ARCHITEKTURA PROCESORA CRUSOE FIRMY TRANSMETA

Marcin Rociek (6-8)
Tomasz Wojtowicz (1-5)

Informatyka, III rok
AGH, Kraków

Spis treści

1 Wstęp

W styczniu 2000 roku firma Transmeta zaprezentowała po raz pierwszy procesory Crusoe, kompatybilną z x86 rodzinę produktów, łączącą wysoką wydajność ze zdumiewająco niskim zużyciem energii. Jak można się było spodziewać, wprowadzono nową technologię projektowania i budowania procesorów, która pomogła w rozwoju tych produktów. Zaskoczeniem jest jednak to, że podstawą tej nowej technologii jest oprogramowanie: oszczędności w zużyciu energii pochodzą przede wszystkim z zastąpienia dużych ilości tranzystorów oprogramowaniem.

Procesory Crusoe zawierają sprzętową jednostkę otoczoną warstwą oprogramowania. Sprzęt stanowi procesor VLIW (*Very Long Instruction Word*) zdolny do wykonywania równocześnie aż czterech operacji w każdym cyklu zegara. Zestaw instrukcji tego procesora VLIW nie wykazuje żadnego podobieństwa do zestawu instrukcji x86. Został on zaprojektowany tylko i wyłącznie do szybkich i zużywających niewiele energii układów wykorzystujących konwencjonalne układy CMOS. Otaczająca procesor warstwa programowa umożliwia stworzenie programom x86 wrażenie pracy na prawdziwym procesorze x86. Warstwa programowa nazwana została oprogramowaniem Code Morphing, ponieważ dynamicznie przekształca instrukcje x86 na instrukcje VLIW. Oprogramowanie Code Morphing posiada kilka zaawansowanych cech, pozwalającyh osiągnąć wysoką wydajność już na niskim poziomie. Oprogramowanie Code Morphing korzysta również z pewnych mechanizmów ułatwiających jego działanie, które są wbudowane w sam procesor. Innymi słowy, projektanci z firmy Transmeta „wmontowali" pewne funkcje w sprzęt, inne zaś w oprogramowanie, stosownie do potrzeb i przeznaczenia danego procesora. Różne potrzeby i przeznaczenie może w przyszłości prowadzić do innego rozdziału funkcji pomiędzy sprzętem a oprogramowaniem.

Technologia oprogramowania Code Morphing zmienia całkowicie podejście do projektowania mikroprocesorów. Pokazując, że praktyczne mikroprocesory mogą być zaimplementowane jako hybrydy sprzętu i oprogramowania, Transmeta dramatycznie rozszerzyła przestrzeń, którą projektanci mikroprocesorów mogą badać w celu znalezienia optymalnego rozwiązania. Prace nad nowym procesorem mogą być rozdzielone na dwie równolegle pracujące grupy: jedną zajmującą się tylko częścią sprzętową procesora, oraz drugą opracowującą nową wersję oprogramowania emulującego.

2 Perspektywy nowej technologii

Projektanci z firmy Transmeta odseparowali zestaw instrukcji x86 od sprzętu, co umożliwia tworzenie procesorów o zupełnie innej konstrukcji niż konwencjonalne implementacje x86. Z tego samego powodu sprzętowa część procesora może być w przyszłości całkowicie zmieniona bez wpływu na działanie oprogramowania x86. Każdy nowy projekt samego procesora wymaga jedynie nowej wersji oprogramowania Code Morphing, które przekształci instrukcje x86 na instrukcje nowego procesora.

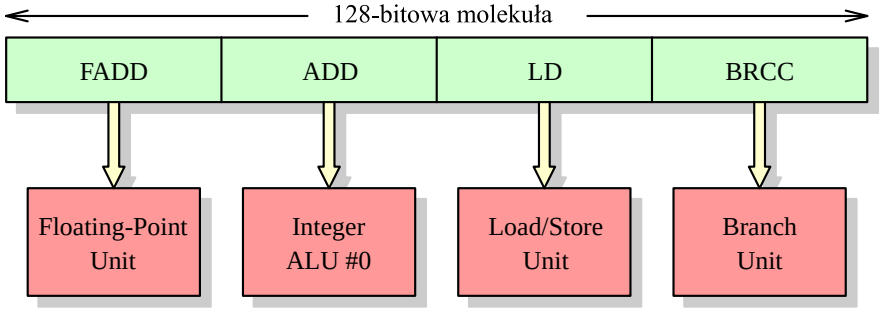
Dwa pierwsze modele procesora Crusoe, tj. TM3200 i TM5400 zostały skonstruowane tak, aby jak najbardziej zminimalizować rozmiar oraz zużycie energii. Poprzez eliminacje

około trzech czwartych tranzystorów, które byłyby potrzebne do budowy konwencjonalnego procesora o podobnej wydajności, projektanci zredukowali zużycie energii oraz wielkość samej struktury półprzewodnikowej. Jednakże w przyszłych wersjach procesora nacisk może być położony na inne czynniki i stosownie do tego mogą być użyte inne techniki implementacji.

Samo oprogramowanie Code Morphing oferuje możliwość zwiększenia wydajności bez konieczności zmian sprzętowej części procesora. Aktualnie jest ono ucieleśnieniem pierwszej generacji nowej technologii, która może być dalej optymalizowana wraz z postępem doświadczeń i eksperymentów. Ponieważ oprogramowanie Code Morphing zwykle rezyduje w standardowej pamięci Flash ROM na płycie głównej, jego ulepszone wersje mogą być nawet wprowadzane do już działających systemów wykorzystujących Crusoe.

3 Podstawy procesora Crusoe

Zgodność procesora Crusoe z x86 jest zapewniona przez zastosowanie oprogramowania Code Morphing. Sprzętową częścią procesora jest bardzo prosty procesor VLIW o dużej wydajności, zawierający dwa moduły obliczeń stałoprzecinkowych, moduł obliczeń zmiennoprzecinkowych, moduł komunikacji z pamięcią (zapis / odczyt), oraz moduł rozgałęzień kodu. W procesorze Crusoe instrukcję stanowi 64- lub 128-bitowe słowo, zwane *molekułą*, składające się z maksymalnie czterech instrukcji (podobnych do RISC'owych), nazywanych *atomami*. Wszystkie atomy znajdujące się w jednej molekule wykonywane są równolegle, zaś format molekuły bezpośrednio wskazuje na to do której jednostki procesora ma trafić dany atom, co bardzo upraszcza część procesora odpowiedzialną za dekodowanie instrukcji. Rysunek 1 pokazuje przykładową 128-bitową molekułę oraz bezpośrednie odwzorowanie odpowiedniej części molekuły na odpowiednią jednostkę procesora. Molekuły wykonywane są w zwykłej kolejności, nie istnieje więc konieczność istnienia skomplikowanego sprzętu wykonującego instrukcje nie w kolejności. Aby procesor mógł rzeczywiście wykorzystać pełnię swej mocy, molekuły muszą być wypełnione atomami tak bardzo jak to tylko możliwe. Szczegóły realizacji tego omówione są szczegółowo w rozdziale dotyczącym oprogramowania Code Morphing.

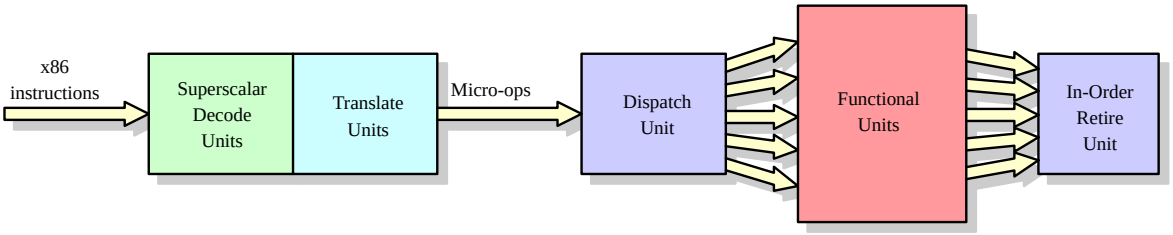


Rysunek 1: Pojedyncza 128-bitowa instrukcja (molekuła) może składać się z maksymalnie czterech atomów, które są wykonywane równocześnie.

Jednostka stałoprzecinkowa posiada 64 rejestry, od `%r0` do `%r63`. Zwykle oprogramowanie Code Morphing wykorzystuje część z nich do przetrzymywania stanów rejestrów x86, inne zaś są wykorzystywane jako rejestry przechowujące wewnętrzne stany systemu bądź rejestry tymczasowe.

W poniższym opracowaniu przyjęto taką konwencję zapisu kodu asemblerowego, w której jedna molekuła zapisana jest w jednej linii, a poszczególne atomy oddzielone są średnikami. Pierwszy rejestr atomu jest rejestrem docelowym operacji. Przyrostek „`.c`” dodany do operandu oznacza, że operacja zmienia flagi. Gdy rejestr przechowuje zawartość rejestru x86, używana jest nazwa rejestru x86 (przykładowo `%eax` zamiast mniej mówiącej nazwy `%r0`).

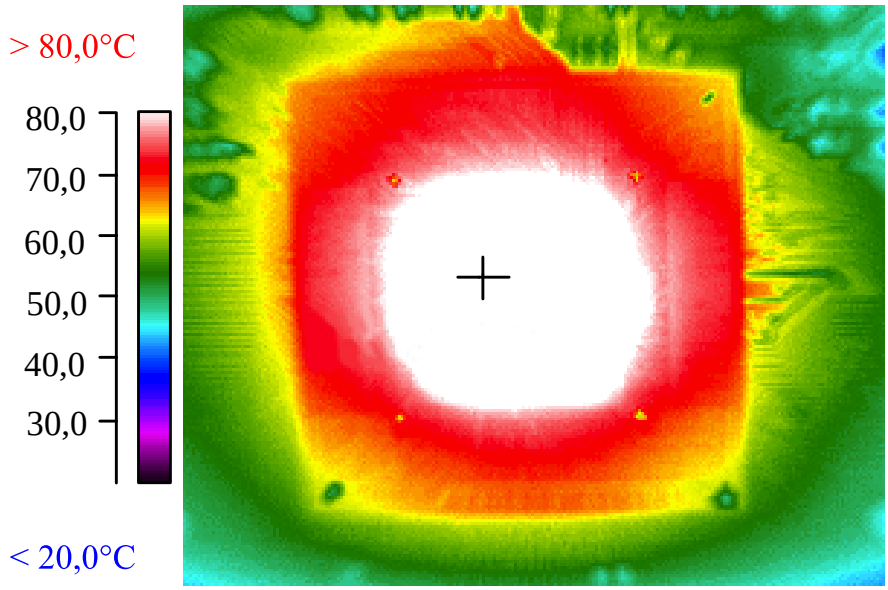
Superskalarne procesory x86 wykonujące operacje nie w kolejności, takie jak Pentium II czy Pentium III również posiadają wiele oddzielnych jednostek funkcjonalnych które mogą wykonywać równolegle mikrooperacje. Rysunek 2 przedstawia typowy schemat sprzętu wykorzystywanego przez te procesory do translacji instrukcji x86 na mikrooperacje i szeregowania tychże mikrooperacji w taki sposób, aby w maksymalnym stopniu zostały wykorzystane jednostki funkcjonalne procesora. Ponieważ jednostka rozmieszczająca mikrooperacje w jednostkach funkcjonalnych (*Dispatch unit*) zmienia kolejność mikrooperacji (w celu maksymalnego wykorzystania jednostek funkcjonalnych) potrzebna jest kolejna część procesora (*In-Order Retire Unit*) zapewniająca efektywne zrekonstruowanie kolejności oryginalnych instrukcji x86 i gwarantująca że efekty ich działania wystąpią w prawidłowej kolejności. Oczywiście sprzętowa część procesora tego typu jest znacznie bardziej skomplikowana od prostego procesora VLIW wykorzystanego w procesorze Crusoe.



