VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

ASSEMBLER PRO TAYLOROVU ŘADU

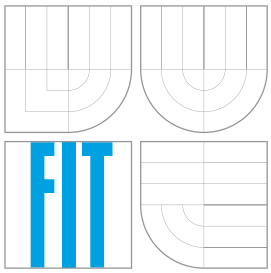
BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

VÁCLAV VALENTA

BRNO 2007

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

ASSEMBLER PRO TAYLOROVU ŘADU

ASSEMBLER FOR THE TAYLOR SERIES

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — VÁCLAV VALENTA
AUTHOR

VEDOUCÍ PRÁCE — doc. Ing. JIŘÍ KUNOVSKÝ, CSc.
SUPERVISOR

BRNO 2007

Zadání bakalářské práce/5913/2006/xvalen09

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav inteligentních systémů Akademický rok 2006/2007

Zadání bakalářské práce

Řešitel: **Valenta Václav**
Obor: Informační technologie
Téma: **Assembler pro Taylorovu řadu**
Kategorie: Modelování a simulace

Pokyny:

1. Seznamte se s problematikou analytického řešení obyčejných diferenciálních rovnic.
2. Seznamte se s problematikou numerického řešení obyčejných diferenciálních rovnic a jejich řešením v TKSL.
3. Analyzujte požadované matematické opetrace a naprogramujte je v assembleru.
4. Zaměřte se na assembler pro mikrokontroléry

Literatura:

- Dle zadání vedoucího

Při obhajobě semestrální části projektu je požadováno:

- - první 2 body zadání

Podrobné závazné pokyny pro vypracování bakalářské práce naleznete na adrese http://www.fit.vutbr.cz/info/szz/

Technická zpráva bakalářské práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap (20 až 30% celkového rozsahu technické zprávy).

Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním paměťovém médiu (disketa, CD-ROM), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí: **Kunovský Jiří, doc. Ing., CSc.**, UITS FIT VUT
Datum zadání: 1. listopadu 2006
Datum odevzdání: 15. května 2007

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta informačních technologií
Ústav inteligentních systémů
612 66 Brno, Božetěchova 2

L.S.

doc. Dr. Ing. Petr Hanáček
vedoucí ústavu

LICENČNÍ SMLOUVA
POSKYTOVANÁ K VÝKONU PRÁVA UŽÍT ŠKOLNÍ DÍLO

uzavřená mezi smluvními stranami

1. Pan

Jméno a příjmení: **Václav Valenta**
Id studenta: 84118
Bytem: M. Krškové 638, 391 81 Veselí nad Lužnicí
Narozen: 17. 07. 1985, Tábor
(dále jen "autor")

a

2. Vysoké učení technické v Brně

Fakulta informačních technologií
se sídlem Božetěchova 2/1, 612 66 Brno, IČO 00216305
jejímž jménem jedná na základě písemného pověření děkanem fakulty:

.....
(dále jen "nabyvatel")

Článek 1
Specifikace školního díla

1. Předmětem této smlouvy je vysokoškolská kvalifikační práce (VŠKP):
bakalářská práce

Název VŠKP: Assembler pro Taylorovu řadu
Vedoucí/školitel VŠKP: Kunovský Jiří, doc. Ing., CSc.
Ústav: Ústav inteligentních systémů
Datum obhajoby VŠKP: .....

VŠKP odevzdal autor nabyvateli v:
tištěné formě počet exemplářů: 1
elektronické formě počet exemplářů: 2 (1 ve skladu dokumentů, 1 na CD)

2. Autor prohlašuje, že vytvořil samostatnou vlastní tvůrčí činností dílo shora popsané a specifikované. Autor dále prohlašuje, že při zpracovávání díla se sám nedostal do rozporu s autorským zákonem a předpisy souvisejícími a že je dílo dílem původním.
3. Dílo je chráněno jako dílo dle autorského zákona v platném znění.
4. Autor potvrzuje, že listinná a elektronická verze díla je identická.

Článek 2
Udělení licenčního oprávnění

1. Autor touto smlouvou poskytuje nabyvateli oprávnění (licenci) k výkonu práva uvedené dílo nevýdělečně užít, archivovat a zpřístupnit ke studijním, výukovým a výzkumným účelům včetně pořizování výpisů, opisů a rozmnoženin.
2. Licence je poskytována celosvětově, pro celou dobu trvání autorských a majetkových práv k dílu.
3. Autor souhlasí se zveřejněním díla v databázi přístupné v mezinárodní síti:
   - ☒ ihned po uzavření této smlouvy
   - ☐ 1 rok po uzavření této smlouvy
   - ☐ 3 roky po uzavření této smlouvy
   - ☐ 5 let po uzavření této smlouvy
   - ☐ 10 let po uzavření této smlouvy

   (z důvodu utajení v něm obsažených informací)
4. Nevýdělečné zveřejňování díla nabyvatelem v souladu s ustanovením § 47b zákona č. 111/ 1998 Sb., v platném znění, nevyžaduje licenci a nabyvatel je k němu povinen a oprávněn ze zákona.

Článek 3
Závěrečná ustanovení

1. Smlouva je sepsána ve třech vyhotoveních s platností originálu, přičemž po jednom vyhotovení obdrží autor a nabyvatel, další vyhotovení je vloženo do VŠKP.
2. Vztahy mezi smluvními stranami vzniklé a neupravené touto smlouvou se řídí autorským zákonem, občanským zákoníkem, vysokoškolským zákonem, zákonem o archivnictví, v platném znění a popř. dalšími právními předpisy.
3. Licenční smlouva byla uzavřena na základě svobodné a pravé vůle smluvních stran, s plným porozuměním jejímu textu i důsledkům, nikoliv v tísni a za nápadně nevýhodných podmínek.
4. Licenční smlouva nabývá platnosti a účinnosti dnem jejího podpisu oběma smluvními stranami.

V Brně dne: .....

.....	.....
Nabyvatel	Autor

Abstrakt

Cílem této práce je seznámení se s numerickým řešením diferenciálních rovnic. Řešení bude prováděno mikrokontrolérem HC08. Analyzují se zde základní operace potřebné pro výpočet a algoritmy které tento výpočet doprovází.

Klíčová slova

Diferenciální rovnice, taylorova řada, mikrokontrolér, HC08, assembler

Abstract

The objective of this work is to get familiar with numerical solution of differential equations. The solution is made by specialized microprocessor HC08. Basic arithmetic operations and algorithms which helps to make precise results are analyzed here.

Keywords

Differential equations, the taylor series, microprocessor, HC08, assembler

Citace

Václav Valenta: Assembler pro Taylorovu řadu, bakalářská práce, Brno, FIT VUT v Brně, 2007

Assembler pro Taylorovu řadu

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana doc. Ing. Jiřího Kunovského CSc. Uvedl jsem všechny literární prameny, z nichž jsem čerpal.

. . . . . .
Václav Valenta
2. května 2007

Poděkování

Chtěl bych poděkovat panu doc. Kunovskému za pomoc a podporu při vytváření tohoto technického díla.

Obsah

Kapitola 1

Úvod

Již od nepaměti se snaží člověk poznávat a pochopit přírodní děje. Dříve byla spoustě nevysvětlitelných jevů, jako je blesk, přisuzovaná boží podstata. V renesanci s nástupem osvícenství se začaly hledat jiné cesty, jak tyto jevy vysvětlit a to děláme až do současnosti. Objevili jsme přírodní zákony, které nám umožňují pochopit a využít je ve svůj prospěch. Naprostou nutností je tyto jevy matematicky popsat. Jakmile existuje matematický model, je možné sestavit i simulační model a provádět experimenty na počítači, čímž se snažíme napodobit skutečné přírodí děje, např. vedení elektrického proudu. V základech fyziky stojí diferenciální rovnice, jejichž aplikaci můžeme najít ve většině oblastí lidského vědění.

Jednoduché diferenciální rovnice se mohou řešit analyticky. Výhoda aplikace analytického řešení diferenciálních rovnic spočívá v jejich přesnosti a i rychlosti. Bohužel je tento způsob použitelný pouze v jednoduchých situacích. V naprosté většině reálných aplikací se vyskytují diferenciální rovnice, jejichž analytické řešení je velmi obtížné, nebo zcela nemožné. V těchto situacích nastupují numerické metody, k jejichž hlavním nevýhodám patří větší časová složitost a nepřesnost. Numerické řešení nám pouze aproximuje hledanou funkci a musíme si uvědomit, že metoda, kterou používáme, nemusí být pro danou rovnici stabilní. Řešení je potom zcela špatné. V reálném světě také můžeme narazit na řešení tzv. tuhých systémů, které vyžadují k řešení speciální integrační metody.

Cílem mé bakalářské práce bylo ověřit funkčnost a jednoduchost algoritmu Taylorovy řady na extrémní architektuře. Byl zvolen mikrokontrolér řady HC08, abychom ověřili, že je možné implementovat tento algoritmus na téměř jakékoli platformě. Jednoduchost této metody také spočívá v tom, že je možné ji implementovat přímo v assembleru bez použití vyššího programovacího jazyka.

Kapitola 2 se zabývá analytickým řešením diferenciálních rovnic a jejich teoretickým rozborem. Numerické řešení a algoritmus **Taylorova rozvoje** je popsán v **kapitole 3**.

Analýza požadavků na program je v **kapitole 4**.

Popis cílové platformy, popis řešení, algoritmus výpočtu a použití programu pro řešení rovnic je v **kapitole 5**.

S možnostmi pro budoucí vývoj se můžeme seznámit v **kapitole 6**.

Řešení různých fyzikálních úloh s použitím programu je vysvětleno v **kapitole 7**.

Kapitola 2

Analýza diferenciálních rovnic

2.1 Typy diferenciálních rovnic

Diferenciální rovnicí rozumíme matematickou rovnici, ve které vystupují jako proměnné derivace funkcí. Diferenciální rovnice můžeme rozdělit podle typu obsažených derivací na

- **Obyčejné diferenciální rovnice** – obsahují derivace hledané funkce pouze podle jedné proměnné.
- **Parciální diferenciální rovnice** – obsahují derivace hledané funkce podle více proměnných.

Řádem diferenciální rovnice rozumíme nejvyšší řád derivace, který je v ní obsažen. Za řád soustavy diferenciálních rovnic je považována nejvyšší derivace, která se v soustavě vyskytuje.

Dále hovoříme o tzv. lineárních diferenciálních rovnicích. Zde se hledaná funkce nevyskytuje jako argument jiné funkce a nenalezneme zde ani žádné součiny mezi jejími derivacemi. Zde je obecný tvar lineární diferenciální rovnice:

$$y^n + a_{n-1}(x)y^{n-1} + \cdots + a_1(x)y' = f(x) \tag{2.1}$$

- n představuje řád diferenciální rovnice
- x je nezávislá proměnná
- y^k je k-tá derivace hledané funkce $y(x)$
- $a_k(x)$ jsou koeficienty
- $f(x)$ je pravá strana diferenciální rovnice. V případě, že $f(x) = 0$ hovoříme o homogenní diferenciální rovnici.

Další informace o rozdělení diferenciálních rovnic je možné nalézt v literatuře [1].

Řešením máme na mysli takovou funkci, která má příslušné derivace a vyhovuje diferenciální rovnici. Řešení dělíme na

- **obecné** – tato řešení obsahují libovolnou konstantu. Můžeme prohlásit, že obecných řešení existuje nekonečně mnoho.
- **partikulární** – známe konstantu, tedy řešení je právě jedno. V případě jednoduchých rovnic je možné toto řešení spočítat analyticky. Většina případů je ale pro analytické řešení příliš obtížná a proto se použije řešení numerické.

2.2 Analytické řešení diferenciálních rovnic

Zde bych chtěl ukázat vzorové analytické řešení jednoduché diferenciální rovnice, čímž chci demonstrovat obtížnost tohoto způsobu řešení a praktickou nemožnost implementace obecného algoritmu pro řešení. Blíže se tomuto tématu věnuje [2, 1].

