Dariusz Makowski

Katedra Mikroelektroniki i Technik Informatycznych

tel. 631 2720

dmakow@dmcs.pl

http://neo.dmcs.p.lodz.pl/sw

Zakres przedmiotu

- Systemy mikroprocesorowe, systemy wbudowane
- Rodzina procesorów ARM
- Asembler
- Urządzenia peryferyjne
- Programy wbudowane na przykładzie procesorów ARM
- Metodyki projektowania systemów wbudowanych
- Interfejsy w systemach wbudowanych
- Systemy czasu rzeczywistego

Od Acorn Computers Ltd. ARM do ARM Ltd.

Acorn

- Firma utworzona w 1990 roku,
- Powstała z firmy produkującej komputery Acorn Computers,
- ***Firma ARM nie produkuje procesorów***,
- Projektuje i sprzedaje licencje projektów tzw. rdzeni (**I**ntellectual **P**roperty **C**ores) procesorów ARM oraz urządzeń peryferyjnych,
- Produkuje narzędzia, płyty prototypowe, narzędzia do debugowania, standardy.

Wybrane firmy produkujące procesory zgodne z architekturą ARM

Agi-lent, AKM, Alcatel, Altera, Atmel, Broadcom, Chip Express, Cirrus Logic, Digital Semiconductor, eSilicon, Fujitsu, GEC Plessey, Global UniChip, HP, Hyundai, IBM, Intel, ITRI, LG Semicon, LSI Logic, Lu-cent, Matsushita, Micrel, Micronas, Mitsubishi, Freescale, NEC, OKI, Philips, Qu-alcomm, Rockwell, Rohm, Samsung, Samsung, Sanyo, Seagate, Seiko Epson, Sharp, Sony, STMicroelectronics, Symbios Logic, Texas Instru-ments, Xilinx, Yamaha, Zeevo, ZTEIC, ...

Zastosowanie procesorów ARM

Historia mikroprocesorów ARM

1985 – Sophie Wilson and Steve Furber projektują pierwszy procesor RISC w firmie Acorn Computers Limited, Cambridge, ARM = **A**corn (**A**dvanced) **R**ISC **M**achine

1985 – prototypowe procesory ARM 1 (wersja architektury v1)

1986 – procesory ARM 2 opuszczają fabrykę (32-bit, 26-bit adres, 16 rejestrów 16 bitowych, 30.000 tranzystorów, wersja architektury v2/v2a, 8 MHz)

1980 – Apple Computer i VLSI Technology rozpoczynają współpracę nad kolejną wersja rdzenia procesora ARM (ARMv3)

1990 – wyłania się nowa firma Advanced RISC Machines Ltd. odpowiedzialna za dalszy rozwój procesorów ARM

1991 – pojawia się pierwszy procesor będący wynikiem współpracy z firmą Apple – procesor z jądrem ARM 6 (stosowany w jako procesor ARM 610 w Apple Newton PDA, wersja architektury v3, 33 MHz)

1995 – firma ARM wprowadza na rynek bardzo udany model procesora z rdzeniem **ARM7TDMI** (**architektura ARMv4T**) oraz StrongARM (Intel, 233 MHz)

2001 – firma ARM wprowadza kolejny model procesora z rdzeniem **ARM9TDMI** (**architektura ARMv5TEJ**, 220 MHz)

2003 – procesory serii ARM 11 (archtektura rdzenia ARMv6, 1 GHz, technologia 65 nm)

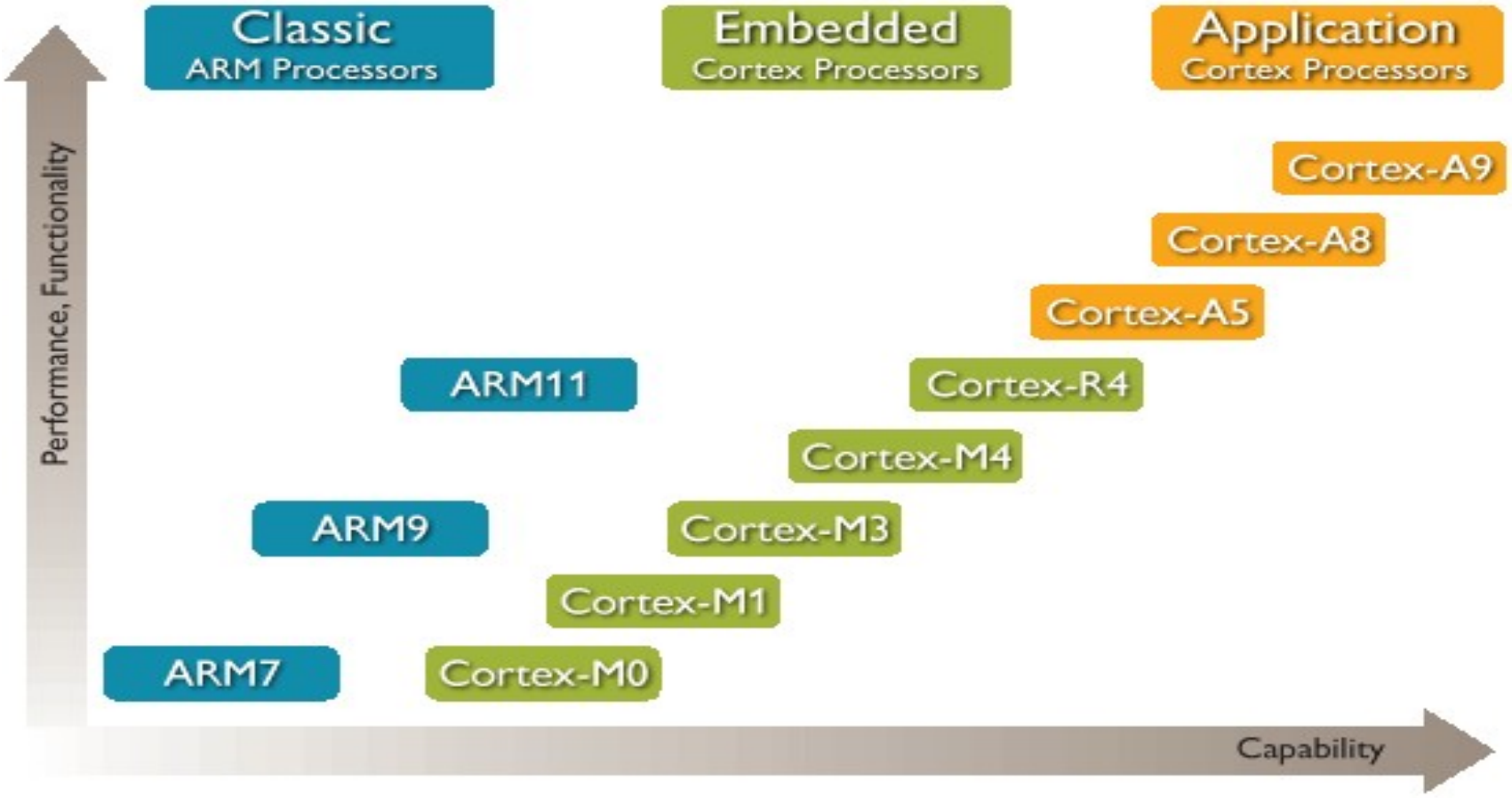
2004 – procesor z rdzeniem Cortex M3 (ARMv7-M, 100 MHz)

2008 – kolejny model procesora **ARM Cordex A8/A9** (architektura **ARMv7**, 2 GHz), **Cortex-R/M**

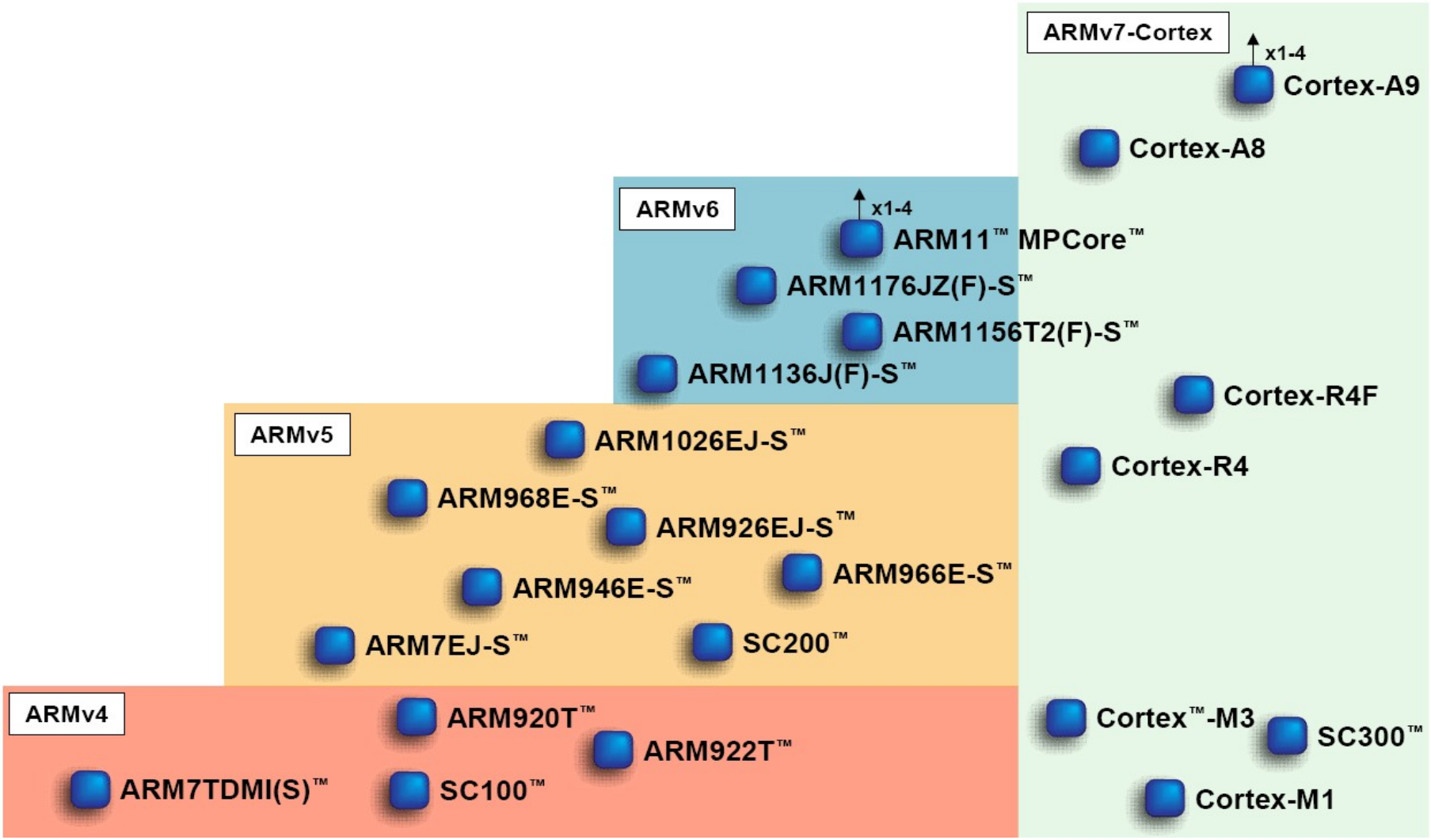
Dostępne wersje architektury rdzenia ARM

- Liczba oraz rodzaj instrukcji, które realizują jednostki wykonawcze dostępnych rdzeni procesorów ARM zależą od wersji rdzenia (V1 - V7).
- Dostępne procesory AMR budowane są w oparciu o następujące wersje architektury:
  - Układy z rdzeniami ARM7, wykorzystywane w aplikacjach o niskim poborze energii, wykorzystują podstawowy zestaw instrukcji udostępniany przez wersję 4T,
  - Układy z rdzeniami ARM9 (ARM926EJ-S, ARM1026EJ-S) zbudowane są w oparciu o architekturę w wersji 5 (instrukcje DSP, rozbudowane instrukcje arytmetyczne MLA, itd...),
  - Układy z rdzeniem ARM11 wykorzystują architekturę instrukcji w wersji 6 (instrukcje operujące na bajtach, instrukcje ułatwiające operacje graficzne),
  - Układy z rodziny Cortex A8/A9, M3, R4 zbudowane są w oparciu o listę instrukcji w wersji 7 (nowy zestaw instrukcji 16 bitowych – Thumb 2, rozbudowany system oszczędzania energii oraz bezpieczeństwa).

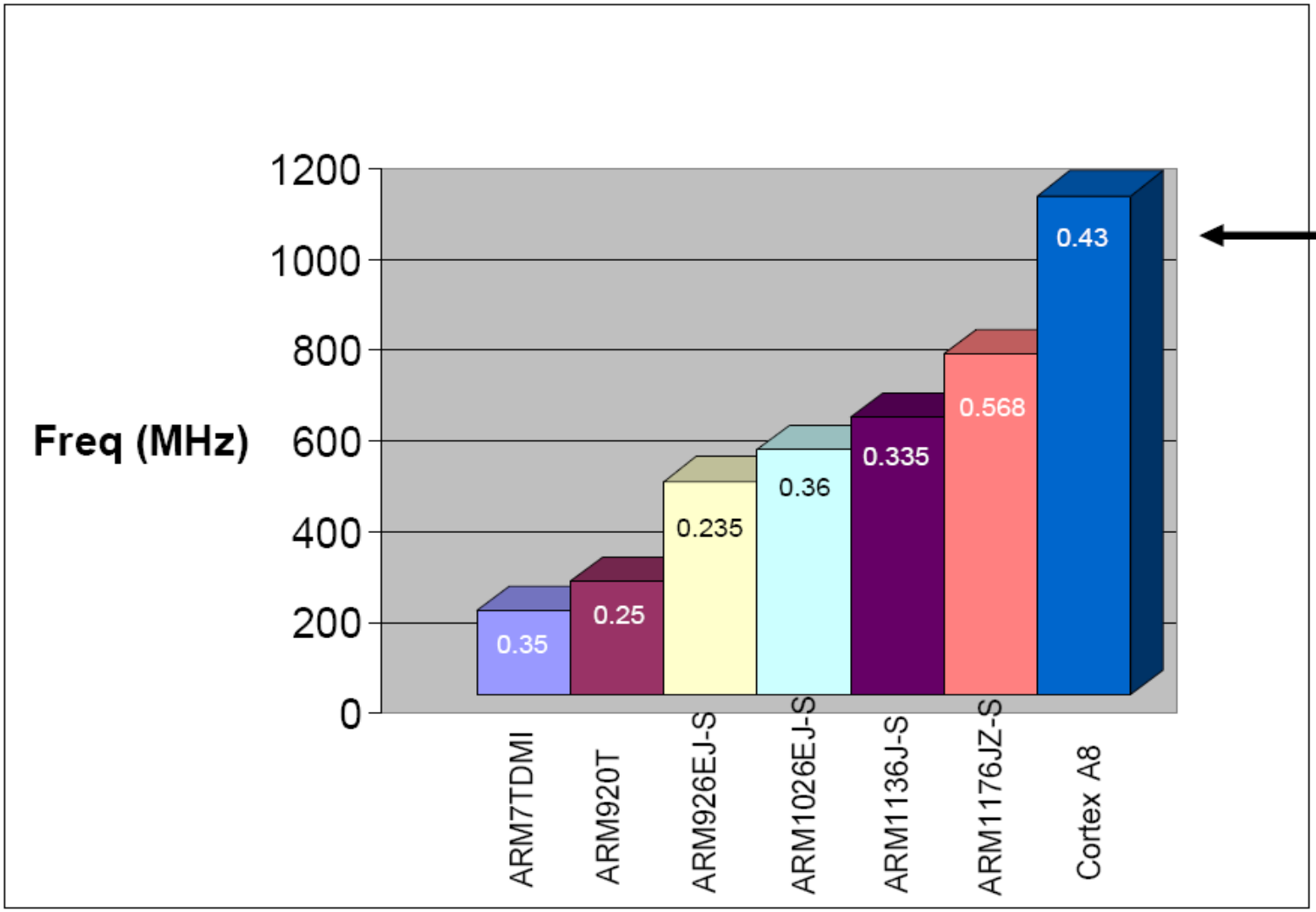
Rodzina procesorów ARM oraz ich zastosowanie



Architektury i dostępne rdzenie procesorów ARM



Porównanie wydajności procesorów

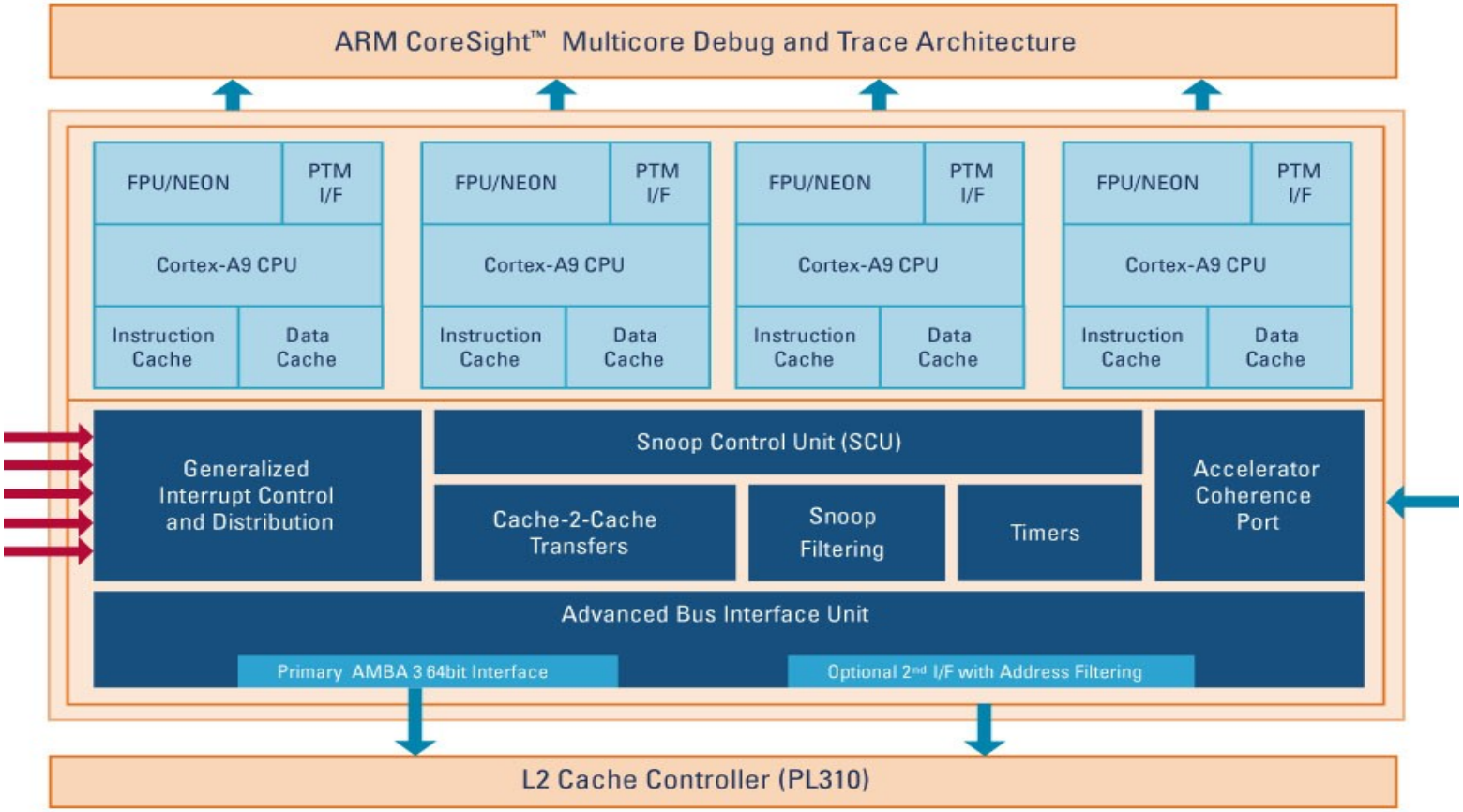


ARM – silna dominacja na rynku

- Firma ARM sprzedaje obecnie ponad 10 milionów rdzeni procesorów każdego dnia (szacuje się, że wyprodukowano ponad 15 miliardów procesorów z rdzeniami ARM).

