Matematická knihovna pro mikropocítacovou radu Motorola HC08

Michal Brázda

Bakalárská práce 2006

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky
Ústav aplikované informatiky
akademický rok: 2005/2006

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Michal BRÁZDA**
Studijní program:	**B 3902 Inženýrská informatika**
Studijní obor:	**Informační technologie**
Téma práce:	**Matematická knihovna pro mikropočítačovou řadu Motorola HC08**

Zásady pro vypracování:

Seznamte se s principy uložení čísel s plovoucí řádovou čárkou u mikropočítačů a zpracujte literární rešerši na toto téma.
Seznamte se s funkcemi a zdrojovým kódem matematické knihovny pro mikropočítač Motorola HC11.
Navrhněte a realizujte knihovnu podprogramů pro práci s čísly s plovoucí řádovou čárkou v jazyce symbolických adres pro mikropočítač Motorola HC08.
Zpracujte dokumentaci vytvořené knihovny především s ohledem na její programátorské využití.

Rozsah práce: 60 stran

Rozsah příloh: 28 stran

Forma zpracování bakalářské práce: **tištěná/elektronická**

Seznam odborné literatury:

1. ROZEHNAL, Z.: Mikrokontroléry HC 11. BEN – technická literatura, Praha 2001
2. RYCHLÍK, J.: Programovací techniky, KOPP, České Budějovice 1994
3. VAŠEK V., VAŠEK L.: Programování mikropočítačů. FT VUT v Brně, Zlín 1989
4. DOLINAY, J.: Programové moduly pro řídící systém s HC11, Bakalářská práce. FT Zlín, 1999
5. Freescale semiconductor Inc.: Referenční manuály k HC08 – firemní literatura, 2002

Vedoucí bakalářské práce: **Ing. Jan Dolinay**
Ústav automatizace a řídicí techniky

Datum zadání bakalářské práce: **14. února 2006**

Termín odevzdání bakalářské práce: **16. června 2006**

Ve Zlíně dne 14. února 2006

L.S.

prof. Ing. Vladimír Vašek, CSc.
pověřený děkan

doc. Ing. Ivan Zelinka, Ph.D.
ředitel ústavu

ABSTRAKT

Cílem této bakalárské práce je vytvorit matematickou knihovnu zdrojových kódu pro operace s reálnými císly uloženými ve formátu plovoucí desetinné cárky pro 8-bitový mikropocítac Motorola M68HC08. Tato knihovna již byla sestavena pro mikropocítac Motorola M68HC11. Prestože oba výše zmínené mikropocítace vychází ze stejného CPU jádra, které používal jejich predchudce M6800, jsou vzájemne nekompatibilní z duvodu optimalizací CPU. Z tohoto duvodu bylo treba sestavit novou matematickou knihovnu tak, aby byla použitelná na mikropocítaci M68HC08. V teoretické cásti práce je popsán problém inkompatibility obou mikropocítacu a práce s císly s plovoucí desetinnou cárkou. V praktické cásti jsou popsány vytvorené funkce z hlediska uživatele i programátora.

Klícová slova: plovoucí desetinná cárka, mikropocítac, Motorola, M68HC11, M68HC08

ABSTRACT

The objective of this bachelor thesis is to create a mathematic source code library for manipulation with real numbers saved in floating point format for the 8-bit microcomputer Motorola M68HC08. This library has already been created for the microcomputer Motorola M68HC11. Although both microcomputers are arisen from the same CPU used by their ancestor M6800, they are incompatible as they have different CPU optimalizations. That is why there was a need for creating a new mathematical library applicable for the microcomputer M68HC08. The problems of incompatibility between the microcomputers and the work with floating point numbers are described in the theoretical part of the work. Created functions are described for users and coders in the practical part.

Keywords: floating point, microcomputer, Motorola, M68HC11, M68HC08

Podekování:

Mé díky patrí ing. Janu Dolinayovi za odborné konzultace a hlavne za pomoc pri odhalovaní programátorských „fíglu“, bez kterých by má práce byla o hodne težší, ne-li nemožná.

Motto:

Buh stvoril celá císla; vše ostatní je dílem cloveka.

Murphyho pocítacové zákony - Binomické pravidlo

Ve Zlíne …….

Podpis diplomanta

OBSAH

ÚVOD

Pri použití 8-bitových mikropocítacu rady Motorola M68HC08, muže nastat potreba pracovat scísly splovoucí desetinnou cárkou. Jelikož CPU tohoto mikropocítace neobsahuje žádnou FPU jednotku, která by operace s císly v plovoucí desetinné cárce podporovala, vyvstává potreba rešit tento problém softwarove. K tomuto úcelu byla vytvorena matematické knihovna podporující základní operace s císly uloženými v tomto formátu a zároven podprogramy pro prevody z tohoto formátu napr. na textový retezec nebo zaokrouhlené celé císlo.

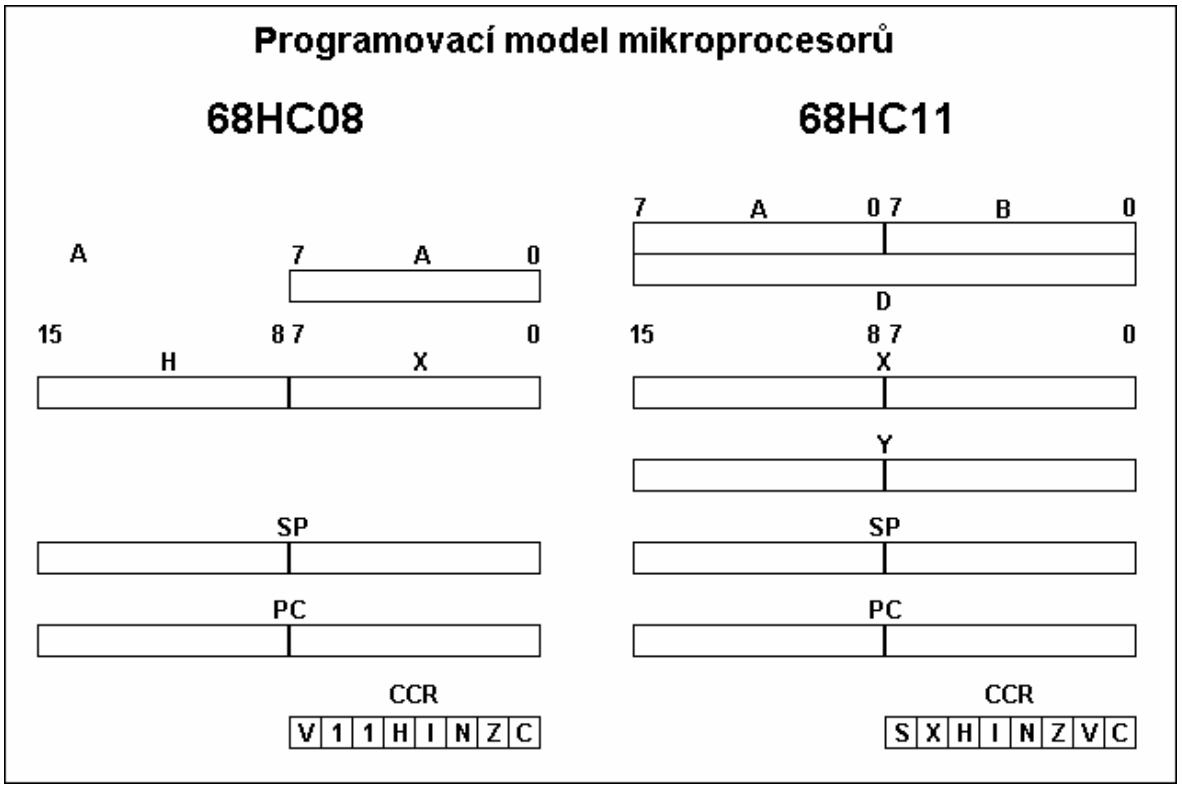
Pri tvorbe této matematické knihovny se vychází z její funkcní verze napsané pro mikropocítacovou radu Motorola M68HC11 sestavenou Gordonem Doughmanem. I když mikropocítace M68HC11 a M68HC08 mají jako spolecného predchudce mikropocítac Motorola M6800, jsou jejich CPU natolik rozdílné, že neexistuje jejich vzájemná softwarová kompatibilita mezi zdrojovými texty psanými v jazyce symbolických adres.

I. TEORETICKÁ CÁST

1 POROVNÁNÍ MIKROPOCÍTACU MOTOROLA RADY HC11 A HC08

Mikropocítace Motorola rady HC08 mají ve své výbave jeden osmi-bitový akumulátor a dva osmi-bitové registry H a X, které lze sloucit do jednoho šestnácti-bitového adresového registru H:X. Oproti tomu rada HC11 má k dispozici dva osmi-bitové akumulátory A a B slucitelné do jednoho šestnácti-bitového akumulátoru a dva šestnácti-bitové indexové registry X a Y. Oba mikropocítace také obsahují šestnácti-bitový ukazatel zásobníku SP (Stack Pointer) a šestnácti-bitový registr programového cítace PC (Program Counter), které jsou u obou mikropocítacu shodné. Zcela rozdílný je ovšem registr príznaku nebo též stavový registr CCR (Condition Code Register).

Tyto odlišnosti mají samozrejme za následek vzájemnou softwarovou a hardwarovou nekompatibilitu. Pro ilustraci je zde priložen prehledný nákres programovacího modelu obou rad.



Obr. 1.1 Programovací model mikropocítacu Motorola rady HC08 a HC11

Práve díky programovacímu modelu je celkem podstatný rozdíl i vinstrukcních sadách obou mikroprocesoru. Pocet instrukcí Motoroly HC11 je v porovnání s HC08 skoro dvojnásobný.

Programování mikropocítacu HC08 a HC11 je celkem podobné. Hlavním prínosem v programování rady HC08 je bezesporu prímá adresace dat uložených vzásobníku pomocí programového cítace. U rady HC11 je tato adresace možná pouze pres registry X a Y.

Predstavme si situaci, kdy potrebujeme použít kvýpoctu hodnotu uloženou na ctvrté pozici od horního konce zásobníku. Následující zdrojové kódy predstavují jedno z možných rešení. Zde si musíme uvedomit, že u Motoroly HC11 se instrukcí TSX uloží do registru X ukazatel na Stack Pointer snížený o jednicku.

```
Motorola HC08:    lda   5,SP   ; nacti do A hodnotu na zásobníku

Motorola HC11:    tsx            ; vlož do X ukazatel na SP snížený o 1
                  ldaa  4,X    ; nacti do A hodnotu ze zásobníku na pozici 4
```

2 PROGRAMOVÁNÍ V JAZYCE SYMBOLICKÝCH ADRES

Jazyk symbolických adres je nejnižším programovacím jazykem prakticky používaném pri programování pocítacu a mikropocítacu. Z hlediska rozkladu úlohy odpovídá programování v tomto jazyku programování ve strojovém kódu. Každé strojové instrukci je prirazena instrukce jazyka symbolických adres. Jazyk symbolických adres je tedy strojove závislým jazykem, nebot programátor se musí seznámit se všemi vlastnostmi pocítace na úrovni jeho strojových instrukcí. Instrukce jsou však na rozdíl od strojového jazyka zapisovány symbolickou formou. Symboly jsou vetšinou tvoreny mnemotechnickými zkratkami slov prirozeného jazyka (predevším anglictiny). Zápis programu je tak citelnejší a srozumitelnejší než zápis ve strojovém kódu.

Preklad programu z jazyka symbolických adres do strojového jazyka obstarává prekladac (tzv. asembler). Díky strojovému prekladu lze zmeny programu, jeho premístení v operacní pameti pocítace, nebo zmeny sestavy mikropocítace vyrešit podstatne jednodušeji, než ve strojovém jazyku.

Strojová závislost jazyka symbolických adres sebou nese výhody i nevýhody. Výhodou je bezprostrední vztah k základní strukture pocítace, což zpusobuje, že programy v jazyku symbolických adres mají obvykle menší nároky na rozsah operacní pameti než programy zapsané ve vyšších programovacích jazycích. Také jejich optimalizace z hlediska doby potrebné pro výpocet programu je snazší.

Nevýhodou jazyka symbolických adres je nemožnost prenést programy na jiný pocítac, pracnost programování a omezené možnosti syntaktických a sémantických kontrol programu pri prekladu i výpoctu

2.1 Príkazy jazyka symbolických adres

Príkazy jazyka symbolických adres se delí do dvou skupin: *symbolické instrukce* a *pseudoinstrukce*.

Symbolické instrukce prímo odpovídají strojovým instrukcím.

Pseudoinstrukce jsou príkazy pro prekládající program; nemají ekvivalent v preloženém strojovém programu.

Posloupnost príkazu jazyka symbolických adres budeme nazývat zdrojovým programem. Prekladac prekládá tento program do cílového (výsledného) programu.

2.2 Symbolické instrukce

Každá symbolická instrukce se skládá ze trí polí: pole náveští, pole operacního znaku a pole operandu; obvykle je zapsána v jednom rádku zdrojového programu. Jednotlivá pole instrukce jsou oddelena predepsanými znaky, tzv. oddelovaci, napr. mezerami nebo cárkami, nebo je pevne predepsán formát rádku (napr. pri zápisu na derné štítky).

Krome techto polí muže za oddelovacem následovat ješte komentár, jenž prekladac interpretuje jako textový retezec, který beze zmen prevádí do výpisu prekladu; jeho obsah nemá vliv na výsledný program. Komentár slouží k orientaci v programu jak pri zápisu a odladování, tak pri údržbe programu. Je významnou složkou dokumentace programu.

V poli náveští se uvádí symbolické jméno adresy, na níž má být instrukce umístena. Odpovídající pametové místo se této symbolické adrese prirazuje až pri prekladu nebo pri zavádení programu do operacní pameti pocítace. Pole náveští je nepovinné a využívá se tehdy, když se na instrukci odvolává jiná instrukce, napr. skoková nebo vyvolávací. Do pole náveští se smí stejná symbolická adresa zapsat jen jednou. Symbolické jméno adresy se tvorí podle stejných syntaktických pravidel jako ostatní symbolická jména (tzv. identifikátory) definovaná uživatelem. Tato pravidla predepisují predevším délku textového retezce abecedne císlicových znaku tvorících symbolické jméno i abecedu, do které musí použité znaky patrit. Je samozrejmé, že definovaná jména nesmí být stejná jako klícová slova jazyka, používaná k pojmenování operacních znaku, registru atd. Je-li zdrojový program clenen do nekolika programových modulu, pak se mnohdy omezuje rozsah platnosti symbolických jmen na jednotlivé moduly. Podle rozsahu platnosti delíme symbolická jména na globální a lokální. Globální symbolická jména platí v celém zdrojovém programu, lokální jen v rámci programového modulu. Výhoda tohoto rozdelení je zrejmá. Jestliže programové moduly vypracovávají ruzní programátori, stací, když se dohodnou na globálních symbolických jménech, na než se programové moduly odvolávají a jejichž prostrednictvím spolu komunikují.

V poli operacního znaku se uvádí symbolické jméno instrukce, totožné se symbolickým jménem operace, jejíž provedení instrukce predepisuje. Symbolická jména instrukcí patrí ke klícovým slovum jazyka a tvorí je mnemotechnické zkratky.

V poli operandu se specifikují operandy instrukce. Bezadresové instrukce nevyžadují operandy a jejich pole operandu musí zustat prázdné. Vyžaduje-li instrukce specifikaci více než jednoho operandu, zapisují se operandy v predepsané posloupnosti, napr. nejdríve zdrojový operand, pak operand príjemce, a oddelují se predepsaným oddelovacem. Jako operandy mohou vystupovat obsahy registru, pametových míst a prímé operandy. Mohou být ruzných typu: slabika, slovo, vícenásobné slovo, s dvojkovým, dvojkove desítkovým zobrazením dat nebo zobrazením dat s pohyblivou rádovou cárkou atd. Vetšinou jazyky symbolických adres poskytují prostredky pro práci s jednoduchými promennými, tj. velicinami, jejichž hodnota se muže behem provádení programu menit. Pri odvolávkách na operandy instrukcí se musí symbolicky specifikovat použitá adresovací metoda.

Prímé operandy se mohou specifikovat bud prímo jako císlo ve zvolené císelné soustave, nebo jako abecedne císlicový znak patrící ke zvolené abecede, nebo i pomocí výrazu, jejichž hodnotu pak vypocítá prekladac. Popis jazyka musí obsahovat popis prípustných operandu ve výrazech, poradí jejich vyhodnocování a jejich význam. Prímé operandy lze též specifikovat symbolickým jménem.

2.3 Pseudoinstrukce

Pseudoinstrukce jsou príkazy prekladaci, jenž se jimi v procesu prekladu rídí. Patrí k nim príkazy pro alokaci programu a dat, príkazy pro rezervaci pametových míst, definicní príkazy (deklarace) ruzných objektu jazyka, napr. datových struktur nebo dílcích programu, a príkazy pro rízení prekladu a jeho výpisu.

Pseudoinstrukce mívají obvykle stejný formát jako symbolické instrukce. Jejich zápis obsahuje rovnež tri pole a muže být doplnen komentárem. Pole náveští se v nekterých pseudoinstrukcích nepoužívá, v jiných, napr. v definicních pseudoinstrukcích, se do nej zapisuje symbolické jméno, které pseudoinstrukce prirazuje definovanému objektu. Pole operacního znaku obsahuje symbolické jméno pseudoinstrukce, které patrí ke klícovým slovum jazyka.

Obsah pole operandu a jejich význam závisí na pseudoinstrukci stejne jako v prípade symbolických instrukcí.

Základním príkazem umožnujícím predepsat umístení instrukce je príkaz ORG, jehož operandem je hodnota, na kterou se po provedení príkazu nastaví cítac adres, používaný prekladacem jako ukazatel instrukcí. Príkladem definice je pseudoinstrukce EQU, prirazující symbolické jméno konstante, adrese nebo výrazu. K pseudoinstrukcím pro rízení prekladu patrí napr. instrukce pro rízení podmíneného prekladu, umožnující vytváret ruzné varianty programu. Pseudoinstrukce pro rízení výpisu umožnuje predepsat stránkování výpisu, opatrit novou stránku predepsaným záhlavím, potlacit tisk nekterých informací apod.

2.4 Prekladac

Program zapsaný v jazyku symbolických adres se predkládá do výsledného jazyka pomocí prekladace nazývaného asembler. (Pojem asembler se nekdy nesprávne používá pro oznacení jazyka symbolických adres, v anglictine se však rozlišuje mezi prekladacem (assembler) a jazykem (assembly language)). Stejne jako pri prekladu z jednoho prirozeného jazyka do jiného používá prekladac slovník uložený v pameti pocítace. Prekladac cte postupne instrukce zdrojového jazyka, prirazuje symbolickým jménum operacních znaku a operandu strojový kód a prirazuje strojovým instrukcím absolutní nebo relativní adresy pametových míst.

Preklad instrukce samotné je pomerne jednoduchý, nebot svýjimkou makroinstrukcí jedné zdrojové instrukci odpovídá jedna výsledná instrukce. Potíže vznikají jen tehdy, je-li nekteré pole strojové instrukce závislé na alokaci programu v operacní pameti. Odkazy na následující instrukce nelze pak zajistit jediným pruchodem zdrojového textu. Proto vetšina prekladacu pracuje dvou-pruchodne a používá dvou slovníku. Jedním z nich je nemenný slovník, obsahující klícová slova jazyka, a to jména operacních znaku a nekterých operandu, napr. jména registru. Tento slovník je jistým vyjádrením sémantických pravidel jazyka, nebot prisuzuje význam prípustným kombinacím znaku. Symbolickému jménu ADD prisuzuje napr. význam príkazu k operaci scítání. Druhým slovníkem je slovník uživatelsky definovaných symbolických jmen adres a operandu, jenž se vytvárí pri prvním pruchodu zdrojovým textem. Prirazení absolutních nebo relativních adres symbolickým adresám pri prvním pruchodu umožnuje cítac adres inicializovaný pseudoinstrukcemi ORG, jehož obsah je behem pruchodu zvýšen pri každém

prechodu symbolické instrukce o hodnotu odpovídající poctu slov, popr. slabik jejího strojového ekvivalentu. Další položky tohoto slovníku jsou definovány pseudoinstrukcemi EQU apod. Pri druhém pruchodu zdrojovým textem dochází k vlastnímu prekladu.

2.5 Absolutní a premistující prekladace

Výsledný strojový kód porízený prekladacem muže být bud v absolutním, nebo relativním tvaru. Výsledkem cinnosti jednodušších prekladacu je absolutní strojový kód, jenž muže být alokován jen v oblasti operacní pameti zadané pseudoinstrukcemi ORG. Pri požadavku na premístení výsledného strojového kódu v operacní pameti je nutné zdrojový program znovu preložit, nebot se musí znovu porídit slovník absolutních adres a zmenit všechny položky strojového kódu závislé na umístení, napr. operandy skokových a vyvolávacích instrukcí.

Pri clenení programu na programové moduly není príliš výhodné prirazovat symbolickým adresám absolutní adresy již ve fázi prekladu. To totiž ješte nemusí být známa struktura fyzického adresového prostoru uživatelského systému, napr. clenení operacní pameti na bloky, realizované pevnými pametmi a pametmi pro ctení a zápis. Do zdrojového programu je mnohdy treba ješte zaclenit nekteré z univerzálních programových modulu dodávaných výrobcem nebo vypracované uživatelem knihovny programových modulu. Pokud by byly v knihovne uloženy v absolutním strojovém kódu, bylo by jejich zaclenení spojeno se znacnými potížemi, napr. s novým prekladem. Pri vývoji programového vybavení se také casto mení umístení jednotlivých programových modulu. Proto je úcelné od sebe oddelit procesy prekladu a umístení (alokace). Pri prekladu pak prirazuje prekladac strojovým instrukcím relativní adresy vzhledem k pocátku programového modulu; obvykle se tato pocátecní adresa volí nulová. Množina adres prístupná prekladaci tvorí tzv. logický adresový prostor. Výsledkem práce premístujícího prekladace je program v tzv. relativním tvaru, tj. s relativními adresami z logického adresového prostoru.

