Technika Mikroprocesorowa

Dariusz Makowski

Katedra Mikroelektroniki i

Technik Informatycznych

tel. 631 2648

dmakow@dmcs.pl

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

Sprawy formalne

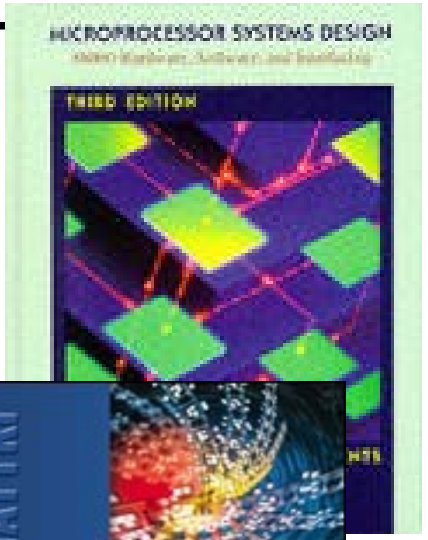
1. Zaliczenie
2. Projekt z Techniki Mikroprocesorowej
3. Materiały do wykładu

http://neo.dmcs.p.lodz.pl/tm/index.html

Literatura

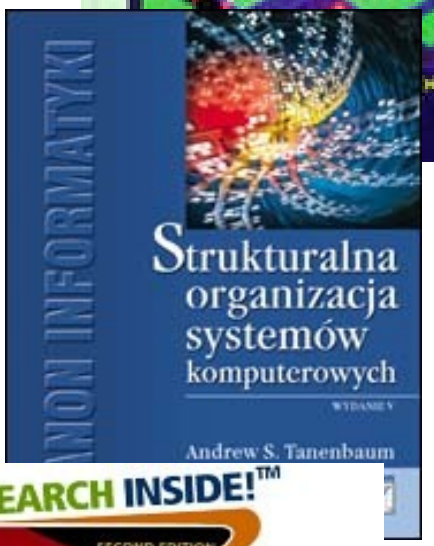
Literatura obowiązkowa:

Alan Clements, “Microprocessor Systems Design: 68000 Hardware, Software and Interfacing”, 3rd Edition, PWS '97

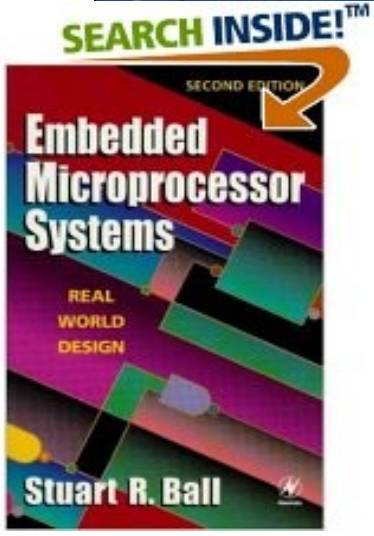


Literatura uzupełniająca:

Andrew S. Tanenbaum „Strukturalna organizacja systemów komputerowych”, wydanie V, Helion 2006



Stuart R. Ball, “Embedded Microprocessor Systems”
2nd Edition, Butterworth-Heinemann 2000