- **Procesory ARM z rdzeniem w wersji ARMv7 (Cortex):**
  - Procesory do zastosowań komputerowych (Application Processors):
    - **Cortex-A** – silne procesory przeznaczony do obsługi systemów operacyjnych (cyfrowe dekodery, np. HDTV, DVD, komputery Netbook, telefony komórkowe, PDA, smartofony, cyfrowe ramki do zdjęć, itd...),
  - Procesory do zastosowań wbudowanych (Embedded Processors):
    - **Cortex-R** – procesory przeznaczone do zastosowań w systemach czasu rzeczywistego (Real-time Applications, niski pobór mocy, szybka reakcja na zdarzenia zewnętrzne, np. motoryzacja: ABS, urządzenia peryferyjne komputera PC, sterowanie HDD, Ethernet, sterowniki drukarek),
    - **Cortex-M** – procesory będące odpowiednikami mikrokontrolerów (niski pobór mocy, niska cena, np. sterowanie procesami przemysłowymi, inteligentne czujniki, motoryzacja – sterownie procesami innymi niż RT, M3 - tylko Thumb2).

ARM Cortex A9 w konfiguracji MPCore



Technologia MPCore pozwala na budowę SoC – cztery rdzenie procesorów A9

Procesory z rdzeniem ARM

- Procesory ARM są szeroko stosowane w systemach wbudowanych i systemach o niskim poborze mocy, ze względu na energooszczędną architekturę
- Procesor ARM jest jednym z najczęściej stosowanych procesorów na świecie. Jest używany w dyskach twardych, telefonach komórkowych, routerach, kalkulatorach a nawet w zabawkach dziecięcych
- Obecnie zajmuje ponad 75% rynku 32-bitowych CPU dla systemów wbudowanych
- Najbardziej udanym projektem ARM był procesor ARM7TDMI szeroko stosowany w systemach wbudowanych
- Moc obliczeniowa procesorów ARM umożliwia instalację na tym procesorze systemu operacyjnego, z zaimplementowanymi mechanizmami wielowątkowości, z możliwością wykorzystania zawartego w systemie stosu TCP/IP, systemu plików (np. FAT32).
- Systemy operacje: dystrybucje Embedded Linux (Embedded Debian, Embedded Ubuntu), Windows CE, Symbian, NUTOS (Ethernet), ...

Zastosowanie procesorów ARM

Porównanie wybranych procesorów ARM

Family	Architecture Version	Core	Feature	Cache (I/D)/MMU	Typical MIPS @ MHz
ARM6	ARMv3	ARM610	Cache, no coprocessor	4K unified	17 MIPS @ 20 MHz
ARM7	ARMv3	ARM7500FE	Integrated SoC. "FE" Added FPA and EDO memory controller.	4 KB unified	55 MIPS @ 56 MHz
ARM7TDMI	ARMv5TEJ	ARM7EJ-S	Jazelle DBX, Enhanced DSP instructions, 5-stage pipeline	8 KB	120 MIPS @ 133 MHz
StrongARM	ARMv4	SA-110	5-stage pipeline, MMU	16 KB/16 KB, MMU	235 MIPS @ 206 MHz
ARM8	ARMv4	ARM810[7]	5-stage pipeline, static branch prediction, double-bandwidth memory	8 KB unified, MMU	1.0 DMIPS/MHz
ARM9TDMI	ARMv4T	ARM920T	5-stage pipeline	16 KB/16 KB, MMU	245 MIPS @ 250 MHz
ARM9E	ARMv5TEJ	ARM926EJ-S	Jazelle DBX, Enhanced DSP instructions	variable, TCMs, MMU	220 MIPS @ 200 MHz
ARM10E	ARMv5TE	ARM1020E	VFP, 6-stage pipeline, Enhanced DSP instructions	32 KB/32 KB, MMU	300 MIPS @ 325 MHz
XScale	ARMv5TE	PXA27x	MMX and SSE instruction set, four MACs,	32 Kb/32 Kb, MMU	800 MIPS @ 624 MHz
ARM11	ARMv6	ARM1136J(F)-S	SIMD, Jazelle DBX, VFP, 8-stage pipeline	variable, MMU	740 @ 532-665 MHz
Cortex	ARMv7-A	Cortex-A8	Application profile, VFP, NEON, Jazelle RCT, Thumb-2, 13-stage superscalar pipeline	variable (L1+L2), MMU+TrustZone	>1000 MIPS@ 600 M-1 GHz

Rdzeń procesora ARM

Architektura ARM (1)

Rdzeń procesora ARM – procesor zgodny z architekturą ARM zaprojektowany w języku opisu sprzętu (najczęściej VHDL lub Verilog) dostarczony jako makrokomórka (ang. macro-cell) lub IP (ang. Intellectual Property).

Cechy rdzeni procesorów ARM:

- Przeznaczony do dalszej rozbudowy – procesory, SoC
- 32-bitowy procesor zgodny z architekturą RISC
- Wbudowana jednostka zarządzania pamięcią MMU
- Zoptymalizowany pod względem niskiego poboru mocy
- Różne tryby pracy:
  - 32-bitowe instrukcje ARM
  - 16-bitowe instrukcje Thumb
  - Instrukcje języka Java - Jazelle DBX
- Praca w trybie Big lub Little Endian
- Szybka obsługa przerwań (FIR – Fast Interrupt Response), aplikacje czasu rzeczywistego
- Pamięć wirtualna
- Lista wydajnych instrukcji (zoptymalizowane na podstawie architektury RISC oraz CISC)
- Sprzętowe wsparcie dla języków wyższego poziomu

Architektura ARM (2)

Rdzeń procesora ARM wykorzystuje architekturę RISC:

- Zredukowana liczba instrukcji,
- Brak bezpośredniego odwołania do pamięci – tylko instrukcje operujące Load/Store na pamięci,
- Duża liczba dostępnych rejestrów ogólnego przeznaczenia,
- Architektura superskalarna.

Różnice w odniesieniu do klasycznych procesorów RICS:

- Instrukcje wzbogacone o dodatkowe funkcje:
  - Instrukcje Thumb/Thumb2,
  - Instrukcje DSP,
  - Warunkowe wykonywanie instrukcji,
  - 32 bitowy przesuwnik bitowy,
- Instrukcje umożliwiające operacje na wielu rejestrach,
- Tryby adresowania z pre-aktualizacją i post-aktualizacją.

Architektura ARM (3)

Nomenklatura:

ARM {x} {y} {z} {T} {D} {M} {I} {E} {J} {F} {S}

- **x** – rodzina rdzenia
- **y** – zarządzanie pamięcią/ochrona pamięci (opcjonalnie)
- **z** – pamięć cache (opcjonalnie)
- **T** – Thumb, rdzeń może wykonywać 16-bitowe rozkazy
- **D** – Debug, rdzeń wyposażony w moduł JTAG umożliwiający debugowanie
- **M** – Multiplier, wyposażony w sprzętowy układ mnożący (32x32 = 64 bit)
- **I** – ICE, moduł pozwalający na debugowanie w układzie (ang. In-Circuit Emulator, breakpoints)
- **E –** Enhanced DSP instructions, instrukcje przydatne podczas przetwarzania sygnałów
- **J** – Jazelle,
- **F** – Floating-point, moduł wspomagający obliczenia zmiennoprzecinkowe
- **S** – Synhesizible version, wersja sytezowalna – dostępne kody źródłowe dla narzędzi EDA

Przykłady:

ARM7TDMI **ARM9TDMI-EJ-S**

Architektura ARM (4)

- **Wersja 1, v1**
  - podstawowe operacje arytmetyczno/logiczne,
  - programowe przerwania,
  - 8 i 32 bitowe operacje na danych,
  - 26-bitowy adres
- **Wersja 2, v2**
  - jednostka mnożąca oraz mnóż i sumuj (MAC),
  - dostępny koprocesor,
  - operacje służące do synchronizacji wątków,
  - 26-bitowy adres
- **Wersja 3, v3**
  - nowe rejestry CPSR, SPSR, MRS, MSR,
  - dostępne dodatkowe tryby pracy Abort i Undef,
  - 32-bitowy adres

Architektura ARM (5)

- **Wersja 4, v4**
  - pierwsza oficjalnie zatwierdzona architektura
  - w prowadzono 16-bitowe operacje na danych
  - tryb pracy THUMB
  - dodatkowy uprzywilejowany tryb pracy procesora (privileged mode)
  - możliwość inkrementacji PC o podwójne słowo (64 bit)
- **Wersja 5, v5**
  - ulepszona współpraca procesora pomiędzy trybami ARM/THUMB, możliwość zmiany trybu procesora w czasie wykonywania programu
  - dodana instrukcja CLZ
  - możliwość użycia programowych pułapek
  - wsparcie dla pracy wieloprocesorowej
- **Wersja 6, v6**
  - ulepszona jednostka zarządzania pamięcią MMU (Management Memory Unit)
  - sprzętowe wsparcie dla przetwarzania dźwięku i obrazu (FFT, MPEG4, SIMD, etc...)
  - ulepszona obsługa wyjątków (nowa flaga w rejestrze PSR)

Rozkazy procesora ARM

Z punktu widzenia wykonywanych instrukcji procesor ARM może pracować w jednym z następujących trybów:

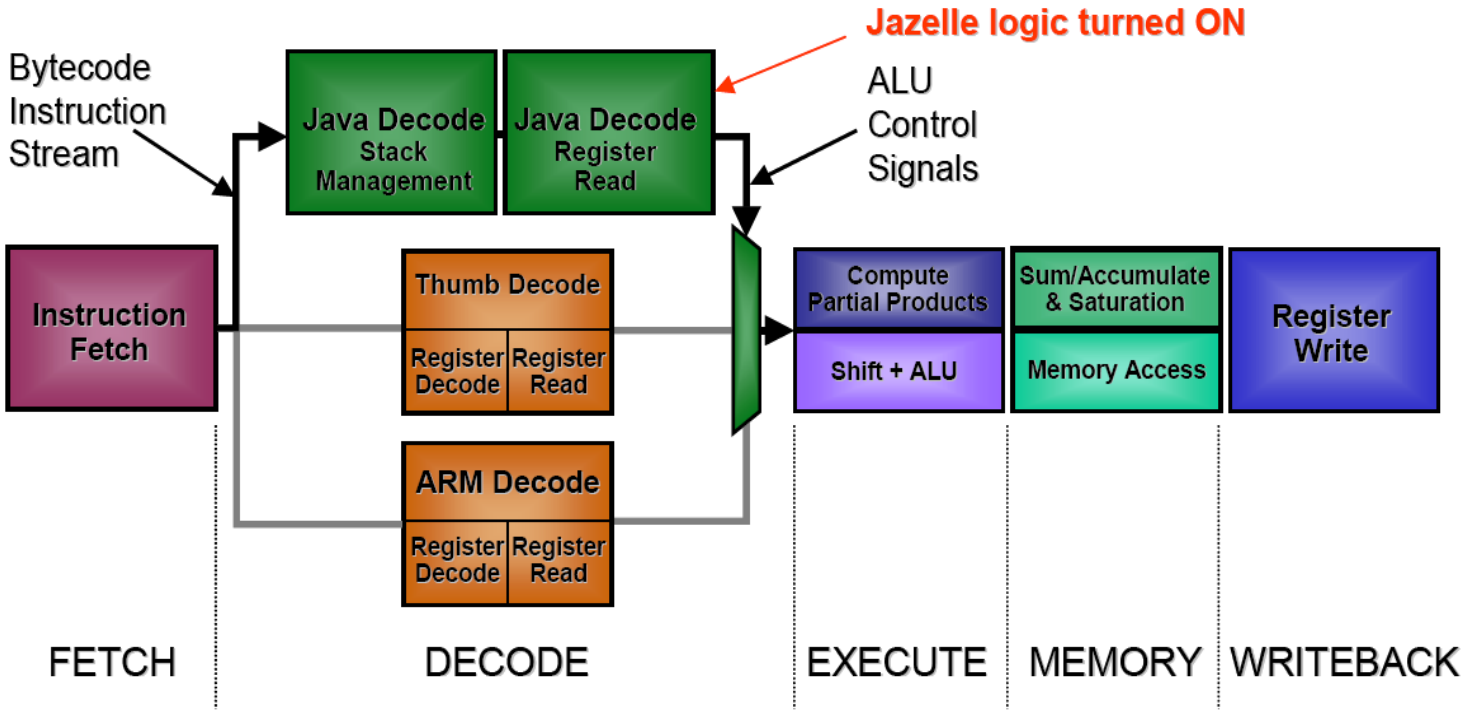
- **ARM** – definiuje 32-bitowe instrukcje (kod programu musi być wyrównany do granicy 4 bajtów),
- **Thumb**, **Thumb-2** – definiuje zestaw 16-bitowych instrukcji zoptymalizowanych pod kątem (kod programu musi być wyrównany do granicy 2 bajtów, wszystkie rejestry pracują w trybie 32 bitowym),
- **Jazelle v1** – tryb pozwalający na bezpośrednie wykonywanie instrukcji zgodnych ze specyfikacją języka Java (bez użycia maszyny wirtualnej JVM, 1000 Caffeine Marks @ 200MHz)

Table A2-3 The T and J bits

J	T	Instruction set
0	0	ARM
0	1	Thumb
1	0	Jazelle
1	1	RESERVED

Wsparcie dla języka Java

- Oznaczenie rdzenia procesora z literą 'J'
- Dynamiczne podmiana rejestrów oraz stosu
- Sprzętowa realizacja dekodera instrukcji – nie maszyna wirtualna



Model programowy – rejestry procesora

Procesor ARM posiada łącznie 37 rejestrów (wszystkie są 32 bitowe):

- **PC (r15)** – licznik programu (Program Counter)
- **CPSR** – rejestr statusowy, obecny status (Current Program Status Register)
- **SPSR** – rejestr statusowy, dostępne w różnych trybach uprzywilejowania (Saved Program Status Register)
- **LR (r14)** – rejestr powrotu (Link Register), wykorzystywany podczas tworzenia ramki stosu (instrukcje skoku do funkcji)
- **SP (r13)** – zwykle używany jako wskaźnik stosu (Stack Pointer)
- **r0 - r12** – rejestry ogólnego przeznaczenia
- **r0-r7 –** rejestry nie są bankowane
- **r8-r14 –** rejestry bankowane (częściowo podmieniane podczas pracy w różnych trybach)

Uwaga :

- Nie wszystkie rejestry są dostępne w różnych trybach uprzywilejowania procesora

Licznik programu

W przypadku wykonywania instrukcji ARM:

- Wszystkie instrukcje są 32 bitowe,
- Wszystkie instrukcje muszą być wyrównane do granicy 4 bajtów,
- PC wykorzystuje tylko 30 bitów (D31..D2).

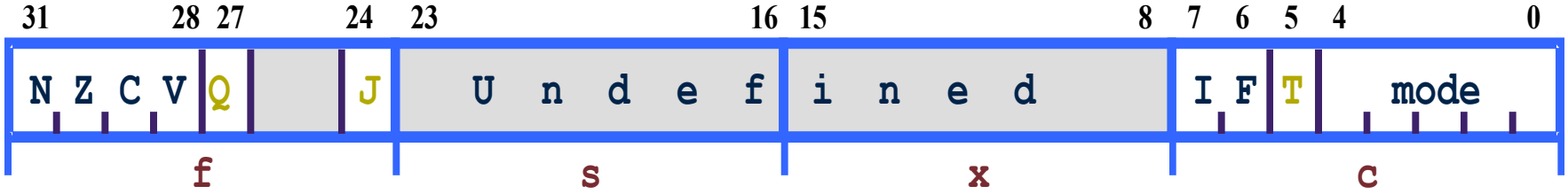
W przypadku wykonywania instrukcji Thumb:

- Wszystkie instrukcje są 16 bitowe,
- Wszystkie instrukcje muszą być wyrównane do granicy 2 bajtów,
- PC wykorzystuje tylko 31 bitów (D31..D1).

W przypadku pracy w trybie Jazelle:

- Wszystkie instrukcje są 8 bitowe,
- Procesor odczytuje zawsze 4 kolejne instrukcje,
- PC wykorzystuje tylko 30 bitów (D31..D2).

Rejestr statusowy SPSR (1)



Wskaźniki stanu

- V – przepełnienie podczas operacji ALU w kodzie U2 (oVerflow)
- C – przeniesienie/pożyczka podczas operacji ALU
- Z – ujemny wynik podczas operacji ALU
- N – ujemny wynik operacji ALU lub „mniejszy niż”

Wskaźniki dostępne jedynie dla architektury 5TE/J

- J – Procesor w trybie Jazelle
- Q – Sticky Overflow – wskaźnik nasycenia podczas operacji ALU (instrukcje DSP: QADD, QDADD, QSUB or QDSUB, lub rezultat operacji SMLAxy or SMLAWx przekracza 32-bity)

Przerwania

- I=1 Przerwania IRQ wyłączone
- F=1 Przerwania FIQ wyłączone

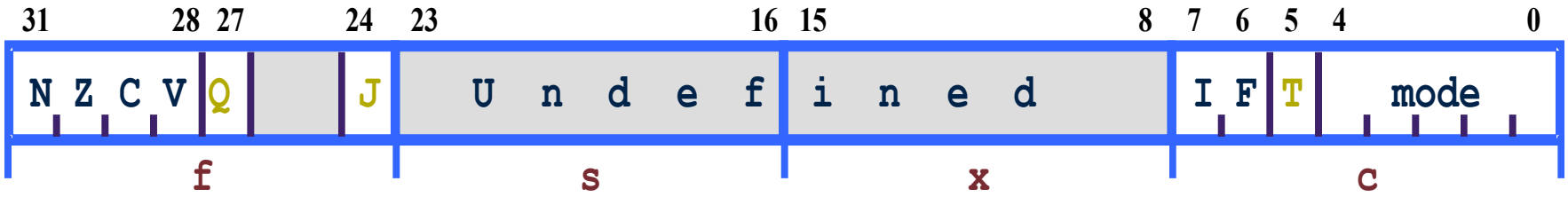
Wskaźniki dostępne dla arch. xT

- T=0 Tryb pracy ARM
- T=1 Tryb pracy Thumb

Tryb pracy procesora

- Definiują jeden z 7 trybów
- Instrukcje operujące na rejestrze CPSR/SPSR:
  - MSR/MRC

Rejestr statusowy SPSR (2)



Nie wolno modyfikować bitów określonych jako undefined. Są one pozostawione do wykorzystania w przyszłych architekturach procesorów ARM.

Próba modyfikacji takich bitów może zakończyć się nieprzewidywalnym zachowaniem programu uruchomionego na nowszej wersji procesora (np. wyjątek).

