2 PROCES TVORBY PROGRAMOV PRE SIGNÁLOVÉ PROCESORY

Ciele cvičenia:

- opis základných vývojových nástrojov dostupných pre DSP
- princíp tvorby knižničných funkcií v asembleri pre procesor Blackfin
- demonštrácia základných vlastností integrovaného prostredia VisualDSP++
- opis zlomkového (fractional) formátu

2.1 ÚVOD

Vysoký výpočtový výkon DSP je len jedným z nutných predpokladov pre úspešnú realizáciu výkonných systémov ČSS na báze DSP. Na dosiahnutie vysokého výpočtového výkonu v praktických aplikáciách má podstatný (a často dokonca dominantný) vplyv kvalita dostupných **vývojových prostriedkov**. Aj keď **klasické DSP** s aritmetikou v pevnej rádovej čiarke (napr. Analog Devices ADSP218x, Motorola DSP560xx, Texas Instrument TMS30C5x a pod.) sú v súčasnosti z hľadiska typov vývojových prostriedkov porovnateľné s univerzálnymi jednočípovými mikroprocesormi[1], vzhľadom na typickú požiadavku ich činnosti v reálnom čase je **nízko-úrovňové programovanie** v asembleri často jedinou reálnou alternatívou pri tvorbe programového vybavenia pre túto triedu DSP. V rámci cvičení v predchádzajúcich rokoch, keď bol na cvičeniach nosným DSP procesor Analog Devices ADSP2181, bol dôraz kladený predovšetkým na programovanie v asembleri [1], ktoré patrí medzi **základné prostriedky** na tvorbu programov.

Súčasné **moderné DSP**, medzi ktoré patria[2] aj procesory firmy **Analog Devices na báze jadra Balckfin** využívané v rámci cvičení, sú však podstatne výkonnejšie[3] ako klasické DSP. Táto skutočnosť vyžaduje **kvalitatívne odlišný prístup** k efektívnej tvorbe programového vybavenia. Programovanie v asembleri je využívané predovšetkým pre tvorbu efektívnych knižničných funkcií resp. pre optimalizáciu

[1] V oblasti univerzálnych (jednočipových) mikroprocesorov sú v súčasnosti široko využívané vyššie programovacie jazyky (typicky jazyk C, C++, Embedded C, …), operačné systémy reálneho času a pod. Tieto vývojové prostriedky existujú aj pre klasické DSP, ich využitie v reálnych systémoch na báze DSP je však do určitej miery limitované požiadavkou na prácu v reálnom čase resp. snahou o minimalizáciu veľkosti programových pamätí. V praktických aplikáciách sú preto tieto prostriedky zvyčajne kombinované s kódom v asembleri, ktorý je využitý napr. vo forme optimalizovaných knižníc resp. priamo vložených asemblerovských príkazov.

[2] Pre upresnenie je potrebné uviesť, že procesory Analog Devices s jadrom Blackfin sú kombináciou architektúry DSP a klasického RISC procesora. Uvedená kombinácia je v súčasnosti jednou z najlepších architektúr medzi DSP s aritmetikou s pevnou rádovou čiarkou. Problematika bude podrobnejšie analyzovaná v rámci prednášok

[3] Taktovacie frekvencie moderných DSP sa blížia k hranici 1 GHz, existujú 2-jadrové a viac-jadrové verzie procesorov, procesory umožňujú adresovanie stoviek megabajtov adresového priestoru, ...

kritických častí kódu. Zvyšok aplikácie je v súčasnosti typicky realizovaný vo vyššom programovacom jazyku. Uvedenému trendu sú prispôsobené aj cvičenia z predmetu SPvT, kde programovanie **v jazyku C** bude tvoriť základ programovania cieľových DSP.

Príklady jednotlivých vývojových prostriedkov a väzieb medzi nimi budú demonštrované na vývojových prostriedkoch pre signálové procesory Analog Devices s jadrom Blackfin, ktoré sú súčasťou integrovaného vývojového prostredia **VisualDSP++** [2]. Tieto prostriedky budú využívané aj v rámci ďalších cvičení pri vysvetľovaní základných vlastností jadra procesora Blackfin ako aj pri práci s reálnymi technickými prostriedkami (vývojové dosky EZ-KIT21535 Lite, EZ-KIT21533 Lite a EZ-KIT21561 Lite od firmy Analog Devices).

2.2 VÝVOJOVÉ PROSTRIEDKY PRE DSP

Do tejto kategórie patria predovšetkým asemblery, linkovacie programy, knižničné funkcie, simulátory, vývojové dosky a emulátory.

2.2.1 ASEMBLERY

Asembler je program, ktorý prekladá špecifické inštrukcie procesora zapísané v zdrojovom ASCII súbore do binárneho **objektového kódu** (object code) pre cieľový DSP. Zvyčajne tento objektový kód (pokiaľ je v tzv. **relatívnom tvare**) vyžaduje dodatočnú transformáciu (**relokáciu** a **linkovanie**), ktorá sa realizuje linkovacím programom (**linkerom**). Linker generuje vykonateľný binárny kód pre cieľový DSP. Vo fáze linkovania je možné použiť dostupné **knižničné funkcie**.

Väčšina súčasných asemblerov sú tzv. **macro asemblery**, ktoré umožňujú definovať (alebo používať preddefinované) parametrizovateľné bloky kódu (**makrá**), ktoré sú do zdrojového kódu vkladané počas prekladu. Makrá umožňujú programátorovi zmenšiť množstvo zdrojového kódu, ktorý je treba udržiavať (čím je možné do určitej miery zvýšiť spoľahlivosť programov) ako aj eliminovať nadbytočné inštrukcie, ktoré je potrebné použiť pri volaní podprogramov a sú preto v programoch pre DSP veľmi často využívané. Nasledujúci kód dokumentuje použitie asemblerovského makra pri realizácii FIR filtra pomocou klasického DSP Motorola DSP5600x, ktorý obsahuje jednu MAC jednotku:

```
; definícia makra „FIR", ktoré implementuje FIR filter s N koeficientmi
FIR     macro  N
        clr    a
        rep    #N-1
        mac    x0,y0,a          x:(r0)+,x0      y:(r4)+,y0
        macr   x0,y0,a (r0)-
        endm

; inicializácia smerníkov na koeficienty a dáta
        move   #data,r0
        move   #koeficienty,r4

; volanie makra pre FIR filter s 512 koeficientmi
        FIR    512
```

Už na prvý pohľad je vidno zásadný rozdiel medzi efektívnym programom pre DSP a programom pre typický jednočípový procesor (napr. na báze jadra Intel 8051). Asembler pre DSP umožňuje zápis podstatne „širšieho" operačného kódu (napr. inštrukcia mac), čo je dané vnútornou architektúrou dátových ciest v DSP.

