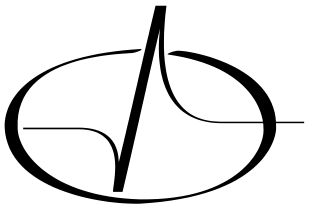
Vyšší odborná škola a Střední průmyslová škola elektrotechnická Olomouc



STŘEDOŠKOLSKÁ ODBORNÁ ČINNOST

Obor 18. Informatika

Lojza - návrh a realizace μprocesoru

DESIGN AND REALIZATION OF MICROPROCESSOR

Autor: Jan Vykydal

7. května 2014

Prohlášení

Prohlašuji, že jsem svou práci SOČ vypracoval samostatně a použil jsem pouze podklady (literaturu, projekty, SW atd.) uvedené v seznamu vloženém v práci SOČ.

Prohlašuji, že tištěná verze a elektronická verze soutěžní práce SOČ jsou shodné. Nemám závažný důvod proti zpřístupňování této práce v souladu se zákonem č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon) v platném znění.

V Olomouci dne 7. května 2014 podpis: .....

Poděkování

Chtěl bych poděkovat jedné moc milé dívce za vymyšlení jména projektu. Dále bych chtěl poděkovat naší škole a hlavně Ing. Zuzaně Veselé za zapůjčení hradlového pole. Také bych chtěl poděkovat panu Ing. Václavu Křížanovi za objasnění některých nejasností z oblasti Booleovy algebry. Velký dík patří také panu Ing. Marku Nožkovi za skvělé výkony, které podává při výuce elektronických počítačů.

ANOTACE

Tato práce popisuje μprocesor a jeho architekturu, jsou zde rozebírány jednotlivé bloky μprocesoru a popisována jejich funkce. Cílem tohoto projektu bylo rozšíření autorových znalostí z oblasti číslicové a μprocesorové techniky, získání nových zkušeností s návrhem rozsáhlých číslicových obvodů s využitím VHDL. Výsledkem tohoto projektu je μprocesor pojmenovaný Lojza. Tento μprocesor má osmi bitovou architekturu nazvanou WPU8. Instrukce procesoru vycházejí z RISC architektury. Koncepce μprocesoru vychází z návrhu Von Neumanna.

Klíčová slova: μprocesor, paměť, RAM, zásobník, aritmeticko-logická jednotka, registr, řadič, VHDL, hradlo, logický obvod, logická funkce, dekodér, kombinační obvod, sekvenční obvod, klopný obvod, sběrnice, architektura, RISC.

ANNOTATION

This work specifies the microprocessor and its architecture, analyses all microprocessor units and describes their functions. The purpose of this project is the extension of the author's knowledge of digital and microprocessor electronics, to gain a new experience in design of digital circuits with the use of VHDL. The result of this project is the microproscessor called Lojza. The architecture of the processor is an 8-bit architecture called WPU8 wich is derived from Von Neumann's design. The instruction set of the processor was developed according to RISC design strategy.

Key words: microprocessor, memory, RAM, stack, ALU, register, control unit, VHDL, logic gate, logic circuit, logic function, decoder, boolean circuit, sequential logic, flip-flop, bus, RISC.

Obsah

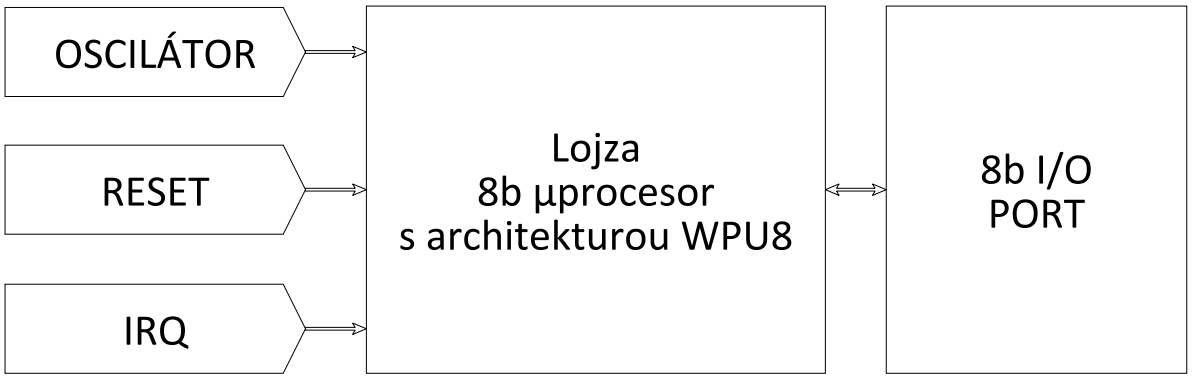
1 Úvod

Už jako malého mě fascinovaly počítače, dlouhou dobu pro mě bylo záhadou, jak tyto černé skříňky pracují, tato zvídavost ve mně zůstala. Postupem času jsem nashromáždil poměrně dost informací o procesorech a výpočetní technice, díky čemuž mohl začít vznikat tento projekt.

V roce 2013 jsem propadl číslicové technice. Díky této zálibě začal vznikat můj závěrečný projekt druhého ročníku, první verze mé aritmeticko-logické jednotky, realizované na kontaktním poli, pomocí integrovaných obvodů 74xx. Postupem času se moje návrhy logických obvodů stávaly komplikovanější a mnohem rozsáhlejší, až jsem se dostal do situace, kdy realizace na kontaktních polích, byly velice obtížné a nepraktické. Proto jsem se začal poohlížet po nové technologii. Zalíbila se mi hradlová pole, a tak jsem se začal učit s nimi pracovat. Po pár týdnech jsem byl schopný své staré konstrukce přepsat do VHDL. Ukázalo se, že je to správná cesta, protože obvody ve VHDL vypadaly docela přehledně a šly snadno upravovat. Když jsem měl ve VHDL kompletně udělanou mou starou aritmeticko-logickou jednotku, tak jsem ji začal rozšiřovat o nové funkce, jako je násobení a dělení. Když byla aritmeticko-logická jednotka kompletní, tak jsem si říkal, že je škoda, že ji musím obsluhovat ručně a nedá se řídit programem.

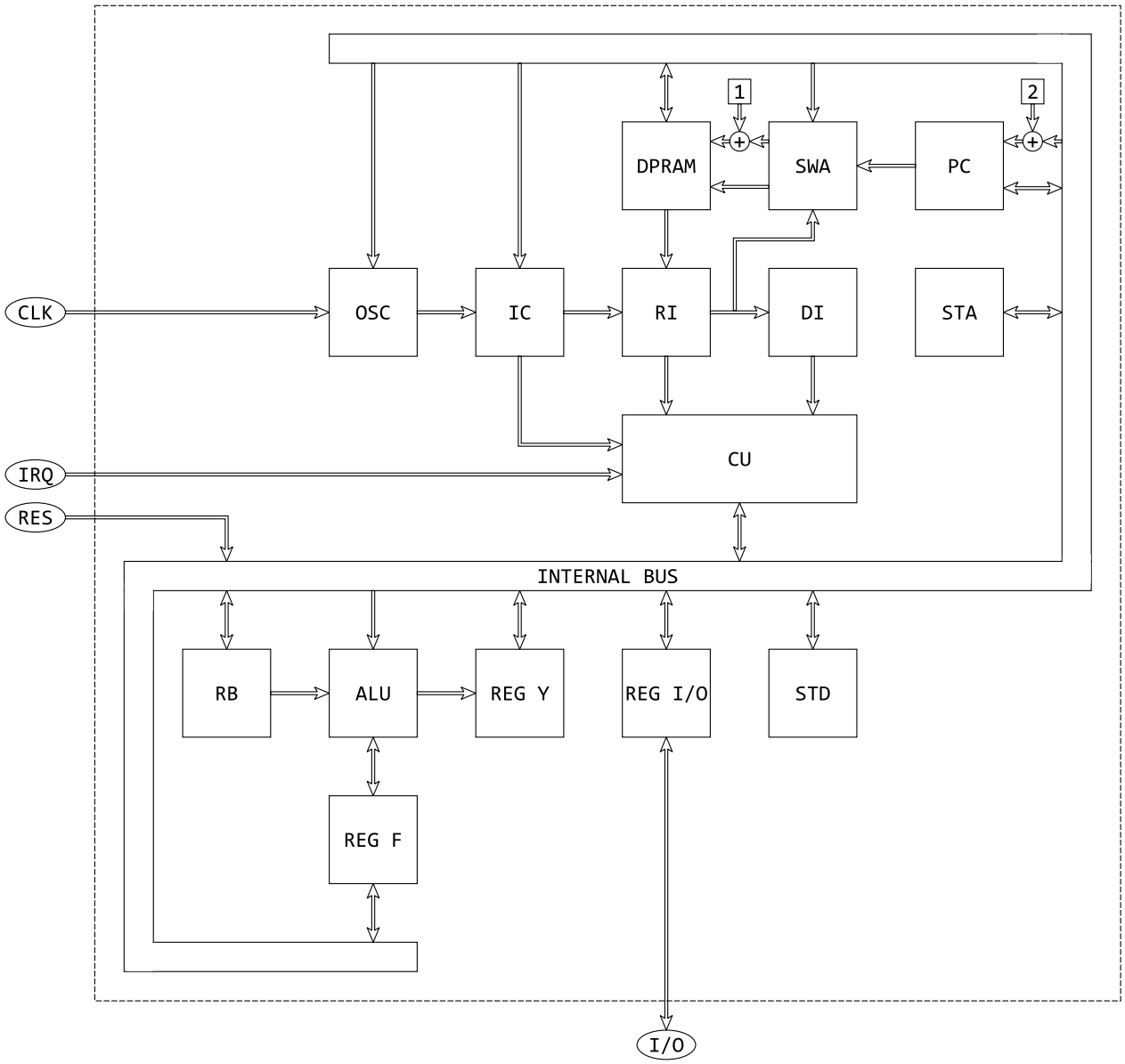
1.1 Cíle projektu

Výše uvedené důvody se staly motivací, pro vnik projektu Lojza, který si kladl za cíl vytvořit kompletní osmi bitový μprocesor, který by byl schopný vykonávat program uložený v paměti RAM, která by byla implementována stejně jako procesor na hradlovém poli. Netrvalo dlouho a vznikla architektura nazvaná WPU8 (název je odvozen z autorovi přezdívky a osmička označuje velikost zpracovávaného slova).