Model programowy – dostępne tryby pracy procesora

Tryb pracy procesora (operating mode) - określa jakie zasoby procesora są dostępne, np. dostępne rejestry, obszary pamięci, urządzenia peryferyjne.

Procesory ARM mogą pracować w jednym z siedmiu trybów pracy:

- **User –** tryb użytkownika (nieuprzywilejowany) przeznaczony do wykonywania programów użytkownika,
- **FIQ –** tryb obsługujący przerwania i wyjątki o wysokich priorytetach (szybki, r8-r14)
- **IRQ –** obsługa przerwań z niskim priorytetem (low/normal priority)
- Supervisor – tryb pracy superużytkownika, dostęp do wszystkich zasobów procesora (dostępny po resecie lub przerwanie programowe)
- **Abort –** obsługa wyjątków związanych z pamięcią (memory access violations)
- **Undef –** obsługa nieznanych/błędnych rozkazów
- **System –** tryb pracy superużytkownika, dostęp do rejestrów takich jak w trybie User jednak możliwy dostęp do różnych obszarów pamięci

Mode	Abbreviation	Privileged	Mode[4:0]
Abort	abt	yes	10111
Fast interrupt request	fiq	yes	10001
Interrupt request	irq	yes	10010
Supervisor	svc	yes	10011
System	sys	yes	11111
Undefined	und	yes	11011
User	usr	no	10000

Adresy wyjątków

Exception type	Mode	VE[a]	Normal address	High vector address
Reset	Supervisor		0x00000000	0xFFFF0000
Undefined instructions	Undefined		0x00000004	0xFFFF0004
Software interrupt (SWI)	Supervisor		0x00000008	0xFFFF0008
Prefetch Abort (instruction fetch memory abort)	Abort		0x0000000C	0xFFFF000C
Data Abort (data access memory abort)	Abort		0x00000010	0xFFFF0010
IRQ (interrupt)	IRQ	0	0x00000018	0xFFFF0018
		1	IMPLEMENTATION DEFINED	
FIQ (fast interrupt)	FIQ	0	0x0000001C	0xFFFF001C
		1	IMPLEMENTATION DEFINED	

*Adres pod jakim znajduje się tablica funkcji obsługujących wyjątki ustalany jest sprzętowo podczas restartowania procesora (funkcja zależna od architektury, >=ARMv4).

Priorytety wyjątków

Priority		Exception
Highest	1	Reset
	2	Data Abort (including data TLB miss)
	3	FIQ
	4	IRQ
	5	Imprecise Abort (external abort) - ARMv6
	6	Prefetch Abort (including prefetch TLB miss)
Lowest	7	Undefined instruction SWI

Model programowy – rejestry dostępne w trybie User oraz System

Current Visible Registers

User Mode

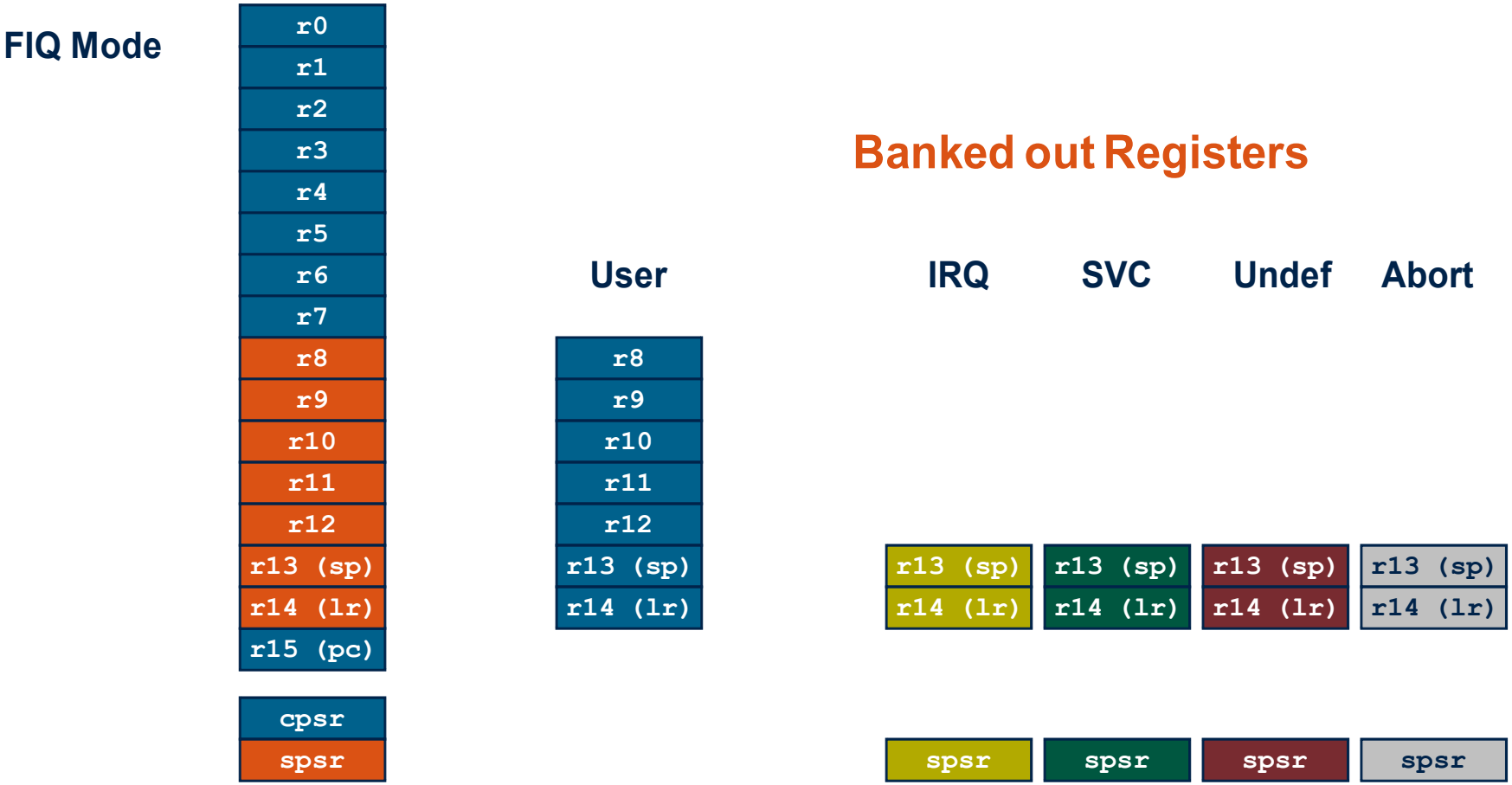
r0
r1
r2
r3
r4
r5
r6
r7
r8
r9
r10
r11
r12
r13 (sp)
r14 (lr)
r15 (pc)
cpsr

Banked out Registers

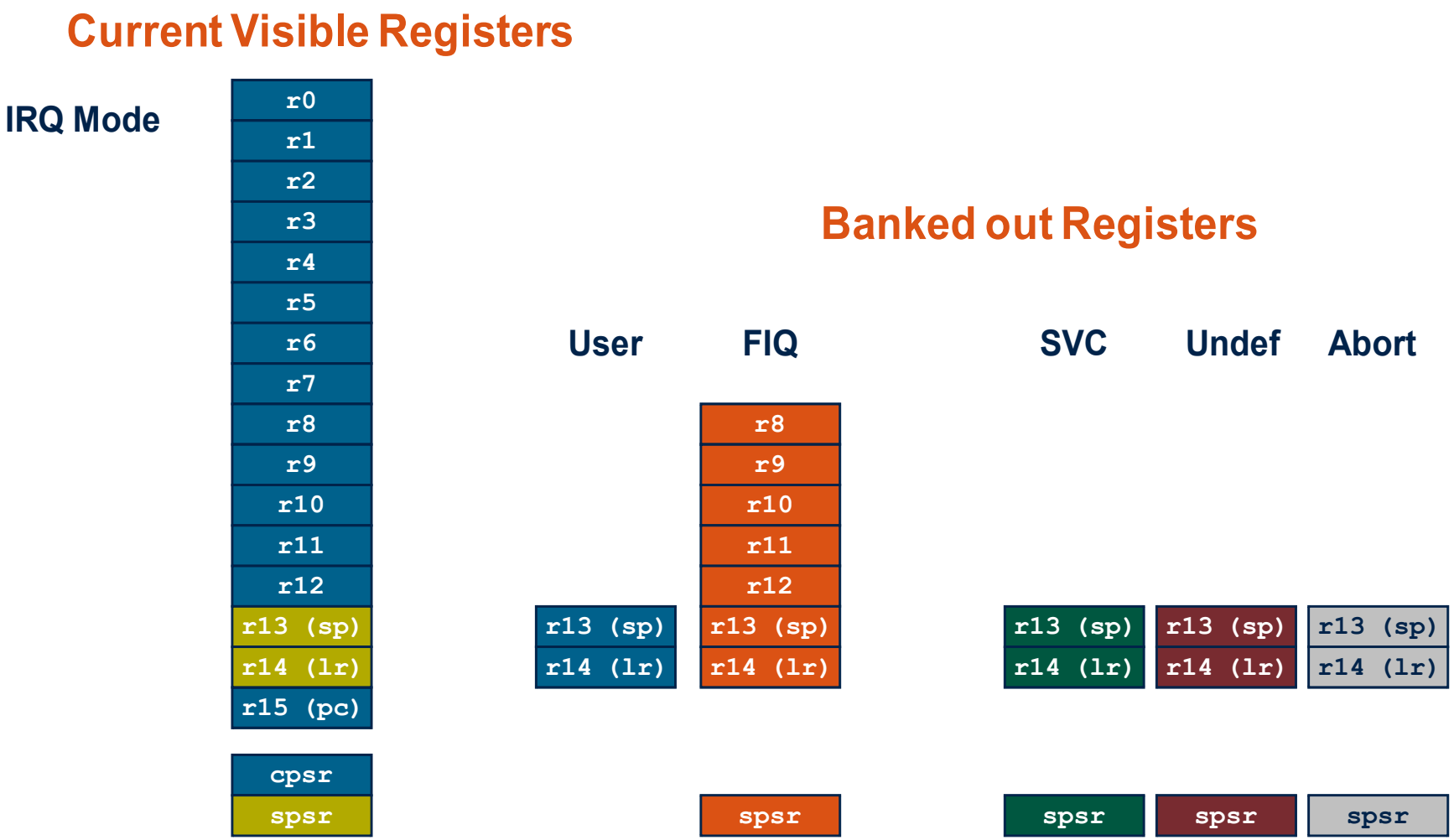
FIQ	IRQ	SVC	Undef	Abort
r8				
r9				
r10				
r11				
r12				
r13 (sp)	r13 (sp)	r13 (sp)	r13 (sp)	r13 (sp)
r14 (lr)	r14 (lr)	r14 (lr)	r14 (lr)	r14 (lr)
spsr	spsr	spsr	spsr	spsr

Model programowy – rejestry dostępne w trybie FIQ

Current Visible Registers



Model programowy – rejestry dostępne w trybie IRQ



Model programowy – rejestry dostępne w trybie Supervisor

Current Visible Registers

SVC Mode

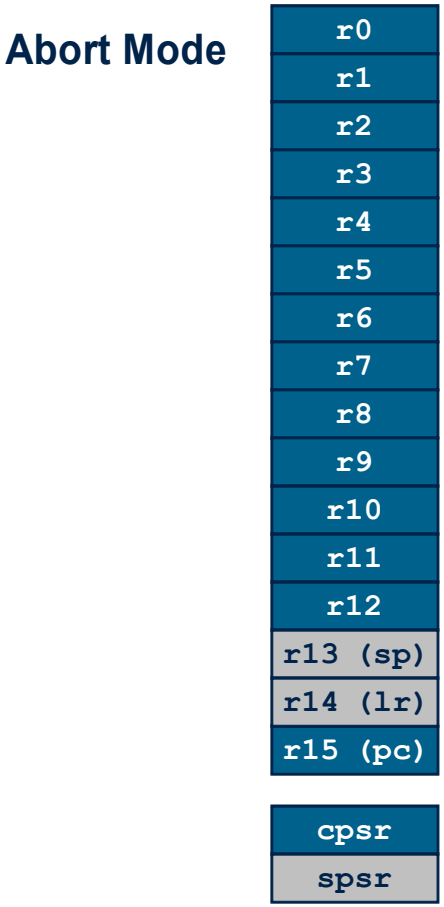
r0
r1
r2
r3
r4
r5
r6
r7
r8
r9
r10
r11
r12
r13 (sp)
r14 (lr)
r15 (pc)
cpsr
spsr

Banked out Registers

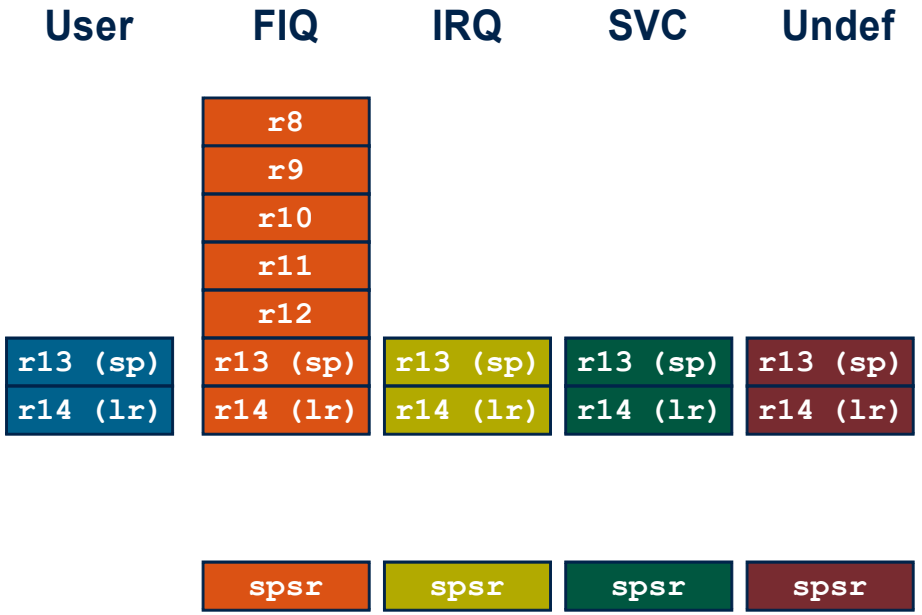
User	FIQ	IRQ	Undef	Abort
	r8			
	r9			
	r10			
	r11			
	r12			
r13 (sp)	r13 (sp)	r13 (sp)	r13 (sp)	r13 (sp)
r14 (lr)	r14 (lr)	r14 (lr)	r14 (lr)	r14 (lr)
	spsr	spsr	spsr	spsr