Zaujímavou alternatívou ku klasickým asemblerom sú tzv. *algebraické asemblery*, u ktorých je snaha používať štýl písania asemblerovských programov, ktoré pripomínajú algebraický zápis algoritmu. Tento prístup využíva napr. firma Analog Devices, čo dokumentuje nasledujúci kód FIR filtra (bez využitia makier) pre procesor ADSP21xx s jednou MAC jednotkou:

```
.ENTRY      fir;
fir:    MR = 0, MX0 =  DM( I0, M1),    MY0 = PM(I4, M5);
        DO sop UNTIL CE;
sop:            MR = MR + MX0*MY0( SS ), MX0=DM(I0, M1), MY0=PM(I4, M5);
        MR = MR+MX0*MY0(RND);
        IF  MV  SAT  MR;
        RTS
```

Dátové cesty moderných DSP obsahujú väčší počet MAC jednotiek. Nasledujúci kód zapísaný v algebraickom asembleri dokumentuje program pre realizáciu FIR filtra v jadre procesora Blackfin, ktoré obsahuje dve MAC jednotky:

```
            ...
            LSETUP(E_MAC_ST, E_MAC_END) LC1=P2>>1;                  //Loop 1 to Nc/2 - 1
            A1 = R2.L*R1.L, A0 = R2.H*R1.H  ||  R2.H=W[I2++]  ||  [I3++]=R3;
E_MAC_ST:   A1 += R0.L*R2.H,  A0 += R0.L*R2.L  || R2.L=W[I2++]  || R0=[I1--];
E_MAC_END:
           A1 += R0.H*R2.L, A0 += R0.H*R2.H   ||  R2.H=W[I2++] ;
```

Kód jasne odráža skutočnosť, že efektívne[4] programovanie v asembleri je pre moderné DSP značne náročná úloha a preto je v praxi žiadúce minimalizovať priame programovanie v asembleri.

2.2.2 KNIŽNIČNÉ FUNKCIE

Pre aplikácie na báze DSP majú optimalizované knižnice kľúčový význam. Aj keď typické jadrá algoritmov ČSS implementované pomocou DSP sú relatívne krátke, ich optimalizácia je z pohľadu programátora značne náročná úloha, ktorá vyžaduje dobrú znalosť architektúry cieľového DSP. Naviac je často potrebné zvážiť možnosti modifikovať samotný algoritmus ČSS do tvaru, ktorý je pre danú architektúru vhodnejší, čo zvyčajne vyžaduje špecifické znalosti z oblasti teórie ČSS. Je logické, že výrobcovia DSP poskytujú pre svoje procesory **optimalizované kódy**, ktoré umožňujú relatívne efektívnu implementáciu základných algoritmov ČSS (FIR, IIR, FFT, DCT…). Tieto kódy sú dostupné v rámci WWW stránok jednotlivých výrobcov (napr. [3], [4], [5]), aplikačných príručiek a špecializovaných kníh (napr. [6], [7], [8]).

Aj keď je často možné tieto kódy ďalej vylepšiť, sú uvedené kódy dobrým štartovacím bodom pre vývoj aplikačných programov. Je dokonca výhodné pri zoznamovaní sa s architektúrou DSP využívať hotové programy resp. funkcie. Klasický postup, t.j. počiatočné zvládnutie celej inštrukčnej sady a následná snaha o písanie vlastných efektívnych programov je málo účinným prístupom k zvládnutiu programovania DSP. V ďalších cvičeniach budeme preto analyzovať takéto (relatívne jednoduché) kódy pre FIR filter, IIR filter a FFT a na týchto príkladoch budeme vysvetľovať podstatné vlastnosti procesorov ADSP Blackfin.

Knižničné funkcie je možné pomocou asemblera preložiť do relatívneho objektového tvaru a spojiť pomocou programu – **knihovníka** do jedného súboru –

[4] Pri programovaní je treba uvažovať aj závislosti medzi jednotlivými inštrukciami, ktoré môžu spôsobiť vsúvanie nop inštrukcií v riadiacej jednotke DSP. Typickým príkladom je čakanie na operand, ktorý bol zapísaný do pamäti v predchádzajúcej inštrukcii.

knižnice. Knižnice je potom možné využívať napr. ako aplikačné balíky (napr. knižnica pre číslicovú filtráciu, FFT a pod) a výrazne tak skrátiť dobu vývoja. Vývojové nástroje často obsahujú štandardné knižničné funkcie pre ČSS. Súčasťou prostredia VisualDSP++ sú napr. funkcie aj funkcie pre FIR, IIR a FFT z tzv. DSP Run-Time Library [9]:

```
void fir_fr16( const fract16 x[ ], fract16 y[ ], int n, fir_state_fr16 *s );
void iir_fr16( const fract16 x[ ], fract16 y[ ], int n, iir_state_fr16 *s );
void iirdf1_fr16( const fract16 x[ ],  fract16 y[ ], int n, iirdf1_fr16_state *s );
void cfft_fr16(const complex_fract16 in[ ], complex_fract16 t[], complex_fract16 out[ ],
                const complex_fract16 w[ ], int wst, int n, int block_exponent, int scale_method );
```

Časť funkcií z knižnice DSP Run-Time Library bude využitá v nasledujúcich cvičeniach resp. zadaniach. Podrobnejšie informácie o uvedených funkciách sú dostupné v on-line manuáloch prostredia VisualDSP++. Súčasťou prostredia sú aj optimalizované zdrojové kódy knižničných funkcií v asembleri.