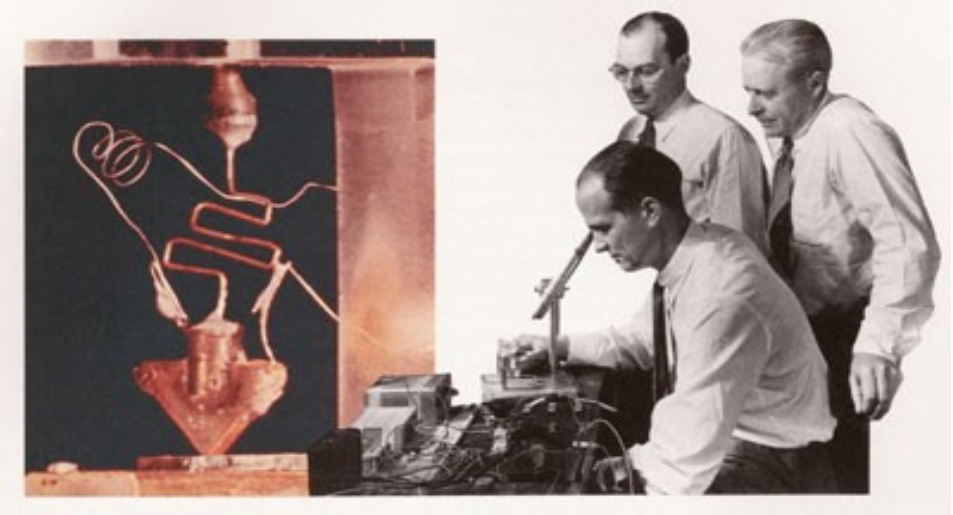
Zakres przedmiotu

1. Wstęp do systemów mikroprocesorowych.
2. Współpraca procesora z pamięcią. Pamięci półprzewodnikowe.
3. Architektura systemów mikroprocesorowych.
4. Współpraca procesora z urządzeniami peryferyjnymi.
5. Przykładowy system mikroprocesorowy.
6. Architektura procesorów 32-bitowych na przykładzie układów Freescale 68k/ColdFire.
7. Architektura mikrokontrolerów 8-bitowych.

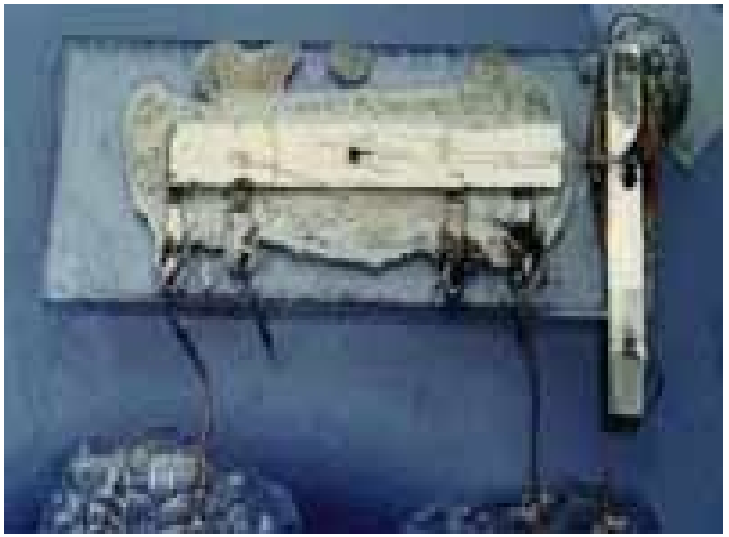
Historia mikroprocesorów (1)

1940 – Russell Ohl – demonstracja złącza półprzewodnikowego (dioda germanowa, bateria słoneczna)

1947 – Shockley, Bardeen, Brattain prezentują pierwszy tranzystor



Pierwszy tranzystor, Bell Laboratories



Pierwszy układ scalony, TI

1958 – Jack Kilby wynalazł pierwszy układ scalony

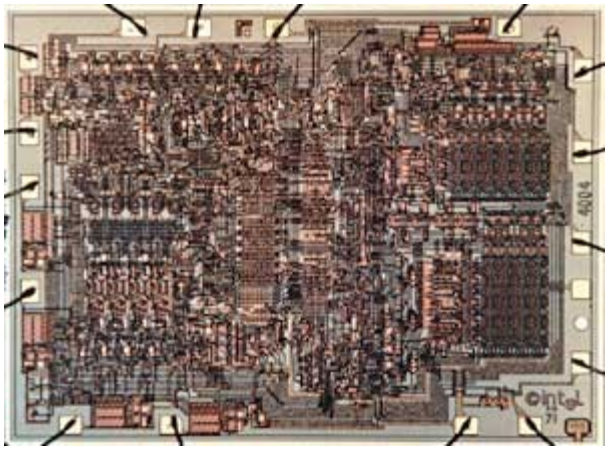
1967 – Laboratorium Fairchild oferuje pierwszą pamięć nieulotną ROM (64 bity)

1969 – Noyce i Moore opuszczają laboratorium Fairchild, powstaje niewielka firma INTEL. INTEL produkuje pamięci SRAM (64 bity). Japońska firma Busicom zamawia 12 różnych układów realizujących funkcje kalkulatorów.

