Programování jednočipových mikropočítačů a kompaktních logických automatů.

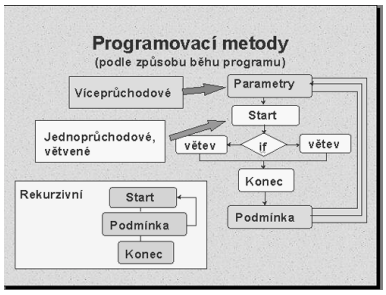
Z.Rozehnal, katedra mikroelektroniky, ČVUT-FEL, Praha

Ačkoliv na první pohled působí spojení jednočipových mikropočítačů a kompaktních logických automatů v nadpisu nesourodě, opak tohoto zdánlivého nesouladu je skutečností. Kompaktní logické automaty mají v oboru řídící techniky až příliš mnoho společného s jednočipovými mikropočítači. V některých případech dokonce vše, jak výhody, tak nevýhody.

1. Základní rysy kompaktních automatů

Programovatelné logické automaty a to zvláště ty, které se pohybují v nižších cenových relacích, jsou konstruovány za použití jednočipových mikropočítačů. Jednočipové mikropočítače použité v konstrukcích přinášejí samozřejmě relativně příznivou koncovou cenu výrobku. To můžeme považovat za výhodu. Další výhodou je obvykle možnost relativně jednoduchého uživatelského programování. Pozor – nemáme na mysli v tomto případě jednoduchou parametrizaci regulační funkce, nýbrž plnohodnotné naprogramování logické funkce podle požadavků uživatele či povahy řešeného problému. Tuto funkci umožňuje operační jádro systému, které však díky mnohdy omezenému výkonu jednočipového mikropočítače nemůže zabezpečit všechny požadavky obvykle kladené na funkce operačního systému. Tuto zdánlivou nedokonalost do jisté míry vyvažuje nasazení těchto kompaktů v oblasti autonomních regulačních systémech ve většině případů s nulovým či omezeným přístupem z nadřízeného řídícího systému. Zde se však poprvé dotýkáme základního problému programování těchto automatů, které musí být provedeno kvalitně a s hlubokou znalostí problematiky programování. Programátor se totiž nemůže obvykle spolehnout na to, že některé chyby za něj odhalí a zachrání operační systém. Ten totiž po pravdě řečeno v těchto automatech není.

2. Programovací metody (z hlediska běhu programu)



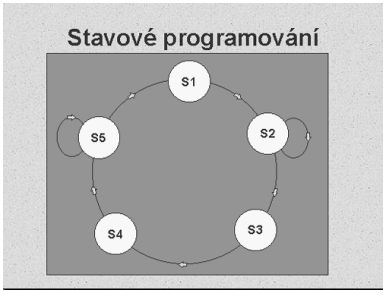
Programovací metody z obecného hlediska, můžeme studovat například z pohledu na způsob běhu programu. Jiným pohledem může být pohled z hlediska reakce na událost (skutečnost, realitu či vnější signál). Věnujme se nejprve pohledu, kterým zkoumáme způsob běhu programu. Podle běhu můžeme programy rozdělit na jednoprůchodové, víceprůchodové (vícenásobné volání) a rekurzivní. Pro programování logických automatů nebo chceme-li jednočipových mikropočítačů není např. rekurzivně napsaný program zrovna ten způsob, který bychom preferovali a mohli ho doporučit. Pokud je to jenom trochu možné, vyhneme se mu. Logické automaty jsou nasazovány ve většině případů do reálného prostředí tj. pořizují vstupní data či parametry pro běh programu ze signálů. Ty mohou být do značné množství zatíženy (rušeny) náhodným signálem (šumem). Vzhledem k tomu, že rekurze je vhodná pro opakující se činnost, kdy není přesně znám počet opakování, není signál zatížený šumem zcela jistě tím pravým signálem, který bychom v rekurzi potřebovali do podmínky jejího ukončení. Dále nám v rekurzivním volání podprogramů brání mnohdy omezený rozsah zásobníku jednočipového mikropočítače a neposlední řadě i výrobci automatů, kteří rekurzi v programovacím jazycích usilovně hledají a systematicky ničí chybovým hlášením překladače. Víceprůchodové řešení programu je možné připustit, ovšem s poznámkou, že maximální počet průchodů musí být předem známý a konstantní. Jedině tak je možné se účinně bránit „zacyklení“.

