ZESZYTY NAUKOWE POLITECHNIKI ŚLĄSKIEJ 2000
Seria: ELEKTRONIKA z. 12 Nr kol. 1492

Jacek IZYDORCZYK
Politechnika Śląska, Instytut Elektroniki

PROCESORY SYGNAŁOWE
Część 1. Historia i teraźniejszość

Streszczenie. Artykuł poświęcony jest omówieniu architektury procesorów sygnałowych. Architektura ta zmienia się z biegiem czasu, a właściwie wraz z rozwojem technologii wytwarzania układów scalonych. W pierwszej części artykułu omówiono najstarszą z nadal używanych architektur, tzn. architekturę rodziny TMS320C1x/2x/2xx/5x opracowaną przez firmę Texas Instruments. Przedyskutowano także rozszerzenie architektury mikroprocesorów ogólnego przeznaczenia firmy Intel (IA32) o rozkazy przetwarzania sygnałów. Jest to tzw. rozszerzenie architektury MMX. Nie mamy tu wprawdzie do czynienia z klasycznym procesorem sygnałowym, ale zastosowane rozwiązanie ilustruje znakomicie zasadę przetwarzania SIMD. Idea procesora SIMD jest obecnie najtańszą metodą rozbudowy istniejącej architektury procesora sygnałowego o elementy przetwarzania równoległego.

DIGITAL SIGNAL PROCESSORS
Part 1. History and today

Summary. The article is about architecture of Digital Signal Processors. The architecture is evolving with time or more precisely with technology of integrated circuits fabrication. Part 1 of the article shows core architecture of TMS320C1x/2x/2xx/5x family. It is the oldest architecture, which is still in use by Texas Instruments. Architecture of MMX extension of Intel microprocessors is showed too. It is an illustration of a SIMD processor. SIMD is a cheap method to insert elements of parallel computing into existing architecture of DSP processor.

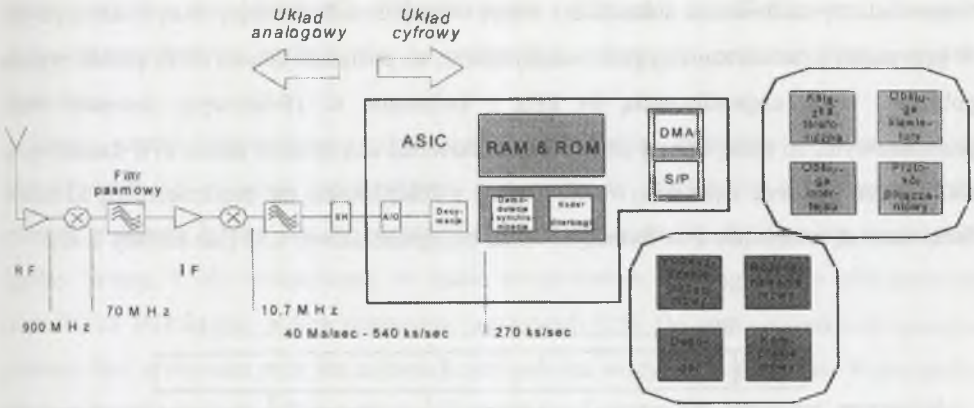
1. Procesory sygnałowe w telekomunikacji

Wartość obrotów na światowym rynku procesorów sygnałowych osiągnęła w roku 1997 ponad 3 miliardy USD. Co roku notuje się wzrost obrotów o 35%-40%, co oznacza prawie potrojenie wielkości rynku co trzy lata. Przewiduje się, że w roku 2002 wartość obrotów

sięgnie 14 miliardów USD [25]. Najciekawsze jest to, że wzrost ten wynika w dużej mierze z zapotrzebowania na procesory sygnałowe przez telekomunikację. W szczególności chodzi tutaj o cyfrową telefonię komórkową oraz transmisję danych cyfrowych poprzez publiczną sieć telefoniczną (modemy dla komputerów PC).

Telefonia komórkowa stanowi największy segment rynku procesorów sygnałowych. Żaden cyfrowy aparat telefoniczny nie może się obejść bez procesora sygnałowego — patrz rysunek 1. Dużą ich liczbę zawiera także każda stacja bazowa. Należy także uwzględnić zastosowania w klasycznych telefonach bezprzewodowych, aparatach telefonicznych z automatyczną sekretarką wyposażonych w pamięć półprzewodnikową oraz faksach. W krajach wysoko rozwiniętych dochodzi do tego wielce obiecujący rynek związany z wprowadzaniem przez licznych operatorów telekomunikacyjnych systemów radiodostępu abonenckiego (ang. wireless local loop). Cyfrowa telefonia komórkowa oprócz zwykłych usług polegających na przesyłaniu głosu pozwala na transmisję danych cyfrowych, w tym na dostęp do Internetu.

Internet to technologia, która w dużym stopniu pobudza popyt na rynku modemów dla komputerów PC. Ludzie używają i chcą używać Internetu w pracy, w domu dla rozrywki, w interesach. Mogą nawet za jego pośrednictwem wygodnie załatwić zakupy. Modem służy w tym przypadku do przesłania danych cyfrowych od abonenta do dostawcy usług internetowych (oraz w kierunku przeciwnym) poprzez publiczną sieć telefoniczną. Połączenie abonenta z publiczną siecią telekomunikacyjną ma jednak zwykle charakter analogowy i dane cyfrowe muszą być przed wysłaniem zabezpieczone, przetworzone i „wtłoczone" w kanał o paśmie przepuszczania około 3 kHz i przepustowości nie przekraczającej 64 kb/s. Podobnie dzieje się w przypadku cyfrowej telefonii komórkowej — dane cyfrowe (zakodowany sygnał mowy) muszą zostać przesłane drogą radiową, czyli kanałem analogowym, w którym warunki propagacji mogą zmieniać się z sekundy na sekundę. W obu przypadkach procesor sygnałowy realizuje bardzo podobne zadania:



Rys. 1. Schemat blokowy telefonu komórkowego GSM [25]
Fig. 1. Block diagram of a GSM telephone handset [25]

dokonuje korekcji zniekształceń wprowadzanych do odbieranego sygnału przez kanał transmisyjny (ang. equalization),

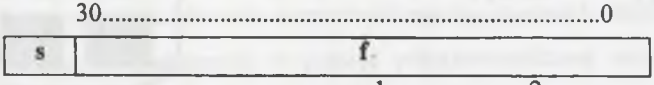
- dokonuje elektronicznej eliminacji echa (modem analogowy, modem ISDN),
- odtwarza częstotliwość nośną nadajnika (modem analogowy),
- odtwarza częstotliwość sygnalizacji nadajnika,
- dokonuje modulacji i demodulacji sygnału,
- realizuje kodowanie i dekodowanie kanałowe,
- dokonuje kompresji (dekompresji) cyfrowego sygnału mowy,
- dokonuje bezstratnej kompresji (dekompresji) danych cyfrowych (modem analogowy),
- eliminuje echo akustyczne (aparaty głośnomówiące).

Chociaż zadania te podsumowano tylko w kilku punktach, to nie są to zadania trywialne. Samo omówienie ich podstaw teoretycznych zajmuje zwykle prawie tysiącstronicowy podręcznik akademicki [27, 28, 29]. Wynikające stąd algorytmy cyfrowego przetwarzania sygnałów mają jednak pewną liczbę cech wspólnych, które przesądzają o takiej a nie innej budowie (architekturze) procesorów sygnałowych.

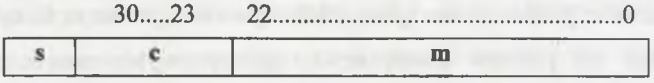
Sygnały cyfrowe reprezentowane są zwykle i przetwarzane ze stosunkowo niewielką dokładnością. Typowy kodek telefoniczny dokonuje próbkowania sygnału analogowego z częstotliwością 8 kHz, a następnie wartości próbek koduje w formacie A-law[1]. Jest to nic innego jak bardzo krótki, jednobajtowy zmiennoprzecinkowy format zapisu liczb [30] zapewniający zachowanie odstępu sygnał-szum kwantowania na poziomie 38-44 dB.

[1] W USA jest to format μ-law.

Odpowiada to zachowaniu dokładności mniej więcej dwóch znaczących cyfr dziesiętnych. W przypadku przetwarzania sygnału akustycznego na poziomie jakości hi-fi, próbki sygnału pobierane są z częstotliwością 44 kHz i kodowane w 16-bitowym formacie stałoprzecinkowym, co mniej więcej odpowiada zachowaniu dokładności pięciu cyfr dziesiętnych. Dalsze przetwarzanie liczb odbywa się zwykle z dokładnością nie przekraczającą 32 bitów. Stosowane są w zasadzie dwa formaty. Format stałoprzecinkowy 1.31 (lub krótszy 1.15):



Najstarszy bit słowa ma wagę -1, następny 2^{-1}, następny 2^{-2} aż do najmłodszego bitu o wadze 2^{-31}. Reprezentowana w ten sposób liczba wyraża się wzorem $x=(-1)^s\cdot(0.f)$ i leży w lewostronnie domkniętym przedziale [-1,1). Dokładność przeprowadzanych w tym formacie obliczeń sięga dziewięciu znaczących cyfr dziesiętnych, ale niewielki zakres reprezentowanych liczb jest pewnym utrudnieniem dla programisty. Procesory sygnałowe przetwarzające liczby w formacie stałoprzecinkowym nazywane są w żargonie stałoprzecinkowymi. Tak się składa, że właściwie wszystkie procesory prezentowane w tym artykule to procesory stałoprzecinkowe. Wynika to z faktu, że arytmetyka stałoprzecinkowa jest łatwiejsza do zaimplementowania w układzie scalonym (mniej bramek) i nowe idee dotyczące procesorów sygnałowych zawsze testowane były najpierw na procesorach stałoprzecinkowych. Drugi ze stosowanych formatów zapisu liczb to format zmiennoprzecinkowy. Jest to krótka, 32-bitowa forma zapisu zmiennoprzecinkowego przewidywanego przez normę IEEE-754 [26]:



Liczbę zmiennoprzecinkową tworzą dwie liczby stałoprzecinkowe: mantysa i cecha. Wartość liczby dana jest wzorem $x=(-1)^s\cdot(1.m)\cdot 2^{(c-127)}$. W ten sposób mogą być reprezentowane liczby z zakresu od 10^{-38} (mniej niż masa elektronu wyrażona w kilogramach) do 10^{38} (więcej niż odległość od najdalszych obserwowanych galaktyk wyrażona w metrach) z zachowaniem dokładności 7 dziesiętnych cyfr znaczących. Jest to dla cyfrowego przetwarzania sygnałów wystarczający zakres liczb oraz dokładność obliczeń. W związku z tym mamy istotną różnicę między zmiennoprzecinkowym procesorem sygnałowym a mikroprocesorem ogólnego przeznaczenia, np. Pentium. Mianowicie ten

ostatni liczy „za dokładnie". Wykonywanie obliczeń w 80-bitowym formacie zmiennoprzecinkowym wydaje się w przypadku cyfrowego przetwarzania sygnałów zbyteczne.