Nasledujúci kód dokumentuje princíp využitia asemblera pri vytvorení vlastnej knižničnej C funkcie s prototypom

```
int a_dot_c_asm( int *a, int *c );
```

a telom funkcie optimalizovanom v asembleri:

```
/*****************************************************************************
 kniznicna funkcia realizuje nasledujucu C funkciu

 const int N = 20;

 int a_dot_c_asm( int *a, int *c ) {
   int i;
   int output = 0.0;
   for( i=0; i<N; i++ )  {
                output += ( a[i] * c[i] );
   }
   return( output );
}
*****************************************************************************/

// ASSEMBLY DOT PRODUCT FUNCTION
// File: dotprod_func.asm

.section my_asm_section;
.global _a_dot_c_asm;

_a_dot_c_asm:
        P0 = R0;                                        // adresa prvku a[0]
        I0 = R1;                                        // adresa prvku c[0]
        P1 = 19;                                        // N-1
        R0 = 0;                                         // output = 0
        NOP;
        R1 = [P0++];
        R2 = [I0++];
        LSETUP (begin_loop, end_loop) LC0 = P1;
begin_loop:     R1 *= R2;
                R2 = [I0++];
end_loop:       R0= R0 + R1 (NS) || R1 = [P0++] || NOP;         // NS -nesturovany vysledok
        R1 *= R2;
        R0 = R0 + R1;
        RTS;
_a_dot_c_asm.end:
```

Aj keď inštrukčná sada, direktívy[5] asembleru a programovací model jadra Balckfin zatiaľ neboli preberané, asemblerovský kód je pomerne čitateľný (detaily budú vysvetlené v rámci cvičenia). Zdrojové kódy reálnych knižničných funkcií sú zvyčajne zložitejšie, pretože okrem implementácie samotnej funkcie musia riešiť napr. kontrolu odovzdávaných parametrov, správne zarovnanie kritických sekcii kódu v pamäti a pod.

2.2.3 SIMULÁTORY

Simulátory sú programy, ktoré simulujú na cieľovej platforme (zvyčajne PC s operačným systémom Windows) prácu procesora na úrovni jednotlivých inštrukcií. Z pohľadu optimalizácie kódu pre DSP majú simulátory kľúčovú úlohu a umožňujú odhaliť problémy vznikajúce napr. vplyvom **zreťazenia**[6], prípadne zvýšiť efektivitu kódu identifikáciou inštrukcií, medzi ktorými sa vytvárajú tzv. „pipeline bubbles", t.j. okamihy, kedy zreťazené technické prostriedky nie sú plne využité. Vhodným preusporiadaním inštrukcií je často možné zvýšiť využitie prostriedkov procesora a tým aj rýchlosť implementovanej funkcie. V rámci cvičení bude využívaný simulátor, ktorý je súčasťou integrovaného prostredia VisualDSP++. Prostredie VisualDSP++ podporuje okrem klasickej simulácie aj tzv. **kompilovanú simuláciu** (Compiled Simulation). Klasická simulácia pracuje **interpretačným spôsobom**, kedy simulátor najskôr dekóduje a následne interpretuje každú simulovanú inštrukciu. Tento princíp simulácie je pre zložitejšie simulácie pomalý. Kompilovaná simulácia realizuje „preklad" simulovaného programu a nevyžaduje opakované dekódovanie inštrukcií. Výsledná simulácia je tak podstatne rýchlejšia.

2.2.4 VÝVOJOVÉ DOSKY, EMULÁTORY

Pri vývoji zariadení na báze DSP je cieľom realizovať systém ČSS, ktorý pracuje s reálnymi vstupnými resp. výstupnými dátami. V určitej etape vývoja je dôležité mať k dispozícii reálne technické zariadenie (často sa technické a programové prostriedky vyvíjajú paralelne a cieľové zariadenie nie je v úvodnej fáze vývoja k dispozícii). Na overenie činnosti algoritmov ČSS v reálnom čase sú vhodné **vývojové dosky**, ktoré typicky obsahujú všetky základné bloky číslicového signálového procesora (obmedzovacie a rekonštrukčné filtre, AD a DA prevodníky, DSP, pamäte) . Všetci hlavní výrobcovia DSP poskytujú lacné vývojové moduly (v cenách 70-300 $), ktoré je možné dokonca v prípade menej náročných zariadení využiť aj ako konečné technické riešenie systému ČSS. Výhodné je, pokiaľ cieľový DSP podporuje **emuláciu na čipe**[7], čo zlepšuje možnosti ladenia priamo v reálnych podmienkach pri minimálnych nárokoch na dodatočné technické vybavenie.

[5] Direktívy asembleru sú súčasťou zdrojových kódov. Na rozdiel od inštrukcií, ktoré sú príkazmi, ktoré realizuje po preklade do strojového kódu cieľový procesor, sú direktívy príkazmi pre program, ktorý realizuje samotný preklad. V predchádzajúcom príklade sú použité direktívy **.section**, **.global**, **LSETUP**.

[6] Zreťazenie (pipelining) je jednou zo všeobecných metód zvýšenia výkonnosti mikroprocesorov a je v oblasti DSP široko využívaná. Využitie zreťazenia bude opísané v rámci prednášok.

[7] Špeciálna metóda ladenia, ktorá využíva ladiace obvody implementované priamo v čipe procesora. Ladiace obvody moderných DSP umožňujú aj prenos ladiacich informácií v reálnom čase bez prerušenia činnosti DSP. Firma Analog Devices používa pre uvedený prenos termín **spätný telemetrický kanál** (Background Telemetric Channel).

Využitie klasických **emulátorov** je v praxi veľmi problematické predovšetkým vzhľadom na stále sa zvyšujúcu taktovaciu frekvenciu DSP a používanie prevažne SMD technológie, čo vylučuje nasadenie klasickej **emulačnej hlavice**.

