostáva z dvoch častí:

- Kritická cesta pre výpočet súčtu:

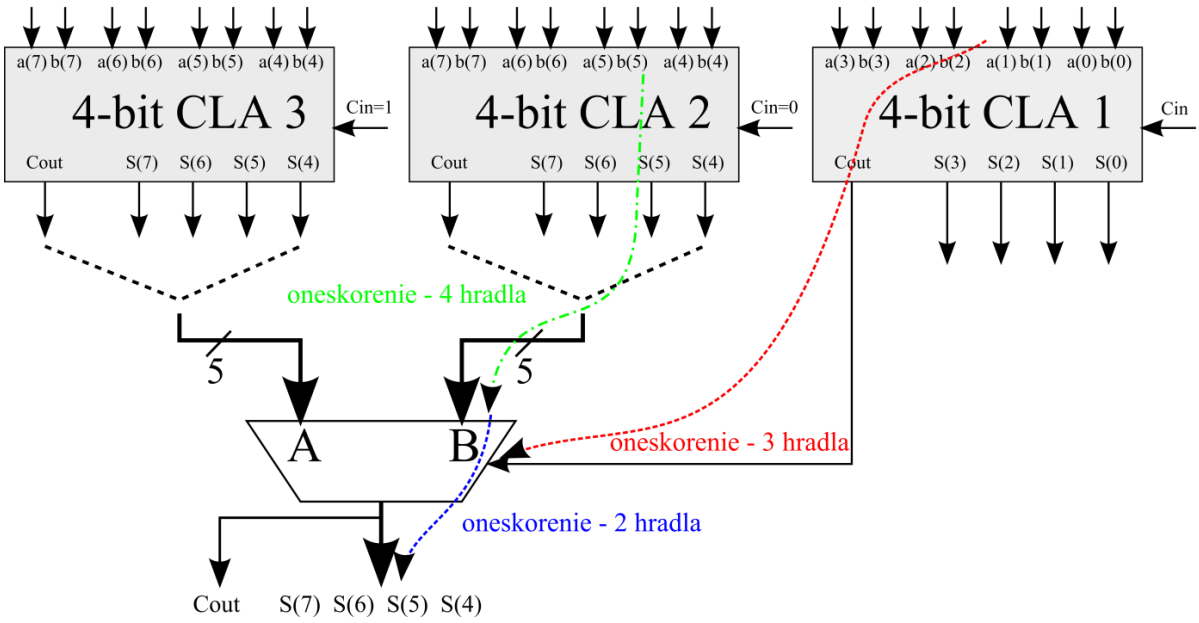
$$tp_{suma} = tp_{XOR} + tp_{AND} + tp_{OR} + tp_{XOR} \quad (1)$$

- Kritická cesta pre prenos:

$$tp_{carry} = tp_{XOR} + tp_{AND} + tp_{OR} \quad (2)$$

1.1.2 Sčítačka s výberom prenosu (Carry- Select Adder)

Myšlienkou tejto techniky je výber prenosu a tým skrátenie kritickej cesty. Na obr. č. 1.2 je znázornená jedna z variant sčítačky CSA. Osembitové operandy sú rozdelené na polovicu a privedené na tri rovnaké 4-bitové sčítačky. Zatiaľ čo sú prvé štyri bity oboch operandov sčítavané, sčíta sa aj druhá polovica operandu a to paralelne pre oba možné prípa-



Obr. č. 1.2 *8-bitová sčítačka s výberom prenosu a znázornením kritickej cesty*

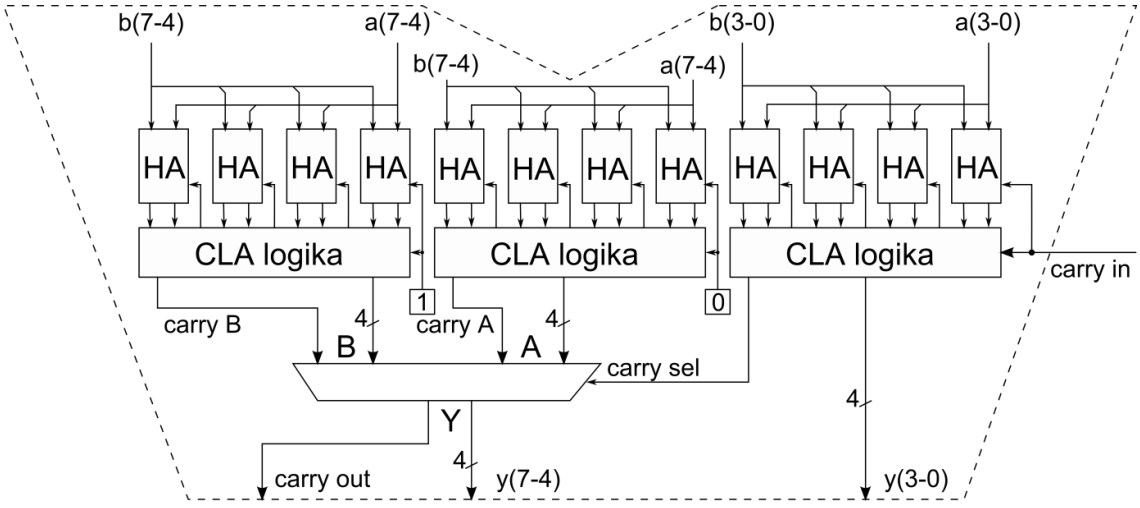
dy prenosu z prvého bloku. Následne sa už iba vyberie ta správna možnosť. Kritická cesta sa skráti takmer o polovicu, pretože je potrebné počítať s oneskorením, ktoré tvorí multiplexor. Skrátenie kritickej cesty je ale na úkor navýšenia hradiel o niečo vyše tretinu.

Kritická cesta znázornená na obr. č. 1.2 prechádza paralelne totožnými sčítačkami *CLA 2* a *CLA 3* a multiplexorom. Celkové oneskorenie je tvorené 6 hradlami, kde 2 hradlá pripadajú na multiplexor a 4 hradlá pre výpočet sumy. Oneskorenie pre riadenie multiplexora pozostáva z 3 hradiel.

1.2 Návrh sčítačky pre aritmeticko-logickú jednotku

Aby pri vyučovaní a prezentovaní vzorových príkladov bola patrná aj problematika sčítačiek, rozhodol som sa nevyužiť behaviorálny popis sčítačky v jazyku VHDL, ale navrhnúť sčítačku z menších elementov. Návrh vychádza z princípov CLA a CSA. Obr. č 1.3 prezentuje štruktúru sčítačky zostavenú z troch typov prvkov. Prvým je polovičná sčítačka (HA), druhým je logika CLA urýchľujúca prenos a posledným prvkom je 5-bitový multiplexor. Ten vyberá prenos na základe výpočtu prvých 4 bitov.

Aby mohla byť sčítačka vložená do zreťazenej linky, je nutné ju ešte doplniť o logiku indikujúcu pretečenie (OV, OVF), príznak znamienka (N) a príznak nulového výsledku (Z). Začleneniu sčítačky do ALU a tým aj do zreťazeného spracovania sa venuje kapitola popisujúca *blok vykonávania inštrukcie*. Aj keď tento typ sčítačky sa z hľadiska štruktúry príliš nehodí na implementovanie do FPGA, cieľom bolo poukázať na problematiku sčítačiek.



Obr. č. 1.3 *Návrh 8- bitovej sčítačky typu CSA pomocou použitia troch 4- bitových sčítačiek CLA.*

2 Zreťazené spracovanie inštrukcií

Zapracovanie zreťazeného spracovania inštrukcií do koncepcie návrhu mikroarchitektúr procesorov sa stalo jedným z pilierov architektúry RISC. Bolo to podriadené myšlienke, aby behom jedného strojového cyklu došlo k vykonaniu jednej inštrukcie. Okrem zreťazeného spracovania inštrukcií, o ktorom pojednáva táto časť, existuje aj zreťazené spracovanie aritmetických operácií a zreťazené spracovanie procesov. Prv než budú predostreté hlbšie vzťahy medzi rôznymi modifikáciami zreťazeného spracovania, je vhodné zadefinovať parametre, ktoré sú ovplyvňované návrhom zreťazenej linky.

Hĺbkou sa označuje počet jednotlivých funkčných blokov idúcich za sebou, ktoré sú oddelené registrovým poľom. *Šírkou* zreťazenej linky rozumieme počet paralelne radených výkonných jednotiek umiestnených v stupni pre vykonávanie inštrukcie. Môže sa jednať o jednotky ako aritmeticko-logická jednotka s pevnou radovou čiarkou, aritmeticko-logická jednotka s pohyblivou radovou čiarkou. Samozrejme počet týchto jednotiek nemusí byť striktne jedna. Takto popísaná architektúra s viacerými jednotkami pre vykonávanie sa nazýva *superskalárna architektúra.*

Uvažujme funkčný blok tvorený kombinačnou logikou s istou funkciou, ktorá pre túto úvahu je nepodstatná. Každé hradlo kombinačnej logiky ma isté oneskorenie a následne celý blok ma dané časové oneskorenie tvorené štruktúrou a typom hradiel. To znamená, že na tento blok musíme privádzať vstupné signály s takou frekvenciou, aby bolo rešpektované požadované oneskorenie celého bloku. Ak označíme oneskorenie celého bloku *D,* následne je možné vyjadriť priepustnosť tohto systému:

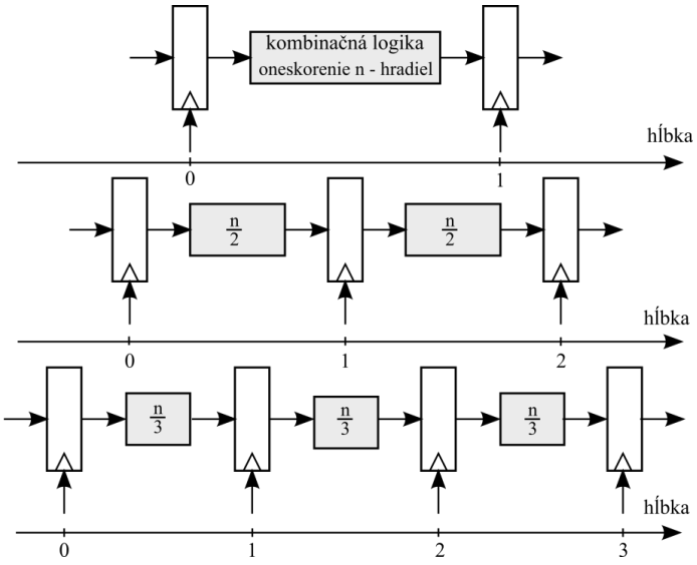
$$P = \frac{1}{D} \qquad (1)$$

Ten istý funkčný blok je ale možné rozdeliť na viacero častí a tým sa dosiahne zvýšenie priepustnosti systému. Ak následne označíme obecne aj počet častí, na ktoré je možne obvod rozdeliť, písmenom *k*, potom môžeme rovnicu č. 1 upraviť nasledovne:

$$P = \frac{1}{\frac{D}{k}} \qquad (2)$$

Na obr. č. 2.1 je znázornený tento postup delenia pre dva prípady, *k=2* a *k=3*. Aby bolo

zrýchlenie systému využité v maximálnej miere využitím tejto techniky, je nutné deliť systém na časti s približne rovnakým časovým oneskorením. Takto popísané zrýchlenie je ale ideálne a v reálnom systéme je nutné počítať s oneskorením vkladaných registrových polí, ktoré sú použité pre oddelenie jednotlivých častí systému.



Obr. č. 2.1 *Rozdelenie jedného funkčného bloku na viacero elementov*

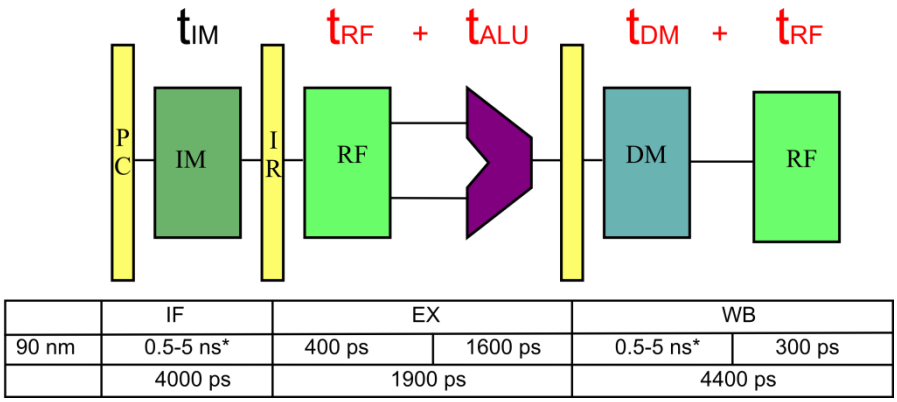
2.1 Limity zreťazeného spracovania

Ako každá technika, aj zreťazené spracovanie inštrukcií má svoje limity, s ktorými je nutné počítať pri návrhu samotnej architektúry a mikroarchitektúry zreťazenej linky. Čím viac sa snažíme zvýšiť výkon a približujeme sa tak k limitom, tým viac musíme počítať návrhom, ktorý spočíva nielen v precíznom návrhu miktoarchitektúry a časovania, ale aj s optimalizáciou na úrovni tranzistorov na samotnom kremíku. Z hľadiska toho, čo limity spôsobuje, môžeme tieto rozdeliť:

1. Limity spôsobené implementáciou
    i. Oneskorenie logických hradiel (Logic Delay)
    ii. Chvenie hodinového taktovania (Clock Skew)
    iii. Stabilnosť dát počas zápisovej hrany (Latch Overhead)
2. Limity spôsobené štruktúrou zreťazenej linky
3. Limity spôsobné programami

2.2 Analýza skalárnych zreťazených liniek s nízkou hĺbkou

Pri návrhu jadra procesoru je nutné sa zaoberať hĺbkou a povahou zreťazenej linky, ktorá je jeho hlavným pilierom. Na obr. č. 2.2 **Chyba! Nenalezen zdroj odkazů.**je znázornená zreťazená linka o hĺbke troch stupňov s latenciami funkčných blokov. Výhodou je jednoduchosť z pohľadu návrhu (napr. jednoduché časovanie, väzby medzi jednotlivými stupňami). Takto navrhnuté zreťazené linky s dobre navrhnutou mikroarchitektúrou (je kladený prísny ohľad na spotrebu), s predikciou skokov a sofistikovanou sadou prerušení, sú základom pre jadrá moderných mikrokontrolérov. Ako príklad je možné uviesť jadro Cortex-M3/M4 firmy ARM.



	IF	EX		WB	
90 nm	0.5-5 ns*	400 ps	1600 ps	0.5-5 ns*	300 ps
	4000 ps	1900 ps		4400 ps	

Obr. č. 2.2 *3-stupňová zreťazená linka s uvedenými latenciami jednotlivých blokov.*

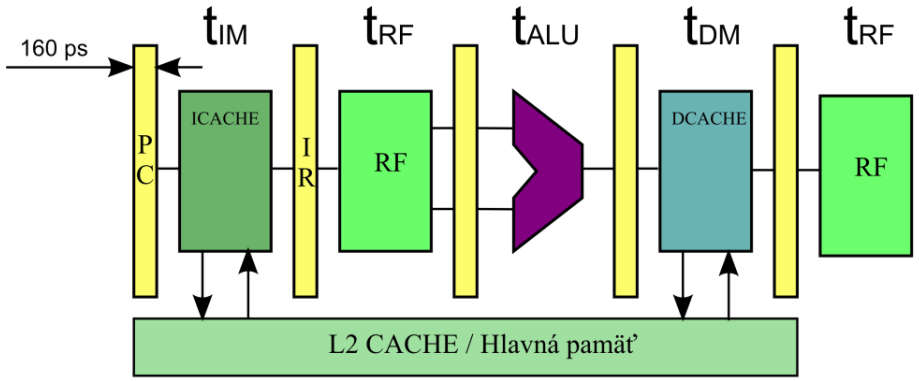
Pravdepodobne najznámejšia koncepcia zreťazenej linky tvoriaca samotné jadro procesora má *5 stupňov*. Fázy spracovania sa delia do jednotlivých etáp, čo podliehalo istému vývoju a technickým možnostiam[1] doby, v ktorej boli tieto koncepcie navrhované. Filozofia 5-stupňovej zreťazenej linky spočíva v rozdelení výkonu inštrukcie na 5 ucelených úsekov. Avšak oneskorenia signálov pri prechode blokmi, ktoré pracovali s pamäťou, boli výrazne vyššie ako u blokov, ktoré pracovali s registrovým poľom. Táto nevýhoda sa dá do istej miery zmierniť vložením pamätí cache. Jednotlivé fázy 5-stupňovej zreťazenej linky sú:

- Načítanie inštrukcie -IF
  - Sprístupnenie ICACHE a kontrola tagu.
- Dekódovanie inštrukcie -DEC
- Vykonávanie inštrukcie -EX

[1] Takt procesora, prístupová doba k pamätiam a jej štruktúra, veľkosť potrebnej operačnej pamäte.

- Prístup k pamäti -MA
  - Sprístupnenie DCACHE a kontrola tagu.
- Spätný zápis do registrového poľa -WB

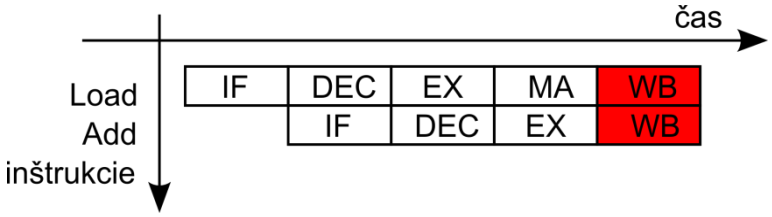
V porovnaní s trojstupňovou zreťazenou linkou, ktorá je na obr. č. 2.2, má táto koncepcia menšie latencie jednotlivých stupňov, čo bolo dosiahnuté rozdelením stupňa č. 2 a č. 3 a zaradením oddeľovacích registrov. Takto dosiahnuté zrýchlenie nie je lineárne, pretože je nutné počítať s oneskorením a dodržaním stabilnosti obsahu dát pri aktívnych hranách vložených registrov.



	IF	DEC (RF)	EX	MA	WB
90 nm	0.5-5 ns*	400 ps	1600 ps	0.5-5 ns*	300 ps
	4000 ps	650-900 ps	1600 ps	4400 ps	300 ps

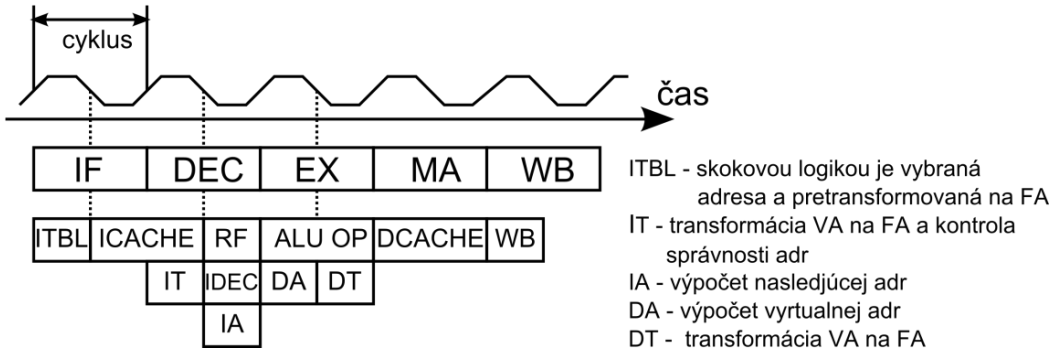
Obr. č. 2.3 *5-stupňová zreťazená linka s uvedenými latenciami jednotlivých blokov*

Pri prúdovom spracovaní v 5-stupňovej zreťazenej linky už dochádza k tomu, že rôzne inštrukcie nepotrebujú rovnakú hĺbku tejto linky. Túto skutočnosť demonštruje obr.č. 2.4 na ktorom sú znázornené operácie načítanie z pamäti a sčítanie. Prvá inštrukcia *„i, Load“* zapisuje načítané dáta z pamäte do registrového poľa v čase (cykle), v ktorom inštrukcia *„i+1, add“* taktiež môže zapisovať do registrového poľa. Dostávame sa tak ku štrukturálnemu hazardu, kde je tento problém možné vyriešiť ďalšou zápisovou bránou. Toto riešenie má ale za následok zvýšenie plochy čipu, následné zvýšenie T_{clk} a v neposlednom rade požiadavky na sofistikovanejšiu dekódovaciu a detekčnú logiku. To je dôvod, prečo väčšina RISC procesorov, využívajúcich 5-stupňovú zreťazenú linku, má inštrukčný set a mikroarchitektúru navrhnutú spôsobom, že aritmeticko-logické operácie prechádzajú akoby prázdnym cyklom MA.