Druga cecha algorytmów cyfrowego przetwarzania sygnałów to konieczność wykonywania ich w czasie rzeczywistym. Algorytmy te składają się z ogromnej liczby operacji mnożenia liczb i ich dodawania. Na przykład ocenia się, że algorytm modulatora zgodny normą V.34, wykonywany w czasie rzeczywistym, wymaga mocy obliczeniowej rzędu 20-25 MIPS (ang. milion instruction per second) [25]. Do tego wszystkie te operacje powinny być wykonane przy jak najmniejszym poborze mocy przez procesor. W przypadku telefonu komórkowego czy pagera wymaganie to jest absolutnie nadrzędne. W przypadku stacji bazowej czy modemu komputerowego obniżenie pobieranej mocy to niższe koszty eksploatacji, a przede wszystkim większa niezawodność urządzenia. Wydaje się, że o ile zmniejszanie poboru energii na każdą wykonaną instrukcję to wyłączna domena technologii elektronowej, o tyle zwiększanie wydajności procesorów sygnałowych to problem ich architektury (budowy wewnętrznej).

2. W pogoni za wydajnością

Architektura procesorów sygnałowych w dużej mierze podporządkowana jest wymogowi bardzo szybkiego przetwarzania liczb. Podstawowym problemem jest konstrukcja i scalenie szybkiego układu mnożącego, który dostarcza wyniku mnożenia w czasie jednego, dwu lub najwyżej trzech taktów zegarowych [3]. Sztuka ta udała się dopiero na początku lat osiemdziesiątych. Od tego momentu można mówić o pojawieniu się procesorów sygnałowych. Następnym problemem było takie zorganizowanie przetwarzania, aby jak najlepiej wykorzystać zasoby sprzętowe procesora i aby żadna jego część nie pozostawała bezczynna podczas realizacji programu.

2.1. Architektura von Neumana

Na początku swej historii komputery pomyślne były jako systemy sterowane przepływem operacji [4]. Stanowisko realizacji (procesor):

- pobiera rozkaz z pamięci operacyjnej,
- dekoduje jego znaczenie,
- pobiera potrzebne do realizacji instrukcji (rozkazu) dane (argumenty),

- wykonuje operację wskazaną przez rozkaz,
- zapamiętuje wyniki.

W tym momencie procesor przechodzi do realizacji następnej instrukcji itd. Pierwsze mikroprocesory działały ściśle według tego schematu, ale szybko okazało się, że dla zwiększenia prędkości przetwarzania trzeba sięgnąć po udoskonalenia tej koncepcji, które wcześniej przetestowane zostały na dużych maszynach cyfrowych[2].

2.1.1. Przetwarzanie potokowe

Jednym z wcześniejszych pomysłów przyspieszenia realizacji rozkazów było zorganizowanie przetwarzania w postaci potoku instrukcji [1, 2]. Realizacja rozkazu podzielona była zawsze na pewną liczbę kolejnych etapów. Powyżej w punktach przedstawiono zaledwie jedną propozycję podziału procesu wykonywania rozkazu na etapy. W tym przypadku podziału dokonano na pięć etapów, ale ani sposób podziału ani liczba etapów nie jest z góry ustalona. Istnieją procesory które realizują instrukcje w czterech etapach, tak jak np. stałoprzecinkowy procesor sygnałowy firmy Analog Devices ADSP-2181 [31], i są takie, które potrzebują do realizacji pewnych rozkazów nawet szesnastu etapów przetwarzania, np. zmiennoprzecinkowy procesor sygnałowy TMS320C6701 firmy Texas Instruments [9]. Etapy realizowane są przez kolejne jednostki procesora połączone w kaskadę (potok), a wyniki przetwarzania w danym etapie są danymi wejściowymi dla przetwarzania w etapie następnym. W takim przypadku bezczynne czekanie układów procesora, które zakończyły już swoją pracę, na całkowite zakończenie wykonywania rozkazu, jest jawnym trwonieniem czasu i zasobów. Każdy z elementów procesora po zakończeniu pracy nad jednym rozkazem przechodzi natychmiast do przetwarzania następnego rozkazu. Powstaje w ten sposób coś na kształt linii montażu samochodów [2]. W każdej chwili procesor zajęty jest przetwarzaniem kilku rozkazów jednocześnie, przy czym każdy rozkaz jest na innym etapie wykonania. Efektywny czas wykonania rozkazu[3], niezależnie od liczby etapów potrzebnych na jego realizację, jest równy czasowi przetwarzania na jednym etapie realizacji. Idea ta sięga lat sześćdziesiątych. Pierwsze maszyny, w których ją zaimplementowano, to IBM 360/195, CDC STAR, MU5 (Uniwersytet w Manchester) [1].

---

[2] Według wiedzy autora ostatnim mikroprocesorem ogólnego przeznaczenia, który działał *ściśle* według przedstawionego schematu, był Z80 firmy Zilog.

[3] Oczywiście, po wypełnieniu się potoku instrukcji.

2.1.2. Architektura typu Harvard

Zwiększona szybkość przetwarzania danych powoduje na ogół liczne konflikty podczas dostępu do pamięci operacyjnej. Jeżeli maszyna korzysta ze wspólnej pamięci programu i danych, to w przypadku przetwarzania potokowego często dochodzi do próby jednoczesnego pobrania nowego rozkazu oraz pobrania argumentu dla rozkazu, który został już częściowo przetworzony. Takie konflikty są, oczywiście, w pewien sposób rozwiązywane przez układy procesora, ale za każdym razem prowadzi to do chwilowego zatrzymania potoku rozkazów. Najlepiej gdyby konfliktów takich nie było. Można ich w dużym stopniu uniknąć, stosując dwa oddzielne układy pamięci i dwie oddzielne magistrale łączące te pamięci z procesorem. Jeden układ pamięci przechowuje rozkazy (pamięć programu), a drugi przechowuje dane (pamięć danych). W ten sposób powstaje architektura typu Harvard. Jest to stary pomysł, bowiem od najdawniejszych czasów czas dostępu do pamięci operacyjnej był zbyt długi w stosunku do szybkości procesora centralnego. Architektura typu Harvard była tu jednym z pomysłów[4] powalającym na utrzymanie wysokiego tempa przetwarzania w obecności stosunkowo wolnej pamięci.

2.1.3. Procesory macierzowe (SIMD)

Wiele algorytmów, np. te dotyczące cyfrowego przetwarzania sygnałów, rozwiązywania równań różniczkowych cząstkowych, wykonuje wielokrotnie te same operacje na różnych zestawach danych. Powstaje zatem pomysł, aby wyposażyć maszynę cyfrową w wiele identycznych jednostek, z których każda wykonuje za każdym razem ten sam rozkaz, ale na innych danych. Procesory o takiej architekturze nazywane są procesorami macierzowymi lub procesorami wektorowymi. W terminologii angielskiej są to procesory SIMD — single instruction multiple data. Ponieważ już w czasie II wojny światowej używano maszyn cyfrowych do rozwiązywania równań różniczkowych cząstkowych, procesory macierzowe pojawiły się bardzo wcześnie. Jednym z pierwszych przykładów była maszyna Illiac IV, skonstruowana na Uniwersytecie Illinois, a zbudowana w zakładach Burroughs Corporation.

[4] Obok np. pracy pamięci z przeplotem [2].

2.2. Przetwarzanie równoległe

Dalsze zwiększenie prędkości przetwarzania danych przez procesor związanie jest z możliwością równoległego przetwarzania danych. Dokładne rozważenie problemu prowadzi do koncepcji systemów sterowanych przepływem argumentów [4, 5]. Idea polega na tym, że rozkazy tworzące algorytm wykonywane są natychmiast wtedy, gdy dostępne są argumenty potrzebne do ich realizacji. Przez argumenty rozumiemy tutaj niezbędne dane oraz układy procesora, które są w stanie zrealizować wskazaną przez rozkaz operację. Prowadzi to do sytuacji, w której procesor centralny wyposażony jest w wiele jednostek zdolnych do wykonywania rozkazów równolegle. Rozkazy są pobierane „szerokim strumieniem" i kierowane do realizacji przez poszczególne jednostki. Koncepcja takiego procesora równoległego może być zrealizowana przynajmniej na dwa sposoby.