2.3 VÝVOJOVÉ PROSTRIEDKY VYUŽÍVANÉ NA CVIČENIACH

V rámci cvičení budú využívané vývojové prostriedky pre signálové procesory na báze jadra Analog Devices Blackfin (ADSP2153x, ADSP21561) dodávané firmou Analog Devices. Tieto programy sú voľne dostupné na WWW stránkach firmy Analog Devices

www.analog.com/dsp/tools

vo forme 90-dňovej testovacej verzie, ktorú je možné zdarma aktivovať po získaní aktivačného kódu (položka **VisualDSP++ Test Drive Registration**). Počas cvičení budú využívané predovšetkým tieto programy:

1) asembler **easmblkfn.exe**
2) C/C++ prekladač **ccblkfn.exe**
3) linker **linker.exe**
4) Simulátor (dostupný z integrovaného prostredia podobne ako aj asembler, prekladač a linker)

Ďalšie programy ako knihovník, rôzne konverzné a pomocné programy, operačný systém reálneho času sú tiež dostupné v rámci prostredia VisualDSP++, počas cvičení však nebudú využívané[8]. Počas cvičení budú využívané optimalizované kódy pre procesory Balckfin, ktoré sú súčasťou prostredia VisualDSP++ resp. sú dostupné na WWW stránke firmy Analog Devices

http://www.analog.com/processors/blackfin/technicalLibrary/codeExamples/

Tieto kódy patria do kategórie voľne šíriteľných programov (freeware) a reprezentujú kódy pre algoritmy z rôznych oblastí ČSS.

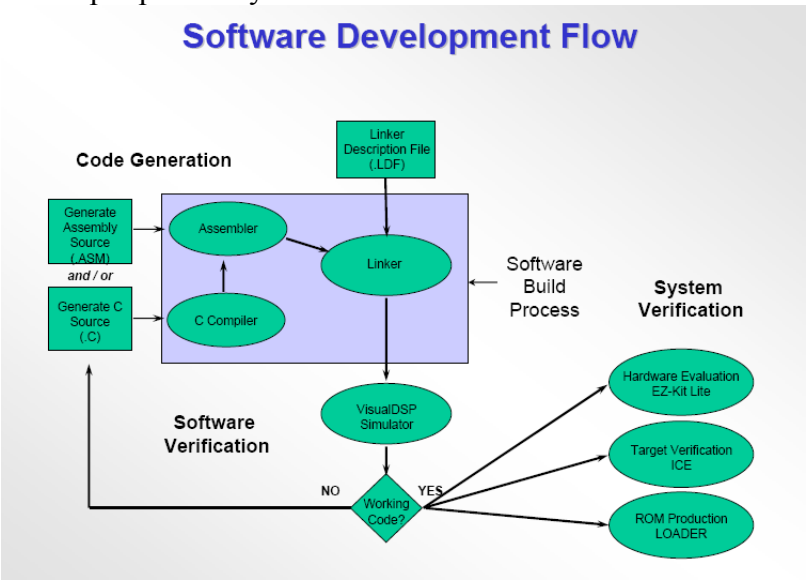
Veľká časť informácií je platná aj pre iné rodiny DSP firmy Analog Devices – ADSP21xx, ADSP SHARC a ADSP Tiger SHARC, pre ktoré je tiež možné využiť prostredie VisualDSP++.

Integrované vývojové prostredie VisualDSP od firmy Analog Devices využíva špecifické prípony pre generované výstupné súbory. Informácie o projekte sú uložené v súbore s príponou **.dpj**. Zdrojové kódy majú zvyčajne prípony **.asm**, **.h**, **.c**. Kód preložený do relatívneho objektového tvaru má príponu **.doj** a kód preložený do absolútneho objektového tvaru má príponu **.dxe**. Technické prostriedky pre ktoré je vytváraný výsledný program sú opísané v tzv. **Linker Description File** s príponou **.ldf**. Knihovník vytvára zlúčením viacerých súborov s príponami **.doj** knižnicu s príponou **.dlb**. Proces tvorby programového kódu pre signálové procesory Analog Devices, použité programové prostriedky a ich vzájomná väzba sú znázornené na Obr.1.

Tento obrázok reprezentuje možnosti a praktickú postupnosť využitia dostupných vývojových prostriedkov. Vykonateľný kód bude v počiatočných cvičeniach

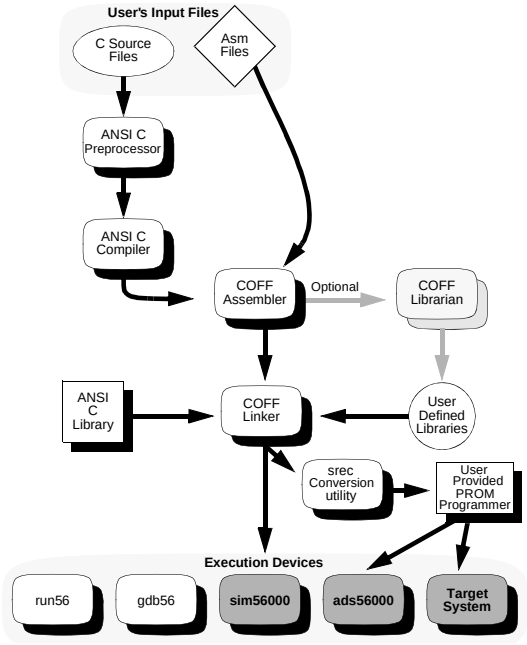
[8] Cieľom cvičení je vysvetliť základné vlastnosti DSP, čo budeme dokumentovať na relatívne jednoduchých príkladoch pri ktorých vystačíme s asemblerom, C prekladačom, linkerom a simulátorom. Programy ako napr. knihovník je výhodné využívať pri väčších projektoch (ako napr. diplomové úlohy), pri ktorých môžu výrazne zrýchliť a sprehľadniť generovanie výsledného kódu.

simulovaný pomocou simulátora a neskôr vykonávaný v cieľovom systéme – vývojovej doske EZ-KIT Lite pre procesory Balckfin.



Obr.1 Štruktúra programových prostriedkov pre DSP firmy Analog Devices

Na Obr.2 je pre porovnanie znázornený proces tvorby kódu pre klasické DSP firmy Motorola. Obrázok podrobnejšie naznačuje aj možné využitie **prekladača** (kompilátora) z vyššieho programovacieho jazyka C a **knihovníka** (Librarian) pre tvorbu užívateľských knižníc. Samozrejme tieto princípy sú využívané aj v prostredí VisualDSP++ resp. aj vo vývojových prostrediach pre jednočípové mikroprocesory.