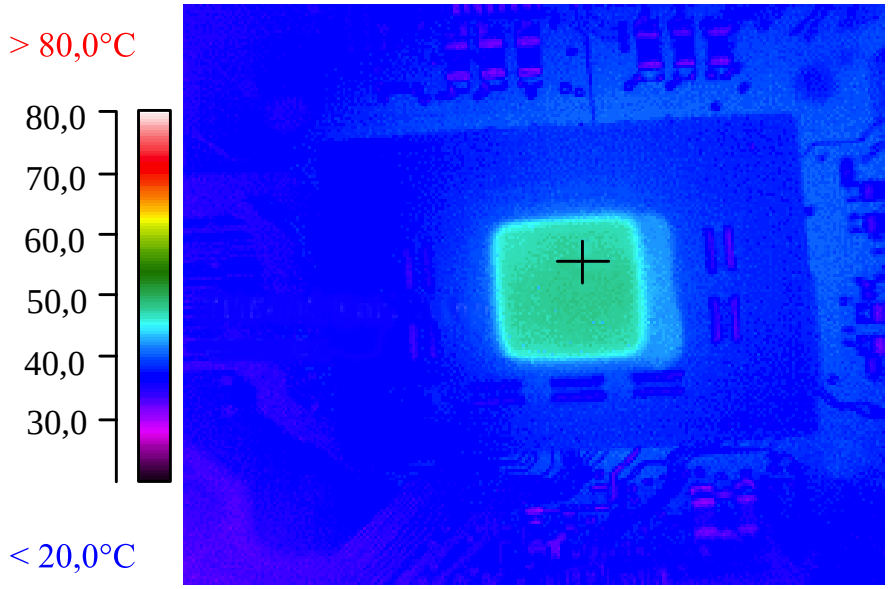
Rysunek 2: Tradycyjne procesory wykorzystują sprzęt w celu stworzenia mikrooperacji, które mogą być wykonane równolegle.

Ponieważ zbiór instrukcji x86 jest dość złożony, jednostki odpowiedzialne za dekodowanie i rozmieszczanie instrukcji wymagają dużych ilości tranzystorów, które pobierają dużo mocy. Cały układ wytwarza ciepło w przybliżeniu proporcjonalne do ilości tranzystorów. Tablica 1 porównuje rozmiary mobilnych procesorów Intela oraz procesory Crusoe.

Rysunki 3 i 4 pokazują różnice w zużyciu energii (pokazaną jako temperaturę) pomiędzy procesorami Pentium III i Crusoe na których uruchomiony jest programowy odtwarzacz DVD (*Digital Versatile Disk*). Procesor TM5400 nie wymaga w ogóle chłodzenia, podczas gdy Pentium III może rozgrzać się do temperatury powodującej uszkodzenie, jeśli nie jest zastosowane odpowiednie chłodzenie.



Rysunek 3: Procesor Pentium III odtwarzający DVD z maksymalną temperaturą 105,5°C.



Rysunek 4: Procesor Crusoe TM5400 odtwarzający DVD z maksymalną temperaturą 48,2°C.

	Mobile PII	Mobile PII	Mobile PIII	TM3120	TM5400
Technologia	0,25μm	0,25μm shrink	0,18μm	0,22μm	0,18μm
Pamięć podręczna L1	32kB	32kB	32kB	96kB	128kB
Pamięć podręczna L2	0	256kB	256kB	0	256kB
Wielkość struktury	130mm^2	180mm^2	106mm^2	77mm^2	73mm^2

Tablica 1: Oprogramowanie Code Morphing pozwala na znaczne uproszczenie procesora

4 Aktualnie produkowane modele

4.1 TM 3200

Procesor Crusoe TM 3200 zawiera:

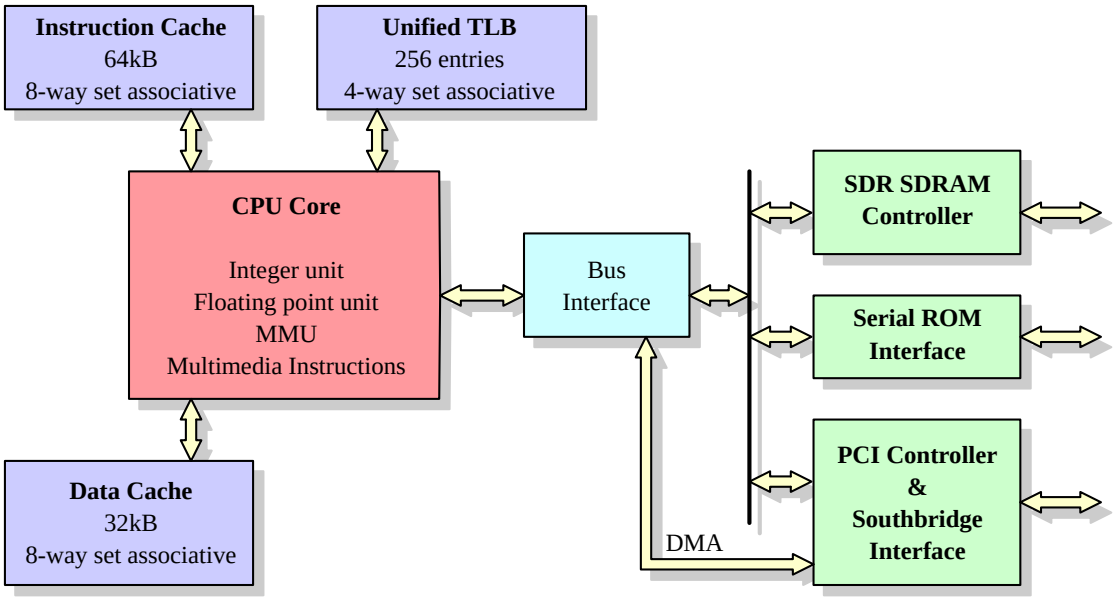
- Procesor VLIW oraz oprogramowanie Code Morphing dostarczające kompatybilne z x86 rozwiązanie dla urządzeń przenośnych
- Jądro procesora pracujące z częstotliwością 366 i 400 MHz
- Zintegrowana pamięć podręczna: 64kB dla kodu i 32kB dla danych
- Zintegrowany mostek północny zawierający:
  - Kontroler pamięci SDR SDRAM, 66-133 MHz, 3,3V
  - Kontroler szyny PCI (zgodny z PCI 2.1), 33 MHz, 3,3V
- Zaawansowany system zarządzania energią oraz bardzo niskie zużycie energii wydłużają życie źródła zasilania
- Pełne wsparcie dla SMM (*System Management Mode*)
- Obudowa BGA, 474-pinowa

Procesor TM 3200 oprócz samego jądra VLIW zawiera 64kB pamięci podręcznej kodu, 32kB pamięci podręcznej danych, 64-bitowy kontroler pamięci SDR SDRAM oraz 32-bitowy kontroler PCI. Te dodatkowe jednostki funkcjonalne, które zwykle są częścią systemu otaczającego mikroprocesor, pozwalają na tworzenie wysoce zintegrowanych i tanich urządzeń przenośnych. Jądro procesora zasilane jest napięciem 1,5V, co prowadzi do niskiego zużycia mocy, nawet przy dużych częstotliwościach pracy. Podczas typowej pracy, pobór mocy jest rzędu 15mW.

4.1.1 Architektura

Procesor zawiera jednostki do obliczeń stało- i zmiennoprzecinkowych, pamięć podręczną dla danych i dla kodu, jednostkę zarządzającą pamięcią oraz jednostkę instrukcji multimedialnych. Ponadto, poza tymi tradycyjnymi składnikami procesora, w procesorze

zawarte są: kontroler pamięci SDR SDRAM, kontroler magistrali PCI oraz kontroler pamięci ROM, które to urządzenia przeważnie wchodzą w skład układów współpracujących z procesorem. Schemat blokowy procesora TM3200 pokazany jest na rysunku 5.



Rysunek 5: Schemat blokowy architektury procesora TM 3200

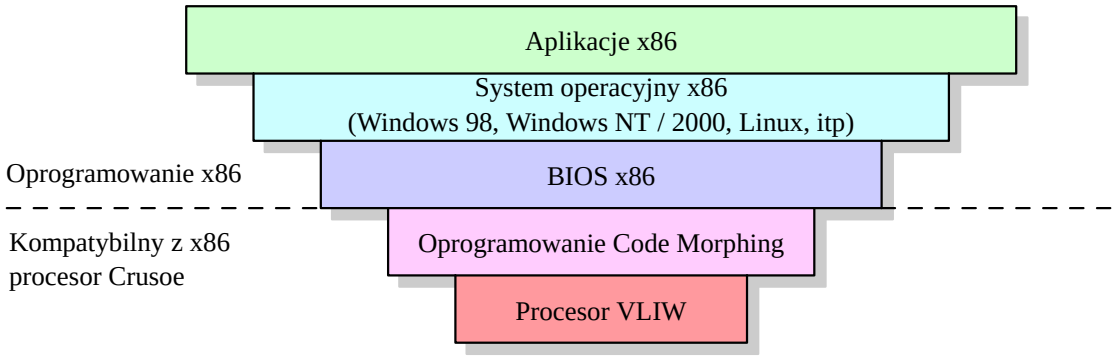
Jądro procesora

Architektura samego jądra procesora Crusoe jest stosunkowo prosta w porównaniu z aktualnymi standardami. Oparta jest na 128-bitowych instrukcjach VLIW (*Very Long Instruction Word*). Architektura procesora jest prosta między innymi dlatego, że prawie zupełnie nie jest wykorzystywana do zarządzania instrukcjami (przestawianie instrukcji, przewidywanie skoków itp) - odpowiedzialność za te czynności spadła prawie w całości na warstwę programową. Umożliwiło to uproszczenie sprzętu i prostą jego implementację zawierającą 7-częściowy potok dla instrukcji stałoprzecinkowych oraz 10-częściowy dla zmiennoprzecinkowych. Zmniejszenie ilości tranzystorów wymaganych do zbudowania procesora pozwala na duży wzrost stosunku wydajności do zużycia mocy w porównaniu ze zwykłymi procesorami x86.