Po prekladu se jednotlivé moduly spojí ve výsledný program pomocí spojovacího programu (linker). Pri dynamické alokaci se logickému adresovému prostoru programu prirazuje fyzický adresový prostor až pri výpoctu. Pri statické alokaci zprostredkovává toto prirazení pri príprave programu tzv. umístovací program (locater) nebo až pri zavádení programu do operacní pameti mikropocítace zavádecí program (loader). Prekladac však musí behem prekladu porizovat seznam položek strojového kódu modifikovaných pri premístení. Tento seznam slouží jako

pomucka pro další programy, které premístitelné programové moduly dále zpracovávají. Z toho duvodu i knihovní moduly v relativním tvaru obsahují údaje nutné pro premístení. Z výsledného programu se tyto údaje odstranují až po definitivním odladení. Pokud by program vyžadoval další zásah spojený s premístením, bylo by nutné jej znovu preložit.

Pri spojování programových modulu je treba rešit problém odvolávek na globální symbolická jména umístená vne zpracovávaného programového modulu. K deklaraci globálních symbolických jmen se používá pseudoinstrukcí; premístující prekladac pak na základe techto deklarací porizuje pro návazné programy seznam globálních symbolických jmen prekládaného modulu. Rovnež je nutné vyznacit odvolávky na symbolická jména, která nejsou v prekládaném modulu obsažena. K tomu se obvykle používá tzv. príznaku, tj. dohodnutých klícových slov jazyka, pripojených k odvolávce.

Spojovací program pri zjištení vnejší odvolávky prohledává seznam globálních symbolických jmen ostatních spojovaných programových modulu. Nalezne-li odpovídající symbolické jméno, pak muže odvolávku uspokojit, v opacném prípade uvede neuspokojenou odvolávku ve výpisu.

Nekteré prekladace umožnují vytvorit spolecný pametový prostor pro všechny programové moduly napr. tak, že cástem logických adresových prostoru jednotlivých programových modulu pridelí týž fyzický adresový prostor. Prístup ke spolecným promenným lze na rozdíl od predcházejícího prípadu získat prostrednictvím lokálních symbolických jmen jednotlivých programových modulu.

2.6 Výpis prekladu

Prekladac porizuje záznam o své cinnosti formou výpisu, jenž obvykle obsahuje zdrojový program, výsledný program a další informace o prekladu, napr. výpis syntaktických chyb a jejich specifikace, seznam symbolických adres a jim pridelené absolutní nebo relativní adresy, seznam vnejších odvolávek, globálních jmen apod. Seznamy obsahují i císla príkazu, v nichž byl symbol definován a použit. Nekdy se tyto informace oznacují jako krížové reference.

Strojový kód se pro lepší citelnost zaznamenává ve zvolené císelné soustave. Pseudoinstrukcemi pro rízení výpisu lze predepsat císelnou soustavu a tisk nekterých dodatecných informací.

3 REPREZENTACE CÍSEL S PLOVOUCÍ DESETINNOU CÁRKOU

Každá numerická hodnota racionálního císla uloženého ve formátu plovoucí desetinné cárky, v sobe nese polohu desetinné cárky. Z tohoto duvodu je krome bitu, které jsou rezervovány pro uložení významných císlic numerické hodnoty, nutné pro každou numerickou hodnotu rezervovat i další bity, pomocí nichž je urcena mocnina o nejakém základu (typicky 2, 8, 10 ci 16), kterou musí být významné císlice vynásobeny resp. vydeleny. První cást císla uloženého ve formátu s plovoucí desetinnou cárkou se nazývá mantisa, druhá cást exponent. Obecný formát uložení a zpusob získání puvodního císla je následující:

$$x_{FP} = b^e * m$$

kde:

x_{FP} značí reprezentovanou numerickou hodnotu z podmnožiny reálných císel

b je báze, nekdy také nazývaná *radix*

e je hodnota exponentu (muže být i záporná)

m je mantisa, která muže být i záporná

Konkrétní formát numerických hodnot reprezentovaných v systému plovoucí rádové tecky závisí predevším na volbe báze (radixu) a také na poctu bitu rezervovaných pro uložení mantisy a exponentu. V minulosti existovalo mnoho ruzných formátu plovoucí desetinné cárky, dnes se však ustálilo použití formátu specifikovaných v norme IEEE 754.

3.1 Formát plovoucí desetinné cárky a norma IEEE 754

Norma IEEE 754 specifikuje nejenom vlastní formát uložení numerických hodnot v systému plovoucí desetinné cárky, ale i pravidla implementace operací stemito hodnotami, vcetne konverzí. Konkrétne je v této norme popsáno:

- Základní (*basic*) a rozšírený (*extended*) formát uložení numerických hodnot.
- Zpusob provádení základních matematických operací: scítání, odecítání, násobení, delení, zbytek po delení, druhá odmocnina a porovnání.

- Pravidla konverze mezi celocíselnými formáty a formáty s plovoucí desetinnou cárkou.
- Zpusob konverze mezi ruznými formáty s plovoucí desetinnou cárkou..
- Zpusob konverze základního formátu s plovoucí desetinnou cárkou na retezec císlic.
- Práce s hodnotami NaN (Not a Number – nejedná se o císlo) a výjimkami.

Zde si uvedme podstatu reprezentace hodnot a zpusob provádení základních matematických operací. Vybraná podmnožina racionálních císel muže být vyjádrena vztahem:

$$X_{FP} = (-1)^S * 2^{EXP-BIAS} * m$$

kde:

$\mathbf{X_{FP}}$ znací reprezentovanou numerickou hodnotu zpodmnožiny racionálních císel (ta je zase podmnožinou císel reálných). Díky vyhrazeným (speciálním) hodnotám je možné rozlišit kladnou a zápornou nulu i kladné a záporné nekonecno. Také se muže uložit necíselná hodnota: *NaN – (Not a Number)*, která je výsledkem nekterých matematicky nedefinovaných operací, napríklad 0/0 nebo 0^0.

2 je *báze*, nekdy také nazývaná *radix*. U IEEE 754 je to vždy dvojka, protože výpocty s bází dve jsou pro císlicové obvody nejjednodušší. V minulosti se používaly i jiné báze, napríklad 8, 16 nebo i 10.

EXP je vždy kladná hodnota exponentu posunutého o základní hodnotu - bias

BIAS je hodnota, díky které je uložený exponent vždy kladný. Tato hodnota se vetšinou volí dle vztahu: $bias=2^{eb-1}-1$, kde *eb* je pocet bitu vyhrazených pro exponent. Pro specifické úcely je však možné zvolit i jinou hodnotu.

m je mantisa, která je u formátu IEEE 754 vždy kladná

s je znaménkový bit nabývající hodnoty 0 nebo 1. Pokud je tento bit nulový, je reprezentovaná hodnota $\mathbf{X_{FP}}$ kladná, v opacném prípade se jedná o zápornou hodnotu. Vzhledem ktomu, že je jeden bit vyhrazen na uložení znaménka, je možné rozlišit kladnou a zápornou nulu.

Podle bitové šírky císel **EXP**, **BIAS** a **m** se rozlišují základní a rozšírené formáty FP císel. Norma IEEE 754 pritom explicitne zminuje dva základní formáty: jednoduchá presnost a dvojitá presnost.

3.2 Základní formát single dle normy IEEE 754

Prakticky všechny v dnešní dobe používané matematické koprocesory (*FPU*), programovatelné grafické procesory (*GPU*) nebo systémové i aplikacní knihovny, které pracují s císelnými hodnotami uloženými ve formátu pohyblivé desetinné cárky, podporují formát jednoduché presnosti, který také bývá nazýván *single*; v nekterých programovacích jazycích pak *float*. Tento formát je charakteristický tím, že se pro uložení numerické hodnoty používá triceti dvou bitu (4 byty), což pro mnoho aplikací predstavuje dobrý pomer mezi rozsahem hodnot, presností a nároky na úložný prostor, nehlede na to, že mnoho architektur používá 32 bitové sbernice. Onech 32 bitu je rozdeleno do trech cástí. Vprvní cásti (predstavované nejvyšším bitem) je uloženo znaménko, následuje osm bitu pro uložení posunutého exponentu a za nimi je zbývajících 23 bitu, které slouží pro uložení mantisy. Celé triceti dvoubitové slovo s hodnotou vyjádrenou pomocí plovoucí desetinné cárky tedy vypadá následovne:

bit	31	30 29 ... 24 23	22 21 ... 3 2 1 0
význam	s	exponent (8 bitu)	mantisa (23 bitu)

tab. 3.1 Význam jednotlivých bitu ve standard u IEEE 754

Exponent je pritom posunutý o hodnotu *bias*, která je nastavena na 127, protože je použit výše uvedený vztah: $bias = 2^{eb-1}-1$

a po dosazení *eb=8 (bitu)* dostaneme: $bias = 2^{8-1}-1=2^{7}-1=128-1=127$

Vzorec pro vyjádrení reálné hodnoty vypadá následovne: $X_{single} = (-1)^{S} * 2^{EXP-127} * m$

Uložení znaménka císelné hodnoty je jednoduché: pokud je znaménkový bit nastavený na jednicku, jedná se o zápornou hodnotu, vopacném prípade jde o hodnotu kladnou. Exponent je uložený vtakzvané *posunuté forme,* tj. jako binárne zakódované celé císlo v rozsahu 0..255. Po vyjádrení neposunutého exponentu dostáváme rozsah -127..128, obe krajní hodnoty jsou však použity pro speciální úcely, proto dostáváme rozsah exponentu -126..127 pro normalizovaná císla (krajními hodnotami jsou takové exponenty, které mají všechny bity bud jednickové nebo

naopak nulové). Ješte si však musíme ríci, jakým zpusobem je uložena mantisa. Ta je totiž vetšinou (až na velmi malá císla) normalizovaná, což znamená, že se do mantisy ukládají pouze hodnoty v rozsahu <1,0;2,0-e>. Vzhledem k tomu, že první bit umístený pred binární teckou je u tohoto rozsahu vždy nastavený na jednicku, není ho zapotrebí ukládat, což znamená, že ušetríme jeden bit z 32-bitového slova. Pro normalizované hodnoty platí následující vztah:

$$X_{single} = (-1)^S * 2^{EXP-127} * (1.M)_2$$

kde **M** je hodnota bitového vektoru mantisy, tj.:

$$M = m_{22}^{-1} + m_{21}^{-2} + m_{20}^{-3} + \ldots + m_1^{-22} + m_0^{-23}$$

Rozsah hodnot, jež je možné reprezentovat pomocí jednoduché presnosti v normalizovaném tvaru je $-3{,}4\times10^{38}$ až $3{,}4\times10^{38}$. Nejnižší reprezentovatelná (normalizovaná) hodnota je rovna $1{,}17549\times10^{-38}$, denormalizovaná pak $1{,}40129\times10^{-45}$. Zpusob výpoctu techto hodnot je uveden v tabulce:

hexadecimální hodnota	výpocet FP	dekadický výsledek	normalizováno
0x00000001	$2^{-126}\times2^{-23}$	$1{,}40129\times10^{-45}$	ne
0x00800000	2^{-126}	$1{,}17549\times10^{-38}$	ano
0x7F7FFFFF	$(2-2^{-23})\times2^{127}$	$3{,}4\times10^{38}$	ano

tab. 3.2 Maximální reprezentovatelné hodnoty pro standard IEEE 754

Význam tech exponentu, které mají minimální a maximální hodnotu, tj. jsou bud nulové, nebo mají hodnotu 255 (obe samozrejme pred posunem) je prehledne uveden vnásledující tabulce:

s-bit	exponent	mantisa	význam	šestnáctkove
0	0<e<255	>0	normalizované kladné císlo	
1	0<e<255	>0	normalizované záporné císlo	
0	0	>0	denormalizované kladné císlo	
1	0	>0	denormalizované záporné císlo	
0	0	0	kladná nula	0x00000000
1	0	0	záporná nula	0x80000000
0	255	0	kladné nekonecno	0x7F800000
1	255	0	záporné nekonecno	0xFF800000
0	255	>0	NaN – not a number	

1	255	>0	NaN – not a number	

tab. 3.3 Normalizované hodnoty pro standard IEEE 754

Denormalizovaná císla jsou takové hodnoty, u kterých není první (explicitne nevyjádrený) bit mantisy roven jednicce, ale naopak nule. Výpocty s temito velmi malými hodnotami nejsou presné, zejména pri násobení a delení. Pri ukládání denormalizovaných císel je exponent vždy nastaven na nejnižší hodnotu, tj. -126 a nejvyšší (explicitne neukládaný) bit mantisy je nulový, nikoli jednickový, jak je tomu u normalizovaných hodnot. Hodnota typu *NaN* vznikne v prípade, že je použita operace s nejasným výsledkem, napríklad 0/0, 0^0 nebo, a to v praxi snad nejcasteji, pri odmocnování záporných císel. Nekonecná hodnota vzniká typicky pri delení nulou (zde je možné zjistit znaménko), nebo pri vyjádrení funkcí typu *log(0)* atd.

3.3 Soucet a rozdíl dvou FP hodnot

Pri scítání dvou hodnot uložených ve formátu pohyblivé desetinné cárky podle normy IEEE 754 by se melo postupovat podle následujícího algoritmu:

1. Nejprve je zjišteno, zda se scítání ci odecítání neprovádí se speciálními hodnotami, tj. s hodnotou *NaN* (not a number) ci s nekonecny. Výsledek souctu ci rozdílu provedeného se speciálními hodnotami je uveden v tabulce na konci popisu algoritmu.

2. Poté je provedeno nalezení vetšího z obou scítaných císel. Porovnávají se však pouze hodnoty exponentu, nikoli mantis (zde se ukazuje výhodnost použití exponentu posunutých o *bias*, nebot pri jejich porovnávání nemusíme brát v úvahu znaménko). Pokud mají obe scítaná ci odecítaná císla stejné exponenty, preskocí se následující bod.

3. Dále se vypocítá císelný rozdíl exponentu, tj. $diff = e_1 - e_2$. Mantisu menšího císla je nutné posunout doprava o tolik bitu, kolik ciní rozdíl exponentu (samozrejme po vydelení rozdílu hodnotou 2^n, ve skutecnosti se pouze porovnávají exponenty bez zbytecného prevodu na jejich reálnou hodnotu). Bitový posun mantisy odpovídá násobení ci delení dané hodnoty mocninou císla 2, stejne tak zvýšení ci snížení hodnoty exponentu. Pro jiné báze (radixy) to však neplatí.

4. Podle znaménkových bitu obou operandu je vycíslen bud soucet nebo rozdíl jejich mantis. Zde se jedná o "klasický" soucet pomocí vícebitové scítacky s postupným ci zrychleným

prenosem. Pri odecítání je nutné vzít vúvahu, že se nejedná o císla uložená ve dvojkovém doplnku, tj. odecítání je ponekud složitejší.

5. Pokud hodnota výsledku pretece (tj. výsledek se nevejde do poctu bitu mantisy), posunou se bity v mantise doprava o jeden bit a exponent se zvýší o jednicku. Z toho vyplývá, že soucet je nutné provádet na scítacce minimálne o jeden bit širší, než je bitová šírka mantisy (postací prenos CARRY).
6. V prípade, že mantisa nemá nastaven nejvyšší bit na jednicku, posouvá se (i nekolikrát) doleva a exponent se pritom snižuje. To je takzvaná normalizace, která zajistí, že nejvyšší bit mantisy je vždy jednickový a tudíž nemusí být v mantise uložen.
7. Dále se musí vyjádrit znaménko výsledku: v prípade scítání se jednoduše zkopíruje znaménko vetšího císla, u odecítání se musí vzít vúvahu i pozice císla voperaci, tj. zda se jedná o menšenec ci menšitel.
8. Po vycíslení znaménka se provede zaokrouhlení hodnoty podle práve nastaveného zaokrouhlovacího režimu. V norme jsou specifikovány ctyri zpusoby zaokrouhlení (tzv. rounding modes): zaokrouhlení smerem k nule, zaokrouhlení smerem ke kladnému nekonecnu, zaokrouhlení smerem k zápornému nekonecnu a konecne zaokrouhlení k nejbližšímu sudému reprezentovatelnému císlu (pojmenování posledního zaokrouhlovacího režimu není presné, protože rozhodnutí, zda se jedná o sudé císlo, je zapotrebí provést pouze pro hranicní hodnoty).
9. Pokud behem výpoctu dojde ktomu, že exponent dosáhne své maximální hodnoty, dojde k pretecení výsledku, bude jako výsledek operace scítání ci odecítání vráceno kladné nebo záporné nekonecno, podle samostatne vypocítaného znaménka.
10. Pokud naopak hodnota exponentu dosáhne své minimální hodnoty, dochází k podtecení. Pri nem se bud generuje výjimka, nebo se hodnota výsledku nastaví na kladnou ci zápornou nulu, opet podle samostatne vypocítaného znaménka výsledku.

Kvuli pretékání, podtékání a zaokrouhlování hodnot behem scítání/odcítání prestávají platit nekterá matematická pravidla, napríklad **A+B-A==B** ci **(A+B)+C=A+(B+C)** a naopak zacínají platit pravidla nová, treba pri reálném programování velmi nebezpecné a záludné **A+B=A** (platí samozrejme pouze pro nekteré hodnoty **A** a **B**). V prvním bodu algoritmu pro soucet ci

rozdíl dvou hodnot je zjišťováno, zda se výpocet neprovádí se speciálními hodnotami. Výsledek operace v prípade použití speciálních hodnot je ukázán v následující tabulce.

Hodnota X	Hodnota Y	Výsledek operace X+Y
konecná	+8	+ 8
konecná	-8	- 8
+ 8	konecná	+ 8
- 8	konecná	- 8
+ 8	+ 8	+ 8
- 8	- 8	- 8
+8	- 8	NaN
-8	+ 8	NaN
NaN	libovolná	NaN
libovolná	NaN	NaN

tab. 3.4 Výsledek operace scítání/odcítání pri použití speciálních hodnot

3.4 Soucin dvou FP hodnot

Soucin dvou numerických hodnot uložených ve formátu plovoucí desetinné cárky je opet – vzhledem k rozdelení hodnot na mantisu a exponent – komplikovanejší než prosté poslání obsahu dvou registru do násobicky. Nejprve si teoreticky ukažme, jakým zpusobem se vynásobí dve hodnoty reprezentované svou mantisou a exponentem. Vstupními hodnotami jsou císla x_1 a x_2:

$$X_1 = (-1)^{S1} * 2^{EXP1} * m_1$$
$$X_2 = (-1)^{S2} * 2^{EXP2} * m_2$$

Soucin techto hodnot mužeme vyjádrit vzorcem:

$$X1\,X\,2 = (-1)^{S1} * (-1)^{S2} * 2^{EXP1+EXP2} * m_1 * m_2$$

Prakticky to znamená, že (pokud nebudeme uvažovat nutnou normalizaci a zaokrouhlení výsledku) se mantisy obou hodnot navzájem vynásobí a exponenty se sectou. Nejjednodušší je operace se znaménkovými bity – na ne se aplikuje bitová operace nonekvivalence (*XOR:* $\oplus$). Vzhledem k tomu, že se musíme v co nejvetší míre vyhnout pretecení, podtecení a denormalizaci hodnot, je násobení dvou FP hodnot složitejší a provádí se následujícím postupem:

1. Podobne jako u operace scítání ci odcítání, i pri násobení je nejprve zjišteno, zda do operace nevstupují speciální hodnoty. S temi je zacházeno podle následující tabulky:

Hodnota X1	Hodnota X2	Výsledek operace X1*X2
nenulová kladná	+8	+8
nenulová kladná	-8	-8
nenulová záporná	+8	-8
nenulová záporná	-8	+8
nula (0,0)	+8	NaN
nula (0,0)	-8	NaN
+8	nenulová kladná	+8
-8	nenulová kladná	-8
+8	nenulová záporná	-8
-8	nenulová záporná	+8
+8	nula (0,0)	NaN
-8	nula (0,0)	NaN
+8	+8	+8
+8	-8	-8
-8	+8	-8
-8	-8	+8
NaN	libovolná	NaN
libovolná	NaN	NaN

tab. 3.5 Výsledek operace násobení pri použití speciálních hodnot

2. Exponent výsledku se rovná souctu obou exponentu, tj. provede se operace EXP=*EXP1*+*EXP2*. Od tohoto výsledku je nutné odecíst posun exponentu (**bias**), a to z toho duvodu, že v mantise jsou uloženy hodnoty s binární cárkou umístenou za prvním bitem a prostým vynásobením mantis by vlastne vznikl výsledek posunutý o nekolik (binárních) rádu smerem doleva.

3. Výsledná mantisa vznikne soucinem obou mantis, tj. provede se operace $m = m_1 * m_2$.

4. Pokud výsledná mantisa pretece, je bitove posunuta doprava o *n* bitu a exponent je o tuto hodnotu *n* zvýšen.

5. Když nemá mantisa nejvyšší bit roven 1, posouvá se doleva a exponent se naopak snižuje (*normalizace*). Tato operace se muže provádet i nekolikrát.

6. Pokud se po souctu exponentu prekrocí maximální hodnota (dojde k *pretecení*), je výsledkem nekonecno (kladné ci záporné, to záleží na znaménkových bitech operandu).
7. Pokud se po souctu exponentu prekrocí minimální hodnota (dojde k *podtecení*), je výsledkem nula (kladná ci záporná, opet záleží na znaménkových bitech obou operandu)
8. Výsledný znaménkový bit je vyjádren nonekvivalencí obou znaménkových bitu, tj. $s = s_1 \oplus s_2$.
9. Po vycíslení znaménka se provede *zaokrouhlení* hodnoty podle práve nastaveného zaokrouhlovacího režimu.

3.5 Podíl dvou FP hodnot

Pri výpoctu podílu dvou hodnot se využívá následující vztah:

$$X_1 / X_2 = (-1)^{S1} * (-1)^{S2} * 2^{EXP1-EXP2} * m_1 / m_2$$

Vlastní vydelení se muže provádet bud jako samostatná operace (tak to ciní dnešní FPU), která vyžaduje delicku, nebo je – v prípade nutnosti šetrení logickými obvody – možné použít postupné odecítání delitele od delence (algoritmus známý ze základních škol). Tretí možností, kterou implementují nekteré FPU a GPU, je výpocet prevrácené hodnoty delitele pomocí specializovaných postupu a následné vynásobení delence prevrácenou hodnotou delitele. Všechny tri postupy však musí zarucit, že se pri delení nulou bude zachovávat znaménko výsledku (tj. kladné ci záporné nekonecno) a také se zajistí, aby se detekovala nepovolená operace 0/0.