Model programowy – rejestry dostępne w trybie Abort

Current Visible Registers

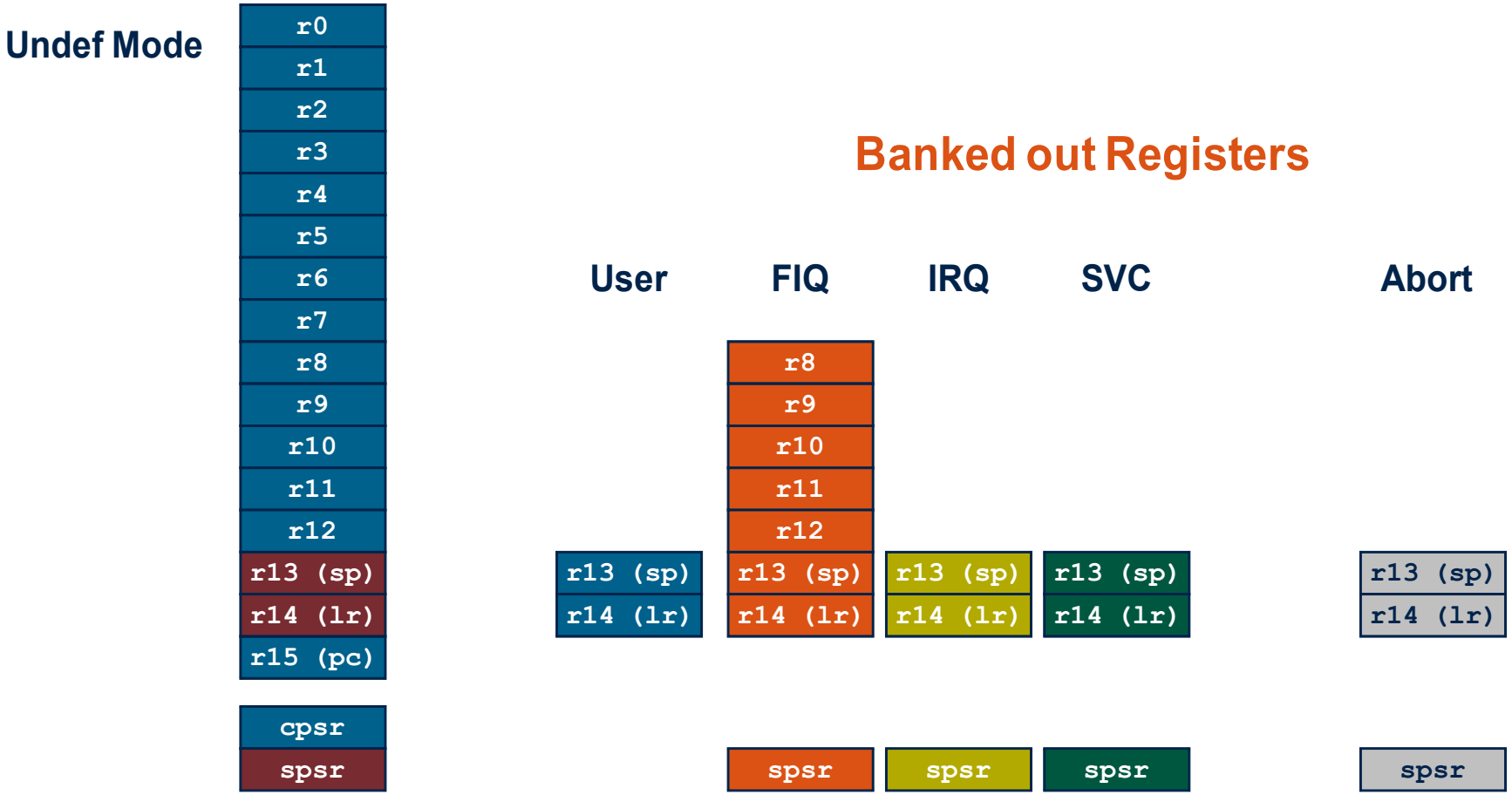


Banked out Registers

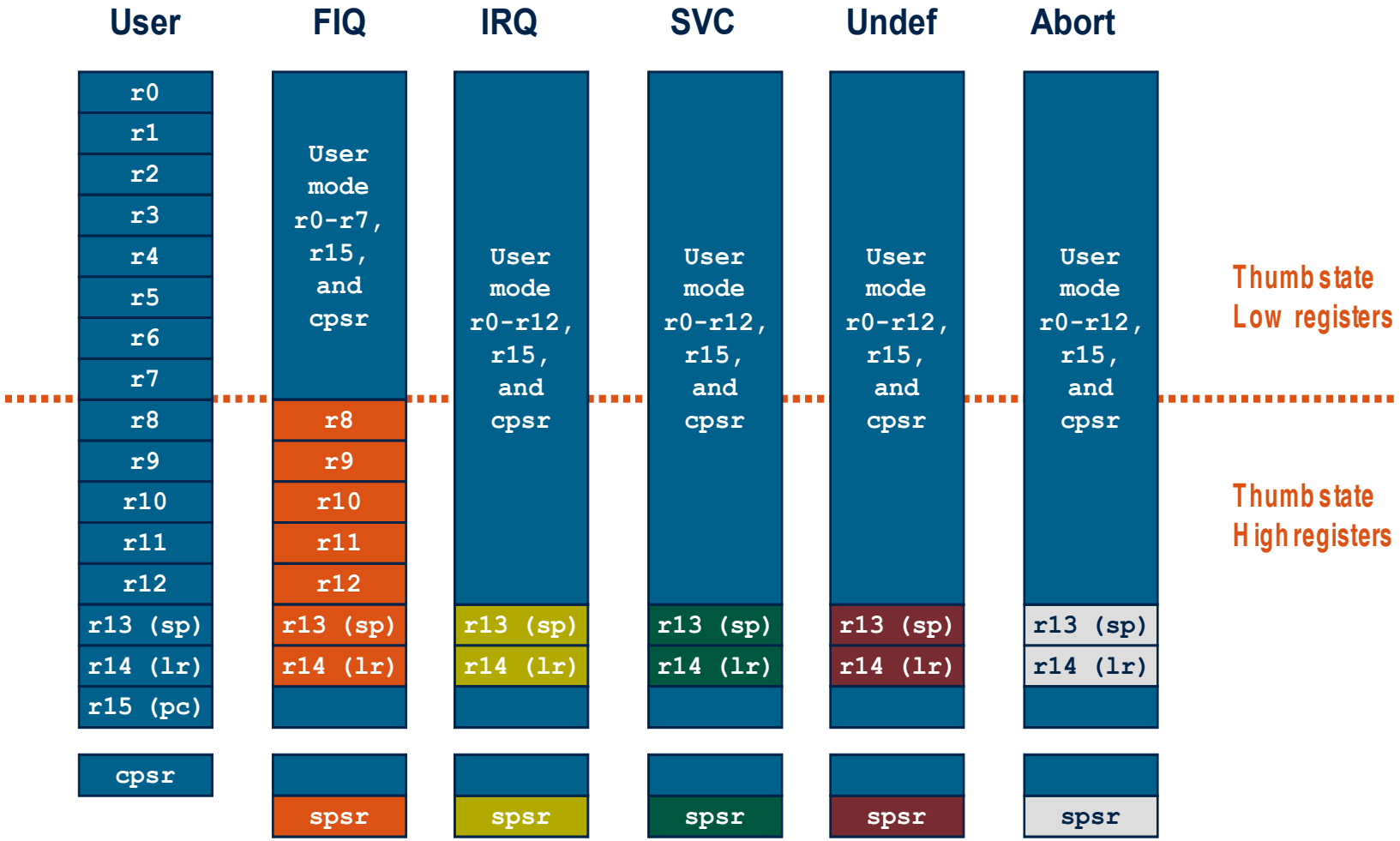


Model programowy – rejestry dostępne w trybie Undef

Current Visible Registers



Model programowy – podsumowanie dostępnych rejestrów



Note: System mode uses the User mode register set

Rejestry dostępne na różnych poziomach uprzywilejowania

Modes						
	Privileged modes					
		Exception modes				
User	System	Supervisor	Abort	Undefined	Interrupt	Fast interrupt
R0	R0	R0	R0	R0	R0	R0
R1	R1	R1	R1	R1	R1	R1
R2	R2	R2	R2	R2	R2	R2
R3	R3	R3	R3	R3	R3	R3
R4	R4	R4	R4	R4	R4	R4
R5	R5	R5	R5	R5	R5	R5
R6	R6	R6	R6	R6	R6	R6
R7	R7	R7	R7	R7	R7	R7
R8	R8	R8	R8	R8	R8	R8_fiq
R9	R9	R9	R9	R9	R9	R9_fiq
R10	R10	R10	R10	R10	R10	R10_fiq
R11	R11	R11	R11	R11	R11	R11_fiq
R12	R12	R12	R12	R12	R12	R12_fiq
R13	R13	R13_svc	R13_abt	R13_und	R13_irq	R13_fiq
R14	R14	R14_svc	R14_abt	R14_und	R14_irq	R14_fiq
PC	PC	PC	PC	PC	PC	PC
CPSR	CPSR	CPSR	CPSR	CPSR	CPSR	CPSR
		SPSR_svc	SPSR_abt	SPSR_und	SPSR_irq	SPSR_fiq

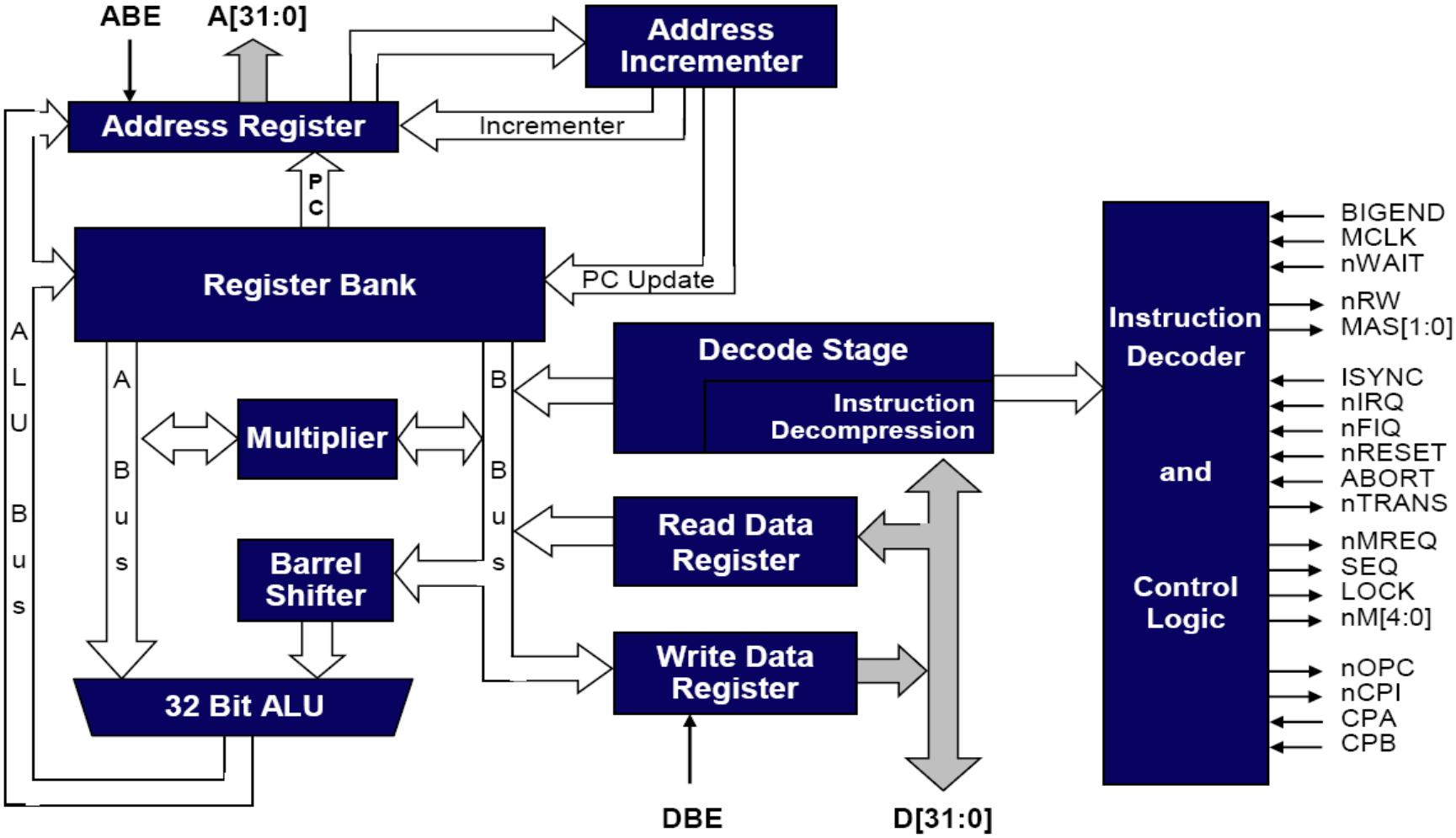
indicates that the normal register used by User or System mode has been replaced by an alternative register specific to the exception mode

Rdzenia procesora ARM7xxx

Cechy rdzenia ARM7TDM:

- Architektura rdzenia ARMv4,
- Trójstopniowy potok wykonawczy,
- Procesor stałoprzecinkowy,
- Wykorzystywane w telefonach komórkowych, odtwarzaczach mp3, itd...

Struktura rdzenia ARM7TDM



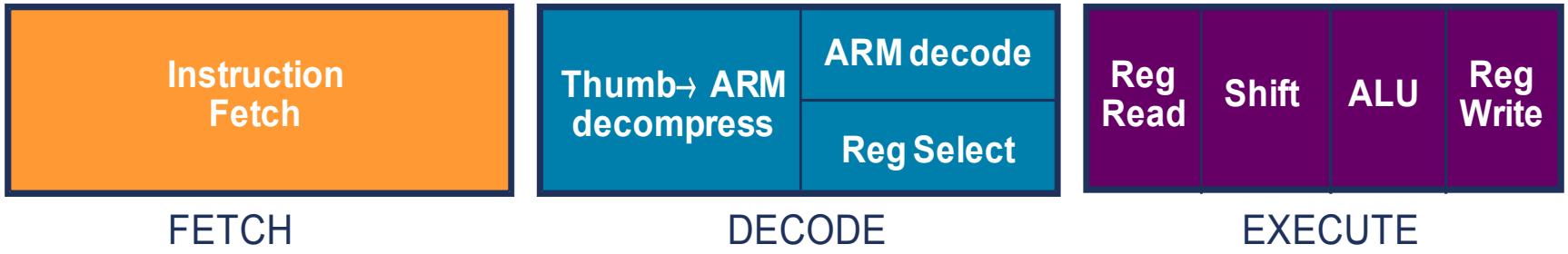
Rdzenia procesora ARM9xxx

Cechy rdzenia ARM9TDM:

- Architektura rdzenia ARMv5,
- Pięciostopniowy potok wykonawczy,
- Wbudowana jednostka zarządzania pamięcią MMU (wsparcie dla systemów operacyjnych WinCE, Linux, Symbian),
- Rozdzielona pamięć cache dla instrukcji i programu,
- Procesor stałoprzecinkowy,
- Wbudowana jednostka zarządzania pamięcią MMU (wsparcie dla systemów operacyjnych WinCE, Linux, Symbian),
- Zestaw instrukcji ARM, Thumb, (Java),
- Wykorzystywane w zaawansowanych telefonach komórkowych, urządzeniach telekomunikacyjnych, itd...

Potok wykonawczy instrukcji, ARM7 vs ARM9

ARM7TDMI



ARM9TDMI

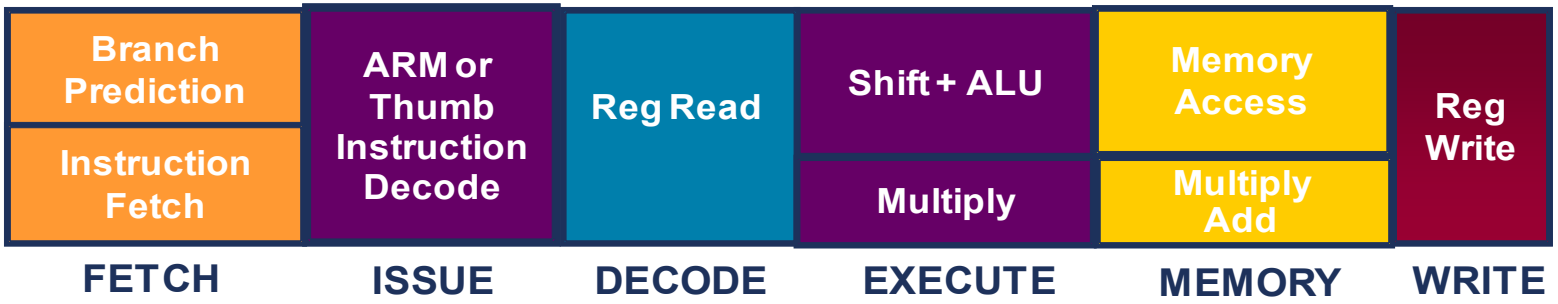
Rdzenia procesora ARM11xxx

Cechy rdzenia ARM11TDM:

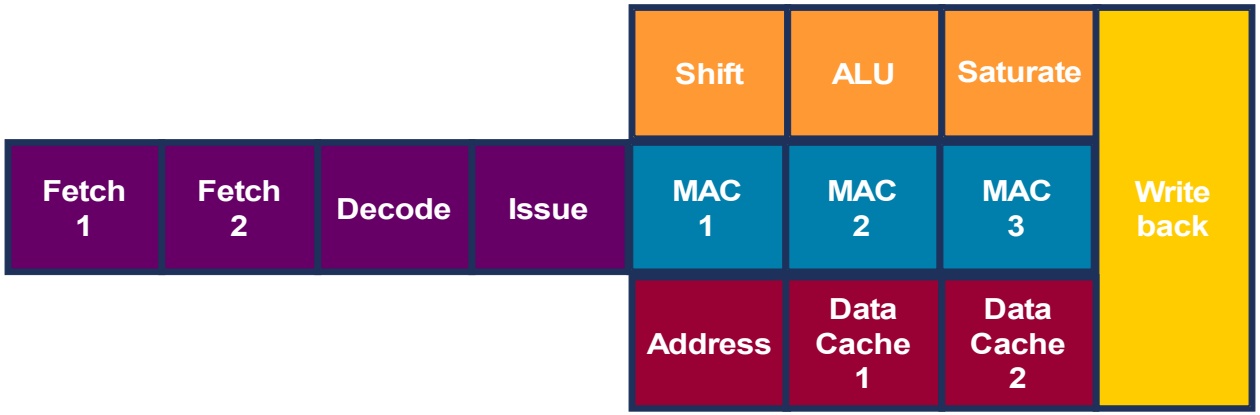
- Architektura rdzenia ARMv6,
- Siedmiostopniowy potok wykonawczy,
- Lepsza wydajność oraz obniżony pobór mocy,
- Rozdzielona pamięć cache dla instrukcji i programu,
- Dodatkowe instrukcje DSP oraz SIMD,
- Wykorzystywane w PDA, smartphonach, przenośnych grach komputerowych, itd...

Potok wykonawczy instrukcji, ARM10 vs ARM11

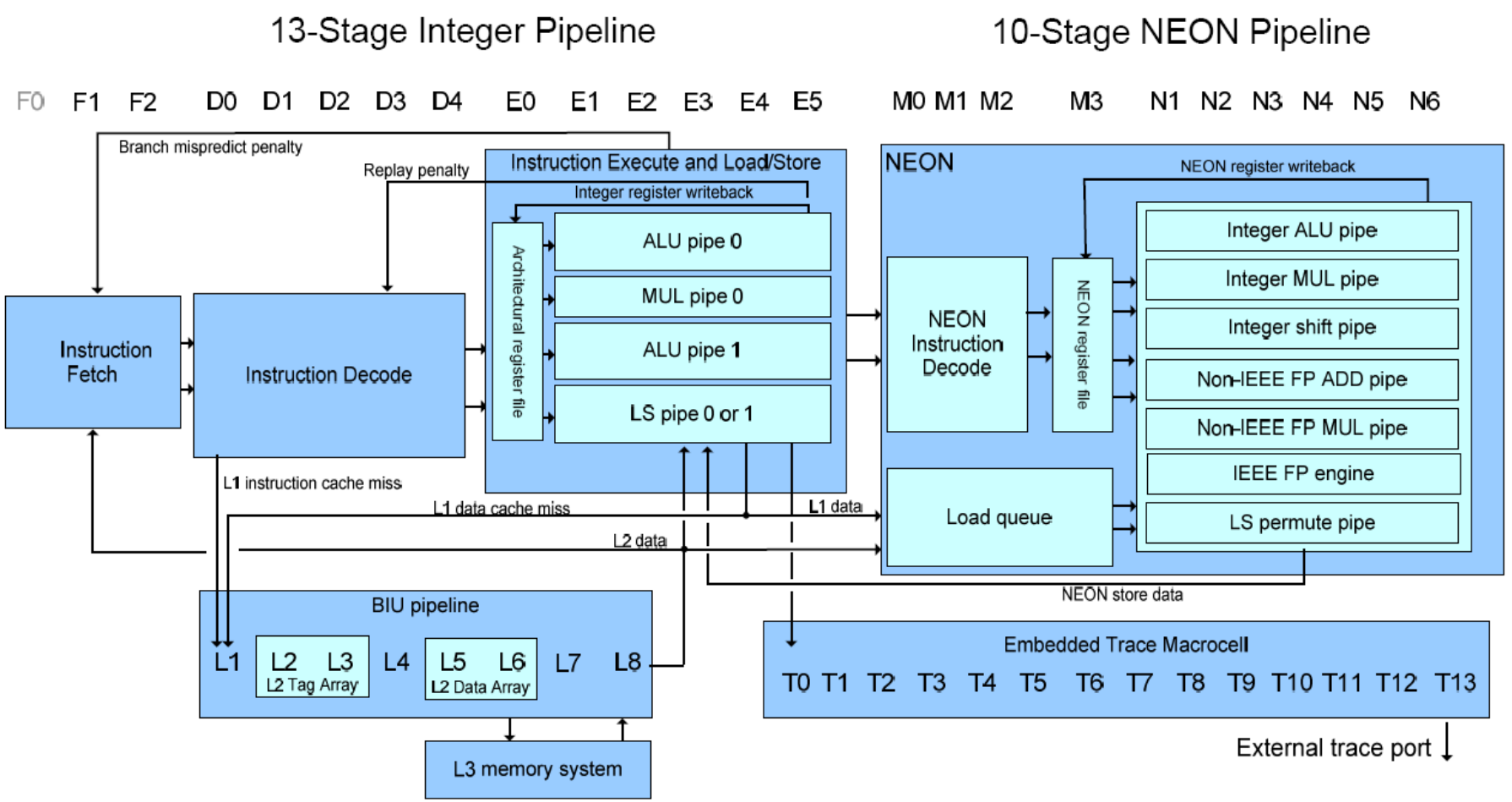
ARM10



ARM11



Potok wykonawczy procesora Cortex-A8



Układy do odmierzania czasu - timery procesora

Timery

Timer – urządzenie peryferyjne procesora przeznaczone do odmierzania określonych przedziałów czasu (zliczania elementarnych cykli zegarowych). Po odmierzeniu wymaganego okresu czasu timer zwykle generuje przerwanie. Timery wykorzystywane są do odmierzania czasu systemowego, przełączania wątków, generacji opóźnień.

