VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

NÁVRH PROCESORU VE VHDL

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE Miroslav Dvořák
AUTHOR

BRNO 2009

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

NÁVRH PROCESORU VE VHDL

VHDL DESIGN OF CPU

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — Miroslav Dvořák
AUTHOR

VEDOUCÍ PRÁCE — Ing. Martin Straka
SUPERVISOR

BRNO 2009

Abstrakt

Práce se zabývá historií a vývojem procesorů, jejich použitím a zaměřuje se na jeho základní funkční bloky. Nastiňuje návrh vlastní architektury jednoduchého procesoru, který je implementován pomocí programovacího jazyka pro popis číslicových obvodů VHDL.

Abstract

The work deals with history and development of processors, its usage and focuses on its basic function blocks. It outlines design of own architecture of simple processor, which is implemented in programming language VHDL, that is used for description of digital circuits.

Klíčová slova

Procesor, Operační paměť, Aritmeticko logická jednotka, Registr, Řadič, Sběrnice, Instrukce, Operace, Operand, Mikroprogram, Mikroinstrukce, Řídící signál

Keywords

Processor, Computing memory, Arithmetical logical unit, Register, Controller, Bus, Instruction, Operation, Operand, Microprogramme, Microinstruction, Control signal

Citace

Dvořák Miroslav: *Návrh procesoru ve VHDL*, bakalářská práce, Brno, FIT VUT v Brně, 2009

Návrh procesoru ve VHDL

Prohlášení

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

……
Miroslav Dvořák
20.5.2009

Poděkování

V úvodu práce bych rád poděkoval **Ing. Martinu Strakovi** za jeho odborné konzultace, náměty, připomínky a doporučený materiál.

Obsah

1 Úvod

Cílem této práce je podrobnější rozbor funkčních bloků procesoru, jakožto řídící jednotky především osobních počítačů, vestavěných systémů a dalších zařízení. Díky sledování historických stop, je možné diskutovat vývoj různých architektur, koncepcí a trendů, které ovlivnily celé odvětví vývoje a výroby procesorů. Zmíněn bude také jazyk VHDL, určený pro popis číslicových systémů. Předvedeny budou základní metody popisu obvodů v tomto jazyce pomocí názorných příkladů. Na základě zmíněných informací bude nakonec navržen a v jazyce VHDL implementován jednoduchý model procesoru, který by měl být schopen provádět základní aritmetické a logické operace. Ten bude doplněn jednoduchou operační pamětí pro možné provádění simulací a testů návrhu na jednoduchých programech. Mimo návrh samotného procesoru se v práci budeme zabývat také návrhem mikroprogramového řadiče a jeho včleněním do navržené architektury.

2 Jazyk VHDL

VHDL je programovací jazyk z rodiny HDL, který je určen k popisu hardware. Používá se k návrhu a simulacím digitálních integrovaných obvodů. Zkratka VHDL ve skutečnosti znamená VHSIC Hardware Description Language, kde VHSIC je zkratka pro „ Very-High-Speed Integrated Circuit. VHDL je typovaným programovacím jazykem. Disponuje prostředky pro popis paralelismu, explicitního vyjádření času a konektivity. Jazyk VHDL se stal IEEE standardem v roce 1987 a byl revidován v roce 1997. [1, 5, 15]

Alternativou k jazyku VHDL, je jazyk Verilog vycházející ze syntaxe jazyků C a Ada, který je také používán pro popis, návrh a testování číslicových obvodů. Použitelnost obou jazyků je z hlediska jejich modelovacích schopností téměř totožná. Jedním z rozdílů mezi jazyky jsou datové typy. VHDL umožňuje uživateli definovat a používat vlastní datové struktury, kdežto Verilog se v základu omezuje na ty nejjednodušší datové typy a neumožňuje vytváření nových, tím se stává Verilog na druhou stranu o něco jednodušším jazykem pro začátečníky. VHDL je také přizpůsobeno znovupoužitelnosti napsaného kódu, zejména funkcí, jelikož je umožňuje seskupovat do balíčků a ty následně přidávat do různých dalších designů. Jazyk Verilog takový koncept nepodporuje a všechny funkce musí být uvnitř modulu. Proto je také VHDL vhodnější pro tvorbu větších projektů.

2.1 VHDL a podpora návrhu

VHDL umožňuje na nejobecnější úrovni popsat chování i velmi složitých elektronických obvodů. Disponuje prostředky pro popis vlastností navrhovaného objektu, což umožňuje strukturovaný návrh. Navíc součástí jazyka VHDL jsou prostředky, kterými můžeme popsat paralelně probíhající děje, což

je velký rozdíl oproti sekvenčnímu kódu běžných programovacích jazyků. Samozřejmě, jelikož se jedná o jazyk, zaměřený na popis obvodů, má programátor k dispozici prostředky, které pracují s událostmi vlastními těmto obvodům. Je například schopen zachytit vzestupné a sestupné hrany signálu, nebo pouhou změnu úrovně.

2.2 VHDL a podpora testování

Od programování v běžných jazycích pro vývoj software se programování v jazyce VHDL liší právě testováním. Je běžné souběžně s návrhem vyvíjet zvláštní design zvaný testbench, jehož úkolem je testovat návrh v čase. Je založen na principu přivedení vstupních budících signálů a sledování odezvy námi projektovaného designu na tyto signály.

2.3 Základní konstrukce a styly popisu architektury

Pozn.: následující kapitola vychází z [1, 5, 15]

Při popisu hardware pomocí jazyka VHDL se nejčastěji využívá dvou základních konstrukcí:

1. entita (entity) – pomocí entity se definuje rozhraní prvku (porty), každý port musí mít svůj název, typ a mód.
   Módy portů:
   - OUT – výstupní port, data vystupují tímto portem z entity
   - IN – vstupní port , data vstupují tímto portem do entity
   - INOUT – vstupněvýstupní port, data mohou pomocí tohoto portu vstupovat i vystupovat do/z entity
   - BUFFER – výstup se zpětnou vazbou
   - LINKAGE – spojovací port, používá se, když není přesně znám směr toku dat

2. architektura (architecture) – popisuje chování jednotlivých entit. Pro jednu entitu může existovat více architektur.

Pro popis architektury je možno využít jeden ze tří stylů popisu nebo je navzájem kombinovat. Každý z těchto stylů představuje různou úroveň abstrakce návrhu.

- Behaviorální popis - neboli popis chování je nejvyšší úroveň abstrakce. Popis obvodu je realizován pomocí operací v čase. Právě role času, je rozdílem od nižších úrovní abstrakce.
- Dtaflow popis – Popis na úrovni datových toků umožňuje popisovat pohyb dat v navrhovaném systému

- Strukturální popis – je možné použít jak pro popis na nízké úrovni, třeba úroveň tranzistorů, tak i na vysoké úrovni, například blokový diagram

2.4 Příklad syntaxe

Pozn.: příklad založen na [5]

2.4.1 Behaviorální popis

```
-- (tohle je komentář)

-- import std_logic z knihovny IEEE
library IEEE;
use IEEE.std_logic_1164.all;

-- definice entity (vstupní a výstupní porty)
entity muj_or is
  port (IN1, IN2 : in std_logic; OUT1: out std_logic);
end entity;

-- behaviorální popis architektury (chování definované entity),v tomto případě se jedná o člen OR.

architecture behavioral of muj_or is
begin
  OUT1 <= IN1 or IN2;
end behavioral;
```

2.4.2 Dataflow popis

```
-- import std_logic z knihovny IEEE
library IEEE;
use IEEE.std_logic_1164.all;

-- definice entity (vstupní a výstupní porty)
entity muj_or is
  port (IN1, IN2 : in std_logic; OUT1: out std_logic);
end entity;

-- architektura popsaná pomocí toku dat, v tomto případě se jedná o člen OR.

architecture dataflow of muj_or is
begin
  with IN2 select
    OUT1 <= IN1 when '0',
            `1` when `1`;
end behavioral;
```

2.4.3 Strukturální popis

```
library IEEE;
use IEEE.std_logic_1164.all;

-- definice entity je v tomto případě prázdná, jedná se o horní vrstvu
návrhu, propojující mezi sebou vrstvy nižší, v tomto případě předává
signál RESET z komponenty tb do komponenty cpu.
entity top is
  end entity;

architecture structural of top is
  signal top_RESET : std_logic;
  component cpu_com port (RESET : in std_logic;); end component;
  component tb_com port (RESET : out std_logic;); end component;

begin
  cpu: entity work.cpu
   port map(RESET => top_RESET);
  tb: entity work.tb
   port map(RESET => top_RESET);
end structural;
```

3 Procesor

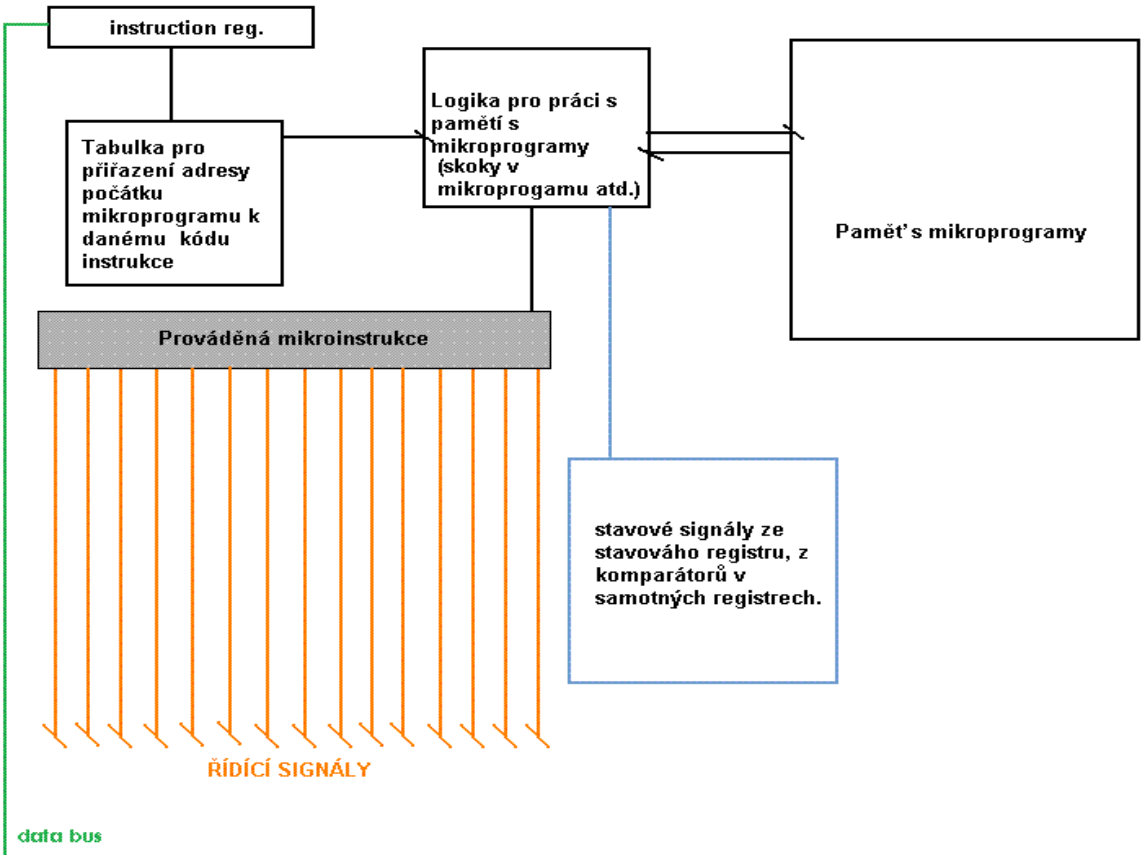
Pojmem procesor se nejčastěji označuje programovatelný integrovaný obvod, jehož úlohou je číst postupně instrukce uložené v paměti a podle nich zpracovávat data. Hovoříme tedy o součástce, přítomné ve spoustě zařízení, kde v každém plní svoji specifickou funkci. Jako procesor, nebo také mikroprocesor označujeme ale nejčastěji centrální jednotku počítače neboli CPU. Speciálním druhem procesorů jsou takzvané mikrokontrolery. Jedná se o zařízení v nichž jsou mimo procesoru integrovány také časovače, čítače, paměť, generátory přerušení a porty pro různé vstupy a výstupy. Systém ovládaný mikrokontrolerem označujeme jako vestavěný systém, pro příklad můžeme uvést mobilní telefon, mikrovlnnou troubu, a jiné.

3.1 Součásti a funkce procesoru

3.1.1 Řadič

Samotný procesor je řízen řadičem procesoru, ten zajišťuje dekódování načtených instrukcí, řídí načtení příslušných operandů a posléze dle kódu instrukce zajistí její patřičné provedení v některé z dalších jednotek procesoru. Řadič procesoru může být navržen jako konečný stavový automat, nebo mikroprogramový řadič viz obr. 3-1 Architektura mikroprogramového řadiče. Obě varianty mají své

výhody i nevýhody a obě řešení jsou využita v dnešních procesorech. Řadič realizovaný konečným stavovým automatem, je většinou použit na jednodušší instrukce, jejichž provedení spočívá v několika málo krocích, tyto instrukce jsou pak provedeny v poměrně krátkém čase a proto je takovéto řešení výhodné u často používaných triviálních instrukcí. Mikroprogramový řadič je tvořen pamětí, která obsahuje mikroprogramy pro každou instrukci, a logikou, pro zpracování mikroprogramu. Po dekódování instrukce je tedy zahájeno provádění příslušného mikroprogramu, jehož výsledkem je dokončení instrukce. Takovéto řešení poskytuje možnost implementovat složité a rozsáhlé instrukce. Zároveň, jelikož jsou všechny mikroprogramy uloženy v paměti, nabízí se snadná cesta ke změnám v instrukční sadě takového procesoru.

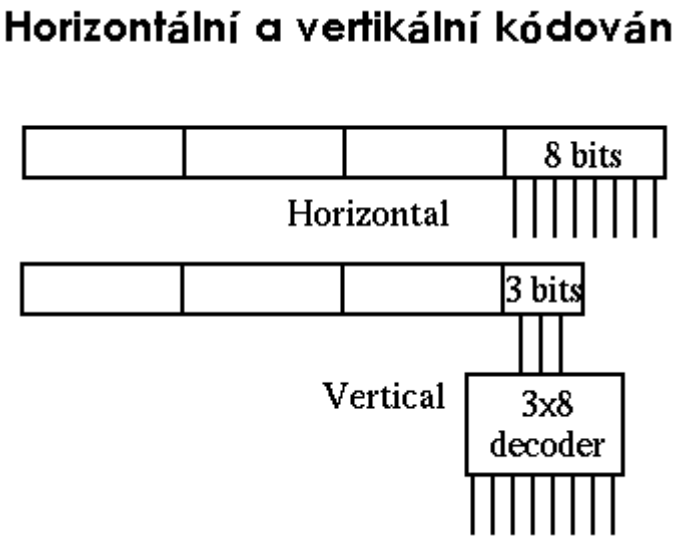


Obr.: 3-1: Architektura mikroprogramového řadiče

Většina mikroinstrukcí se dá rozdělit na základní bloky, z nichž každý kontroluje určitou část procesoru, např.:

- Blok, který ovládá datové cesty (načítání operandů, ukládání výsledků)
- Blok, který řídí výkonnou jednotku (např. ALU)
- Blok, který řídí vstupně výstupní porty procesoru (např. systémová sběrnice)
- Blok, který kontroluje vykonávání programu jako celku (např. ovládá program counter).
- Blok, pro kontrolu podmínek a řízení skoků v mikroprogramu

Jednotlivé bloky mohou k ovládání používat velmi jednoduché signály, například jednoduchou ALU jednotku, bude stačit řídit pomocí opcode, což mohou být i pouhé 4 signály. Některé bloky mohou být složitější a k ovládaní používají velkou spoustu signálů, takovým blokem může být blok pro řízení datových cest, u složitějších procesorů, může být zapotřebí nastavovat velké množství multiplexorů atd. V takovém případě by celá mikroinstrukce mohla být velmi dlouhá a vyvstává tak požadavek na velkou paměť pro uchování mikroprogramů. Podle těchto aspektů je voleno mezi horizontálním a vertikálním kódováním mikroinstrukcí viz obr.: 3-2 kódování mikroinstrukcí. [13] Horizontální kódovaní znamená, že každý signál je v mikroinstrukci reprezentován svým příslušným bitem. Například pokud máme 8 registrů, tak v mikroinstrukci bude blok pro 8 řídících signálů, jedná se o způsob zápisu mikroinstrukcí, který je náročný na paměťový prostor, ovšem vykonávání takového mikroprogramu je rychlejší. Vertikální kódování je založeno na myšlence sdružit skupiny signálů, z nichž pouze jeden může být v určitý čas aktivní a zakódování informace o aktivaci tohoto signálu do kratšího kódu, který ovšem musí být následně dekódován přídavnou logikou. Například, pokud máme 8 registrů a pouze do 1 můžeme v určitý čas zapisovat, nemusíme mikroinstrukcí přímo ovládat daných 8 signálů, ale postačí nám 3 signály, které budou představovat kód pro logiku, která následně dle tohoto kódu aktivuje signál pro daný registr. Toto řešení zkracuje mikroinstrukce, snižuje požadavky na velikost paměti mikroprogramů, určitým způsobem snižuje přímou čitelnost kódu mikroinstrukce a celých mikroprogramů, ale díky nutnosti použít přídavnou logiku, je pomalejší. U jednodušších datových cest je možné neovládat je přímo mikroprogramem, ale při dekódování instrukce, použít logiku na bázi konečného stavového automatu, která podle adres operandů, nastaví příslušné datové cesty.



Obr.: 3-1: Kódování mikroinstrukcí

Další důležitou vlastností mikroinstrukcí je, zda vykonávají mikroprogram sekvenčně, v každé mikroinstrukci jediný krok, nebo zda se některé kroky paralelizují do jediné mikroinstrukce. V případě vykonávání více kroků v jedné mikroinstrukci, je zapotřebí dbát velké opatrnosti, aby se prováděné kroky navzájem neovlivňovaly, nepoužívaly shodné prostředky nebo stejnou část datových cest.

3.1.2 Registry

Dalším prvkem procesoru jsou registry, jedná se o rychlé pracovní paměti malé kapacity, které mohou sloužit k různým účelům. Nejčastěji rozeznáváme registry obecné, ty se dále dělí na pracovní a univerzální, a poté registry řídící, mezi ně řadíme stavové registry, registry adres instrukcí nebo index-registry [13]. Takový soubor registrů nazýváme sadou registrů. Některé procesory mohou mít takových sad více, je to výhodné zejména při vstupu do podprogramu, přerušení nebo přepínání mezi úlohami (typické pro RISC architekturu). Není pak nutné přesouvat veškerý obsah registrů do paměti a následně jej navracet z paměti zpět, ale jednoduše pro provedení například podprogramu se přepne na následující sadu registrů a při výstupu z podprogramu se vrátíme k sadě původní. Tento návrh může výrazně zrychlit vykonávání například rekurze, ale zároveň je ekonomicky velice nákladný, jelikož cena výroby takových rychlých pamětí, kterými jsou registry tvořeny, je vysoká. Registry mohou být i součástí řadiče procesoru, zde mohou sloužit například k uchování hodnot mezivýsledků.

