ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ

FAKULTA ELEKTROTECHNICKÁ

KATEDRA MĚŘENÍ

DIPLOMOVÁ PRÁCE

Model číslicového počítače v Cadence

Digital Computer Model in Cadence

Bc. Jana Ščuková

Vedoucí práce: Ing. Vladimír Janíček, Ph.D.

ZADÁNÍ DIPLOMOVÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Ščuková** Jméno: **Jana** Osobní číslo: **434908**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra měření**

Studijní program: **Otevřená informatika**

Studijní obor: **Počítačové inženýrství**

II. ÚDAJE K DIPLOMOVÉ PRÁCI

Název diplomové práce:

Model číslicového počítače v Cadence

Název diplomové práce anglicky:

Digital Computer Model in Cadence

Pokyny pro vypracování:

1) Proveďte rozbor funkce číslicového počítače von Neumannovské architektury a popište obvodovou realizaci jednotlivých bloků.
2) Navrhněte vlastní koncept modelu číslicového počítače.
3) Navrhněte jednotlivé bloky realizující aritmetické a logické operace (ALU, operační a programová paměť, řadič).
4) Navrhněte obvodové zapojení v Cadence.
5) Funkcionalitu otestujte na ukázkových úlohách.

Seznam doporučené literatury:

1) ILS kurzy Cadence (http://learning.cadence.com/)
2) Digital Systems: From Logic Gates to Processors, J-P Deschamps, Springer, ISBN-13: 978-3319411972
3) Digital Logic and Microprocessor Design with Interfacing, E. O. Hwang, ISBN-13: 978-1305859456

Jméno a pracoviště vedoucí(ho) diplomové práce:

Ing. Vladimír Janíček, Ph.D., katedra mikroelektroniky FEL

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) diplomové práce:

Datum zadání diplomové práce: **24.01.2019** Termín odevzdání diplomové práce: _____

Platnost zadání diplomové práce:
do konce letního semestru 2019/2020

Ing. Vladimír Janíček, Ph.D. podpis vedoucí(ho) práce	podpis vedoucí(ho) ústavu/katedry	prof. Ing. Pavel Ripka, CSc. podpis děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

Diplomantka bere na vědomí, že je povinna vypracovat diplomovou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací. Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v diplomové práci.

Datum převzetí zadání Podpis studentky

© ČVUT v Praze, Design: ČVUT v Praze, VIC

Čestné prohlášení autora práce

Prohlašuji, že jsem předloženou práci vypracovala samostatně a že jsem uvedla veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

Bc. Jana Ščuková

V Praze dne …….. ……

Podpis autora práce

Poděkování

Ráda bych poděkovala Ing. Vladimíru Janíčkovi, PhD. za podporu, trpělivost a pozitivní přístup při zpracování této diplomové práce.

Dále bych ráda poděkovala své rodině za podporu po celou dobu mého studia.

Anotace

Cílem diplomové práce je návrh procesoru za pomocí CMOS logiky a ověření jeho funkce v komplexních simulacích v prostředí Cadence Virtuoso. Práce je koncipována jako doplňkový materiál při výuce elektronických předmětů pro studenty informatiky.

Klíčová slova

Cadence, digitální návrh, simulace, logická hradla, procesor, datová paměť, instrukční paměť, řadič, ALU, Euklidův algoritmus

Annotation

Aim of this diploma theses is design of processor in CMOS logic and … of its function by complex simulations in Cadence Virtuoso. Theses may be used as additional materiál during electronical lectures for students of informatics.

Key words

Cadence, digital design, simulation, logic gates, processor, data memory, instruction memory, control unit, ALU, Euclidean algorithm

OBSAH

Seznam tabulek ... 13
Seznam obrázků ... 13
1 Úvod ... 17
1.1 Vývojové prostředí ... 17
2 Teoretický rozbor ... 18
3 Blokové schéma procesoru ... 19
3.1 Sada instrukcí ... 21
4 Datová paměť ... 23
4.1 Paměťová jednotka ... 23
4.1.1 Read-Modify-Write ... 27
4.2 Datová paměť procesoru ... 29
4.2.1 Dekodér adresy ... 29
4.2.2 Zapojení datové paměti ... 33
5 Instrukční paměť ... 42
5.1 Instrukční paměťová jednotka ... 42
5.2 Instrukční paměť procesoru ... 45
6 Aritmeticko-logická jednotka ... 50
6.1 Podporované operace ... 50
6.1.1 Sčítání, odčítání ... 50
6.1.2 Bitový posun ... 53
6.1.3 AND, OR, XOR ... 56
6.2 Zapojení ALU ... 62
7 Řadič ... 69
8 Procesor – komplexní simulace ... 71
8.1 Sekvenční procesor s čítačem instrukcí ... 71
8.1.1 Program ... 71
8.1.2 Instrukční paměť ... 71
8.1.3 Zapojení procesoru ... 72
8.1.4 Simulace ... 77
8.1.5 Layout ... 82
8.2 Sekvenční procesor bez čítače instrukcí ... 83
8.2.1 Program ... 83
8.2.2 Instrukční paměť ... 83
8.2.3 Zapojení procesoru ... 83

8.2.4 Simulace 84
8.2.5 Layout 88
8.3 Program se skoky 89
8.3.1 Program 89
8.3.2 Instrukční paměť 89
8.3.3 Zapojení procesoru 91
8.3.4 Simulace 95
8.3.5 Layout 99
9 Závěr 100
10 Použitá literatura 101
11 Seznam použitých zkratek a symbolů 103
12 Seznam příloh 104

SEZNAM TABULEK

Tabulka 1: Kódy jednotlivých operací a jejich význam 21
Tabulka 2: Pravdivostní tabulka adresního dekodéru 29
Tabulka 3: Pravdivostní tabulka multiplexoru 36
Tabulka 4: Detekce přetečení součtu dvou čísel A+B 50
Tabulka 5: Pravdivostní tabulka operace AND 56
Tabulka 6: Pravdivostní tabulka operace OR 58
Tabulka 7: Pravdivostní tabulka operace XOR 60
Tabulka 8: Přehled hodnot výstupních hodnot ALU v závislosti na řídicích signálech 62
Tabulka 9: Uložení zakódovaných instrukcí v instrukční paměti – sekvenční program s čítačem instrukcí 71
Tabulka 10: Zakódované instrukce pro Euklidův algoritmus 89
Tabulka 11: Průběh simulace Euklidova algoritmu 95
Tabulka 12: Počet tranzistorů v diplomové práci 100

SEZNAM OBRÁZKŮ

Obrázek 1: Blokové schéma procesoru 19
Obrázek 2: Ukázka pořadí operací prováděných při sčítání dvou čísel 20
Obrázek 3: Proces zpracování instrukcí 22
Obrázek 4: Schéma paměťové jednotky 23
Obrázek 5: Detail paměťové jednotky 23
Obrázek 6: Symbol paměťové jednotky 24
Obrázek 7: Schéma simulace paměťové jednotky 24
Obrázek 8: Průběh simulace paměťové jednotky 1 25
Obrázek 9: Průběh simulace paměťové jednotky 2 26
Obrázek 10: Průběh simulace paměťové jednotky 3 26
Obrázek 11: Schéma simulace Read-Modify-Write 27
Obrázek 12: Průběh simulace Read-Modify-Write 28
Obrázek 13: Schéma dekodéru 1 z 16 30
Obrázek 14: Symbol dekodéru 30
Obrázek 15: Schéma simulace dekodéru 1 z 16 31
Obrázek 16: Průběh simulace dekodéru 1 z 16 - vstupní signály 31
Obrázek 17: Průběh simulace dekodéru 1 z 16 - out0 až out7 32
Obrázek 18: Průběh simulace dekodéru 1 z 16 - out8 až out15 32
Obrázek 19: Schéma datové paměti procesoru 33
Obrázek 20: Schéma datové paměti procesoru – zápis 34
Obrázek 21: Schéma zapojení datové paměti – zápis – výřez 35
Obrázek 22: Symbol multiplexoru 36
Obrázek 23: Multiplexor 1 z 16 36
Obrázek 24: Schéma zapojení datové paměti procesoru – čtení A – výřez 37
Obrázek 25: Schéma zapojení datové paměti procesoru – čtení A 38
Obrázek 26: Schéma zapojení datové paměti – čtení B – výřez 38
Obrázek 27: Schéma zapojení datové paměti – čtení B 39

Obrázek 28: Symbol datové paměti 39
Obrázek 29: Schéma simulace datové paměti 40
Obrázek 30: Průběh simulace datové paměti – A 41
Obrázek 31: Průběh simulace datové paměti – B 41
Obrázek 32: Schéma instrukční paměťové jednotky 42
Obrázek 33: Symbol instrukční paměťové jednotky 42
Obrázek 34: Schéma simulace instrukční paměťové jednotky 43
Obrázek 35: Průběh simulace instrukční paměťové jednotky – horních 8 bitů 44
Obrázek 36: Průběh simulace instrukční paměťové jednotky – dolních 8 bitů 44
Obrázek 37: Schéma instrukční paměti 45
Obrázek 38: Schéma instrukční paměti – zápis 46
Obrázek 39: Instrukční paměť – zápis – výřez 47
Obrázek 40: Schéma instrukční paměti – čtení 48
Obrázek 41: Instrukční paměť – čtení – výřez 49
Obrázek 42: Symbol instrukční paměti 49
Obrázek 43: Schéma 8-bitové sčítačky/odčítačky 51
Obrázek 44: Symbol 8-bitové sčítačky/odčítačky 51
Obrázek 45: Schéma simulace 8-bitové sčítačky/odčítačky 52
Obrázek 46: Průběh simulace 8-bitové sčítačky/odčítačky 52
Obrázek 47: Schéma 8-bitového aritmetického posunu doleva 53
Obrázek 48: Symbol 8-bitového aritmetického posunu doleva 53
Obrázek 49: Simulace 8-bitového posunu doleva 54
Obrázek 50: Průběh simulace 8-bitového posunu doleva – vstupní data 54
Obrázek 51: Průběh simulace 8-bitového posunu doleva – výstupní data 54
Obrázek 52: Schéma 8-bitového posunu doprava 55
Obrázek 53: Symbol 8-bitového posunu doprava 55
Obrázek 54: Simulace 8-bitového posunu doprava 55
Obrázek 55: Průběh simulace 8-bitového posunu doprava – vstupní signály 56
Obrázek 56: Průběh simulace 8-bitového posunu doprava - výstupní signály 56
Obrázek 57: Schéma zapojení 8-bitového AND 57
Obrázek 58: Symbol 8-bitového AND 57
Obrázek 59: Simulace 8-bitového AND 57
Obrázek 60: Průběh simulace 8-bitového AND 58
Obrázek 61: Schéma zapojení 8-bitového OR 58
Obrázek 62: Symbol 8-bitového OR 58
Obrázek 63: Simulace 8-bitového OR 59
Obrázek 64: Průběh simulace 8-bitového OR 59
Obrázek 65: Schéma zapojení 8-bitového XOR 60
Obrázek 66: Symbol 8-bitového XOR 60
Obrázek 67: Schéma zapojení simulace XOR 61
Obrázek 68: Průběh simulace XOR 61
Obrázek 69: Schéma ALU multiplexoru pro jeden bit 62
Obrázek 70: Symbol ALU multiplexoru pro jeden bit 62
Obrázek 71: Schéma zapojení výstupního bloku ALU 63
Obrázek 72: Symbol výstupního bloku ALU 64
Obrázek 73: Symbol ALU 65
Obrázek 74: Schéma zapojení ALU 66
Obrázek 75: Schéma zapojení simulace ALU 67

Obrázek 76: Průběh simulace ALU 68
Obrázek 77: Schéma zapojení řadiče 69
Obrázek 78: Schéma dekodéru instrukce 69
Obrázek 79: Schéma zapojení kontrolní jednotky 70
Obrázek 80: Schéma zapojení instrukční paměti sekvenčního procesoru s čítačem instrukcí ... 72
Obrázek 81: Schéma čítače instrukcí 73
Obrázek 82: Symbol čítače instrukcí 73
Obrázek 83: Zapojení simulace čítače instrukcí 73
Obrázek 84: Průběh simulace čítače instrukcí 73
Obrázek 85: Schéma zapojení procesoru s čítačem instrukcí 74
Obrázek 86: Schéma zapojení procesoru s čítačem instrukcí – výřez – instrukční paměť, čítač instrukcí, řadič 74
Obrázek 87: Schéma zapojení procesoru s čítačem instrukcí – výřez – datová paměť, ALU, výstup 75
Obrázek 88: Schéma zapojení procesoru s čítačem instrukcí – výřez – datová paměť, switch .. 75
Obrázek 89: Schéma zapojení 8-bitového přepínače 76
Obrázek 90: Symbol zapojení 8-bitového přepínače 76
Obrázek 91: Schéma zapojení 1-bitového přepínače 77
Obrázek 92: Symbol 1-bitového přepínače 77
Obrázek 93: Symbol procesoru s čítačem instrukcí 77
Obrázek 94: Schéma zapojení simulace procesoru s čítačem instrukcí 78
Obrázek 95: Průběh výstupních signálů z datové paměti (A) – procesor s čítačem instrukcí 79
Obrázek 96: Průběh výstupních signálů z datové paměti (B) – procesor s čítačem instrukcí 79
Obrázek 97: Průběh vstupních signálů do datové paměti – procesor s čítačem instrukcí 80
Obrázek 98: Průběh řídicích signálů detekujících typ operace – procesor s čítačem instrukcí .. 80
Obrázek 99: Průběh výstupních signálů procesoru s čítačem instrukcí 81
Obrázek 100: Layout procesoru s čítačem instrukcí 82
Obrázek 101: Schéma zapojení procesoru bez čítače instrukcí 83
Obrázek 102: Přičítání 1 k instrukční adrese 83
Obrázek 103: Schéma paměti 1 bitu pro příští cyklus 84
Obrázek 104: Symbol paměti 1 bitu pro příští cyklus 84
Obrázek 105: Schéma paměti 4 bitů pro příští cyklus 84
Obrázek 106: Symbol paměti 4 bitů pro příští cyklus 84
Obrázek 107: Průběh výstupních signálů z datové paměti (A) – procesor bez čítače instrukcí .. 85
Obrázek 108: Průběh výstupních signálů z datové paměti (B) – procesor bez čítače instrukcí .. 85
Obrázek 109: Průběh vstupních signálů do datové paměti – procesor bez čítače instrukcí 86
Obrázek 110: Průběh řídicích signálů detekujících typ operace – procesor bez čítače instrukcí 86
Obrázek 111: Průběh výstupních signálů procesoru bez čítače instrukcí 87
Obrázek 112: Layout procesoru bez čítače instrukcí 88
Obrázek 113: Zapojení instrukční paměti Euklidova algoritmu 90
Obrázek 114: Schéma zapojení procesoru – Euklidův algoritmus 91
Obrázek 115: Vyhodnocení hodnoty příští instrukce – Euklidův algoritmus 91
Obrázek 116: Schéma detekce skoku – Euklidův algoritmus 92
Obrázek 117: Schéma zapojení A=B (8 bitů) 93
Obrázek 118: Symbol zapojení A=B (8 bitů) 93
Obrázek 119: Schéma zapojení A=B (4 bity) 93
Obrázek 120: Symbol zapojení A=B (4 bity) 93
Obrázek 121: Schéma zapojení A<B (8 bitů) 94

Obrázek 122: Symbol zapojení A<B (8 bitů) 94
Obrázek 123: Schéma zapojení A<B (4 bity) 94
Obrázek 124: Symbol zapojení A<B (4 bity) 95
Obrázek 125: Průběh výstupních signálů z datové paměti (A) – Euklidův algoritmus 96
Obrázek 126: Průběh výstupních signálů z datové paměti (B) – Euklidův algoritmus 96
Obrázek 127: Průběh vstupních signálů do datové paměti – Euklidův algoritmus 97
Obrázek 128: Průběh řídicích signálů detekujících typ operace – Euklidův algoritmus 97
Obrázek 129: Průběh výstupních signálů procesoru – Euklidův algoritmus 98
Obrázek 130: Layout procesoru – Euklidův algoritmus 99

1 Úvod

Tato diplomová práce volně navazuje na mou bakalářskou práci z roku 2017 [1]. Obě práce mají společné zaměření na možnost využití jako doplňkového materiálu při výuce předmětů se zaměřením na elektroniku pro studenty informatiky. Zatímco v bakalářské práci jsem kladla důraz hlavně na vysvětlení principů digitálního návrhu a tvorbu jednotlivých součástek, v této diplomové práci se zabývám implementací komplexního integrovaného obvodu v podobě procesoru. Nezbytnou součástí je propojení elektroniky s ostatními tématy, se kterými se student informatiky reálně během svého studia na fakultě setká – architektura počítačů, konečné automaty a jiné.

Při implementaci jsem stejně jako v předchozí práci postupovala hierarchicky od jednoduchých po komplexní obvody a funkčnost každého bloku dostatečně popsala a otestovala.

Sama jsem si prošla oběma etapami studijního programu Otevřená informatiky na FEL ČVUT – bakalářským oborem Počítačové systémy a magisterským oborem Počítačové inženýrství. Při tvorbě práce jsem si dala za cíl se zaměřit na klíčové oblasti obou studií a vytvořit práci, která by mohla pomoci ostatním studentům se zájmem o informatiku, elektroniku a jejich vzájemné propojení. Z tohoto důvodu se mohou zdát upozaděna jiná neméně zajímavá a důležitá témata, která ale nereflektují potřeby cílového čtenáře.

Cíl práce není vytvoření procesoru, který obsahuje velké množství složitých instrukcí, ale naopak srozumitelně ukázat, že i tak obsáhlá jednotka jako procesor se ve skutečnosti skládá z částí, které nejsou složité na pochopení. Proto jsem kladla důraz na rozložení návrhu do srozumitelného blokového schématu, kde každý blok tvoří samostatně fungující jednotku a jednotlivé bloky mezi sebou vzájemně interagují a předávají si informace. Zároveň jsem vymyslela vlastní sadu instrukcí podporovaných procesorem takových, aby nebyly složité na pochopení čtenářem a zároveň by bylo možné s nimi vytvořit komplexní program.

Ve své práci předpokládám základní znalosti principů elektronického návrhu a Booleovské algebry, které čtenář získal například v předmětech bakalářské etapy studia. Pokud však tyto znalosti chybí, mohu vřele doporučit například online tutoriály Massachusetts Institute of Technology (MIT) nebo záznamy jejich přednášek na Youtube (např. [2], [3]) zápis předmětů na FEL jako jsou Struktury integrovaných systémů [4] nebo Integrované systémy na čipu [5], popř. dohledat si informace ve zmíněné bakalářské práci či jiné oborné literatuře vlastní volby.

1.1 Vývojové prostředí

Všechny schémata a simulace jsou vytvořeny pod školní licencí vývojového prostředí Cadence Virtuoso v knihovně gpdk090 obsahující tranzistory v 90nm technologii. Schémata jsou vytvořena ve schematic, poté je vytvořen symbol součástky, a nakonec je chování součástky odsimulováno v testovacím zapojení v ADE L.

Konkrétní zapojení simulace je u většiny součástek podobné. Na vstupy jsou připojeny zdroje pulsního napětí vpulse z knihovny analogLib. Tyto zdroje generují obdélníkový signál, který je možný přizpůsobit vlastním potřebám – nastavit periodu signálu, zpoždění, úrovně napěti, délku nástupné a sestupné hrany, délku pulsu apod. Výstupy ze součástky jsou naopak připojené na kondenzátor, aby bylo možné změřit výstupní napětí.

Automatický layout finálních součástek byl vytvořen v Layout XL.

2 TEORETICKÝ ROZBOR

Pro správné fungování počítače, konkrétně procesoru, je nutné si přesně definovat své požadavky. Podle nich se pak bude odvíjet jeho architektura, tedy fyzické a logické propojení jednotlivých funkčních bloků.

Počítače můžeme rozdělit do skupin hned podle několika kritérií.

První dělení je dle typu architektury procesoru na 2 základní typy: von Neumann a Harvard. Procesor klasické architektury typu von Neumann se skládá z ALU zodpovědné za vykonání instrukcí, řadiče (control unit) zodpovědného za rozvrhování vykonání instrukcí, vstupních a výstupních periférií a paměti k uložení dat a instrukcí. [6] Tato architektura má společnou instrukční a datovou paměť. Z tohoto důvodu nelze instrukce a data přenášet současně po sběrnici. [7]

Harvardská architektura navazuje na von Neumannovskou, na rozdíl od ní má ale oddělenou instrukční a datovou paměť. Proto lze instrukce číst současně s přenosem dat a šířka přenosové cesty instrukcí může být jiná než šířka přenosové cesty dat. [7]

Další dělení je dle typu použitého procesoru na CISC a RISC. CISC je původní technologie rozšířená hlavně v počátcích počítačů. Obsahuje rozsáhlý soubor instrukcí a složité operace se dějí na úrovni HW, což umožňuje jednoduché přidávání podpory nových instrukcí. Přestože CISC procesor může obsahovat nadbytečné instrukce, poskytuje lepší překlad z instrukcí vysokoúrovňových programovacích jazyků na strojové instrukce. To vše je za cenu vyšší složitosti, nižší rychlosti a potřeby větší paměti k uložení instrukce. [6] [8]

Původně byly počítače pouze typu CISC a implementaci instrukcí zjednodušil mikroprogramový řadič. Takový řadič byl ale složitější a pomalejší než klasický obvodově řešený – 20 % instrukcí využívá 80 % času. Počítače typu RISC s redukovaných souborem instrukcí obsahují pouze nezbytné základní operace a složité operace se řeší na úrovni SW. [9] Výhodou této technologie oproti CISC je, že všechny instrukce mají stejnou délku a jejich vykonání trvá stejnou dobu. [6]

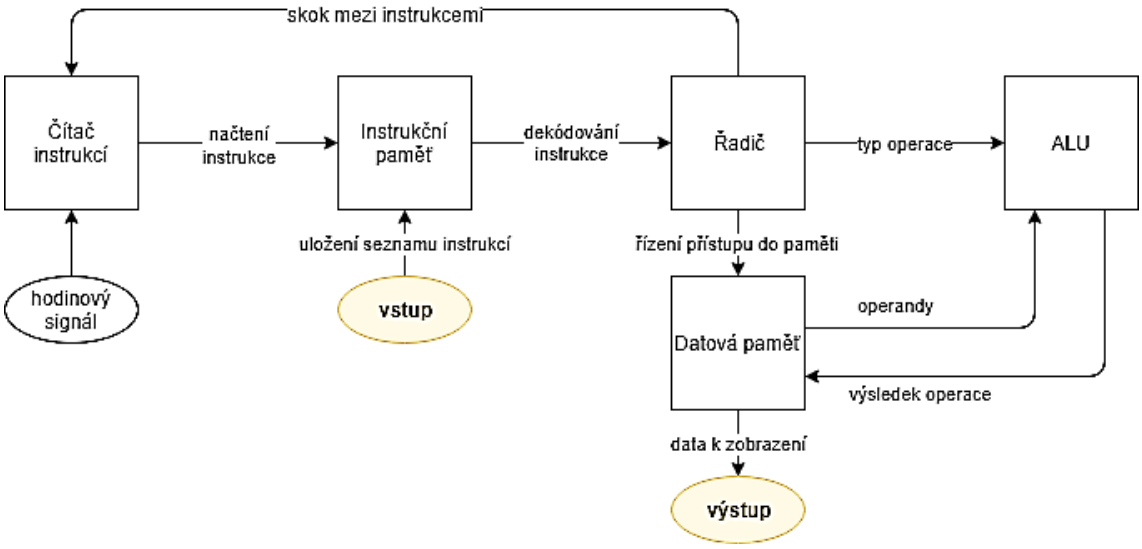
Dnešní počítače využívají kombinaci RISC a CISC a nelze je zařadit ani do jedné kategorie. [8]

Práce procesoru je řízena instrukcemi. Pojem instrukční sada (ISA) se používá pro definici programově viditelných instrukcí v procesoru a tvoří tak hranici mezi hardwarem a softwarem. Podle přístupu jednotlivých instrukcí do paměti je možné architekturu procesorů rozdělit na 2 skupiny: register-memory, ve které může být instrukce přímý přístup do paměti, a load-store, ve které je přístup do paměti možný pouze pomocí instrukcí *load* a *store*. Použití modelu load-store je v procesoru velmi populární. Například v MIPS64 tak pracuje procesor s 64-bitovými celými čísly v pevné řádové čárce (integers) a 32-bitovými nebo 64-bitovými čísly v plovoucí řádové čárce (floating-point). [8]

Zpracování instrukcí má obvykle 5 základních fází: uvolnění instrukce (instruction fetch), dekódování instrukce (instruction decode), uvolnění operandů (operand fetch), vykonání operace (execute operation) a uložení výsledku (result store). [9]

3 BLOKOVÉ SCHÉMA PROCESORU

Pro návrh procesoru je potřeba si předem rozvrhnout jeho schéma stávající se z jednotlivých bloků, jejich fyzické a logické propojení. Je nutné, aby každý blok zastával svou specifickou funkci a zároveň, aby práce všech bloků byla synchronizována v čase. V tom případě bude procesor produkovat správné, předvídatelné a opakovatelné výsledky.



Obrázek 1: Blokové schéma procesoru

Implementovaný procesor se skládá z pěti hlavních částí, kterými jsou:

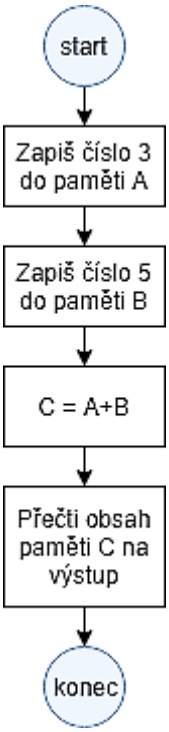
- čítač instrukcí
- instrukční paměť
- řadič
- ALU
- datová paměť

Před zahájením jakýchkoliv výpočetních operací je nutné uložit do instrukční paměti seznam zakódovaných instrukcí, který je po celou dobu běhu programu jednotný a neměnný. Při běhu programu plní čítač instrukcí funkci ukazatele na právě zpracovávanou instrukci. Instrukce je načtena z instrukční paměti a v řadiči je dekódován její typ.

Procesor pracuje na principu load-store architektury, což znamená, že instrukční sadu lze rozdělit na dva typy instrukcí: přístup do paměti a ALU operace. Při zpracování datových instrukcí dochází k interakci mezi uživatelským vstupem/výstupem pomocí dvou operací: *STORE*, tedy zápisu uživatelských dat na zvolenou adresu v paměti, a *LOAD*, tedy přečtení obsahu dané adresy v datové paměti na výstup. Všechny ostatní typy operací pracují již pouze v rámci procesoru a s daty uloženými v datové nebo instrukční paměti. Nedochází tedy k jakékoliv další interakci s uživatelem.

To má za důsledek, že konkrétní matematická operace nikdy nebude probíhat mezi dvěma konkrétními čísly, ale vždy mezi dvěma místy v paměti. Na obr. 2 je znázorněna ukázka pořadí

operací při sčítání dvou čísel (3+5). Je nezbytné, aby čísla byla nejprve uložena do paměti, poté teprve může dojít k operaci sčítání, výsledek je opět uložen do paměti, z které může být vyveden na výstup.



Obrázek 2: Ukázka pořadí operací prováděných při sčítání dvou čísel

3.1 SADA INSTRUKCÍ

Procesor umí zpracovat vlastní sadu 16-bitových instrukcí. Každá binárně zakódovaná instrukce začíná 4-bitovým opcode (kódem instrukce), který jednoznačně identifikuje její typ. Následujících 12 bitů se může lišit dle typu instrukce. V následující tabulce je shrnuta instrukční sada podporovaná procesorem a vysvětlení tvaru jednotlivých instrukcí.

Tabulka 1: Kódy jednotlivých operací a jejich význam

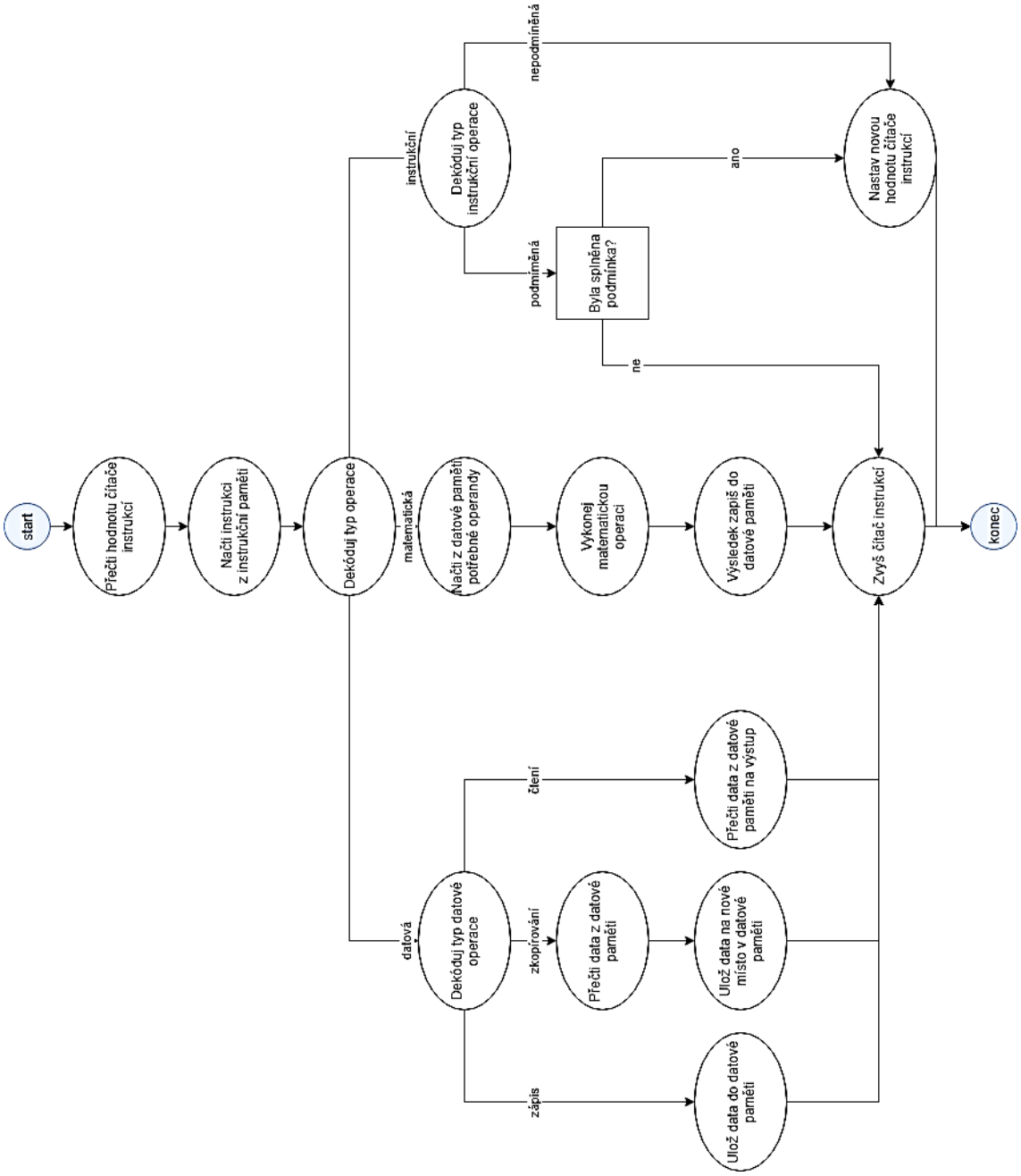
opcode[3:0]	operace	tvar instrukce	vysvětlení
0000	--	0000 ---- ---- ----	*neimplementováno*
0001	ADD	0001 XXXX YYYY ZZZZ	MEM[X] = MEM[Y] + MEM[Z]
0010	SUB	0010 XXXX YYYY ZZZZ	MEM[X] = MEM[Y] – MEM[Z]
0011	SHIFT LOGIC LEFT	0011 XXXX YYYY 0000	MEM[X] = MEM[Y] << 1
0100	SHIFT LOGIC RIGHT	0100 XXXX YYYY 0000	MEM[X] = MEM[Y] >> 1
0101	AND	0101 XXXX YYYY ZZZZ	MEM[X] = MEM[Y] & MEM[Z]
0110	OR	0110 XXXX YYYY ZZZZ	MEM[X] = MEM[Y] | MEM[Z]
0111	XOR	0111 XXXX YYYY ZZZZ	MEM[X] = MEM[Y] ^ MEM[Z]
1000	--	1000 ---- ---- ----	*neimplementováno*
1001	LOAD	1001 0000 XXXX 0000	out = MEM[X]
1010	STORE	1010 XXXX YYYY YYYY	MEM[X] = Y
1011	MOVE	1011 XXXX YYYY 0000	MEM[X] = MEM[Y]
1100	JUMP	1100 XXXX 0000 0000	PC = X
1101	JUMP IF EQUAL	1101 XXXX YYYY ZZZZ	if (MEM[Y] == MEM[Z]) then PC = X
1110	JUMP IF LESS	1110 XXXX YYYY ZZZZ	if (MEM[Y] < MEM[Z]) then PC = X
1111	--	1111 ---- ----	*neimplementováno*

Procesor musí umět nejen rozlišit jednotlivé instrukce podle jeho opcode, ale zároveň musí instrukce správně zpracovat a zajistit, aby všechny jeho součásti spolupracovaly. Například z paměti musí být přečteny uložené hodnoty dle adresy paměťové buňky, matematické operace musí být v ALU správně vypočteny nebo musí být správně vyhodnoceny podmínky, zda má dojít k instrukčnímu skoku. Takovýto rozhodovací proces shrnuje následující obr. 3.

Procesor rozeznává celkem 3 typy operací. Prvním typem jsou datové operace, tedy operace, které přímo interagují s datovou pamětí. Konkrétně se jedná o instrukci *STORE*, při které dochází k uložení do datové paměti čísla, jenž je součástí instrukce. Další takovou instrukcí je *LOAD*, která data uložená v paměti přivede na výstup. Poslední datovou instrukcí je *MOVE*, která překopíruje data z jedné paměťové buňky do jiné.

Dalším typem operací jsou matematické, tedy aritmetické a logické operace, při kterých dochází k interakci s ALU. Na vstupu je jeden nebo dva operandy, dle konkrétní instrukce, tyto operandy jsou přivedeny na vstup ALU a výstup z ALU je uložen do paměti. Mezi tyto instrukce patří sčítání, odčítání, AND, OR, XOR a bitové posuny.

Posledním typem operací jsou instrukční. Při nich může dojít ke změně čítače instrukcí v závislosti na tom, zda je operace nepodmíněná či podmíněná a zda byla podmínka splněna.



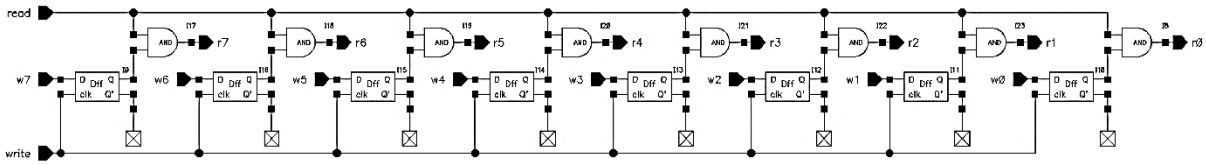
Obrázek 3: Proces zpracování instrukcí

4 DATOVÁ PAMĚŤ

Jednou ze základních jednotek procesoru je datová paměť, tedy struktura, ve které je možné uchovat vstupní hodnoty, mezivýsledky či výstupy z jednotlivých operací. Implementovaná paměť je v tomto procesoru 8-bitová s 4-bitovým adresováním, je tedy možné ukládat celkem 16 osmibitových hodnot do vzájemně nezávislých paměťových jednotek.

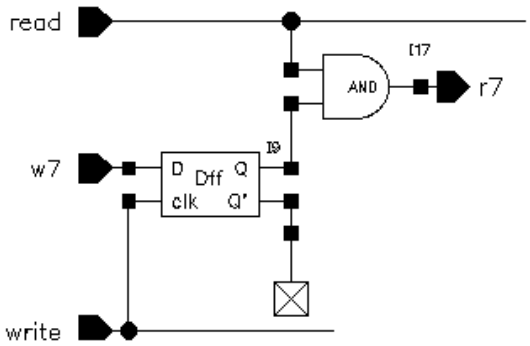
4.1 PAMĚŤOVÁ JEDNOTKA

Datová paměť procesoru je složená z paměťových jednotek. Jednotlivé paměťové jednotky se skládají z osmi D klopných obvodů (DFF), které uchovávají hodnotu z osmi vstupních datových pinů *w[7:0]* a převádějí je na výstup *r[7:0]*. Procesy ukládání do paměti a čtení z paměti jsou řízené řídicími signály *read* a *write*. Sestavení a princip fungování DFF je vysvětlen v mé bakalářské práci [1], proto není v této diplomové práci zahrnut.



Obrázek 4: Schéma paměťové jednotky

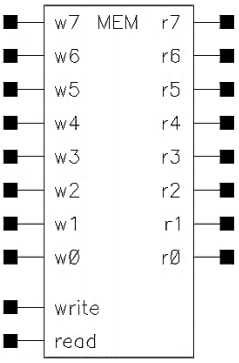
Paměťová jednotka je struktura, do které je možno uložit 8-bitové číslo. Procesy čtení a zápisu jsou synchronizované pro celou jednotku, není tedy možné ukládat nebo modifikovat bity jednotlivě, vždy pouze celý byte najednou. Stejně tak dochází vždy ke čtení celého bytu, není nijak možné přečíst pouze některé bity.



Obrázek 5: Detail paměťové jednotky

Na obr. 5 je znázorněn detail paměťové jednotky pro uložení nejvyššího bitu. Je vidět, že centrální částí je DFF, na jehož vstup je přivedena hodnota *w[7]*, kterou chceme do paměti zapsat. Obecně při zápisu do DFF dochází s náběžnou hranou hodin (*clk*). Na vstup znázorňující hodinový signál je tedy přiveden signál *write* a při náběžné hraně tohoto signálu dojde k zápisu hodnoty do paměti. Poté, co byla hodnota do paměti zapsána, je možné jí z paměti i přečíst (*Q*). K tomu dojde pouze při kladné logické hodnotě signálu *read.* Proces čtení a zápisu do paměti se tedy liší v tom, že zatímco proces zápisu je hranový a dojde k němu tedy při náběžné hraně, proces čtení je úrovňový a hodnota se zobrazí na výstupu po celou dobu, kdy signál *read* má úroveň log. 1. Negovaný výstup klopného obvodu (*Q'*) nemá v paměťové buňce žádnou roli, proto je připojen do noConn symbolu z knihovny basic, což značí, že výstup je nepřipojen a ignorován.

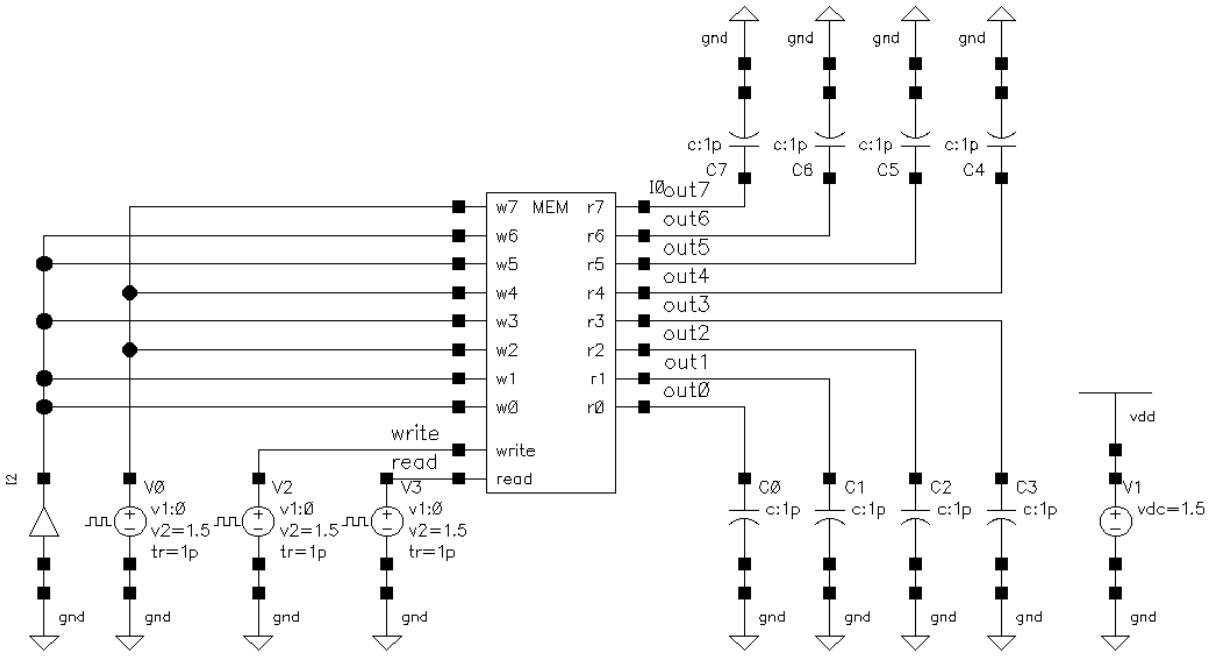
Pro otestování součástky ve schématech je nutné vytvořit její symbol, který reprezentuje součástku jako celek.



Obrázek 6: Symbol paměťové jednotky

Simulace funkčnosti paměťové jednotky je jedna z nejdůležitějších v celé práci, protože procesor čerpá data výhradně z datové paměti. Pro správnou funkčnost procesoru je tedy nezbytné mít jistotu, že všechna uložená data jsou platná a aktuální.

Postup následujících simulací je podobný – na vstupy přivedeny data, poté jsou pomocí signálu *write* zapsána a následně jsou pomocí signálu *read* přečtena a tím jsou přivedena na výstup.