3.6 Standard IEEE - príklad

Pri prevodu desetinného císla vyjádreného vdesítkové soustave na tvar podle standard u IEEE 754 je postup následující:

1) prevedeme celou cást do binární soustavy

2) prevedeme desetinnou cást do binární soustavy

3) skryjeme nejvýznamnejší bit

4) urcíme exponent

5) urcíme znaménkový bit

Príklad prevodu

desítkové císlo 49,65625 prevést na Floating Point formát podle standard u IEEE

1)

49:2	=	24	zbytek	1		6:2	=	3	zbytek 0
24:2	=	12	zbytek	0		3:2	=	1	zbytek 1
12:2	=	6	zbytek	0		1:2	=	0	zbytek 1

tedy celá cást 49,0 je v binárním kódu 110001

2)

$0,65625 * 2 = 1,3125$	$1,3125 \geq 1$	sepíšeme	1
$1,3125 - 1 = 0,3125$			
$0,3125 * 2 = 0,625$	$0,625 < 1$	sepíšeme	0
$0,625 * 2 = 1,25$	$1,25 \geq 1$	sepíšeme	1
$1,25 - 1 = 0,25$			
$0,25 * 2 = 0,5$	$0,5 < 1$	sepíšeme	0
$0,5 * 2 = 1,0$	$1 = 1$	sepíšeme	1

Desetinná cást 0,65625 je tedy v binárním kódu 10101

3) skryjeme nejvýznamnejší bit

110001,10101 => 10001,10101

4) urcíme exponent tzn. o kolik míst musíme posunout desetinnou cárku, aby byla pred nejvýznamnejším bitem.

10001,10101 => ,1000110101 => 5 míst

Exponent je 5. Po prictení +127 dostaneme 132 což je binárne 10000100

5) protože je císlo kladné znaménkový bit bude 0

Výsledek prevodu císla 49,6525 na binární tvar FP císla vypadá takto:

$01000010010001101010000000000000_b = 4246A000_h$

Pro lepší orientaci je výsledek znázornen v tabulce:

císlo bitu	31	30 23	22-0
hodnota (binárne)	0	10000100	10001101010000000000000

tab. 3.6. Zápis císla 49,625 podle standard u IEEE

3.7 Matematická knihovna - príklad

V matematické knihovne je zápis císla splovoucí desetinnou cárkou ponekud odlišný. Pro usnadnení práce a její efektivnost je pri reprezentaci císla použito 5 bytu.

Prvním bytem je exponent shodný s reprezentací použitou u IEEE formátu. Další tri byty jsou urceny pro mantisu desetinného císla. Poslední, tedy pátý, byte je znaménkový a urcuje je-li císlo kladné (hodnota 00h) nebo záporné (hodnota FFh).

císlo bitu	39-32	31-8	7-0
význam	exponent (8 bitu)	mantisa (24 bitu)	znaménkový byte (8 bitu)

tab. 3.7. Významy jednotlivých bitu FP císla v matematické knihovne

Samotný prevod na císlo na splovoucí desetinnou cárkou tedy probíhá podobe jen s malými rozdíly.

1) prevedeme celou cást do binární soustavy

2) prevedeme desetinnou cást do binární soustavy

3) urcíme exponent

4) urcíme znaménkový byte

Príklad prevodu:

desítkové císlo 49,65625 prevést na Floating Point formát používaný vmatematické knihovne

1)

49:2 = 24 zbytek 1 24:2 = 12 zbytek 0

12:2 = 6 zbytek 0 3:2 = 1 zbytek 1

6:2 = 3 zbytek 0 1:2 = 0 zbytek 1

tedy celá část 49,0 je v binárním kódu 110001

2)

$0{,}65625 * 2 = 1{,}3125$	$1{,}3125 \geq 1$	sepíšeme	1
$1{,}3125 - 1 = 0{,}3125$			
$0{,}3125 * 2 = 0{,}625$	$0{,}625 < 1$	sepíšeme	0
$0{,}625 * 2 = 1{,}25$	$1{,}25 \geq 1$	sepíšeme	1
$1{,}25 - 1 = 0{,}25$			
$0{,}25 * 2 = 0{,}5$	$0{,}5 < 1$	sepíšeme	0
$0{,}5 * 2 = 1{,}0$	$1 = 1$	sepíšeme	1

Desetinná část 0,65625 je tedy v binárním kódu 10101

3) urcíme exponent tzn. o kolik míst musíme posunout desetinnou cárku tak, aby byla za nejvýznamnejším bitem.

110001,10101 => 1,1000110101 => 5 míst

Exponent je 5. Po prictení +127 dostaneme 132 což je binárne 10000100

4) protože je císlo kladné znaménkový byte bude 00h

Výsledek prevodu císla 49,6525 na binární tvar FP císla vypadá takto:

$1000\ 0100\ 1100\ 0110\ 1010\ 0000\ 0000\ 0000\ 0000\ 0000_{b} = 84C6A00000_{h}$

Pro lepší orientaci je výsledek znázornen v tabulce

císlo bitu	39-32 (exponent)	31 – 8 (mantisa)	7 – 0 (znaménkový byte)
hodnota (binárne)	10000100	11000110101000000000000	00000000
hodnota (hexadecimálne)	84	C6A000	00

tab. 3.8. Zápis císla 49,625 v matematické knihovne

II. PRAKTICKÁ ČÁST

4 UŽIVATELSKÝ POPIS PODPROGRAMU MATEMATICKÉ KNIHOVNY

4.1 Funkce pro prevody

4.1.1 ASC2FLT

Funkce ASC2 FLT prevádí textový retezec na císlo spohyblivou desetinnou cárkou uložené v akumulátoru FLTACC1. Absolutní hodnota maxima vkládaného císla je približne $3{,}4\times10^{38}$, minimální pak $1{,}17549\times10^{-38}$. Obsah FLTACC2 zustává po ukoncení podprogramu zachován.

Každý textový retezec musí být zakoncen ukoncovacím znakem a nesmí obsahovat mezery. Teoreticky stací pokud je ukoncovací znak rozdílný od ASCII hodnot císlic („0“=030_h; „9“=039_h) nebo znaku „E“ (45_h), „e“ (65_h), „.“ ($2E_h$), „+“ ($2B_h$), „-“ ($2D_h$). Ovšem pro lepší názornost by mel být ukoncen znakem 00_h.

Deklarace textových retezcu urcených k prevodu na formát s plovoucí desetinnou cárkou muže tedy vypadat napríklad takto:

```
cislo1      ds      '49.26'          cislo4      ds      '12.1E15'
            fcb     0                            fcb     0

cislo2      ds      '-0.656'         cislo5      ds      '-.525'
            fcb     0                            fcb     0

cislo3      ds      '15.3e-12'       cislo6      ds      '-1E-14'
            fcb     0                            fcb     0
```

Obr. 4.1 Príklady deklarace textových retezcu

Pokud procedura skoncí úspešne, akumulátor FLTACC1 obsahuje hodnotu ekvivalentní textovému retezci. Jestliže je behem prevodu nalezena chyba v textovém retezci, pak je nastaven príznak chyby (carry bit) a v akumulátoru A navrácen chybový kód. Obsah akumulátoru FLTACC1 muže obsahovat nulu, ale také nesmyslná data.

4.1.2 FLT2ASC

Funkce FLT2ASC prevádí císlo s plovoucí desetinnou cárkou uložené v akumulátoru FLTACC1 na textový retezec o maximální velikosti 14 bytu. Vstupním parametrem této funkce je registr H:X, který obsahuje ukazatel na rezervovanou oblast dat, ve které bude výsledný textový retezec uložen. Výstupem této funkce je opet ukazatel na textový retezec. Tato funkce nemuže skoncit nestandardne.

Výsledný textový retezec vytvorený pomocí tohoto podprogramu bude mít následující strukturu:

- první znak je vyhrazen pro znaménko (mínus nebo mezera)
- maximálne osm znaku je vyhrazeno pro desítkovou cást i s desetinnou teckou
- maximálne ctyri znaky vyhrazeny pro desítkový exponent (E-12; E9; E-7)
- poslední byte je vyhrazen pro ukoncovací znak (00h)

Výsledný retezec max. 14 bytu													
1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**
-	3	.	1	4	1	5	00h						
	8	.	2	5	6	E	-	2	3	00h			
-	1	.	2	3	4	5	6	8	E	-	1	8	00h
	1	4	4	00h									
	.	2	3	8	8	7	00h						

pozn. 00h – znak konce stringového retezce

tab. 4.1.Ukázky výsledku prevodu reálného císla na string

4.1.3 UINT2FLT

Funkce UINT2FLT prevede šestnácti-bitové celé císlo bez znaménka, na reálné císlo ve tvaru požadovaném matematickou knihovnou. Maximální rozsah vstupních hodnot této funkce je 0 – 65535. Jako vstupní parametr slouží registr H:X, ve kterém je uloženo celé císlo bez znaménka urcené k prevodu. Výsledek prevodu je pak uložen v akumulátoru FLTACC1. Tato funkce nemuže skoncit s chybou.

4.1.4 SINT2FLT

Funkce SINT2FLT prevede šestnácti-bitové celé císlo se znaménkem na císlo s plovoucí desetinnou cárkou ve tvaru požadovaném matematickou knihovnou. Maximální rozsah této funkce je (-32767) – 32767. Jako vstupní parametr slouží registr H:X, ve kterém je uloženo celé císlo se znaménkem urcené k prevodu. Výstupem je císlo s pohyblivou desetinnou cárkou uložené vFLTACC1 ve formátu požadovaném matematickou knihovnou. Tato funkce nemuže skoncit s chybou.

4.1.5 FLT2INT

Funkce FLT2INT prevede císlo s plovoucí desetinnou cárkou uložené vakumulátoru FLTACC1 na šestnácti-bitové celé císlo uložené v registru H:X. Pokud je císlo uložené v tomto akumulátoru kladné je rozsah takto prevedeného císla 0 - 65535. Je-li ovšem císlo v akumulátoru záporné je maximální rozsah prevádeného císla (-1) – (-32767). Pokud je císlo v akumulátoru FLTACC1 prekrocí požadovaný rozsah podprogram vrátí príznak nesprávného prevodu (carry bit) a v akumulátoru A je navrácen kód chyby.

Tato funkce nebere v úvahu zaokrouhlování hodnot cili pokud budeme chtít prevést císla 45,2 a 45,9 pak výsledek bude v obou prípadech roven 45.

4.1.6 FLT2INTROUND

Funkce FLT2INTROUND má v podstate stejné vstupy, výstupy a omezení jako funkce FLT2INT. Návratová hodnota vprípade neúspešného prenosu je rovnež shodná. Rozdíl mezi temito dvema funkcemi je ovšem vtom, že tato funkce „myslí“ na zaokrouhlení císla pred jeho prevodem. To znamená že císlo 45,2 vrátí hodnotu 45, ale císlo 45,9 vrátí hodnotu 46.

4.2 Matematické funkce

4.2.1 FLTADD, FLTSUB

Funkce FLTADD provede soucet dvou reálných císel uložených v akumulátorech FLTACC1 a FLTACC2. Jejich soucet je vrácen v akumulátoru FLTACC1. Pokud behem scítání dojde k pretecení ci podtecení výsledku je nastaven príznak chyby a v akumulátoru A je

navrácen její kód. Výsledná hodnota FLTACC1 je potom závislá na typu chyby. Jedná-li se o pretecení FLTACC1 obsahuje maximální hodnotu císla (kladné nebo záporné nekonecno), pri podtecení zásobníku obsahuje tento akumulátor nulu.

Funkce FLTSUB odcítá od akumulátoru FLTACC1 hodnotu FLTACC2. Podíl techto hodnot je opet uložen ve FLTACC1. Reprezentace chybového ukoncení je shodná s funkcí FLTADD.

4.2.2 FLTMULT

Funkce FLTMUL vynásobí císlo uložené v akumulátoru FLTACC1 císlem v akumulátoru FLTACC2. Výsledek této funkce je predán zpet v akumulátoru FLTACC1. Pokud dojde k pretecení nebo podtecení hodnoty výsledku je v akumulátoru FLTACC1 vrácena maximální hodnota císla s plovoucí desetinnou cárkou (kladné nebo záporné nekonecno) nebo nula. Zároven je nastaven príznak chyby násobení (carry bit) a v akumulátoru A je vrácen kód chyby.

4.2.3 FLTDIV

Funkce FLTDIV vydelí císlo uložené ve FLTACC1 císlem uloženým ve FLTACC2. Podíl techto hodnot je vrácen v akumulátoru FLTACC1. Pokud je zaregistrován pokus o delení nulou, je nastaven príznak chyby a podprogram je beze zmeny hodnot obou akumulátoru ukoncen. Pokud dojde k pretecení nebo podtecení výsledku je hodnota FLTACC1 maximální (kladné nebo záporné nekonecno) nebo nulová. Opet je nastaven príznak chybného výpoctu a v akumulátoru A je vrácen kód chyby.

4.2.4 FLTSQRT

Funkce FLTSQRT, jak už název napovídá, vrátí druhou odmocninu císla uloženého ve FLTACC1. Pokud je císlo v akumulátoru záporné pak funkce nastaví príznak chyby výpoctu (carry bit) a v akumulátoru A vrátí kód chyby.

4.2.5 FLTABSx, FLTCHSIGx, FLTSGNx (x = 1 nebo 2)

Všechny níže popsané funkce nemají žádné vstupní parametry ani žádné výstupní parametry.

Funkce FLTABS1 a FLTABS2 vrátí absolutní hodnotu císla s plovoucí desetinnou cárkou. Akumulátor, u kterého se provádí zmena, je urcen koncovým císlem v názvu funkce.

Funkce FLTCHSIG1 a FLTCHSIG2 zmení u požadovaného akumulátoru znaménko na opacné. Zmena se opet týká akumulátoru urceného koncovým císlem v názvu funkce.

Funkce FLTSGN1 a FLTSGN2 provedou vyhodnocení funkce SGN (x). Tato funkce vrátí 1 jedná-li se o kladné císlo a (-1) jedná-li se o císlo záporné. Pokud je císlo nula vrátí tato funkce nulu. Zmena se do tretice týká jen akumulátoru uceného koncovým císlem funkce.

4.2.6 FLTROUND

Funkce FLTROUND zaokrouhlí císlo vakumulátoru FLTACC1 na celé císlo. Funkce nemá žádné vstupy ani výstupy.

4.3 Ostatní funkce

4.3.1 GETFLTACCx (x =1 nebo 2)

Funkce GETFLTACC1 a GETFLTACC2 slouží k nactení hodnot akumulátoru FLTACC1 a FLTACC2 zadaných pomocí ctyr-bytového císla s plovoucí desetinnou cárkou uložené ve standardizovaném formátu podle normy IEEE 754. Vstupním parametrem techto funkcí je adresa ctyr-bytového prostoru predaná v registru H:X, ve kterém je normalizované císlo uloženo.

4.3.2 PUTFLTACCx (x = 1 nebo 2)

Funkce PUTFLTACC1 a PUTFLTACC2 slouží k uložení hodnot akumulátoru FLTACC1 a FLTACC2 ve ctyr-bytovém formátu císla s plovoucí desetinnou cárkou ve standardizované forme podle normy IEEE 754. Vstupním parametrem techto funkcí je adresa ctyr-bytového prostoru predaná v registru H:X, do kterého se císlo s plovoucí desetinnou cárkou bude ukládat.

4.3.3 PSHACCx (x = 1 nebo 2)

Funkce PSHACC1 a PSHACC2 slouží, jak název napovídá, k uložení zálohy akumulátoru na zásobníku. Funkce nemá žádné vstupy ani výstupy. Obsah akumulátoru A je po prubehu zachován. Registr H:X ovšem zachován není.

4.3.4 PULACCx (x = 1 nebo 2)

Funkce PULACC1 a PULACC2 slouží k nactení zálohy akumulátoru ze zásobníku. Funkce nemá žádné vstupy ani výstupy. Obsah akumulátoru A není prubehem funkce zmenen což ovšem neplatí pro indexový registr H:X

4.3.5 FLTMOV1TO2, FLTMOV2TO1, CH1AND2

Funkce FLTMOV1TO2 a FLTMOV2TO1 slouží k prepsání hodnot v akumulátorech FLTACC1 A FLTACC2 akumulátorem opacným. Tyto funkce neovlivní obsah akumulátoru A ani obsah indexového registru H:X.

Funkce CH1AND2 mezi sebou prohodí obsahy akumulátoru FLTACC1 A FLTACC2. Tato funkce také nemení obsah akumulátoru A, ani obsah indexového registru H:X.

4.3.6 FLTCMP

Funkce FLTCMP porovnává obsah akumulátoru FLTACC1 sobsahem akumulátoru FLTACC2. Pokud jsou tyto akumulátory shodné je ve stavovém registru CCR nastaven príznak nuly (tj. Zbit je nastaven na 1). Je-li FLTACC1 menší než FLTACC2, pak je ve stavovém registru CCR nastaven príznak záporného císla (tj. N bit je nastaven na 1). Carry bit a príznak pretecení (C, V) jsou na konci podprogramu vždy nulové. Pokud jsou po prubehu funkce príznak nuly a príznak záporného císla nulové, je FLTACC1 vetší než FLTACC2.

4.3.7 INTFRAC

Funkce INTFRAC rozdelí císlo uložené vakumulátoru FLTACC1 na cást celou a cást desetinnou. Pokud desetinná cást existuje, je její hodnota vrácena vakumulátoru FLTACC1. Existuje-li celá cást pak je vrácena v akumulátoru FLTACC2. Nastane-li prípad, že jedna z cástí neexistuje, pak príslušný akumulátor obsahuje nulu.

5 PRINCIPIÁLNÍ POPIS PODPROGRAMU MATEMATICKÉ KNIHOVNY

5.1 Funkce pro prevody

5.1.1 ASC2FLT

Funkce ASC2FLT provádí konverzi FP císla zadaného textovým retezcem na 5-bytový formát reálného císla používaný matematickou knihovnou.

Jako vstupní parametr funkce je použit indexový registr H:X, který obsahuje ukazatel na první znak prevádeného retezce. Výstupem je FLTACC1. Pokud prevod neprobehne v porádku, je nastaven príznak chybného ukoncení programu (carry bit) a vakumulátoru A je vrácen kód chyby.

Po zavolání procedury se nejprve do zásobníku uloží ukazatel na textový retezec. Dále se do zásobníku uloží FLTACC2, protože pri behu procedury pro prevod bude používán a musí být zajišteno jeho restaurování pri ukoncení procedury. Nakonec jsou do zásobníku uloženy 2 byty, které budou pozdeji sloužit jako lokální promenné. Po inicializaci zásobníku je vynulován FLTACC1.

Pocet Bytu	Význam
2	Návratová adresa pro instrukci RTS
2	Ukazatel na retezec urcený k prevodu
4	Záloha FLTACC2 ve ctyr-bytovém formátu
1	Lokální promenná – udává znaménko desítkového exponentu (EXPSIGN)
1	Lokální promenná – udává velikost desítkového exponentu
1	Není rezervován pri inicializaci zásobníku. Slouží pouze k docasnému uložení hodnot za behu podprogramu

tab. 5.1. Využití zásobníku funkcí ASC2FLT

Po inicializaci je provedena kontrola správnosti formátu zadaného retezce. Pokud je zadán nesprávne je nastaven príznak chybného ukoncení programu (Carry bit) a v akumulátoru A je vrácena hodnota odpovídající chybe prevodu.

Pokud kontrola probehla vporádku pokracuje se vprevodu podle formátu zadaného retezce. Byl-li zadán retezec s cástí celou a desetinou, je nejprve prevedena celá cást desetinného císla.

Samotný prevod císla funguje na principu postupného ctení znaku z leva smerem do prava a pricítáním ekvivalentních hodnot k mantise. Nejprve je puvodní mantisa uschována do FLTACC2. Poté je vynásobena deseti a nakonec je prictena hodnota odpovídající ASCII znaku, který chceme vložit. Pokud dojde k pretecení mantisy, pak je její puvodní hodnota zpetne vložena z FLTACC2 a v prevodu se pokracuje dále bez pridávání hodnot do mantisy. Jsme-li u prevodu desetinné cásti retezce pak je pri každém prictení k mantise navíc dekrementován exponent FLTACC1.

Pokud prevod narazí v retezci na znak „e“ nebo „E“ pokracuje prevodem této cásti retezce. V této fázi jsou využity lokální promenné EXPSIGN a PWR10EXP. Je-li hned za znakem „e“ znaménko mínus, je promenná EXPSIGN nastavena na FFh. V ostatních prípadech zustává hodnota této promenné nezmenena. Poté je exponent preveden a výsledek prevodu uložen v promenné PWR10EXP. Pokud má promenná EXPSIGN hodnotu FFh dochází k negaci výsledku uloženého v PWR10EXP. Následne je hodnota této promenné prictena k exponentu FLTACC1.

Po ukoncení nacítání retezce se nejprve otestuje FLTACC1, nemá-li nulovou hodnotu. Pokud ano procedura skocí na ukoncovací cást podprogramu. Pokud je nenulový, dojde k uschování exponentu do lokální promenné PWR10EXP. Poté je inicializována hodnota exponentu FLTACC1 na hodnotu 7Eh (exponent je nula) + maximální posun 18h (max. posun mantisy je 24 bitu doleva) a zavolán podprogram pro normalizaci FP císla.

Normalizace posouvá mantisu vlevo dokud není nejvyšší bit mantisy roven jedné a zároven pri každém posunu dekrementuje exponent FLTACC1.

Po normalizaci je testována promenná PWR10EXP. Je-li nulová, procedura skocí na ukoncovací cást. Je-li kladná, je do FLTACC2 vložena hodnota 10,0. V opacném prípade je vložena hodnota 0,10 a poté negována hodnota vPWR10EXP. Dále je FLTACC1 násoben FLTACC2 dokud není promenná PWR10EXP nulová.

Pri ukoncování podprogramu pro prevod ASCII retezce na reálné císlo jsou nejprve ze zásobníku odebrány pomocné promenné, následne je obnoven akumulátor FLTACC2 a ze zásobníku je odebrán ukazatel na ASCII retezec. Nakonec jsou smazány príznak chyby (Carry bit) a akumulátor A.

5.1.2 FLT2ASC

Funkce FLT2ASC provádí konverzi FLTACC1 na textový retezec o velikosti maximálne 14 bytu, což je dále možno použít napríklad k zobrazení výsledku na LCD display.