2.2.1 Homogenní diferenciální rovnice

Zvolil jsem jednoduchou rovnici vhodnou pro demonstraci.

$$y' + y = 0 \tag{2.2}$$

U této rovnice sestavíme tzv. charakteristickou rovnici.

$$\lambda + 1 = 0 \tag{2.3}$$

Jakmile známe hodnotu λ, můžeme zapsat obecné řešení rovnice ve tvaru

$$y = Ce^{\lambda t} \tag{2.4}$$

Abychom byli schopni toto řešení prakticky využít, musíme si vybrat z nekonečného množství rovnic právě jednu. Proto vstupuje do hry počáteční podmínka. Tu zvolíme vhodně jako $y(0) = 1$. Dosadíme do obecného řešení a vypočítáme konstantu C.

$$1 = Ce^0 \tag{2.5}$$

$$C = 1$$

Pro zpětnou kontrolu dosadíme do původní rovnice, kde nám vyjde

$$(e^{-t})' + e^{-t} = 0 \tag{2.6}$$
$$-e^{-t} + e^{-t} = 0 \tag{2.7}$$
$$0 = 0 \tag{2.8}$$

Čímž jsme ověřili správnost řešení. Teorie a příklady jsou v [2, 1].

2.2.2 Nehomogenní diferenciální rovnice

Zde jsem zvolil opět rovnici prvního řádu. Nehomogenní diferenciální rovnice má pravou stranu nenulovou.

$$y' + y = e^t \tag{2.9}$$

Opět začneme s charakteristickou rovnicí.

$$\lambda + 1 = 0 \tag{2.10}$$
$$\lambda = -1 \tag{2.11}$$

Nyní, protože řešíme nehomogenní diferenciální rovnici, bude řešení ve tvaru

$$y = C(t)e^{\lambda t} \tag{2.12}$$

$C(t)$ je funkce času, kterou je nutné dále dopočítat. Využijeme tzv. variaci konstant. K tomu potřebujeme dopočítat derivaci pravé strany.

$$y' = C(t)'\mathrm{e}^{-t} - C(t)\mathrm{e}^{-t} \tag{2.13}$$

Dosazením do původní rovnice můžeme vyjádřit funkci $C(t)$.

$$\begin{aligned} C(t)'\mathrm{e}^{-t} - C(t)\mathrm{e}^{-t} + C(t)\mathrm{e}^{-t} &= \mathrm{e}^{t} && (2.14)\\ C(t)' &= \mathrm{e}^{2t} && (2.15) \end{aligned}$$

Abychom získali funkci $C(t)$, musíme obě strany rovnice integrovat.

$$C(t) = \int \mathrm{e}^{2t}\mathrm{d}t \tag{2.16}$$

Použijeme substituční metodu řešení integrálu a nakonec dostaneme výsledek.

$$C(t) = \frac{1}{2}\mathrm{e}^{2t} + c \tag{2.17}$$

Když funkci $C(t)$ dosadíme do řešení rovnice, vyjde nám $y = \frac{1}{2}\mathrm{e}^{t} + c$. Opět jsme obdrželi obecné řešení, kde figuruje konstanta, jejíž hodnotu lze dopočítat z počáteční podmínky. Nástin analytického řešení nehomogenních diferenciálních rovnic prvního i vyššího řádu je v literatuře [1, 2].

Je nutné podotknout, že použití charakteristické rovnice je omezeno do 3. řádu, kde nastává problém při řešení rovnic vyšších řádů. Dále jsme omezeni schopností integrování. Výpočet itegrálu složitější funkce může být problematický, ne-li zcela nemožný. Například integrál $\int \mathrm{e}^{x^2}\mathrm{d}x$ je analyticky neřešitelný.

V případě, že známe analytické řešení, je jeho použití jistě výhodnější z hlediska přesnosti a rychlosti výpočtu.

Kapitola 3

Numerické řešení diferenciálních rovnic

Nejprve bych rád pohovořil o tom, co vlastně považujeme za *numerické řešení* diferenciální rovnice. Při analytickém řešení získáme funkci. Tato funkce může být i úsek počítačového kódu, který pro libovolné číslo z definičního oboru této funkce je schopen spočítat odpovídající hodnotu. U numerického řešení je to jinak. Výsledkem je množina bodů, jež nám aproximují graf hledané funkce. Tedy nevíme jak na sobě jednotlivé body navazují.

Jestliže budeme potřebovat alespoň přibližnou hodnotu pro libovolné vstupní číslo z definičního oboru, je vhodné využít nějakou interpolační funkci, například Lagrangeův nebo Newtonům interpolační polynom.

3.1 Úpravy diferenciálních rovnic

Vždy když budeme numericky řešit nějakou diferenciální rovnici, musíme si uvědomit, že naše použitá metoda nebude umět řešit rovnici v jakémkoli tvaru. Nejčastěji budeme požadovat, aby rovnice byla pouze prvního řádu taková, kde derivovaný člen je na levé straně. Tedy např.

$$y' = 5y + 4 \tag{3.1}$$

Jestliže je systém popsán diferenciální rovnicí vyššího řádu než prvního, musíme ji převést na soustavu rovnic prvního řádu. Převod můžeme uskutečnit s pomocí dvou metod – **metoda postupné integrace** a **metoda snižování řádu derivace**. Obě tyto metody jsou podrobně popsány v [6].

3.1.1 Metoda postupné integrace

Principem je převod nejvyšší derivace na levou stranu rovnice a všechny ostatní na pravou stranu rovnice. Potom dochází postupně k integraci a zavedení nové proměnné. V podstatě přidáme další diferenciální rovnici prvního řádu. Tato metoda generuje soustavu diferenciálních rovnic, které vyhovují kritériím numerických metod.

Zde bych uvedl příklad použití metody postupné integrace. Mějme diferenciální rovnici 2. řádu.

$$p^2y + 2py + y = p^2x + 3px + 2x \tag{3.2}$$

Nejprve osamostatníme člen s nejvyšší derivací.

$$p^2y = p^2x + p(3x - 2y) + (2x - y) \tag{3.3}$$

Budeme poprvé integrovat obě strany rovnice.

$$py = px + (3x - 2y) + \frac{1}{p}(2x - y) \tag{3.4}$$

Zavedeme novou proměnnou např. $w_1 = \frac{1}{p}(2x - y)$. Celá rovnice vypadá takto

$$py = px + (3x - 2y + w_1) \tag{3.5}$$

Nyní opět budeme integrovat obě strany rovnice. Po integraci zavedeme novou proměnnou např. $w_2 = \frac{1}{p}(3x - 2y + w_1)$ a dostáváme výslednou soustavu 2 diferenciálních a jednu algebraickou rovnici.

$$\begin{aligned} y &= x + w_2 && (3.6)\\ w_1' &= 2x - y && (3.7)\\ w_2' &= 3x - 2y + w_1 && (3.8) \end{aligned}$$

3.1.2 Metoda snižování řádu derivace

Tato metoda využívá substituci pro převod na soustavu rovnic prvního řádu. Není použitelná v situacích, kdy vstupní veličina se v rovnici vyskytuje v derivaci.

Jako příklad uvádím diferenciální rovnici druhého řádu.

$$p^2y - 3py + 2y = x \tag{3.9}$$

Zavedeme pomocné proměnné $w_0 = y$, $w_1 = py$ a $w_2 = p^2y$. Dostáváme soustavu dvou diferenciálních rovnic a jedné algebraické rovnice.

$$\begin{aligned} w_2 &= x + 3w_1 - 2w_0 && (3.10)\\ w_1' &= w_2 && (3.11)\\ w_0' &= w_1 && (3.12) \end{aligned}$$

3.2 Rozdělení numerických metod

Nyní uvedu některé základní metody, které je možné použít jako alternativu k metodě **taylorova rozvoje**, a některé jejich výhody i nevýhody.

Časové okamžiky, pro které počítáme odpovídající hodnotu, nevybíráme náhodně. Rozdíl dvou po sobě jdoucích bodů na časové ose

$$t_{i+1} - t_i = h \tag{3.13}$$

se nazývá **integrační krok**. Integrační krok ovlivňuje přesnost výpočtu. Pro příliš velký integrační krok může dojít k nestabilitě metody a následnému rozkmitání celého systému. Naopak při použití příliš malého integračního kroku roste chyba aritmetických výpočů. Ideální situace je, když integrační krok není po celou dobu výpočtu konstantní, ale přizpůsobujeme jeho velikost počítané funkci.

Numerické metody lze rozdělit na

- **jednokrokové** – počítají novou hodnotu pouze z předchozí hodnoty. Obecně bývají méně přesné než vícekrokové metody.
- **vícekrokové** – počítají novou hodnotu z několika předchozích hodnot. Obecně jsou přesnější než jednokrokové, ale je nutné použití speciálního postupu pro výpočet prvních několika členů.

3.2.1 Eulerova metoda

Je jednoduchou jednokrokovou metodou. Využívá první dva členy Taylorova rozvoje.

$$y_{i+1} = y_i + hy'_i \tag{3.14}$$

Když se nad touto metodou zamyslíme z hlediska geometrie, zjistíme, že pravá strana je rovnicí přímky, kde y'_i je směrnice tečny v bodě t_i.

Dosahuje rozumné přesnosti pro malé h. Výhodou je, že používá pouze první derivaci, proto lze zadanou rovnici prvního řádu jednoduše dosadit do vzorce. Další podrobnosti lze nalézt v literatuře [3, 6].

3.2.2 Metody Runge-Kutta

Je důležitá skupina jednokrokových metod. Obecný tvar Runge-Kuttovy metody je

$$y_{i+1} = y_i + h(w_1k_1 + \cdots + w_sk_s) \tag{3.15}$$

- $k_1 = f(x_i, y_i)$
- $k_n = f(x_i + \alpha_n h, y_i + h\sum_{j=1}^{n-1} \beta_{nj}k_j), \quad n = 2, \cdots, s$
- konstanty w_i, α_i a β_{nj} jsou vhodně volené konstanty pro maximální řád metody.

Popis metod Runge-Kutta lze najít v literatuře [3, 6].

3.2.3 Metoda Adams-Bashforth

Uvádím zde tuto metodu pouze pro doplnění příkladu vícekrokové metody. Počítá následující hodnotu ze čtyř předchozích podle vzorce

$$y_{n+1} = y_n + \frac{h}{24}(55f_n - 59f_{n-1} + 37f_{n-2} - 9f_{n-3}) \tag{3.16}$$

Problém nastává při spuštění výpočtu, kdy je nutné použít jednokrokovou metodu.

3.3 Taylorova řada

Zadáním této bakalářské práce je použití **taylorova rozvoje**. Obecný vzorec taylorovy řady v bodě i je

$$y_{i+1} = y_i + hy'_i + \frac{h^2}{2!}y_i^{(2)} + \frac{h^3}{3!}y_i^{(3)} + \cdots + \frac{h^n}{n!}y_i^{(n)} \tag{3.17}$$

Největší nevýhodou této metody je nutnost znalosti vyšších derivací funkce.

Alternativní a lépe názorný zápis **Taylorova rozvoje** je

$$y_{i+1} = DY0_i + DY1_i + DY2_i + DY3_i + \cdots + DYp_i \tag{3.18}$$

Výhody tohoto zápisu vysvětlím níže v textu. Zdůrazňuji, že tento zápis je pro vlastní řešení klíčový.

3.3.1 Příklad použití Taylorovy řady

Mějme jednoduchou rovnici

$$y' = y \tag{3.19}$$
$$y(0) = y_0 \tag{3.20}$$

Podle zadání platí, že

$$y = y' = y'' = y''' = \cdots = y^{(n)} \tag{3.21}$$

Celý **taylorův rozvoj** bude vypadat takto:

$$y_{i+1} = y_i + hy_i + \frac{h^2}{2!}y_i + \cdots + \frac{h^n}{n!}y_i \tag{3.22}$$

Nyní můžeme využít výhodu zápisu (3.18) a zapsat jednotlivé členy jako

$$DY0_i = y_i \tag{3.23}$$
$$DY1_i = hDY0_i = hy_i \tag{3.24}$$
$$DY2_i = \frac{h}{2}DY1_i = \frac{h^2}{2!}y_i \tag{3.25}$$
$$DY3_i = \frac{h}{3}DY2_i = \frac{h^3}{3!}y_i \tag{3.26}$$
$$\vdots \tag{3.27}$$
$$DYp_i = \frac{h}{p}DY(p-1)_i = \frac{h^p}{p!}y_i \tag{3.28}$$

V zápisu jednotlivých členů **taylorovy řady** můžeme vidět, že je možné spočítat n-tý člen rekurentně pomocí členu předchozího. V této vlastnosti tkví síla **taylorovy řady**. Je totiž možné získat rekurentní vztah z každé rovnice.