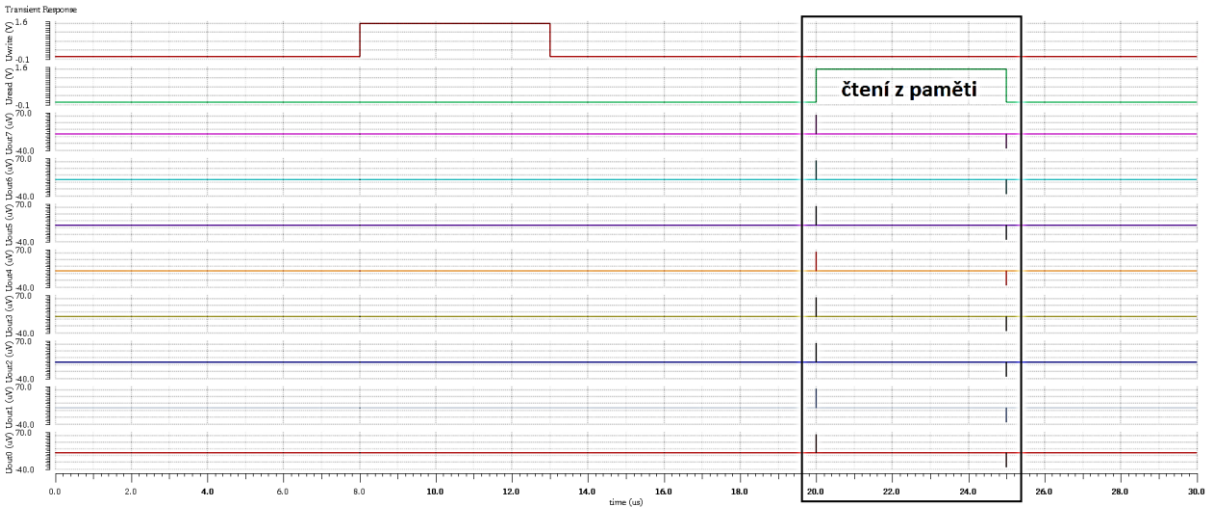
Obrázek 7: Schéma simulace paměťové jednotky

Nejprve je nutno ověřit funkčnost operací čtení a zápisu. V první simulaci jsou data přivedena na vstupy až po náběžné hraně signálu *write*. K zápisu zamýšlených dat tedy nedojde, protože i když jsou data přítomna po dobu, kdy má signál *write* hodnotu log. 1, nejsou přítomna v kritickou chvíli, tedy právě při náběžné hraně. Časování signálů je následující:

1. V čase t_1 = 10 µs jsou na vstupy přivedena data *w[7:0]*, délka pulsu je 10 µs a délka náběžné i sestupné hrany je 1 ps.
2. V čase t_2 = 8 µs je nastaven signál *write* na hodnotu log. 1, délka pulsu je 5 µs a délka náběžné i sestupné hrany je 1 ps. Jelikož k zapsání hodnoty do registru dochází

s náběžnou hranou, v čase t_2 nejsou na vstupech *w[7:0]* přítomna data z kroku 1 a k jejich zápisu nedojde.

3. V čase t_3 = 20 µs je nastaven signál *read* na hodnotu log. 1, délka pulsu je 5 µs a délka náběžné i sestupné hrany je 1 ps. Díky tomu dojde k propsání hodnot uložených v paměťové jednotce na výstup v čase t_3.



Obrázek 8: Průběh simulace paměťové jednotky 1

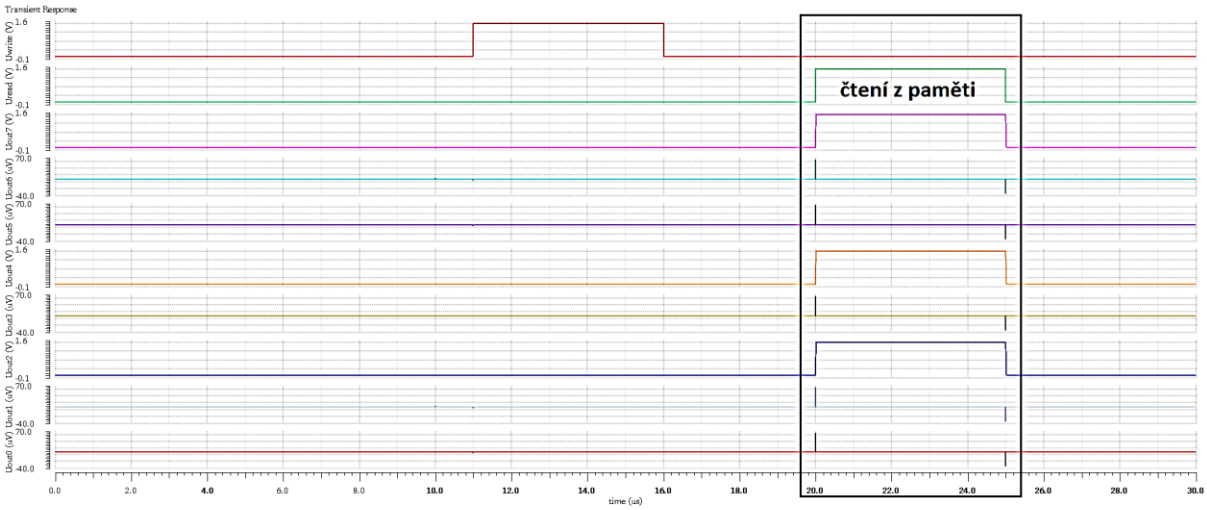
Na obr. 8 jsou zobrazeny následující signály (ze shora dolů): *write*, *read*, *out7* až *out0*.

Z průběhu signálů na výstupech je vidět, že všechny výstupy mají hodnotu log. 0. K zápisu hodnot na vstupech totiž nedošlo, protože data nebyla na vstupech přítomna v okamžiku náběžné hrany.

V další simulaci již předchází přivedení dat na vstupy náběžné hraně signálu *write*. Časování signálů je následující:

1. V čase t_1 = 10 µs jsou na vstupy přivedena data *w[7:0]*, délka pulsu je 10 µs a délka náběžné i sestupné hrany je 1 ps.
2. V čase t_2 = 11 µs je nastaven signál *write* na hodnotu log. 1, délka pulsu je 5 µs a délka náběžné i sestupné hrany je 1 ps. Jelikož k zapsání hodnoty do registru dochází s náběžnou hranou, v čase t_2 jsou na vstupech *w[7:0]* přítomna data z kroku 1 a k jejich zápisu dojde.
3. V čase t_3 = 20 µs je nastaven signál *read* na hodnotu log. 1, délka pulsu je 5 µs a délka náběžné i sestupné hrany je 1 ps. Díky tomu dojde k propsání hodnot uložených v paměťové jednotce na výstup v čase t_3.

Na obr. 9 jsou opět viděny výstupní signály *write*, *read* a *out7* až *out0*.



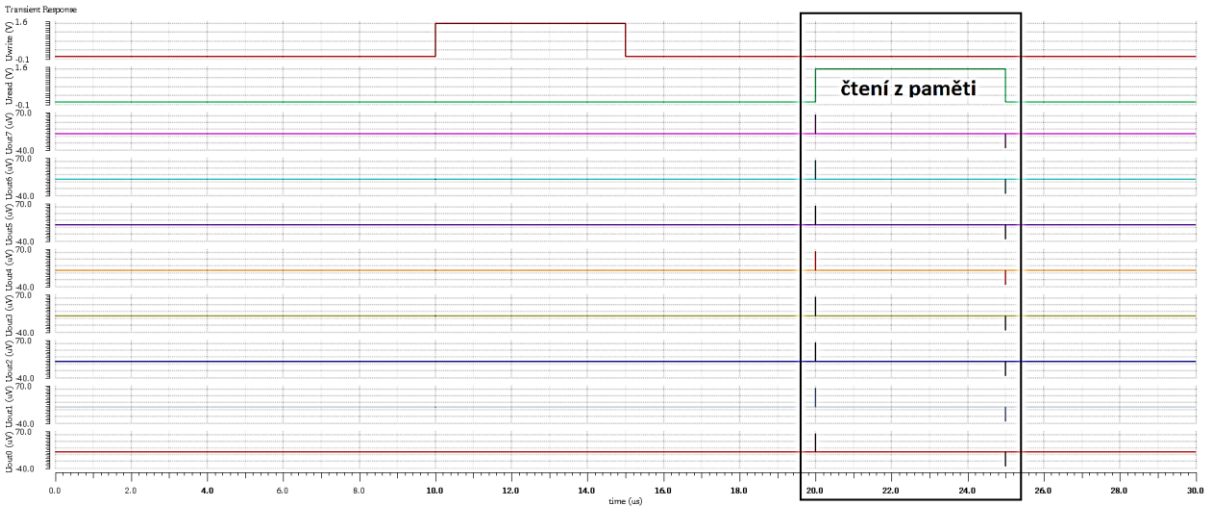
Obrázek 9: Průběh simulace paměťové jednotky 2

Z průběhu signálů je vidět, že na všech výstupech jsou po celou dobu zápisu napěťové úrovně log. 0, až do chvíle, než dojde ke čtení z paměti. To dokazuje, že data ze vstupů byla opravdu zapsána na náběžnou hranu signálu *write* do paměti.

Poslední možná situace, která může nastat je v případě, že náběžné hrany vstupních dat *w[7:0]* a řídicího signálu *write* se překrývají. Časování signálů je pak následující:

1. V čase t_1 = 10 µs jsou na vstupy přivedena data *w[7:0]*, délka pulsu je 10 µs a délka náběžné i sestupné hrany je 1 ps.
2. Zároveň, v čase t_1 = 10 µs je nastaven signál *write* na hodnotu log. 1, délka pulsu je 5 µs a délka náběžné i sestupné hrany je 1 ps. Jelikož k zapsání hodnoty do registru dochází s náběžnou hranou, v čase t_2 nejsou na vstupech *w[7:0]* přítomna data z kroku 1 a k jejich zápisu nedojde.
3. V čase t_3 = 20 µs je nastaven signál *read* na hodnotu log. 1, délka pulsu je 5 µs a délka náběžné i sestupné hrany je 1 ps. Díky tomu dojde k propsání hodnot uložených v paměťové jednotce na výstup v čase t_3.

Na obr. 10 jsou opět zobrazeny výstupní signály v pořadí *write*, *read*, *out7* až *out0*.



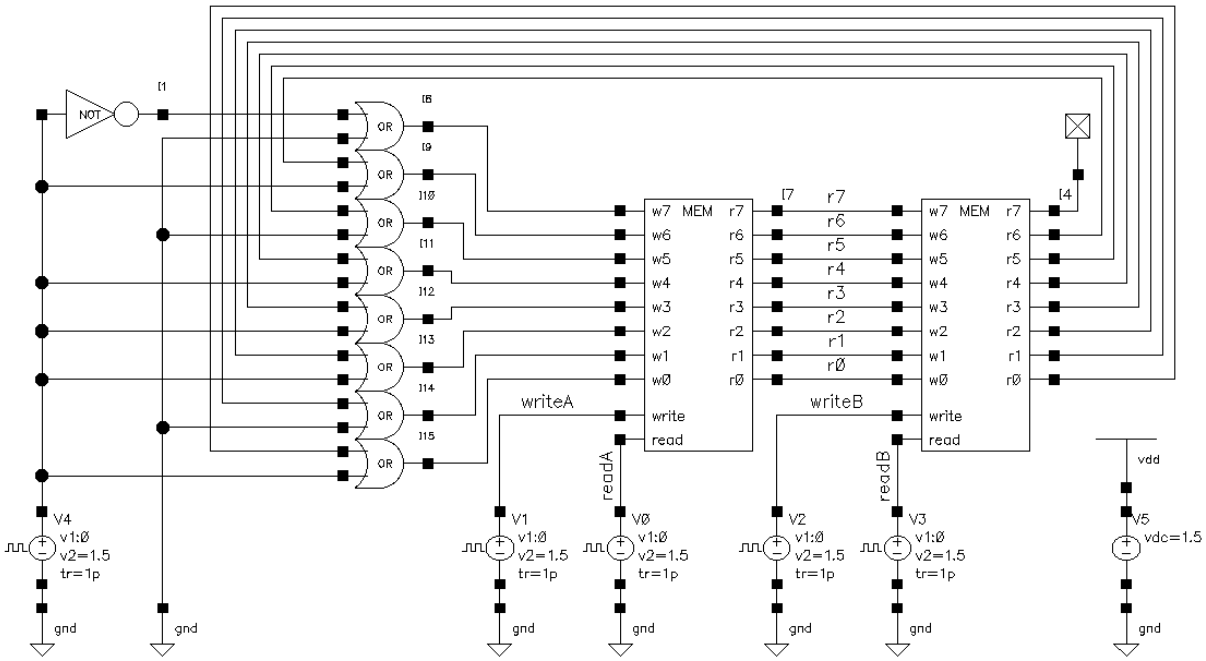
Obrázek 10: Průběh simulace paměťové jednotky 3

Z průběhu signálů na obr. 10 je vidět, že k zapsání vstupních dat opravdu nedošlo. Je tedy nezbytné, aby data k zapsání byla v okamžiku nástupné hrany signálu *write* přítomna na vstupech.

4.1.1 Read-Modify-Write

Protože paměťová jednotka uchovává všech 8 bitů najednou, není možné změnit jednotlivé bity zvlášť. Pro změnu pouze jednoho či více bitů v konkrétním registru je nutné použít systém Read-Modify-Write (RMW). To znamená, že obsah paměťové jednotky je přečten do jiné, upraven a poté opět zapsán do původní jednotky.

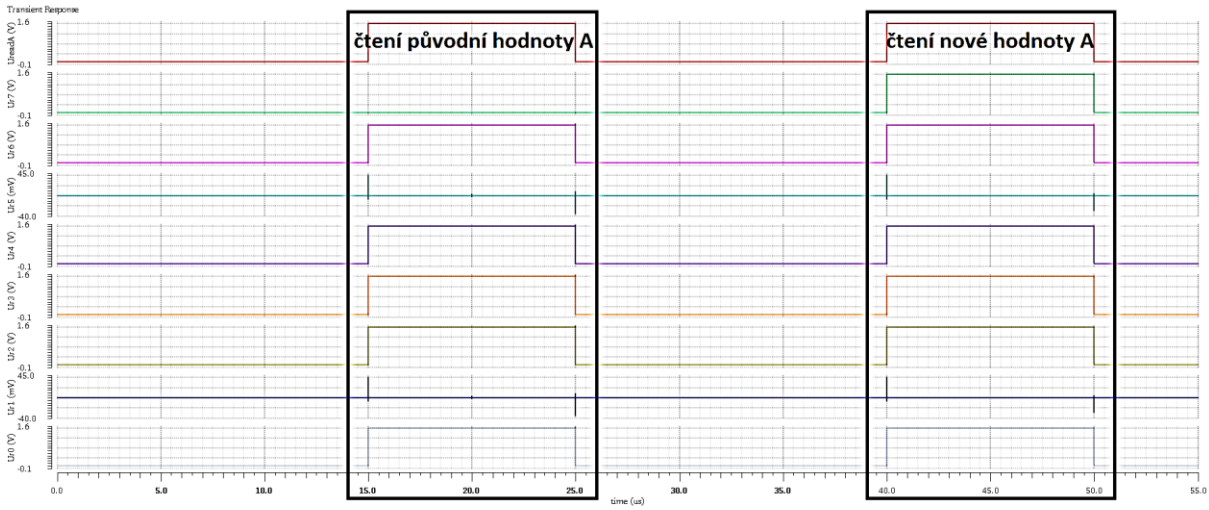
Teoreticky by bylo možné data upravit bez prostředníka, tedy paměťové jednotky navíc. V případě této simulace je však použita, aby se dal vysledovat přesný tok dat.



Obrázek 11: Schéma simulace Read-Modify-Write

Pro jasnost je levá paměťová jednota obsahující původní data pojmenována A, druhá pomocná paměťová jednotka je pojmenována B. V případě této simulace je časování následující:

1. V čase t_1 = 0 µs jsou na vstup A přivedena původní data.
2. V čase t_2 = 5 µs jsou data zapsána do A. V tomto okamžiku má jednotka původní hodnotu, kterou chceme změnit.
3. V čase t_3 = 15 µs jsou data přečtena z A.
4. V čase t_4 = 20 µs jsou zapsána do B.
5. V čase t_5 = 25 µs jsou data přečtena z B a modifikována.
6. V čase t_6 = 30 µs jsou modifikovaná data zapsána zpět do A.
7. V čase t_7 = 40 µs jsou data přečtena z A, čímž je ověřeno, že data byla opravdu pozměněna.



Obrázek 12: Průběh simulace Read-Modify-Write

Důležitý poznatek je, že ani při použití tohoto přístupu dvou různých paměťových jednotek není možné v jednom okamžiku zahájit čtení dat z jedné a zároveň zápis dat do jiné. Je to z toho důvodu, že při zápisu do paměti s náběžnou hranou musí být data bezpodmínečně připravena na vstupech a to nejsou.

4.2 DATOVÁ PAMĚŤ PROCESORU

4.2.1 Dekodér adresy

Obecně je dekodér digitální obvod, který detekuje přítomnost specifické vstupní kombinace a indikuje přítomnost specifické výstupní kombinace. Dekodér obsluhuje n vstupů a 1 až 2n výstupů, na kterých indikuje přítomnost jedné nebo více n-bitových kombinací. [10]

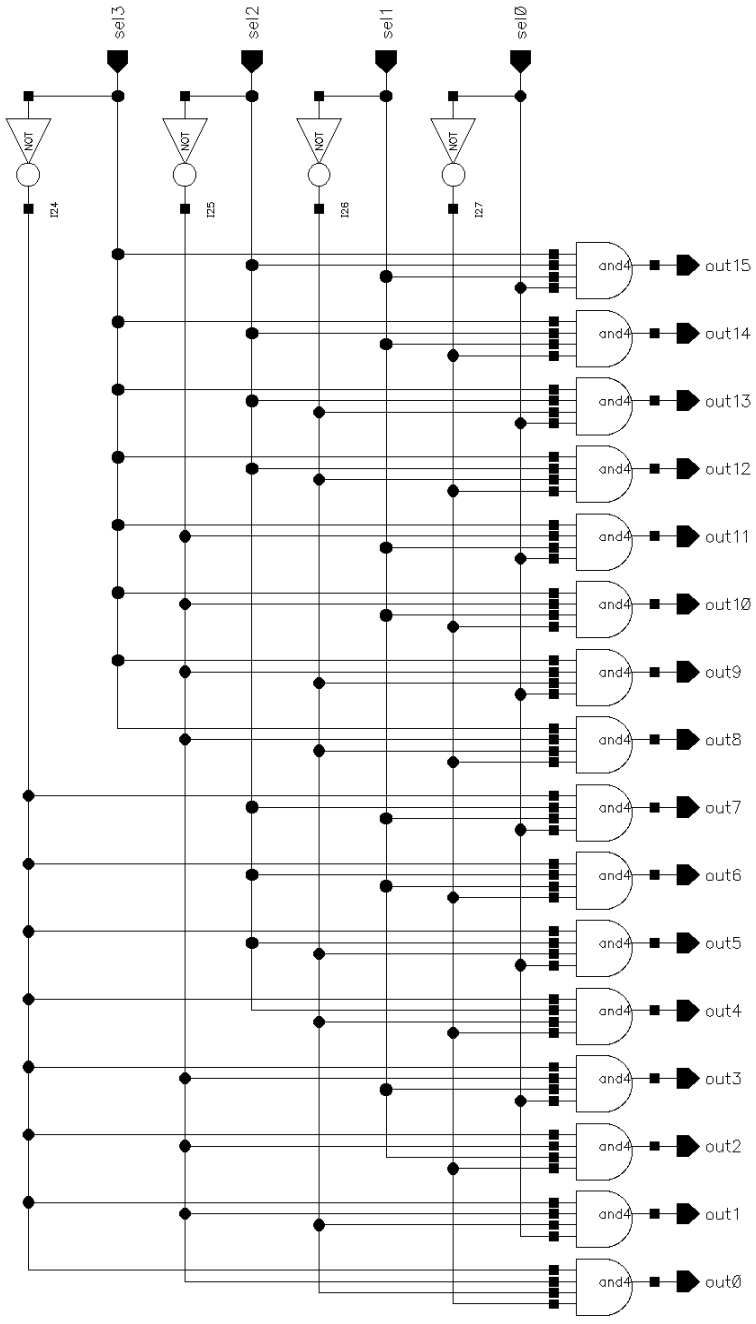
Pro jednoznačnou identifikaci paměťové buňky je jednotlivým registrům přiřazena unikátní čtyřbitová adresa. Tato adresa je dekódována pomocí tzv. one-hot binárního dekodéru, tedy typu dekodéru, ve kterém má vždy právě jeden výstup hodnotu logické 1, zatímco všechny ostatní výstupy mají hodnotu logické 0. Všechny kombinace výstupních hodnot jsou uvedeny v tabulce 2.

Pojmenování výstupů v tomto dekodéru odráží jejich chování. Signál out_i má hodnotu log. 1 právě tehdy, když vstupní adresní kombinace má dekadickou hodnotu i.

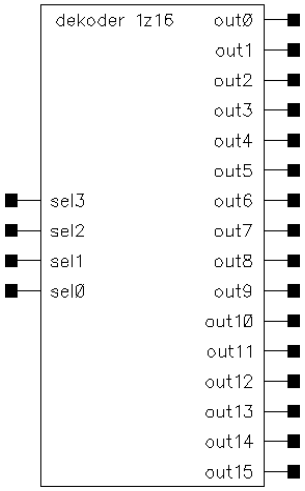
Tabulka 2: Pravdivostní tabulka adresního dekodéru

vstupy				výstupy															
addr3	addr2	addr1	add0	out15	out14	out13	out12	out11	out10	out9	out8	out7	out6	out5	out4	out3	out2	out1	out0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
0	1	0	1	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
0	1	1	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
0	1	1	1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
1	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
1	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
1	0	1	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
1	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
1	1	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
1	1	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Schéma zapojení tohoto dekodéru je přesně podle tabulky 2. Na vstupu jsou celkem 4 adresní vodiče a tedy možných $2^4 = 16$ vstupních kombinací. Každý výstupní pin má hodnotu log. 1 pouze při jedné určité logické kombinaci a zároveň v jeden časový okamžik má hodnotu log. 1 právě jeden výstupní pin.



Obrázek 13: Schéma dekodéru 1 z 16

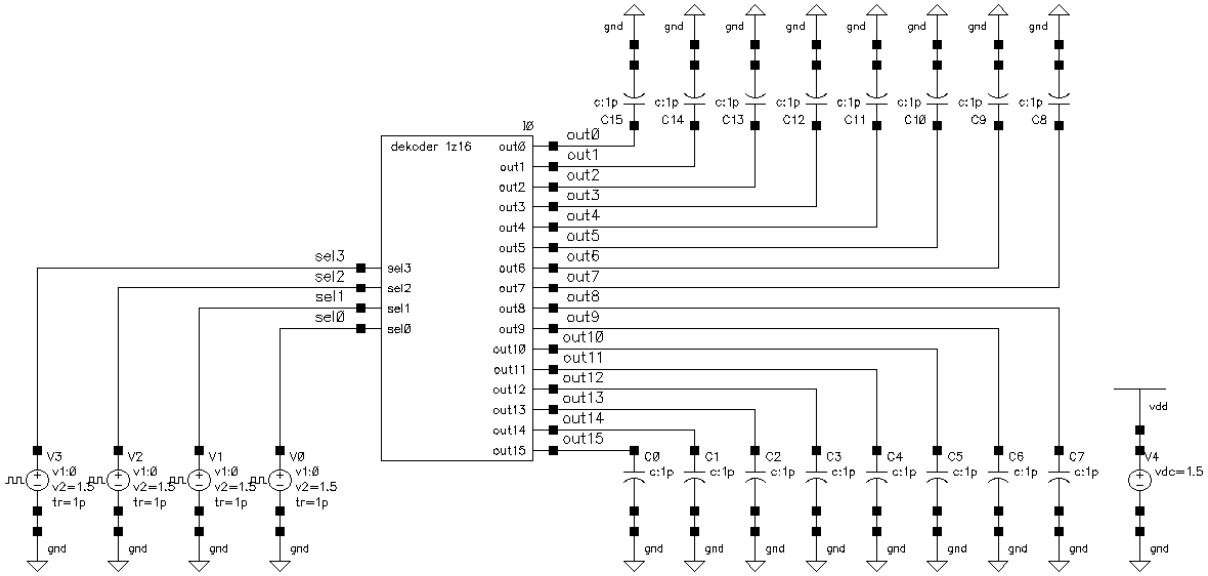


Obrázek 14: Symbol dekodéru

V případě dekodéru je postup simulace poměrně evidentní. Na vstup jsou postupně přivedeny všechny kombinace adres a jsou sledovány hodnoty na výstupu.

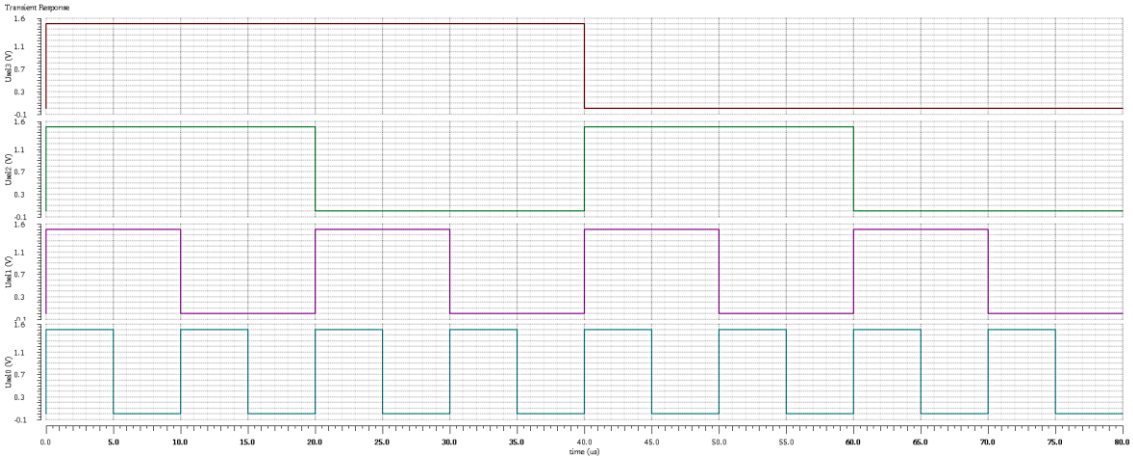
Tyto kombinace jsou vygenerovány následujícími zdroji pulsního napěti:

- sel3: délka periody 80 µs, délka pulsu 40 µs, délka náběžné i sestupné hrany 1 ps
- sel2: délka periody 40 µs, délka pulsu 20 µs, délka náběžné i sestupné hrany 1 ps
- sel1: délka periody 20 µs, délka pulsu 10 µs, délka náběžné i sestupné hrany 1 ps
- sel0: délka periody 10 µs, délka pulsu 5 µs, délka náběžné i sestupné hrany 1 ps



Obrázek 15: Schéma simulace dekodéru 1 z 16

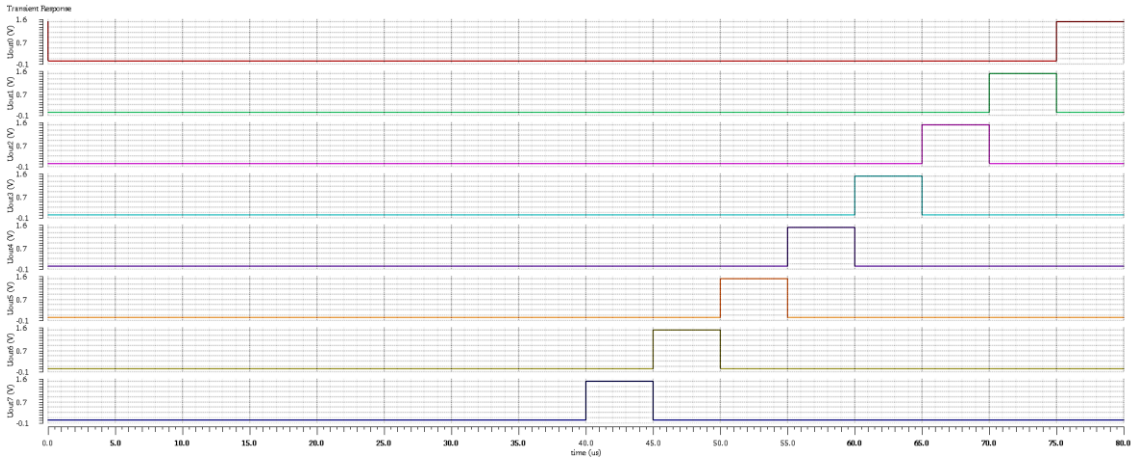
Na obr. 16 jsou zobrazeny všechny vygenerované kombinace vstupního napětí na vodičích sel3 až sel0.



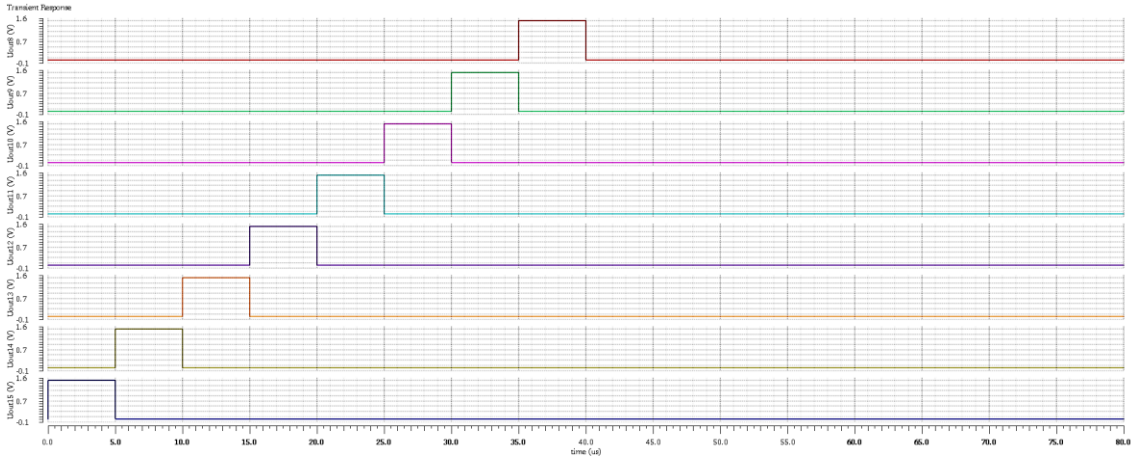
Obrázek 16: Průběh simulace dekodéru 1 z 16 - vstupní signály

Z průběhu je patrné, že v simulaci je za hodnotu sel dosazena hodnota sestupně od 15 do 0.

Z následujících průběhu simulace na obr. 17 a obr. 18 je patrné, že v jeden časový okamžik má právě jeden výstupní pin úroveň log. 1. Průběh signálů tedy přesně odpovídá chování popsanému v tabulce 2.



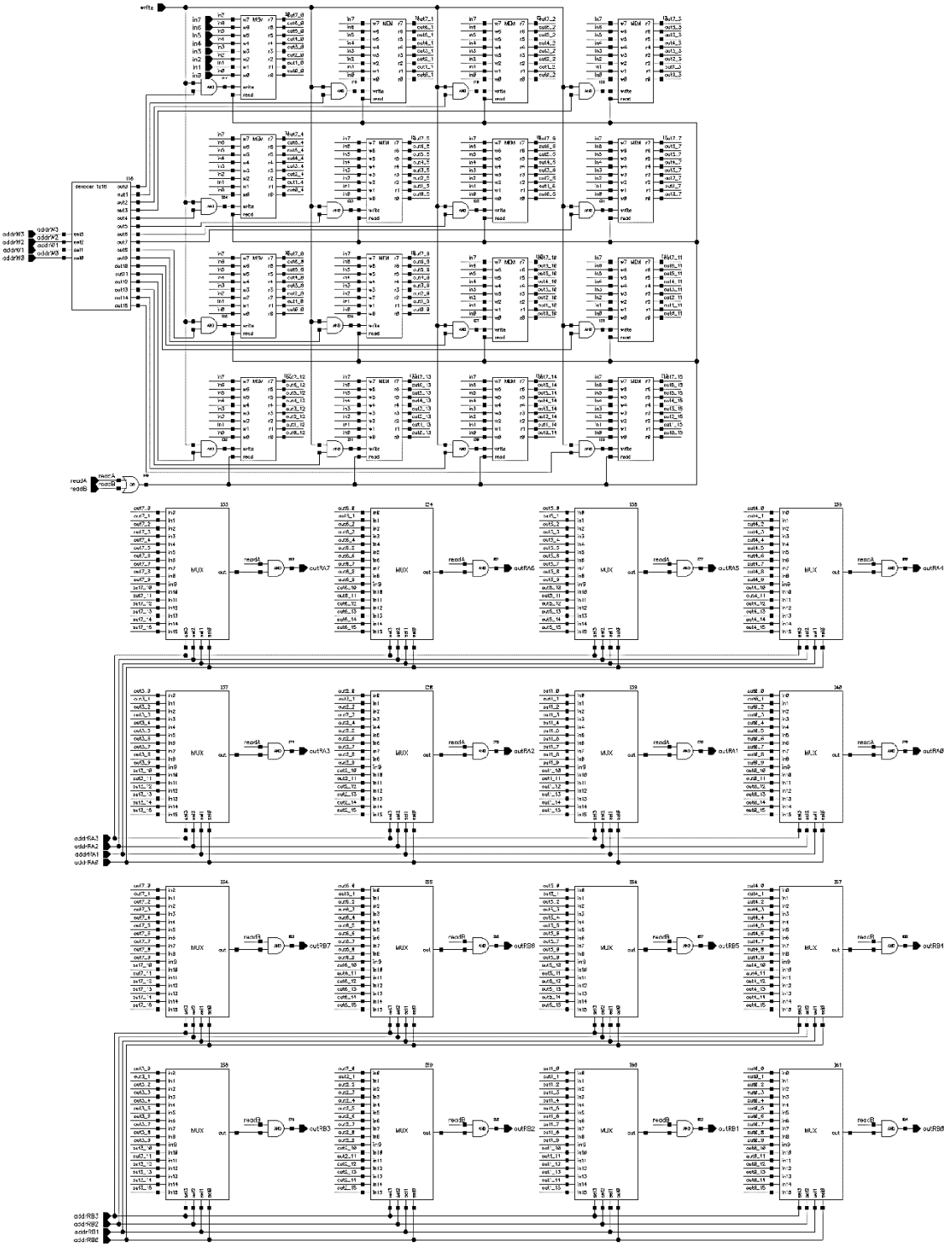
Obrázek 17: Průběh simulace dekodéru 1 z 16 - out0 až out7



Obrázek 18: Průběh simulace dekodéru 1 z 16 - out8 až out15

4.2.2 Zapojení datové paměti

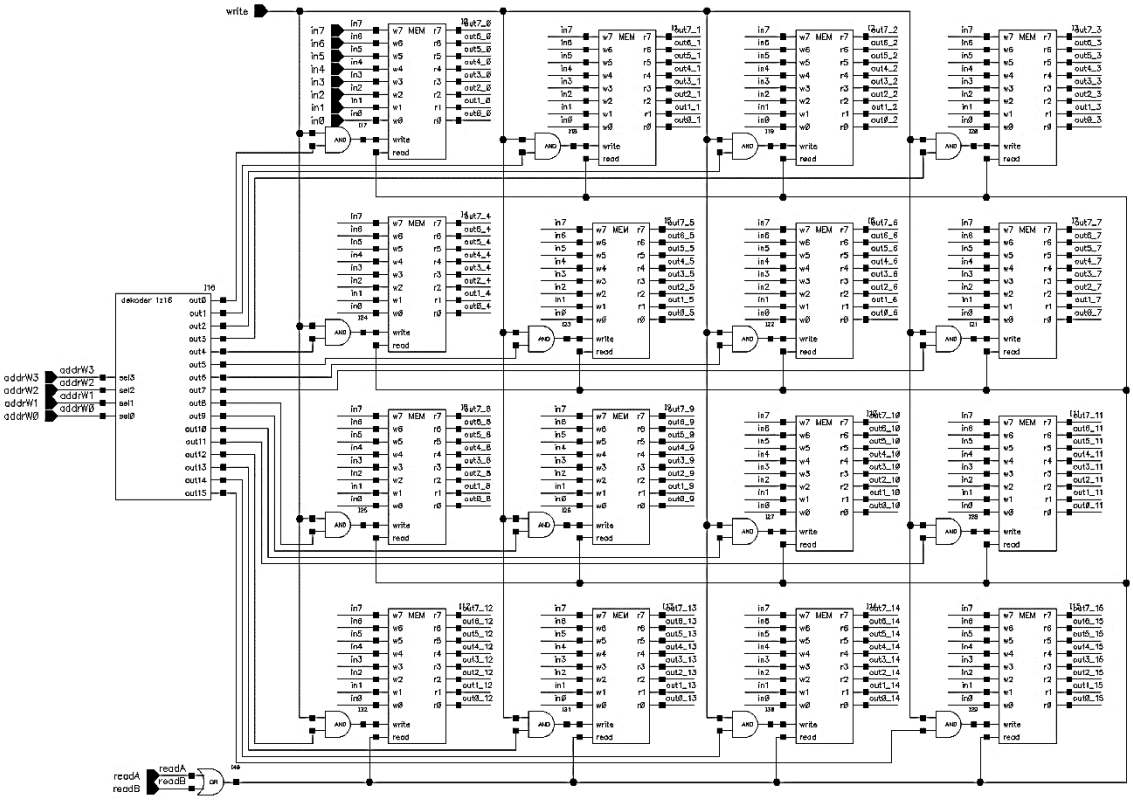
Pro práci procesoru je nutná možnost ukládat větší množství dat. Datová paměť procesoru se skládá z 16 paměťových jednotek (viz kapitola 4.1), každá má svoji unikátní 4-bitovou adresu. Do každé z těchto jednotek je možné na základě adresy zapisovat data nebo z ní data číst.



Obrázek 19: Schéma datové paměti procesoru

Datovou paměť lze pomyslně rozdělit na dvě funkční části: zápis a čtení.

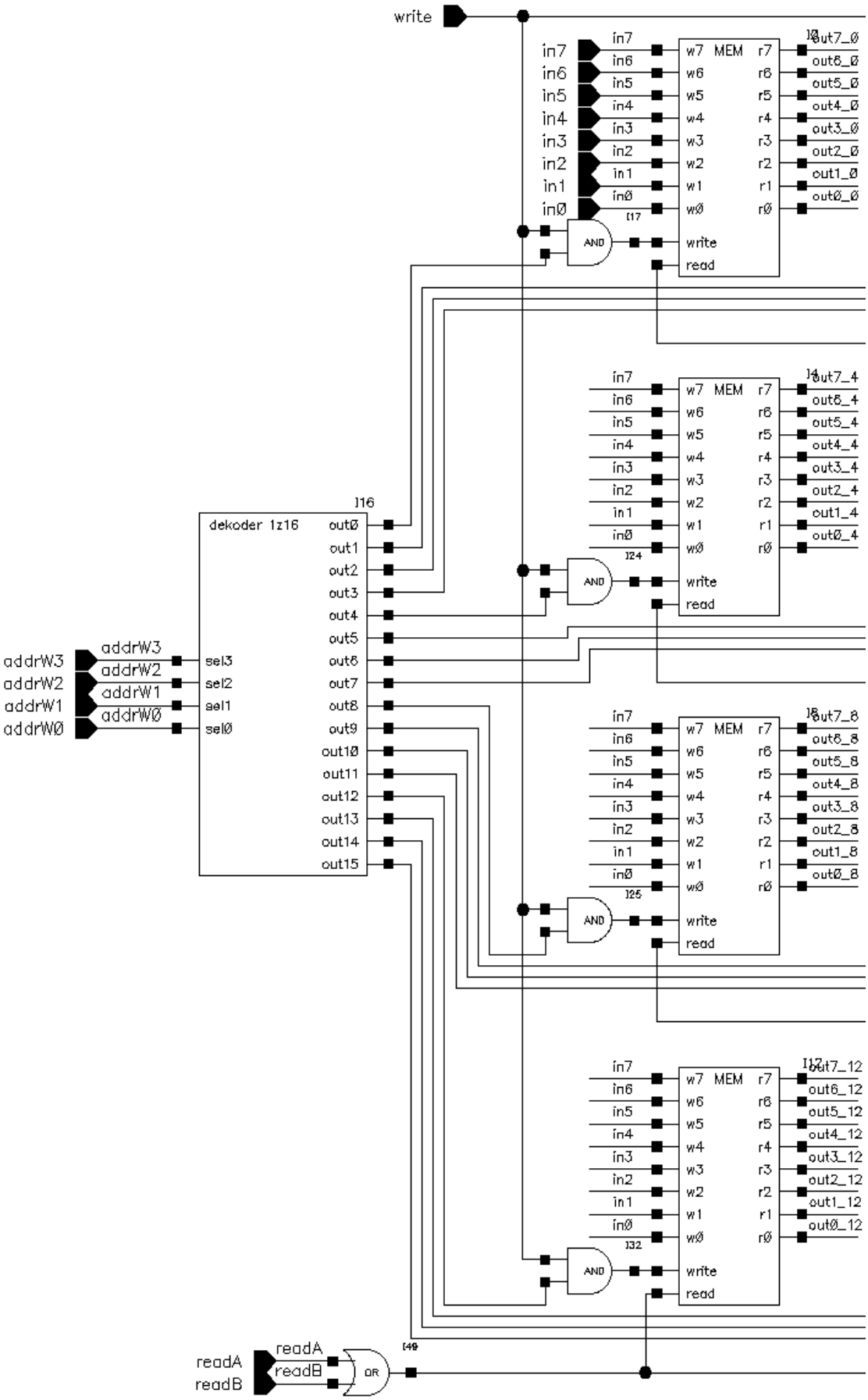
K umožnění zápisu obsahuje datová paměť procesoru celkem osm vstupních datových signálů *in[7:0]*, na které jsou přivedena data k zapsání do paměti. Dále je nutné nastavit adresu konkrétní paměťové jednotky *addrW[3:0]* a zápis se provede nastavením signálu *write* na hodnotu log. 1.



Obrázek 20: Schéma datové paměti procesoru – zápis

Do každé paměťové jednotky je zapojen jeden výstup z adresního dekodéru. Právě tento výstup definuje konkrétní adresu této paměťové jednotky. Vstupní datové signály jsou totiž přivedeny na všechny paměťové jednotky, ale k zápisu dojde pouze u té, jejíž adresa je nastavena pomocí datových vodičů.

Na obr. 21 je zobrazen detail zapojení datové paměti a paměťových jednotek s adresami 0, 4, 8 a 12. Například zápis do paměťové jednotky s adresou 0 nastane přivedením hodnoty 0000 na piny addrW[3:0] a nastavením vodiče write na hodnotu log. 1, načež výstup z hradla AND, který je přiveden do paměťové jednotky, bude mít také úroveň log. 1.

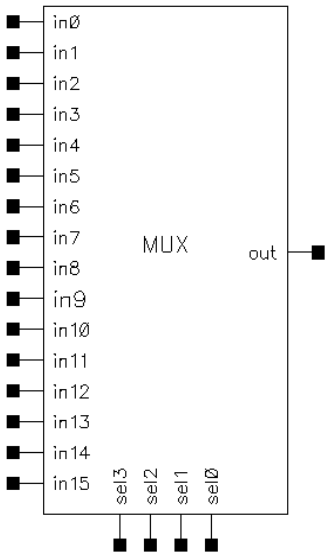


Obrázek 21: Schéma zapojení datové paměti – zápis – výřez

Dále obsahuje datová paměť dvě sady pro čtení – jednu pro paměťovou jednotku A a druhou pro B. Každá sada se skládá z osmi multiplexorů. [1] Multiplexor je elektronická součástka, která převádí na výstup jeden ze vstupů v závislosti na kombinaci logických hodnot výběrových vodičů. Všechny možné vstupně-výstupní kombinace jsou zobrazeny níže v tabulce 3.