Jako vstupní parametr funkce slouží FLTACC1. Dále je použit ukazatel na rezervovaný 14-ti bytový buffer predaný vregistru H:X. Návratovou hodnotou je ukazatel na první znak prevedeného stringu, opet predaný v registru H:X. Tvar výstupního textového retezce je podrobne popsán v uživatelské casti této práce. Tato funkce nemuže být ukoncena s chybou.

Po zavolání procedury je nejdríve uložen na zásobník ukazatel na buffer do kterého se budou vkládat jednotlivé ASCII znaky. Poté je otestován FLTACC1 na nulovou hodnotu. Pokud FLTACC1 obsahuje nulu, je do bufferu zapsán ASCII kód nuly a ukoncovací znak stringu s hodnotou 00h. Je-li FLTACC1 rozdílný od nuly, prijde na radu inicializacní cást programu.

V inicializacní casti programu jsou nejprve na zásobník uschovány oba akumulátory pro práci s reálnými hodnotami uloženými ve formátu plovoucí desetinné cárky. Následne je vyhrazeno 5 bytu pro lokální promenné.

Pocet Bytu	Význam
2	Návratová hodnota pro instrukci RTS
1	Ukazatel na textový buffer
5	Záloha FLTACC1 v peti-bytovém formátu
4	Záloha FLTACC2 ve ctyr-bytovém formátu
1	Lokální promenná – cítac znaku
1	Lokální promenná – akumulátor desítkového císla
1	Lokální promenná – cítac znaku za desetinou teckou
1	Lokální promenná – cítac násobení/delení
1	Lokální promenná – ukazatel na další znak v bufferu
2	Nejsou rezervovány pri inicializaci zásobníku. Slouží pouze k docasnému uložení hodnot za behu podprogramu

tab. 5.2. Využití zásobníku funkcí FLT2ASC

Vlastní podprogram nejdríve otestuje znaménko reálného císla uloženého v akumulátoru FLTACC1. Jedná-li se o záporné císlo, je jako první znak retezce uloženo znaménko mínus. V opacném prípade je znaménko vynecháno. Dále podprogram pro prevod upraví FLTACC1 násobením hodnotami 0,1 nebo 10, tak aby výsledná velikost císla uloženého v tomto akumulátoru byla v rozmezí 999999,9 - 9999999. Cítac násobení/delení obsahuje po této úprave pocet provedených kroku. Poté je k FLTACC1 prictena hodnota 0,5 pro zaokrouhlení FP císla. Do akumulátoru A je uložen exponent FLTACC1 a z jeho hodnoty je odecten základ exponentu 07Fh. Výsledek je negován a je zvýšen o maximální velikost mantisy zmenšenou o jednicku ($23_d = 17_h$). Pokud není akumulátor A roven nule pokracuje podprogram smyckou, která rotuje mantisu FLTACC1 smerem doprava a dekrementuje akumulátor A. Po této procedure dochází k inicializaci cítace znaku za desetinnou teckou. Je-li cítac menší jak nula (tzn. jeho hodnota je záporná), pak prevádené císlo je v absolutní hodnote menší než 1. V tomto prípade je jako další znak do textového retezce vložena desetinná tecka.

Dále se pokracuje s pomocí tabulky decdig, ve které jsou uloženy konstanty pro delení mantisy v logaritmické posloupnosti. Nejprve se od císla uloženého v FLTACC1 postupne opakuje odectení aktuální hodnoty z tabulky. Pocet takto provedených odectení se pocítá v decimálním akumulátoru. Pokud dojde pri odcítání k podtecení je tato procedura ukoncena a posledne odecítaná hodnota je zpetne pripoctena ke zbytku v FLTACC1. V desítkovém akumulátoru dostaneme pocet probehlých odectení (bez posledního ve kterém došlo k pretecení). Prictením hodnoty 030h k desítkovému akumulátoru dostaneme ASCII kód znaku, který bude vložen do textového retezce. Je-li v tomto bode cítac znaku za desetinou teckou nulový, vložíme do retezce znak desetinné tecky. Dále zvýšíme ukazatel na tabulku decdig a po kontrole, není-li nulový cítac znaku, pokracujeme v prevodu.

Dosáhne-li cítac znaku nuly, provede se kontrola retezce. Je-li na jeho konci nula, pokracuje se od konce retezce smerem na zacátek a zkoumá se zda-li není ASCII znaku odpovídajících nule víc. Pokud ano, jsou tyto z výsledného retezce vynechány.

Jestliže je hodnota cítace násobení/delení nulová, preskocí se vyhodnocování exponentu a program je korektne ukoncen. Má-li tento cítac nenulovou hodnotu, pak je exponent vyhodnocen a pripojen na konec textového retezce.

Korektní ukoncení podprogramu vloží na konec retezce ukoncovací znak 00h, odebere ze zásobníku všechny lokální promenné, obnoví stav akumulátoru FLTACC1 a FLTACC2 a do registru H:X vloží ukazatel na textový retezec.

5.1.3 UINT2FLT

Podprogram UINT2FLT prevádí šestnácti-bitové celé císlo bez znaménka na císlo s plovoucí desetinou cárkou uložené v FLTACC1. Jako vstupní parametr je použit registr H:X, ve kterém je uloženo celé císlo urcené k prevodu na formát pro matematickou knihovnu a jako výstup slouží FLTACC1. Tento podprogram nemuže skoncit s chybou.

Pri zavolání podprogramu se nejprve na zásobník uloží císlo urcené k prevodu. Poté je císlo otestováno není-li nulové. Pokud ano, je vložena nula do akumulátoru FLTACC1 a podprogram je po restaurování zásobníku ukoncen.

Pokud císlo v registru H:X je rozdílné od nuly, je uloženo do mantisy FLTACC1. Následne je mantisa normalizována tak, aby nejvyšší bit mantisy byl 1. Výsledek normalizace je poté uložen do akumulátoru FLTACC1. Nakonec je ze zásobníku odebráno prevádené císlo a jsou aktualizovány exponent a znaménkový byte akumulátoru FLTACC1.

5.1.4 SINT2FLT

Podprogram SINT2FLT prevádí šestnáctibitové celé císlo se znaménkem na reálné císlo uložené v FLTACC1. Jako vstupní parametr je použit registr H:X, ve kterém je prevádené císlo uloženo a jako výstup slouží akumulátor FLTACC1. Tento podprogram nemuže skoncit s chybou.

Pri zavolání podprogramu je nejdríve otestováno jedná-li se o záporné císlo. Jestliže je císlo kladné, následuje skok na podprogram pro prevod celého císla bez znaménka. Pokud je prevádené císlo záporné, podprogram uloží toto císlo na zásobník. Poté je proveden dvojkový doplnek prevádeného císla. Výsledek je zvýšen o jednicku címž dostaneme absolutní hodnotu prevádeného císla. Nakonec je inicializována promenná urcující znaménko a hodnotu exponentu císla s plovoucí desetinou cárkou a podprogram pokracuje dále od náveští umísteného ve funkci UINT2FLT.

5.1.5 FLT2INT

Podprogram FLT2INT prevádí FP císlo uložené v akumulátoru FLTACC1 na šestnáctibitové celé císlo se znaménkem. Vstupním parametrem je císlo s plovoucí desetinnou cárkou uložené v akumulátoru FLTACC1. Výstupní parametr je šestnáctibitové celé císlo uložené v registru H:X. Tento podprogram nebere v úvahu zaokrouhlování hodnot. To znamená, že je na celé císlo prevedena pouze celá cást reálného císla. Na desetinnou cást není brán žádný zretel.

Na zacátku je vytvoren dvou-bytový pracovní prostor na zásobníku. Porovnáním exponentu daného reálného císla s hodnotou 7Eh zjistíme, jedná-li se o císlo s celou cástí. Pokud má exponent nižší hodnotu jedná se o ryze desetinné císlo, do výsledkového registru vložena hodnota nula a program je korektne ukoncen.

Pokud reálné císlo obsahuje celou cást pak pokracujeme v prevodu. Exponent je uložen do akumulátoru A, je z nej vytvoren dvojkový doplnek a jeho hodnota je zvýšena o 144. Je-li výsledek menší nebo roven nule, jedná se o císlo vetší než maximální rozsah celého císla bez znaménka. Podprogram je ukoncen s chybou. Kód chyby je uložen v akumulátoru A a výsledkový registr H:X je vynulován. V opacném prípade je výsledkem pocet nutných posunu smerem doprava. Tato hodnota je uložena do registru X pro pozdejší zpracování.

V tomto bode jsou nacteny horní dva byty mantisy a uloženy do pracovního prostoru. Je opakována postupná rotace smerem vpravo dokud není promenná v registru X nulová. Tímto postupem získáme celou cást desetinného císla zarovnanou vpravo. Pokud je znaménko císla v FLTACC1 kladné, muže být prevod korektne ukoncen. Je-li záporné, vytvoríme dvojkový doplnek pracovního prostoru a zvýšíme jeho hodnotu o jednicku. Pokud je výsledek dvojkového doplnku kladný, byla prekrocena absolutní hodnota maximálního rozsahu celého císla se znaménkem. Podprogram vrátí prázdný výsledkový registr H:X a v akumulátoru A je navrácen kód chybového hlášení.

Pri korektním ukoncení podprogramu je výsledek prevodu odebrán ze zásobníku do výsledkového registru H:X.

5.1.6 FLT2INTROUND

Podprogram FLTROUND vrací vregistru H:X zaokrouhlené šestnáctibitové celé císlo uložené v akumulátoru FLTACC1.

Na zacátku podprogramu je na zásobníku inicializován prostor pro výsledek prevodu. Poté jsou na zásobníku uloženy oba akumulátory pro císla s plovoucí desetinnou cárkou. Do FLTACC2 je vložena konstanta 0,5 a následne je tato hodnota prictena k akumulátoru FLTACC1. Poté následuje prevod císla ve FLTACC1 na šestnáctibitové celé císlo. Pokud prevod probehl v porádku je výsledek uschován z registru H:X do lokální promenné na zásobníku, protože pri pokracování bude jeho hodnota znicena. To je zpusobeno obnovením puvodních hodnot akumulátoru FLTACC1 a FLTACC2 ze záloh na zásobníku. Poté je výsledek vložen ze zásobníku zpet do registru H:X a podprogram muže být ukoncen.

Pokud došlo pri prevodu na celé císlo k chybe, jsou ze zásobníku obnoveny akumulátory FLTACC1 a FLTACC2. Do registru H:X je vložena nula, nastaven príznak chyby a podprogram muže být ukoncen.

5.2 Matematické funkce

5.2.1 FLTADD, FLTSUB

Funkce FLTSUB slouží k odcítání dvou císel s plovoucí desetinnou cárkou. Tato funkce je velmi jednoduchá. Nejdríve zmení znaménko FLTACC2 poté zavolá funkci pro scítání a nakonec vrátí puvodní znaménko do FLTACC2.

Funkce FLTADD secte císla s plovoucí desetinnou cárkou v akumulátorech FLTACC1 a FLTACC2. Pokud dojde k pretecení nebo podtecení výsledku je vrácena v akumulátoru FLTACC1 hodnota nula nebo maximální hodnota císla s plovoucí desetinnou cárkou (kladné nebo záporné nekonecno). V akumulátoru A je uložen kód vzniklé chyby.

Pri zavolání podprogramu je nejdríve na zásobník uložena hodnota stack pointeru. Dále jsou na zásobníku inicializovány tri lokální promenné. Za behu podprogramu jsou na zásobník pridávána další data pro správnou funkci podprogramu. Maximální obsazení zásobníku je zobrazeno v tabulce.

Pocet Bytu	Význam
2	Návratová hodnota pro instrukci RTS
1	Ukazatel na návratovou adresu
1	Promenná obsahující znaménko výsledku
1	Promenná obsahující ukazatel na mantisu akumulátoru urcenou k rotaci
1	Promenná obsahující ukazatel na mantisu druhého akumulátoru
1	Pocítadlo nutných posunu mantisy
6	Pracovní prostor pro funkci
1	Není nikde rezervován. Slouží pouze k docasnému uložení hodnot za behu podprogramu

tab. 5.3 Využití zásobníku funkcí FLTADD

Nejdríve je otestován exponent akumulátoru FLTACC1. Pokud je nulový podprogram zkopíruje FLTACC2 do FLTACC1 a je ukoncen. Dále je otestován FLTACC2 a obsahuje-li nulu, je výsledkem FLTACC1 a podprogram je ukoncen.

Poté podprogram urcí podle znamének obou akumulátoru, jedná-li se o scítání (obe znaménka jsou stejná), nebo o odcítání (znaménka akumulátoru jsou rozdílná). Výsledek porovnání znamének je uložen do lokální promenné.

Dále je od exponentu FLTACC1 odecten exponent FLTACC2. Pokud je po odcítání nastaven carry bit, je císlo uložené ve FLTACC2 vetší než FLTACC1, výsledek je negován a porovnán s maximální hodnotou posunu mantisy. Pokud je výsledek vyšší, je akumulátor FLTACC2 presunut do FLTACC1 a podprogram je ukoncen. V opacném prípade je do lokální promenné na zásobníku uložena hodnota pocítadla posunu mantisy. Také jsou do lokálních promenných uloženy adresy mantis akumulátoru.

Je-li výsledek odcítání exponentu záporný, je výsledek porovnán s maximální hodnotou posunu mantisy. Pokud je výsledek vyšší je podprogram ukoncen. V opacném prípade je do lokální promenné uložena hodnota pocítadla posunu mantisy. Také jsou do lokálních promenných uloženy adresy mantis akumulátoru.

První tri byty pracovního prostoru jsou inicializovány hodnotou znaménka uloženého v lokální promenné na zásobníku. Do registru H:X je nacten ukazatel na mantisu akumulátoru urceného k posouvání. Celá mantisa tohoto exponentu je násobena od nejvyššího bytu po nejnižší hodnotami znaménka uloženými v pracovním prostoru a tvorí dalších 3 byty pracovního prostoru. Dále je mantisa posunuta v pracovním prostoru tak, aby jednotlivé bity mantisy svou váhou

odpovídaly bitum v mantise druhého císla. Následne jsou obe mantisy secteny a výsledky uloženy do akumulátoru FLTACC1.

Nyní je otestováno znaménko uložené v lokální promenné. Pokud je kladné a zároven carry bit není po posledním scítání nastaven, podprogram muže skoncit. Pokud je carry bit nastaven pak je rotací doprava vložen do mantisy a exponent je inkrementován a pokud nedosáhl nulové hodnoty, podprogram muže skoncit. Jestliže exponent dosáhl nuly, došlo k pretecení mantisy, podprogram vloží do FLTACC1 maximální hodnotu (kladné nebo záporné nekonecno), nastaví kód chyby i príznak chyby a je ukoncen.

Je-li ovšem znaménko mínus a carry bit není nastaven, je zmeneno znaménko FLTACC1 na opacné. Dále je provedena zmena znaménka celé mantisy. Program pokracuje na test normalizace mantisy.

Bylo li znaménko mínus a carry bit byl nastaven, není potreba menit znaménko FLTACC1 a je otestováno jestli mantisa výsledku potrebuje normalizaci. Pokud normalizace není potreba, podprogram je ukoncen. Má-li být normalizace provedena, je otestován nejvyšší byte mantisy na nulu. Je-li skutecne nulový, nastává hrubá normalizace mantisy. Jsou posunuty oba dva spodní byty mantisy o byte výše a spodní byte je smazán. Znovu je zkontrolováno, není-li nejvyšší byte nulový, pokud ano je prostrední byte mantisy presunut na horní byte mantisy. Pri techto krocích je exponent vždy snížen o osm a pokud došlo k jeho pretecení je vrácena v akumulátoru FLTACC1 nula a podprogram je ukoncen s chybou.

Pokud pro hrubé normalizaci není císlo stále normalizováno, je pristoupeno k normalizaci jemné. Tato posouvá mantisu o jeden bit doleva dokud není první bit mantisy jedna. Zároven s posuny je exponent snižován o jednicku a hlídán jestli nedošlo k pretecení. Jestliže k pretecení došlo, je ve FLTACC1 vrácena maximální hodnota akumulátoru a podprogram je ukoncen s chybou. Pokud vše probehlo v porádku muže být podprogram pro scítání po odstranení lokálních promenných a pracovního prostoru ze zásobníku standard ne ukoncen

5.2.2 FLTMUL

Funkce FLTMUL vynásobí akumulátor FLTACC1 akumulátorem FLTACC2. Pokud dojde k pretecení nebo podtecení pri násobení, program nastaví príznak chyby a vrátí kód chyby v akumulátoru A.

Podprogram nejdríve otestuje oba akumulátory na nulovou hodnotu. Je-li alespon jeden z nich nulový, podprogram vloží nulu do FLTACC1 a je ukoncen.

Pokud jsou oba akumulátory nenulové je urceno znaménko výsledku a uloženo do znaménkového bytu FLTACC1. Dále jsou secteny exponenty obou akumulátoru. Pokud je výsledek záporný a carry bit je nastaven, došlo k pretecení, podprogram vloží do FLTACC1 maximální hodnotu (kladné nebo záporné nekonecno) a je ukoncen schybou. Je-li výsledek scítání kladný a carry bit není nastaven, pak došlo k podtecení císla, podprogram vloží do FLTACC1 nulu a je ukoncen s chybou.

Jestliže nedošlo k pretecení ani podtecení výsledku souctu zásobníku, pokracujeme v prevodu inicializací exponentu prictením hodnoty 82h a jeho uložením do FLTACC1. Na zásobníku jsou inicializovány 4 byty jako pracovní prostor. Poté je zavolána funkce na celocíselné násobení obou mantis.

Výsledkem celocíselného násobení obou mantis je 6- bytové celé císlo. Protože jsou ale spodní dva byty po ukoncení násobení nepotrebné (do mantisy se mohou vložit maximálne 3 byty), jsou zahozeny a výsledek tohoto násobení je omezen pouze na tri horní byty plus jeden, který bude pozdeji treba k zaokrouhlení výsledku.

Po provedení celocíselného násobení je výsledek otestován, obsahuje-li na nejvyšším bitu jednicku. Pokud ne, je celý ctyr-bytový prostor rotován doleva a inkrementován exponent. Obsahuje-li nejnižší byte výsledku na nejvyšším bitu jednicku, je ješte potreba mantisu zaokrouhlit. Horní tri byty výsledku jsou poté presunuty do mantisy akumulátoru FLTACC1 a poslední ctvrtý byte je zahozen. Pokud zustal po zaokrouhlování exponent nulový, došlo k pretecení císla, do FLTACC1 je nactena nula a podprogram je ukoncen s chybou. V opacném prípade probehlo scítání v porádku a podprogram je standard ne ukoncen.

5.2.3 FLTDIV

Funkce FLTDIV vydelí akumulátor FLTACC1 akumulátorem FLTACC2. Dojde-li v prubehu funkce k chybe, je nastaven príznak nesprávného ukoncení a v akumulátoru A je navrácen kód této chyby.

Na zacátku podprogramu je testován FLTACC2 na nulou. Pokud je tento delitel nulový je podprogram ukoncen schybou a vakumulátoru A je vložen chybový kód. Dále je na nulu otestován FLTACC1 a pokud nulu obsahuje je podprogram ukoncen bez chyby.

Jestliže není ani jeden z akumulátoru roven nule, podprogram pokracuje uschováním obsahu akumulátoru FLTACC2 na zásobníku. Poté je urceno znaménko výsledku, je uloženo do znaménkového bytu akumulátoru FLTACC1 a na zásobníku je vytvoren šesti-bytový pracovní prostor. Nakonec je na zásobníku vytvorena lokální promenná cítace prubehu podprogramu inicializovaná na hodnotu 24 a pomocný jednobitový registr sloužící kuschovávání hodnot za behu podprogramu.

Pocet bytu	Význam
2	Návratová hodnota pro instrukci RTS
4	Záloha akumulátoru FLTACC2
6	Pracovní prostor pro funkci FLTDIV
1	Cítac prubehu podprogramu
1	Pomocný registr

tab. 5.4 Využití zásobníku funkcí FLTDIV

Následuje porovnání mantis obou akumulátoru. Pokud je mantisa FLTACC2 vetší než mantisa FLTACCC1, je exponent FLTACC2 zvýšen o jednicku a pokud nedošlo k pretecení, pokracuje rotací mantisy FLTACC2. Dále nacte exponent FLTACC2, neguje jej, a od výsledku odecte exponent FLTACC1. Pokud je výsledek kladný a carry bit není nastaven došlo k podtecení a podprogram je ukoncen schybou. Je-li výsledek záporný a carry bit je nastaven, došlo kpretecení a podprogram je opet ukoncen schybou. Pokud po výpoctu není nastaven carry bit, k výsledku je prictena základní hodnota mantisy a výsledný exponent je uložen do FLTACC1.

Poté je provedena záloha mantisy akumulátoru FLTACC1 a následne je do této mantisy odectena mantisa akumulátoru FLTACC2. Pokud je nejvyšší bit mantisy jedna, je mantisa FLTACC1 obnovena ze zálohy. Pokud je nejvyšší bit nulový je do nejvyšší pozice výsledkové mantisy rotován carry bit a mantisa FLTACC1 je rotována doleva. Dále je dekrementován cítac prubehu programu a pokud nedosáhl nuly je celý postup opakován. Po dosažení nulové hodnoty cítace je provedeno ješte jedno odcítání mantis kvuli zaokrouhlení výsledku. Pokud je po tomto odcítání nastaven carry bit došlo kpretecení a kmantise podílu je prictena jednicka. Jinak je

mantisa podílu zkopírována ze svého místa na zásobníku do akumulátoru FLTACC1. Pokud pricítáním jednicky nedošlo kpretecení této mantisy je podprogram úspešne ukoncen. Jestliže k pretecení došlo, je carry bit rotací celé mantisy vložen na její nejvyšší bit, exponent je zvýšen o jednicku a pokud nedošlo k jeho pretecení, podprogram muže opet úspešne ukoncen.

5.2.4 FLTSQRT

Funkce FLTSQRT vypocítá druhou odmocninu zcísla uloženého ve FLTACC1. Je-li hodnota císla uložená v akumulátoru FLTACC1 záporná, podprogram skoncí s chybou a v akumulátoru A predá její kód.

Nejdríve podprogram otestuje jestli není akumulátor FLTACC1 nulový. Pokud ano, podprogram je ukoncen. Dále otestuje, není-li hodnota v akumulátoru záporná. Nastane-li takovýto prípad, do akumulátoru A je vložen kód chyby, je nastaven príznak chyby a podprogram je ukoncen.