Problematiku by měl přiblížit trochu složitější příklad. Mějme rovnici

$$y' = \sin t, \quad y(0) = 1 \tag{3.29}$$

Přímé řešení této rovnice by bylo velmi složité při získávání vyšších derivací. Proto uděláme substituci $z = \sin t$ a spočítáme si derivaci funkce z.

$$z = \sin t \tag{3.30}$$
$$z' = \cos t \tag{3.31}$$

Dále musíme dopočítat počáteční podmínku, kterou získáme dosazením do následující rovnice.

$$z = sin(t) \tag{3.32}$$
$$z(0) = sin(0) \tag{3.33}$$
$$z(0) = 0 \tag{3.34}$$

Rovnice $z' = \cos t$ má počáteční podmínku

$$w = cos(t) \tag{3.35}$$
$$w(0) = cos(0) \tag{3.36}$$
$$w(0) = 1 \tag{3.37}$$

Řešení by opět bylo moc složité, proto zavedeme další substituci $w = \cos t$. Budeme opět potřebovat i první derivaci.

$$w = \cos t \tag{3.38}$$
$$w' = -\sin t \tag{3.39}$$
$$w' = -z \tag{3.40}$$

Důležité je, že máme rekurentní vztah, takzvanou zpětnou vazbu systému. Vznikla nám soustava tří diferenciálních rovnic.

$$y' = z, \quad y(0) = 1 \tag{3.41}$$
$$z' = w, \quad z(0) = 0 \tag{3.42}$$
$$w' = -z, \quad w(0) = 1 \tag{3.43}$$

Řešení soustavy můžeme zapsat pomocí tří **taylorových rozvojů**.

$$y_{i+1} = DY0_i + DY1_i + DY2_i + \cdots + DYp_i \tag{3.44}$$
$$z_{i+1} = DZ0_i + DZ1_i + DZ2_i + \cdots + DZp_i \tag{3.45}$$
$$w_{i+1} = DW0_i + DW1_i + DW2_i + \cdots + DWp_i \tag{3.46}$$

Jednotlivé členy můžeme spočítat jako

$$DY0_i = y_i \tag{3.47}$$
$$DY1_i = hDZ0_i \tag{3.48}$$
$$DY2_i = \frac{h}{2} DZ1_i \tag{3.49}$$
$$DY3_i = \frac{h}{3} DZ2_i \tag{3.50}$$
$$\vdots \tag{3.51}$$
$$DYn_i = \frac{h}{n} DZ(n-1)_i \tag{3.52}$$

Členy řady pro výpočet funkce z jsou tyto:

$$DZ0_i = z_i \tag{3.53}$$
$$DZ1_i = hDW0_i \tag{3.54}$$
$$DZ2_i = \frac{h}{2} DW1_i \tag{3.55}$$
$$DZ3_i = \frac{h}{3} DW2_i \tag{3.56}$$
$$\vdots \tag{3.57}$$
$$DZn_i = \frac{h}{n} DW(n-1)_i \tag{3.58}$$

A konečně pro funkci w:

$$DW0_i = w_i \tag{3.59}$$
$$DW1_i = -hDZ0_i \tag{3.60}$$

$$DW2_i = -\frac{h}{2}DZ1_i \tag{3.61}$$

$$DW3_i = -\frac{h}{3}DZ2_i \tag{3.62}$$

$$\vdots \tag{3.63}$$

$$DWn_i = -\frac{h}{n}DZ(n-1)_i \tag{3.64}$$

Z rovnic vyplývá, že nám odpadla nevýhoda obecného řešení, kde neznáme vyšší derivace. Také se nám omezila chyba výpočtu pouze na aritmetickou chybu při zaokrouhlování a na chybu způsobenou integračním krokem h.

Další obrovskou výhodou, která je patrná ze vzorců, je možnost paralelního výpočtu. Vidíme, že máme soustavu tří rovnic, kde každá rovnice by mohla být řešena na vlastním procesoru. Docházelo by k výměně hodnot jednotlivých členů řady, poté by došlo k výpočtu dalšího členu řady, který by byl opět distribuován atd.

Kapitola 4

Analýza požadavků programu

Mým hlavním cílem bylo navrhnout takový systém, ve kterém je oddělena vlastní aritmetika od řídících algoritmů.

Budeme požadovat po programu, aby byl schopen

- *načíst soustavu diferenciálních rovnic*
- *změnit integrační krok*
- *změnit čas ukončení výpočtu*
- *spustit výpočet a posílat výsledky*
- *přerušit výpočet v libovolném okamžiku*

Tento návrh předpokládá zadání soustavy lineárních diferenciálních rovnic prvního řádu ve tvaru $y' = ay$.

Změna integračního kroku a času ukončení výpočtu mohou vést k zpřesnění výpočtu.

Spuštění výpočtu způsobí řešení zadané soustavy. Současně by měl být na výstupu kromě výsledků i čas, pro který jsou platné vypočítané hodnoty. Přerušení výpočtu v libovolném okamžiku je vhodné pro ladění programu.

Nyní bych pohovořil k vlastní architektuře mikrokontroléru. Zvolili jsme mikrokontrolér firmy Motorola řady HC08. To je osmibitový stroj pracující na M68HC05 architektuře. Jeho frekvence sběrnice je 8MHz. Pro vývoj programu je ale podstatná paměť RAM, které má 512 bytů a paměť ROM, které je k disposici 12KB. Další podstatnou informací je, že procesor je typu big-endian, tedy že nižší významové byty ukládá k vyšším adresám.

Mikrokontrolér disponuje komunikačním modulem SCI a SPI, tedy sériové a paralelní rozhraní umožnující navázat kontakt s okolním světem. Je schopen adresovat až 64KB díky 16bitovému adresovému registru.

Podrobný popis architektury mikrokontroléru HC08 je v literatuře [5].

Tato platforma je vhodným příkladem extrémní architektury, tedy takové architektury, která má nějaká omezení, např. málo paměti. aj. Experimentujeme s **taylorovou řadou** na této platformě ne kvůli praktickému využití, ale protože chceme ukázat, jednoduchost tohoto algoritmu. Je schopný počítat relativně přesně i právě na HC08. V budoucnu je v plánu implementace celého algoritmu do hardwaru, proto je hlavním vývojovým jazykem assembler. Assembler je jazyk, kde píšeme přímo instrukce pro procesor.

Kapitola 5

Návrh řešení

Kdybych shrnul požadavky na program, tak nejprve budeme muset zhodnotit vlastní architekturu a přizpůsobit naše požadavky jejím možnostem. Dále budeme muset navrhnout vhodnou reprezentaci čísel a aritmetické operace, které nad nimi budeme provádět.

Pro komunikaci využijeme modul SCI, který je možné připojit k sériovému portu na PC abychom mohli celý výpočet snadno řídit, např. programem *Hyperterminál*. Zadávání dat by mohlo být uděláno v režimu dotazování (polingu) a požadované ukončení běhu programu v průběhu výpočtu by mělo být s využitím přerušení.

Abychom mohli zadávat jednotlivé diferenciální rovnice, měl by být přítomen i překladač pro převod rovnice do tvaru, jemuž rozumí vlastní program.

Po zvážení těchto požadavků mohu navrhnout hrubé rozdělení do základních programových modulů. To je důležité při psaní v jazyce assembler, aby bylo možné výsledný program udržovat a rozšiřovat.

- **výpočetní** modul
- **komunikační** modul
- **řídící** modul

Existenci *výpočetního* a *řídícího* modulu jsem udělal z důvodu nezávislosti vlastního algoritmu **taylorovy řady** na použité reprezentaci čísel. Dále jsem chtěl od sebe oddělit nízkoúrovňové a vysokoúrovňové operace.

5.1 Reprezentace čísel

Abychom mohli vůbec provádět výpočty, je nutné zvolit vhodnou číselnou reprezentaci, tedy dát jednotlivým bitům sémantiku.

Jelikož mikrokontrolér HC08 má pouze 512 bytů RAM paměti, zvolil jsem délku čísla na 32 bitů. Čísla jsou v pevné řádové čárce v doplňkovém kódu, kde nejvyšší bit představuje znaménko, následujících 7 bitů představuje celou část čísla a zbylých 24 bitů představuje desetinnou část.

Toto číslo bude mít přesnost maximálně na 6 desetinných míst. Před desetinnou čárkou figuruje 7 bitů, které dodávají rozsah od 0 do 127. S přihlédnutím k reprezentaci tedy dostáváme rozsah čísel od -128.000000 do $127,999999$. Tento rozsah by měl stačit na jednoduché experimenty.

Zbývá navrhnout, jak daná čísla předávat a používat. Šířka čísla je 32 bitů a to je moc pro předávání čísel hodnotou. I ukazatel má 16 bitů a není vhodný pro předávání. Jde mi o to, že při volání funkce by se kopírovalo velké množství dat na programový zásobník. Docházelo by ke

zbytečným přesunům dat z paměti do zásobníku a potom následné čtení ze zásobníku. Podobné je to i v případě ukazatelů do paměti. V extrémním případě by mohlo dojít k zahlcení zásobníku a následnému pádu programu.

Asi nejlepším řešením je použití takzvaných rukojetí (anglicky “handle”). Dále v textu budu používat termín **odkaz**. To je osmibitová hodnota, kterou můžeme ukázat na jedno číslo. Nevýhoda tohoto přístupu spočívá v tom, že je nutné udělat správu paměti. O správě paměti se zmíním dále. Zatím budeme používat pro reprezentaci jednoho čísla odkaz, který zabírá v paměti právě jeden byte.

5.2 Architektura HC08

V této kapitole bych chtěl popsat přizpůsobování cílové architektury potřebám programu. Hlavně jde o rozdělení paměti RAM.

5.2.1 Rozdělení paměti RAM

Po přečtení manuálu [5], zjistíme, že po startu a nastavení programového zásobníku zabírá tento 192 bytů RAM paměti. Pro potřeby programu bude stačit polovina, tj. 96 bytů paměti RAM.

Další paměť bude potřebovat komunikační modul. Pro uložení právě načtených dat jsem vyhradil 40 bytů paměti RAM a pro vyrovnávací paměť výstupních dat je k dispozici 8 bytů paměti RAM. Dále budou potřeba 2 příznakové byty. Velikost tohoto modulu se ustálila na 50 bytech.

Největší část paměti bude zabírat modul, který se stará o uložení čísel. Modul správy paměti. Pro hodnoty jsem vyhradil 264 bytů a pro funkci správy paměti je potřeba ještě dalších 17 bytů dat.

Řídící modul využívá pro svoje potřeby 54 bytů paměti RAM. Uschovává v sobě převážně odkazy na vypočítané hodnoty a struktury reprezentující řešené rovnice.

Aby bylo jednodušší provádění výpočtů, uchýlil jsem se k použití tzv. virtuálních registrů. Virtuální registr je místo v paměti, do kterého mohu krátkodobě odložit data. Celkem jsem navrhl tři osmibitové a tři šestnáctibitové virtuální registry. V globálním pohledu na program je potřeba ještě uložit další data. Mezi ně patří **slabika chyb** a **příznakové slovo**. V součtu potřebujeme dalších 12 bytů RAM paměti.

Ve zkratce shrnu využití paměti RAM

- 96 bytů programový zásobník
- 50 bytů komunikační modul
- 264 bytů pro uložení číselných hodnot
- 17 bytů pro správu paměti
- 54 bytů pro řídící modul
- 12 bytů pro virtuální registry a stavové informace

Pokud sečteme všechny požadavky, dostáváme celkem 493 bytů. To je 96% využití paměti RAM. Zde by se mohla vést zajímavá diskuse, zda by bylo možné napsat tento program ve vyšším programovacím jazyce při takovém využití systémových prostředků.

5.2.2 Rozdělení paměti ROM

Paměť ROM (read only memory) slouží k uchování konstantních dat a programu. Celkem je k dispozici 12KB paměti ROM na platformě HC08.

Rozhodl jsem se uchovávat v této paměti kromě vlastního programu i číselné konstanty ve stejném formátu jako v paměti RAM, tedy 32 bitů délka, dolních 24 bitů pro desetinnou část a horních 8 bitů pro znaménko a celou část. Je nutné také zohlednit transparentní použití číselných hodnot jak z paměti RAM, tak z paměti ROM. Tuto problematiku bude řešit modul správy paměti.

Dále zde budou uloženy textové řetězce usnadňující komunikaci mezi uživatelem a mikrokontrolérem.

5.2.3 Ostatní nastavení

Po spuštění programu se provede zápis do konfiguračních registrů mikrokontroléru. Ve zkratce mohu říct, že není potřeba podpora obvodů COP, LVI a ani dalších speciálních obvodů. Program používá standardní konfiguraci. Jediný hardwarový modul, který je používán, je SCI jednotka pro komunikaci. Rychlost komunikace máme nastavenou na 9600 Baudů. Komunikujeme po 8 bitech bez parity a jedním stopbitem.