Pojem jednoprůchodový či víceprůchodový program užíváme z hlediska dosažení výsledku jako reakce na vstupní signál. Víceprůchodový program je například vždy PID regulátor. Zde je dobře znám problém identifikace regulované soustavy a nastavení vhodného tlumení regulační smyčky.

3. Programovací metody (z hlediska reakce na skutečnost)

Z hlediska reakce na skutečnost rozlišujeme v zásadě dva typy programovaní a to programování nevázané v čase a programování v reálném čase. Pro programování nevázané v čase můžeme použít jednovláknové či vícevláknové programování (podporuje operační systém Windows) nebo objektové programování. Pozor zde není pojmem objektové programování myšleno objektově orientované programování, ale programování podle objektu (modelu) používané v simulacích. Pro programování jednočipových mikropočítačů a tím i logických automatů v reálném čase se dá doporučit stavový způsob programování, událostní způsob programování nebo kombinace obou způsobů. Programování za využití Petriho sítí není zatím dostatečně probádáno a není zde jasný přístup, který by vedl jednoznačně k cíli. Zdá se, že Petriho sítě budou dobře použitelné pro relativně rozsáhlé systémy s vágním popisem vlastností. Tato oblast je však zatím zcela jistě mimo výkonové možnosti jednočipových mikropočítačů. Je nutné říci, podle dosud nabytých zkušeností z výuky stavového programování, že dobré zvládnutí této metody působí, díky poněkud nepřirozené (odvyknuté) logice uvažování, nesnáze i erudovaným programátorům.

4. Stavové programování

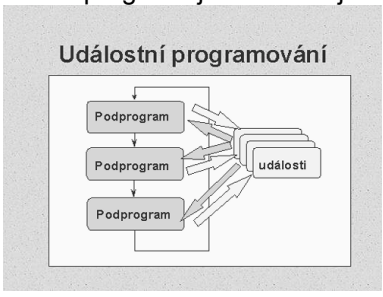


Metoda stavového programování se hojně využívá pro programování víceúlohových aplikací běžících v reálném čase. Princip metody stavového programování je v zásadě jednoduchý a je postaven na principu podmíněného příkazu pro akci a pro přechod na další stav. Každý proces, který je naprogramován pomocí stavového programování musí držet ve speciální proměnné číslo stavu v němž se aktuálně nachází. Pro příklad uveďme jednoduchý příklad programu (v jazyce C) nějakého procesu, který je naprogramován pomocí stavového programování.

```
switch (stav)
        {
        case 0:
                // volaní podprogramu pro provedení akce
                if (akce0()) then stav++;
        break;
        case 1:
                if (akce1()) then stav = 0;
        break;
        default:
                stav = 0; // ošetření nedefinovaných stavů
        break;
        }
```

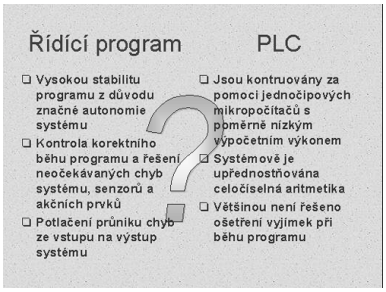
5. Událostní programování

Pokud programujeme složitější aplikace v nichž běží paralelně více úloh a potřebujeme-li tyto aplikace vzájemně svázat přenosy dat, je výhodné jednotlivé úlohy naprogramovat stavovým programováním a vazbu mezi úlohami řešit pomocí programování událostního. Programování je postaveno na myšlence předávání zpráv (událostí) přes společnou datovou strukturu (v jednoduchých případech) nebo prostřednictvím tzv. manažeru zpráv (případy složitější). Zprávy musí být vhodně kódovány tj. musí být známo co dané číslo zprávy znamená, jaké má zpráva parametry a jakou odezvu vrací. Principielně je událostní způsob programování jednoduchý, nicméně při bližším průzkumu můžeme narazit na značné potíže. Velmi obtížně se například



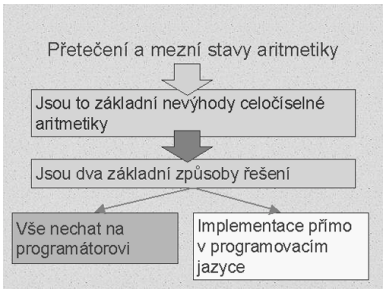
hledají chyby typu „deadlock“ a problémy mohou vznikat i při hledání řešení pro nezpracované zprávy (musíme předejít jejich hromadění).