Pokud nedošlo ani k jednomu z predchozích prípadu, je na zásobníku inicializována hodnota cítace prubehu programu na ctyri. Do zásobníku je následne uložena mantisa a exponent akumulátoru FLTACC1. FLTACC1 je zkopírován do FLTACC2.

Poté je do akumulátoru A nacten exponent FLTACC2 a je od nej odecten základ a prictena jednicka. Pokud vyjde exponent kladný je vydelen dvemi a je do nej zpet pricten základ exponentu. Jestliže vyšel exponent záporný, je pouze vydelen dvemi. Poté je nový exponent uložen zpet do FLTACC2. Probehne delení akumulátoru a k výsledku je pricten zmenený akumulátor FLTACC2. Exponent výsledku ve FLTACC1 je snížen o jednicku a následne je akumulátor FLTACC1 presunut do akumulátoru FLTACC2. Akumulátor FLTACC1 je obnoven ze zálohy na zásobníku a je dekrementován cítac prubehu. podprogramu. Celý postup se opakuje dokud není cítac prubehu roven nule.

Dojde-li cítac knule, je výsledek uložený ve FLTACC2 presunut do FLTACC1, ze zásobníku je odebrána záloha puvodního akumulátoru FLTACC1 a ze zálohy je obnovena puvodní hodnota akumulátoru FLTACC2. Nakonec podprogram smaže akumulátor A a príznak chyby a je ukoncen.

5.2.5 FLTABSx, FLTCHSIGx, FLTSGNx (x = 1 nebo 2)

Funkce FLTABS pouze zmení u císla s plovoucí desetinnou cárkou uloženého ve zvoleném akumulátoru znaménkový byte na 00h (kladné císlo).

Funkce FLTCHSGN mení u císla s plovoucí desetinnou cárkou uloženého ve zvoleném akumulátoru znaménkový byte na opacnou hodnotu (FF_h => 00_h, 00_h => FF_h).

Funkce FLTSGN nacte adresu prvního bytu zvoleného akumulátoru. Poté do tohoto akumulátoru vloží hodnotu 1.0. Znaménko daného akumulátoru zustane zachováno.

5.2.6 FLTROUND

Podprogram FLTROUND zaokrouhlí císlo ve FLTACC1 na celé císlo. Výsledek vrátí opet ve FTACC1.

Podprogram nejdríve uschová FLTACC2 na zásobníku. Poté do tohoto akumulátoru vloží konstantu 0,5 a pricte ji k akumulátoru FLTACC1. Následne provede rozdelení FLTACC1 na cást celou a cást desetinnou. Celá cást uložená ve FLTACC2 je pak zkopírována do FLTACC1. Nakonec je FLTACC2 obnoven ze zálohy uložené na zásobníku a podprogram je ukoncen.

5.3 Ostatní funkce

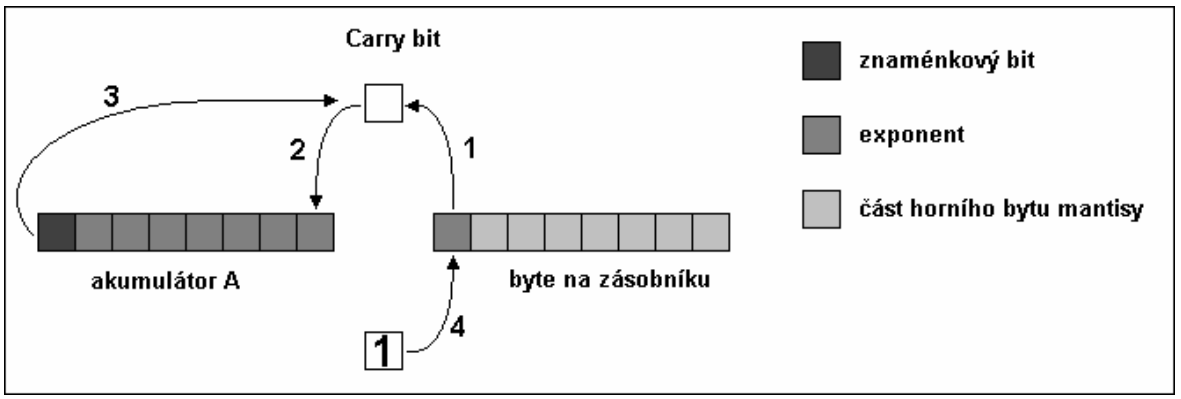
5.3.1 GETFLTACCx (x = 1 nebo 2)

Funkce GETFLTACC1 a GETFLTACC2 prevede ctyr-bytový formát císla s plovoucí desetinnou cárkou na formát používaný v matematické knihovne.

Vstupním parametrem techto funkcí je registr H:X, který ukazuje na první byte standard izovaného císla. Výstupem je císlo uložené ve zvoleném akumulátoru uložené ve formátu vyžadovaném knihovnou. Program nemuže skoncit s chybou.

Nejprve jsou nacteny spodní dva byty a následne uloženy do spodních dvou bytu mantisy akumulátoru. Poté je nacten horní byte mantisy zadaného standard izovaného císla a uložen do zásobníku. Do akumulátoru A je vložen nejvyšší byte obsahující znaménko a cást exponentu. Lokální promenná na zásobníku obsahující cást mantisy a 1 bit exponentu je rotována doleva

s prenosem do carry bitu (1). Prenos z carry bitu je dále rotován do akumulátoru A (2) (vznikne kompletní exponent). Znaménkový bit je po rotaci akumulátoru uložen v carry (3). Je-li po rotaci carry bit nastaven, jedná se o záporné císlo a do znaménkového bytu FLTACC je vložena hodnota FFh. Pokud je carry bit nulový zustává znaménkový byte nulový. Výsledný exponent uložený v akumulátoru A je uložen do exponentu FLTACC. Dále je provedena rotace vpravo a na nejvyšší bit cásti mantisy uložené na zásobníku je vložena jednicka (4). Mantisa je poté uložena na místo ve FLTACC.



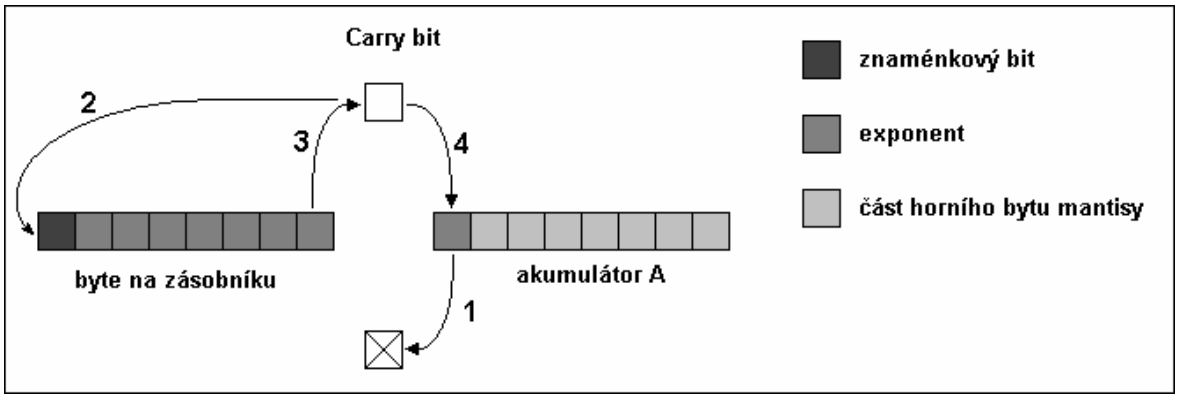
Obr. 5.1 Schéma funkce GETFLTACCx

5.3.2 PUTFLTACCx (x = 1 nebo 2)

Funkce PUTFLTACC1 a PUTFLTACC2 prevede peti-bytový formát císla s plovoucí desetinnou cárkou používaný v matematické knihovne na formát ctyr-bytový.

Vstupním parametrem techto funkcí je registr H:X, který ukazuje ukazatel na první byte oblasti, do které se bude prevedené císlo ukládat. Program nemuže skoncit s chybou.

Mechanismus této funkce je opacný než u funkce GETFLTACCx. Nejdríve se zkopírují spodní dva byty mantisy do spodních dvou bytu rezervovaného prostoru. Poté je nacten exponent FLTACC a uložen na zásobník. Do akumulátoru A je vložena hodnota nejvyššího bytu FLTACC. Tato hodnota je rotována smerem doleva a prenos nejvyššího bitu mantisy je zahozen (1). Porovnáním znaménkového bytu FLTACC je nastaven carry bit. Hodnota carry bitu je rotací doprava prenesena do exponentu uloženého v lokální promenné na zásobníku (2). Prenos nejnižšího bitu zexponentu je opet pres carry bit (3) vložen rotací doprava na pozici nejvyššího bitu mantisy (4). Oba výsledky jsou poté uloženy do prostoru vpameti vyhrazeného pro ctyr-bytový formát císla s plovoucí desetinnou cárkou.



Obr. 5.2 Schéma funkce PUTFLTACCx

5.3.3 PSHACCx (x = 1 nebo 2)

Funkce PSHACC1 a PSHACC2 slouží k uschování akumulátoru na zásobníku. Na zacátku podprogramu je ze zásobníku nactena návratová adresa pro instrukci RTS. Poté jsou na zásobníku rezervovány 4 byty pro uložení akumulátoru a vrácena návratová adresa. Ho registru H:X je uložen ukazatel na vrchol zásobníku a jeho hodnota je zvýšena o dve. Tím to dostaneme v registru H:X ukazatel na 4-bytovou oblast dat do které muže být akumulátor uložen pomocí funkce PUTFLTACCx, na kterou je pokracováno skokem. O návrat zpodprogramu se již postará funkce PUTFLTACCx

5.3.4 PULACC x (x = 1 nebo 2)

Funkce PULACC1 a PULACC2 slouží k odebrání zálohy akumulátoru ze zásobníku. Na zacátku funkce je do registru H:X uložen ukazatel zásobníku (SP) a jeho hodnota zvýšena o dve. Takto získáme ukazatel na oblast dat na zásobníku ve které je akumulátor uložen. Poté je zavolána funkce GETFLTACCx která nacte do akumulátoru hodnotu uloženou na zásobníku . Po nactení akumulátoru je ze zásobníku odebrána návratová adresa pro instrukci RTS. Ukazatel na zásobník je zvýšen o ctyri a do zásobníku je zpetne vložena návratová adresa. Poté je podprogram ukoncen.

5.3.5 FLTMOV2TO1, FLTMOV1TO2, CH1AND2

Funkce pro presuny nemají žádné vstupní ani výstupní parametry. Jejich realizace je jednoduchá. FLTMOV2TO1 a FLTMOV1TO2 premístí jeden akumulátor do druhého prostým

kopírováním dat zdrojového akumulátoru do akumulátoru cílového. Funkce CH1AND2 prohodí obsahy obou akumulátoru bez využití zásobníku. Žádná z výše popsaných funkcí neovlivní hodnoty akumulátoru A ani indexového registru H:X.

5.3.6 FLTCMP

Funkce FLTCMP provádí porovnání dvou císel s plovoucí desetinnou cárkou. Vstupem této funkce jsou akumulátory FLTACC1 a FLTACC2, výstupem pak jsou príznaky vregistru CCR.

Podprogram nejdríve prozkoumá, je-li znaménko FLTACC2 záporné. Pokud ano, pokracuje kontrolou znaménka u FLTACC2.

Jsou-li obe porovnávaná císla záporná, pokracuje se porovnáváním exponentu FLTACC2 s exponentem FLTACC1. Pokud jsou exponenty shodné, pokracuje obdobne porovnáváním všech bytu mantisy. Narazí-li na rozdílné hodnoty, skocí na vyhodnocení porovnávání a nastavení príslušných príznaku.

Jsou-li obe porovnávaná císla kladná, pokracuje se porovnáváním exponentu FLTACC1 s exponentem FLTACC2. Pri shode techto exponentu vzájemne porovnává byty mantisy od nejvyššího po nejnižší. Najde-li dva rozdílné byty, pokracuje skokem na vyhodnocení porovnávání.

Je-li císlo FLTACC1 kladné a FLTACC2 záporné, podprogram preskocí porovnávání exponentu a mantisy a skocí na vyhodnocení.

Vyhodnocení nejdríve uschová do akumulátoru A registr CCR. Poté v této záloze smaže príznak pretecení, záporné hodnoty a carry bit. Pokud je carry bit v registru CCR nulový, nastaví se vakumulátoru príznak záporného císla. Pri ukoncení podprogramu je registr CCR prepsán výslednou hodnotou uloženou v akumulátoru A.

5.3.7 INTFRAC

Funkce INTFRAC rozdelí císlo s plovoucí desetinnou cárkou v akumulátoru FLTACC1 na cást celou a cást desetinou. Celá cást je na konci podprogramu uložena ve FLTACC2 a cást desetinná ve FLTACC1.

Nejdríve podprogram zkopíruje FLTACC1 do FLTACC2. Poté nacte exponent a od jeho hodnoty odecte základ exponentu ($7E_h$) zvýšený o 2^{24} .

Pokud je výsledek kladný, císlo urcené k rozdelení neobsahuje žádnou desetinnou cást. V tomto prípade podprogram odecte FLTACC2 od FLTACC1. Výsledkem tohoto odcítání je nula ve FLTACC1 a podprogram muže být ukoncen.

Jestliže je výsledek operace sexponentem záporný, podprogram pokracuje inicializací pocítadla maximálního poctu inkrementace exponentu na hodnotu 3 a uloží tuto hodnotu do lokální promenné vytvorené na zásobníku. Do registru H:X nacte ukazatel na nejspodnejší byte mantisy akumulátoru FLTACC2.

K výsledku manipulace sexponentem uloženém vakumulátoru A je prictena hodnota 8 (posun mantisou o 8 bitu smerem doleva).

Pokud je výsledek vakumulátoru A záporný, je smazán aktuální byte mantisy, protože obsahuje desetinnou cást císla. Dále je snížen ukazatel na aktuální byte mantisy tak, že ukazuje na byte vlevo od aktuálního bytu, a je dekrementován cítac poctu inkrementací exponentu. Pokud není tento cítac roven nule pokracuje se dále prictením hodnoty osm a opakováním nekolika predchozích kroku. Dosáhne-li cítac hodnoty nula a hodnota akumulátoru A je stále záporná, jedná se o císlo bez celé cásti, do FLTACC2 je vložena nula a podprogram je ukoncen.

Pokud je výsledek vakumulátoru A kladný, pak máme vakumulátoru A uložen pocet bitu celé cásti v aktuálním bytu mantisy. V lokální promenné na zásobníku je inicializován maskovací byte. Postupnou dekrementací akumulátoru A a aritmetickou rotací lokální promenné je vytvoren maskovací byte. Logickým soucinem je z aktuálního bytu mantisy vymazána desetinná cást císla s plovoucí desetinou cárkou. Ve FLTACC2 zbývá pouze celá cást císla. Od puvodní hodnoty uložené v akumulátoru FLTACC1 je FLTACC2 odecten a ve FLTACC1 zustane pouze cást desetinná.

ZÁVER

Cílem této bakalárské práce bylo vytvorit matematickou knihovnu zdrojových kódu pro operace s reálnými císly uloženými ve formátu plovoucí desetinné cárky pro 8-bitový mikropocítac Motorola M68HC08

V teoretické cásti je popsán problém inkompatibility obou mikropocítacu a práce s císly s plovoucí desetinnou cárkou. V praktické cásti jsou popsány vytvorené funkce z hlediska uživatele i programátora. V príloze práce je pak zpracováno nekolik príkladu na použití knihovny a strucná verze uživatelského popisu funkcí. Pro úplnost je jako príloha také pridána instrukcní sada mikropocítace Motorola M68HC08.

Programování samotné matematické knihovny však prinášelo znacné problémy. Nejvetším z nich bylo omezení pri práci s jednotlivými akumulátory a registry, kdy funkce sestavené pro mikropocítacovou radu M68HC11 používaly prakticky neustále 16-bitový akumulátor D a v nekterých prípadech i soucasne oba indexové registry X a Y. Bohužel takovýmito prostredky rada M68HC08 nevládne a proto bylo nekteré funkce treba vytvorit zcela od základu. Principy všech funkcí ovšem byly zachovány tak, aby odpovídaly standardu IEEE 754. Výsledná knihovna obsahuje funkce pro soucet, rozdíl, soucin, podíl a druhou odmocninu císla s plovoucí desetinnou cárkou. Také obsahuje funkce pro konverzi vstupních a výstupních hodnot a pár doplnujících funkcí pro presuny, které by mely usnadnit psaní programu využívajících tuto knihovnu.

Na CD priloženém k této práci naleznete mimo samotné matematické knihovny také aktuální verzi simulátoru WinIDE pro mikroprocesory Motorola M68HC08, od firmy P&E Microcomputer Systems.

SEZNAM POUŽITÉ LITERATURY

[1] Vána, Vladimír. Zacínáme pracovat s mikrokontroléry Motorola HC08 NITRON. 1. vyd. Praha: BEN, 2003. ISBN 80-7300-124-1.

[2] Rozehnal, Zdenek. Mikrokontroléry Motorola HC11. 1. vyd. Praha: BEN, 2001. ISBN 80-86056-77-5

[3] Starý, Jaroslav. Mikropocítac a jeho programování. 2. vyd. Praha: SNTL, 1987, Typové císlo L11-B3-III-41f/12032

[4] Rychlík, Jan. Programovací techniky. 1. vyd. KOOP: Ceské Budejovice, 1992. ISBN 80-901051-7-3

[5] Seriál Fixed point arithmetic. [online] Tišnovský, Pavel. [2006-6-9]. Dostupný z WWW: <http://www.root.cz/serialy/fixed-point-arithmetic/>

[6] Vašek.V., Vašek l. Programování mikropocítacu. FT VUT v Brne, Zlín 1994

[7] Dolinay, J. Programové moduly pro rídící systém s HC11, Bakalárská práce. FT Zlín, 1999

[8] Freescale semiconductor Inc. Referencní materiály k HC08, Firemní literatura 2002

SEZNAM POUŽITÝCH SYMBOLU A ZKRATEK

FP	Floating point – plovoucí desetinná cárka
FLTACCCx	Akumulátor pro císlo s plovoucí desetinnou cárkou (x = 1 nebo 2)
FPU	Floating Point Unit – aritmetická jednotka pro výpocty s plovoucí desetinnou cárkou
GPU	Graphic Processing Unit – procesor grafické karty
CPU	Central Processing Unit – aritmeticko logická jednotka (mikroprocesor)

SEZNAM OBRÁZKU

SEZNAM TABULEK

SEZNAM PRÍLOH

PRÍLOHA P I: PRÍKLADY PROGRAMU VYUŽÍVAJÍCÍCH MATEMATICKOU KNIHOVNU

Príklad 1: Výpocet obsahu kruhu o polomeru 5 m. (priklad1.asm)

```
RAMStart        EQU     $0080           ; definuje pocátek RAM pameti pro promenné
ROMStart        EQU     $EE00           ; definuje pocátek ROM pameti pro program
                org     RAMStart        ; Zacátek pameti RAM
vysledek        ds      14              ; 14-bytový buffer pro textový retezec
; Floating point akumulátor 1
FLTACC1EXP      FCB     $00             ; deklarace znaménkového bytu
FLTACC1MAN      FCB     $00,$00,$00     ; deklarace mantisy
FLTACC1MANZN            FCB             $00     ; deklarace znaménkového bytu
; Floating point akumulátor 2
FLTACC2EXP      FCB     $00             ; deklarace znaménkového bytu
FLTACC2MAN      FCB     $00,$00,$00     ; deklarace mantisy
FLTACC2MANZN            FCB             $00     ; deklarace znaménkového bytu
;- MAIN = hlavni program ----------------------------------------------------------------------
---
        org     RomStart        ; zacátek pameti ROM
PI      FCB     $040,$049,$0F,$0DB; 3,1415927

Main:
        rsp                     ; reset ukazatele na stack pointer
        clra                    ; inicializace akumulátoru A
        clrx                    ; inicializace registru X
        cli                     ; inicializace masky prerušení
start:  ldhx    #05             ; nacti do H:X hodnotu prumeru
        jsr     UINT2FLT        ; vlož tuto hodnotu do FPACC1
        jsr     FLTMOV1TO2      ; kopíruj prumer do FLTACC2
        jsr     FLTMUL          ; vypocítej druhou mocninu
        ldhx    #PI             ; nacti adresu císla PI
        jsr     GETFLTACC2      ; vlož hodnotu PI do akumulátoru FLTACC2
        jsr     FLTMUL          ; vynásob umocnený prumer s císlem PI
        ldhx    #vysledek       ; nacti adresu místa pro prevedení výsledku na text
        jsr     FLT2ASC         ; preved výsledek na textový retezec
        jmp     start           ; skok na zacátek programu

;- KONEC MAIN ----------------------------------------------------------------------------
;-- FUNKCE MATEMATICKÉ KNIHOVNY
                                        .
                                        .
                                        .
;-- KONEC FUNKCÍ MATEMATICKÉ KNIHOVNY
;- RESET VECTOR -------------------------------------------------------------------------
```

```
; místo pro vektor reset
        org     $FFFE
        dw      Main            ; FFFE - Reset Vector
;- KONEC RESET VECTORU ----------------------------------------------------------------------
-------
```

