# Vážení zákazníci,

dovolujeme si Vás upozornit, že na tuto ukázku knihy se vztahují autorská práva, tzv. copyright.

To znamená, že ukázka má sloužit výhradně pro osobní potřebu potenciálního kupujícího (aby čtenář viděl, jakým způsobem je titul zpracován a mohl se také podle tohoto, jako jednoho z parametrů, rozhodnout, zda titul koupí či ne).

Z toho vyplývá, že není dovoleno tuto ukázku jakýmkoliv způsobem dále šířit, veřejně či neveřejně např. umisťováním na datová média, na jiné internetové stránky (ani prostřednictvím odkazů) apod.

*redakce nakladatelství BEN – technická literatura*
redakce@ben.cz

# 3 JEDNOČIPOVÉ MIKROPOČÍTAČE – ZÁKLADNÍ POJMY

Jak již bylo uvedeno v úvodu, považujeme mikropočítač či jednočipový mikropočítač za stavový sekvenční automat. Tím máme na mysli fakt, že zpracování dat (signálu) je prováděno po nedělitelných krocích, v nichž prochází mikropočítač známou posloupností stavů, přičemž průchod těmito stavy závisí kromě momentálních hodnot datových signálů ještě na jejich předchozích hodnotách.

Pod pojmem „jednočipový mikropočítač“ rozumíme spojení mikroprocesoru, paměti a vstupně/výstupních obvodů do jediného celku s konstrukcí na jediném čipu. Prioritně jsou tedy jednočipové mikropočítače konstruovány pro použití ve funkci samostatných řídicích jednotek. Vzhledem k integraci různorodých obvodů na jediném čipu jsou konstrukce jednočipových mikropočítačů výrazněji omezeny ve velikostech paměťových prostorů, rychlosti apod., než je tomu u prostých mikropočítačů, které obvykle nejsou vybaveny žádnou samostatně programově přístupnou vstupně/výstupní periferií.

## 3.1 ARCHITEKTURA JEDNOČIPOVÉHO MIKROPOČÍTAČE

Mikropočítače obecně, a pochopitelně i jednočipové mikropočítače, mohou být konstruovány v jednom ze dvou základních typů architektury. K dispozici je architektura typu von Neumann nebo architektura harvardská.

Mikropočítače konstruované v těchto architekturách se zásadně odlišují přístupem ke konstrukci paměti dat. Zatímco architektura typu von Neumann má pro data i program společnou paměť, architektura harvardská striktně odlišuje paměť programu a paměť dat. Obě řešení mají své výhody i nevýhody. Základní výhodou, která se může při nepozornosti programátora stát katastrofickou nevýhodou, je v architektuře von Neumann možnost vykonávat data jako program a program inter-

pretovat jako data. V harvardské architektuře to možné není (alespoň ne jednoduchým způsobem), neboť ze svého principu na to není mikropočítač s touto architekturou konstruován. Pravdou však je to, že obě architektury je možné s pomocnými obvody převádět z jedné na druhou. U architektury harvardské není výsledek převodu zcela rovnocenný a odpovídající architektuře von Neumann. Je tomu tak proto, že v harvardské architektuře jsou oba adresové prostoty, tj. prostor pro data i program, alokovány od adresy 0 a navíc není možné vkládat data do programu a naopak bez respektování těchto základních obvodových vlastností.

## 3.2
## VLASTNOSTI A ARCHITEKTURA OBVODŮ PERIFERIÍ

V předchozím textu jsme se zmiňovali o paměti programu a paměti dat. Kromě těchto paměťových prostorů oplývají jednočipové mikropočítače i paměťovým prostorem pro ovládání periferií ať už integrovaných nebo externích. Pokud je paměťový prostor periferií ve společném paměťovém prostoru s pamětí dat, říkáme, že periferie jsou paměťově mapovány.