Procesor TM3200 zawiera 64kB pamięci podręcznej poziomu 1 (L1) dla instrukcji, oraz 32kB pamięci podręcznej L1 dla danych. Taka architektura pamięci podręcznej gwarantuje maksymalną wewnętrzną przepustowość pamięci dla aplikacji wykorzystywanych w urządzeniach przenośnych, zachowując przy tym wysoką wydajność przy małym zużyciu energii.

Poza klasycznymi jednostkami sprzętowymi przeznaczonymi do wykonywania operacji logicznych, arytmetycznych, przesuwających oraz zmiennoprzecinkowych, procesor Crusoe

posiada również jednostki zupełnie odmienne od tych spotykanych na co dzień w procesorach x86. Dla ułatwienia procesu translacji kodu x86 na kod jądra VLIW, generowane są takie same kody warunkowe jak w x86. Ponadto sprzęt operuje na identycznych jak x86 80-bitowych liczbach zmiennoprzecinkowych. Również bufor TLB (*Translation Look-aside Buffer*) wykorzystuje te same bity zabezpieczające oraz takie samo odwzorowanie adresów jak procesory x86. Część programowa procesora wykorzystuje te cechy do emulowania wszelkich innych cech architektury x86. Oprogramowanie współpracujące z jądrem to oprogramowanie Code Morphing, stanowiące wraz z sprzętem procesor kompatybilny z x86. Hierarchia oprogramowania pokazana jest na rysunku 6.



Rysunek 6: Hierarchia oprogramowania procesora Crusoe

Typowym zachowaniem oprogramowania Code Morphing jest wykonywanie pętli która dekoduje i wykonuje instrukcje x86. Przez pierwsze kilka razy, pewien kod x86 (sekwencja instrukcji) jest dekodowany bajt po bajcie, zamieniany na odpowiadającą mu sekwecję instrukcji VLIW i wysyłany do wykonania. Po kilkukrotnym wykonaniu tego samego bloku kodu x86, oprogramowanie Code Morphing zamienia ciąg instrukcji x86 na wysoce zoptymalizowany i bardzo szybki ciąg instrukcji VLIW, który następnie zostaje wykonany przez procesor, oraz dodatkowo zapisany w pamięci podręcznej programu translującego do przyszłego użycia. Jeśli ten sam blok kodu x86 ma zostać wykonany ponownie, jego zoptymalizowana wersja jest od razu wykonywana i nie ma potrzeby ponownego dekodowania i przekształcania.

Zintegrowany kontroler pamięci SDR SDRAM

Kontroler pamięci SDR SDRAM współpracuje z maksymalnie czterema bankami, równoważnymi dwóm modułom SO-DIMM (*Small Outline Dual In-Line Memory Module*), pracującymi z pojedynczą częstotliwością (SDR - *Single Data Rate*), które mogą być skonfigurowane jako 64-bitowe lub 72-bitowe moduły SO-DIMM. Moduły te mogą być 64Mb, 128Mb lub 256Mb. Wszystkie moduły muszą wykorzystywać SDRAM o tej samej częstotliwości, lecz nie ma ograniczeń co do mieszania różnych konfiguracji SO-DIMM w każdym ze slotów SO-DIMM.

Ustawienie częstotliwości pracy interfejsu pamięci SDR SDRAM dokonywane jest podczas sekwencji uruchamiania. Chociaż procesor może współpracować z interfejsem SDR pracującym z częstotliwościami w zakresie od $1/2$ do $1/15$ częstotliwości jądra procesora, zalecaną częstotliwością interfejsu jest częstotliwość z zakresu 66 - 133MHz. Ponadto zalecane jest aby w jednym module SO-DIMM nie było więcej niż 8 układów, aby zachować wymaganą częstotliwość pracy przy zachowaniu poprawnego sygnału.

Zintegrowany kontroler PCI

Procesor Crusoe zawiera kontroler magistrali PCI zgodny z PCI 2.1. Magistala PCI jest 32-bitowa, pracuje z częstotliwością 33MHz i jest kompatybilna z sygnałami o poziomie $3,3V$, nie toleruje jednak sygnałów $5V$. Kontroler PCI zawiera mostek do urządzeń PCI, układ arbitrujący oraz kontroler DMA (*Direct Memory Access*).

Magistrala PCI może zapisywać i odczytywać z prędkością 132MBps w blokach po 4kB. Kontroler PCI zarządza odczytami i zapisami PCI - DRAM. 64-bajtowy bufor zapisu CPU→PCI konwertuje sekwencyjne zapisy I/O na przesłania PCI. Kontroler DMA zarządza odczytami i zapisami PCI - DRAM. 64-bajtowy bufor zapisu PCI→DRAM konwertuje 64-bajtowe przesłanie na osiem indywidualnych par adres / dane. 64-bajtowy bufor odczytu DRAM→PCI pozwala na kontynuację operacji odczytu po trafieniu do bufora.

Szeregowy interfejs ROM

Szeregowy 5-pinowy interfejs ROM zawarty w procesorze Crusoe jest wykorzystywany do odczytu z szeregowej pamięci flash ROM. Ta nieulotna pamięć, o rozmiarze 1MB jest używana do przechowywania oprogramowania Code Morphing. Podczas procesu uruchamiania oprogramowanie to jest kopiowane z pamięci ROM do specjalnej przestrzeni w pamięci SDRAM. Po skopiowaniu oprogramowanie Code Morphing wymaga od 8 do 16MB pamięci, która to przestrzeń jest niewidzialna dla oprogramowania x86.

4.1.2 Kompatybilność oprogramowania

Połączenie procesora Crusoe wraz z oprogramowaniem Code Morphing stanowi jednostkę zdolną do wykonywania oprogramowania stworzonego dla architektury x86 bez konieczności jego ponownej kompilacji. Systemy komputerowe oparte o procesor Crusoe mogą pracować z wszelkimi standardowymi systemami operacyjnymi oraz aplikacjami kompatybilnymi z x86, włączając w to m.in: Microsoft Windows 95, Windows 98, Windows NT oraz Linux'a.

4.1.3 Zużycie energii i zarządzanie energią

Jądro procesora Crusoe zasilane jest napięciem 1,5V, przy czym zużycie energii jest bardzo małe nawet podczas pracy z wysoką wydajnością. Procesor w pełni wspomaga zgodne z ACPI tryby zarządzania energią pracując w jednym z pięciu możliwych stanów zużycia energii: *Normal*, *Auto Halt*, *Quick Start*, *Deep Sleep* oraz *Off*. Stany te mogą być używane

do redukcji zużycia energii przez procesor w czasie gdy system wymaga bardzo małej (lub wcale) aktywności procesora.

Tablica 2 pokazuje rekomendowane stany procesora dla każdego z globalnych stanów systemu ACPI. Typowe zużycie energii przez procesor pokazane jest na tablicach 3 oraz 4.

Stan systemu ACPI			**Stan procesora**	**SDRAM**	**Zegar**
G0/S0	(Working)	C0	Normal	Normal	Running
		C1	Auto Halt	Normal	Running
		C2	Quick Start	Self Refresh	Running
		C3	Deep Sleep	Self Refresh	Clocks Stopped
G1/S1	(Sleeping)		Deep Sleep	Self Refresh	PLL Shut Down
G1/S2	(Suspend to RAM)		Deep Sleep	Self Refresh	PLL Shut Down
G1/S3	(Suspend to RAM)		Off	Self Refresh	PLL Shut Down
G1/S4	(Suspend to Disk)		Off	Off	Off
G2/S5	(Soft Off)		Off	Off	Off
G3	(Mechanical Off)		Off	Off	Off

Tablica 2: Stany systemu zarządzania energią procesora Crusoe

Zadanie Stan systemu		Jądro procesora	Mostek północny	Uwagi
Web Browser	(C0)	1,6 W	0,4 W	1, 2
MP3 Playback	(C0)	1,0 W	0,4 W	1, 3
Auto Halt	(C1)	0,5 W	0,4 W	1, 4
Quick Start	(C2)	0,2 W	0,2 W	1, 5
Deep Sleep	(C3)	0,0 W	0,015 W	1, 6
Off / Instant On		0,0 W	0,0 W	1, 7

Uwagi:

1. Wartości mocy mierzone przy działającym system zarządzania energią oraz włączonym systemie LongRun.
2. Odtwarzanie DVD przy pomocy odtwarzacza Win DVD 2000 w systemie Windows 98SE.
3. Odtwarzanie MP3 przy pomocy programu MMJukebox w systemie Windows 98SE.
4. Wejście w stan *Auto Halt* poprzez wykonanie instrukcji `HLT`.
5. Wejście w stan *Quick Start* poprzez zatwierdzenie `STPCLK#`.
6. Wejście w stan *Deep Sleep* poprzez zatwierdzenie `SLEEP#` oraz zatrzymanie `CLKIN`, będąc w stanie *Quick Start*.
7. Wejście w stan *Off / Instant On* poprzez przejście systemu w stan *Suspend to RAM* lub *Suspend to Disk*.

Tablica 3: Typowe zużycie energii przez procesor TM3200, 366 MHz, 1,5V

Zadanie Stan systemu		Jądro procesora	Mostek północny	Uwagi
Web Browser	(C0)	1,8 W	0,4 W	1, 2
MP3 Playback	(C0)	1,0 W	0,4 W	1, 3
Auto Halt	(C1)	0,5 W	0,4 W	1, 4
Quick Start	(C2)	0,2 W	0,2 W	1, 5
Deep Sleep	(C3)	0,0 W	0,015 W	1, 6
Off / Instant On		0,0 W	0,0 W	1, 7

Uwagi: zob. tablica 3.

