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

Jacek IZYDORCZYK
Politechnika Śląska, Instytut Elektroniki

PROCESORY SYGNAŁOWE
Część 2. Czyli niedaleka przyszłość

Streszczenie. Druga część artykułu o architekturze procesorów sygnałowych poświęcona jest tendencjom rozwojowym. W charakterze ilustracji wybrano dwa rozwiązania. Są to: 1) procesor sygnałowy TMS320C6x firmy Texas Instruments ilustrujący ideę procesorów o bardzo długim rozkazie (VLIW); 2) mikrokontroler TC-1 firmy Infineon (dawniej Siemens) integrujący w postaci jednej superskalarnej maszyny funkcję klasycznego mikrokontrolera oraz zupełnie niezłego procesora sygnałowego. Ilustruje to dwie tendencje w tej dziedzinie. Z jednej strony dąży się do stworzenia układu specjalizowanego o jak największej mocy obliczeniowej (TMS320C6x). Z drugiej strony powstają konstrukcje potencjalnie wolniejsze, ale bardziej uniwersalne, uwalniające konstruktora od stosowania w jednym urządzeniu (np. telefonie GSM) wielu mikroprocesorów.

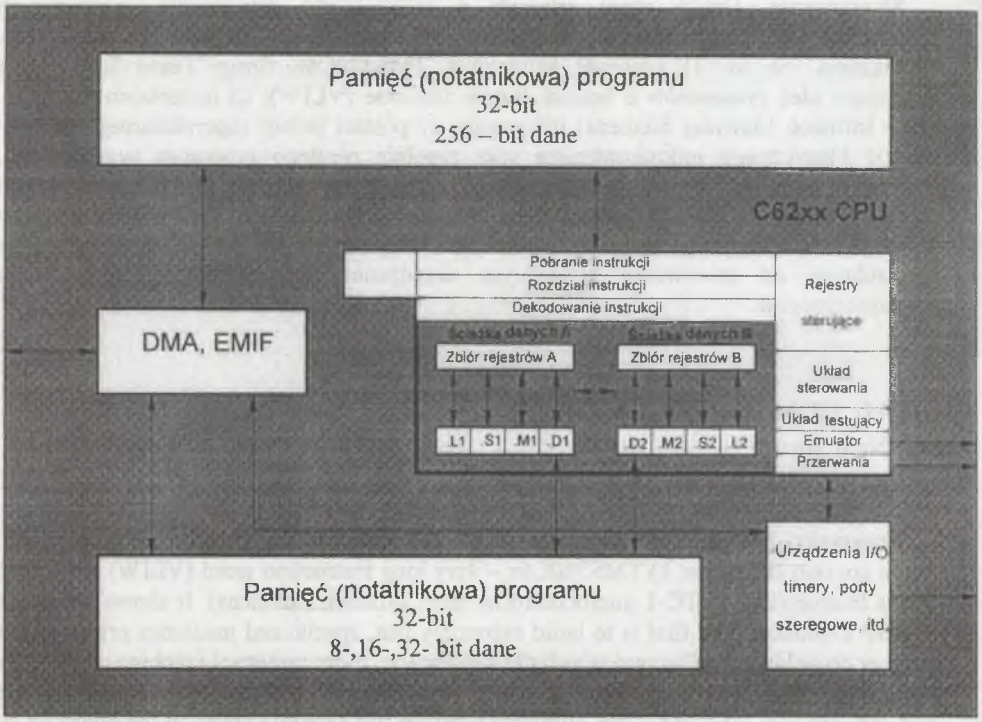
DIGITAL SIGNAL PROCESSORS
Part 2. Near future

Summary. The second part of the article is devoted to the near future of the DSP. There are two examples: 1) TMS320C6x – very long instruction word (VLIW) DSP from Texas Instruments; 2) TC-1 microcontroller from Infineon (Siemens). It shows two ways of DSP evolution. The first is to build extremely fast, specialized machines primarily for number crunching. The second is to build superscalar, more universal machine capable for multiprocessing. Such a machine integrate capabilities of conventional microcontroller and conventional DSP and it can eliminate need for two separate chips on the board eg. of GSM handset.

1. VelociTI – czyli procesory VLIW firmy Texas Instruments

Po wielu latach sukcesów rynkowych firma Texas Instruments zdecydowała się unowocześnić i ujednolicić architekturę oferowanych procesorów sygnałowych. W ten sposób powstała specyfikacja architektury VLIW (ang. very long instruction word) nazwana przez fir-

mę VelociTI [9]. Architektura odnosi się do układów przetwarzających liczby w formacie stałoprzecinkowym oraz do układów przetwarzających liczby zapisane w formacie zmiennoprzecinkowym. W roku 1997 na rynku pojawił się układ TMS320C6201 — pierwszy z rodziny o nowej architekturze [6]. Układ ten, przetwarzający liczby stałoprzecinkowe, taktowany zegarem o częstotliwości 200 MHz może osiągać szczytową wydajność 1600 MIPS. Producent przewiduje, że znajdzie on zastosowanie w modemach i wokoderach przeznaczonych do obsługi wielu kanałów jednocześnie[1] oraz do przetwarzania obrazów [9]. Drugi układ z rodziny to TMS320C6701 — procesor zmiennoprzecinkowy, który taktowany zegarem 166 MHz osiąga szczytową wydajność dochodzącą do 1000 Mflops.



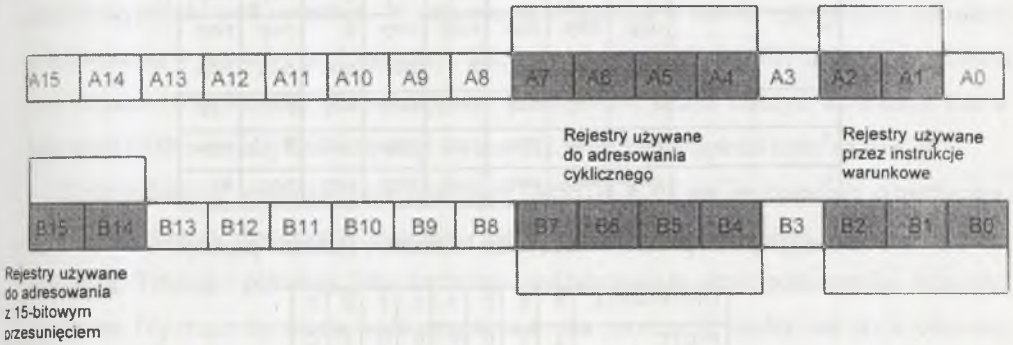
Rys.1. Schemat blokowy jednostki centralnej procesora TMS320C6201 [18]
Fig.1. Block diagram of TMS320C6201 CPU [18]

1.1. Jednostka centralna TMS320C6201

Jednostka centralna układu TMS320C6201 przedstawiona jest na rysunku 1. Układ zawiera osiem jednostek wykonawczych podzielonych na dwie grupy i oznaczonych L1, S1,

[1] Obie aplikacje stanowią standardową część stacji bazowej cyfrowej telefonii komórkowej.

M1, D1 oraz L2, S2, M2, D2. Każda z grup ma do dyspozycji szesnaście 32-bitowych rejestrów — patrz rysunek 2. Dla pierwszej grupy jednostek są to rejestry A0-A15, dla grupy drugiej są to rejestry B0-B15. Każda jednostka wykonawcza ma swobodny dostęp do rejestrów swojej grupy. Każda grupa jednostek ma jeden port pozwalający na pobieranie lub zapisywanie danych w pamięci. W ten sposób w danym cyklu tylko jedna jednostka wykonawcza w każdej z grup może zapisywać (odczytywać) pamięć. Podobny port pozwala grupie jednostek na dostęp do rejestrów drugiej grupy jednostek.



Rys.2. Rejestry procesora TMS320C6201
Fig.2. Registers of the TMS320C6201 CPU

Instrukcje pobierane są z pamięci programu w porcjach po 256 bitów (ang. fetch packet), które nazwiemy liniami. Porcja taka zawiera osiem 32-bitowych instrukcji, z których każda odnosi się do jednej jednostki wykonawczej. Najmłodszy bit każdej z instrukcji określany jest angielską nazwą P-bit. Jeżeli P-bit instrukcji X równy jest 1, to następna instrukcja Y będzie wykonywana równocześnie z instrukcją X[2]. W ten sposób tworzone są łańcuchy instrukcji wykonywanych równocześnie przez różne jednostki wykonawcze. Łańcuch taki nazywany jest w terminologii wprowadzonej przez firmę Texas Instruments execute packet. Każda linia instrukcji pobranych przez jednostkę centralną (fetch packet) musi zawierać całkowitą liczbę łańcuchów (execute packet), tzn. że P-bit ostatniej[3] z pobranych instrukcji musi być równy 0. Zachodzą przy tym trzy możliwości [14]:

- Wszystkie pobrane instrukcje (fetch packet) mają wyzerowany P-bit:

[2] Pod warunkiem, że odnoszą się do innych jednostek wykonawczych.

[3] Magistrala służąca pobieraniu instrukcji jest magistralą 256-bitową. Osiem instrukcji pobieranych jest równocześnie. Należałoby zatem mówić o najmłodszej (najmniej znaczącej) z pobranych instrukcji.