## 3.3
## SBĚRNICE MIKROPOČÍTAČE

Aby mohl mikropočítač pracovat s pamětí programu, dat či obsluhovat periferie, musí být schopen generovat pro tyto obvody řadu signálů, tj. musí být s nimi schopen komunikovat. K této komunikaci slouží sběrnice. Pojmem sběrnice označujeme skupinu signálů sloužících pro řízení komunikace a komunikaci s okolním světem mikropočítače. Každý mikropočítač má k dispozici alespoň tři typy sběrnice. Adresová sběrnice je u mikropočítačů, a zvláště pak jednočipových, obvykle jednosměrná s orientací od mikropočítače k okolí. Slouží k přenosu adresy dat či adresy kódu instrukce. Sběrnice datová je konstruována jako obousměrná a slouží k přenosu dat, která jsou adresována adresovou sběrnicí. Posledním typem sběrnice je sběrnice řídicí. Tuto sběrnici tvoří signály určené k řízení přenosu dat po datové sběrnici ve spolupráci se sběrnicí adresovou. Řídicí signály obvykle označují platnost adresy a dat na příslušných sběrnicích a slouží tak k synchronizaci mikropočítače s okolním světem.

## 3.4
# ČASOVÁNÍ MIKROPOČÍTAČE

Každý mikropočítač (mějme na paměti, že jde o sekvenční automat) vykonává zadaný program krok po kroku. Tento základní stavební kámen programu nazýváme instrukcí. Každá instrukce je složena alespoň z jednoho strojového cyklu a strojový cyklus je složen alespoň z jedné fáze. Dříve byl místo termínu fáze používán termín takt, který je dnes v podstatě synonymem k pojmu strojového cyklu.

Strojovým cyklem označujeme obvykle dobu, která uplyne mezi dvěma přístupy na adresovou sběrnici. Než se tak stane, musí mikropočítač projít alespoň jednou, zpravidla však výrazně větším počtem fází.

Mikropočítače pracují téměř výhradně synchronně, tj. jejich činnost je řízena tzv. hodinovým signálem. Hodinový signál (míníme tím skutečný řídicí signál mikropočítače) je signálem s nejkratší periodou (nejvyšším kmitočtem), se kterým mikropočítače pracují. Nemusí být totožný s kmitočtem krystalu oscilátoru, který se pro generování hodinového kmitočtu používá. Hodinový kmitočet mikropočítače může být i řádově větší či menší než kmitočet krystalu. Z tohoto důvodu je naprostým omylem názor, že čím větší kmitočet krystalu tím větší je výpočetní výkon. Toto porovnávání výkonu může být pravdou pouze tehdy, pokud porovnáváme mikropočítače v rámci jednoho typu. S daleko horším přiblížením toto platí mezi jednotlivými řadami mikropočítačů od jediného výrobce. Zmíněné porovnání už vůbec nelze použít napříč trhem mikropočítačů produkovaných různými výrobci s použitím odlišné konstrukce. Chceme-li přesto použít nějakého kmitočtu pro srovnání a odhad výpočetního výkonu daného mikropočítače vůči nám známému, musíme použít právě informaci o rychlosti strojového cyklu. Ta totiž vyjadřuje mezní kmitočet přenosu dat po sběrnici, jehož je mikropočítač schopen. Mikropočítač může být třeba tisíckrát rychlejší než přenos dat po sběrnici a stejně nebude schopen poskytovat smysluplný výsledek rychleji, než je rychlost přenosu dat. Nedodáme-li po sběrnici data, nemá mikropočítač co počítat a jeho rychlost nám není nic platná. Dalším úskalím je, že v tomto porovnávání obvykle zapomeneme na skutečnost v rozdílu stavby a konstrukce ve smyslu, zda jde o mikropočítač typu CISC nebo typu RISC. V praxi totiž není podstatná rychlost výpočtu triviální operace, ale to, zda mikropočítač bude schopen dostatečně rychle zpracovat naši úlohu, tj. problém daleko složitější, než je obyčejná aritmetická operace. Zmíněného jevu zapomínání na typ konstrukce jsme však bohužel svědky i u samotných výrobců při srovnávání vlastního výrobku s výrobky konkurence.