Tablica 4: Typowe zużycie energii przez procesor TM3200, 400 MHz, 1,5V

4.2 TM 5400

Procesor Crusoe TM 5400 zawiera:

- Procesor VLIW oraz oprogramowanie Code Morphing dostarczające kompatybilne z x86 rozwiązanie dla urządzeń przenośnych
- Jądro procesora pracujące z częstotliwością 500-700 MHz
- Zintegrowana pamięć podręczna poziomu L1: 64kB dla kodu i 64kB dla danych, oraz 256kB poziomu L2
- Zintegrowany mostek północny zawierający:
  - Kontroler pamięci DDR SDRAM, 100-133 MHz, 2,5V
  - Kontroler pamięci SDR SDRAM, 66-133 MHz, 3,3V
  - Kontroler szyny PCI (zgodny z PCI 2.1), 33 MHz, 3.3V
- Zaawansowany system zarządzania energią oraz bardzo niskie zużycie energii wydłużają życie źródła zasilania
  - 1-2 W przy pracy 500-700 MHz, 1,2-1,6 V - wykonywanie typowych aplikacji multimedialnych
  - 50 mW w trybie *Deep sleep*
- Pełne wsparcie dla SMM (*System Management Mode*)
- Obudowa BGA, 474-pinowa

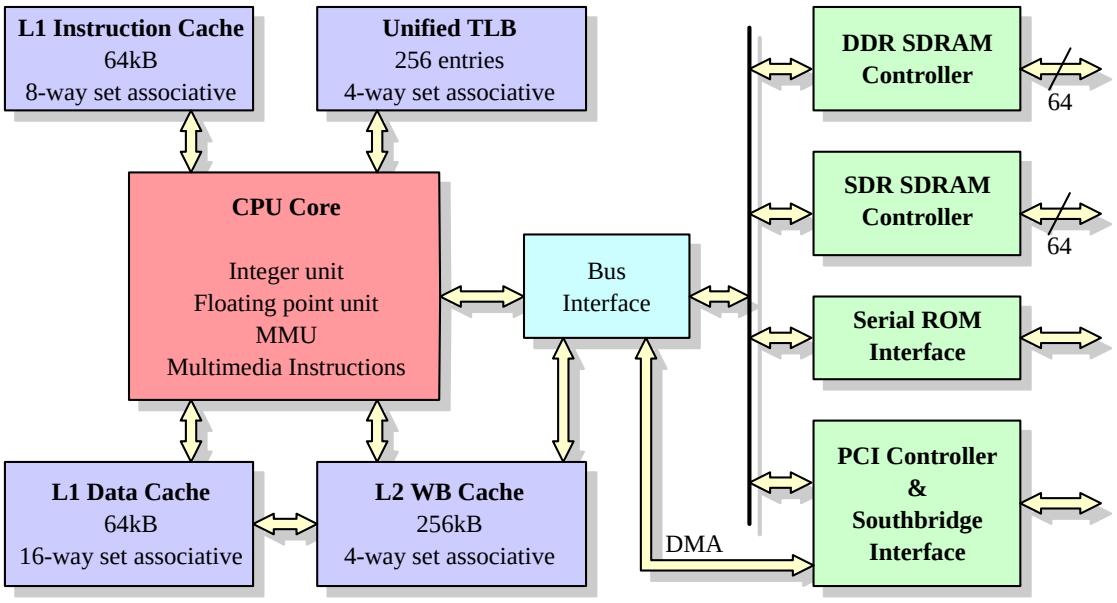
Procesor TM 5400 oprócz samego jądra VLIW zawiera po 64kB pamięci podręcznej dla kodu i danych, 256kB pamięci podręcznej poziomu L2 pracującej w trybie *write-back*, 64-bitowy kontroler pamięci DDR SDRAM, 64-bitowy kontroler pamięci SDR SDRAM oraz 32-bitowy kontroler PCI. Jądro procesora zasilane jest napięciem z zakresu 1,2-1,6V, co prowadzi do niskiego zużycia mocy, nawet przy dużych częstotliwościach pracy. Podczas typowej pracy, pobór mocy jest rzędu 50mW.

4.2.1 Architektura

Schemat blokowy procesora TM5400 pokazany jest na rysunku 7. Jak widać różni się od procesora TM3200 dwoma dodatkowymi blokami, tj. kontrolerem pamięci DDR SDRAM oraz pamięcią podręczną poziomu L2.

Jądro procesora

Procesor TM5400 zawiera, podobnie jak TM3200, 64kB pamięci podręcznej poziomu L1 dla kodu. Większa natomiast jest niż w przypadku modelu TM3200 ilość pamięci podręcznej dla danych, wynosząca również 64kB. Ponadto model 5400 zawiera 256kB pamięci podręcznej poziomu L2, pracującej w trybie *write-back*, której to nie było w modelu TM3200.



Rysunek 7: Schemat blokowy architektury procesora TM 5400

Zintegrowany kontroler pamięci DDR SDRAM

Interfejs pamięci DDR SDRAM jest najwydajniejszym układem współpracującym z pamięcią dostępnym w procesorze Crusoe. Kontroler ten może współpracować tylko i wyłącznie z DDR (*Double Data Rate*) SDRAM, przesyłając dane z częstotliwością dwukrotnie większą niż częstotliwość zegara samego interfejsu. Kontroler może współpracować z maksymalnie czterema bankami DDR SDRAM, równoważnymi dwóm modułom DIMM, używając 64-bitowego interfejsu.

Jako pamięć mogą pracować układy 64Mb, 128Mb lub 256Mb. Dla zapewnienia jak największej wydajności układy pamięci powinny być przylutowane do płyty głównej, nie zaś umieszczone na modułach DIMM. Aby zredukować obciążenie sygnału powinny być to tylko układy $\times 8$, $\times 16$ lub $\times 32$.

Ustawienie częstotliwości interfejsu pamięci DDR SDRAM dokonuje się podczas procesu uruchamiania. Chociaż procesor może współpracować z częstotliwościami od $1/2$ do $1/15$ częstotliwości jądra, to zalecaną częstotliwością jest częstotliwość z przedziału 100-133MHz.

Pozostałe składniki

Pozostałe specyficzne składniki procesora Crusoe, tzn:

- Kontroler pamięci SDR SDRAM
- Zintegrowany kontroler PCI
- Szeregowy interfejs ROM

są identyczne jak w przypadku modelu TM3200.

4.2.2 Zużycie energii i zarządzanie energią

Jądro procesora Crusoe TM5400 jest zasilane napięciem z przedziału 1,2-1,6V, przy bardzo małym poborze mocy nawet podczas wykonywania operacji wymagających dużej wydajności. Model TM5400 zawiera system LongRun, będący technologią pozwalającą na adaptacyjne zarządzanie zużyciem mocy. System LongRun dynamicznie zmniejsza zużycie mocy przez jądro procesora w zależności od wykonywanych zadań.

Zmiana zużycia mocy następuje przez regulowanie częstotliwości zegara oraz napięcia zasilającego jądro procesora. Sterowanie to odbywa się przy pomocy pewnych protokołów adaptacyjnego zarządzania energią, które monitorują obciążenie procesora i kontrolują jego moc i stopień wydajności. System LongRun jest wystarczająco efektywny w przypadku aplikacji, które dominującą część czasu spędzają w stanie aktywnym.

Ponadto procesor Crusoe w pełni wspiera zgodne z ACPI tryby zarządzania energią pracując w jednym z pięciu możliwych stanów zużycia energii: *Normal*, *Auto Halt*, *Quick Start*, *Deep Sleep* oraz *Off*. Stany te mogą być używane do redukcji zużycia energii przez procesor w czasie gdy system wymaga bardzo małej (lub wcale) aktywności procesora.

Zalecane stany procesora dla każdego z globalnych stanów systemu ACPI zostały pokazane na tablicy 2. Natomiast typowe zużycie energii przez procesor TM5400 pokazane jest na tablicy 5.

Zadanie Stan systemu		**Jądro procesora**	**Mostek północny**	**Uwagi**
Web Browser	(C0)	1,27 W	0,1 W	1, 2
MP3 Playback	(C0)	0,53 W	0,05 W	1, 3
Auto Halt	(C1)	0,22 W	0,1 W	1, 4
Quick Start	(C2)	0,17 W	0,03 W	1, 5
Deep Sleep	(C3)	0,02 W	0,03 W	1, 6
Off / Instant On		0,0 W	0,0 W	1, 7

Uwagi: zob. tablica 3.

Tablica 5: Typowe zużycie energii przez procesor TM5400, 500-700 MHz, 1,2-1,6 V

4.3 TM 5600

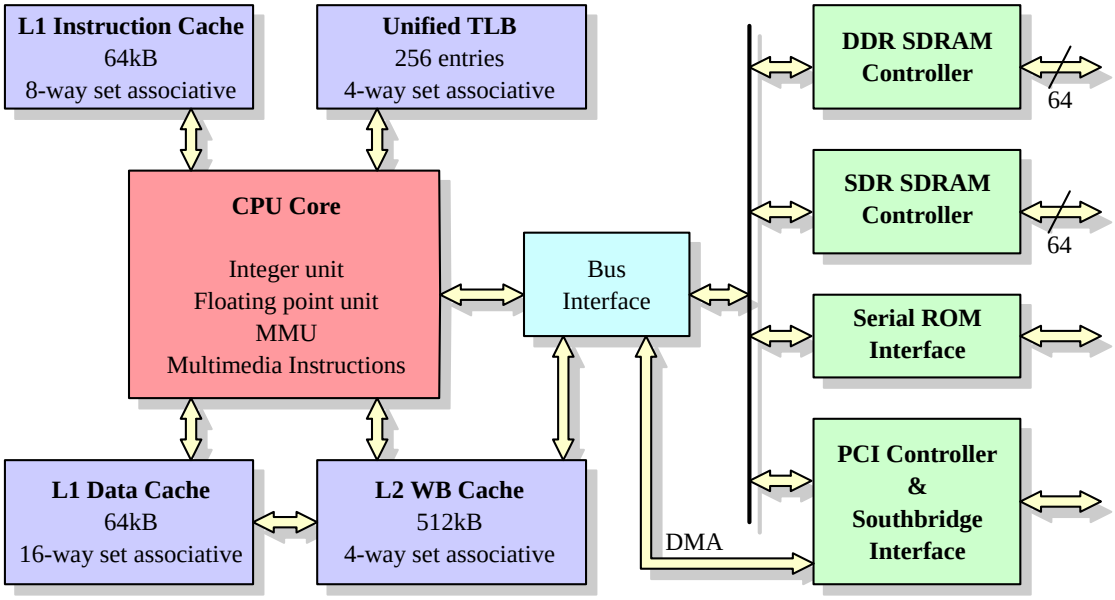
Procesor Crusoe TM 5600 zawiera:

- Procesor VLIW oraz oprogramowanie Code Morphing dostarczające kompatybilne z x86 rozwiązanie dla urządzeń przenośnych
- Jądro procesora pracujące z częstotliwością 500-700 MHz
- Zintegrowana pamięć podręczna poziomu L1: 64kB dla kodu i 64kB dla danych, oraz 512kB poziomu L2
- Zintegrowany mostek północny zawierający:
  - Kontroler pamięci DDR SDRAM, 100-133 MHz, 2,5V
  - Kontroler pamięci SDR SDRAM, 66-133 MHz, 3,3V
  - Kontroler szyny PCI (zgodny z PCI 2.1), 33 MHz, 3.3V
- Zaawansowany system zarządzania energią oraz bardzo niskie zużycie energii wydłużają życie źródła zasilania
  - 1-2 W przy pracy 500-700 MHz, 1,2-1,6 V - wykonywanie typowych aplikacji multimedialnych
  - 100 mW w trybie *Deep sleep*
- Pełne wsparcie dla SMM (*System Management Mode*)
- Obudowa BGA, 474-pinowa

Procesor TM 5600 oprócz samego jądra VLIW zawiera po 64kB pamięci podręcznej dla kodu i danych, 256kB pamięci podręcznej poziomu L2 pracującej w trybie *write-back*, 64-bitowy kontroler pamięci DDR SDRAM, 64-bitowy kontroler pamięci SDR SDRAM oraz 32-bitowy kontroler PCI. Jądro procesora zasilane jest napięciem z zakresu 1,2-1,6V, co prowadzi do niskiego zużycia mocy, nawet przy dużych częstotliwościach pracy. Podczas typowej pracy, pobór mocy jest rzędu 100mW.