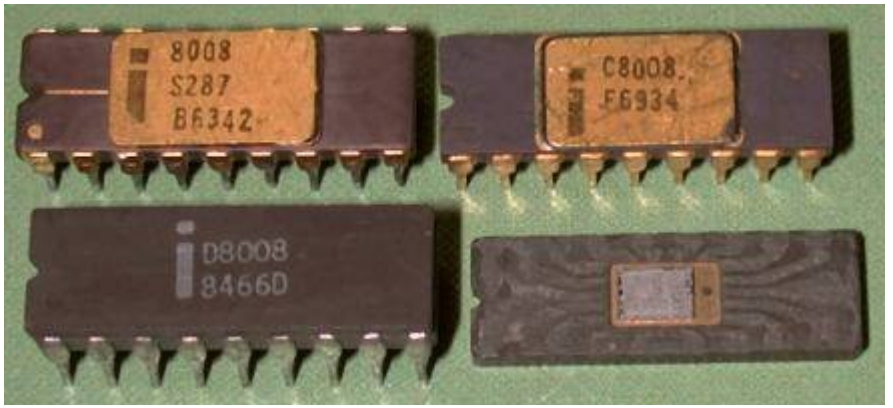
Historia mikroprocesorów (2)

1970 - **F14 CADC** (Central Air Data Computer) mikroprocesor zaprojektowany przez Steve'a Gellera i Raya Holta na potrzeby armii amerykańskiej (myśliwiec F-14 Tomcat)

1971 - **Intel 4004** 4-bitowy procesor realizujące funkcje programowalnego kalkulatora **(**powszechnie uznaje się za pierwszy na świecie mikroprocesor), 3200 tranzystorów. INTEL wznawia pracę nad procesorami, Faggin z Fairchild pomaga rozwiązać problemy.



Zdjęcie 4-bitowego procesora INTEL 4004



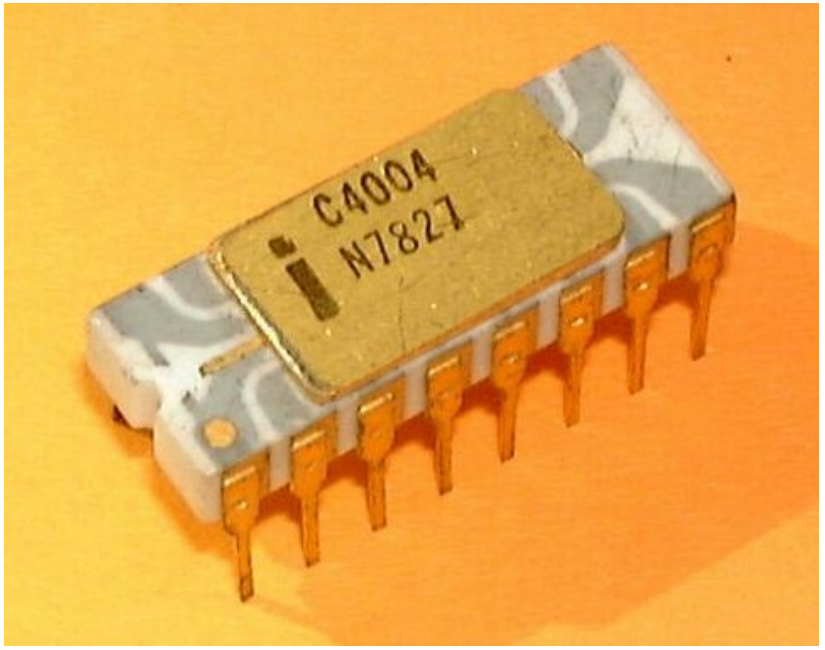
8-bitowe procesory INTEL-a

1972 – Faggin rozpoczyna prace nad 8-bitowym procesorem INTEL 8008. Rynek zaczyna się interesować układami “programowalnymi” - procesorami.