Alespoň jeden příklad za všechny. Firma Microchip ve svém katalogu srovnává své mikropočítače populární řady 16CXX s konkurencí co do rychlosti a výpočetního výkonu. Pro toto srovnání použila rychlost vykonání instrukce „swap“. Instrukce prohazuje vrchní čtyři bity se spodními v osmibitovém slově. Až potud by to šlo celkem vzít. Bohužel si jako soupeře vybrala mimo jiné i mikropočítač, který tuto instrukci nemá a musí místo ní vykonat čtyři jiné (instrukce posunu). Výsledky testu jsou vyvedeny v grafu, kde mikropočítače řady 16XX znamenitě vítězí o několik koňských délek. Tento způsob testu se snad může objevit v televizní reklamě, ale v konstrukčním katalogu je jeho uvedení zcela nepřípustné.

## 3.5
## ŘADIČ A ARITMETICKO-LOGICKÁ JEDNOTKA

V odborné literatuře bývá často uváděna věta typu „*Mikropočítač je vystavěn okolo osmibitové aritmeticko-logické jednotky*“. Tato věta je sice hezká, ale nepostihuje fakt, že aritmeticko-logická jednotka je v mikropočítači obvod sice důležitý nikoli však zcela hlavní. Složitost této jednotky a poměřujeme-li složitostí obvodu jeho význam v zařízení, ani zdaleka nedosahuje složitosti obvodu řadiče, který je skutečným mozkem mikropočítače. Řadič se skládá, alespoň v hrubém dělení, z registru instrukce, dekodéru instrukce, řídicích a časovacích obvodů mikroprogramu. Řadič se stará o veškeré časování dějů v mikropočítači a o generování řídicích signálů pro datové přenosy po datové sběrnici. Aritmeticko-logická jednotka pak provádí na základě řídicích signálů řadiče aritmetické či logické operace.

Aritmetické operace jsou řešeny obvykle obvodem sčítačky doplněným přenosem z nižšího a do vyššího řádu a obvodem pro realizaci záporného čísla pomocí tzv. dvojkového doplňku.

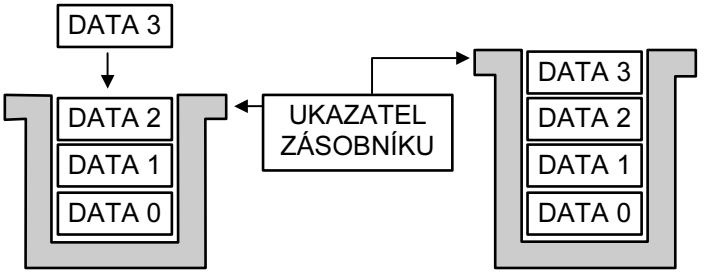
Logické operace obvykle řeší posuvný registr s paralelními vstupy a výstupy a řídicí logikou posunu vlevo či vpravo. Ve složitějších případech bývá posuvný registr nahrazen obvodem typu „barel shifter“, který umožňuje realizovat vícebitové posuny. Obvod typu „barel shifter“ je možné poměrně snadno realizovat sadou multiplexerů.

## 3.6
## PAMĚŤ

Do obvodů aritmeticko-logické jednotky počítáme též vyrovnávací registry operandů a jeden nebo více registrů pro uložení výsledku výpočtu. Pro označení registru výsledku používáme pojem střadač.

Každý mikropočítač je určen pro zpracování dat a je tedy nutné ho vybavit pamětí pro jejich ukládání. Datovou paměť mikropočítače dělíme podle rychlosti přístupu k datům a podle způsobu přístupu k nim na zápisník, zásobník a obecnou datovou paměť. Paměti typu zápisník a zásobník jsou téměř výhradně integrovány společně s řadičem a aritmeticko-logickou jednotkou. Obecná paměť je mnohdy realizována alespoň z části vnějším paměťovým obvodem. Pravidlo o lokalizaci obecné datové paměti a někdy i zásobníku není možné chápat striktně a konkrétní řešení této lokalizace téměř vždy závisí na požadovaném rozsahu paměťových obvodů pro danou aplikaci.

Zápisník je paměť, která slouží pro dočasné uchování dat. Pojem dočasné nevyjadřuje fakt, že se snad data z této paměti ztrácejí, ale fakt, že k paměti typu zápisník má mikropočítač nejrychlejší přístup. Velikost paměti zápisníku je obvykle velmi omezená a zápisník je tedy především určen k uchovávání mezivýsledků při složitějších výpočtech. Zápisník je konstruován jako paměť s libovolným přístupem (překlad slova „Random“ slovem „náhodný“ při označení paměti RAM – Random Access Memory je v tomto kontextu zavádějící).