Obr. č. 2.4 *Porovnanie operácií načítanie a sčítanie*

I keď niektoré architektúry využívajú zdanlivo 5-stupňové zreťazené linky, ich mikroarchitektúra však pri pozornej a hlbšej analýze vykazuje väčšiu hĺbku, ako je prezentovaná navonok. Takéto linky reagujú v niektorých fázach spracovania inštrukcie (hlavne v paralelne radených častiach jednotlivých blokov) aj na zostupné hrany. Dôvodom je práca s pamäťami cache. Konkrétne samotné sprístupnenie dátoveho a riadiaceho obsahu z cache a následná verifikácia jeho správnosti. Takýto prístup volili napríklad procesory MIPS začiatkom deväťdesiatych rokov minulého storočia. Napr. jadro LX4180 (rok 2001), vychádzajúce z rodiny MIPS32 4K, má upravenú ZL[2] tak, že aritmeticko-logické operácie sa spúšťajú zostupnou hranou v druhom stupni. Podrobné informácie o tejto architektúre sa nachádzajú v [12].



Obr. č. 2.5 *Detailnejšie zobrazenie štruktúry 5- stupňovej ZL reagujúcej aj na zostupné hrany. Jedná z variant ZL MIPS R3000*

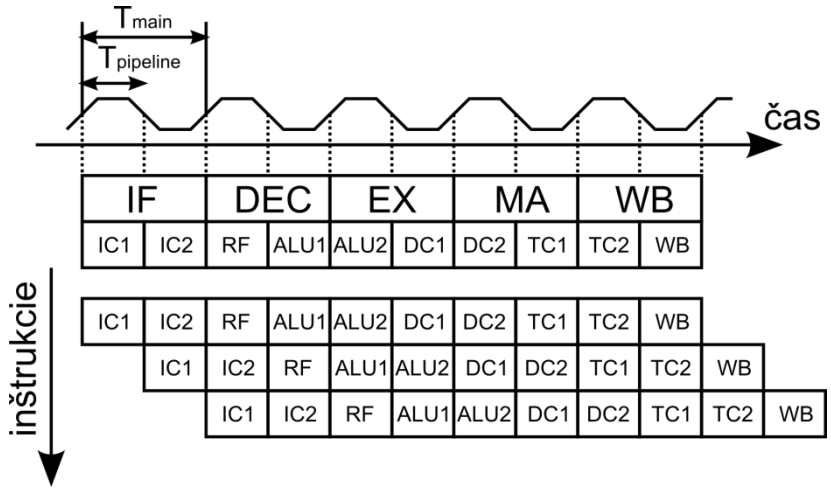
Výhodou 5-stupňovej zreťazenej linky je jej jednoduchosť, elegancia, absencie dátových závislostí typu WAW a WAR. Ďalej takzvaný oneskorený skok, ktorý bol

[2] Zreťazenú linku tvorí 5 stupňov v atypickom poradí: 1) Práca s inštrukčnou pamäťou cache, 2) Vykonávanie inštrukcie, 3) Práca s dátovou pamäťou cache, 4) 5) Zápis do registrového poľa

navrhnutý práve pre túto štruktúru zreťazenej linky. Superskalárne a superzreťazené linky vychádzali a boli odvodené práve z tejto koncepcie.

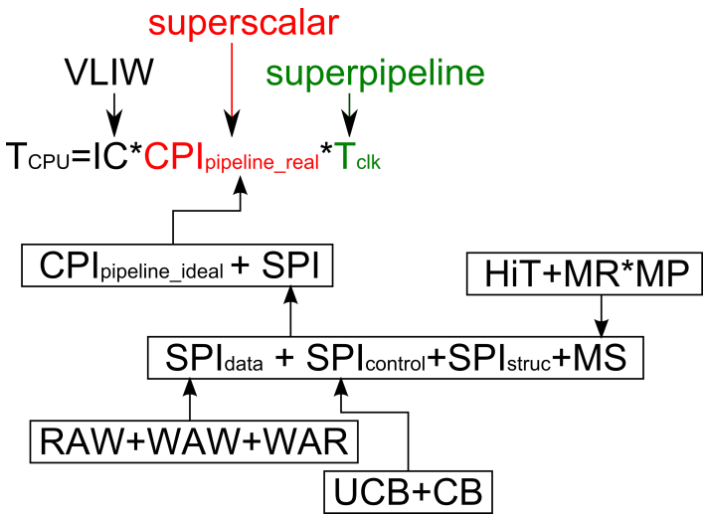
2.3 Modifikácie linky pre zreťazené spracovanie. Superskalárna a superzreťazená linka

5-stupňová zreťazená linka bola konštruovaná s ohľadom na priblíženie sa hodnote CPI = 1. Je to však nedosiahnuteľná hodnota vzhľadom k hazardom, ktoré sa v takomto systéme vyskytujú. Smery, ktorými sa môžeme vydať pri zvyšovaní výkonnosti systému zreťazenej linky, je viacero. Rovnica zasadená do obr. č. 2.6 vyjadruje, na čom závisí doba vykonávania programu a kde všade je možné hľadať cesty k zvyšovaniu výkonnosti zreťazenej linky a tým mikroprocesora. Jednou cestou je napríklad zvyšovanie priepustnosti zreťazenej linky zvyšujúcou sa frekvenciou. Pri zvyšovaní taktovacej frekvencie narazíme na problém oneskorenia signálu prechádzajúcim blokom a hradlami, ktoré tvoria jednotlivý stupeň. Koncepcia 5-stupňovej zreťazenej linky sa tak stane nedostačujúca. Preto je nutné zvýšiť hĺbku zreťazenej linky a rozdeliť tak spracovanie inštrukcie do viacerých blokov. To je cesta k superzreťazenej linke. Zvýšenie taktovacej frekvencie nie je jedinou motiváciou pre rozčlenenie spracúvanej inštrukcie do viacerých blokov. Pridávaním ďalších výkonných jednotiek, ako násobičky, deličky a sčítačky v pohyblivej radovej čiarke, sa dostávame k problému nerovnomernej latencie spomínaných jednotiek. Ak by sme nerozčlenili aj stupeň pre vykonanie inštrukcie, boli by sme nútení čakať na najpomalšiu jednotku bloku, ale zároveň aj na najpomalšiu jednotku celej koncepcie zreťazenej linky.



Obr. č. 2.6 *Možná implikácia supezreťazenej linky na 5-stupnovú ZL*

Prechod k superzreťazenej linke je možný taktiež využitím oboch aktívnych hrán a zároveň úpravou kritických blokov, ako sú pamäťové časti a výkonná jednotka ALU. Takýto prístup naznačuje **Chyba! Nenalezen zdroj odkazů.** Redukovaním niektorých blokov, použitím lepšej technológie či sofistikovanejším návrhom bloku (napr. návrhom rýchlejších sčítačiek) spolu s zaradením niektorých blokov paralelne (hlavne kontrola Tagu cache pamätí), je možné z 10-stupňovej hĺbky ZL dosiahnuť výkonnú ZL o hĺbke ôsmich stupňov. Podobným spôsobom sa rodili niektoré varianty zreťazenej (superzraťazenej) linky procesora MIPS R4000.



Obr. č. 2.7 *Rovnica vyjadrujúca dobu výpočtu programu CPU v závislosti na nedokonalosti zreťazeného spracovania*

V predchádzajúcom odstavci bolo načrtnuté pridávanie viacerých funkčných jednotiek. Pri klasickej skalárnej zreťazenej linke vždy môže pracovať iba jedna jednotka zaradenia v stupni EX a zvyšné ostanú nevyužité. Preto sa prechádzalo postupne na superskalárnu architektúru, ktorá využíva viacero funkčných jednotiek radených paralelne. Táto metóda je cestou k znižovaniu $CPI_{pipeline\_real}$. Na to je ale potrebné dekódovať viacero inštrukcií v jednom takte a poslať do správnej rady, kde sa otestujú závislosti medzi inštrukciami, ktoré sú vykonávane za sebou, ale aj paralelne. Následne záleží na povahe zreťazenej superskalárnej linky, ktorá sa všeobecne delí na:

- Statická superskalárna ZL (in-order superscalar)
- Dynamická superskalárna ZL (out-of-order superscalar)

Zatiaľ čo prvá skupina spracúvava 2-5 inštrukcií paralelne v programovom poradí a tým pádom je veľmi závislá na kvalitnom prekladači, druhá skupina s podporou špekulatívneho vykonávania inštrukcií a hardwarového premenovania registrov vykonáva a inštrukcie mimo poradia. Tieto hardwarové techniky boli vyvinuté k zámeru čo najviac znížiť závislosti, ktoré vedú k pozastavovaniu zreťazenej linky.

Cesta k znižovaniu CPI z pohľadu superskalárnej zreťazenej linky nie je jednoduchá, pretože takáto štruktúra začína byť náchylná na ďalšie závislosti, ktoré v 5-stupňovej skalárnej ZL absentovali, ako napr. dátové závislosti typu WAW a WAR.

Dnešné moderné mikroprocesory kombinujú[3] výhody superzreťazenej a superskálarnej ZL. A ako už bolo spomenuté, s výhodami, ktoré prinášajú, je nutné riešiť problém závislostí, s ktorými sa potyká takýto vysoko sofistikovaný systém. Jeden z takýchto problémov, ktorý nemusí byť na prvý pohľad zrejmý, sa nachádza v rovnici, ktorá vyjadruje výkonnosť zreťazenej linky, a je označený ako MS (Memory stals). MS pozostáva z pozastavenia ZL kvôli samotnému čítaniu pamäte MR (Memory read) a takzvanej pokute MP (Memory penalty), ktorá súvisí s výpadkom dát alebo inštrukcií z pamäte cache. HiT reprezentuje dobu, za ktorú manažment správy pamäte zistí, či sú požadované dáta v pamäti cache. Výpadky spôsobujú práve chýbajúce dáta v pamäti cache, ktoré sa nachádzajú vo vyšších úrovniach pamäte cache alebo v hlavnej pamäti. Doba výpadku záleží od úrovne, do ktorej musí manažment správy pamäte siahnuť po potrebné a korektné dáta, a od povahy (šírka bloku a miera asociativity) , veľkosti a úrovní pamätí cache. Problematika správy pamäte je obšírna a parametre sa navzájom značne ovplyvňujú, preto je vhodné prispôsobovať pamäťový systém konkrétnej ZL s rešpektovaním technológií, ktoré sú k dispozícii.

Z pohľadu porovnania oboch štruktúr je z hľadiska implementácie menej náročnejšia superzreťazená linka. Vyžaduje menej hardwarových prostriedkov pre správu a riadenie zreťazenej linky, čo má vplyv na potrebnú plochu na čipe a tým aj na cenu a samotný vývoj. V tab. č. 2.1 sa nachádza prehľad základných vlastností moderných zreťazených liniek niekoľkých jadier verzie *v7* od firmy ARM.

[3] Takáto kombinácia zreťazenej linky sa často nazýva Super-Super a procesory využívajúce takúto zreťazenú linku sú označované za špekulatívne a dynamicky plánované procesory. Ako príklad je možné uviesť Pentium 4 od Intelu a Opteron založený na architektúre K8 (Hammer) od AMD, ďalej PowerPC 970, ktorý má základ v rodine Power4 od IBM, MIPS R12K a iné. Taktiež MIPS R10000 ako prvý procesor MIPS s dynamickým plánovaním, predstavený už v roku 1996, disponoval pred-dekódovacou jednotkou. Tá zapisuje do inštrukčnej cache pamäte pred-dekódované inštrukcie. Nasleduje 7-stupňová zreťazená linka, z ktorej 5 stupňov patrí výkonným jednotkám.

Tab. č. 2.1 *Prehľad základných vlastností zreťazených liniek verzie v7 firmy ARM*

Jadro	*Hĺbka ZL*	*Povaha ZL*	*Taktovacia frekvencia [MHz]*
Cortex–M3	3	Skalárna + BP	50-100
Cortex–M4	3	Skalárna +BP	150
Cortex–R4	8	Superskalárna	250-620
Cortex–A5	8	Superskalárna	500-1000
Cortex–A8	13	Superskalárna	600-1000 (65nm LP, G)
Cortex–A15	17-24	Superskalárna	2500

2.4 Hazardy

Zreťazená linka je sama o sebe náchylná k závislostiam už svojou povahou a štruktúrou. Architektúra zreťazenej linky svojimi rysmi do istej miery ovplyvňuje povahu závislostí a mieru náchylnosti k istej skupine závislosti. Obecne sa konflikty rozdeľujú do následných troch skupín:

- Dátové hazardy
- Štrukturálne hazardy
- Kontrolné hazardy

2.4.1 Dátové závislosti

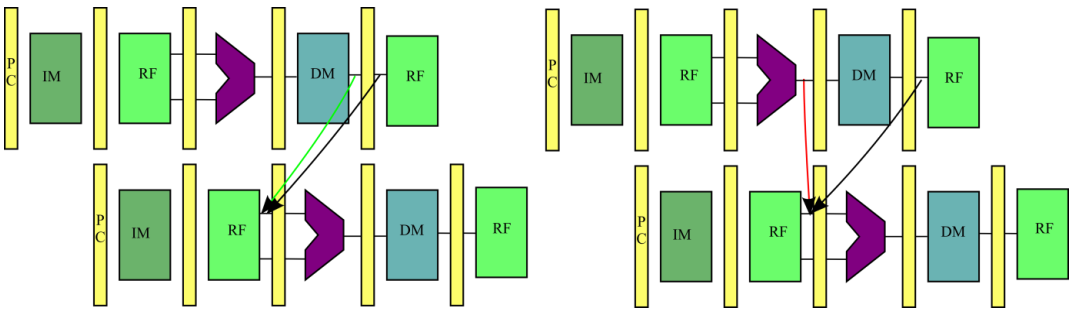
Dátové hazardy, taktiež označované ako údajové závislosti, sú spojene, ako už samotný názov napovedá, s konzistenciou dát. Ide predovšetkým o tri prípady a to RAW, WAR a WAW (pri závislosti RAR ku konfliktom nedochádza). V nasledujúcom texte budú objasnené tieto tri typy dátových závislostí. Väčšia pozornosť bude venovaná závislostiam RAW, pretože sú jediné, ktoré sa vyskytujú v skalárnej zreťazenej linke. Závislosti typu WAR a WAW sa vyskytujú v superskalárnom zreťazenom spracovaní inštrukcií. Z hľadiska náročnosti problematiky a časových možností je týmto technikám venovaná podstatne menšia časť a to len na teoretickej úrovni.

2.4.1.1 Dátová závislosť RAW (Read after Write)

K tejto závislosti dochádza, ak chceme použiť obsah registra, do ktorého ešte nebola zapísaná hodnota výsledku z predchádzajúcej inštrukcie. Tým pádom by došlo k načítaniu nesprávneho obsahu. Pre názornosť uvažujeme dve po sebe idúce inštrukcie „*i1*“ a „*i2*“.

$i1$: $lw\ r8, X\ r4$;

$i2$: $add\ r2, r8, r6;\quad r2 = r8 + r6$



Obr. č. 2.8 *a) konflikt RAW typu načítanie – použitie b) konflikt RAW typu použitie-použitie*

Prvá inštrukcia načítava z dátovej pamäti dáta, ktoré sú zapísané do registra *r8.* Keďže *i1* používa *r8* ako cieľový a inštrukcia *i2* idúca hneď po *i1* používa *r8* ako zdrojový register, správny obsah register *r8* (pre inštrukciu *i2*) nadobudne až za dva strojové cykly. To značí, že inštrukcia *i2* je závislá na *i1* spôsobom *RAW načítanie - použitie.* Z obr. č. 2.8 a) je zreteľné, že obsah, ktorý ma byť uložený do cieľového registra *r8,* môžeme skratkou prepraviť z výstupu dátovej pamäte na vstup registru zreťazenej linky pomocou rozšírenia multiplexorov. Výhodou tohto riešenia je fakt, že skrátime pozastavenie vykonávania inštrukcie *i2* z dvoch cyklov na jeden strojový cyklus. Nevýhoda, okrem nutnosti použiť širšie multiplexory, tkvie v tom, že obsah nebol zapísaný do registra *r8* a pri opätovnom použití daných dát budeme musieť ich znova z pamäti načítavať. Samozrejme je nutné implementovať zložitejšiu réžiu výberu správnych dát. Táto technika sa v anglickej literatúre nazýva *forwarding* alebo *bypassing.*

Okrem konfliktu RAW typu načítanie – použitie dochádza k RAW konfliktom aj pri dvoch po sebe idúcich aritmetických operáciách. V tomto prípade RAW konfliktu ide o typ *výpočet – požitie* . Z obr. č. 2.8 b) je vidieť, že tento konflikt sa dá úplne odstrániť pridaním ďalšej zbernice z výstupu ALU na vstup ALU. Keďže sa jedná o rovnakú techniku, nevýhody sú totožné z predchádzajúcim prípadom.

2.4.1.2 Dátová závislosť WAR (Write after Read)

Tieto závislosti sú doménou superskalárnych procesorov, ktoré vykonávajú inštrukcie mimo poradia. Táto technika je v anglickej literatúre označovaná ako „out of order“. Pre názornosť predpokladajme dve po sebe idúce inštrukcie:

$$i1\text{:}\ \ div\ r8, r3, r4;$$
$$i2\text{:}\ \ sub\ r3\ r8, r6;$$

Ak sa vykoná inštrukcia *i2* pred inštrukciou *i1,* dôjde k chybe, pretože sa prepíše hodnota v *r3* a dostaneme chybný výsledok pri vykonávaní inštrukcie „*i1“*. Riešením je umiestnenie pamäťového modulu za každú jednotkou, ktorý uchová hodnoty výsledkov tak, aby sa následne mohli ukladať do registrového poľa v poradí. Samozrejme je následne kladená požiadavka na réžiu takéhoto vykonávania a spracovania inštrukcií.

2.4.1.3 Dátová závislosť WAW (Write after Write)

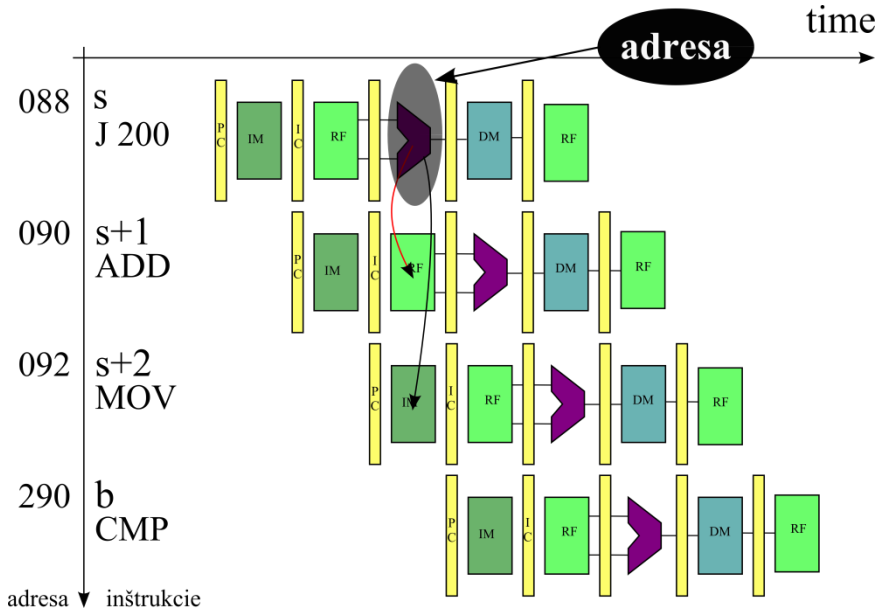
Tak ako predchádzajúca závislosť, tak aj závislosť typu WAW sa vyskytuje v zreťazených linkách superskalárnych procesorov. Zavedením dvoch výpočtových jednotiek s rôznou latenciou strojových cyklov nastane to, že jednotka, ktorá vykonáva inštrukcie s menšiu latenciu, zapíše výsledok skôr ako jednotka, ktorá ma väčšiu latenciu vykonávania. Ako príklad uveďme dve jednotky. Jednotka sčítačky ADD s latenciou 1 strojový cyklus a iteračné pole pre násobenie a delenie MULT s latenciou 3 strojové cykly. Na jednotku MULT je vyslaná inštrukcia. O takt neskôr prichádza inštrukcia na jednotku ADD. Jednotka pri ďalšom hodinovom impulze môže zapisovať, avšak jednotka MULT stále pracuje na výsledku inštrukcie. Dôležité je v tejto situácii to, ako na sebe dané dve inštrukcie závisia. Ak závisia, dostávame sa ku konfliktu a k problému s koherenciou dát. Pomer latencií použitých výkonných jednotiek môže veľmi jednoducho vyústiť aj k štrukturálnym závislostiam na bloku registrového poľa.