3.1.3 Aritmeticko logická jednotka (ALU)

Prvkem, kterému je svěřeno vykonávání aritmeticko-logických operací v pevné řádové čárce je jednotka ALU (Arythmetical-Logic Unit). Jednotek ALU může být v jednom procesoru více. Tato koncepce zohledňuje fakt, že v případě dvou po sobě jdoucích instrukcích, operandy druhé instrukce nemusí být závislé na výsledku první operace. Takovéto instrukce není nutné tedy zpracovat sériově, ale je možné je provést paralelně, což opět urychlí vykonávání kódu.

3.1.4 Jednotka pro operace v plovoucí řádové čárce (FPU)

Jednotkou, která v osobních počítačích měla nejprve podobu externího koprocesoru je jednotka FPU, dnes už je ve stolních počítačích integrována přímo do čipu CPU. Jedná se o blok, který vykonává operace v plovoucí řádové čárce. Návrh architektury tohoto prvku je velmi náročný a zároveň má velký vliv na rychlost procesoru, jelikož operace v plovoucí řádové čárce jsou jedny z nejnáročnějších, běžně užívaných operací. Jednotek FPU může být v jednom čipu opět více.

3.1.5 Další části procesoru

Existuje spousta dalších funkčních jednotek, které v dnešní době tvoří procesor. Zmíním alespoň rychlé vyrovnávací paměti tzv. cache, které mívají zpravidla několik úrovní (u nejnovějších procesorů pro osobní PC L1, L2 a L3), z nichž ta nejnižší bývá tvořena nejrychlejší pamětí a zároveň z ekonomického hlediska bývá nejmenší. V dnešní době mají jak procesory AMD (od architektury K8), tak Intel (od Core i7) integrován paměťový řadič. V procesorech osobních počítačů nejsou instrukce načítány postupně po jedné a následně až dekódovány a provedeny, ale existují fronty, do kterých jsou instrukce načítány dříve než má dojít k jejich zpracování, v této frontě jsou již částečně dekódovány. Největším problémem této koncepce, je metodika předvídání skoků, které mají za následek, že všechny instrukce v takové frontě se najednou mohou stát nepotřebnými, s tím souvisí jednotky předvídání a logiky skoků.

3.2 Základní parametry procesoru

Využití procesoru, jakožto základní výpočetní a řídící jednotky systému je velice rozšířená koncepce a pro každé takové zařízení, může být esenciální odlišný parametr.

U stolních počítačů je bezesporu nejdůležitějším parametrem procesoru jeho rychlost, která je z velké části závislá na pracovní frekvenci procesoru. Musíme si ale uvědomit, že i samotná architektura a další činitelé ovlivňují výsledný výpočetní výkon CPU. Ne vždy tedy musí platit, že čím vyšší frekvence, tím vyšší výpočetní výkon. Příkladem z běžného užití může být porovnání architektur NetBurst a Core nebo novější Nehalem. Typickým zástupce architektury NetBurst může být procesor Pentium 4 Prescott, tyto procesory dosahovaly pracovní frekvence až 3,8GHz, zatímco procesory architektur Core nebo Nehalem pracují na kmitočtech okolo 3GHz, přesto je jejich výpočetní výkon vyšší.

Přesnějším parametrem pro měření rychlosti procesoru je počet operací provedených za jednu sekundu, udává se v jednotkách MIPS (milonů operací za sekundu). Pro představu můžeme použít srovnání architektur Core a Nehalem, kdy nejvýkonnější představitel architektury Core – QX9700 dosáhl v benchmarku 7-Zip 12187MIPS a nejvýkonnější zástupce architektury Nehalem – Corei7 965 dosáhl ve stejném benchmarku hodnoty 18322MIPS.

Velmi důležitá je také instrukční sada. Některé procesory (CISC) mohou disponovat velmi rozsáhlou instrukční sadou, zahrnující i složité a speciální instrukce, u některých zase může být žádoucí, aby obsahovaly jen potřebné minimum instrukcí (RISC), které jsou jednoduché a jejich provádění je rychlé [14].

Další vlastností je šířka operandu v bitech (šířka slova), zde platí neustálý trend rozšiřování. Jedny z prvních procesorů, jako například Intel 4004 z roku 1971, byly 4 bitové, dnešní procesory osobních počítačů jsou již většinou 64bitové. Toto číslo vyjadřuje kolik bitů může mít operand zpracovatelný v jednom kroku.

Frekvence datové sběrnice je další parametr, který ovlivňuje především rychlost práce s operační pamětí.

Parametrem, který začal hrát v poslední době velkou roli u procesorů pro osobní počítače je počet jader, koncepce více-jádrových CPU umožňuje využívat paralelního zpracovávání instrukcí.

3.3 Podle jakých kritérií můžeme dělit procesory, také DSP a mikroprocesory

Procesory můžeme dělit hned podle několika kritérií, například podle délky operandu v bitech, podle struktury procesoru nebo podle počtu jader.

- dělení podle délky operandu v bitech

  Délka operandu v bitech, udává šířku operandu, který je schopný procesor zpracovat v jednom krorku. Například 8bitový procesor je schopný pracovat s čísly 0 – 255 přímo, pokud je číslo větší (operand je širší) zpracovává se číslo v několika krocích.
  Nejjednodušší 4bitové a 8bitové procesory se používají nejčastěji pro zabudované tzv. embedded systémy (počítačová klávesnice, pračka, mikrovlnná trouba…)
  Pro složitější aplikace, jako jsou mobilní telefony, přenosné videohry apod., se používají 8bitové nebo 16bitové procesory.
  Ty nejsložitější zařízení, kam řadíme třeba osobní počítače, používají potom 32bitové a dnes již nejčastěji 64bitové procesory.

- dělení podle vnitřní architektury

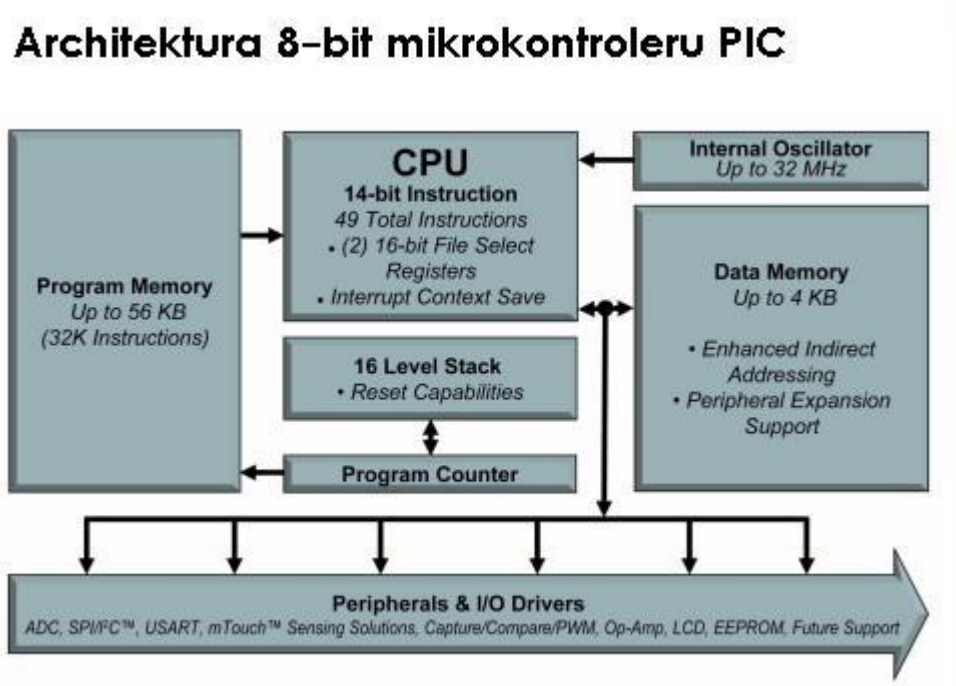
  Podle tohoto dělení jsou procesory zařazeny do skupin RISC nebo CISC (můžeme se setkat i s procesory, které obsahují prvky typické jak pro jednu, tak druhou skupinu).
  Procesory RISC disponují menší instrukční sadou, která je založena především na jednoduchých, snadno proveditelných instrukcích, tato architektura má velmi omezené prostředky pro práci s pamětí, většina z nich si vystačí s instrukcemi „LOAD“ a „STORE“. Nevýhodou této koncepce je větší spotřeba paměti na program.
  Procesory CISC mají zase rozsáhlé instrukční sady, obsahující i složité instrukce, jejich provádění může být i poměrně časově náročné, výhodou je menší spotřeba paměti a jednodušší kód programu, ale na druhé straně ve většině případů vykoupená

delším výpočetním časem, než by tomu bylo u stejného problému řešeného pomocí jednodušších instrukcí architektury RISC.

- Dělení podle speciální funkce

Jednočipový mikropočítač (mikrokontrolér – MCU)

[2] Procesor s univerzálním jádrem, s kterým jsou současně zaintegrovány základní periferní obvody, takže je schopen samostatné funkce. Za průkopníky v této kategorii můžeme považovat 8bitový procesor Intel i8051, který poprvé integroval všechny základní periferie (jádro procesoru, paměť RAM, EEPROM, čítače a časovače) na jediném čipu a 16bitový technologický procesor Siemens SAB 80C166, který poprvé integroval A/D převodníky, komunikační linky a masivní systém čítačů/časovačů/přerušení (následníky řady 80166 dnes vyrábí Infineon (řada C167 a C166 SV2) a SGS Thomson (řada ST10)). Příklad architektury viz obr.: 3-3 Architektura MCU PIC.

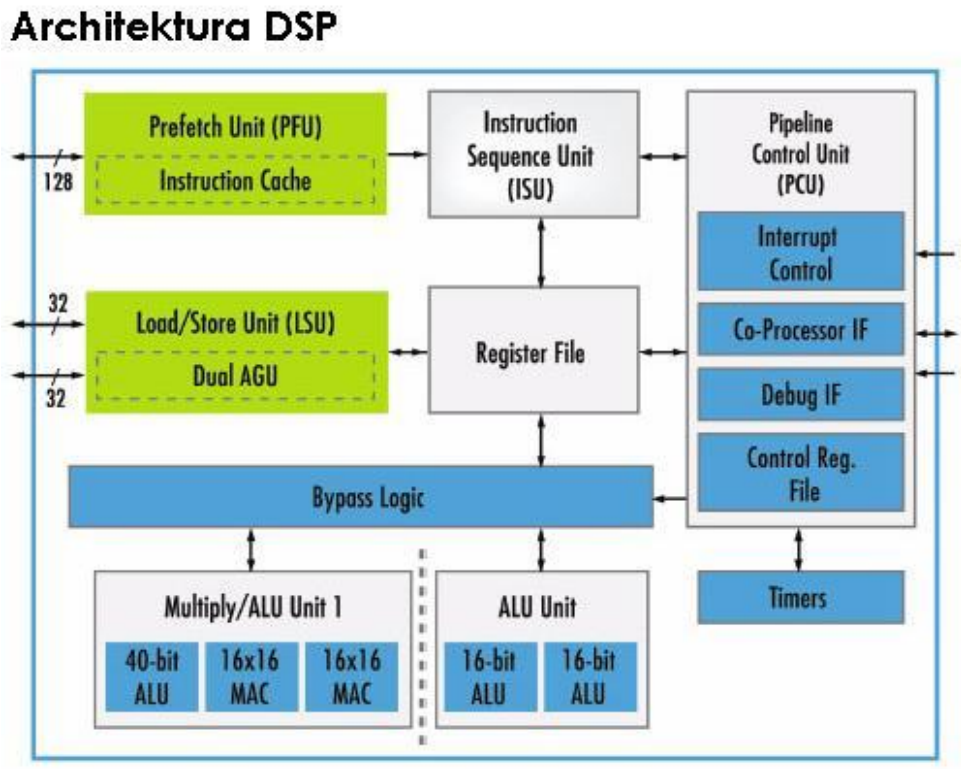


Obr.: 3-2: Architektura MCU PIC

Digitální signálový procesor (DSP)

[3] DSP processor, je takový druh procesoru, který je specializovaný na zpracování signálu. Jejich architektura je přizpůsobena tak, aby byly schopny co nejrychleji vykonávat jednoduché matematické algoritmy (viz obr.: 3-4 architektura DSP), používané při zpracování signálů. DSP procesory se používají často jako filtr signálu, své využití naleznou v telekomunikačním, nebo také hudebním průmyslu. Často můžeme rovněž pozorovat snahu o spojení výhod DSP a jednočipových

mikropočítačů ať už je to cestou rozšiřování DSP o periferie nebo rozšiřováním mikrokontrolérů o DSP jednotky [2].



Obr.: 3-3: Architektura DSP

- dělení dle počtu jader

  Zvyšování počtů jader na jednom čipu je trendem doby blízko minulé zejména na poli procesorů pro osobní počítače. Procesory tedy můžeme dělit na jednojádrové a vícejádrové. Prvním náznakem tohoto trendu byla technologie HT (hyperthreading)[4] u precosorů Intel Pentium 4. Tato technologie je založena na myšlence, že většina procesů není schopna využít veškerou výpočetní kapacitu jádra procesoru a procesor je schopen takto nevyužité výpočetní jednotky přidělit jinému procesu. Fyzicky se tedy vlastně stále jedná o jednojádrový procesor, ale operační systém s ním již zachází jako s vícejádrovým. První čistokrevný vícejádrový procesor firmy Intel pro širokou veřejnost byl Pentium D a u firmy AMD jím byl Athlon64 X2. Jak plyne z názvů, tak první varianty měly 2 jádra, ty byly velmi krátce následovány 4 jádrovými modely a za zmínku stojí i velice rozpačitě přijímaný 3 jádrový procesor firmy AMD (Phenom X3). Koncepce vícejádrových procesorů není zcela jednoduchá, nejedná se totiž o naprosto samostatné jednotky, ale sdílí mezi sebou i některé systémové prostředky jako například vyrovnávací paměť vyšší úrovně (většinou druhé a vyšší). Tento fakt je založen na myšlence snadného předávání dat mezi jednotlivými jádry. Paralelismus s sebou ale přináší samozřejmě i stinné

stránky. Za prvé se jedná o navyšování „paralelního výkonu“, to znamená, že takovýto procesor poskytne vyšší výpočetní sílu jen aplikacím, které jsou rozloženy na více vláken a mohou být tedy zpracovány paralelně, to má za následek potřebu jak nových postupů při programování takových aplikací, tak i nových vývojových nástrojů, které tuto koncepci zohledňují. Za druhé zvyšování počtu jader na jednom čipu bude problematické i z hlediska návrhu samotného procesoru, jelikož právě propojení a správa sdílených prostředků mezi jádry bude velmi problematická. U procesorů pro osobní počítače, které zde byly dosud diskutovány, se jedná o jádra ekvivalentní, to znamená, že každé jádro je schopno tentýž úkol zpracovat shodným způsobem. Ale toto nemusí být jediná cesta, například architektura procesoru IBM, který je využíván v herních konzolích Sony Playstation 3, používá jedno komplexnější jádro, které přerozděluje práci mezi 6 jednodušších DSP, jež jsou vzájemně propojeny hradlovým polem. Zde se tedy setkáváme se specializací jednotlivých částí na specifické úkony.

4 Historie vývoje procesorů

Dnes jsme zvyklí, že CPU počítače nebo většina MCU se nám vejde pohodlně do dlaně, ale samozřejmě ne vždy tomu tak bylo. Procesory počítačů první generace byly elektromechanické a hlavními součástkami byly elektronky. Takový procesor mohl zabírat celou skříň, nebo dokonce několik skříní. K podstatným změnám došlo až v 70. letech 20. století s příchodem třetí generace počítačů, založené na integrovaných obvodech. Takový procesor se skládal z několika desítek integrovaných obvodů nazývaných procesorové řezy. Ve chvíli, kdy míra integrace pokročila natolik, že byl celý procesor integrován jako jeden čip, hovoříme o mikroprocesoru. Prvním mikroprocesorem byl čip Intel 4004. Byl čtyřbitový a pracoval na kmitočtu 108kHz. Následovala spousta 8mi bitových procesorů na různých frekvencích. Až k procesoru 8086, který byl prvním 16bitovým procesorem Intelu a položil základy architektury počítačů x86. Tato architektura byla na začátku 21. století postupně nahrazena 64bitovou architekturou, ale stále (v roce 2009) ještě nezanikla a je hojně používána. [10,11]

4.1 Koncepce počítače s uloženými instrukcemi

První návrhy počítačů se hodně lišily od toho, co známe v dnešní době. Neexistovalo nic jako program, který by představoval sled instrukcí, které mají být provedeny nad určitými daty. Nastavení určité operace v každém kroku muselo být provedeno ručně a až poté byla operace vykonána.

Koncepce číslicového počítače přináší pojem „program“, který představuje jakýsi návod pro počítač jak docílit ze vstupních dat požadovaného výsledku. Tento algoritmický postup spolu s implementací jednotlivých instrukcí provádějící operace umožnil vznik univerzálních počítačů a projevil se jako velmi efektivní. [12]

Existují 2 základní koncepce digitálního počítače:

- Von Neumanova koncepce – využívá oddělené paměti pro data a program, tato varianta může být odolnější na programátorské chyby při práci s pamětí.
- Harvardská koncepce – spojuje paměť dat i programu

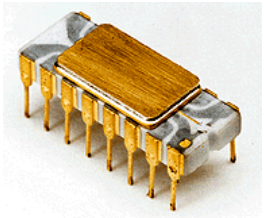
Současné osobní počítače nespadají vyhraněně ani do jedné z nich. Mají sice jedinou operační paměť RAM, v které jsou uložena jak data, tak program, ovšem disponují také technikami, které umožňují část paměti, kde se nachází pouze program, uzamknout pro zápis a z této oblasti je možné jen číst. Harvardské schéma je využíváno u jednočipových počítačů a jiných vestavěných systémů (mobilní telefony, PDA…) a zejména u signálových DSP procesorů, kde dovoluje zpracovávat data velmi rychle [12].