Instrukcja	A	B	C	D	E	F	G	H
P-bit	0	0	0	0	0	0	0	0

W związku z tym instrukcje te wykonywane są sekwencyjnie, np. tak:

Takt zegara	L1	S1	D1	M1	L2	S2	D2	M2
1	nop	Nop	**A**	nop	nop	nop	nop	nop
2	nop	**B**	nop	nop	nop	nop	nop	nop
3	nop	Nop	nop	nop	nop	**C**	nop	nop
4	nop	Nop	nop	nop	nop	**D**	nop	nop
5	nop	Nop	nop	**E**	nop	nop	nop	nop
6	**F**	Nop	nop	nop	nop	nop	nop	nop
7	nop	Nop	nop	nop	nop	nop	**G**	nop
8	nop	Nop	nop	nop	nop	nop	nop	**H**

- Pobrane instrukcje (fetch packet) tworzą kilka łańcuchów (execute packet):

Instrukcja	A	B	C	D	E	F	G	H
P-bit	1	1	0	1	0	0	1	0

i wykonywane są częściowo równolegle:

Takt zegara	L1	S1	D1	M1	L2	S2	D2	M2
1	nop	**B**	**A**	nop	nop	nop	**C**	nop
2	nop	Nop	**E**	nop	**D**	nop	nop	nop
3	**F**	Nop	nop	nop	nop	nop	nop	nop
4	nop	Nop	nop	nop	nop	nop	**H**	**G**

- Pobrane instrukcje (fetch packet) tworzą jeden łańcuch (execute packet):

Instrukcja	A	B	C	D	E	F	G	H
P-bit	1	1	1	1	1	1	1	0

i wykonywane są wszystkie równolegle:

Takt zegara	L1	S1	D1	M1	L2	S2	D2	M2
1	**A**	**B**	**C**	**D**	**E**	**F**	**G**	**H**

Ze względu na szybkość przetwarzania danych najkorzystniejsza wydaje się sytuacja, kiedy wszystkie pobrane instrukcje są wykonywane równolegle i wszystkie jednostki wykonawcze w każdym cyklu zegarowym są „zajęte" użyteczną pracą. Trudno się jednak spodziewać, że uda się w ten sposób zakodować każdy algorytm. Najczęściej będziemy mieli

do czynienia z „częściową" równoległością przetwarzania. Istotna trudność polega jedynie na tym, że granice łańcuchów instrukcji (execute packet) nie będą się pojawiały regularnie co osiem instrukcji. Mamy wtedy do dyspozycji przynajmniej dwie strategie postępowania. W trosce o ilość pamięci zajmowanej przez kod programu łańcuchy instrukcji zostaną „przerwane" co osiem instrukcji. W ten sposób nie zmienimy rozmiarów kodu, ale przestaje to być kod optymalny pod względem szybkości przetwarzania. Optymalność kodu można zachować uzupełniając go o instrukcje NOP[4] tak, aby granice łańcuchów instrukcji (execute packet) wypadały co 8 instrukcji. W najgorszym przypadku w kodzie optymalnym instrukcje grupowane są w łańcuchy po 5 instrukcji. Każdy łańcuch musi być wtedy uzupełniony trzema instrukcjami NOP, co daje wzrost objętości kodu o 60%. Jeżeli założyć, że średnia liczba instrukcji NOP w każdej linii instrukcji wynosi 1,5, to wzrost objętości kodu wynosi 30%.

Zbiór jednostek wykonawczych procesora TMS320C6201 nie jest zupełnie symetryczny, choć wiele często używanych instrukcji może być wykonywane przez więcej niż jedną jednostkę. Tablica 1 pokazuje, jakie instrukcje wykonywane są przez poszczególne jednostki procesora. Na etapie tworzenia kodu programu trzeba przydzielić każdej instrukcji jednostkę procesora, która ją wykona. Na tym etapie zapada też decyzja, które instrukcje będą wykonywane jednocześnie oraz uzupełnia się kod programu o instrukcje NOP. Wszystkie operacje wykonywane są na argumentach pobieranych z rejestrów procesora. Wynik działania instrukcji zapamiętywany jest także w rejestrze. Oddzielono od siebie w ten sposób operacje dostępu do pamięci (odczyt i zapis) od właściwych operacji przetwarzania. Zbiór instrukcji jest zbiorem typu RISC (ang. reduced instruction set computer), tzn. że operacje wykonywane przez instrukcje są w miarę proste i niezależne od siebie. Ponieważ procesor posiada duży zbiór uniwersalnych rejestrów, możliwa jest konstrukcja kompilatorów generujących kod bardzo zbliżony do kodu optymalnego[5] pod względem szybkości przetwarzania danych [9]. Szczegółową listę instrukcji procesora TMS320C6201 można znaleźć w [14]. Godne podkreślenia są przy tym następujące cechy tej listy:

- procesor realizuje arytmetykę z nasyceniem,
- sprzętowo wspomaga realizację algorytmu dzielenia liczb stałoprzecinkowych,

[4] Instrukcja oznaczająca „nic nie rób" (ang. no operation).

[5] Punktem odniesienia są tu algorytmy kodowane ręcznie na poziomie asemblera.

Tablica 1

Przyporządkowanie instrukcji poszczególnym jednostkom wykonawczym procesora TMS320C6201 [9]

L	S	D	M
Dodawanie liczb stałoprzecinkowych	Dodawanie liczb stałoprzecinkowych	Dodawanie liczb stałoprzecinkowych	Mnożenie liczb stałoprzecinkowych
Operacje logiczne	Operacje logiczne	Ładowanie i zapamiętywanie danych	
Zliczanie bitów	Operacje na bitach		
	Przesuwanie		
	Ładowanie stałych		
	Instrukcje skoku i wywołania podprogramu		

- instrukcje zliczania nadmiarowych bitów znaku ułatwiają programową realizację arytmetyki zmiennoprzecinkowej oraz skalowanie grupy liczb pojawiających się np. w stałoprzecinkowym algorytmie FFT,
- instrukcje porównania umieszczają wynik (jedynkę lub zero) w rejestrach ogólnego przeznaczenia — patrz rysunek 2,
- instrukcja ADD2 (SUB2) pozwala na dodanie (odjęcie) zawartości dwóch rejestrów z pominięciem przeniesienia z pozycji binarnej 15 na 16 — w ten sposób jedna instrukcja wykonuje dwa dodawania (odejmowania) liczb 16-bitowych,
- procesor wykonuje operacje na wybranych grupach bitów — ekstrakcja, ustawianie, zerowanie,
- najmniejszą adresowalną jednostką danych jest bajt,
- adresowanie odbywa się z wykorzystaniem rejestrów ogólnego przeznaczenia. Adresy są 32-bitowe. Daje to przestrzeń adresową 4 GB. Adresowanie cykliczne (bufory cykliczne) mogą być realizowane z wykorzystaniem wybranych rejestrów — patrz rys.5.

Wszystkie instrukcje procesora TMS320C6201 mogą być wykonywane warunkowo. Testowana jest wtedy zawartość jednego z rejestrów A1, A2, B0, B1, B2, przy czym zero oznacza tradycyjnie fałsz, a wartość różna od zera prawdę. Każda z instrukcji, niezależnie od tego czy warunek jest spełniony czy też nie, wchodzi w etap wykonania. Dla instrukcji, której warunek nie jest spełniony, wynik działania nie jest zapisywany do rejestru przeznaczenia. Podobnie jest w przypadku instrukcji zapisu w pamięci. Jeżeli warunek nie jest spełniony, nie jest uruchamiany mechanizm dostępu do pamięci. Instrukcje warunkowe pozwalają na

eleganckie konstruowanie pętli if-else bez wykonywania instrukcji skoku[6]. W tablicy 2 pokazany jest fragment programu napisanego w języku C. Program składa się właściwie z dwóch instrukcji if-else:

- pierwsza instrukcja sprawdza zmienną b2. Jeżeli jest ona różna od zera, liczbę przechowywaną przez zmienną a5 powiększa się o liczbę przechowywaną przez zmienną b4,
- druga instrukcja sprawdza zmienną a0. Jeżeli jest równa zero, następuje operacja mnożenia, w przeciwnym wypadku zerowana jest zmienna b0 i inkrementowana jest zmienna a5.

Ten sam program napisany w asemblerze procesora TMS320C6201 przedstawiony jest w tablicy 3. Cały program składa się zaledwie z jednego łańcucha instrukcji wykonywanych w tym samym cyklu zegarowym[7] (execute packet). Pierwszej instrukcji programu napisanego w języku C odpowiada pierwsza instrukcja programu asemblerowego — warunkowa akumulacja zawartości rejestru B4 w rejestrze A5. Pozostałe trzy instrukcje asemblerowe realizują drugą instrukcję if-else. Wszystkie trzy instrukcje będą wykonywane jednocześnie, ale ze względu na wzajemnie wykluczające się warunki zostanie utrwalony tylko wynik realizacji pierwszej z nich (część if instrukcji w języku C) albo wynik działania pozostałych dwóch (część else instrukcji w języku C). Nie ma tu żadnych skoków i związanych z tym opóźnień.

Tablica 2

Przykładowy fragment programu w języku C zawierający instrukcje if-else

```
if (b2) a5 += b4;
if (a0) b0 = a0 * (unsigned) (b1 >> 16);
else
   {
   b0 = 0;
   a5 += 1;
   }
```

[6] Efektywny czas wykonania instrukcji skoku warunkowego w przypadku procesora z rozbudowanym przetwarzaniem potokowym (a takim procesorem jest TMS320C6201) może być równy nawet kilkudziesięciu okresom zegara taktującego.

[7] W asemblerze kolejne instrukcje łączone są w łańcuchy za pomocą dwóch pionowych kresek „||" wstawionych między łączone instrukcje.

Tablica 3

Realizacja instrukcji if-else w asemblerze procesora TMS320C6201

```
     [B2]   ADD          .L1X    A5, B4, A5
||   [A0]   MPYLSHU      .M2X    A0, B1, B0
||   [!A0]  ZERO         .S1     B0
||   [!A0]  ADD          .D1     A5, 1, A5
```

1.2. Potok instrukcji

Ograniczenia współczesnej submikronowej technologii elektronowej [15] skłaniają konstruktorów do tworzenia procesorów o bardzo rozwiniętym przetwarzaniu potokowym. Tak też jest w przypadku procesora TMS320C6201. Przetwarzanie potokowe składa się z trzech zasadniczych etapów:

- pobranie instrukcji obejmujące cztery podetapy — cykle zegarowe:
  - generowanie adresu następnej linii instrukcji lub wyliczenie adresu instrukcji skoku,
  - przekazanie adresu do pamięci programu,
  - sprawdzenie, czy pobierana linia zapisana jest w pamięci notatnikowej; jeżeli pobranie linii instrukcji wymaga sięgnięcia do pamięci zewnętrznej, wstrzymywane jest przetwarzanie całego potoku instrukcji na czas potrzebny do pobrania instrukcji z pamięci zewnętrznej; jest to jedyny przypadek, kiedy wstrzymywany jest *cały* potok instrukcji,
  - przekazanie pobranej instrukcji jednostce centralnej.
- dekodowanie instrukcji, które realizowane jest w dwóch podetapach:
  - w pobranej linii instrukcji wydzielane są łańcuchy instrukcji wykonywanych równolegle (execute packets),
  - dekodowanie kolejnych łańcuchów instrukcji i przekazywanie ich odpowiednim jednostkom w celu realizacji; dekodowanie każdego łańcucha instrukcji trwa jeden cykl zegarowy i dopóki nie zostaną zdekodowane wszystkie łańcuchy tworzące linię, wstrzymuje się pobieranie kolejnych linii instrukcji,
- wykonanie instrukcji — ta część potoku zawiera pięć etapów[8] przetwarzania E1-E5. Wykonywanie większości instrukcji kończy się jednak już po pierwszym etapie tak, jak ilustruje to tabela 4. Tabela uwzględnia:
  - result latency — liczbę cykli zegarowych niezbędnych dla ukończenia instrukcji,

[8] Zmiennoprzecinkowy procesor TMS320C6701 o architekturze VelociTI potrzebuje nawet do dziesięciu etapów w celu zakończenia instrukcji arytmetycznych podwójnej precyzji.