Obr.2 Štruktúra programových prostriedkov pre DSP firmy Motorola

Príklad

*Precvičte ladenie priloženého C projektu **dotprod.zip** na výpočet skalárneho súčinu v prostredí VisualDSP++ zobrazenom na Obr.3.Precvičte základné ladiace kroky:*

- *preloženie programu (**menu Project**)*
- *krokovanie programu (**menu Debug**),*
- *možnosti zobrazenia rôznych registrov procesora (**menu Register**)*
- *zobrazenie C premenných (menu **View->Debug Windows-> Expressions, Locals**)*
- *krokovanie kódu v okne **Disassembly***
- *nastavenie bodov zastavenia (klik myšou na riadok C-kódu, **menu Settings->Breakpoints**)*
- *výpis funkcie printf( ) v okne **Output Window***

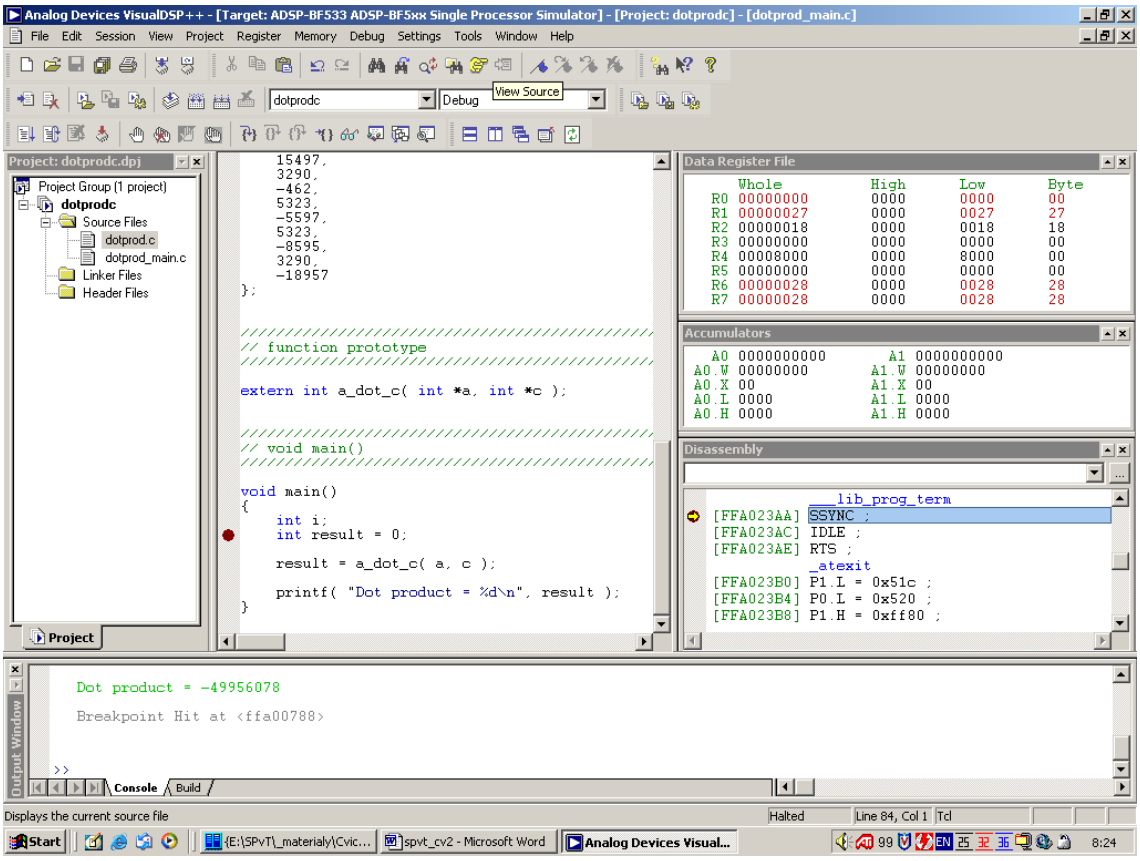
Pri ladení odpovedzte na nasledujúce otázky:

a) Koľko inštrukcií procesora je realizovaných v jednej iterácii funkcie a_dot_c( )? Porovnajte zistený počet s počtom inštrukcií v hlavnej slučke funkcie a_dot_c_asm( ) na str.4._

b) Čo sa stane po ukončení funkcie main( )?

c) Kde by bol v reálnej aplikácii smerovaný výstup funkcie printf ()?

d) Aký kód je vykonávaný procesorom pred spustením funkcie main? (processor začína vykonávať kód na adrese start: 0xFFA00000) _



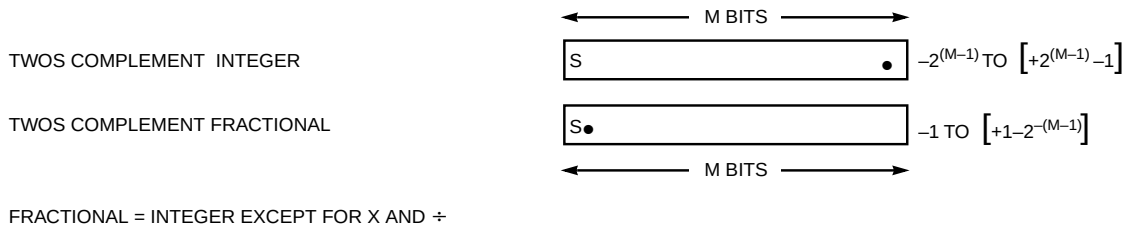
Obr.3 Ladenie príkladu v prostredí VisualDSP++

2.4 ZLOMKOVÝ FORMÁT ČÍSEL V DSP

Základnou úlohou pre ktorú sú DSP optimalizované je matematické spracovanie číslicových údajov. Medzi typické algoritmy pre ktoré sú komerčne dostupné DSP optimalizované patria číslicová filtrácia a algoritmus FFT. Základným dátovým typom DSP **s pevnou rádovou čiarkou** (medzi ktoré patria aj procesory Analog Devices s jadrom Balckfin, ako aj staršie procesory Motorola DSP56xxx [10] a Analog Devices ADSP21xx [11]) patrí **zlomkový formát** (fractional format, firma Analog Devices používa pre ich 16-bitové DSP tiež označenie **fractional representation 1.15**), pomocou ktorého sú čísla v intervale $\langle -1,1)$ reprezentované v tvare M bitov (**dĺžka slova**) $b_m \in \{0,1\}$, $m = 0,1,2,\ldots,M-1$, ktoré vyjadrujú číslo v tvare

$$x_{pevna} = (-1)b_0 + \sum_{m=1}^{M-1} b_m 2^{-m} \tag{2.1}$$

ktorý sa od klasického celočíselného formátu 16.0 (ktorý reprezentuje záporné čísla pomocou dvojkového doplnku) odlišuje polohou desatinnej čiarky, čo je dokumentované na nasledujúcom obrázku.