Obrázek 1: Blokové schéma μprocesoru propojeného s okolím

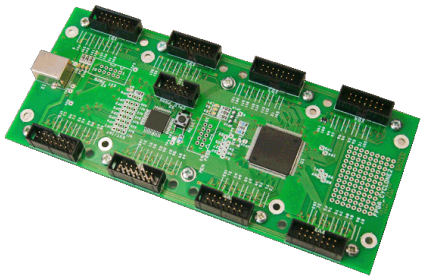
2 Koncept



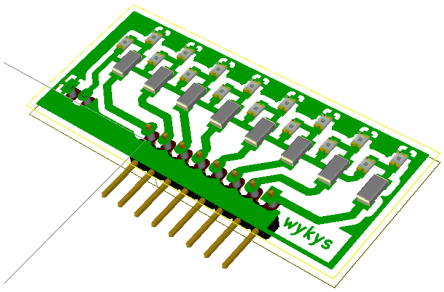
Obrázek 2: Blokové schéma μprocesoru

Blokové schéma se skládá z patnácti hlavních částí: řadič, aritmeticko-logická jednotka, paměť SSDPRAM, registrová banka, registr instrukcí, dekodér instrukcí, adresový přepínač, programový čítač, zásobník pro ukládání adres, zásobník pro ukládání dat, registr příznaků, registr výsledků, vstupně výstupní registr, instrukční čítač, hodinový korektor. Funkce jednotlivých bloků budou rozebírány v dalších sekcích.

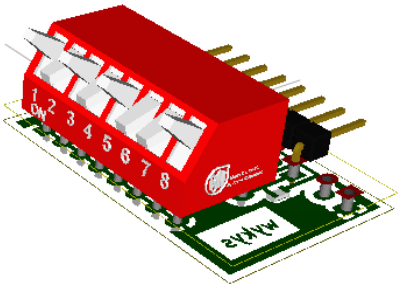
3 Vývojový kit



Obrázek 3: Vývojový kit UNICELL



Obrázek 4: LED indikátor výstupů



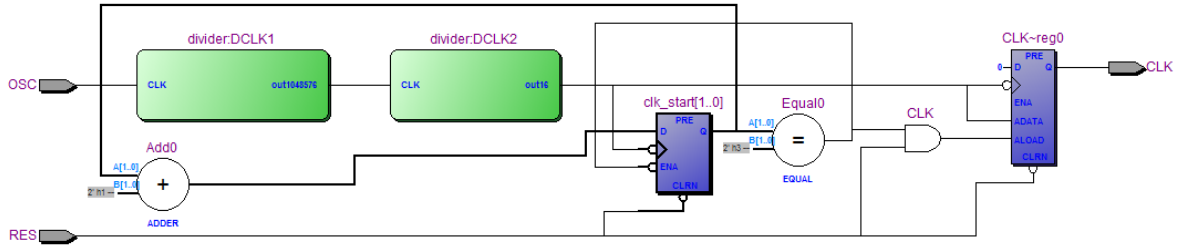
Obrázek 5: dip spínač pro testování vstupů

Procesor byl vyvíjen a testován na vývojovém kitu UNICELL od společnosti UNITES Systems a.s. Kit je osazen hradlovým polem Cyclon II, jedná se o typ EP2C5T144C8N, který je taktován na frekvenci 24 MHz. Na kitu je integrovaný programátor USB Blaster. K tomuto kitu jsem si vytvořil vlastní testovací moduly, které mi usnadnily a urychlily vývoj jednotlivých částí μprocesoru. Díky tomu, že vývoj probíhá na hradlovém poli, tak si mohu na vývody hradlového pole přivést libovolný vstup, či výstup z kteréhokoliv bloku μprocesoru, což mi umožňuje téměř okamžitě vidět, zda-li se výsledný obvod chová správně.

4 Synchronizační obvody

4.1 Hodinový korektor

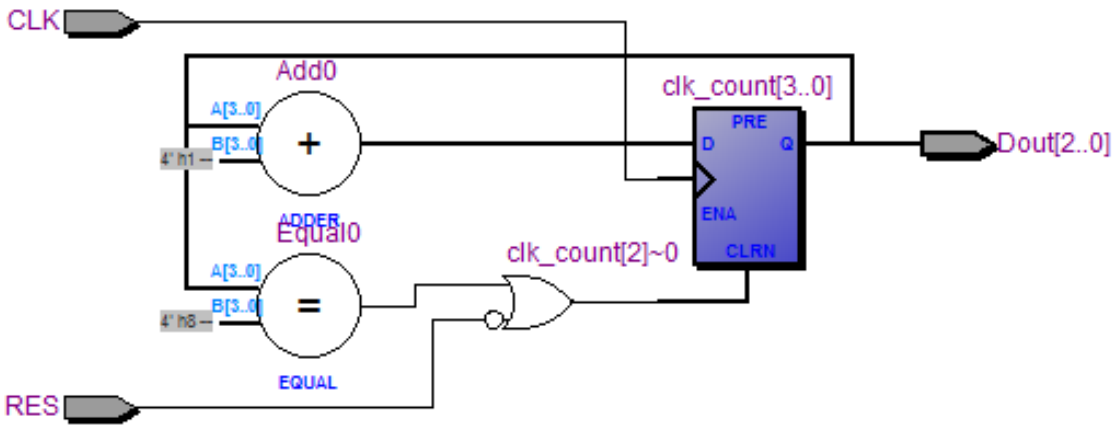
Tento obvod řídí hodinový signál μprocesoru, dále jen CLK. Při přivedení napájení k μprocesoru obvod počká dokud nezachytí tři sestupné hrany z oscilátoru a důvodu aby procesor nezačal pracovat v už probíhající nástupné hraně, což by vedlo k nedefinovaným stavům a poté do do obvodu začne dodávat synchronizační pulzy. Pokud je na signál RES přivedena logická úroveň L, tak jsou hodinové pulzy zastaveny a po přivedení logické hodnoty H na signál RES dochází opět k inicializaci obvodu. Obvod umožňuje frekvenci výstupního hodinového signálu CLK snižovat, pomocí vestavěných předděliček.