Kalkulator z procesorem Intel 4004

W skład każdego kalkulatora wchodziły cztery układy Intel 4001 (czyli łącznie 1KB), dwa 4002, dwa 4003 i jeden układ 4004.

- 4001 - 2 kbitowy (czyli 256 bajtów) pamięć ROM z 4-bitowym portem wejścia/wyjścia (4-bit mask-programmable I/O port),
- 4002 - 320 bitowa pamięć RAM z 4-bitowym portem we/wy,
- 4003 - 10-bitowy rejestr przesuwny - układ rozszerzający możliwości wejścia/wyjścia (10-bit serial-in, parallel-out shift register),
- **4004 - 4-bitowy CPU (4-bit parallel central processing unit).**



Historia mikroprocesorów (3)

1974 – INTEL wprowadza na rynek ulepszona wersję 8008, procesor Intel 8080. Faggin opuszcza firmę intel i zakłada firmę o nazwie Zilog. Motorola oferuje 8-bitowy procesor 6800 (NMOS, 5 V).

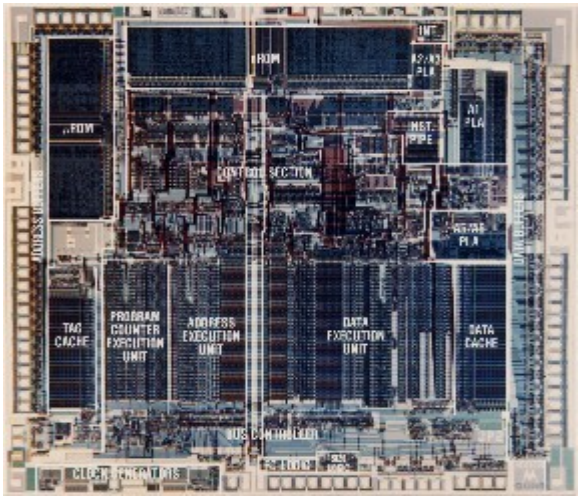
1976 – Zilog oferuję procesor Z80, INTEL pierwsza wersje procesora Intel 8048.

1978 – Pierwszy 16-bitowy procesor 8086 (ulepszony 8080).

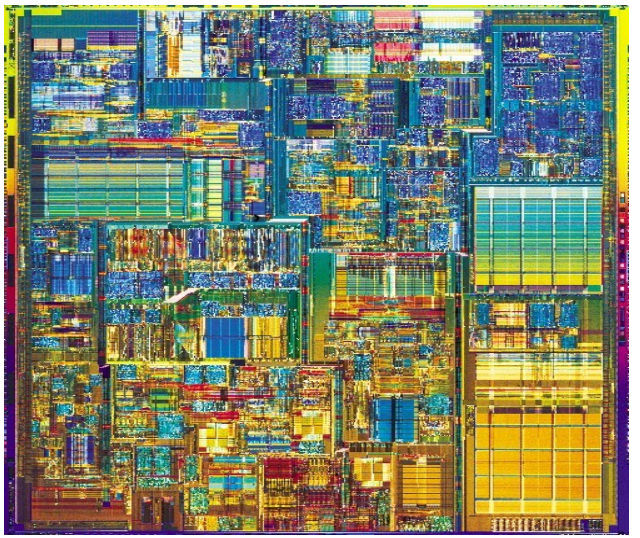
1979 – Motorola oferuje 16-bitowy procesor 68000.

1980 – Motorola wprowadza nowy 32-bitowy procesor 68020, 200,000 tranzystorów.