4.2 Chronologický přehled nejvýznamnějších procesorů Intel a AMD pro PC

Pozn.: následující kapitola čerpá z [10,11]

1971

- **Intel 4004** (na obr.: 4-1 Intel 4004) – 4bitový, pracovní frekvence 108kHz, asi 1000x nižší výpočetní výkon než současné procesory, jeho součástí není FPU



Obr.: 4-1: Intel 4004

1978

- **Intel 8086** – první 16bitový procesor Intelu, základ architektury počítačů x86, pracovní frekvence 5 – 10 MHz

1979

- **AMD Am2901** – poprvé se objevuje na trhu společnost Advanced Micro Devices, 4bitový procesor

1986

- **Intel 80386DX** – navazuje na architekturu x86, ale rozšiřuje ji natolik, že se počítá jako zakladatel architektury i386. Pracovní frekvence 16-33MHz. Hojně využívaný v osobních počítačích. Má 32bitovou adresovou sběrnici. Nemá interní cache. 3 režimy (reálný,chráněný, virtuální). Sada registrů rozšířena o 32bit registry. Úsporná verze I80386SX určena pro notebooky (16-25MHz)

1989

- **Intel 80486** –plně 32bitový procesor. První procesor s dostatečným výkonem pro grafické a multimediální operace. Pracovní frekvence 25-100MHz. Jeho součástí je matematický koprocesor pro operace v plovoucí desetinné čárce. Na trhu se objevila opět i úsporná varianta I80486SX.

1993

- **Intel Pentium P5** (viz obr. 4-2 Pentium) – Pentium je nástupce 80486, jeho pracovní frekvence je 60MHz, součástí je matematický koprocesor a 16kB L1 cache.

Obr.: 4-2: Intel Pentium

1995

- **AMD K5** – Procesor vyvinutý od začátku firmou AMD. Jeho FPU koprocesor byl pomalejší než u Pentií, proto se nijak výrazněji neprosadil

1997

- **Intel Pentium MMX** – disponoval 32kB L1 cache, pracovní frekvence se pohybovala od 133 do 233MHz. Byl vybaven speciální sadou instrukcí, zvanou MMX, která slouží pro operace s multimédii.
- **AMD K6** – Byl o něco výkonnější než Pentium a Pentium Pro na shodné frekvenci, disponoval 57mi instrukcemi MMX
- **Intel Pentium II** – založen na modifikovaném jádře P6, disponoval navíc MMX instrukcemi, 233MHz - 500MHz, 32kB L1 cache a 512kB L2 cache, která pracovala na 50% rychlosti CPU, poslední modely měly poloviční velikost L2 cache,ale zato shodný takt s jádrem. Odlehčená verze Celeron, menší L2 cache (první modely žádná L2 cache). Modely Celeron doprovází prakticky všechny výkonné procesory Intel dodnes.

1998

- **AMD K6-2** – Přímá konkurence procesorů Pentium II a Celeron. Disponoval 64kB L1 cache Pracoval na frekvencích 233-550MHz.

1999

- **Intel Pentium III** – frekvence 450MHz - 1400MHz (Tualatin). Hlavním rozdílem oproti PentiuII bylo implementování SSE instrukční sady, jako odpověď na instrukční sadu 3Dnow! Od AMD. Na obrázku 4-3, můžeme vidět rozdíl mezi provedením pro Slot a Socket.

Obr.: 4-3: Pentium III

- **AMD K7 – Athlon** – První procesor s názvem Athlon, které ovládli trh po roce 2000. První modely pracovaly na frekvenci 500MHz.

2000

- **Intel Pentium 4** procesory se superskalární architekturou NetBurst, která slibovala vysoké pracovní frekvence, kterých díky problémům s chlazením a vysoké spotřebě nebylo nikdy dosaženo. Frekvence 1,4GHz - 3,8GHz. U jádra Northwood se objevuje nová Instrukční sada SSE2 a jádro Prescott (65nm výrobní proces) nově i SSE3. Počátek technologie Hyper-Threading (viz. [4]). V roce 2005 z Pentia 4 vzešla dvoujádrová verze v podobě Pentia D.

2001

- **AMD Athlon XP** (viz. obr. 4-4)–Na stejném taktu jako Pentium 4 dosahovaly nepoměrně vyššího výpočetního výkonu. Stejně tak v porovnání s prvními Athlony, proto bylo od tohoto modelu zavedeno poměrné značení, které mělo udávat na jaké frekvenci by musel pracovat původní Athlon, aby dosáhl stejného výkonu.

Obr.: 4-4: AMD Athlon XP

2003

- **Intel Pentium M** – procesor určený pro mobilní počítače, je založen na jádru Pentia III a jeho architektura je mnohem výkonnější, než architektura NetBurst Pentia 4. Pentium M 1,6GHz má přibližně ekvivalentní výkon jako Pentium 4 2,4GHz.

2004

- **AMD Athlon64** – nástupce Athlonů XP, jedná se o první 64bitové procesory dostupné pro širokou veřejnost. Došlo k integraci paměťového řadiče do čipu procesoru. S tím souvisí nahrazení sběrnice FSB sběrnicí Hyper Transport. Athlon64 se dočkal ještě v tomto roce také modifikace pro mobilní počítače. O rok později z řady E vzešel dvoujádrový Athlon X2.

2005

- **Intel Core Duo** – První zástupce procesorů postavených na architektuře Core, která historicky vyšla z architektury P6. Jedná se o první dvoujádrový mobilní procesor Intelu, který položil základy velmi úspěšné řadě procesorů Core 2.

2006

- **Intel Core 2 Duo/Quad** – Dvoujádrové/čtyřjádrové (spíše 2x2 než nativní 4jádro) procesory založené na architektuře Core, primárně určeny pro stolní počítače, ale velice rychle se objevily i modely pro laptopy. Instrukční sada těchto procesorů obsahuje i 64bitové (EM64), instrukce, přibyla ještě sada SSE 4.1. Vyráběny 65nm a 45nm výrobními procesy.
- **AMD Phenom** – dlouho očekávaná odpověď firmy AMD na architekturu Core. Procesory Phenom existuje ve 2, 3 a 4 jádrové verzi. Novinkou u architektury K10 je integrování L3 cache přímo do procesoru.

2008

- **Intel Celeron Dual-Core** – první dvoujádrový procesor určený do levného segmentu. L2 cache o velikosti pouhých 512kB sdílená pro obě jádra.
- **Intel Core i7** (viz. orb. 4-5)– Architektura Nehalem, 45nm výrobní proces. Instrukční sada rozšířena o instrukce SSE4.2. 4 fyzická jádra + technologie HT poskytují 8 logických výpočetních jednotek. FSB nahrazena sběrnicí QuickPath Interconnect. Integrován PCI-Express a paměťový řadič přímo. L3 cache integrována také přímo do procesoru.

Obr.: 4-5: Intel Core i7

- **AMD Phenom II** – jádro K10.5 se stalo nástupcem jader K10 Phenomů. Jsou vyráběny 45nm výrobním procesem. Integrován byl řadič DDR3 pamětí a L3 je oproti K10 třikrát větší (6MB)
- **Intel Atom** –mobilní procesor, díky 45nm výrobnímu procesu je úžasně malý (15mm v průměru) a dosahuje velice nízké spotřeby do 10W.

4.3 Stručný přehled patic pro procesory osobních počítačů

Patice procesoru je součást základní desky, která nám umožní připojit procesor do systému. V historii byly procesory pájeny napevno k základní desce, to ovšem neumožňovalo jejich výměnu a tak s postupem vývoje vznikly patice. Jednalo se buď o sloty nebo sockety (viz. obrázek 4-3), dnes se setkáváme již zpravidla pouze se sockety. Sloty měly usnadňovat výměnu procesoru, koncepce zahrnovala prostředníka v podobě přídavné karty na níž se nacházel procesor a ta se teprve vkládala do slotu na základní desce. Stručný přehled nejvýznamnějších patic viz. tabulka 4-1.

Patice	*Pinů*	*Podporované procesory*
Socket 1	**169**	80486DX,80486SX,80486DX2,80486DX4
Slot 1	**242**	Pentium II, Pentium Pro, Celeron, Pentium III
Socket 370	**370**	FCPGA - Pentium III, Pentium IV, Celeron, VIA Cyrix III, VIA C3(800-1GHz)
Socket A	**462**	AMD ATHLON(Thunderbird,Palomino),AMD DURON(Morgan)
Socket 423	**423**	Pentium IV(Wilamette)
Socket 478	**478**	Pentium IV (Northwood a některé Prescott)
Socket 754	**754**	AMD Athlon 64, AMD Sempron, AMD Turion 64, AMD Mobile Athlon 64
Socket 775	**775**	Pentium 4, Pentium 4 Extreme, Celeron D, Pentium D, Pentium Dual-Core, Core 2 Duo, Core 2 Extreme, Core 2 Quad, Xeon, Celeron ('Core'), Celeron Dual-Core
Socket 939	**939**	Athlon64, Athlon64 X2, Athlon64 FX, Opteron, Sempron
Socket AM2/+	**940**	Athlon64, Athlon64 X2, Athlon64 FX, Opteron, Sempron, Phenom
Socket AM3	**941**	Phenom II
Socket 1366	**1366**	Core i7

Tabulka 4-1: Přehled patic

5 Současné trendy ve vývoji CPU

Základní snahou, kterou lze sledovat při vývoji od samého počátku až po současnost je zvyšování výpočetního výkonu. U osobních počítačů se hodně dlouhou dobu jednalo o zvyšování výkonu pouze při zpracování jediného vlákna. Zde bylo nejjednodušší cestou dosahování co nejvyšších pracovních frekvencí. V současné době se ovšem toto odvětví stále více zaměřuje na paralelní zpracování vláken. Tato koncepce se poprvé začala hojně využívat na serverových stanicích, pro které byly vyráběny základní desky schopné pojmout více procesorů. Dnes již existují čipy, které v sobě integrují více fyzických jader a známé jsou také technologie, které dovolují přidělit nevyužité výpočetní prostředky samotného jádra některému dalšímu vláknu (HT)[4]. Tomuto modelu paralelního zpracování se začínají postupně přizpůsobovat i ostatní části počítače. Souvisí s tím například nová koncepce vnější sběrnice procesoru, nebo více-kanálový přístup do operační paměti.

U počítačů před třetí generací se také vyvíjel způsob konstrukce procesoru (použité součástky) a vzhledem k obrovským rozměrům tehdejších strojů, byla pochopitelně velká snaha o jejich zmenšení. Ta vyústila v 70. letech 20. století v použití nově objeveného integrovaného obvodu. Snahy o zmenšování můžeme pozorovat i dnes a to v neustálém zdokonalování výrobního procesu, což vede ke snižování nákladů na výrobu, nižší spotřebě a následně také menšímu zahřívání.

Posledním, snadno pozorovatelným trendem u osobních počítačů současnosti, je snaha přenést některé specializované výpočty (např. grafické, audio…) z CPU na jiné specializované procesory. Tak se dočkáváme grafických procesorů (GPU) s obrovským výpočetním potenciálem nebo také audio procesorů, jejichž výkon je větší než výkon samotného CPU před několika málo lety.

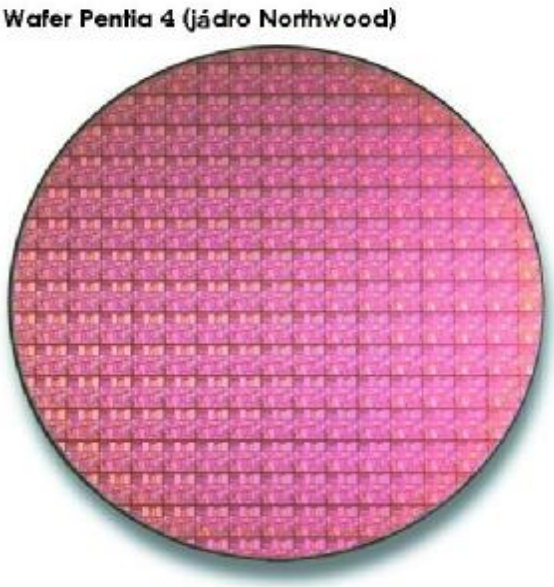
5.1 Výrobní procesy

Pozn.: kapitola vychází z [6]

V současnosti je procesor polovodičová součástka, založená na křemíkové destičce s několika dalšími příměsemi. V blízké minulosti byl nejčastější příměsí hliník, v současnosti se stále více používá měď, jelikož je lepší vodič. Sériové výrobě procesoru předchází spousta fází. Jako první je vytvořen softwarový model RTL (Register Transfer Logic), tento model je pro jednoduchost testován na nízkých frekvencích v řádu několika Hz. Současně s ním je testům podrobován i další softwarový model nazvaný Arcsim, tento již pracuje na frekvencích špičkového výkonu. Další testy jsou prováděny na hardwarovém modelu, ten často dosahuje velkých rozměrů (jedna místnost). Toto stádium vývoje procesoru se nazývá emulace. Pokaždé, když se během testů objeví chyba, dojde k celkové revizi modelu a jeho opravě. Opravený model poté dostává nové označení. Po sérii testů na modelu ve skutečné velikosti následuje poslední fáze, než je pro sériovou výrobu daného procesoru spuštěna celá výrobní linka. Touto fází je testování procesoru na spoustě běžně používaného hardwaru, ověřování kompatibility a opomenuty nejsou ani výkonnostní testy v podobě benchmarkových programů.

Po zahájení sériové výroby je každý vyrobený kus podroben sérii testů trvající pár desítek sekund. V prvních sériích se často objevují chyby a stejně tak ve všech dalších sériích se mohou vyskytnout vadné kusy. Tyto krátké testy mají prověřit funkčnost vyrobeného procesoru před tím, než je vypuštěn do prodeje. V roce 1994 se do prodeje dostaly procesory firmy Intel, které chybovaly ve velice speciálních výpočtech. Společnost Intel nejdříve nabízela bezplatnou náhradu pouze těm zákazníkům, kteří byli schopni prokázat, že dané výpočty skutečně potřebují. Až později ustoupila a bezplatnou náhradu poskytla všem poškozeným. Tato zkušenost z minulosti jen potvrzuje, jak je série testů před vypuštěním procesoru do prodeje důležitá a může výrobci ušetřit nemalé náklady, které by mohly vzniknout v případě, že by se snažil procesor vypustit na trh co nejdříve na úkor testování.

Na začátku sériové výroby procesoru stojí výroba waferu (viz. obrázek 5-1). Wafer je plátek polovodičového materiálu, v případě výroby procesorů je tímto materiálem křemík. Pro výrobu waferů má většina výrobců procesorů vlastní výrobní zařízení, jejichž označení je dáno velikostí waferů, které produkují. V současnosti jsou nejčastěji využívány wafery o průměru 300mm a firmy Intel, TSMC a Samsung pracují odděleně na vývoji 450mm waferů.



Obrázek 5-1: Wafer procesoru Intel Pentium 4 s jádrem Northwood

Následný výrobní proces čipu je založen na úpravách krystalu křemíku, který se cílenými zásahy ultrafialového záření stává na určitých místech vodivým a jinde nevodivým a posléze se na něj integrují tranzistory. Postup výroby je nazýván podle nejmenší součástky, kterou je možno daným procesem vyrobit. To znamená, že například procesor Intel 386 vyrobený výrobním procesem 1,5µm mohl obsahovat nejmenší prvky, které měřily právě 1,5µm. Cílem dnešních výrobců je dosáhnout co nejmenšího výrobního procesu. To s sebou přináší možnost integrovat více součástek na jeden čip, snižuje se tím spotřeba produktu a samozřejmě také výrobní náklady. Ovšem pro každý nový výrobní proces musí často výrobce postavit celou novou továrnu s výrobními linkami, zázemím pro testování atd. Postupné zmenšování výrobního procesu probíhá asi o 70% každé 2-3 roky. Pro výrobu současných procesorů osobních počítačů je nejčastěji využíván 45nm výrobní proces (objevují se ještě i starší modely na 65nm výrobním procesu) a předpokladem je, že do konce roku 2009 bude jak AMD, tak Intel schopen vyrábět své čipy 32nm výrobním procesem. [7,8]

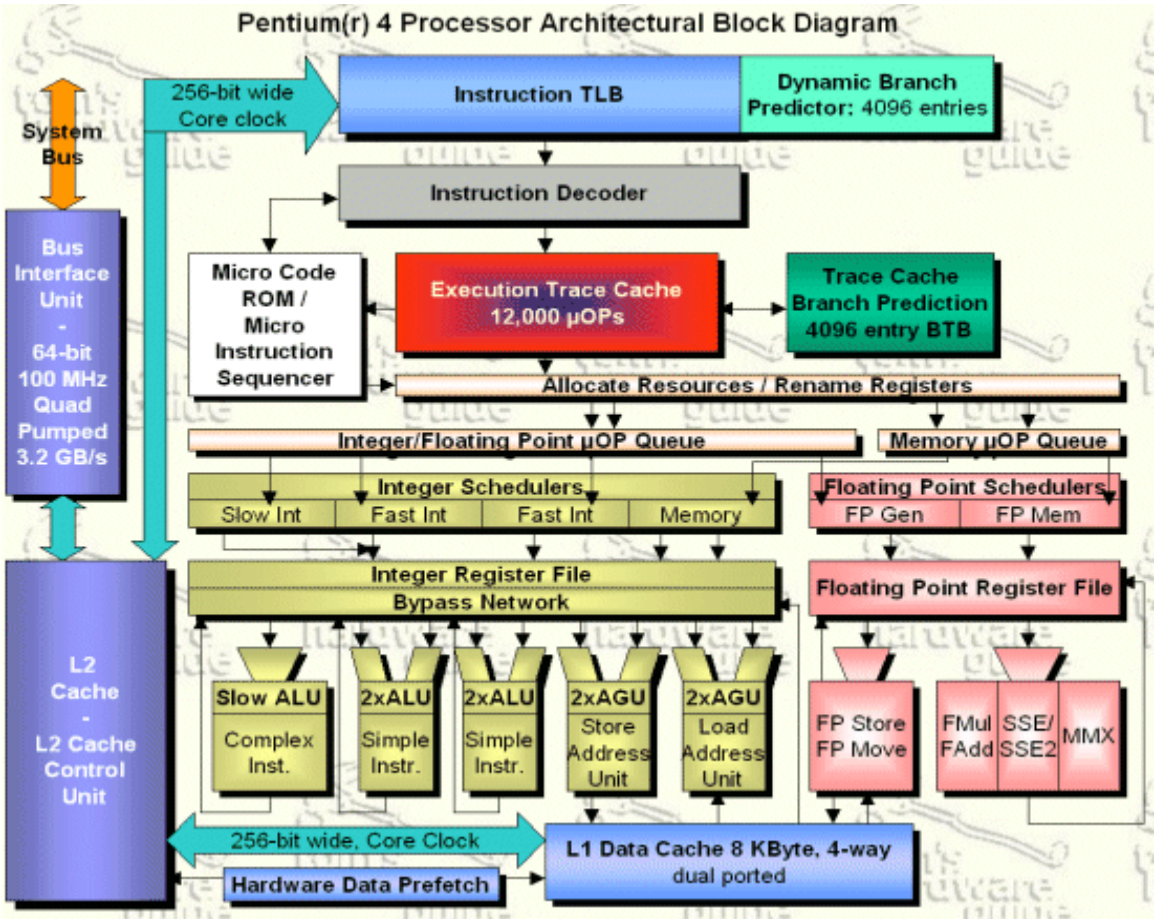
5.2 Architektury procesorů osobních počítačů v blízké minulosti a současnosti

Architekturou rozumíme vnitřní uspořádání jednotlivých částí procesoru a rozhraní pro komunikaci s okolím. Většina architektur prodělala řadu změn, za dobu co byla na trhu, ať už se jednalo pouze o novější revize jader, nebo dokonce přechod k úplně novému socketu. Jádra procesoru dostávají většinou velice zvláštní jména a těžko bychom v těchto označeních hledali nějaký systém. První procesory byly schopny zpracovávat v jeden okamžik jen jednu instrukci, později se objevily skalární architektury, které zavedly frontu instrukcí, do níž jsou instrukce načítány před samotným

provedením (ještě v době provádění některé předchozí instrukce) a jsou zde také částečně dekódovány. Dalším stupněm bylo zdvojení těchto front a také zdvojení samotných prováděcích jednotek, které nazýváme jako super-skalární architektura. Dále došlo k dělení jednotlivých sekcí, které mají na starost specifické zpracování instrukce na další podsekce, v nichž dochází k zřetězenému zpracování instrukcí. Pak také můžeme rozlišovat harvardskou architekturu (oddělená paměť pro data a program) nebo Von Neumanovu architekturu, data a program sdílí jednu paměť.