Obrázek 6: Schéma hodinového korektoru

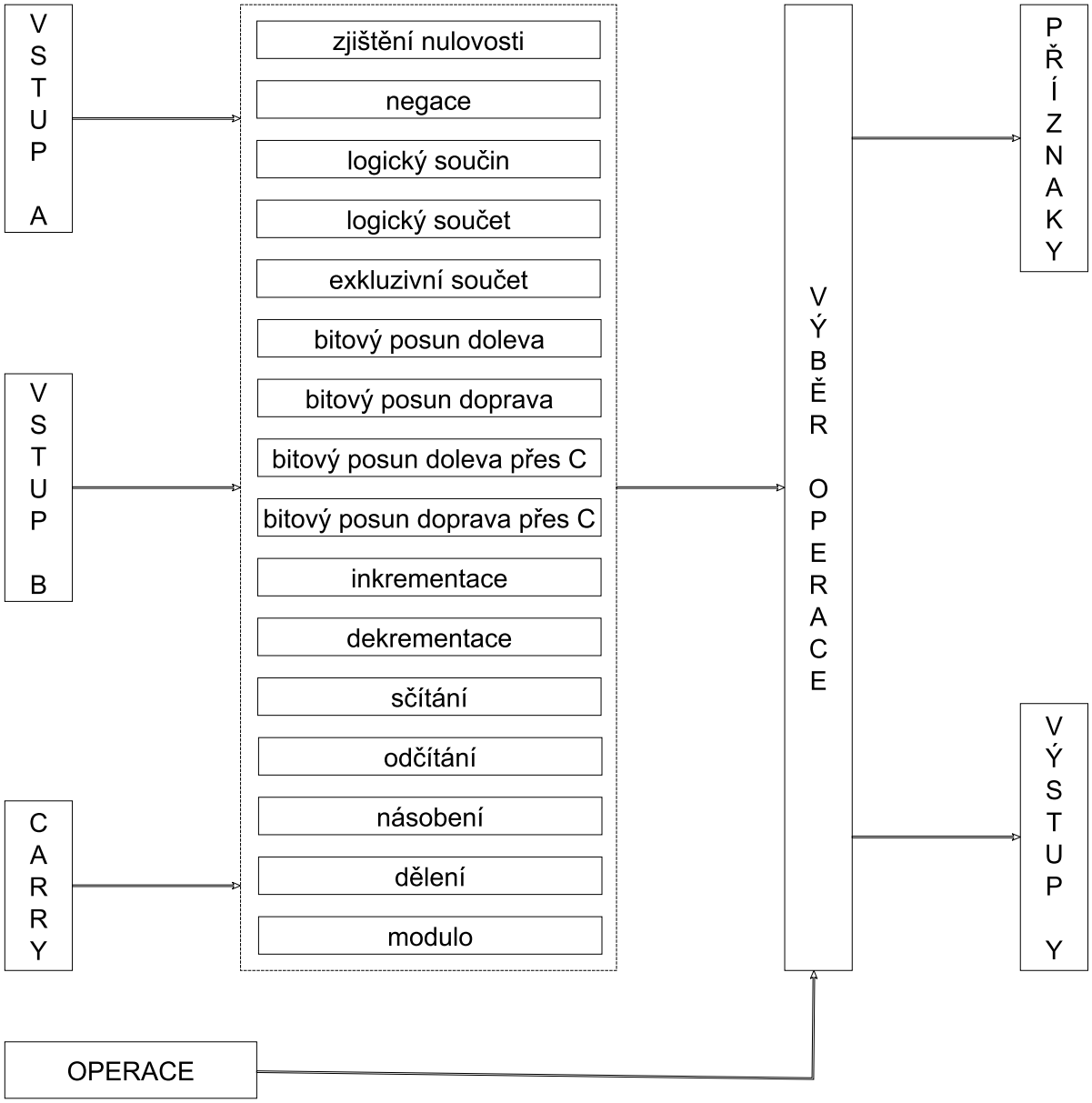
4.2 Instrukční čítač

Jedná se o tříbitový čítač, který sloučí pro určování stavu řadiče a registru instrukcí. Tento čítač má asynchronní nulování v logické úrovni L.



Obrázek 7: Schéma instrukčního čítače

5 ALU



Obrázek 8: Blokové schéma ALU

5.1 Popis bloku

ALU (Arithmetic Logic Unit) český název aritmeticko-logická jednotka, je část procesoru, která je určena k vykonávání aritmetických a logických operací s celými čísly. ALU je navržena modulárně (jednotlivé funkce jsou realizovány pomocí samostatných modulů), díky čemuž se snadno dají funkce jednotky modifikovat. Většina modů ALU je navržena tak univerzálně že nacházejí uplatnění v dalších částech procesoru. Aritmeriko-logická jednotka je navržena pro výpočty s 8 b čísly, tudíž je maximální hodnota výsledku $2^8 = 255$. Kromě výsledku poskytuje ALU další informace v podobě příznaků. C - Carry je nastaven na úroveň H při operacích INC, DEC, ADD, SUB tehdy, pokud dojde přenosu z nejvyššího řádu, to je

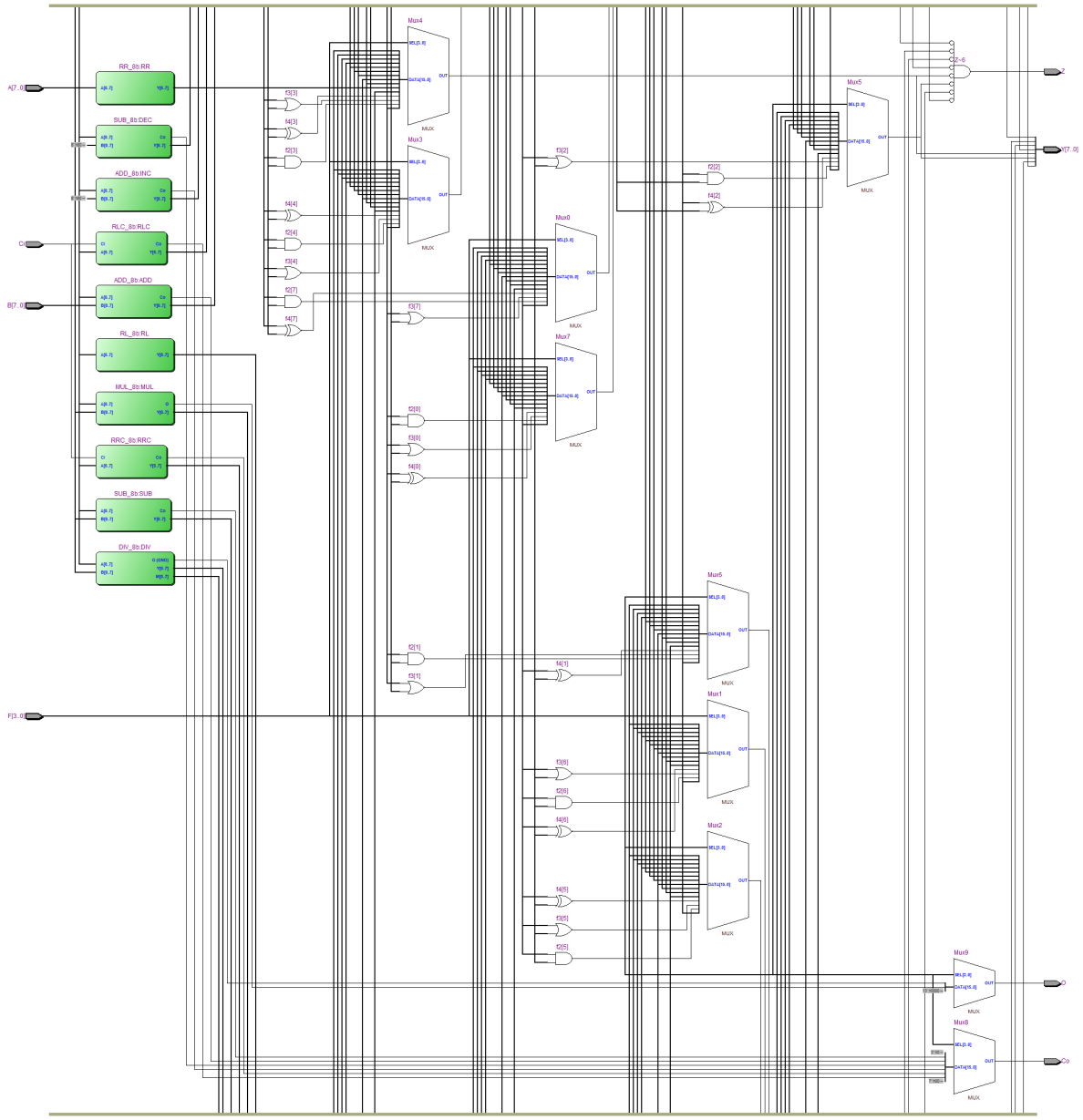
ze 7. bitu. Další příznakem je O - Overflow neboli přetečení. Je nastaven do úrovně H pokud se výsledek operace nevejde do registru. Overflow je nastavován při operacích MUL, DIV a MOD. Posledním příznakem, který tato ALU používá je Z - Zerro příznak nulovosti, nastaví do úrovně H tehdy, je-li výsledek nulový. Tento příznak ovlivňují všechny operace ALU.

5.2 Tabulka funkcí

V tabulce jsou shrnuty všechny funkce ALU, znaky A a B symbolizují vstupní registry.

Název funkce	**Kód funkce**	**Operand 1**	**Operand 2**
FLG	0000	A	-
NOT	0001	A	-
AND	0010	A	B
OR	0011	A	B
XOR	0100	A	B
RL	0101	A	-
RR	0110	A	-
RLC	0111	A	-
RRC	1000	A	-
INC	1001	A	B
DEC	1010	A	B
ADD	1011	A	B
SUB	1100	A	B
MUL	1101	A	B
DIV	1110	A	B
MOD	1111	A	B

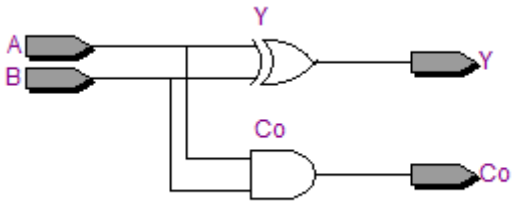
Tabulka 1: Funkce aritmeticko-logické jednotky