5.3 Modul správy paměti

Tento modul stojí v samotných základech výpočetního algoritmu. Umožňuje nám dívat se na čtyřbytové úseky paměti jako na jedno číslo. Jeho úkolem je poskytovat volnou paměť podle potřeby. Každé číslo lze identifikovat jednoznačnou osmibitovou hodnotou. Ta musí zapouzdřovat jak čísla uložená v paměti ROM, tak čísla uložená v paměti RAM. Dále musí uchovávat informace o tom, která paměť je již využívána, a která je volná pro použití ve výpočtu.

5.3.1 Popis odkazu na číslo

Data všech hodnot jsou uložena ve dvou souvislých blocích paměti. První blok je v paměti RAM a druhý je v paměti ROM. Pro nás je podstatné, že se na tyto bloky můžeme dívat jako na pole 32 bitových hodnot. Abychom mohli přistoupit do pole, musíme znát index. My máme celkem dvě pole, tudíž potřebujeme znát index a ještě je nutné vědět, do které paměti máme přistupovat.

Máme pouze dvě pole, a tak nám bude stačit jeden bit pro identifikaci pole. Zbylých sedm bitů může představovat index. Obecně lze jakýkoli odkaz zapsat takto:

M I I I I I I I

- **M** – představuje typ paměti nebo také blok do kterého se bude přistupovat
- **I** – index do paměti určené **M**.

Sedm bitů stačí, protože každé číslo má délku čtyři byty a celkově lze naadresovat až dvakrát 512 bytů paměti.

Odkaz má i jednu speciální hodnotu. Tato hodnota představuje prázdný nebo neplatný odkaz a všechny její bity jsou nulové.

Tento odkaz má takovou hodnotu:

0 0 0 0 0 0 0 0

5.3.2 Stavové slabiky správy paměti

Tento modul musí uchovávat stavové informace. Musí vědět kolik čísel lze uložit, kde v paměti začínají vlastní hodnoty, které číslo je volné a které je obsazené. K těmto úkolům slouží stavové slabiky. Jsou to tyto tři:

- *počet* – představuje maximální počet čísel, které je možné současně uložit. Tato hodnota je využita při paměťové aritmetice.
- *adresa* – 16 bitová hodnota reprezentující adresu do paměti RAM, kde je uloženo první číslo.
- *bitové pole*. O každém čísle je možné prohlásit, zda je využité nebo není. To je jednobitová informace. Proto je možné použít bitové pole, kde každý bit představuje jedno číslo a pozici v bitovém poli je možné dopočítat z odkazu.

5.3.3 Základní funkce modulu

Úkoly tohoto modulu jsou:

- **Získání volného čísla** – prochází bitovým polem dokud nenarazí na volné číslo, nebo dokud nedorazí na konec. Jakmile najde volné číslo, vytvoří nový odkaz a toto číslo zabere. Jestliže není volná paměť k dispozici program na to upozorní. Výpočet nebude možný.
- **Uvolnění použitého čísla** – pouze vymaže příznak v bitovém poli.
- **Získání adresy čísla** – obecně lze spočítat 16 bitovou adresu do paměti z odkazu na číslo podle vzorce

$$M_{16} = BASE_{16} + ((REF_8 \quad and \quad [01111111]_2))_{16} * 4 \tag{5.1}$$

  Výraz $BASE_{16}$ představuje bázovou adresu, kterou získáme z nejvyššího bitu REF_8. Nejprve vynulujeme nejvyšší bit v odkazu, potom celý odkaz rozšíříme na 16 bitů a vynásobíme šířkou čísla, tedy čtyřmi.

5.4 Komunikační modul

Tento modul se stará o komunikaci mezi PC (uživatelem) a řídícím jádrem celého programu. Výměna dat probíhá po sériové lince. Uživatel zadává na terminál příkazy, které jsou přeneseny do mikrokontroléru a systém na ně musí nějak zareagovat.

Komunikace se fyzicky odehrává na dvou úrovních

- **vstupní** – zahrnuje příkazy, které zadává uživatel. Tento modul je rozpozná a předá jádru výpočtu.
- **výstupní** – představuje výstup dat zpět na terminál zahrnujíce výsledky výpočtu i reakce na každý příkaz.

Z hlediska abstrakce je lepší se dívat na tuto komunikaci jako na datové proudy známé z vyšších programovacích jazyků. Tento pohled je totiž obecnější a vhodnější pro pozdější rozšiřování programu.

5.4.1 Vstupní proud dat

Zde je využíváno jak režimu polingu (dotazování), tak režimu přerušení. Stav programu bychom mohli nazvat *čekání na příkazy*, kde se využívá režim dotazování, protože přerušení není potřeba a v průběhu *výpočtu* se používá režim přerušení.

Vstupní paměť je velká 40 bytů. Takto velkou vyrovnávací paměť jsem zvolil z důvodu uschování celého zadání diferenciální rovnice. Kvůli bezpečnosti jsem se rozhodl pro využití pouze 39 bytů. Čtyřicátý byte je nastaven vždy na hodnotu 0. Je to ochrana proti nechtěnému čtení mimo vyhrazenou vyrovnávací paměť.

Pro analýzu vstupních dat potřebujeme následující tři příznaky, které jsou uloženy v jednom bytu. Jejich rozdělení v paměti je následující

$$D\ D\ D\ D\ D\ D\ P\ N \tag{5.2}$$

- **načtení N** – příkaz je načten.
- **přetečení P** – příkaz je větší než vyrovnávací paměť.
- **délka D** – délka příkazu v bytech. Teoreticky může délka vyrovnávací paměti být až 64 bytů.

Při načtení znaku z SCI se provede následující analýza:

1. Znak se porovná se znakem odpovídající klávese `Enter`. Jestliže se jedná o klávesu `Enter`, nastaví se příznak načtení příkazu. Jinak se skočí na bod *3*.
2. Program zjistí, jestli nedošlo dříve k přetečení, protože potom by příkaz pravděpodobně byl nekompletní a chybný. V opačném případě je příkaz uložen v paměti celý a je nutné provést jeho analýzu. Rozborem každého příkazu se podrobně zabývám níže.
3. Program zjistí, jestli je ve vyrovnávací paměti ještě volné místo. Jestliže toto volné místo existuje, znak se uloží. Inkrementuje se příznak obsazené délky vyrovnávací paměti. V opačném případě nemůže být znak uložen. Dojde k zahození tohoto znaku a nastavení příznaku přetečení. Následuje předání řízení řídícímu modulu.

5.4.2 Analýza příkazu

Každý příkaz se skládá ze dvou částí. V první části je konstantní kód příkazu, který přesně identifikuje o jakou instrukci se jedná. Druhou částí je parametr.

Vlastní zpracování parametrů příkazů provádí řídící modul, který pro každý příkaz poskytuje metodu pro zpracování. Komunikační modul zjistí o který příkaz se jedná a zavolá obslužnou funkci. V paměti ROM jsou uloženy informace o každém příkazu. Nejprve je délka textové části v bytech, která nám říká, kolik bytů maximálně se má porovnávat. Následuje ukazatel na funkci, která se má zavolat v případě rozpoznání tohoto příkazu, a na závěr je textová podoba příkazu. Tímto způsobem je zajištěna hromadnost algoritmu vyhledávání příkazu a následná reakce.

Jelikož je textová podoba příkazu různě dlouhá, bylo nutné ještě vytvořit mapovací pole všech příkazů, kde jsou uloženy jejich počáteční adresy. Mimo to se paměti nachází délka tohoto pole. Kdybychom chtěli přidat další příkaz, stačilo by přidat novou strukturu obsahující délku příkazu v bytech, adresu metody, která jej obsluhuje a textový zápis příkazu. Následně bychom měli přidat záznam do mapovacího pole a zvětšit jeho délku o jedna. Systém porovnávání se o vše již postará sám.

5.4.3 Popis příkazů

Celkem jsem implementoval dvanáct příkazů, jimiž lze ovládat řešení rovnic. Jejich počet nemusí být konečný. V následujícím výčtu bych je chtěl všechny popsat.

- `RESET ALL` – provede vymazání všech rovnic a uvede procesor do stavu po spuštění. Tento příkaz nemá žádný parametr.
- `ADE` – přidá novou rovnici. Původní znění bylo `ADD EQ`, ale z důvodu velikosti vstupní vyrovnávací paměti musím šetřit každý byte a musel jsem zkrátit tento příkaz na tři znaky. Po tomto příkazu je jako parametr rovnice. Například rovnici $y' = 35.0y, \quad y(0) = 0.0$ zapíšeme jako `ADE y'=35.0y&0.0`. Syntaxe je stejná jakou používá jazyk v programu TKSL, abych nemusel vymýšlet nový jazyk se stejnou sémantikou. Ještě bych doplnil, že před každou proměnnou musí být nějaký koeficient, ikdyž se jedná o číslo 1.0. Co se řídících proměnných týká, tak možné jsou pouze tyto čtyři: `x,y,z,w`. Proč pouze tyto čtyři bude vysvětleno v kapitole o řídícím modulu.
- `REMOVE EQ=` odstraní rovnici, která byla zadána. Má jeden parametr – proměnnou – kterou je uvozena rovnice. Například `REMOVE EQ=y` odstraní ze seznamu rovnici, která začíná `y'=`. . . .
- `SET TMAX=` nastaví čas, kdy má skončit výpočet. Požaduje jeden parametr a to je číslo. `SET TMAX=10.0` nastaví čas ukončení výpočtu na 10.0.
- `SET H=` tímto příkazem nastavíme integrační krok. V současnosti je parametr konkrétní číselná hodnota. Co se týká možného vývoje do budoucna, tak bych mohl implementovat umělou inteligenci, která by si přizpůsobovala velikost integračního sama, čímž by docházelo k zvýšení přesnosti výpočtu.
- `SET INPUTFORMAT=`, `SET OUTPUTFORMAT=` nastavuje vstupní resp. výstupní formát čísel pro všechny příkazy. Jako parametr mohou figurovat tyto tři textové řetězce: `BIN` pro binární podobu, `DEC` pro desítkovou podobu a `HEX` pro šestnáctkovou podobu.
- `ENABLE OUTPUT=`, `DISABLE OUTPUT=` povoluje, nebo zakazuje programu provádět výstup určité proměnné. Představme si, že řešíme soustavu rovnic a zajímá nás konkrétní jedna proměnná, např. y. Ostatní proměnné x a z nás nezajímají. Provedeme následující příkazy.

  1. `ENABLE OUTPUT=y`
  2. `DISABLE OUTPUT=x`
  3. `DISABLE OUTPUT=z`

  Po přidání rovnice je implicitně povolen výstup jejích hodnot.
- `HALT` – tímto příkazem je možné okamžitě ukončit běžící výpočet.
- `RUN` – aby byl výpočet spuštěn, použijeme tohoto příkazu.

Na závěr tohoto popisu bych chtěl zdůraznit, že cílem této práce bylo navrhnout systém řešení rovnic, nikoliv zcela bezpečný překladač příkazů. Systém je zabezpečen v rozumné míře, např. při zadávání příkazů nemůžete přepsat jinou paměť než vyrovnávací vstupní.

5.4.4 Výstupní proud dat

Je implementován osmibytovou vyrovnávací pamětí a jedním osmibitovým příznakem. Opět se díváme na vyrovnávací paměť jako na pole bytů a onen osmibitový příznak je index, kam se bude zapisovat příští znak. V případě, že je příznak nulový, celá vyrovnávací paměť je prázdná. Jestliže se dostane na hodnotu osm, dochází k vyprázdnění vyrovnávací paměti. Je využit režim dotazování a po odeslání všech dat dojde k vynulování příznaku.

Naskýtá se otázka proč používat vyrovnávací paměť, když je možné data přímo posílat po SCI. Tento způsob by se mohl hodit v budoucnu. Kdybychom použili například režim přerušení, mohlo by dojít k podstatnému urychlení výpočtu, protože v současnosti je výpočet brzděn posíláním dat, které by tak mohlo odpadnout.

Výstupní proud se také stará o výstup číselných hodnot. Podle nastavení výstupního formátu posílá odpovídající znaky na výstup. Popis formátů čísel bude popsán níže.