- delay slots — opóźnienie — liczbę łańcuchów instrukcji, następujących po zadanej instrukcji X, dla których wynik działania instrukcji X jest jeszcze niedostępny; opóźnienie jest zawsze równe liczbie cykli zegarowych niezbędnych do ukończenia instrukcji minus jeden,
- functional unit latency — czas, liczony w cyklach zegarowych, po upływie którego jednostka realizująca instrukcję może podjąć wykonywanie następnej instrukcji; dla procesora TMS320C6201 czas ten zawsze wynosi jeden, co oznacza, że w każdym cyklu zegarowym każda jednostka może rozpocząć realizację kolejnej instrukcji[9].

Tablica 4 wskazuje, że największe opóźnienie występuje podczas realizacji instrukcji skoku. W istocie instrukcja skoku realizowana jest już w pierwszym cyklu E1 etapu wykonania i powoduje zmianę adresu linii instrukcji, która będzie pobierana. Potok instrukcji ani nie jest wstrzymywany, ani nie jest unieważniany. Powoduje to, że jeszcze pięć łańcuchów instrukcji (execute packets) pobranych po instrukcji skoku będzie wykonanych zanim nastąpi wykonanie pierwszej instrukcji wskazywanej przez argument instrukcji skoku. Jednym słowem procesor TMS320C6201 dysponuje jedynie instrukcją skoku z opóźnieniem, a opóźnienie wynosi pięć cykli zegarowych (instrukcji, łańcuchów instrukcji). W każdym łańcuchu instrukcji mogą pojawić się dwie instrukcje skoku — jedna wykonywana przez jednostkę S1, a druga przez jednostkę S2. Procesor zachowa się w sposób przewidywalny jedynie wtedy, gdy obie instrukcje są instrukcjami warunkowymi, a warunki wzajemnie się wykluczają.

Tablica 4

Czas realizacji instrukcji procesora TMS320C6201

Instrukcja	Result latency	Delay slots	Functional unit latency
Instrukcje mnożenia: MPY, SMPY	2	1	1
Instrukcja ładowania danych: LD(B,H,W,D)	5	4	1
Instrukcja skoku: B	6	5	1
Wszystkie pozostałe instrukcje	1	0	1

Instrukcja pobrania danych z pamięci (LD) jest jedyną instrukcją procesora TMS320C6201, która wymaga na etapie realizacji pięciu cykli zegarowych E1-E5. W etapie E1 obliczany jest adres argumentu; w etapie E2 adres przekazywany jest do podsystemu pamięci; w etapie E3 procesor sprawdza, czy argument znajduje się w pamięci notatnikowej,

[9] Czas ten może być większy dla innych procesorów należących do rodziny o architekturze VelociTI.

a jeżeli nie, to cały potok instrukcji wstrzymywany jest w celu realizacji dostępu do zewnętrznej pamięci RAM; w etapie E4 dane przekazywane są z podsystemu pamięci do właściwej jednostki centralnej;w etapie E5 następuje zapisanie danych w rejestrze. Ze względu na opisany mechanizm dla czterech instrukcji (execute packets) następujących po instrukcji LD dane pobierane przez instrukcję LD nie są jeszcze dostępne. Natomiast modyfikacje rejestru służącego do adresowania widoczne są już dla następnej instrukcji po instrukcji LD. Instrukcja zapisywania w pamięci (ST) realizowana jest w ciągu trzech cykli zegarowych E1-E3. W etapie E1 obliczany jest adres argumentu; w etapie E2 adres przekazywany jest do podsystemu pamięci, w etapie E3 dane przekazywane są do podsystemu pamięci. Dzięki odpowiedniej konstrukcji podsystemu pamięci wydaje się, że kombinacja instrukcji ST i LD wykonywana jest zawsze tak, jak gdyby nie istniały jakiekolwiek opóźnienia w ich wykonaniu. Możliwe są bowiem tylko trzy sytuacje [14]:

- bezpośrednio po instrukcji ładowania (LD) z komórki pamięci o adresie X występuje instrukcja składowania (ST) do tej samej komórki X. Kiedy instrukcja ST znajduje się na etapie E3 wykonania i zapisywane dane przekazywane są do podsystemu pamięci, instrukcja LD jest na etapie wykonywania E4 i odczytywane dane dostarczane są do jednostki centralnej. W rezultacie stare dane z komórki o adresie X są ładowane do rejestru, a nowe dane (pochodzące z procesora) zapisywane do komórki X,
- bezpośrednio po instrukcji składowania (ST) w komórce pamięci o adresie X występuje instrukcja ładowania (LD) z tej samej komórki X. Kiedy instrukcja ST znajduje się na etapie E3 wykonania i nowe dane przekazywane są do podsystemu pamięci, instrukcja LD jest na etapie wykonywania E2 i do podsystemu pamięci przekazywany jest dopiero adres argumentu instrukcji LD. W rezultacie nowe dane zapisywane są w komórce o adresie X i te same dane ładowane są do rejestru procesora,
- instrukcja LD ładowania z komórki o adresie X jest wykonywana równolegle (połączona w łańcuch) z instrukcją składowania ST w tej samej komórce. W rezultacie wykonania tych instrukcji stara wartość z pamięci zapisana zostaje w rejestrze procesora, a nowa wartość (pochodząca z procesora) zapisana w komórce X.

1.3. Przerwania

Procesor TMS320C6201 ma możliwość reakcji na 14 przerwań. Są to: zerowanie procesora — RESET, przerwanie niemaskowane — NMI, 12 przerwań maskowanych INT4-INT15. Źródłem dwóch pierwszych przerwań są zawsze sygnały doprowadzone do wyprowadzeń zewnętrznych procesora. Część przerwań maskowanych wykorzystywanych jest przez urządzenia peryferyjne zintegrowane wraz z procesorem, zaś pozostałe mogą pochodzić od urządzeń zewnętrznych. Każdemu przerwaniu przypisany jest priorytet. Najbardziej uprzywilejowane jest zerowanie procesora, następnie przerwanie niemaskowane i przerwania maskowane, dla których stopień uprzywilejowania maleje wraz ze wzrostem numeru, tzn. najbardziej uprzywilejowanym wśród przerwań maskowanych jest przerwanie INT4, a najmniej uprzywilejowanym jest przerwanie INT15.

Zerowanie procesora odbywa się niezależnie od jego stanu wewnętrznego. Obsługę przerwania niemaskowanego może uniemożliwić jedna z poniższych okoliczności:

- aktualnie obsługiwane jest przerwanie niemaskowane, które pojawiło się wcześniej,
- właśnie wykonana została instrukcja skoku. Procesor blokuje automatycznie system przerwań na pięć taktów zegarowych[10], aby prawidłowo zakończyć wykonywanie instrukcji, które znalazły się już w potoku[11].

Obsługę przerwania maskowanego może uniemożliwić jedna z poniższych okoliczności:

- aktualnie obsługiwane jest przerwanie niemaskowane, które pojawiło się wcześniej,
- właśnie wykonana została instrukcja skoku i system przerwań procesora jest zablokowany na okres pięciu taktów zegarowych,
- programowo zablokowano obsługę wszystkich przerwań maskowanych przez wyzerowanie bitu GIE (ang. global interrupt enable) w rejestrze sterującym pracą procesora CSR (ang. control status register),
- programowo zamaskowano obsługę zgłaszanego przerwania przez wpisanie do rejestru masek IER (ang. interrupt enable register) zera na pozycji odpowiadającej zgłaszanemu przerwaniu.

Zgłoszenie przerwania polega na przejściu zewnętrznego sygnału zgłoszenia przerwania ze stanu niskiego do stanu wysokiego. Procesor wykrywa pojawienie się sygnału żądania przerwania w czasie nie przekraczającym trzech cykli zegarowych. W tym czasie sprawdza

---

[10] Nawet jeżeli wykonanie instrukcji skoku warunkowego nie doszło do skutku.

[11] Przerwanie którejkolwiek z pięciu instrukcji (execute packets) następujących po instrukcji skoku powoduje, że ślad zachowywany przez procesor podczas przechodzenia do procedury obsługi przerwania nie pozwala na odtworzenie stanu potoku przed przerwaniem.

się, czy może zostać podjęte wykonywanie procedury obsługi przerwania, tzn. czy system przerwań jest aktywny, czy przerwanie nie jest zamaskowane itd. Jeżeli procedura obsługi zgłaszanego przerwania może być podjęta, to:

- blokowane jest przyjmowanie dalszych przerwań,
- unieważniane są wszystkie instrukcje znajdujące się w potoku, które nie doszły jeszcze do etapu wykonania,
- do rejestru IRP (ang. interrupt return pointer) wpisywany jest adres najstarszej z unieważnionych instrukcji. Powrót z procedury obsługi przerwania polega na wykonaniu skoku do instrukcji wskazywanej przez rejestr IRP,
- w rejestrze IFR (ang. interrupt flag register) ustawiany jest bit odpowiadający zgłaszanemu przerwaniu.

W następnym cyklu zegarowym:

- w potoku instrukcji wymuszany jest skok do procedury obsługi przerwania,
- na zewnątrz uaktywniany jest sygnał IACK sygnalizujący urządzeniom zewnętrznym, że procesor podjął obsługę zgłoszonego przerwania.

Jeszcze jeden cykl zegarowy później zerowana jest flaga w rejestrze IFR odpowiadająca obsługiwanemu przerwaniu. W rezultacie od chwili zgłoszenia przerwania do chwili rozpoczęcia etapu E1 pierwszej instrukcji procedury obsługi przerwania upływa 11 cykli zegarowych, czyli zaledwie 55 ns. Procesor nie wspomaga sprzętowo zagnieżdżania przerwań.

Procedury obsługi przerwań tworzą tablicę wektorów przerwań. Tablica wektorów przerwań może zostać umieszczona (prawie) gdziekolwiek w przestrzeni adresowej programu. Każdemu z przerwań odpowiada w tablicy jedna 256-bitowa procedura obsługi tworząca jedną linię instrukcji (fetch packet). Jeżeli tak krótka procedura obsługi przerwania jest niewystarczająca, należy wykonać skok do właściwej procedury obsługi przerwania. Należy pamiętać, że każdy skok, w tym skok realizujący powrót z procedury obsługi przerwania, wykonywany jest z opóźnieniem.

Blokowanie systemu przerwań po wykonaniu instrukcji skoku powoduje, że ciasne pętle programowe, których rozmiar nie przekracza pięciu instrukcji (execute packets), nie mogą być przerwane. Podczas realizacji takiej pętli po wykonaniu instrukcji skoku potok instrukcji zawiera zawsze następną instrukcję skoku. W programach realizujących algorytmy cyfrowego przetwarzania sygnałów takie ciasne i czasochłonne pętle pojawiają się bardzo często. Aby

umożliwić obsługę przerwań, pętlę taką należy sztucznie powiększyć tak, aby zawierała przynajmniej sześć instrukcji. W tym celu można posłużyć się wieloetapową instrukcją NOP. Argumentem tej instrukcji jest liczba cykli zegarowych potrzebnych na jej realizację.