.....



Motorola 68020



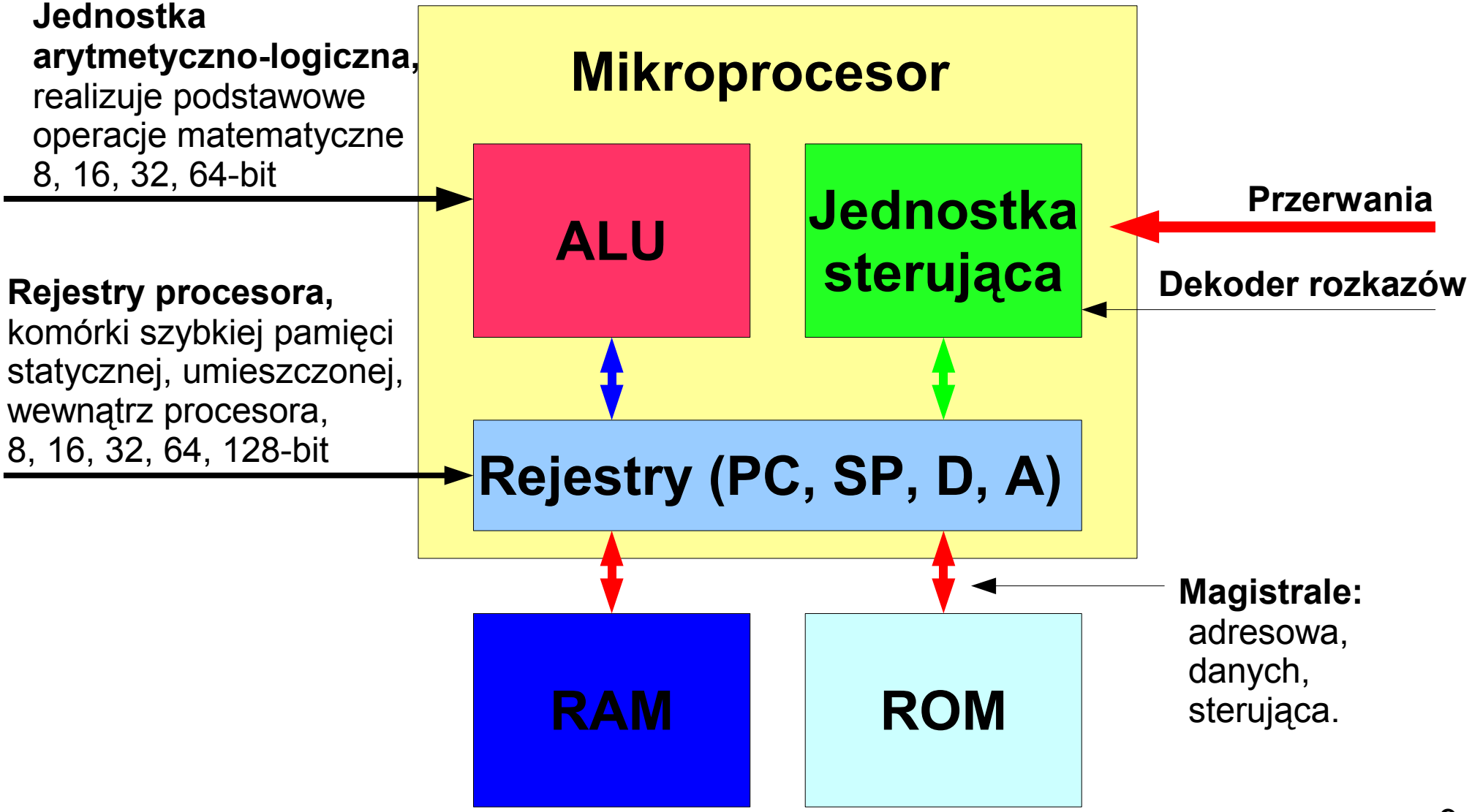
Intel, Pentium 4 Northwood

Intel 386, 486, Pentium I, II, III, IV, Centrino, Pentium D, Duo/Quad core, ...