Príklad 2: A/D prevodník s rozsahem 0-10V a rozlišením 16-bitu. (priklad2.asm)

```
RAMStart        EQU     $0080           ; definuje pocátek RAM pameti pro promenné
ROMStart        EQU     $EE00           ; definuje pocátek ROM pameti pro program
                org     RAMStart        ; Zacátek pameti RAM
vysledek        ds      14              ; 14-bytový buffer pro textový retezec

; Floating point akumulátor 1
FLTACC1EXP      FCB     $00             ; deklarace znaménkového bytu
FLTACC1MAN      FCB     $00,$00,$00     ; deklarace mantisy
FLTACC1MANZN            FCB             $00     ; deklarace znaménkového bytu

; Floating point akumulátor 2
FLTACC2EXP      FCB     $00             ; deklarace znaménkového bytu
FLTACC2MAN      FCB     $00,$00,$00     ; deklarace mantisy
FLTACC2MANZN            FCB             $00     ; deklarace znaménkového bytu
;- MAIN = hlavni program ----------------------------------------------------------------------
---
                org     ROMStart        ; Zacátek pameti ROM
Main:
        rsp                             ; reset ukazatele na stack pointer
        clra                            ; inicializace akumulátoru A
        clrx                            ; inicializace registru X
        cli                             ; inicializace masky prerušení
start:  ldhx    #0A                     ; nacti maximální hodnotu napetí
        jsr     UINT2FLT                ; vlož ji do FLTACC1
        jsr     FLTMOV1TO2              ; uschovej tuto hodnotu do FLTACC2
        ldhx    #0FFFF                  ; nacti maximální velikost výstupu prevodníku (65535)
        jsr     UINT2FLT                ; vlož ji do FLTACC1
        jsr     CH1AND2                 ; prohod FLTACC1 a FLTACC2
                                        ; delenec = 10, delitel = 65535
        jsr     FLTDIV                  ; spocítej rozlišení A/D prevodníku
        jsr     FLTMOV1TO2              ; uschovej rozlišení do FLTACC2
        ldhx    #06462                  ; nacti údaj získaný prevodem (6462 h = 25698 d)
        jsr     UINT2FLT                ; vlož jej do FLTACC1
        jsr     FLTMUL                  ; vynásob výsledek prevodu s rozlišením prevodníku
        ldhx    #vysledek               ; nacti adresu prostoru pro textový retezec
        jsr     FLT2ASC                 ; vytvor textový retezec v pameti
        jmp     start
;- KONEC MAIN ----------------------------------------------------------------------------------
;-- FUNKCE MATEMATICKÉ KNIHOVNY
                                        .
                                        .
                                        .
;-- KONEC FUNKCÍ MATEMATICKÉ KNIHOVNY
;- RESET VECTOR --------------------------------------------------------------------------------
```

```
; místo pro vektor reset
      org    $FFFE
      dw     Main          ; FFFE - Reset Vector
;- KONEC RESET VECTORU ----------------------------------------------------------------
---------
```

Príklad 3: Výpocet korenu kvadratické rovnice vobecném tvaru $Ax^2 + Bx + C = 0$. A=3, B=5, C=1 (priklad3.asm)

```
RAMStart        EQU     $0080           ; definuje pocátek RAM pameti pro promenné
ROMStart        EQU     $EE00           ; definuje pocátek ROM pameti pro program

                org     RAMStart        ; Zacátek pameti RAM

X1              ds      14              ; 14-bytový buffer pro textový retezec pro X1
X2              ds      14              ; 14-bytový buffer pro textový retezec pro X2

; Floating point akumulátor 1

FLTACC1EXP      FCB     $00             ; deklarace znaménkového bytu
FLTACC1MAN      FCB     $00,$00,$00     ; deklarace mantisy
FLTACC1MANZN            FCB             $00     ; deklarace znaménkového bytu

; Floating point akumulátor 2

FLTACC2EXP      FCB     $00             ; deklarace znaménkového bytu
FLTACC2MAN      FCB     $00,$00,$00     ; deklarace mantisy
FLTACC2MANZN            FCB             $00     ; deklarace znaménkového bytu

                org     ROMStart        ; Zacátek pameti ROM
;- MAIN = hlavni program ---------------------------------------------------------------
---
Main:
        rsp                     ; reset ukazatele na stack pointer
        clra                    ; inicializace akumulátoru A
        clrx                    ; inicializace registru X
        cli                     ; inicializace masky prerušení
start:  ldhx    #3              ; nacti hodnotu A
        jsr     SINT2FLT        ; vlož ji do FLTACC1
        jsr     FLTMOV1TO2      ; kopíruj hodnotu A do FLTACC2
        ldhx    #1              ; nacti hodnotu C
        jsr     SINT2FLT        ; ; vlož ji do FLTACC1
        jsr     FLTMUL          ; Vynásob A a C
        jsr     FLTMOV1TO2      ; Výsledek kopíruj do FLTACC2
        ldhx    #4              ; Nacti 4
        jsr     SINT2FLT        ; a vlož do FLTACC1
```

```
            jsr     FLTMUL          ; vynásob
            jsr     PSHACC1         ; schovej výsledek na akumulátor (4 * A * C)
            ldhx    #5              ; nacti hodnotu B
            jsr     SINT2FLT        ; vlož ji do FLTACC1
            jsr     FLTMOV1TO2      ; kopíruj hodnotu B do FLTACC2
            jsr     FLTMUL          ; vynásob Akumulátory (B^2)
            jsr     PULACC2         ; nacti ze zálohy výsledek 4 * A * C
            jsr     FLTSUB          ; spocítej diskriminant D = B^2 - 4 * A * C
            jsr     FLTSQRT         ; vypocítej odmocninu z diskriminantu
            jsr     PSHACC1         ; uschovej odmocninu z diskriminantu na zásobník
            jsr     FLTMOV1TO2      ; vlož diskriminant do FLTACC2
            ldhx    #5              ; nacti B
            jsr     SINT2FLT        ; a ulož jej do FLTACC1
            jsr     FLTCHSIG1       ; zmen znaménko hodnoty B
            jsr     FLTSUB          ; odecti od B diskriminant
            jsr     FLTMOV1TO2      ; schovej výsledek do FLTACC2
            ldhx    #6              ; nacti hodnotu 6  (2*A = 2*3 = 6)
            jsr     SINT2FLT        ; vlož ji do FLTACC1
            jsr     CH1AND2         ; prohod akumulátory 1 a 2 kvuli delení
            jsr     FLTDIV          ; vydel hodnoty
            ldhx    #X1             ; nacti adresu textového bufferu pro koren X1
            jsr     FLT2ASC         ; preved výsledek na text
            jsr     PULACC2         ; nacti do FLTACC2 zálohu odmocniny diskriminantu
            ldhx    #5              ; nacti hodnotu B
            jsr     SINT2FLT        ; vlož hodnotu B do FLTACC1
            jsr     FLTCHSIG1       ; zmen znaménko hodnoty B
            jsr     FLTADD          ; secti FLTACC1 a FLTAC2
            jsr     FLTMOV1TO2      ; výsledek ulož do FLTACC2
            ldhx    #6              ; nacti hodnotu 6  (2*A = 2*3 = 6)
            jsr     SINT2FLT        ; vlož ji do FLTACC1
            jsr     CH1AND2         ; prohod akumulátory 1 a 2 kvuli delení
            jsr     FLTDIV          ; vydel hodnoty
            ldhx    #X2             ; nacti adresu textového bufferu pro koren X2
            jsr     FLT2ASC         ; preved výsledek na text
            jmp     start
;- KONEC MAIN ------------------------------------------------------------------------------

;-- FUNKCE MATEMATICKÉ KNIHOVNY
                                                .
                                                .
                                                .
;-- KONEC FUNKCÍ MATEMATICKÉ KNIHOVNY

;- RESET VECTOR ---------------------------------------------------------------------------
; místo pro vektor reset
      org   $FFFE
```

```
        dw      Main            ; FFFE - Reset Vector
;- KONEC RESET VECTORU ----------------------------------------------------------------------
---------
```

Príklad 4: Výpocet absolutní hodnoty komplexního císla . (priklad4.asm)

```
RAMStart        EQU     $0080           ; definuje pocátek RAM pameti pro promenné
ROMStart        EQU     $EE00           ; definuje pocátek ROM pameti pro program
                org     RAMStart        ; Zacátek pameti RAM
realna:         db      '10'
                fcb     $00
imaginarni:     db      '5'
                fcb     $00
vysledek        ds      4
; Floating point akumulátor 1
FLTACC1EXP              FCB             $00             ; deklarace znaménkového bytu
FLTACC1MAN              FCB             $00,$00,$00     ; deklarace mantisy
FLTACC1MANZN            FCB             $00             ; deklarace znaménkového bytu
; Floating point akumulátor 2
FLTACC2EXP              FCB             $00             ; deklarace znaménkového bytu
FLTACC2MAN              FCB             $00,$00,$00     ; deklarace mantisy
FLTACC2MANZN            FCB             $00             ; deklarace znaménkového bytu
                org     ROMstart        ; Zacátek pameti ROM
;- MAIN = hlavni program ------------------------------------------------------------------------
---
Main:
            rsp                         ; reset ukazatele na stack pointer
            clra                        ; inicializace akumulátoru A
            clrx                        ; inicializace registru X
            cli                         ; inicializace masky prerušení
start:      ldhx    #realna             ; nacti adresu reálné složky
            jsr     ASC2FLT             ; vlož její hodnotu do FLTACC1
            jsr     FLTMOV1TO2          ; kopíruj hodnotu RE do FLTACC2
            jsr     FLTMUL              ; RE^2
            jsr     PSHACC1             ; schovej výsledek na zásobník
            ldhx    #imaginarni         ; nacti adresu imaginární složky
            jsr     ASC2FLT             ; vlož její hodnotu do FLTACC1
            jsr     FLTMOV1TO2          ; kopíruj IM di FLTACC2
            jsr     FLTMUL              ; IM^2
            jsr     PULACC2             ; vlož zálohu ze zásobníku do FLTACC2
            jsr     FLTADD              ; secti obe složky
            jsr     FLTSQRT             ; Vypocítej odmocninu
            ldhx    #vysledek           ; nacti adresu pameti kam bude uložen výsledek
            jsr     PUTFLTACC1          ; ulož výsledek ve 4B formátu podle IEEE 754
;- KONEC MAIN -----------------------------------------------------------------------------------
;-- FUNKCE MATEMATICKÉ KNIHOVNY
                                        .
                                        .
;-- KONEC FUNKCÍ MATEMATICKÉ KNIHOVNY
;- RESET VECTOR ---------------------------------------------------------------------------------
```

```
; místo pro vektor reset
        org     $FFFE
        dw      Main            ; FFFE - Reset Vector
;- KONEC RESET VECTORU ----------------------------------------------------------------------
-------
```

PRÍLOHA P II: SHRNUTÍ FUNKCÍ MATEMATICKÉ KNIHOVNY

Funkce pro konverzi

ASC2FLT prevod ASCII retezce na reálné císlo
Vstup: H:X – ukazatel na retezec
Výstup: FLTACC1, A – chybový kód

FLT2ASC prevod reálného císla na ASCII retezec
Vstup: FLTACC1, H:X ukazatel na 14-bitový buffer
Výstup: H:X ukazatel na zacátek ascii retezce

UINT2FLT prevod celého císla bez znaménka na reálné císlo
Vstup: H:X – celé císlo bez znaménka
Výstup: FLTACC1

SINT2FLT prevod celého císla se znaménkem na reálné císlo
Vstup: H:X – celé císlo se znaménkem
Výstup: FLTACC1

FLT2INT prevod reálného císla na 16-bit celé císlo (bez zaokrouhlení)
Vstup: FLTACC1
Výstup: H:X – celé císlo, A kód chyby

FLT2INTROUND prevod reálného císla na 16-bit celé císlo (se zaokrouhlením)
Vstup: FLTACC1
Výstup: H:X – celé císlo, A kód chyby

GETACCx konverze císla ve formátu IEEE 754 na formát knihovny
x = 1 nebo 2
Vstup: H:X ukazatel pozici hodnoty uložené v pameti
Výstup: FLTACCx

PUTACCx konverze císla ve formátu knihovny na formát IEEE 754
x = 1 nebo 2
Vstup: H:X ukazatel na 4-bytovou oblast v pameti, FLTACCx
Výstup: H:X ukazatel na zacátek prevedené hodnoty

Funkce pro presuny

FLTMOV1TO2 kopírování FLTACC1 do FLTACC2
Vstup: FLTACC1
Výstup: FLTACC2

FLTMOV2TO1 kopírování FLTACC1 do FLTACC2
Vstup: FLTACC2

Výstup: FLTACC1

CH1AND2 prohození hodnot akumulátoru
Vstup: FLTACC1,FLTACC2
Výstup: FLTACC2, FLTACC1

Matematické funkce

FLTADD soucet dvou reálných císel
Vstup: FLTACC1 - scítanec1 , FLTACC2 - scítanec 2
Výstup: FLTACC1 - soucet, A – kód chyby

FLTSUB rozdíl dvou reálných císel
Vstup: FLTACC1 - menšenec, FLTACC2 - menšitel
Výstup: FLTACC1 - rozdíl, A kód chyby

FLTMUL soucin dvou reálných císel
Vstup: FLTACC1 – cinitel 1, FLTACC2 cinitel 2
Výstup: FLTACC1 - soucin, A kód chyby

FLTDIV podíl dvou reálných císel
Vstup: FLTACC1 – delenec, FLTACC2 – delitel
Výstup: FLTACC2 – podíl, A kód chyby

FLTSQRT druhá odmocnina reálného císla
Vstup: FLTACC1
Výstup: FLTACC1, A kód chyby

FLTROUND zaokrouhlení císla ve FLTACC1
Vstup: FLTACC1
Výstup: FLTACC1

Ostatní funkce

PSHACCx úschova akumulátoru na zásobník
x = 1 nebo 2
Vstup: žádný
Výstup: žádný

PULACCx obnovení akumulátoru ze zásobníku
x = 1 nebo 2
Vstup: žádný
Výstup: žádný

FLTCMP Porovnání dvou reálných císel
Vstup: FLTACC1, FLTACC2

Výstup: registr príznaku CCR

INTFRAC Rozdelení reálného císla na celou a desetinnou cást
Vstup: FLTACC1
Výstup: FLTACC1 – desetinná cást, FLTACC2 - celá cást

FLTABSx absolutní hodnota reálného císla
x =1 nebo 2
Vstup: FLTACCx
Výstup: FLTACCx

FLTSGNx vyhodnocení funkce SGN(X)
x = 1 nebo 2
Vstup: FLTACCx
Výstup: FLTACCx

FLTCHSIGx Zmena znaménka reálného císla
x = 1 nebo 2
Vstup: FLTACCx
Výstup: FLTACCx

Návratové kódy matematické knihovny

Pokud pri prubehu nekterých funkcí dojde kchybe, je nastaven príznak chyby (carry bit) a v akumulátoru A je vrácen kód této chyby.

A kumulátor A	**Popis chyby**
1	Chyba prevodu ASCII retezce na reálné císlo. ASCII retezec byl zadán nesprávne
2	Pretecení reálného císla pres maximální hodnotu (císlo > $3{,}4\times10^{38}$)
3	Podtecení reálného císla císlo je príliš malé (císlo < $1{,}17549\times10^{-38}$)
4	Delení nulou
5	Císlo je moc velké nebo malé pro prevod na celé císlo maximální rozsah je 65535 – (-32767)
6	Odmocnina ze záporného císla není definována

PRÍLOHA P III: INSTRUKCNÍ SADA PRO MIKROPOCÍTAC MOTOROLA M68HC08

Reference Guide

M68HC08RG/AD
Rev. 2, 3/2002

M68HC08
Family Reference Guide

MOTOROLA intelligence everywhere™ | digital dna™

Programming Model

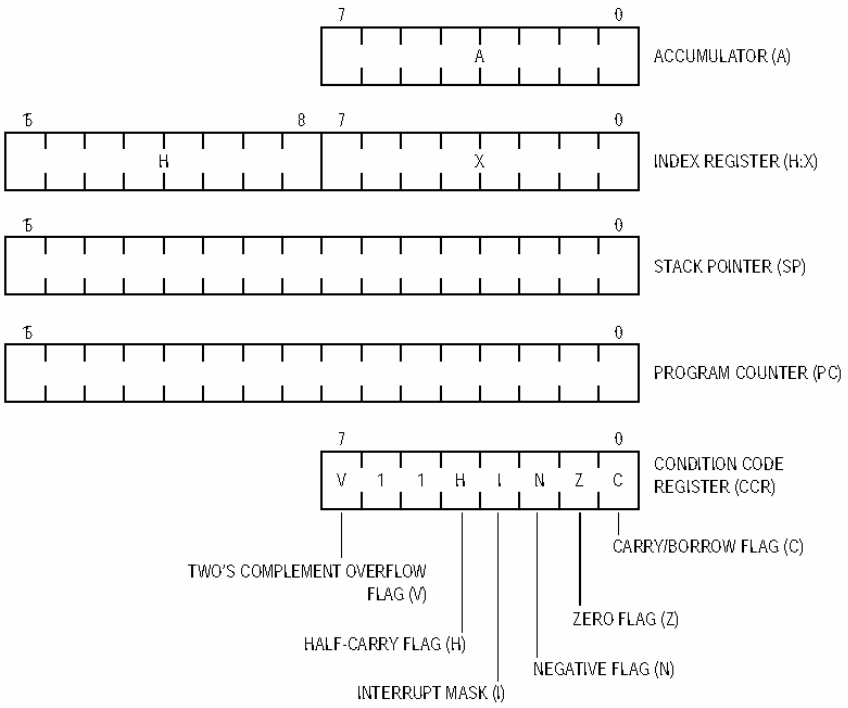
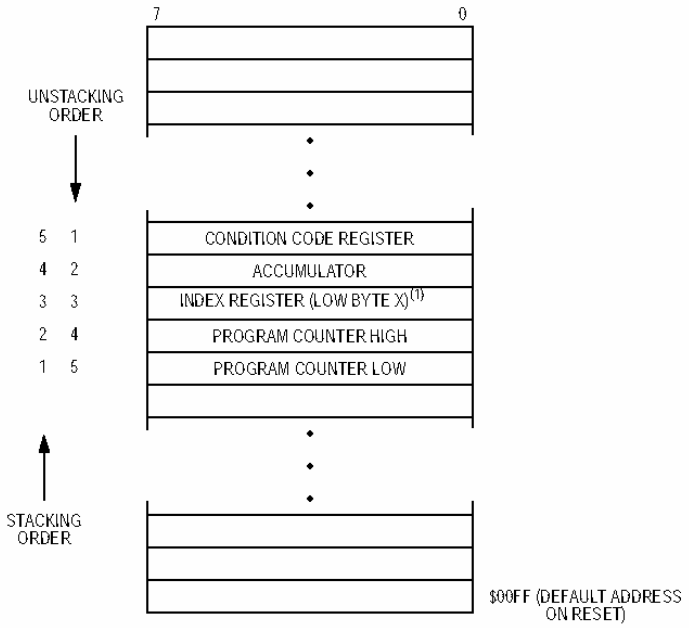


Figure 1. Programming Model

Stacking



1. High byte (H) of index register is not stacked.

Figure 2. Interrupt Stack Frame

NOTE: *To maintain compatibility with the M6805 Family, H (the high byte of the index register) is not stacked during interrupt processing. If the interrupt service routine modifies H or uses the indexed addressing mode, it is the user's responsibility to save and restore it prior to returning.*

```
IRQINT      PSHH
            |
            |Interrupt service routine
            |
            |
            PULH
            RTI
```

Figure 3. H Register Storage

Interrupt Vector Locations

Table 1. M68HC08 Vectors

Address	Reset	Priority
FFFE	Reset	1
FFFC	SWI	2
FFFA	IREQ[0]	3
:	:	:
FF02	IREQ[124]	127
FF00	IREQ[125]	128

Notation Used in Instruction Set Summary

See Table 2 for the instruction set summary.

Operators

( ) = Contents of register or memory location shown inside parentheses
← = Is loaded with (read: “gets”)
& = Boolean AND
| = Boolean OR
⊕ = Boolean exclusive OR
× = Multiply
÷ = Divide
: = Concatenate
+ = Add
– = Negate (two’s complement)
« = Sign extend

CPU Registers

A = Accumulator
CCR = Condition code register
H = Index register, higher order (most significant) eight bits
X = Index register, lower order (least significant) eight bits
PC = Program counter
PCH = Program counter, higher order (most significant) eight bits
PCL = Program counter, lower order (least significant) eight bits
SP = Stack pointer

Memory and Addressing

M	=	A memory location or absolute data, depending on addressing mode
M:M + $0001	=	A 16-bit value in two consecutive memory locations. The higher-order (most significant) eight bits are located at the address of M, and the lower-order (least significant) eight bits are located at the next higher sequential address.
rel	=	The relative offset, which is the two's complement number stored in the last byte of machine code corresponding to a branch instruction

Condition Code Register (CCR) Bits

V	=	Two's complement overflow indicator, bit 7
H	=	Half carry, bit 4
I	=	Interrupt mask, bit 3
N	=	Negative indicator, bit 2
Z	=	Zero indicator, bit 1
C	=	Carry/borrow, bit 0 (carry out of bit 7)

Bit Status BEFORE Execution of an Instruction (*n* = 7, 6, 5, ... 0)(1)

M*n*	=	Bit *n* of memory location used in operation
A*n*	=	Bit *n* of accumulator
H*n*	=	Bit *n* of index register H
X*n*	=	Bit *n* of index register X
b*n*	=	Bit *n* of the source operand (M, A, or X)

1. For 2-byte operations such as LDHX, STHX, and CPHX, *n* = 15 refers to bit 15 of the 2-byte word or bit 7 of the most significant (first) byte.

Bit Status AFTER Execution of an Instruction(1)

R*n*	=	Bit *n* of the result of an operation (*n* = 7, 6, 5, ... 0)

1. For 2-byte operations such as LDHX, STHX, and CPHX, *n* = 15 refers to bit 15 of the 2-byte word or bit 7 of the most significant (first) byte.

CCR Activity Figure Notation

–	=	Bit not affected
0	=	Bit forced to 0
1	=	Bit forced to 1
↕	=	Bit set or cleared according to results of operation
U	=	Undefined after the operation

Machine Coding Notation

`dd`	=	Low-order eight bits of a direct address $0000–$00FF (high byte assumed to be $00)
`ee`	=	Upper eight bits of 16-bit offset
`ff`	=	Lower eight bits of 16-bit offset or 8-bit offset
`ii`	=	One byte of immediate data
`jj`	=	High-order byte of a 16-bit immediate data value
`kk`	=	Low-order byte of a 16-bit immediate data value
`hh`	=	High-order byte of 16-bit extended address
`ll`	=	Low-order byte of 16-bit extended address
`rr`	=	Relative offset

Explanation of Italic Expressions in Source Form Column

n	=	Any label or expression that evaluates to a single integer in the range 0–7
opr8i	=	Any label or expression that evaluates to an 8-bit immediate value
opr16i	=	Any label or expression that evaluates to a 16-bit immediate value
opr8a	=	Any label or expression that evaluates to an 8-bit value. The instruction treats this 8-bit value as the low order eight bits of an address in the direct page of the 64-Kbyte address space ($00xx).
opr16a	=	Any label or expression that evaluates to a 16-bit value. The instruction treats this value as an address in the 64-Kbyte address space.
oprx8	=	Any label or expression that evaluates to an unsigned 8-bit value; used for indexed addressing
oprx16	=	Any label or expression that evaluates to a 16-bit value. Since the MC68HC08S has a 16-bit address bus, this can be either a signed or an unsigned value.
rel	=	Any label or expression that refers to an address that is within –128 to +127 locations from the next address after the last byte of object code for the current instruction. The assembler will calculate the 8-bit signed offset and include it in the object code for this instruction.