Architektura procesora nakłada pewne ograniczenia na sposób pisania programów, które mogą być przerwane w celu obsłużenia przerwania. Przykładem „patologicznego" pod tym względem programu może być ten zamieszczony w pierwszej części tablicy 5. Instrukcja LD pobiera z pamięci dane i umieszcza je w rejestrze A1. Ze względu na opisane wcześniej opóźnienia w realizacji instrukcji LD dane te będą dostępne dopiero dla instrukcji mnożenia MPY. Tymczasem programista skorzystał ze „starej" zawartości rejestru A1. W obecności przerwań program może działać zupełnie inaczej. Załóżmy, że przerwanie nastąpi po przekazaniu instrukcji LD do wykonania, a przed rozpoczęciem wykonywania instrukcji dodawania. Po powrocie z procedury obsługi przerwania w rejestrze A1 będzie już nowa wartość pobrana przez instrukcję LD. Poprawna postać programu znajduje się w drugiej części tablicy 5. Instrukcja LD zapisuje pobrane dane do rejestru A6 i ewentualne przerwanie nie wpływa na sposób działania programu.

Tablica 5

Wpływ przerwań na poprawność działania programu realizowanego przez procesor TMS320C6201

```
;
; Program działający wadliwie w obecności przerwań
;

   LDW     .D1   *A0,A1       ; A1 <- [A0]
   ADD     .L1    A1,A2,A3    ; A3 <- A2 + A1(stare)
   NOP            3           ; odczekaj trzy cykle zegarowe
   MPY     .M1    A1,A4,A5    ; A5 <- A4 * A1(nowe)
    .
    .
    .
;
; Program działający poprawnie w obecności przerwań
;

   LDW     .D1   *A0,A6       ; A6 <- [A0]
   ADD     .L1    A1,A2,A3    ; A3 <- A2 + A1
   NOP            3           ; odczekaj trzy cykle zegarowe
   MPY     .M1    A6,A4,A5    ; A5 <- A4 * A6
```

1.4. Pamięć

Przestrzeń adresowa programu oraz przestrzeń adresowa danych są w układzie TMS320C6201 rozdzielone. Razem z jednostką centralną zintegrowano 64 Kbajtów pamięci programu. Pamięć ta może pomieścić 16 K instrukcji albo 2 K linii programu (fetch packets).

Jeżeli wewnętrzna pamięć programu jest zdecydowanie za mała dla realizacji zadań postawionych przed systemem, można ją skonfigurować jako pamięć notatnikową [18].

Układ TMS320C6201 zawiera także 64 Kbajty pamięci danych. Pamięć danych podzielona jest na dwa 32 K bajtowe bloki umieszczone „jeden za drugim" w przestrzeni adresowej danych. Każdy z bloków podzielony jest na cztery banki pamięci o długości słowa 16 bitów i pracujące z przeplotem [2, 1]. W ten sposób w większości przypadków możliwy jest jednoczesny dostęp do pamięci dwóch „uprawnionych" do tego jednostek wykonawczych D1 i D2 (oraz kontrolera DMA). Konflikt powstaje jedynie wtedy, gdy obie jednostki w tym samym cyklu zegarowym usiłują się dostać do tego samego banku pamięci, co ilustruje tablica 6. W tym przypadku przetwarzanie zostaje zatrzymane na jeden dodatkowy takt zegarowy tak, aby umożliwić dwa kolejne dostępy do tego samego banku pamięci. Jeżeli dojdzie do konfliktu pomiędzy instrukcją ładowania LD i instrukcją składowania ST, zawsze najpierw wykonywana jest instrukcja ładowania LD.

Dostęp do zewnętrznej pamięci programu i do zewnętrznej pamięci danych realizowany jest za pośrednictwem specjalizowanego układu sterującego o nazwie EMIF (ang. external memory interface). Pozwala on na:

- dołączenie do układu procesora szerokiej gamy pamięci RAM,
- szczytową szybkość przesyłania danych pomiędzy pamięcią zewnętrzną a procesorem równą 800 Mbajtów/s,
- możliwość zaadresowania do 52 Mbajtów pamięci. Pamięci podzielone są na cztery bloki (ang. chip-enable space). Trzy z nich mogą zawierać do 16 Mbajtów pamięci, a czwarty 4 Mbajty pamięci,
- adresowanie pojedynczych bajtów,
- możliwość pobierania danych z pamięci ROM o organizacji ośmio- i szesnasto-bitowej,
- współdzielenie pamięci z innymi procesorami.

Tablica 6

Schemat konfliktów podczas dostępu do wewnętrznej pamięci danych procesora TMS320C6201

	D1	bajt								półsłowo				słowo	
D2	LSB adresu	000	001	010	011	100	101	110	111	000	010	100	110	000	100
Bajt	000	■	■							■				■	
	001	■	■							■				■	
	010			■	■						■			■	
	011			■	■						■			■	
	100					■	■					■			■
	101					■	■					■			■
	110							■	■				■		■
	111							■	■				■		■
półsłowo	000	■	■							■				■	
	010			■	■						■			■	
	100					■	■					■			■
	110							■	■				■		■
słowo	000	■	■	■	■					■	■			■	
	100					■	■	■	■			■	■		■

Zacieniowane pole oznacza konflikt

Układ EMIF pozwala na dołączanie różnych typów pamięci RAM bez konieczności stosowania dodatkowych układów logicznych. W szczególności dotyczy to:

- synchronicznej, statycznej pamięci RAM pracującej w trybie zakładkowania (SBBSRAM). Częstotliwość zegara taktującego może być równa częstotliwości zegara taktującego procesor lub dwukrotnie niższa,
- synchronicznej dynamicznej pamięci RAM (SDRAM) taktowanej zegarem o częstotliwości równej połowie częstotliwości zegara procesora. Układ EMIF zapewnia właściwe odświeżanie pamięci,
- wszelkiego rodzaju pamięci asynchronicznych SRAM, ROM, flash itd. Istnieje możliwość programowej zmiany liczby cykli oczekiwania na gotowość pamięci.

Najmniejszą adresowalną porcją pamięci jest jeden bajt. Z pamięci pobierać można bajty, półsłowa (2 bajty) i słowa (4 bajty). Adres każdego półsłowa musi być parzysty, a adres każdego słowa musi być podzielny przez cztery. Do pamięci mogą sięgać tylko dwie instrukcje: ładowanie LD i składowanie ST. Instrukcje te mogą być wykonane tylko przez jednostkę D1 lub D2. Dostępne jest tylko adresowanie pośrednie przez rejestr z indeksowaniem. Źródłem indeksu może być rejestr wewnętrzny procesora, krótka 5-bitowa stała lub stała 16-bitowa. Rejestr służący do adresowania może być inkrementowany (dekrementowany) przed lub po wykonaniu dostępu do pamięci. Adresowanie pośrednie za pośrednictwem rejestrów A4-A7 lub B4-B7 pozwala na zorganizowanie buforów kołowych.

1.5. Peryferia

Procesory z rodziny TMS320C62xx integrowane są wraz z pewną liczbą urządzeń peryferyjnych. Poniżej przedstawiony jest krótki ich przegląd. Jednym z nich jest kontroler DMA [9]. Kontroler DMA pozwala na przeprowadzenie transmisji danych między dwoma obszarami pamięci lub pomiędzy pamięcią a urządzeniem zewnętrznym bez konieczności angażowania procesora centralnego. Transmisja może odbywać się z szybkością dochodzącą do 800 Mbajtów/s. Dostępne są cztery programowalne kanały transmisji oraz kanał piąty przeznaczony dla obsługi portu HIP. Każdy kanał posiada swój własny zestaw rejestrów zawierających adres źródłowy, adres przeznaczenia oraz licznik przesłań. Zaimplementowane schematy indeksowania pozwalają na przesyłanie danych pobieranych z komórki o ustalonym adresie oraz przesyłanie danych tworzących tablicę jedno- lub dwuwymiarową. Przesyłanie danych może być wyzwalane przez zdarzenia, których źródłem jest urządzenie zewnętrzne lub urządzenie zintegrowane razem z procesorem. Sygnalizuje jednostce centralnej, za pomocą przerwania, zakończenie transmisji lub pojawienie się błędów. Każdy kanał może być zaprogramowany w trybie autoinicjalizacji tak, aby podjął następną transmisję bez konieczności interwencji ze strony jednostki centralnej. Może także obsługiwać transmisję półdupleksową — od pamięci do urządzenia zewnętrznego i w kierunku odwrotnym. Podczas obsługi procedury zerowania procesora — RESET — kontroler DMA może zostać wykorzystany dla przesłania programu z zewnętrznej pamięci ROM do wewnętrznej pamięci RAM.

Układ TMS320C6201 posiada zintegrowane dwa synchroniczne porty szeregowe podobne do tych, które firma stosuje w układach TMS320C2x/C5x/C54x. Cechą szczególną procesora TMS320C6201 jest możliwość obsługiwania transmisji z portu szeregowego do pamięci oraz w kierunku odwrotnym przez kontroler DMA. Port może prowadzić transmisję z szybkością

do 100 Mbitów/s. Chodzi tu, oczywiście, o dwa strumienie danych każdy o wymienionej przepływności — jeden to dane transmitowane od procesora do urządzenia zewnętrznego, drugi strumień stanowią dane transmitowane do procesora. Port jest portem wielokanałowym przystosowanym do współpracy z układami obsługującymi protokoły transmisji T1, E1 oraz MVIP.

Układ timera — programowanego 32-bitowego licznika — może być taktowany zegarem zewnętrznym albo zegarem powstałym w wyniku podziału częstotliwości zegara taktującego procesor przez cztery. Ta ostatnia możliwość pozwala na odmierzanie odcinków czasu o długości przekraczającej jedną minutę z rozdzielczością 20 ns. W zależności od sposobu zaprogramowania na wyjściu timera generowany jest pojedynczy impuls lub fala prostokątna. O zakończeniu odliczania czasu timer może informować procesor generując żądanie przerwania.

Kolejnym urządzeniem peryferyjnym jest port dla komunikacji z jednostką nadrzędną HIP (ang. host interface port). Jest to 16-bitowy układ pozwalający na dostęp do pamięci wewnętrznej procesora TMS320C6201 przez procesor nadrzędny. Port może być taktowany zegarem o częstotliwości do 50 MHz i prowadzić transmisję z prędkością do 100 Mbajtów/s. Konstrukcja portu pozwala na użycie praktycznie dowolnego mikroprocesora lub mikrokontrolera w charakterze jednostki nadrzędnej bez konieczności używania rozbudowanych logicznych układów pośredniczących.