***Obr. 3.1 Zápis dat na zásobník a modifikace ukazatele***

Zásobník (Stack) je paměť typu LIFO. LIFO je zkratka z anglického označení Last-In First-Out vyjadřujícího skutečnost, že v tomto typu paměti je přístup k uloženým datům sekvenční, tj. poslední zapsaná hodnota je přečtena jako první. K zásobníku se ještě váže pojem vrchol zásobníku, tj. paměťová buňka, na niž ukazuje (adresuje ji) ukazatel zásobníku *obr. 3.1*. Ukazatel zásobníku je inkrementován resp. dekrementován při zápisu resp. čtení dat v případě, že zásobník narůstá směrem k vyšším adresám. Používá se i řešení opačné, tj. zásobník narůstá směrem k nižším adresám. V tomto druhém typu řešení je ukazatel zásobníku při ukládání dat dekrementován a při jejich

čtení inkrementován. Zásobník se obvykle využívá pro krátkodobé uložení dat především proto, že se o něj dělí program (uživatel) a CPU mikropočítače. CPU mikropočítače používá zásobník pro uložení návratové adresy v okamžiku volání podprogramu nebo přerušení. V případě přerušení pak mikropočítače MOTOROLA automaticky ukládají na zásobník též celý programovací model, tj. obsah všech interních registrů CPU. Vhodný soubor instrukcí pro práci se zásobníkem pak umožňuje jednoduché předávání parametrů při volání podprogramů ve vyšších programovacích jazycích. Ve velké většině případů totiž generátory kódu vyšších programovacích jazyků používají pro předávání parametrů podprogramů a funkcí právě zásobník jako obecnou strukturu dat.

Obecná datová paměť mikropočítače slouží k ukládání dat a větších datových struktur. Obvykle má k obecné datové paměti mikropočítač nejpomalejší přístup. Co do kapacity je však tato paměť největší.

Dělíme-li paměť mikropočítače z hlediska rychlosti přístupu, je nutné si uvědomit, v čem tento rozdíl spočívá. Rychlost přístupu k paměti v popisovaném dělení není dána obvodovou konstrukcí samotné paměti, ale určuje ji metoda, kterou používá řadič mikropočítače pro generování úplné adresy dat uložených v paměti.

Názorně je možné popsat situaci na příkladu osmibitového mikropočítače s paměťovým prostorem 64 kB. Je zřejmé, že obsluha datové paměti o rozsahu 64 kB si při přístupu na libovolnou buňku paměti vyžádá 16bitovou úplnou adresu. Zjišťování šestnáctibitové adresy je pro osmibitový mikropočítač relativně zdlouhavá záležitost. Musí totiž pro přenos či přečtení adresy použít dva přístupy na sběrnice a v následujícím přístupu přečíst nebo zapsat data.

Tato situace nastává tehdy, když mikropočítač adresu zjišťuje. Celou operaci čtení nebo zápisu dat je možné urychlit tím, že část adresy mikropočítač „vymyslí“ a část adresy skutečně zjistí. V případě osmibitového mikropočítače vyjde výhodně, když mikropočítač „vymyslí“ horní polovinu šestnáctibitové adresy.

To je zařízeno tak, že horní polovina adresy je v mikropočítači generována implicitně a spodní část se zjišťuje. Používá se řešení, že mikropočítač dosazuje nulovou hodnoty vyššího byte adresy, zatímco nižší byte adresy se zjišťuje. V tomto případě je tedy ke stanovení úplné adresy potřeba pouze jeden přístup na sběrnice a v následujícím přístupu je již možné načítat nebo ukládat data. Nevýhoda tohoto zkráceného adresování spočívá v tom, že máme k dispozici pouze 256 různých adres (zjišťujeme pouze spodní polovinu adresy, horní je implicitně rovna 0). Takto specifickou oblast paměti pak můžeme označit jako

zápisník, neboť se vyznačuje nejrychlejším přístupem, kterého je mikropočítač vůči datové paměti schopen. Nutno podotknout, že popisované řešení je implementováno u jednočipových mikropočítačů Motorola rodiny HC11.