Address Modes

INH	=	Inherent (no operands)
IMM	=	8-bit or 16-bit immediate
DIR	=	8-bit direct
EXT	=	16-bit extended
IX	=	16-bit indexed no offset
IX+	=	16-bit indexed no offset, post increment (CBEQ and MOV only)
IX1	=	16-bit indexed with 8-bit offset from H:X
IX1+	=	16-bit indexed with 8-bit offset, post increment (CBEQ only)
IX2	=	16-bit indexed with 16-bit offset from H:X
REL	=	8-bit relative offset
SP1	=	Stack pointer relative with 8-bit offset
SP2	=	Stack pointer relative with 16-bit offset

Table 2. Instruction Set Summary (Sheet 1 of 8)

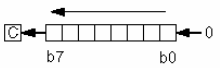
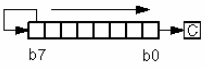
Source Form	Operation	Description	Effect on CCR						Address Mode	Opcode	Operand	Cycles
			V	H	I	N	Z	C				
ADC #*opr8i* ADC *opr8a* ADC *opr16a* ADC *oprx16*,X ADC *oprx8*,X ADC ,X ADC *oprx16*,SP ADC *oprx8*,SP	Add with Carry	A ← (A) + (M) + (C)	↕	↕	–	↕	↕	↕	IMM DIR EXT IX2 IX1 IX SP2 SP1	A9 B9 C9 D9 E9 F9 9ED9 9EE9	ii dd hh ll ee ff ff	2 3 4 4 3 2 5 4
ADD #*opr8i* ADD *opr8a* ADD *opr16a* ADD *oprx16*,X ADD *oprx8*,X ADD ,X ADD *oprx16*,SP ADD *oprx8*,SP	Add without Carry	A ← (A) + (M)	↕	↕	–	↕	↕	↕	IMM DIR EXT IX2 IX1 IX SP2 SP1	AB BB CB DB EB FB 9EDB 9EEB	ii dd hh ll ee ff ff	2 3 4 4 3 2 5 4
AIS #*opr8i*	Add Immediate Value (Signed) to Stack Pointer	SP ← (SP) + (M) M is sign extended to a 16-bit value	–	–	–	–	–	–	IMM	A7	ii	2
AIX #*opr8i*	Add Immediate Value (Signed) to Index Register (H:X)	H:X ← (H:X) + (M) M is sign extended to a 16-bit value	–	–	–	–	–	–	IMM	AF	ii	2
AND #*opr8i* AND *opr8a* AND *opr16a* AND *oprx16*,X AND *oprx8*,X AND ,X AND *oprx16*,SP AND *oprx8*,SP	Logical AND	A ← (A) & (M)	0	–	–	↕	↕	–	IMM DIR EXT IX2 IX1 IX SP2 SP1	A4 B4 C4 D4 E4 F4 9ED4 9EE4	ii dd hh ll ee ff ff	2 3 4 4 3 2 5 4
ASL *opr8a* ASLA ASLX ASL *oprx8*,X ASL ,X ASL *oprx8*,SP	Arithmetic Shift Left (Same as LSL)		↕	–	–	↕	↕	↕	DIR INH INH IX1 IX SP1	38 48 58 68 78 9E68	dd ff ff	4 1 1 4 3 5
ASR *opr8a* ASRA ASRX ASR *oprx8*,X ASR ,X ASR *oprx8*,SP	Arithmetic Shift Right		↕	–	–	↕	↕	↕	DIR INH INH IX1 IX SP1	37 47 57 67 77 9E67	dd ff ff	4 1 1 4 3 5
BCC *rel*	Branch if Carry Bit Clear	Branch if (C) = 0	–	–	–	–	–	–	REL	24	rr	3

Table 2. Instruction Set Summary (Sheet 2 of 8)

Source Form	Operation	Description	Effect on CCR						Address Mode	Opcode	Operand	Cycles
			V	H	I	N	Z	C				
BCLR *n*,*opr8*a	Clear Bit n in Memory	Mn ← 0	–	–	–	–	–	–	DIR (b0) DIR (b1) DIR (b2) DIR (b3) DIR (b4) DIR (b5) DIR (b6) DIR (b7)	11 13 15 17 19 1B 1D 1F	dd dd dd dd dd dd dd dd	4 4 4 4 4 4 4 4
BCS *rel*	Branch if Carry Bit Set (Same as BLO)	Branch if (C) = 1	–	–	–	–	–	–	REL	25	rr	3
BEQ *rel*	Branch if Equal	Branch if (Z) = 1	–	–	–	–	–	–	REL	27	rr	3
BGE *rel*	Branch if Greater Than or Equal To (Signed Operands)	Branch if (N ⊕ V) = 0	–	–	–	–	–	–	REL	90	rr	3
BGT *rel*	Branch if Greater Than (Signed Operands)	Branch if (Z) | (N ⊕ V) = 0	–	–	–	–	–	–	REL	92	rr	3
BHCC *rel*	Branch if Half Carry Bit Clear	Branch if (H) = 0	–	–	–	–	–	–	REL	28	rr	3
BHCS *rel*	Branch if Half Carry Bit Set	Branch if (H) = 1	–	–	–	–	–	–	REL	29	rr	3
BHI *rel*	Branch if Higher	Branch if (C) | (Z) = 0	–	–	–	–	–	–	REL	22	rr	3
BHS *rel*	Branch if Higher or Same (Same as BCC)	Branch if (C) = 0	–	–	–	–	–	–	REL	24	rr	3
BIH *rel*	Branch if IRQ Pin High	Branch if IRQ pin = 1	–	–	–	–	–	–	REL	2F	rr	3
BIL *rel*	Branch if IRQ Pin Low	Branch if IRQ pin = 0	–	–	–	–	–	–	REL	2E	rr	3
BIT #*opr8i* BIT *opr8*a BIT *opr16*a BIT *oprx16*,X BIT *oprx8*,X BIT ,X BIT *oprx16*,SP BIT *oprx8*,SP	Bit Test	(A) & (M) (CCR Updated but Operands Not Changed)	0	–	–	↕	↕	–	IMM DIR EXT IX2 IX1 IX SP2 SP1	A5 B5 C5 D5 E5 F5 9ED5 9EE5	ii dd hh ll ee ff ff ee ff ff	2 3 4 4 3 2 5 4
BLE *rel*	Branch if Less Than or Equal To (Signed Operands)	Branch if (Z) | (N ⊕ V) = 1	–	–	–	–	–	–	REL	93	rr	3
BLO *rel*	Branch if Lower (Same as BCS)	Branch if (C) = 1	–	–	–	–	–	–	REL	25	rr	3
BLS *rel*	Branch if Lower or Same	Branch if (C) | (Z) = 1	–	–	–	–	–	–	REL	23	rr	3
BLT *rel*	Branch if Less Than (Signed Operands)	Branch if (N ⊕ V) = 1	–	–	–	–	–	–	REL	91	rr	3
BMC *rel*	Branch if Interrupt Mask Clear	Branch if (I) = 0	–	–	–	–	–	–	REL	2C	rr	3
BMI *rel*	Branch if Minus	Branch if (N) = 1	–	–	–	–	–	–	REL	2B	rr	3

Table 2. Instruction Set Summary (Sheet 3 of 8)

Source Form	Operation	Description	Effect on CCR						Address Mode	Opcode	Operand	Cycles
			V	H	I	N	Z	C				
BMS *rel*	Branch if Interrupt Mask Set	Branch if (I) = 1	–	–	–	–	–	–	REL	2D	rr	3
BNE *rel*	Branch if Not Equal	Branch if (Z) = 0	–	–	–	–	–	–	REL	26	rr	3
BPL *rel*	Branch if Plus	Branch if (N) = 0	–	–	–	–	–	–	REL	2A	rr	3
BRA *rel*	Branch Always	No Test	–	–	–	–	–	–	REL	20	rr	3
BRCLR *n*,*opr8a*,*rel*	Branch if Bit *n* in Memory Clear	Branch if (Mn) = 0	–	–	–	–	–	↕	DIR (b0) DIR (b1) DIR (b2) DIR (b3) DIR (b4) DIR (b5) DIR (b6) DIR (b7)	01 03 05 07 09 0B 0D 0F	dd rr dd rr dd rr dd rr dd rr dd rr dd rr dd rr	5 5 5 5 5 5 5 5
BRN *rel*	Branch Never	Uses 3 Bus Cycles	–	–	–	–	–	–	REL	21	rr	3
BRSET *n*,*opr8a*,*rel*	Branch if Bit *n* in Memory Set	Branch if (Mn) = 1	–	–	–	–	–	↕	DIR (b0) DIR (b1) DIR (b2) DIR (b3) DIR (b4) DIR (b5) DIR (b6) DIR (b7)	00 02 04 06 08 0A 0C 0E	dd rr dd rr dd rr dd rr dd rr dd rr dd rr dd rr	5 5 5 5 5 5 5 5
BSET *n*,*opr8a*	Set Bit *n* in Memory	Mn ← 1	–	–	–	–	–	–	DIR (b0) DIR (b1) DIR (b2) DIR (b3) DIR (b4) DIR (b5) DIR (b6) DIR (b7)	10 12 14 16 18 1A 1C 1E	dd dd dd dd dd dd dd dd	4 4 4 4 4 4 4 4
BSR *rel*	Branch to Subroutine	PC ← (PC) + $0002 push (PCL); SP ← (SP) – $0001 push (PCH); SP ← (SP) – $0001 PC ← (PC) + *rel*	–	–	–	–	–	–	REL	AD	rr	4
CBEQ *opr8a*,*rel* CBEQA #*opr8i*,*rel* CBEQX #*opr8i*,*rel* CBEQ *oprx8*,X+,*rel* CBEQ ,X+,*rel* CBEQ *oprx8*,SP,*rel*	Compare and Branch if Equal	Branch if (A) = (M) Branch if (A) = (M) Branch if (X) = (M) Branch if (A) = (M) Branch if (A) = (M) Branch if (A) = (M)	–	–	–	–	–	–	DIR IMM IMM IX1+ IX+ SP1	31 41 51 61 71 9E61	dd rr ii rr ii rr ff rr rr ff rr	5 4 4 5 4 6
CLC	Clear Carry Bit	C ← 0	–	–	–	–	–	0	INH	98		1
CLI	Clear Interrupt Mask Bit	I ← 0	–	–	0	–	–	–	INH	9A		2

Table 2. Instruction Set Summary (Sheet 4 of 8)

Source Form	Operation	Description	Effect on CCR						Address Mode	Opcode	Operand	Cycles
			V	H	I	N	Z	C				
CLR *oprx8*,X CLR ,X CLR *oprx8*,SP	Clear	M ← $00 A ← $00 X ← $00 H ← $00 M ← $00 M ← $00 M ← $00	0	–	–	0	1	–	DIR INH INH INH IX1 IX SP1	3F 4F 5F 8C 6F 7F 9E6F	dd ff ff	3 1 1 1 3 2 4
CMP #*oprx8*,X CMP ,X CMP *oprx8*,SP	Compare Accumulator with Memory	(A) – (M) (CCR Updated But Operands Not Changed)	↕	–	–	↕	↕	↕	IMM DIR EXT IX2 IX1 IX SP2 SP1	A1 B1 C1 D1 E1 F1 9ED1 9EE1	ii dd hh ll ee ff ff ee ff ff	2 3 4 4 3 2 5 4
COM *oprx8*,X COM ,X COM *oprx8*,SP	Complement (One's Complement)	M ← ($\overline{M}$)= $FF – (M) A ← ($\overline{A}$) = $FF – (A) X ← ($\overline{X}$) = $FF – (X) M ← ($\overline{M}$) = $FF – (M) M ← ($\overline{M}$) = $FF – (M) M ← ($\overline{M}$) = $FF – (M)	0	–	–	↕	↕	1	DIR INH INH IX1 IX SP1	33 43 53 63 73 9E63	dd ff ff	4 1 1 4 3 5
CPHX #*opr* CPHX *opr*	Compare Index Register (H:X) with Memory	(H:X) – (M:M + $0001) (CCR Updated But Operands Not Changed)	↕	–	–	↕	↕	↕	IMM DIR	65 75	jj ii+1 dd	3 4
CPX #*oprx8*,X CPX ,X CPX *oprx8*,SP	Compare X (Index Register Low) with Memory	(X) – (M) (CCR Updated But Operands Not Changed)	↕	–	–	↕	↕	↕	IMM DIR EXT IX2 IX1 IX SP2 SP1	A3 B3 C3 D3 E3 F3 9ED3 9EE3	ii dd hh ll ee ff ff ee ff ff	2 3 4 4 3 2 5 4
DAA	Decimal Adjust Accumulator After ADD or ADC of BCD Values	$(A)_{10}$	U	–	–	↕	↕	↕	INH	72		2
DBNZ *oprx8*,X,*rel* DBNZ ,X,*rel* DBNZ *oprx8*,SP,*rel*	Decrement and Branch if Not Zero	Decrement A, X, or M Branch if (result) ≠ 0 DBNZX Affects X Not H	–	–	–	–	–	–	DIR INH INH IX1 IX SP1	3B 4B 5B 6B 7B 9E6B	dd rr rr rr ff rr rr ff rr	5 3 3 5 4 6
DEC *oprx8*,X DEC ,X DEC *oprx8*,SP	Decrement	M ← (M) – $01 A ← (A) – $01 X ← (X) – $01 M ← (M) – $01 M ← (M) – $01 M ← (M) – $01	↕	–	–	↕	↕	–	DIR INH INH IX1 IX SP1	3A 4A 5A 6A 7A 9E6A	dd ff ff	4 1 1 4 3 5
DIV	Divide	A ← (H:A)÷(X) H ← Remainder	–	–	–	–	↕	↕	INH	52		7

Table 2. Instruction Set Summary (Sheet 5 of 8)

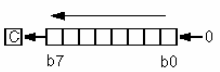
Source Form	Operation	Description	Effect on CCR V	H	I	N	Z	C	Address Mode	Opcode	Operand	Cycles
EOR #*opr8i* EOR *opr8a* EOR *opr16a* EOR *oprx16*,X EOR *oprx8*,X EOR ,X EOR *oprx16*,SP EOR *oprx8*,SP	Exclusive OR Memory with Accumulator	A ← (A ⊕ M)	0	–	–	↕	↕	–	IMM DIR EXT IX2 IX1 IX SP2 SP1	A8 B8 C8 D8 E8 F8 9ED8 9EE8	ii dd hh ll ee ff ff	2 3 4 4 3 2 5 4
INC *opr8a* INCA INCX INC *oprx8*,X INC ,X INC *oprx8*,SP	Increment	M ← (M) + $01 A ← (A) + $01 X ← (X) + $01 M ← (M) + $01 M ← (M) + $01 M ← (M) + $01	↕	–	–	↕	↕	–	DIR INH INH IX1 IX SP1	3C 4C 5C 6C 7C 9E6C	dd ff ff	4 1 1 4 3 5
JMP *opr8a* JMP *opr16a* JMP *oprx16*,X JMP *oprx8*,X JMP ,X	Jump	PC ← Jump Address	–	–	–	–	–	–	DIR EXT IX2 IX1 IX	BC CC DC EC FC	dd hh ll ee ff ff	2 3 4 3 3
JSR *opr8a* JSR *opr16a* JSR *oprx16*,X JSR *oprx8*,X JSR ,X	Jump to Subroutine	PC ← (PC) + *n* (*n* = 1, 2, or 3) Push (PCL); SP ← (SP) – $0001 Push (PCH); SP ← (SP) – $0001 PC ← Unconditional Address	–	–	–	–	–	–	DIR EXT IX2 IX1 IX	BD CD DD ED FD	dd hh ll ee ff ff	4 5 6 5 4
LDA #*opr8i* LDA *opr8a* LDA *opr16a* LDA *oprx16*,X LDA *oprx8*,X LDA ,X LDA *oprx16*,SP LDA *oprx8*,SP	Load Accumulator from Memory	A ← (M)	0	–	–	↕	↕	–	IMM DIR EXT IX2 IX1 IX SP2 SP1	A6 B6 C6 D6 E6 F6 9ED6 9EE6	ii dd hh ll ee ff ff	2 3 4 4 3 2 5 4
LDHX #*opr* LDHX *opr*	Load Index Register (H:X) from Memory	H:X ← (M:M + $0001)	0	–	–	↕	↕	–	IMM DIR	45 55	ii jj dd	3 4
LDX #*opr8i* LDX *opr8a* LDX *opr16a* LDX *oprx16*,X LDX *oprx8*,X LDX ,X LDX *oprx16*,SP LDX *oprx8*,SP	Load X (Index Register Low) from Memory	X ← (M)	0	–	–	↕	↕	–	IMM DIR EXT IX2 IX1 IX SP2 SP1	AE BE CE DE EE FE 9EDE 9EEE	ii dd hh ll ee ff ff	2 3 4 4 3 2 5 4
LSL *opr8a* LSLA LSLX LSL *oprx8*,X LSL ,X LSL *oprx8*,SP	Logical Shift Left (Same as ASL)		↕	–	–	↕	↕	↕	DIR INH INH IX1 IX SP1	38 48 58 68 78 9E68	dd ff ff	4 1 1 4 3 5

Table 2. Instruction Set Summary (Sheet 6 of 8)

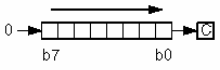
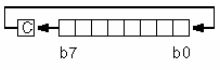
Source Form	Operation	Description	Effect on CCR						Address Mode	Opcode	Operand	Cycles
			V	H	I	N	Z	C				
LSR *oprx8*,X LSR ,X LSR *oprx8*,SP	Logical Shift Right		↕	–	–	0	↕	↕	DIR INH INH IX1 IX SP1	34 44 54 64 74 9E64	dd ff ff	4 1 1 4 3 5
MOV *oprx8*,*oprx8* MOV *oprx8*,X+ MOV #*oprx8*,*oprx8* MOV ,X+,*oprx8*	Move	$(M)_{destination} \leftarrow (M)_{source}$ H:X ← (H:X) + $0001 in IX+/DIR and DIR/IX+ Modes	0	–	–	↕	↕	–	DIR/DIR DIR/IX+ IMM/DIR IX+/DIR	4E 5E 6E 7E	dd dd dd ii dd dd	5 4 4 4
MUL	Unsigned multiply	X:A ← (X) × (A)	–	0	–	–	–	0	INH	42		5
NEG *oprx8*,X NEG ,X NEG *oprx8*,SP	Negate (Two's Complement)	M ← – (M) = $00 – (M) A ← – (A) = $00 – (A) X ← – (X) = $00 – (X) M ← – (M) = $00 – (M) M ← – (M) = $00 – (M) M ← – (M) = $00 – (M)	↕	–	–	↕	↕	↕	DIR INH INH IX1 IX SP1	30 40 50 60 70 9E60	dd ff ff	4 1 1 4 3 5
NOP	No Operation	Uses 1 Bus Cycle	–	–	–	–	–	–	INH	9D		1
NSA	Nibble Swap Accumulator	A ← (A[3:0]:A[7:4])	–	–	–	–	–	–	INH	62		3
ORA #*oprx8* ORA *oprx8* ORA *oprx16* ORA *oprx16*,X ORA *oprx8*,X ORA ,X ORA *oprx16*,SP ORA *oprx8*,SP	Inclusive OR Accumulator and Memory	A ← (A) | (M)	0	–	–	↕	↕	–	IMM DIR EXT IX2 IX1 IX SP2 SP1	AA BA CA DA EA FA 9EDA 9EEA	ii dd hh ll ee ff ff ee ff ff	2 3 4 4 3 2 5 4
PSHA	Push Accumulator onto Stack	Push (A); SP ← (SP) – $0001	–	–	–	–	–	–	INH	87		2
PSHH	Push H (Index Register High) onto Stack	Push (H); SP ← (SP) – $0001	–	–	–	–	–	–	INH	8B		2
PSHX	Push X (Index Register Low) onto Stack	Push (X); SP ← (SP) – $0001	–	–	–	–	–	–	INH	89		2
PULA	Pull Accumulator from Stack	SP ← (SP + $0001); Pull (A)	–	–	–	–	–	–	INH	86		2
PULH	Pull H (Index Register High) from Stack	SP ← (SP + $0001); Pull (H)	–	–	–	–	–	–	INH	8A		2
PULX	Pull X (Index Register Low) from Stack	SP ← (SP + $0001); Pull (X)	–	–	–	–	–	–	INH	88		2
ROL *oprx8*,X ROL ,X ROL *oprx8*,SP	Rotate Left through Carry		↕	–	–	↕	↕	↕	DIR INH INH IX1 IX SP1	39 49 59 69 79 9E69	dd ff ff	4 1 1 4 3 5

Table 2. Instruction Set Summary (Sheet 7 of 8)