2.4.2 Riadiace hazardy (závislosti)

K ďalšiemu narušovaniu plynulosti zreťazeného spracovania dochádza pri vykonávaní podmienených a nepodmienených skokových operácií. Pri spracovávaní inštrukcií v danom poradí sa narušuje sled vykonávaných inštrukcií tým, že po výsledku podmienky sa má skočiť na inú adresu. U nepodmienených skokových operácii nie je žiadna podmienka a skáče sa vždy. Keďže vypočítaná adresa a výsledok testu podmienky je známy až v stupni EX[4] a uvažovali by sme 5 stupňovú zreťazenú linku, stratili by sme vždy minimálne 2 strojové cykly behom jednej podmienenej skokovej inštrukcie.

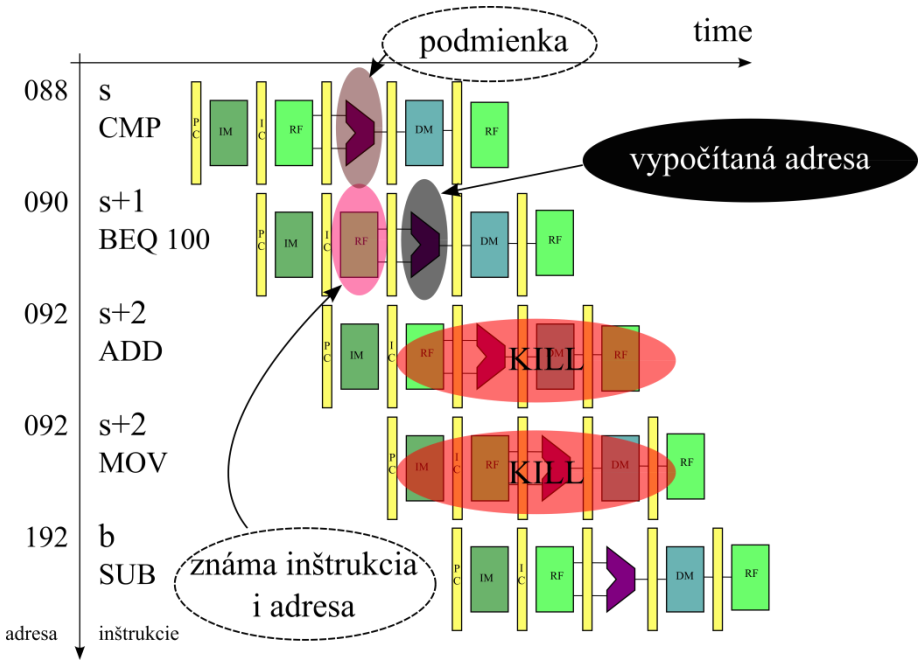
Predpokladajme skokovú inštrukciu „*s“* a jej vykonanie v treťom stupni. Zatiaľ čo je inštrukcia „*s“* vykonávaná, „*s+1“* je dekódovaná a „*s+2“* je načítavaná. Túto skutočnosť znázorňuje 0. Obe operácie „*s+1“* a „*s+2“* sa zahadzujú a čaká sa dokončenie operácie *„b“,* ktoré nastane po piatich taktoch.

[4] Stupeň č.3 (EXECUTE) , kde je umiestnená FX sčítačka. Pri tejto úvahe sa predpokladá základná zreťazená linka, nijako zvlášť modifikovaná. U nepodmienených skokových operácii je možné pridaním sčítačky do stupňa č. 2 (DECODE) ušetriť jeden strojový cyklus, pretože už je známa inštrukcia a adresa a nie je potrebné čakať na výsledok podmienky.



Obr. č. 2.9 *Riadiaca závislosť pri nepodmienenej skokovej operácii*

Ak by sa jednalo o inštrukciu podmieneného skoku znázorneného na obr. č. **Chyba! Nenalezen zdroj odkazů.**, tak v prípade negatívneho testu podmieneného výrazu skokovej inštrukcie sa skok neprevedie a načítané inštrukcie *„s+2“* a *„s+3“* môžu[5] dobehnúť v poradí, v akom boli načítané. V tomto prípade k inštrukcii *„b“* nedôjde a program bude pokračovať s inštrukciami *„s+4“*, *„s+5“*, *„s+6“* atď. až do ďalšieho vetvenia programu či nepodmieneného skoku.



[5] V prípade ak nepoužívame pozitívnu statickú predikciu skokov.

Obr. č. 2.10 *Riadiaca závislosť pri podmienenej skokovej operácii*

Iná situácia nastáva v prípade pozitívneho výsledku testu. Tu záleží na danej inštrukčnej sade a architektúre zreťazenej linky, za koľko strojových cyklov môže nastať načítanie inštrukcie na adrese, kam sa vetví program. Mnou navrhnutý inštrukčný set nesie podmienku a adresu v jednej inštrukcii, ale podmienka sa testuje na základe porovnávania registrov v predchádzajúcej inštrukcii v stupni EX, kde sa uložia príznakové bity. Ak by sme predpokladali základnú štruktúru zreťazenej linky, tak k načítaniu inštrukcie *„b“* by došlo až pri tretej aktívnej hodinovej hrane. To znamená, že sú stratené obe inštrukcie (*„s+2“* a *„s+3“*) idúce za skokovou inštrukciou *„s+1“*, tak ako to je znázornené na obr. č. 2.10**.**

Modifikáciou[6] môžeme dosiahnuť to, že namiesto dvoch stratených strojových cyklov stratíme len jeden. Tejto metóde bude venovaná pozornosť na aplikačnej úrovni v kapitole pre návrh zreťazenej linky.

MIPS procesory majú navrhnutý inštrukčný set tak, že nesú adresy dvoch porovnávacích registrov a adresu na skok v jednej inštrukcii. V prípade oboch[7] sčítačiek umiestnených v stupni EX je priepustnosť zreťazenej linky pozastavená na dva strojové cykly.

2.4.3 Štrukturálne závislosti

V závislostiach tohto typu, ako už názov napovedá, ide o štruktúru blokov a ich implementáciu v zreťazenom spracovaní dát. Tieto závislosti sa tiež nazývajú závislosťami prostriedkov, pretože ak jedna inštrukcia využíva nejaký prostriedok zreťazenej linky, je nutné čakať dokiaľ sa daný prostriedok uvoľní, aby ho mohla ďalšia inštrukcia využiť. Ako príklad je možno uviesť blok pamäte, predovšetkým pamäť cache, ktorú je potrebné použiť v bloku pre načítanie inštrukcií a zároveň[8] pre uloženie dát. Ďalším príkladom môže byť výkonná jednotka v zložitejších štruktúrach superskalárnych procesorov, ktoré vykonávajú inštrukcie mimo poradia.

Štrukturálne závislosti sa prejavujú aj na registrovom poli zakomponované v bloku pre dekódovanie inštrukcie. Implementácia dvojportového[9] registrového poľa v zreťazenom spracovaní inštrukcií by bola prakticky nepoužiteľná. Aby mohli byť inštrukcie vykonávané

[6] Pridaním sčítačky do stupňa DEC.
[7] Sčítačka FX pre komparáciu a sčítačka pre výpočet adresy. Obecne majú vyššie rady MIPS procesorov viacero sčítačiek. Napr. MIPS R10000 ma 2 sčítačky typu integer a jednu pre výpočet adresy. Ďalej obsiahnuté sčítačky pre výpočet v pohyblivej desatinnej čiarke.
[8] Predpokladá sa dvojportová (čítanie, zápis) pamäť, ktorá nie je rozdelená na inštrukčnú a dátovú.
[9] Jedna brána pre zápis a jedna brána pre čítanie.

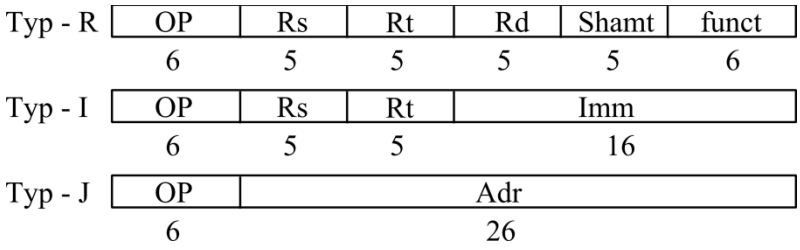
v jednom strojovom cykle, musí[10] registrové pole disponovať minimálne dvoma čítacími bránami a jednou zápisovou bránou. Obecne sú tak tieto závislosti potlačované pridávaním funkčných jednotiek, zberníc a brán.

[10] V tejto úvahe sa predpokladá skalárna zreťazená linka. Pri zreťazenej superskalárnej linke záleží na povahe navrhnutej linky, hlavne na počte výkonných jednotiek a taktiež záleží na počte samotných registrových polí. Pokročilejšie architektúry disponujú viacerými registrovými poľami. Napríklad registrové pole prepojené na jednotky, ktoré operujú s dátovým typom integer a registrové pole prepojené na jednotky, ktoré pracujú v pohyblivej radovej čiarke. Takéto registrové pole disponuje aj siedmimi bránami.

3 Filozofia návrhu inštrukčného setu.

Koncepcia a metodika návrhu inštrukčného setu je kritický proces, pri ktorom je nutné dávať veľký dôraz na celkovú komplexnosť inštrukčného setu. To znamená, že je potreba eliminovať takzvané „hluché miesta“, ktoré môžu spôsobovať, že niektoré časti programu napísané vo vyššom programovacom jazyku sa budú neefektívne prekladať do strojového kódu. Na jednej strane musíme zohľadňovať hľadisko softwaru a na druhej strane hľadisko hardwaru, s ktorým je návrh inštrukčného setu veľmi úzko previazaný. Z hľadiska hardware je dôležité, aké adresovacie módy sú použité a ich počet. Tomu odpovedá cena a zložitosť sýstému. Podrobný popis hardware sa nachádza v kapitole pre návrh mikroarchitektúry.

Pri 32 bitovej *Load-Store* architektúre môžeme všeobecne rozdeliť inštrukcie do troch skupín. Toto delenie vychádza z formy usporiadania segmentov v inštrukcii (rozsah a umiestnenie segmentu pre operačný kód, cieľový, zdrojový register, prípadne konštanta a pod.) a začali ho používať RISC procesory MIPS. Prvú skupinu tvoria inštrukcie, ktoré vykonávajú aritmeticko-logické operácie a pracujú výhradne s registrovým poľom. Preto sa často inštrukcie tohto typu označujú písmenom *R* (R ako Register). Ďalšou všeobecne známou skupinou je skupina typu *I* (I ako Immediate). Inštrukcie tohto typu majú tú vlastnosť, že uchovávajú v sebe hodnotu, s ktorou sa môže ihneď pracovať (pričítať, odčítať, uložiť do zvoleného registra a pod.). Poslednou z troch skupín je skupina *typu J* (J ako Jump), ktorá predstavuje inštrukcie skokových operácií.



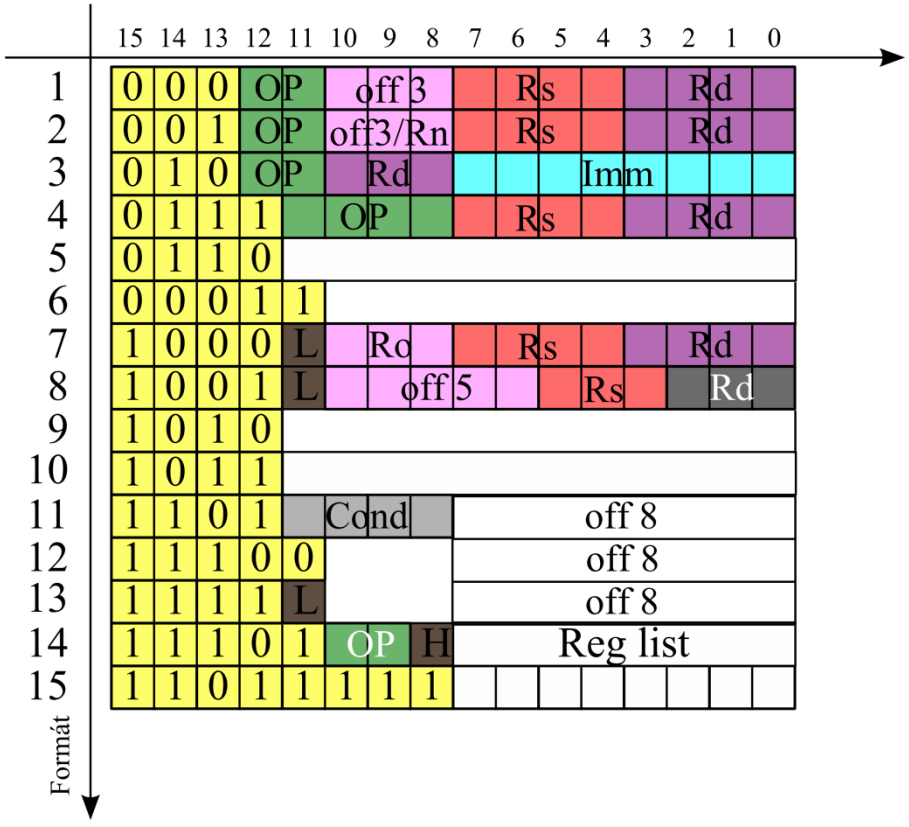
Obr. č. 3.1 *Jedna z variánt typov inštrukcií procesorov MIPS*

Takto popísané typy majú dĺžku 32 bitov. Keďže náš inštrukčný set je navrhovaný pre dĺžku 16 bitov, čo je polovica, za úsporu 16 bitov na jednu inštrukciu zaplatíme zložitejším návrhom. To sa odzrkadlí v nerovnomernej segmentácii a rozložení funkčných polí pre rôzne typy inštrukcií. Tento dôsledok vedie k väčšiemu počtu formátov, hlavne pri snahe dosiahnuť vysokú komplexnosť inštrukčného setu. Z hľadiska hardwaru je taktiež náročnejšie navrhovanie dekódovacej časti zreťazenej linky. Nerovnomerné rozloženie segmentov

prispieva k navyšovaniu počtu multiplexorov. Mnou navrhnutý inštrukčný set vychádza z inštrukčného setu pre mikroprocesory rodiny ARM nesúci názov Thumb. Dôvodom pre tento krok bolo to, aby sa študenti zoznámili s takým inštrukčným setom, ktorý je podobný nejakému na trhu, keďže sa bude jednať o procesor pre výukové účely.

Pôvodná koncepcia, z ktorej vychádza tento návrh, obsahuje 19 formátov a adresuje sa vždy iba 8 registrov. Buď ide o spodnú alebo hornú polovicu registrového pola. Popis toho inštrukčného setu sa nachádza v [8].

Rozšírením segmentov z 3 na 4 bity pre adresovanie registrového poľa sa docielil prístup k všetkým 16 všeobecne prístupným registrom. Odpadajú tak z hardwarového hľadiska dva multiplexory, ktoré prepínali medzi horným a dolným segmentom registrového poľa. Ďalej odpadajú inštrukcie pre prenos medzi spomínanými segmentmi. Keďže sa niektoré segmenty v inštrukcii zväčšia, tak niektoré sa budú musieť zmenšiť, aby sa zachovala konštantná dĺžka 16 bitovej inštrukcie. To ma za následok zníženie počtu formátov a isté zjednodušenie na úkor komplexnosti.



Obr. č. 3.2 *Inštrukčná mapa*

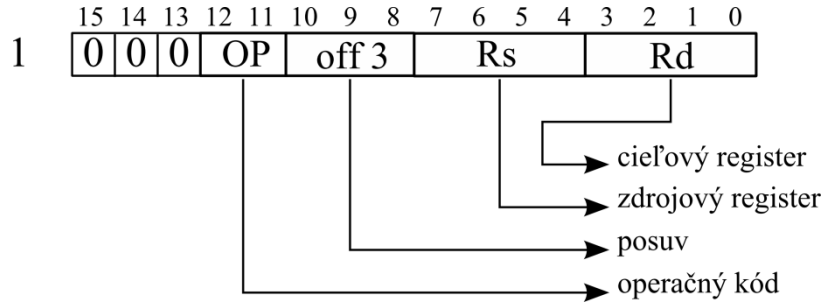
Celkovo bolo navrhnutých pätnásť[11] formátov, ktoré sú medzi sebou odlíšené

[11] Niektoré formáty sú prázdne a niektoré nie sú úplne využité. Naskytuje sa tak možnosť inštrukčný set doplňovať a vylepšovať, čo je vhodné pre výukové účely.

takzvaným inštrukčným kľúčom . Na obr. č. 3.2 je inštrukčný kľúč znázornený poľom žltej farby. Je vidieť, že toto pole inštrukčného kľúča je premennej dĺžky a to z dôvodu, aby sa dosiahlo prijateľného pomeru medzi počtom inštrukčných formátov a počtom potrebných bitov pre daný formát a jeho funkciu. Keďže samotná inštrukčná sada je stále vo vývoji, boli ponechané štyri voľne formáty na prípadné doplnenie a rozšírenie jej aplikačných možností.

Formáty navrhnutého inštrukčného setu

Formát č. 1: Aritmeticko-logické operácie: bitový posun



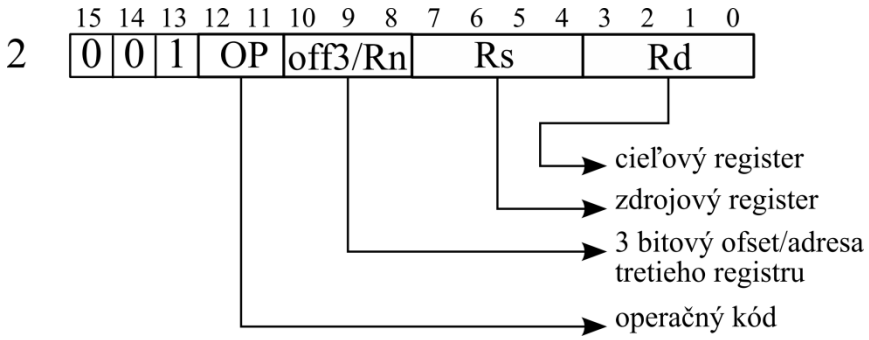
Obr. č. 3.3 *Formát č. 1*

Tento formát obsahuje 3 inštrukcie, ktoré pracujú s dvoma registrami a ofsetom na troch bitoch, v ktorých je uložená informácia o posune. Segment OP značí operačný kód a jeho kódový význam je popísaný v tab. č. 3.1. Ak si všimneme inštrukčnú mapu na obr. č 3.2, je vidieť, že bity číslo 15,14,13 sú totožné v dvoch formátoch. A to konkrétne vo formáte č.1 a č. 6. Preto z hľadiska jednoznačného dekódovania je kódový význam 11 (posledný riadok tab. č. 3.1) operačného kódu rezervovaný.

Tab. č. 3.1 *Inštrukcie formátu 1*

Operačný kód	***Asembler***	***Popis***
00	Voľné	-
01	LSL Rd, Rs, #Offset3	Posun doprava o hodnotu v ofsete
10	LSR Rd, Rs, #Offset5	Posun doľava o hodnotu v ofsete
11	-	Rezervované pre formát 6

Formát č. 2: Aritmeticko-logické operácie: sčítanie, odčítanie



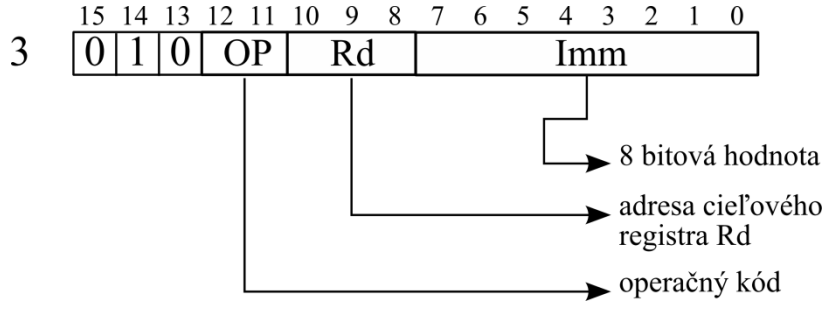
Obr. č. 3.4 *Formát č. 2*

Tento formát obsahuje 4 inštrukcie, ktoré sú druhou podmnožinou aritmeticko-logických operácií a pracujú fixne so *zdrojovým registrom Rs* a *cieľovým registrom Rd*. Operačný kód vyberá medzi sčítaním a odčítaním, a to buď za použitia *tretieho registra Rn* alebo trojbitovej konštanty (čísla 1 až 8). Kódový význam je popísaný v tab. č. 3.2.
Poznámka: Ako register Rn môže byť vybraný register z dolného registrového poľa.