5.5 Výpočetní modul

Tento modul poskytuje základní nízkoúrovňové matematické operace. Z hlediska hierarchie jsou toto operace stojící mezi správou paměti a operacemi vysoké úrovně řídícího modulu. Všechny binární operace, které jsou používány v programu, jsou tzv. nedestruktivní. To znamená, že mají tři parametry. Načtou hodnoty ze dvou parametrů a výsledek uloží do třetího parametru. Nedochází ke zničení původní hodnoty žádného ze vstupních parametrů.

Podíváme-li se na rovnici **taylorova rozvoje** 3.17, zjistíme, že základními aritmetickými operacemi použitými jsou sčítání, násobení, umocňování, faktoriál a dělení. Ze vztahů 3.23 až 3.28 zjistíme, že umocňování i faktoriál můžeme nahradit násobením při postupném výpočtu členů řady. Co se dělení týká, vždy dělíme konstantou. Tuto vlastnost můžeme využít a v paměti ROM si můžeme uchovat reciproké hodnoty těchto konstant. Následně místo dělení budeme násobit převrácenou hodnotou, čímž jsme zjistili, že nám bude stačit pouze sčítání, násobení a pole konstant v paměti ROM. S pomocí těchto operací můžeme ale spočítat pouze členy **taylorovy řady**.

Pro výpočet budeme potřebovat ještě operaci pro porovnání dvou čísel. Kvůli této operaci bude ještě nutné implementovat funkci pro rozdíl dvou čísel a pro porovnání rovnosti na nulu.

Poslední operace týkající se aritmetiky čísel jsou inicializační operace. Je užitečné mít funkci, která inicializuje hodnotu na nulu. Kromě této operace jsou potřeba ještě konverzní operace pro převod z desítkové podoby do vnitřní reprezentace.

5.5.1 Aritmetické operace

Tyto operace jsou pro vlastní výpočet klíčové. Program řešením těchto operací stráví naprostou většinu času. Proto by bylo vhodné je implementovat do hardwaru. V podstatě se jedná pouze o tři operace, *sčítání*, *násobení* a ve velmi malé míře i *odčítání*.

Sčítání a odčítání jsou podobné operace. Implementoval jsem obě dvě z důvodu zvýšení rychlosti provádění operací. Sčítání jsem implementoval jako čtyři dílčí součty s přetečením a odečítání se provede čtyřmi dílčími rozdíly s výpůjčkou. Díky této jednoduché implementaci se jedná o velmi rychlou operaci. V průměru trvají asi 800 cyklů mikrokontroléru. Musím poznamenat, že více než polovinu z toho tvoří výpočet adres operandů a kontroly na platnost těchto operandů.

Při implementaci násobení jsem chtěl maximálně využít poskytované operace mikrokontroléru. Proto jsem využil vnitřní osmibitovou násobičku. Na každé 32bitové číslo se můžeme dívat jako na čtyřciferné číslo ve 256 číselné soustavě. Výsledek bude na dvojnásobné délce, tedy na 64 bitech. Vlastní násobení můžeme provést jako čtyři dílčí součty čtyř součinů. Vše dokresluje následující

schéma.

$$AAAA * EFGH = H * AAAA + (G * AAAA) * 256 + (F * AAAA) * 256^2 + (E * AAAA) * 256^3$$

Násobení mocninou 256 je možné provést jako bitový posun o násobek čísla osm, tedy o jeden celý byte.

Předchozí algoritmus je funkční v případě, že obě čísla jsou kladná. Proto jsem implementaci rozdělil do dvou částí. Hierarchicky bychom mohli říct, že jedna je nadřazena druhé. První část se stará o násobení dvou 32bitových kladných čísel. Toto násobení probíhá podle výše zmíněného algoritmu. Druhá část hlídá správné násobení záporných čísel. Nejprve provede kopii čísel na zásobník, aby bylo možné u záporných čísel provést změnu znaménka. Může se totiž násobit i číslem uloženém v paměti ROM, do které nelze zapisovat. U záporných čísel dojde ke změně znaménka a uložení příznaků. Spustí se první část výpočtu, která probíhá na zásobníku. Po ukončení výpočtu dochází k ořiznutí výsledné hodnoty z osmi na čtyři byty. Dochází ke ztrátě informací. Dále se podle příznaků vypočítá znaménko výsledku a potom se může změnit znaménko u výsledku. Celkem se na zásobníku potřebuje 25 bytů. Dvakrát čtyři byty jako vstupní operandy, osm bytů pro výsledek, osm bytů pro dílčí součin a jeden byte jako příznak pro výpočet znaménka.

Násobení je časově nejsložitější aritmetická operace. Má verze na HC08 potřebuje asi 3500 cyklů. Opět zahrnuje i kontroly platnosti operandů. Může dojít k chybě přetečení při ořezávání, kterou reflektuje chybová slabika.

5.5.2 Porovnávací operace

Tyto operace se využijí pro rozpoznání zda daná přesnost již stačí a dále se potřebují při porovnávání časové proměnné s maximální časovou proměnnou, čímž dojde k ukončení výpočtu.

Porovnávání čísel provádí následující dvě operace

- Porovnání dvou hodnot – provede odečtení a využívá porovnání na nulu. Jestliže výsledek je nenulový, porovnává se znaménko výsledku se znaménky vstupních hodnot.
- Porovnání na nulu – zjistí, jestli je požadovaná hodnota nulová. Tuto funkci jsem implementoval tak, aby i čísla blízká nule byla považována jako nulová. Funkce vyhodnotí číslo za nulová, když absolutní hodnota čísla bude menší než $3 * 10^{-7}$ neboli

$$|val| < 0.0000003 \tag{5.3}$$

5.5.3 Konverzní operace

Z hlediska vývoje do budoucna jsem se rozhodl umožnit zadávání hodnot čísel v různých formátech. V současnosti se jedná o tři možné formáty.

- **desítkový tvar** – číslo je zadáno ve tvaru *x.y*, kde *x* je celá část čísla a *y* je desetinná část čísla. Tento formát je nastaven pro vstupní data implicitně po restartu programu.
- **šestnáctkový tvar** – číslo se skládá ze čtyř bytů. Každý byte lze vyjádřit jako dvojciferné hexadecimální číslo. Tento formát představuje osmiciferné číslo. Pro převod čísel mezi jednotlivými formáty jsem jakou součást této práce vytvořil program, který je možné nalézt v příloze této zprávy.
- **binární tvar** – přímo kopíruje obsah paměti, kde je dané číslo uloženo. Tento způsob je vhodný především tam, kde přímo komunikuje stroj s jiným strojem. Došlo by ke snížení objemu přenášených dat a odpadla by i zpětná konverze do vnitřní podoby.

Převod z desítkové soustavy probíhá za pomoci číselných konstant uložených v paměti ROM. Tyto konstanty obsahují hodnoty 100.0 10.0 1.0 0.1 …. Při čtení textového řetězce se nejprve zjistí pozice desetinné čárky a případné znaménko. Potom dochází k násobení a sčítání mezivýsledků. Zpětný převod není zatím implementován, protože by vyžadoval operaci dělení.

Převod mezi šestnáctkovým tvarem a vnitřní reprezentací je velmi jednoduchý kvůli pevné délce čísla. Každý znak představuje hexadecimální hodnotu odpovídajícího nibblu.

5.5.4 Ostatní operace

Zde se jedná pouze o dvě operace nízké úrovně pracující přímo s jednotlivými byty.

- *kopírování* hodnoty – kopíruje hodnotu ze zdroje do cíle
- *nahrání nuly* – inicializuje cílové číslo na nulu.
- *konverze z celého čísla* – touto funkcí provedeme konverzi jednoho bytu na číslo ve vnitřní reprezentaci. Tato operace je používána při konverzi z desítkové soustavy.

5.6 Řešení chyb

Systém zachytávání chyb by mohl být považován za další modul. Za modul ho ale nemůžeme považovat, protože je propleten v celém programu. Jeho základním úkolem je podchytit chyby a upozornit o nich uživatele.

Systém celkem definuje dvanáct možných pojmenovaných chybových stavů. Podotýkám, že ne všechny chyby jsou použity a jsou zde pro možné pozdější rozšíření programu. V paměti RAM má vyhrazen jeden osmibitový příznak, kam se ukládá chybový kód. Při obsloužení chybového kódu dochází k vymazání tohoto příznaku a následnému výpisu chyby uživateli.

Následující seznam popíše možné chybové stavy. Každému jménu odpovídá číselný kód.

- `OrderTooLong` – načtený příkaz se nevejde do vstupní vyrovnávací paměti.
- `UnknownOrder` – načtený příkaz nebyl rozpoznán.
- `InvalidOperand` – aritmetické operaci byl přiřazen nulový odkaz.
- `NotEnoughMemory` – není volná paměť. Systém potřebuje uložit novou hodnotu, ale systém správy paměti poslal nulový odkaz.
- `Overflow` – v aritmetické operaci došlo k přetečení, čímž se znehodnotil výsledek.
- `Unimplemented` – tento příznak je nastaven, když dochází k volání operace, která není implementovaná. V současnosti se jedná se např. o převod čísel z vnitřní reprezentace do desítkové podoby.
- `BadHandle` – program se pokusil přistoupit na nulový odkaz.
- `HandleNotUsed` – ve výpočtu je používán odkaz, který ještě nebyl vytvořen.
- `OutOfRange` – toto chybové hlášení vytváří konverzní operace, když se převáděné číslo nevejde do rozsahu vnitřní reprezentace.
- `BadFormatEq` – byla zadána rovnice v neznámém tvaru.
- `NotValidVariable` – program rozpoznává pouze `x`,`y` ,`z`,`w` proměnné, u ostatních dochází k této chybě.

5.7 Řídící modul

Tento modul se týká řízení jak výpočtu **taylorova rozvoje**, tak celého programu. Jsou zde definovány i obsluhy všech příkazů. Nejprve bych chtěl pohovořit o omezeních, která s sebou přináší platforma HC08.

5.7.1 Typy řešených rovnic

Vzhledem k tomu, že paměti RAM je velice omezené množství, program může najednou uložit pouze 66 různých koeficientů. Z toho jich je potřeba pro řešení rovnic čtrnáct. Zbytek je možné použít jako koeficienty rovnic. Rozhodl jsem se implementovat řešení soustavy až čtyř diferenciálních lineárních rovnic, pro které používám čtyři proměnné – x, y, z a w. Dohromady může být využito až šestnáct koeficientů a potřeba jsou počáteční hodnoty pro každou rovnici a zároveň vyhrazené místo pro nově spočítané řešení.

Kromě těchto koeficientů jsem zahrnul ještě časovou proměnnou, která je podstatná pro ukončení výpočtu, dalším koeficientem je čas ukončení výpočtu a posledním důležitým koeficientem je integrační krok.

Zbylo k dispozici ještě asi 25 volných hodnot, z nichž některé mohou být využity pro výpočet.

5.7.2 Analyzátor rovnic

Nejprve bych pohovořil o způsobu uložení rovnic v paměti a tvaru, v jakém je program schopen tyto rovnice řešit. Každá proměnná má vlastní odkaz (ten je jiný od číselného odkazu) tedy osmibitovou hodnotu, která jednoznačně tuto proměnnou určuje. Tento odkaz také ukazuje na jednu z těchto čtyř rovnic.

$$x' = a_1x + b_1y + c_1z + d_1w \tag{5.4}$$
$$y' = a_2x + b_2y + c_2z + d_2w \tag{5.5}$$
$$z' = a_3x + b_3y + c_3z + d_3w \tag{5.6}$$
$$w' = a_4x + b_4y + c_4z + d_4w \tag{5.7}$$

V paměti jsou uloženy celkem čtyři rovnice. Ty jsou zaznamenány jako jednoduché paměťové struktury. Každá struktura má velikost 10 bytů. První dva byty jsou odkazy na čísla, kde prvním číslem je nově počítaná hodnota a druhým číslem je předchozí hodnota. Před započetím výpočtu je zde uschována počáteční podmínka. Za těmito odkazy následují čtyři dvojice, kde první byte ve dvojici je odkaz na koeficient a druhý byte je odkaz na rovnici.

Jak můžeme vidět, zápis této soustavy rovnic není hromadný. Může být zapsáno omezené množství rovnic, ale věřím, že vzhledem k možnostem platformy HC08 to bude stačit.