Motorola 68030, 68040, 68060, PowerPC, ColdFire, ...

Mikroprocesor

Mikroprocesor to układ cyfrowy wykonany jako pojedynczy układ scalony o wielkim stopniu integracji zdolny do wykonywania operacji cyfrowych według dostarczonych mu instrukcji.



Architektura procesora CISC

Cechy architektury CISC (Complex Instruction Set Computers):

- Duża liczba rozkazów (instrukcji),
- Niektóre rozkazy potrzebują dużej liczby cykli procesora do wykonania,
- Występowanie złożonych, specjalistycznych rozkazów,
- Duża liczba trybów adresowania,
- Do pamięci może się odwoływać bezpośrednio duża liczba rozkazów,
- Mniejsza od układów RISC częstotliwość taktowania procesora,
- Powolne działanie dekodera rozkazów, ze względu na dużą ich liczbę i skomplikowane adresowanie

Przykłady rodzin procesorów o architekturze CISC to:

- x86
- **Motorola 68000**
- PDP-11 (Dec)
- AMD

Architektura procesora RISC

Cechy architektury RISC (Reduced Instruction Set Computer):

- Zredukowana liczba rozkazów. Upraszcza to znacznie dekoder rozkazów.
- Redukcja trybów adresowania, dzięki czemu kody rozkazów są prostsze,
- Ograniczenie komunikacji pomiędzy pamięcią, a procesorem. Do przesyłania danych pomiędzy pamięcią, a rejestrami służą dedykowane instrukcje (load, store) .
- Zwiększenie liczby rejestrów (np. 32, 192, 256),
- Dzięki przetwarzaniu potokowemu (ang. pipelining) wszystkie rozkazy wykonują się w jednym cyklu maszynowym.

Przykłady rodzin mikroprocesorów o architekturze RISC:

- IBM 801 (zaprojektowany w 1970 r.)
- PowerPC (Motorola, IBM)
- MIPS (zaprojektowany w 1984 r. Przez firmę MIPS Computer Systems)
- Alpha (64-bitwy procesor zaprojektowany przez firmę DEC)
- ARM (32-bitowy procesor, zaprojektowany przez Acorn Computers w 1983)
- Motorola 88000 (32-bitowy procesor)
- ColdFire (32-bitowy procesor)
- SPARC (32, 64-bitowy procesor zaprojektowany przez firmę SUN)
- PA-RISC (32-bitowy procesor, Hewlett-Packard)
- Atmel AVR (8-bitowy mikrokontroler)

Obecnie produkowane procesory Intela z punktu widzenia programisty są widziane jako CISC, ale ich rdzeń jest zgodny z RISC. Rozkazy CISC są rozbijane na mikrorozkazy (ang. microops), które są następnie wykonywane przez szybki blok wykonawczy zgodny z architekturą RISC.