Další výrobci řeší zápisník mikropočítačů jiným způsobem, nicméně výsledná charakteristika tohoto typu paměti dat jako paměti s libovolným a nejrychlejším přístupem platí obecně.

V případě zásobníku je přístup k datům sekvenční a zásobník tedy není paměť s libovolným přístupem. Adresa místa (položky) v zásobníku, na které mikropočítač právě přistupuje, se uchovává v ukazateli zásobníku. Ukazatel zásobníku uchovává úplnou adresu vrcholu zásobníku a tím určuje místo přístupu do zásobníku pro mikropočítač. Adresa se v tomto případě nezjišťuje, a tudíž zápis a čtení dat může proběhnout a obvykle probíhá v jediném cyklu sběrnic. Nevýhoda přístupu k datům přes zásobník je právě v tom, že máme přístupnou vždy pouze jednu položku zásobníku, tj. jeho vrchol.

## 3.7
# BĚH PROGRAMU A ZPRACOVÁNÍ DAT

Program, který mikropočítač vykonává, tvoří jednotlivé instrukce mikropočítače. Pomocí základních instrukcí mikropočítače je tvořen výsledný algoritmus programu. Mikropočítač spouští program bezprostředně po ukončení sekvence základních nastavení. Tato sekvence probíhá v okamžiku resetu (inicializace) a kromě jiných úkolů řeší obvykle pomocí časového zpoždění ustálení kmitočtu oscilátoru, základní nastavení módu periferií a základní nastavení velikostí paměťových prostorů pro program data atd.

Program se spouští od zadané adresy programové paměti. Tato specifická adresa je zapsána řadičem do programového čítače. Tento specializovaný registr je určen k uchování adresy programové paměti a ukazuje na paměťovou buňku s instrukcí, která se bude vykonávat. V průběhu vykonávání instrukce je hodnota programového čítače inkrementována, čímž se zajišťuje přechod na další instrukci. Obsah programového čítače je zcela změněn, pokud se vykonává instrukce skoku. Dá se tedy říci, že instrukce skoku je speciální instrukce pro přenos dat do programového čítače. Z pohledu přenosu dat se tedy instrukce skoku nikterak neodlišuje od instrukcí, které používáme pro načítání dat např. při aritmetických operací. Jediná odlišnost je v tom, že cílem přenosu dat při instrukci skoku není aritmeticko-logická jednotka, ale programový čítač.

Spouštění programu od specifické adresy může být prakticky řešeno dvěma způsoby. První řešení se označuje termínem „restart“ adresa, druhé řešení pak termínem „reset“ vektor **(pozor! Reset vektor není totožný se stavem reset)**. V řešení typu „restart“ je adresa pro spuštění přímo dána konstrukcí řadiče. V řešení typu „reset“ vektor je adresa spuštění programu dána nepřímo. V tomto případě je programový čítač naplněn hodnotou, kterou řadič načte z adresy „reset“ vektoru. Adresa „reset“ vektoru je součástí tabulky vektorů přerušení. Jedná se o vyhrazenou, pevně určenou část programové paměti, která tvoří tabulku hodnot. Za naplnění těchto hodnot odpovídá programátor. Má-li se spustit program z adresy „reset“ vektoru, pak první, co řadič udělá je přečtení hodnoty z příslušné položky tabulky přerušovacích vektorů. V druhém kroku uloží přečtenou hodnotu do programového čítače, čímž vlastně provede instrukci skoku a spustí tak program.

Základní běh programu je možné ovlivnit pomocí tzv. přerušení. Přerušení je zásadní zásah do běhu programu, který spočívá v tom, že se dokončí rozpracovaná instrukce základního programu, uloží se návratová adresa a provede se volání specifického podprogramu, který nazýváme obslužný program přerušení či prostě obsluhou přerušení. V okamžiku volání obsluhy přerušení je nezbytné uložit obsah všech pracovních registrů mikropočítače (obvykle na zásobník). Dále se zakáže vyvolání přerušení od zdroje, který je právě obsluhován, a spustí se první instrukce obslužného programu přerušení. Popisované úkony při volání přerušení dělá obvykle mikropočítač automaticky. Pokud je automaticky nedělá, je nanejvýš vhodné úkony udělat v úvodních instrukcích obsluhy přerušení. Dodržením tohoto doporučení předejdeme mnoha chybám, které se jen velmi těžko odhalují.