Co se analýzy textové podoby rovnice týká, nepoužil jsem žádný sofistikovaný algoritmus. Dojde jednoduše k načtení první proměnné. Tak zjistíme do jaké z těchto čtyř rovnic se bude zapisovat. Poté je spouštěna konverze čísla do vnitřní reprezentace a po ní systém rozpozná jméno proměnné, se kterou se koeficient má násobit. Z tohoto důvodu je nutné zadávat vždy koeficient ikdyž se jedná o hodnotu 1.0. Cílem práce nebylo udělat plně odolný a robustní analyzátor vstupních dat, proto je tato implementace dostačující.

5.7.3 Řízení programu

Řízení probíhá ve dvou režimech. V jakém režimu se program právě nachází je možné zjistit z *příznakového slova*. Jeho první bit ve vyšším bytu odpovídá aktuálnímu režimu. V režimu nula se

čeká na vstup od uživatele, který musí zadat rovnice. V režimu jedna probíhá výpočet.

V *příznakovém slově* jsou dále uloženy běhové informace. Další dva bity odpovídají za formát vstupních dat, tedy jestli vstupní čísla budou překládána jako hexadecimální, desítková nebo jako binární. Podobné je to u dalších dvou bitů, které zodpovídají za formát výstupních dat. Pro možné pozdější rozšíření jsem vyhradil ještě jeden bit, který by měl povolit, nebo zakázat dynamickou volbu integračního kroku.

Dále má řízení programu uloženy tři číselné odkazy. Jedná se o aktuální časovou proměnnou. Časovou konstantu, která určuje ukončení výpočtu, a poslední proměnnou je integrační krok.

Pro řízení výpočtu jsou dále potřeba ještě dva příznakové byty. V prvním jsou uloženy všechny rovnice, které se účastní výpočtu. Mohu říci, že odkazy na rovnice a jim odpovídající proměnné tvoří mocninnou řadu o základu 2. Tedy proměnné x odpovídá číslo 1, proměnné y odpovídá číslo 2, proměnné z odpovídá 4, a w odpovídá 8. Díky tomuto rozdělení je teoreticky možné v programu mít až osm proměnných a k nim osm rovnic.

Poslední příznak zodpovídá za povolení výstupu. Při výpočtu nás totiž mohou zajímat pouze některé výsledky. Ostatním, které nepotřebujeme znát můžeme zakázat výstup. O tuto funkci se stará právě tento příznakový byte.

5.7.4 Princip výpočtu

Připomeňme si obecnou soustavu rovnic 5.4 až 5.7. V programu jsem naimplementoval její obecné řešení.

Základem je postupně počítat nové členy řady za pomoci předchozích členů řady. Celou rovnici můžeme zapsat pomocí čtyř **Taylorových řad**.

$$x_1 = x_0 + DX1_0 + DX2_0 + DX3_0 + \ldots \tag{5.8}$$
$$y_1 = y_0 + DY1_0 + DY2_0 + DY3_0 + \ldots \tag{5.9}$$
$$z_1 = z_0 + DZ1_0 + DZ2_0 + DZ3_0 + \ldots \tag{5.10}$$
$$w_1 = w_0 + DW1_0 + DW2_0 + DW3_0 + \ldots \tag{5.11}$$

Z tohoto zápisu je vidět, že potřebujeme hodnoty z předchozího času. Zbývá nám spočítat jednotlivé členy všech čtyř řad. Podle vzorce 3.17 můžeme odvodit druhé členy řad.

$$DX1_0 = h * x' = h * (a_1x_0 + b_1y_0 + c_1z_0 + d_1w_0) \tag{5.12}$$
$$DY1_0 = h * y' = h * (a_2x_0 + b_2y_0 + c_2z_0 + d_2w_0) \tag{5.13}$$
$$DZ1_0 = h * z' = h * (a_3x_0 + b_3y_0 + c_3z_0 + d_3w_0) \tag{5.14}$$
$$DW1_0 = h * w' = h * (a_4x_0 + b_4y_0 + c_4z_0 + d_4w_0) \tag{5.15}$$

Třetí členy řady spočítáme z druhých členů řady takto:

$$DX2_0 = \frac{h}{2} * (a_1DX1_0 + b_1DY1_0 + c_1DZ1_0 + d_1DW1_0) \tag{5.16}$$
$$DY2_0 = \frac{h}{2} * (a_2DX1_0 + b_2DY1_0 + c_2DZ1_0 + d_2DW1_0) \tag{5.17}$$
$$DZ2_0 = \frac{h}{2} * (a_3DX1_0 + b_3DY1_0 + c_3DZ1_0 + d_3DW1_0) \tag{5.18}$$
$$DW2_0 = \frac{h}{2} * (a_4DX1_0 + b_4DY1_0 + c_4DZ1_0 + d_4DW1_0) \tag{5.19}$$

A pro jistotu uvedu i čtvrté členy řady, které můžeme dopočítat ze třetích členů řady.

$$DX3_0 = \frac{h}{3} * (a_1 DX2_0 + b_1 DY2_0 + c_1 DZ2_0 + d_1 DW2_0) \quad (5.20)$$

$$DY3_0 = \frac{h}{3} * (a_2 DX2_0 + b_2 DY2_0 + c_2 DZ2_0 + d_2 DW2_0) \quad (5.21)$$

$$DZ3_0 = \frac{h}{3} * (a_3 DX2_0 + b_3 DY2_0 + c_3 DZ2_0 + d_3 DW2_0) \quad (5.22)$$

$$DW3_0 = \frac{h}{3} * (a_4 DX2_0 + b_4 DY2_0 + c_4 DZ2_0 + d_4 DW2_0) \quad (5.23)$$

Souhrn těchto vzorců nám ukazuje způsob, kterým můžeme obecně spočítat libovolný počet členů **Taylorovy řady**.

5.7.5 Implementace výpočtu

Jak můžeme vidět z předchozích vzorců 5.12 až 5.23, budeme potřebovat místo v paměti pro aktuálně počítané členy řady a pro předchozí členy řady. Toto je dohromady osm proměnných. Dále jsou potřeba ještě čtyři pomocné proměnné, protože aritmetické operace jsou tzv. nedestruktivní a je nutné jim předat volný parametr.

Celý výpočet probíhá podle následujícího algoritmu. Chtěl bych pro názornost tento popis rozdělit do dvou částí. Nejprve bych popsal řízení výpočtu.

1. *inicializace* – proběhne inicializace časové konstanty na nulu.
2. *výpočet* nových hodnot pro čas $t + h$. Bude rozebrán podrobně níže.
3. *zobrazení výsledků* – podle příznakové slabiky pro výstup jednotlivých proměnných proběhne zobrazení nejprve časové proměnné a potom všech výsledků, jejichž výstup je povolen.
4. *výpočty pro další krok* – přičtení integračního kroku k časové proměnné. Proběhne také porovnání časové konstanty s koncovým časem a případně dojde k ukončení běhu výpočtu. Posledním důležitým krokem je poznačení nově spočítaných hodnot jako staré, protože ve výpočtu potřebujeme hodnoty z předchozího kroku. Na začatku toto neproběhne, protože počáteční podmínky jsou automaticky považovány jako starší hodnoty. Tyto proměnné označíme jako `OldX`, která představuje matematicky x_{t-h} a `NewX` matematicky znamená x_t. Kvůli názornosti zde používám pouze proměnnou x. Podobné proměnné jsou i pro y, z a w.

   Pokračuje se skokem na krok 2.

Zde se budu věnovat algoritmu výpočtu. Používám zde proměnné `NewDX` a `OldDX`. Podobně i pro ostatní proměnné. Proměnnou `NewDX` používám pro výpočet nového členu řady, zatímco v `OldDX` je uložena hodnota předchozího členu řady.

Ve výpočtu se používá ještě čítač členů `I`. V následujicím textu ho pro názornost raději neuvedu, ale budeme předpokládat, že jeho hodnota začíná na nule a při výpočtu každého členu se jeho hodnota zvětšuje o jedničku.

1. *Inicializace* – Nastavíme proměnnou `NewX` na nulu. Dále musíme inicializovat `NewDX`. Podle vzorce 5.12 víme, že pouze překopírujeme hodnotu `OldX` do `NewDX`.

```
NewX = 0        NewDX = OldX
NewY = 0        NewDY = OldY
```

```
NewZ = 0      NewDZ = OldZ
NewW = 0      NewDW = OldW
```

2. *První člen řady* – proběhne přičtení `NewDX` k `NewX`. Následuje kopírování hodnoty z `NewDX` do `OldDX`.

```
NewX += NewDX     OldDX = NewDX
NewY += NewDY     OldDY = NewDY
NewZ += NewDZ     OldDZ = NewDZ
NewW += NewDW     OldDW = NewDW
```

3. *Výpočet dalšího členu řady* – Probíhá výpočet hodnoty pravé strany rovnice za pomoci `OldDX`, `OldDY`, `OldDZ` a `OldDW`. Použije se čtyř vzorců.

```
NewDX = a1*OldDX + b1*OldDY + c1*OldDZ + d1*OldDW
NewDY = a2*OldDX + b2*OldDY + c2*OldDZ + d2*OldDW
NewDZ = a3*OldDX + b3*OldDY + c3*OldDZ + d3*OldDW
NewDW = a4*OldDX + b4*OldDY + c4*OldDZ + d4*OldDW
```

4. *Násobení* – Následuje násobení integračním krokem a převrácenou hodnotou čítače, která nám říká jaký člen řady právě počítáme.

```
NewDX = H * (1/I) * NewDX
NewDY = H * (1/I) * NewDY
NewDZ = H * (1/I) * NewDZ
NewDW = H * (1/I) * NewDW
```

5. *Součet* – přičte se nově vypočítaný člen řady `NewDX` k proměnné `NewX` a překopírujeme hodnotu z `NewDX` do `OldDX`.

```
NewX += NewDX      OldDX = NewDX
NewY += NewDY      OldDY = NewDY
NewZ += NewDZ      OldDZ = NewDZ
NewW += NewDW      OldDW = NewDW
```

6. *kontrola přesnosti* – jestliže naposled vypočítaný člen nebyl nulový, pokračujeme krokem 3, jinak končíme výpočet.

Tento Algoritmus je klíčový v celé této práci. Na první pohled by se mohlo zdát, že není nejefektivnější, ale je nutné si uvědomit, že je implementován v jazyce assembler, proto budeme muset sestoupit až na nejnižší úroveň abstrakce a odpustit si výhody vyššího programovacího jazyka.

Kapitola 6

Možný vývoj do budoucna

Metoda **Taylorovy řady** je velmi perspektivní kvůli vysoké přenosti a možnosti paralelizace. Kdybych měl posoudit možnosti, které máme pro zefektivnění výpočtu. Určitě bych nejprve implementoval všechny základní instrukce do hardwaru a zvětšil paměť RAM, abychom mohli uložit čísla, která budou mít větší délku než 32 bitů. Místo formátu pevné řádové čárky bych použil raději formát plovoucí řádové čárky podle standardu IEEE 754. Možnosti využití formátu plovoucí řádové čárky můžete najít v literatuře [4].

6.1 Možná paralelizace

Řešení rozsáhlých soustav diferenciálních rovnic je možné provádět paralelně. Jako příklad paralelního algoritmu bych chtěl uvést možnou verzi tohoto programu. Pro jednoduchost uvedu pouze výpočet členů řady s inicializací a zároveň budeme předpokládat, že `Procesor 1` řeší rovnici proměnné *x*, `Procesor 2` řeší rovnici *y* atd.

Každý procesor má ještě vlastní čítač `I`, který nám říká, o jaký člen řady se jedná.

Následující příkazy se vykonávají paralelně.

	Procesor 1	Procesor 2	Procesor 3	Procesor 4
1.				
2.	`NewX = 0`	`NewY = 0`	`NewZ=0`	`NewW=0`
3.	`I = 0`	`I = 0`	`I = 0`	`I = 0`
4.	`NewDX = OldX`	`NewDY = OldY`	`NewDZ = OldZ`	`NewDW = OldW`
5.	`NewX += NewDX`	`NewY += NewDY`	`NewZ += NewDZ`	`NewW += NewDW`
6.	`OldDX = NewDX`	`OldDY = NewDY`	`OldDZ = NewDZ`	`OldDW = NewDW`
7.	Nyní musí dojít k distribuci proměnných `OldDX`, `OldDY`, `OldDZ` a `OldDW` mezi procesory.			
8.	`I += 1`	`I += 1`	`I += 1`	`I += 1`
9.	`NewDX=a1*OldDX`	`NewDY=a2*OldDX`	`NewDZ=a3*OldDX`	`NewDW=a4*OldDX`
10.	`NewDX+=b1*OldDY`	`NewDY+=b2*OldDY`	`NewDZ+=b3*OldDY`	`NewDW+=b4*OldDY`
11.	`NewDX+=c1*OldDZ`	`NewDY+=c2*OldDZ`	`NewDZ+=c3*OldDZ`	`NewDW+=c4*OldDZ`
12.	`NewDX+=d1*OldDW`	`NewDY+=d2*OldDW`	`NewDZ+=d3*OldDW`	`NewDW+=d4*OldDW`

13. `NewDX =H*NewDX  NewDY =H*NewDX  NewDZ =H*NewDZ  NewDW =H*NewDW`
14. `NewDX =NewDX/I  NewDY =NewDY/I  NewDZ =NewDZ/I  NewDW =NewDW/I`
15. `NewX += NewDX  NewY += NewDY  NewZ += NewDZ  NewW += NewDW`
16. Kontrola nově spočítaných členů řady. Jestliže dosáhly požadované přesnosti, může výpočet skončit, jinak se musí skočit na příkaz 6.