2.2.1. Procesory o bardzo długim rozkazie (VLIW)

Rozkazy procesora o architekturze VLIW w sposób jawny zawierają informację, które z nich będą przetwarzane jednocześnie. Argumentem (jawnym lub domyślnym) każdego takiego rozkazu jest jednostka wykonawcza procesora, która będzie rozkaz realizować. W jednym cyklu pobiera się wiele rozkazów jednocześnie, stąd angielska nazwa architektury VLIW – very long instruction word. Algorytm z postaci szeregowej, którą programiście łatwiej stworzyć i przeanalizować, do postaci równoległej musi być zatem przetworzony na etapie kompilacji programu. W praktyce wymusza to korzystanie przy pisaniu programów z kompilatora języka C/C++ lub innego języka wysokiego poziomu. Przykładem procesorów o takiej architekturze są procesory sygnałowe z rodziny TMS320C6xx. Architekturę typu VLIW lub jej elementy będzie miała także nowa rodzina procesorów ogólnego przeznaczenia firmy Intel[5]. Jest rzeczą ciekawą, że podobną koncepcję zrealizowano już dawno w maszynach typu CDC6600 i Cyber 74. W tym przypadku rozkazy pobierane były kolejno, ale efekt równoległego przetwarzania uzyskiwano dzięki temu, że czas pobrania i dekodowania rozkazu był pomijalnie mały w stosunku do czasu realizacji rozkazu przez jedną z wielu jednostek wykonawczych [1].

[5] Firma Intel nazwała architekturę nowych procesorów angielskim akronimem EPIC od słów Explicitly Parallel Instruction Computing [12].

2.2.2. Procesory superskalarne

W przypadku procesorów superskalarnych programista nie jest do końca świadom istnienia wielu jednostek wykonawczych. Program pisany jest tak, jak dla maszyny, która przetwarza rozkazy szeregowo, ewentualnie z wykorzystaniem przetwarzania potokowego. Procesor rozdziela w locie strumień rozkazów do poszczególnych jednostek wykonawczych dbając o to, aby nie dochodziło do konfliktu argumentów i aby wynik działania całości układu był dokładnie taki sam, jak w przypadku przetwarzania rozkazów szeregowo. Architekturę superskalarną mają wszystkie współczesne procesory ogólnego przeznaczenia, tzn. Pentium, PowerPC, Alpha itd. Długi czas powstrzymywano się przed stosowaniem architektury superskalarnej w przypadku procesorów sygnałowych. W roku 1998 pojawił się jednak pierwszy, superskalarny, stałoprzecinkowy procesor sygnałowy ZSP16401 [32].

3. Żywa skamielina — TMS320C5x

Procesory TMS320C5x (tablica 1) reprezentują w miarę współczesną gałąź dużej rodziny stałoprzecinkowych procesorów sygnałowych firmy Texas Instruments. Rodzina ta jest z pewnością najstarszą linią procesorów sygnałowych, która nadal jest oferowana na rynku. Najstarszym reprezentantem tej rodziny jest procesor TMS32010 wprowadzony na rynek przez firmę Texas Instruments w roku 1982 [6]. Jeszcze w tym samy roku magazyn „Electronic Products" przyznał układowi TMS32010 tytuł „Product of the Year" [8]. Był to początek olbrzymiego sukcesu rynkowego firmy Texas Instruments. Sukces ten spotęgowało wykorzystanie procesora TMS320C17[6] w jednej z pierwszych (a z pewnością najbardziej agresywnie reklamowanej) zabawek rozpoznających ludzki głos. Była to słynna lalka „Julie Doll". Procesor TMS32010 taktowany zegarem 20 MHz pozwalał na wykonywanie do 5 milionów instrukcji w ciągu sekundy (5 MIPS). Wkrótce pojawiły się następne układy wykonane w bardziej energooszczędnej technologii CMOS. W latach 1984-1995 firma Texas Instruments wprowadziła na rynek kolejne wersje stałoprzecinkowych procesorów sygnałowych o udoskonalonej architekturze. Były to TMS320C2x (1984), następnie TMS320C5x (1989), TMS320C2xx (1995) i TMS320C54x (1995). Wszystkie procesory (z wyjątkiem 54x) zachowują kompatybilność „w górę" na poziomie kodu źródłowego. Oznacza

[6] Jedna z mutacji procesora TMS32010. Litera C pojawiająca się w nazwie procesora wskazuje, że został wykonany w technologii CMOS, znacznie bardzie energooszczędnej niż technologia bipolarna wykorzystywana dla produkcji oryginalnego układu TMS32010.

to, że program napisany w asemblerze dla procesora, TMS320C10 może być asemblowany bez zmian w celu wykonywania go na procesorze TMS320C50. Stwierdzenie to pozostaje prawdziwe za każdym razem, gdy numer procesora, dla którego program był pierwotnie pisany, jest mniejszy od numeru procesora, który będzie program wykonywał [7]. Oznacza to, że trzon architektury tych procesorów pozostał nie zmieniony od początku lat osiemdziesiątych. Uwzględniając czas potrzebny wówczas na opracowanie tak skomplikowanego układu scalonego, należy przyjąć, że architektura rodziny swymi korzeniami sięga połowy lat siedemdziesiątych. Można zatem powiedzieć, że procesory te stanowią prawdziwą żywą skamielinę. Przetrwały do dzisiaj niewiele zmieniając się od czasów, gdy głównym problemem technologicznym było umieszczenie w strukturze układu scalonego tablicowego układu mnożącego 16-bitowe, stałoprzecinkowe liczby w ciągu jednego cyklu zegarowego. O tym, że był to poważny problem, przekonujemy się sprawdzając liczbę cykli zegarowych niezbędnych dla przemnożenia dwóch 16-bitowych, stałoprzecinkowych liczb przez procesor Intel 80286. Układ ten wprowadzono na rynek w tym samym roku, co procesor TMS32010. Intel 80286 potrzebował na wykonanie instrukcji IMUL[7] 24 cykle zegarowe, co trwało 4 μs przy taktowaniu procesora zegarem 6 MHz, podczas gdy TMS32010 zadowalał się czterema cyklami, co trwało 200 ns dla zegara 20 MHz. Rzeczywiście, architektura układu TMS32010 zorganizowana jest wokół układu mnożącego i jednostki arytmetyczno-logicznej z akumulatorem, które zajmują dużą część chipa.

[7] Mnożenie dwóch 16-bitowych liczb ze znakiem.

Tablica 1

Rodzina procesorów TMS320C5x

Układ TMS320	ID	Wielkość pamięci wbudowanej			Port		Zasilanie [V]	Okres zegara [ns]	Obudowa
		DARAM[A]	SARAM[B]	ROM	Szeregowy	Równoległy			
'C50	PQ	1056	9K	2K	2	64K	5	50/35/25	132 pin BQFP
'LC50	PQ	1056	9K	2K	2	64K	3.3	50/40/25	132 pin BQFP
'C51	PQ	1056	1K	8K	2	64K	5	50/35/25/20	132 pin BQFP
'C51	PZ	1056	1K	8K	2	64K	5	50/35/25/20	100 pin TQFP
'LC51	PQ	1056	1K	8K	2	64K	3.3	50/40/25	132 pin BQFP
'LC51	PZ	1056	1K	8K	2	64K	3.3	50/40/25	100 pin TQFP
'C52	PJ	1056	—	4K	1	64K	5	50/35/25/20	100 pin QFP
'C52	PZ	1056	—	4K	1	64K	5	50/35/25/20	100 pin TQFP
'LC52	PJ	1056	—	4K	1	64K	3.3	50/40/25	100 pin QFP
'LC52	PZ	1056	—	4K	1	64K	3.3	50/40/25	100 pin TQFP
'C53	PQ	1056	3K	16K	2	64K	5	50/35/25	132 pin BQFP
'C53S	PZ	1056	3K	16K	2	64K	5	50/35/25	100 pin TQFP
'LC53	PQ	1056	3K	16K	2	64K	3.3	50/40/25	132 pin BQFP
'LC53S	PZ	1056	3K	16K	2	64K	3.3	50/40/25	100 pin TQFP
'LC56	PZ	1056	6K	32K	2	64K	3.3	50/35/25	100 pin TQFP
'C57S	PGE	1056	6K	2K	2	64K[D]	5	50/35/25	144 pin TQFP
'LC57	PBK	1056	6K	32K	2	64K[D]	3.3	50/35/25	128 pin TQFP
'LC57S	PGE	1056	6K	2K	2	64K[D]	3.3	50/35	144 pin TQFP

[A] DARAM — pamięć o dwóch portach; [B] SARAM – pamięć o jednym porcie

3.1. Układ mnożący i jednostka arytmetyczna z akumulatorem

Rys. 2 pokazuje uproszczony schemat blokowy jednostki obliczeniowej procesora TMS320C50. Centralnym elementem jest tutaj układ mnożący pozwalający na wykonanie mnożenia dwóch 16-bitowych liczb. Jeden z argumentów operacji mnożenia znajduje się zawsze w rejestrze TREG0, natomiast drugi argument może zostać pobrany z przestrzeni adresowej danych albo z przestrzeni adresowej programu. Oba argumenty interpretowane są jako liczby całkowite w zapisie dopełnienie do 2. Wynik mnożenia umieszczany jest w 32-bitowym rejestrze PREG. Stąd poprzez układ przesuwnika wynik może zostać przekazany do jednostki arytmetyczno-logicznej (ALU). Przesuwnik umieszczony między rejestrem PREG i ALU pozwala na przesunięcie wyniku mnożenia podczas przesyłania go do ALU. Zachodzą przy tym cztery następujące możliwości [8]:

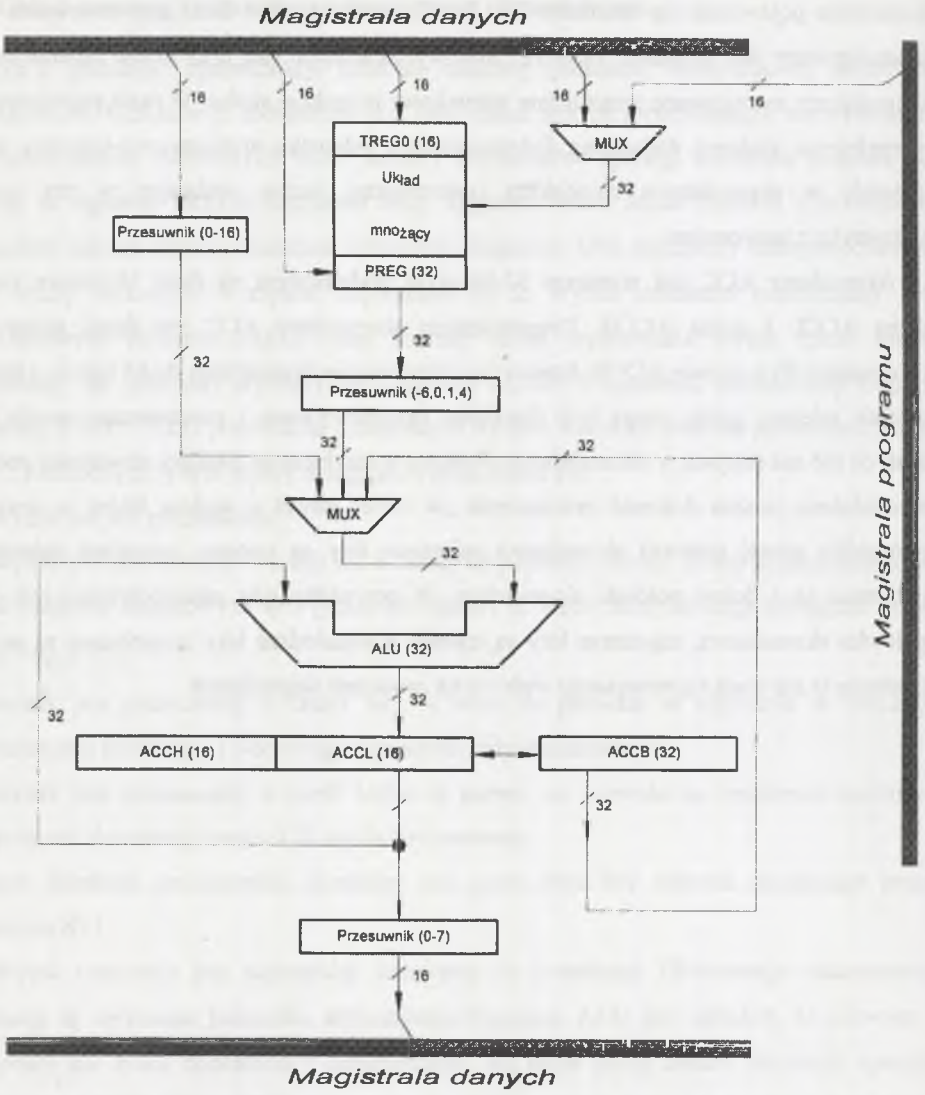
- wynik nie jest przesuwany,
- wynik jest przesuwany o jeden bit w lewo, co pozwala mnożyć liczby stałoprzecinkowe w formacie ułamkowym 1.15 (jeden bit znaku i 15 bitów, których waga zmniejsza się od 2^{-1} do 2^{-15}),
- wynik jest przesuwany o cztery bity w lewo, co pozwala na używanie w instrukcji mnożenia krótkiego, 13-bitowego argumentu natychmiastowego,
- wynik jest przesuwany o sześć bitów w prawo, co pozwala na uniknięcie nadmiaru podczas akumulacji nawet 128 wyników mnożenia.

Sposób działania przesuwnika określony jest przez dwa bity rejestru sterującego pracą procesora ST1.

Wynik mnożenia jest najczęściej dodawany do zawartości 32-bitowego akumulatora. Operację tę wykonuje jednostka arytmetyczno-logiczna. ALU jest układem 32-bitowym i wykonuje nie tylko dodawanie i odejmowanie, ale także pełny zestaw bitowych operacji logicznych (AND, OR i XOR) oraz operacje porównywania liczb i testowania wybranych bitów dowolnego słowa pamięci danych (testowanie stanu flagi). Domyślnym argumentem każdej operacji realizowanej przez ALU jest akumulator ACC. Drugi argument, dla operacji dwuargumentowych, poprzez przesuwnik pobierany jest z rejestru PREG albo z pamięci danych. Argument pochodzący z pamięci danych może być co najwyżej liczbą 16-bitową. Stąd przed dostarczeniem do ALU, argument ten może być przesunięty nawet o 16 bitów tak, aby mógł się znaleźć w dowolnym miejscu 32-bitowego portu wejściowego ALU. Jednostka arytmetyczno-logiczna podczas wykonywania dodawania (odejmowania) zapamiętuje w rejestrze sterującym ST1 bit przeniesienia z najstarszej pozycji akumulatora C oraz

sygnalizuje pojawienie się nadmiaru OV. Wynik porównywania liczb oraz testowania flagi zapamiętywany jest w postaci flagi TC. Stan wymienionych flag oraz liczba zapamiętana w akumulatorze są testowane przez liczne warunkowe instrukcje skoku. W razie pojawienia się przepełnienia podczas dodawania (odejmowania), jednostka arytmetyczno-logiczna może umieścić w akumulatorze największą (najmniejszą) liczbę, realizując w ten sposób arytmetykę z nasyceniem.

Akumulator ACC jest rejestrem 32-bitowym podzielonym na dwie 16-bitowe części: dolną ACCL i górną ACCH. Uzupełnieniem akumulatora ACC jest drugi akumulator (akumulator B) o nazwie ACCB. Stanowi on rozszerzenie akumulatora do 64 bitów, a równocześnie miejsce, gdzie mogą być chwilowo przechowywane i przetwarzane wynik,i dla których nie ma miejsca w akumulatorze. Podczas przesyłania do pamięci zawartości połówki akumulatora, można dokonać przesunięcia „w locie" nawet o siedem bitów w lewo. W przypadku górnej połówki akumulatora najstarsze bity są tracone, natomiast najmłodsze pobierane są z dolnej połówki akumulatora. W przypadku gdy zapamiętywana jest dolna połówka akumulatora, najstarsze bity są tracone a najmłodsze bity uzupełniane są zerami. Operacje te nie mają najmniejszego wpływu na zawartość akumulatora.



Rys.2. Schemat blokowy jednostki centralnej procesora TMS320C50 [8]
Fig.2. Block diagram of TMS320C50 CPU [8]

3.2. Pamięć

Program oraz dane umieszczone są w osobnych przestrzeniach adresowych. Każda z nich obejmuje 64K słowa. Słowo maszynowe zawiera szesnaście bitów. Wewnątrz układu scalonego funkcjonują dwie osobne magistrale: magistrala danych i magistrala programu. Na zewnątrz układu obie magistrale danych (danych i programu) oraz obie magistrale adresowe

(adres danych i adres programu) są multipleksowane. Na chipie razem z procesorem umieszczona jest zawsze pewna ilość pamięci, czy to (EP)ROM, czy RAM, którą można umieścić w dowolnie wybranej przestrzeni adresowej — patrz tablica 1. Jeżeli system uzupełnimy o zewnętrzną pamięć RAM, możemy zachować zalety architektury typu Harvard mimo ograniczonej liczby wyprowadzeń zewnętrznych układu scalonego realizującego układ. Oznacza to, że mamy możliwość jednoczesnego sięgania do pamięci programu, jak i do pamięci danych.

Pierwsze 128 komórek pamięci danych zarezerwowane jest dla realizacji rejestrów procesora. W przypadku modeli z rodziny TMS320C5x praktycznie wszystkie rejestry (jest ich razem 28) z wyjątkiem akumulatorów ACC i ACCB są zrealizowane w postaci komórek pamięci o adresie mniejszym od 128. Tego typu oszczędnościowe pomysły funkcjonują w świecie komputerów już od bardzo dawna. Jednym z wcześniejszych i bardziej znanych przykładów były najtańsze konfiguracje maszyn IBM 360.

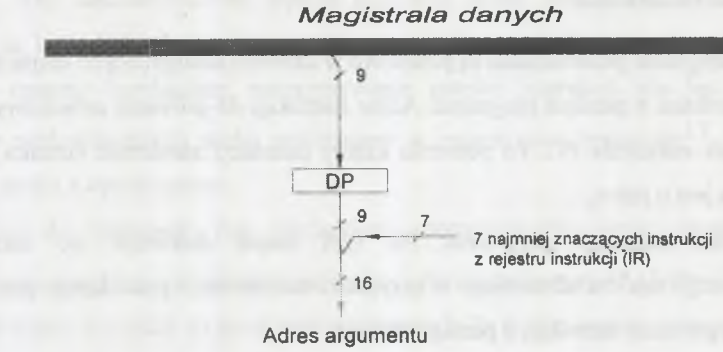
Programista dysponuje kilkoma trybami adresowania pamięci. Są to:

- adresowanie natychmiastowe z krótkim argumentem. Ośmiobitowy argument instrukcji stanowi część samej instrukcji. Przykładem jest instrukcja dodawania ADD #0FFh (dodaj do akumulatora liczbę 255) kodowana w postaci jednego słowa #B8FFh. Argument instrukcji został podkreślony,
- adresowanie natychmiastowe z długim operandem. Szesnastobitowy argument instrukcji stanowi drugie słowo instrukcji. Przykładem jest instrukcja dodawania ADD #01234h (dodaj do akumulatora liczbę 4660) kodowana w postaci dwóch słów #BF90h #1234h. Argument instrukcji został podkreślony,
- adresowanie bezpośrednie. Adres szesnastobitowego argumentu instrukcji powstaje w wyniku złożenia dwóch części. Pierwsza — bardziej znacząca — część adresu pobierana jest z rejestru wskaźnika strony DP (ang. data pointer). Dziewięciobitowy rejestr DP jest częścią rejestru sterującego pracą procesora ST0. Mniej znacząca część adresu argumentu to siedem najmniej znaczących bitów słowa instrukcji. W ten sposób bez zmiany zawartości rejestru DP można zaadresować zaledwie 128 komórek pamięci. Sposób tworzenia adresu w trybie adresowania bezpośredniego ilustruje rys.3a,