5.2.1 Architektury procesorů pro PC společnosti Intel

Blízká minulost procesorů firmy Intel by se dala datovat někdy od vypuštění prvního Pentia Pro v roce 1995. Jeho super-skalární architektura (P6) byla postupně vylepšována, Zvláštností u Pentia MMX mimo nové sady instrukcí byla zvláštní sběrnice pro komunikaci s L2 cache. U Pentií II již byla L2 cache integrována přímo do procesoru. Od Pentií II Intel odvodil „osekané“ procesory s označením Celeron. Podstatné rozdíly mimo ceny byly ve velikosti vyrovnávacích pamětí a často také v instrukční sadě. Například první Celeron (jádro Convington), který vzešel z Pentia II měl kmitočet 266MHz a neměl žádnou vyrovnávací paměť druhé úrovně, což mělo za následek znatelné snížení výpočetního výkonu v porovnání s plnokrevným Pentiem II, na druhé straně jej bylo možné velice snadno přetaktovat, třeba až k pracovní frekvenci o hodnotě 400MHz. Následující Pentia III dostala další sadu multimediálních instrukcí v podobě sady SSE. V roce 2000 společnost Intel vypustila procesory se superskalární architekturou s velmi dlouhými instrukčními pipeline, které měly dosahovat velmi vysokých pracovních frekvencí. Jednalo se o architekturu NetBurst Pentia 4. Blokový diagram takové architektury je na obrázku 5-2. Označení Pentium 4 se udrželo na trhu téměř 6 let. Procesory s tímto označením mají velký frekvenční rozptyl, od 1,4GHz v roce 2000, až po 3,8GHz v letech 2004-2005. Pro procesor Intel Pentium 4 byly za jeho života vyvinuty hned 3 druhy socketů, byly jimi: Socket 423, Socket 478 a ještě dodnes hojně používaný Socket 775. Společnost Intel v době vypuštění prvního Pentia 4 předpokládala, že s novými výrobními procesy, se tuto architekturu podaří škálovat až k 10GHz. Tato očekávání se bohužel nenaplnila. Pentia 4 jsou dodnes známa jako „horké procesory“, odpadní teplo, které tyto procesory vyzařovaly se stalo spolu s vysokou spotřebou hlavní bariérou v pokroku k vyšším frekvencím. Intel si hodně sliboval od přechodu na 90nm výrobní proces (Pentium 4 Prescott). Bohužel vyzařované teplo dosahovalo stále velmi vysokých hodnot. Před úplným odchodem architektury Netburst byl ještě posledním výkřikem do tmy „slepenec“ dvou jader Pentia 4 na jednom čipu, nazvaný Pentium D. Zde se také naposledy objevilo označení vlajkové lodi firmy Intel slovem „Pentium“.



Obr.: 5-2: Pokročilá architektura Procesoru Intel Pentium 4

Po celou tu dobu, kdy se Intel na poli stolních počítačů snažil dosahovat vysokých frekvencí, tak na mobilních platformách pokračoval s Pentiem M (architektura P6), za jehož předchůdce, by se dalo označit Pentium III. Po Pentiích M se sice i v noteboocích objevily na chvíli Pentia 4, ale v této sekci to byl již krok zcela nesprávným směrem a tak se záhy dostaly na trh první procesory Core Duo a Core Solo (jak z názvu vyplývá, dvoujádrová a jednojádrová varianta). Na těchto čipech byly dále založeny procesory Core 2, které se objevily jak ve dvoujádrové, i čtyřjádrové variantě. Byly vyráběny zpočátku 65nm výrobním procesem a od konce roku 2007 i 45nm výrobním procesem. Tyto procesory poskytovaly již v základu velice slušný výkon, kdy na o několik stovek MHz nižších taktech porážely ve výkonových benchmarcích i ty nejvýkonnější procesory založené na architektuře Netburst. Dále se vyznačovaly poměrně nízkou spotřebou, která s přechodem na dokonalejší výrobní proces ještě poklesla a tím pádem se snížilo i množství vyzářeného tepla. Za zmínku jistě stojí, že procesory Core 2 byly všeobecně velice vhodné pro přetaktování, některé lepší kusy se povedlo přetaktovat až o 60% za použití lepšího vzduchového chladiče. Tato architektura dominovala na trhu přes 2 roky. Nejvýkonnější modely doplňují postupně během této doby zajímavé low-endové kousky, ze kterých bychom rozhodně neměly opomenout Pentrium Dual-core. Jedná se o procesor téměř shodný s plnokrevnými Core 2 Duo, ovšem s nižším taktem FSB (zpravidla 800MHz) a také menší

L2 cache (obvykle poloviční velikosti než u plnokrevného Core 2 Duo). Zde musím podotknout, že název „Pentium Dual-core“ byl ze strany Intelu poměrně nerozvážným krokem. Spousta kupujících je totiž považovala za procesory Pentium D na překonané architektuře Netburst. Proto, i když by Pentium Dual-core mohl představovat ideální volbu pro domácí počítač v poměru cena/výkon, jeho prodej tomu zdaleka neodpovídá. Pentium Dual-core samozřejmě není tou rizí low-end variantou architektury Core, tou je totiž řada procesorů, která opět nese označení Celeron. Tyto procesory mají pouze 512kB L2 cache a to i ve dvoujádrovém provedení Celeron Dual-core. Z tohoto faktu je zřejmé, že Intel se hodlá jednoznačně ubírat dále cestou Multijádrových procesorů a to i u těch nejlevnějších modelů. Úspěšnou architekturu Core 2 nahradila s příchodem roku 2009 architektura Nehalem. Jejími zástupci jsou procesory Core i7 a ohlášené Core i5, které mají být vypuštěny někdy mezi červnem a zářím roku 2009. Architektura Nehalem je dalším přímým následníkem, ovšem přináší řadu změn. Jednou z nich je integrace paměťového řadiče do čipu procesoru, jedná se konkrétně o řadič DDR3 pamětí. Další změnou je použití nástupce sběrnice FSB, který se nazývá QuickPath Iterconnect [7]. Tato sběrnice zajišťuje každému jádru přístup k vstupně/výstupním prostředkům systému. Rychlost této sběrnice se liší dle modelu od 4,8GT/s do 6,4GT/s. Procesory Core i7 mají fyzicky 4 jádra, ovšem každé z nich navíc disponuje technologií Hyper Threading, takže ve výsledku operační systém přerozděluje procesy mezi 8 logických jader. Procesory Core i7 mají 3 úrovně vyrovnávací paměti. Z toho L1 a L2 samostatně pro každé ze 4 jader a L3 je sdílená.

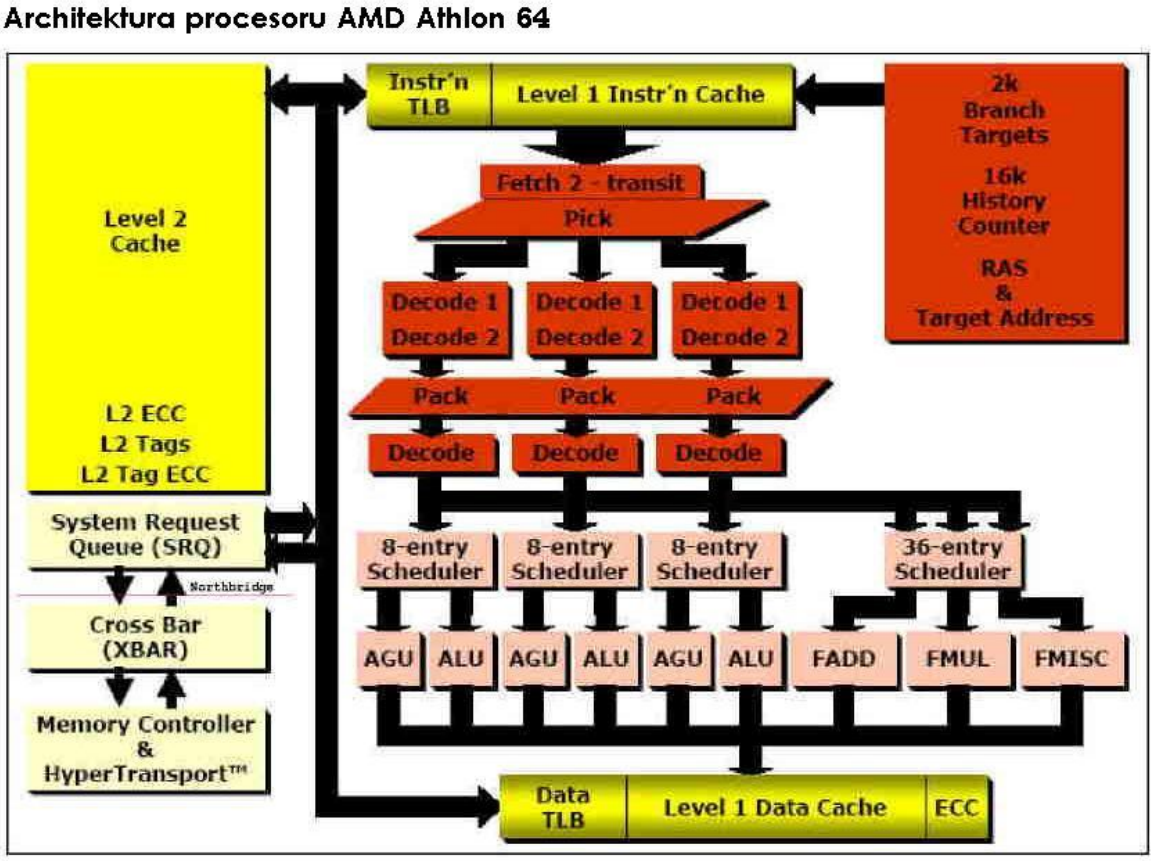
Nakonec ještě zmíníme řady Xenon a Itanium, což jsou procesory určeny k využití především v serverových stanicích, v jejich designu je kladen důraz na paralelní zpracování procesů, tomu odpovídají i, v porovnání s modely procesorů pro osobní počítače té doby, velké vyrovnávací paměti všech úrovní. První procesor pojmenovaný Xenon vyšel v roce 1998 a byl založen na Pentiu II.

5.2.2 Architektury procesorů pro PC společnosti AMD

Prvním procesorem vyvinutým od začátku do konce v laboratořích AMD byl projekt K5. Jednalo se o procesor standardu X86 a AMD jej navrhovalo zcela od základu. Jelikož chyběly zkušenosti s návrhem skalární architektury a celý projekt byl pro AMD prvním svého druhu, byly termíny dokončení odkládány. Procesor se nakonec dostal do prodeje v roce 1995. K5 byl postaven na RISCovém paralelním jádře, které tvořilo 5 celočíselných jednotek a jedna jednotka pro práci s plovoucí desetinnou čárkou. Procesor byl ale pomalejší než Pentium na shodné frekvenci, vyrovnával se spíše Pentiu Pro, proto nebyl příliš rozšířen.

Architektura K5 ale položila základ architektuře K6, vydané v roce 1997. Jeho jádro vyvinula společnost NexGen, kterou AMD koupila. K6 již na rozdíl od svého předchůdce disponoval MMX instrukcemi a jeho výpočetní výkon předčil procesory Pentium na shodném taktu. Další v pořadí byla architektura K6-2, tyto procesory AMD poprvé vybavilo vlastní instrukční sadou nazvanou 3Dnow!,

která slouží pro práci s audiem a videem. Následovaly ještě další změny v podobě K6-III a K6-III P, kde šlo především o vylepšení systému vyrovnávací paměti. V červnu 1999 byla vypuštěna architektura K7, která dostala označení Athlon. K7 se dočkal i nové patice, když byl Slot7 nahrazen SlotemA. Novinkou byl hlavně nový design jednotky pro zpracování instrukcí nazvané Super-Pipeline a superskalární jednotka pro operace v plovoucí desetinné čárce. První procesory byly vyrobeny 0,25 mikronovým procesem, ale od roku 1999 i 0,18 mikronovým procesem a dosahovaly frekvencí 500-600MHz. Díky novému výrobnímu procesu a tím pádem i nižší spotřebě dosáhly procesory Athlon jako první kmitočtu 1GHz v oblasti procesorů pro osobní počítače. V červnu 2000 byly řady K7 obohaceny o model Thunderbird, ten se vyskytoval jak ve verzi pro SlotA, tak pro SocketA. Byl dodáván na frekvencích 600-1400MHz. Oproti předchozímu Athlonu došlo opět ke změnám v systému vyrovnávacích pamětí. Projekt Thunderbird, byl největším úspěchem AMD za tehdejších 10 let. Po vzoru Intelu i AMD vyšlo vstříc poptávce po levnějších procesorech a postavilo na jádře plnokrevných Athlonů, levnější modely procesorů se zmenšenou L2 cache a sníženým kmitočtem FSB (200MHz efektivně oproti 266MHz efektivně u Athlonu), tyto procesory dostaly název Duron. Modely Thunderbird konkurovaly úspěšně Pentiím III a prvním Pentiím 4. Když ovšem Intel uvedl P4 na frekvenci 1,7GHz, bylo nutné vydat novou revizi zvanou Palomino, známou spíše jako Athlon XP. Nový redesign se ukázal jako velmi úspěšný, výpočetní výkon byl zhruba o 10% než u Thunderbirdu na shodné frekvenci a zároveň jeho spotřeba byla o 20% nižší. Došlo také k implementaci instrukcí sady SSE a stejně tak byla zachována sada 3Dnow! Jelikož širší veřejnost, byla zvyklá porovnávat rychlost procesoru dle vnitřního kmitočtu a často nezohledňovala architekturu, bylo velmi dobrým marketingovým tahem začít značit procesory i poměrným výkonem. To v praxi znamenalo, že za názvem procesoru byla ještě číslice se znaménkem „+“ na konci a ta představovala hodnotu frekvence, na jaké by musel pracovat model Thunderbird, aby dosáhl stejného výkonu. Tato filozofie se obzvlášť osvědčila, když se Intelu začalo dařit dosahovat s Pentii 4 vyšších frekvencí a zůstala tak vlastně, pro běžného uživatele, jediným srovnáním. Toto poměrné značení se udrželo v praxi i u dalších nástupců až do posledního Athlonu, které roku 2008 nahradila architektura Phenom (K10). V levnějším segmentu trhu byly čipy Duron nahrazeny procesory Sempron, které vychází opět z jádra procesoru Athlon XP (Thoroughbred/Thorton/Barton…), ale disponují menší L2 cache. Doba Athlonů XP byla pro společnost AMD velmi úspěšná, náskok před Intelem se povedlo ještě zvětšit, když AMD v roce 2003 bylo první, kdo uvedl na trh první nativní 64-bitový procesor, jednalo se o architekturu K8, která pak dostala označení Athlon 64. K8 přineslo také novinku v podobě integrovaného paměťového řadiče s tím je spojeno nahrazení stávajícího socketu A socketem 939. Athlon 64 byl několikrát revidován a na trhu se držel celých 5 let. Nástin jeho blokové architektury je na obrázku 5-3.



Obr.: 5-3: Pokročilá architektura procesoru AMD Athlon 64

V roce 2006 AMD vypustilo 2 jádrové Athlony označeny X2, které byly prvním nativním dvoujádrem a měly mnohem propracovanější interní design oproti Pentiu D. Ale s příchodem architektury Core (2) společnosti Intel končí nadvláda AMD na trhu, to byl impulz k rychlému dokončení architektury K10 (Phenom), aby zvýšilo svoji konkurenceschopnost. Bohužel, vypuštění Phenomu AMD skutečně uspěchalo a tak se dostaly ven první chybné série. Samozřejmostí je integrovaný řadič paměti, L3 cache společná pro všechna jádra integrovaná přímo na čipu a sběrnice Hyper transport, která zajišťuje komunikaci jednotlivých jader s I/O zařízeními. Mezi zvláštnosti architektury K10 je nekonvenční 3 jádrový model značený Phenom X3. Poslední vyvinutou architekturou je K10.5 v podání Phenomu II X4, které disponují L3 cache až 6MB a řadičem DDR3 pamětí.

AMD má také procesory určené pro použití především v serverových stanicích, jsou označovány Opteron, první byly založeny na architektuře K8 a velkou výhodou oproti konkuretním čipům Intelu byl nativní jak 32bit režim, tak 64bit režim, jelikož Xenon 32bit režim pouze emuloval a docházelo k degradaci výkonu.

5.3 Zvyšování výkonu vs. snižování spotřeby

Existuje spousta požadavků na centrální výpočetní jednotky, ať už se jedná o přesnost, různé rozšířené speciální funkce nebo spolehlivost, ale po celou dobu historického vývoje byla hlavním požadavkem zejména rychlost. Rychlost byla parametrem, kterým se poměřovaly nejen osobní počítače, ale i jiná zařízení, která byla založena na modelu procesoru jakožto řídícího a hlavního výpočetního prvku. V posledních letech se ovšem neustále zvyšuje poptávka po mobilitě těchto zařízení. Je těžké si představit digitální fotoaparát, který požaduje napájení ze sítě. Tento moderní požadavek přinutil vývojáře rozdělit své produkty na ty, které jsou zaměřeny na vysoký výkon, bez ohledu na spotřebu (jednoduchým příkladem může být grafické jádro AMD R600), a na produkty, které se snaží při zachování dostatečného výkonu spořit energii, aby mohly být napájeny z přenosných elektrických zdrojů. To vedlo k vývoji buď zcela odlišných nebo alespoň upravených procesorových řad pro notebooky, takové „úsporné centrální jednotky jsou vyvíjeny i pro mobilní telefony, různé MP3 přehrávače, videokamery a jiná přenosná zařízení, u nichž je esenciální doba po kterou jsou schopny pracovat s omezeným množstvím energie.