6. Srovnání požadavků na programové vybavení a možností poskytovaných jednočipovými mikropočítači



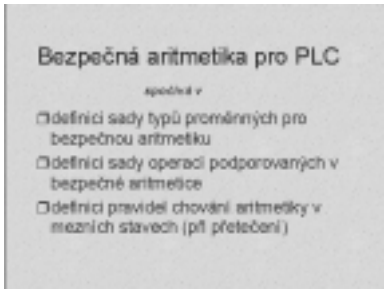
Obrázek shrnuje základní požadavky na programové vybavení programovatelných logických automatů. Vzhledem k tomu, že tyto programovatelné automaty jsou postaveny za pomoci jednočipových mikropočítačů (především z cenových důvodů), je mnohdy obtížné tyto požadavky splnit. Základním problémem je relativně nízký výpočetní výkon, který je u jednočipových mikropočítačů k dispozici. To si obvykle uvědomují i výrobci těchto automatů. Z tohoto důvodu automaty disponují především celočíselnou aritmetikou, která je pochopitelně méně náročná na výkon procesoru než aritmetika v pevné nebo plovoucí řádové čárce. Další problém, který u těchto typů automatů není obvykle řešen vůbec a nebo pouze okrajově, je problém výjimek (chyb) vzniklých při běhu programu (např. dělení nulou, přetečení indexu pole, rozsahu proměnných atp.).

7. Přetečení a mezní stavy aritmetiky



Velmi častými jevy, které musí programátor při návrhu řídícího systému uvažovat, jsou přetečení a mezní stavy celočíselné aritmetiky. Tyto jevy patří mezi základní nevýhody celočíselné aritmetiky. Zatímco pro aritmetiku v plovoucí řádové čárce jsou obvykle definovány koncové hodnoty +MAX a -MAX s nimiž je možné dále počítat nebo je alespoň detekovat, po aritmetiku celočíselnou takové definice v naprosté většině případů neexistují. Je to hned ze dvou důvodů. Prvním z nich je to, že aritmetické operace vycházejí mnohem náročněji, než bez detekce těchto stavů V druhé řadě pak proto, že v některých případech je možné přetečení proměnné velmi elegantně využít (například při odměřování délky impulsu pomocí jednoduchého časovače). Pokud se však přesto chceme nějakým způsobem touto problematikou zabývat nebo ji řešit, dospějeme k názoru, že jsou v podstatě dvě možné cesty řešení. Veškerou odpovědnost za tyto jevy přenést na programátora koncové řídící úlohy. Toto řešení je však spíše „neřešení“. Druhou variantou je podpořit záchyt mezních stavů jak „operačním“ systémem programovatelného automatu, tak překladačem ve fázi překladu zdrojového textu.

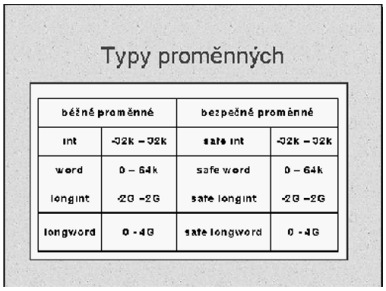
8. Bezpečná aritmetika pro PLC



Pokus o řešení problému přetečení a mezních stavů aritmetiky byl proveden ve spolupráci s firmou MicroPEL (výrobce programovatelných logických automatů). Projekt byl nazván „Bezpečná aritmetika pro PLC“. Následující obrázky shrnují některé zajímavé poznatky z tohoto projektu. Za základní popis či definici bezpečné aritmetiky můžeme považovat prosté tvrzení, že je to aritmetika, která za žádných okolností nepřeteče. Pokud chceme podpořit bezpečnou aritmetiku jak systémově, tak ve fázi překladu, musíme definovat sadu typů proměnných, se kterými bude bezpečná aritmetika pracovat. Dále pak musíme definovat sadu aritmetických operací, které bude tato aritmetika podporovat a v neposlední řadě musíme definovat pravidla chování aritmetiky v mezních stavech tj. při „přetečení“.