Procesor TMS320C6201 jest także wyposażony w układ oszczędzania energii. Procesor można programowo wprowadzić w stan o niskim poborze energii. Istnieją trzy takie stany:

- power down 1: Zatrzymana zostaje jednostka centralna, natomiast wszystkie urządzenia peryferyjne zintegrowane wewnątrz procesora działają. Wyjście z tego stanu następuje w wyniku pojawienia się niezamaskowanego żądania przerwania. Przerwanie może pochodzić tak od urządzenia zewnętrznego, jak i od urządzenia zintegrowanego wraz z jednostką centralną,
- power down 2: Zatrzymana zostaje jednostka centralna oraz wszystkie urządzenia peryferyjne z wyjątkiem pętli PLL. Wyjście z tego stanu możliwe jest tylko przez zerowanie procesora sygnałem RESET,
- power down 3: Stan taki sam jak power down 2, lecz pętla PLL procesora także nie działa — nie są generowane żadne sygnały zegarowe.

1.6. Przykład programowania

Programowanie maszyny cyfrowej o takim stopniu przetwarzania równoległego, jaki prezentuje układ TMS320C6201, różni się istotnie od programowania konwencjonalnego mikroprocesora lub mikroprocesora superskalarnego. Dokładne omówienie tego zagadnienia wychodzi poza ramy tego artykułu, a zainteresowani Czytelnicy muszą zostać odesłani do literatury [19, 20]. Tutaj przedstawiona zostanie jedynie jedna z wielu stosowanych technik o angielskiej nazwie software pipelining, co w swobodnym tłumaczeniu na język polski oznacza programowe zakładkowanie. Niech przykładowy algorytm oblicza iloczyn skalarny dwóch wektorów[12]:

$$c = \vec{a} \cdot \vec{b} = \sum_{i=0}^{N-1} a_i \cdot b_i$$

Składowe pierwszego wektora zapisane są w tablicy ***a***, natomiast składowe drugiego wektora zapisane są w tablicy ***b***. Iloczyn skalarny to suma iloczynów kolejnych składowych obu wektorów. W tablicy 7 przedstawiony jest program napisany w asemblerze procesora TMS320C6201 obliczający taki iloczyn w sposób całkowicie pomijający możliwości przetwarzania równoległego. Obliczanie iloczynu odbywa się w pętli. Pierwsza instrukcja programu inicjuje w rejestrze A1 licznik pętli. Druga instrukcja zeruje akumulator A7. Od tego miejsca zaczyna się właściwe „ciało" pętli. Pierwsze dwie instrukcje pętli ładują do procesora po jednej składowej każdego z mnożonych wektorów. Aby potrzebne dane znalazły się w rejestrach wewnętrznych procesora, trzeba odczekać dodatkowo 4 cykle zegarowe. Wtedy można przemnożyć składowe. Na wynik trzeba poczekać dodatkowo przez jeden takt zegarowy. Dalej następuje akumulacja wyniku mnożenia w rejestrze A7, dekrementacja licznika pętli zawartego w rejestrze A1 i skok warunkowy na początek pętli. Nim jednak skok zostanie wykonany, upłynie pięć taktów zegarowych, które procesor spędzi na wykonywaniu instrukcji NOP. W ten sposób każdy przebieg pętli wymaga szesnastu taktów zegarowych. Dla wektorów o stu składowych daje to razem z dwiema początkowymi instrukcjami programu 1602 cykle zegarowe.

---

[12] Zauważmy od razu, że obliczanie iloczynu skalarnego jest bardzo zbliżone do obliczania odpowiedzi filtru FIR.

Tablica 7

Obliczanie iloczynu skalarnego – przetwarzanie szeregowe

```
;
;  Liczba cykli = 16*N+2; N=100 => Liczba cykli = 1602
;

     MVK  .S1 100, A1          ; A1 <- N == liczba iteracji
     ZERO .L1 A7               ; A7 <- 0
LOOP:
     LDH  .D1 *A4++,A2         ; A2 <- [A4] == a[i] ; A4 <- A4 + 2
     LDH  .D1 *A3++,A5 ;       ; A5 <- [A5] == b[i] ; A5 <- A5 + 2
     NOP   4                   ; trwa ładowanie!
     MPY  .M1 A2,A5,A6         ; A6 <- a[i] * b[i]
     NOP                       ; trwa mnożenie!
     ADD  .L1 A6,A7,A7         ; A7 <- A7 + a[i] * b[i]
     SUB  .S1 A1,1,A1          ; A1 <- A1 - 1
[A1] B    .S2 LOOP             ; jeżli to nie koniec to skok
     NOP 5                     ; wyczyść potok
     ; teraz skok
```

Czas wykonywania przedstawionego programu można istotnie zmniejszyć uwzględniając możliwości przetwarzania równoległego, jakie stwarza procesor TMS320C6201, oraz uwzględniając opóźnienie, z jakim realizowana jest instrukcja skoku. Przykład ulepszonego programu przedstawia tablica 8. Składowe obu wektorów ładowane są jednocześnie parami — składowa o indeksie parzystym + składowa o indeksie nieparzystym — z wykorzystaniem instrukcji ładowania słowa 32-bitowego LDW. Zmniejsza to liczbę instrukcji w pętli o jeden. Składowe o parzystych indeksach przetwarzane są przez pierwszą grupę jednostek wykonawczych, a składowe o nieparzystych indeksach przez grupę drugą. Zmniejsza to liczbę przebiegów pętli o połowę. Liczba instrukcji w pętli zmniejszyła się także, ponieważ w czasie oczekiwania na załadowanie składowych mnożonych wektorów dokonano dekrementacji licznika pętli i zainicjowano wykonanie instrukcji skoku. W ten sposób zaoszczędzono dalszych 7 instrukcji NOP. W sumie daje to pętlę złożoną z ośmiu instrukcji. Do przemnożenia dwóch wektorów o stu składowych potrzeba 402 taktów zegarowych[13] — prawie cztery razy mniej niż w poprzednim programie.

[13] Z uwzględnieniem inicjacji pętli i końcowej akumulacji.

Tablica 8

Obliczanie iloczynu skalarnego – przetwarzanie równoległe

```
;
;  Liczba cykli = 4*N+2; N=100 => Liczba cykli = 402
;

     MVK  .S1 50,A1          ; A1 <- 50 == N/2
||   ZERO .L1 A7             ; A7 <- 0
||   ZERO .L2 B7             ; B7 <- 0
LOOP:
     LDW  .D1 *A4++,A2       ; A2 <- a[i] a[i+1] ; A4 <- A4 + 4
||   LDW  .D2 *B4++,B2       ; B2 <- b[i] b[i+1] ; B4 <- B4 + 4
     SUB  .S1 A1,1,A1        ; A1 <- A1 - 1
[A1] B    .S2 LOOP           ; jeżli to nie koniec to skocz
     NOP   2                 ; trwa ładowanie składowych !
     MPY  .M1X A2,B2,A6      ; A6 <- a[i] * b[i]
||   MPYH .M2X A2,B2,B6      ; B6 <- a[i+1] * b[i+1]
     NOP                     ; trwa mnożenie
     ADD  .L1 A6,A7,A7       ; A7 <- A7 + A6
||   ADD  .L2 B6,B7,B7       ; B7 <- B7 + B6
; tu następuje skok
     ADD .L1X A7,B7,A4       ; iloczyn skalarny = A4 <- A7 + B7
```

Program przedstawiony w tablicy 8 podczas jednego przebiegu pętli dokonuje równolegle mnożenia składowych wektorów dla dwóch kolejnych indeksów oraz akumulacji wyników. Zauważmy, że każda instrukcja pętli wykonywana jest przez inną jednostkę wykonawczą. Można zatem wszystkie instrukcje wykonać jednocześnie łącząc je w jeden łańcuch (execute packet). W tym przypadku ze względu na zależności między argumentami poszczególnych instrukcji realizowane będą elementy kilku pętli jednocześnie. Odpowiedni program pokazany jest w tablicy 9. W chwili gdy wykonywany jest n-ty raz łańcuch instrukcji wskazywany przez etykietę LOOP realizowana jest:

- akumulacja iloczynu składowych o indeksie 2n-12 oraz 2n-11,
- mnożenie składowych o indeksie 2n-10 ***a[2n-10]·b[2n-10]*** oraz składowych o indeksie 2n-9 ***a[2n-9]·b[2n-9]***,
- dekrementacja licznika pętli oraz skok do etykiety LOOP,
- ładowanie dwóch kolejnych składowych pierwszego wektora, tzn. składowej o indeksie 2n — ***a[2n]*** — oraz składowej o indeksie 2n++1 — ***a[2n+1]***,
- ładowanie dwóch kolejnych składowych drugiego wektora, tzn. składowej o indeksie 2n — ***b[2n]*** — oraz składowej o indeksie 2n+1 — ***b[2n+1]***.

Licznik pętli dekrementowany jest od wartości N/2+1 (N — liczba składowych mnożonych wektorów) do wartości 0. W tym momencie przestaje być wywoływana pętla LOOP, natomiast akumulatory zawierają już iloczyny składowych o indeksie N-12 oraz N-11. Pętla LOOP jest jednak wykonywana dalej dopóty, dopóki tworzące ją instrukcje znajdują się w potoku. Oznacza to, że pętla zostanie wykonana jeszcze pięć razy tak, aby ostatecznie

dokonać akumulacji iloczynów składowych o indeksach N-2 i N-1. Jeżeli uwzględnimy instrukcje zaczynające program, których zadaniem jest zerowanie zmiennych oraz wypełnienie potoku instrukcji kopiami pętli LOOP, oraz ostatnią instrukcję obliczającą iloczyn skalarny, to do przemnożenia dwóch wektorów o stu składowych każdy potrzeba zaledwie 57 taktów zegarowych. Siedem razy mniej niż w przypadku przetwarzania równoległego i 28 razy mniej niż w przypadku przetwarzania szeregowego. Jednak nie ma róży bez kolców — tak ciasna pętla programowa nie może zostać przerwana w celu obsłużenia przerwania[14].