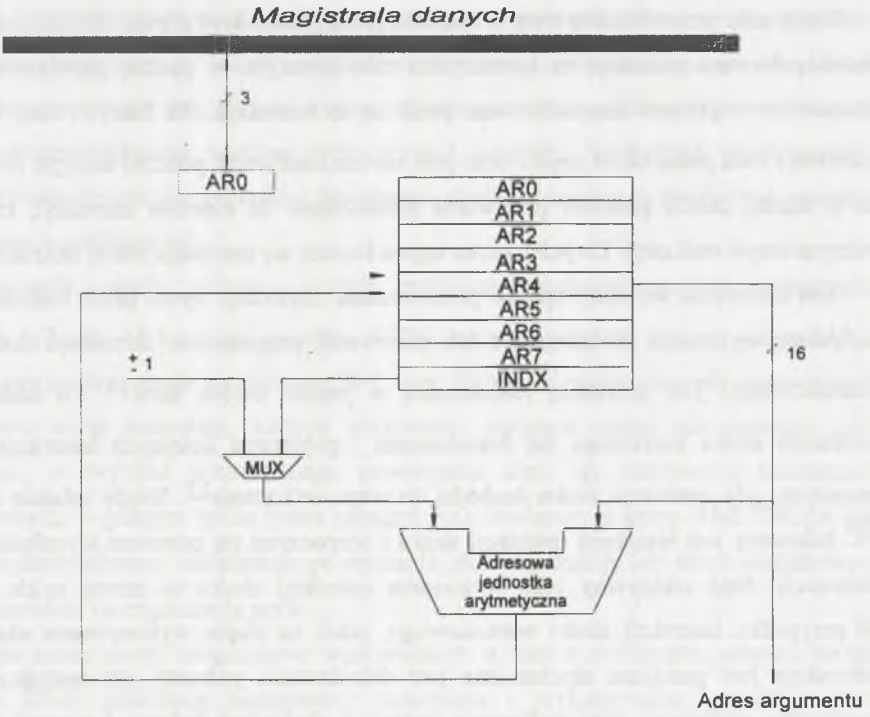
- adresowanie pośrednie przez rejestr. Źródłem adresu argumentu jest jeden z ośmiu 16-bitowych rejestrów adresowych AR0-AR7[8]. Do adresowania używany jest rejestr wskazywany przez 3-bitowy wskaźnik ARP (ang. address register pointer) będący częścią rejestru sterującego pracą procesora ST0. Ilustruje to rys.3b. Każda instrukcja wykorzystująca adresowanie pośrednie pozwala na:
  - dokonanie inkrementacji (dekrementacji) używanego rejestru adresowego; inkrementacja (dekrementacja) odbywa się zawsze po pobraniu argumentu; zawartość rejestru adresowego może zostać zwiększona o 1 (inkrementacja) albo o wartość zapisaną w rejestrze indeksowym INDX;
  - wskazanie innego rejestru adresowego (zmiana wartości wskaźnika ARP).

Mechanizm adresowania pośredniego pozwala na zorganizowanie w pamięci buforów cyklicznych i adresowania z odwróceniem kolejności bitów, bardzo pomocnego podczas kodowania algorytmu szybkiej transformacji Fouriera FFT. W procesorze TMS320C5x zaimplementowano ponadto inne mechanizmy adresowania. Nie są one jednak dostępne dla każdej instrukcji pobierającej argument z pamięci — korzystają z nich tylko wybrane instrukcje (ang. dedicated register addressing mode).

[8] Procesory z rodziny TMS320C1x zawierają tylko dwa takie rejestry.



A) Mechanizm adresowania bezpośredniego
A) Direct addressing



B) Mechanizm adresowania pośredniego przez rejestr
B) Indirect addressing

Rys. 3.Tryby adresowania dostępne w procesorze TMS320C50
Fig. 3. Memory addressing by TMS320C50 CPU

3.3. Sterowanie procesorem

Instrukcje programu przetwarzane są potokowo w czterech następujących etapach:

- pobranie rozkazu z pamięci programu. Adres instrukcji do pobrania przechowywany jest przez licznik rozkazów PC. Po pobraniu każdej instrukcji zawartość licznika rozkazów zwiększana jest o jeden,
- dekodowanie instrukcji programu. Na tym etapie dokonuje się inkrementacji (dekrementacji) rejestru adresowego w przypadku adresowania pośredniego przez rejestr,
- pobranie argumentu instrukcji z pamięci danych,
- wykonanie instrukcji. Na tym etapie dokonuje się zapisania wyniku działania instrukcji w pamięci.

Każdy etap przetwarzania trwa w zasadzie przez jeden okres zegara taktującego procesor. Jeżeli pobierane instrukcje są instrukcjami zakodowanymi w postaci pojedynczego słowa (znakomita większość instrukcji) oraz jeżeli są to instrukcje, dla których etap wykonania (czwarty) trwa jeden okres zegara oraz jeśli nie ma konfliktów podczas dostępu do pamięci[9], to w każdej chwili procesor przetwarza jednocześnie do czterech instrukcji, które są na różnym etapie realizacji. Co jeden okres zegara kończy się realizacja jednej instrukcji.

Ten niezwykle wydajny sposób przetwarzania zakłócany bywa przez instrukcję skoku, instrukcję wywołania podprogramu lub przerwanie programowe. Instrukcja skoku (skoku warunkowego) jest instrukcją zakodowaną w postaci dwóch słów[10]. Po zdekodowaniu instrukcji skoku zaprzestaje się dekodowania i pobierania kolejnych instrukcji[11], aż do momentu, gdy instrukcja skoku dochodzi do etapu wykonania[12]. Wtedy właśnie do rejestru PC ładowany jest argument instrukcji skoku i rozpoczyna się ponowne wypełnianie potoku instrukcji. Stąd efektywny czas wykonania instrukcji skoku to cztery cykle zegarowe. W przypadku instrukcji skoku warunkowego, jeżeli na etapie wykonywania okaże się, że instrukcja jest pomijana uruchamiane jest dekodowanie pobranej już instrukcji. W tym przypadku efektywny czas wykonania instrukcji skoku jest krótszy i wynosi dwa cykle

---

9 Na przykład w przypadku procesora TMS320C50 9K słów pamięci RAM dostępnych na chipie można skonfigurować tak, aby była dostępna jednocześnie w przestrzeni adresowej danych i programu. Pamięć ta posiada jednak tylko jeden port wejścia-wyjścia. W przypadku gdy program i przetwarzane dane umieszczone są w tej właśnie pamięci, dochodzi do konfliktu pomiędzy jednostką procesora pobierającą kolejną instrukcję a jednostką pobierającą argument innej instrukcji.

10 Drugie słowo to adres w pamięci programu, do którego zostanie przeniesione sterowanie.

11 W tym momencie pobrana jest już instrukcja następująca po instrukcji skoku.

zegarowe. Aby zminimalizować wpływ instrukcji skoku na efektywność przetwarzania, dostępne są instrukcje skoku (skoku warunkowego) z opóźnieniem. W przypadku takiej instrukcji opisany mechanizm wstrzymywania potoku instrukcji nie jest uruchamiany. W efekcie oprócz instrukcji skoku realizowane są jeszcze dwie instrukcje[13], następujące po instrukcji skoku z opóźnieniem.

Podobny do opisanego jest mechanizm wstrzymywania potoku instrukcji podczas wykonywania instrukcji (warunkowego) wywołania podprogramu[14]. Jedyna różnica polega na tym, że kiedy dochodzi do przekazania sterowania do podprogramu, na stosie sprzętowym umieszczany jest automatycznie adres instrukcji następującej po instrukcji skoku. Wspomniany stos sprzętowy może przechowywać do ośmiu adresów powrotu z podprogramu. Instrukcja powrotu z podprogramu powoduje pobranie adresu ze stosu sprzętowego i umieszczenie go w liczniku rozkazów. Procesory TMS320C5x mają wbudowany mechanizm pozwalający programowo rozszerzyć stos sprzętowy w obszarze pamięci danych. W celu zminimalizowania wpływu wykonywania instrukcji wywołania podprogramu i powrotu z podprogramu na szybkość przetwarzania danych dostępne są wersje tych instrukcji wykonywane z opóźnieniem.

W przypadku wielu instrukcji istnieje możliwość wielokrotnego ich powtarzania. Wystarczy tylko w 16-bitowym rejestrze RPTC zapisać liczbę N większą od zera[15], a następna instrukcja zostanie powtórzona N+1 razy. Dzięki tej własności oraz przetwarzaniu potokowemu wiele instrukcji, których wykonanie wymaga więcej niż jednego cyklu zegarowego, w wyniku wielokrotnego powtarzania staje się efektywnie instrukcjami wykonywanymi w jednym cyklu (patrz paragraf 3.6). Procesory z grupy TMS320C5x mają także zaimplementowany mechanizm powtarzania bloku instrukcji bez straty dodatkowych cykli zegarowych na organizację pętli.

Procesor może zostać programowo wprowadzony w stan o obniżonym poborze energii. Instrukcja IDLE powoduje zatrzymanie pobierania i wykonywania instrukcji. Porty szeregowe pozostają aktywne i kontynuują pracę. Instrukcja IDLE2 zatrzymuje także pracę portów szeregowych, dzięki czemu następuje dalszy spadek poboru energii. Z tego stanu procesor może zostać wyprowadzony jedynie poprzez przerwanie sprzętowe.