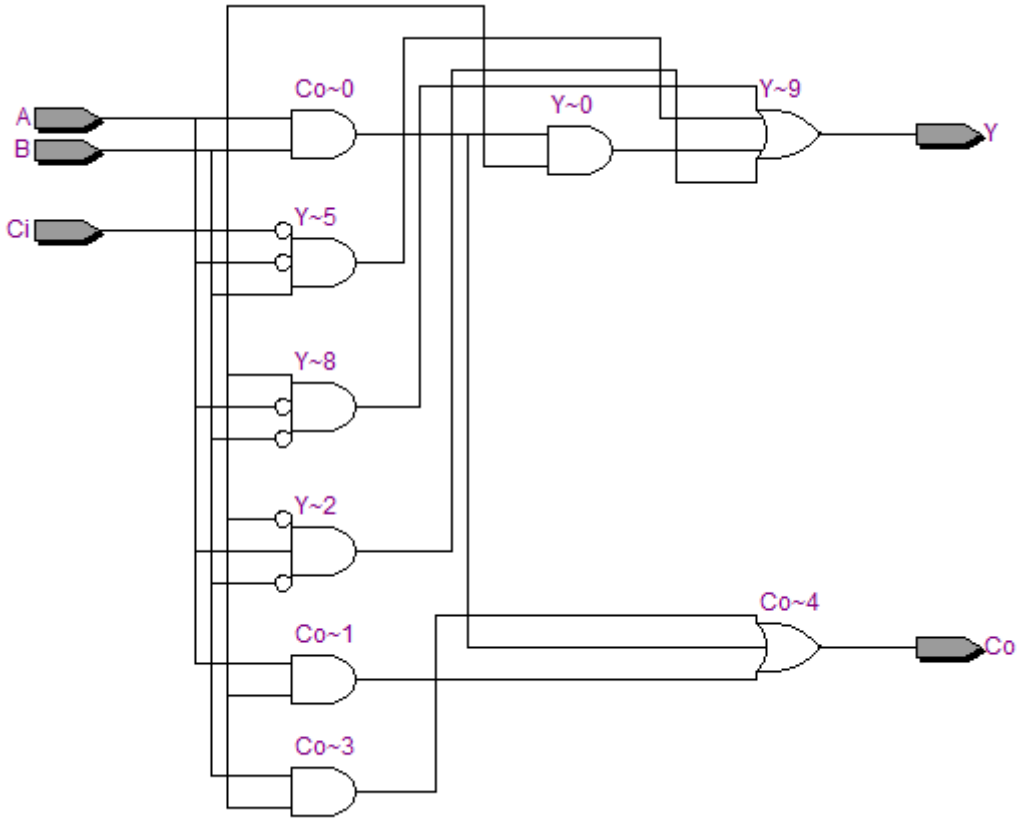
Obrázek 9: Schéma zapojení ALU

5.3 Základní části ALU

ALU zvládá poměrně pestrou škálu operací, k jejich realizaci si však vystačí téměř s jedním typem obvodů. Jedním z nejdůležitějších obvodů je sčítačka. Pro realizaci sčítaček používám dva typy zapojení. Zapojení bez vstupního přenosu, tzv. poloviční sčítačku a sčítačku se vstupním přenosem, tzv. úplnou sčítačku. Zřetězením těchto sčítaček získávám další složitější obvody a to 8 b sčítačku bez vstupní přenosu a 8 b sčítačku se vstupním přenosem.



Obrázek 10: Poloviní sčítačka



Obrázek 11: Úplná sčítačka

5.4 Funkce ALU

5.4.1 FLG

Tato funkce vrátí Y = A totožnou hodnotu, kterou dostane na vstup, ale ovlivní Zerro Flag. Toho můžu využít například při kontrole Nulovosti registru.

5.4.2 NOT

Tato funkce vrátí negovanou hodnotu, kterou dostane na vstup, ovlivní Zerro Flag. Je realizovaná pomocí 8 invertorů.

5.4.3 AND

Tato funkce vrátí Y = A AND B, ovlivní Zerro Flag. Je realizovaná pomocí 8 hradel AND.

5.4.4 OR

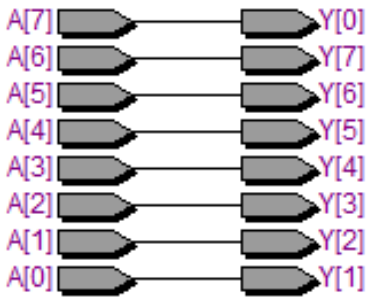
Tato funkce vrátí Y = A OR B, ovlivní Zerro Flag. Je realizovaná pomocí 8 hradel OR.

5.4.5 XOR

Tato funkce vrátí Y = A XOR B, ovlivní Zerro Flag. Je realizovaná pomocí 8 hradel XOR.

5.4.6 RL

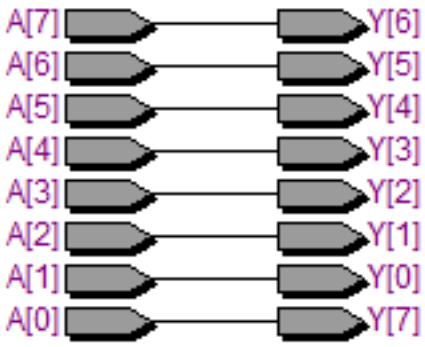
Tato funkce vrátí Y = RL(A), tj, A posunuté o jeden bit do leva. Ovlivní Zerro Flag. Je realizovaný pomocí přeskupení vodičů.



Obrázek 12: Schéma bitového posunu do leva

5.4.7 RR

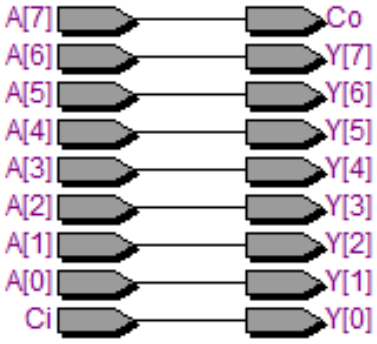
Tato funkce vrátí Y = RR(A), tj, A posunuté o jeden bit doprava. Ovlivní Zerro Flag. Funguje na na podobném principu jako RL.



Obrázek 13: Schéma bitového posunu do prava

5.4.8 RLC

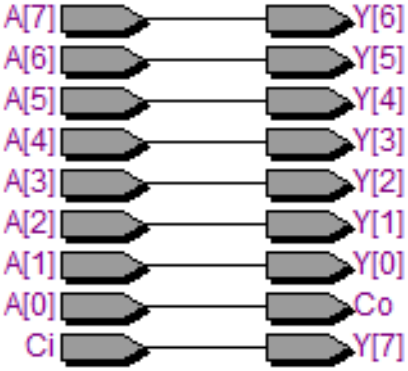
Tato funkce vrátí Y = RLC(A), tj, A posunuté o jeden bit doleva, přes Carry bit. Ovlivní Zerro Flag a Carry Flag. Carry bit je umístěn za vstupem A, tudíž když dojde k posunu doleva, tak hodnoty bitu Carry_{IN} se dostane na pozici Y[0], Y[7] se přesune do Carry_{OUT}. A[6-0] je přesunuta do Y[7-1].



Obrázek 14: Schéma bitového posunu do leva přes carry

5.4.9 RRC

Tato funkce vrátí Y = RRC(A), tj, A posunuté o jeden bit doprava, přes Carry bit. Ovlivní Zerro Flag a Carry Flag. Uspořádáni bitů je totožné jako u funkce RLC. Při posunu dochází k přesunu hodnoty z Carry_{IN} do Y[7], z A[7-1] do Y[6-0] a do Carry_{OUT} je přesunut bit A[7].



Obrázek 15: Schéma bitového posunu doprava přes Carry

5.4.10 INC

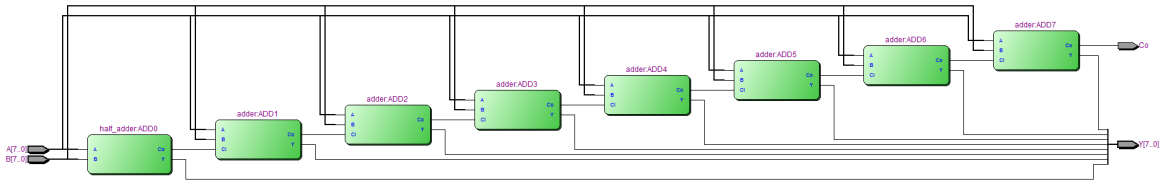
Tato funkce vrátí Y = A + 1. Ovlivní Zerro Flag a Carry Flag. Je realizovaná pomocí jedné sčítačky, Na vstup bitového vektoru B je napevno připojena hodnota $(01)_{HEX}$, díky čemuž dochází k inkrementaci výsledku právě o jedničku.

5.4.11 DEC

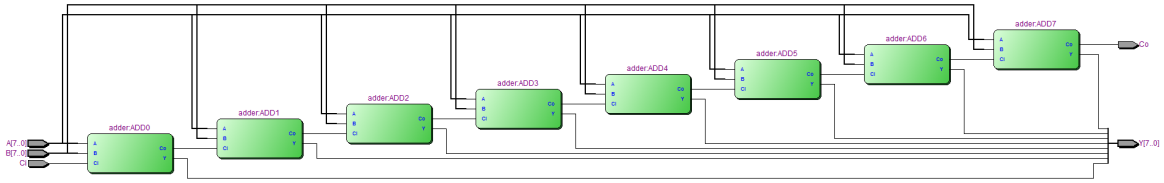
Tato funkce vrátí Y = A - 1. Ovlivní Zerro Flag a Carry Flag. Je realizována pomocí 8 b odčítačky, na vstup bitového vektoru B je připojena hodnota $(01)_{HEX}$.