Praxe ukazuje, že je vždy bezpečnější uložit pracovní registry (střadač apod.), než to neudělat. Nejčastější chybu, která při nesystematickém ukládání registrů vznikne, můžeme označit jako chybu následnou či chybu druhé iterace. Tato chyba totiž vznikne při opravách programu, kdy se soustředíme na opravu chyby a opomeneme zkontrolovat, zda některý z pracovních registrů přepisujeme. Při velké smůle se chyba může projevit až za poměrně dlouhou dobu. Tato doba je dána zázněj vzniklým při běhu hlavního programu a periodickém volání přerušení. Zázněj vznikne tehdy, když doba potřebná k vykonání celé hlavní programové smyčky včetně volání přerušení je velmi blízká periodě volání přerušení.

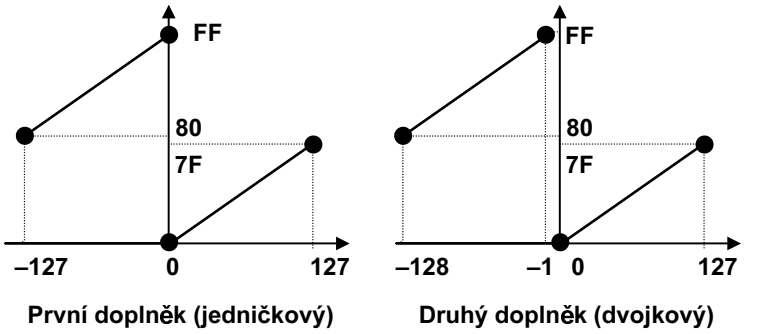
Vzhledem k tomu, že mikropočítač na daný podnět zajišťuje volání přerušení automaticky, setkáváme se opět se dvěma způsoby konstrukce tohoto volání. Stejně jako v případě spouštění programu po inicializaci mikropočítače (resetu) jde v případě volání přerušení o načtení

počáteční adresy obslužného programu přerušení do programového čítače. Mechanizmus načtení počáteční adresy se zajišťuje v mikropočítači konstrukcí řadiče a načtení adresy může být tedy přímé, typu „restart“ nebo nepřímé, typu „vektor přerušení“. Postup získání adresy se v obou případech kryje s postupem popsaným v případě spouštění programu (reset mikropočítače).

## 3.8
## ČÍSLA A JEJICH ZOBRAZENÍ

Mikropočítače pracují v číselné soustavě se základem dvě. Dvojkovou soustavu, jak jí říkáme, tvoří pouze dvě číslice 0 a 1. V dvojkové soustavě tvoříme čísla vyšších hodnot než jedna pomocí vyšších řádů. Jde o obdobu soustavy desítkové, v níž tvoříme pomocí vyšších řádů čísla s hodnotou větší než devět. Až potud je zcela jedno, v jaké soustavě počítáme. Odlišnost mezi oběma soustavami se objeví při práci se zápornými čísly. Zatímco v desítkové soustavě použijeme k označení záporného čísla prosté znaménko mínus, v soustavě dvojkové zobrazujeme záporná čísla odlišným způsobem, a to pomocí prvního nebo druhého (též dvojkového) doplňku. Důvod odlišného zobrazení záporných čísel je prostý a má kořeny v konstrukci aritmeticko-logické jednotky a vhodné obvodové interpretaci zmíněného znaménka mínus. Obvodově je aritmetická část této jednotky řešena sčítačkou. Aby bylo možné použít tuto sčítačku i pro realizaci rozdílu, bylo nutné zvolit vhodné zobrazení záporných čísel. Pro vhodné zobrazení záporných čísel je možné zvolit buď tzv. první doplněk nebo druhý doplněk (zobrazení se také označují termíny jedničkový a dvojkový doplněk).