Przykładów układów służących do odmierzania czasu:

Timer PIT (ang. Periodic Interval Timer, Programmable Interrupt Timer),

Timer Czasu Rzeczywistego RTT (ang. Real-Time Timer),

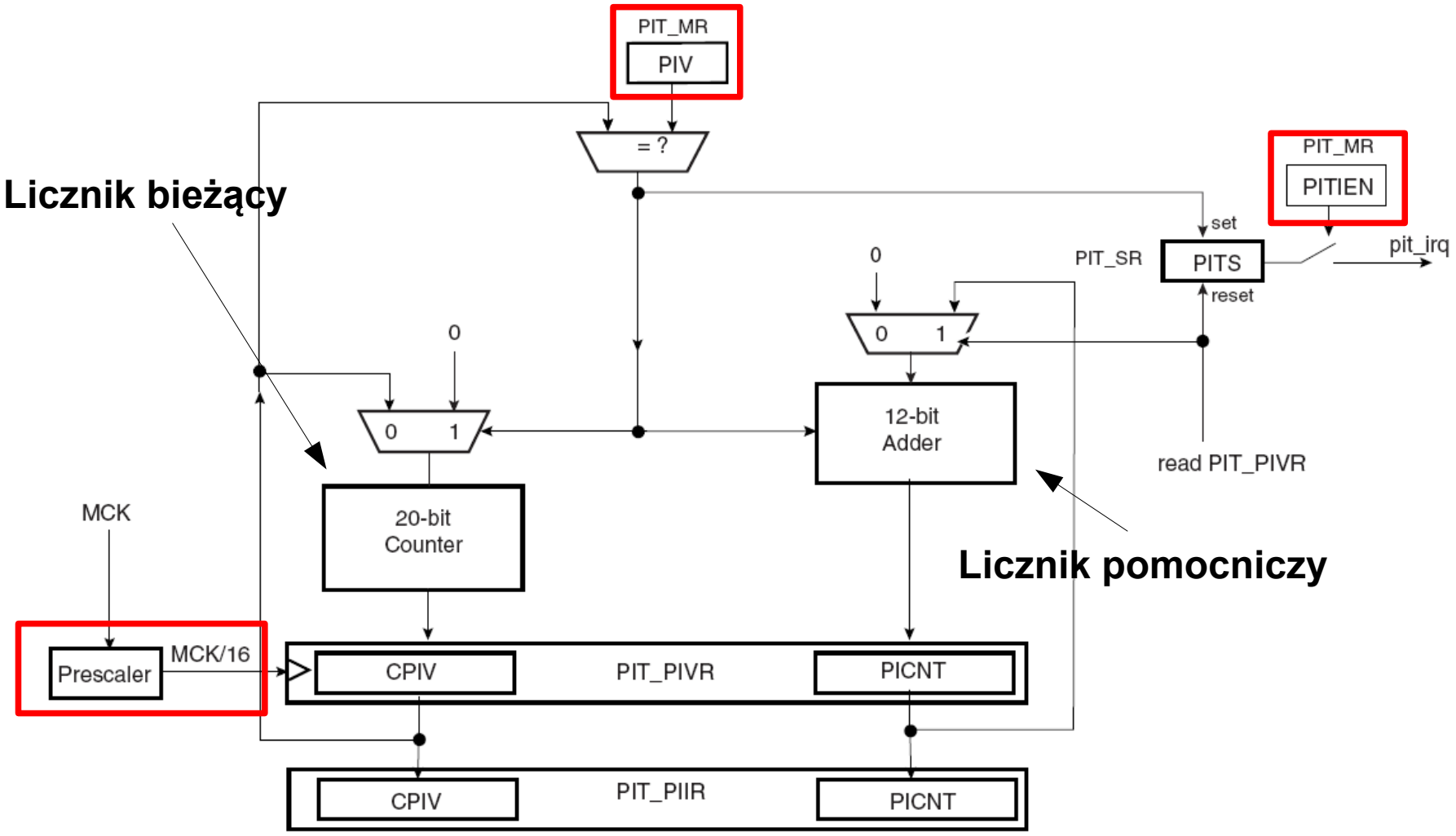
Timer PWM (ang. Pulse Width Modulation),

Timer uniwersalny TC (ang. Timer Counter),

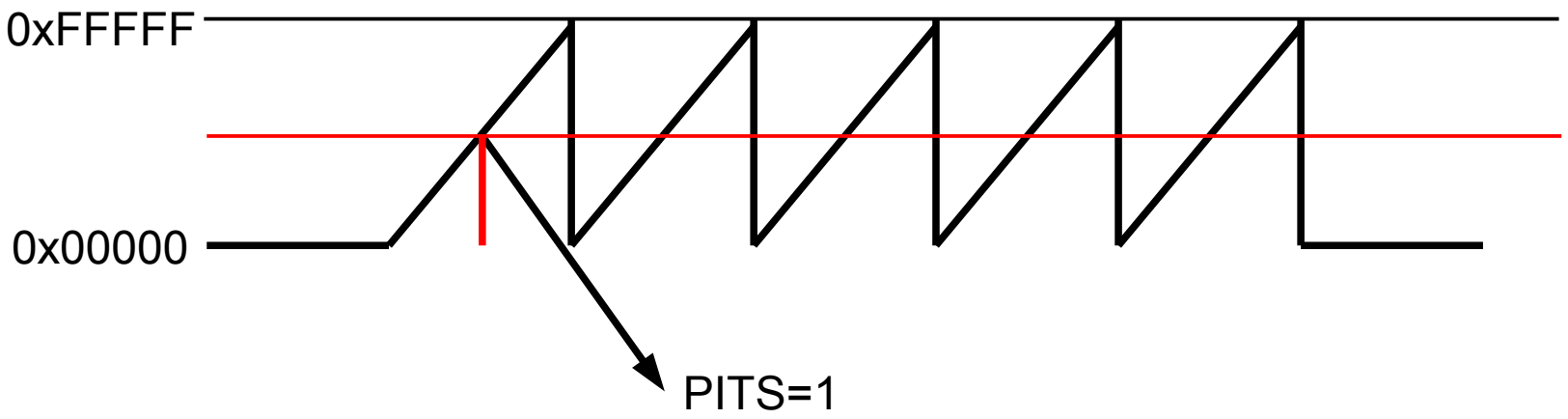
Timer Watch-dog WDT.

Moduł timera PIT (Periodic Interval Timer)

Schemat blokowy timera PIT



Automatyczne przeładowanie timera

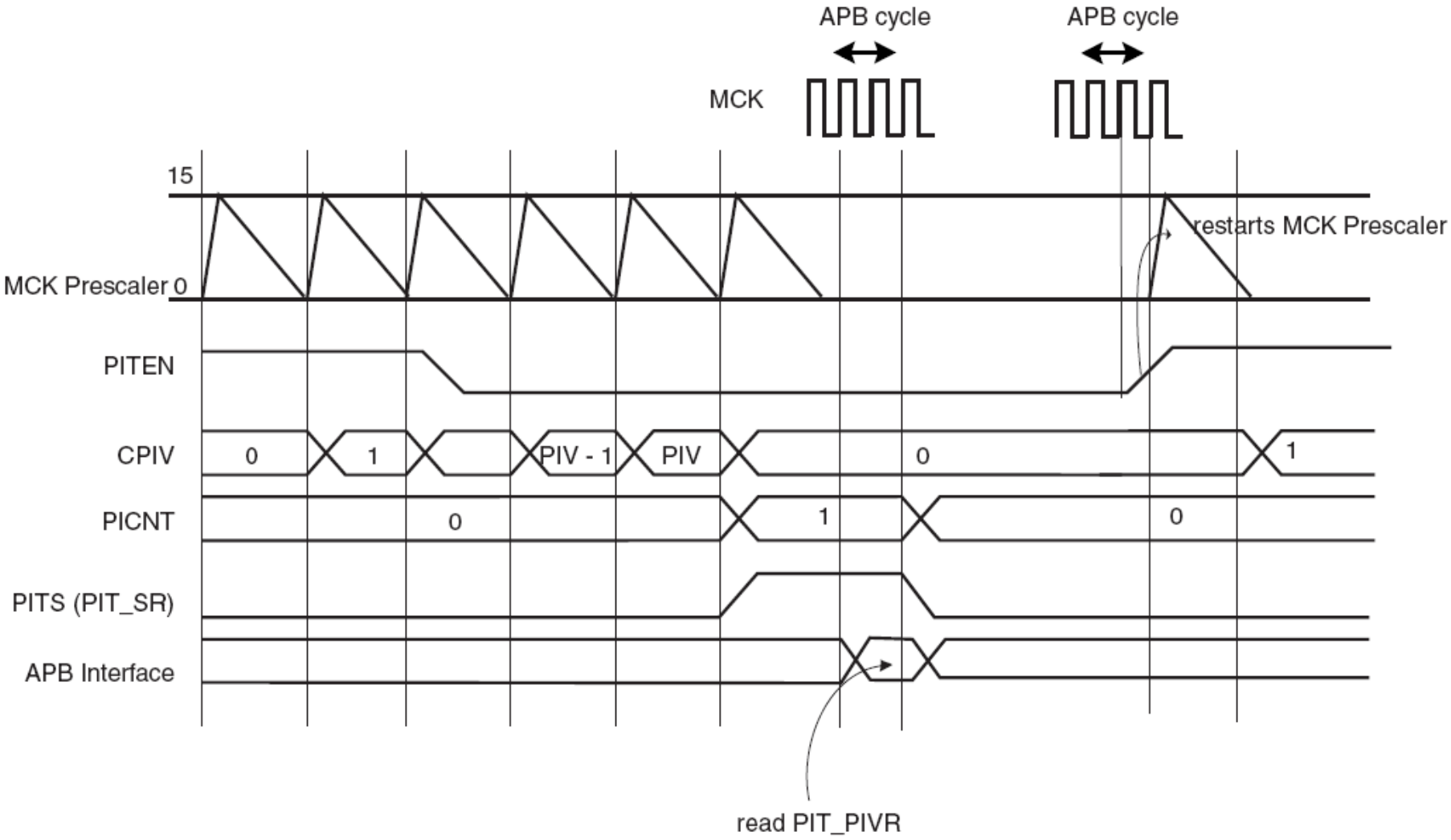


Okres generowanych przerwań:

(PIV_VALUE+1)/(16 * Clk)

Clk = 100 MHz, PIV = 0x1000 => t_{PIT} = 2,5 us

Przebiegi obrazujące pracę timera PIT



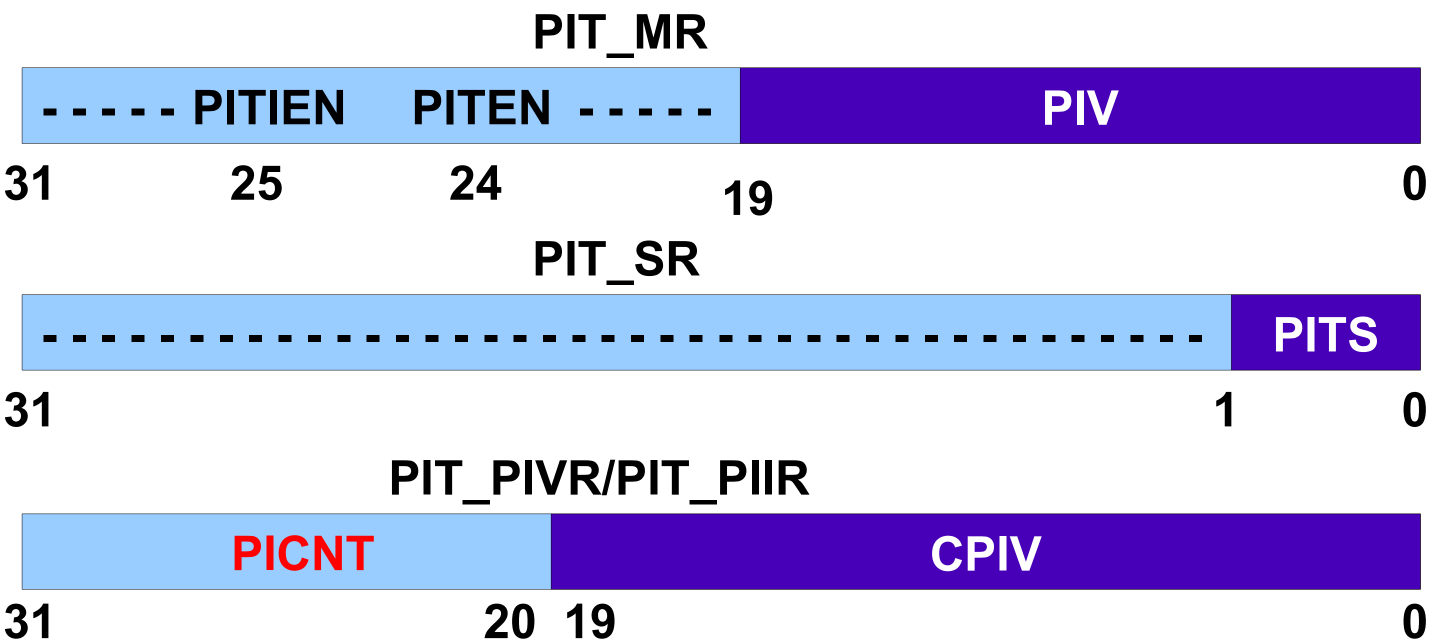
Rejestry timera PIT

Table 16-1. Register Mapping

Offset	Register	Name	Access	Reset
0x00	Mode Register **Address:** 0xFFFFFD30	PIT_MR	Read-write	0x000F_FFFF
0x04	Status Register	PIT_SR	Read-only	0x0000_0000
0x08	Periodic Interval Value Register	PIT_PIVR	Read-only	0x0000_0000
0x0C	Periodic Interval Image Register	PIT_PIIR	Read-only	0x0000_0000

```
typedef struct S_PIT {          /*   Register name          R/W    Reset val.      Offset
AT91_REG  PIT_MR;               // PIT Mode Register        R/W   0x000F.FFFF      0x00
AT91_REG  PIT_SR;               // PIT Status Register      R     0x0000.0000      0x04
AT91_REG  PIT_PIVR;             // PIT Per. Int. Val. Reg.  R     0x0000.0000      0x08
AT91_REG  PIT_PIIR;             // PIT Per. Int. Image Reg. R     0x0000.0000      0x0C
}  S_PIT, *PS_PIT;
/* blok rejestrów portów I/O PIOA...PIOE */
#define PIT     ((PS_PIT)    0xFFFFFD30)     // (PIT) Base Address
```

Rejestry timera PIT

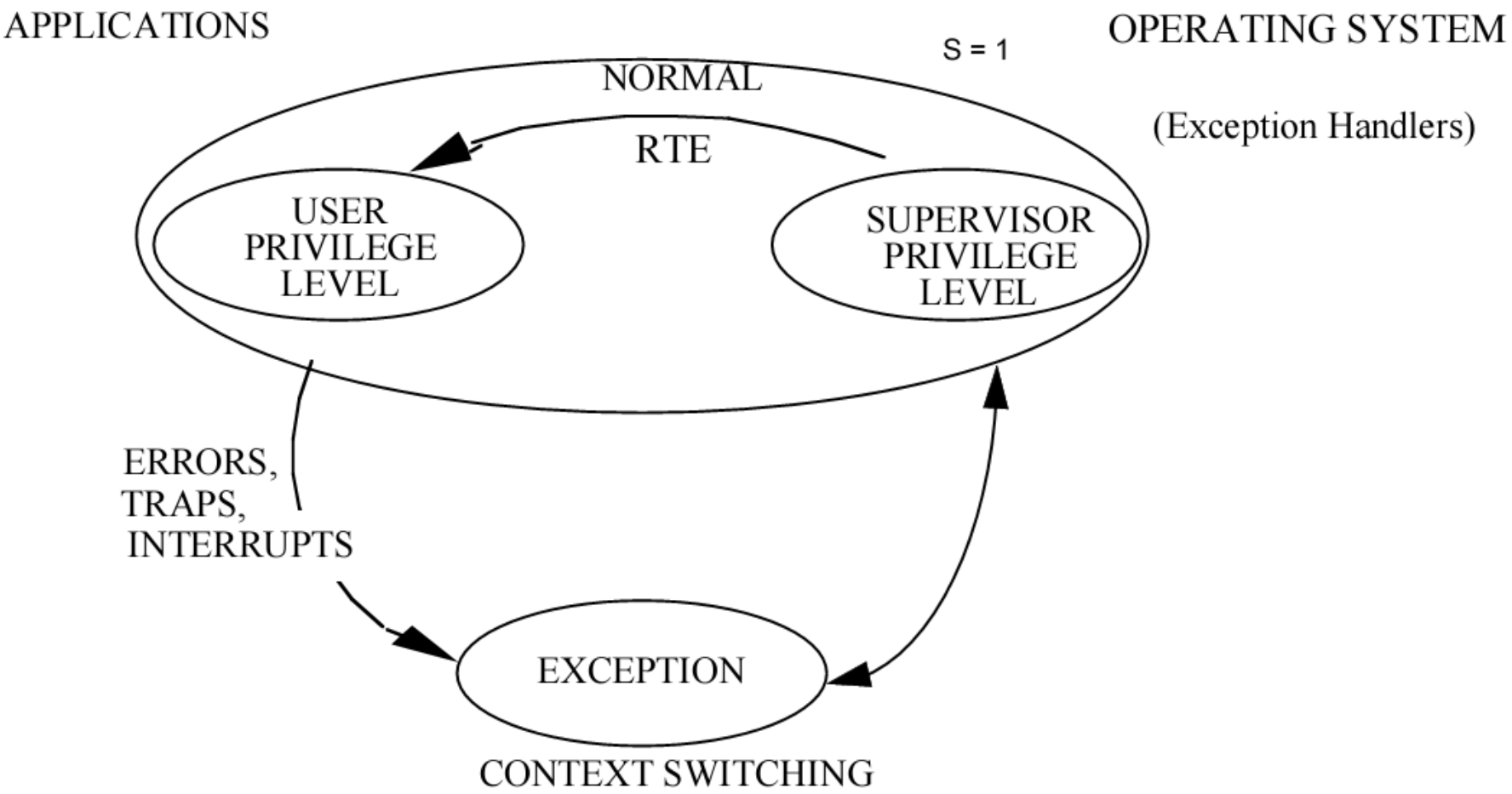


Obsługa przerwania

- Timer PIT może generować przerwania (np. wykorzystywane przez planistę systemów operacyjnych).
- Przerwanie generowane jest w momencie przepełnienia 20 bitowego licznika CPIV. Ustawiana jest flaga PITS oraz generowane jest przerwanie (przy założeniu, że nie jest maskowane),
- Procesor przerywa wykonywanie programu i wykonuje skok do funkcji obsługującej przerwania systemowe (przerwanie SYS o numerze ID równym 1),
- Procedura obsługi przerwania powinna obsłużyć wszystkie przerwania systemowe (przerwanie SYS może zostać zgłoszone przez kilka urządzeń peryferyjnych procesora, np. port szeregowy, licznik Watch-Dog, itd...),
- Procedura obsługująca przerwania powinna odczytać rejestr PIVR w celu skasowania flagi PIT oraz potwierdzenia zakończenia procedury obsługi przerwania systemowego.
- Potwierdzenie obsługi przerwania dla kontrolera przerwań AIC (rejestr EOICR) wykonywane jest automatycznie w niskopoziomowej procedurze obsługi przerwania (asembler).

Kontroler przerwań AIC
(Advanced Interrupt Controller)

Obsługa sytuacji wyjątkowych



Wyjątki

Wyjątek (ang. exception) – mechanizm kontroli przepływu danych występujący w mikroprocesorach oraz we współczesnych językach programowania służący do obsługi zdarzeń wyjątkowych, a w szczególności sytuacji błędnych. Szczególnym przypadkiem wyjątku jest przerwanie.

Wyjątki dzielimy na:

- przerwania (ang. interrupts),
- niepowodzenia (ang. fault),
- błędy nienaprawialne (ang. abort),
- pułapki (ang. trap).

Procesory ARM obsługują dwa rodzaje przerwań:

- **FIQ** (Fast interrupt) – przerwania z szybką obsługą,
- **IRQ** (Interrupt) – przerwania normalne.