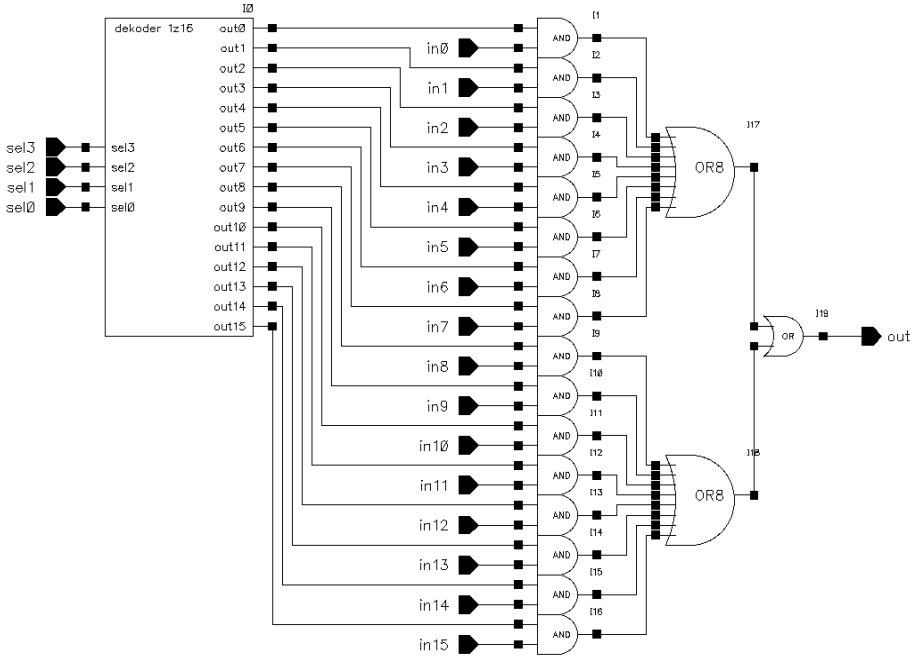
Tabulka 3: Pravdivostní tabulka multiplexoru

sel3	sel2	sel1	sel0	out
0	0	0	0	in0
0	0	0	1	in1
0	0	1	0	in2
0	0	1	1	in3
0	1	0	0	in4
0	1	0	1	in5
0	1	1	0	in6
0	1	1	1	in7
1	0	0	0	in8
1	0	0	1	in9
1	0	1	0	in10
1	0	1	1	in11
1	1	0	0	in12
1	1	0	1	in13
1	1	1	0	in14
1	1	1	1	in15



Obrázek 22: Symbol multiplexoru

Zapojení multiplexoru přesně odpovídá jeho pravdivostní tabulce. Dekodér má hodnotu log. 1 vždy přesně v jednom případě kombinace vstupních hodnot (*sel[3:0]*). To znamená, že pokud bude výstup z dekodéru mít hodnotu log. 0, při zapojení s logickými hradly typu AND bude výstup z tohoto hradla vždy mít hodnotu log. 0. Naopak pokud bude výstup z dekodéru mít hodnotu log. 0, na výstupu hradla bude hodnota *in*. Ve výsledku bude na maximálně jednom výstupu z AND hradel hodnota log. 1.

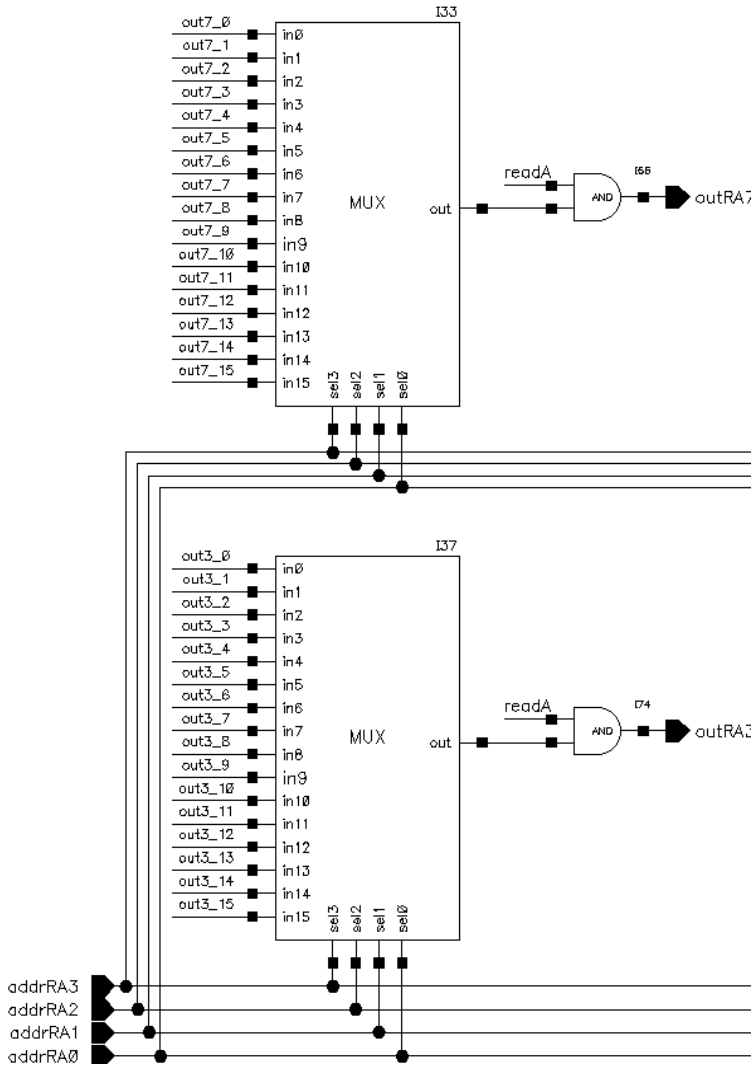


Obrázek 23: Multiplexor 1 z 16

Na rozdíl od operace zápisu, kdy je možné v konkrétním okamžiku zapisovat pouze do jedné paměťové jednotky, je možné číst zároveň z dvou paměťových jednotek (A,B) zároveň. Datová paměť procesoru totiž obsahuje dvě sady adresních vodičů *addrRA[3:0]* a *addrRB[3:0]*, dvě sady výstupních datových vodičů *outRA[7:0]* a *outRB[7:0]* a dva řídicí vodiče *readA* a *readB*.

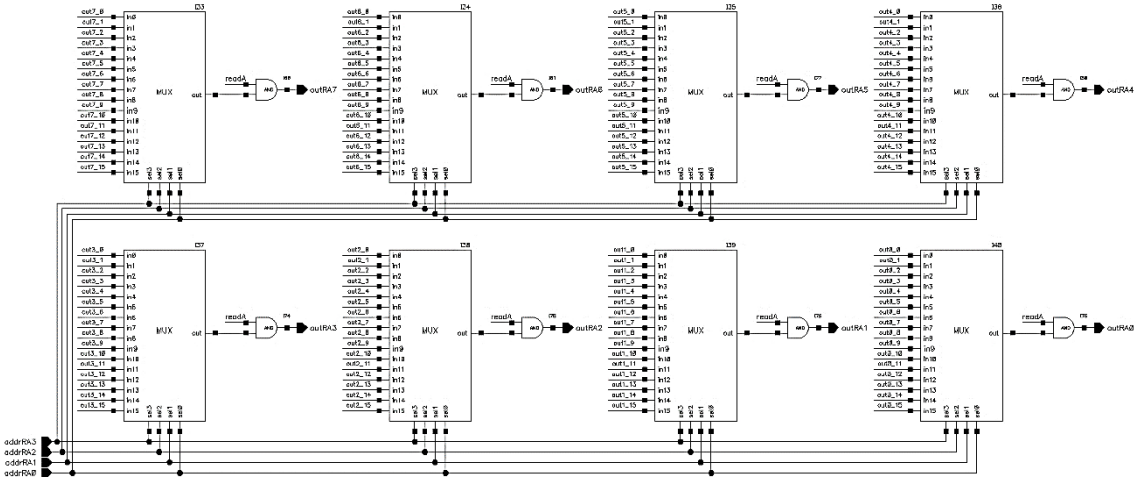
Pro čtení z jednotky A je nutné nastavit adresu *addrRA[3:0]*, poté nastavit řídicí vodič *readA* na hodnotu log. 1 a díky tomu se objeví obsah konkrétní paměťové jednotky na výstupu *outRA[7:0]*.

Níže na obr. 24 je zobrazen výřez ze čtecí části datové paměti, konkrétně bity č. 7 a 3. Na vstup multiplexoru jsou přivedeny ze všech adres bity konkrétní pozice, tedy sedmé, popř. třetí pozice, a na výstup je propsán bit z té paměťové jednotky, jejíž adresa je zadána řídicími signály *addrRA[3:0]*.



Obrázek 24: Schéma zapojení datové paměti procesoru – čtení A – výřez

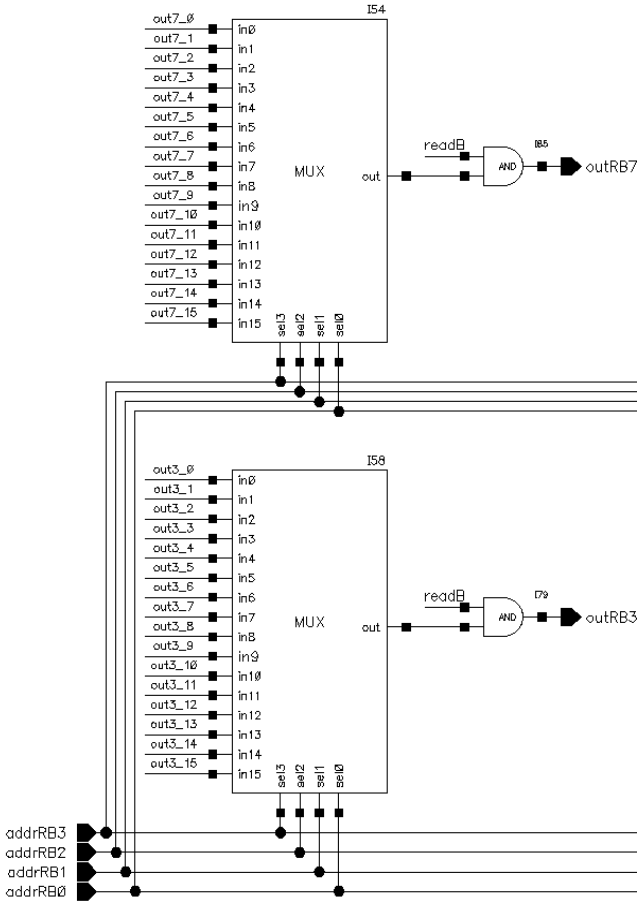
Zapojení celé části paměti starající se o zobrazení hodnot na výstupu A je zobrazeno na obr. 25.



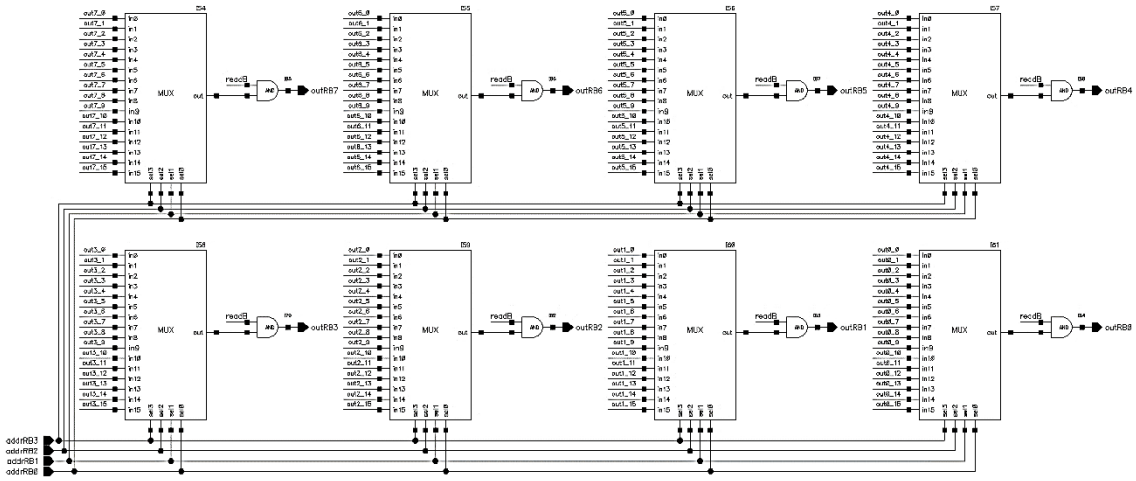
Obrázek 25: Schéma zapojení datové paměti procesoru – čtení A

Obdobně při čtení z jednotky B je nutné nastavit adresu *addrRB[3:0]*, poté nastavit řídicí vodič *readB* na hodnotu log. 1 a následně se objeví obsah této paměťové jednotky na výstupu *outRB[7:0]*.

Výřez ze čtecí části datové paměti pro B, rovněž pro sedmý a třetí bit, je zobrazen níže na obr. 26. Princip fungování je naprosto stejný jako u A, stejně jako vzhled zapojení celé části starající se o čtení na výstup B zobrazené na obr. 27.



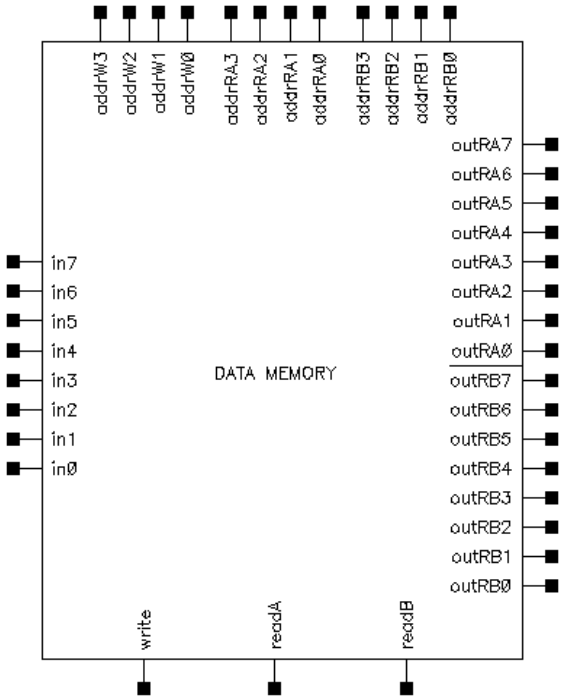
Obrázek 26: Schéma zapojení datové paměti – čtení B – výřez



Obrázek 27: Schéma zapojení datové paměti – čtení B

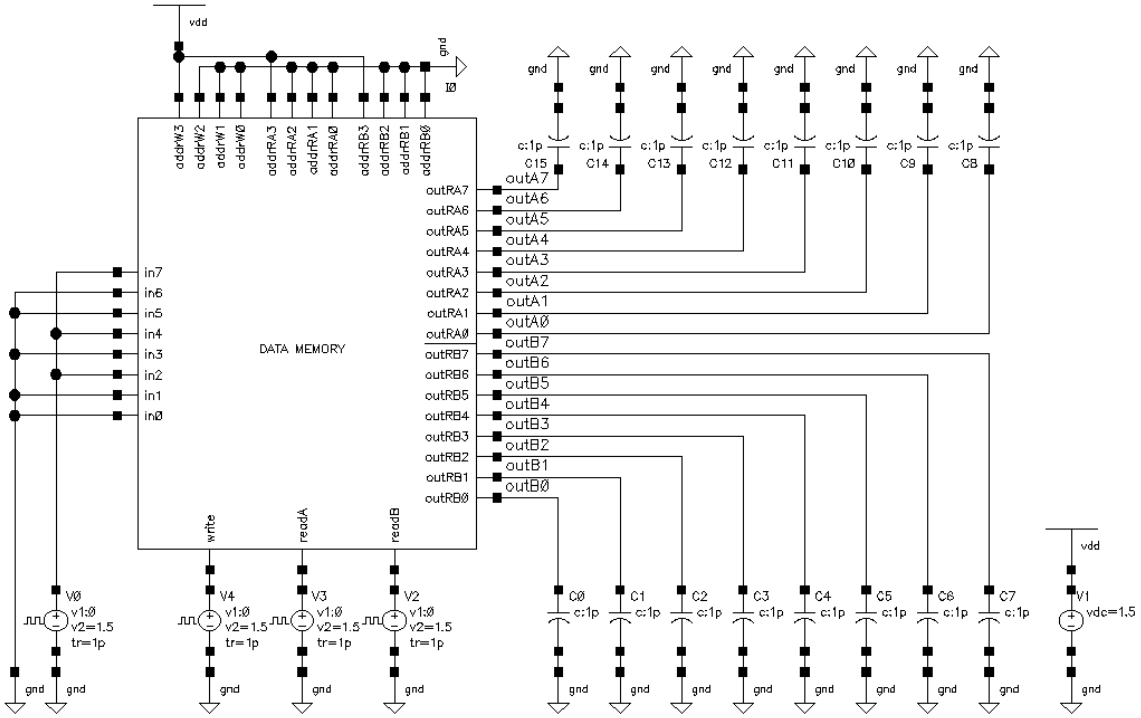
Čtení z různých paměťových jednotek je na sobě nezávislé, není nutné číst ze dvou paměťových jednotek zároveň, jedná se pouze o výhodnou možnost. Stejně tak je možné číst z dvou různých paměťových jednotek nebo číst pouze z jedné, ale dvakrát a dále s každou sadou dat naložit jinak.

Jako poslední je nutné vytvořit symbol datové paměti procesoru, který je zobrazen na obr. 28. Takto je možné použít součástku ve finálním zapojení.



Obrázek 28: Symbol datové paměti

Pro použití datové paměti je klíčové jí otestovat stejně jako ostatní součástky.



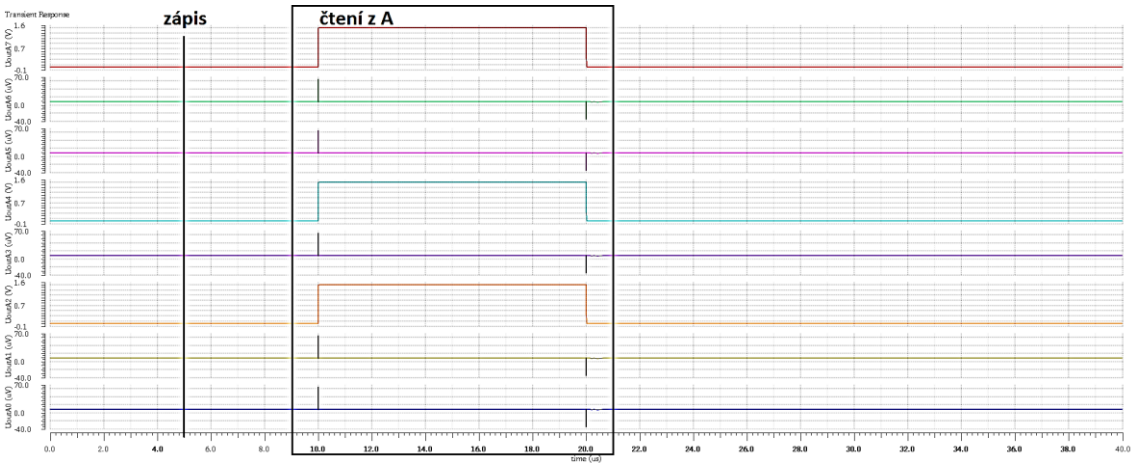
Obrázek 29: Schéma simulace datové paměti

Časování simulace je následující:

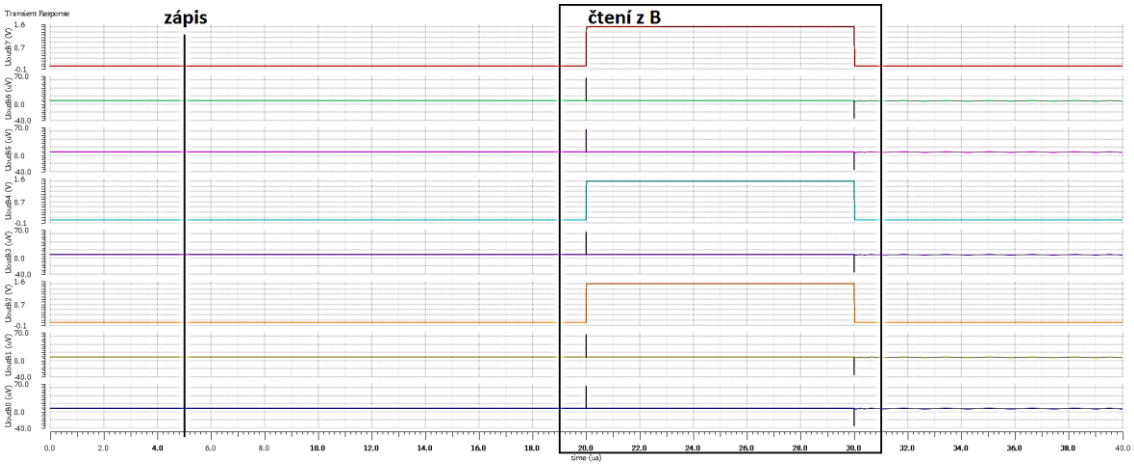
1. V čase t_1 = 0 µs jsou na vstupy *in[7:0]* přivedena data s délkou pulsu 10 µs a délkou náběžné i sestupné hrany 1 ps.
2. V čase t_2 = 5 µs jsou data zapsána do paměťové jednotky s adresou *addrW[3:0] = 1000* nastavením signálu *write* na úroveň log. 1 s délkou pulsu 5 µs a délkou náběžné a sestupné hrany 1 ps.
3. V čase t_3 = 10 µs jsou data přečtena z adresy *addrRA[3:0] = 1000* na výstup A, tedy nastavením signálu *readA* na úroveň log. 1 s délkou pulsu 10 µs a délkou náběžné a sestupné hrany 1 ps.
4. V čase t_4 = 20 µs jsou data přečtena z adresy *addrRB[3:0] = 1000* na výstup B, tedy nastavením signálu *readB* na úroveň log. 1 s délkou pulsu 10 µs a délkou náběžné a sestupné hrany 1 ps.

Níže na obr. 30 a 31 je zobrazen průběh této simulace. V čase t_2 = 5 µs došlo k zapsání dat do datové paměti, protože na výstupu jsou zobrazena data, jak při čtení uložených dat na výstup A v čase t_3 = 10 µs, tak na výstup B v čase t_4 = 20 µs. Na obrázcích je svislou čarou znázorněn moment, ve kterém došlo k zápisu dat na nastavenou adresu.

Z průběhů signálů je patrné, že k zápisu do konkrétní paměťové jednotky opravdu došlo, protože data jsou přítomna na výstupu v okamžiku čtení. Zároveň při čtení na výstup A a na výstup B jsou přečtená data totožná. Tento fakt dokazuje, že nezáleží na tom, z kterého výstupu čteme data.



Obrázek 30: Průběh simulace datové paměti – A



Obrázek 31: Průběh simulace datové paměti – B

5 INSTRUKČNÍ PAMĚŤ

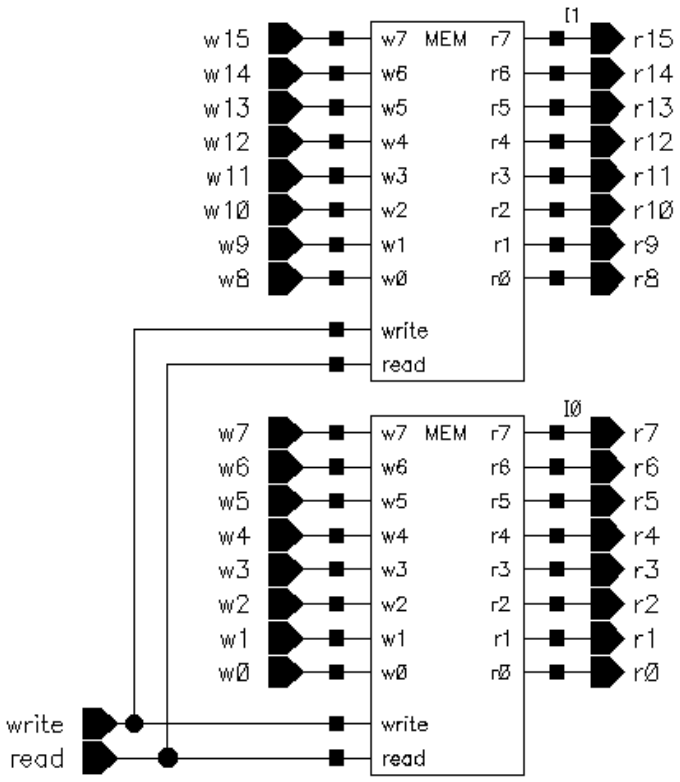
Procesor vykonává svojí práci na základě instrukcí, které jsou uloženy v instrukční paměti. Instrukce je způsob zakódování konkrétní matematicko-logické nebo paměťové operace.

Instrukce bývají po celou dobu práce procesoru neměnné a předem dané. To znamená, že jakékoliv chování procesoru by nemělo nijak ovlivnit data uložená v instrukční paměti. Stejně tak zápis do instrukční paměti by měl být pouze jeden, a to před počátkem běhu programu procesoru. Za běhu programu již procesor pouze čte instrukce uložené v instrukční paměti a zpracovává je s ohledem na jejich typ a obsah.

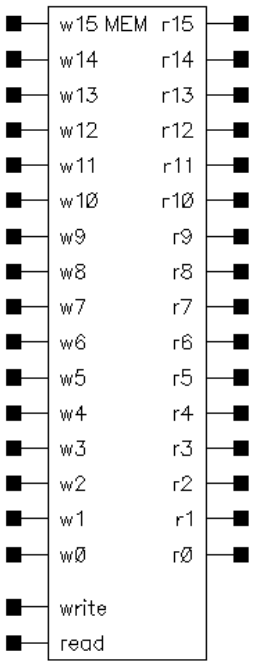
Seznam možných typů instrukcí a jejich vysvětlení je uveden v kapitole 3.

5.1 INSTRUKČNÍ PAMĚŤOVÁ JEDNOTKA

Způsob ukládání do instrukční paměti je obdobný jako v případě datové paměti s jediným rozdílem, že data uložená v datové paměti jsou osmibitová, zatímco instrukce se skládají z 16 bitů. Jelikož data uložená v instrukční paměti jsou dvojnásobné délky než data uložená v datové paměti, ke tvorbě datové paměti lze použít již vytvořenou datovou paměťovou jednotku (kap. 4.1).



Obrázek 32: Schéma instrukční paměťové jednotky

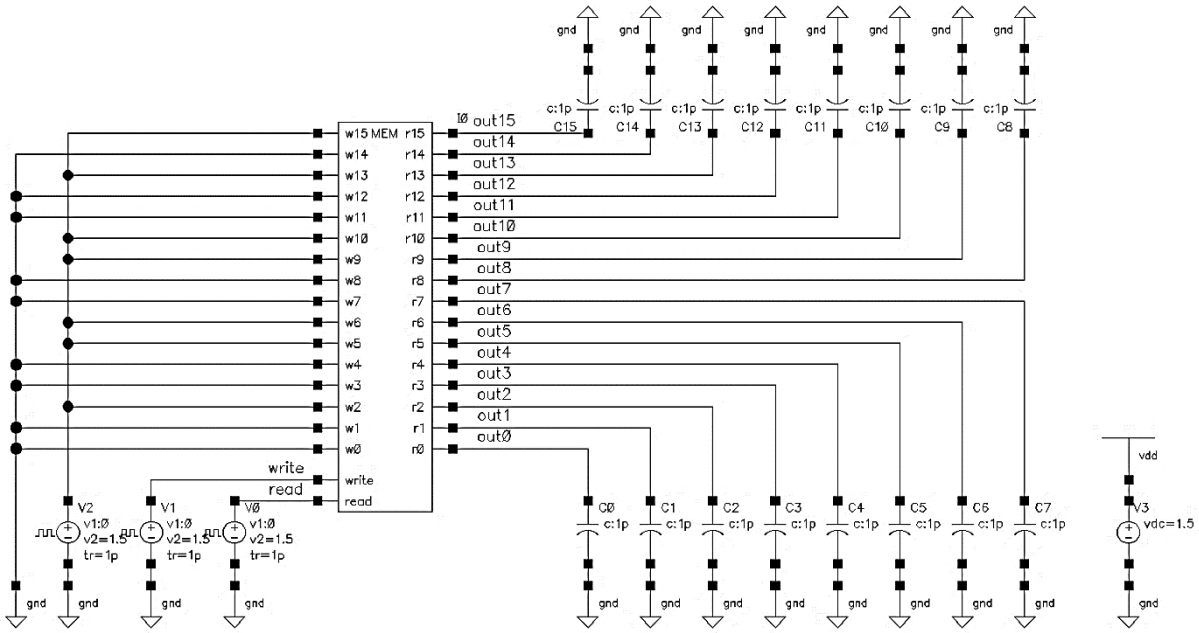


Obrázek 33: Symbol instrukční paměťové jednotky

Pro vytvoření instrukční paměťové jednotky je možné použít dvě datové paměťové jednotky – jedna k uložení horních 8 bitů a druhá k uložení dolních 8 bitů. Protože jsou obě ovládány stejnými řídicími signály *read* a *write*, dojde k současnému zápisu všech 16 bitů. V případě

instrukční paměti tedy stejně jako v případě datové paměti není možné pracovat pouze s některými vybranými bity, vždy je potřeba přečíst nebo zapsat všech 16 bitů najednou.

Pro simulaci instrukční paměti platí stejné zásady, jako jsou uvedeny u simulace datové paměti. Tou hlavní je dodržení správného načasování. Data, která mají být zapsána do instrukční paměti, musí být přítomna na vstupu v okamžik, kdy je nastaven řídicí signál *write* na hodnotu log. 1.

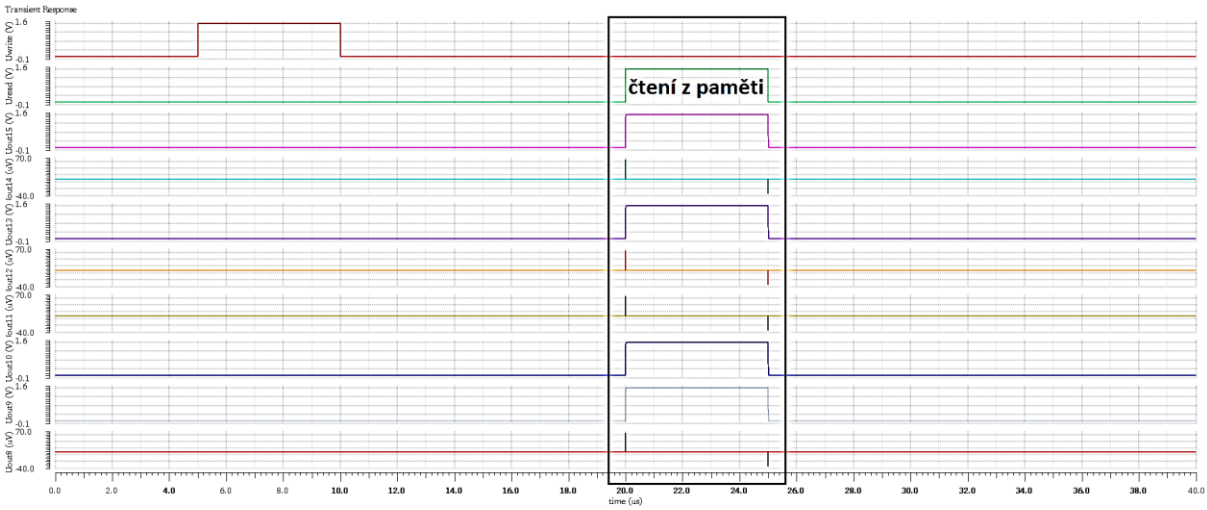


Obrázek 34: Schéma simulace instrukční paměťové jednotky

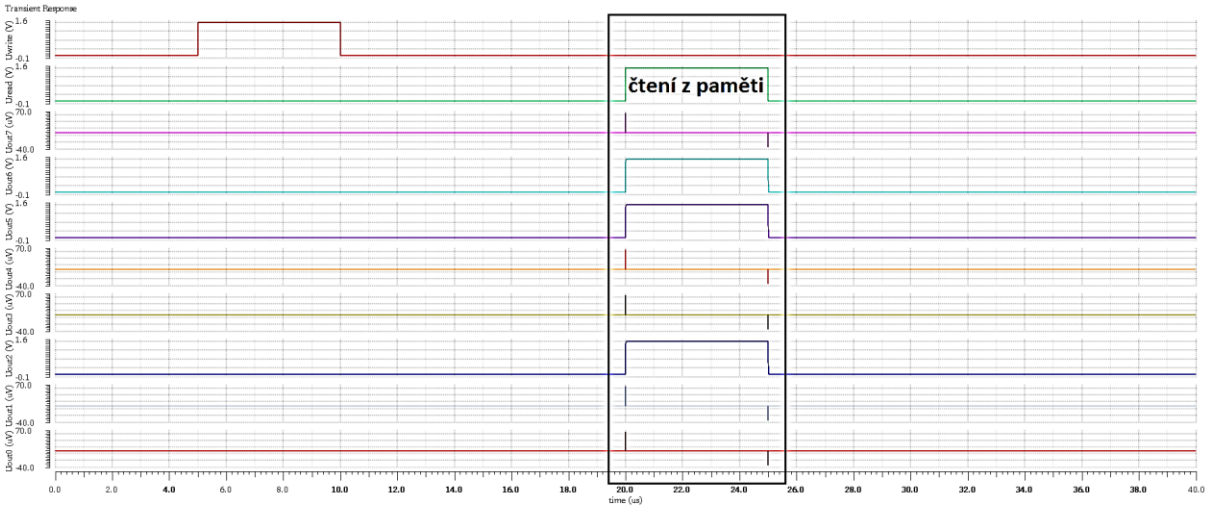
Časování signálů je následující:

1. V čase t_1 = 0 µs jsou na vstupy přivedena data *w[15:0]*, délka pulsu je 10 µs a délka náběžné i sestupné hrany je 1 ps.
2. V čase t_2 = 5 µs je nastaven signál *write* na hodnotu log. 1, délka pulsu je 5 µs a délka náběžné i sestupné hrany je 1 ps. Jelikož k zapsání hodnoty do registru dochází s náběžnou hranou, v čase t_2 jsou na vstupech *w[15:0]* přítomna data z kroku 1 a k jejich zápisu dojde.
3. V čase t_3 = 20 µs je nastaven signál *read* na hodnotu log. 1, délka pulsu je 5 µs a délka náběžné i sestupné hrany je 1 ps. Díky tomu dojde k propsání hodnot uložených v paměťové jednotce na výstup v čase t_3.

Na následujících obrázkách jsou zobrazeny průběhy této simulace – nejprve horních 8 bitů a poté dolních 8 bitů.



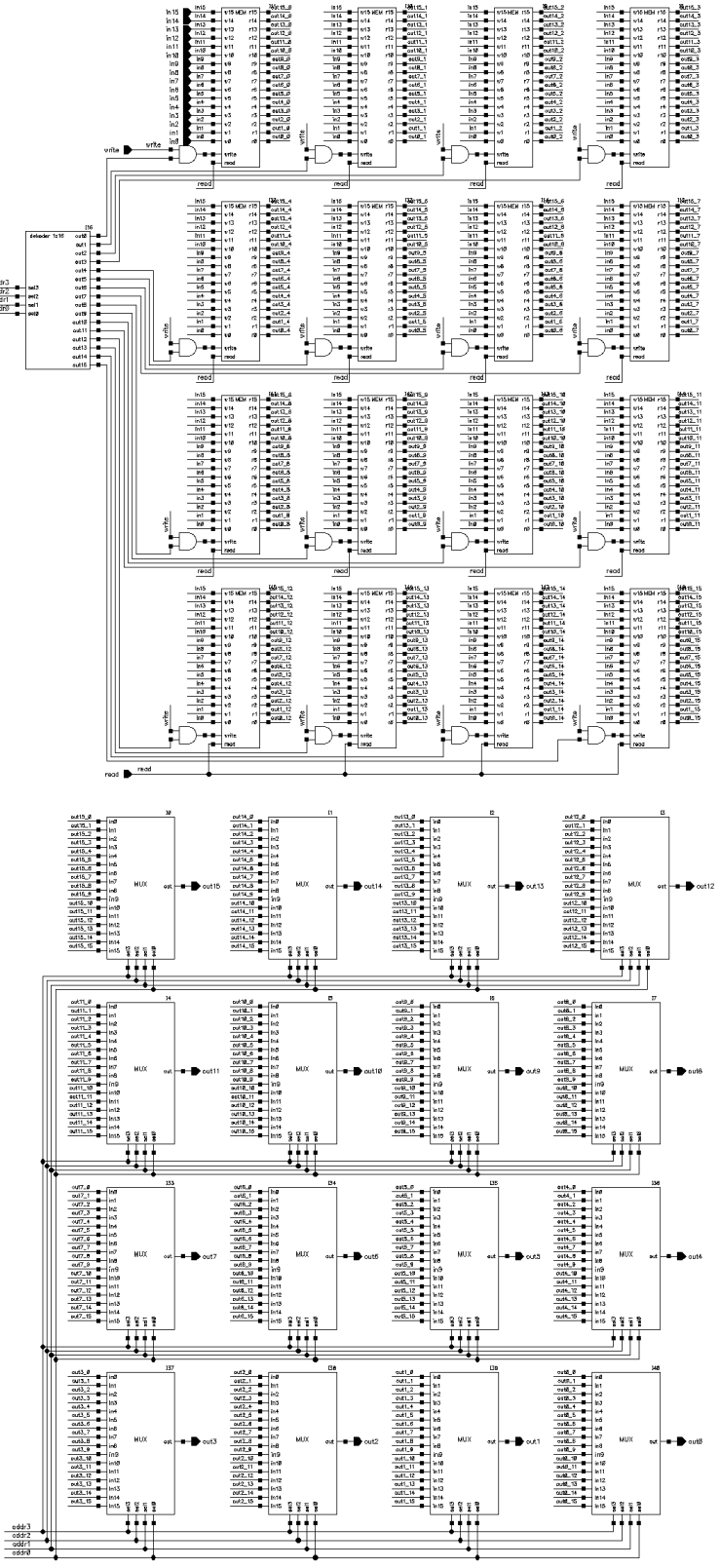
Obrázek 35: Průběh simulace instrukční paměťové jednotky – horních 8 bitů



Obrázek 36: Průběh simulace instrukční paměťové jednotky – dolních 8 bitů

5.2 INSTRUKČNÍ PAMĚŤ PROCESORU

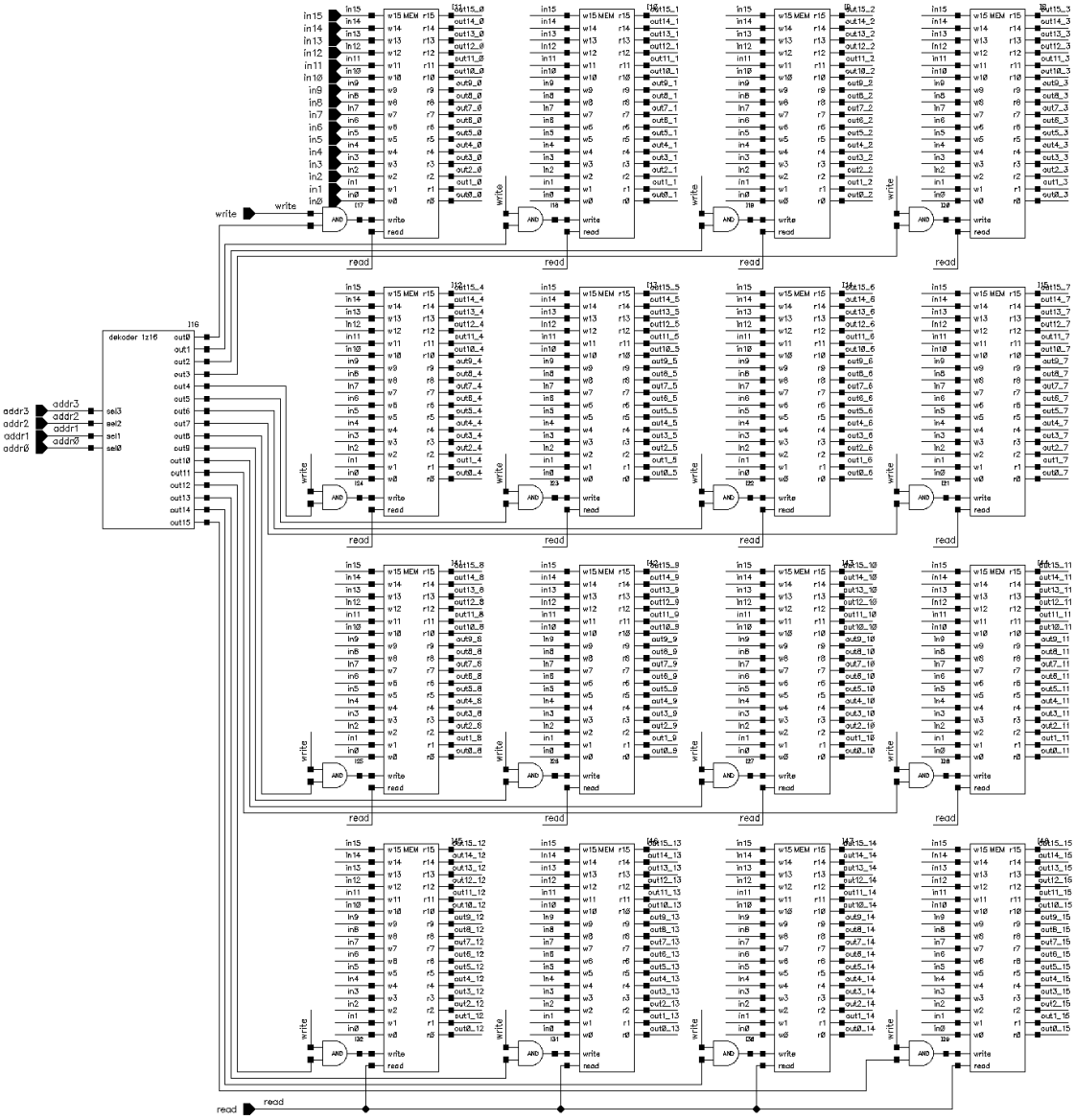
V procesoru je možnost zabudování prakticky stejné instrukční paměti jako datové. Schéma instrukční a datové paměti se liší pouze ve velikosti. Zatímco data lze v procesoru uchovávat 8-bitová, instrukce mají délku 16 bitů. Adresování je také 4-bitové, obdobně jako u datové paměti procesoru. Najednou lze tedy uložit až 16 instrukcí.