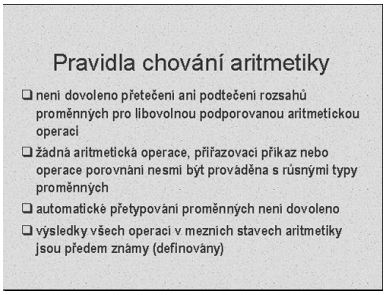
9. Typy proměnných

Typy proměnných, pro které má smysl definovat bezpečnou aritmetiku, jsou uvedeny v tabulce. Jedná se o celočíselné znaménkové a neznaménkové typy. Typ byte byl záměrně vynechán neboť z důvodu malého číselného rozsahu se v podstatě pro aritmetické výpočty nepoužívá. Proměnnou o rozsahu



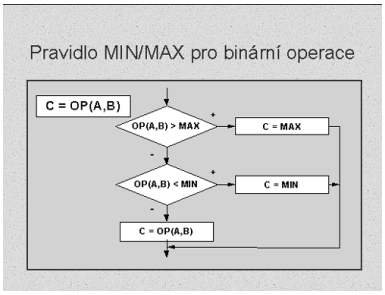
jednoho byte spíše používáme pro uložení znaku popř. ve formě pole byte pro uložení řetězce znaků. Z tabulky dále plyne, že rozsahy bezpečných typů proměnných se kryjí s rozsahy běžných typů proměnných. Není to tedy nic, co by se z pohledu typů proměnných vymykalo běžnému standardu. Bezpečné proměnné musí mít tak nějaký příznak, který vyjadřuje chování proměnné. V našem případě tento příznak tvoří slovo „safe“.

10. Pravidla chování aritmetiky



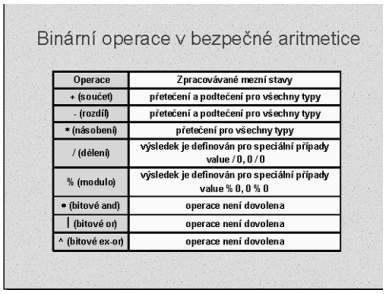
Pravidla chování bezpečné aritmetiky jsou prostá. Požadujeme, aby při libovolné aritmetické operaci a pro libovolný typ bezpečné proměnné nedošlo k přetečení nebo podtečení rozsahu. Toto pravidlo se může zdát jasné a zcela zřejmé. Když se však nad ním zamyslíme z hlediska programátora, můžeme získat pocit, že pokud se aritmetika bude chovat tímto způsobem, nebudeme schopni s ní pracovat. Neustále nás bude napadat při psaní libovolného řádku zdrojového textu, zda výpočet dopadne tak, že aritmetika nepřeteče. To je ale to, co by nás mělo napadat i v okamžiku, kdy pracujeme s běžnými typy proměnných. U řady programátorů budí vlastnosti bezpečné aritmetiky smíšené pocity a v zásadě jsou jim tyto vlastnosti nepřirozené. Opak je však skutečností. Programátoři si zcela automaticky zvykli, odhlížet od problematiky přetečení a pracují s běžnými proměnnými a běžnou aritmetikou zcela bez ostychu a prostě věří, že vše dobře dopadne. To je však hluboký omyl. Ekvivalent zmíněného pravidla zákazu přetečení nebo podtečení můžeme vysledovat na chování operačního zesilovače. Pokud by se choval operační zesilovač stejně „absurdně“ jako běžná aritmetika, tak by jeho mezní odezva na zvyšování diference vstupních napětí nekončila v saturaci, ale výstup by plynule přešel z maximální kladné hodnoty na maximální zápornou hodnotu výstupního napětí. Je zřejmé, že takové chování operačního zesilovače by u většiny elektroniků budilo přinejmenším stejně smíšené pocity, jako mají programátoři při prvním kontaktu s myšlenkou bezpečné aritmetiky.
Dalším důležitým pravidlem je, že jakákoliv podporovaná aritmetická operace je povolena pouze mezi proměnnými stejného typu. Tento požadavek je sice zřejmý, nicméně se jedná o snad nejčastěji porušované pravidlo (mezi největší hříšníky patří obvykle programátoři v jazyce C).
Důležitou možností pro programátora je možnost přetypování proměnné. Je to sice z hlediska čistoty pojetí bezpečné aritmetiky možnost diskutabilní, ale z praktických důvodů je možné ji povolit. Nezbytné však je, aby přetypování nedělal překladač o své vůli sám, ale pouze na výslovný pokyn programátora.
Když už při použití aritmetické operace dojde na přetečení nebo podtečení, musí být předem známo a definováno, jak se aritmetika zachová. Programátor tak může předvídat další vývoj výpočtu po přetečení nebo podtečení. V případě potřeby pak může výpočet korigovat doplňkovými pravidly tak, aby se řízení daného systému nezhroutilo.