Toto je obecný popis algoritmu, který není závislý na žádné konkrétní architektuře.

6.2 Přizpůsobování integračního kroku

Výpočet můžeme zrychlit a zpřesnit s použitím dynamicky se měnícího integračního kroku. Bylo by vhodné navrhnout algoritmus přizpůsobování velikost kroku podle aktuálně vypočítaných hodnot. Například pokud by absolutní hodnota jejich rozdílu byla větší než stanovená mez, došlo by k zjemnění kroku, např. k rozpůlení a následně by probíhal výpočet podle menšího kroku. Naopak, pokud by byla absolutní hodnota rozdílu dvou po sobě jdoucích hodnot menší než jiná mez, mohlo by dojít ke zvětšení integračního kroku.

Kapitola 7

Praktický příklad použití

Na závěr bych chtěl ukázat, jak lze prakticky použít tento program pro řešení jednoduché fyzikální úlohy. Návod jak program používat a popis všech důležitých funkcí můžete nalézt v programové dokumentaci.

7.1 Rovnoměrně zrychlený pohyb

Chtěl bych porovnat řešení jednoduché fyzikální úlohy rovnoměrně zrychleného pohybu. Mějme hmotný bod, který se pohybuje po přímce s konstantním zrychlením $a = 5m/s^2$ a jeho počáteční rychlost je nulová.

Tento příklad můžeme řešit pomocí jednoduchého vzorce, který můžeme najít ve fyzikálních tabulkách.

$$s = s_0 + v_0 t + \frac{1}{2}at^2 \tag{7.1}$$

Víme, že $s_0 = 0$ a $v_0 = 0$, tudíž obecně můžeme napsat, že dráha je v čase t rovna $s = \frac{1}{2}at^2$.

Nyní stejný příklad vyřešíme s použitím programu. Víme, že

$$v = \frac{ds}{dt} = s' \tag{7.2}$$

$$a = \frac{dv}{dt} = v' = s'' \tag{7.3}$$

Můžeme napsat, že celý příklad je řešením diferenciální rovnice druhého řádu

$$s'' = a \tag{7.4}$$

Abychom mohli zadat tuto rovnici do programu, musíme ji převést do požadovaného tvaru. Použijeme metodu postupné integrace.

$$x'' = a \tag{7.5}$$

$$p^2 x = a \tag{7.6}$$

$$px = \frac{1}{p}a \quad y = \frac{1}{p}a \quad y' = a \tag{7.7}$$

$$x = \frac{1}{p}y \tag{7.8}$$

Do programu zadáme tuto soustavu dvou diferenciálních rovnic

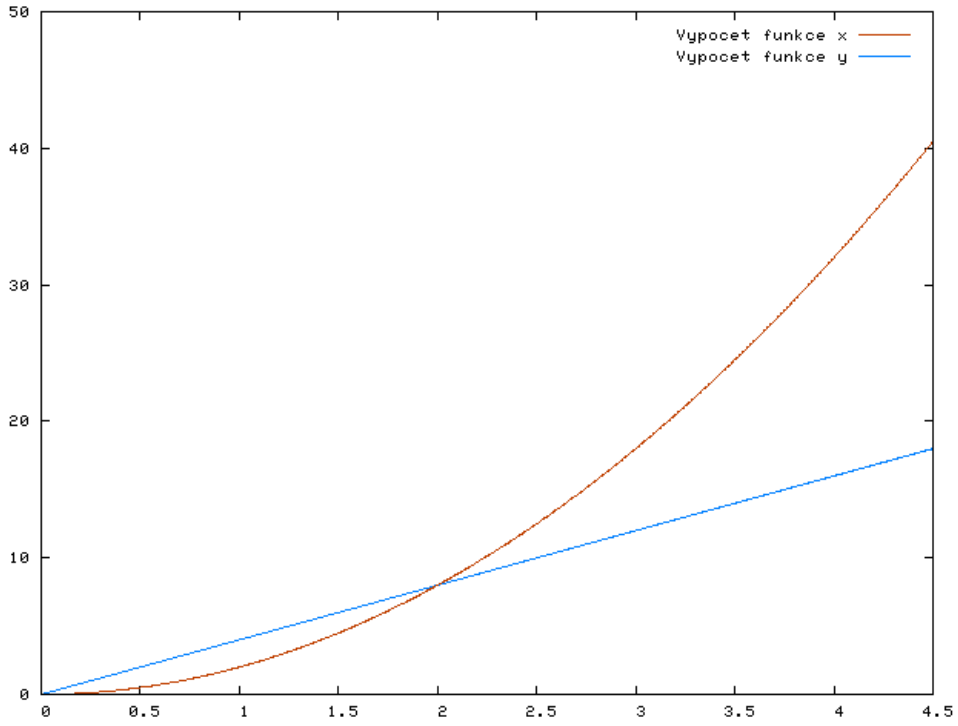
$$x' = y \tag{7.9}$$
$$y' = a \tag{7.10}$$

Obě počáteční podmínky budou nulové.

Budeme požadovat zjištění dráhy pro čas $t = 3s$. Z fyzikálního vzorce dostaneme hodnotu $s(t) = 18\,m$.

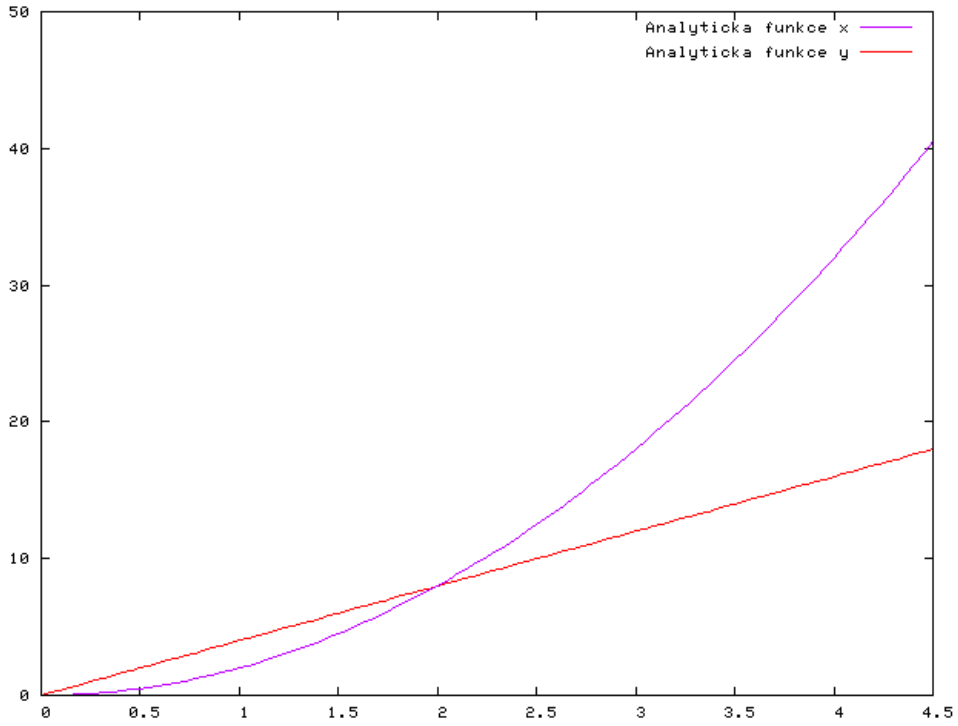
Nyní přejděme k vlastnímu zadání do programu. Nejprve nastavíme časovou konstantu a integrační krok. Výsledný graf je na obrázku 7.1.

```
SET H=0.01
SET TMAX=10.0
ADE x'=1.0y&0.0
ADE y'=4.0&0.0
```



Obrázek 7.1: Výpočet provedený programem na HC08 s $H = 0.01$

Výsledky jsou ve dvou grafech, protože v jednom obrázku se obě řešení zcela kryla. Pro náš referenční čás vyšel výsledek $s = 18,00016\,m$. Chyba je dána omezenou přesností, velikostí integračního kroku a zaokrouhlováním. Analytické řešení je na obrázku 7.2.



Obrázek 7.2: Analytické řešení příkladu zrychleného pohybu.

7.2 Vybíjení kondenzátoru

Jako druhý příklad jsem si připravil vybíjení kondenzátoru zapojeného sériově s odporem. Schéma je zakresleno na obrázku. Víme však, že kondenzátor je nabit na jeden volt. $u_C = 1V$, $R = 10^6\,\Omega$ a $C = 10^{-6}F$. Schéma je na obrázku 7.3.

Obvod popíšeme následující rovnicí:

$$0 = R\,i + u_C \tag{7.11}$$

$$i = \frac{-u_C}{R} \tag{7.12}$$

Ze základního vztahu $u_C' = \frac{1}{C}\,i$ můžeme napsat výslednou rovnici jako

$$u_C' = \frac{1}{RC}(-u_C) \quad u_C(0) = 1 \tag{7.13}$$

Po dosazení dostáváme homogenní diferenciální rovnici prvního řádu.

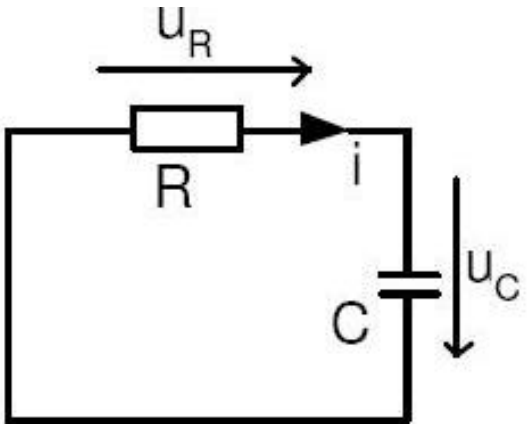
$$u_C' + u_C = 0 \quad u_C(0) = 1 \tag{7.14}$$

Analyticky vyřešíme tuto rovnici s pomocí charakteristické rovnice, která je $\lambda + 1 = 0$. Výsledek je tedy

$$u_C = Ke^{-t} \tag{7.15}$$

Z počáteční podmínky určíme konstantu $K = 1$. Výsledná funkce je tedy

$$u_C = e^{-t} \tag{7.16}$$



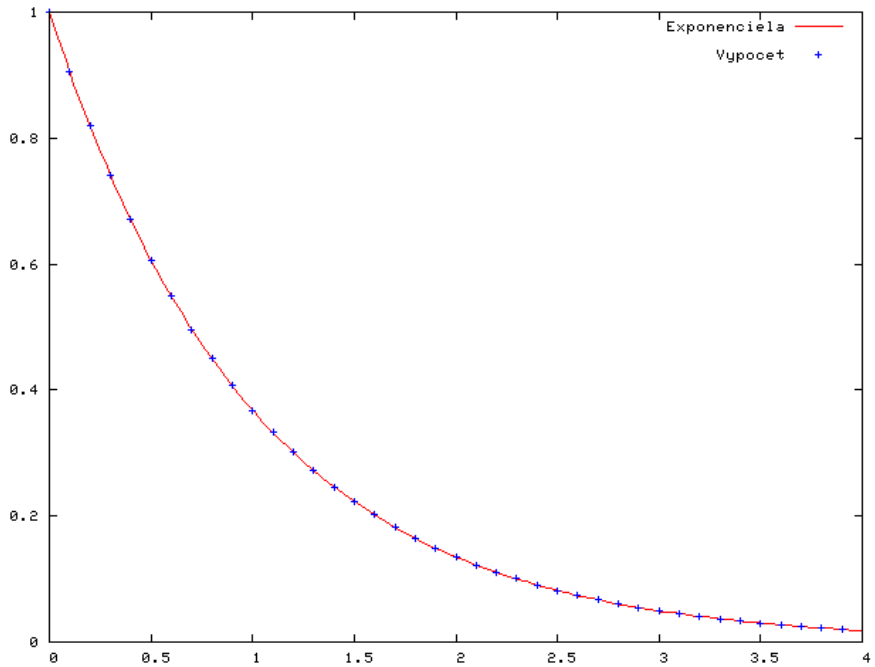
Obrázek 7.3: El. obvod

Stanovíme referenční čas $t = 2$. Po dosazení zjistíme, že $e^{-2} = 0.135335$.