Mimo speciální návrhy celých úsporných architektur (Intel Atom) se objevily i technologie, které jsou jakýmsi kompromisem mezi výkonem a spotřebou. Většina takových technologií je založena na výkonných čipech, jejichž řízení je ovlivňováno sledováním potřeby výkonu. Ve chvíli, kdy jednotka nemusí podávat vysoký výkon, je snížena její pracovní frekvence (např.: pomocí multiplikátoru) a tím je možno snížit i napájecí napětí. Ve chvíli, kdy je výkon opět potřeba, nastaví se nominální hodnoty. Orientační hodnoty maximální spotřeby jednotlivých zástupců procesorů jsou uvedeny v tabulkách 5-1 (CPU pro stolní počítače) a 5-2 (CPU pro mobilní počítače) [9].

CPU	*Pracovní frekvence (MHz)*	*Spotřeba (W)*
Intel Pentium	100	10,1
Intel Pentium II	266	38,6
Intel Pentium III (EB)	600	15,8
AMD Athlon XP 2000+	1667	70,5
AMD Athlon 64 3200+	2000	67,0
Intel Pentium 4 560J	3600	115,0
Intel Pentium D940	3200	130,0
AMD Athlon 64 X2 3800+	2000	89,0
Intel Core 2 Duo (Wolfdale)	3000	65,0
AMD Phenom 9600 X4	2300	95,0
Intel Core 2 Quad (Q6600)	2400	105,0

Tabulka 5-1: Přehled spotřeby vybraných CPU pro stolní PC

CPU	*Pracovní frekvence (MHz)*	*Spotřeba (W)*
Intel Pentium M	1800	27,0
AMD Turion 64	1800	35,0
Intel Celeron M	1800	30,0
AMD Turion X2	1800	31,0
Intel Core DuoT2300	1830	31,0

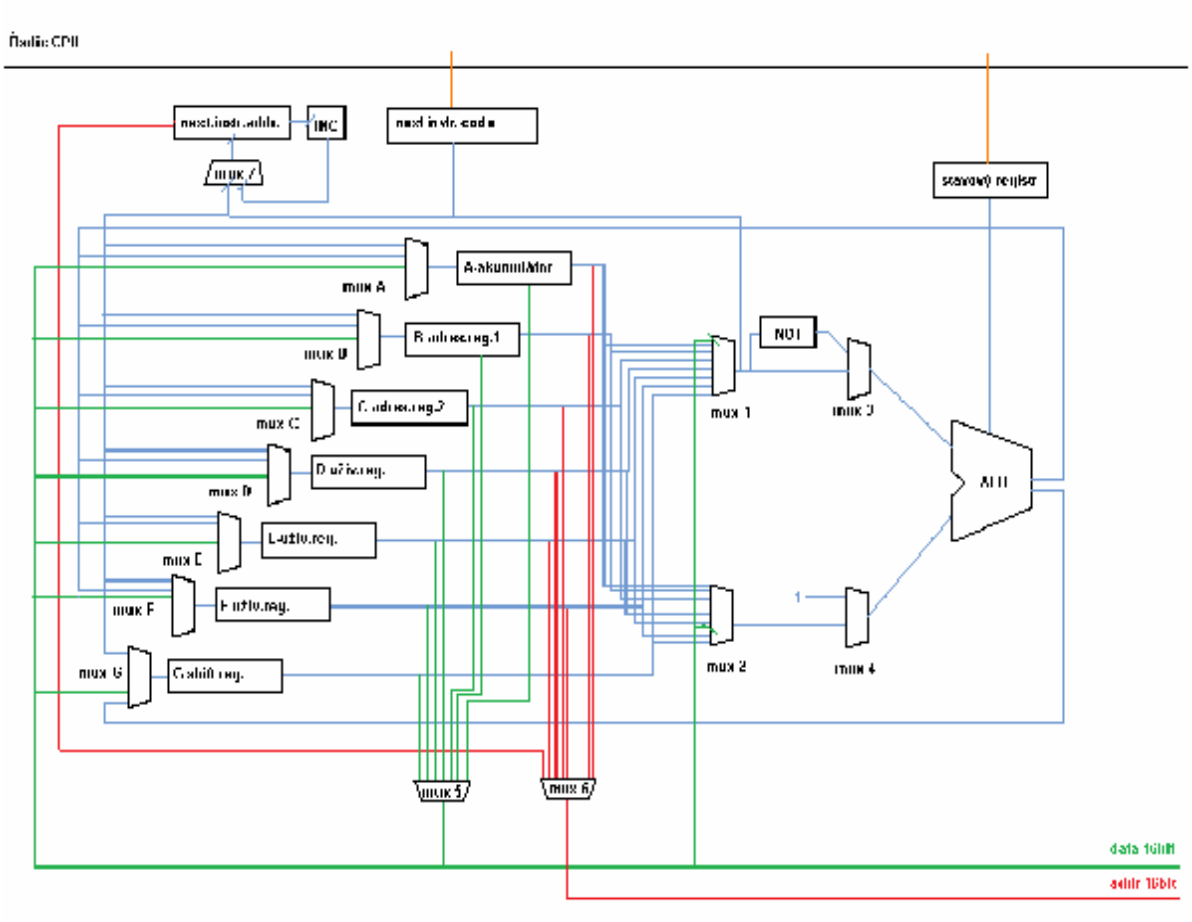
Tabulka 5-2: Přehled spotřeby vybraných CPU pro mobilní PC

6 Návrh vlastní architektury jednoduchého procesoru

Celý návrh je zaměřen především na jednoduchost a funkčnost. Procesor je navržen pro práci se sadou těch nejzákladnějších a nejdůležitějších instrukcí. Dále by měl obsahovat dostatečný počet registrů pro snadné vykonání jednodušší rutiny (programu). Součástí návrhu je také základní operační paměť RAM. Tato paměť je společná pro data i instrukce (Von Neumannova architektura). Komunikaci procesoru s pamětí zajišťuje sběrnice skládající se ze 3 částí (datová, adresová, řídící). Všechny části procesoru jsou řízeny mikroprogramovým řadičem (typické pro CISC architekturu), tento řadič umožňuje snadno provádět úpravy v instrukční sadě.

6.1 Základní charakteristiky vlastního návrhu

Jedná se o 16bitový procesor, jehož architektura (viz. obrázek 6-1) je navržena s ohledem na vykonávání základních aritmeticko logických operací. Předpokládaná instrukční sada, aplikovatelná na tuto architekturu zahrnuje celočíselné aritmetické operace, dále jednoduché logické operace, bitové operace jako rotace a posuny, operace práce s pamětí a registry a samozřejmě v neposlední řadě instrukce podmíněných i nepodmíněných skoků. Procesor řídí mikroprogramový řadič. Uživatel má pro práci s daty k dispozici 7 uživatelských 16bitových registrů - akumulátor, 2 adresové registry, 3 univerzální registry a záchytný registr pro případ přetečení výsledku ALU. Procesor umožňuje pracovat i s operandy v paměti pomocí adresových registrů. Výsledky operací nad dvěma operandy jsou zpravidla ukládány do akumulátoru. Dalšími registry jsou stavový registr a registr adresy následující instrukce a kódu zpracovávané instrukce.



Obr.: 6-1: Vlastní architektura procesoru

6.2 Sada registrů

Procesor disponuje celkem devíti 16bitovými a jedním osmibitovým registrem. Celkem z toho 7 16bitových registrů je uživatelských. Tento počet by měl být plně postačující k provádění základních operací i jednoduššího programu. Poskytují místo pro případné dočasné uložení hodnot, s kterými se nebude v dalším kroku pracovat, ale o pár kroků později budou znovu použity. 8bitový registr je registrem stavovým (na obrázku 6-2), ve kterém jsou uloženy speciální příznaky provedených operací. Například,zda došlo k přetečení výsledku ALU operace, nebo zda byl výsledek nulový a také se zde ukládá výsledek porovnávací operace. Tento registr není adresován, jelikož uživatel do něj nemá přímý přístup, příznaky lze testovat pouze pomocí speciálních skokových instrukcí. Registr je automaticky nulován na začátku operace, která mění dané příznaky. Jedná se o tyto instrukce: RESET, ADD, SUB, MUL, INC, DEC, INV, AND, OR, XOR, NEG, SHFR, SHFL, CMP. Dále je registr resetován na konci každé operace skoku.

Příznaky na jednotlivých bitech:

Obrázek 6-2: Stavový registr

Zero flag	Overflow flag					Příznak rovnosti1	Příznak rovnosti2
0	1	2	3	4	5	6	7

- Zero flag – výsledek operace v ALU = 0
- Overflow flag – výsledek přesáhl 16bit ve dvojkovém doplňkovém kódu
- Příznak rovnosti – po provedení instrukce CMP nad oeprandy op1 a op2 nastaví bity následovně

  op1 = op2 => bit6 = 1, bit7 = 1

  op1 = op2 => bit6 = 1, bit7 = 0

  op1 = op2 => bit6 = 0, bit7 = 1

Dalším registrem, ke kterému uživatel nemá přímý přístup je registr v němž je uložena adresa následující instrukce a registr v němž je uložen kód instrukce. Plnění těchto registrů správnou hodnotou je čistě v rukou řadiče a jsou zajištěny jako součást mikroprogramů. Pokud není prováděna instrukce skoku, je adresa jednoduše inkrementována, pokud je kladně vyhodnocena operace skoku, je registr naplněn adresou, která je uložena v registru, jehož adresa je dána jako operand instrukce skoku.

Další 16bitové registry jsou uživatelské a jsou tedy adresovány pro možný přístup.

- Akumulátor – uživatelský registr A (00001)
  Do tohoto registru jsou uloženy výsledky všech operací nad dvěma operandy, které vrací přímo výsledek .
  Obsah registru je přepsán těmito operacemi: RESET, ADD, SUB, MUL, INV, AND, OR, XOR, NEG

- Adresový registr (1) – uživatelský registr B (10001)
  Pokud je adresa tohoto registru dána jako operand instrukce, procesor vystaví hodnotu uloženou v tomto registru na adresovou sběrnici a přečte z paměti RAM z dané adresy data na datovou sběrnici a ta budou použita jako operand pro operaci v ALU.
  Obsah registru je přepsán operací: RESET
- Adresový registr (2) – uživatelský registr C (10010)
  Pokud je adresa tohoto registru dána jako operand instrukce, procesor vystaví hodnotu uloženou v tomto registru na adresovou sběrnici a přečte z paměti RAM

z dané adresy data na datovou sběrnici a ta budou použita jako operand pro operaci v ALU.
Obsah registru je přepsán operací: RESET

- Uživatelský registr D (00011)
  Primární určení tohoto registru je uchování hodnoty jako operandu pro operaci ALU jednotky. Dále může registr sloužit pro odložení nějaké hodnoty, která bude opět záhy potřeba v jiném registru, popřípadě jako čítač.
  Obsah registru je přepsán operací: RESET
- Uživatelský registr E (00100)
  Určení tohoto registru je shodné s registrem D.
  Jeho obsah je přepsán operací: RESET
- Uživatelský registr F (00101)
  Určení tohoto registru je shodné s registrem D, navíc slouží pro dočasné uchování hodnoty při operaci MSTORE.
  Obsah registru je přepsán operací: MSTORE, RESET
- Shift/Overflow registr (G) (00111)
  Do tohoto registru jsou uloženy bity 17 a vyšší, které mohou přetéct při operaci násobení. Zároveň je možné tento registr použít ke stejným účelům jako registr D.
  Obsah registru je přepsán operacemi: RESET, MUL

6.3 ALU jednotka

ALU jednotka vykonává celkem 14 matematických nebo logických operací. Prováděné matematické operace jsou základní celočíselné operace jako sčítání, odčítání, inkrementace, dekrementace a násobení celých čísel. Operace dělení, jakožto operace primárně určená pro FPU implementována není. Celočíselné dělení lze ovšem velmi snadno nahradit programovou smyčkou. Například s použitím čítače od nuly a při každém průchodu odečteme dělitele od dělence, smyčku vykonáváme tak dlouho, dokud je výsledek po odečtení větší než dělitel, až tato podmínka nebude splněna, tak hodnota v čítači je výsledkem celočíselného dělení. Stejně tak hodnota získaná po posledním provedeném odečtení je výsledkem operace modulo, proto není implementována ani operace modulo přímo jako operace ALU jednotky. Z logických operací jsou implementovány AND, OR, XOR. Negace je provedena pomocí předřazeného členu a ALU jednotkou již prochází nezměněný výsledek přímo do akumulátoru. Samozřejmostí jsou operace bitových rotací a posunů, které jsou ve VHDL kódu implementovány jako vlastní funkce nad logickým vektorem hodnot. Poslední operací je operace porovnání, která nevrací na výstup žádný výsledek, jen nastaví příslušné příznaky ve stavovém registru.

6.4 Jednoduchá RAM

Synchronní operační paměť je implementována jako jednoduché pole logických vektorů, kde adresa vystavená na adresovou sběrnici představuje přímo index hodnoty v tomto poli. Paměť je propojena s procesorem 34bitovou nesdílenou sběrnicí. 16 vodičů slouží jako 16bitová datová sběrnice, dalších 16 vodičů jako 16bitová adresová sběrnice a 2 vodiče přenáší řídící signály. Jedná se o signály READ a WRITE, vždy je provedena operace podle toho, který signál je zrovna nastaven na 1. Synchronizaci zajišťuje externě generovaný hodinový signál shodný pro všechny moduly procesoru. Ve chvíli, kdy paměť nemá provádět žádnou operaci, jsou oba signály nastaveny na 0. Synchronizaci zajišťuje externě generovaný hodinový signál shodný pro všechny moduly procesoru. 16bitová adresová sběrnice umožňuje adresovat až 65532 paměťových buněk, což je ovšem zbytečně mnoho, předpokládám, že pro funkčnost takto jednoduchého procesoru, by mělo postačit 1000buňek RAM, ale celková šířka 16bit zůstane zachována, kvůli jednodušší práci s 16bit univerzálními registry. Řadič paměti RAM je pro jednoduchost návrhu včleněn do řadiče procesoru. Paměť navíc disponuje naprosto identickou sběrnicí, pomocí níž je možné ji naplnit při spuštění testu patřičným vzorkem dat a sledem instrukcí z modulu testbench.

6.5 Mikroprogramový řadič

Mikroprogramový řadič se v návrhu objevuje jako jisté rozšíření, jeho výhodou oproti řadiči, který by byl realizován jako konečný stavový automat je snadné provádění úprav v instrukční sadě. Mikroprogramový řadič se skládá z logiky pro dekódování instrukcí jehož funkcí je také spuštění patřičného mikroprogramu, pomocí kterého má být daná instrukce vykonána. Samotné mikroprogramy jsou uloženy ve zvláštní paměti ROM, která pomocí 10bitové sběrnice (nepředpokládám nutnost indexovat více místa pro mikroprogramy tohoto procesoru) přijímá adresu mikroinstrukce, kterou má navrátit. Samotná mikroinstrukce se skládá ze signálů, které ovládají všechny řídící prvky v procesoru (horizontální kódování) a posledního příznakového bitu, který značí, že daná mikroinstrukce byla poslední v mikroprogramu (hodnota 0) nebo bude následovat další mikroinstrukce (hodnota 1) . Horizontální kódování jsem zvolil především proto, že architektura procesoru není natolik složitá, aby se mikroinstrukce stala neúnosně dlouhou a psaní mikroprogramu v horizontálním kódu je jednodušší a přehlednější. Pro opravdu každou instrukci, i podle toho, jaký registr je použit pro čtení operandu, musí v takovéto implementaci existovat přesný speciální mikroprogram. I přes to, že mnohem výhodnější by bylo použít nastavování datových cest zvláštní logikou za použití adres operandů, jsem tento postup nezvolil. Důvodem je, že řízení takto jednoduchého procesoru spočívá především v ovládání datových cest a jen malá část se stará o ovládání ALU, program counteru atd. a řadič by tak dostal z velké části podobu konečného stavového automatu, což odporuje zadání. Ovšem rozhodně bych tento způsob řešení doporučil jako

další rozšíření návrhu. Vzhledem k této skutečnosti nejsou implementovány operace nad všemi registry, ale jejich implementace (především operací aritmeticko logických) je omezena nad uživatelské registry D a E. Implementace nad ostatní registry by spočívala v triviálních úpravách již hotových mikroprogramů, což vzhledem k samotné povaze mikroprogramového řadiče (snadná úprava instrukční sady změnou mikroprogramů v paměti ROM) není problém, ale mnohem přínosnější by bylo ovládání datových cest realizovat pomocí pevné logiky, jak bylo zmíněno výše.

6.6 Formát instrukce

Celá instrukce, jelikož se jedná o 16bitový procesor se skládá z 16ti bitů. Těchto 16 bitů je rozděleno na tři části (jak naznačuje obrázek 6-3), z nichž každá zvlášť nese určitou informaci. První informace je uložena na prvních 6ti bitech (zleva) a je to informace popisující operační kód, čili určuje, jaká operace bude prováděna. Druhá část a třetí část mají rozsah 5bitů každá a jedná se o adresu registru, s kterým bude operováno.

Instrukce v tomto návrhu za žádných okolností nepřenáší konstantní hodnotu a vždy nese pouze odkaz v podobě adresy na danou hodnotu. Začleněním mikroprogramového řadiče, v podobě, která byla popsána v kapitole 6.5 je instrukce vyhodnocována jako jeden celek a není dekódována po částech, ale architektura s touto koncepcí dekódování instrukce na jednotlivé sekce a jejich zvláštní zpracování počítá jako možné rozšíření, proto jsou rozsahy jednotlivých částí instrukcí pevně dány, aby se usnadnilo případné dekódování instrukce.