Tablica 9

Obliczanie iloczynu skalarnego – programowe zakładkowanie

```
;
;  Liczba cykli = N/2+7; N=100 => Liczba cykli = 57
;
        B     .S2 LOOP            ; skok do pętli
||      MVK   .S1 51,A1           ; A1 <- N/2 + 1 == 51
        B     .S2 LOOP            ; skok do pętli
        B     .S2 LOOP            ; skok do pętli
||      ZERO  .L1 A7              ; suma0 == A7 <- 0
||      ZERO  .L2 B7              ; suma1 == B7 <- 0
        B     .S2 LOOP            ; skok do pętli
||      ZERO  .L1 A6              ; wynik_mnożenia0 == A6 <- 0
||      ZERO  .L2 B6              ; wynik_mnożenia1 == B6 <- 0
        B     .S2 LOOP            ; skok do pętli
||      ZERO  .L1 A2              ; mnożna  == A2  <- 0
||      ZERO  .L2 B2              ; mnożnik == B2  <- 0
LOOP:
        ADD   .L1 A6,A7,A7        ; sum0 += a[2n-12] * b[2n-12]
||      ADD   .L2 B6,B7,B7        ; sum1 += a[2n-11] * b[2n-11]
||      MPY   .M1X A2,B2,A6       ; a[2n-10] * b[2n-10]
||      MPYH  .M2X A2,B2,B6       ; a[2n-9] * b[2n-9]
||[A1]  ADD   .S1 -1,A1,A1        ; dekrementuj licznik pętli
||[A1]  B     .S2 LOOP           ; skok do pętli
||      LDW   .D1 *A4++,A2        ; ładuj a[2n] & a[2n+1] z pamięci
||      LDW   .D2 *B4++,B2        ; ładuj b[2n] & b[2n+1] z pamięci
; tu następuje skok
        ADD .L1X A7,B7,A4         ; iloczyn_skalarny = suma0 + suma1
```

2. Supermikrokontroler — projekt TriCore

Współczesna technologia elektronowa coraz częściej daje możliwość konstrukcji całych urządzeń cyfrowych w postaci pojedynczej struktury scalonej. Układ taki składa się zazwyczaj z:

[14] Istnieje jeszcze jeden mankament — realizując algorytm sięgamy w pamięci poza tablicę a[i] oraz poza tablicę b[i].

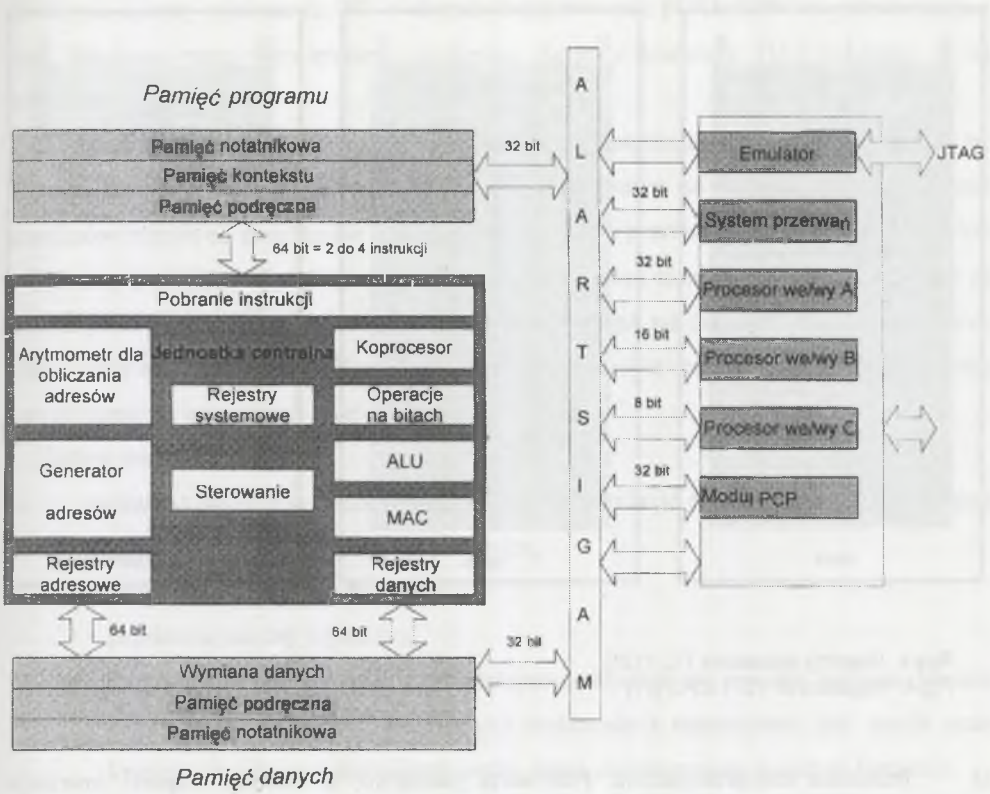
- szybkiego, 32-bitowego mikrokontrolera konstrukcyjnie przystosowanego do przetwarzania liczb z szybkością porównywalną z szybkością procesora sygnałowego,
- najróżniejszych pamięci półprzewodnikowych, np. DRAM, SRAM, ROM, flash itp,
- układów logicznych charakterystycznych dla danej aplikacji (ang. ASIC — application specific integrated circuit).

Konfiguracja taka pozwala na programową realizację wielu zadań, które dotychczas wykonywano sprzętowo. Na ogół zmniejsza to koszt projektu, ponieważ można wykorzystać go w całości lub w części w innych projektach. Pozwala to na eliminację ewentualnych błędów poprzez wymianę oprogramowania, nawet jeżeli urządzenie jest już u klienta. Daje klientowi możliwość instalacji nowej wersji oprogramowania o udoskonalonych własnościach użytkowych. Na przykład w przypadku cyfrowego telefonu komórkowego całość przetwarzania cyfrowego może być realizowana przez jeden układ. W typowej obecnie konstrukcji stosowane są przynajmniej dwa układy. Procesor sygnałowy zaangażowany jest w celu kompresji i dekompresji sygnału mowy, kodowania i dekodowania danych cyfrowych, realizacji modulacji i demodulacji w paśmie podstawowym. Jednym słowem realizuje protokoły transmisji leżące w warstwie fizycznej modelu ISO/OSI. Natomiast mikrokontroler realizuje protokoły transmisji należące do warstwy połączeniowej oraz obsługuje interfejs użytkownika [13]. Te same zadania może realizować jeden szybki, 32-bitowy mikrokontroler o charakterystyce zbliżonej do charakterystyki procesora sygnałowego [11]. Przykładem takiego układu jest procesor TC-1 — pierwszy z rodziny układów o architekturze TriCore, opracowanej przez firmę Siemens Microelectronics Inc.[15] Cechy charakterystyczne tego układu to:

- architektura będąca hybrydą mikrokontrolera i stałoprzecinkowego procesora sygnałowego,
- szybkie przełączanie zadań. Procesor TC-1 może przejść od realizacji zadania charakterystycznego dla mikrokontrolera do zadania charakterystycznego dla procesora sygnałowego w czasie zaledwie dwóch cykli zegarowych!,
- możliwość integracji razem z pamięcią półprzewodnikową o dużej pojemności, co podnosi wydajność systemu, jego niezawodność oraz zmniejsza zużycie energii[16],

[15] Od 1 kwietnia 1999 firma Siemens SMI jest wydzielona z koncernu Siemens i nosi nazwę Infineon Inc.

[16] Oczywiście, w stosunku do systemu, którego pamięć umieszczona została w innym układzie scalonym.



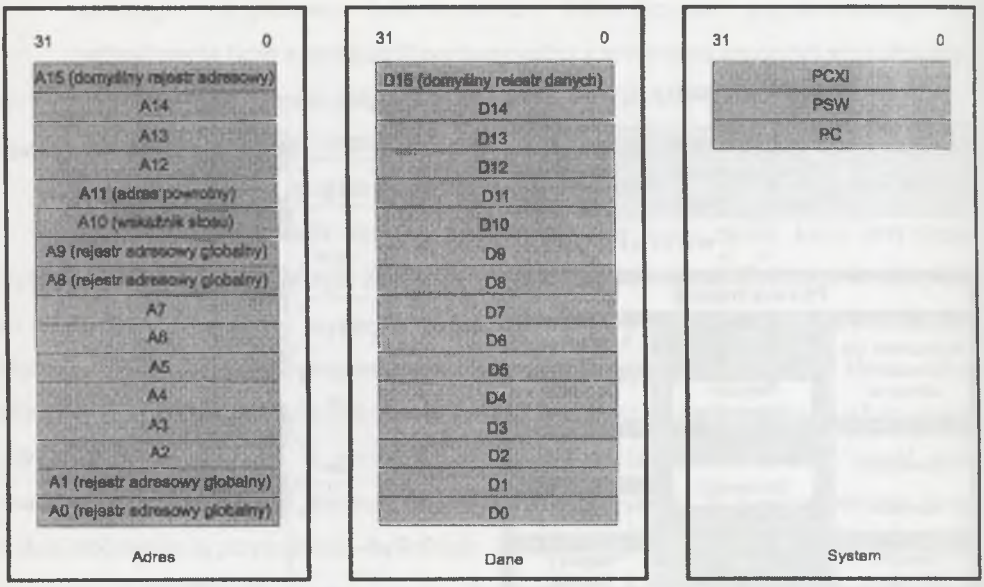
Rys.3. Schemat blokowy procesora TC-1 [21]
Fig.3. Block diagram of TC-1 CPU [21]

- możliwość dołączania zewnętrznych urządzeń peryferyjnych praktycznie bez pośredniczących układów logicznych,
- lista instrukcji zakodowana w postaci słów 16-bitowych i 32-bitowych tak, aby typowe programy były możliwie małe.

Lista instrukcji oraz architektura procesora zbliżona jest do architektury typu RISC (ang. reduced instuction set procesor), co pozwala na konstrukcję efektywnych kompilatorów języka C/C++.

2.1. Jednostka centralna procesora TC-1

Schemat blokowy procesora TC-1 przedstawiony jest na rysunku 3. Jest to procesor o architekturze Harvard, z magistralą danych rozdzieloną od magistrali programu. Procesor jest układem superskalarnym zawierającym trzy jednostki wykonawcze:



Rys.4. Rejestry procesora TC-1 [21]
Fig.4. Registers of TC-1 CPU [21]

- jednostka stałoprzecinkowa. Przetwarza potokowo, w czterech etapach, instrukcje arytmetyczno-logiczne. Przetwarzanie odbywa się z wykorzystaniem dwóch 16-bitowych układów mnożących i jednostki arytmetyczno-logicznej. W potoku tym w jednym takcie zegarowym można wykonać dwa mnożenia połączone z akumulacją. Jednostka dysponuje interfejsem do zewnętrznego układu koprocesora, który może wykonywać np. operacje na liczbach zmiennoprzecinkowych,
- jednostka realizująca operacje ładowania z pamięci LD i składowania w pamięci ST. Wszystkie instrukcje arytmetyczne i logiczne używają w charakterze argumentów rejestrów wewnętrznych procesora. Dane z oraz do pamięci przenoszone są przez parę instrukcji LD/ST przetwarzanych w osobnym, czteroetapowym potoku,
- jednostka realizująca rozkazy skoku. Przetwarza rozkazy potokowo w dwóch etapach.