---

12 Dopiero wtedy ustalone są np. warunki sprawdzane przez skoki warunkowe.

13 Dwie instrukcje o długości jednego słowa lub jedna instrukcja o długości dwóch słów.

14 Nazywanego także skokiem ze śladem.

15 Do tego celu służy specjalna instrukcja RPT.

3.4. Przerwania

W celu zrozumienia mechanizmu obsługi przerwań sprzętowych przez procesory z grupy TMS320C5x warto najpierw przeanalizować działanie instrukcji przerwania programowego. Działa ona w sposób nieco podobny do wywołania podprogramu. Zdekodowanie tej instrukcji powoduje wstrzymanie pobierania kolejnych instrukcji i unieważnienie instrukcji już pobranej. Na stosie sprzętowym umieszcza się adres następnej instrukcji po instrukcji przerwania programowego. Na etapie wykonania sterowanie przekazywane jest do tej pozycji tablicy wektorów przerwań, którą wskazuje argument instrukcji przerwania programowego. Początek tablicy wektorów przerwań może zostać umieszczony gdziekolwiek w przestrzeni adresowej programu pod warunkiem, że zgodzimy się, aby jedenaście najmłodszych bitów jego adresu było równych zeru. Tablica zawiera 32 elementy, z których każdy to dwa słowa pamięci programu. Owe dwa słowa mają tworzyć procedurę obsługi przerwania. Ze względu na szczupłość miejsca mieści się tam zwykle zaledwie skok do właściwej procedury obsługi przerwania. W czasie wykonania instrukcji przerwania następuje zablokowanie przyjmowania dalszych przerwań maskowanych i wymiana kontekstu procesora. To ostatnie oznacza, że zawartość akumulatora ACC, rejestru zawierającego wynik mnożenia PREG, rejestru tymczasowego TREG0, rejestrów sterujących pracą procesora ST0 i ST1 oraz innych istotnych dla wykonywanego programu rejestrów umieszczana jest w rejestrach-duplikatach (ang. shadow registers). Instrukcja powrotu z procedury obsługi przerwania przywraca przerwany kontekst, pobiera ze stosu sprzętowego adres instrukcji występującej po instrukcji przerwania programowego i odblokowuje system przerwań.

Przerwania sprzętowe sygnalizowane mogą być przez urządzenia zewnętrzne za pośrednictwem czterech wejść przerwań maskowanych INT1-INT4 oraz wejścia przerwania niemaskowanego NMI. Źródłem przerwania sprzętowego może być także urządzenie wewnętrzne, np. port szeregowy. Od chwili zgłoszenia przerwania do chwili kiedy procesor podejmuje pierwsze działania upływają trzy okresy zegara taktującego[16]. Wtedy to procesor zaznacza w rejestrze przerwań IFR źródło przerwania, a w następnym cyklu zegarowym pobraną właśnie instrukcję zamienia na instrukcję przerwania programowego. Na stosie sprzętowym umieszcza się adres usuniętej instrukcji. Argumentem wstawionej instrukcji przerwania programowego jest numer pozycji w tablicy wektorów przerwań zawierającej procedurę obsługi zgłaszanego przetrwania sprzętowego. Przyporządkowanie przerwaniom

16 Pod warunkiem, że przerwanie nie jest zamaskowane, a system przerwań jest odblokowany.

sprzętowym procedur obsługi ustalone zostało na zawsze przez producenta procesora [8]. Od tej chwili wszystko dzieje się tak, jak to opisano dla instrukcji przerwania programowego.

3.5. Inne urządzenia wewnętrzne

Procesory z rodziny TMS320C5x integrowane są zwykle z przynajmniej jednym synchronicznym portem szeregowym. Port taki zaprojektowany jest dla komunikacji procesora z układem kodeka. Niektóre modele procesorów wyposażone są w synchroniczny port szeregowy z podziałem kanałów w dziedzinie czasu (ang. time division multiplex - TDM). Istnieje wtedy możliwość prowadzenia transmisji w siedmiu kanałach jednocześnie. Intencją producenta było wykorzystywanie szeregowego portu TDM dla komunikacji między procesorami w systemach wieloprocesorowych [8].

Wszystkie modele procesorów w rodzinie TMS320C5x posiadają zintegrowany timer pozwalający na precyzyjne odmierzanie czasu, bez angażowania w ten proces czasu obliczeniowego procesora.

3.6. Przykład programowania

Tablica 2 zawiera wydruk bardzo krótkiego podprogramu asemblerowego realizującego filtr FIR. Całość obliczeń realizowana jest praktycznie przez wielokrotne powtórzenie jednej instrukcji MADD. Instrukcja ta wykonuje wiele operacji jednocześnie:

- pobiera z pamięci danych jawnie wyszczególniony argument wskazywany przez aktywny rejestr adresowy AR0,
- pobiera z pamięci programu niejawny argument wskazywany przez rejestr BMAR (ang. block move address register),
- mnoży pobrane argumenty, a wynik umieszcza w rejestrze PREG,
- przenosi zawartość komórki wskazywanej przez rejestr AR0 do komórki o adresie o 1 większym,
- dekrementuje zawartość rejestru AR0 i inkrementuje zawartość rejestru BMAR,
- poprzednią zawartość rejestru PREG dodaje do zawartości akumulatora i zapamiętuje w akumulatorze.

Jeżeli tę instrukcję powtórzyć tyle razy, ile wynosi rząd filtru plus 1 (instrukcja RPT), to po dodatkowej akumulacji (instrukcja APAC) w akumulatorze znajdziemy próbkę odpowiedzi filtru. Przed uruchomieniem tego mechanizmu należy:

- próbkę sygnału wejściowego umieścić na początku linii opóźniającej filtru FIR,
- w rejestrze AR0 umieścić adres ostatniego elementu linii opóźniającej,
- do rejestru BMAR załadować adres tablicy, w której zapamiętano w odwrotnej kolejności próbki odpowiedzi impulsowej filtru,
- wyzerować akumulator oraz rejestr PREG (instrukcja ZAP).

Powtarzana instrukcja MADD, dzięki przetwarzaniu potokowemu, staje się efektywnie instrukcją wykonywaną w jednym cyklu zegarowym.

Budowa jednostki centralnej procesorów z rodziny TMS320C5x, implementacja praktycznie jednego rejestru wewnętrznego, lista instrukcji pozwalająca na realizację filtru FIR praktycznie za pomocą jednej instrukcji uzasadnia stwierdzenie, że wczesne procesory sygnałowe były w istocie programowalnymi filtrami cyfrowymi.

Tablica 2

Procedura realizująca filtr FIR napisana w asemblerze procesorów TMS320C5x

```
* Parametry:
*     próbka sygnału wejściowego na początku linii
*     ar0 -> koniec linii opóźniającej
*     ar1 -> rząd filtru (N)
*     bmar -> tablica współczynników filtru
* Zwraca:
*        acc(hi)  = sygnał wyjściowy
*
* Uwagi:
*      Linia opóźniająca zawiera N+1 elementów
*  + za końcem jedna komórka dodatkowa
*
* (C) Jacek Izydorczyk, 10.ii.1999

   .ps
UFir:
   ZAP                    ; ACC<-0; PREG <-0
   MAR      *,AR1         ; ARP -> AR0

   RPT      *,AR0         ; powtórz aż do wyczerpania
   MADD     *-            ; tablicy współczynników
   APAC                   ; ACC <- ACC+PREG

   SFL                    ; dla zachowania formatu 1.15
   ret                    ; powrót
```

4. Technologia MMX

Należy się spodziewać, że w niedalekiej przyszłości większość obliczeń wykonywanych przez komputery będzie związana z cyfrowym przetwarzaniem sygnałów [16]. Ma na to wpływ gwałtowne rozpowszechnianie się tzw. aplikacji mulimedialnych oraz rozwój rynku coraz bardziej skomplikowanych gier wykorzystujących technologię wirtualnej rzeczywistości. Aplikacje tego typu wykonują zadania obliczeniowe o następujących cechach charakterystycznych:

- liczby zapisane są w szesnasto- lub ośmiobitowym formacie stałoprzecinkowym,
- większość obliczeń to operacje mnożenia i akumulacji wyniku,
- obliczenia zorganizowane są w postaci licznych, zagnieżdżonych i ciasnych pętli programowych,
- wiele operacji można wykonywać równolegle.

W istocie są to cechy charakterystyczne dla algorytmów cyfrowego przetwarzania sygnałów. W związku z tym wiele firm wytwarzających mikroprocesory postanowiło rozszerzyć ich architekturę tak, aby sprzętowo wspomagać wykonywanie algorytmów DSP (ang. digital signal processing). Przykładem są instrukcje VIS wprowadzone do architektury SPARC przez firmę Sun Microsystems, instrukcje MDMX procesorów MIPS V firmy Silicon Graphics, instrukcje MVI dla procesorów Alpha firmy DEC[17] czy instrukcje MAX2 wprowadzone do architektury PA-RISC prze firmę Hewlett-Pacard [16]. Ale niewątpliwie najbardziej znanym przykładem jest rozszerzenie przez firmę Intel architektury IA-32 o instrukcje MMX (ang. multimedia extension) [24]. Cechą charakterystyczną tej technologii jest wprowadzenie do zbioru instrukcji procesora 57 nowych instrukcji wykonujących operacje na kilku argumentach jednocześnie. W ten sposób procesory zgodne z architekturą IA-32[18] nabrały z jednej strony cech procesorów sygnałowych, a z drugiej cech procesorów macierzowych SIMD (ang. single instruction multilple data).

Intencją autorów rozszerzenia MMX było zachowanie zgodności architektury z całą dotychczasową bazą oprogramowania. W szczególności chodziło o to, aby pojawienie się rozszerzenia MMX nie wymagało modyfikowania istniejących już systemów operacyjnych, tzn. MS-DOS, MS-Windows 3.1/95/98/NT, OS/2 i Unix. Dlatego zdecydowano, że kontekst jednostki wykonującej instrukcje MMX będzie wspólny dla niej oraz dla jednostki wykonują-

[17] Obecnie Intel.