5.4.12 ADD

Tato funkce vrátí Y = A + B. Ovlivní Zerro Flag a Carry Flag. Je realizována pomocí 8 b sčítačky bez vstupního přenosu. Na její bitový vektor A je přivedena hodnota z registru A a na její bitový vektor B je přivedena hodnota z registru B.



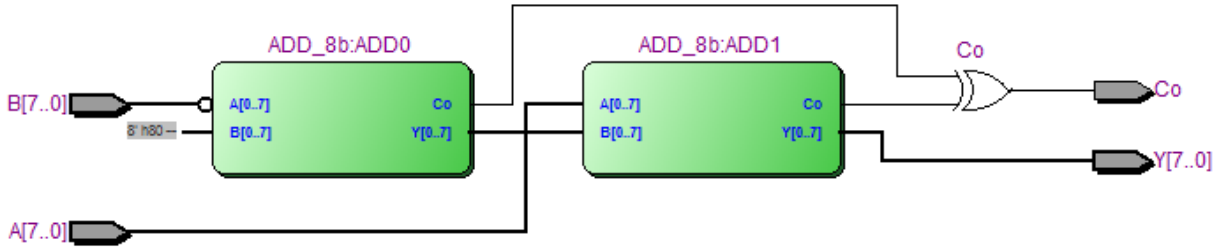
Obrázek 16: Schéma obvodu 8 b sčítačky bez vstupního přenosu



Obrázek 17: Schéma obvodu 8 b sčítačky se vstupním přenosem

5.4.13 SUB

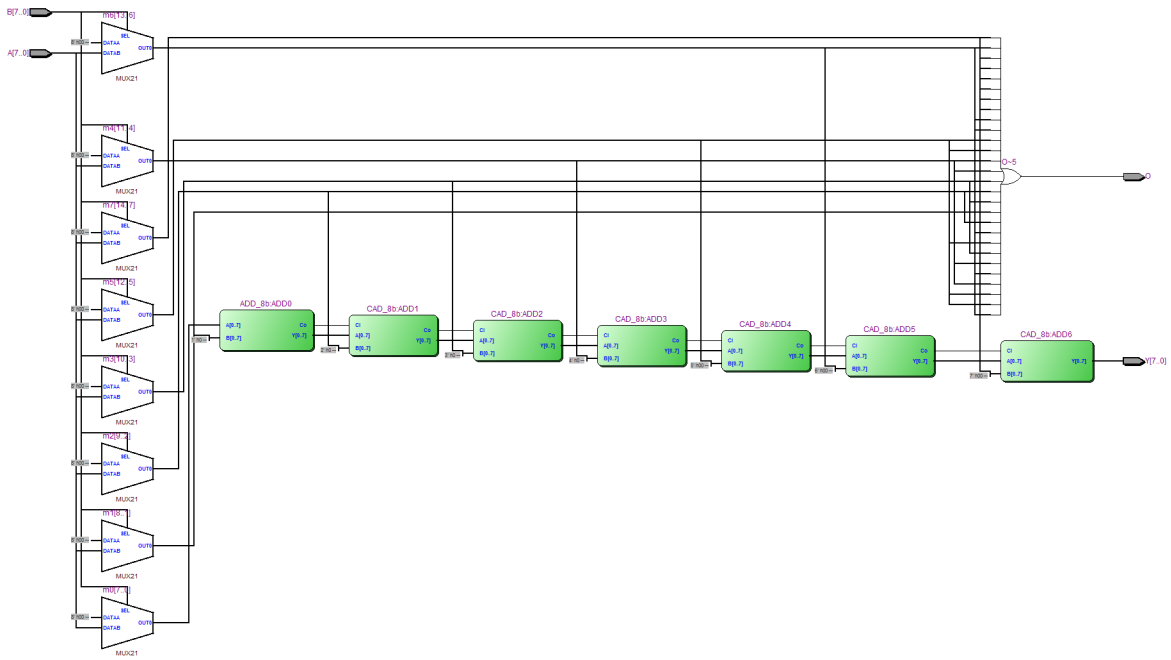
Tato funkce vrátí Y = A - B. Ovlivní Zerro Flag a Carry Flag. Tato funkce je realizovaná pomocí dvou 8 b sčítaček bez vstupního přenosu. Jedna sčítačka slouží k vytvoření dvojkového doplňku k B a druhá sčítačka sečte dvojkový doplněk B a A a tím získáme funkce Y = A - B.



Obrázek 18: Schéma obvodu pro funkci SUB

5.4.14 MUL

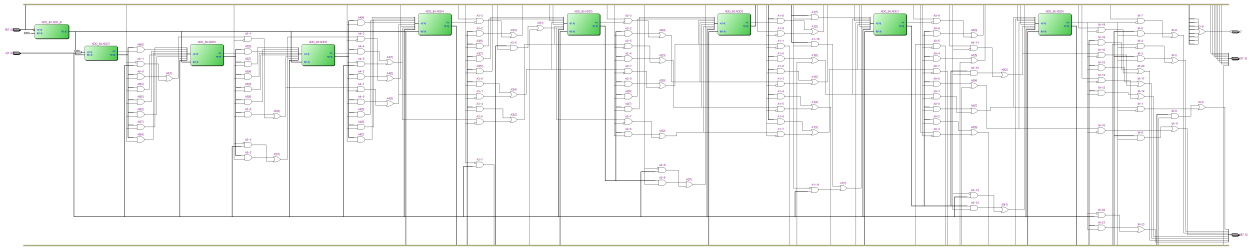
Tato funkce vrátí Y = A · B. Ovlivní Zerro Flag a Overflow Flag. Pomocí multiplexorů se zjistí, zda-li je daný bit v L nebo H a přiřadím mu odpovídající 8 b číslo, tj, buďto $(00)_{HEX}$ nebo $(FF)_{HEX}$. Tyto 8 b číslo jsou sečtena pomocí 1 8 b sčítačky bez vstupního přenosu a k ní kaskádově připojených šesti sčítaček se vstupním přenosem. Obvod byl odvozen ze sčítání pod sebou. Každý bajt je posunut o jedno místo doleva a poté dojde k sečtení bitů na pozicích 7-0. Pomocí 28 b hradla OR , které sčítá všechny bity, které jsou mimo rozsah 7-0, je zjištěna hodnota příznaku Overflow.



Obrázek 19: Schéma obvodu pro funkci MUL

5.4.15 DIV

Tato funkce vrátí Y = $\frac{A}{B}$. Ovlivní Zerro Flag a Overflow Flag. Obvod byl inspirován algoritmem pro dělení mnohočlenu mnohočlenem. V binární podobě je celá operace zjednodušena na porovnávání dělence a dělitele. Pokud je dělenec větší nebo roven jeho děliteli, je výsledný bit roven jedné. To jestli je dělenec větší nebo roven děliteli zjistíme tak, že pomocí 8 b sčítačky vytvoříme dvojkový doplněk dělitele. Tento dvojkový doplněk dělitele následně sčítáme postupně s celým číslem. Postupujeme od MSB. Pokud je po součtu Carry rovno jedné, tak je dělenec větší nebo roven děliteli. Hodnota Carry bitu je zároveň výsledkem operace na dané bitové úrovni. Pokud je Carry rovno nule, tak si vezmeme kromě dalšího bitu i bit předchozí. Pokud je Carry roven jedné, tak další bit připíšeme za výsledek součtu.



Obrázek 20: Schéma obvodu pro funkce DIV a MOD

5.4.16 MOD

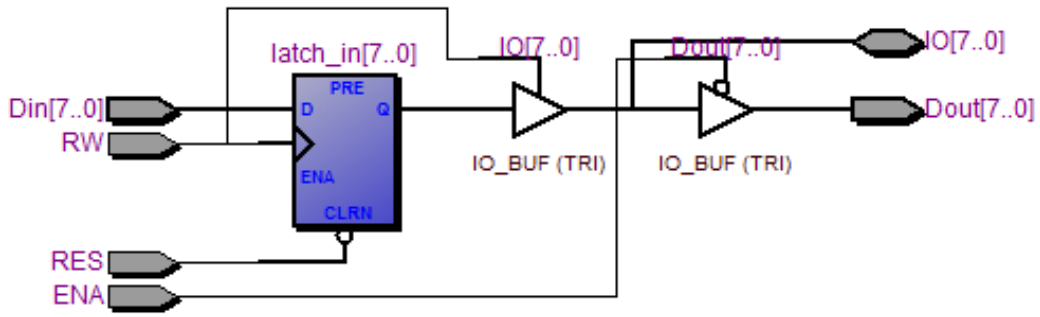
Tato funkce vrátí Y = A MOD B, tj. zbytek po dělení. Ovlivní Zerro Flag a Overflow Flag. Funkce je realizována pomocí zbytku po dělení, tj, posledního součtu dvojkového doplňku s přenosem, který vznikne při dělení.