Obsługa wyjątków

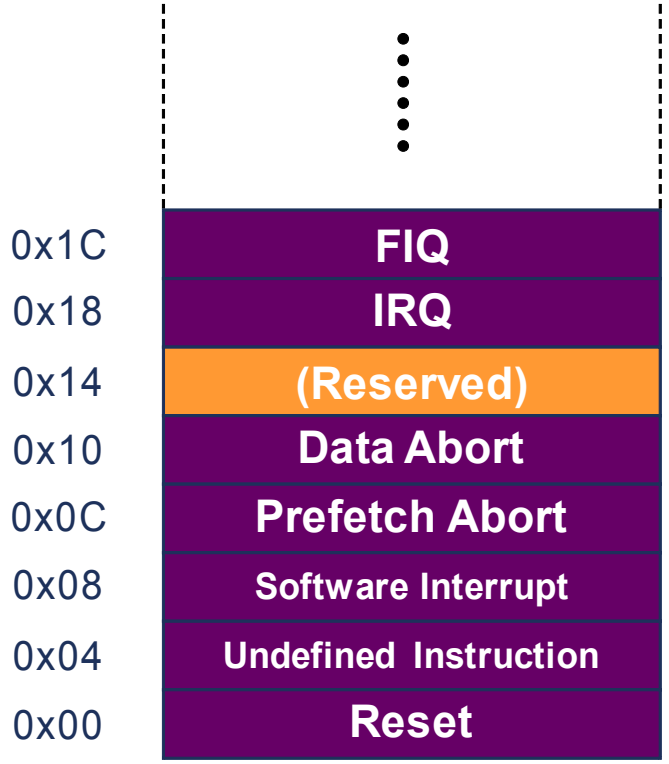
Wystąpienie wyjątku w przypadku procesorów ARM powoduje:

- Przełączenie banku rejestrów, CPSR->SPSR_<mode>
- Ustawienie odpowiednich bitów rejestru CPSR
  - Zmiana trybu pracy do trybu ARM
  - Przełączenie w tryb obsługi wyjątków
  - Wyłączenie przerwań
- Zapisanie obecne rejestru do rejestru LR_<mode>
- Ustawienie rejestru PC na adres odpowiedniego wektora obsługującego dany wyjątek

W przypadku powrotu z wyjątku uchwyt obsługujący dany wyjątek powinien:

- Odtworzyć zawartość rejestrów zapisanych na stosie (r0-r13, pozostałe rejestry odtwarzane automatycznie-w zależności od trybu pracy, np. CPSR z SPSR_<mode>)
- Odtworzyć rejestr PC z LR_<mode>

Powyższe operacje można wykonać wyłącznie w trybie ARM



Vector Table

Vector table can be at **0xFFFF0000** on ARM720T and on ARM9/10 family devices

Przerwania

Przerwanie (ang. interrupt) lub żądanie przerwania (IRQ – Interrupt ReQuest) – sygnał powodujący zmianę przepływu sterowania, niezależnie od aktualnie wykonywanego programu. Pojawienie się przerwania powoduje wstrzymanie aktualnie wykonywanego programu i wykonanie przez procesor kodu procedury obsługi przerwania, uchwytu przerwania (ang. interrupt handler).

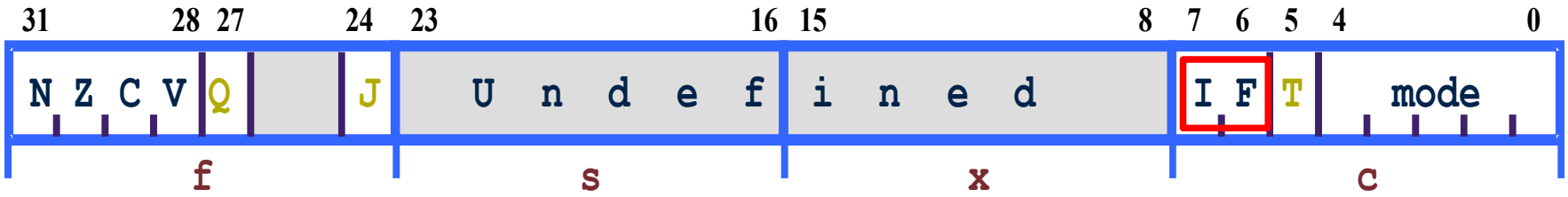
W procesorach ARM wystąpienie przerwania powoduje ustawienie flagi w rejestrze statusowym AIC sygnalizującej żądanie obsługi przerwania. Jeżeli system przerwań jest aktywny (rdzeń procesora) oraz dane przerwanie nie jest zamaskowane (sterownik przerwań) następuje przyjęcie przerwania przez procesor - skok do programu obsługującego przerwanie.

Przykłady przerwań:

- odebranie lub zakończenie transmisji danej przez port szeregowy,
- zmiana stanu wyprowadzenia portu procesora.

Stan urządzenia można sprawdzać programowo, jednak wymaga to ciągłego sprawdzania stanu rejestru statusowego. Taka operacja nazywana jest odpytywaniem (ang. poolling). Powoduje to znaczne obciążenie procesora, np. transmisja jednego znaku zajmuje ok. 100 us (procesor może w tym czasie wykonać kilka tysiące operacji).

Rejestr statusowy SPSR procesora ARM



Wskaźniki stanu

- V – przepełnie podczas operacji ALU (oVerflow)
- C – przeniesienie/pożyczka podczas operacji ALU
- Z – ujemny wynik podczas operacji ALU
- N – ujemny wynik operacji ALU lub „mniejszy niż”

Wskaźniki dostępne jedynie dla architektury 5TE/J

- J – Procesor w trybie Jazelle
- Q – Sticky Overflow – wskaźnik nasycenia podczas operacji ALU (QADD, QDADD, QSUB or QDSUB, lub rezultat operacji SMLAxy or SMLAWx przekracza 32-bity)

Przerwania

- I=1 Przerwania IRQ wyłączone
- F=1 Przerwania FIQ wyłączone

Wskaźniki dostępne dla arch. xT

- T=0 Tryb pracy ARM
- T=1 Tryb pracy Thumb

Tryb pracy procesora

- Definiują jeden z 7 trybów operacyjnych rdzenia procesora

Obsługa przerwania

Obsługa przerwania obejmuje wszystkie operacje od momentu wykrycia sygnału przerwania do pobrania pierwszej instrukcji obsługującej dane przerwanie.

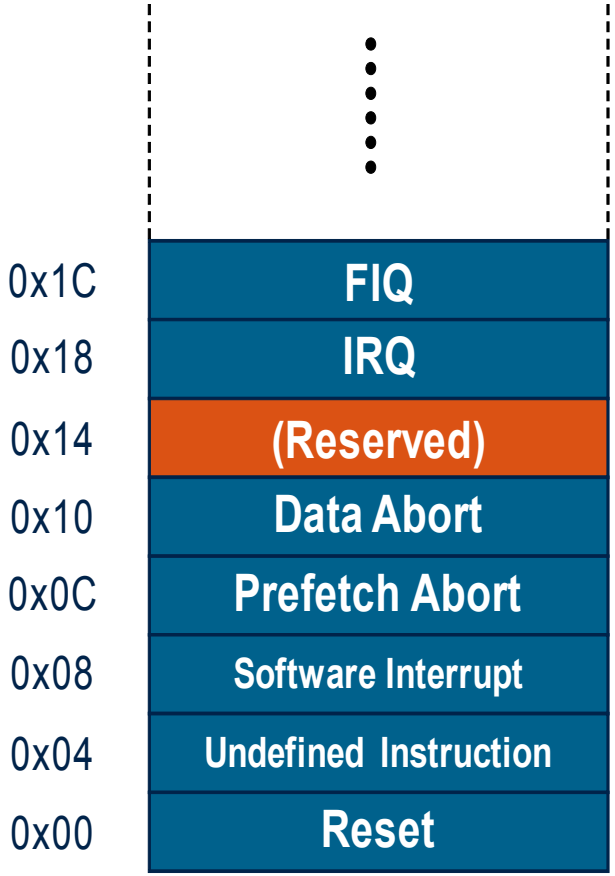
1. a) Wykonanie kopii CPSR → SPSR oraz PC (r15) → Link Register (r14),
   b) Przejście do trybu ARM (z trybu Thumb lub Jazelle),
   c) Przejście do trybu obsługi przerwań (FIQ/IRQ) lub wyjątków,
   d) Ustawienie maski IRQ na poziomie zgłaszanego przerwania (lub wyłączenie przerwań).
   e) Przełączenie banku rejestrów (uaktywnienie rejestru SPSR).
2. Określenie wektora obsługiwanego wyjątku (przerwania).
3. Obliczenie adresu pierwszej instrukcji procedury obsługującej dany wyjątek (przerwanie).

Powrót z obsługi sytuacji wyjątkowych

1. a) Odtworzenie rejestru CPSR (r15),
   b) PC (Link Register r14),
   c) Powrót do wykonywanego rozkazu.

Tablica wektorów przerwań umieszczona po resecie pod adresem 0x0.

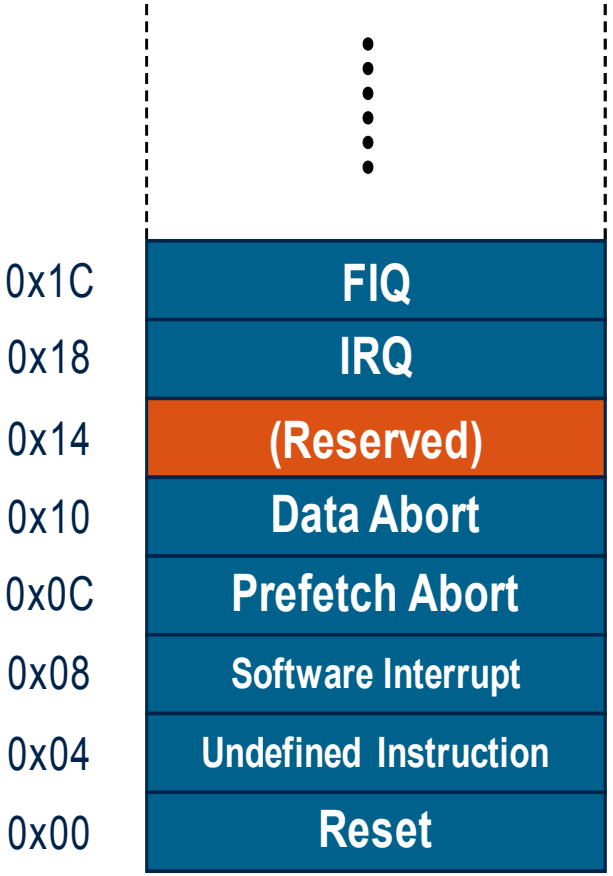
Tablice można przesunąć pod adres 0xFFFF.0000 (ARM 7/9/10).



Fragment pamięci

Tablica wyjątków

```
LDR     PC, =FIQ_Addr
LDR     PC, =IRQ_Addr
NOP ; Reserved vector
LDR     PC, =Abort_Addr
LDR     PC, =Prefetch_Addr
LDR     PC, =SWI_Addr
LDR     PC, =Undefined_Addr
LDR     PC, =Reset_Addr
```



Memory image

Obsługa wyjątków (1)

```
IRQ_Addr:
/*- Manage Exception Entry */
/*- Adjust and save LR_irq in IRQ stack */
        sub      lr, lr, #4
        stmfd    sp!, {lr}
/*- Save r0 and SPSR in IRQ stack */
        mrs      r14, SPSR
        stmfd    sp!, {r0,r14}
/*- Write in the IVR to support Protect Mode */
/*- No effect in Normal Mode */
/*- De-assert the NIRQ and clear the source in Protect Mode */
        ldr      r14, =AT91C_BASE_AIC
        ldr      r0 , [r14, #AIC_IVR]
        str      r14, [r14, #AIC_IVR]
...
/*- Branch to the routine pointed by the AIC_IVR */
        mov      r14, pc
        bx       r0                          /* Branch to IRQ handler */
...
/*- Restore adjusted  LR_irq from IRQ stack directly in the PC */
        ldmia    sp!, {pc}^
```

Obsługa wyjątków (2)

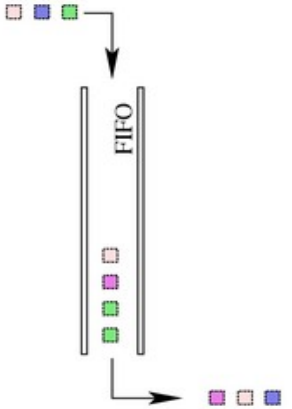
```
/* lowlevel.c */
/*-------------------------------------------------------------------------
 * Function Name       : default_spurious_handler
 * Object              : default handler for spurious interrupt
 *-------------------------------------------------------------------------*/
void default_spurious_handler(void)
{
    dbgu_print_ascii("-F- Spurious Interrupt\n\r ");
    while (1);
}

/*-------------------------------------------------------------------------
 * Function Name       : default_fiq_handler
 * Object              : default handler for fast interrupt
 *-------------------------------------------------------------------------*/
void default_fiq_handler(void)
{
    dbgu_print_ascii("-F- Unexpected FIQ Interrupt\n\r ");
    while (1);
}
```

Struktura stosu (1)

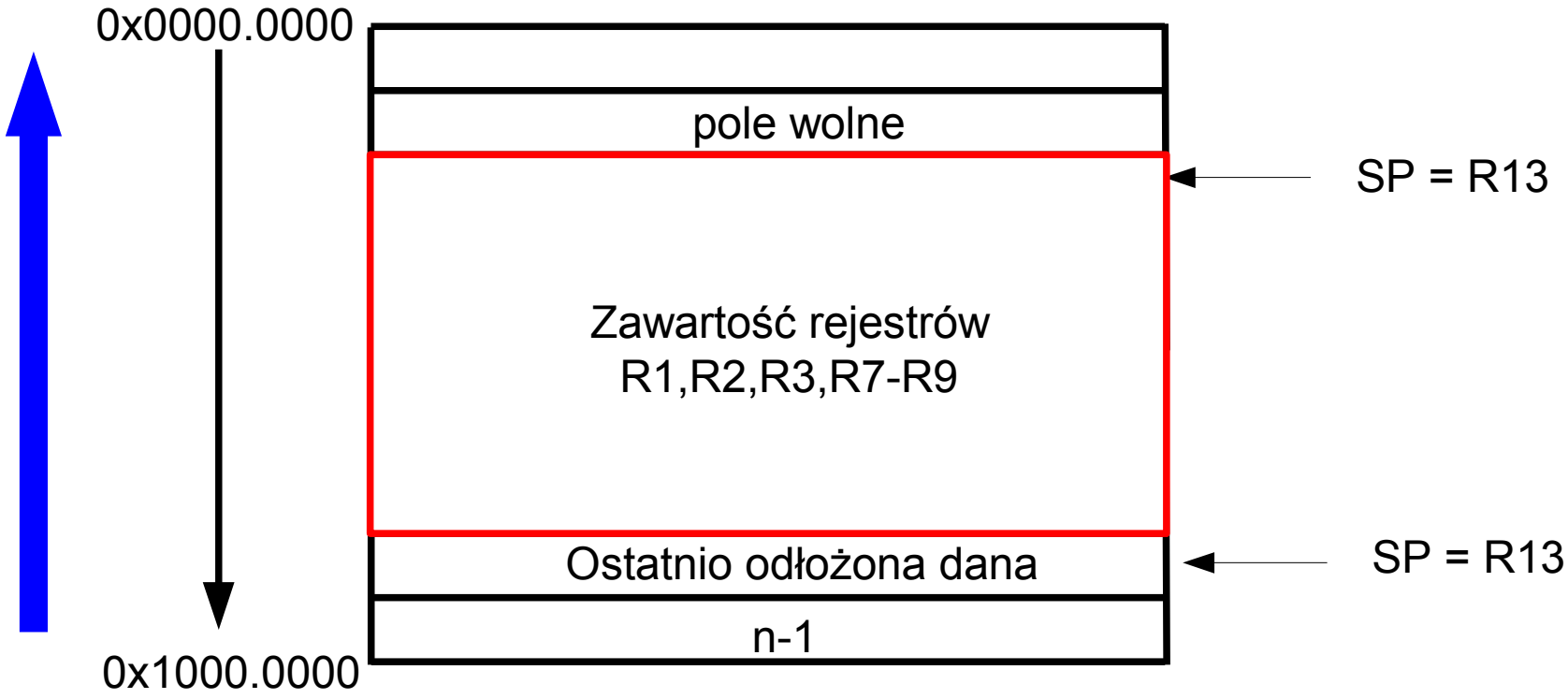
Stos (ang. stack lub LIFO Last-In, First-Out) - liniowa struktura danych, w której dane odkładane są na wierzch stosu i z wierzchołka stosu są zdejmowane. Ideę stosu danych można zilustrować jako stos położonych jedna na drugiej książek – nowy egzemplarz kładzie się na wierzch stosu i z wierzchu stosu zdejmuje się kolejne egzemplarze. Elementy stosu poniżej wierzchołka stosu można wyłącznie obejrzeć, aby je ściągnąć, trzeba najpierw po kolei ściągnąć to, co jest nad nimi

FIFO (ang. First In, First Out) - przeciwieństwem stosu LIFO jest kolejka, bufor typu FIFO (pierwszy na wejściu, pierwszy na wyjściu), w którym dane obsługiwane są w takiej kolejności, w jakiej zostały dostarczone (jak w kolejce do kasy)

First–in First–out (FIFO)

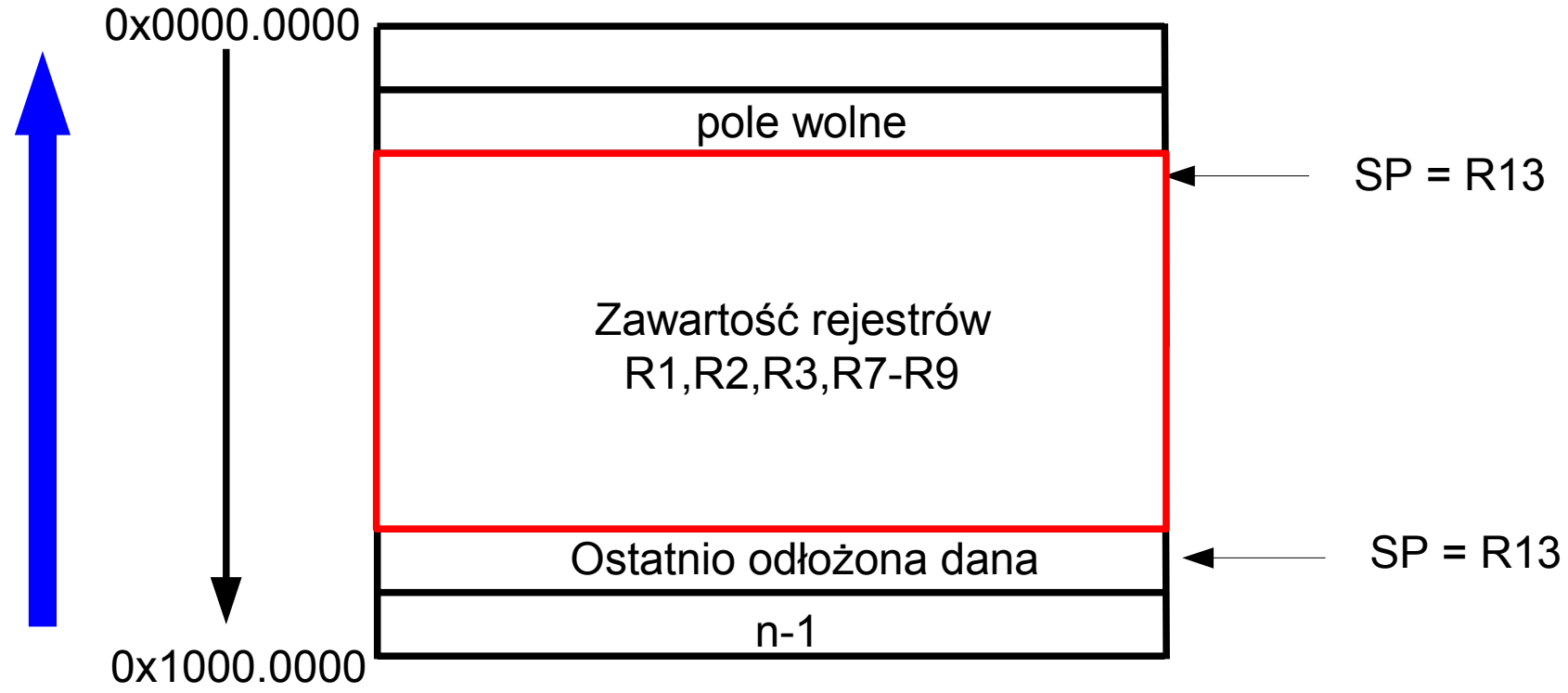


Struktura stosu – odłożenie danej na stos




```
STMDB   SP!, {lista rejestrów}      | zmniejszenie SP o 24, odłożenie zawartości rejestrów  na
STMDB   SP!, {R1,R2,R3,R7-R9}         stos,
```