Nyní zkusíme tuto úlohu vyřešit v mém programu. Zadáme následující posloupnost příkazů.

```
SET H=0.1
SET TMAX=5.0
ADE y'=-1.0y&1.0
RUN
```

Pro náš referenční čas $t = 2$ je výsledek $y(t) = 0.135336$. Modré body tvoří graf funkce počítané mikrokontrolérem a červený je graf analytického řešení $y = e^{-t}$ 7.4.



Obrázek 7.4: Graf řešení mikrokontrolérem (modře) a analytické řešení (červeně)

Kapitola 8

Závěr

Cílem této práce bylo ověřit funkčnost a jednoduchost algoritmu **taylorovy řady** v extrémních podmínkách. Výsledky byly velmi přesné. Maximální přesnost mnou navržené aritmetiky je 10^{-6} a řešení se s analytickým řešením shodovalo až do řádu 10^{-4}. Při výpočtu může být použito maximálně 16 členů **Taylorovy řady**, protože hodnoty dalších členů jsou, pro zde použitý formát čísel, nulové.

Práce slouží především jako vodítko před implementací tohoto algoritmu do hardwaru. Mým příspěvkem v této práci je, že jsem dokázal jednoduše implementovat algoritmus výpočtu, který efektivně řeší diferenciální rovnice na platformě HC08.

Co se možného vývoje do budoucna týká, předpokládám implementaci alespoň základních aritmetických operací do hardwaru a použití jiného formátu čísel, který by byl přesnější. Lákavou možností je paralelizace, díky které můžeme řešit efektivně i velmi rozsáhlé soustavy rovnic. Přemýšlet můžeme i o umělé inteligenci pro dynamické přizpůsobování integračního kroku. Více se tomuto tématu věnuje vlastní kapitola. Zároveň by se mohla rozšířit funkčnost programu tak, aby řešil i některé nelineární diferenciální rovnice.

Literatura

[1] Bartsch, H. J.: *Matematické vzorce*. Mladá fronta, 2000, iSBN 80-204-0607-7.

[2] Cifřík: Diferenciální rovnice. Technická zpráva, Pedagogická fakulta Karlovy Univerzity, 2001/2002.

[3] Fajmon, B.: *Matematika 3*. VUT FEKT, 2005.

[4] Kraus, M.: *Elementární procesor specializovaného paralelního systému*. Diplomová práce, FIT, VUT, 2006.

[5] Motorola: *M68HC08 Microcontrollers*. Rev. 2 vydání, 2002.

[6] Peringer, P.: *Studijní opora předmětu IMS*. VUT, 2007.

Použité zkratky a symboly

HC08 – typ osmibitového mikrokontroléru firmy Motorola.

COP Computer Operating Proprerly – obvody hlídající správný běh programu. V případě detekce chyby dojde k restartování programu.

LVI Low Voltage Inhibit – obvody hlídající velikost napájecího napětí.

SCI – sériové komunikační rozhraní.

SPI – paralelní komunikační rozhraní.

RAM Random Acces Memory – volatilní paměť, kam se ukládají data potřebná po dobu běhu programu.

ROM Read Only Memory – nevolatilní paměť, kde je uložen kód programu a důležité konstanty.

Seznam příloh

Dodatek A – doprovodné programy

Dodatek B – grafy různých rovnic

Dodatek A

Doprovodné programy

Aby bylo možné jednodušeji ladit hlavní program, vytvořil jsem ještě několik dalších pro PC v jazyke C#. Zde bych chtěl popsat jejich ovládání.

A.1 Program Numbers

Tento program slouží pro převod mezi číselnými reprezentacemi. Po spuštění musíme nastavit, jak dlouhé číslo používáme a kolik bitů tvoří desetinnou část. Potom si zvolíme vstupní a výstupní formát. Poté můžeme zapsat číslo do editačního řádku a následně i převést.

A.2 Program Ibp

Je program pro zpracování výsledků, které posílá mikrokontrolér. Obsahuje dva panely. Na prvním panelu je textové pole, kam se zkopírují data vyslaná mikrokontrolérem, nebo můžeme tento program přímo připojit k sériovému portu. Potom si vyberete index sloupce v editačním řádku, protože ve výstupu mohou být výsledky až čtyř rovnic. Můžete si zvolit i barvu a tlačítkem *Přečíst výsledky* necháme program vykreslit graf funkcí.

Na druhém panelu je kreslen graf. Máme zde tlačítka $<$ a $>$, kterými se můžeme posunovat po ose x, ale jednodušší je použít myš pro posun v libovolném směru. Dále jsou zde tlačítka $+$ a $-$, kterými můžeme měnit měřítko pohledu. Nad těmito tlačítky je editační řádek, kam můžeme napsat referenční funkci v prefixovém zápisu. Tlačítkem *Barva* můžeme změnit barvu výsledného grafu a tlačítko *Přidat* nám tuto funkci zobrazí. Dále se k tomu váže editační řádek *Granularita*, kde můžeme říct, kolik bodů pro jednu funkci se má generovat.

Uložit obr do souboru je další funkce pro uložení výsledku (obrázku). Speciální tlačítko *Uložit do souboru*, které nám uloží u zadané funkce (index funkce je v řádku pod tlačítkem) do souboru hodnoty osy *x* a k němu příslušející *y*.

Dodatek B

Grafy různých rovnic

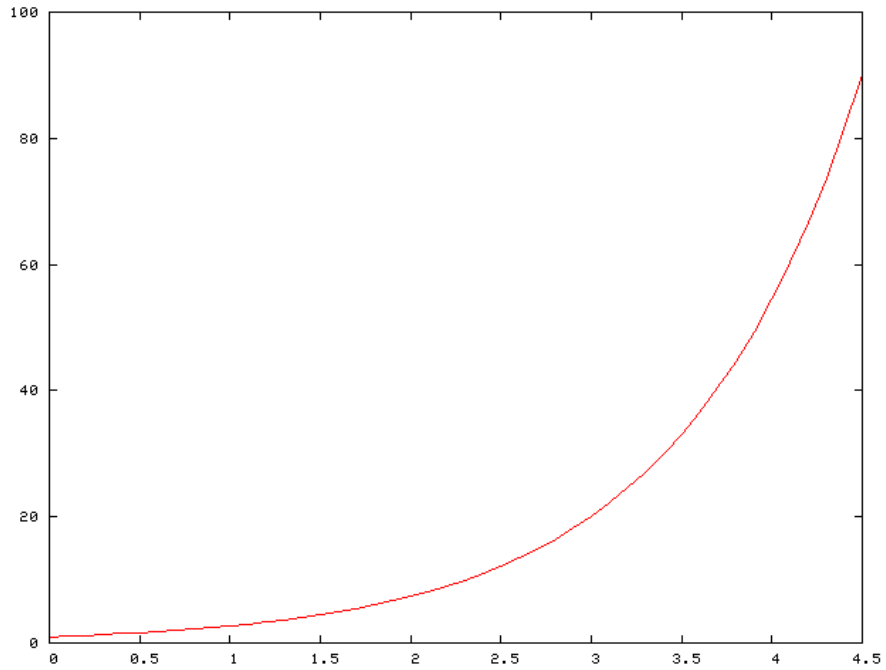
Zde bych chtěl uvést výsledky některých řešených diferenciálních rovnic.

B.1 Exponenciela

Základní diferenciální rovnice.

$$y' = y \quad y(0) = 1 \tag{B.1}$$

Řešením je funkce $y = e^t$.

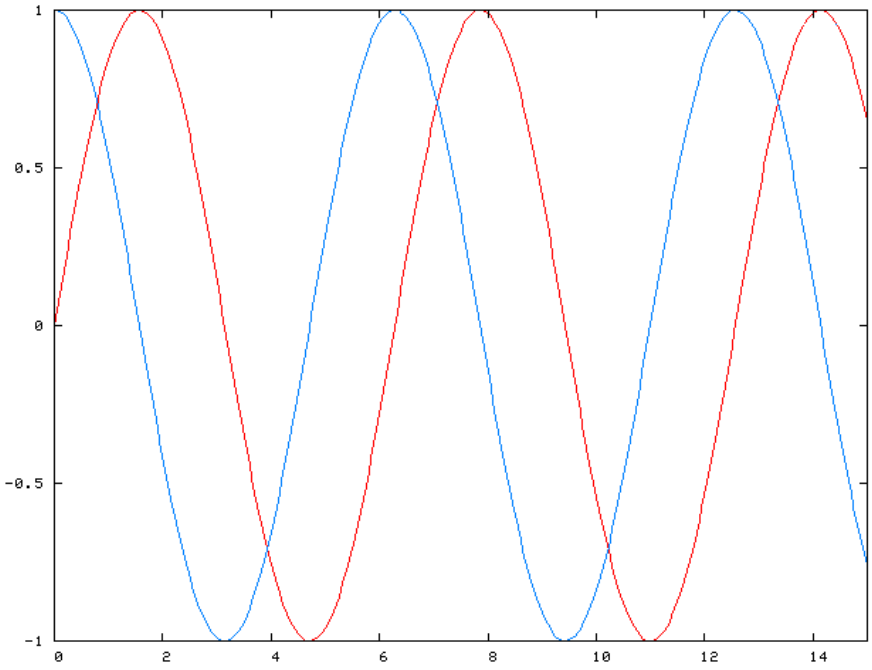


Obrázek B.1: Graf exponenciely

B.2 Sinus, kosinus

Řešením soustavy rovnic je sinus a kosinus $z = cos(t)$ a $y = sin(t)$.

$$y' = z \quad y(0) = 0 \tag{B.2}$$
$$z' = -y \quad z(0) = 1 \tag{B.3}$$

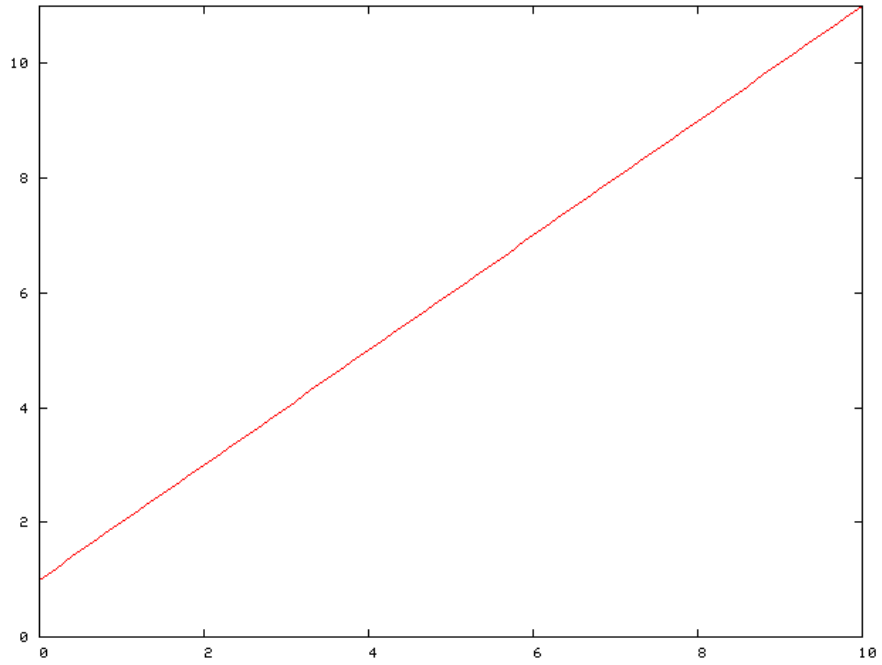


Obrázek B.2: Sinus (červeně) a kosinus (modře)

B.3 Přímka

Řešíme rovnici. Výsledek je přímka o rovnici $y = t + 1$.

$$y' = 1 \quad y(0) = 1 \tag{B.4}$$



Obrázek B.3: Přímka je řešením rovnice