První doplněk vznikne ze zadaného čísla záměnou jedniček za nuly a nul za jedničky. Jedná se tedy o prostou operaci negace bit po bitu. Podíváme-li se na možné výsledky, zjistíme, že záporná čísla mají vždy v nejvyšším řádu jedničku. Dále zjistíme že součet kladného a záporného čísla realizuje rozdíl. Patrně též objevíme, že dvojí aplikace prvního doplňku na dané číslo ponechá číslo beze změny nebo chceme-li, vytvoří číslo původní. Přijdeme však patrně i na jednu nevýhodu a tou je fakt, že v zobrazení čísel pomocí prvního doplňku máme k dispozici dva kódy pro nulu. První doplněk aplikovaný na hodnotu nula vrací na všech řádech jedničky. Protože však neexistuje záporná nula, můžeme toto zobrazení aplikovat pouze tehdy, jestliže provedeme dodatečnou korekci hodnoty, tj. když v případě, že je výsledkem operace „záporná“ nula, provedeme nad výsledkem dodatečnou operaci prvního doplňku (viz *obr. 3.2*).




***Obr. 3.2 Zobrazení celých čísel v prvním a druhém doplňku***

Nevýhodu se dvěma nulami odstraňuje zobrazení záporných čísel pomocí druhého (dvojkového) doplňku. Číslo s opačným znaménkem vytvoříme z čísla zadaného tak, že v prvním kroku provedeme jeho negaci a v druhém kroku přičteme k výsledku jedničku. Pro zobrazení v druhém doplňku platí stejná pravidla jako v zobrazení pomocí doplňku prvního, avšak s výhodou odstranění dvojí nuly. Za každou výhodu je ale nutné zaplatit. V případě druhého doplňku platíme tím, že rozsah zobrazení je pro záporná čísla vždy o jedničku větší než pro čísla kladná, tj. např. pro osmibitové číslo je rozsah zobrazení v druhém doplňku od –128 po 127.

Dalším problémem při zobrazování a práci s čísly je zobrazení a aritmetické operace s reálnými čísly. Reálná čísla můžeme zobrazovat buď v plovoucí řádové čárce nebo v pevné řádové čárce. Zobrazení v plovoucí řádové čárce je v technice mikropočítačů a především jednočipových mikropočítačů záležitostí matematických knihoven a mikropočítače s tímto zobrazením bez značné programové podpory právě knihovních funkcí nejsou schopny pracovat. Mikropočítače jsou však, alespoň v některých případech, schopny pracovat se zobrazením reálných čísel v pevné řádové čárce. Toto zobrazení nazýváme „fractional“. Pro čísla v zobrazení „fractional“ platí, že jsou vždy z intervalu ⟨0, 1) tj. vždy menší než 1. V některých implementacích zobrazení „fractional“ obsahují čísla v tomto zobrazení i znaménko a jsou tudíž z intervalu (–1, 1). Pevná řádová čárka je vždy umístěna vlevo od nejvyššího řádu (bitu) a dekadický ekvivalent čísla vypočítáme podle vztahu:

$$A = a_n\, 1/2^1 + a_{n-1}\, 1/2^2 + \ldots + a_0\, 1/2^{n+1}$$

kde „A“ je žádané číslo, „a“ je bit řádu „n“ a „n“ označuje řád příslušného bitu čísla. Pro názornost uveďme příklad. Mějme osmibitové číslo vyjádřené v dvojkové soustavě takto:

$$11000001_B$$

Při výpočtu dekadického ekvivalentu zadaného čísla vyjdeme z výše uvedeného vztahu a výpočet bude tedy vypadat:

$$1.1/2^1 + 1.1/2^2 + \ldots + 1.1/2^{256} =$$
$$= 1.0{,}5 + 1.0{,}25 + \ldots 1.0{,}00390625 =$$
$$= 0{,}75390625$$

a dekadický ekvivalent uvedeného čísla vyjde 0,75390625.

Zobrazení typu „fractional“ můžeme doplnit celočíselnou aritmetikou pro práci s hodnotami nad 1, přičemž informaci o znaménku ponese pak celočíselná část zobrazení. Tímto triviálním postupem získáme poměrně slušnou aritmetiku v pevné řádové čárce.