6 bitů – operační kód	5 bitů - adresa registru prvního operandu	5 bitů – adresa registru druhého operandu

Obr.: 6-3: Formát instrukce

6.7 Instrukční sada, pro kterou byla architektura navržena

SYSTÉMOVÉ INSTRUKCE (2)

0b00001 HALT	OP1:-	OP2:-	
0b00010 NOP	*OP1:-*	*OP2:-*	

OPERACE PŘESOUVÁNÍ DAT (4)

0b000011 CLOAD	OP1:cíl reg (A,B,C,D…)	OP2:-	konstanta na další pozici v paměti
0b000100 MLOAD	OP1:cíl reg (A,B,C,D…)	OP2:zdroj	
0b000101 MOV	OP1:cíl reg (A,B,C,D…)	OP2:zdroj reg (A,B,C,D…)	
0b000110 MSTORE	OP1:zdroj reg (A,B,C,D…)	OP2: cíl	

ARITMETICKÉ INSTRUKCE (5)

0b000111 ADD	OP1:reg	OP2:reg	výsledek do: A
0b001000 SUB	*OP1:reg*	*OP2:reg*	*výsledek do: A*
0b001001 MUL	*OP1:reg*	*OP2:reg*	*výsledek do: A*
0b001100 INC	OP1:reg	OP2:-	výsledek do: původní registr
0b001101 DEC	OP1:reg	OP2:-	výsledek do: původní registr

LOGICKÉ INSTRUKCE (4)

0b001110 AND	OP1:reg	OP2:reg	výsledek do: A
0b001111 OR	OP1:reg	OP2:reg	výsledek do: A
0b010000 XOR	OP1:reg	OP2:reg	výsledek do: A
0b010001 NEG	OP1:reg	OP2:-	výsledek do: A

INSTRUKCE BITOVÝCH OPERACÍ (4)

0b010010 ROTR	*OP1:zdroj*	*OP2:*	*výsledek do: původní registr*
0b010011 ROTL	*OP1:zdroj*	*OP2:*	*výsledek do: původní registr*
0b010110 SHFR	*OP1:zdroj*	*OP2: o kolik bit rot*	*výsledek do: pův.reg.*
0b010111 SHFL	*OP1:zdroj*	*OP2: o kolik bit rot*	*výsledek do: pův.reg.*

INSTRUKCE POROVNÁNÍ A SKOKU (9)

0b011111 CMP	OP1: reg	OP2: reg	příznak ve stavovem registru
0b011000 BRANCH	OP1:cíl skoku	OP2:-	(změní PC na OP1)
0b011001 BRZERO	OP1:cíl skoku	OP2:	(změní PC na OP1, statusreg 0 = 1)
0b011100 BROVERFL	OP1:cíl skoku	OP2:-	(změní PC na OP1, statusreg 1 = 1)
0b100000 JE	OP1: cíl skoku	OP2:-	
0b100001 JG	OP1: cíl skoku	OP2:-	
0b100011 JGE	OP1: cíl skoku	OP2:-	
0b100100 JL	OP1: cíl skoku	OP2:-	
0b101100 JLE	OP1: cíl skoku	OP2:-	

6.8 Adresy uživatelských registrů

Za pomoci přehledu instrukční sady (kapitola 6.7) a adres jednotlivých uživatelských registrů (tabulka 6-1) je možné celkem jednoduše tvořit rutiny ve strojovém kódu.

Registr	A	B	C	D	E	F	G
Adresa	00001	10001	10010	00011	00100	00101	00111

Tabulka 6-1: Adresování uživatelských registrů

7 Implementace

Návrh je implementován v jazyce VHDL, jako vývojové prostředí pro implementaci i simulaci byl použit nástroj ModelSim XE III/Starter 6.2g. Pro syntézu návrhu byl použit nástroj Xilinx ISE 9.2i.

7.1 Popis implementace

Celá implementace je rozdělena do větších funkčních bloků a následně strukturálně propojena. Nakonec je připojen testovací modul, který zavede testovací data a generuje vnější řídící signály.

7.1.1 Blok CPU (cpu.vhd)

V tomto funkčním bloku jsou implementovány registry, prvky které řídí tok dat (především multiplexory) a samotná ALU jednotka. Každý registr je implementován jako proces, citlivý na signály RESET (nulování) a CLK (hodiny). S vzestupnou hranou hodinového signálu jsou registry, pokud jsou nastaveny pro přijetí informace (signál in), schopny přijmout data z patřičného multiplexoru. Multiplexory jsou implementovány jako dalaflow prvky, které od řadiče obdrží kód „cesty“, odkud mají přijmout data, tato data uchovají ve vlastním logickém vektoru, odkud si je registr nastavený pro zápis nové hodnoty vyzvedne. Obdobným způsobem jsou navrženy všechny mutiplexory i registry. Samostatná jednotka ALU je navržena na co nejvyšší úrovni abstrakce, mimo operace bitových rotací a posunů, které jsou implementovány jako zvláštní funkce nad logickými vektory. Výsledky operací nad dvěma operandy jsou uloženy do akumulátoru (registr A). Operace nad jedním registrem, jako je inkrementace apod. vrací výsledek do původního registru. Možnost přetečení při operaci násobení 2 16bitových vektorů je řešena pomocí 32bitového bufferu, z nějž je výsledek rozdělen na 16bitový výsledek a 16 bitů považovaných za přetečené, ty jsou uloženy v overflow registru G.Vstupními porty bloku CPU jsou jednotlivé řídící signály pro všechny prvky, hodinový, resetovaní a aktivační signál a také sběrnice. Jelikož datová sběrnice je obousměrná (propojení modulu SRAM a CPU) je pro validnost syntézy implementována zvlášť část pro příchozí a odchozí tok dat. Přičemž při nevyužití sběrnice pro odchozí data je sběrnice uvedena do stavu vysoké impedance.

7.1.2 Blok SRAM (sram.vhd)

Jedná se o synchronní operační paměť společnou jak pro data, tak program (Von Neumannova architektura) implementovanou pomocí pole 16bitových vektorů. Díky 16 bitové adresové sběrnici, je možno adresovat paměťové pole až do hodnoty 65536. Paměť je při implementaci omezena na 1000 16bitových vektorů. Důvodem je, že tato hodnota je bohatě postačující jak pro uložení dat, tak jednoduchého programu, navíc syntéza (na C2Quad 6600@3,8GHz) implementace s větší operační pamětí trvá neúnosně dlouho (cca 30min). Paměť je synchronizována externím hodinovým signálem CLK. A s procesorem je propojena pomocí adresové, paměťové a řídící částí sběrnice. Navíc pro zavedení dat pomocí testbenche je připojena ještě na další datovou a adresovou sběrnici, na jejímž druhém konci je modul testbench, který pomocí těchto sběrnic naplní pamět před aktivací procesoru testovacími daty.

7.1.3 Blok řadiče (controller.vhd)

Řadič procesoru je ve výsledné implementaci řešen jako plně mikroprogramový, to znamená, že žádná část není řešena konečným stavovým automatem. Více k této koncepci v kapitole 10 - Závěr a možná rozšíření. Řadič je řízen signály, které generuje modul testbench. Jedná se o hodinový signál CLK a aktivační signál CE, který slouží k aktivaci nebo deaktivaci (podle úrovně signálu) celého procesoru. Zároveň řadič disponuje vlastním interním signálem halt, který je řízen programově a je tak pomocí něj například zastaveno vykonávání programu. Jediná inicializace procesoru, na úrovni 1 signálu CE , není řešena mikroprogramově. Jako reakce na horní hladinu inicializačního signálu CE je spuštěna rutina, která má za úkol načíst instrukci, na kterou ukazuje hodnota v registru program counter (next instr. addr.). Přičemž se počítá s tím, že před prvotním spuštěním se procesor resetuje vnějším signálem RESET (v tomto případě generovaným modulem testbench), což vede k tomu, že v registru program counter je hodnota ‚0', která zajistí čtení z nulté adresy operační paměti, kde je očekávána první instrukce. V případě, že se nejedná o prvotní inicializaci, ale procesor byl například odstaven nulovou úrovní signálu CE a následně opět aktivován změnou signálu CE na ‚1', tak v program counteru zůstane adresa, která, pokud byl předchozí mikroprogram dokončen, ukazuje na následující instrukci programu v paměti. Na základě načtené instrukce, z registru v modulu cpu, která má být provedena je nastavena adresa do paměti ROM s mikroprogramy. Mikroprogram je následně sekvenčně prováděn až do chvíle, kdy je indikována poslední mikroinstrukce mikroprogramu (poslední bit mikroinstrukce je 0). Ve chvíli, kdy skončí provádění Mikroprogramu, je už v instrukčním registru připravena další instrukce programu. Každý mikroprogram na závěr svého provádění načte z paměti podle hodnoty v program cntr. registru následující instrukci do instrukčního registru a inkrementuje program counter. A může začít provádění nové instrukce. Načítané mikroinstrukce z paměti rom jsou kódovány horizontálně, to znamená, že každý řídící signál má v mikroinstrukci svou hodnotu, která jej přímo řídí.

7.1.3.1 Mikroprogramové ovládání řídících prvků procesoru

V tabulkách 7-1 až 7-9 je znázorněna konfigurace jednotlivých multiplexorů z obrázku 6-1, které jsou pomocí přiřazených kódů ovládány mikroprogramovým řadičem.

multiplexor (A-F) ovládá vstupní cesty do registrů A - F		kód
Vstupy:	výstup ALU	01
	výstup Mux. 1 (především pro operaci MOV z jednoho registru do jiného)	10
	datová sběrnice	11
	zavřeno	00
Výstup:	registr (A-F)	

Tabulka 7-1: Řízení multiplexorů uživatelských registrů A-F

G-multiplexor ovládá vstupní cesty do registru G		kód
Vstupy:	výstup z ALU při přetečení (horních 16bit)	01
	výstup Mux. 1 (především pro operaci MOV z jednoho registru do jiného)	10
	datová sběrnice	11
	zavřeno	00
Výstup:	registr G	

Tabulka 7-2: Řízení multiplexoru registru G

Mux 1		kód
Vstupy:	registr A	001
	registr B	010
	registr C	011
	registr D	100
	registr E	101
	registr F	110
	registr G	111
	datová sběrnice	000
Výstupy:	mux A-G, negátor, mux 3	

Tabulka 7-3: Řízení multiplexoru 1

Mux 2		kód
Vstupy:	registr A	001
	registr B	010
	registr C	011
	registr D	100
	registr E	101
	registr F	110
	registr G	111
	datová sběrnice	000
Výstupy:	mux 4	

Tabulka 7-4: Řízení multiplexoru 2

Mux 3		kód
Vstupy:	negátor	0
	mux 1	1
Výstup:	ALU	

Tabulka 7-5: Řízení multiplexoru 3

Mux 4		kód
Vstupy:	mux2	0
	„jednička“	1
Výstup:	ALU	

Tabulka 7-6: Řízení multiplexoru 4

Mux 5		kód
Vstupy:	registr A	001
	registr B	010
	registr C	011
	registr D	100
	registr E	101
	registr F	110
	registr G	111
Výstupy:	datová sběrnice	

Tabulka 7-7: Řízení multiplexoru 5

Mux 6		kód
Vstupy:	registr A	001
	registr B	010
	registr C	011
	registr D	100
	registr E	101
	registr F	110
	Registr adresy následující instrukce	111
	closed	000
Výstupy:	Adresová sběrnice	

Tabulka 7-8: Řízení multiplexoru 6

Mux 7		kód
Vstupy:	mux1(skok)	0
	inkrementační jednotka	1
Výstup:	registr pro adresu příští instrukce	

Tabulka 7-9: Řízení multiplexoru 7

7.1.3.2 Mikroprogramové kódy operací ALU

V tabulce 7-10 jsou jednotlivým instrukcím, vykonávaným jednotkou ALU přiřazený operační kódy, pomocí nich je jednotka ALU ovládána mikroprogramovým řadičem.

Operace	ADD	SUB	MUL	INC	DEC	AND	OR	XOR	NOP	ROR	ROL	SHR	SHL	CMP
kód	1010	0001	0010	0101	0110	0111	1000	1001	0000	1011	1100	1101	1110	1111

Tabulka 7-10: Kód operace ALU jednotky

7.1.4 Paměť ROM s mikroprogramy (controller_rom.vhd)

Všechny mikroprogramy jsou uloženy jako sekvence po sobě jdoucích mikroinstrukcí v paměti rom na po sobě následujících adresách. Podle instrukce je tedy získána pouze adresa první mikroinstrukce a další provádění mikroprogramu je již prováděno sekvenčně. Ve výsledné implementaci nejsou přítomny mikroprogramy pro provádění instrukcí nad všemi registry. Pro názornou ukázku funkčnosti návrhu jsou sice implementovány všechny uvažované instrukce, ovšem pouze nad specifickými registry (viz tabulka 7-11). Postup při rozšíření provádění instrukcí nad dalšími registry je podrobně diskutován v kapitole rozšíření. Modul paměti ROM řadiče, je řízen signály hodin, který je generován modulem testbench a dále signálem READMROM, který generuje samotný řadič v případě, že požaduje zaslání mikroinstrukce. Modul je propojen s řadičem pomocí adresové sběrnice, jíž se přenáší adresa mikroinstrukce, která má být načtena. Načtená mikroinstrukce je poslána řadiči přes mikroinstrukční sběrnici a ten ji poté dekóduje a nastaví příslušné signály řídící modulu cpu.

Výčet implementovaných mikroprogramově prováděných instrukcí (jak vyplývá z kapitoly 7.1.4)

Instrukce	Op1	Op2	popis
CLOAD	D, E	-	Načte konstantu na další pozici v paměti do reg. D nebo E
MLOAD	F	B	Načte hodnotu v paměti na adrese určené registrem B do reg. F
MSTORE	F	B	Uloží hodnotu registru F do paměti na adresu určenou reg B
MOV	A	B	Zkopíruje hodnotu registru A do registru B
ADD	D	E	Sečte hodnoty v registrech D a E, výsledek uloží do A
SUB	E	D	Odečte hodnotu v registru D od hodnoty v reg. E, výsledek do A
MUL	D	E	Vynásobí hodnoty v reg D a E, výsledek uloží do A
INC	E	-	Inkrementuje hodnotu v registru E
DEC	A	-	Dekrementuje hodnotu v registru A
AND	E	D	Provede logické násobení s registry E a D, výsledek do A
OR	E	D	Provede logický součet hodnot registrů E a D, výsledek do A
XOR	E	D	Exklusive OR hodnot v registrech E a D, výsledek do A
NEG	A	-	Provede negaci hodnoty v registru A
SHR	E	D	Bitový posun vpravo hodnoty v registru E o hodnotu registru D => A
SHL	E	D	Bitový posun vlevo hodnoty v registru E o hodnotu registru D => A
ROR	E	D	Bitová rotace vpravo hodnoty v registru E o hodnotu registru D => A
ROL	E	D	Bitová rotace vlevo hodnoty v registru E o hodnotu registru D => A
CMP	E	A	Porovná hodnoty v registrech E a A, výsledek ve stavovém registru
BRANCH	B	-	Nepodmíněný skok na adresu danou hodnotou v registru B
BRZERO	B	-	Skok na adr. v reg. B, pokud byl výsledek operace ALU 0
BROVERFL	B	-	Skok na adr. v reg. B, pokud bošlo při násobení k přetečení
JE	B	-	Skok na adr. v reg. B, poku je výsledek srovnání - rovnost
JG	B	-	Skok na adr. v reg. B, pokud je výsledek srovnání – větší
JGE	B	-	Skok na adr. v reg. B, pokud je výsledek srovnání – větší nebo rovno
JL	B	-	Skok na adr. v reg. B, pokud je výsledek srovnání – menší
JLE	B	-	Skok na adr. v reg. B, pokud je výsledek srovnání–menší nebo rovno
NOP	-	-	Neprovede nic
HALT	-	-	Konec programu, není řízena mikroprogramově

Tabulka 7-11: Implementované instrukce

7.1.4.1 Příklad mikroprogramu (operace ADD mezi registry D a E)

V této podobě se vyskytují mikroinstrukce v paměti ROM pro mikroprogramy, jednotlivé bity ovládají v použitém horizontálním kódování vždy jeden signál. Jako příklad je uvedena operace sečtení hodnot uložených v registrech D a E. Celý mikroprogram sestává z 6 mikroinstrukcí, první nastaví datové cesty operandů a přivede je na vstup ALU, nastaví operaci příslušnou operaci ALU jednotky. V dalším kroku přivede výsledek z výstupu ALU jednotky do registru A. V dalším kroku inkrementuje hodnotu v registru s adresou následující instrukce, následně vystaví novou adresu na adresovou sběrnici a signálem READ si vyžádá vystavení hodnoty instrukce na datovou sběrnici. Tu uloží v předposledním kroku do registru pro instrukci, která má být vykonána. Poslední mikroinstrukce slouží jako zarážka neboli konec mikroprogramu. Podstatná je nulová hodnota bitu 0 (úplně vpravo), která uvolňuje procesor z vykonávání tohoto mikroprogramu a umožňuje vykonání dalšího.

Mikroinstrukce číslo 1

```
00010010110000000000000000000000101000000000001
```

```
Nastaví mux1 vstup z registru D, mux2 vstup z registru E, mux3
přímý průchod, mux4 přímý průchod, ALU operace ADD,
```

Mikroinstrukce_číslo 2

```
00000000000000000001000000000000000001000000001
```

Výstup ALU do akumulátoru -mux A ALUres, reg A in nastaví na 1

Mikroinstrukce číslo 3

```
10000000000000000100000000000000000000000000001
```

```
inkrementuje hodnotu registru nextInstrAddr
```

Mikroinstrukce číslo 4

```
00000000000000111000000000000000000000000000001
```

vystaví hodnotu registru nextInstrAddr na adresovou sběrnici

Mikroinstrukce číslo 5

```
00100000000000000000000000000000000000000000101
```

Řídí načtení další instrukce z paměti a uloží hodnotu na datové sběrnici do regNextInstr.

Mikroinstrukce číslo 6

```
00000000000000000000000000000000000000000000000
```

```
konec mikroprogramu
```

7.1.5 Strukturní modul top (top.vhd)

Jedná se o modul propojující všechny ostatní moduly mezi sebou navzájem a mapuje jednotlivé vstupní a výstupní porty z jednoho modulu na správné porty v patřičném dalším modulu. Je to sekce na níž je nejvhodnější provádět simulace, jelikož jsme na ní schopni sledovat veškerou komunikaci jednotlivých modulů.

7.1.6 Modul testbench (tb.vhd)

Jedná se o modul, který má sloužit pro testování vytvářeného návrhu. Může také generovat všechny vnější řídící signály, jako je hodinový signál, signál reset, nebo aktivační signál. Je to jakási zjednodušená emulace okolního prostředí (technického vybavení), pro nějž byl procesor vyvinut.

Přiložený testbench provádí následující operace:

- Generuje hodinový signál
- Před spuštěním procesoru jej resetuje pomocí signálu RESET
- Po zavedení testovacího programu do paměti SRAM aktivuje procesor signálem CE

Podoba testovacího programu, který je modulem testbench zaveden do operační paměti a následně vykonán procesorem viz. kapitola 8.1.1.