Obrázek 37: Schéma instrukční paměti

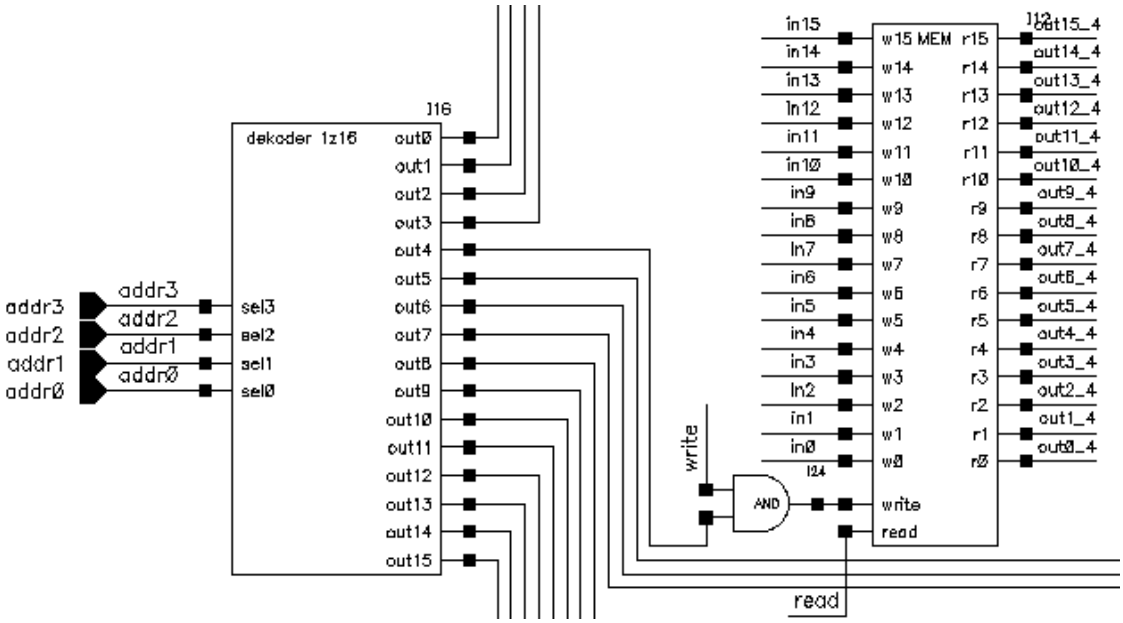
Stejně jako v případě datové paměti lze i instrukční paměť rozdělit na dvě funkční části: zápis a čtení.

Při zápisu je nutné přivést data na šestnáct vstupních datových signálů *in[15:0]*, nastavit adresu paměťové jednotky *addr[3:0]* a nastavit signal *write* na hodnotu log. 1.



Obrázek 38: Schéma instrukční paměti – zápis

V instrukční paměti je zapojen adresní dekodér z kapitoly 4.2.1, jehož výstup určuje adresu konkrétní instrukce. Vstupní signály jsou i tentokrát přiveden na všechny instrukční paměťové jednotky a zápis bude proveden u té, jejíž adresa je nastavena pomocí řídicích signálů *addr[3:0]*.

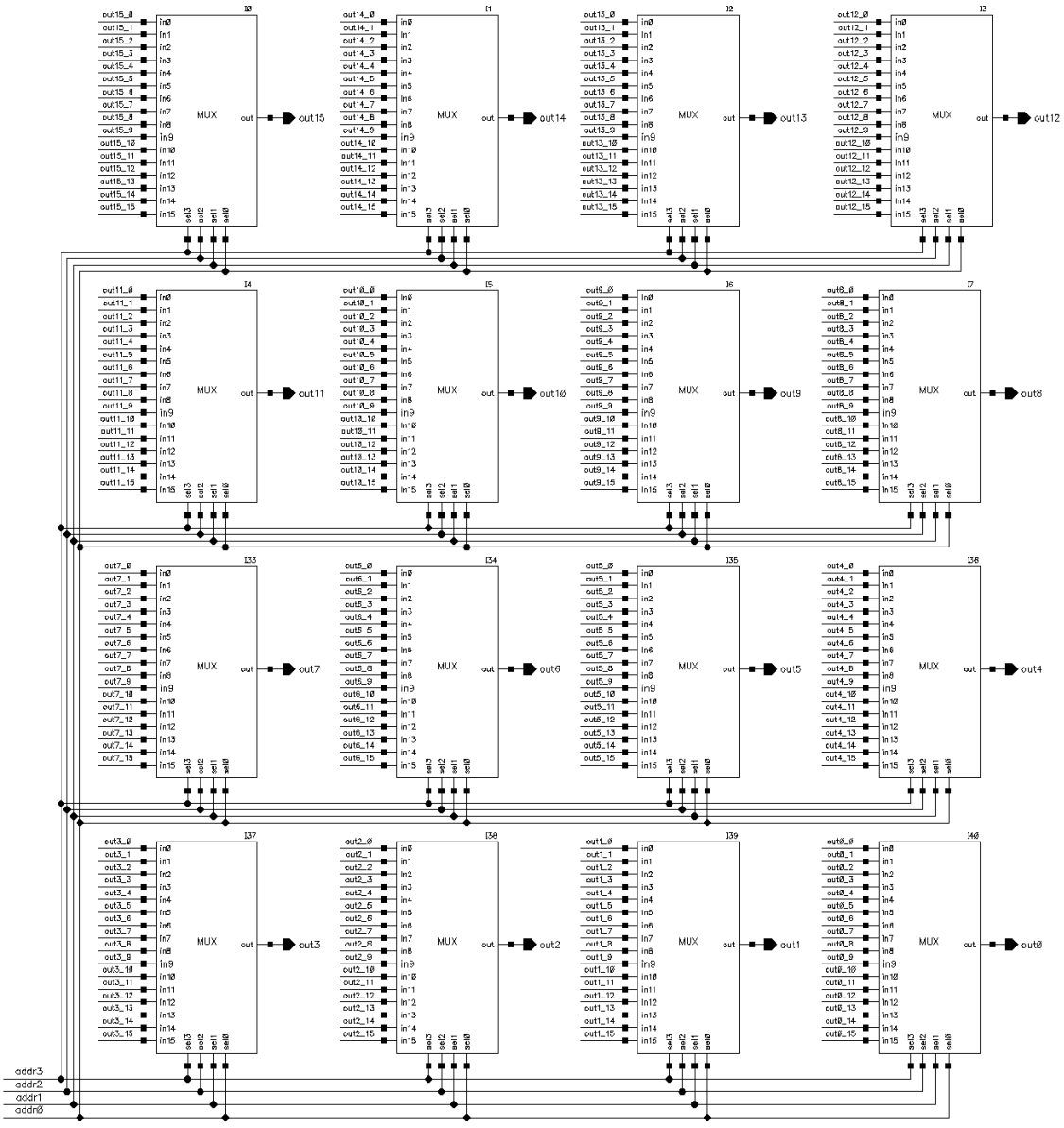


Obrázek 39: Instrukční paměť – zápis – výřez

Proces čtení z instrukční paměti se ale v mnohém liší od té datové. Hlavním rozdílem je fakt, že z instrukční paměti není možné číst z více adres najednou. Důvodem je to, že procesor nepodporuje paralelní zpracování instrukcí. V jednom konkrétním časovém okamžiku je možné zpracovávat pouze jednu instrukci a není tedy důvod, aby bylo možné číst více instrukcí najednou.

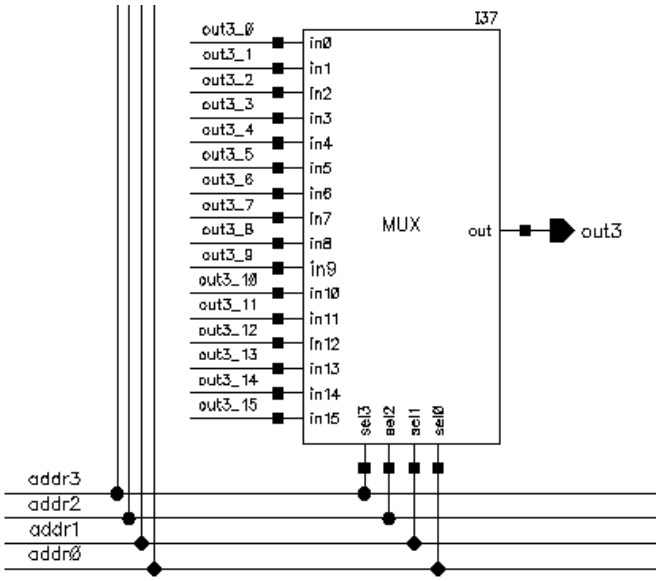
O čtení se stará šestnáct multiplexorů, jejichž funkcionalita je znázorněna v tab.3. Při čtení z instrukční paměti je nutné nastavit adresu *addr[3:0]*, poté nastavit signál *read* na hodnotu log. 1 a data uložená v konkrétní instrukční paměťové jednotce se objeví na výstupu *out[15:0]*.

Při čtení z instrukční paměti se stejně jako u té datové různé paměťové jednotky vzájemně neovlivňují a nejsou na sobě nikterak závislé.



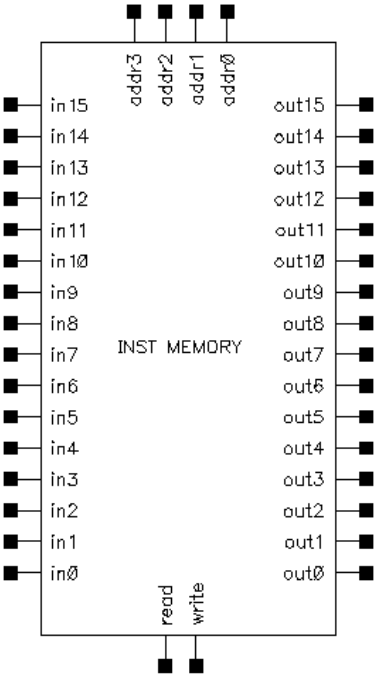
Obrázek 40: Schéma instrukční paměti – čtení

Níže na obr. 41 je zobrazen výřez z čtecí části instrukční paměti. V něm je obsažen multiplexor starající se o zobrazení výstupního bitu č. 3 dle řídicího signálu addr[3:0].



Obrázek 41: Instrukční paměť – čtení – výřez

Symbol reprezentující instrukční paměť je zobrazen na obr. 42.



Obrázek 42: Symbol instrukční paměti

6 ARITMETICKO-LOGICKÁ JEDNOTKA

Aritmeticko-logická jednotka (ALU) je součást procesoru vykonávající instrukce. Většinou přijímá na vstupu operandy a typ instrukce a na výstupu produkuje výsledek. [6]

V tomto případě, jednotka typicky přijme dvě 8-bitová čísla A a B ve formě dvojkového doplňku, vykoná příslušnou operaci a na výstup vypíše výsledek této operace, opět ve formě 8-bitového čísla.

Dvojkový doplněk (2's complement) je způsob vyjádření čísel se znaménkem, ve kterém se pro n bitů dají vyjádřit čísla v rozsahu od $-(2^{n-1})$ do $(2^{n-1}-1)$. Záporná hodnota čísla se vyjádří z kladné hodnoty inverzí bitů a přičtením 1. [10] Znaménko je vyjádřeno v nejvyšším bitu – pokud se MSB rovná log. 1, pak je číslo záporné, v opačném případě je kladné.

6.1 PODPOROVANÉ OPERACE

V implementovaném procesoru je podporováno celkem 7 matematických operací s osmibitovými daty:

- sčítání (A+B)
- odčítání (A-B)
- aritmetický bitový posun doleva (A << 1)
- aritmetický bitový posun doprava (A >> 1)
- AND (A AND B)
- OR (A OR B)
- XOR (A XOR B)

6.1.1 Sčítání, odčítání

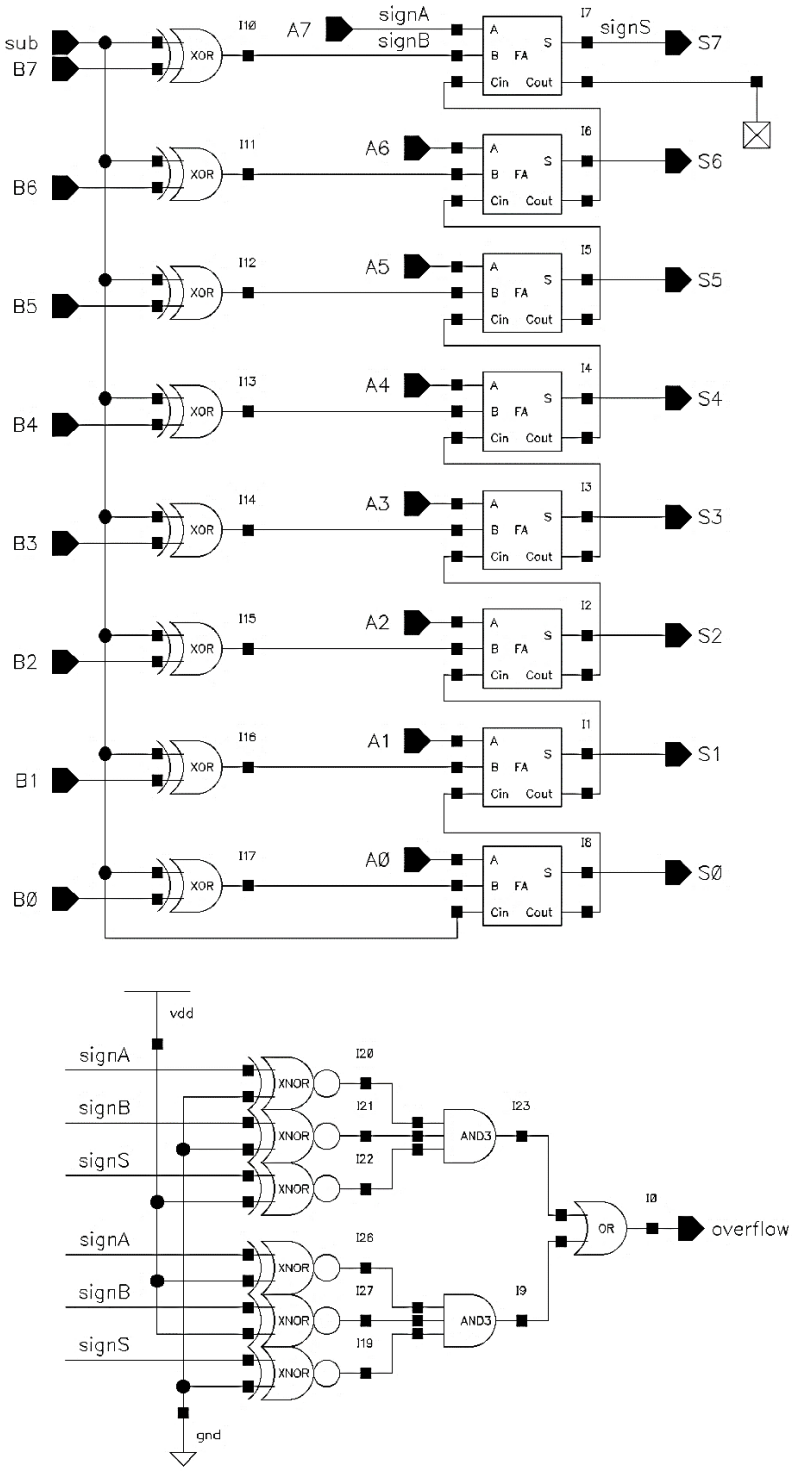
Sčítání a odčítání je zajištěno jedním společným obvodem – 8-bitovou sčítačkou/odčítačkou [1]. Jedná se o součástku, na jejímž vstupu jsou dva 8-bitové operandy *A[7:0]* a *B[7:0]* a řídicí signál *sub*. Výstup pak tvoří 8-bitová hodnota *S[7:0]*. Pokud má signál sub úroveň log. 1, na výstupu se zobrazí rozdíl A-B, v opačném případě se zobrazí součet A+B.

Je důležité dávat pozor na přetečení neboli *overflow*, které je indikováno u dvojkového doplňku nesprávným znaménkem výsledku. Pokud po sečtení dvou čísel se stejným znaménkem má výsledek opačné znaménko, zcela jistě došlo k přetečení. [10] Tato detekce ve zmíněné práci [1] chyběla, proto byla doplněna. Aby byla možná detekce přetečení i u odčítání, je důležité nejprve převést rozdíl čísel A-B na součet opačného čísla, teda A+(-B).

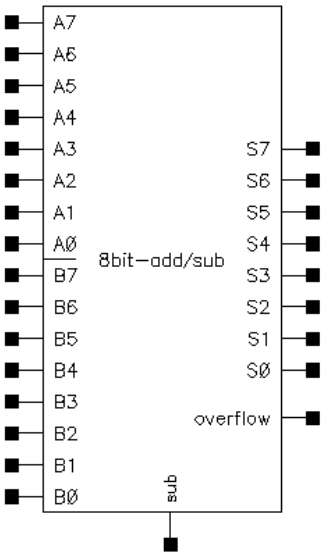
Tabulka 4: Detekce přetečení součtu dvou čísel A+B

znaménko A	znaménko B	znaménko S	přetečení
+	+	+	ne
+	+	-	ano
-	-	+	ano
-	-	-	ne
+	-	+, -	ne
-	+	+, -	ne

Pokud má signál *sub* hodnotu log. 1, je realizována operace odčítání. XOR hradlo s jedním vstupem úrovně log. 1 způsobí konverzi bitů (viz tabulka 7) a přičtením 1 vznikne opačná hodnota operandu B ve dvojkovém doplňku. Došlo tedy k otočení znaménka u operandu B a na výstupu se tedy zobrazí součet A+(-B), což je rozdíl A-B. Naopak, pokud má signál sub hodnotu log. 0, XOR hradlo nemá žádný vliv na operandy a na výstupu se zobrazí součet A+B.

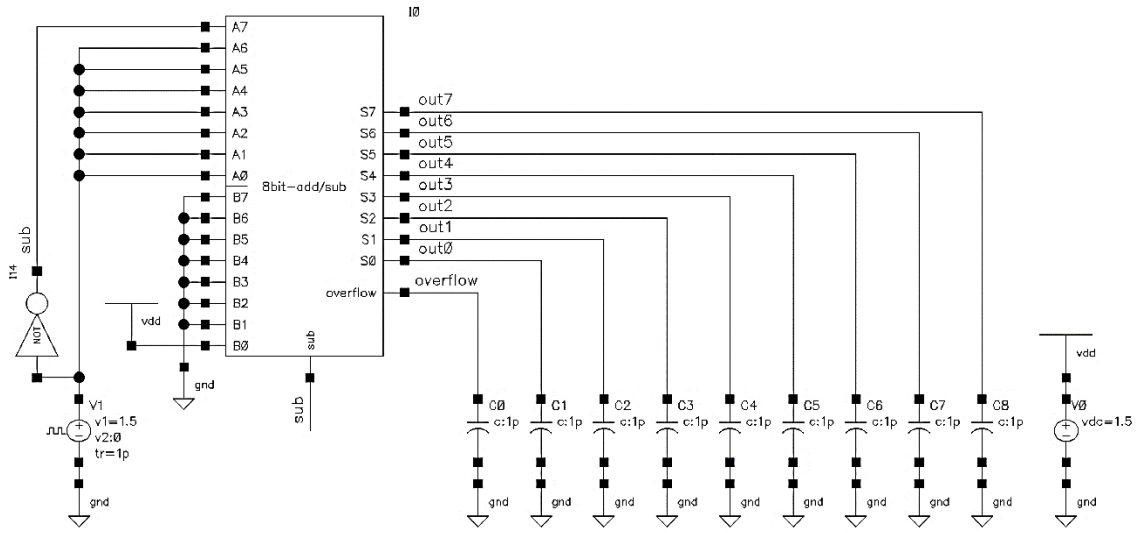


Obrázek 43: Schéma 8-bitové sčítačky/odčítačky



Obrázek 44: Symbol 8-bitové sčítačky/odčítačky

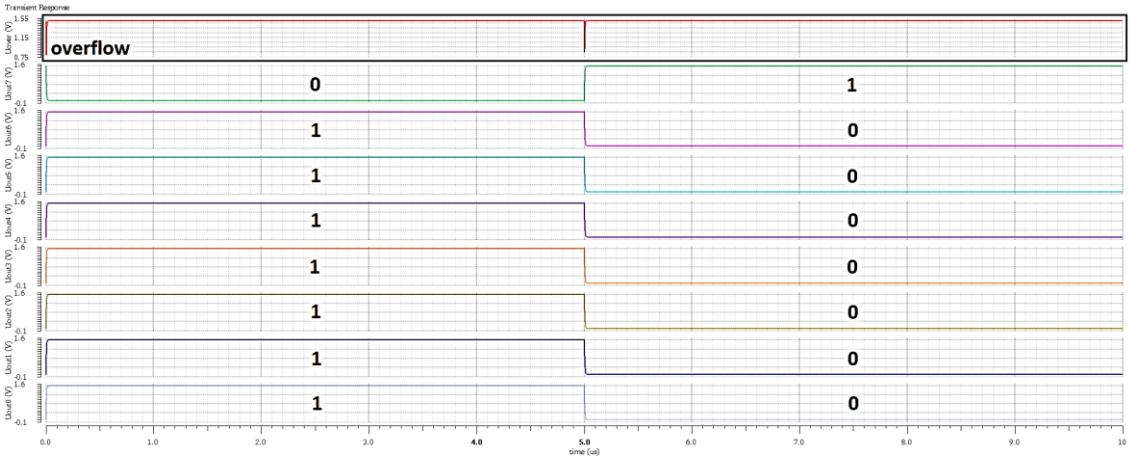
Při simulaci této sčítačky/odčítačky jsou vyzkoušeny 2 krajní případy. Prvním případem je odečtení 1 od nejmenšího možného 8-bitového čísla zapsaného v dvojkovém doplňku, tedy -128 (binárně zapsáno 1000 0000). Druhý prověřovaný případ je přičtení 1 k největšímu možnému 8-bitovému číslu zapsanému v dvojkovém doplňku, tedy k 127 (binárně zapsáno 0111 1111). V obou těchto případech by mělo dojít k přetečení.



Obrázek 45: Schéma simulace 8-bitové sčítačky/odčítačky

Rozdíl čísel -128 a 1 je v konečném důsledku součet čísel -128 a -1 (binárně zapsáno 1111 1111). Výsledný binárně zapsaný součet je tedy: 1000 0000 + 1111 1111 = ~~1~~ 0111 1111. Výsledkem tedy nemůže být očekávaných -129, protože tato hodnota je mimo rozsah 8-bitového čísla v dvojkovém doplňku. Ve skutečnosti výsledek 0111 1111 odpovídá dekadickému číslu 127. Jelikož oba operandy měly záporné znaménko a výsledek má kladné znaménko, zcela jistě došlo k přetečení.

Stejně tak součet čísel 127 a 1 (binárně zapsáno 0000 0001) je: 0111 1111 + 0000 0001 = 1000 0000. Výsledek očekáváme 128, ale protože tato hodnota je také mimo rozsah 8-bitového čísla ve dvojkovém doplňku, ve skutečnosti odpovídá výsledek 1000 0000 dekadickému číslu -128. Oba operandy měly kladné znaménko, výsledek má záporné znaménko, tedy i v tomto případě došlo k přetečení.

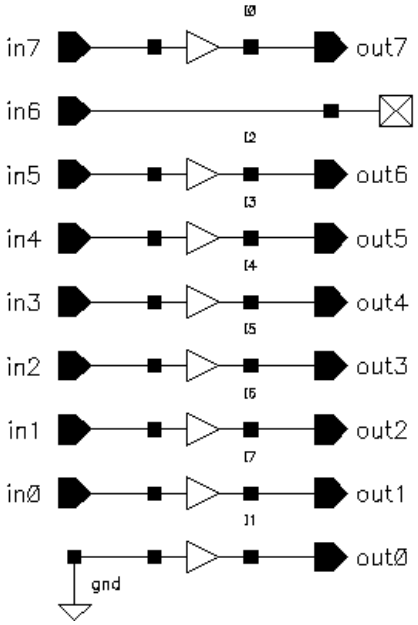


Obrázek 46: Průběh simulace 8-bitové sčítačky/odčítačky

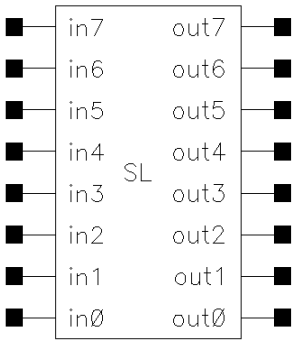
6.1.2 Bitový posun

Existují dva druhy bitových posunů: aritmetický a logický.

Logický posun doleva znamená, že všechny bity v binárně zapsaném čísle jsou posunuty o jednu pozici doleva, nejnižší bit je doplněn log. 0. Obdobně logický posun doprava znamená, že všechny bity v binárně zapsaném čísle jsou posunuty o jednu pozici doprava a nejvyšší bit je doplněn log. 0. Tento typ logického posunu je vhodný pro binární čísla bez znaménka a vyjadřuje násobení dvěma v případě posunu doleva, resp. dělení dvěma v případě posunu doprava. Aritmetický posun je vhodný pro posun binárně zapsaných čísel se znaménkem, protože na rozdíl od toho logického zachovává znaménko. To znamená, že v případě obou posunů zůstává logická hodnota MSB zachována. [11]



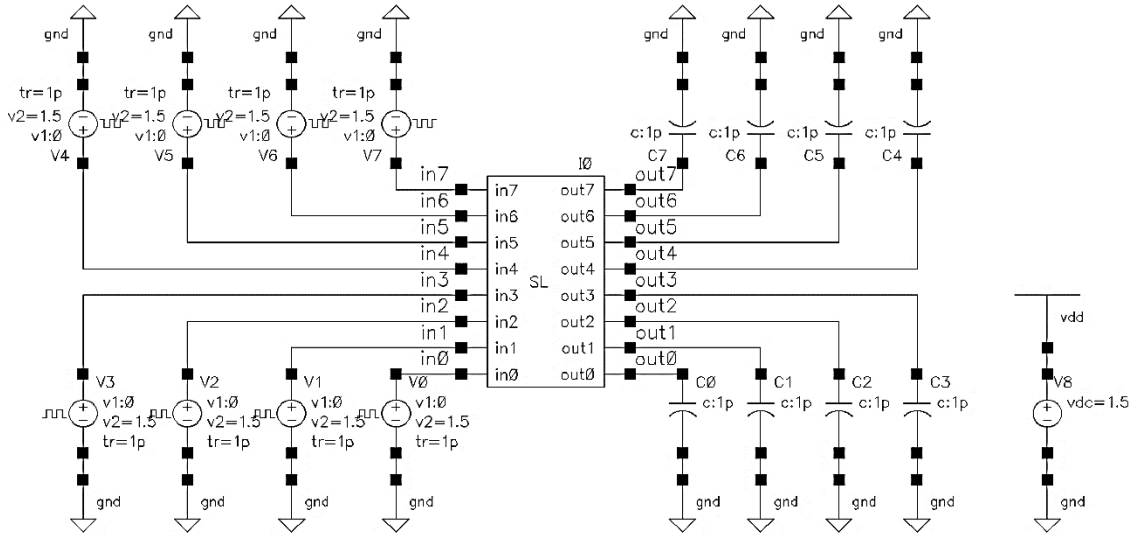
Obrázek 47: Schéma 8-bitového aritmetického posunu doleva



Obrázek 48: Symbol 8-bitového aritmetického posunu doleva

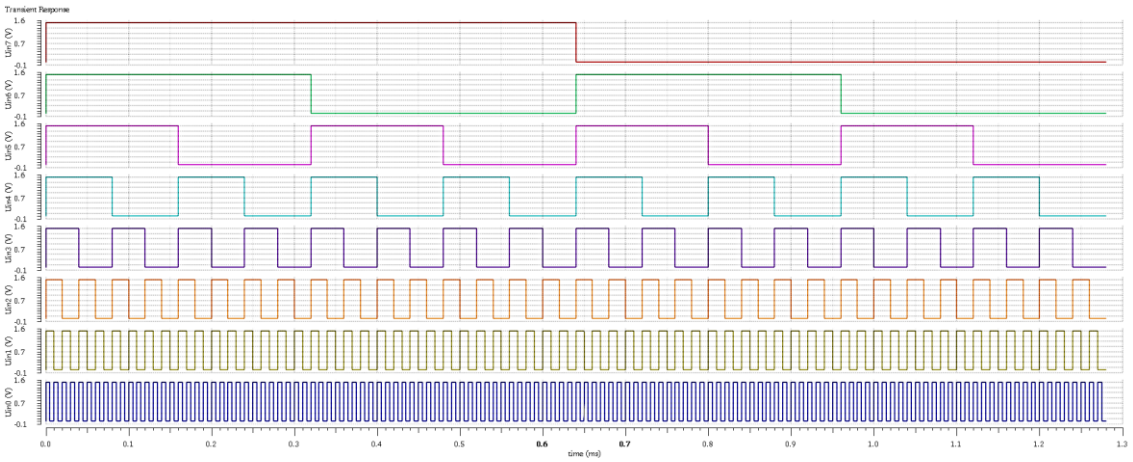
Pro simulaci byl zvolen takový postup, že na vstupy součástky byly přivedeny následující zdroje pulsního napětí:

- out7: perioda 1280 µs, délka pulsu 640 µs, délka náběžné i sestupné hrany 1 ps
- out6: perioda 640 µs, délka pulsu 320 µs, délka náběžné i sestupné hrany 1 ps
- out5: perioda 320 µs, délka pulsu 160 µs, délka náběžné i sestupné hrany 1 ps
- out4: perioda 160 µs, délka pulsu 80 µs, délka náběžné i sestupné hrany 1 ps
- out3: perioda 80 µs, délka pulsu 40 µs, délka náběžné i sestupné hrany 1 ps
- out2: perioda 40 µs, délka pulsu 20 µs, délka náběžné i sestupné hrany 1 ps
- out1: perioda 20 µs, délka pulsu 10 µs, délka náběžné i sestupné hrany 1 ps
- out0: perioda 10 µs, délka pulsu 5 µs, délka náběžné i sestupné hrany 1 ps



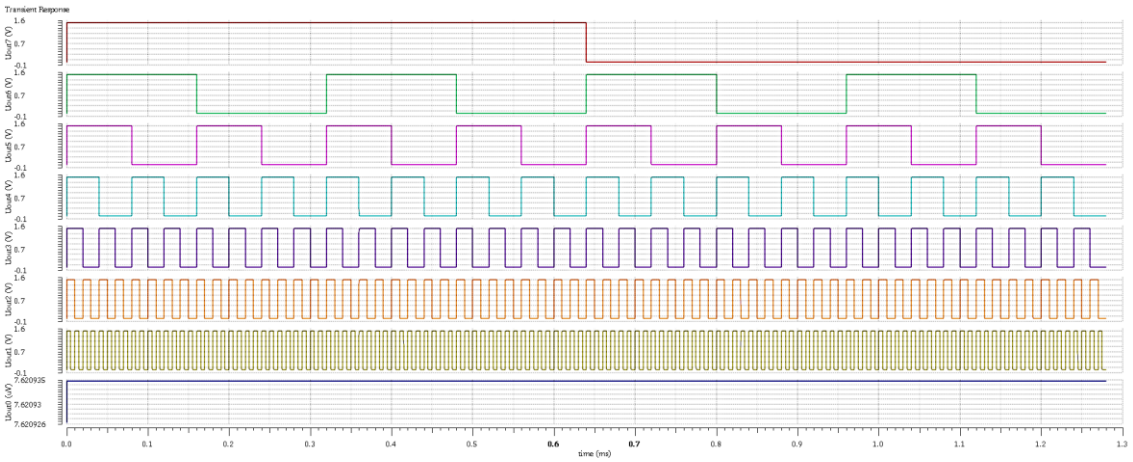
Obrázek 49: Simulace 8-bitového posunu doleva

Takto zapojené zdroje napětí postupně v čase přivádí do součástky všechny možné 8-bitové kombinace a je sledován výstup.



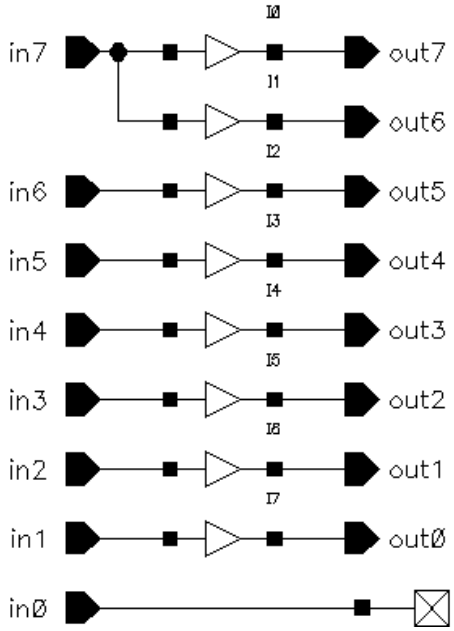
Obrázek 50: Průběh simulace 8-bitového posunu doleva – vstupní data

Na obr. 51 je zobrazen napěťový průběh výstupních signálů (ze shora dolů out7 až out0). Je z něj patrné, že oproti obr. 50 jsou signály posunuty o jeden, kromě out7, který zůstal stejný, a out0, který má stabilně úroveň log. 0.

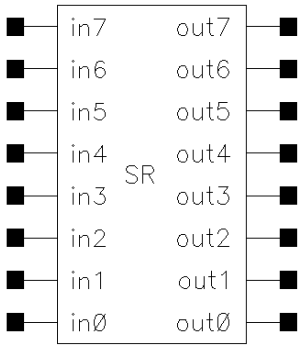


Obrázek 51: Průběh simulace 8-bitového posunu doleva – výstupní data

V zapojení aritmetického posunu doprava je opět překopírováno znaménko nejvyššího bitu, zatímco hodnoty ostatních bitů jsou posunuty o jednu pozici doprava.

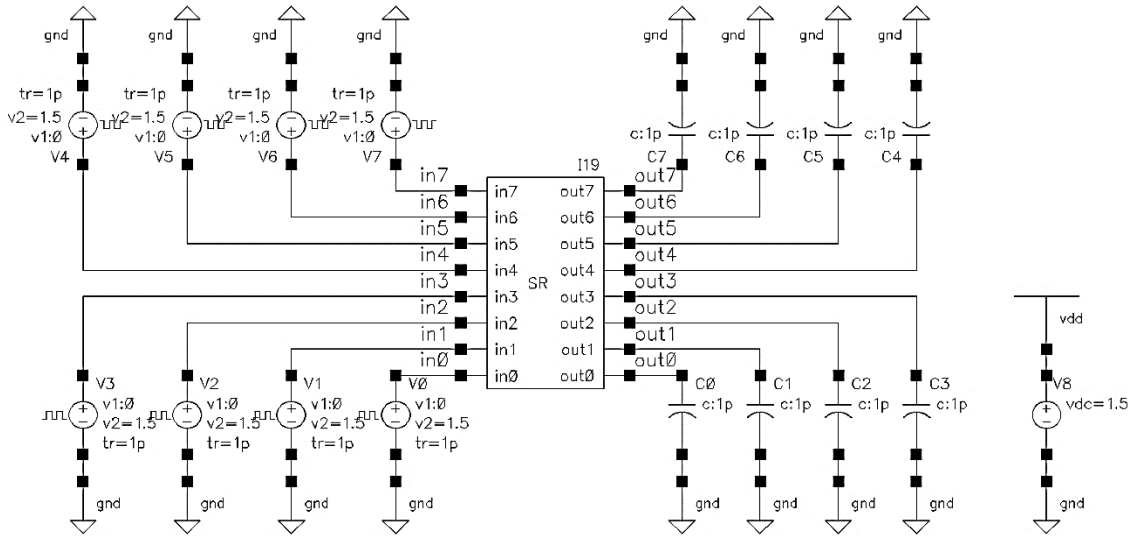


Obrázek 52: Schéma 8-bitového posunu doprava



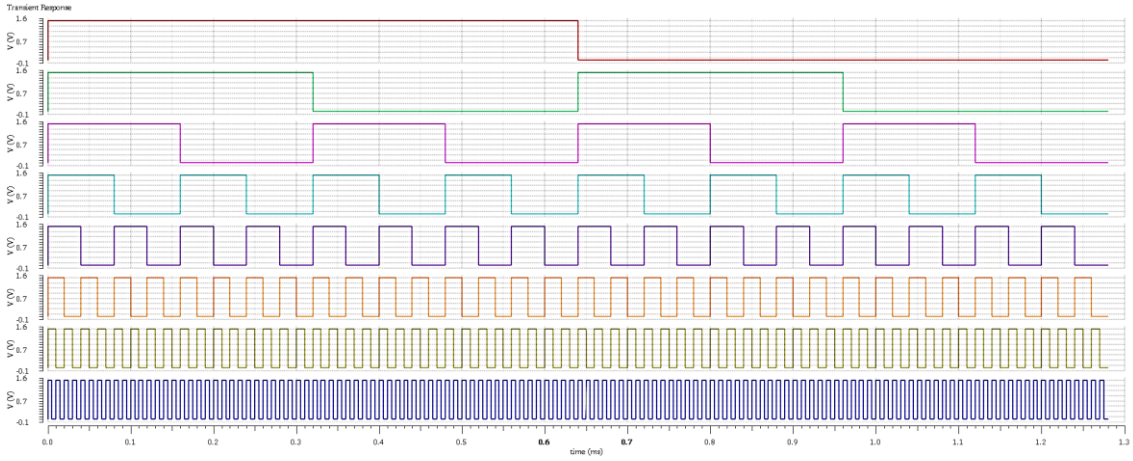
Obrázek 53: Symbol 8-bitového posunu doprava

Zapojení testovací simulace je stejné jako v případě simulace posunu doleva (viz obr. 49). Na vstupech jsou taktéž zapojeny zdroje pulsního napětí, každý má vždy dvojnásobnou periodu než předchozí.



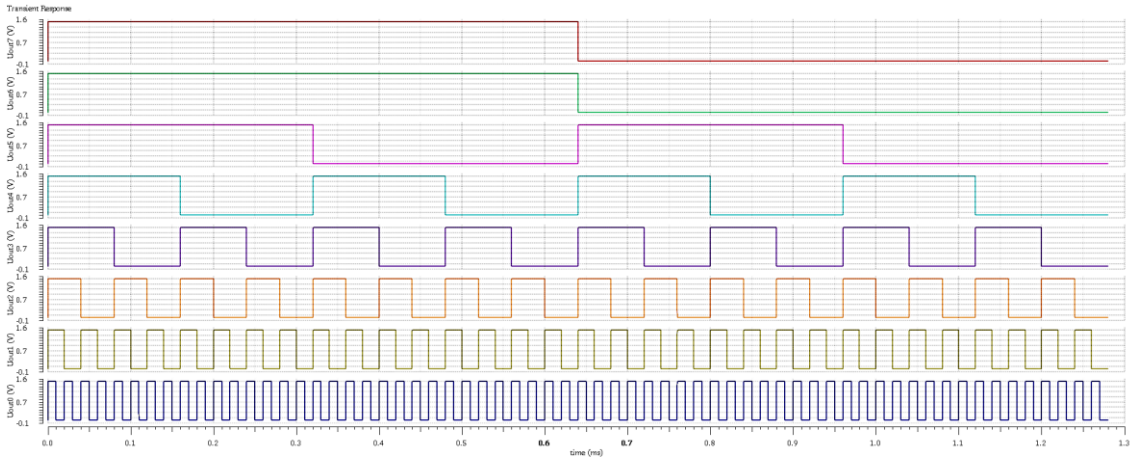
Obrázek 54: Simulace 8-bitového posunu doprava

Průběh vstupních signálů je zobrazen níže na obr. 55. Tento obrázek je totožný s obr. 50.



Obrázek 55: Průběh simulace 8-bitového posunu doprava – vstupní signály

Průběh výstupních signálů (ze shora dolů out7 až out0) přesně odpovídá předpokladům. To znamená, že signál out7 zůstal nezměněn, tedy znaménko bylo zkopírováno. Ostatní signály jsou posunuty o jedno místo doprava.



Obrázek 56: Průběh simulace 8-bitového posunu doprava - výstupní signály

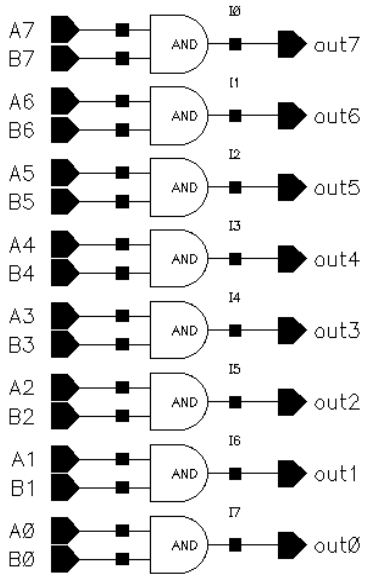
6.1.3 AND, OR, XOR

Další podporovanou operací je logický součin neboli operace AND [1]. Při operaci s dvěma jednobitovými vstupy má výstup hodnotu log. 1 pouze tehdy, když oba operandy měly také hodnotu log. 1, viz tabulka 5.

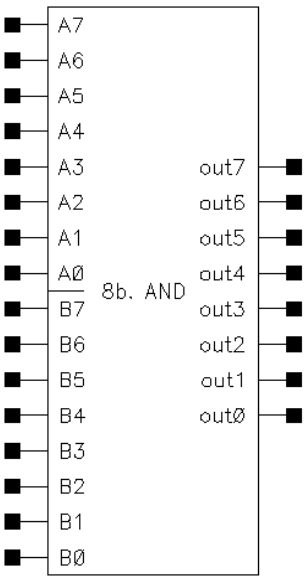
Tabulka 5: Pravdivostní tabulka operace AND

A	B	out
0	0	0
0	1	0
1	0	0
1	1	1

Tato operace je rozšířena na 8-bitová čísla. Na vstupu se nachází dva operandy *A[7:0]*, *B[7:0]*, výsledkem je opět 8-bitové číslo *out[7:0].* Pak je operace AND provedena mezi těmi bity z *A* a *B*, které se nachází na stejném pořadí, např. *out[7] = A[7] AND B[7], out[6] = A[6] AND B[6]*.

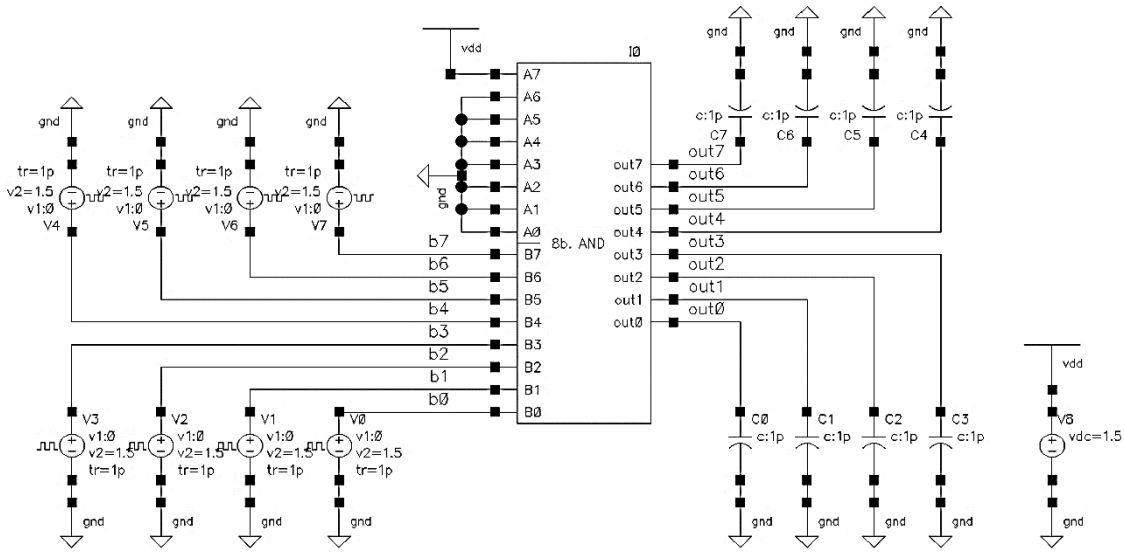


Obrázek 57: Schéma zapojení 8-bitového AND



Obrázek 58: Symbol 8-bitového AND

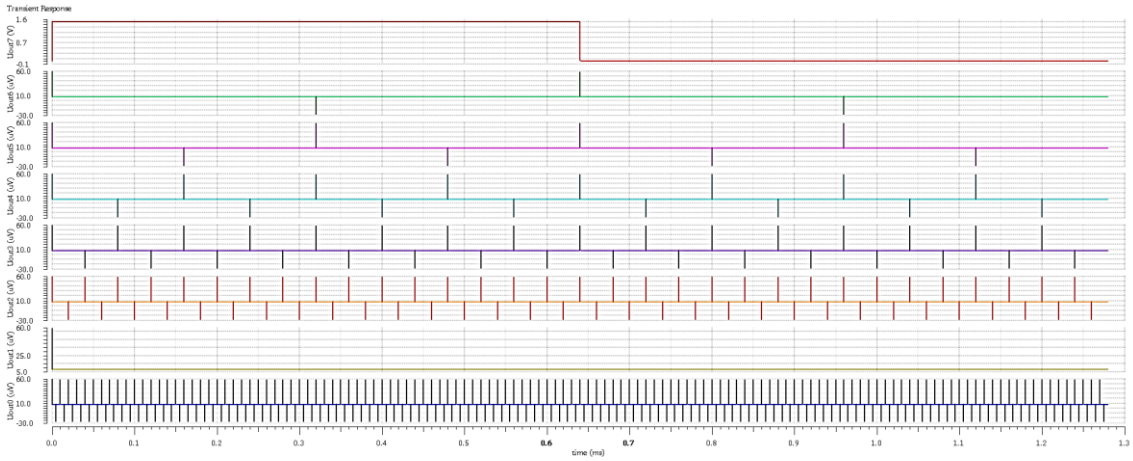
Při simulaci zapojení bylo využito faktu, že pokud při operaci AND má alespoň jeden z operandů úroveň log. 0, výsledek bude vždy úroveň log. 0. Naopak pokud má jeden z operandů úroveň log. 1, bude mít výsledek hodnotu druhého operandu – viz tabulka 5.