Tab. č. 3.2 *Inštrukcie formátu 2*

Operačný kód	***Asembler***	***Popis***
00	ADD Rd, Rs, Rn	Rd <- Rs + Rn
01	ADD Rd, Rs, #Offset3	Rd <- Rs + Offset3
10	SUB Rd, Rs, Rn	Rd <- Rs - Rn
11	SUB Rd, Rs, #Offset3	Rd <- Rs - Offset3

Formát č. 3: Aritmeticko-logické operácie



Obr. č. 3.5 *Formát č. 3*

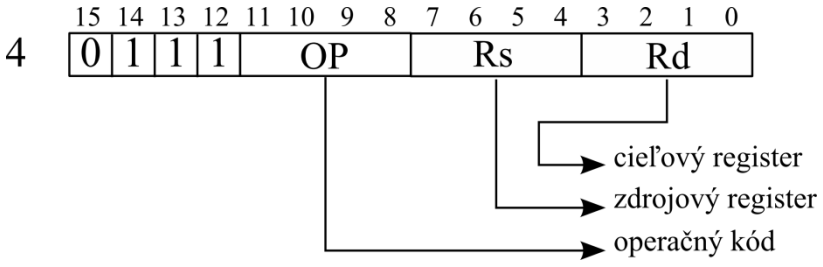
V tomto formáte sú obsiahnuté inštrukcie ďalšej podmnožiny aritmeticko-logických operácií, ktorá pracuje z jediným a to *cieľovým/zdrojovým registrom Rd/Rs.* Segmenty, ktoré boli v predchádzajúcom formáte vyčlenené pre zdrojový a cieľový register, v tomto formáte predstavujú 8 bitovú hodnotu, s ktorou je možné ihneď pracovať. Operačný kód vyberá medzi presunom, komparáciou, sčítaním a odčítaním. Kódový význam je popísaný v tab. č. 3. 2.
Poznámka: Ako register Rd/Rs môže byť vybraný iba register z dolného registrového poľa a adresa Rd je totožná s Rs!

Tab. č. 3.3 *Inštrukcie formátu 3*

Operačný kód	***Asembler***	***Popis***
00	MOV Rd, #Offset8	Rd <- Offset8
01	CMP Rd, #Offset8	Rd-Offset8
10	ADD Rd, #Offset8	Rd <- Rs + Offset8
11	SUB Rd, #Offset8	Rd <- Rs – Offset8

Formát č. 4: Operácie s aritmeticko-logickou jedntkou

Formát č. 4 v sebe obsahuje inštrukcie, ktoré vykonávajú v poradí už tretiu podmnožinu aritmeticko-logických operácii s tým rozdielom, že sa pracuje s dvomi registrami. To znamená, že *cieľový register Rd* je použitý ako zdrojový pre ľavý operand a zároveň ako *cieľový register Rd* pre výsledok operácie. Matematicky to znamená, že ľavý operand bude po prevedení aritmeticko-logickej operácie stratený. Oproti prvej podmnožine aritmeticko-logických operácii vyjadrenej formátom č. 2, sa v tejto podmnožine nachádza samozrejme omnoho viac inštrukcií, čo sa dosiahlo rozšírením operačného kódu na úkor segmentu pre adresovanie tretieho registra. Kódové významy spolu asemblerom a popisom sa nachádzajú v tab. č. 4.



Obr. č. 3.6 *Formát č. 4*

Tab. č. 3.4 *Kódové významy inštrukcii formátu 4*

Operačný kód	*Asembler*	*Popis*
0000	AND Rd, Rs	Rd <- Rd AND Rs
0001	EOR Rd, Rs	Rd <- Rd EOR Rs
0010	OR Rd, Rs	Rd <- Rd OR Rs
0011	NND Rd, Rs	Rd <- NAND Rs
0100	NEG Rd, Rs	Rd=- Rs
0101	-	-
0110	-	-
0111	-	-
1000	ADC Rd, Rs	Rd <- Rd + Rs + CY
1001	SBC Rd, Rs	Rd <- Rd - Rs - CY
1010	CMP Rd, Rs	Rd- Rs
1011	CMP NRd, Rs	Rd+ Rs
1100	-	-
1101	-	-
1110	-	-
1111	-	-

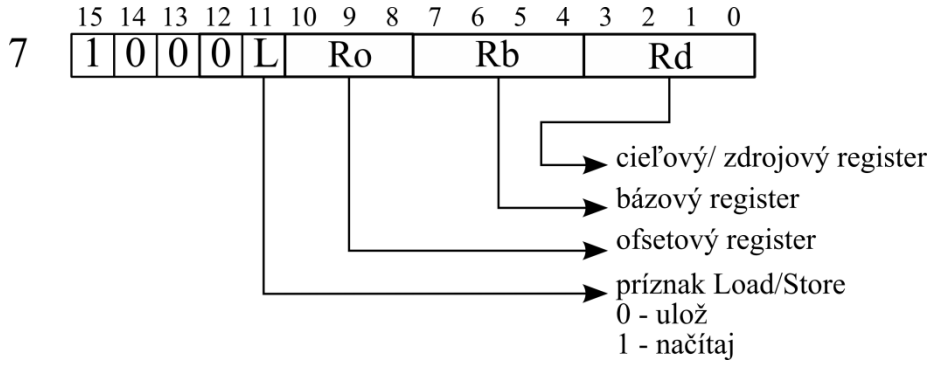
Formát č. 5 a č. 6

Formát je voľný a umožňuje implementovanie ďalších inštrukcií.

Formát č. 7: Presuny medzi pamäťou a GPR

Prvý z formátov, ktorý obsahuje dve inštrukcie typu Load/Store. Pracuje s dvoma adresami registrov na 4 bitoch a jedným ofsetovým registrom na 3 bitoch. Segment označený ako Rd je pri príznaku L=0 zdrojovým registrom a jeho hodnota sa zapíše do časti pamäte, ktorú tvoria hodnoty uložené v bázovom a ofsetovom registri. V prípade, ak príznak L = 1, nastáva situácia, kedy segment Rd plní úlohu adresy cieľového registra. Opäť bázový a ofsetový register tvoria adresu, ale v tomto prípade je hodnota na tejto adrese dátovej pamäte presunutá do registrového poľa na adresu cieľového registra. Bit L rozlišuje medzi

operáciami načítanie hodnoty z dátovej pamäti do registrového poľa a uloženie hodnoty registru do dátovej pamäti.



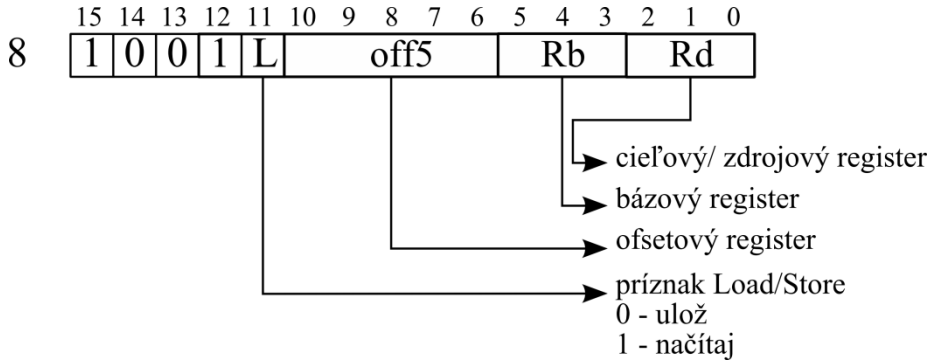
Obr. č. 3.7 *Formát č. 7*

Tab. č. 3.5 *Kódové významy inštrukcií formátu 7*

L	***Asembler***	***Popis***
0	STR Rd, [Rb, Ro]	Ulož do pamäte hodnotu registera adresovaného segmentom Rd. Adresa vypočítaná bazovým registrom Rb a ofsetovým registrom Ro.
1	LDR Rd, [Rb, Ro]	Načítaj z pamäte hodnotu a ulož ju do registra adresovaného segmentom Rd. Adresa vypočítaná bazovým registrom Rb a ofsetovým registrom Ro.

Formát č. 8 Presuny medzi pamäťou a GPR

Tento formát je v poradí druhý, ktorý obsahuje inštrukcie typu Load/Store. Konkrétne sa jedná o dve inštrukcie podobného charakteru ako v predchádzajúcom formáte s jedným rozdielom, a to tým, že miesto ofsetového registra je použitý ofset držaný v samotnej inštrukcii. Teoreticky by bolo možné dodržať segmentáciu a miesto adresy ofsetového registra uchovávať ofset na troch bitoch, ale z praktického hľadiska to nie je veľmi efektívne. Aby bolo možné dosiahnuť širšieho ofsetu, je nutné tieto bity ubrať z iných segmentov. Jedinou cestou je zníženie adresy bázového a cieľového, respektíve zdrojového registru.



Obr. č. 3.8 *Formát č. 8*

Tab. č. 3.6 *Kódové významy inštrukcií formátu 8*

L	***Asembler***	***Popis***
0	STR Rd, [Rb, offset5]	Ulož do pamäte hodnotu registra adresovaného Rd. Adresa vypočítaná bázovým registrom Rb a 5 bitovým ofsetom.
1	LDR Rd, [Rb, offset5]	Načítaj z pamäte hodnotu a ulož ju do registra adresovaného segmentom Rd. Adresa je vypočítaná bázovým registrom Rb a 5 bitovým ofsetom.

Formát č. 9

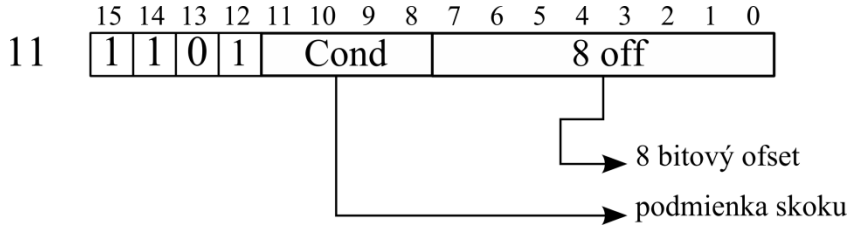
- Formát rezervovaný implementácii inštrukcii PUSH a POP.

Formát č. 10

- Formát je voľný a umožňuje implementovanie ďalších inštrukcií.

Formát č. 11: Podmienené skoky

Tento formát obsahuje prvú podmnožinu skokových operácií. Konkrétne ide o podmnožinu podmienených skokov. Celkovo sa jedná o 14 inštrukcii, ktoré sú zakódované v podmienke. Kódový význam je v tab. č. 3.8.



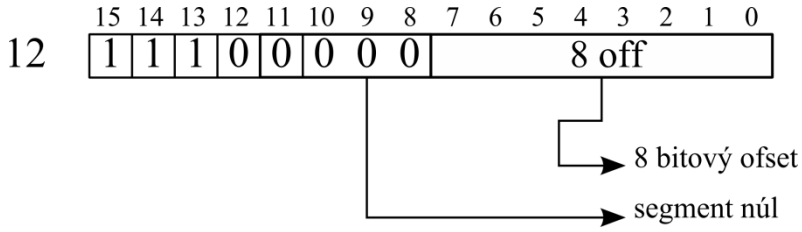
Obr. č. 3.9 *Formát č. 11*

Tab. č. 3.7 *Kódové významy inštrukcií formátu 11*

podmienka	***Asembler***	***Popis***
0000	BEQ label	Skok, ak je nastavený príznak Z (zero)
0001	BNE label	Skok, ak je zhodený príznak Z (zero)
0010	BCS label	Skok, ak je nastavený príznak C
0011	BCC label	Skok, ak je zhodený príznak C
0100	BMI label	Skok, ak je nastavený príznak N(negatívny)
0101	BPL label	Skok, ak je zhodený príznak N(výsledok pozitívny alebo nula)
0110	BVS label	Skok, ak je nastavený príznak V(indikácia pretečenia)
0111	BVC label	Skok, ak je zhodený príznak V
1000	BHI label	Skok, ak je C = 1 a Z = 0
1001	BHS label	Skok, ak je C = 0 a Z = 1
1010	BGE label	Skok, ak N = 1 a V = 1 alebo N = 0 a V = 0 (väčší alebo rovný)
1011	BLT label	Skok ak N = 1 a V = 1 alebo N = 0 a V = 0 (väčší alebo rovný)
1100	BGT label	Skok, ak N = 1 a V = 0 alebo N = 0 a V = 1 (menší ako)
1101	BLE label	Skok, ak Z = 1 alebo N = 1 a V = 0 alebo N = 0 a V = 1 (menší ako alebo rovný)
1110	-	-
1111	Rezervované	Formát 15

Formát č. 12 Nepodmienený skok

Tento formát implementuje operáciu nepodmieneného relatívneho skoku. Segment núl môže byť pri modifikácií architektúry využitý k rozšíreniu adresovacieho priestoru.



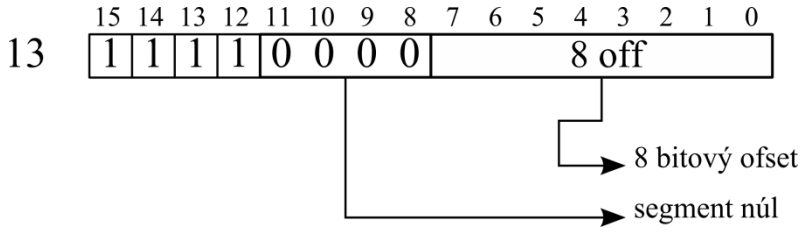
Obr. č. 3.10 *Formát č. 12*

Tab. č. 3.8 *Kódové významy inštrukcií formátu 12*

Asembler	***Popis***
BR label	Hodnota relatívneho skoku

Formát č. 13: Nepodmienený skok

Formát č. 13 implementuje operáciu nepodmieneného absolútneho skoku. Segment núl môže byť pri modifikácií architektúry využitý k rozšíreniu adresovacieho priestoru.



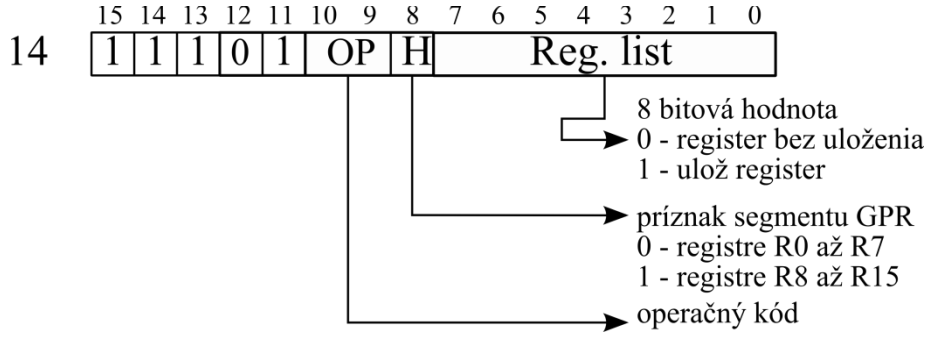
Obr. č. 3.11 *Formát č. 13*

Tab. č. 3.9 *Kódové významy inštrukcií formátu 13*

Asembler	***Popis***
BA label	Hodnota absolútneho skoku

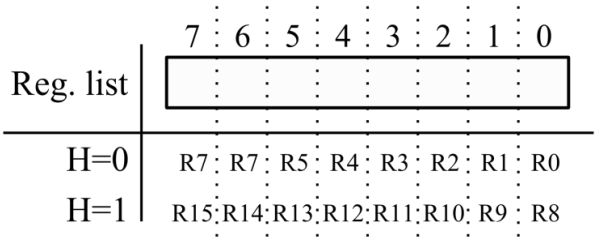
Formát č. 14: Presuny typu PUSH/POP

Tento formát v sebe uchováva 8 inštrukcií pre prácu so zásobníkom. Osem bitové pole označené na obr. č. 16 ako reg. list dáva informáciu o tom, ktoré registre sa majú ukladať do zásobníka. Jednotka na danej pozícii v tomto segmente označuje, že register bude uložený do zásobníka. V opačnom prípade sa register neukladá.



Obr. č. 3.12 *Formát č. 14*

To, ktorý bit odpovedá akému registru, prehľadne znázorňuje obr. č. 17. Vždy ide buď o hornú polovicu alebo dolnú polovicu registrového poľa príznaku H. Kódový význam operačného kódu a príznaku H je stručne popísaný v tab. č. 14.



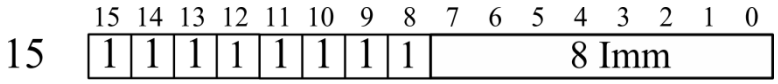
Obr. č. 3.13 *Segment registrový list*

Tab. č. 3.10 *Kódové významy inštrukcií formátu 14*

Operačný kód	***H***	***Asembler***	***Popis***
00	0	PUSH {Rlist}	Vlož požadované registre do zásobníka. (Dolná časť registrového poľa). SP je aktualizovaný
01	0	PUSH {LR, Rlist}	Vlož LR a požadované registre do zásobníka. (Dolná časť registrového poľa). SP je aktualizovaný
10	0	POP {Rlist}	Načítaj do požadovaných registrov hodnoty uložené v zásobníku. (Dolná časť registrového poľa). SP je aktualizovaný
11	0	POP {PC, Rlist}	Načítaj do požadovaných registrov hodnoty uložené v zásobníku + PC. (Dolná časť registrového poľa). SP je aktualizovaný
00	1	PUSH {Rlist}	Vlož požadované registre do zásobníka (Horná časť registrového poľa). SP je aktualizovaný
01	1	PUSH {LR, Rlist}	Vlož LR a požadované registre do zásobníka. (Horná časť registrového poľa). SP je aktualizovaný
10	1	POP {Rlist}	Vlož požadované registre do zásobníka (Horná časť registrového poľa). SP je aktualizovaný
11	1	POP {PC, Rlist}	Načítaj do požadovaných registrov hodnoty uložené v zásobníku + PC. (Horná časť registrového poľa). SP je aktualizovaný

Formát 15: Softwarové prerušenie

Posledný formát obsahuje jednu inštrukciu softwarového prerušenia. Pri vykonávaní tejto inštrukcie sú potrebné úkony z uložením nasledujúcej inštrukcie, aby sa po skončení prerušenia vrátil beh programu do miesta jeho prerušenia. Štruktúra formátu a binárny obsah kľúča je zhodný s formátom inštrukčného setu Thumb (formát 17).



Obr. č. 3.14 *Formát č. 15*

Tab. č. 3.11 *Sumarizačný prehľad navrhnutých inštrukcií.*

formát	***Stručný popis***	***Počet využitých inštrukcií***	***Počet nevyužitých inštrukcií***
1	Bitový posun	2	1
2	Aritmeticko-logické operácie	4	0
3	Aritmeticko-logické operácie	4	0
4	Operácie s ALU	9	7
5	Rezervované	-	-
6	Rezervované	-	-
7	Presuny medzi pamäťou a GPR	2	0
8	Presuny medzi pamäťou a GPR	2	0
9	Rezervované	-	-
10	Rezervované	-	-
11	Podmienené skoky	14	1
12	Nepodmienený skok	1	0
13	Nepodmienený skok	1	0
14	Presuny typu PUSH/POP	8	0
15	Softwarové prerušenie	1	0
celkovo	-	48	9

4 Návrh linky pre zreťazené spracovanie

Táto kapitola diplomovej práce prezentuje vlastný návrh implementácie inštrukčnej sady do zreťazenej linky procesora. Taktiež si táto kapitola kladie za cieľ zašpecifikovať vlastnosti architektúry, z ktorej budú následne odvodzované úlohy jednotlivých blokov začlenených v zreťazenej linke. Návrh štruktúry pre zreťazené spracovanie je jednou z častí návrhu kompletného mikroprocesora. Preto je nutné koordinovať návrh s ďalšími časťami umiestnenými na čipe ako pamäť cache, správa a manažment pamäte, riadenie spotreby, zbernicový systém, DMA, prípadne koprocesor. Z hľadiska návrhu komplexnej zreťazenej linky je vhodné diverzifikovať návrh do dvoch rovín.