4.3.1 Architektura

W zasadzie model TM5600 różni się od modelu TM5400 tylko i wyłącznie wielkością pamięci podręcznej poziomu L2. Opis architektury wewnętrznej jest więc zbyteczny i został ograniczony jedynie do przedstawienia schematu blokowego (rysunek 8).



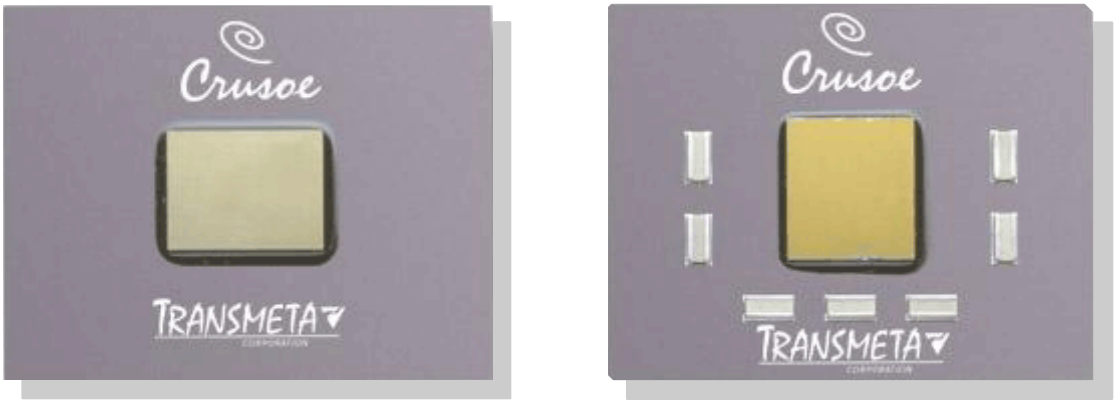
Rysunek 8: Schemat blokowy architektury procesora TM 5600

4.4 Porównanie

Tablica 6 przedstawia zbiorcze porównanie podstawowych parametrów aktualnie produkowanych modeli. Ponadto na rysunku 9 przedstawiono zewnętrzny wygląd dwóch z przedstawianych tu modeli.

	TM3200	TM5400	TM5600
Technologia	0,22μm	0,18μm	0,18μm
Zakres częstotliwości	333-400MHz	500-700MHz	500-700MHz
Pamięć podręczna L1	96KB	128KB	128KB
Pamięć podręczna L2	-	256KB	512KB
Pamięć podstawowa	SDRAM (66-133MHz)	DDR-SDRAM (100-166MHz)	DDR-SDRAM (100-66MHz)
Rozszerzenie pamięci		SDRAM (66-133MHz)	SDRAM (66-133MHz)
Mostek północny	Zintegrowany	Zintegrowany	Zintegrowany
Obudowa	474 BGA	474 BGA	474 BGA

Tablica 6: Porównanie podstawowych parametrów aktualnie produkowanych modeli



Rysunek 9: Fotografie procesorów Crusoe TM3200 oraz TM5400.

5 Zapowiadane modele

W styczniu 2001 roku na stronie Transmety pojawił się dokument *Enabling a New World of Mobile Internet Computing* [5]. Znaleźć w nim można enigmatyczne informacje na temat nowych modeli procesora Crusoe planowanych na najbliższy okres. Widać wyraźnie, że jak na razie kontynuowane będą dwie linie procesorów:

- TM5x00
  Linia zapoczątkowana procesorami TM5400 i TM5600, przeznaczonymi głównie do pracy w środowisku Windows. Kontynuatorem ma być procesor TM5800 wykonany w technologii $0,13\mu$m (poprzednie modele wykonane były w technologii $0,18\mu$m). Ponadto zwiększona ma być ilość pamięci podręcznej L2 (nawet do 1MB) oraz częstotliwość pracy (większa od 700MHz).

- TM3x00
  Procesory te przeznaczone są głównie do pracy w środowisku Mobile Linux. Kolejnymi procesorami po TM3200 mają być TM3300 i TM3400, wykonane w technologii $0,18\mu$m, pracujące z większą częstotliwością i posiadające większą ilość pamięci podręcznej (procesor TM3400 zostanie wzbogacony nawet o pamięć podręczną L2).

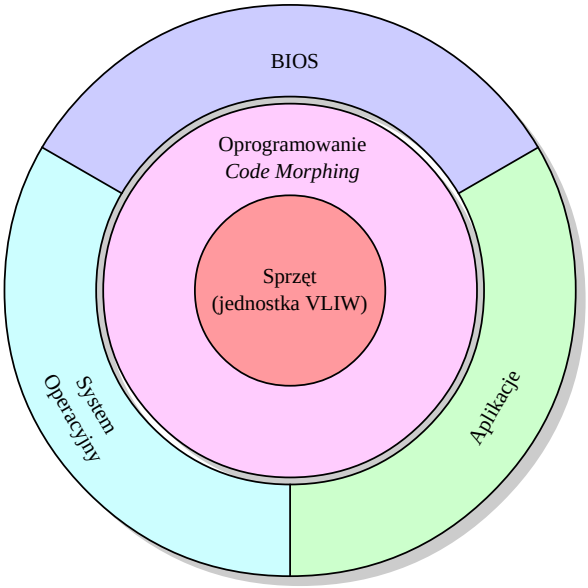
W tablicy 7 zestawiono podstawowe parametry zapowiadanych modeli.

	TM3300	TM3400	TM5800
Technologia	0,18μm	0,18μm	0,13μm
Zakres częstotliwości	400MHz	400-500MHz	> 700MHz
Pamięć podręczna L1	128KB	?	?
Pamięć podręczna L2	?	256KB	512KB-1MB

Tablica 7: Porównanie podstawowych parametrów zapowiadanych modeli

6 Oprogramowanie Code Morphing

Oprogramowanie Code Morphing jest przede wszystkim systemem dynamicznej translacji, programem który kompiluje instrukcje przeznaczone dla jednej architektury sprzętowej (na przykład instrukcje x86) na instrukcje przeznaczone dla innej architektury sprzętowej (w tym przypadku dla procesora VLIW operującego na 128 bitowych instrukcjach). Oprogramowanie Code Morphing jest zainstalowane w pamięci ROM i jest pierwszym programem jaki zostaje uruchomiony podczas inicjalizacji procesora. Oprogramowanie Code Morphing jest jedyną rzeczą widzianą przez kod x86. Jedynym programem napisanym specjalnie dla sprzętowego jądra VLIW jest właśnie oprogramowanie Code Morphing. Rysunek 10 przedstawia zależność pomiędzy kodem x86, oprogramowaniem Code Morphing oraz samym procesorem Crusoe.



Rysunek 10: Oprogramowanie Code Morphing jest jedynym pośrednikiem pomiędzy sprzętową jednostką VLIW a oprogramowaniem zgodnym z x86.

Ponieważ oprogramowanie Code Morphing oddziela oprogramowanie x86 (włączając w to BIOS oraz system operacyjny) od naturalnego zestawu instrukcji jednostki sprzętowej, tenże zestaw instrukcji może zostać odgórnie zmieniony, w ogóle nie wpływając na funkcjonalność oprogramowania x86. Jedynym programem, który musi być napisany dla nowego zestawu instrukcji jest właśnie oprogramowanie Code Morphing, a czynność ta jest wykonywana jednorazowo przez producenta przy okazji każdej zmiany architektury wewnętrznej procesora. Możliwość taka została już zresztą przez Transmetę wykorzystana: zestaw instrukcji modelu TM5400 jest rozszerzoną wersją (niekompatybilną w żadną stronę) zestawu instrukcji modelu TM3200 i z tego też względu, w obu modelach pracują inne wersje oprogramowania Code Morphing. Modele te różnią się między sobą, ponieważ są przeznaczone do różnych celów: model TM3200 przeznaczony jest do stosowania w przenośnych ultralekkich urządzeniach specjalizowanych, pracujących głównie pod kontrolą systemu Mobile Li-

nux: WebPadach, komunikatorach, palmtopach itp. Natomiast model TM5400 przeznaczony jest do notebooków oraz subnotebooków pracujących głównie w środowisku Windows; charakteryzuje się przy tym wysoką wydajnością porównywalną z Pentium III 500Mhz.

Ukrycie sprzętu wraz ze swoim specyficznym zestawem instrukcji pod powłoką warstwy programowej pozwala uniknąć problemu, który w przeszłości hamował powszechny rozwój architektur VLIW. W przypadku tradycyjnego procesora VLIW wszystkie szczegóły jego architektury (potoki itp.) muszą być znane kompilatorowi, dlatego też jakakolwiek zmiana architektury pociąga za sobą konieczność przekompilowania wszystkich istniejących programów binarnych tak, aby mogły być uruchomione na nowym sprzęcie. Warto zwrócić uwagę, że nawet tradycyjne procesory x86 cierpią na podobny problem: mimo, że stare programy będą działać poprawnie na nowym procesorze, zwykle wymagają ponownej kompilacji, aby mogły w pełni wykorzystać to, co oferuje nowa implementacja procesora. Ten problem nie istnieje jednakże w przypadku procesorów Crusoe, w których warstwa programowa Code Morphing zawsze „przekompilowuje" i optymalizuje kod x86, który jest wykonywany.

Elastyczność jaką ze sobą niesie programowe przekształcanie kodu nie jest niestety za darmo: procesor musi przeznaczyć część swoich cykli na wykonywanie oprogramowania Code Morphing. Cykli, które tradycyjny procesor x86 przeznaczyłby na wykonywanie aplikacji. Aby osiągnąć wystarczająco dobrą wydajność procesora, firma Transmeta bardzo szczegółowo zaprojektowała oprogramowanie Code Morphing, w celu dostarczenia maksymalnej wydajności przy minimalnym koszcie.

6.1 Granica pomiędzy sprzętem a oprogramowaniem

Stworzenie wirtualnego procesora x86 jest dużym wyzwaniem ze względu na złożoność architektury x86. Wybór tego, które funkcje mają zostać zaimplementowane sprzętowo, a które programowo, uzależniony jest od kosztów, złożoności, ogólnej wydajności i zużycia energii. Istnieje wiele możliwości tego wyboru, a uzależniony jest on od popytu czy najnowszych dostępnych technologii.