W ten sposób w każdym cyklu zegarowym może kończyć się realizacja nawet trzech instrukcji. W praktyce procesor taktowany zegarem 100 MHz, ma wydajność 130 MIPS [21].

Procesor TC-1 posiada trzydzieści dwa 32-bitowe rejestry wewnętrzne. Szesnaście z nich A0-A15 przeznaczonych jest do adresowania. Pozostałe szesnaście rejestrów D0-D15 to rejestry danych. Niektóre z tych rejestrów są wyróżnione. Na przykład D15 jest domyślnym rejestrem danych, natomiast rejestr A10 to wskaźnik stosu. Poza tym procesor zawiera trzy

32-bitowe rejestry systemowe: PC — licznik rozkazów oraz PCXI, PSW — przechowujące dane dotyczące stanu wewnętrznego procesora. Rejestry procesora TC-1 pokazane są na rysunku 4.

W zasadzie wszystkie instrukcje procesora TC-1 są instrukcjami 32-bitowymi. Jednak te, które są używane najczęściej, mają swoje wersje 16-bitowe, co pozwala na zmniejszenie rozmiarów typowych programów oraz podnosi szybkość przetwarzania — w jednym cyklu pobiera się i kieruje do przetwarzania cztery zamiast dwóch instrukcji. Krótkie instrukcje są instrukcjami dwuargumentowymi, a nie trójargumentowymi, tak jak ich 32-bitowe odpowiedniki. Często stosują argumenty domyślne — rejestry adresowe i rejestr danych. Ich argumenty natychmiastowe oraz adresy względne są krótkie.

Zestaw instrukcji obejmuje:

- instrukcje ładowania i składowania — jedyne instrukcje dokonujące wymiany danych między procesorem a pamięcią operacyjną,
- instrukcje arytmetyczno-logiczne:
  - przesłania między rejestrami,
  - dodawanie i odejmowanie liczb stałoprzecinkowych ze znakiem lub bez. Istnieje osobny zbiór instrukcji realizujących dodawanie z nasyceniem, gdy wynik przekroczy największą (najmniejszą) liczbę, którą można zapisać w danym formacie,
  - mnożenie dwóch 32-bitowych liczb stałoprzecinkowych połączone z dodawaniem (akumulacją). Mnożone liczby mogą być liczbami ułamkowym w formacie 1.31 lub liczbami całkowitymi,
  - instrukcje wspomagające programową realizację dzielenia liczb stałoprzecinkowych,
  - obliczanie wartości bezwzględnej, wartości bezwzględnej z różnicy, wybieranie większego lub mniejszego argumentu,
  - warunkowe instrukcje dodawania i odejmowania. Realizacja instrukcji uzależniona jest od wartości jednego z argumentów (rejestru). Jeżeli jest różny od zera, instrukcja jest wykonywana. W przeciwnym przypadku procesor pomijają,,
  - instrukcje logiczne realizujące operacje na każdej parze odpowiadających sobie bitów argumentów: AND, OR, XOR, NAND, NOR, XNOR. Dwie instrukcje dokonują dodatkowo negacji jednego z argumentów ANDN i ORN,
  - instrukcje zliczające nadmiarowe bity znaku, początkowe zera oraz początkowe jedynki,
  - przesunięcia arytmetyczne w lewo i w prawo o dowolną liczbę bitów,

- ekstrakcja dowolnego pola bitowego argumentu,

- instrukcje arytmetyczne realizujące operacje na spakowanych argumentach. Każdy z trzydziestodwubitowych rejestrów-argumentów przechowuje dwa argumenty 16-bitowe lub cztery argumenty 8-bitowe. Operacja arytmetyczna (dodawanie, odejmowanie, mnożenie, obliczanie wartości bezwzględnej) realizowana jest dla odpowiadających sobie argumentów 16-bitowych lub 8-bitowych. W ten sposób procesor realizuje elementy przetwarzania wektorowego (ang. SIMD — single instruction multiple data). W przypadku cyfrowego przetwarzania sygnału akustycznego, 16-bitowa reprezentacja liczb jest na ogół wystarczająca, natomiast w przypadku obrazów wystarcza nawet reprezentacja 8-bitowa,
- instrukcje porównania — wynik porównania umieszczony jest w rejestrze ogólnego przeznaczenia, przy czym zero oznacza fałsz, natomiast wartość różna od zera oznacza prawdę,
- operacje logiczne wykonywane na pojedynczych bitach argumentów,
- operacje arytmetyczne i operacje porównania wykonywane na rejestrach adresowych. Pozwalają na programową realizację trybów adresowania, które nie są wspomagane bezpośrednio przez procesor,
- skoki warunkowe i bezwarunkowe. Wykonanie instrukcji warunkowej uzależnione jest od zawartości rejestru ogólnego przeznaczenia, przy czym zero oznacza fałsz, a wartość różna od zera prawdę,
- instrukcje systemowe pozwalające na manipulacje zawartością rejestrów sterujących pracą procesora.

2.2. Pamięć oraz tryby adresowania

Najmniejszym adresowanym przez procesor TC-1 elementem danych jest jeden bajt. Ponieważ adresy są 32-bitowe, daje to przestrzeń adresową o pojemności 4 GB. Jest to wspólna przestrzeń danych, programu[17] oraz urządzeń peryferyjnych. Przestrzeń adresowa podzielona jest na szesnaście 256 KB segmentów o numerach od 0 do 15. Numer segmentu tworzą cztery najstarsze bity adresu. Pierwsze 16 KB każdego segmentu można zaadresować w trybie adresowania bezpośredniego [21]. Ostatnie dwa segmenty (o numerze 14 i 15) przeznaczone są dla urządzeń peryferyjnych.

[17] Mimo, że magistrala programu i magistrala danych są rozdzielone.

Procesor oprócz bajtów może adresować słowa 16-, 32- i 64-bitowe. Dostępnych jest przy tym siedem trybów adresowania. Tryby adresowania procesora TC-1 zostały zebrane w tablicy 10. Oprócz adresowania bezpośredniego dostępne są różne warianty adresowania pośredniego przez rejestr. Mnogość trybów adresowania ułatwia konstrukcję kompilatora języka C/C++ oraz pozwala na łatwy dostęp do typowych struktur danych, tzn. tablic, stosów, buforów cyklicznych itp.

Tablica 10

Tryby adresowania procesora TC-1 [21]

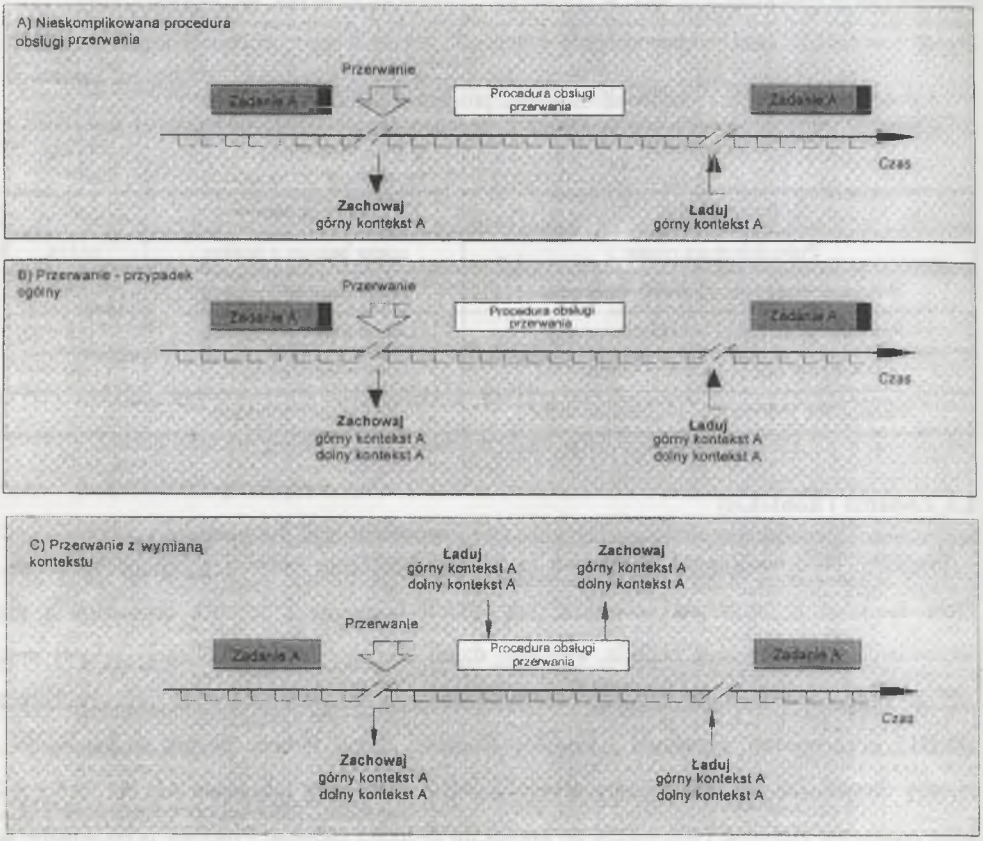
Tryb adresowania	Argumenty	Poślizg (l. Bitów)
Bezpośrednie	adres 18 bitowy	-
Pośrednie przez rejestr	rejestr bazowy + poślizg	10 / 16
Pośrednie przez rejestr z preinkrementacją	rejestr bazowy + poślizg	10
Pośrednie przez rejestr z postinkrementacją	rejestr bazowy + poślizg	10
Adresowanie pośrednie cykliczne	dwa rejestry adresowe	10
Adresowanie pośrednie z odwróceniem bitów	dwa rejestry adresowe	-

2.3. Zadania i konteksty

Procesor TC-1 wspomaga sprzętowo szybkie przełączanie zadań i wymianę kontekstu. Przez kontekst rozumie się zawartość wszystkich rejestrów procesora niezbędnych dla wznowienia wykonywania zadania. Procesor dzieli kontekst na „górny" (ang. upper) oraz „dolny" (ang. lower). Kontekst górny tworzą rejestry systemowe PCXI, PSW, rejestry danych D8-D15 oraz rejestry adresowe A10-A15. Kontekst dolny tworzy licznik rozkazów PC, rejestry danych D0-D7 oraz rejestry adresowe A2-A7. Rejestry adresowe A0, A1, A8 i A9 mają znaczenie globalne i nie należą do kontekstu żadnego zadania. Procesor przechowuje w pamięci operacyjnej połączoną listę obszarów CSW (ang. context switch area). Każdy z tych obszarów składa się z szesnastu 32-bitowych słów, które przechowują dokładnie jeden kontekst górny lub jeden kontekst dolny. Dzięki specjalnej magistrali łączącej procesor oraz zintegrowaną z nim pamięć czas przesłania kontekstu do pamięci wynosi zaledwie dwa takty zegarowe [21]. Wymiana kontekstu następuje w przypadku obsługi przerwania oraz wywołania podprogramu.