11. Pravidlo MIN/MAX pro binární operace



Pokud budeme implementovat pravidla bezpečné aritmetiky programově, usoudíme zcela jistě, že pravidlo MIN/MAX bude vhodnou interpretací požadavku na zákaz přetečení nebo podtečení. Přestože se toto pravidlo se jeví zcela přirozeně a jeho implementace se zdá být jednoduchá, můžeme nalézt příklady, kdy dojde při jeho použití k mírně překvapivému chování aritmetiky. I když překvapení není vždy zcela vítaný jev, můžeme se s ním smířit. Horší však je, že v některých případech nelze pravidlo MIN/MAX dosti dobře použít.

12. Binární operace pro bezpečnou aritmetiku

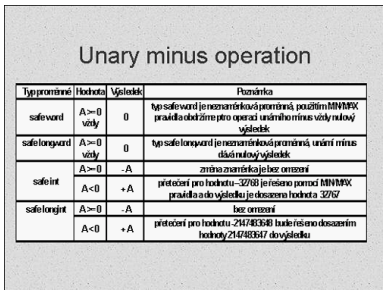


V bezpečné aritmetice povolujeme všechny typy binárních aritmetických operací včetně operace modulo tj. operace zjištění zbytku po celočíselném dělení. Bitové operace nad datovými typy bezpečné aritmetiky (safe int, safe word atd.) nejsou dovoleny. Je to z důvodu pricipiálního i věcného. Bitové operace nad aritmetickými typy jsou sice hojně využívané (zvláště programátory v jazyce C) nicméně zde nemohou být povoleny zvláště pro znaménkové typy aritmetických proměnných. Pokud by tyto operace byly povoleny, pak například vymazáním nejvyššího bitu proměnné by došlo nejen ke změně znaménka, ale i k zásadní změně absolutní hodnoty proměnné (např. u typu safe int by vymazání nejvyššího bitu u hodnoty –1 vedlo k hodnotě 32767). Navíc, pokud bychom změnu nejvyššího bitu považovali za změnu znaménka, produkovala by tato bitová operace jiné výsledky než násobení proměnné hodnotou –1 (to je též operace změny znaménka). Z pricipiálních důvodů pak nejsou bitové operace povoleny ani s neznaménkovými typy proměnných.

13. Přetypování v bezpečné aritmetice

Přetypování proměnné je sice v bezpečné aritmetice povoleno, neprobíhá však automaticky jako např. u mnohých překladačů jazyka C. Přetypování proměnné může provést překladač pouze na výslovné přání programátora. Je to z praktických důvodů. Přetypování totiž v bezpečné aritmetice vede k vedlejším efektům, které si programátor musí vždy řádně uvědomit. Ukázkou mohou být oba uvedené příklady. Pokud bychom řešili uvedené případy přetypování například v jazyce C, řada překladačů by explicitní přetypování vůbec nevyžadovala. Pokud ano, pak by po přetypování dávaly uvedené výrazy s velkou pravděpodobností totožné výsledky. V bezpečné aritmetice tomu tak být nemusí. Operandy A a B jsou typu word. V prvním případě je tedy nejprve proveden součet v závorce. Protože se jedná o součet běžných proměnných, není přetečení zakázáno. Pokud k němu dojde, nic se neděje a výsledek součtu je přetypován na bezpečnou proměnnou. V druhém případě dojdeme za podobných předpokladů k odlišnému výsledku. Zde se nejprve provede přetypování, tj. z běžných proměnných se stanou bezpečné proměnné (pozor - v uvedeném případě se negeneruje pro operace přetypování žádný kód, přetypování probíhá pouze uvnitř překladače). Tyto proměnné jsou sečteny pomocí bezpečné operace součtu, která ovšem na rozdíl od předchozího případu nepřeteče. Díky MIN/MAX pravidlu se dosadí v takovém případě maximální hodnota tj. 65535.