W swoich pierwszych procesorach, firma Transmeta zdecydowała się przeznaczyć oprogramowanie do złożonego procesu dekodowania instrukcji x86 i generowania równoległych molekuł kodu, które sprzęt wykonuje wykorzystując bardzo prostą i szybką jednostkę VLIW. Warstwa programowa jest dodatkowo wspomagana przez kilka funkcji sprzętowych. Granica pomiędzy sprzętem a oprogramowaniem może przebiegać w innym miejscu, w zależności od konkretnego produktu.

6.2 Dekodowanie i przemieszczanie instrukcji

Konwencjonalne superskalarne procesory x86 pobierają po kolei instrukcje z pamięci, następnie zamieniają je na mikrorozkazy, których kolejność jest zamieniana, i które następnie są wykonywane równolegle przez różne jednostki. Natomiast oprogramowanie Code Morphing (będące rozwiązaniem programowym) potrafi przekształcić cały blok kodu x86 na raz, podczas gdy superskalarny procesor x86 przekształca osobno każdą instrukcję. Ponadto procesor x86 przekształca instrukcję za każdym razem, gdy jest ona wykonywana, natomiast oprogramowanie Code Morphing przekształca instrukcję *raz*, zapisując wynikowy kod

(VLIW) w pamięci podręcznej translatora. Następnym razem, gdy instrukcja x86 (już przekształcona) ma być wykonana, system omija proces przekształcania i bezpośrednio wykonuje już istniejący, zoptymalizowany kod, umieszczony w pamięci podręcznej. Zaimplementowanie procesu przekształcania w warstwie programowej otwiera nowe możliwości i wyzwania. Ponieważ tradycyjny procesor musi przekształcać instrukcje za każdym razem, gdy je wykonuje, musi to robić bardzo szybko. To poważnie ogranicza ilość transformacji, których może dokonać. Rozwiązanie programowego przekształcania kodu może zamortyzować koszty translacji wielokrotnie wykonywanego bloku kodu, pozwalając na użycie bardziej wyrafinowanych metod przekształcania kodu. Ponadto ilość zużytej na translację energii jest niewielka w porównaniu do tradycyjnego procesora, który dokonuje translacji za każdym razem. Wreszcie oprogramowanie przekształcające kod może *zoptymalizować* wygenerowany kod zmniejszając ilość instrukcji, które mają zostać wykonane. Innymi słowy: oprogramowanie Code Morphing zwiększa szybkość wykonywania się programów, zmniejszając przy tym zużycie energii.

6.3 Wykorzystywanie pamięci podręcznej

Pamięć podręczna translatora wraz z oprogramowaniem Code Morphing rezyduje w osobnej przestrzeni pamięci, która jest niedostępna dla kodu x86. (Dla zwiększenia wydajności, podczas inicjalizacji oprogramowanie Code Morphing kopuje się z pamięci ROM do pamięci DRAM). Rozmiar tej przestrzeni adresowej może być ustalony podczas inicjalizacji, może również być zmieniany przez system operacyjny.

Oprogramowanie Code Morphing używa pamięci podręcznej do wielokrotnego wykorzystywania już przekształconego kodu. Gdy dany blok kodu ma być powtórnie wykonany, a był już uprzednio przekształcony i umieszczony w pamięci podręcznej, zostaje wykonana z maksymalną prędkością właśnie ta jego przekształcona wersja.

Niektóre programy mierzące wydajność procesora wykonują wiele zróżnicowanych procedur, które mają niewielką powtarzalność. Koszty translacji kodu, którą wykonuje oprogramowanie Code Morphing są w takim przypadku bardzo uwypuklone. Ponadto wraz z czasem wykonywania się aplikacji, oprogramowanie Code Morphing „uczy się” programu i coraz bardziej go udoskonala prowadząc do coraz szybszego czasu wykonywania się. Dzisiejsze programy testujące wydajność nie były pisane z myślą o procesorach, które wykonują programy coraz szybciej; oceniają więc czas poświęcony głównie na uczenie się aplikacji a nie faktyczny zysk, jaki to przynosi. W rezultacie niektóre programy testujące niezbyt dokładnie wyznaczają wydajność procesorów Crusoe.

W przypadku typowych aplikacji, w których bloki kodu wykonywane są bardzo często, oprogramowanie Code Morphing ma szansę zoptymalizować wykonywany kod i zamortyzować koszty wstępnej translacji. Jako przykład może posłużyć program odtwarzający DVD - zanim zostanie wyświetlona pierwsza klatka filmu, kod dekodera DVD będzie już całkowicie przekształcony i zoptymalizowany, procesor nie będzie już tracił czasu na translację kodu podczas samego odtwarzania filmu. Rozwiązanie polegające na wielokrotnym wykorzystywaniu raz przekształconego kodu oferuje bardzo wysoką wydajność w przypadku rzeczywistych aplikacji.

6.4 Filtrowanie

Powszechnie wiadomo, że bardzo niewielka część kodu programu (w większości przypadków mniej niż 10%, czasem nawet tylko 1%) odpowiada za 95% ogólnego czasu wykonywania się programu. Dlatego system translacji kodu musi rozważnie dobierać wielkość czasu, który powinien przeznaczyć na translację i optymalizację kodu x86. Oczywiście najlepiej by było, gdyby najwięcej uwagi poświęcał na najczęściej wykonywane fragmenty kodu i nie marnował czasu procesora na optymalizację kodu, który zostanie wykonany tylko raz.

Oprogramowanie Code Morphing dysponuje szeroką gamą możliwości przekształcania kodu x86; począwszy od interpretacji (gdzie koszty translacji są niewielkie, aczkolwiek kod x86 jest wykonywany wolniej), poprzez prostą translację (wykorzystującą bardzo proste metody generacji kodu), aż do bardzo złożonej optymalizacji (która mimo iż zabiera najwięcej czasu, prowadzi do dużo szybszego wykonywania się zoptymalizowanego kodu). Decyzja o tym, w jaki sposób przekształcać następne fragmenty kodu podejmowana jest na podstawie informacji zebranych z poprzednio wykonanych fragmentów.

6.5 Przewidywanie skoków i wybór ścieżki

Jedną z wielu metod używanych przez oprogramowanie Code Morphing do zbierania informacji o aktualnie wykonywanym kodzie jest dodawanie specjalnych instrukcji do kodu wytworzonego w czasie translacji, których jedynym celem jest zbieranie informacji o częstotliwości wykonywania danego bloku programu, lub np. o historii skoków wykonywanych w danym bloku. Dane te wykorzystywane są później przy podejmowaniu decyzji które części programu mają zostać zoptymalizowane i przekształcane, oraz kiedy ma to nastąpić. Na przykład jeśli dany skok warunkowy x86 jest częściej wykonywany, system tak zoptymalizuje kod, że lepiej będzie zoptymalizowany właśnie ten blok kodu najczęściej wykonywany. Z drugiej zaś strony, dla skoków które nie są od razu tak jednoznaczne (np. wykonywane w połowie przypadków), translator może zadecydować aby wykonać oba bloki kodu, a dopiero później wybrać właściwy blok do zoptymalizowania. Informacja o tym jak często dany blok kodu x86 jest wykonywany, pomaga w podejmowaniu decyzji w jakim stopniu blok ten należy zoptymalizować. Podejmowanie takich decyzji w tradycyjnych sprzętowych implementacjach x86 byłoby niezmiernie trudne.

6.6 Dokonywanie przekształcenia

Poniżej zamieszczony jest prosty przykład ilustrujący w jaki sposób oprogramowanie Code Morphing zamienia blok instrukcji x86 na równoważny kod VLIW procesora Crusoe. Załóżmy, że algorytmy filtrowania i przewidywania skoków wybrały poniższy blok czterech instrukcji (od A do D) do translacji.

```
A. addl  %eax,(%esp)     // załadować daną ze stosu, dodać do %eax
B. addl  %ebx,(%esp)     // to samo, tylko że dodać do %ebx
C. movl  %esi,(%ebp)     // załadować %esi zawartością komórki pamięci
D. subl  %ecx,5          // odjąć 5 od zawartości rejestru %ecx
```

W pierwszym przejściu system translacji dekoduje instrukcje x86 i przekształca je w prostą sekwencję operacji podstawowych (atomów). Na tym etapie wciąż jest raczej łatwo dostrzec analogię pomiędzy oryginalnym i wygenerowanym kodem. (Rejestry %r30 i %r31 są używane do tymczasowego przechowywania danych wczytywanych z pamięci. Przyrostek „.c” dodany do instrukcji oznacza ustawienie flag).

```
ld    %r30,[%esp]     // załadować daną ze stosu do rejestru tymczasowego
add.c %eax,%eax,%r30  // dodać do %eax (ustawiając odpowiednie flagi)
ld    %r31,[%esp]     // ponownie załadować daną ze stosu
add.c %ebx,%ebx,%r31  // dodać do %ebx
ld    %esi,[%ebp]     // załadować daną z~pamięci do rejestru %esi
sub.c %ecx,%ecx,5     // odjąć 5 od zawartości rejestru %ecx
```

W drugim przejściu optymalizator stosuje powszechnie stosowane przez kompilatory metody optymalizacji, tj. usuwanie powtarzających się wyrażeń, nie zmieniających się pętli, czy niepotrzebnych instrukcji (włączając w to niepotrzebne ustawianie flag). To są przykłady optymalizacji, których sprzętowe implementacje nie mogą zrobić: programowy system translacji jest w stanie właśnie *usunąć* atomy ze strumienia instrukcji, a nie tylko np. zmienić kolejność ich wykonania. W podanym przykładzie wszystkie (oprócz ostatniej) instrukcje ustawiające flagi są niepotrzebne, oraz jeden z atomów ładujących daną z pamięci jest nadmiarowy. Etap ten prowadzi to do zmniejszenia ilości atomów, które muszą zostać wykonane.

```
ld    %r30,[%esp]     // załadować daną ze stosu tylko raz
add   %eax,%eax,%r30  // dodać do %eax
add   %ebx,%ebx,%r30  // do %ebx dodać daną już raz załadowaną
ld    %esi,[%ebp]
sub.c %ecx,%ecx,5     // tylko ta instrukcja ustawia flagi
```

W ostatnim przejściu zamieniana jest kolejność atomów i następuje grupowanie ich w pojedyncze molekuły. Proces ten jest bardzo podobny do tego, który w rzeczywistości wykonują procesory sprzętowe wykonujące kod nie w kolejności (oczywiście w miarę możliwości). Jednakże użycie do tego celu oprogramowania umożliwia użycie bardziej efektywnych algorytmów i branie pod uwagę większych (niż w przypadku rozwiązań sprzętowych) fragmentów kodu. Po tej operacji oryginalny kod x86 został zredukowany do dwóch molekuł:

```
1. ld %r30,[%esp];   sub.c %ecx,%ecx,5
2. ld %esi,[%ebp];   add %eax,%eax,%r30;   add %ebx,%ebx,%r30
```

Należy zwrócić uwagę na dwie rzeczy:

- Mimo iż molekuły są wykonane przez sprzęt w kolejności, wykonują one pracę oryginalnych instrukcji x86 ustawionych nie w kolejności.
- Molekuły te już wykorzystują równoległość na poziomie instrukcji, mogą więc być wykonane przez prostą (przez to szybką i zużywającą niewiele energii) jednostkę VLIW. Sprzęt nie musi przeprowadzać skomplikowanych operacji zamieniania kolejności instrukcji.