Podział komputerów

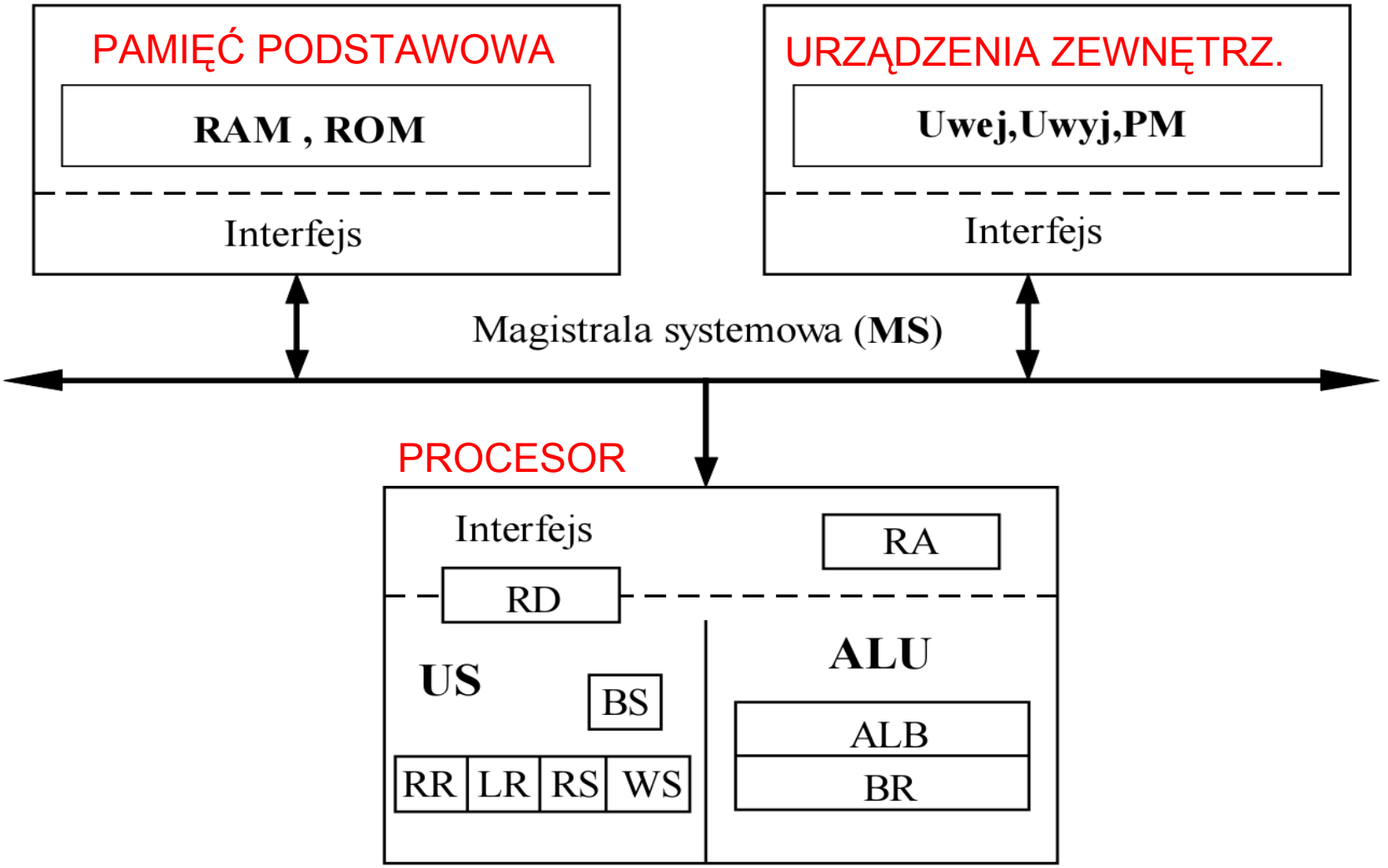
Mikrokomputery:

- **uniwersalne (desktop, Personal Computer)** – funkcjonalność urządzenia zależy głównie od posiadanego oprogramowania,
- **wbudowane (embedded)** – komputer, maszyna, sterownik przeznaczony do realizacji określonego zadania, np. sterowanie pralką automatyczną.

Architektura systemu komputerowego

Architektura polega na ścisłym podziale komputera na trzy podstawowe części:

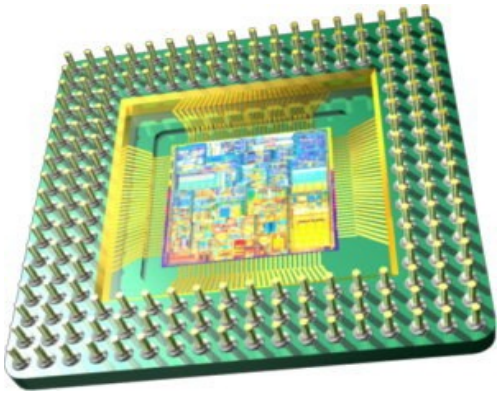
- procesor,
- pamięć (zawierająca dane oraz program),
- urządzenia wejścia/wyjścia (I/O).