Obr.4 Porovnanie celočíselnej a zlomkovej reprezentácie

Formáty 1.15 a 16.0 sú formáty znamienkových čísel s desatinnou čiarkou maximálne posunutou vľavo resp. vpravo. DSP sú optimalizované pre zlomkový formát, principiálne však môžu využívať aj ďalšie formáty, ktoré sú pre $M = 16$ zobrazené na Obr.5.

Dĺžka slova v komerčne využívaných DSP s pevnou rádovou čiarkou je $M = 16$ a $M = 24$ bitov[9]. Základným dôvodom, prečo je využívaná zlomková reprezentácia je jej tesnejšia väzba s formátom **pohyblivej rádovej čiarky**, ktorý sa bežne využíva pri vedecko-technických výpočtoch (medzi ktoré patria aj algoritmy ČSS). Napríklad všetky bežne dostupné programy pre návrh číslicových filtrov poskytujú koeficienty v pohyblivej rádovej čiarke, čo je zvyčajne možné pretransformovať do zlomkového formátu jednoduchou **zmenou mierky** (t.j. predelením hodnôt mierkovou konštantou).

[9] 16-bitové DSP sa typicky využívajú v telekomunikačnej technike a 24-bitové DSP sú dominantné predovšetkým pre audio aplikácie. Procesory Motorola DSP5600x a DSP563xx sú 24-bitové DSP. Všetky DSP firmy Analog Devices, ktoré využívajú aritmetiku s pevnou rádovou čiarkou sú 16-bitové.

Ranges for 16 bit Formats

FORMAT		Largest Positive Value (0x7FFF) In Decimal	Largest Negative Value (0x8000) In Decimal	Value of 1 LSB (0x0001) In Decimal
1.15	Fractional	0.999969482421875	–1.0	0.000030517578125
2.14		1.999938964843750	–2.0	0.000061035156250
3.13		3.999877929687500	–4.0	0.000122070312500
4.12		7.999755859375000	–8.0	0.000244140625000
5.11		15.999511718750000	–16.0	0.000488281250000
6.10		31.999023437500000	–32.0	0.000976562500000
7.9		63.998046875000000	–64.0	0.001953125000000
8.8		127.996093750000000	–128.0	0.003906250000000
9.7		255.992187500000000	–256.0	0.007812500000000
10.6		511.984375000000000	–512.0	0.015625000000000
11.5		1023.968750000000000	–1024.0	0.031250000000000
12.4		2047.937500000000000	–2048.0	0.062500000000000
13.3		4095.875000000000000	–4096.0	0.125000000000000
14.2		8191.750000000000000	–8192.0	0.250000000000000
15.1		16383.500000000000000	–16384.0	0.500000000000000
16.0	Integer	32767.000000000000000	–32768.0	1.000000000000000

Obr.5 Číselné formáty pre znamienkové čísla a $M = 16$

Ďalšou výhodnou vlastnosťou zlomkového formátu je vlastnosť, že ak $x, y \in \langle -1,1)$, potom platí

$$z = x * y \in \langle -1,1) \tag{2.2}$$

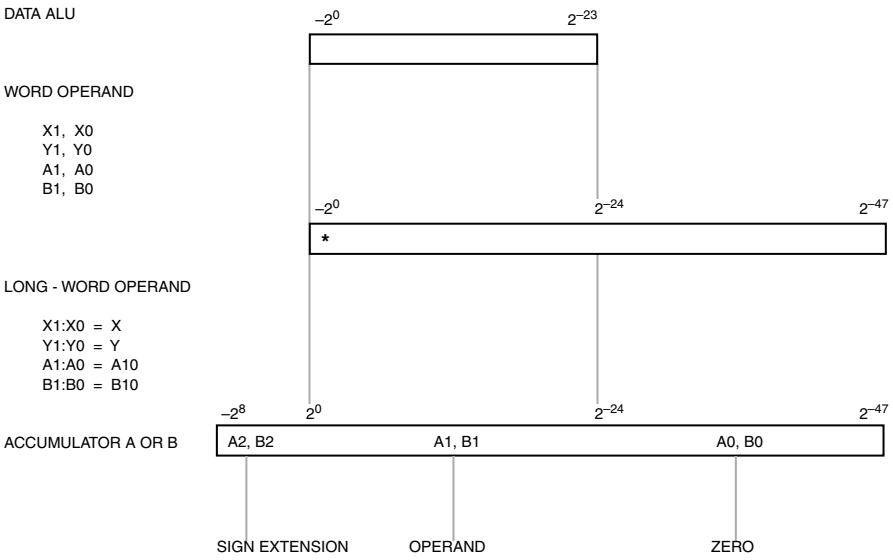
a teda pri operácií násobenia nedochádza k pretečeniu výsledkov mimo interval $\langle -1,1)$. K pretečeniu mimo zlomkový interval $\langle -1,1)$ dochádza len[10] pri operácií sčítania resp. odčítania.

V rámci štruktúry DSP (presnejšie **dátových registrov DSP**) typicky existuje niekoľko typov dátových registrov s rôznou presnosťou. V signálových procesoroch DSP56xxx ktoré využívajú **jednu MAC** jednotku napr. existujú 24-bitové registre (X0,X1,Y0,Y1) a 56-bitové registre (**akumulátory**) A, B, ktorých štruktúra a váhy jednotlivých bitov sú znázornené na nasledujúcom Obr.6.