6 Registry

V procesoru je použito několik druhů registrů. Většina z nich je založena na některém typu klopného obvodu D. Většina registrů má kromě vlastní paměťové buňky ještě další logické členy, pro specifikování zapisovacích a čtecích podmínek.

6.1 RIO

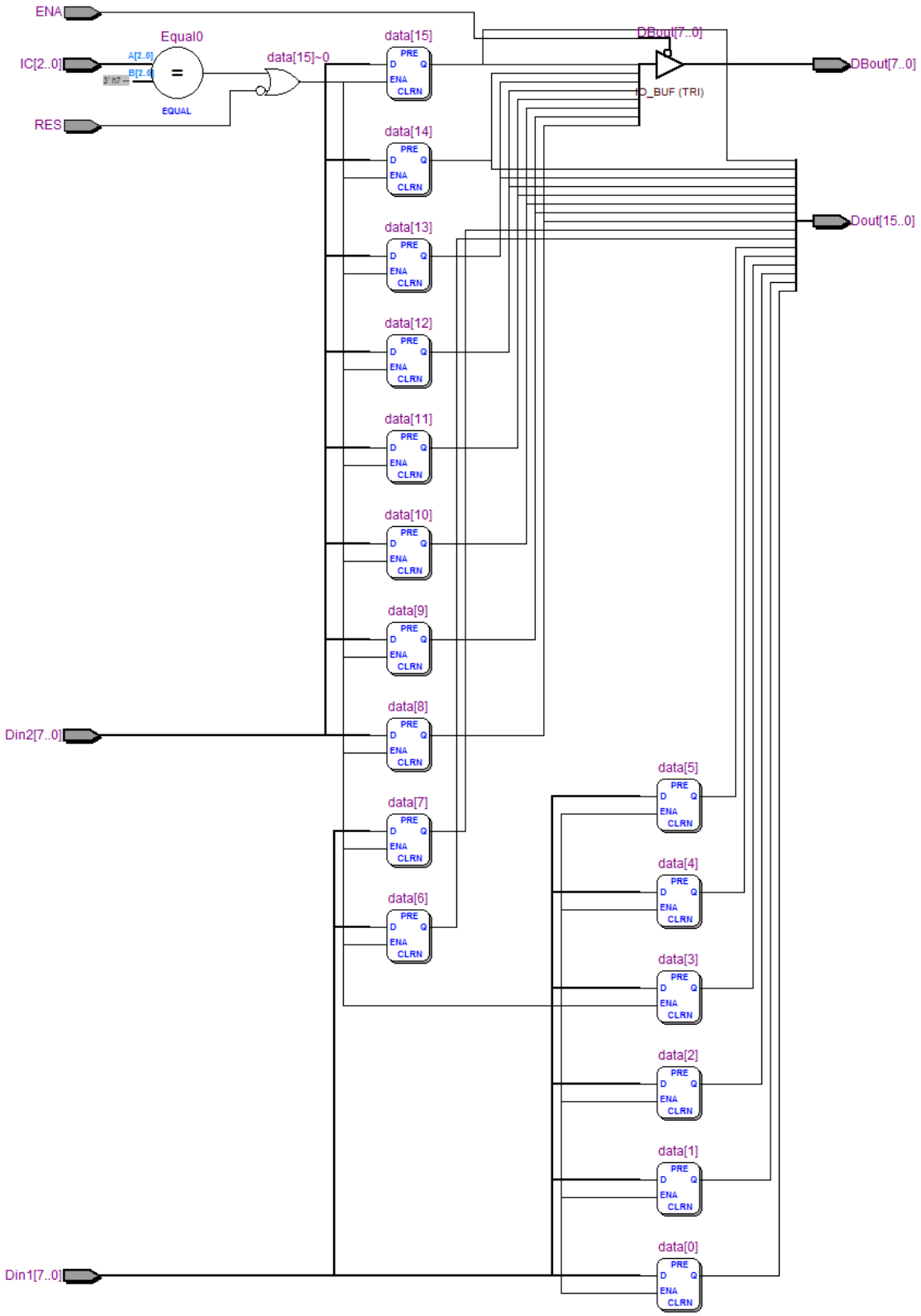
Tento registr umožňuje obsluhovat vstupně výstupní port procesoru. Je tedy určen pro komunikaci s okolím. Ovládá se pomocí signálů RES, ENA, RW. Pokud je RES v logické úrovni L, tak je obsah registru nulován. ENA slouží k připojení vysoké impedance, pokud je tento signál v logické úrovni H, díky tomu se dá registr připojit na sběrnici. Pomocí signálu RW je do registru buď zapsána hodnota (při nástupné hraně) a nebo čtena hodnota z I/O portu pokud je RW v logické úrovni L.



Obrázek 21: Schéma vstupně výstupního registru

6.2 RI

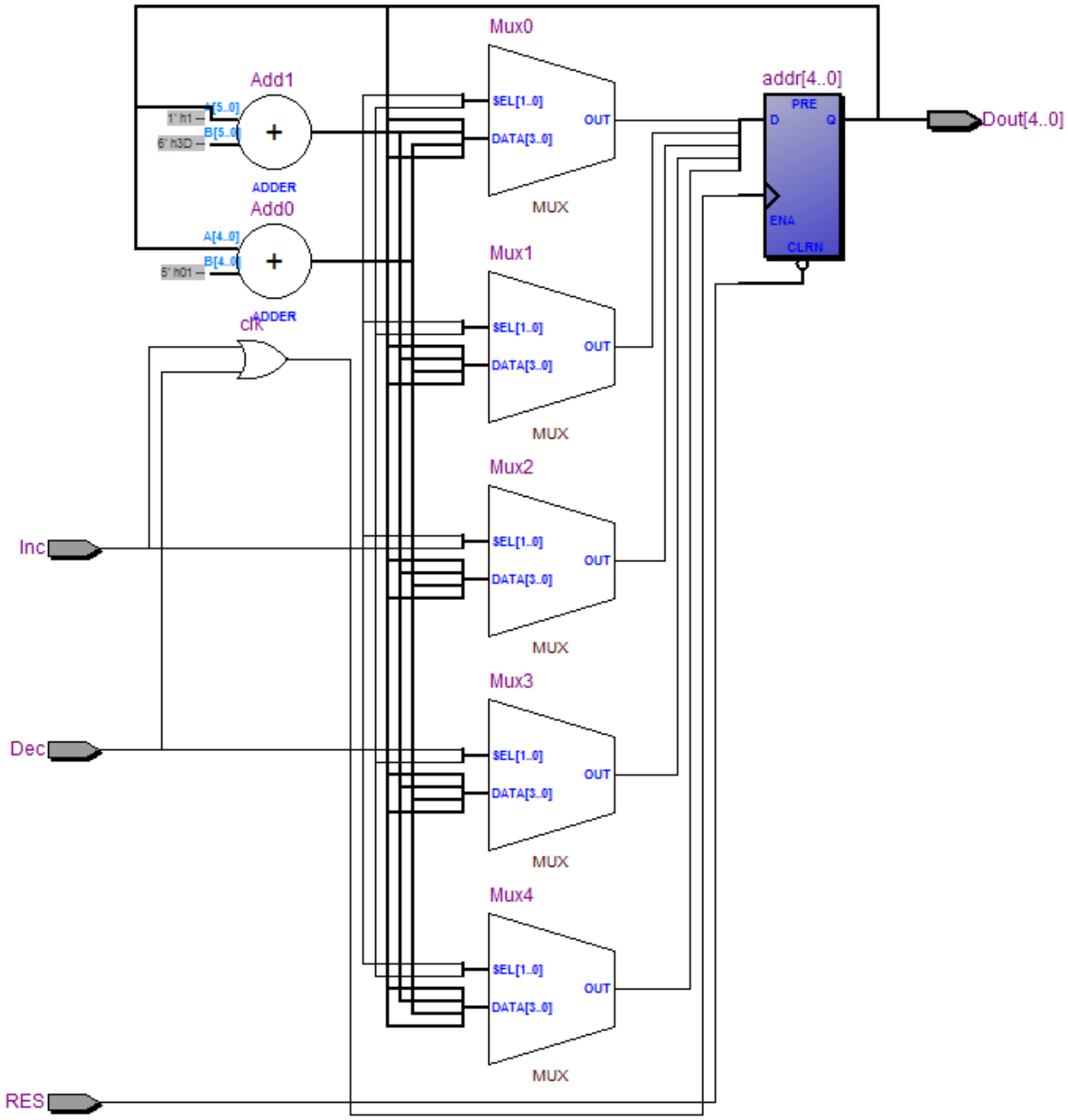
RI (Registr instrukcí) je speciální registr, který slouží pro uložení právě prováděné adresy. Instrukční cyklus procesoru trvá 8 period oscilátoru. Takže je tento registr navrhnut tak, aby svou hodnotu změnil právě jednou za 8 period hodinového signálu. Registr se ovládá pomocí signálů CLK, Din1, Din2 a RES, Obvod má asynchronní reset, pokud je na signálu RES hodnota L, dojde k reseru při kterém je vynicena aktualizaci obsahu regisru. Signály Din1 a Din2 jsou 8 b vektory, slouží pro připojení registru k DPRAM. Pokud je na signálu CLK zjištěna nástupná hrana, tak se zvýší hodnota vnitřního čítače. Pokud čítač dosáhne hodnoty $(111)_{BIN}$ tak se uloží do registru hodnota, která je na bitových vektorech Din1 a Din2 a vnitřní čítač se vynuluje.