- Syntéza základnej štruktúry zreťazeného spracovania
- Aplikácie metód pre riešenie závislostí

4.1 Syntéza základnej štruktúry zreťazeného spracovania

Návrh základnej štruktúry zreťazeného spracovania, od špecifikovania architektúry až po časovanie modelu, je zahrnutý do niekoľkých krokov:

- určenie hĺbky linky zreťazeného spracovania a pridelenie úloh jednotlivým blokom,
- špecifikovanie jednotlivých blokov architektúry na základe úlohy, ktorú majú vykonávať,
- návrh mikroarchitektúry jednotlivých funkčných blokov, s predprípravou pre „*bypassing*“,
- časovanie modelu zreťazeného spracovania, vylepšovanie a úpravy.

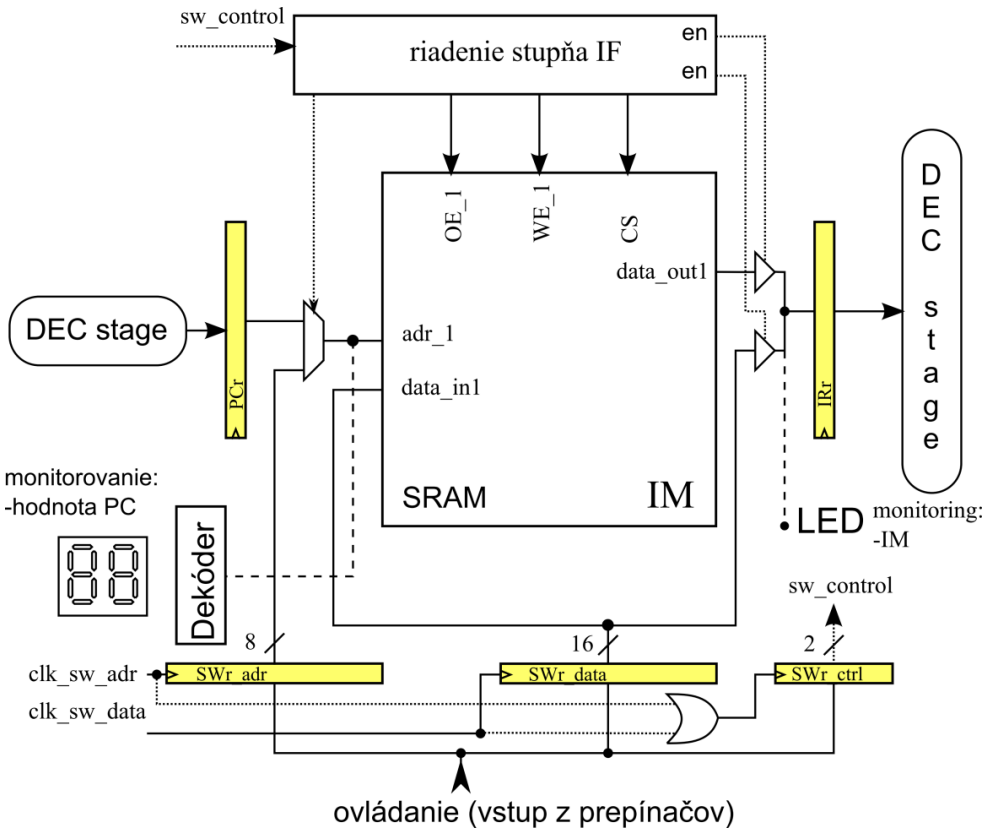
Ako prvé je nutné charakterizovať architektúru. Návrh bude vychádzať z modelu oddelených zberníc pre inštrukcie a pre ukladanie dát, čiže z harwardskej architektúry. Šírky adresovacích zberníc vzhľadom na jednoduchší návrh, prehľadnosť a možnosti hardwarovej vizualizácie, boli stanovené na 8 bitov. To značí, že inštrukčná a dátová pamäť disponuje kapacitou o veľkosti 256 bytov. Ďalším hlavným rysom architektúry je prístup k pamäti dát. Ukladanie do pamäte, respektíve načítanie z pamäte, bude možné jedine pomocou inštrukcií typu *LOAD* a *STORE*. Hĺbka navrhovanej zreťazenej linky je určená týmito piatimi stupňami:

1. Stupeň pre načítavanie inštrukcií - (IF)
2. Stupeň pre dekódovanie inštrukcií - (DEC)
3. Stupeň pre vykonávanie inštrukcie - (EX)
4. Stupeň pre prístup k pamäti dát - (MA)
5. Stupeň pre zápis do registrového poľa - (WB)

V nasledujúcich podkapitolách budú špecifikované úlohy jednotlivých stupňov spolu s návrhom ich mikroarchitektúr. Pri návrhu sa muselo pristúpiť k definovaniu niekoľkých skratiek z dôvodu zvýšenia jednoznačnosti a jednoduchšej identifikácií jednotlivých elementov. Jednotlivé oddeľovacie registre sú označené veľkým písmenom stupňa, ktorému predchádzajú, a malým písmenom *r* ako register. Za podčiarkovníkom (podtržítkom) je následne došpecifikovávaná funkcia tohto registru. Napríklad EXr_ALU_ctrl označuje register oddeľujúci stupeň DEC a EX a je určený k oddeleniu signálov pre riadenie ALU jednotky. U multiplexorov, respektíve ich riadiacich signálov, bolo použité značenie následného tvaru: MUX_1_DEC označuje multiplexor číslo jedna v dekódovacom stupni. Výnimku tvoria signály pre riadenie ALU jednotky, ktoré nesú v názve svoju funkciu v záujme lepšej identifikácie pri testovaní a demonštrovaní na príkladoch.

4.1.1 Blok pre načítavanie inštrukcií

Tento blok tvorí prvý stupeň zreťazenej linky a jeho funkciou je načítavať inštrukcie uložené v pamäti inštrukcií a sprístupniť ich následne bloku pre dekódovanie inštrukcie pro-



Obr. č. 4.1 *Mikroarchitektúra stupňa pre načítavanie inštrukcií s rozhraním programovania inštrukčnej pamäte.*

stredníctvom inštrukčného registra. Aby tento blok mohol byť programovateľný, musí obsahovať rozhranie, ktoré to bude umožňovať. Pre tento účel obsahuje blok registre *SWr_adr, SWr_data* a *SWr_ctrl*, ktoré oddeľujú vkladanie hodnôt adries a inštrukcií vykonávané prostredníctvom prepínačov vývojovej dosky DE2-115. Zápis do týchto registrov je uskutočňovaný pomocou signálov *clk_sw_adr* (pre register *SWr_adr*), a *clk_sw_data* (pre register *SWr_data*). Register *SWr_ctrl* reaguje prostredníctvom hradla OR na oba signály. Tieto hodinové signály sú emulované tlačidlami key3 a key2, čo prezentuje obr. č. 5.2 (kapitola 5).

Hlavným elementom bloku je dvojportová statická RAM (SRAM), ktorá je určená pre uchovanie inštrukcií. Réžiu zápisu a čítania SRAM, prepínania multiplexora MUX1 a povoľovanie trojstavových obvodov zabezpečuje obvod *control_IF* na základe hodnoty vektoru *sw_control.* Navrhovaný význam vektoru je popísaný v tab. č. 6.1 .

Tab. č. 4.1 *Navrhovaný význam vektoru sw_control*

sw_control	***Funkcia***
00	CPU je v režime chodu. Krokovaním za pomoci key1 sú emulované taktovacie hodiny. Inštrukcie sú načítavané z inštrukčnej pamäte a postupujú zreťazenou linkou.
01	Do zreťazenej linky je možné vkladať inštrukcie za pomoci prepínačov. Odpadá tak nutnosť programovania.
10	CPU v režime programovania. Pomocou prepínačov sa nastaví adresa a vloží inštrukcia.
11	Voľba umožňuje krokovaním prezrieť hodnoty uložené v pamätiach.

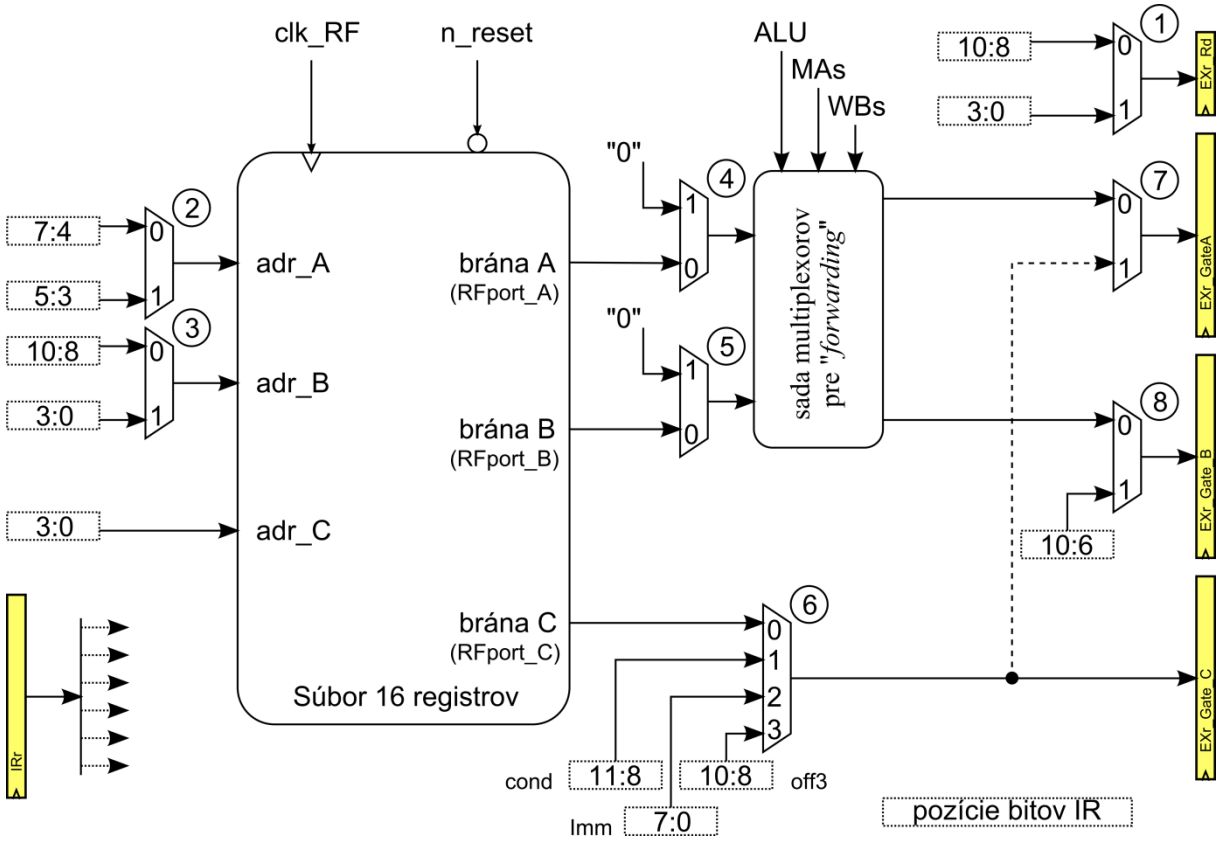
4.1.2 Blok pre dekódovania inštrukcií

Tento blok môžeme rozdeliť do dvoch častí. Tou prvou je časť datová, ktorá na základe adries držaných v IR sprístupní požadované registre. Respektíve môže byť použitá časť IR, ktorá nesie hodnoty priamo v inštrukcii k okamžitému použitiu[12]. Druhou časťou je

[12] Inštrukčné formáty č. 2, 3, 8.

časť riadiaca a jej hlavným prvkom je dekodér inštrukcie.

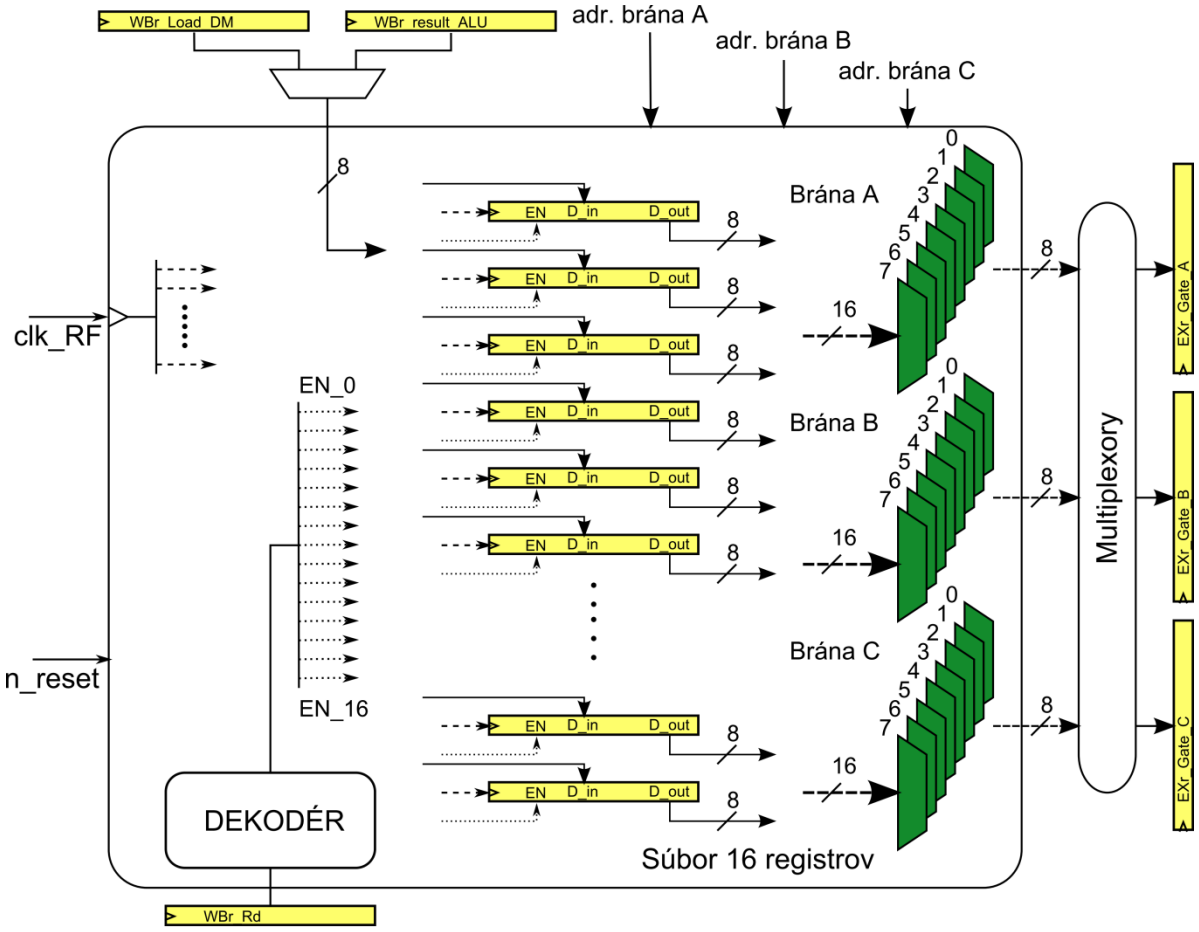
Návrh časti, ktorá sprístupňuje požadované registre k ALU operáciám, je znázornený na obr. č. 4.2, kde hlavným prvkom je osembitové registrové pole so štyrmi bránami[13]. Keďže inštrukčný set je koncipovaný tak, že jednotlivé segmenty adries nie sú umiestnené striktne na tých istých bitových pozíciách v IR, je nutné zaradiť multiplexory na adresné vstupy registrového poľa. Dva multiplexory pre výber adresy (MUX2, MUX3) a jeden pre výber cieľového registra (MUX1) nie je ani zďaleka vysoká cena pre prínos komplexnosti inštrukčnej sady. Ďalším špecifickým multiplexorom, ktorý je nutné zaradiť z hľadiska komplikovanejšieho a sofistikovanejšieho inštrukčného setu, je MUX č. 6. Ten vyberá medzi čítacou bránou C registrového poľa a segmentmi v inštrukcii, ktoré nesú priamo prístupné hodnoty. Ďalej je možné týmto multiplexorom separovať podmienku a vložiť ju na bránu C dekódovacieho stupňa. To umožní laborovanie so skokovými inštrukciami. Poslednou možnosťou je vloženie ofsetu neseného v inštrukcií č. 2 za pomoci multiplexorov MUX6 a MUX7 na bránu A.



Obr. č. 4.2 *Dátová časť dekódovacieho bloku*

[13] Ide o tri brány pre čítanie obsahu registrového poľa a jednu zápisovú bránu.

Povahovo inú kategóriu tvoria multiplexory pripravené pre návrh zberníc pre *„bypassing“*. Často sa táto technika nazýva aj *„forwarding“*. Jej úlohou je zmierniť dopady dátových závislostí. Riadenie týchto multiplexorov nepodlieha dekodéru inštrukcie ale obvodu pre správu závislosti, ktorý ma vyššiu prioritu a dokáže povoľovať a zakazovať hodiny jednotlivým oddeľovacím registrom. Ich pozícia sa na obr. č. 4.2 kvôli prehľadnosti nachádza v bloku *„sada multiplexorov pre forwarding“*.



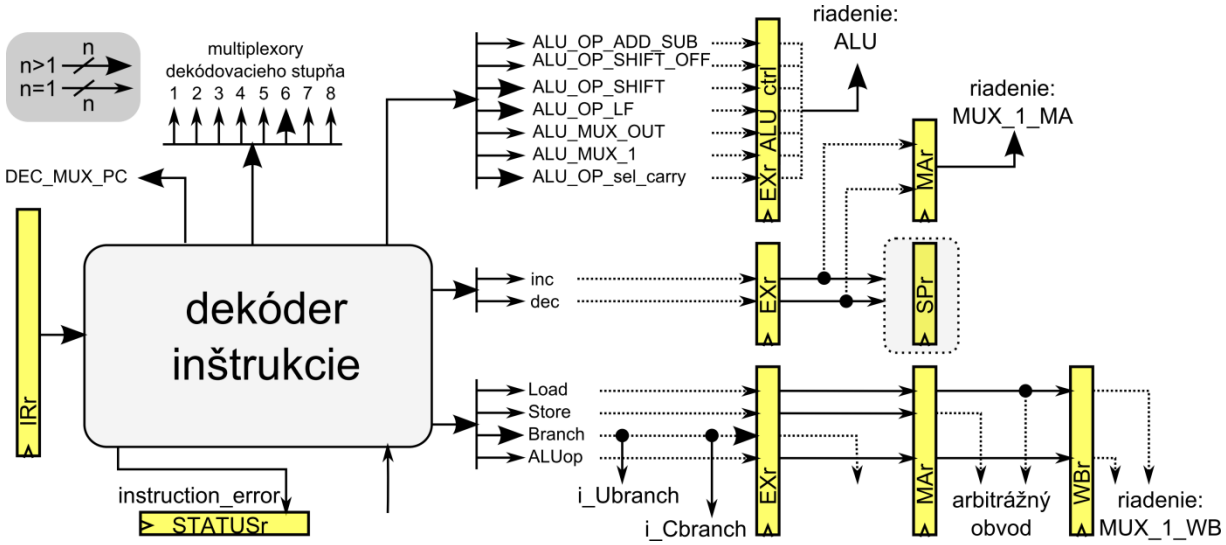
Obr. č. 4.3 *Znázornenie mikro-architektúry registrového poľa s povoľovaním zápisu.*

Na obr. č. 4.3 je znázornený detail návrhu mikroarchitektúry registrového poľa s troma bránami pre čítanie operandov a jednou zápisovou branou. Každá brána, ktorá slúži pre čítanie, potrebuje k svojej činnosti osem 16-vstupových multiplexorov. Vzhľadom k tejto skutočnosti je nutné uvážiť počet jednotlivých brán a ich prínos k architektúre zreťazenej linky ako aj inštrukčnému súboru. Na prvý pohľad by sa mohlo zdať, že pre zreťazenú linku

s takou povahou inštrukčného paralelizmu stačia dve brány určené k čítaniu a jedna brána k zápisu. S výnimkou jednej inštrukcie (navrhnutého inštrukčného súboru) je predchádzajúce tvrdenie správne. Výnimku tvorí ukladanie obsahu registra do pamäte dát.