7.2 Shrnutí implementace

- Počet navržených bloků: 6
  (cpu, controller, controller_rom, sram, top, tb)
- Celkový počet registrů výkonné části procesoru: 10 z toho 7 uživatelských
  (všechny až na stavový registr 16ti bitové)
- Počet navržených multiplexorů ve výkonné části procesoru: 14
  (7 pro uživatelské registry, 4 pro řízení vstupu operandů do ALU, 2 pro řízení vstupu na datovou a adresovou část sběrnice, 1 pro řízení vstupu do registru adresy následující instrukce)
- Počet operací vykonávaných ALU: 14
  (ADD, SUB, MUL, INC, DEC, AND, OR, XOR, IO/NEG/NOP, ROR, ROL, SHR, SHL, CMP)
- Počet signálů nutných k mikroprogramovému řízení výkonné části procesoru: 42
  7 signálů řídí povolení zápisu do uživatelských registrů,
  29 signálů řídí multiplexory,
  4 signály přenáší kód operace do ALU jednotky,
  2 signály řídí zápis do instrukčních registrů,
- Počet bitů mikroinstrukce (velikost řídícího slova): 46 (horizontální kódování)
  42 bitů ovládá signály výkonné části procesoru,
  2 bity řídí signály operační paměti (READ/WRITE),
  1 bit nutný k identifikaci konce mikroprogramu,
  1 bit na pozici (1) je nevyužit – rezervní bit,
- Počet řídících signálů v návrhu: 8
  Globální hodinový signál CLK (generován modulem tb),

Globální resetovaní signál RESET (generován modulem tb),
Signál pro aktivaci procesoru CE (generován modulem tb),
Signály pro řízení operační paměti READ, WRITE (generován modulem controller),
Signály pro řízení operační paměti při zavádění testovací rutiny READTB, WRITETB,
Signál čtení z paměti ROM s mikroprogramy REDMROM (generován modulem controller)

- Počet sběrnic v návrhu: 3
  34bitová mezi CPU a SRAM (16bit data, 16bit adresa, 2 bity řídící),
  34bitová mezi TB a SRAM (16bit data, 16bit adresa, 2 bity řídící) – zavedení testovací rutiny
  10bitová ADDRROM přenáší adresu mikroinstrukce, která má být navrácena z ROM řadiči.
- Velikost operační paměti: 1000 16bitových bloků

8 Simulace

Simulace návrhu byla provedena za pomoci již výše zmíněného nástroje ModelSim XE III/Starter 6.2g. Způsob testování byl založen na modulu testbench. Jeho prostřednictvím, je možné do operační paměti zavést kód programu, popřípadě hodnoty konstant, před aktivací samotného procesoru. K těmto účelům je implementována speciální sběrnice propojující modul testbench s modulem operační paměti. Sběrnice je rozdělena na adresovou, datovou a řídící část. Pro zavedení programu je tedy nutné v jazyce VHLD popsat časovou posloupnost generování signálů přivedených na vstup sběrnice a pomocí správných řídících signálů zajistit jejich uložení do operační paměti. Z toho plyne, že program je nutné zapsat ve strojovém kódu podle kapitol 6.7 a 6.8 (tabulka 6-1). Jako příklad si uvedeme jednoduchý program, který načte z paměti do registru D hodnotu konstanty 1 (tedy instrukce CLOAD) a ukončí se (instrukce HALT). Po zavedení programu do operační paměti se pomocí signálu CE aktivuje procesor a proces, který řídil načítání testovacího programu se uvede do stavu nečinnosti, díky příkazu `wait for` *a dlouhého časového úseku (ten by měl být delší než bude doba trvání simulace).*

```
-- příprava před zavedením kódu do operační paměti
CE <= '0'; --deaktivace procesoru
RESET <= '1';   --resetovací signál nastaven na hladinu 1 (RESET)
wait for 10 ns; --signál reset bude aktivní po dobu 10ns
RESET <= '0';   --po uplynutí tédo doby je opět deaktivován
READTB <= '0';  --řídící signál paměti na operaci READ, je defaultně
                --na nízké úrovni (z paměti nečteme)
-- naplnění paměti
wait until CLK = '1'; --počká na horní hladinu hodinového signálu
```

```
DATABUSTB <= "0000110001100000"; --instrukce CLOAD do registruD na
                                 --datovou část sběrnice
ADDRBUSTB <= "0000000000000000"; --adresa na adresovou sběrnici
WRITETB <= '1'; --aktivace řídícího signálu paměti WRITE
wait until CLK = '0'; --počká na nízkou hladinu hodinového signálu
WRITETB <= '0'; -- a deaktivuje řídící signál pro zápis do paměti

wait until CLK = '1'; --počká na horní hladinu hodinového signálu
DATABUSTB <= "0000100000000000"; --instrukce HALT na
                                 --datovou část sběrnice
ADDRBUSTB <= "0000000000000001"; --na adresu v paměti 1
WRITETB <= '1'; --aktivace řídícího signálu paměti WRITE
wait until CLK = '0'; --počká na nízkou hladinu hodinového signálu
WRITETB <= '0'; --a deaktivuje řídící signál pro zápis do paměti
wait until CLK = '1'; --počká na horní hladinu hodinového signálu
CE <= '1'; --a pomocí signálu CE proběhne aktivace procesoru
wait for 100000 ns; --zbytek času je tento proces modulu testbench
                    --neaktivní
```

Dalším procesem, který je zařazen v modulu testbench je generátor hodinového signálu, ve zdrojových souborech je generována změna úrovně hodinového signálu každých 10ns. Tento hodinový signál je distribuován pomocí modulu top (nejvyšší vrstvy) do ostatních bloků návrhu. Po kompilaci zdrojových souborů v prostředí ModelSim je možné spustit simulaci. Pro simulaci je nejvhodnější zvolit modul top, který zastřešuje celý návrh a je z něj patrná komunikace mezi jednotlivými částmi. Délka simulace závisí na složitosti a délce rutiny, kterou budeme simulovat. Nejčastěji se délka provádění jedné instrukce pohybuje okolo 7-9hodinových cyklů (140-180ns, při periodě hodinového signálu 20ns) + je nutné počítat ještě s dobou po kterou je program zaváděn modulem testbench do operační paměti.

8.1 Výsledky simulací

8.1.1 Podoba testovacího programu

- Pro účely simulace byla vytvořena speciální rutina v programu testbench. Je v ní použita většina implementovaných instrukcí včetně 2 skokových. Program obsahuje 2 cykly a délka celé rutiny je 23,8μs (po instrukci HALT). Symbolický zápis tohoto programu je součástí přílohy (viz.: příloha3).

8.1.2 Průběhové diagramy testovacího programu

Grafická dokumentace k této kapitole je součástí přílohy (viz.: příloha2).

- Obrázek 8-1: simulace modulu top: vidíme průběh zavádění programu z modulu tb do sram.

  Důležité jsou signály top_write_tb, což je řídící signál pro zápis do operační paměti, dále vektory top_addrbus_tb a top_databus_tb, které reprezentují adresovou a datovou část sběrnice. Je možné nalézt průběh hodinového signálu top_clk. A na konci diagramu, kdy už je program zaveden do operační paměti, můžeme pozorovat vzestupnou hranu signálu top_ce, která představuje aktivační signál procesoru.

- Obrázek 8-2: simulace modulu top: tento diagram zachycuje provádění instrukce

  CLOAD E 7 na nejsvrchnější vrstvě procesoru (vrstva propojovacích signálů mezi moduly), tato instrukce je zajímavá tím, že během svého vykonávání načítá hodnotu, kterou očekává na následující adrese v paměti ihned za samotnou instrukcí. Na diagramu si všimněme vektoru top_addrrom, který ukazuje na požadovanou adresu mikroinstrukce v paměti ROM, zároveň je signál top_readmrom uveden na horní hladinu a začne vykonávání mikroprogramu, který je sekvenčně načítán z RAM. Jako první je inkrementován registr uchovávající adresu následující instrukce v paměti ROM. Tuto skutečnost poznáme podle hodnoty 1 signálu top_mux7_sel. Hned v dalším kroku je inktementovaná adresa pomocí mux 6 vystavena na adresovou sběrnici. To pozorujeme změnami řídícího vektoru top_mux6 sel a přímo na samotné adresové sběrnici top_addrbus. V následujícím kroku je uveden signál top_read na hladinu 1, což je povel pro operační paměť vystavit na datovou sběrnici data, na které ukazuje hodnota na adresové sběrnici. To se děje synchronně se sestupnou hranou hodinového signálu. Mezitím už je nastaven řídící signál multiplexoru pro registr E (top_muxE_sel), aby přivedl na vstup registru data z datové sběrnice, stejně tak je aktivní signál top_rege_in, který povoluje zápis hodnoty předané multiplexorem E do registru E. V dalším kroku je opět inkrementována hodnota v registru pro adresu následující instrukce, posléze je hodnota stejným mechanismem vystavena na adresovou sběrnici, opět je aktivován signál READ, který řídí činnost paměti, ta na datovou sběrnici vystaví tentokrát hodnotu následující instrukce. Hodnota je přečtena a uložena do registru regNextInstr, toto poznáme díky signálu top_regnextinstr_in, který má v danou chvíli hodnotu 1.

- Obrázek 8-3: zachycuje provádění shodné operace CLOAD E 7 jako je znázorněno na obrázku 8-2, ovšem tentokrát z vnitřní struktury procesoru, mimo signály, které jsme mohli pozorovat již na svrchní vrstvě na obrázku 8-2, je zde důležitá jedna věc a to změna hodnoty v registru E, z předchozí hodnoty na požadovanou hodnotu 7, která byla tímto úspěšně načtena z paměti.
- Obrázek 8-4: opět instrukce CLOAD E 7, tentokrát diagram průběhu v modulu řadiče. Proces vykonávání mikroprogramu začíná, když je signál in_progress na nízké úrovni, to značí, že není vykonáván žádný mikroprogram a je tedy možné zahájit vykonávání instrukce, která je v registru NextInstr. Společně s dekódováním instrukce je nastavena hodnota signálu in_progress na 1, což signalizuje, že je právě prováděn nějaký mikroprogram. Po dekódování instrukce je známa adresa první mikroinstrukce mikroprogramu uloženého v paměti ROM. Ta je vystavena na sběrnici ADDRROM a s následující vzestupnou hranou hodinového signálu řadič obdrží patřičnou mikroinstrukci, v tu samou chvíli již požádá o další mikroinstrukci. Ty jsou čteny sekvenčně (zřejmé podle hodnot vystavených na sběrnici ADDRROM v průběhu vykonávání mikroprogramu), Mikroinstrukce je rozkódována a dle jejího kódu jsou aktivovány řídící signály procesoru. Po provedení poslední mikroinstrukce, se hodnota signálu in_progress vrátí opět na hodnotu 0, tím je řadič připraven k vykonávání další instrukce, která byla načtena do příslušného registru, konečnou sekvencí mikroinstrukcí v předchozím mikroprogramu.

9 Syntéza do FPGA

9.1 FPGA

FPGA je zkratka pro field-programmable-gate-array, přeloženo do češtiny hradlové pole. Jedná se o polovodičové zařízení, které je možné konfigurovat k provádění logických funkcí. Nejčastějším způsobem je popis obvodu pomocí zdrojového kódu v HDL jazyce a jeho následnou syntézou do FPGA. Klasické FPGA je tvořeno na okrajích I/O bloky a uvnitř logickými bloky (viz. obrázek 9-1), které mohou provádět jak komplexní logické funkce, tak prosté funkce jako AND nebo OR, bloky jsou propojeny propojovacími cestami. Dalšími přítomnými prvky jsou moduly RAM, násobičky atd.

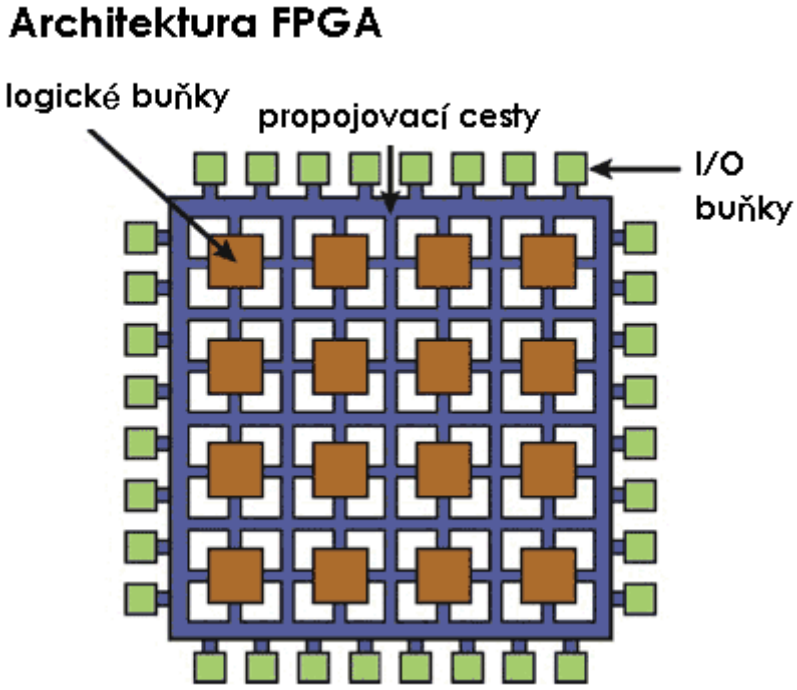
Mezi I/O bloky mohou být také zpožďovací linky na násobení frekvence atd. Velikost FPGA se udává počtem těchto logických bloků, například (10x10).

Architektura od firmy Xilinx má většinou takovou koncepci, že každá buňka (SLICE) obsahuje dvě LUT (Look Up Table), do níž můžeme vložit jakoukoliv čtyřvstupovou logickou funkci, a dva klopné obvody, pro každou LUT jeden.

Technologie FPGA je schopna pojmout jakoukoliv aplikaci. Nejčastěji se používají pro synchronní design, jen v případě velmi jednoduchých aplikací se jedná o design asynchronní. Tato technologie je nasazována opravdu téměř na cokoliv, od realizace DSP počínaje, přes obranné a zdravotnické systémy, až po simulaci vyvíjených HW návrhů.

Díky paralelizaci a zřetězení dosahují FPGA vysokého výpočetního výkonu. Maximální taktovací rychlost těch nejlevnějších FPGA se pohybuje i nad 100MHz a jsou schopny zvládnout i obsluhu DDR II pamětí na kmitočtu 133MHz (266MHz efektivně).

Návrh designu pro FPGA bývá vytvořen buď v nějakém HDL (VHDL / Verilog) jazyce nebo jako schématický diagram. Nástroje pro zpracování těchto návrhů vytvářejí takzvaný netlist (technologickou mapu pro specifické FPGA), na jehož základě je možné ověřit správnou funkčnost návrhu, z něj je poté vytvořen binární soubor, který je možné nahrát většinou pomocí software dodávaného výrobcem daného FPGA do FPGA zařízení. [16]



Obrázek 9-1: Architektura FPGA

9.2 Výsledky syntézy vlastního návrhu

Syntéza byla provedena pro cílové zařízení FPGA Virtex 5 (modelové označení xc5vlx30-3ff324), pomocí nástroje Xilinx ISE 9.2i. Výsledný odhad po provedení syntézy návrhu je v tabulce 9-1.

Logické bloky	Použito	K dispozici	Vytížení
Počet registrů logických buněk	640	19200	3%
Počet LUT logických buněk	1383	19200	7%
Počet plně využitých Bit Slices	341	1682	20%
Počet propojených vstupně/výstupních portů	0	220	0%
Počet BUFG/BUFGCTRL	1	32	3%
Počet DSP48E	1	32	3%

Tabulka 9-1: odhad využití FPGA návrhem

10 Závěr a možná rozšíření

Při celém návrhu bylo dbáno na jednoduchost a použitelnost. Vytvořená sada instrukcí a jí přizpůsobená architektura, by měla být schopna obsloužit jednodušší programy, založené na celočíselné aritmetice a základních logických operacích. Sada registrů poskytuje dostatečný prostor i pro ukládání mezivýsledků a tím minimalizujeme nutnost pracovat s operační pamětí.. Implementovaný mikroprogramový řadič umožňuje velice snadno zasahovat do instrukční sady procesoru. Jeho nevýhodou, v podobě jak je implementován, je nutnost pro každou instrukci napsat mikroprogram i pro jednotlivé kombinace vstupních, popřípadě výstupních registrů. Tato vlastnost je důsledkem řízení datových cest pomocí mikroinstrukcí. Mnohem efektivnější by bylo dekódovat instrukci na operační kód a adresy operandů a pro nastavení datových cest použít jednoduchou logiku pracující na základě adres operandů. Ovšem většina řídících signálů procesoru se týká právě datových cest a jen minimum je použito pro vykonání samotného operačního kódu a tak by použití mikroprogramového řadiče nebylo příliš efektivní, jelikož by byla část procesoru řízená mikroprogramy opravdu velmi malá. Proto pro účely zadání jsou všechny segmenty procesoru řízeny mikroprogramově a z výše zmíněného důvodu nejsou implementovány instrukce pro všechny kombinace registrů, ale vždy pouze pro nějaké specifické registry. Z povahy mikroprogramového řadiče vyplývá, že případné doplnění instrukcí pro operace nad dalšími registry je již jen triviální přepisování mikroprogramů.

Instrukční sadu by bylo dále vhodné rozšířit o instrukce IN a OUT, které by umožnily připojení nějakého externího zařízení.

Dalším vhodným rozšířením by byla optimalizace načítání následujících instrukcí, které v současné chvíli probíhá vždy na konci každého mikroprogramu, ale u některých instrukcí by bylo možné tento krok provádět alespoň částečně ještě při vykonávání instrukce.

Konečná syntéza kódu do FPGA byla příčinou spousty úprav ve zdrojových kódech, jelikož nástroje pro syntézu odmítají pracovat s některými konstrukcemi, které jsou i podle simulací v nástroji ModelSim použitelné. Jednalo se zejména o části spojené se synchronizací pomocí hodinového signálu nebo sběrnic s obousměrným tokem dat, které bylo nutné implementovat jako třístavové sběrnice. Syntéza byla také časově velmi náročná, obzvláště při pokusu o syntézu modulu operační paměti v plném rozsahu daném šířkou adresové sběrnice. Konečné simulace a testy potvrdily, že navržená architektura je plně funkční a splnila kritéria na nichž byl postaven její návrh, ovšem bylo by vhodné provést výše zmíněná rozšíření k získání větší efektivity návrhu.