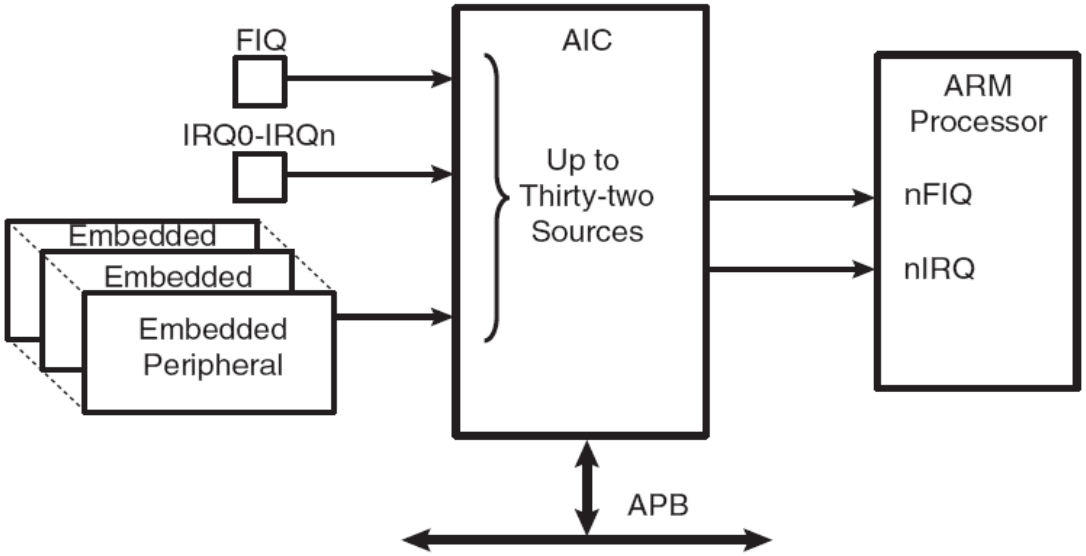
Struktura stosu – zdjęcie danej ze stosu




```
LDMIA   SP!, {lista rejestrów}      | zwiększenie SP o 24, odłożenie zawartości rejestrów  na
LDMIA   SP!, {R1,R2,R3,R7-R9}         stos,
```

Advanced Interrupt Controller

Schemat blokowy sterownika przerwań procesora ARM



- Obsługa przerwań wektorowych,
- Obsługa do 32 przerwań zewnętrznych i wewnętrznych,
- Możliwość maskowania dowolnego przerwania,
- Obsługa przerwań nIRQ i szybkich nFIR (ang. Fast Interrupt Request),
- 8 poziomów priorytetów (0- najniższy, 7- najwyższy),
- Obsługa przerwań wyzwalanych poziomem lub zboczem.

Schemat blokowy sterownika przerwań procesora ARM

- Sterownik przerwań wykorzystuje zegar systemowy. Zegar doprowadzany jest przez cały czas pracy procesora (nie ma możliwości odcięcia zegara).
- Przerwania mogą zostać wykorzystane do wyprowadzenia procesora ze stanu uśpienia (Idle mode).
- Przerwanie o numerze 0 (FIQ) jest zawsze przerwaniem typu FIQ.
- Przerwanie o numerze 1 (SYS) sumą logiczną przerwań od urządzeń peryferyjnych procesora. W procedurze obsługi przerwania należy określić urządzenie/a zgłaszające przerwanie/a.
- Przerwania o numerach 2-31 (PID2-PID31) mogą zostać dołączone do urządzeń peryferyjnych (użytkownika) lub portów I/O.
- Sterownik obsługuje przerwania wyzwalane poziomem lub zboczem sygnału.

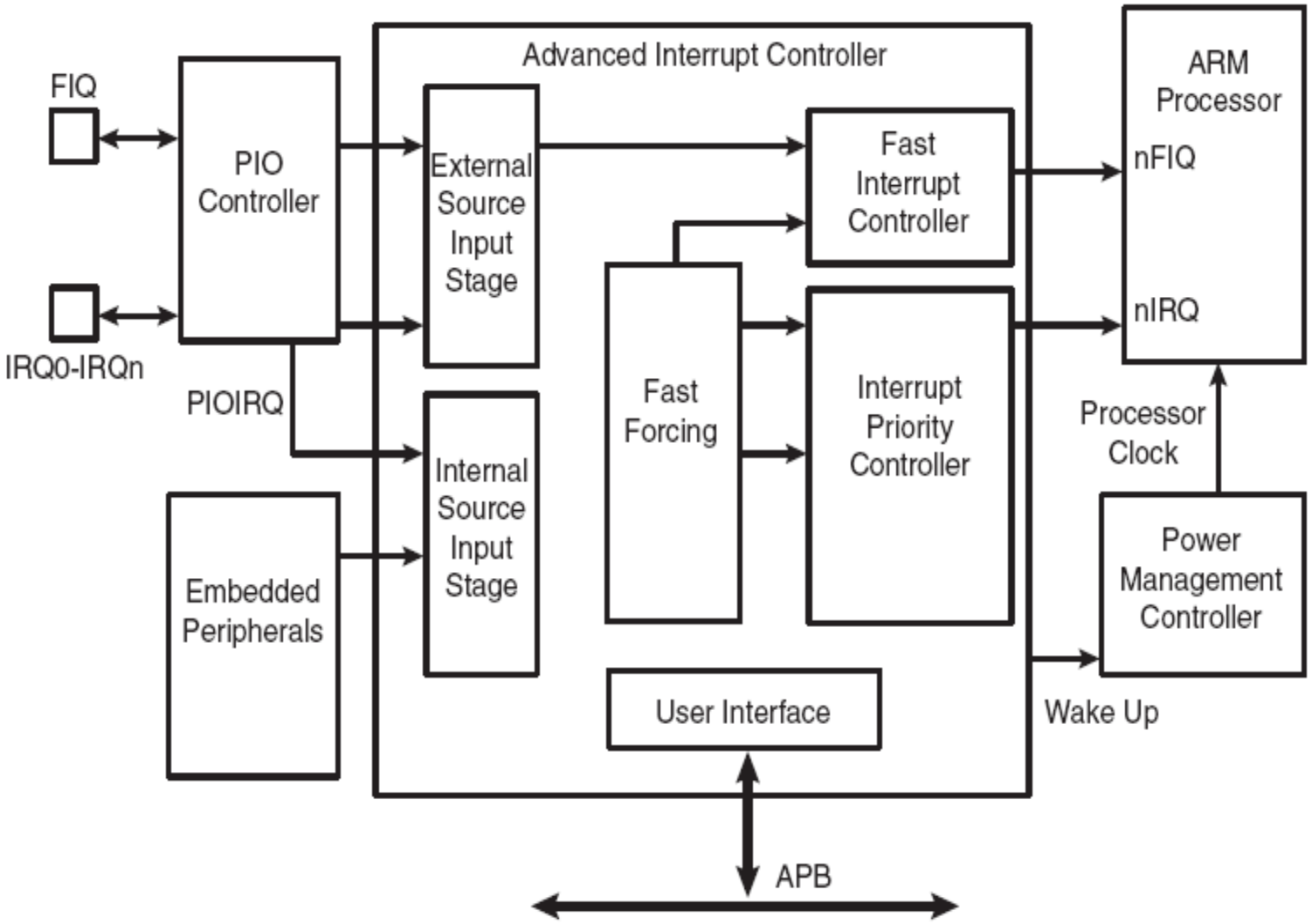
Przerwanie współdzielone

Blok urządzeń systemowy (AT91C_ID_SYS) dysponuje jednym, wspólnym przerwaniem systemowym SYS (ang. shared interrupt) o numerze ID=1, które obejmuje następujące urządzenia:

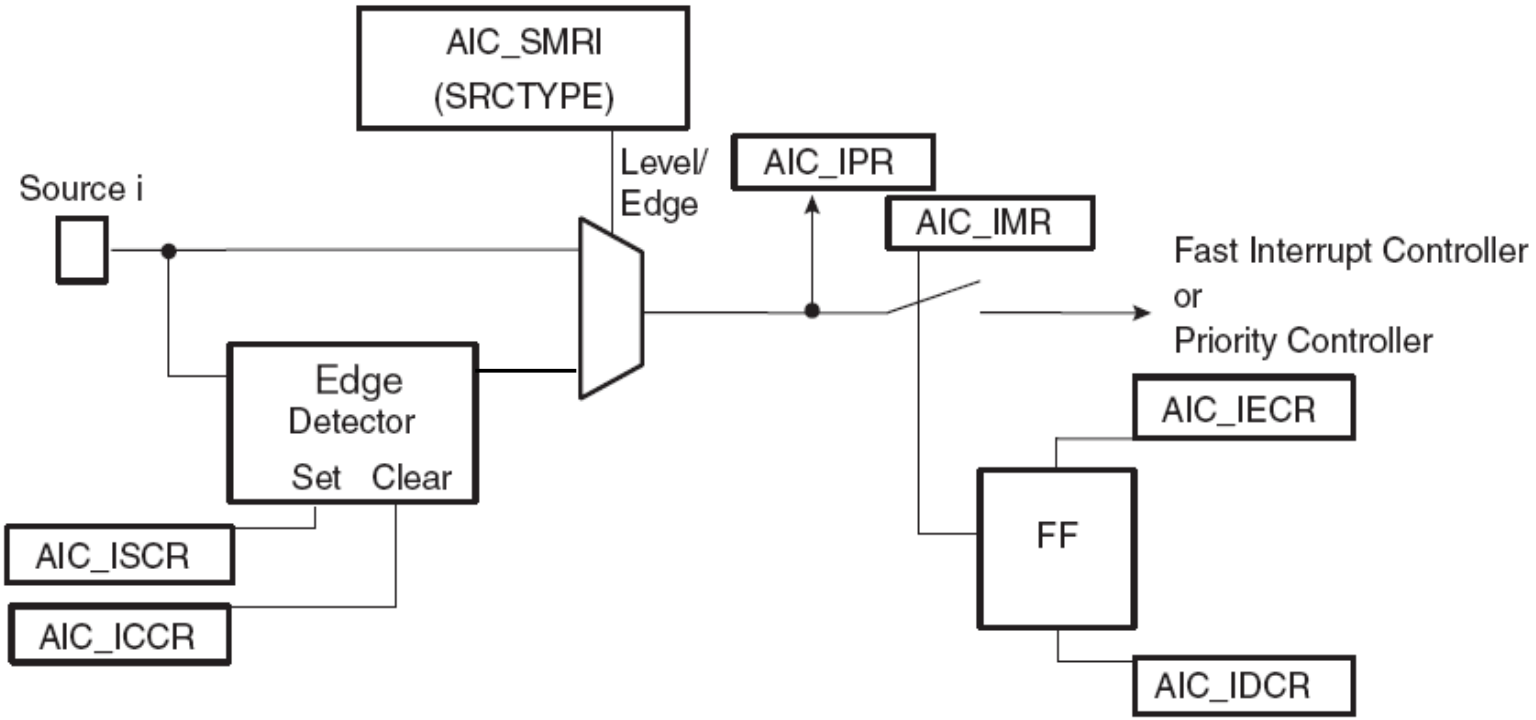
- timery PIT, RTT, WDT,
- interfejs diagnostyczny DBGU,
- Sterownik DMA moduł PMC,
- Układ zerowania procesora RSTC,
- Sterownik pamięci MC.

W procedurze obsługi przerwania SYS należy sprawdzić kolejno stan wszystkich urządzeń (odmaskowane przerwanie). Jeżeli przerwanie jest aktywne należy sprawdzić flagę sygnalizującą przerwanie w rejestrze statusu danego urządzenia. Jeżeli flaga jest ustawiona należy wykonać program związany z obsługą przerwania od danego urządzenia.

Schemat blokowy sterownika AIC

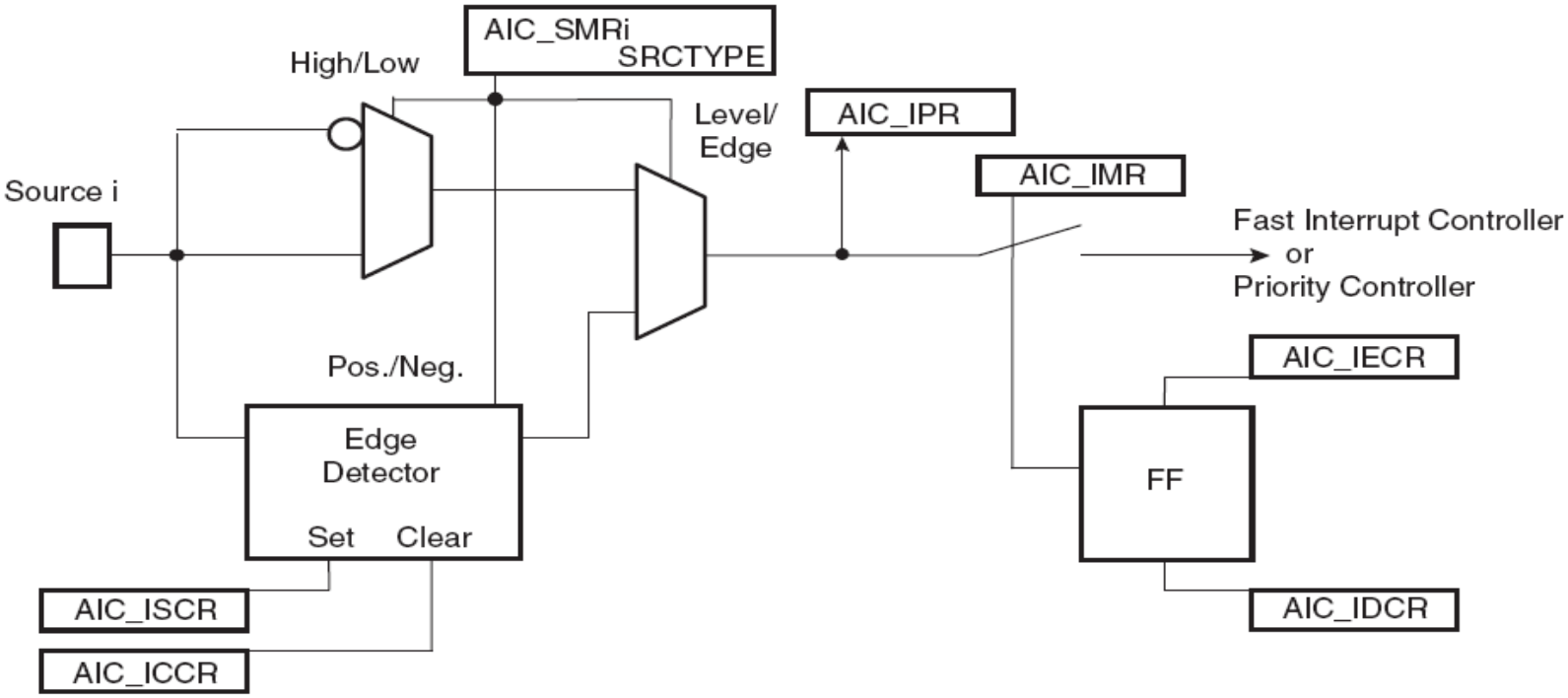


Przerwania wewnętrzne



- Maska przerwań – AIC_IECR/IDCR (status → AIC_IMR),
- Wyczyszczenie flagi przerwania podczas odczytu rejestru AIC_IVR (przerwania FIQ → AIC_FVR),
- Status przerwań dostępny w rejestrze AIC_IPR

Przerwania zewnętrzne



- Możliwość wyboru zbocza opadającego/narastającego lub poziomu niskiego/wysokiego do generacji przerwań.

Definicja numerów urządzeń peryferyjnych

```
// ********************************************************************************
//               PERIPHERAL ID DEFINITIONS FOR AT91SAM9263
// ********************************************************************************
#define AT91C_ID_FIQ    ( 0) // Advanced Interrupt Controller (FIQ)
#define AT91C_ID_SYS    ( 1) // System Controller
#define AT91C_ID_PIOA   ( 2) // Parallel IO Controller A
#define AT91C_ID_PIOB   ( 3) // Parallel IO Controller B
#define AT91C_ID_PIOCDE ( 4) // Parallel IO Controller C, Parallel IO Controller D, Parallel IO Controller E
#define AT91C_ID_US0    ( 7) // USART 0
#define AT91C_ID_US1    ( 8) // USART 1
#define AT91C_ID_US2    ( 9) // USART 2
#define AT91C_ID_MCI0   (10) // Multimedia Card Interface 0
#define AT91C_ID_MCI1   (11) // Multimedia Card Interface 1
#define AT91C_ID_CAN    (12) // CAN Controller
#define AT91C_ID_TWI    (13) // Two-Wire Interface
#define AT91C_ID_SPI0   (14) // Serial Peripheral Interface
```

ID=0, ID=30-31 przerwania zewnętrzne, pozostałe to przerwania wewnętrzne.