Pri naozaj precíznom návrhu mikoarchitektúry (hlavne pri väčšom počte brán) vzhľadom k úrovniam samotného návrhu, ktorému bola venovaná prvá kapitola, je nutné sa venovať aj štruktúre samotných multiplexorov. Tá ovplyvňuje na jednej strane počet jednotlivých elementárnych prvkov (hradiel, tranzistorov) a tým plochu čipu, a na strane druhej latenciu samotného multiplexoru.

Riadiaca časť dekódovacieho bloku je znázornená na obr. č. 4.4 a jej hlavným prvkom je dekodér inštrukcie, ktorý na základe IR špecifikuje inštrukciu a pomocou multiplexorov v dekódovacom stupni sprístupní správne segmenty IR. Ďalšou jeho úlohou je, že podľa inštrukcie nastaví riadiace signály pre ALU jednotku a ukazovateľ na zásobník v nasledujúcom stupni. Užitočné je tiež navrhnúť informačné signály, ktoré nesú informáciu o povahe nesenej inštrukcie v stupňoch zreťazenej linky. Tie následne možno jednoducho použiť pri riadení v niektorom zo stupňov zreťazenej linky. Priepustnosť signálov v ZL a ich vyvedenie v potrebných stupňoch je znázornené na obr. č. 4.4 . Naskytuje sa aj úvaha použiť dekodér pre každý stupeň zvlášť a rozdeliť tak zložitosť dekodéra inštrukcie na viacero jednoduchších dekódovacích blokov. Vzhľadom na povahu a štruktúru inštrukčnej sady sa



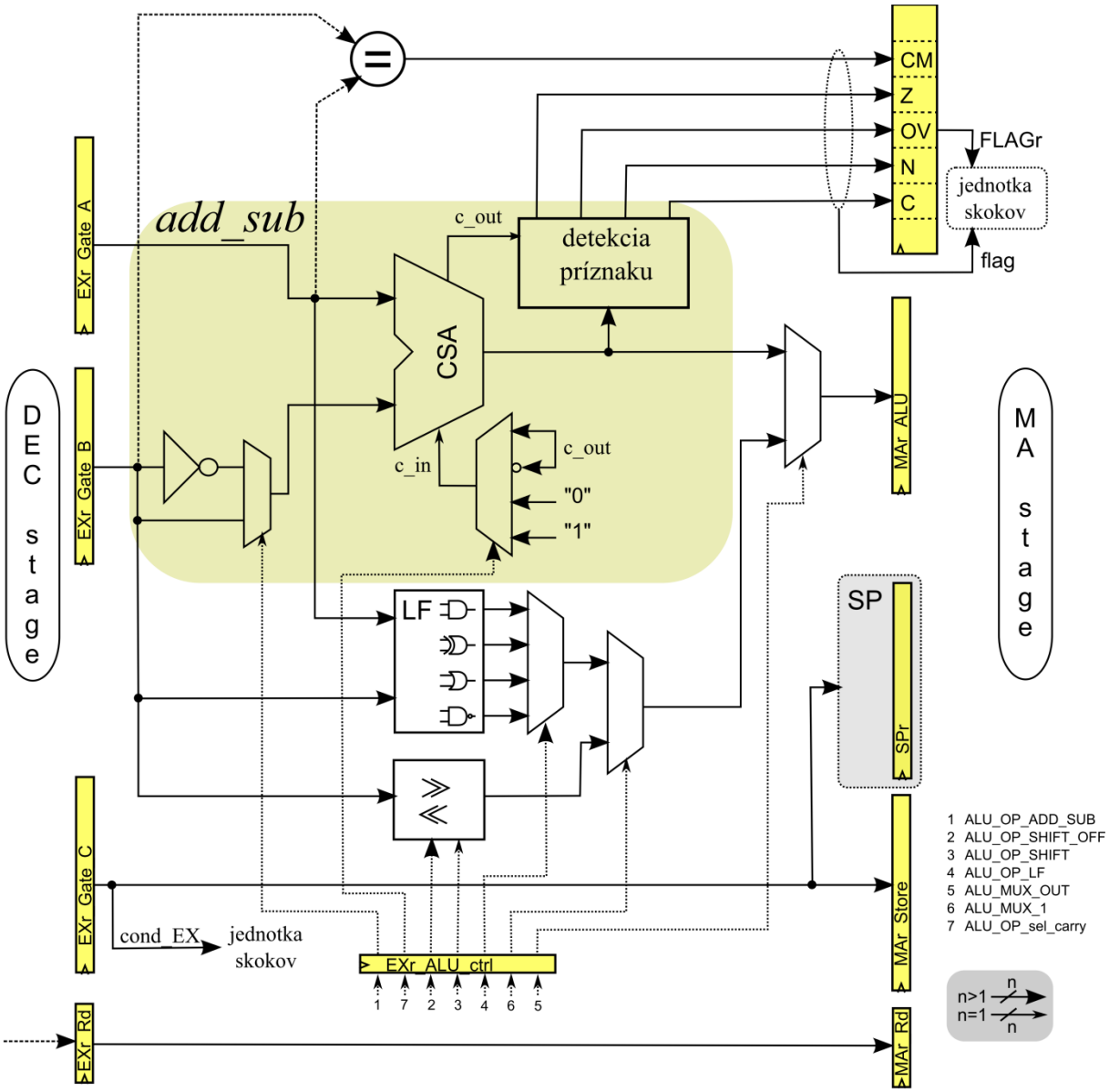
Obr. č. 4.4 *Riadiaca časť dekódovacieho bloku. Riadiace signály v závislosti na zreťazenej linke*

zdá voľba jedného dekodéra inštrukcie výhodnejšia. Jednotlivé segmenty inštrukčnej sady nie sú striktne na tej istej pozícii v IR. Ďalšou vlastnosťou, ktorá hovorí v prospech centrálneho dekodéra, je nejednotná dĺžka spomínaných segmentov. Na tomto príklade je vidieť veľmi úzke previazanie návrhu inštrukčnej sady a samotnej mikroarchitektúry. V prípade neúspešného dekódovania inštrukcie dekodér zapisuje chybu inštrukcie do statusového registra (bit *instruction_error*).

4.1.3 Blok pre vykonávanie inštrukcií

Dominantou tohto bloku je aritmeticko-logická jednotka. Keďže stupeň disponuje iba jednou aritmetickou jednotkou, je na obr. č. 4.5 znázornená priamo mikroarchitektúra ALU. Z dôvodu prehľadnosti nie sú zakomponované do obrázka registre a signály prechádzajúce blokom k stupňom MA a WB. Tomuto aspektu sa venuje obr. č. 4.4. Operandy pre aritmeticko-logické operácie sú brány A a B privádzane na oddeľovacie registre EXr_Gate_A a EXr_Gate_B. Podľa výberu sčítania a odčítania riadiacim signálom ALU_OP_ADD_SUB sa na port A multiplexora ALU_MUX_OUT dostáva výsledok tejto operácie. Korektnosť výsledku kontroluje jednotka pre detekciu príznakov a špecifikuje tak niekoľko informácií o výsledku operácie. Príznakové bity sa po dobu jedného taktu ukladajú do registra príznakov FLAGr. Paralelne k operáciám sčítanie /odčítanie sa vykonávajú logické operácie AND, XOR, OR, NAND. Výber je uskutočňovaný multiplexorom pomocou signálu ALU_OP_LF. Medzi registrom EXr_Gate_C a MAr_Store sa nachádza dátová zbernica, ktorá je využívaná k ukladaniu hodnoty všeobecne prístupného registra do dátovej pamäte. Taktiež umožňuje nastavenie ukazovateľa na ľubovoľnú hodnotu a nesenie podmienky skoku, čo zvyšuje flexibilitu architektúry vzhľadom k experimentovaniu so skokovými operáciami (implementácia oneskorených skokov). Práve z dôvodu experimentovania sú privedené príznakové bity z výstupu a vstupu FLAGr. K demonštrácii užitočnosti uvažujme dve po sebe idúce inštrukcie. Komparačná inštrukcia porovná dve hodnoty, na základe ktorých indikuje príznakové bity. Tento proces je vykonávaný v stupni EX. Paralelne s týmto procesom je ale vykonávané dekódovanie skokovej inštrukcie v stupni DEC. K tomu, aby sa jednotka pre výpočet nasledujúcej adresy mohla rozhodnúť ešte počas tohto strojového cyklu, potrebuje informáciu o hodnote príznakových bitov.

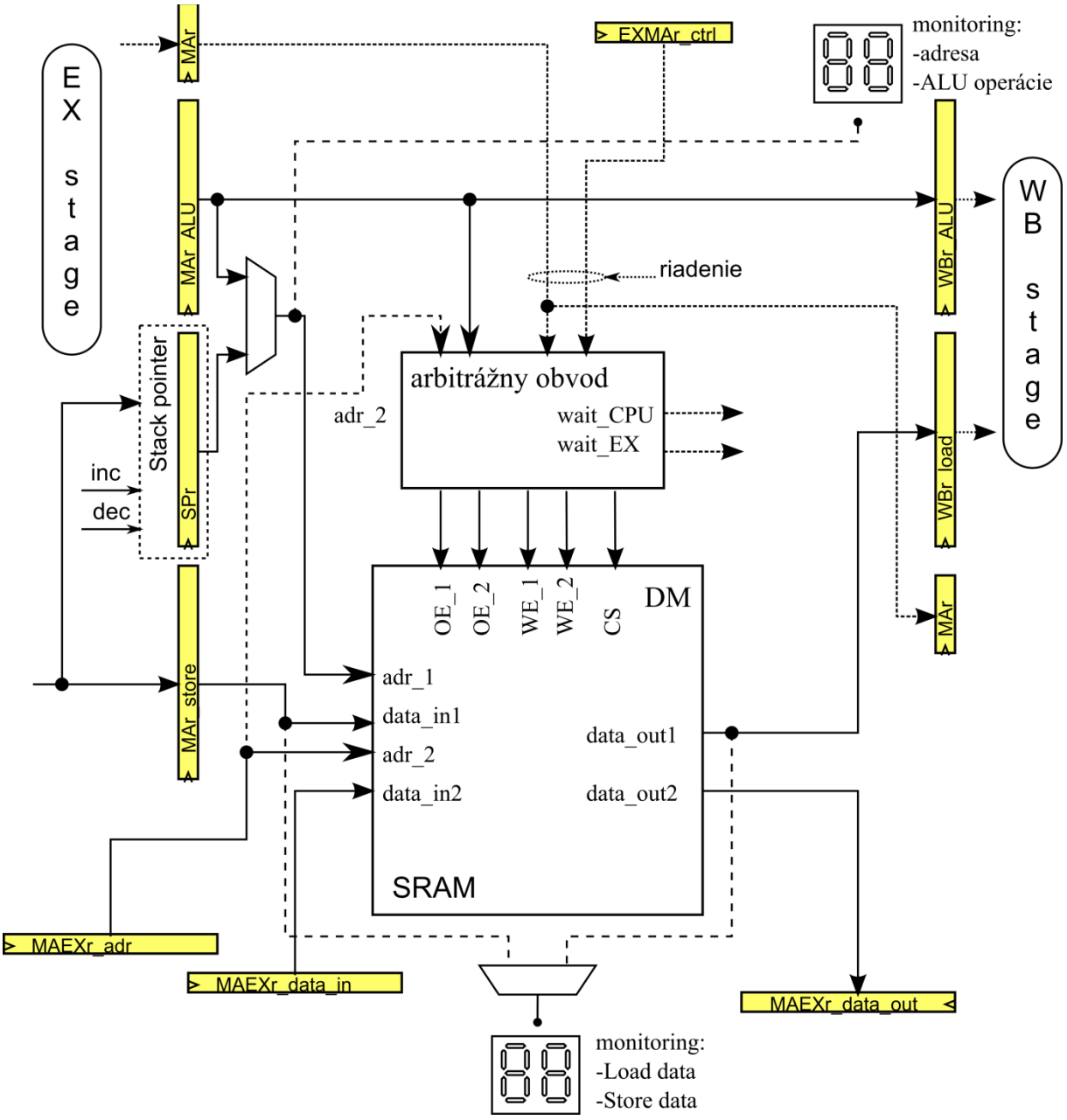
Registre EXr_Rd a Mar_Rd zabezpečujú postupný prechod cieľovej adresy pre zápis do registra (brána RD registrového poľa) stupňom EX zreťazenej linky.



Obr. č. 4.5 *Znázornenie mikroarchitektúry stupňa pre vykonávanie inštrukcie*

4.1.4 Blok pre prístup do pamäti

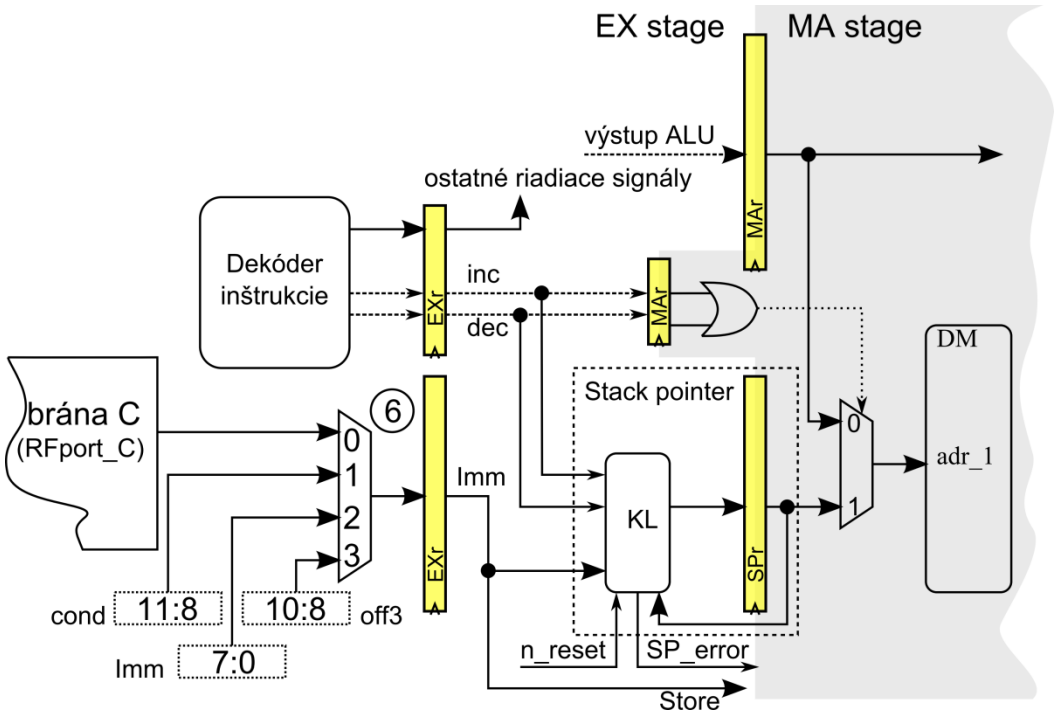
Tento blok bol navrhovaný, aby umožňoval prístup do pamäte nielen prostredníctvom zreťazenej linky, ale aj externým prístupom. To znamená vydať požadované dáta na externú bránu a nezaťažovať pritom procesom zreťazenú linku. Keďže dátová RAM je 4-portová, mikroarchitektúra umožňuje aj proces zápisu. Na obr. č. 4.6 je znázornená štruktúra stupňa a zároveň aj ukazuje, ktoré hodnoty sú monitorované a vizualizované na prípravku.



Obr. č. 4.6 *Znázornenie mikroarchitektúry stupňa pre prístup do pamäti dát*

Vzhľadom k tomu, že pamäť dát je 4-portová, je nutné zaistiť, aby nedochádzalo ku konfliktom pri zápise oboch brán na jedno pamäťové miesto. Túto funkciu plní arbitrážny obvod, ktorý taktiež plní funkciu kontroly koherencie dát. V mikroprocesoroch sa o tieto záležitosti stará správa pamäti. V niektorých prípadoch ide o vysoko sofistikované obvody, hlavne pri implementácii niekoľkoúrovňových cache pamätí a DMA kanálov. Diplomová práca si nekladie za cieľ návrh správy pamätí, preto bol implementovaný jednoduchý

arbitrážny obvod zabraňujúci hlavným konfliktom, ktorý môže byť v budúcnosti modifikovaný.



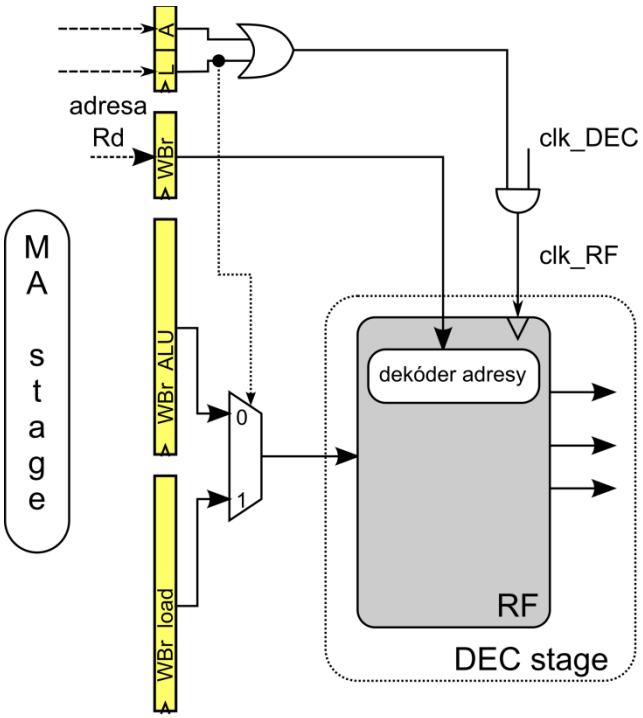
Obr. č. 4.7 *Detail na návrh ukazovateľa zásobníka od dekódovania až po náväznosť na stupeň prístupu do pamäti dát.*

Neodmysliteľnou súčasťou adresovania architektúry „načítaj-ulož“ (Load-Store) je adresovanie pomocou ukazovateľa na zásobník. Aj keď fyzicky je jeho kombinačná logika v stupni pre vykonávanie inštrukcie, ukazovateľ na zásobník ako celok je popisovaný v tejto podkapitole. Taktiež z hľadiska implementácie do VHDL je SP *„portmapovaný“* do bloku pre prístup k pamäti, aby ho nebolo nutné deliť na dve časti (kombinačnú logiku a register). Celá koncepcia jednotlivých entít stupňov zreťazenej linky je koncipovaná tak, že oddeľovacie registre sú iba na úrovni vstupu[14]. Vynechaním oddeľovacieho registra pre signály inkrement (inc) a dekrement (dec), vzhľadom ku kombinačnej logike, sa umožní jednoduché zakomponovanie ukazovateľa na zásobník do mikroarchitektúry. Oddelením týchto signálov pomocou registra a využitím hradla OR je dosiahnuté elegantné riadenie multiplexora, ktorý prepína medzi adresovaním pomocou SP a vypočítanou adresou z ALU jednotky, ako prezentuje obr. č. 4.7 .

[14] Pri zložení zreťazenej linky z jednotlivých blokov tak vznikne značne zložitá sekvenčná entita, ktorej bloky sú oddelené registrami na vstupe aj na výstupe.

4.1.5 Blok pre zápis do registrového poľa

Posledný stupeň zreťazenej linky zabezpečuje spätný zápis do registrového poľa. Štruktúrou a povahou je najjednoduchší. Obsahuje iba oddeľovacie registre a multiplexor pre výber medzi výsledkom z ALU jednotky a hodnotami načítanými z pamäte dát. Mikroarchitektúra je znázornená na obr. č. 4.8 . Znázornené bity *L* (Load- načítanie) a *A (ALU operácie)* nesú informáciu o povahe inštrukcie. Aby v ostatných prípadoch nebolo možné zapisovať do registrového poľa a prepísať tak hodnoty niektorého z registrov, sú hodiny clk_DEC zakázané.



Obr. č. 4.8 *Znázornenie zápisu do registrového poľa*

4.2 Aplikácie metód pre riešenie závislostí

Z komplexného pohľadu na aplikácie pre riešenie závislosti je nutné sa zaoberať nasledujúcimi bodmi:

- návrh zberníc pre bypassing (forwarding),
- návrh metód pre indikáciu dátových závislostí,
- návrh metód pre indikáciu štrukturálnych závislostí,
- riešenie skokových (riadiacich) závislostí a implementácia metód (predikcia skokov, pridanie sčítačky do stupňa DEC a pod.),
- návrh jednotiek pre riadenie zberní typu *„bypassing“*, pozastavovanie zreťazenej linky,
- implementácia prerušení.