Obrázek 22: Schéma registru instrukcí

6.3 SP

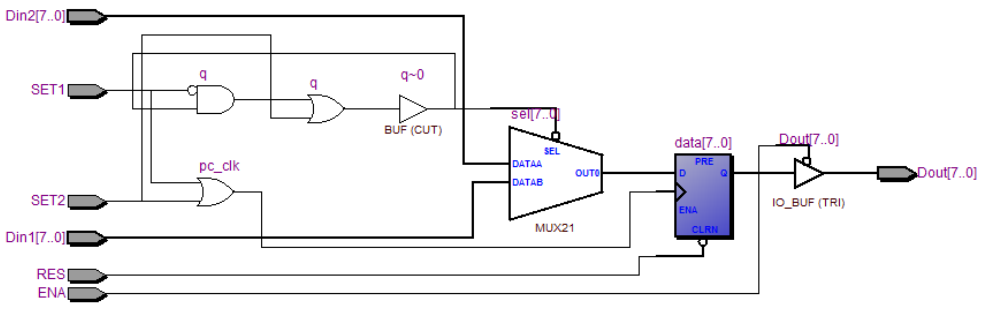
SP (Stack Pointer) Ukazatel zásobníku je speciální registr, který slouží pro uložení adresy vrcholu zásobníku. Jedná se o 5 b registr, protože používám 32 B zásobníky. Ovládá se pomocí signálů INC, DEC a RES. Signál RES slouží jako asynchronní nulování. Signál INC inkrementuje hodnotu registru o jedničku a signál DEC dekrementuje hodnotu registru o jedničku. Signály INC a DEC reagují na nástupnou hranu, signál RES reaguje na úroveň L.



Obrázek 23: Schéma registru SP

6.4 PC

PC (Program Counter) Programový čítač je speciální registr, který slouží pro uložení adresy právě zpracovávané instrukce. Je ovládán pomocí signálů RES, SET1, SET2, Ain1 a Ain2. Pomocí signálu RES je vytvořeno asynchronní nulování, stejné jako u předchozích registrů. Signály SET1 a SET2 slouží k přiřazení hodnoty do registru při nástupní hraně napětí právě na jednom z těchto signálů. Pokud je nástupná hrana zaznamenána na SET1 je do registru přiřazena hodnota na Ain1, pokud je nástupná hrana zachycena na signálu SET2, tak do registru je přiřazena hodnota z Ain2.



Obrázek 24: Schéma registru PC

7 RB

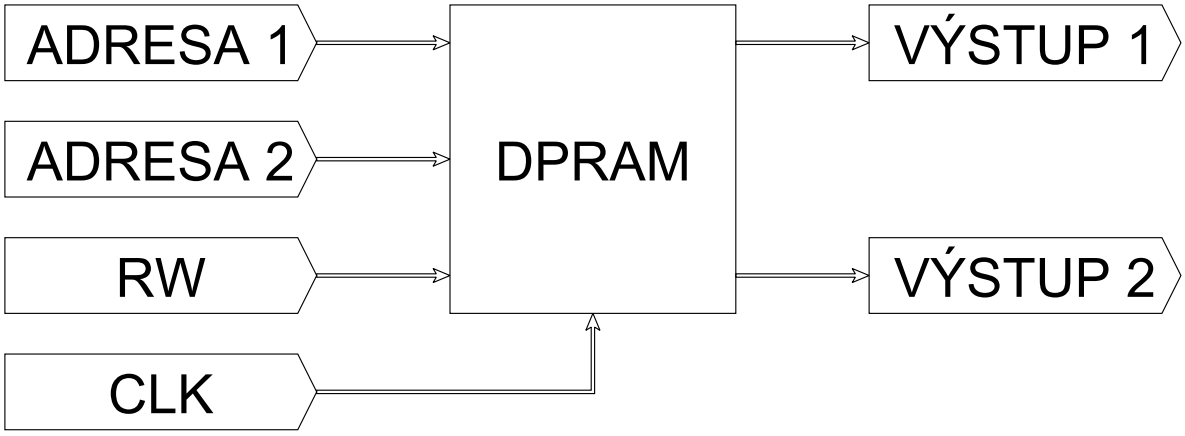
7.1 Popis bloku

RB (Registers Bank) česky registrová banka, je obvod který seskupuje několik registrů do jednoho. Výhodou tohoto zapojení je, snížení počtu řídících signálů, vedoucích z řadiče k registrům. Registrová banka se ovládá pomocí šesti signálů. RW při nástupné hraně zapíše do registru vybraného pomosí signálu SEL1 data, která jsou na signálovém vektoru Din. ENA tento signál v logické úrovni L povolí výstup na Dout1 a Dout2, jinak je Dout1 a Dout2 ve stavu vysoké impedance. SEL1 vybírá pracovní registr pro čtení i zápis, SEL2 vybírá registr pouze pro výstup na Dout2.

8 Paměti

V procesoru jsou kromě registrů použity paměti o velikosti několika desítek bajtů, jedná se o paměti typu zásobník nebo RAM.

8.1 SSDPRAM



Obrázek 25: Schéma zásobníku

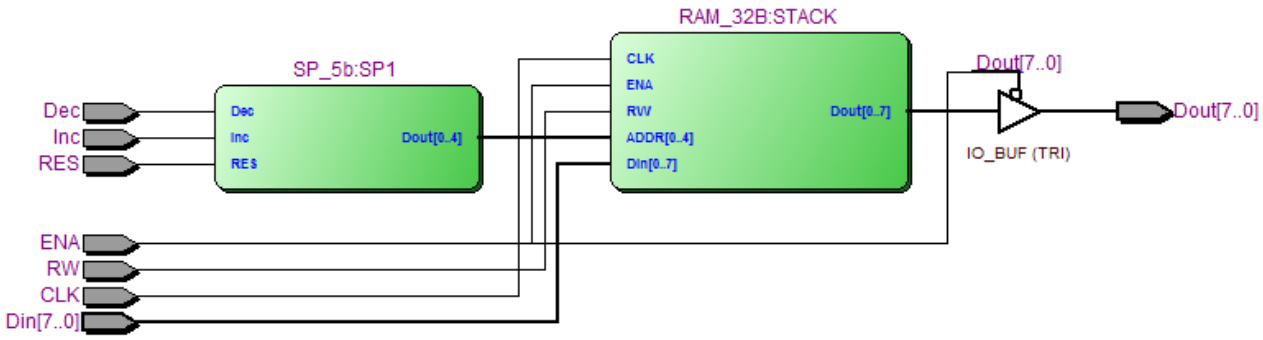
SSDPRAM (Static Synchoronut Dual Port Random Access Memory) dále jen RAM Jedná se o sériovou synchronní paměť speciálně navrženou pro tuto aplikaci. Má velikost 256 B. Její zvláštností je, že má vstupy pro dvě 8 b adresy a dva 8 b výstupy. Ale pouze na jednom portu se dá s paměti číst i do ní zapisovat. Díky této paměti může být v jednom taktu oscilátoru načtena celá 16 b instrukce. Zapisuje / čte se do ní při nástupní hraně hodinového signálu. Zápis a čtení se pozná pomocí signálu RW. Pokud je signál RW roven nule, tak je nastaven mód čtení, pokud je na RW zachycena nástupná hrana, tak se zapíše do paměti hodnota, která je na vstupu Din.

8.2 SSRAM

SSRAM(Static Synchoronut Random Access Memory) Jedná se o sériovou synchronní paměť o velikosti 32 B, která je v tomto μprocesoru používána v zásobnících. Zápis a čtení je totožné s pamětí SSDPRAM.