Obrázek 59: Simulace 8-bitového AND

Na vstup 8-bitového AND byl tedy zapojen vstupní signál *A[7:0]* mající stabilní hodnotu 1000 0000. Takovéto zapojení bude na výstup *out7* kopírovat nejvyšší bit druhého operandu, tj. *B[7]*, a všechny ostatní bity budou mít na výstupu úroveň log. 0. Pro ověření tohoto faktu ve všech možných případech byly na vstupy *B[7:0]* zapojeny zdroje pulsního napětí s násobky period.

Níže na obr. 60 je znázorněno, že opravdu pouze nejvyšší bit si zachoval v čase svou hodnotu. Ostatní bity mají úroveň log. 0, přestože se to z průběhu signálů tak nemusí zdát. Píky jsou ovšem v řádu µV, způsobené vzájemným ovlivněním náběžných a sestupných hran.



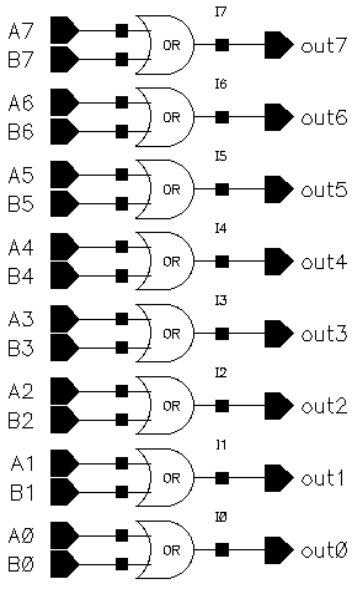
Obrázek 60: Průběh simulace 8-bitového AND

Obdobně jako logický součin je implementován i logický součet, tedy operace OR [1]. Tato operace pro dva bity produkuje výsledek log. 1 právě tehdy, když alespoň jeden z operandů má hodnotu log. 1. Pouze pokud mají oba operandy úroveň log. 0, má i výsledek úroveň log. 0.

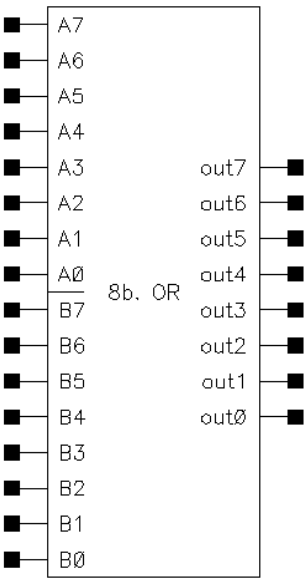
Tabulka 6: Pravdivostní tabulka operace OR

A	B	out
0	0	0
0	1	1
1	0	1
1	1	1

Procesor opět implementuje tuto operaci rozšířenou na 8-bitové operandy. Na jednotlivých pozicích *out* se vypíše logický součet stejných pozic operandů *A* a *B*.

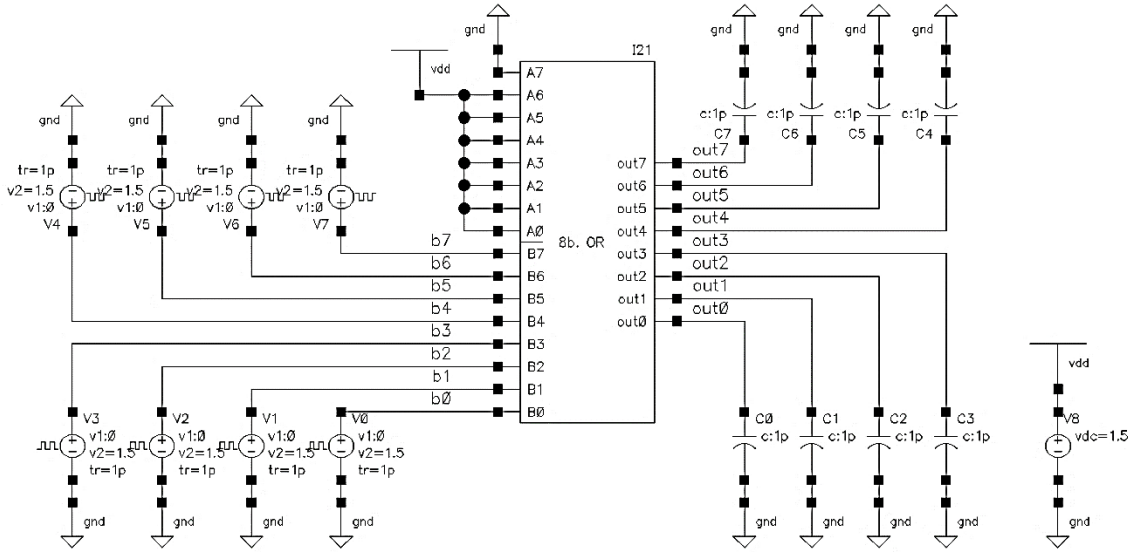


Obrázek 61: Schéma zapojení 8-bitového OR



Obrázek 62: Symbol 8-bitového OR

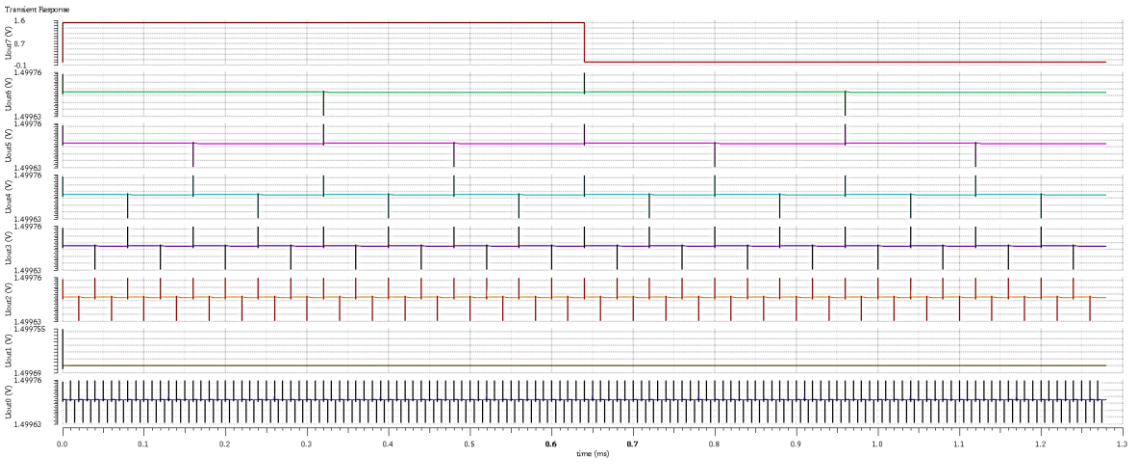
V simulaci je využito faktu, že pokud při operace OR má alespoň jeden z operandů úroveň log. 1, výsledek bude mít vždy úroveň log. 1. Naopak pokud má jeden z operandů úroveň log. 0, bude mít výsledek hodnotu druhého operandu – viz tabulka 6.



Obrázek 63: Simulace 8-bitového OR

Na vstup 8-bitového OR byl zapojen vstupní signál *A[7:0]* s trvalou hodnotou 0111 1111. Takovýto vstup bude mít za následek, že na výstup out7 bude zkopírován nejvyšší bit druhého operandu, tj. *B[7]*, a ostatní bity budou mít na výstupu úroveň log. 1. I v tomto případě byly pro ověření na vstupy *B[7:0]* zapojeny zdroje pulsního napětí s násobky period.

Na obr. 64 jsou zobrazeny průběhy signálů, z nichž je patrné, že nejvyšší bit si zachoval svůj průběh, zatímco ostatní bity mají úroveň log. 1 s píky v řádu µV.



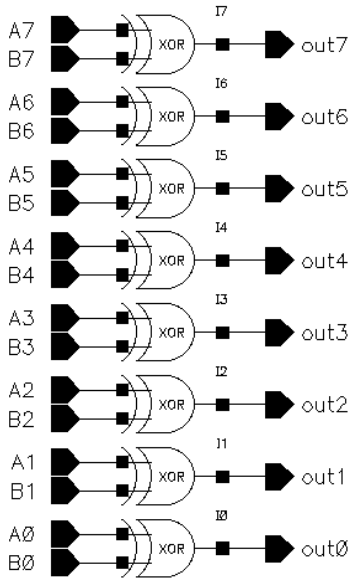
Obrázek 64: Průběh simulace 8-bitového OR

Poslední implementovanou logickou funkcí v ALU je XOR. Tato operace má výstupní hodnotu log. 1 právě tehdy, když *A* a *B* mají opačné vstupní úrovně.

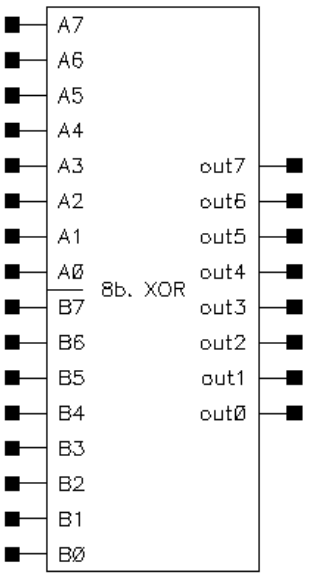
Tabulka 7: Pravdivostní tabulka operace XOR

A	B	out
0	0	0
0	1	1
1	0	1
1	1	0

Osmibitové rozšíření této operace znamená, výsledný bit má hodnotu log. 1 právě tehdy, když bity *A* a *B* na stejné pozici měly různou hodnotu. To má za důsledek, že pokud výsledné osmibitové číslo *out* má hodnotu 0000 0000, pak se operandy *A[7:0]* a *B[7:0]* rovnají.

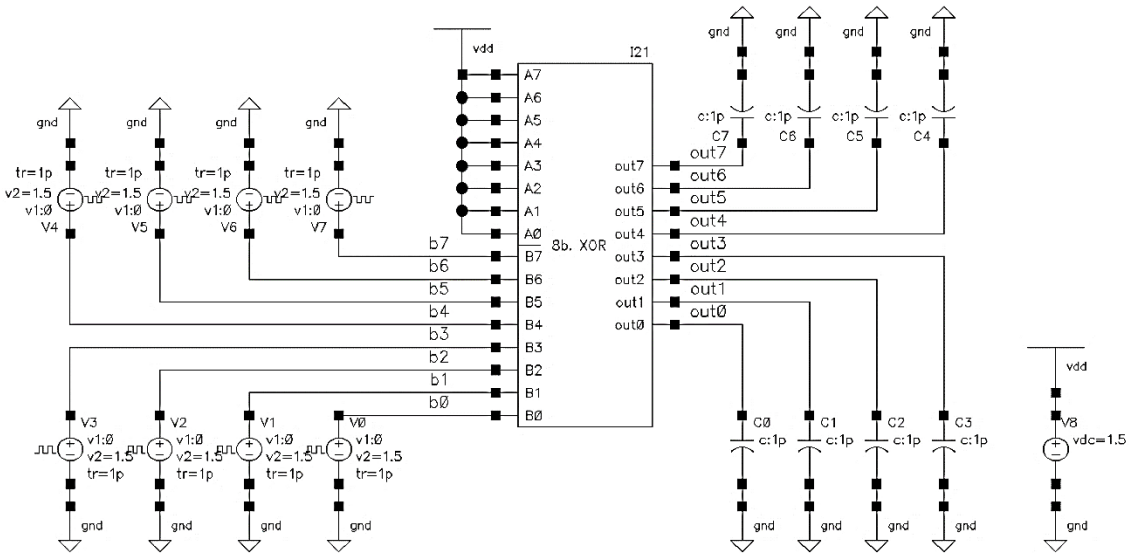


Obrázek 65: Schéma zapojení 8-bitového XOR



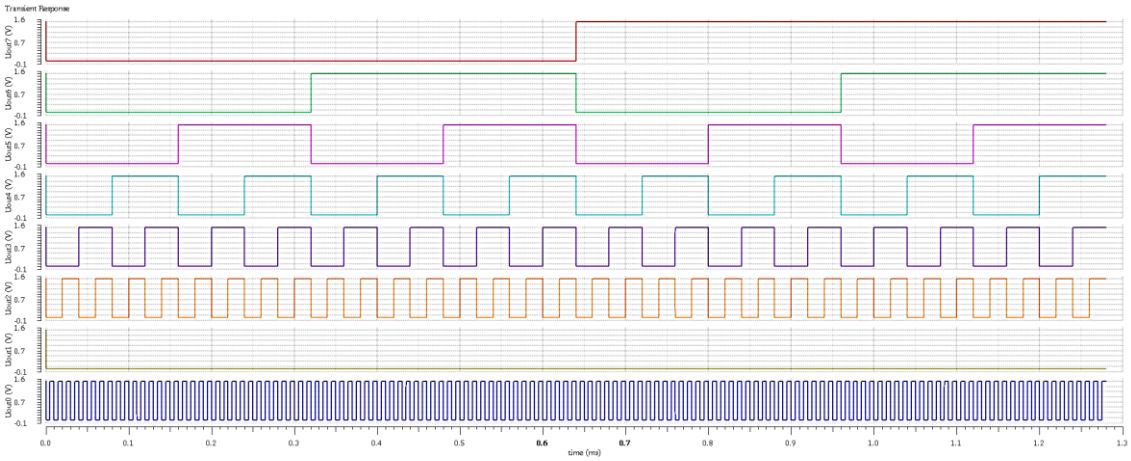
Obrázek 66: Symbol 8-bitového XOR

V simulačním schématu byly nastaveny všechny bity operandu A na úroveň log. 1. Při operaci XOR, pokud jeden z operandů má úroveň log. 1, pak se na výstupu zobrazí opačná hodnota než má druhý operand – viz tabulka 7.



Obrázek 67: Schéma zapojení simulace XOR

Na vstup 8-bitového XOR byl zapojen vstupní signál *A[7:0]* s trvalou hodnotou 1111 1111. Tento operand způsobí opačnou logickou úroveň druhého operandu. Jelikož na vstupy *B[7:0]* jsou zapojeny zdroje pulsního napětí s násobky period, dojde k otočení logických úrovní a v čase nebude hodnota *B[7:0]* klesat od 1111 1111 po 0000 0000, ale poroste od 0000 0000 po 1111 1111 – viz obr. 68.



Obrázek 68: Průběh simulace XOR

6.2 ZAPOJENÍ ALU

Aritmeticko-logická jednotka je součástka, ve které dochází k provedení aritmetické nebo logické operace nad konkrétními daty. Na vstupu jednotky se nachází 7 řídicích signálů – *ADD*, *SUB*, *SHIFT-LEFT*, *SHIFT-RIGHT*, *AND*, *OR*, *XOR*. Dále se na vstupu nachází dvě 8-bitové hodnoty *A[7:0]*, *B[7:0]* reprezentující konkrétní čísla v dvojkovém doplňku. V procesoru je ALU zapojena tak, že v jednom konkrétním okamžiku má maximálně jeden z řídicích signálů úroveň log. 1.

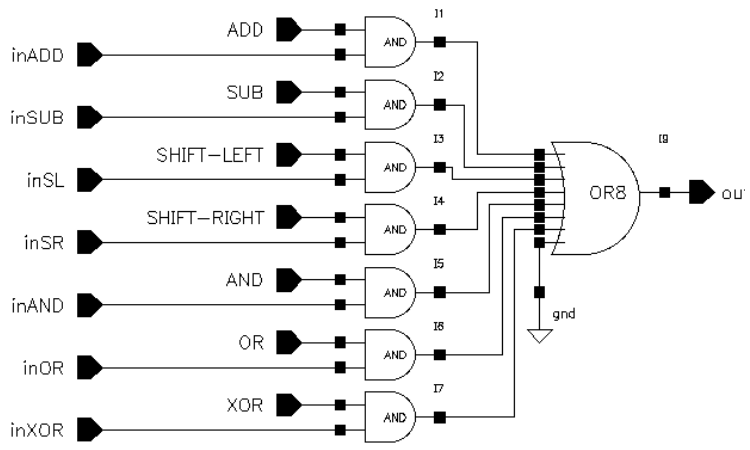
Schéma ALU se skládá z několika bloků. Hlavní část se skládá z bloků realizujících jednotlivé podporované operace. Tyto bloky jsou vysvětleny v kapitole 6.1.

Proces zpracování je následující. Nejprve jsou na vstup přivedeny typicky dvě 8-bitová čísla *A[7:0]* a *B[7:0]*. Tato čísla jsou přivedena na vstupy všech podbloků reprezentující jednotlivé matematické operace a výstupy z těchto operací jsou převedeny do „výstupního bloku“. Tento „výstupní blok“ vybírá, který z konkrétních výstupů bude přiveden na výstup z celé ALU, na základě detekce právě prováděné operace, tedy na základě nastavení řídicích signálů z tab. 8.

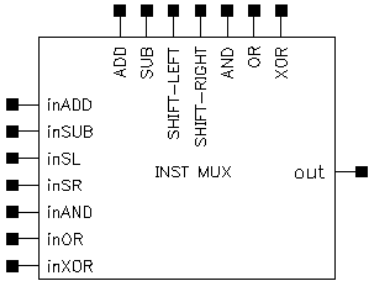
Tabulka 8: Přehled hodnot výstupních hodnot ALU v závislosti na řídicích signálech

stav řídicích signálů							**operace**	**výstupní signál**
ADD	SUB	SHIFT-LEFT	SHIFT-RIGHT	AND	OR	XOR		out[7:0]
1	0	0	0	0	0	0	OUT = A+B	inADD[7:0]
0	1	0	0	0	0	0	OUT = A-B	inSUB[7:0]
0	0	1	0	0	0	0	OUT = A << 1	inSL[7:0]
0	0	0	1	0	0	0	OUT = A >> 1	inSR[7:0]
0	0	0	0	1	0	0	OUT = A AND B	inADD[7:0]
0	0	0	0	0	1	0	OUT = A OR B	inOR[7:0]
0	0	0	0	0	0	1	OUT = A XOR B	inXOR[7:0]

Výstupní hodnota je vyhodnocována pro každý bit zvlášť. Na vstupu se nachází 7 řídicích signálů (*ADD*, *SUB*, *SHIFT-LEFT*, *SHIFT-RIGHT*, *AND*, *OR*, *XOR*) a 7 vstupních hodnot (*inADD*, *inSUB*, *inSL*, *inSR*, *inAND*, *inOR*, *inXOR*). Na konkrétní výstupní pozici se bude nacházet ta hodnota, jejíž řídicí signál je na vstupu nastaven na hodnotu log. 1.

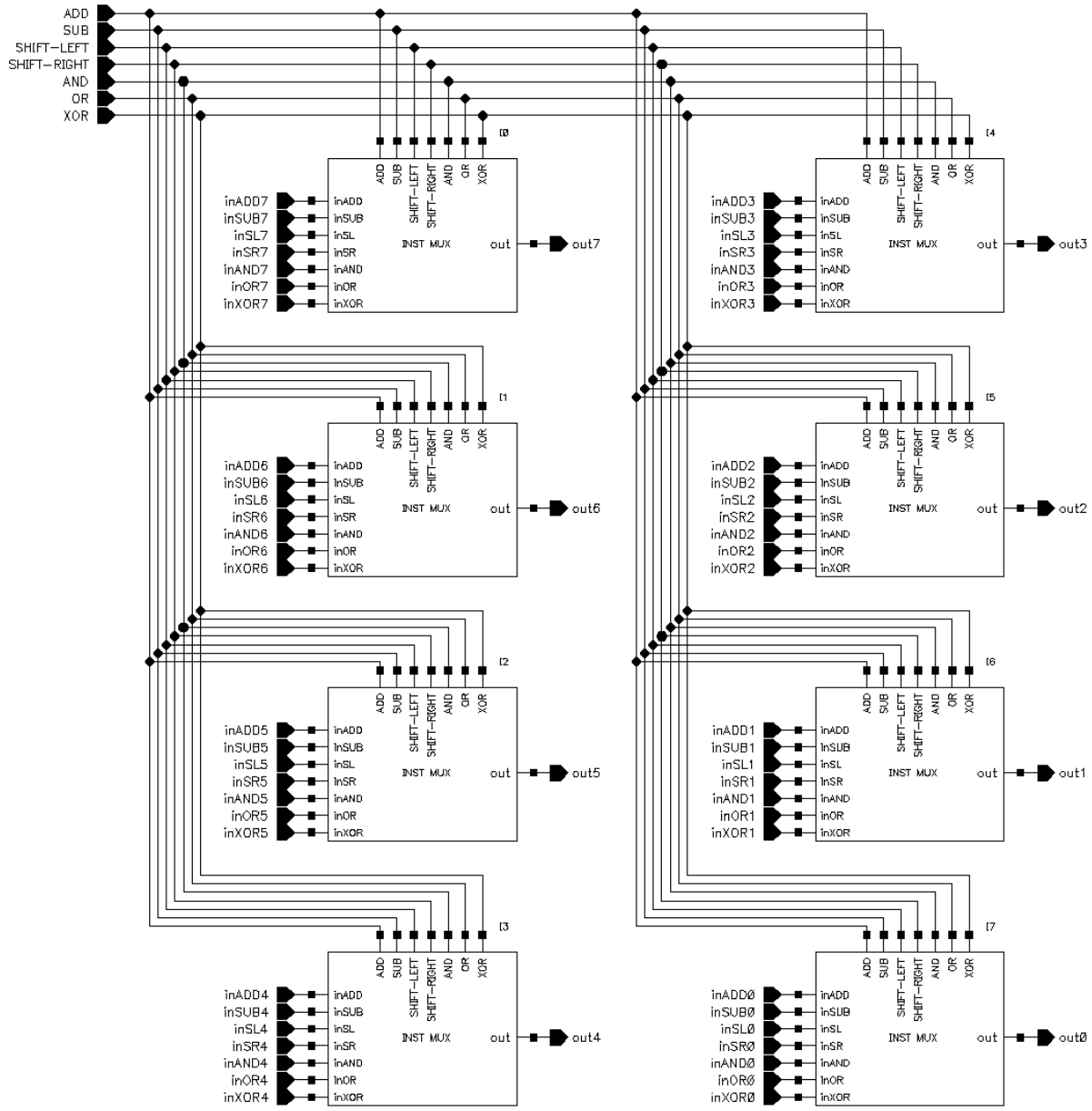


Obrázek 69: Schéma ALU multiplexoru pro jeden bit



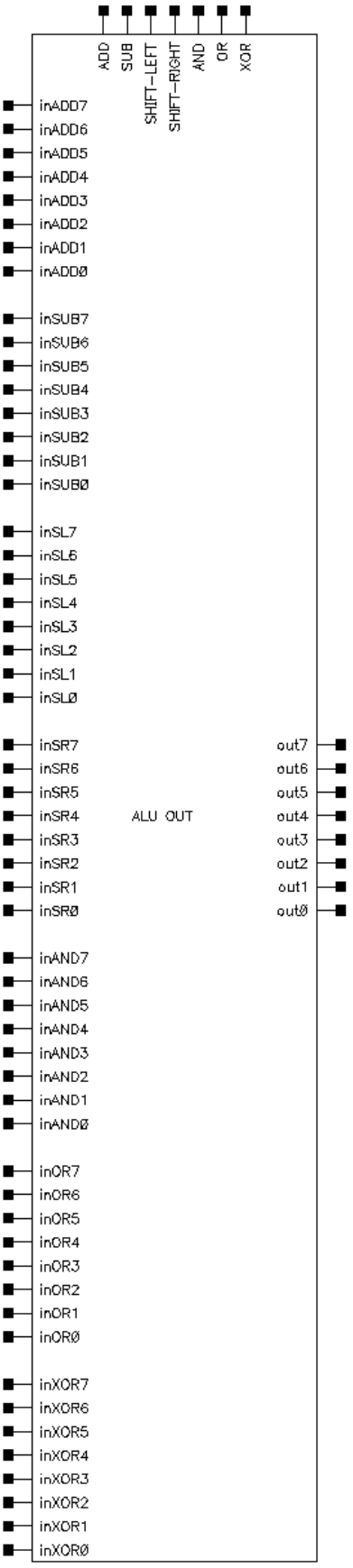
Obrázek 70: Symbol ALU multiplexoru pro jeden bit

Multiplexory pro jednotlivé bity jsou zapojeny do společného schématu a synchronizovány společnými řídicímu signály. Schéma tohoto zapojení je zobrazeno níže na obr. 71.



Obrázek 71: Schéma zapojení výstupního bloku ALU

Symbol výstupního bloku je zobrazen na obr. 72.

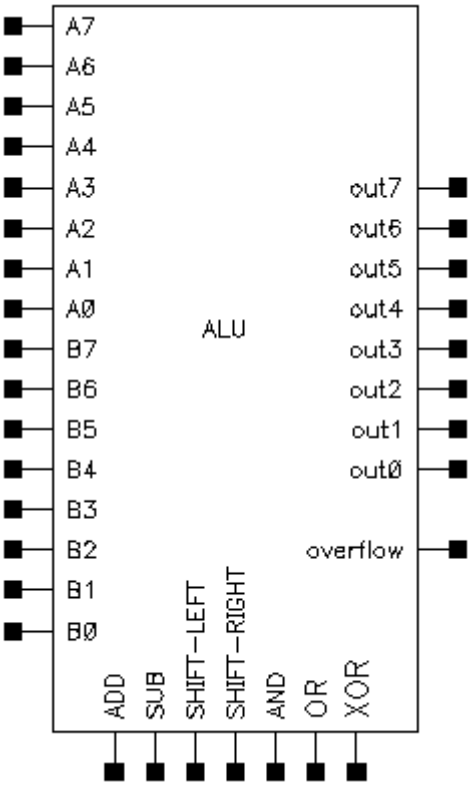


Obrázek 72: Symbol výstupního bloku ALU

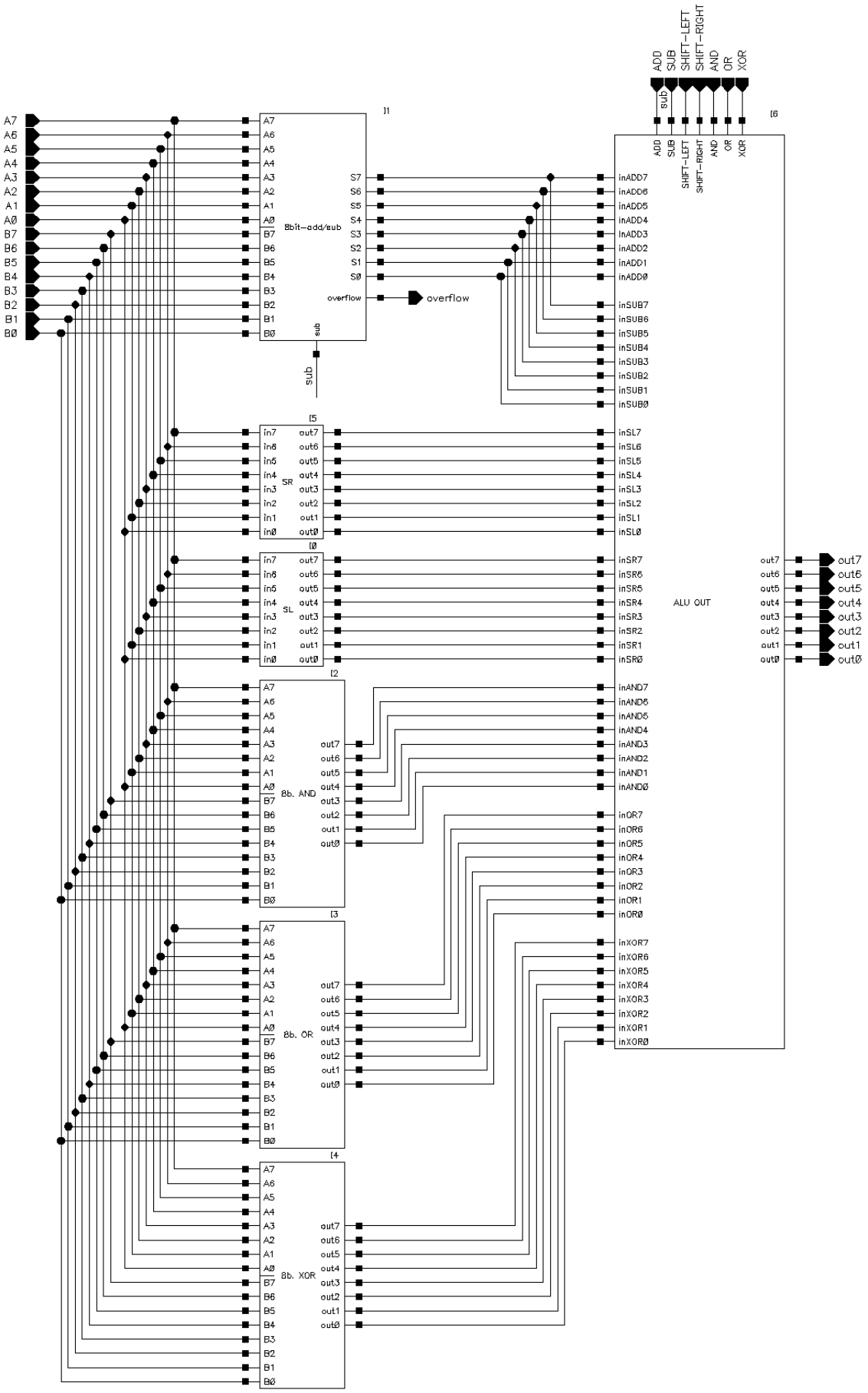
Ve schématu zapojení aritmeticko-logické jednotky se tedy nachází následující podbloky:

- 8-bitová sčítačka/odčítačka – viz obr. 43
- 8-bitový posun doleva – viz obr. 47
- 8-bitový posun doprava – viz obr. 52
- 8-bitový AND – viz obr. 51
- 8-bitový OR – viz obr. 55
- 8-bitový XOR – viz obr. 59

Do celé aritmeticko-logické jednotky jsou přivedeny vstupní signály *A[7:0]* a *B[7:0]*. Tyto hodnoty jsou rozvedeny do všech podbloků, výstupy z podbloků jsou přivedeny do výstupního bloku (viz obr. 71), který určuje konkrétní hodnotu na výstupu z jednotky.

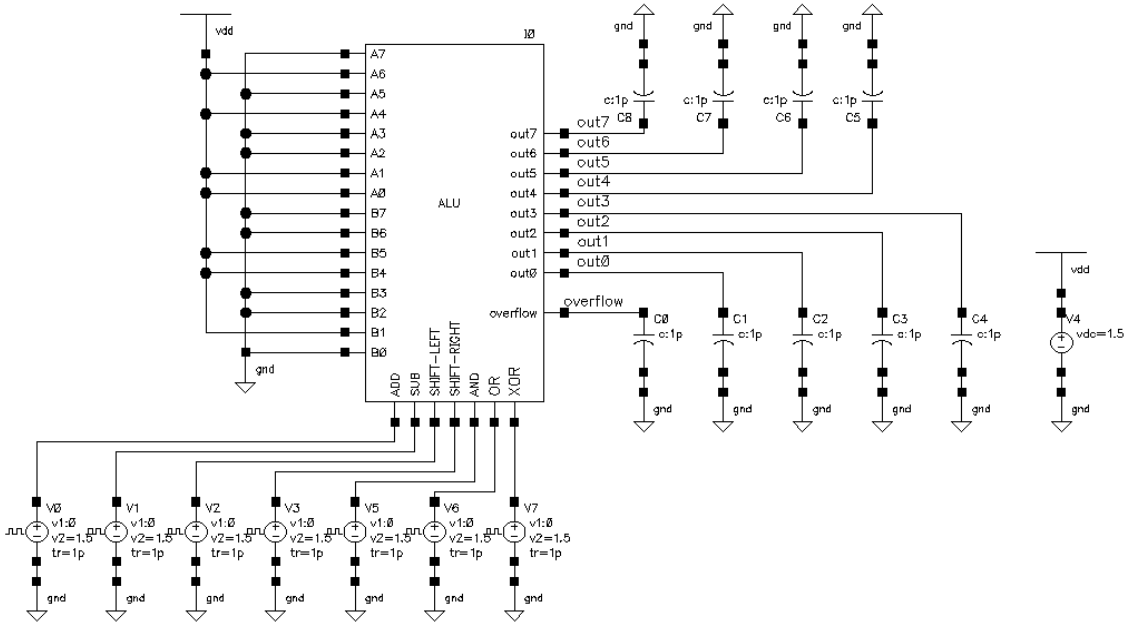


Obrázek 73: Symbol ALU



Obrázek 74: Schéma zapojení ALU

Stejně jako všechny ostatní součástky je důležité otestovat i ALU.



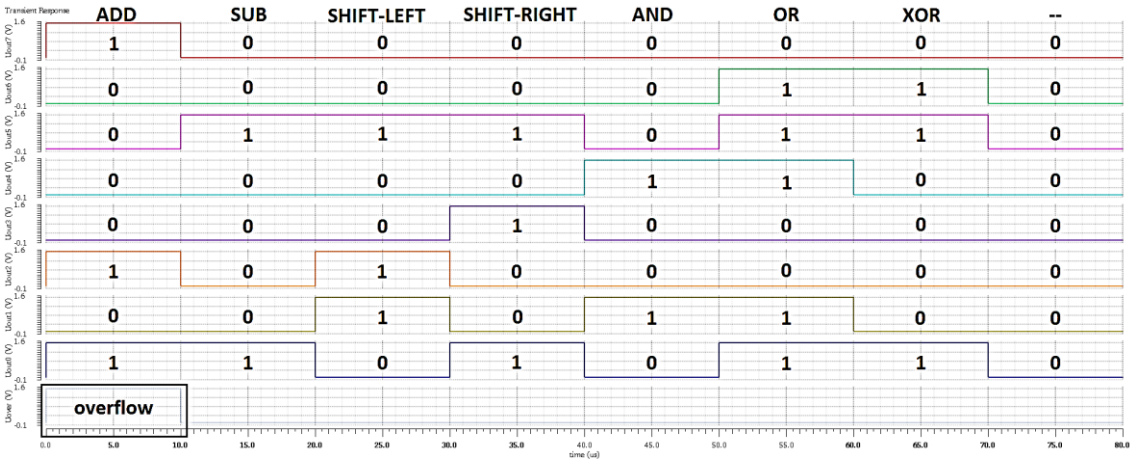
Obrázek 75: Schéma zapojení simulace ALU

Na datové vstupy byly připojeny konkrétní hodnoty: *A[7:0] = 0101 0011*, *B[7:0] = 0011 0010*.

Na řídicí vstupy signalizující konkrétní operaci byly přivedeny zdroje pulsního napětí následovně:

- ADD: perioda 80 µs, délka pulsu 10 µs, délka náběžné i sestupné hrany 1 ps
- SUB: perioda 80 µs, délka pulsu 10 µs, zpoždění 10 µs, délka náběžné i sestupné hrany 1 ps
- SHIFT-LEFT: perioda 80 µs, délka pulsu 10 µs, zpoždění 20 µs, délka náběžné i sestupné hrany 1 ps
- SHIFT-RIGHT: perioda 80 µs, délka pulsu 10 µs, zpoždění 30 µs, délka náběžné i sestupné hrany 1 ps
- AND: perioda 80 µs, délka pulsu 10 µs, zpoždění 40 µs, délka náběžné i sestupné hrany 1 ps
- OR: perioda 80 µs, délka pulsu 10 µs, zpoždění 50 µs, délka náběžné i sestupné hrany 1 ps
- XOR: perioda 80 µs, délka pulsu 10 µs, zpoždění 60 µs, délka náběžné i sestupné hrany 1 ps

Níže na obr. 76 je zobrazen průběh výstupních signálů v čase *out[7:0]*. V konkrétním časovém okamžiku je vždy aktivní pouze jeden z řídicích signálů definující typ operace, poslední časový úsek není aktivní žádný. To má za důsledek, že na výstupu je zobrazen výsledek v závislosti na konkrétní zvolené matematické operaci. V posledním časovém úseku má výsledek hodnotu 0000 0000, jelikož nebyl zvolen žádný typ operace.

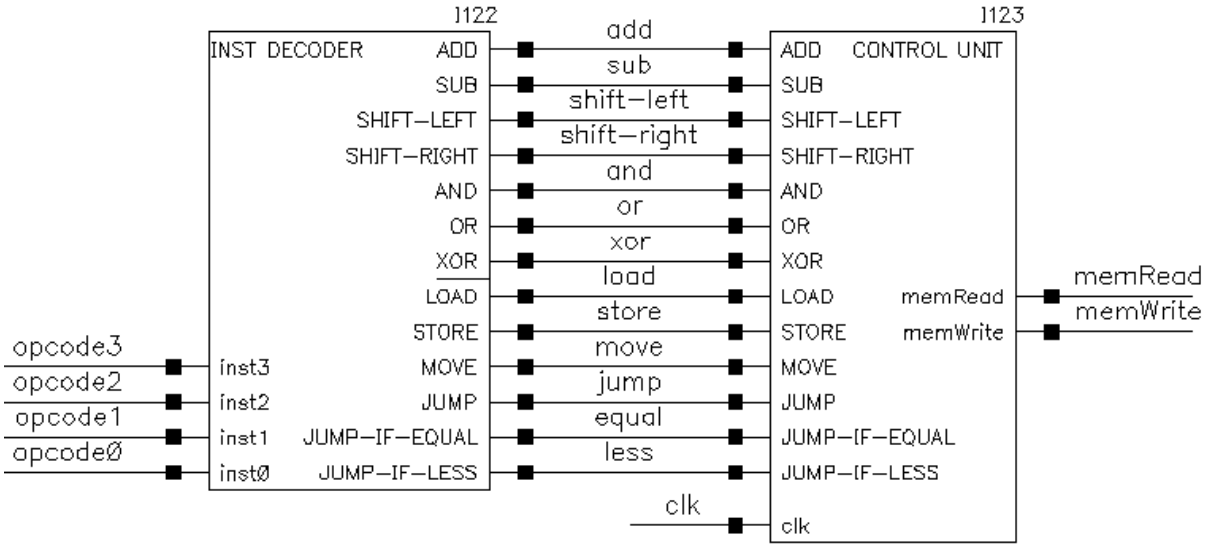


Obrázek 76: Průběh simulace ALU

7 ŘADIČ

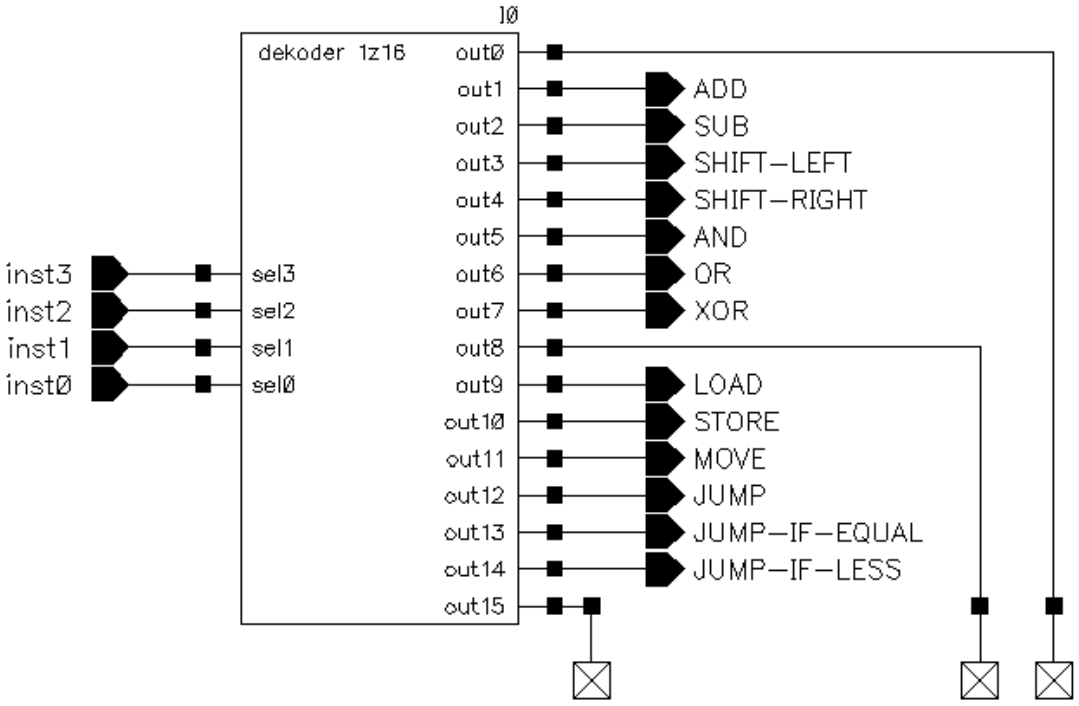
Řadič (control unit) je součást procesoru starající se o rozvrhování vykonávání instrukcí. [6]

Běh programu v procesoru je v této práci synchronizovaný pomocí hodinového signálu v závislosti na typu právě běžící instrukce. Proto pod pojmem řadič se rozumí část procesoru skládající se z dekodéru instrukce a kontrolní jednotky.



Obrázek 77: Schéma zapojení řadiče

První samostatná jednotka tvořící řadič je dekodér instrukce. Pomocí vstupního signálu inst[3:0] značícího opcode, neboli horní 4 bity v binárně zakódované instrukce, je dekódován typ instrukce. Seznam podporovaných instrukcí a jejich opcode je uveden v kapitole 3 v tabulce 1.