[18] Pentium, Pentium Pro, Pentium II, Pentium III, AMD K-6, Cyrix M2, Cyrix MII.

cej operacje zmiennoprzecinkowe (koprocesora). Praktycznie oznacza to, że rejestry będące argumentami nowych instrukcji to osiem rejestrów koprocesora zmiennoprzecinkowego, które teraz nazywają się MM0-MM7. Wykorzystywane są sześćdziesiąt cztery bity, które w przypadku operacji zmiennoprzecinkowych reprezentują mantysę liczby. System operacyjny przechodząc do realizacji nowego zadania zachowuje stan koprocesora zmiennoprzecinkowego za pomocą instrukcji FSAVE, natomiast odtworzenie stanu koprocesora następuje z wykorzystaniem instrukcji FRSTR. W ten sposób system operacyjny „nieświadomie" zachowuje i odtwarza stan jednostki MMX.

4.1. Format przetwarzanych danych

Jednostka przetwarzająca instrukcje MMX jest w istocie jednostką arytmetyki stałoprzecinkowej[19]. Przetwarzane liczby całkowite mogą być zapisane w formacie zajmującym jeden bajt, słowo (16 bitów), podwójne słowo (32 bity) lub poczwórne słowo (64 bity). Liczby grupowane są w 64-bitowe ciągi binarne, zapisywane w rejestrach MM0-MM7 i przetwarzane. Na przykład pojedynczy piksel grafiki reprezentowany jest w postaci ośmiu bitów (bajt). Osiem takich pikseli może zostać umieszczonych w jednym, poczwórnym słowie pamięci. Realizacja instrukcji MMX polega na pobraniu takiego poczwórnego słowa[20], wykonaniu operacji na wszystkich ośmiu pikselach jednocześnie i zapisaniu wyniku w jednym z rejestrów MMX0-MMX7. Arytmetyka stałoprzecinkowa zmusza programistę do dokładnego przemyślenia realizowanego algorytmu tak, aby na żadnym z etapów przetwarzania nie doszło do niekontrolowanego przekroczenia zakresu reprezentowanych liczb.

4.2. Przegląd instrukcji

Instrukcje MMX realizują następujące operacje:

- podstawowe działania arytmetyczne, tzn. dodawanie, odejmowanie, mnożenie, mnożenie połączone z akumulacją, przesunięcia arytmetyczne w lewo i w prawo,
- porównywanie liczb,

---

19 Procesor Pentium, Cyrix MII i AMD K-6 posiadają jedną jednostkę przetwarzającą instrukcje MMX. Procesor Pentium II i Pentium Celleron posiadają dwie takie jednostki działające współbieżnie.

20 Magistrale wewnętrzne procesora są 64-bitowe.

- konwersję formatów — pakowanie liczb w ciągi 64-bitowe, konwersję z formatów krótkich (np. bajt) do formatów dłuższych (np. słowo),
- podstawowe operacje logiczne AND, NAND, OR, XOR wykonywane na parach odpowiadających sobie bitów,
- operacje przesunięcia logicznego w lewo i w prawo,
- ładowanie danych z pamięci do rejestrów procesora, składowanie danych w pamięci operacyjnej i przesłania między rejestrami procesora.

Każda instrukcja może być wykonywana na danych o innym (krótszym lub dłuższym) formacie. Każdej takiej instrukcji odpowiada inny kod binarny, co pokazano w tablicy 3.

W charakterze przykładu rozważmy instrukcję mnożenia połączonego z akumulacją wyników:

PMADDWD MM0, MM1

Zawartość rejestrów MM0 oraz MM1 traktowana jest jak czwórka 16-bitowych liczb zapisanych w formacie stałoprzecinkowym:

MM0=	-100	-62	17	-30
MM1=	64	-66	42	-40

W wyniku mnożenia odpowiadających sobie par liczb otrzymujemy cztery 32-bitowe liczby:

-6400	4092	714	1200

Pierwsza para iloczynów powstała w wyniku przemnożenia odpowiadających sobie liczb leżących w górnej części rejestrów MM0 i MM1. Druga para iloczynów powstała w wyniku przemnożenia odpowiadających sobie liczb leżących w dolnej części rejestrów MM0 i MM1. Pierwsze dwa iloczyny są dodawane do siebie i umieszczane w rejestrze w górnej części rejestru MM0. W dolnej części rejestru MM0 umieszczana jest suma drugiej pary iloczynów:

MM0=	-2308	1914

Firma Intel twierdzi [23], że jeżeli oba argumenty instrukcji PMADDWD są rejestrami jednostki MMX, to procesor Pentium jest w stanie średnio realizować jedną taką instrukcję co 1.5 taktu zegarowego. Jeśli jeden z argumentów musi zostać pobrany z pamięci, jedna instrukcja PMADDWD jest realizowana co dwa takty zegarowe. Dla typowej obecnie częstotliwości zegara komputera PC mieszczącej się w granicach od 300 MHz do 500 MHz daje to od 600 milionów do 1.333 miliarda mnożeń połączonych z akumulacją na sekundę!

Tablica 3

Zbiór instrukcji MMX rozszerzających architekturę IA-32

Typ	Mnemonika	N	Opis
Arytmetyczne	PADD[B,W,D]	3	Dodawanie bez nasycenia [8,16,32 bity]
	PADDS[B,W]	2	Dodawanie liczb ze znakiem [8,16,32 bity] nasycenie
	PADDUS[B,W]	2	Dodawanie liczb bez znaku [8,16 bitów], nasycenie
	PSUB[B,W,D]	3	Odejmowanie bez nasycenia [8,16,32 bity]
	PSUBS[B,W]	2	Odejmowanie liczb ze znakiem [8,16 bitów] nasycenia
	PSUBUS[B,W]	2	Odejmowanie liczb bez znaku [8,16 bitów], nasycenie
	PMULHW	1	Mnożenie górnej części rejestru
	PMULLW	1	Mnożenie dolnej części rejestru
	PMADDWD	1	Mnożenie połączone z akumulacją
Porównanie	PCMPEQ[B,W,D]	3	Porównanie liczb [8,16,32 bity] czy równe?
	PCMPGT[B,W,D]	3	Porównanie liczb [8,16,32 bity] czy większa?
Konwersja formatu	PACKUSWB	1	Pakuj słowa w bajty (bez znaku, z nasyceniem)
	PACKSS[WB,DW]	2	Pakuj [słowa w bajty, podwójne słowa w poczwórne słowa] (znak i nasycenie)
	PUNPCKH [BW,WD,DQ]	3	Rozpakuj starsze [8,16,32 bity] z rejestru MMX
	PUNPCKL [BW,WD,DQ]	3	Rozpakuj młodsze [8,16,32 bity] z rejestru MMX
Logiczne	PAND	1	AND
	PANDN	1	AND NOT
	POR	1	OR
	PXOR	1	XOR
Przesunięcia	PSLL[W,D,Q]	6	Przesunięcie logiczne w lewo [8,16,32 bity]
	PSRL[W,D,Q]	6	Przesunięcie logiczne w prawo [8,16,32 bity]
	PSRA[W,D]	4	Przesunięcie arytmetyczne w prawo [8,16 bitów]
Przesłania	MOV[D,Q]	4	Przesłanie [32,64 bitów] z / do rejestru MMX
stan FP&MMX	EMMS	1	Zeruj stan MMX

N liczna różnych kodów binarnych

4.3. Programowanie

Ze względu na zastosowane rozwiązania w konstrukcji jednostki MMX firma Intel nie zaleca stosowania na przemian instrukcji MMX oraz instrukcji wykonujących działania na liczbach w formacie zmiennoprzecinkowym. Główne przesłanki tego zalecenia są następujące:

- wykonanie instrukcji MMX powoduje ustawienie wskaźnika stosu rejestrów zmiennoprzecinkowych TOS (ang. top of stack) na zero. Instrukcje zmiennoprzecinkowe tracą w ten sposób wskaźnik stosu rejestrów, a tym samym dalsza realizacja operacji zmiennoprzecinkowych przebiega błędnie,
- instrukcja MMX wpisując do rejestru MMX dane powoduje, że w części tego rejestru używanej przez instrukcje zmiennoprzecinkowe dla przechowywania cechy, wpisywane są same jedynki. Taka zawartość rejestru zmiennoprzecinkowego nie reprezentuje żadnej liczby zmiennoprzecinkowej i próba wykonania jakiejkolwiek operacji zmiennoprzecinkowej na tym rejestrze kończy się zgłoszeniem przerwania do jednostki centralnej,
- użycie dowolnej z instrukcji MMX powoduje, że tracona jest zawartość rejestru TAG (rejestr TAG wypełniany jest zerami),
- częste przełączanie między instrukcjami MMX i instrukcjami zmiennoprzecinkowymi może spowodować znaczne zmniejszenie prędkości przetwarzania.

Jeżeli program przeprowadza obliczenia na liczbach zmiennoprzecinkowych oraz obliczenia z wykorzystaniem instrukcji MMX, firma zaleca:

- podział programu na osobne moduły, z których jeden przeprowadza tylko obliczenia na liczbach zmiennoprzecinkowych, a drugi tylko obliczenia z wykorzystaniem instrukcji MMX,
- nie należy przekazywać przez rejestry parametrów między jednym i drugim modułem,
- w chwili zakończenia obliczeń z wykorzystaniem instrukcji MMX należy użyć instrukcji EMMS w celu opróżnienia stosu rejestrów zmiennoprzecinkowych,
- kończąc obliczenia zmiennoprzecinkowe należy pozostawić pusty stos rejestrów.