8.3 Stack

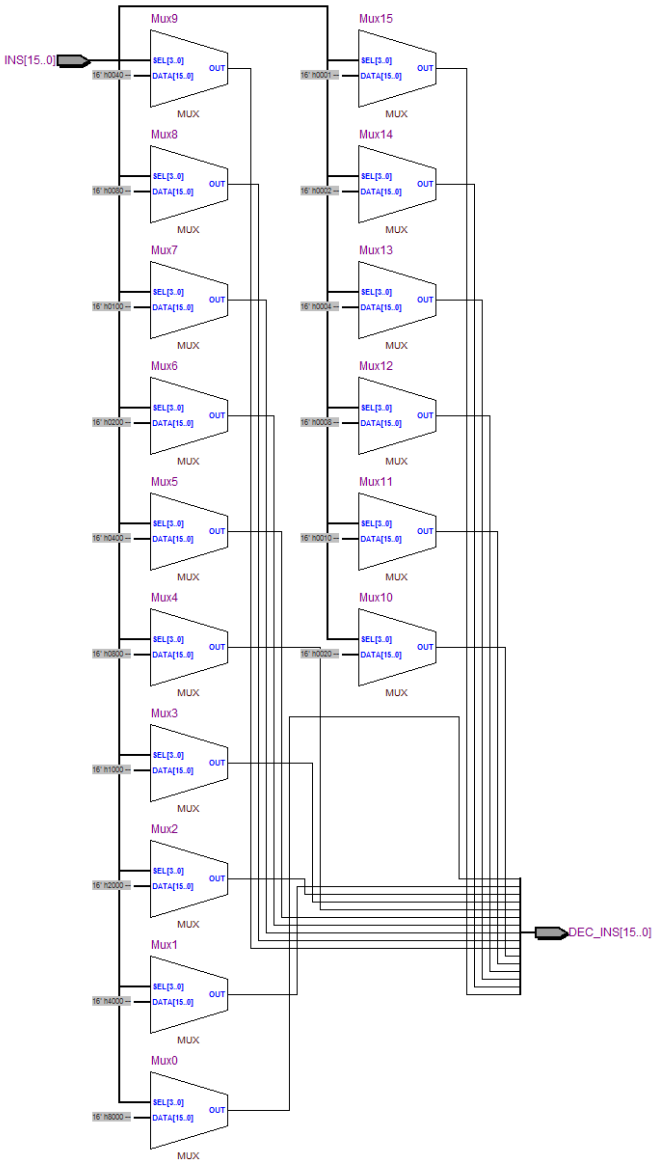
Stack - zásobník, je paměť vytvořená pomocí registru SP a 32 B paměti RAM. Zásobníky jsou paměti typu LIFO (Last In First Out). V romto procesoru jsou využity pro uchovávání návratových adres a pro zálohování hodnot registrů.



Obrázek 26: Schéma zásobníku

9 DI

Dekodér instrukcí, dekóduje data z registru instrukcí a dává do výstupního bitového vektoru úroveň H na odpovídající polohu instrukce, tuto polohu v bitovém vektoru následně vyhodnocuje CU (Control Unit).



Obrázek 27: Schéma dekodéru instrukcí

10 CU

10.1 Popis bloku

CU (Control Unit) český název řadič nebo řídící jednotka, je jedna z nejdůležitějších částí procesoru. Řídí takřka všechny části procesoru, na základě informací které dostane z DI (Dekodéru instrukcí), RI (Registru Instrukcí), IC (Instrukčního čítače) a signálu RES a IRQ. Bohužel, ale tento blok ještě není zcela odladěn a stále mám problém s instrukcemi PUSH a POP.

Jak již bylo zmíněno výše, tento procesor je navrhován jako RISC, tudíž jsou všechny instrukce stejně dlouhé, momentálně trvá instrukční cyklus, (tj. doba načtení dat z RAM a jejich zpracování) osm period hodinového signálu. Z toho vyplývá, že procesor mám momentálně při taktovací frekvenci 24 MHz je výpočetní výkon 3 MIPS.

10.2 Tabulka instrukcí

Název instrukce	**Kód funkce (hex)**	**Operand 1**	**Operand 2**
JMP	0	A	-
JC	1	A	-
JO	2	A	-
JZ	3	A	-
JNC	4	A	-
JNO	5	A	-
JNZ	6	A	-
CALL	7	A	-
RET	8	-	-
LOA	9	R	D
MOV	*A*	R	R
SAV	*B*	A	R
PUSH	*C*	R	-
POP	*D*	R	-
ALI	*E*	R	R/-

Tabulka 2: Instrukční soubor μprocesoru

10.3 JMP

JMP (Jump) česky skok. Tato instrukce změní obsah PC (Programového Čítače) na adresu, která je uvedena v následujícím bytu v RAM za bytem říkajícím proveď instrukci JMP.

10.4 JC

JC (Jump Carry) Tato instrukce provede skok tehdy, pokud je Carry bit v logické úrovni H. Instrukce bude opět skákat na adresu, která je uvedena za bytem identifikujícím instrukci JC.

10.5 JO

JO (Jump Overflow) Tato instrukce skočí tehdy, je-li bit signalizující Overflow v logické úrovni H.

10.6 JZ

JZ (Jump Zerro) Tato instrukce provede skok pokud je Zerro flag v logické hodnotě H. Pro všechny podmínkové skoky platí, pokud není splněna podmínka, tak se zvýší hodnota adresy, na kterou ukazuje PC o dva.

10.7 JNC

JNC (Jump Not Carry) instrukce se chová stejně jako instrukce JC, s tím rozdílem, že se provede skok tehdy, je-li Carry bit v logické úrovni L.

10.8 JNO

JNO (Jump Not Overflow) Tato instrukce je podobná instrukci JO, s tím rozdílem, že skočí, pokud je hodnota Carry bitu v logické úrovni L.

10.9 JNZ

JNZ (Jump Not Zerro) Tato instrukce je podobná jako instrukce JZ, liší se pouze v tom, že skok bude proveden, pokud bude úroveň na Zerro flagu v logické úrovni L.

10.10 CALL

CALL česky volání, tato instrukce změní obsah registru PC na adresu, která je uvedena za bytem identifikujícím tuto instrukci. Předtím než to ale provede, uloží aktuální obsah registru PC do STA (Stack Address) zásobníku adres.

10.11 RET

K hodnotě v STA (Stack Address) přičte dvojku, aby se dostal na novou instrukci a tuto hodnotu uloží do PC. Díky instrukcím CALL a RET, tedy je možné realizovat na Lojzovi podprogramy.

10.12 LOA REGISTER, DATA

Tato instrukce přesune hodnotu RAM do registru, při vykonávání této instrukce ale nedochází k přímému přístupu do RAM, jelikož v RI je uložen i následující byte. Takže během instrukce dochází k přenosu obsahu osmi nejvýznamějších bitů z IR do registru, který je uveden za nibblem určujícím instrukci.

10.13 MOV REGISTER, REGISTER

Tato instrukce slouží ke kopírování hodnot mezi registry.

10.14 SAV ADDRESS, REGISTER

Přesune honotu z registru do paměti RAM.

10.15 PUSH

Tato instrukce zatím není plně doladěna.

10.16 POP

Tato instrukce má zatím stejné neduhy jako instrukce PUSH.

10.17 ALI

Provede jednu ze šestnácti možných aritmeticko-logických operací a výslednou hodnotu uloží do registru Y. Tato instrukce jako jediná dokáže ovlivňovat příznaky procesoru.

11 Závěr

11.1 Zhodnocení

Cíl projektu byl splněn, byl navržen a realizován procesor, bohužel z časových důvodů zatím nefungují dvě instrukce, PUSH a POP. Tento projekt byl pro autora velikým přínosem, jelikož se naučil pracovat se zcela novým typem obvodům, což bylo jedním z cílů tohoto projektu.

Navržený μprocesor sice nepatří mezi nejvýkonnější procesory na světě, ale to ani nebylo autorovým záměrem.

11.2 Pokračování projektu

Do budoucna bych chtěl zajistit funkčnost všech instrukcí, tedy dodělat instrukce PUSH a POP, také bych chtěl napsat assembler pro tento procesor. Díky znalostem získaným při Lojzově vývoji bych chtěl pokračovat v navrhování procesorů. Lákají mě zásobníkové architektury a tak možná začnu vyvíjet zásobníkový procesor, ve kterém bych mohl využít i moduly vytvořené při tomto projektu. Pak bych chtěl zkusit využít toho, že Lojza a jeho architektura WPU8 je navrhnuta jako RISC a tak bych chtěl vytvořit další procesor, ve kterém bych aplikoval zřetězené zpracování instrukcí.

Seznam obrázků

Seznam tabulek

Literatura

[1] *Ing. Jiří Král:* **Řešené příklady ve VHDL - Hradlová pole pro začátečníky** BEN - technická literatura, Praha 2010

[2] *Pavel Tišnovský:* **Pohled do nitra mikroprocesoru** [online] [cit. 13-4-2014] Dostupné z: `http://www.root.cz/clanky/pohled-do-nitra-mikroprocesoru/`

[3] *Doc. Ing. Jaromír Kolouch, CSc.:* **Informace o programovatelných logických obvodech** [online] [cit. 13-4-2014] Dostupné z: `http://www.urel.feec.vutbr.cz/k̃olouch/pld/`

[4] *UNITES Systems a.s.:* **Stránky výrobce vývojového kitu** [online] [cit. 13-4-2014] Dostupné z: `http://www.unicell.cz/`

[5] *Vlastimil Slinták:* **Uvod do jazyka VHDL** [online] [cit. 13-4-2014] Dostupné z: `http://uart.cz/563/uvod-do-pld-a-jazyka-vhdl/`

[6] *Altera Corporation:* **Dokumentace k hradlovému poli Cyclone II** [online] [cit. 13-4-2014] Dostupné z: `http://www.altera.com/literature/hb/cyc2/cyc2_cii5v1.pdf`

A PŘÍLOHA

A.1 Obsah přiloženého DVD

- Zdrojové kódy architektury WPU8
- Zdrojové kódy aplikace využívající Quin McCluskeyho algoritmus pro minimalizaci logických funkcí
- Schémata generovaná programem Quartus II v 13.0 po syntéze kódů
- Fotografie z projektu
- Aplikace Quartus II v 13.0
- Aplikace Python 2.7