2.4. System przerwań

System przerwań procesora TC-1 jest bardzo rozbudowany. Składają się nań liczne moduły SRN (ang. service request node). Każdemu przerwaniu, poprzez programowanie odpowiadającego mu modułu SNR, nadaje się właściwy priorytet[18]. Procesor dopuszcza bowiem zagnieżdżanie przerwań. Tylko przerwanie o wyższym priorytecie może przerwać



Rys.5. Obsługa przerwań przez procesor TC-1
Fig.5. Interrupt processing by TC-1 CPU

realizację procedury obsługi przerwania o niższym priorytecie. Procedury obsługi przerwań identyfikowane są według priorytetu, tzn. wszystkie przerwania o tym samym priorytecie obsługiwane są przez tę samą procedurę. Tablica wektorów przerwań zawiera kolejno procedury obsługi przerwań o coraz wyższym priorytecie. Każda taka procedura składa się z ośmiu instrukcji. Jeżeli to nie wystarcza, trzeba wykonać skok do właściwej procedury

[18] Możliwe jest także określenie, czy przerwanie będzie obsługiwane przez jednostkę centralną czy też przez procesor peryferyjny (PCP).

obsługi przerwania leżącej poza tablicą wektorów przerwań. Wyboru zgłoszenia przerwania o najwyższym priorytecie dokonuje moduł ICU (ang. interrupt control unit), a następnie zgłasza je do procesora w celu uruchomienia właściwej procedury obsługi przerwania. Procesor podejmuje wtedy następujące działania[19]:

- zachowuje w pamięci górny kontekst,
- blokuje system przerwań,
- zapamiętuje priorytet obsługiwanego przerwania (priorytet przerwania staje się priorytetem jednostki centralnej),
- inicjuje system ochrony pamięci i urządzeń peryferyjnych,
- sięga do tablicy wektorów przerwań po pierwszą instrukcję procedury obsługi przerwania.

Rys.5 pokazuje trzy możliwe scenariusze, jakie może realizować procedura obsługi przerwania. W najprostszym przypadku wystarcza jej do realizacji swojego zadania korzystanie z rejestrów automatycznie zachowanego kontekstu górnego — rys.5a. W ogólnym przypadku procedura obsługi przerwania musi zachować także dolny kontekst — rys.5b. Przerwanie może także powodować wznowienie wykonywania wcześniej rozpoczętego zadania B — rys.5c. Wymaga to na początku sprowadzenia zachowanego w pamięci kontekstu zadania B, a przed zarzuceniem jego dalszego wykonywania, kontekst zadania B musi być zachowany w pamięci.

2.5. Ochrona pamięci oraz urządzeń peryferyjnych [22]

Kiedy system ochrony pamięci procesora TC-1 jest uaktywniony, sprawdza „legalność" każdego dostępu do pamięci systemu. Dotyczy to tak zapisu, jak i odczytu oraz pobrania instrukcji. System ten zbudowany jest na bazie zbioru rejestrów wyznaczających granice obszarów chronionych. Dla każdego obszaru chronionego określony jest typ dopuszczalnych operacji. Istnieje zawsze kilka (od dwóch do czterech) takich zbiorów, przechowywanych w obszarze CSFR (ang. core special function registers). Każdy ze zbiorów rejestrów ochrony związany jest z jednym z wykonywanych zadań. System ochrony pozwala „przetrwać" kluczowym dla urządzenia zadaniom systemowym, mimo błędnego działania innych, mniej istotnych zadań.

---

[19] O ile np. system przerwań procesora nie jest zablokowany.

System ochrony pamięci chroni także urządzenia peryferyjne — przestrzeń adresowa urządzeń peryferyjnych zajmuje dwa ostatnie segmenty przestrzeni adresowej danych. Dla każdego zadania określony jest stopień uprzywilejowania w stosunku do urządzeń peryferyjnych:

- user-0 Mode: zadanie nie ma prawa dostępu do urządzeń peryferyjnych,
- user-1 Mode: zadanie ma dostęp do wspólnych urządzeń nie objętych ochroną,
- supervisor mode: zadanie ma dostęp do wszystkich zasobów systemu oraz do wszystkich rejestrów systemowych.

2.6. Magistrala FPI

Jednostka centralna procesora TC-1 łączy się z urządzeniami peryferyjnymi umieszczonymi na tym samym chipie za pośrednictwem magistrali FPI (ang. flexible peripherial bus). Jest to uniwersalna magistrala z 32 liniami adresowymi i 64 liniami danych. Taktowana zegarem 100 MHz może przesyłać do 800 MB/s. Magistralę FPI charakteryzuje ponadto możliwość koegzystencji do 16 urządzeń-nadzorców magistrali (ang. master), praca synchroniczna, możliwość transmisji słów 8-, 16-, 32- i 64-bitowych, realizacja pojedynczych przesłań oraz przesłań blokowych.

Urządzenia dołączane do magistrali FPI mogą aktywnie uczestniczyć w realizacji protokołu transmisji (ang. master) lub być urządzeniami pasywnymi (ang. slave), których rejestry są zapisywane lub odczytywane przez urządzenia aktywne.

2.7. Procesor peryferyjny (PCP)

Procesor peryferyjny (ang. PCP — peripherial control processor module) przeznaczony jest dla obsługi urządzeń zewnętrznych zaimplementowanych poza układem procesora. Jego zadaniem jest nadzorowanie transmisji danych od i do obsługiwanych urządzeń. Układ może obsługiwać do 255 kanałów logicznych na zasadzie obsługi żądania przerwania. Obsługiwanych może być nawet 64 urządzeń zewnętrznych [21] bez konieczności przerywania pracy właściwej jednostki centralnej. Procesor PCP może przy tym wykonywać następujące zadania:

- przenosić dane z jednego obszaru pamięci do drugiego lub pomiędzy pamięcią a urządzeniem zewnętrznym,
- przenosić dane pomiędzy pamięcią wewnętrzną procesora PCP i urządzeniem zewnętrznym,
- czytać dane, modyfikować wykonując na nich operacje arytmetyczne lub logiczne i zapisywać wynik z powrotem,
- przenosić dane z jednego miejsca na drugie aż do napotkania wśród przenoszonych danych określonej wartości,
- czytać dane, badać ich wartość i podejmować na tej podstawie stosowne działania,
- czytać dane i dokonywać ich akumulacji.

W ten sposób procesor PCP może wykonywać zadania, które w przeciwnym razie musiałyby zostać wykonane przez jednostkę centralną. Na przykład procesor peryferyjny może:

- załadować do rejestru urządzenia zewnętrznego stałą w celu inicjalizacji timera,
- zmodyfikować pojedynczy bit lub pole bitowe w rejestrze urządzenia zewnętrznego w celu uruchomienia przetwarzania analogowo/cyfrowego,
- uśredniać wyniki pomiarów okresu przebiegu.

Procesor PCP posiada swój własny układ arbitrażowy, który wybiera spośród wszystkich zgłoszeń to o najwyższym priorytecie i inicjuje jego obsługę. Programy realizowane przez procesor PCP mogą być przechowywane w jego własnej pamięci wewnętrznej lub w pamięci danych i wtedy są mu udostępniane za pośrednictwem magistrali FPI.

2.8. Emulator sprzętowy

Procesor TC-1 wyposażony jest w emulator sprzętowy dołączony do systemu za pośrednictwem magistrali FPI. Emulator wspomaga zakładanie pułapek sprzętowych podczas uruchamiania programów, śledzenie zawartości rejestrów procesora oraz pamięci w czasie rzeczywistym, śledzenie działania urządzeń peryferyjnych.

3. Podsumowanie

Architektura wszystkich typów procesorów wydaje się coraz bardziej ujednolicać. Procesory sygnałowe przejmują funkcje mikrokontrolerów, mikroprocesory ogólnego przeznaczenia specjalizują się w wykonywaniu algorytmów cyfrowego przetwarzania sygnałów, mikrokontrolery przygotowywane są do pracy pod kontrolą wielozadaniowego systemu operacyjnego, np. Windows CE, oraz do przetwarzania sygnałów. Być może niedługo hasło jeden miliard tranzystorów, jeden procesor, jeden chip (i jeden system operacyjny — Bill Gates) w każdym żyrandolu stanie się rzeczywistością [17].

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, ZN Pol. Śl., ser. Informatyka, z.24, Gliwice 1993, ss.9-19.
5. Węgrzyn S.:Przyspieszenie realizacji algorytmów w systemach sterowanych przepływem argumentów, ZN Pol. Śl., ser. Informatyka, z.28, Gliwice 1994, ss.67-75.
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. Seshan N.: 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, 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, 1997, (SPRU190).
19. Texas Instruments, TMS320C62xx Programmer's Guide, 1997, (SPRU198).
20. Texas Instruments, TMS320C6x Optimizing C Compiler User's Guide, 1997, (SPRU187A).
21. Siemens A.G.: TriCore Architecture Overview Handbook, 1999.
22. Siemens A.G.: TriCore Architecture Manual, 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. Glover I., Grant P.: 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, nr 10, 1996.

34. Analog Devices: ADSP-21160 Technical Specifivation, Rev 3.0, 1999.

35. Intel: P6 Family of Processors Hardware Developer's Manual, 1998.

36. http://www.cyrix.com/html/products/mii/index.htm

37. Advanced Micro Devices: AMD-K6 Data Sheet, 1998.

Recenzent: Prof.dr hab.inż. Jerzy Skubis

Abstract

The second part of the article about DSP architecture is devoted to the near future of the DSP. There are two examples: 1) TMS320C6x – very long instruction word (VLIW) DSP from Texas Instruments; 2) TC-1 microcontroller from Infineon (Siemens). The article shows two ways of DSP evolution. The first is to build extremely fast, specialized machines primarily for number crunching. TMS320C6x is an example. It is a VLIW machine designed to explore instruction level parallelism (ILP) of digital signal processing algorithms. Instructions are statically mapped into processing units of the machine. Such architecture do not need such complex control unit as an superscalar processor. It results in: 1) fewer bugs in control unit implementation (in comparison to superscalar DSP); 2) shortened path from project to market; 3) a lot of free silicon area which can be used for implementation of fast processing units. The second way of DSP evolution is to build, more universal machines capable for multiprocessing. An example is TC-1 from Infineon. This machine integrates capabilities of conventional microcontroller and conventional DSP. It is superscalar processor with memory protection and hardware aid for multiprocessing. Such a processor can substitute two separate chips: microcontoller and DSP. The result is cheaper and more reliable boards.