Powyższe zalecenia mają charakter ogólny. Tablica 4 przedstawia natomiast program obliczający iloczyn skalarny dwóch wektorów o stu składowych [24, 23]. W programie założono, że składowe obu wektorów zapisane są w tablicach wskazywanych odpowiednio przez rejestry EDX i EDI. Każda składowa jest 16-bitową całkowitą liczbą stałoprzecinkową. Licznik pętli zorganizowano z wykorzystaniem rejestru ECX. Licznik zawiera liczbę równą (-2)*100, czyli liczbę przeciwną do liczby bajtów tablicy zawierającej składowe wektora. W pętli odbywa się:

- ładowanie do rejestru MM1 czterech składowych wektora mnożnej,
- mnożenie składowych pierwszego czynnika przez odpowiadające im składowe drugiego czynnika połączone z dodaniem wyników par mnożeń. Trzydziestodwubitowe wyniki dodawania umieszczone są w rejestrze MM1,
- akumulacja dwóch dodawań w rejestrze MM0.

Po zakończeniu wykonywania pętli w chwili, gdy wyzerowana zostanie zawartość rejestru ECX, górna i dolna część rejestru MM0 zawierają dwie sumy częściowe, które dodane tworzą iloczyn skalarny wektorów. Zawartość akumulatora MM0 zostaje zachowana w rejestrze MM1, górna część rejestru MM0 przesunięta na miejsce dolnej, a następnie 32-bitowe liczby zawarte w rejestrach MM1 i MM0 zostają dodane do siebie. Dolna część rejestru MM0 to obliczany iloczyn skalarny. Wystarczy teraz dokonać zaokrąglenia wyniku do 16-bitów przez spakowanie zawartości akumulatora MM0.

5. Podsumowanie

Procesory sygnałowe funkcjonują na rynku już prawie dwadzieścia lat. Dopiero jednak intensywny rozwój telefonii komórkowej i aplikacji multimedialnych dla komputerów PC w latach dziewięćdziesiątych spowodował ich zastosowanie w sprzęcie powszechnego użytku. Początkowo architektura procesorów sygnałowych podporządkowana była konieczności umieszczenia na chipie układu mnożącego działającego w jednym cyklu zegarowym. Z czasem wzbogacona została o przetwarzanie potokowe. Obecnie wytwarzane procesory sygnałowe mają architekturę SIMD. Wydaje się jednak, że najbliższa przyszłość należy do układów VLIW.

Tablica 4

Program obliczający iloczyn skalarny z wykorzystaniem instrukcji MMX™

```
; Parametry:
;     próbka sygnału wejściowego na początku linii
;     edx -> a[0]
;     edi -> b[0]
;     bmar -> tablica współczynników filtru
; Zwraca:
;     iloczyn skalarny a[i]*b[i] w najmłodszym słowie rejestru MM0

    mov       ecx, 100          ; ecx <- 100 == N
    sal       ecx,1             ; ecx <- ecx*2
    add       edx,ecx           ; edx -> a[N]
    add       edi,ecx           ; edi -> a[N]
    neg       ecx               ; ecx <- (-2)*N

    pxor      mm0,mm0           ; mm0 <- 0

LOOP:
    movq      mm1,[edx+ecx]     ; mm1 <- a[i] a[i+1] a[i+2] a[i+3]
    pmadddw   mm1,[edi+ecx]     ; mm1 <- a[i]*b[i]+a[i+1]*b[i+1]
                                ;        a[i+2]*b[i+2]+a[i+3]*b[i+3]

    padddw    mm0,mm1           ; mm0 <- mm0 + mm1
    add       ecx,8             ; ecx <- ecx + 8
    jnz       LOOP              ; skocz jeśli to nie koniec pętli

    movq      mm1,mm0           ; mm1 <- mm0
    psrlq     mm0,32            ; mm0 <- mm0/2^(32)
    padddw    mm0,mm1           ; mm0 <- mm0 + mm1
    packssdw  mm0,mm0           ; przekształć na format 1.15
```

Literatura

1. Tanenbaum A.S. Organizacja maszyn cyfrowych w ujęciu strukturalnym, WNT, Warszawa 1980.
2. Mano M.M.: Architektura komputerów, WNT, Warszawa 1988.
3. Pochopień B.: Arytmetyka systemów cyfrowych, Skrypt Pol. Śl. nr 1548, Gliwice 1990.
4. Węgrzyn S.: Systemy sterowane przepływem operacji i systemy sterowane przepływem argumentów, Zeszyty Naukowe Pol. Śl., ser. Informatyka, z.24, Gliwice 1993, ss.9-19.
5. Węgrzyn S.: Przyspieszenie realizacji algorytmów w systemach sterowanych przepływem argumentów, Zeszyty Naukowe Pol. Śl., ser. Informatyka, z.28, Gliwice 1994, ss.67-75.
6. [6] http://www.ti.com/sc/docs/integrat/97dec/dspfirst.html
7. Texas Instruments, TMS320C1x/2x/2xx/5x Assembly Language Tools, Users Guide, 1995 (SPRU018D).
8. Texas Instruments, TMS320C5x Users Guide, 1996 (SPRU056C).
9. Nat Seshan, High VelociTI Processing, IEEE Signal Processing Magazine, Vol. 15, no 2, 1998, pp.86-101.
10. Chen T. [ed.]: VLSI Design and Implementation Fuels Signal-Processing Revolution, IEEE Signal Processing Magazine, Vol. 15, No 1, 1998, pp.22-37.
11. Schlett M.: Trends in Embedded-Microprocesor Design, Computer, Vol 31, No 8, 1998, pp.44-49.
12. Dulong C.: The IA-64 Architecture at Work, Computer, Vol 31, No 7, 1998, pp.24-32.
13. Eyre J., Bier J.: DSP Processors Hit the Mainstream, Computer, Vol 31, No 8, 1998, pp.51-59.
14. Texas Instruments, TMS320C62xx CPU and Instruction Set Reference Guide, styczeń 1997 (SPRU189A).
15. Matzke D.: Will Physical Scalability Sabotage Performance Gains?, Computer, Vol 30, No 9, 1997, pp.37-39.
16. Diefendorff K., Dubey P.K.: How Multimedia Workloads Will Change Processor Design, Computer, Vol 30, No 9, 1997, pp.43-45.
17. Patt Y.N., Evers M., Friendly D.H., Stark J.: One Billion Transistors, One Uniprocessor, One Chip, Computer, Vol 30, No 9, 1997, pp.51-57.
18. Texas Instruments, TMS320C62xx Peripherals Reference Guide, styczeń 1997, (SPRU190).
19. Texas Instruments, TMS320C62xx Programmer's Guide, styczeń 1997, (SPRU198).

20. Texas Instruments, TMS320C6x Optimizing C Compiler User's Guide, styczeń 1997, (SPRU187A).
21. Siemens A.G., TriCore Architecture Overview Handbook, 22 luty 1999.
22. Siemens A.G., TriCore Architecture Manual, 14 grudzień 1998.
23. Intel, Using MMX* Technology Instructions to Compute a 16-Bit FIR Filter, Application Note AP-559, http://developer.intel.com/drg/mmx/AppNotes/AP559.htm
24. Intel, MMX Technology Technical Overview. http://developer.intel.com/drg/mmx/Manuals/overview/index.htm
25. Stevens J.: DSPs in Communications, IEEE Spectrum, 1998, pp.39-46.
26. Inacio C., Ombres D.: The DSP Decision: Fixed Point or Floating?, IEEE Spectrum, 1996, pp.72-74.
27. Lee E.A., Messerschmitt D.G: Digital Communication, Kluwer Academic Publishers, 1994.
28. Proakis J.G.: Digital Communications, McGraw Hill, 1995.
29. [29] Ian Glover, Peter Grant: Digital Communications, Prentice Hall Inc., 1998
30. Izydorczyk J., Płonka G., Tyma G.: Teoria sygnałów, wstęp, Wydawnictwo Helion, Gliwice 1999.
31. Analog Devices: ASDP-2100 Family User's Manual, 1995.
32. ZSP Corporation: The ZSP16401 Digital Signal Processor, Product Note v0.6, 27 marzec 1998.
33. Płonka G., Tyma G.: Laboratorium Cyfrowego Przetwarzania Sygnałów Instytutu Elektroniki Politechniki Śląskiej, Przegląd Telekomunikacyjny + Wiadomości Telekomunikacyjne, nr 10, 1996.
34. Analog Devices: ADSP-21160 Technical Specifivation, Rev 3.0, 24 luty 1999.
35. Intel: P6 Family of Processors Hardware Developer's Manual, wrzesień 1998. http://www.cyrix.com/html/products/mii/index.htm
36. Advanced Micro Devices: AMD-K6 Data Sheet, 1998.

Recenzent: Prof.dr hab.inż. Jerzy Skubis

Abstract

The article is about architecture of Digital Signal Processors (DSP). There are some obstacles, specific to applications of digital signal processing, which modulated evolution of DSP architecture. The first: the DSP should multiply numbers very fast but it should not be too accurate. 32-bit fixed point arithmetic is sufficient in most cases. The second: the DSP should be very fast but used in GSM handset could not discharge battery overnight. The third: the DSP should posses all hardware needed for data processing on the board. CPU must be integrated with a lot of fast memory and various peripherals making a microcomputer on a chip. First commercially available DSP – TMS32010 – could not meet all demands but it could multiply two 16-bit fixed point numbers in four clock cycles. Architecture of the TMS32010 have been improved by Texas Instruments and is available on the marked till now as TMS320C1x/2x/2xx/5x/5xx family of DSP. Need for software compatibility frozen a lot of old ideas and compromises in this family of DSP's. An example of newer architecture is MMX extension of Intel IA32 architecture. This extension add some digital signal processing capabilities to the most widespread microprocessor – the heard of every PC computer. The MMX extension is made with single instruction multiple data (SIMD) idea in mind. This idea makes it possible to introduce an element of parallel computing into practically all existing DSP architectures. Therefore MMX has been chosen to illustrate architecture of today DSP. But near future belongs probably to VLIW machines described in the part 2 of this article.