6.7 Sprzętowe wsparcie dla oprogramowania Code Morphing

Dynamiczna translacja kodu wykonana przez konwencjonalny procesor nie prowadziłaby do zadowalających wyników. Natomiast procesor Crusoe osiąga doskonałą wydajność ponieważ został zaprojektowany właśnie do współpracy z dynamiczną translacją. Poniżej opisane są trzy proste cechy sprzętowej części procesora Crusoe wpierające wyjątki, optymalizację operacji na pamięci oraz samomodyfikujący się kod.

6.7.1 Obsługa wyjątków

Bez specjalnego wsparcia sprzętowego bardzo trudno jest systemowi dynamicznej translacji kodu poprawnie modelować semantykę wyjątków docelowej architektury zachowując przy tym dużą wydajność. Spowodowane jest to tym, że semantyka wyjątków narzuca pewne zasady wykonywania instrukcji. Przykładowo, następujący fragment kodu x86:

```
A. addl  %eax,(%esp)
B. addl  %ebx,(%esp)
C. movl  %esi,(%ebp)
D. subl  %ecx,5
```

po przekształceniu na instrukcje jednostki VLIW, wygląda następująco:

```
1. ld %r30,[%esp];   sub.c %ecx,%ecx,5
2. ld %esi,[%ebp];   add %eax,%eax,%r30;   add %ebx,%ebx,%r30
```

Architektura x86 określa w sposób *precyzyjny* sposób postępowania: jeśli jakaś instrukcja powoduje wyjątek, wszystkie instrukcje ją poprzedzające muszą się zakończyć zanim nastąpi zgłoszenie wyjątku, natomiast żadna z instrukcji następujących nie może być wykonana. W podanym powyżej przykładzie atomy pojawiają się w wynikowym kodzie w innej kolejności, niż były w oryginalnym kodzie x86. Wyobraźmy sobie, że wykonanie instrukcji ładującej z pamięci, zawartej w molekule 2 (reprezentującej oryginalną instrukcję C), powoduje błąd ochrony pamięci. Do tego czasu, procesor już zdążył wykonać molekułę 1, w której była instrukcja reprezentująca oryginalną instrukcję D, co narusza precyzyjne reguły wyjątków.

Warto zwrócić uwagę na to, że tradycyjne procesory wykonujące instrukcję nie w kolejności również borykają się z tym problemem. Zwykle wykorzystywane są do tego celu złożone mechanizmy sprzętowe, które opóźniają lub odwracają skutki mikrorozkazów wykonanych „zbyt wcześnie”.

Procesor Crusoe wykorzystuje dużo prostsze rozwiązanie sprzętowe, współpracujące z oprogramowaniem Code Morphing. Wszystkie rejestry przechowujące stan x86 są zdublowane, tzn. istnieją dwie kopie każdego rejestru: robocza oraz zapasowa. Normalne atomy uaktualniają tylko roboczą kopię rejestru. Jeśli wykonywanie przekształconego fragmentu kodu dobiega końca bez pojawienia się wyjątku, zawartość wszystkich rejestrów roboczych kopiowana jest do rejestrów zapasowych przy pomocy operacji *commit*. Jeśli zaś jakaś instrukcja x86 powoduje wyjątek, skutki wykonania instrukcji od początku translacji są odwracane za pomocą operacji *rollback*, która kopiuje zawartość kopii zapasowych rejestrów

spowrotem do rejestrów roboczych. W tym momencie oprogramowanie Code Morphing ponownie wykonuje dany fragment kodu x86 w sposób tradycyjny, tj. bez przestawiania instrukcji, w celu ustalenia położenia instrukcji powodującej wyjątek.

Odwracanie operacji wykonanych na pamięci jest nieco bardziej skomplikowane. Procesor Crusoe przetrzymuje dane przeznaczone do zapisu do pamięci w specjalnym buforze, z którego są do pamięci przepisywane w momencie wykonania operacji *commit*. Jeśli następuje wyjątek, dane po prostu nie są zapisywane z bufora do pamięci.

6.7.2 Operacje na pamięci

Im większą swobodę ma system przekształcający kod w wypełnianiu molekuł pojedynczymi atomami, tym lepszy kod może zwykle wygenerować. Jednym z największych ograniczeń tej wolności są zależności pomiędzy instrukcjami wykonującymi operacje na pamięci. W szczególności, często pożądana jest możliwość umieszczenia instrukcji czytających przed instrukcjami zapisującymi. Jednakże jest to rozwiązanie niepoprawne jeśli instrukcja czytająca wykorzystuje dane zapisane przez poprzedzającą ją instrukcję zapisującą. Ponieważ ciężko jest w czasie translacji sprawdzić, czy jest inaczej, translator zwykle zakłada, że tak jednak jest (problem ten występuje także w tradycyjnych kompilatorach oraz mikroprocesorach).

Procesor Crusoe wykorzystuje innowacyjny mechanizm przeznaczony do rozwiązania tego problemu. Kiedy translator przenosi instrukcję ładującą przed instrukcję zapisującą, konwertuje ją na instrukcję „załaduj i zabezpiecz” (*load-and-protect*), która oprócz załadowania danej zapamiętuje jej adres oraz rozmiar. Instrukcję zapisującą zamienia zaś na instrukcję *store-under-alias-mask*, która dodatkowo sprawdza zabezpieczone obszary pamięci. W przypadku wystąpienia operacji zapisu, która nadpisuje uprzednio załadowaną daną (co się jednak zdarza rzadko), procesor zgłasza wyjątek, który umożliwia podjęcie sekwencji poprawiającej. Wykorzystując ten mechanizm, zamienianie kolejności instrukcji operujących na pamięci jest zawsze bezpieczne. Sprzętowa część procesora dostarcza bowiem bardzo prostego mechanizmu sprzętowego, który wraz z oprogramowaniem pozwala rozwiązać ten męczący problem.

Mechanizm ten pozwala na znacznie więcej niż tylko przestawianie atomów: pomaga usuwać nadmiarowe operacje ładowania/zapisu. Rozpatrzmy przypadek, w którym dana ładowana jest z pamięci dwukrotnie, a pomiędzy tymi operacjami występuje operacja zapisu (taka sekwencja instrukcji występuje bardzo często w procesorach o małej liczbie rejestrów, jak np. x86):

```
ld    %r30,[%x]       // załadowanie danej spod adresu X
...
st    %data,[%y]      // może nadpisać lokację X
ld    %r31,[%x]       // ponowne załadowanie spod lokacji X
use   %r31
```

Jeśli tylko środkowa operacja zapisu nie nadpisuje danej załadowanej za pierwszym razem, drugie ładowanie jest niepotrzebne, lecz zbyt często translator bądź kompilator nie potrafi tego sprawdzić. Użycie poznanego wcześniej mechanizmu pozwala na zabezpieczenie pierwszego ładowania i wyeliminowania drugiego:

```
ldp  %r30,[%x]       // załadowanie spod X~i zabezpieczenie
...
stam %data,[%y]      // ten zapis powoduje wyjątek, jeśli nadpisuje X
use  %r30            // tu można użyć danej z~pierwszego ładowania
```

Można zauważyć, że użycie wcześniej załadowanej danej może być już wcześniej zaplanowane, jeszcze bardziej przyśpieszając wygenerowany kod. Żaden tradycyjny procesor, wykonujący instrukcję nie w kolejności, nie potrafi dokonywać podobnych rzeczy.

6.7.3 Obsługa samomodyfikującego się kodu

Czasami instrukcje x86 rezydujące w pamięci są nadpisywane, gdy system operacyjny ładuje nowy program, bądź gdy program używa kodu samomodyfikującego się. Jeśli dzieje się to z kodem, który już został przekształcony, oprogramowanie Code Morphing musi zostać o tym poinformowane, aby nie wykonywać już nieaktualnego, przekształconego starego kodu. Dzieje się to w sposób następujący: jeśli system przekształca blok kodu x86, zabezpiecza ten obszar pamięci przed zapisem. Dokonywane jest to przez ustawienie specjalnego bitu odpowiadającego danej stronie pamięci w jednostce zarządzającej pamięcią. (Bit ten, jak również inne szczegóły procesora VLIW, jest niedostępny dla kodu x86). Jeśli występuje zapis do zabezpieczonego obszaru pamięci, odpowiedni blok przekształconego kodu jest po prostu unieważniany. Podczas wykonywania się programu, system uczy się jego zachowania i przechodzi do coraz bardziej wyrafinowanych strategii.

6.8 Przykładowa rzeczywista translacja

Opis technologii translacji kodu zamknięty zostanie nieco większym przykładem wziętym z rzeczywistej aplikacji x86 pracującej w systemie Windows NT, obrazującym wyrafinowane możliwości oprogramowania Code Morphing. Następujące 20 instrukcji kodu x86 (które w tradycyjnym procesorze wymagałyby wykonania ponad dwudziestu mikrooperacji):

```
 1.          movl  %ecx,$0x3
 2.          jmp   lbl1
             ...
 3. lbl1:    movl  %edx,0x2fc(%ebp)
 4.          movl  %eax,0x304(%ebp)
 5.          movl  %esi,$0x0
 6.          cmpl  %edx,%eax
 7.          movl  0x40(%esp,1),$0x0
 8.          jle   skip1
 9.          movl  %esi,$0x1
10. skip1:   movl  0x6c(%esp,1),%esi
11.          cmpl  %edx,%eax
12.          movl  %eax,$0x1
13.          jl    skip2
```

```
14.         xorl  %eax,%eax
15. skip2:  movl  %esi,0x308(%ebp)
16.         movl  %edi,0x300(%ebp)
17.         movl  0x7c(%esp,1),%eax
18.         cmpl  %esi,%edi
19.         movl  %eax,$0x0
20.         jnl   exit1
    exit2:
```

zostało zamienione na poniższe 10 instrukcji VLIW:

```
 1. addi %r39,%ebp,0x2fc
 2. addi %r38,%ebp,0x304
 3. ld   %edx,[%r39];     add %r27,%r38,4;     add %r26,%r38,-4
 4. ld   %r31,[%r38];     add %r35,0,1;        add %r36,%esp,0x40
 5. ldp  %esi,[%r27];     add %r33,%esp,0x6c;  sub.c %null,%edx,%r31
 6. ldp  %edi,[%r26];     sel #le,%r32,0,%r35
 7. stam 0,[%r36];        sel #l,%r24,%r35,0;  add %r25,%esp,0x7c
 8. stam %r32,[%r33];     add %ecx,0,3;        sub.c %null,%esi,%edi
 9. st   %r24,[%r25];     or %eax,0,0;         brcc #lt,<exit2>
10. br <exit1>
```

Warto zwrócić uwagę na kilka interesujących rzeczy:

- Instrukcja x86 `JMP` bezwarunkowego skoku nie ma odpowiednika w przekształconym kodzie: algorytm wyboru ścieżki po prostu „podążył” za rozgałęzieniem kodu i kontynuował translację od miejsca, na które wskazywała instrukcja `JMP`.
- Nazwy rejestrów zostały zmienione programowo; nie ma potrzeby aby tę funkcję realizował sprzęt (i zużywał przez to więcej energii).
- Kolejność wykonywania instrukcji została zmieniona w porównaniu z oryginalnym kodem.
- Translator zamienił dwa wewnętrzne skoki warunkowe specjalną instrukcją „wybierającą” (która warunkowo wybiera jedną z dwóch możliwości). W wyniku tego, system Code Morphing wykonuje oba rozgałęzienia kodu i dopiero później wybiera poprawną gałąź. Zmniejszenie ilości rozgałęzień jest bardzo pożądane, ponieważ często powodują one nieefektywność przetwarzania potokowego przez procesor. Nie znane są procesory wykonujące instrukcje nie w kolejności, które całkowicie wyeliminowałyby warunkowe rozgałęzienia kodu.
- W translacji został użyty mechanizm optymalizujący operacje na pamięci (molekuły 5 do 8) w celu umożliwienia przestawienia operacji ładowania przed operacje zapisu i przez to jeszcze efektywniejszego upakowania kodu.