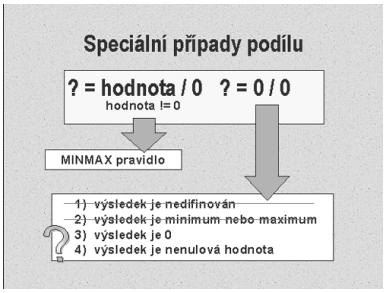
14. Operace „unární mínus“



Operace unárního mínus nahrazuje zápis typu 0-A, kde A je daná proměnná. Protože se jedná v podstatě o binární operaci rozdílu s implicitně dosazeným prvním operandem a současně s výpočtem rozdílu, musí být zajištěno i pravidlo MIN/MAX. Použití pravidla MIN/MAX bude generovat zajímavý efekt. Pro neznaménkové proměnné bude výsledek unárního mínus vždy roven nule. To je sice překvapivé, nicméně výsledek je „správný“. Obsahuje-li proměnná typu safe word nenulovou hodnotu, unární mínus dosadí do výsledku operace hodnotu 0. To je korektní chování, protože pro kladná čísla platí, že pokud nedojde k přetečení, tak rozdíl vždy produkuje menší hodnotu výsledku než je hodnota prvního operandu. Nulový výsledek při přetečení dodržuje, když ne „správnou“ hodnotu výsledku, tedy alespoň „trend“ operace rozdílu kladných čísel.

15. Speciální případy podílu a operace modulo

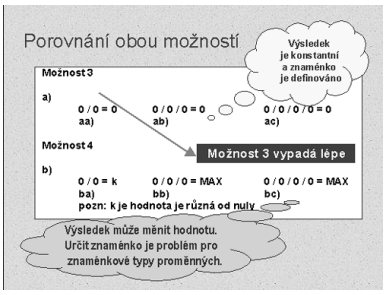
Podíl je jediná aritmetická operace, která je ve svém základě bezpečná. U podílu nemůže nastat přetečení. Bohužel podíl skrývá úskalí dělení nulou. Vyřešit tento problém bohužel není ve všech případech exaktně možné. Pro řešení podílu budeme nulu považovat za kladné číslo v případech, kdy



se jedná o dělení nenulové hodnoty nulou. V těchto případech můžeme použít pravidlo MIN/MAX a dosadit minimální nebo maximální hodnotu rozsahu proměnné. Znaménko výsledku se bude řídit znaménkem čitatele. Případ, kdy dělíme nulu nulou, tímto způsobem řešit nelze. Ve vyšší matematice řešíme výrazy typu nula dělená nulou pomocí výpočtu limity. Výpočet limity však není v řídících systémech s programovatelnými logickými automaty možný. Museli bychom totiž nejen provádět samotný výpočet, ale při každém jeho provádění by bylo nutné vyhodnocovat trend čitatele a jmenovatele. Na takové vyhodnocení není bohužel ani čas ani dostatek paměťového prostoru. Pro tyto případy podílu se musíme uchýlit ke spekulaci a výsledek nějakým rozumným způsobem dodefinovat. Jaké jsou možnosti ? Jsou pricipiálně čtyři.

1. výsledek je nedefinovaný – možnost nelze použít, protože řídící proces nejde ve většině případů je tak beztrestně přerušit a žádat o nápravu
2. výsledek je minimum nebo maximum – možnost je nepoužitelná pro znaménkové typy proměnných, protože bychom neuměli rozumně rozhodnout o znaménku výsledku
3. výsledek je roven 0 – o této možnosti je možné uvažovat
4. výsledek je nenulová hodnota – tato možnost by byla také řešením

16. Porovnání obou možností



Pro volbu možnosti (3 nebo 4) stanovení výsledku dělení můžeme pro stanovení výsledku uvedené operace prozkoumat, jak se bude výsledek chovat při vícenásobném dělení. Pro případ 3 vychází výsledky vícenásobného dělení stále nulové. Pro případ 4 je výsledek prvního dělení stanovená konstanta k. Při opakovaném dělení se výsledek změní z hodnoty „k“ na MAX nebo MIN podle toho zda k bude menší nebo větší než nula. Tím ovšem narážíme na problém. Má být „k“ menší nebo větší než nula ? Z obou možností vybereme možnost 3, protože má jednodušší pravidlo pro stanovení výsledku a výsledek je konstantní. Výsledek též, odpovídá zkratu na odporovém děliči, tj. reálné situaci, která v praktických řídících aplikacích nastává a tudíž jsou programátoři zvyklí takové situace programově ošetřovat.

17. Závěr

Bezpečná aritmetika má krom praktického užití i jisté vedlejší výchovné efekty. Upozorňuje totiž na problémy celočíselné aritmetiky, které si mnozí z programátorů (i velmi zkušených) mnohdy odmítají uvědomit. Dle zkušeností spíše z lenosti než neznalosti.