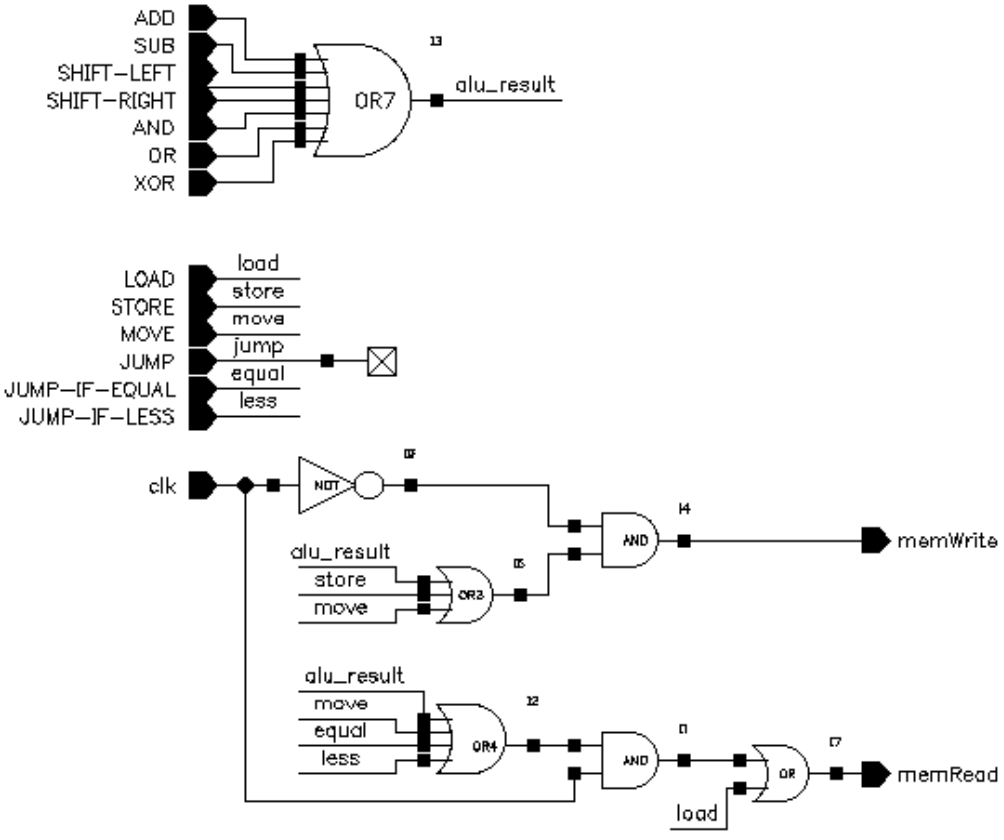
Obrázek 78: Schéma dekodéru instrukce

Zapojení dekodéru instrukce je tvořeno stejným dekodérem, jaký je použit k dekódování adresy v paměti v kapitole 4.2.1 a jednotlivé výstupy z dekodéru jsou připojeny na výstup součástky. V praxi tedy došlo k přejmenování výstupních pinů u implementovaných instrukcí. Ty výstupy z dekodéru, jejichž vstupní kombinace neodpovídá implementované instrukci, jsou připojeny na součástku noConn z knihovny basic. To znamená, že výstup zůstal nezapojen.

Druhá součástka tvořící řadič je kontrolní jednotka, která řídí přístup do datové paměti. Na vstupu má 13 vstupních signálů (*ADD*, *SUB*, *SHIFT-LEFT*, *SHIFT-RIGHT*, *AND*, *OR*, *XOR*, *LOAD*, *STORE*, *MOVE*, *JUMP*, *JUMP-IF-EQUAL*, *JUMP-IF-LESS*). Tyto signály jsou přivedeny z dekodéru instrukce výše a značí druh právě zpracovávané instrukce.

V případě instrukce *STORE* pouze k zápisu do datové paměti. Naopak v případě instrukce *LOAD* dochází pouze ke čtení z datové paměti, stejně jako u skokových instrukcí (*JUMP*, *JUMP-IF-EQUAL*, *JUMP-IF-LESS*). U ostatních instrukcí, tedy u veškerých matematických instrukcí a u instrukce *MOVE* dochází ke čtení z i k zápisu do datové paměti.

Čtení i zápis jsou synchronizovány hodinovým signálem. K zápisu do datové paměti dochází vždy se sestupnou hranou hodin. Pokud se zpracovává instrukce typu *LOAD*, je čtení povoleno po celou periodu hodinového signálu. U ostatních instrukcí dochází ke čtení s náběžnou hranou hodin a je povoleno po celou dobu, kdy má hodinový signál kladnou úroveň.



Obrázek 79: Schéma zapojení kontrolní jednotky

Řadič je součástí schématu procesoru a není pro něj vytvořena samostatná symbolická značka. Je to z toho důvodu, že signály pro jednotlivé instrukce jsou použity i např. v ALU. Proto je otestována jeho činnost až v rámci zapojení v procesoru.

8 PROCESOR – KOMPLEXNÍ SIMULACE

8.1 SEKVENČNÍ PROCESOR S ČÍTAČEM INSTRUKCÍ

První simulace je založena na nejjednodušším typu sekvenčního procesoru, který nepodporuje skokové instrukce a instrukce jsou čteny z instrukční paměti postupně za sebou. Výběr instrukce je řízen čítačem instrukcí a délka zpracování instrukce je jeden hodinový cyklus. Protože instrukční paměť je konstruována na 16 instrukcí, maximální doba běhu programu je 16 hodinových cyklů. V programech zkratka Mx reprezentuje paměťovou jednotku v datové paměti pod adresou x, zkratka Ix reprezentuje instrukci uloženou v instrukční paměti pod adresou x.

8.1.1 Program

Na ověření funkce procesoru byl zvolen následující jednoduchý program založený na vytvořených instrukcích.

```
I0      STORE   M1, 45
I1      STORE   M2, 28
I2      ADD     M3, M2, M1
I3      MOVE    M4, M3
I4      LOAD    M4
```

Tento program nejprve načte číslo 45 do datové paměti na adresu 1. Dále načte číslo 28 do datové paměti na adresu 2. Poté obě čísla sečte a uloží na adresu 3 a v následujícím kroku číslo z adresy 3 uloží na adresu 4. V posledním kroku je vyvedeno na výstup číslo uložené právě pod adresou 4, tedy číslo 73.

8.1.2 Instrukční paměť

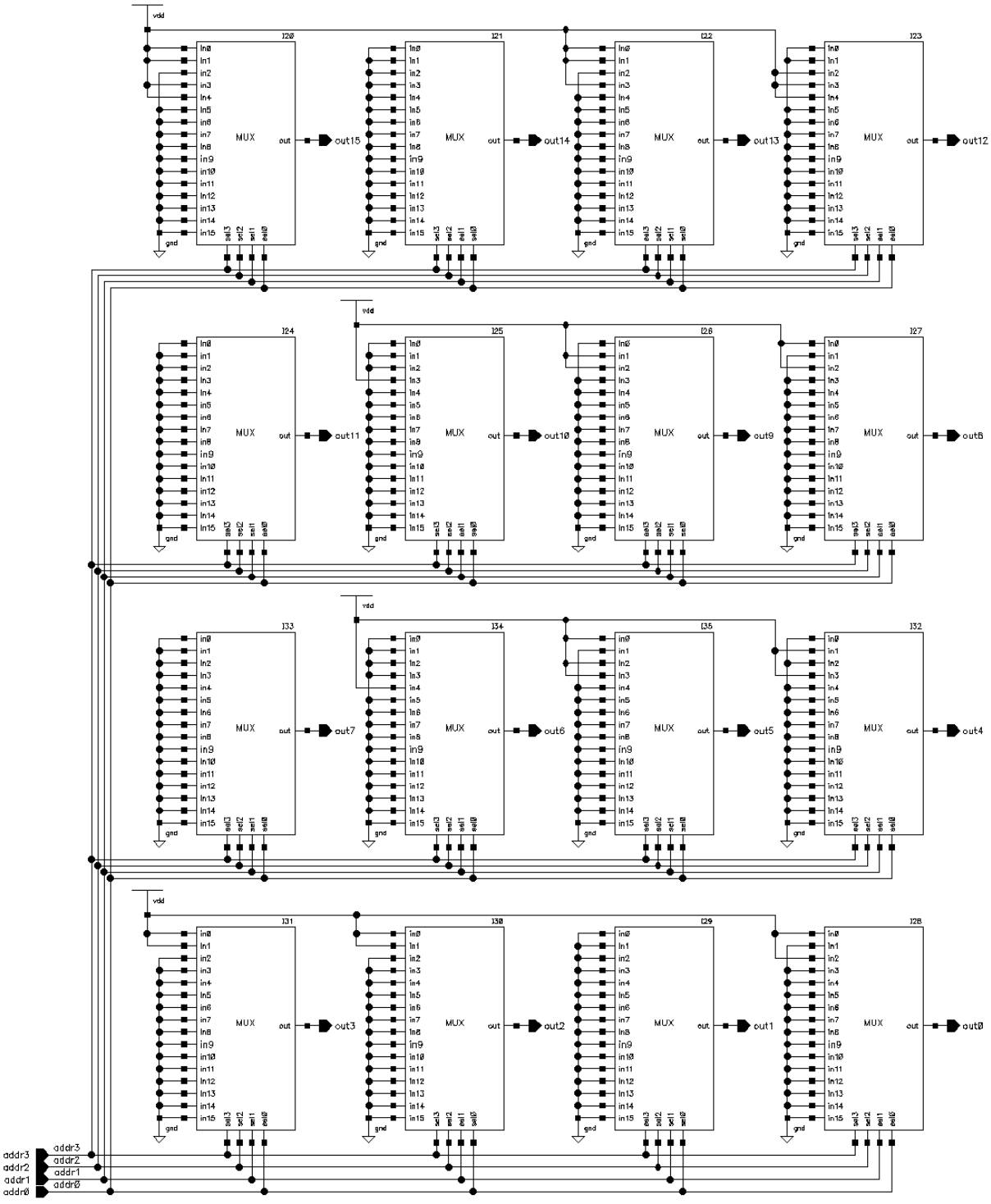
Protože je program procesoru předem daný a pevně uložený v instrukční paměti, není použita uživatelsky přístupná instrukční paměť z kapitoly 5, ale pouze část starající se o čtení instrukcí.

Instrukce z programu byly konvertovány do binárního kódu a uloženy na adresy 0 až 4 následovně:

Tabulka 9: Uložení zakódovaných instrukcí v instrukční paměti – sekvenční program s čítačem instrukcí

addr[3:0]	zakódovaná instrukce v binárním kódu
0000	1010 0001 0010 1101
0001	1010 0010 0001 1100
0010	0001 0011 0010 0001
0011	1011 0100 0011 0000
0100	1001 0000 0100 0000
0101 až 1111	0000 0000 0000 0000

Instrukční paměť tohoto procesoru se tedy skládá z 16 multiplexorů, na jejichž vstupech jsou hodnoty vdd a gnd (log. 1 a log. 0) v takovém pořadí, aby při zadání adresy instrukce na vstupy *addr[3:0]* byla na výstupu zobrazena binární hodnota zakódované instrukce uložené na dané adrese.

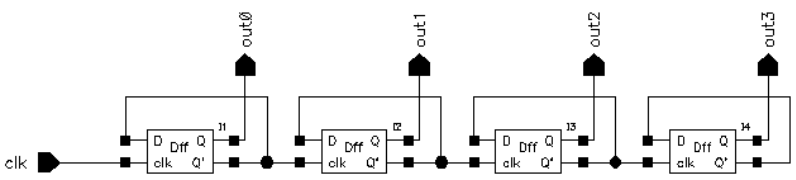


Obrázek 80: Schéma zapojení instrukční paměti sekvenčního procesoru s čítačem instrukcí

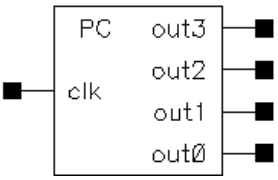
8.1.3 Zapojení procesoru

Funkce procesoru je založena na čítači instrukcí. Tento čítač se skládá ze čtyř sekvenčně zapojených DFF. Na vstup prvního klopného obvodu je přiveden hodinový signál *clk*, u ostatních DFF je vždy na místo hodinového signálu přiveden negovaný výstup *Q'* z předchozího DFF. Výstupy z klopného obvodu *Q* jsou vyvedeny na výstup *out[3:0]* a zároveň jsou propojeny se vstupy *D*.

Toto zapojení má za následek, že výstup z prvního DFF tedy *out0* bude kopírovat hodinový signál s dvojnásobnou periodou, obdobně *out1* se čtyřnásobnou periodou, *out2* s osminásobnou periodou a *out3* s šestnáctinásobnou periodou. Ve výsledku se tedy s každou náběžnou hranou původního hodinového signálu *clk* bude na výstupu *out[3:0]* generovat posloupnost 0000 až 1111.

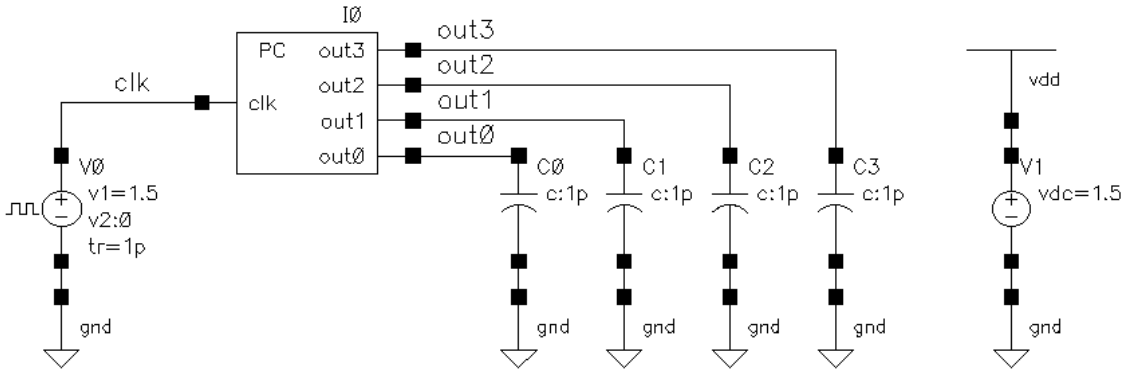


Obrázek 81: Schéma čítače instrukcí



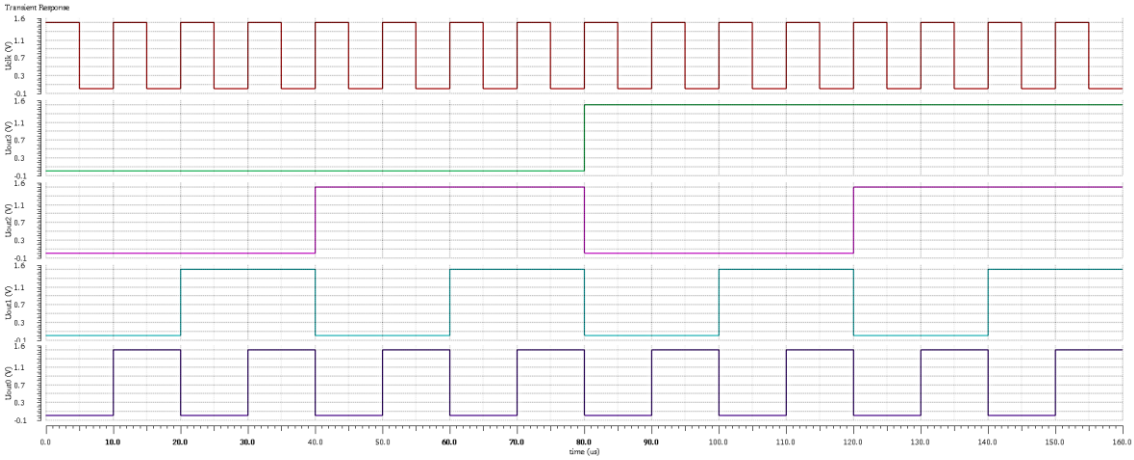
Obrázek 82: Symbol čítače instrukcí

Pro simulaci je na vstup čítače zapojen hodinový signál s periodou 10 µs, délkou pulsu 5 µs a délkou nástupné i sestupné hrany 1 ps.



Obrázek 83: Zapojení simulace čítače instrukcí

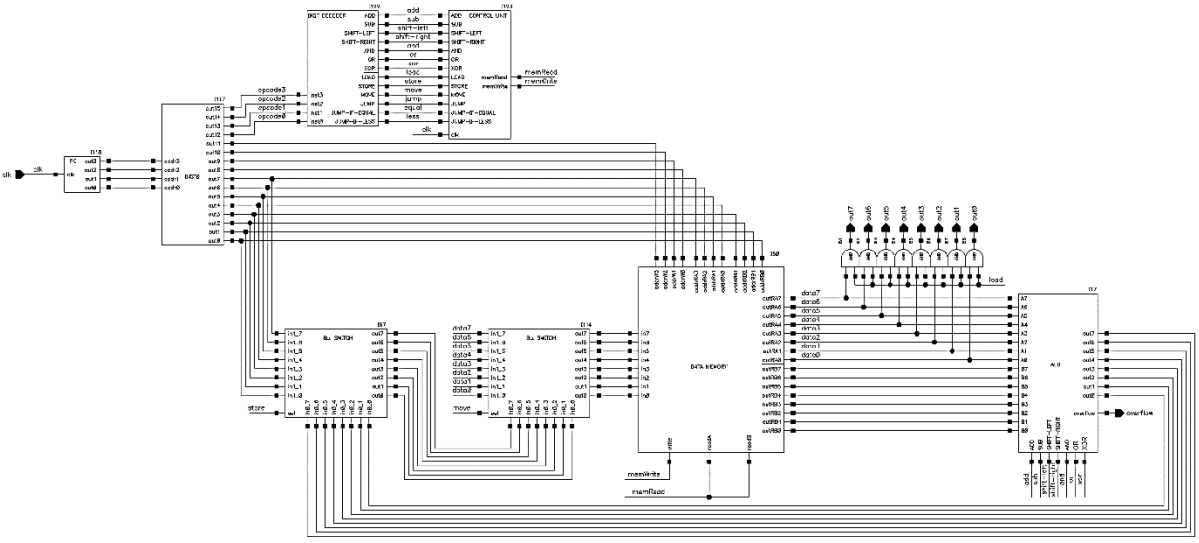
Na výstupu jsou zobrazeny pulsní signály – *out0* s dvojnásobnou, *out1* s čtyřnásobnou, *out2* s osminásobnou a *out3* s šestnáctinásobnou periodou oproti původnímu hodinovému signálu.



Obrázek 84: Průběh simulace čítače instrukcí

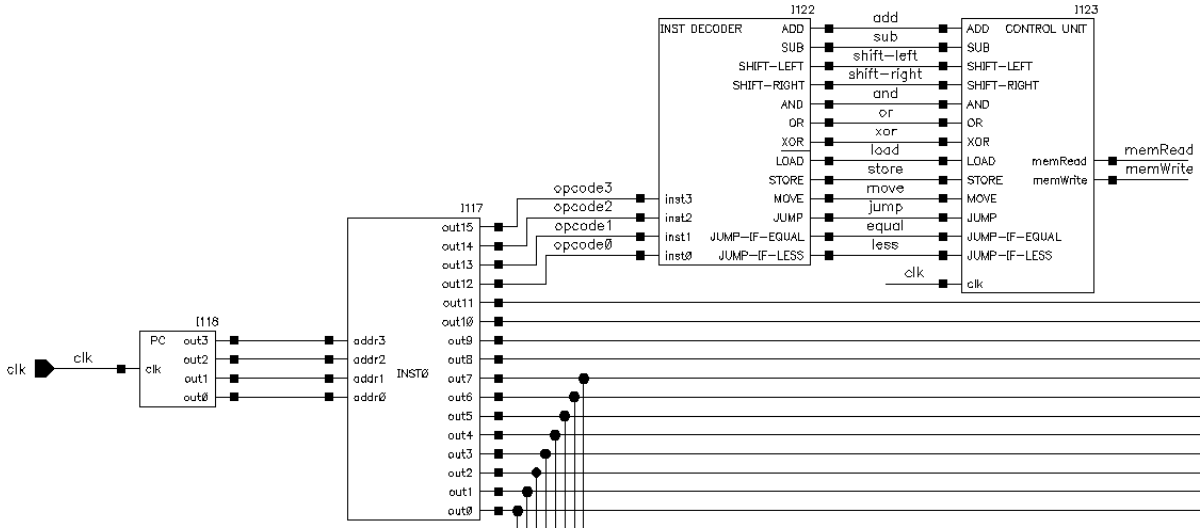
Výstup z tohoto čítače instrukcí je přiveden na vstup instrukční paměti, čímž dochází k postupnému sekvenčnímu vyčítání instrukcí na příslušných adresách.

Kromě čítače instrukcí a instrukční paměti obsahuje procesor řadič, datovou paměť a ALU. Níže na obr. 85 je zobrazeno kompletní zapojení procesoru.



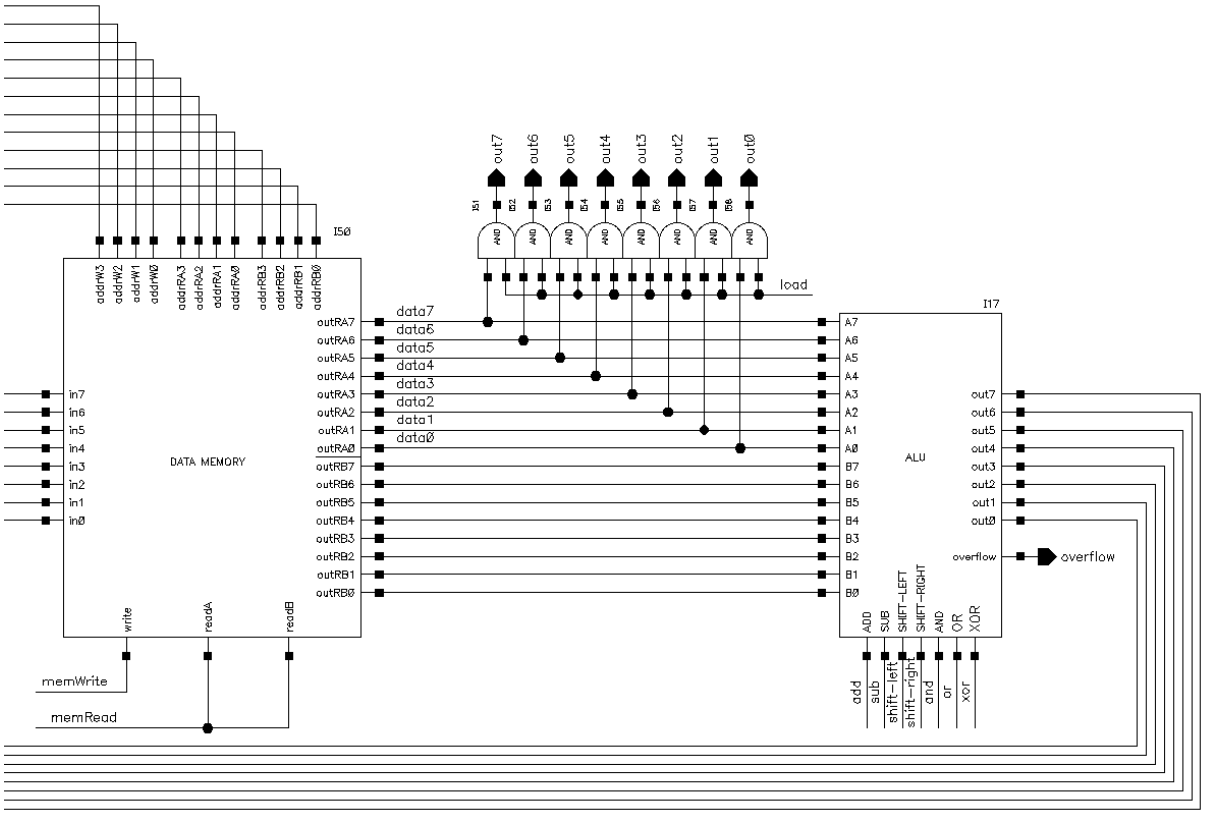
Obrázek 85: Schéma zapojení procesoru s čítačem instrukcí

První část procesoru se stará o správné zpracování instrukcí. Hodinový signál *clk* je připojen přímo na vstup čítače instrukcí, který s každou náběžnou hranou hodin vygeneruje adresu, z níž má být přečtena instrukce z instrukční paměti. Instrukce je vedena do řadiče, jenž dekóduje konkrétní typ instrukce na základě jejího opcode a povoluje zápis do datové paměti a čtení z ní.



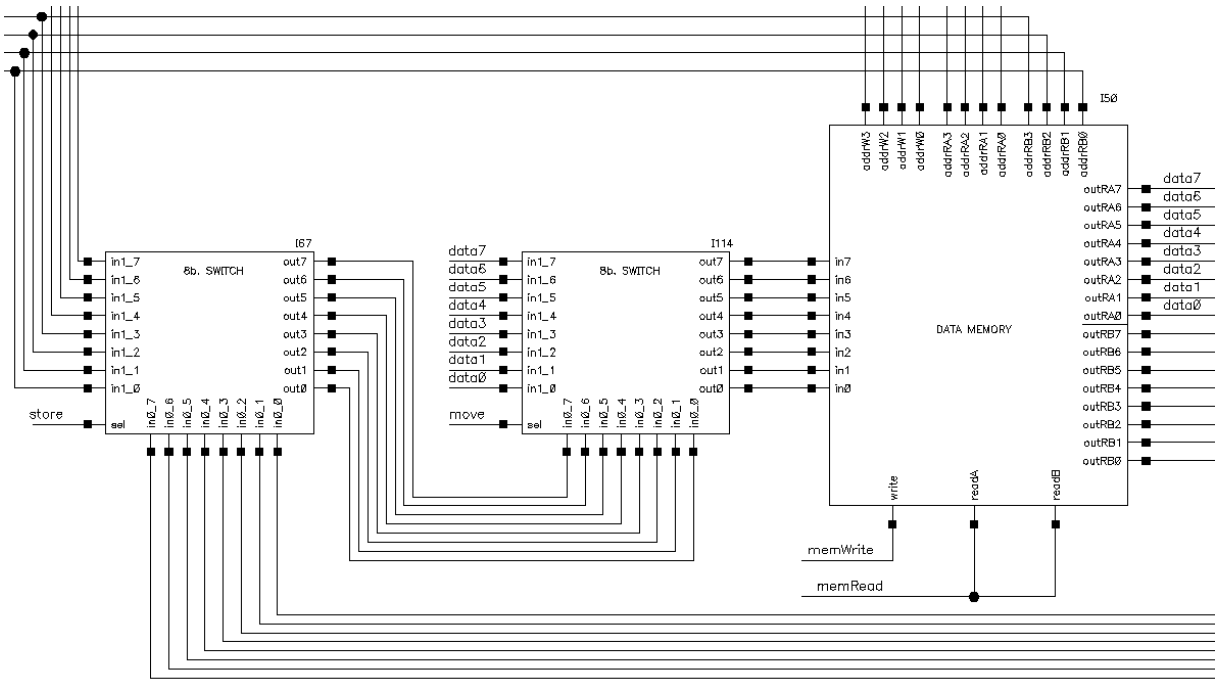
Obrázek 86: Schéma zapojení procesoru s čítačem instrukcí – výřez – instrukční paměť, čítač instrukcí, řadič

Dále jsou adresy jednotlivých operandů přivedeny na vstup datové paměti, dle typu instrukce. Z datové paměti jsou přečtena konkrétní data, která jsou přivede do vstup ALU. V případě instrukce typu *LOAD* je hodnota přečtená z paměti pouze vyvedena na výstup. Pokud se jedná o některou z matematických operací, výsledek z ALU je přiveden zpětnou vazbou na vstup datové paměti a se sestupnou hranou hodin je zapsán do paměti.



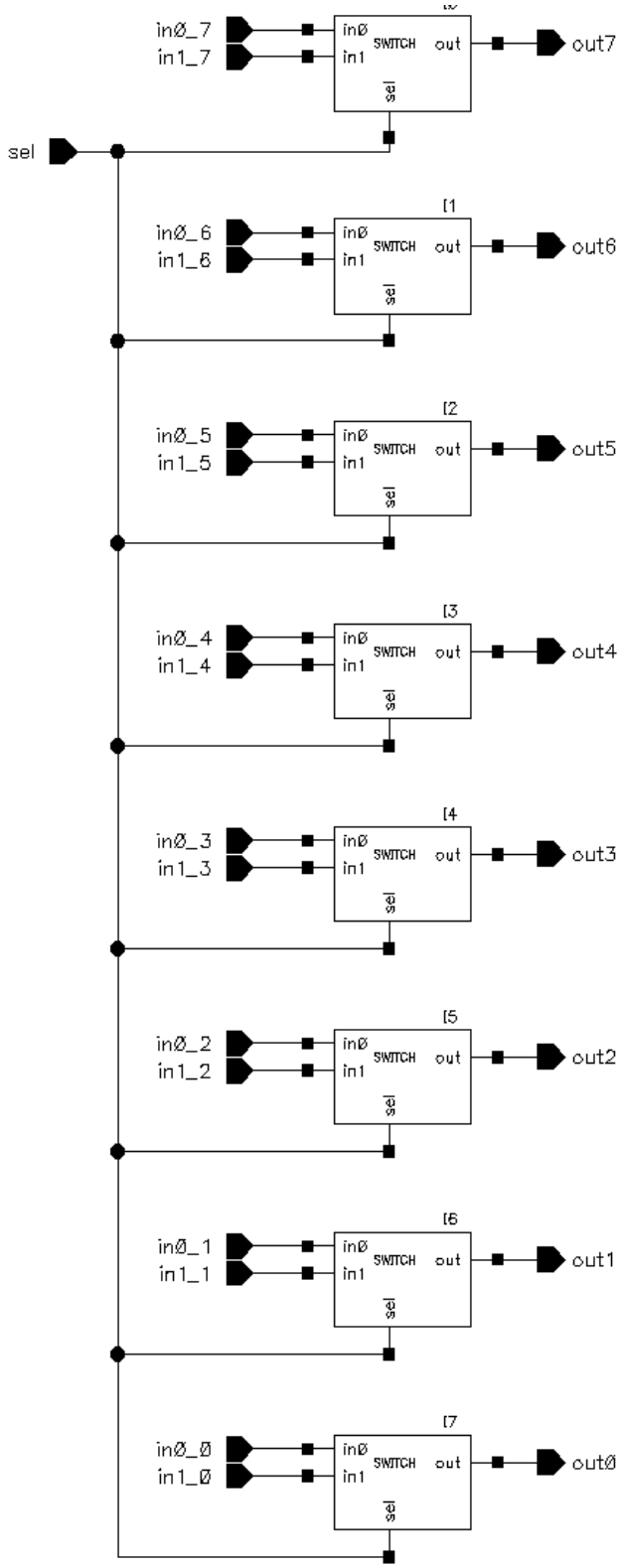
Obrázek 87: Schéma zapojení procesoru s čítačem instrukcí – výřez – datová paměť, ALU, výstup

Dalším případem je instrukce typu *MOVE*, kdy je do paměti překopírována hodnota z jiné paměťové jednotky. Poslední možností je instrukce typu *STORE*, při níž je do datové paměti zapsána přímo číselná hodnota, která je součástí instrukce.

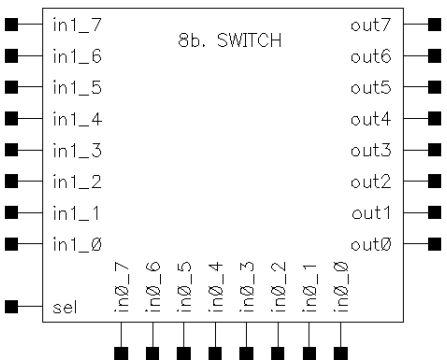


Obrázek 88: Schéma zapojení procesoru s čítačem instrukcí – výřez – datová paměť, switch

Ve schématu na obr. 88 je použit 8-bitový přepínač (switch), který přepíná mezi dvěma 8-bitovými hodnotami *in0_[7:0]* a *in1_[7:0]* v závislosti na řídicím signálu sel. Jestliže má sel úroveň log. 0, bude na výstupu zobrazena hodnota *in0_[7:0]*, v opačném případě *in1_[7:0].*

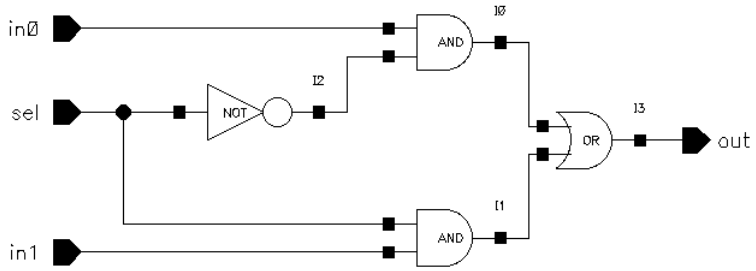


Obrázek 89: Schéma zapojení 8-bitového přepínače

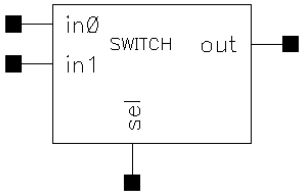


Obrázek 90: Symbol zapojení 8-bitového přepínače

Přepínač je tvořen osmi 1-bitovými přepínači, které jsou synchronizovány pomocí řídicího vodiče *sel*. V důsledku toho jsou všechny bity přepínány současně.



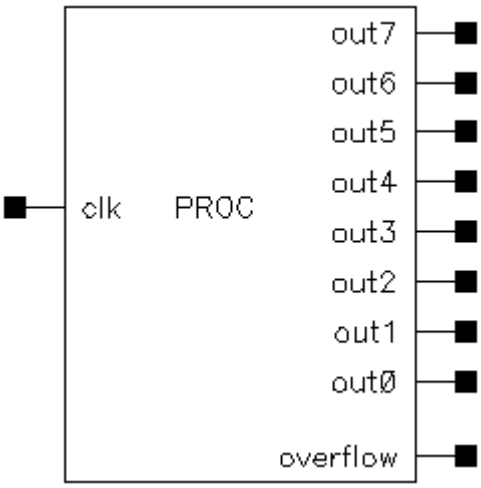
Obrázek 91: Schéma zapojení 1-bitového přepínače



Obrázek 92: Symbol 1-bitového přepínače

Ze schématu 1-bitového přepínače je zřejmé, že pokud má řídicí signál *sel* úroveň log. 0, na výstupu bude zobrazena hodnota *in0*. Naopak pokud má řídicí signál *sel* úroveň log. 1, na výstupu bude zobrazena hodnota *in1*.

Aby bylo možné procesor otestovat, je nutné vytvořit jeho schématickou značku. Protože jeho celá funkce je ovládána vestavěnou instrukční pamětí a hodinovým signálem, má procesor pouze jediný vstupní pin, a to právě hodinový signál *clk*. Na výstupu bude zobrazenou 8-bitové číslo *out[7:0]*, v případě, že právě zpracovávaná instrukce bude typu *LOAD*. Posledním výstupním signálem je kontrolní signál overflow signalizující přetečení při sčítání nebo odčítání (viz kapitola Sčítání, odčítání6.1.1).

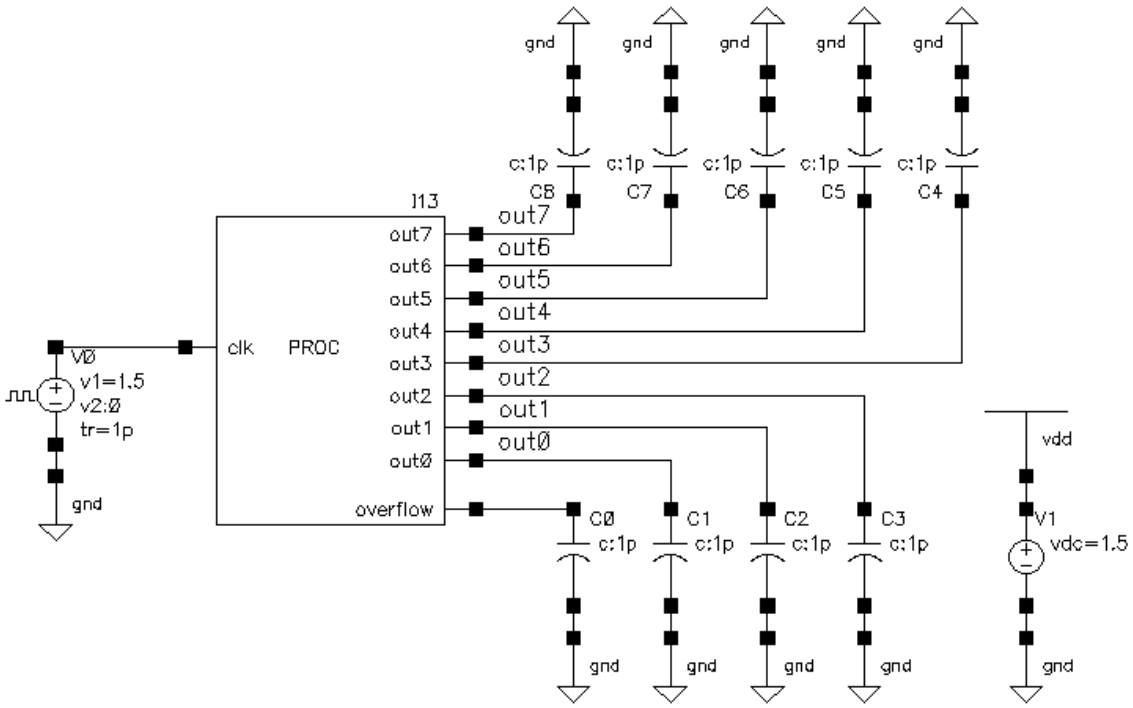


Obrázek 93: Symbol procesoru s čítačem instrukcí

8.1.4 Simulace

Na každém schématu je velmi důležité otestovat jeho funkčnost. V případě běhu programu z 8.1.1 je očekávaný průběh následující:

- 1. perioda hodinového signálu: uložení hodnoty 45 na adr. 1 datové paměti
- 2. perioda hodinového signálu: uložení hodnoty 28 na adr. 2 v datové paměti
- 3. perioda hodinového signálu: uložení součtu z adres 2 a 1 (45+28 = 73) na adr. 3
- 4. perioda hodinového signálu: překopírování hodnoty z adr. 3 na adr. 4
- 5. perioda hodinového signálu: zobrazení hodnoty z adr. 4 na výstupu



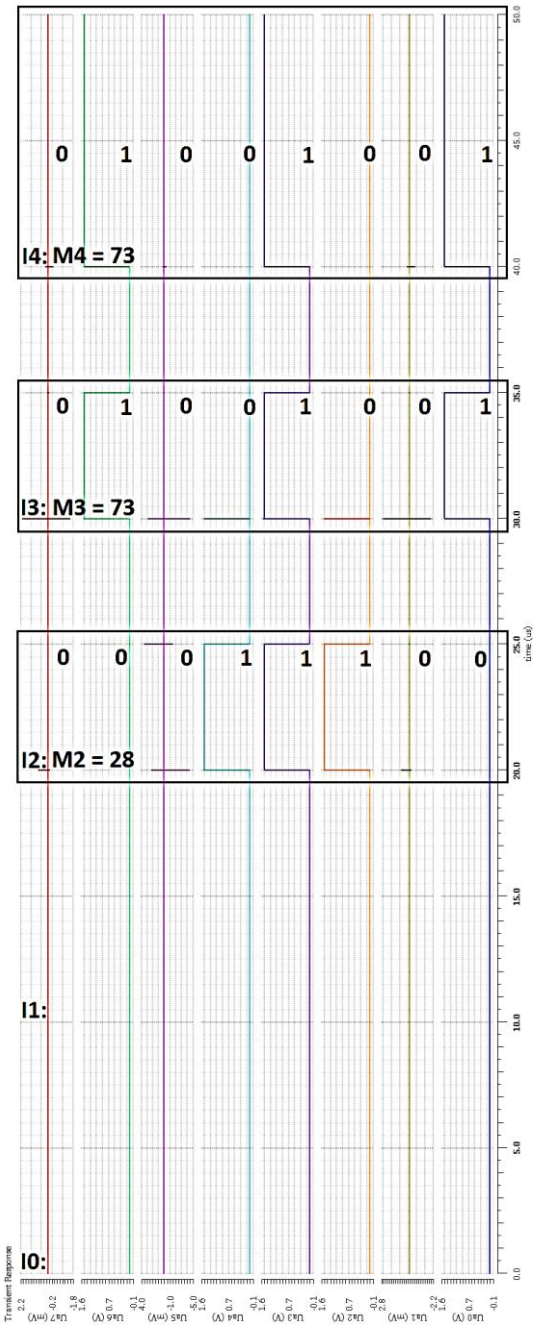
Obrázek 94: Schéma zapojení simulace procesoru s čítačem instrukcí

V simulaci procesoru jsou sledovány tyto signály:

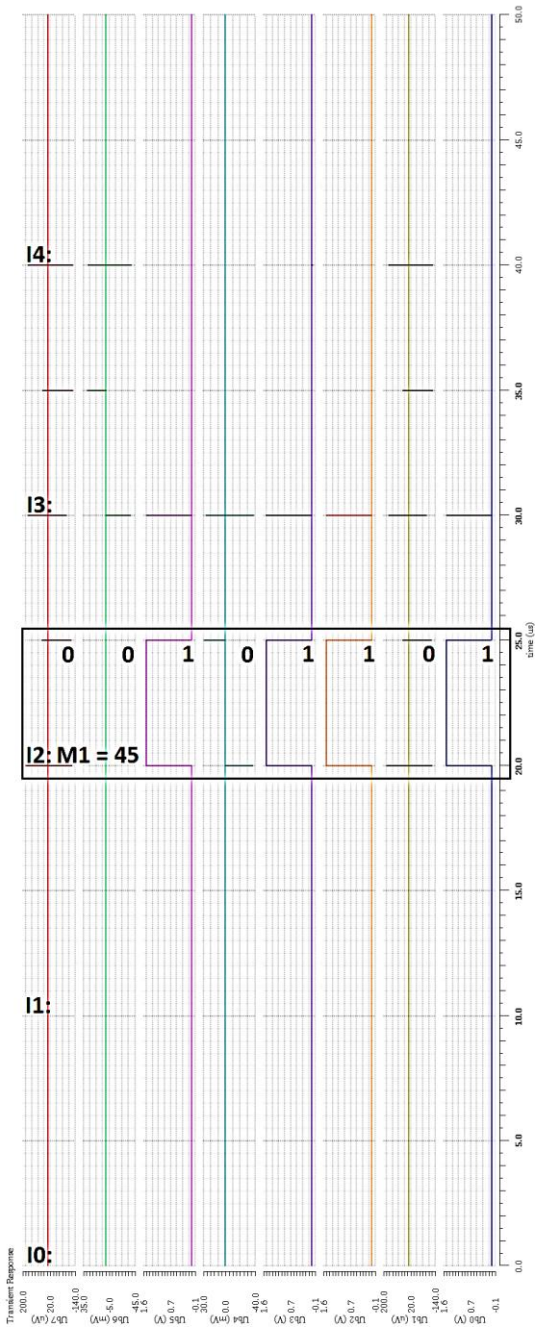
- vstupní signály do datové paměti
- výstupní signály z datové paměti (A,B)
- řídicí signály detekující typ operace
- výstupní signály procesoru

Průběhy těchto signálů jsou zobrazeny níže na obr. 95 až obr. 99.