7 System LongRun

Chociaż podstawowym zadaniem oprogramowania Code Morphing jest zapewnienie kompatybilności z oprogramowaniem x86, dostarcza ono również możliwość współpracy z usługami dostępnymi jedynie w procesorach Crusoe. Przykładem może być system zarządzania energią LongRun, zawarty np. w modelu TM 5400, który jeszcze bardziej minimalizuje, już i tak bardzo niskie zużycie energii.

W przypadku tradycyjnych procesorów x86, przeznaczonych do urządzeń przenośnych, regulacja zużycia energii sprowadza się do naprzemiennej pracy procesora z pełną prędkością, oraz jego wyłączania. Poprzez zmianę proporcji czasów pracy w obu tych stanach istnieje możliwość regulowania wydajności. Jednakże w przypadku takiego rozwiązania procesor może się nagle wyłączyć właśnie kiedy aplikacja wymaga dużej szybkości działania procesora. Powodować to może np. pomijanie klatek podczas odtwarzania filmu, co jest zjawiskiem dostrzegalnym (i nieprzyjemnym) dla użytkownika.

W przeciwieństwie do takiego rozwiązania procesor Crusoe TM 5400 może dostosować pobór mocy nie poprzez wyłączanie się, lecz przez bardzo szybkie przełączanie częstotliwości zegara w locie. Operacja ta wykonywana jest bardzo szybko bez potrzeby ponownego uruchamiania systemu operacyjnego, czy konieczności przejścia przez powolną procedurę uśpienia i ponownego wystartowania z pamięci RAM. Ponadto oprogramowanie może na bieżąco monitorować wymagania procesora i dynamicznie wybierać stosowną częstotliwość zegara (i związane z tym zużycie mocy) wymaganą do pracy aplikacji - nie za dużą i nie za małą. Ponieważ przełączanie trwa niezwykle szybko, nie jest zauważalne dla użytkownika.

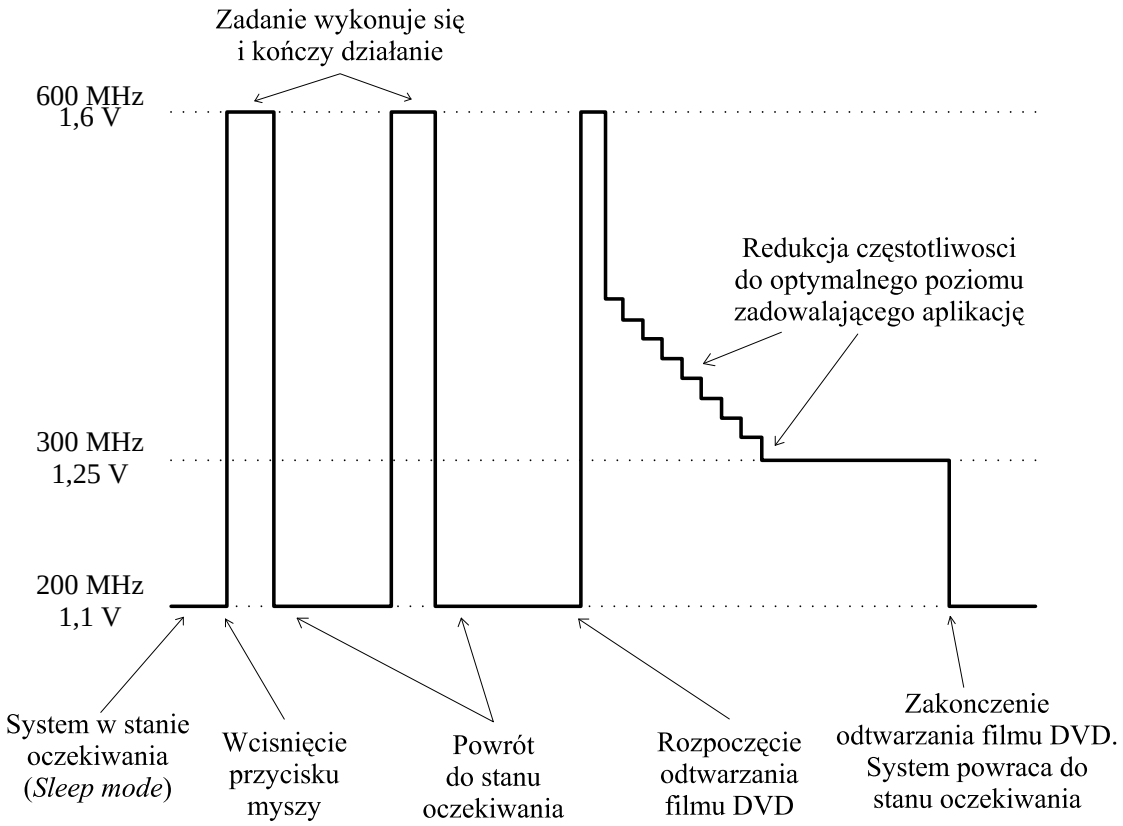
Oprogramowanie Code Morphing może również dostrajać w locie napięcie zasilania procesora (ponieważ przy niższych częstotliwościach pracy może być użyte niższe napięcie). Zużycie mocy zależne jest liniowo od częstotliwości zegara i kwadratowo od napięcia zasilania, dostrajanie obu tych czynników może więc wpłynąć na *sześcienną* redukcję poboru mocy, podczas gdy tradycyjny procesor może regulować zużycie mocy jedynie liniowo. Załóżmy przykładowo, że jakaś aplikacja wymaga jedynie 90% szybkości procesora. W przypadku tradycyjnego procesora zwolnienie o 10% zmniejszy zużycie mocy jedynie o 10%, podczas gdy w tych samych warunkach system LongRun może zmniejszyć moc o prawie 30% - jest to dostrzegalna przewaga.

Rysunek 11 przedstawia przykład działania systemu LongRun - jego wpływ na częstotliwość i napięcie zasilające procesor w zależności od akcji wykonywanych przez użytkownika.

8 Podsumowanie

Opracowany przez firmę Transmeta mikroprocesor zmienia w sposób radykalny dotychczasowe spojrzenie na sposób tworzenia mikroprocesorów. Jak na razie procesory Crusoe produkowane są przede wszystkim z myślą o rynku urządzeń przenośnych, gdzie stosowane procesory zużywają wiele energii zmuszając do wyboru pomiędzy czasem działania baterii a wydajnością urządzenia.

Wyjątkowość procesora Crusoe nie wynika z jakichś wyrafinowanych procesów technologicznych, lecz z „wymyślenia od nowa” podstaw projektowania procesorów. Zamiast



Rysunek 11: Przykład działania systemu LongRun.

projektowania od nowa warstwy sprzętowej procesora zastosowano innowacyjne połączenie sprzętu i oprogramowania. Wykorzystanie oprogramowania do rozkładu złożonych instrukcji na proste instrukcje atomowe oraz do szeregowania i optymalizacji tych atomów tak, aby mogły być wykonywane równocześnie pozwala zaoszczędzić miliony tranzystorów i zmniejszyć zużycie energii o 60%-70% (w porównaniu do standardowych procesorów). Ponadto oprogramowanie to stosuje zaawansowane metody optymalizacji kodu, które są niedostępne w przypadku tradycyjnych procesorów x86. Technologia ta nie jest ograniczona jedynie do instrukcji x86 - wymieniając jedynie oprogramowanie Code Morphing można stworzyć procesor kompatybilny z procesorami Motoroli, a nawet procesor kompatybilny z maszyną wirtualną Javy.

Bibliografia

[1] Alexander Klaiber, *The Technology Behind Crusoe Processors*, Transmeta Corporation, 2/2000. http://www.transmeta.com/crusoe/download/pdf/crusoetechwp.pdf

[2] *Crusoe Processor Model TM3200 Features*, Transmeta Corporation, 5/2000. http://www.transmeta.com/crusoe/download/pdf/TM3200_ProductBrief_5-20-00.pdf

[3] *Crusoe Processor Model TM5400 Features*, Transmeta Corporation, 5/2000. http://www.transmeta.com/crusoe/download/pdf/TM5400_ProductBrief_5-23-00.pdf

[4] *Crusoe Processor Model TM5600 Features*, Transmeta Corporation, 8/2000. http://www.transmeta.com/crusoe/download/pdf/TM5600_ProductBrief_8-2-00.pdf

[5] David R. Ditzel, *Enabling a New World of Mobile Internet Computing*, Transmeta Corporation, 1/2001. http://www.transmeta.com/crusoe/download/pdf/mobile_compute_0100_1-20.pdf http://www.transmeta.com/crusoe/download/pdf/mobile_compute_0100_21-32.pdf

[6] Darek Rzeźnicki, *Jak Crusoe*, PC World Komputer 4/2000, IDG Poland.

[7] Ryszard Sobkowski, *Tajemniczy Crusoe...*, Enter 3/2000, LUPUS.

[8] Jon „Hannibal" Stokes, *Crusoe Explored*. http://arstechnica.com/cpu/1q00/crusoe/

[9] Armin Gerritsen, *The Transmeta Crusoe*, 2/2000. http://cpusite.examedia.nl/docs/crusoe.html

[10] Chris Koiak, Martin McClintock, John Dunn, Ross Byrne, *Crusoe: International Processor of Mystery*, 2/2000. http://friday.editthispage.com/stories/storyReader$247