Valná väčšina jednoduchých procesorov využíva k detekcii a eliminácii kompilátor. Ten závislosti detekuje a podľa povahy konfliktu poprehadzuje vhodne inštrukcie poprípade vloží prázdnu inštrukciu NOP. Keďže sa jedná o návrh výukového a experimentálneho procesoru a nie je k dispozícii žiadny kompilátor, je vhodné navrhnúť techniky, ktoré umožnia detekovať jednotlivé konflikty pri zreťazenom spracovaní. Táto časť sa taktiež bude venovať aj návrhom pre elimináciu niektorých závislostí. Z hľadiska rozsahu problematiky nie je možné sa venovať všetkým bodom, ktoré boli uvedené v úvode tejto podkapitoly.

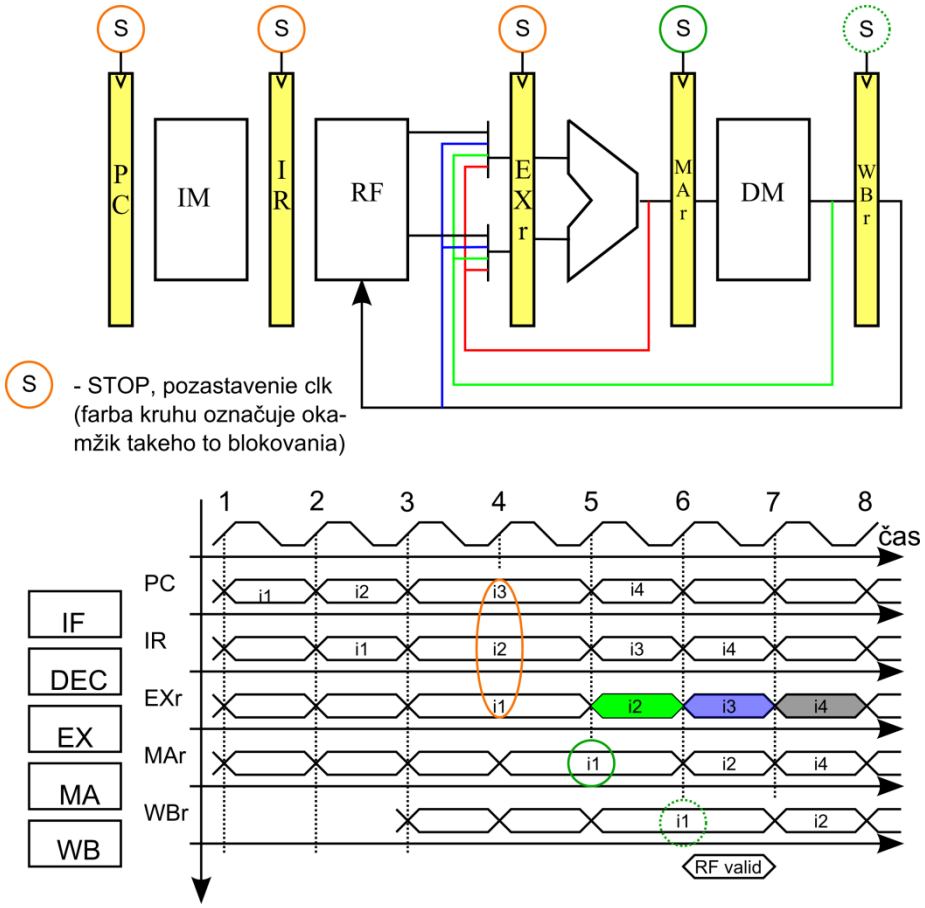
4.2.1 Návrh zbernice pre bypassing

Táto technika sa zameriava na zmiernenie dopadov dátových konfliktov, ktoré vznikajú pri inštrukciách, ktoré na sebe závisia. Obr. č. 4.9 prezentuje dátové skratky (v angličtine sa často technika, ktorá využíva všetkých možných skratiek nazýva *„full bypass*“) a možnosti techniky z hľadiska závislosti medzi inštrukciami prechádzajúcimi stupňami zreťazenej linky. Idealizovaná predstava navrhnutej koncepcie pre detekciu a elimináciu závislosti bude predostretá na príkladoch odkazujúcich sa práve na obr. č. 4.9. Navrhnuté techniky pre elimináciu dátových závislostí sú v réžii hardwaru a radia sa k dynamickým detekčným a eliminačným prostriedkom. Nepotrebujú vkladanie inštrukcií NOP, čo výrazne šetrí miesto v pamäti inštrukcií.

Príklad. č. 1 : použitie –použitie

Predpokladajme, že inštrukcia *„i2“* potrebuje použiť výsledok operácie *„i1“*, jednotka pre správu hazardov deteguje túto závislosť a prepne multiplexor do polohy 1. Ak by tým istým spôsobom bola závislá *„i3“*, využije sa skratka z výstupu pamäte dát (multiplexor do

polohy 2). Obdobným spôsobom pri závislosti inštrukcie *„i4“* na *„i1“* zafunguje skratka zo zápisového portu registrového poľa brány D (multiplexor do polohy 3). Bez zaimplementovania týchto skratiek by bolo možné použiť takúto závislosť až na pozícii *„i5“*, pretože do registrového poľa sa výsledok inštrukcie zapisuje *6-tou aktívnou hranou* a platnosť dát načítaného registru je až pri *7-mej aktívnej hrane* hodinového signálu.



Obr. č. 4.9 *Návrh plného bypassingu s povoľovaním hodín jednotlivých blokov a časovanie pri dátovej závislosti „i1“ a „i2“ spôsobom RAW – načítanie - použitie*

Príklad. č. 2 : načítanie –použitie

V predchádzajúcom príklade sa zakomponovaním skratiek z výstupu blokov (EX, MA, WB) na vstup stupňa pre vykonávanie inštrukcií dosiahlo toho, že nemuselo dochádzať k pozastavovaniu spracovania inštrukcií v zreťazenej linke (či už hardwarovo alebo za použitia inštrukcie NOP). Konflikt, ktorý nastáva pri závislosti *načítanie –použitie* v zreťazenej linke s takouto povahou inštrukčného paralelizmu, nie je možné eliminovať iba skratkami medzi jednotlivými stupňami. Preto je nutné navrhnúť stratégiu pozastavovania niektorých stupňov zreťazenej linky v jednotlivých fázach procesu. Uvažujeme inštrukciu

„*i1“*, ktorá načítava hodnotu z pamäte dát. Inštrukcia *„i2“* idúca po *„i1“* už ale chce danú hodnotu použiť ako jeden z operandov. Do stupňa pre vykonávanie inštrukcií sa *„i2“* nemôže vložiť skôr, ako budú ustálené hodnoty z dátovej RAM na bránach vstupného registra MAr. Táto situácia nastane najskôr na konci 4 cyklu prostredníctvom skratky z výstupu DM (zbernica zelenej farby). Preto je nutné pozastaviť stupne IF, DEC, EX zablokovaním prístupu hodinového signálu oddeľovacím registrom PCr, IRr, EXr. Túto situáciu znázorňuje oranžová elipsa zahŕňajúca *„i3, i2, i1*“. Na konci 4 cyklu je už požadovaná hodnota na vstupe oddeľovacieho registra MAr. Blok pre elimináciu závislostí povolí hodinové signály registrom PCr, IRr, EXr, WBr a zakáže ich oddeľovaciemu registru MAr. Dôvodom je situácia, že počas štvrtého cyklu je inštrukcia *„i1“* obsiahnutá v stupni EX aj MA. Tento proces síce „stojí“ jeden prázdny takt, ale „pokuta“ bez zakomponovania tejto eliminácie by narástla na 3 strojové takty.

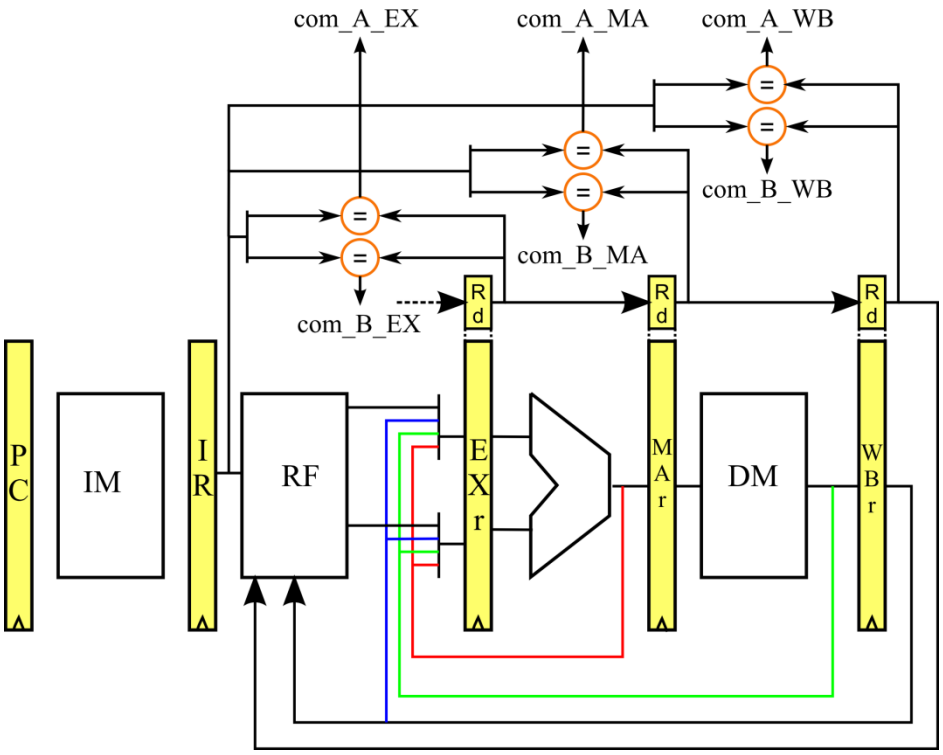
Na zeleno podfarbená „*i2“* na úrovni piateho cyklu naznačuje aktívnosť zbernice z výstupu DM. Podfarbenie inštrukcie „*i3“* na modro naznačuje možný prípad využitia zbernice z výstupu stupňa WB pri závislosti *načítanie -použitie* medzi inštrukciami „*i1“* a *„i3“*. Šedé podfarbenie naznačuje, že pri závislosti medzi inštrukciami „*i1“* a *„i4“* sú hodnoty načítavané už z registrového poľa.

4.2.2 Návrh metód pre indikáciu dátových závislostí

Aby procesy a eliminácie popísané v kapitole pre návrh bypasingu mohli fungovať, je potrebné navrhnúť detekciu závislosti a následne jednotku riadenia. Na detekciu závislosti sa zameriava obr. č. 4.10. Sada šiestich komparátorov porovnáva cieľový register v stupňoch EX, MA a WB s požadovanými operandami v dekódovacom stupni.

Z hľadiska štrukturálnych závislostí, vzhľadom k registrovému poľu, je nutné dodržať podmienku, aby nebolo umožnené zapisovať do registrov, z ktorých je čítané. Ide o porty registrového poľa RA, RB, RC. Pre indikáciu konfliktu na portoch RA a RB je možné použiť už komparátory označené ako *com_B_WB* a *com_A_WB* (doposiaľ využité pre indikáciu dátových konfliktov). Ostáva tak pridať komparačný element, ktorý porovná adresu zápisu do registru (zápisová brána RD registrového poľa) s adresou pre port RC. Po takejto indikácii je nutné zakázať postup inštrukcií v zreťazenej linke a zapísať do statusového registra chybu

prostriedkov. Bez zdvojenia[15] registrového poľa, či použitia rezervačnej stanice a jeho obsluhy, nie je možné sa s týmto konfliktom dynamicky vysporiadať. Taktiež je nutné



Obr. č. 4.10 *Zakomponovanie komparátorov pre indikáciu dátových závislosti*

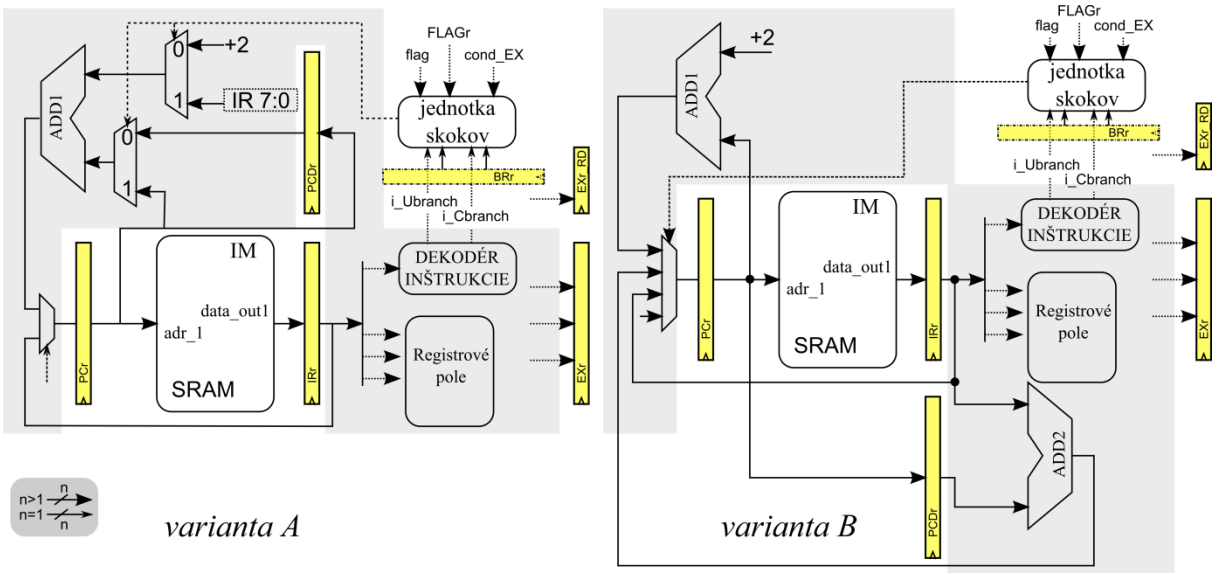
zaoberať sa identifikáciou konfliktu. Môže isť o konflikt prostriedkov, ale taktiež je možná závislosť dát[16].

4.2.3 Implementácia skokov a riešenie riadiacich závislostí

Aby mohol mikroprocesor načítavať inštrukcie, musí dekódovací stupeň obsahovať mikroarchitektúru, ktorá mu umožní vybrať správnu adresu inštrukcie. Práve túto časť dekódovacieho bloku znázorňuje obr. č. 4.11. Okrem základného usporiadania architektúry (sčítačka v stupni EX) boli navrhnuté dve varianty. Zatiaľ čo *varianta A* používa jednu sčítačku, *varianta B* používa dve sčítačky k zmenšeniu latencie výpočtu nasledujúcej adresy

[15] Dobežnou hranou prepis z registrového poľa A do registrového poľa B. Nutné prehodnotiť nárast spotreby (prepisovanie 16-tich registrov každým hodinovým signálom), nárast plochy čipu a implementovať sofistikovanejšie riadenie a kontrolu koherencie dát.
[16] V tomto prípade by sa postupovalo ako bolo navrhnuté v kapitole popisujúcej bypassing.

a tým aj celého dekódovacieho bloku. Tento proces výpočtu adresy je znázornený kruhovým časovým diagramom na obr. č. 4.12.

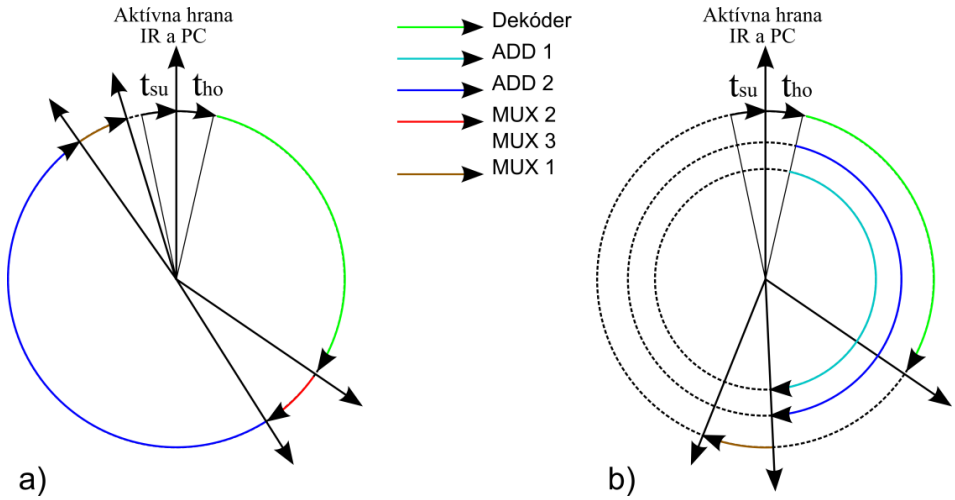


Obr. č. 4.11 *Znázornenie dekódovacieho stupňa s detailom na časť mikroarchitektúry pre vykonávanie skokových operácii: a) za použitia jednej sčítačky v dekódovacom stupni b) za požitia dvoch sčítačiek v dekódovacom stupni. Šedé podfarbenie znázorňuje prvky dekódovacieho stupňa*

Ide o prvok paralelizmu, kde sú spočítané obe možnosti a to nasledujúca adresa a adresa skoku. Následne už iba dekodér inštrukcie rozhodne, ktorá informácia o adrese bude zapísaná do programového čítača. Do VHDL kódu je zapracovaná varianta B. O výpočet nasledujúcej adresy sa stará *jednotka skokov,* ktorá na základe informácií od dekodéra inštrukcie, registra príznakov a podmienky, určí nasledujúcu adresu.

Ako už bolo v teoretickej časti predostreté, zreťazená linka je náchylná pri skokových operáciách na vznik tzv. riadiacích závislostí. Pri nich dochádza k redukcii efektívnosti zreťazeného spracovania. Navrhnutá architektúra má tu vlastnosť, že je známy výsledok adresy, podmienka[17] a samotná informácia o skokovej inštrukcii v jednom takte. Takýto návrh hardwarovo redukuje vzniknuté okienko medzi vypočítanou adresou skoku a načítaním inštrukcie na tejto adrese. Mnou navrhnutá architektúra vykazuje dĺžku okienka jedného až dvoch taktov. Záleží na politike implementovanej v *jednotke skokov.* Riadiace (skokové) závislosti sú riešené pri statickom kóde kompilátorom.

[17] Podmienka (ak ide o podmienený skok) je nesená samotnou skokovou inštrukciou.



Obr. č. 4.12 *Časové diagramy jednotlivých modelov: a) za použitia jednej sčítačky v dekódovacom stupni b) za požitia dvoch sčítačiek v dekódovacom stupni*

Prístupov, ako riešiť takto vniknuté narušenie zreťazeného spracovania, je viacero. Od základných, vložením inštrukcie NOP, až po sofistikovanejšie techniky vkladania inštrukcií, ktoré sú pri sekvenčnom spracovaní pred skokovou inštrukciou. Vkladanie musí byť bezpečné, to znamená, že vloženie takejto inštrukcie nesmie spôsobiť žiadne konflikty. Pri dlhších[18] okienkach sa môže stať, že kompilátor nenájde dostatočné množstvo bezpečných inštrukcii na vloženie. Jedným z možných riešení je vkladať podmienku až za skokovú inštrukciu a predpokladať, že sa skok prevedie vždy. Tým je docielené načítanie inštrukcie na adrese skoku ešte pred vyhodnotením podmienky. Pri správnej predikcii je pokuta 0 taktov, ale pri nesprávnej narastie na 2 takty. Riešenie skokových závislostí, vedúcich k čo najefektívnejšiemu využitiu okienka, je jednou hlavných priorít, pretože v aplikačných programoch je v priemere každá piata inštrukcia skoková.

Bola navrhnutá istá hardwarová koncepcia a predostreté možnosti pri prístupe ku kontrolným závislostiam. Pri kompletovaní architektúry, ladení a odstraňovaní drobných nedostatkov môžu byť vytvorené viaceré verzie s odlišnou skokovou politikou pri rovnakej architektúre. Je tak možné laborovať s technikami, ktoré redukujú nevyužité strojové cykly a demonštrovať na príkladoch ich výhody a nevýhody. Ďalej by bolo možné doplniť architektúru o jednoduchý prediktor skokov a demonštrovať tak na jednoduchých príkladoch možnosť dynamickej redukcie dôsledkov riadiacich závislosti, ktoré sú dnes bežnou súčasťou pokročilejších architektúr.