Akumulátory A=(A2:A1:A0) a B=(B2:B1:B0) sa skladajú z 24-bitových subregistrov A0, A1, B0, B1 a 8-bitových registrov A2, B2. Akumulátory A a B sa využívajú predovšetkým pomocou inštrukcie MAC, ktorá realizuje výpočet

$$Akum = Akum \pm \mathrm{Re}\, g\_X * \mathrm{Re}\, g\_Y \tag{2.3}$$

[10] Operácia delenia je z pohľadu DSP špeciálna operácia, ktorá je podstatne pomalšia ako operácie +,-,*. DSP typicky poskytujú špeciálne inštrukcie, pomocou ktorých je možné realizovať delenie iteračným spôsobom, t.j. na dosiahnutie výsledku delenia je potrebných niekoľko inštrukcií. V procesoroch ADSP je napr. možné využiť inštrukcie DIVS (divide sign) a DIVQ (divide quotient).



Obr.6 Štruktúra registrov v procesoroch Motorola DSP56xxx

Štruktúra registrov v procesoroch Blackfin je podstatne bohatšia, čo je dané predovšetkým **väčším počtom MAC jednotiek** (MAC0 a MAC1), registrovým súborom s ôsmimi 32-bitovými registrami R0, R1, R2, R3, R4, R5, R6, R7, existenciou jednotky posúvača (Barrel Shifter) a podporou SIMD inštrukcií pre video operácie s 8-bitovými operandmi.

Vzhľadom na to, že procesory ADSP s aritmetikou v pevnej rádovej čiarke využívajú menší dynamický rozsah (16 resp. 40 bitov oproti 24 a 56 bitom u DSP5600x), obsahujú procesory ADSP podstatne lepšiu podporu pre aritmetiku v tzv. **blokovej pohyblivej rádovej čiarke** a v aritmetike s dvojnásobnou presnosťou.

Procesory Blackfin majú v jednotke MAC0 akumulátor A0=(A0.X:A0.H:A0.L) a v jednotke MAC1 akumulátor A1=(A1.X:A1.H:A1.L), ktoré majú 40 bitov. Akumulátory sú zložené zo 16-bitových subregistrov A0.H, A0.L, A1.H, A1.L a 8-bitových subregistrov A0.X, A1.X. Ich štruktúra je teda identická so štruktúrou akumulátorov A, B na Obr.6 pre procesory Motorola. Jediný rozdiel je v dĺžke subregistrov a im zodpovedajúcim binárnym váham.

Akumulátory A0, A1 sa v procesoroch Blackfin využívajú predovšetkým pomocou MAC inštrukcie, ktorá realizuje všeobecný výpočet

$$A = A + X * Y \tag{2.4}$$

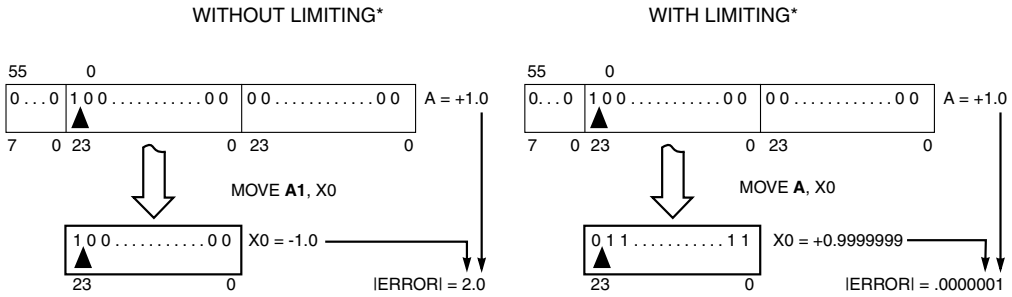
pričom vstupné registre X a Y sú 16-bitové subregistre z registrového súboru R0-7 (R0.L, R0.H, …R7.H) a výstupom A sú akumulátory A0, A1. Typické príklady zápisu:

```
A0 = R3.L * R4.H;                    // nasobenie bez akumulacie
A0 += R3.L * R4.H;                   // nasobenie s akumulaciou
A1 += R3.H * R4.H;
```

Prítomnosť dvoch paralelných MAC jednotiek v jadre procesorov Blackfin umožňuje paralelne vykonať dve MAC inštruckie (podrobnejšie preberané na prednáške) ako napr.:

```
R3.H = (A1 += R1.H * R2.L),  R3.L = (A0 += R1.L * R2.L);
R3 = (A1 += R1.H * R2.L),  R2 = (A0 += R1.L * R2.L);
```

S aritmetikou v procesoroch DSP sú spojené dva nové pojmy – **saturačná aritmetika (saturačný mód v ALU a MAC)** a **konvergentné zaokrúhľovanie (convergent rounding[11]).** Tieto vlastnosti sú demonštrované na Obr.7 a Obr.8 a podrobne vysvetlené počas cvičenia.