Na následujících dvou obrázcích jsou zobrazena data přečtená z datové paměti (A,B). Popisky u obrázků jsou ve tvaru „adresa instrukce: adresa paměti = hodnota“. Pokud má právě zpracovávaná instrukce binární kód OOOO XXXX YYYY ZZZZ, kde OOOO značí opcode, tak YYYY je adresa v datové paměti, ze které je čteno na výstup A, a ZZZZ je adresa v datové paměti, ze které je čteno na výstup B. Toto platí pouze pokud se jedná o instrukci, při které se čte z datové paměti, viz kap. 7.

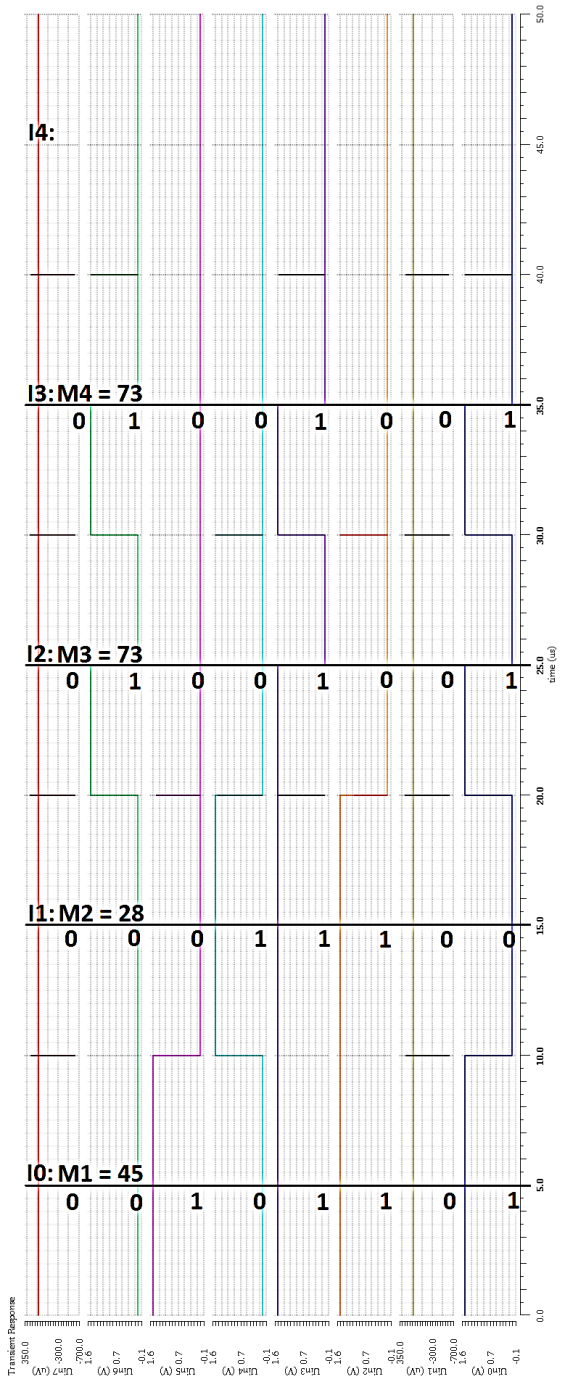


Obrázek 95: Průběh výstupních signálů z datové paměti (A) – procesor s čítačem instrukcí

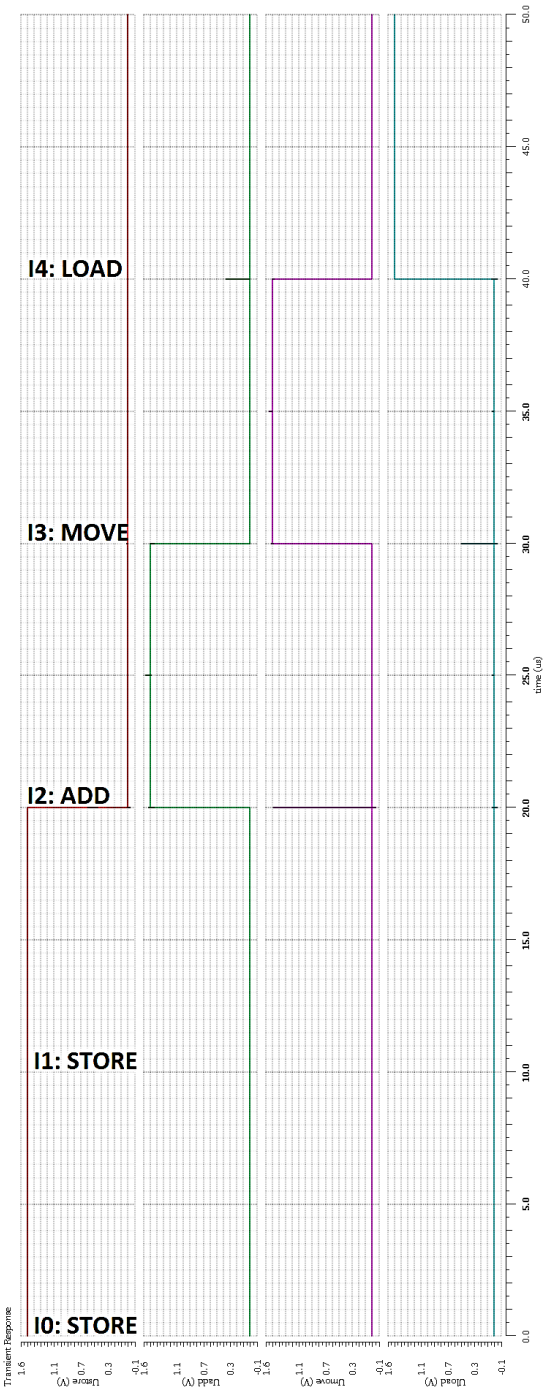


Obrázek 96: Průběh výstupních signálů z datové paměti (B) – procesor s čítačem instrukcí

Na obr. 97 je znázorněn proces zapisování do paměti – buď konkrétního čísla nebo zpětnou vazbou výsledku z ALU. Zápis do paměti je na rozdíl od čtení proces hranový a dochází k němu se spádovou hranou hodinového signálu. Ačkoliv by se mohlo zdát, že k zapsání dat nedojde, tak data jsou přítomna až do spádové hrany hodin, tedy přesně do okamžiku, kdy jsou zapsána do datové paměti.

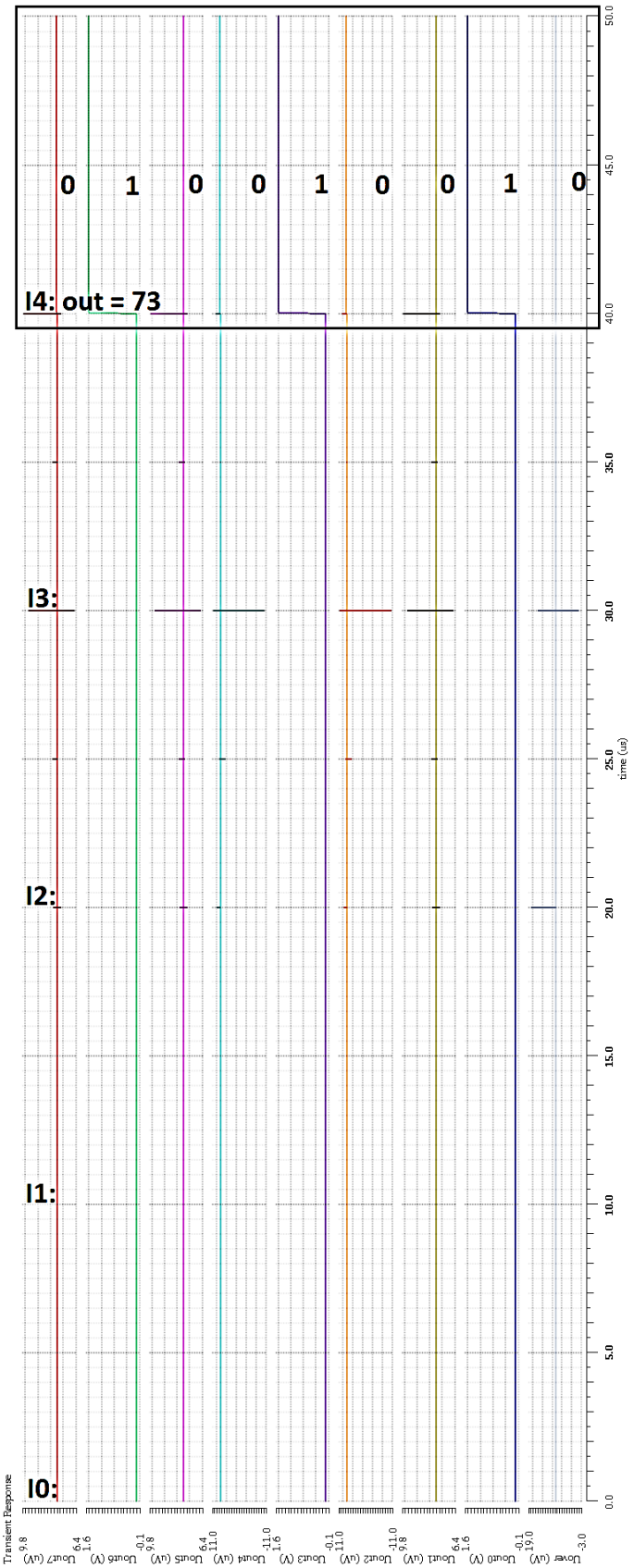


Obrázek 97: Průběh vstupních signálů do datové paměti – procesor s čítačem instrukcí



Obrázek 98: Průběh řídicích signálů detekujících typ operace – procesor s čítačem instrukcí

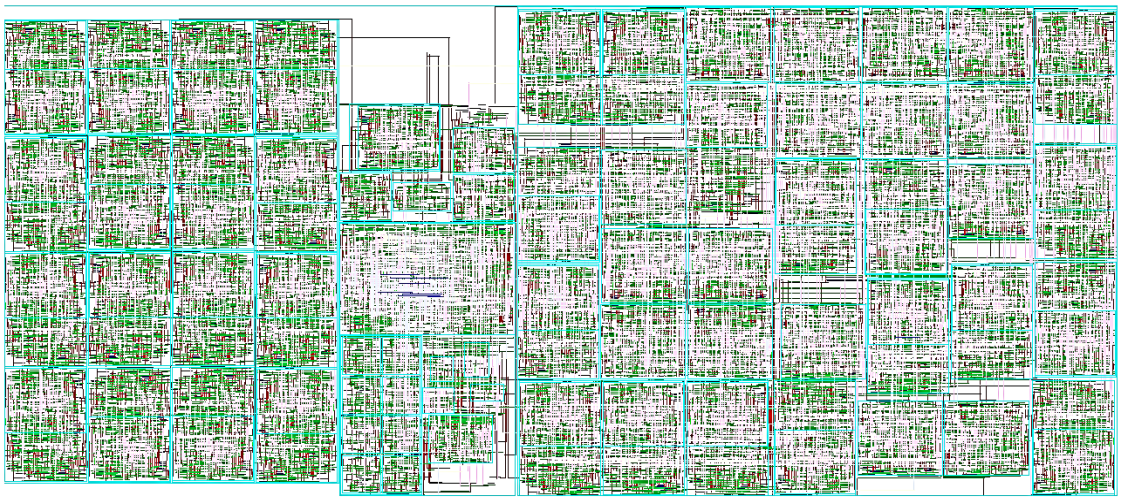
Na obr. 99 je zobrazen stav na výstupních pinech procesoru. Hodnota má úroveň log. 0, pokud není právě zpracovávaná instrukce LOAD. V tom případě je na výstupu zobrazen obsah dané paměťové jednotky.



Obrázek 99: Průběh výstupních signálů procesoru s čítačem instrukcí

8.1.5 Layout

Jako poslední krok byl vygenerován automatický layout procesoru pomocí Layout XL.



Obrázek 100: Layout procesoru s čítačem instrukcí

8.2 SEKVENČNÍ PROCESOR BEZ ČÍTAČE INSTRUKCÍ

Druhý procesor je opět sekvenčního typu, v tomto případě je ale čítač instrukcí vytvořen obvodem, který s každou náběžnou hranou hodin uloží následující adresu přičtením 1 k té současné. V důsledku toho jsou stejně jako v předchozím případě procházeny všechny adresy v instrukční paměti za sebou a práce procesoru trvá celkem 16 instrukčních cyklů.

8.2.1 Program

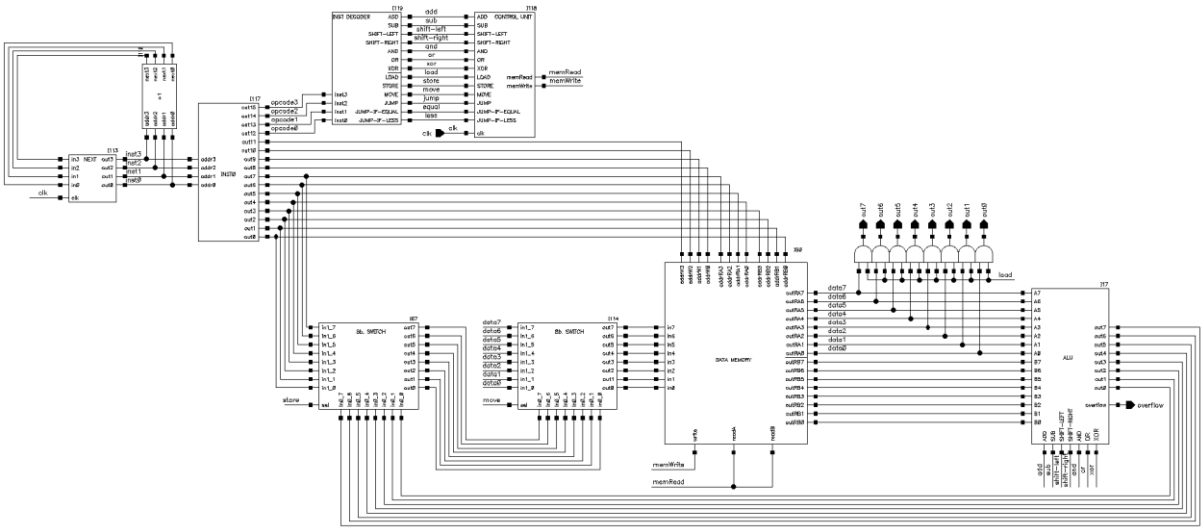
Na ověření funkce tohoto procesoru byl zvolen stejný program jako v kapitole 8.1.1.

8.2.2 Instrukční paměť

Zakódování instrukcí do instrukční paměti je vysvětleno v kapitole 8.1.2.

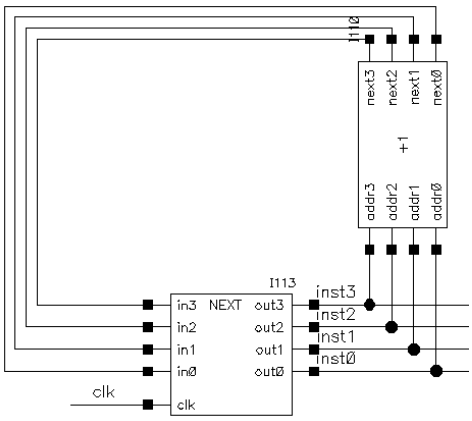
8.2.3 Zapojení procesoru

Zapojení procesoru je v tomto případě velmi podobné jako v kapitole 8.1.3.



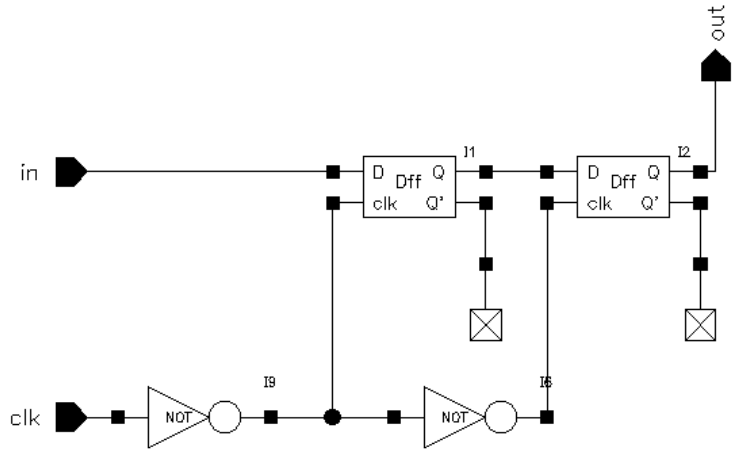
Obrázek 101: Schéma zapojení procesoru bez čítače instrukcí

S každou náběžnou hranou hodinového signálu se k aktuálně zpracovávané adrese instrukce přičte 1 a hodnota se uloží pro příští periodu. Není tedy potřeba čítače instrukcí.

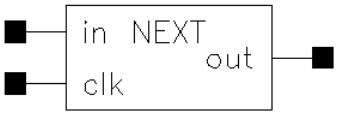


Obrázek 102: Přičítání 1 k instrukční adrese

Ukládání adresy pro příští cyklus je provedeno dvěma sériově zapojenými DFF. Jestliže do prvního D-klopného obvodu zapíšeme hodnotu se sestupnou hranou a do druhého s náběžnou hranou, bude hodnota na výstupu této součástky vždy s další periodou hodinového signálu.

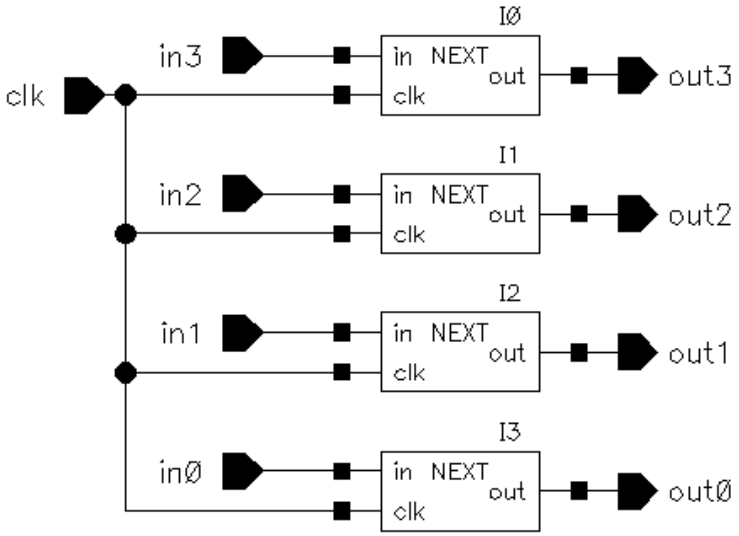


Obrázek 103: Schéma paměti 1 bitu pro příští cyklus

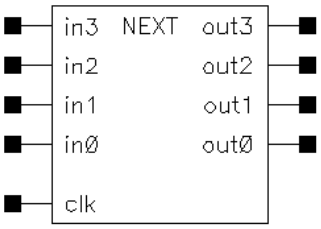


Obrázek 104: Symbol paměti 1 bitu pro příští cyklus

V procesoru je implementovaná paměť pro 4 bity uchovávající adresu instrukce pro příští cyklus. Skládá se ze čtyř pamětí synchronizovaných stejnými hodinami.



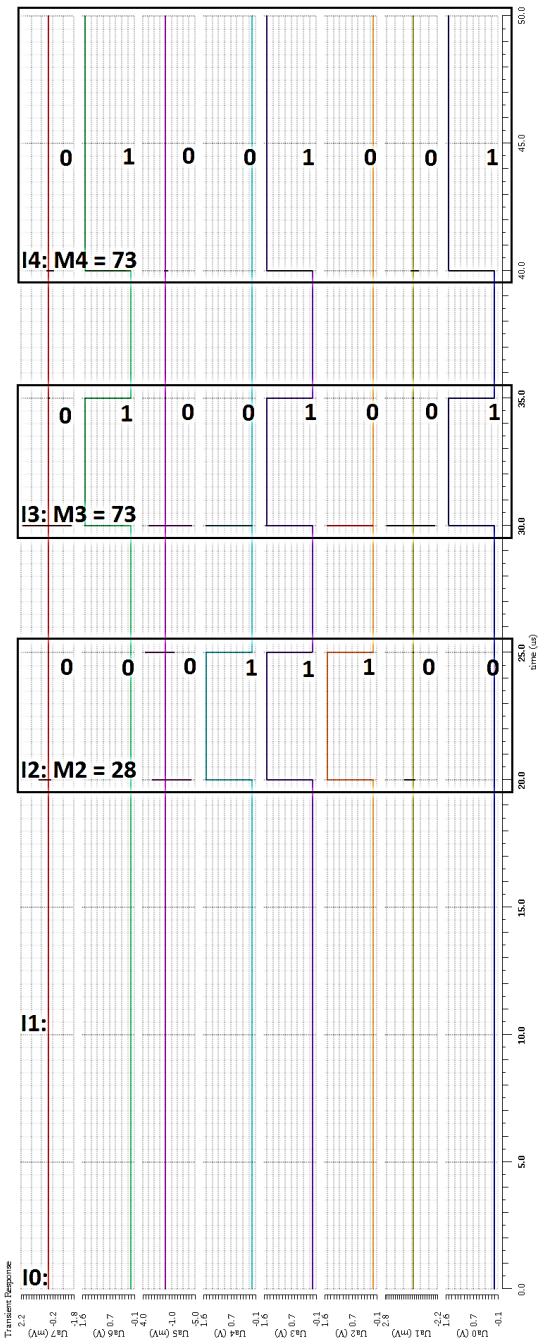
Obrázek 105: Schéma paměti 4 bitů pro příští cyklus



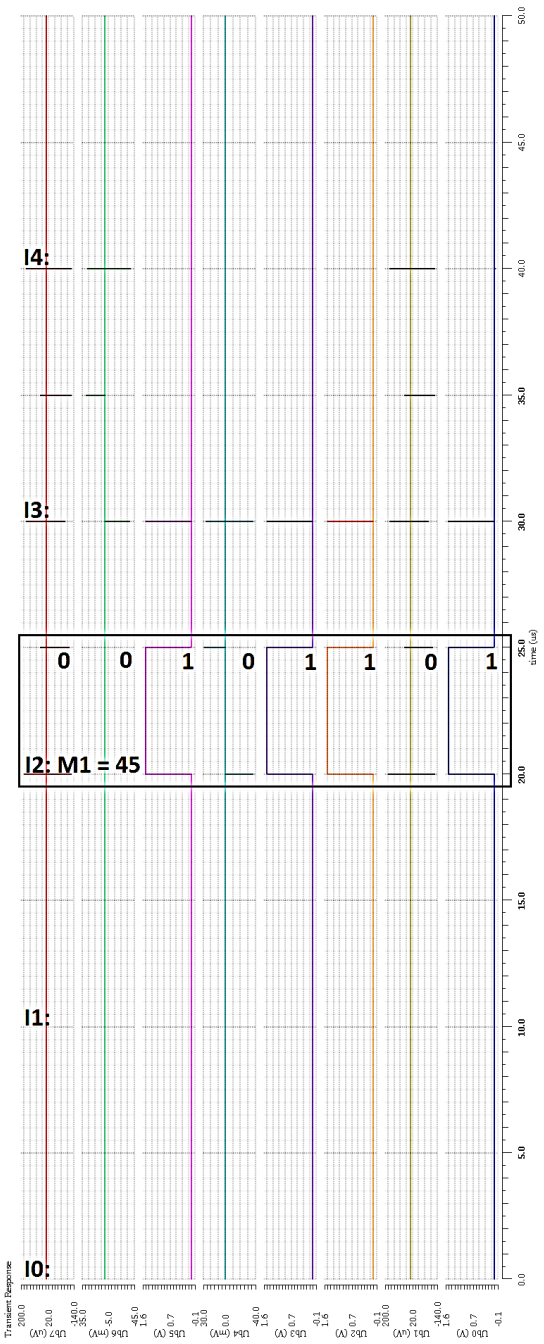
Obrázek 106: Symbol paměti 4 bitů pro příští cyklus

8.2.4 Simulace

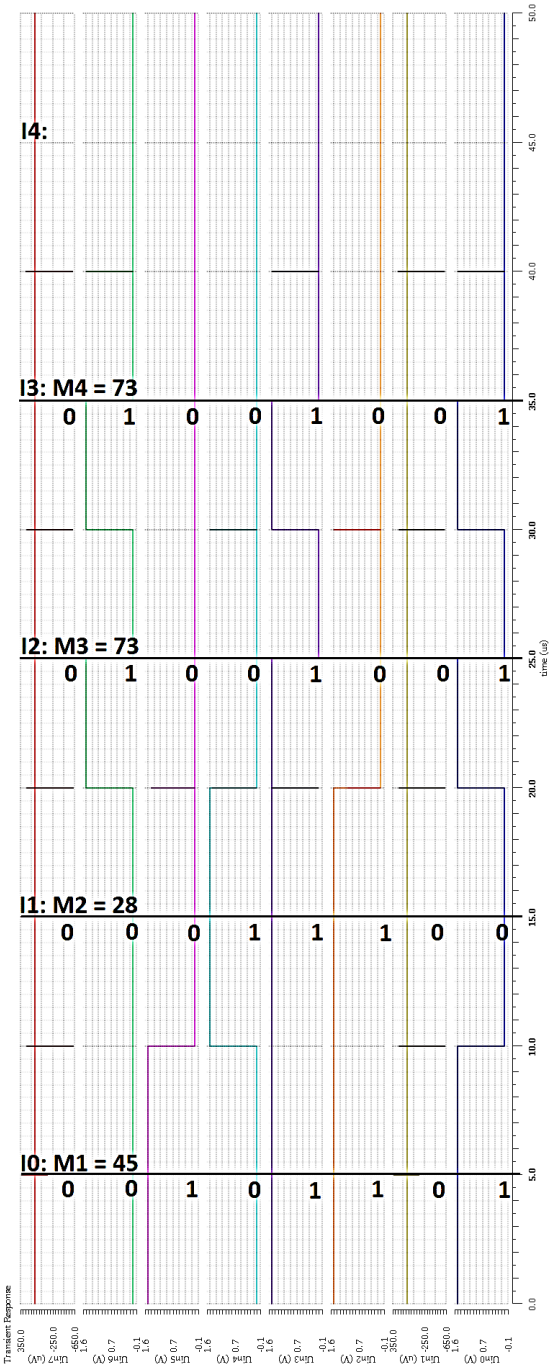
Jelikož je chování tohoto procesoru naprosto stejné jako v kapitole 8.1, odpovídají si i jednotlivé průběhy simulace.



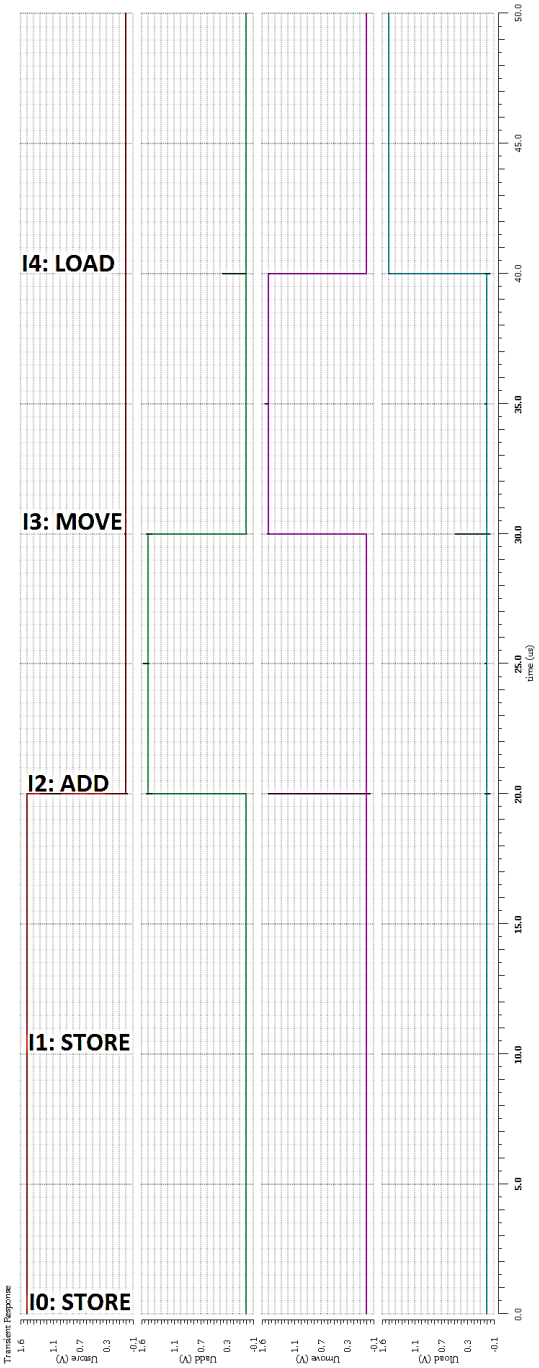
Obrázek 107: Průběh výstupních signálů z datové paměti (A) – procesor bez čítače instrukcí



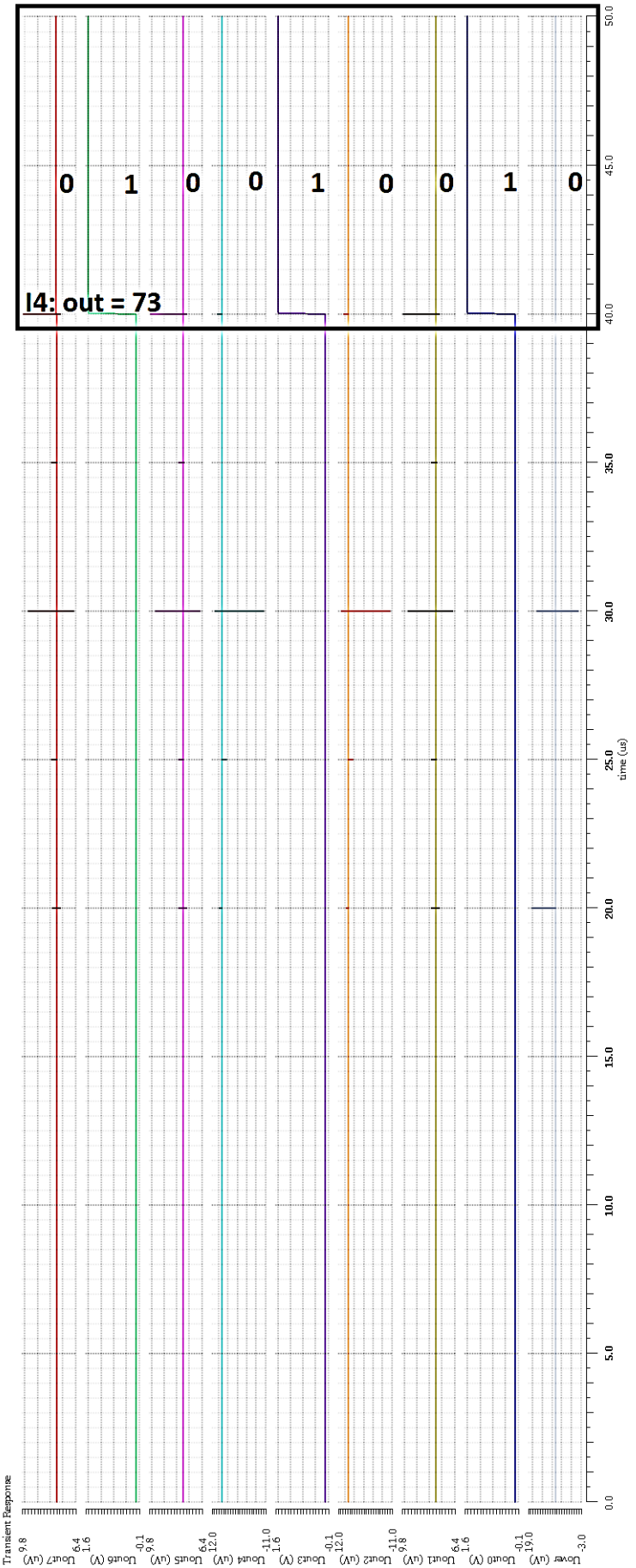
Obrázek 108: Průběh výstupních signálů z datové paměti (B) – procesor bez čítače instrukcí



Obrázek 109: Průběh vstupních signálů do datové paměti – procesor bez čítače instrukcí



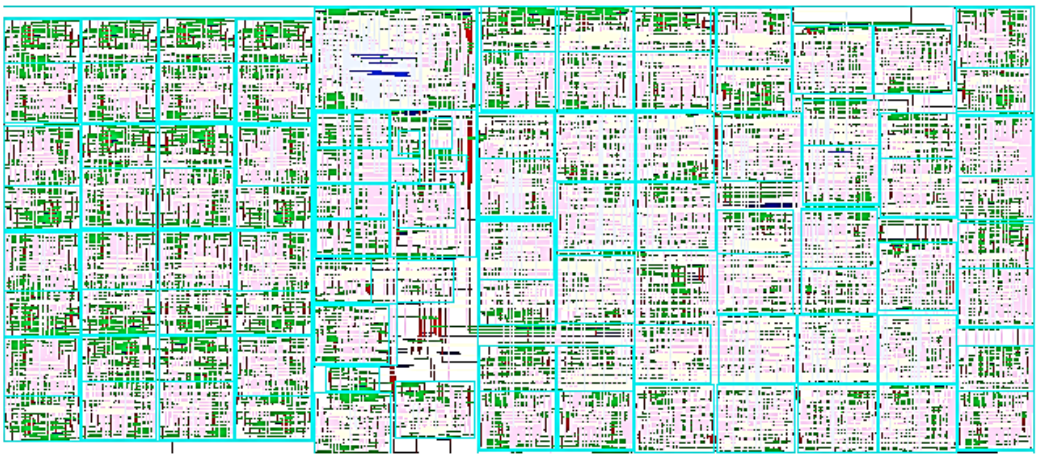
Obrázek 110: Průběh řídicích signálů detekujících typ operace – procesor bez čítače instrukcí



Obrázek 111: Průběh výstupních signálů procesoru bez čítače instrukcí

8.2.5 Layout

Stejně jako v předchozím případě byl nakonec vytvořen automatický layout v Layout XL.



Obrázek 112: Layout procesoru bez čítače instrukcí

8.3 PROGRAM SE SKOKY

Třetí procesor již využívá implementovanou instrukční sadu v plném rozsahu, tj. včetně skokových instrukcí.

8.3.1 Program

Tento typ procesoru je vyzkoušen na jednom ze základních algoritmů pro studenta informatiky. Standardní Euklidův algoritmus je dle [12] definován jako algoritmus pro nalezení největšího společného dělitele dvou přirozených čísel *a*, *b* a má následující kód:

```
procedure gcd (a, b: integer, a > b > 0)
repeat
        r := a mod b;
        a := b;
        b := r;
until b = 0;
output: a;
```

```
procedure gcd (a, b: integer, a > b > 0)
repeat
        r := a
        repeat
                r := r – b;
        until r >= b
        a := b;
        b := r;
until b = 0;
output: a;
```

Instrukční sada implementovaná v tomto procesoru však nepodporuje operaci modulo, proto je nutné tento algoritmus upravit. Jelikož operace modulo znamená zbytek po dělení dvou čísel, stačí číslo *b* odečítat od čísla *a*, dokud a>=b. Takto upravený program bude mít následující kód:

Tento program je v procesoru implementován pro a=15, b=10 a do instrukcí přepsán následovně:

```
I0      STORE           M0, 0
I1      STORE           M1, 15
I2      STORE           M2, 10
I3      JUMP-IF-EQUAL   I11, M2, M0
I4      MOVE            M3, M1
I5      JUMP-IF-LESS    I8, M3, M2
I6      SUB             M3, M3, M2
I7      JUMP            I5
I8      MOVE            M1, M2
I9      MOVE            M2, M3
I10     JUMP            I3
I11     LOAD            M1
```

8.3.2 Instrukční paměť

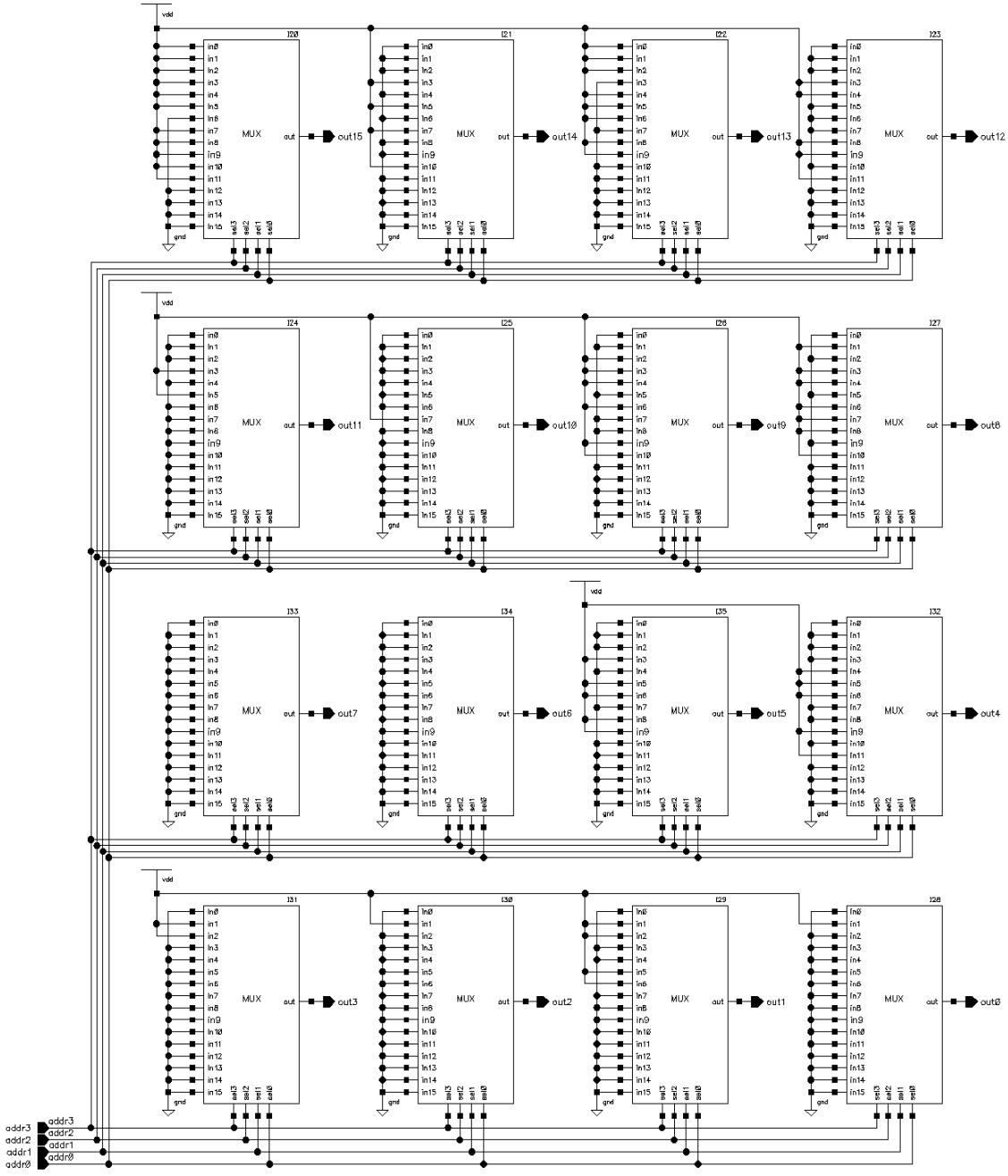
Stejně jako v předchozích případech byly instrukce přepsány do binárního kódu.

Tabulka 10: Zakódované instrukce pro Euklidův algoritmus

addr[3:0]	zakódovaná instrukce v binárním kódu
0000	1010 0000 0000 0000
0001	1010 0001 0000 1111
0010	1010 0010 0000 1010
0011	1101 1011 0010 0000

0100	1011 0011 0001 0000
0101	1110 1000 0011 0010
0110	0010 0011 0011 0010
0111	1100 0101 0000 0000
1000	1011 0001 0010 0000
1001	1011 0010 0011 0000
1010	1100 0011 0000 0000
1011	1001 0000 0001 0000
1100 až 1111	0000 0000 0000 0000

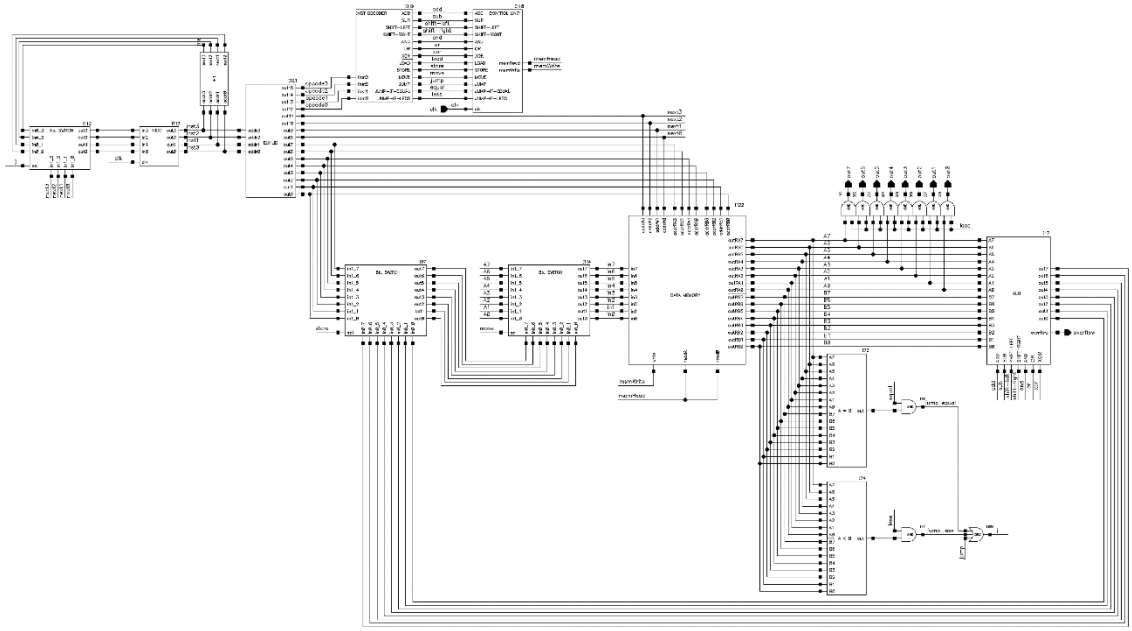
Konkrétní zapojení instrukční paměti tedy vypadá následovně.