## 3.9
## PROGRAMOVACÍ MODEL

Pojmem programovací model označujeme soupis vlastností a prostředků, jimiž je mikropočítač vybaven a které může využít při konstrukci programů programátor. Jedná se obvykle o popis vnitřní struktury mikropočítače v programátorském smyslu, tj. například kolik a jakých střadačů máme k dispozici, kolik a jakých ukazatelů zásobníku můžeme využít, jaké jsou rozsahy paměti, jaká je programová dostupnost periferií apod.

## 3.10
## VÝVOJOVÉ PROSTŘEDKY

Vývojové prostředky pro programování a ladění programů jednočipových mikropočítačů jsou nezastupitelnými nástroji každého programátora. Mohou být různé kvality, různé ceny, mohou poskytovat více či méně komfortní nástroje pro programování a ladění, mohou být nezbytné nebo takové, bez nichž se lze obejít.

Vzhledem k tomu, že se mikropočítače, a zvláště pak jednočipové mikropočítače, programují ve značné míře pomocí zdrojového textu, považujeme vhodný textový editor za naprostý základ vývojových prostředků.

Překladač zdrojového textu do operačních kódů je druhým nezbytným nástrojem, který potřebujeme pro ladění aplikací. Nejlepší variantou je, když překladač a editor spolupracují alespoň na úrovni hlášení chyb v průběhu překladu a na úrovni interaktivního vyhledávání chybových řádků ve zdrojovém textu.

Dalším nezbytným nástrojem je ladicí program (debugger) aplikace. Ladicí program může být v samostatné formě, tj. běží na hostitelském počítači a simuluje činnost reálného mikropočítače. Takový program pak označujeme termínem simulátor.

Druhou variantou je, že ladicí program běží na hostitelském počítači a přes tzv. emulační rozhraní spolupracuje přímo s mikropočítačem nebo s jeho hardwarovou obdobou. V tomto případě nazýváme celý komplex, tj. ladicí program a emulační zařízení, emulátor. Zásadní rozdíl mezi emulátorem a simulátorem spočívá v tom, že při práci na emulátoru pracujeme s reálným prostředím, které zabezpečuje maximální míru kompatibility s reálným mikropočítačem, zatímco při práci na simulátoru je reálné prostředí pouze simulováno, tj. stupeň kompatibility se skutečným mikropočítačem bývá výrazně nižší. Simulátor se hodí a je ho možné doporučit pro konstrukci částí programu, které nezávisí na reálném čase. Jedná se např. o aritmetické výpočty, korekce a přepočítávání dat. Simulátor se naprosto nehodí k ladění komunikace mezi mikropočítačem a okolím.

Hlavní výhodou simulátoru je jeho cena. Bývá řádově levnější než emulátor. Jakýmsi mezistupněm, který se snaží o kompromis mezi cenou zařízení a jeho vlastnostmi a emulačními schopnostmi, jsou vývojové kity či vývojové desky.

Oproti simulátoru mají výhodu v tom, že jsou konstruovány za pomoci mikropočítače, k jehož ladění jsou určeny. Kompatibilita s mikropočítačem ve skutečné aplikaci je zde téměř stoprocentní. Téměř je slovo použité v tomto případě záměrně. K tomu, abychom mohli ladit program aplikace na vývojové desce, potřebujeme obvykle vlastnosti, které mikropočítač nemá. Potřebujeme např. možnost krokování programem, zastavení programu na zadaném místě (breakpoint) atd. K využití těchto vlastností či schopností vývojové desky je nezbytné, aby byla deska vybavena základním komunikačním programem, který tyto ladicí funkce zajišťuje a který nazýváme obvykle „kernel“. Právě vliv tohoto programu na mikropočítač způsobuje ne zcela stoprocentní kompatibilitu s výslednou aplikací mikropočítače.

Kernel obvykle využívá ke komunikaci s ladicím programem na hostitelském počítači některou z periferií (nejčastěji sériový asynchronní komunikační kanál) mikropočítače a ta je pak pro aplikaci nedostupná. I přes tato omezení se vývojové desky hojně používají a díky své ceně jsou dostupné širokému okruhu konstruktérů a programátorů.