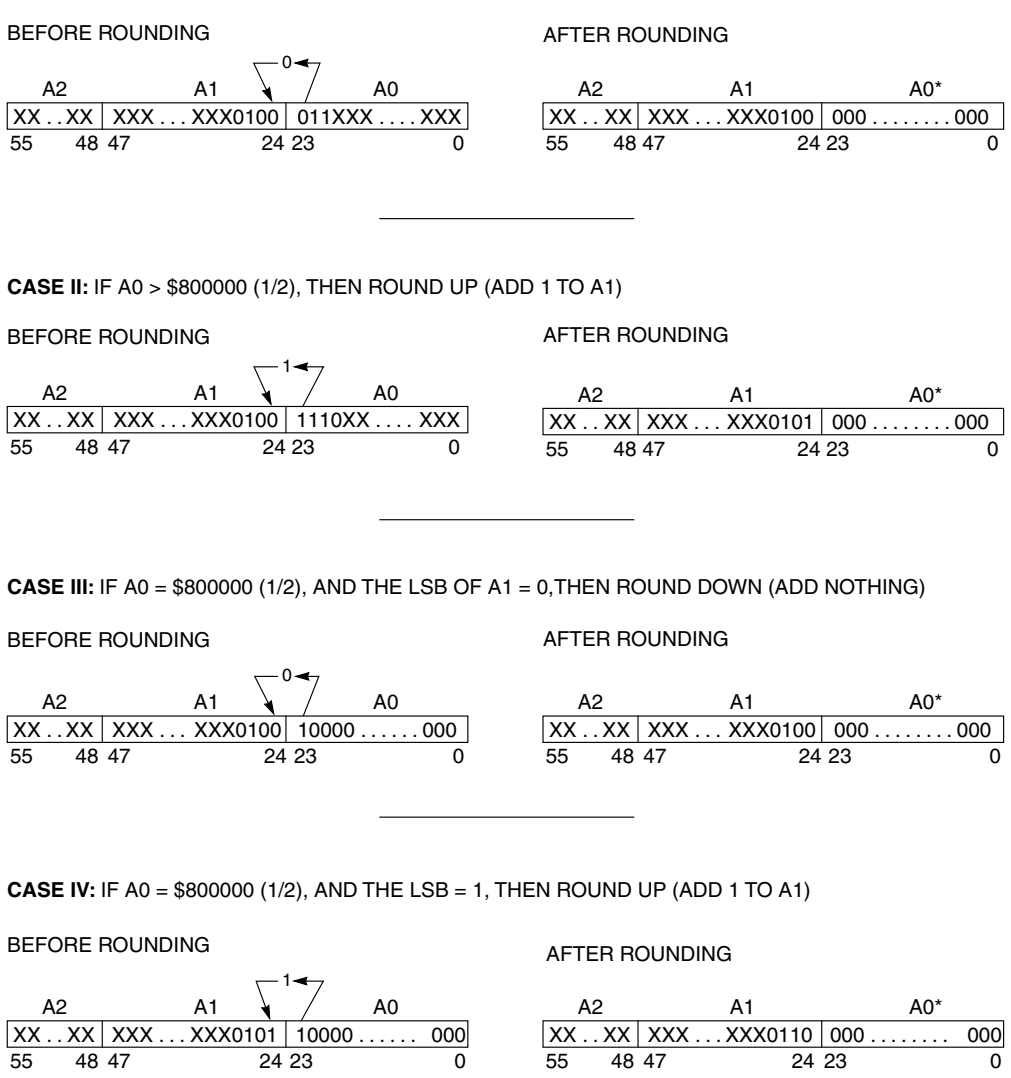
Obr.7 Princíp saturačnej aritmetiky v procesoroch Motorola (56-bitový akumulátor), zhodný princíp sa uplatňuje aj v prípade procesorov ADSP21xx a ADSP Blackfin (40-bitový akumulátor)

Využitie saturačnej aritmetiky v ALU jednotkách je možné v procesoroch Balckfin riadiť využitím modifikátorov **S** (saturate) a **NS** (non saturate)

```
R3 = R1 + R2 (NS);          // 32-bitovy vysledok nebude saturovany
R3 = R1 + R2 (S);           // 32-bitovy sucet bude saturovany
```

V MAC jednotkách je prevažná časť inštrukcií, ktoré pracujú so 40 bitovými akumulátormi A0, A1 automaticky saturovaná. U ostatných inštrukcií je saturácia závislá na type operácie [13].

[11] Novšie procesory vrátane procesorov Blackfin umožňujú aj využitie klasického zaokrúhľovania (tzv. **biased rounding**), ktoré sa využíva napr. pri implementácií kompresných rečových kodekov, ktoré musia poskytovať bitovo zhodný výstup s referenčným kodekom, ktorý je zvyčajne implementovaný v jazyku ANSI C s využitím celočíselnej aritmetiky.



Obr.8 Princíp konvergentného zaokrúhľovania v procesoroch Motorola (56-bitový akumulátor), zhodný princíp sa uplatňuje aj v prípade ADSP21xx (40-bitový akumulátor)

Zaokrúhľovací mód **MAC jednotiek** je v procesoroch Blackfin definovaný hodnotou bitu **RND_MOD** v registri **ASTAT**. Naviac Blakfin ALU poskytuje možnosť realizovať klasické zaokrúhlenie na konkrétnej bitovej pozícii bez ohľadu na nastavenie v registri ASTAT s využitím inštrukcií

```
R3.L = R4 (RND);          // klasicke zaokruhlenie na 16-tom bite
R3.L = R4 + R5 (RND12);   // 32-bitovy sucet a nasledne zaokruhlenie na 12-tom bite
R3.L = R4 + R5 (RND20);   // 32-bitovy sucet a nasledne zaokruhlenie na 20-tom bite
```

LITERATÚRA

[1] http://www.kemt.fei.tuke.sk/Predmety/KEMT412_SPvT/_materialy/Cvicenia/adsp2181/

[2] VisualDSP++, http://www.analog.com/processors/resources/crosscore/visualDspDevSoftware.html

[3] www.analog.com

[4] www.ti.com

[5] http://www.freescale.com

[6] El-Sharkawy, M.: *Real Time Digital Signal Processing Applications With Motorola's DSP56000 Family*. Prentice Hall, Englewood Cliffs, New Jersey, 1990.

[7] Mar, E. (editor): Digital Signal Processing Applications using the ADSP-2100 Family – Volume I. Prentice Hall, Englewood Cliffs, 1992 (dostupné aj v elektronickej forme).

[8] Babst, J. (editor): Digital Signal Processing Applications using the ADSP-2100 Family – Volume II. Prentice Hall, Englewood Cliffs, 1995 (dostupné aj v elektronickej forme).

[9] VisualDSP++ 4.0 C/C++ Compiler and Library Manual for Blackfin Processors. Analog Devices, Inc., Revision 3.0, January 2005.

[10] DSP56000 Digital Signal Processor Family Manual. DSP56KFAMUM/AD, Motorola, Inc., 1992.

[11] ADSP-2100 Family User's Manual, Analog Devices, Inc., 1995.

[12] ADSP-BF533 Blackfin Processor Hardware Reference (Rev.3.1). Analog Devices, Inc., May 2005.

[13] ADSP-BF53x/BF56x Blackfin Processor Programming Reference (Rev.1.1). Analog Devices, Inc., February 2006.