[18] Dlhšie okienko pri jednoduchších architektúrach je následok nezaradenia sčítačky do dekódovacieho stupňa. Z hľadiska pokročilejších a sofistikovanejších architektúr sú takéto okienka bežnou cenou, ktorou sa platí za zvýšenie inštrukčného paralelizmu.

5 Implementácia do VHDL, prípravok a jeho obsluha

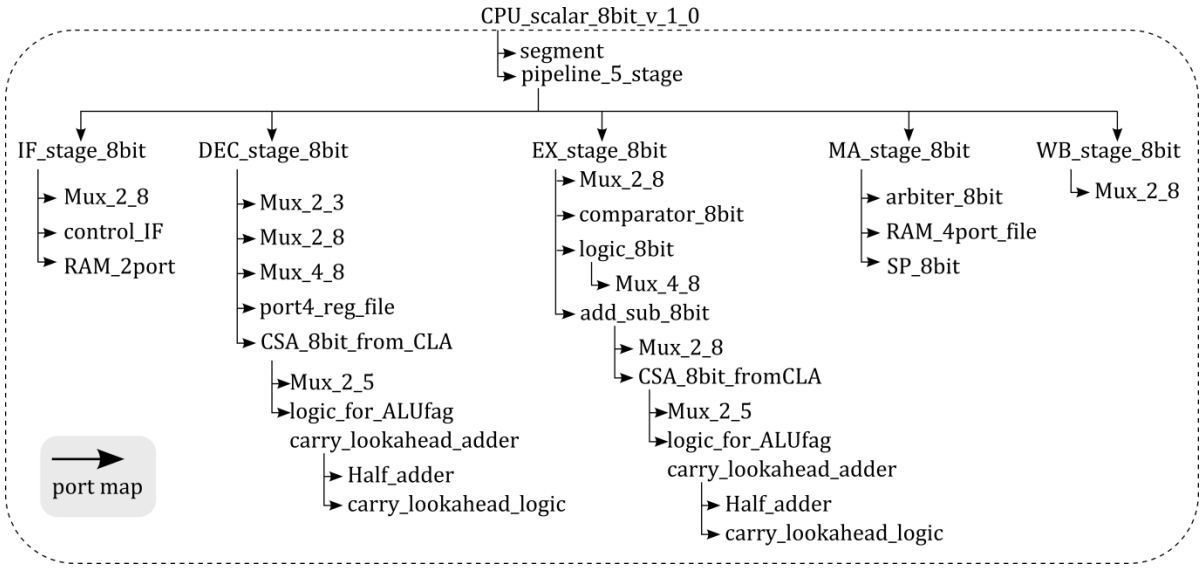
Prvotná koncepcia diplomovej práce počítala s návrhom jednoduchého procesora pre výukové účely s využitím diskrétnych súčiastok a s vizualizáciou pomocou LED diód. To prináša iste koncepčné nevýhody a limity ako nemodifikovateľnosť, náchylnosť k poruchám, finančne náročná výroba DPS vzhľadom k funkcii obvodu. Tieto nevýhody viedli k implementácii do FPGA s vizualizáciou na už hotovej vývojovej doske.

5.1 Implementácia architektúry do jazyka VHDL

Keďže vonkajšie vizualizačné schopnosti prostredníctvom vývojových dosiek sú značne limitované, bolo hlavnou mojou víziou umožniť, aby na architektúre a mikroarchitektúre bolo možne ukázať iste dané koncepčné a značne detailné vlastnosti a chovania jednotlivých elementov. Preto bolo vytvorených mnoho dielčích entít od multiplexorov cez arbitrážne obvody, bloky pamäti a registrového poľa až po jednotlivé stupne zreťazenej linky, ktoré sú navzájom do seba „portmapované“. Pri skompilovaní potrebných súborov je následne možné vo vývojovom prostredí Quartus II od ALTERY pomocou aplikácie *RTL Viewer* „rozklikávať“ jednotlivé elementy mikroprocesora a sledovať tak jednotlivé abstrakčné úrovne jeho štruktúry.

Pomocou vývojového prostredia ModelSIM je možné sledovať priebehy signálov na rôznych úrovniach abstrakcie od adries a hodnôt pre pamäte (ktoré budú môcť byť sledované aj na prípravku) až po detailné signály polovičných sčítačiek a nastavovania príznakových bitov. Ďalšou výhodou je to, že k takémuto štúdiu a experimentovaniu (je možne architektúru upravovať, dolaďovať a rozširovať) nie je potrebný žiadny hardware a študenti sa tomu môžu venovať aj z pohodlia domova.

Na obr. č. 5.1 je znázornená štruktúra entít a ďalej to, ako jednotlivé entity na sebe závisia pri *„portmapovaní“*. Obrázok taktiež prezentuje schopnosť jednoducho identifikovať daný prvok v architektúre (RTL Viewer) a pozorovať aj malé elementy, napr. polovičné sčítačky.



Obr. č. 5.1 *Závislosti entít*

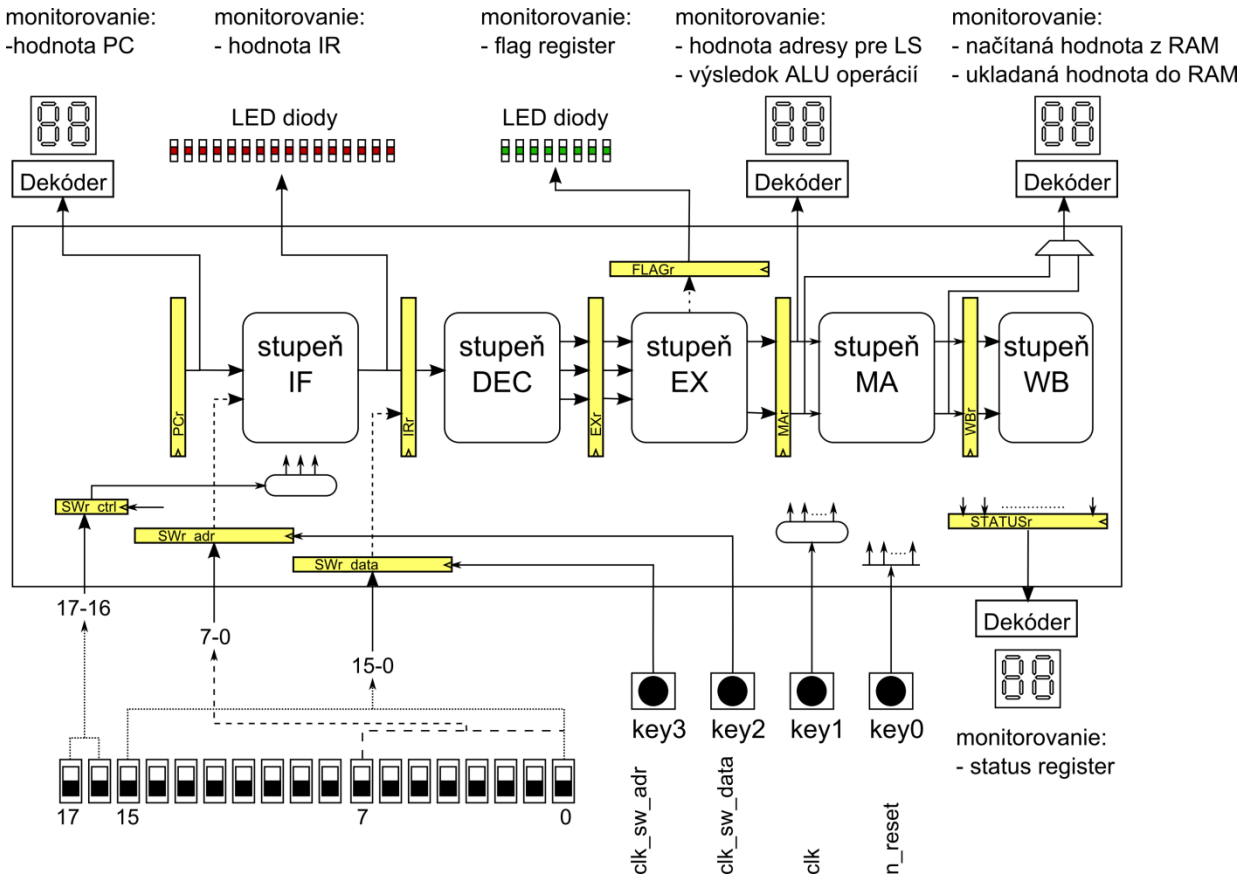
5.2 Implementácia na vývojovú dosku DE2-115

Ako hardwarový vizualizačný prostriedok bola vybraná vývojová doska od firmy TerasIC DE2- 115, ktorá je dostupná na Katedre aplikovanej elektroniky a telekomunikácií vo väčšom množstve a študenti s ňou prichádzajú priamo do kontaktu na predmete PLO a ACZS. Jej štruktúre bol prispôsobený návrh vizualizačných častí a top entita. Na obr. č. 5.2 je znázornené zapracovanie ovládacích a vizualizačných častí do zreťazenej linky. Spomínaná vývojová doska obsahuje 18 prepínačov, preto je nutné niektoré prepínače koncipovať ako multifunkčné. Pre indikáciu sú použité 7-segmentové displeje a LED diódy. Vizualizačné a ovládacie prvky sú popísané v tab. č. 5.1 až tab. č. 5.4.

Tab. č. 5.1 *Funkčné vlastnosti prepínačov v závislosti na tlačidle key3 a key2*

Prepínač	*Aktívne tlačidlo key3*	*Aktívne tlačidlo key2*
0	adresa, bit [0]	dáta, bit [0]
1	adresa, bit [1]	dáta, bit [1]
2	adresa, bit [2]	dáta, bit [2]
3	adresa, bit [3]	dáta, bit [3]
4	adresa, bit [4]	dáta, bit [4]
5	adresa, bit [5]	dáta, bit [5]
6	adresa, bit [6]	dáta, bit [6]
7	adresa, bit [7]	dáta, bit [7]
8	-	dáta, bit [8]

9	-	dáta, bit [9]
10	-	dáta, bit [10]
11	-	dáta, bit [11]
12	-	dáta, bit [12]
13	-	dáta, bit [13]
14	-	dáta, bit [14]
15	-	dáta, bit [15]
16	riadenie, bit [0]	riadenie, bit [0]
17	riadenie, bit [1]	riadenie, bit [1]



Obr. č. 5.2 *Zapracovanie ovládacích a vizualizačných častí do zreťazenej linky.*

Tab. č. 5.2 *Funkčné vlastnosti tlačidiel key3 a key2*

Tlačidlo	***Funkcia***
key0	reset (aktívny pri logickej 0)
key1	signál, ktorý emuluje taktovacie hodiny a umožní tak krokovanie zreťazenej linky
key2	nábežná hrana pre zápis do registru SWr_data
key3	nábežná hrana pre zápis do registru SWr_adr

Tab. č. 5.3 *Funkčné hodnoty indikovaných LED diod*

LED	***Zelená***	***Červená***
0	príznak prenosu	hodnota bitu IR [0]
1	príznak vynulovania	hodnota bitu IR [1]
2	príznak znamienka	hodnota bitu IR [2]
3	príznak pretečenia	hodnota bitu IR [3]
4	-	hodnota bitu IR [4]
5	-	hodnota bitu IR [5]
6	-	hodnota bitu IR [6]
7	-	hodnota bitu IR [7]
8	-	hodnota bitu IR [8]
9	-	hodnota bitu IR [9]
10	-	hodnota bitu IR [10]
11	-	hodnota bitu IR [11]
12	-	hodnota bitu IR [12]
13	-	hodnota bitu IR [13]
14	-	hodnota bitu IR [14]
15	-	hodnota bitu IR [15]

Tab. č. 5.4 *Indikačné vlastnosti použitých 7-segmentových displejov*

7-segmentovka	***Funkcia***
hex1	hodnota obsahu programového čítača
hex2	hodnota výsledku ALU operácie a adresa LS, prípadne obsah ukazovateľa zásobníku
hex3	načítaná a ukladaná hodnota do RAM
hex4	indikácia statusového registra zreťazenej linky

Záver

Koncepcia práce bola zameraná na návrh jadra procesora typu RISC pre výukové účely. Úvodom by bolo vhodné zmieniť, že koncepcia realizácie modelu sa značne vychýlila od koncepcie, s ktorou sa počítalo pri zadávaní diplomovej práce. Pôvodne bolo počítané s návrhom jednoduchého modelu procesora realizovaného pomocou diskretných súčiastok. Z dôvodu realizačných problémov, akými sú vysoké náklady, nemodifikovateľnosť a náchylnosť k poruchám, bol zvolený model implementácie do obvodu FPGA v jazyku VHDL. Významnými vlastnosťami modelu, ktoré hovoria v prospech implementácie architektúry do obvodu FPGA, je vizualizácia prostredníctvom aplikácie *RTL Viewer* v programe *Quartus II,* kde je možné sledovať aj malé funkčné elementy, z ktorých je poskladané celé jadro procesora. Simuláciu modelu je taktiež výhodné realizovať pomocou programu *Modelsim*, kde prostredníctvom tzv. *testbenchov* sú napísane časti programu a dá sa tak pri krokovaní využiť sledovanie akéhokoľvek signálu. To poskytne široké spektrum ukážkových a modelových simulácii vhodných pre výuku, ktorému z pohľadu detailnosti nemôže konkurovať žiadna hardwarová vizualizácia signálov. V neposlednom rade sa táto implementácia vyznačuje vysokou flexibilitou pri modifikovaní a experimentovaní.

Pri tvorbe teoretickej časti sa vychádzalo predovšetkým zo zahraničných knižných publikácií od autorov, ktorý boli pri vzniku realizácií myšlienok zreťazeného spracovania inštrukcií, ako D. A. Patterson a J. L. Hennessy. Bola vytvorená analýza častých typov zreťazených liniek s nízkou hĺbkou na základe vyhodnotených informácií z mnohých odborných periodík.

Jednou z kľúčových častí praktickej realizácie bol návrh inštrukčnej sady, ktorý bol založený na inštrukčnej sade THUMB. Táto inštrukčná sada je istým kompromisom medzi inštrukčným sadami RISC-ového a CISC-ového typu. Štruktúra síce zachováva rovnakú šírku inštrukcie, ale niektoré segmenty, ako registrový kľuč a operačný kód, majú premennú dĺžku, čo sú vlastnosti bližšie CISC-ovej inštrukčnej sade. Na druhú stranu ortogonalita inštrukčnej sady bola zachovaná využitím architektúry *NAČÍTAJ-ULOŽ*. Celkovo bolo vytvorených 15 formátov, čo tvorí v súčasnom stave 48 inštrukcií, z ktorých sú niektoré rezervované pre následne doplnenie a modifikáciu.

V kapitole 4 bol predložený vlastný návrh mikroarchitektúry, ktorá aplikuje zreťazené vykonávanie inštrukcií navrhnutej inštrukčnej sady. Každému stupňu je venovaná vlastná podkapitola s popisom a obrázkami, ktoré prezentujú jeho mikroarchitektúru. V tejto kapitole je tiež pomerne obšírna časť vlastných návrhov pre identifikáciu a elimináciu konfliktov.

K praktickej realizácii architektúry prostredníctvom jazyka VHDL sa pristupovalo na úrovni abstrakcie presunov medzi registrami (Register Transfer Level = RTL). Veľká časť entít je ale písaná vo forme menších blokov, ktoré majú povahu kombinačnej logiky. Tie sú následne portmapované do väčších blokov, ktoré sú začlenené do zreťazenej linky. Táto metóda je síce veľmi prácna, časovo náročná a pseudohradlový popis nemusí viesť k optimálnym výsledkom po syntéze, ale veľmi názorne zobrazuje bloky v rôznych fázach abstrakcie architektúry, čo bolo hlavným cieľom tejto myšlienky. Vzhľadom k časovej a rozsahovej náročnosti problematiky a odchýlenia sa od pôvodnej koncepcie, nebolo možné navrhnutú architektúru zreťazenej linky implementovať v plnom rozsahu a podrobiť tak verifikácii. V súčasnom stave je väčšina blokov navrhnutej architektúry implementovaných do jazyka VHDL a umožňuje prehľadné „rozklikávanie“ architektúry, ale pre finálnu verifikáciu a validáciu sa bude musieť zakomponovať a funkčne doplniť ešte pár blokov. Táto implementácia architektúry bola navrhnutá tak, aby ju bolo možné postupne rozvíjať, ladiť a obohacovať až do finálnej podoby.

Poslednou fázou praktickej realizácie bol návrh hardwarového prípravku pre možnosti simulácií krátkych programov, na ktorých bude možné vysvetľovať isté špecifické vlastnosti architektúry RISC. Navrhovaný vizualizačný hardwarový prípravok je realizovaný na vývojovej doske DE2-115 od firmy Terasic.

V rámci diplomovej práce bolo vytvorených 40 názorných a precíznych obrázkov vo vektorovej grafike, ktoré celú prácu doprevádzajú a poukazujú na princípy, konflikty zreťazeného spracovania. V kapitole 5, ktorá sa venuje návrhu linky pre zreťazené spracovanie, sú obrázky konštruované tak, aby poukázali na jeden dva detaily istej časti zreťazenej linky. Tie v budúcnosti môžu slúžiť ako podklad niektorých prednášok.

Použitá literatúra

[1] David A. Patterson, John L. Hennesyy, *Computer Organization and Desing*, Morgan Kaufman, San Francisco, 2005

[] David M. Harris, Sarah L. Harris : *Digital Desing and Computer Architecture*, Morgan Kaufman

[2] DVOŘÁK, V.: *Architektura procesorů*. Vysoké učení technické v Brně, Brno 1999

[3] Vahid Frank : *Digital deasing*, John Wiley & Sohn, Danvers 2007

[4] YAP, Zi He: *Building A RISC microcontroller in an FPGA*. Faculty of Electrical Engineering, Universiti Teknologi Malaysia, Johor Bahru 2002

[5] Pinker, J. a Poupa, M., *Číslicové systémy a jazyk VHDL*, BEN – technická literatúra, Praha 2006

[6] Steve Furber : *ARM system-on-chip architektur* (2nd Edition),

[7] Hubert Kaeslin: *Digital Integrated Circuit Desing, From VLSI Architecture to CMOS Fabrication*, ETH Zurich, Cambridge University Press 2008

[8] ARM7TDMI Data Sheet ARM DDI 0029E - THUMB Instruction Set

[9] Pinker, J.: *Mikroprocesory a mikropočítače*, BEN – technická literatúra, Praha 2004

[10] Qin Zhao, Bart Mesman, Twan Basten, *Practical Instruction Set Design and Compiler Retargetability Using Static Resource Models*, Eindhoven, IEEE Computer socienty Press, 2002

[11] Asghar Bashteen, Ivy Lui, Jill Mullan, *A superpipeline Approach to the MIPS Architecture*, CA

[12] W. P. Hays, S. Katzman, Ch. Hauck, *7-tage Lexra's New High-Preformace ASIC Processor Pipeline*, Whitepaper Lexra Inc.

[13] Kenneth C. Yeager, *The Mips R10000 Superscalar Microprocessor*, IEEE Micro, 1996

[14] S. Mirapuri, M. Woodacre, N. Vasseghi, *The Mips R4000 Processor*, IEEE Micro, 1992

[14] Ching-Long Su, Alvin M. Despain, *Branch With Masked Squashing in Superpipelined Processors*, Advanced Computer Architecture Laboratory, 21th International Symposium on Computer Architecture, April, 1994

[15] Krste Asanovic, Klaus E. Schauser, David A. Patterson, *Evaluation of a "Stall" Cache: An Efficient Restricted On-chip Instruction Cache*, Computer Science Division, EECS Department University of California, Berkeley

Zoznam príloh umiestnených na CD

Príloha č. 1. : Diplomová práca formáte pdf.

Príloha č. 2. : Vytvorené obrázky v pdf a svg.

Príloha č. 3. : VHDL súbory

Evidenční list

Súhlasím s tým, aby bola moja bakalárska práca požičiavaná k prezenčnému štúdiu v Univerzitnej knižnici ZČU v Plzni.

Dátum: Podpis:

Užívateľ potvrdzuje svojím čitateľným podpisom, že túto bakalársku prácu použil k študijným účelom a prehlasuje, že ju uviedol medzi použité pramene.

Meno	**Fakulta/katedra**	**Dátum**	**Podpis**