Literatura

[1] Synario Design Automation: *VHDL reference manual* [online]. Březen 1997 [cit. 26.4.2009]. Dostupné z URL: http://www.usna.edu/EE/ee462/MANUALS/vhdl_ref.pdf

[2] Kolektiv autorů projektu wikipedia.org: *Procesor* [online]. 23.4.2009 [cit. 26.4.2009]. Dostupné z URL: http://cs.wikipedia.org/wiki/Procesor

[3] Kolektiv autorů projektu wikipedia.org: *Digital signal processing* [online]. 7.4.2009 [cit. 26.4.2009]. Dostupné z URL: http://en.wikipedia.org/wiki/Digital_signal_processing

[4] ©Intel corporation: *Intel® Hyper-Threading Technology (Intel® HT Technology)* [online]. [cit. 26.4.2009] Dostupné z URL: http://www.intel.com/technology/platform-technology/hyper-threading/

[5] Kolektiv autorů projektu wikipedia.org: *VHDL* [online]. 31.3.2009 [cit. 27.4.2009]. Dostupné z URL: http://cs.wikipedia.org/wiki/VHDL

[6] Jaromír Krotký: *Výroba procesorů* [online]. [cit. 27.4.2009] Dostupné z URL: http://www.fi.muni.cz/usr/jkucera/pv109/2002/xkrotky_proc.html#liter

[7] ©Intel corporation: *Technology* [online]. [cit. 27.4.2009]. Dostupné z URL: http://www.intel.com/technology/index.htm?iid=gg_work+home_technology

[8] ©Advanced Micro Devices: *Technical Documentation* [online]. 2009 [cit. 27.4.2009] Dostupné z URL: http://support.amd.com/us/Pages/techdocs.aspx

[9] Kolektiv autorů projektu wikipedia.org: *CPU power dissipation* [online]. 23.4.2009 [cit. 29.4.2009] Dostupné z URL: http://en.wikipedia.org/wiki/CPU_power_dissipation

[10] Jiří Marchalín: *Historie procesorů od počátku po současnost* [online]. Univerzita Karlova v Praze. 2004 [cit. 30.4.2009] Dostupné z URL: http://artax.karlin.mff.cuni.cz/~marcj1am/download-files/hist.pdf

[11] *Cpu-museum.de* [online]. 2009 [cit. 30.4.2009] Dostupné z URL: http://www.cpu-museum.de/

[12] Kolektiv autorů projektu wikipedia.org: *Počítač* [online]. 20.4.2009 [cit. 30.4.2009] Dostupné z URL: http://cs.wikipedia.org/wiki/Po%C4%8D%C3%ADta%C4%8D

[13] Hennessy, J.L., Patterson, D.A. *Computer Architecture - A Quantitative Approach, Third Edition*, Morgan Kaufmann Publishers, New York, 2003. 1000 s. ISBN 1-55860-596-7

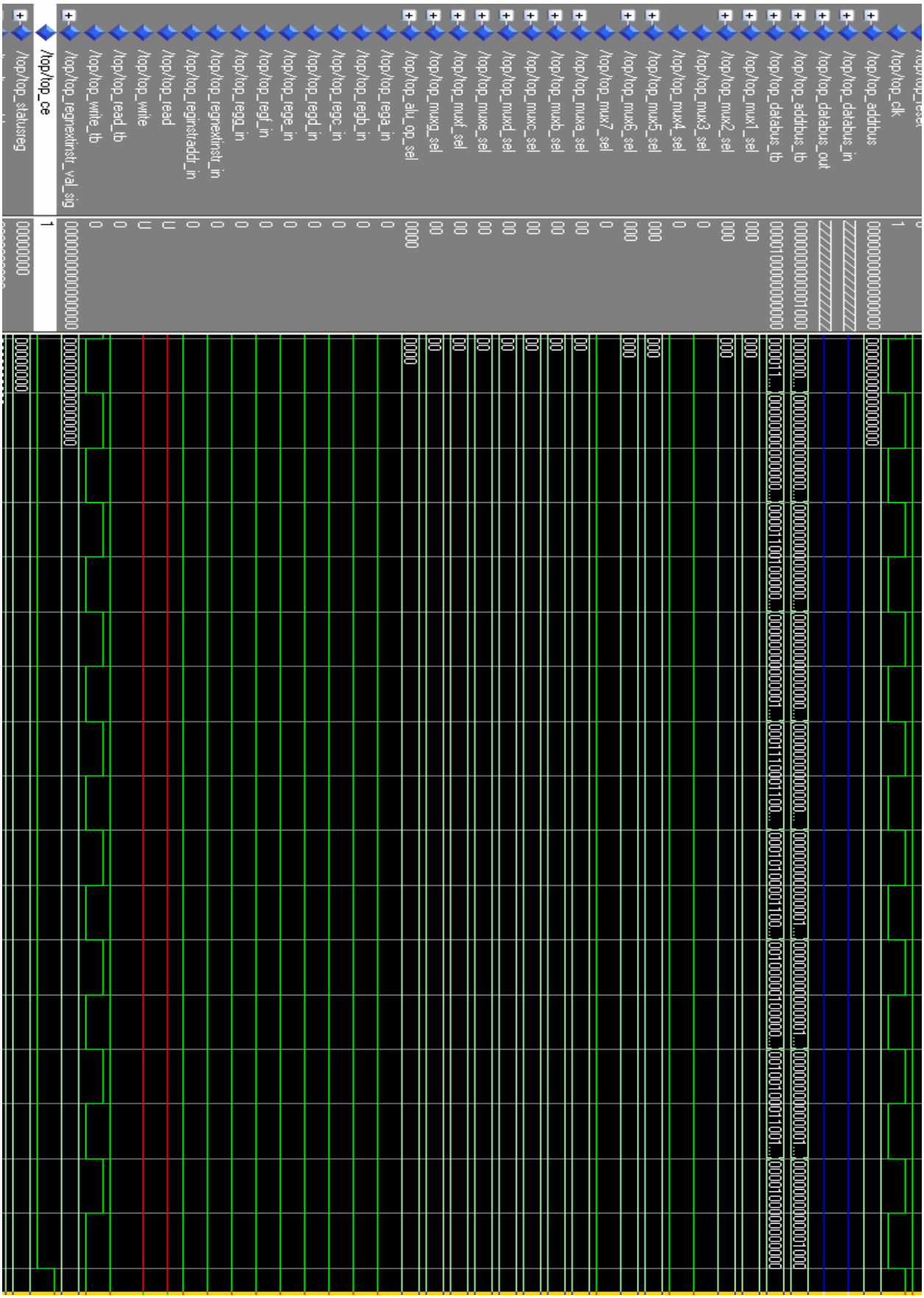
[14] Dvořák, V., Drábek , V. *Architektura procesorů*, VUTIUM, Brno, 1999. 293 s. ISBN 80-214-1458-8

[15] Cohen, B. *VHDL Coding Styles and Methodologies*, Kluwer Academic Publishers, Dordrecht, 2001. 453 s. ISBN 0-7923-8474-1

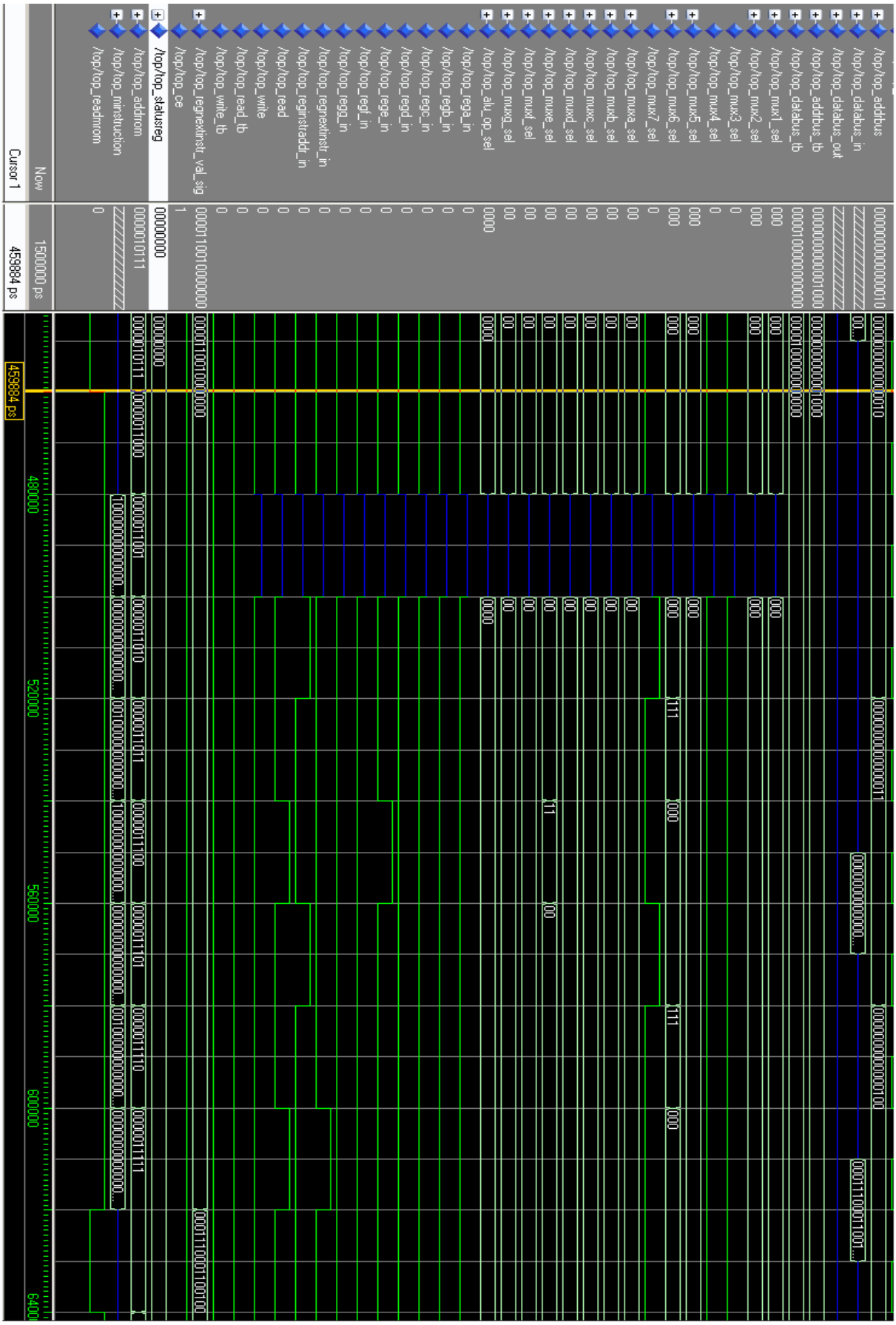
[16] Kolektiv autorů projektu wikipedia.org: *Field-programmable gate array* [online]. 8.5.2009 [cit. 5.5.2009] Dostupné z URL: http://en.wikipedia.org/wiki/Field-programmable_gate_array

Seznam příloh

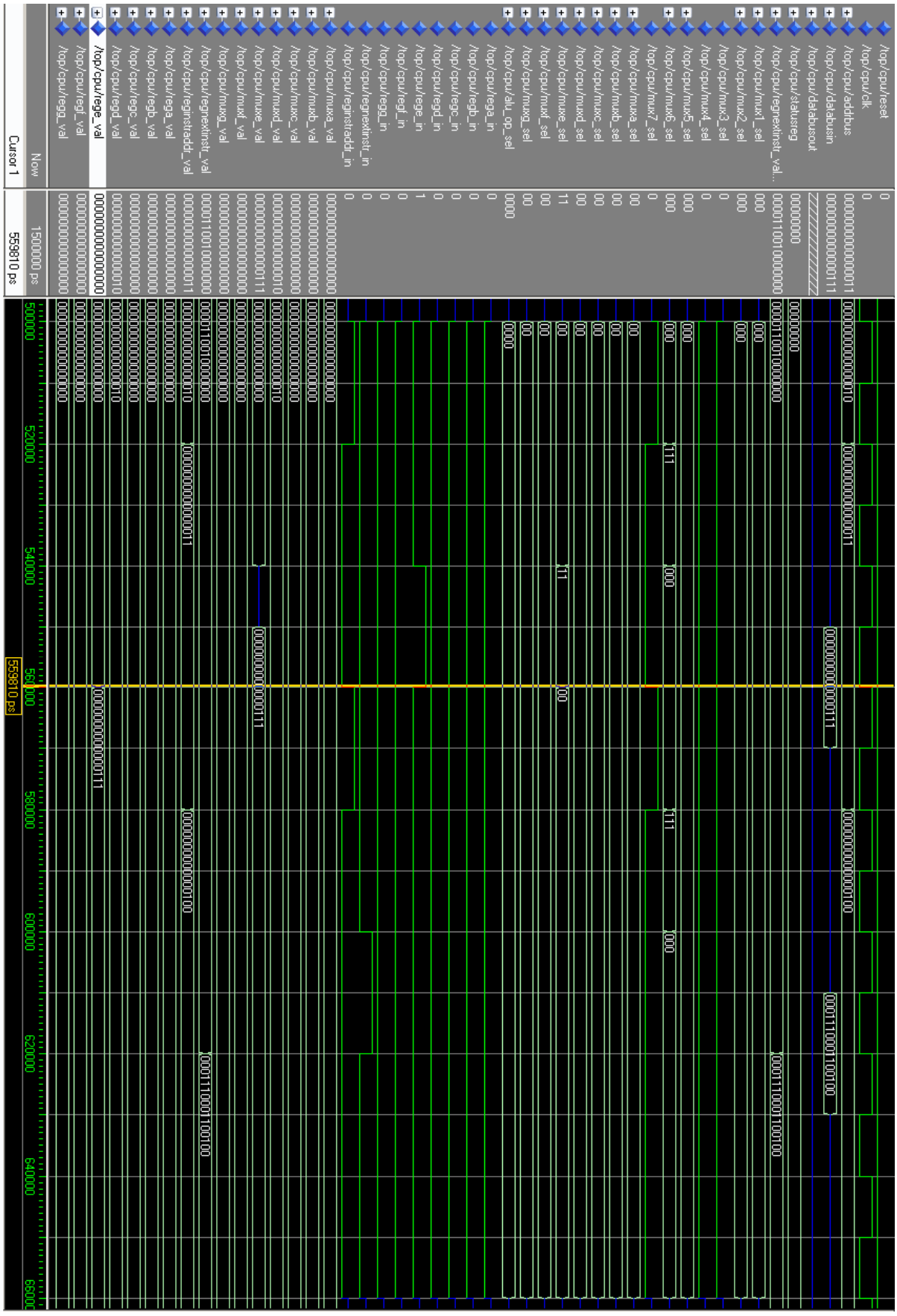
Příloha 1: Zdrojové texty na optickém nosiči (CD)
Příloha 2: Diagramy simulací v nástroji Modelsim ke kapitole 8.1.2 (4 listy A4)
Příloha3 : Symbolický zápis programu v modulu testbench ke kapitole: (1 list A4)



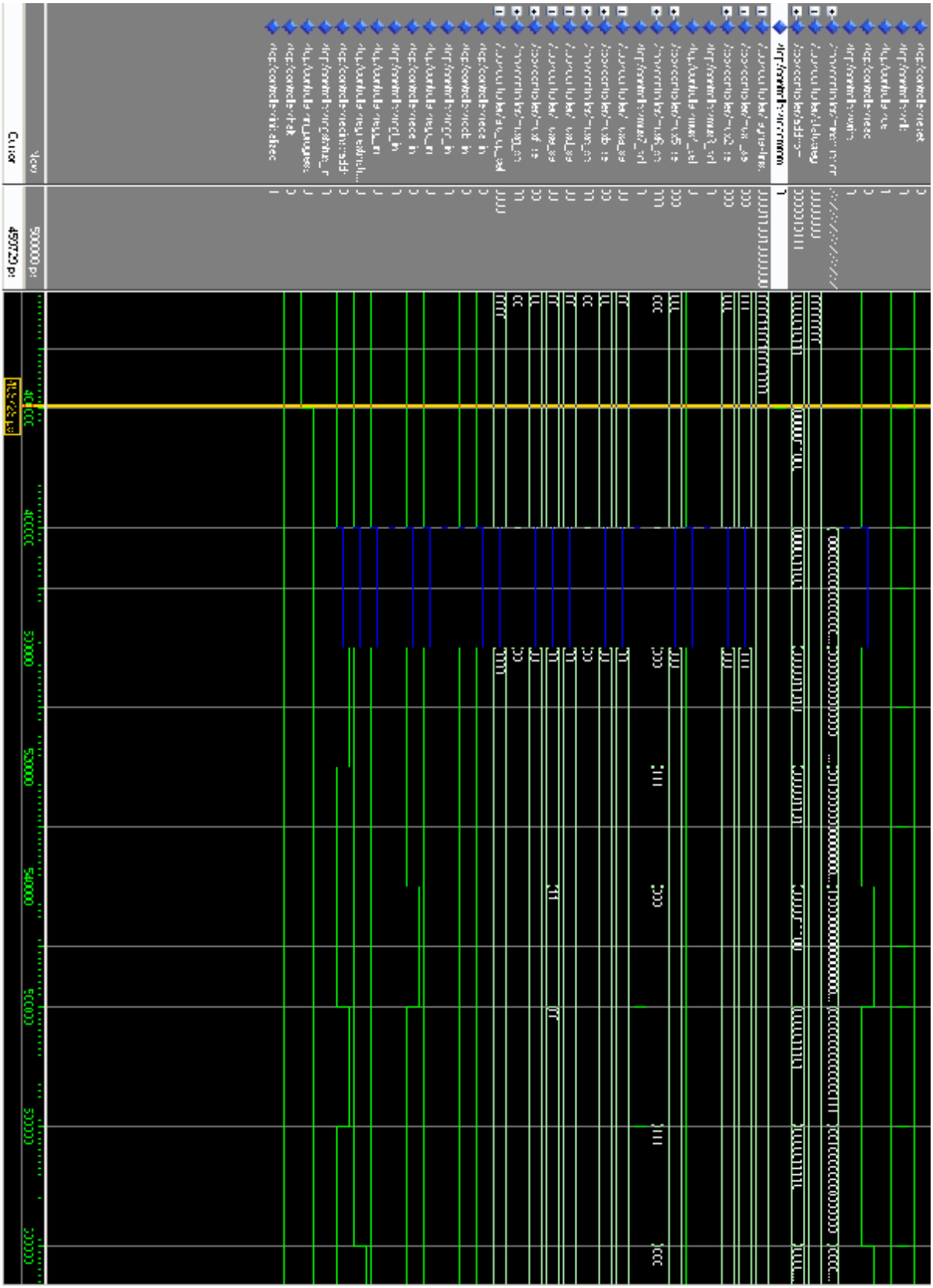
Obrázek 1: Diagram zavádění instrukcí do operační paměti (modul top)



Obrázek 2: Diagram vykonávání operace CLOAD E (modul top)



Obrázek 3: Diagram operace CLOAD E (modul CPU)



Obrázek 4: Diagram operace CLOAD E (modul controller)

Symbolický zápis programu v modulu test bench

Adresa v paměti	Instrukce	Popis
0	CLOAD D, 2	D = 2
2	CLOAD E, 6	E = 6
4	ADD D, E	A = 8
5	MOV A, B	B = 8
6	SUB E, D	A = 4
7	MUL D, E	A = 12
8	INC E	E = 7
9	CMP E, A	Porovnání
10	JLE B	Pokud je A > E skok na instrukci na adrese B = d8
11	SHR E, D	A= 3
12	SHL E, D	A= 52
13	ROR E, D	A= 131075
14	ROL E, D	A= 52
15	AND E, D	A= 0
16	OR E, D	A= 15
17	XOR E, D	A= 15
18	NEG A	A= 262128
19	DEC A	A= 262127
20	MLOAD F, B	F= 49664
21	MSTORE F, B	Uloží hodnotu v F na adresu danou hodnotou v B
22	NEG A	A= 16
23	DEC A	A= 15
24	MOV A, B	B= 15
25	MSTORE F, B	Uloží hodnotu v F na adresu danou hodnotou v B
26	CLOAD E, 35	E= 35
28	ADD D, E	A= 37
29	DEC A	A= 36
30	DEC A	A= 35
31	MOV A, B	B= 35
32	CLOAD D, 1	D= 1
34	SHL E, D	A= 70
35	DEC A	A=A-1
36	CMP E, A	Porovná E a A
37	JL B	Pokud je E menší než A skáče na hodnotu B (35)
38	HALT	KONEC

Tabulka znázorňující symbolický popis testovacího programu.