Obrázek 113: Zapojení instrukční paměti Euklidova algoritmu

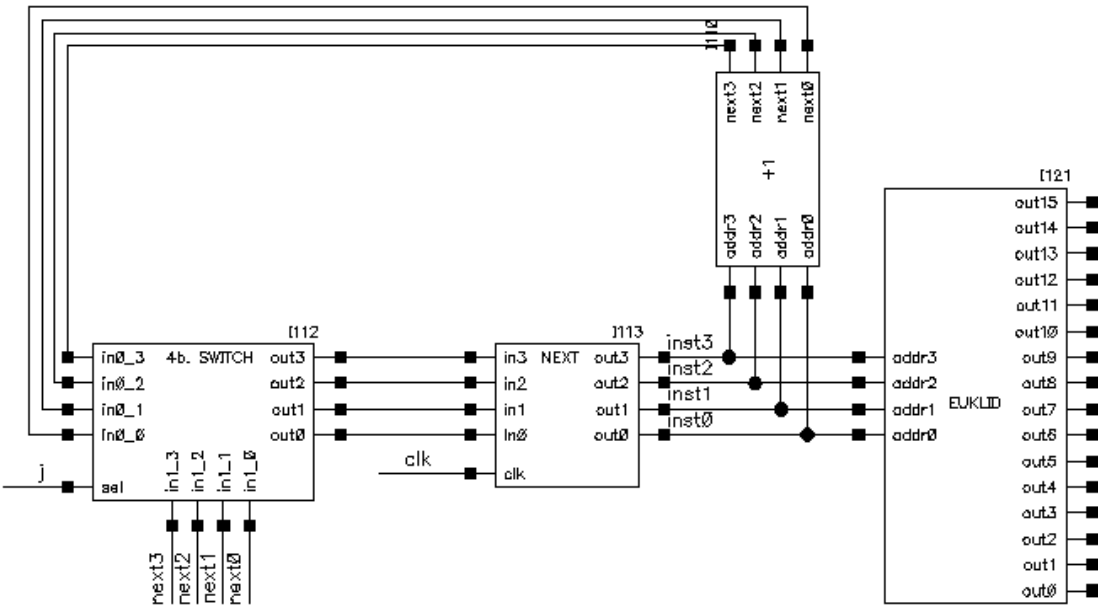
8.3.3 Zapojení procesoru

Schéma zapojení procesoru implementujícího Euklidův algoritmus (obr. 114) je v mnohém podobné schématu zapojení procesoru bez čítače instrukcí (obr. 101). V této kapitole bude kladen důraz pouze na odlišnosti v obou zapojeních, tj. detekce skokové instrukce a vyhodnocení aktuálně zpracovávané instrukce.



Obrázek 114: Schéma zapojení procesoru – Euklidův algoritmus

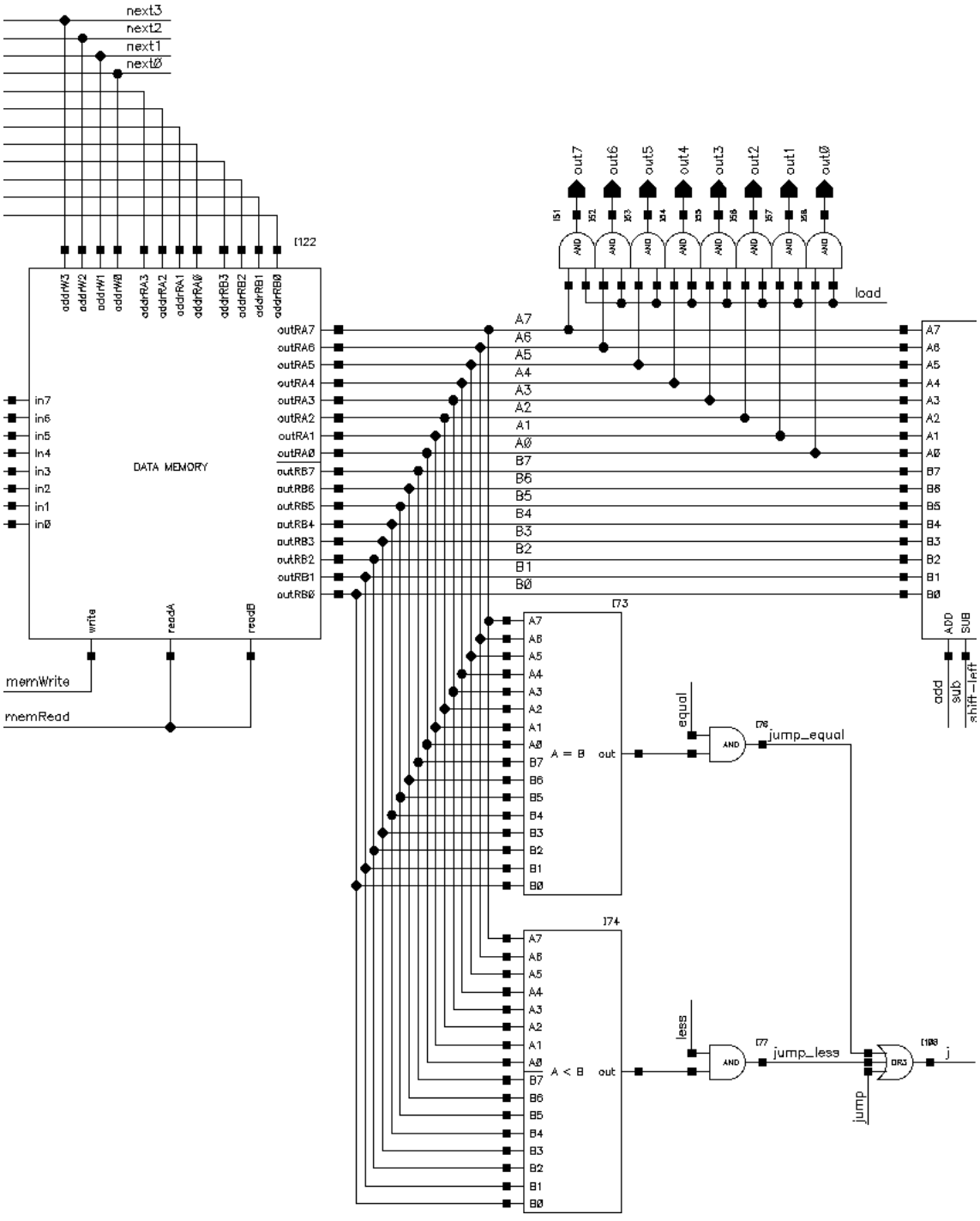
Vyhodnocení aktuálně zpracovávané instrukce je znázorněna na obr. 115. Stejně jako u minulého procesoru je k aktuální adrese přičtena 1, ovšem tato hodnota není ihned zapsána do paměti pro příští hodinový cyklus. Je zapsána nejprve do přepínače, který přepíná mezi touto hodnotou a hodnotou příští instrukce v případě skokové instrukce.



Obrázek 115: Vyhodnocení hodnoty příští instrukce – Euklidův algoritmus

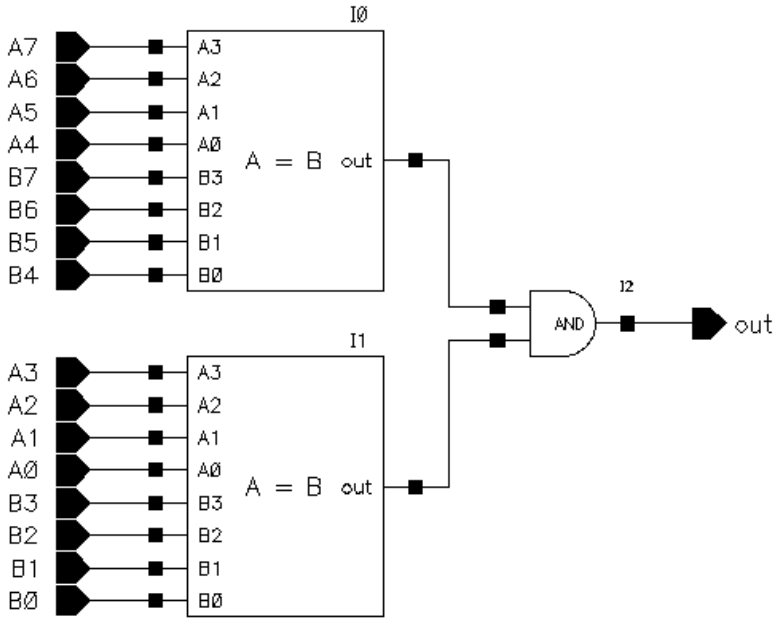
Řídicí signál přepínače z obr. 115 je ovlivněn faktem, zda byla detekována skoková instrukce. Pokud ano, tj. právě zpracovávaná instrukce je typu *JUMP*, *JUMP-IF-EQUAL* nebo *JUMP-IF-LESS*,

a byly splněny její podmínky (žádné v případě *JUMP*), signál *j* bude mít hodnotu log. 1 a v příštím hodinovém cyklu bude vyhodnocována instrukce s adresou *next[3:0]*.

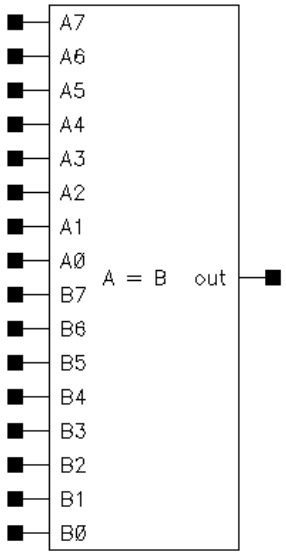


Obrázek 116: Schéma detekce skoku – Euklidův algoritmus

Rovnost dvou čísel (A = B) je vyhodnocována komparátorem zobrazeným na obr. 117. Ze schématu zapojení vyplývá, že rovnost čísel je porovnávána zvlášť pro horní a dolní 4 bity a jestliže se rovnají horní i dolní 4 bity, musejí se rovnat obě 8-bitová čísla.

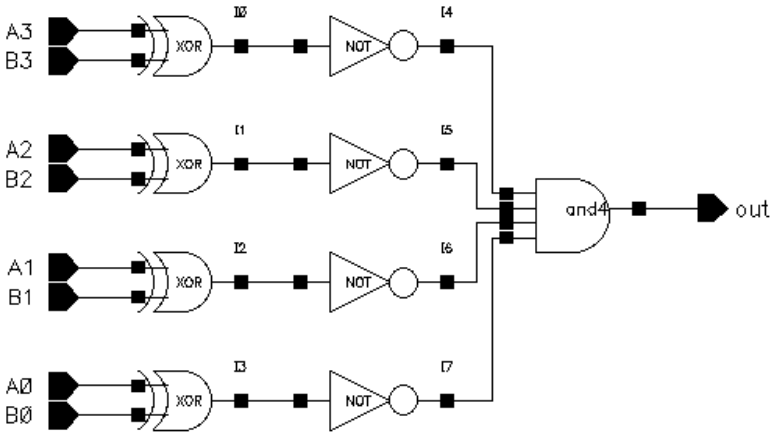


Obrázek 117: Schéma zapojení A=B (8 bitů)

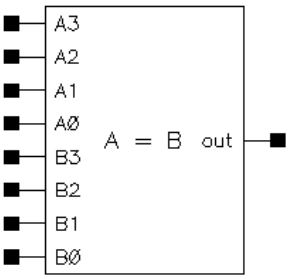


Obrázek 118: Symbol zapojení A=B (8 bitů)

Rovnost 4-bitového čísla je ověřena pomocí hradla XOR. Z tabulky 7 vyplývá, že mají-li dva bity stejnou logickou hodnotu, jejich zapojení do hradla XOR bude mít vždy úroveň log. 0, tj. jeho negace bude mít úroveň log. 1. Proto pokud dva 4-bitové operandy budou mít na stejných pozicích bity stejné logické úrovně, výstupy z hradel XOR budou mít úroveň log. 0, jejich negace log. 1 a na výstupu bude také hodnota log. 1.

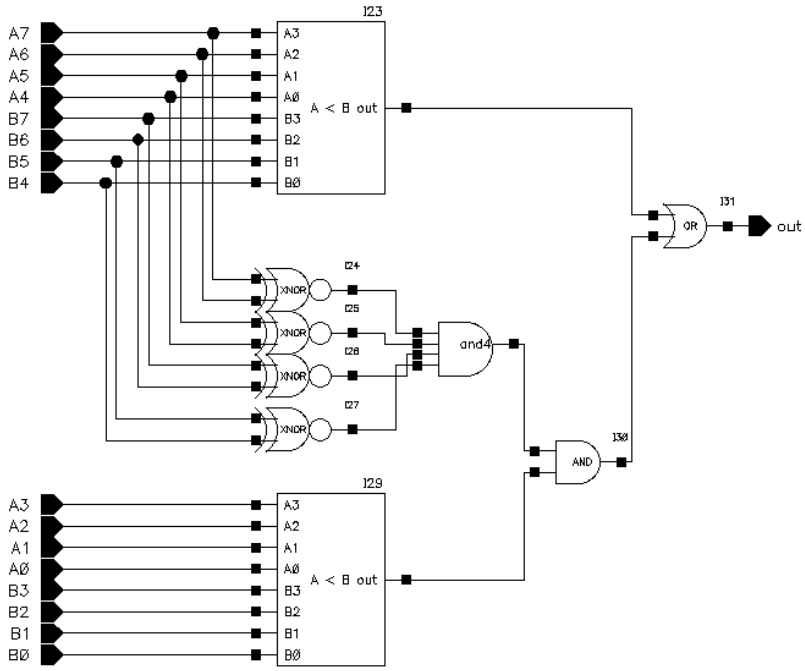


Obrázek 119: Schéma zapojení A=B (4 bity)

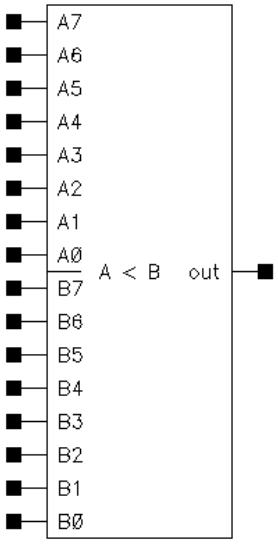


Obrázek 120: Symbol zapojení A=B (4 bity)

Porovnání dvou 8-bitových čísel A < B probíhá podobně jako v případě vyhodnocení rovnosti dvou čísel zvlášť pro horní a dolní 4 bity. Výstup bude mít hodnotu log. 1, pokud byla v horních 4 bitech detekován fakt, že A<B, nebo pokud se horní 4 bity rovnají a detekce A<B byla zaznamenána mezi dolními 4 bity. Rovnost horních 4 bitů je pomocí hradel XNOR [1], které chováním odpovídají hradlu XOR a na výstup připojenému invertoru.

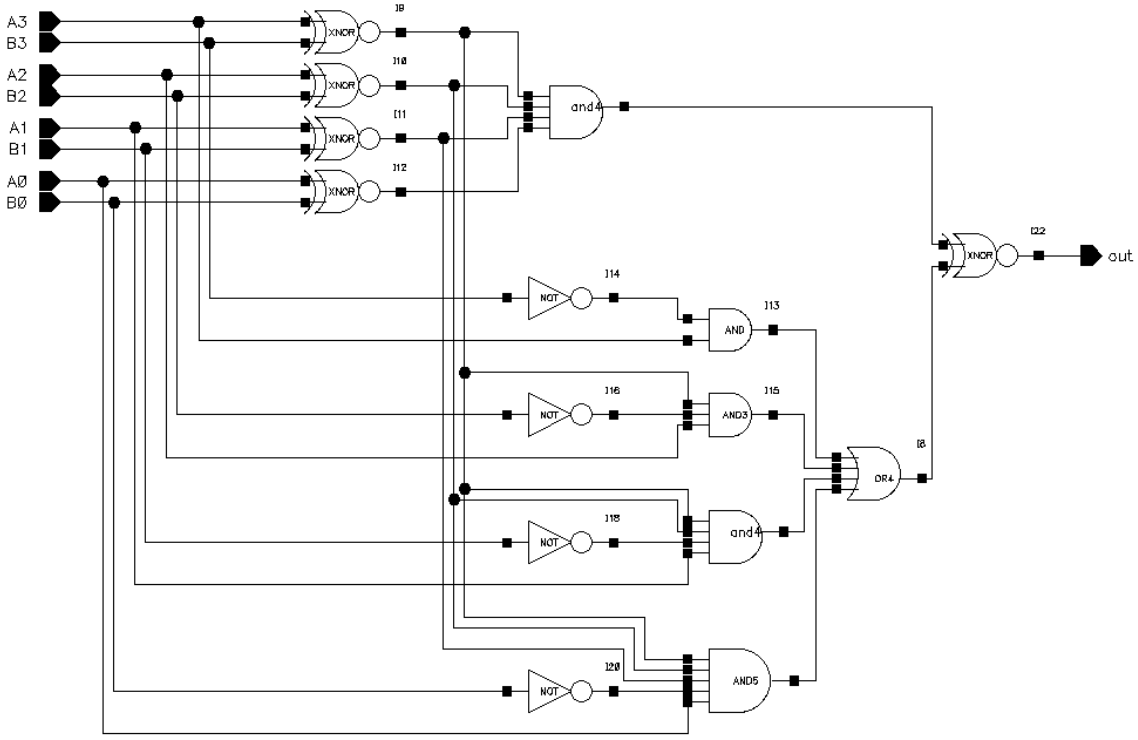


Obrázek 121: Schéma zapojení A<B (8 bitů)

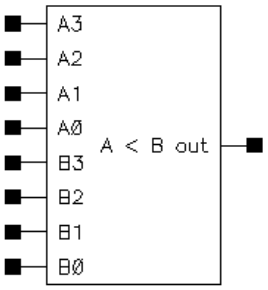


Obrázek 122: Symbol zapojení A<B (8 bitů)

Porovnání 4-bitových operandů A<B je zobrazeno na obr.123 a je převzato z [13]. Na rozdíl od zmíněného zdroje je ale na výstupu zapojeno hradlo XOR místo XNOR. Dvě čísla nemohou být zároveň větší a rovno, takže do hradla nikdy nepovedou dva signály s úrovní log. 1. Pokud právě jeden z nich má hodnotu log. 1, na výstupu bude vždy úroveň log. 0 a 4-bitové operandy jsou si buď rovny nebo platí A>B. Pokud oba signály vedoucí do hradla XNOR mají úroveň log. 0, na výstupu bude hodnota log. 1 a platí A<B.



Obrázek 123: Schéma zapojení A<B (4 bity)



Obrázek 124: Symbol zapojení A<B (4 bity)

Zapojení A<B počítá detekuje správné výsledky pouze pro 8-bitová čísla bez znaménka. Jelikož Euklidův algoritmus, pro který je tento procesor navržen, pracuje pouze s kladnými čísly, nemusí být na tento fakt kladen důraz. Pokud by byl záměr postavit procesor se skokovou instrukcí *JUMP-IF-LESS* pro dva obecné operandy v dvojkovém doplňku, musela by být tato součástka přestavěna s respektem na znaménko operandů.

Pokud tedy byla detekována skoková instrukce a její podmínky byly splněny (viz obr. 116), v paměti pro příští instrukci bude uchována adresa skoku, která je součástí instrukce (viz tabulka 1).

8.3.4 Simulace

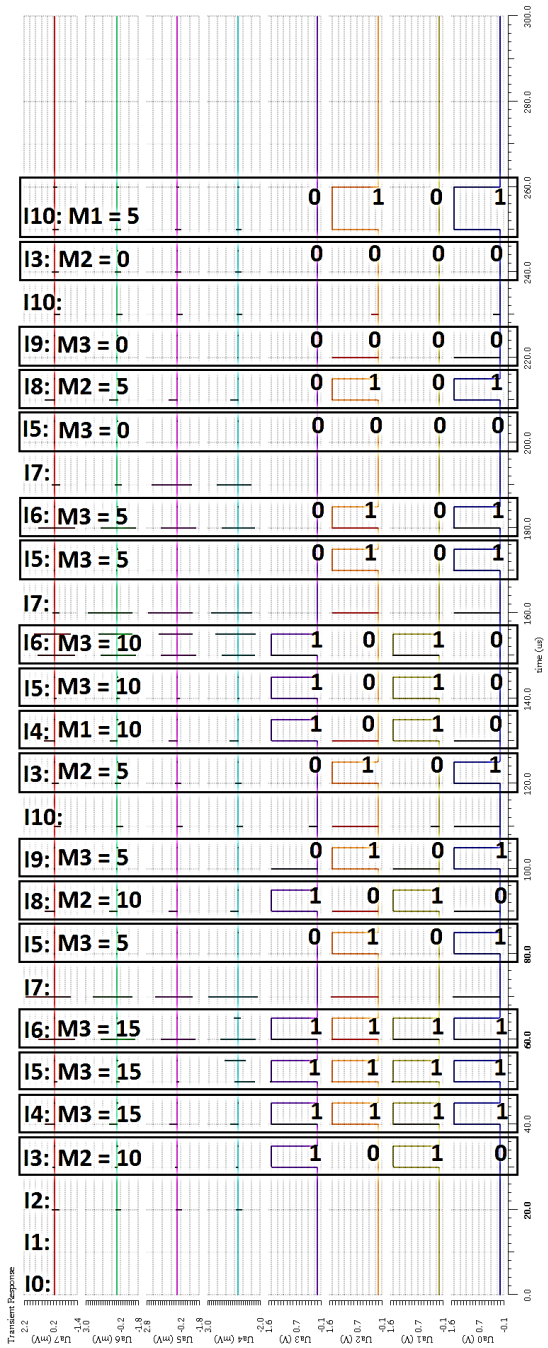
Práce procesoru byla nasimulována na průběhu Euklidova algoritmu s konkrétními čísly a=15, b=10. Průběh instrukcí v jednotlivých hodinových cyklech je zobrazen níže v tabulce 11. Červeně znázorněny jsou konkrétní zápisy do datové paměti.

Tabulka 11: Průběh simulace Euklidova algoritmu

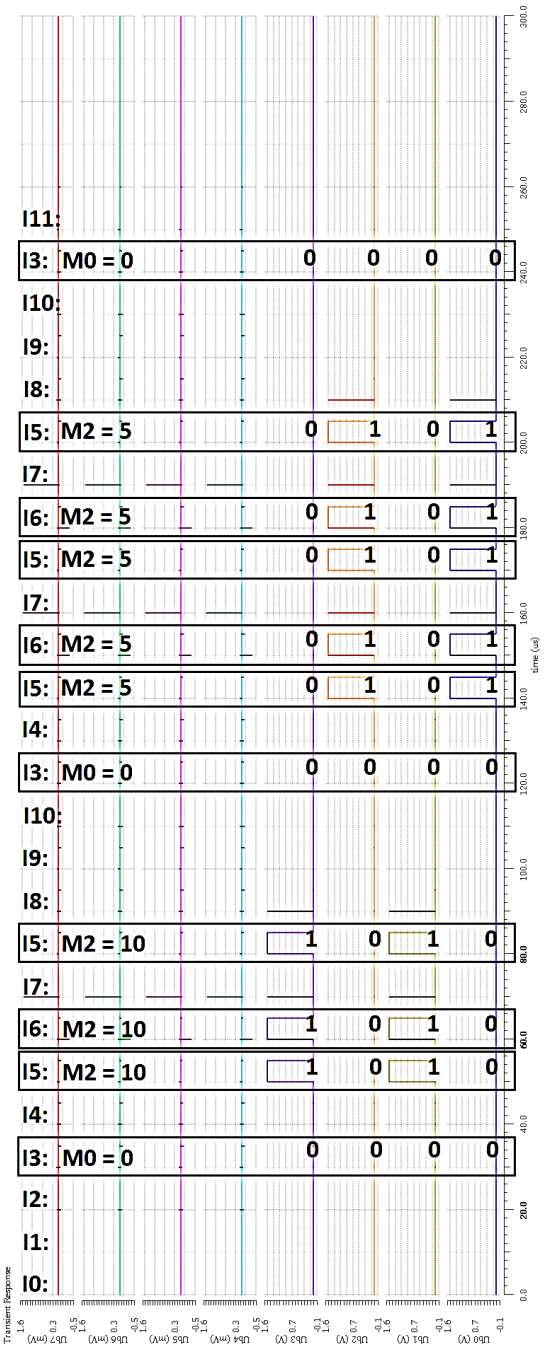
pořadí	adresa instrukce	typ instrukce	vysvětlivky
1	0	STORE	M0 = 0
2	1	STORE	M1 = 15
3	2	STORE	M2 = 10
4	3	JUMP-IF-EQUAL	M2 ≠ M0 (10 ≠ 0)
5	4	MOVE	M3 = 15
6	5	JUMP-IF-LESS	M3 > M2 (15 > 10)
7	6	SUB	M3 = 5
8	7	JUMP	
9	5	JUMP-IF-LESS	M3 < M2 (5 < 10)
10	8	MOVE	M1 = 10
11	9	MOVE	M2 = 5
12	10	JUMP	
13	3	JUMP-IF-EQUAL	M2 ≠ M0 (5 ≠ 0)
14	4	MOVE	M3 = 10
15	5	JUMP-IF-LESS	M3 > M2 (10 > 5)
16	6	SUB	M3 = 5
17	7	JUMP	
18	5	JUMP-IF-LESS	M3 == M2 (5 = 5)
19	6	SUB	M3 = 0
20	7	JUMP	
21	5	JUMP-IF-LESS	M3 < M2 (0 < 5)
22	8	MOVE	M1 = 5

23	9	MOVE	M2 = 0
24	10	JUMP	
25	3	JUMP-IF-EQUAL	M2 == M0 (0 == 0)
26	11	LOAD	
27	12	---	---
28	13	---	---
29	14	---	---
30	15	---	---

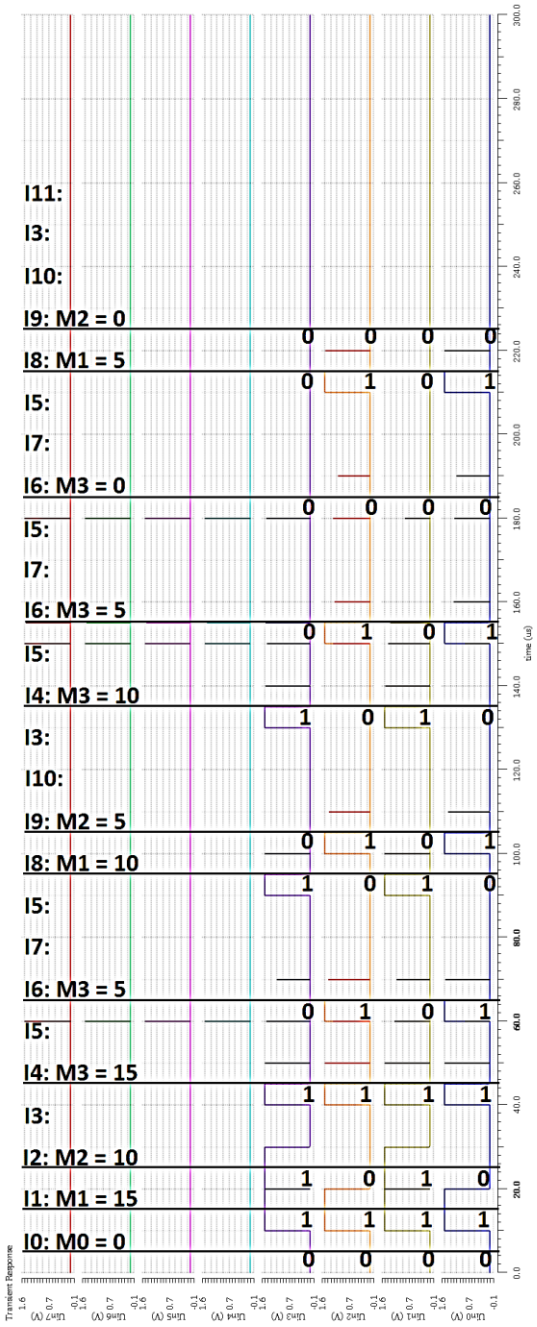
Princip průběhu simulací je obdobný jako v předchozích dvou procesorech a je zobrazen níže.



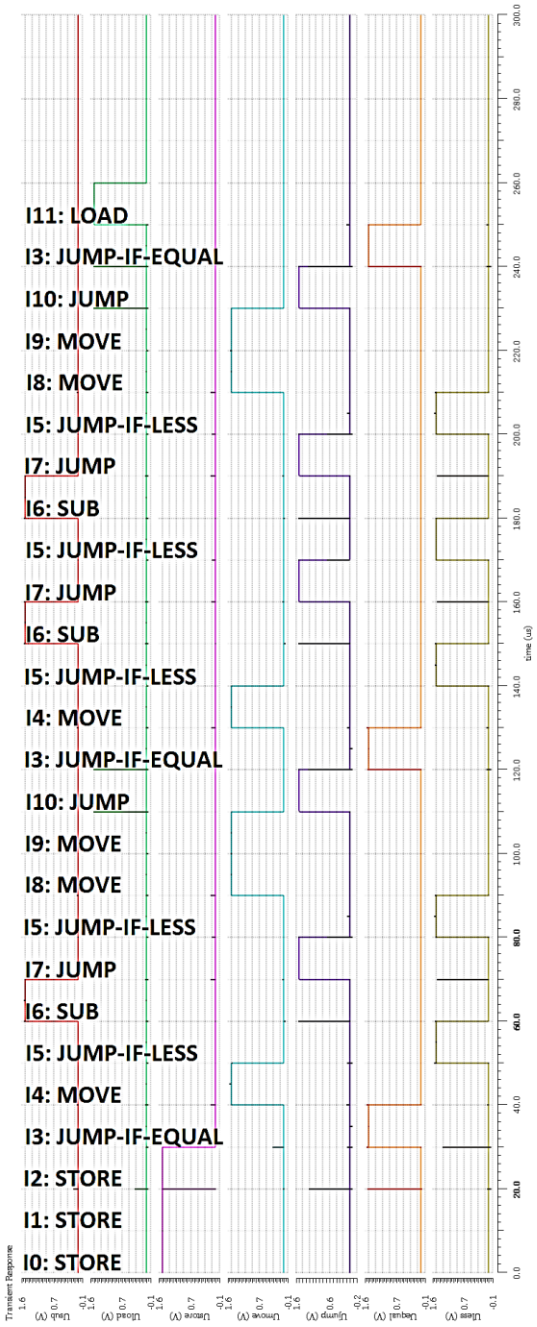
Obrázek 125: Průběh výstupních signálů z datové paměti (A) – Euklidův algoritmus



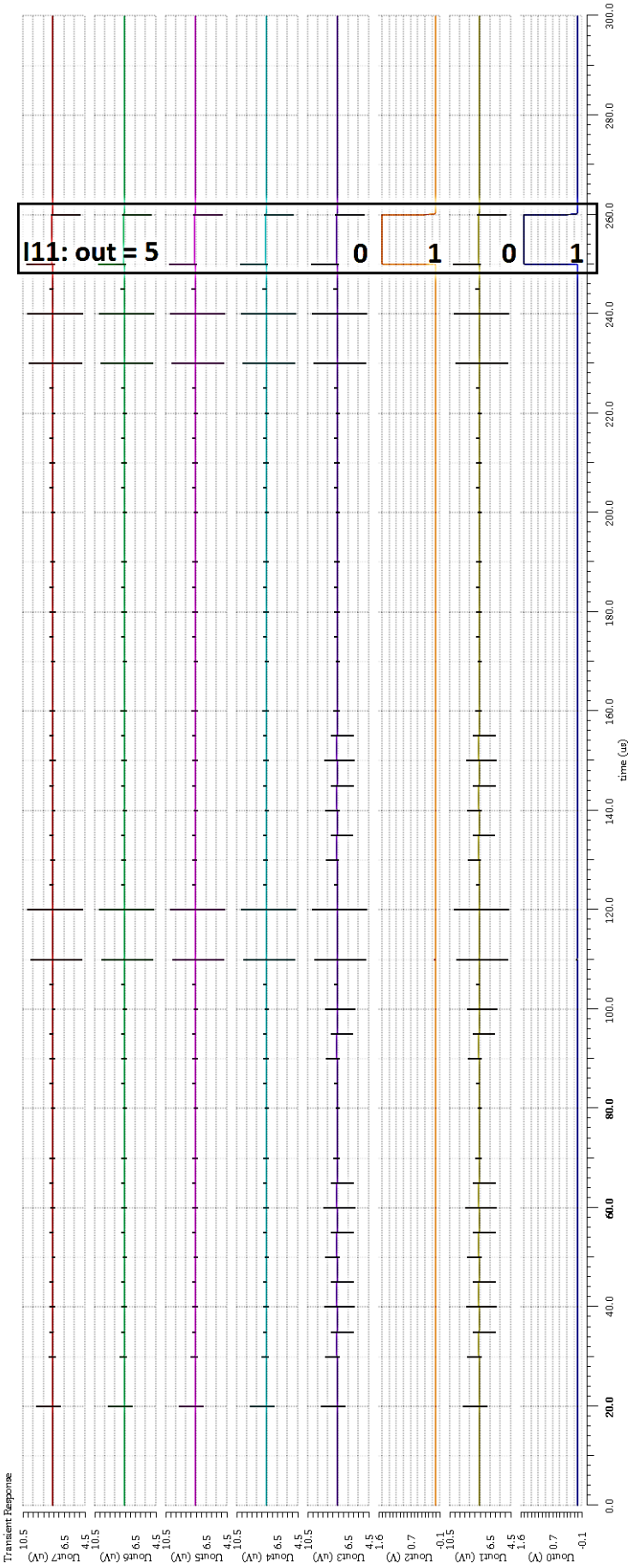
Obrázek 126: Průběh výstupních signálů z datové paměti (B) – Euklidův algoritmus



Obrázek 127: Průběh vstupních signálů do datové paměti – Euklidův algoritmus



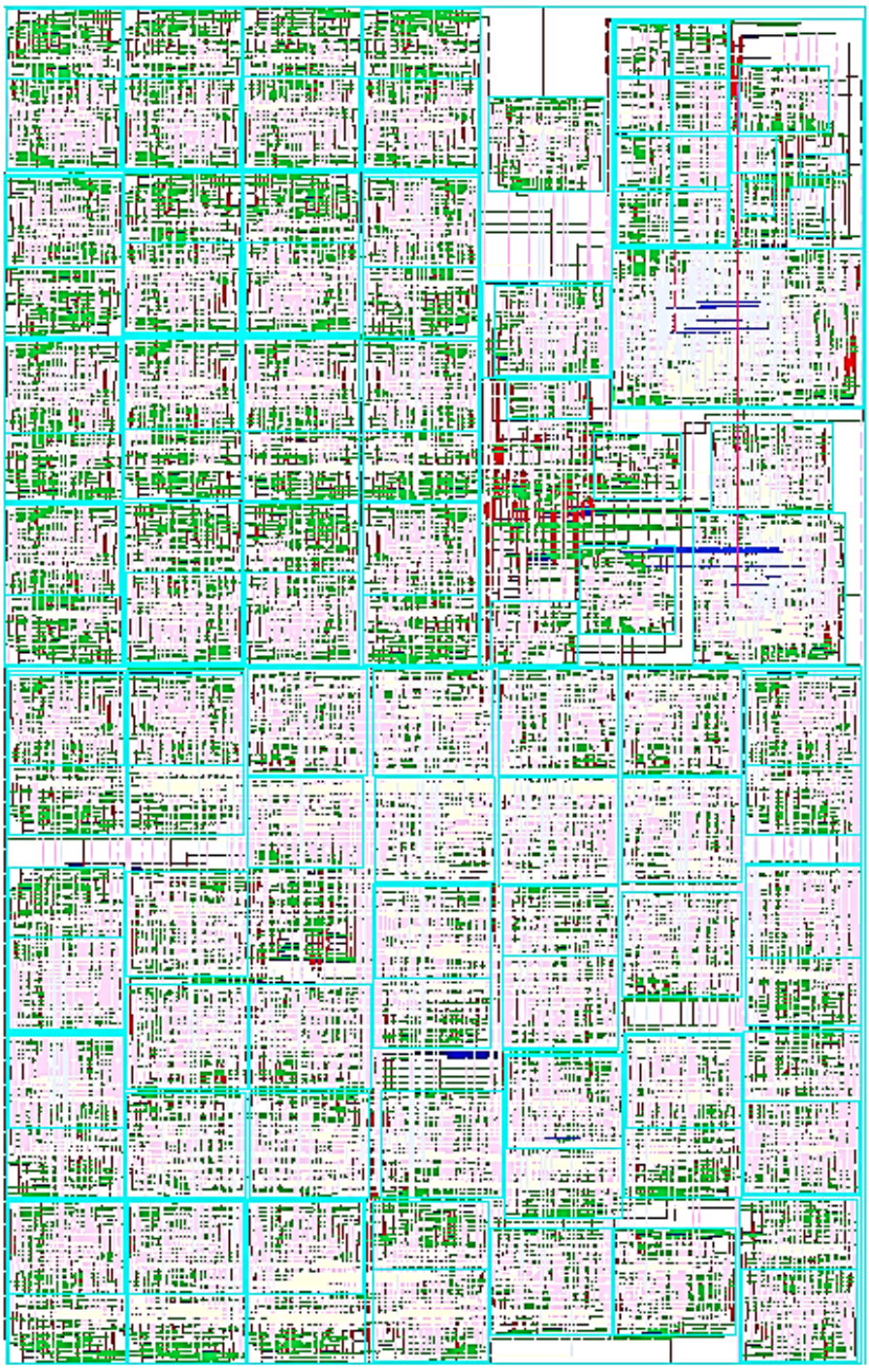
Obrázek 128: Průběh řídicích signálů detekujících typ operace – Euklidův algoritmus



Obrázek 129: Průběh výstupních signálů procesoru – Euklidův algoritmus

8.3.5 Layout

Stejně jako v předchozích případech byl vytvořen automatický layout v Layout XL.



Obrázek 130: Layout procesoru – Euklidův algoritmus

9 ZÁVĚR

Cílem diplomové práce bylo navázat na svoji bakalářskou práci [1] a poodkrýt svět elektroniky pro studenty informatiky trochu z jiného úhlu. Finální zapojení procesoru a ověření funkce na Euklidově algoritmu bylo zvoleno z toho důvodu, že tento algoritmus je přednášen již od 1. semestru bakalářské etapy studia a s výpočtem největšího společného dělitele se student setkává po celou dobu svého studia informatiky.

Diskrétní forma realizace bez použití jakéhokoliv jazyka (např. VHDL, Verilog) byla i zde zvolena záměrně, aby bylo možné ukázat průběh jednotlivých signálů a jejich načasování. Při napsání programu v nějakém jazyce by nebylo možné zobrazit jednotlivé nesrovnalosti v průběhu signálů. Zároveň se domnívám, že při tomto postupu je lépe vidět důležitost hierarchického návrhu a důležitost odladění jednotlivých podbloků.

Záměrem této diplomové práce byla možnost využití jako doplňkového materiálu při výuce elektronických předmětů pro studenty informatiky, proto jsem kladla důraz na srozumitelnost textu pro studenty a zároveň použití takových elektronických součástek, s nimiž má student možnost se setkat v rámci svého studia.

Další výhodou tohoto přístupu je zdůraznění složitosti celého návrhu. Pomocí nástrojů v Layout XL je možné dohledat informace o zapojení, mj. počet tranzistorů NMOS a PMOS.

Tabulka 12: Počet tranzistorů v diplomové práci

	počet tranzistorů	
	NMOS	PMOS
procesor s čítačem instrukcí	10 566	10 566
procesor bez čítače instrukcí	10 794	10 794
procesor – Euklidův algoritmus	11 250	11 250
celkem	32 610	32 610

Celkem bylo tedy v této práci 65 200 tranzistorů. Domnívám se, že tato čísla jsou názornou ukázkou toho, že procesor se zabudovaným zdánlivě jednoduchým algoritmem, jakým je Euklidův algoritmus pro výpočet největšího společného dělitele není jednoduché postavit, alespoň co se týče počtu součástek.

10 POUŽITÁ LITERATURA

[1] J. Ščuková, Tutoriál digitálního návrhu v Cadence pro studenty informatiky, Praha, 2017.

[2] C. Terman, „Computer Architecture MIT 6.004 - L03: CMOS Technology," Youtube, 12 Únor 2013. [Online]. Available: https://www.youtube.com/watch?v=SkbBM6To9F0&list=PLl-RxKmAhMuqy7L1qaBrTw9soWzv9JfMU&index=3. [Přístup získán 20 Únor 2019].

[3] S. H. Wachman, „Computation Structures MIT 6.004 - L02: Combinational Devices and Boolean Algebra," Youtube, 11 Září 2018. [Online]. Available: https://www.youtube.com/watch?v=hJa6mk8apLo. [Přístup získán 20 Únor 2019].

[4] „Popis předmětu - B2M34SIS - ČVUT - Fakulta elektrotechnická," FEL ČVUT, 20 Únor 2019. [Online]. Available: https://www.fel.cvut.cz/cz/education/bk/predmety/46/74/p4674506.html. [Přístup získán 20 Únor 2019].

[5] „Popis předmětu - B4M34ISC - ČVUT - Fakulta elektrotechnická," FEL ČVUT, 20 Únor 2019. [Online]. Available: https://www.fel.cvut.cz/cz/education/bk/predmety/46/47/p4647306.html. [Přístup získán 20 Únor 2019].

[6] A. Yadin, Computer Systems Architecture, CRC Press, 2016.

[7] [Online]. Available: https://moodle.fel.cvut.cz/pluginfile.php/4956/mod_resource/content/1/A7B14SAP-01-uvod.pdf. [Přístup získán 13 Duben 2019].

[8] J. L. Hennessy a D. A. Patterson, Computer Architecture: A Quantitative Approach, 4th Edition, USA: Morgan Kaufmann Publishers, 2007.

[9] J. Zdeněk, „Moodle FEL ČVUT - A7B14SAP - Struktury a architektury počítačů - B151," FEL ČVUT, 2014. [Online]. Available: https://moodle.fel.cvut.cz/pluginfile.php/4998/mod_resource/content/1/A7B14SAP-12-ascii-radic-CISC-RISC.pdf. [Přístup získán 13 Duben 2019].

[10] T. L. Floyd, Digital Fundamentals, Harlow: Pearson Education Limited, 2015.

[11] MUNI, „P000 Architektura počítačů," [Online]. Available: https://is.muni.cz/el/1433/podzim2004/PB151/um/15090.html. [Přístup získán 22 Květen 2019].

[12] P. Habala, „Diskrétní matematika, 6. Dělitelnost a prvočísla," 2012. [Online]. Available: https://math.feld.cvut.cz/habala/teaching/dma/dmknih06.pdf. [Přístup získán 19 Květen 2019].

[13] „Chegg Study,“ [Online]. Available: https://www.chegg.com/homework-help/questions-and-answers/following-two-questions-two-approaches-construct-comparator-two-8-bit-unsigned-integers-bi-q27711610. [Přístup získán 21 Květen 2019].

11 SEZNAM POUŽITÝCH ZKRATEK A SYMBOLŮ

CMOS	Complementary Metal Oxid Semiconductor
ISA	Instruction Set Architecture = instrukční sada
ALU	aritmeticko-logická jednotka
CISC	Complex Instruction Set Computer
RISC	Reduced Instruction Set Computer
HW	hardware
SW	software
CU	control unit
DFF	D flip-flop = D klopný obvod
NMOS	unipolární tranzistor s N-jámou
PMOS	unipolární tranzistor s P-jámou

12 SEZNAM PŘÍLOH

Přiložené CD obsahuje následující:

- adresář *DP2019* – vytvořená knihovna v programu Cadence Virtuoso
- adresář *obrazky* – exportované obrázky zapojení schémat, simulací a jejích průběhů
- adresář *text*
    - text práce ve formě MS Word
    - text práce ve formě pdf