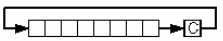
Source Form	Operation	Description	Effect on CCR						Address Mode	Opcode	Operand	Cycles
			V	H	I	N	Z	C				
ROR *opr8a* RORA RORX ROR *oprx8*,X ROR ,X ROR *oprx8*,SP	Rotate Right through Carry		↕	–	–	↕	↕	↕	DIR INH INH IX1 IX SP1	36 46 56 66 76 9E66	dd ff ff	4 1 1 4 3 5
RSP	Reset Stack Pointer	SP ← $FF (High Byte Not Affected)	–	–	–	–	–	–	INH	9C		1
RTI	Return from Interrupt	SP ← (SP) + $0001; Pull (CCR) SP ← (SP) + $0001; Pull (A) SP ← (SP) + $0001; Pull (X) SP ← (SP) + $0001; Pull (PCH) SP ← (SP) + $0001; Pull (PCL)	↕	↕	↕	↕	↕	↕	INH	80		7
RTS	Return from Subroutine	SP ← SP + $0001; Pull (PCH) SP ← SP + $0001; Pull (PCL)	–	–	–	–	–	–	INH	81		4
SBC #*opr8i* SBC *opr8a* SBC *opr16a* SBC *oprx16*,X SBC *oprx8*,X SBC ,X SBC *oprx16*,SP SBC *oprx8*,SP	Subtract with Carry	A ← (A) – (M) – (C)	↕	–	–	↕	↕	↕	IMM DIR EXT IX2 IX1 IX SP2 SP1	A2 B2 C2 D2 E2 F2 9ED2 9EE2	ii dd hh ll ee ff ff ee ff ff	2 3 4 4 3 2 5 4
SEC	Set Carry Bit	C ← 1	–	–	–	–	–	1	INH	99		1
SEI	Set Interrupt Mask Bit	I ← 1	–	–	1	–	–	–	INH	9B		2
STA *opr8a* STA *opr16a* STA *oprx16*,X STA *oprx8*,X STA ,X STA *oprx16*,SP STA *oprx8*,SP	Store Accumulator in Memory	M ← (A)	0	–	–	↕	↕	–	DIR EXT IX2 IX1 IX SP2 SP1	B7 C7 D7 E7 F7 9ED7 9EE7	dd hh ll ee ff ff ee ff ff	3 4 4 3 2 5 4
STHX *opr*	Store H:X (Index Reg.)	(M:M + $0001) ← (H:X)	0	–	–	↕	↕	–	DIR	35	dd	4
STOP	Enable Interrupts: Stop Processing Refer to MCU Documentation	I bit ← 0; Stop Processing	–	–	0	–	–	–	INH	8E		1
STX *opr8a* STX *opr16a* STX *oprx16*,X STX *oprx8*,X STX ,X STX *oprx16*,SP STX *oprx8*,SP	Store X (Low 8 Bits of Index Register) in Memory	M ← (X)	0	–	–	↕	↕	–	DIR EXT IX2 IX1 IX SP2 SP1	BF CF DF EF FF 9EDF 9EEF	dd hh ll ee ff ff ee ff ff	3 4 4 3 2 5 4

Table 2. Instruction Set Summary (Sheet 8 of 8)

Source Form	Operation	Description	Effect on CCR						Address Mode	Opcode	Operand	Cycles
			V	H	I	N	Z	C				
SUB *#opr8i* SUB *opr8a* SUB *opr16a* SUB *oprx16*,X SUB *oprx8*,X SUB ,X SUB *oprx16*,SP SUB *oprx8*,SP	Subtract	A ← (A) – (M)	↕	–	–	↕	↕	↕	IMM DIR EXT IX2 IX1 IX SP2 SP1	A0 B0 C0 D0 E0 F0 9ED0 9EE0	ii dd hh ll ee ff ff ee ff ff	2 3 4 4 3 2 5 4
SWI	Software Interrupt	PC ← (PC) + $0001 Push (PCL); SP ← (SP) – $0001 Push (PCH); SP ← (SP) – $0001 Push (X); SP ← (SP) – $0001 Push (A); SP ← (SP) – $0001 Push (CCR); SP ← (SP) – $0001 I ← 1; PCH ← Interrupt Vector High Byte PCL ← Interrupt Vector Low Byte	–	–	1	–	–	–	INH	83		9
TAP	Transfer Accumulator to CCR	CCR ← (A)	↕	↕	↕	↕	↕	↕	INH	84		2
TAX	Transfer Accumulator to X (Index Register Low)	X ← (A)	–	–	–	–	–	–	INH	97		1
TPA	Transfer CCR to Accumulator	A ← (CCR)	–	–	–	–	–	–	INH	85		1
TST *opr8a* TSTA TSTX TST *oprx8*,X TST ,X TST *oprx8*,SP	Test for Negative or Zero	(M) – $00 (A) – $00 (X) – $00 (M) – $00 (M) – $00 (M) – $00	0	–	–	↕	↕	–	DIR INH INH IX1 IX SP1	3D 4D 5D 6D 7D 9E6D	dd ff ff	3 1 1 3 2 4
TSX	Transfer SP to Index Reg.	H:X ← (SP) + $0001	–	–	–	–	–	–	INH	95		2
TXA	Transfer X (Index Reg. Low) to Accumulator	A ← (X)	–	–	–	–	–	–	INH	9F		1
TXS	Transfer Index Reg. to SP	SP ← (H:X) – $0001	–	–	–	–	–	–	INH	94		2
WAIT	Enable Interrupts; Wait for Interrupt	I bit ← 0; Halt CPU	–	–	0	–	–	–	INH	8F		1

Addressing Modes

Inherent (INH)

The inherent addressing mode has no operand because the opcode contains all information necessary to carry out the instruction. Most inherent instructions are one byte long.

Immediate (IMM)

The operand in immediate mode instructions is contained in the byte(s) immediately following the opcode. The immediate value is one or two bytes, depending on the size of the register involved in the instruction.

Direct (DIR)

Most direct mode instructions can access any of the first 256 memory addresses with two bytes. The first byte is the opcode, and the second byte is the low byte of the operand address. The high byte of the address is assumed to be $00.

Extended (EXT)

Extended mode instructions are three bytes in length and can access any address in a 64-Kbyte memory map. The first byte is the opcode. The following two bytes are the operand addresses.

Indexed (IX, IX1, and IX2)

Indexed mode instructions access data with variable addresses. The effective address (EA) of the operand is determined by the contents of the register (H:X) added to a zero, 8-bit, or 16-bit offset. For one-byte, zero-offset mode instructions (IX), X (index register low) contains the low byte of the EA of the operand. The value of H (index register high) is $00 if none of the HC08 instructions that modify H are used, assuring source code compatibility with HC05 Family instructions. The sum of H:X is the EA of the operand. For two-byte, 8-bit offset mode instructions (IX1) the unsigned bytes in H:X added to the unsigned byte following the opcode constitutes the EA of the operand. For three byte, 16-bit offset mode instructions (IX2), the unsigned bytes in H:X added to the 16-bit unsigned word following the opcode constitute the EA of the operand.

Stack Pointer (SP1 and SP2)

Stack pointer (SP) mode instructions operate like indexed instructions, except that the offset is added to the 16-bit SP. Stack pointer, 8-bit offset instructions (SP1) are three-byte instructions. The EA of the operand is formed by adding the unsigned byte in the SP register to the unsigned byte following the opcode. Stack pointer, 16-bit offset instructions (SP2) are four-byte instructions. The EA of the operand is formed by adding the unsigned bytes in the 16-bit SP register to the 16-bit unsigned word following the opcode.

Relative (REL)

Conditional branch instructions use the relative addressing mode. The EA of the operand depends on whether or not the branch is taken. If a branch is taken, the EA of the operand is formed by adding the signed byte following the opcode to the value of the PC, and the PC is loaded with the EA. If no branch is taken, the EA is the contents of the PC.

Memory to Memory (IMD, DD, IX+D, and DIX+)

Memory to memory immediate to direct (IMD) is a three-byte addressing mode. The operand in the byte immediately following the opcode is stored in the direct page location addressed by the second byte following the opcode.

Memory to memory direct to direct (DD) is a three-byte addressing mode. The operand in the byte immediately following the opcode is stored in the direct page location addressed by the second byte following the opcode.

Memory to memory indexed to direct with post increment of H:X (IX+D) is a two-byte addressing mode. The operand addressed by H:X is stored in the direct page location addressed by the byte following the opcode.

Memory to memory direct to indexed with post increment of H:X (DIX+) is a two-byte addressing mode. The operand in the direct page location addressed by the byte immediately following the opcode is stored in the location addressed by H:X.

Indexed and Indexed 8-Bit Offset with Post Increment (IX+ and IX1+)

Indexed, no offset with post increment mode instructions (IX+) are two-byte instructions that address operands, then increment H:X. The EA of the operand is derived by adding X (low byte) to H (high byte). Indexed, 8-bit offset with post increment mode instructions (IX1+) are three-byte instructions that address operands with variable addresses, then increment H:X. The EA of the operand is derived by adding X (low byte) with H (high byte).

Opcode Map

See **Table 3**.

Hexadecimal to ASCII Conversion

See **Table 4**.

Table 3. M68HC08 Opcode Map

	Bit-Manipulation		Branch	Read-Modify-Write						Control		Register/Memory							
	DIR	DIR	REL	DIR	INH	INH	IX1	SP1	IX	INH	INH	IMM	DIR	EXT	IX2	SP2	IX1	SP1	IX
HIGH / LOW	0	1	2	3	4	5	6	9E6	7	8	9	A	B	C	D	9ED	E	9EE	F
0	5 BRSET0 3 DIR	4 BSET0 2 DIR	3 BRA 2 REL	4 NEG 2 DIR	1 NEGA 1 INH	1 NEGX 1 INH	4 NEG 2 IX1	5 NEG 3 SP1	3 NEG 1 IX	7 RTI 1 INH	3 BGE 2 REL	2 SUB 2 IMM	3 SUB 2 DIR	4 SUB 3 EXT	4 SUB 3 IX2	5 SUB 4 SP2	3 SUB 2 IX1	4 SUB 3 SP1	2 SUB 1 IX
1	5 BRCLR0 3 DIR	4 BCLR0 2 DIR	3 BRN 2 REL	5 CBEQ 3 DIR	4 CBEQA 3 IMM	4 CBEQX 3 IMM	5 CBEQ 3 IX1+	6 CBEQ 4 SP1	4 CBEQ 2 IX+	4 RTS 1 INH	3 BLT 2 REL	2 CMP 2 IMM	3 CMP 2 DIR	4 CMP 3 EXT	4 CMP 3 IX2	5 CMP 4 SP2	3 CMP 2 IX1	4 CMP 3 SP1	2 CMP 1 IX
2	5 BRSET1 3 DIR	4 BSET1 2 DIR	3 BHI 2 REL		5 MUL 1 INH	7 DIV 1 INH	3 NSA 1 INH		2 DAA 1 INH		3 BGT 2 REL	2 SBC 2 IMM	3 SBC 2 DIR	4 SBC 3 EXT	4 SBC 3 IX2	5 SBC 4 SP2	3 SBC 2 IX1	4 SBC 3 SP1	2 SBC 1 IX
3	5 BRCLR1 3 DIR	4 BCLR1 2 DIR	3 BLS 2 REL	4 COM 2 DIR	1 COMA 1 INH	1 COMX 1 INH	4 COM 2 IX1	5 COM 3 SP1	3 COM 1 IX	9 SWI 1 INH	3 BLE 2 REL	2 CPX 2 IMM	3 CPX 2 DIR	4 CPX 3 EXT	4 CPX 3 IX2	5 CPX 4 SP2	3 CPX 2 IX1	4 CPX 3 SP1	2 CPX 1 IX
4	5 BRSET2 3 DIR	4 BSET2 2 DIR	3 BCC 2 REL	4 LSR 2 DIR	1 LSRA 1 INH	1 LSRX 1 INH	4 LSR 2 IX1	5 LSR 3 SP1	3 LSR 1 IX	2 TAP 1 INH	2 TXS 1 INH	2 AND 2 IMM	3 AND 2 DIR	4 AND 3 EXT	4 AND 3 IX2	5 AND 4 SP2	3 AND 2 IX1	4 AND 3 SP1	2 AND 1 IX
5	5 BRCLR2 3 DIR	4 BCLR2 2 DIR	3 BCS 2 REL	4 STHX 2 DIR	3 LDHX 3 IMM	4 LDHX 2 DIR	3 CPHX 3 IMM		4 CPHX 2 DIR	1 TPA 1 INH	2 TSX 1 INH	2 BIT 2 IMM	3 BIT 2 DIR	4 BIT 3 EXT	4 BIT 3 IX2	5 BIT 4 SP2	3 BIT 2 IX1	4 BIT 3 SP1	2 BIT 1 IX
6	5 BRSET3 3 DIR	4 BSET3 2 DIR	3 BNE 2 REL	4 ROR 2 DIR	1 RORA 1 INH	1 RORX 1 INH	4 ROR 2 IX1	5 ROR 3 SP1	3 ROR 1 IX	2 PULA 1 INH		2 LDA 2 IMM	3 LDA 2 DIR	4 LDA 3 EXT	4 LDA 3 IX2	5 LDA 4 SP2	3 LDA 2 IX1	4 LDA 3 SP1	2 LDA 1 IX
7	5 BRCLR3 3 DIR	4 BCLR3 2 DIR	3 BEQ 2 REL	4 ASR 2 DIR	1 ASRA 1 INH	1 ASRX 1 INH	4 ASR 2 IX1	5 ASR 3 SP1	3 ASR 1 IX	2 PSHA 1 INH	1 TAX 1 INH	2 AIS 2 IMM	3 STA 2 DIR	4 STA 3 EXT	4 STA 3 IX2	5 STA 4 SP2	3 STA 2 IX1	4 STA 3 SP1	2 STA 1 IX
8	5 BRSET4 3 DIR	4 BSET4 2 DIR	3 BHCC 2 REL	4 LSL 2 DIR	1 LSLA 1 INH	1 LSLX 1 INH	4 LSL 2 IX1	5 LSL 3 SP1	3 LSL 1 IX	2 PULX 1 INH	1 CLC 1 INH	2 EOR 2 IMM	3 EOR 2 DIR	4 EOR 3 EXT	4 EOR 3 IX2	5 EOR 4 SP2	3 EOR 2 IX1	4 EOR 3 SP1	2 EOR 1 IX
9	5 BRCLR4 3 DIR	4 BCLR4 2 DIR	3 BHCS 2 REL	4 ROL 2 DIR	1 ROLA 1 INH	1 ROLX 1 INH	4 ROL 2 IX1	5 ROL 3 SP1	3 ROL 1 IX	2 PSHX 1 INH	1 SEC 1 INH	2 ADC 2 IMM	3 ADC 2 DIR	4 ADC 3 EXT	4 ADC 3 IX2	5 ADC 4 SP2	3 ADC 2 IX1	4 ADC 3 SP1	2 ADC 1 IX
A	5 BRSET5 3 DIR	4 BSET5 2 DIR	3 BPL 2 REL	4 DEC 2 DIR	1 DECA 1 INH	1 DECX 1 INH	4 DEC 2 IX1	5 DEC 3 SP1	3 DEC 1 IX	2 PULH 1 INH	2 CLI 1 INH	2 ORA 2 IMM	3 ORA 2 DIR	4 ORA 3 EXT	4 ORA 3 IX2	5 ORA 4 SP2	3 ORA 2 IX1	4 ORA 3 SP1	2 ORA 1 IX
B	5 BRCLR5 3 DIR	4 BCLR5 2 DIR	3 BMI 2 REL	5 DBNZ 3 DIR	3 DBNZA 2 INH	3 DBNZX 2 INH	5 DBNZ 3 IX1	6 DBNZ 4 SP1	4 DBNZ 2 IX	2 PSHH 1 INH	2 SEI 1 INH	2 ADD 2 IMM	3 ADD 2 DIR	4 ADD 3 EXT	4 ADD 3 IX2	5 ADD 4 SP2	3 ADD 2 IX1	4 ADD 3 SP1	2 ADD 1 IX
C	5 BRSET6 3 DIR	4 BSET6 2 DIR	3 BMC 2 REL	4 INC 2 DIR	1 INCA 1 INH	1 INCX 1 INH	4 INC 2 IX1	5 INC 3 SP1	3 INC 1 IX	1 CLRH 1 INH	1 RSP 1 INH		2 JMP 2 DIR	3 JMP 3 EXT	4 JMP 3 IX2		3 JMP 2 IX1		2 JMP 1 IX
D	5 BRCLR6 3 DIR	4 BCLR6 2 DIR	3 BMS 2 REL	3 TST 2 DIR	1 TSTA 1 INH	1 TSTX 1 INH	3 TST 2 IX1	4 TST 3 SP1	2 TST 1 IX		1 NOP 1 INH	4 BSR 2 REL	4 JSR 2 DIR	5 JSR 3 EXT	6 JSR 3 IX2		5 JSR 2 IX1		4 JSR 1 IX
E	5 BRSET7 3 DIR	4 BSET7 2 DIR	3 BIL 2 REL		5 MOV 3 DD	4 MOV 2 DIX+	4 MOV 3 IMD		4 MOV 2 IX+D	1 STOP 1 INH	*	2 LDX 2 IMM	3 LDX 2 DIR	4 LDX 3 EXT	4 LDX 3 IX2	5 LDX 4 SP2	3 LDX 2 IX1	4 LDX 3 SP1	2 LDX 1 IX
F	5 BRCLR7 3 DIR	4 BCLR7 2 DIR	3 BIH 2 REL	3 CLR 2 DIR	1 CLRA 1 INH	1 CLRX 1 INH	3 CLR 2 IX1	4 CLR 3 SP1	2 CLR 1 IX	1 WAIT 1 INH	1 TXA 1 INH	2 AIX 2 IMM	3 STX 2 DIR	4 STX 3 EXT	4 STX 3 IX2	5 STX 4 SP2	3 STX 2 IX1	4 STX 3 SP1	2 STX 1 IX

INH	Inherent	REL	Relative	SP1	Stack Pointer, 8-Bit Offset	
IMM	Immediate	IX	Indexed, No Offset	SP2	Stack Pointer, 16-Bit Offset	
DIR	Direct	IX1	Indexed, 8-Bit Offset	IX+	Indexed, No Offset with Post Increment	
EXT	Extended	IX2	Indexed, 16-Bit Offset			
DD	Direct-Direct	IMD	Immediate-Direct	IX1+	Indexed, 1-Byte Offset with Post Increment	
IX+D	Indexed-Direct	DIX+	Direct-Indexed			

*Pre-byte for stack pointer indexed instructions

High Byte of Opcode in Hexadecimal	F
Low Byte of Opcode in Hexadecimal: 0	2 SUB 1 IX

HC08 Cycles
Opcode Mnemonic
Number of Bytes / Addressing Mode

Table 4. Hexadecimal to ASCII Conversion

Hex	ASCII	Hex	ASCII	Hex	ASCII	Hex	ASCII
$00	NUL	$20	SP *space*	$40	@	$60	‘ *grave*
$01	SOH	$21	!	$41	A	$61	a
$02	STX	$22	“ quote	$42	B	$62	b
$03	ETX	$23	#	$43	C	$63	c
$04	EOT	$24	$	$44	D	$64	d
$05	ENQ	$25	%	$45	E	$65	e
$06	ACK	$26	&	$46	F	$66	f
$07	BEL *beep*	$27	‘ *apost.*	$47	G	$67	g
$08	BS *back sp*	$28	(	$48	H	$68	h
$09	HT *tab*	$29	)	$49	I	$69	i
$0A	LF *linefeed*	$2A	*	$4A	J	$6A	j
$0B	VT	$2B	+	$4B	K	$6B	k
$0C	FF	$2C	, *comma*	$4C	L	$6C	l
$0D	CR *return*	$2D	- *dash*	$4D	M	$6D	m
$0E	SO	$2E	. *period*	$4E	N	$6E	n
$0F	SI	$2F	/	$4F	O	$6F	o
$10	DLE	$30	0	$50	P	$70	p
$11	DC1	$31	1	$51	Q	$71	q
$12	DC2	$32	2	$52	R	$72	r
$13	DC3	$33	3	$53	S	$73	s
$14	DC4	$34	4	$54	T	$74	t
$15	NAK	$35	5	$55	U	$75	u
$16	SYN	$36	6	$56	V	$76	v
$17	ETB	$37	7	$57	W	$77	w
$18	CAN	$38	8	$58	X	$78	x
$19	EM	$39	9	$59	Y	$79	y
$1A	SUB	$3A	:	$5A	Z	$7A	z
$1B	ESCAPE	$3B	;	$5B	[	$7B	{
$1C	FS	$3C	<	$5C	\	$7C	|
$1D	GS	$3D	=	$5D	]	$7D	}
$1E	RS	$3E	>	$5E	^	$7E	~
$1F	US	$3F	?	$5F	_ *under*	$7F	DEL *delete*

Hexadecimal to Decimal Conversion

To convert a hexadecimal number (up to four hexadecimal digits) to decimal, look up the decimal equivalent of each hexadecimal digit in Table 5. The decimal equivalent of the original hexadecimal number is the sum of the weights found in the table for all hexadecimal digits.

Table 5. Hexadecimal to/from Decimal Conversion

15 Bit 8				7 Bit 0			
15 12		11 8		7 4		3 0	
4th Hex Digit		3rd Hex Digit		2nd Hex Digit		1st Hex Digit	
Hex	Decimal	Hex	Decimal	Hex	Decimal	Hex	Decimal
0	0	0	0	0	0	0	0
1	4,096	1	256	1	16	1	1
2	8,192	2	512	2	32	2	2
3	12,288	3	768	3	48	3	3
4	16,384	4	1,024	4	64	4	4
5	20,480	5	1,280	5	80	5	5
6	24,576	6	1,536	6	96	6	6
7	28,672	7	1,792	7	112	7	7
8	32,768	8	2,048	8	128	8	8
9	36,864	9	2,304	9	144	9	9
A	40,960	A	2,560	A	160	A	10
B	45,056	B	2,816	B	176	B	11
C	49,152	C	3,072	C	192	C	12
D	53,248	D	3,328	D	208	D	13
E	57,344	E	3,484	E	224	E	14
F	61,440	F	3,840	F	240	F	15

Decimal to Hexadecimal Conversion

To convert a decimal number (up to $65{,}535_{10}$) to hexadecimal, find the largest decimal number in **Table 5** that is less than or equal to the number you are converting. The corresponding hexadecimal digit is the most significant hexadecimal digit of the result. Subtract the decimal number found from the original decimal number to get the *remaining decimal value*. Repeat the procedure using the remaining decimal value for each subsequent hexadecimal digit.

HOW TO REACH US:

USA/EUROPE/LOCATIONS NOT LISTED:

Motorola Literature Distribution;
P.O. Box 5405, Denver, Colorado 80217
1-303-675-2140 or 1-800-441-2447

JAPAN:

Motorola Japan Ltd.; SPS, Technical Information Center,
3-20-1, Minami-Azabu Minato-ku, Tokyo 106-8573 Japan
81-3-3440-3569

ASIA/PACIFIC:

Motorola Semiconductors H.K. Ltd.;
Silicon Harbour Centre, 2 Dai King Street,
Tai Po Industrial Estate, Tai Po, N.T., Hong Kong
852-26668334

TECHNICAL INFORMATION CENTER:

1-800-521-6274

HOME PAGE:

http://www.motorola.com/semiconductors

M68HC08RG/AD