Rejestry sterownika przerwań (1)

Offset	Register	Name	Access	Reset
0x00	Source Mode Register 0	AIC_SMR0	Read-write	0x0
0x04	Source Mode Register 1	AIC_SMR1	Read-write	0x0
---	---	---	---	---
0x7C	Source Mode Register 31	AIC_SMR31	Read-write	0x0
0x80	Source Vector Register 0	AIC_SVR0	Read-write	0x0
0x84	Source Vector Register 1	AIC_SVR1	Read-write	0x0
---	---	---	---	---
0xFC	Source Vector Register 31	AIC_SVR31	Read-write	0x0
0x100	Interrupt Vector Register	AIC_IVR	Read-only	0x0
0x104	FIQ Interrupt Vector Register	AIC_FVR	Read-only	0x0
0x108	Interrupt Status Register	AIC_ISR	Read-only	0x0
0x10C	Interrupt Pending Register(3)	AIC_IPR	Read-only	0x0(1)
0x110	Interrupt Mask Register(3)	AIC_IMR	Read-only	0x0
0x114	Core Interrupt Status Register	AIC_CISR	Read-only	0x0
0x118 - 0x11C	Reserved	---	---	---
0x120	Interrupt Enable Command Register(3)	AIC_IECR	Write-only	---
0x124	Interrupt Disable Command Register(3)	AIC_IDCR	Write-only	---
0x128	Interrupt Clear Command Register(3)	AIC_ICCR	Write-only	---
0x12C	Interrupt Set Command Register(3)	AIC_ISCR	Write-only	---
0x130	End of Interrupt Command Register	AIC_EOICR	Write-only	---
0x134	Spurious Interrupt Vector Register	AIC_SPU	Read-write	0x0
0x138	Debug Control Register	AIC_DCR	Read-write	0x0
0x13C	Reserved	---	---	---
0x140	Fast Forcing Enable Register(3)	AIC_FFER	Write-only	---
0x144	Fast Forcing Disable Register(3)	AIC_FFDR	Write-only	---
0x148	Fast Forcing Status Register(3)	AIC_FFSR	Read-only	0x0
0x14C - 0x1E0	Reserved	---	---	---
0x1EC - 0x1FC	Reserved			

Rejestry sterownika przerwań

```
typedef struct _AT91S_AIC {
    AT91_REG    AIC_SMR[32];    // Source Mode Register
    AT91_REG    AIC_SVR[32];    // Source Vector Register
    AT91_REG    AIC_IVR;        // IRQ Vector Register
    AT91_REG    AIC_FVR;        // FIQ Vector Register
    AT91_REG    AIC_ISR;        // Interrupt Status Register
    AT91_REG    AIC_IPR;        // Interrupt Pending Register
    AT91_REG    AIC_IMR;        // Interrupt Mask Register
    AT91_REG    AIC_CISR;       // Core Interrupt Status Register
    ...
} AT91S_AIC, *AT91PS_AIC;
#define AT91C_BASE_AIC      (AT91_CAST(AT91PS_AIC)   0xFFFFF000) // (AIC)
    Base Address
```

Rejestry sterownika przerwań (2)

AIC_SMR[32]; // Source Mode Register – konfiguracja poziomu oraz sposobu wyzwalania

AIC_SVR[32]; // Source Vector Register – uchwyty obsługujące przerwania

AIC_IVR; // IRQ Vector Register – adres uchwytu do obsługiwanego przerwania

AIC_FVR; // FIQ Vector Register – adres uchwytu do obsługiwanego przerwania

AIC_ISR; // Interrupt Status Register – numer obsługiwanego przerwania

AIC_IPR; // Interrupt Pending Register – rejestr z flagami oczekujących przerwań 0-31

AIC_IMR; // Interrupt Mask Register – rejestr z maskami przerwań 0-31

AIC_CISR; // Core Interrupt Status Register – stan przerwań rdzenia IRQ/FIQ

AIC_IECR; // Interrupt Enable Command Register – rejestr uaktywniający przerwania

AIC_IDCR; // Interrupt Disable Command Register – rejestr wyłączający przerwania

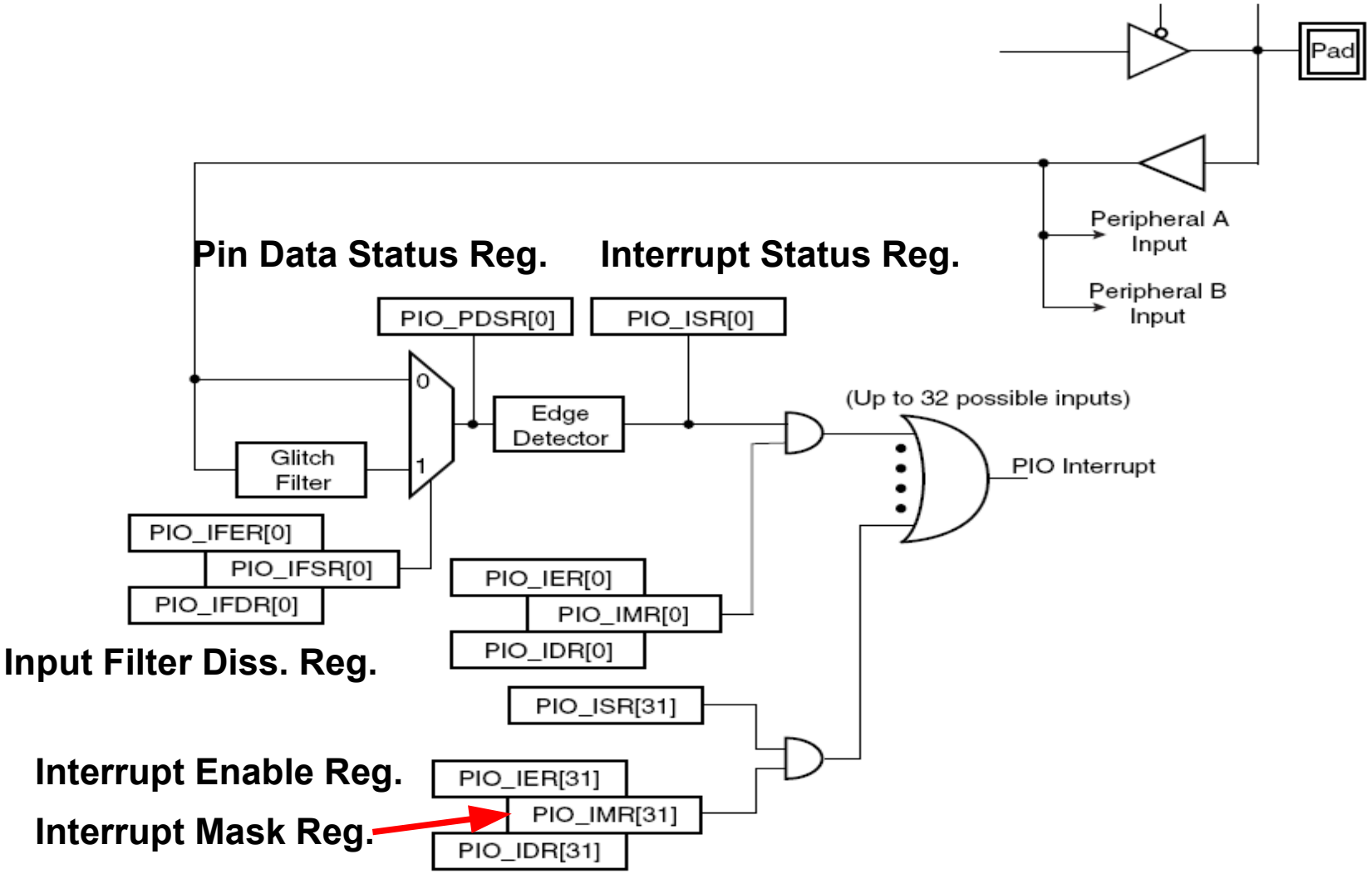
AIC_ICCR; // Interrupt Clear Command Register – rejestr kasujący flagi przerwań

AIC_ISCR; // Interrupt Set Command Register – rejestr ustawiający flagi przerwań

AIC_EOICR; // End of Interrupt Command Register – koniec obsługi przerwania

AIC_SPU; // Spurious Vector Register – uchwyt do przerwania fałszywego

I/O – przerwania



Procedura obsługująca przerwanie od timera PIT i klawiatury

Ustawienie adresu funkcji (handlera) obsługującego przerwanie (adres 32-bitowy)

```
AT91C_BASE_AIC->AIC_SVR[AT91C_ID_SYS] = (unsigned long) TIMER_handler;
```

Procedura obsługi przerwania od timera

```
void TIMER_handler (void) {
    Odczyt rejestru statutowego PITC_PISR  (PIT_SR)
    jeżeli flaga od timera INT_ENABLE jest ustawiona (rejestr PITC_PIMR) to odczyt
       rejestru PITC_PIVR - skasowanie flagi przerwania
    jeżeli nie to inne urządzenie peryferyjne zgłosiło przerwanie – odpowiednia reakcja
}
```

Procedura obsługi przerwania od klawiatury

```
void BUTTON_handler (void) {
    Odczyt rejestru statutowego PIO_ISR - skasowanie flagi przerwania
    jeżeli flaga na odpowiednim bicie rejestru PIO_ISR jest ustawiona to oznacza to
       wciśnięcie przycisku
    Obsługa wszystkich linii wejściowych (może zostać zgłoszone więcej niż jedno
       przerwanie)
}
```

Potwierdzenie obsługi przerwania

- W celu poprawnego zakończenia obsługi przerwania należy:
  - Potwierdzić przerwanie kasując flagę przerwania dla danego urządzenia peryferyjnego, np. odczyt rejestru SR dla portów IO (czasami flaga jest czyszczona automatycznie po wykonaniu operacji związanej z danym urządzeniem, np. zapisanie nowej danej do rejestru nadawczego UART_Tx),
  - Potwierdzić zakończenie procedury obsługi przerwania IRQ lub FIQ przez wykonanie dowolnej operacji na rejestrze EOICR.
  - Programy na laboratorium wykorzystują pomocniczą funkcję do obsługi przerwań IRQ, która automatycznie wykonuje kopię rejestrów oraz obsługuje rejestr potwierdzenia EOICR.

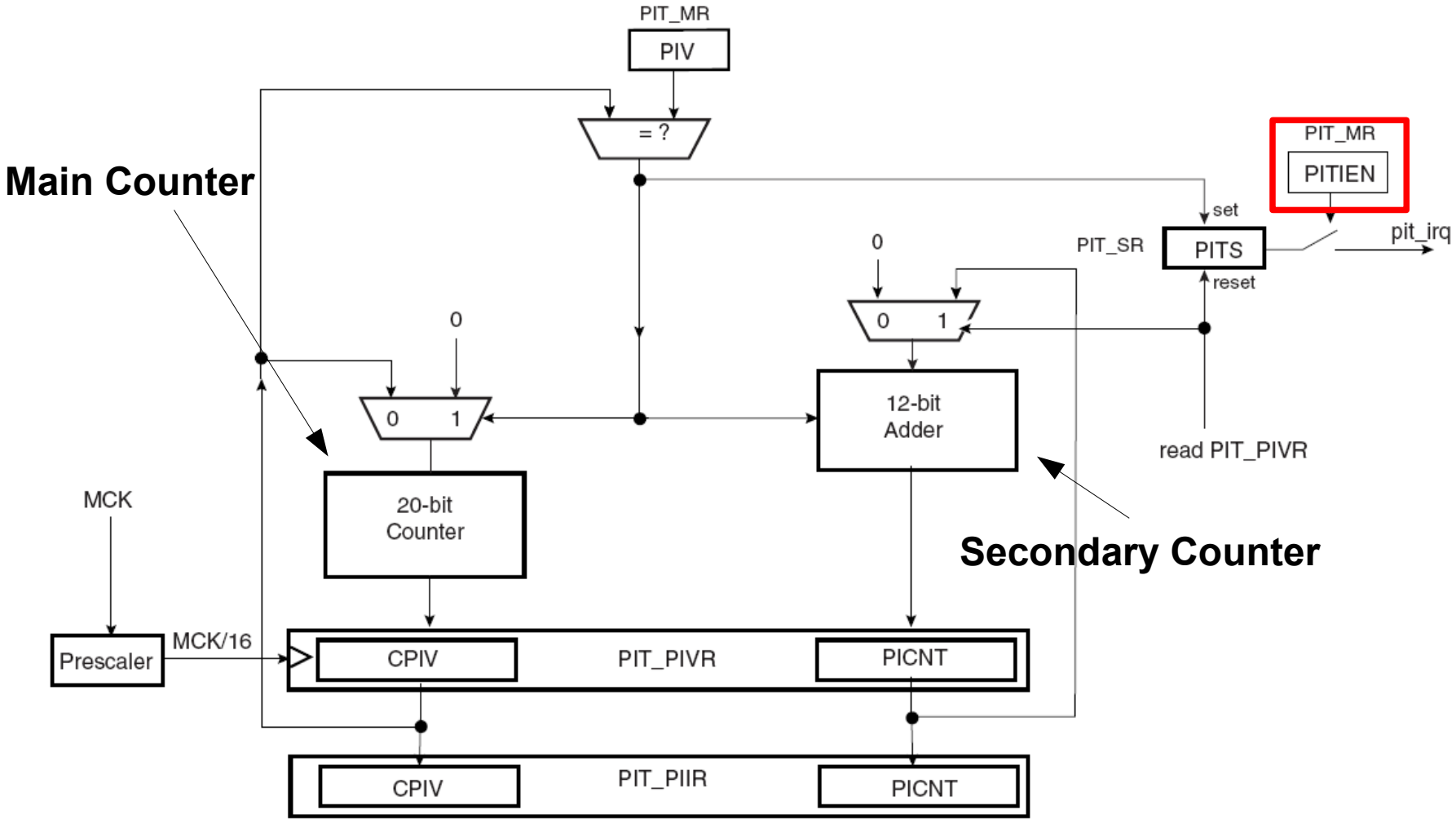
Konfiguracja przerwań od klawiatury

Przyciski dołączone są do portu C – przerwania generowane przez układy wejściowe portu C/D/E (maska AT91C_ID_PIOCDE)

Konfiguracja przerwań od portów C/D/E:

1. Konfiguracja portów procesora jako porty wejściowe (przycisk lewy i prawy)
2. Wyłączenie przerwań generowanych przez porty C/D/E (rejestr AIC_IDCR, AT91C_ID_PIOCDE)
3. Ustawienie wskaźnika do procedury obsługującej przerwanie dla portów C/D/E w tablicy wektorów SVR (AIC_SVR[AT91C_ID_PIOCDE])
4. Konfiguracja poziomu i metody wyzwalania przerwania (rejestr AIC_SMR, wyzwalanie wysokim poziomem AT91C_AIC_SRCTYPE_EXT_HIGH_LEVEL oraz priorytet AT91C_AIC_PRIOR_HIGHEST)
5. Wyczyszczenie flagi przerwania portów C/D/E (rejestr AIC_ICCR)
6. Włącznie przerwań dla obu przycisków (rejestr PIO_IER)
7. Włączenie przerwania od portów C/D/E (AIC_IECR)
8. Włączenie portu IO

Przerwanie od timera PIT



Konfiguracja przrwań od Timera PIT

Timer PIT generuje przerwania o numerze 1 – przerwania od urządzeń peryferyjnych (System Controller, maska AT91C_ID_SYS)

Konfiguracja przerwań od Timera PIT

1. Konfiguracja okresu timera, np. 5 ms
2. Wyłączenie przerwania od Timera PIT na czas konfiguracji (AIC_IDCR, przerwanie nr 1 - urządzenia peryferyjne procesora, stała AT91C_ID_SYS)
3. Ustawienie wskaźnika do procedury obsługującej przerwanie dla urządzeń peryferyjnych w tablicy wektorów AIC_SVR (AIC_SVR[AT91C_ID_SYS])
4. Konfiguracja poziomu i metody wyzwalania przerwania (rejestr AIC_SMR, wyzwalanie AT91C_AIC_SRCTYPE_INT_LEVEL_SENSITIVE, dowolny poziom, np. AT91C_AIC_PRIOR_LOWEST)
5. Wyczyszczenie flagi przerwania urządzeń peryferyjnych (rejestr AIC_ICCR)
6. Włącznie przerwania urządzeń peryferyjnych AT91C_ID_SYS (rejestr AIC_IECR)
7. Włączenie przerwania od Timera (AT91C_PITC_PITIEN)
8. Włączenie Timera PIT (AT91C_PITC_PITEN)
9. Wyzerowanie tzw. licznika lokalnego Timera (zmienna Local_Counter)

Przerwania od transceiwera DBGU (1)

DGBU generates system interrupt (ID number 1) – interrupt from processor peripheral devices (System Controller, mask AT91C_ID_SYS). We have distinguish which device triggered interrupt. A few interrupts can be triggered.

DGBU can generate the following interrupts:

- RXRDY: Enable RXRDY Interrupt
- TXRDY: Enable TXRDY Interrupt
- ENDRX: Enable End of Receive Transfer Interrupt
- ENDTX: Enable End of Transmit Interrupt
- OVRE: Enable Overrun Error Interrupt
- FRAME: Enable Framing Error Interrupt
- PARE: Enable Parity Error Interrupt
- TXEMPTY: Enable TXEMPTY Interrupt
- TXBUFE: Enable Buffer Empty Interrupt
- RXBUFF: Enable Buffer Full Interrupt
- COMMTX: Enable COMMTX (from ARM) Interrupt
- COMMRX: Enable COMMRX (from ARM) Interrupt

Przerwania od transceiwera DBGU (2)

DGBU interrupt handler

```
void DGBU_INT_handler (void) {
        int IntStatus;
        SysIRQCounter++;              /* to have a feeling how many system INTs are triggered */
        IntStatus = DGBU->SR;
        if (IntStatus & DBGU->IMR )   /* interrupt from DGBU */
              if INT from TxD         /* transmitter interrupt */
                    WriteNewData ();  /* be careful INTcan be also generated in case of error */
              else if INT from RxD
                    ReadDataToBuffer();/* INT can be also generated when error occur */
              else
                    other device triggered INT;
}
```

Funkcje obsługujące przerwania w języku C (1)

Deklaracja procedur obsługujących przerwania (kompilator GCC) wymaga użycia dyrektywy preprocesora __attribute__ ((interrupt("IRQ")));

```
void INTButton_handler()__attribute__ ((interrupt("IRQ")));
void INTPIT_handler()__attribute__ ((interrupt("IRQ")));
void Soft_Interrupt_handler()__attribute__ ((interrupt("SWI")));
void Abort_Exception_handler()__attribute__ ((interrupt("ABORT")));
void Undef_Exception_handler()__attribute__ ((interrupt("UNDEF")));
```

Funkcja obsługująca przerwanie nie różni się od klaszycznej funkcji w języku C

```
void INTButton_handler() {
      // standard C function
}
```

Podczas pisania programów na laboratorium nie należy używać atrybutu __attribute__ ((interrupt("IRQ"))) – przerwania zagnieżdżone obsługiwane są przez funkcję umieszczoną w pliku startowym (startup.S).

Funkcje obsługujące przerwania w języku C (2)

```
.size     Open_INTButtons, .-Open_INTButtons
.align    2
.global   INTButton_handler
INTButton_handler:
@ Interrupt Service Routine.
@ args = 0, pretend = 0, frame = 8
@ frame_needed = 1, uses_anonymous_args = 0
str  ip, [sp, #-4]!      // store R12
mov  ip, sp              // move sp to ip
stmfd     sp!, {r1, r2, r3, fp, ip, lr, pc}
sub  fp, ip, #4          // allocate dataframe
sub  sp, sp, #8          // on stack for local
                         // data
```

Właściwy program obsługujący przerwanie

```
sub  sp, fp, #24         // rem. frame from st
ldmfd     sp, {r1, r2, r3, fp, sp, lr}
ldmfd     sp!, {ip}      // recover R12
subs pc, lr, #4          // return from INT
```

```
.size     Open_INTButtons, .-Open_INTButtons
.align    2
.global   INTButton_handler
.type     INTButton_handler, %function
INTButton_handler:
@ args = 0, pretend = 0, frame = 8
@ frame_needed = 1, uses_anonymous_args = 0


mov  ip, sp              // store R12
stmfd     sp!, {fp, ip, lr, pc}
sub  fp, ip, #4          // allocate dataframe
sub  sp, sp, #8          // on stack for local
                         //  data
```

Właściwy program obsługujący przerwanie

```
sub  sp, fp, #12         // rem. frame from stac
ldmfd     sp, {fp, sp, pc} // recover registers
                           // and return from
                           // function
```