Architektura von Neumanna

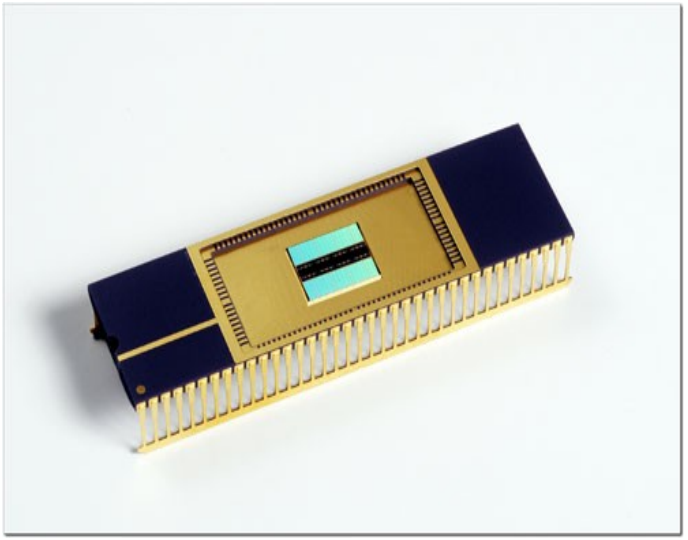
Cechy architektury **von Neumanna:**

- rozkazy i dane przechowywane są w tej samej pamięci,
- nie da sie rozróżnić danych o rozkazów (instrukcji),
- dane nie maja przypisanego znaczenia,
- pamięć traktowana jest jako liniowa tablica komórek, które identyfikowane są przy pomocy dostarczanego przez procesor adresu,
- procesor ma dostęp do przestrzeni adresowej, dekodery adresowe zapewniają mapowanie pamięci na rzeczywiste układy.



Magistrala adresowa

Magistrala danych

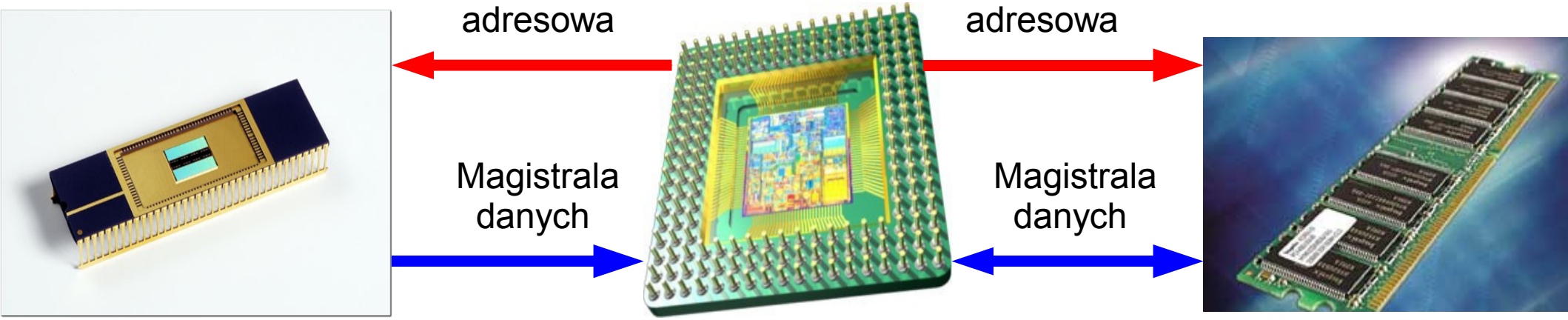


Architektura harwardzka

Prostsza (w stosunku do architektury Von Neumanna) budowa przekłada się na większą szybkość działania - dlatego ten typ architektury jest często wykorzystywany w procesorach sygnałowych oraz przy dostępie procesora do pamięci cache.

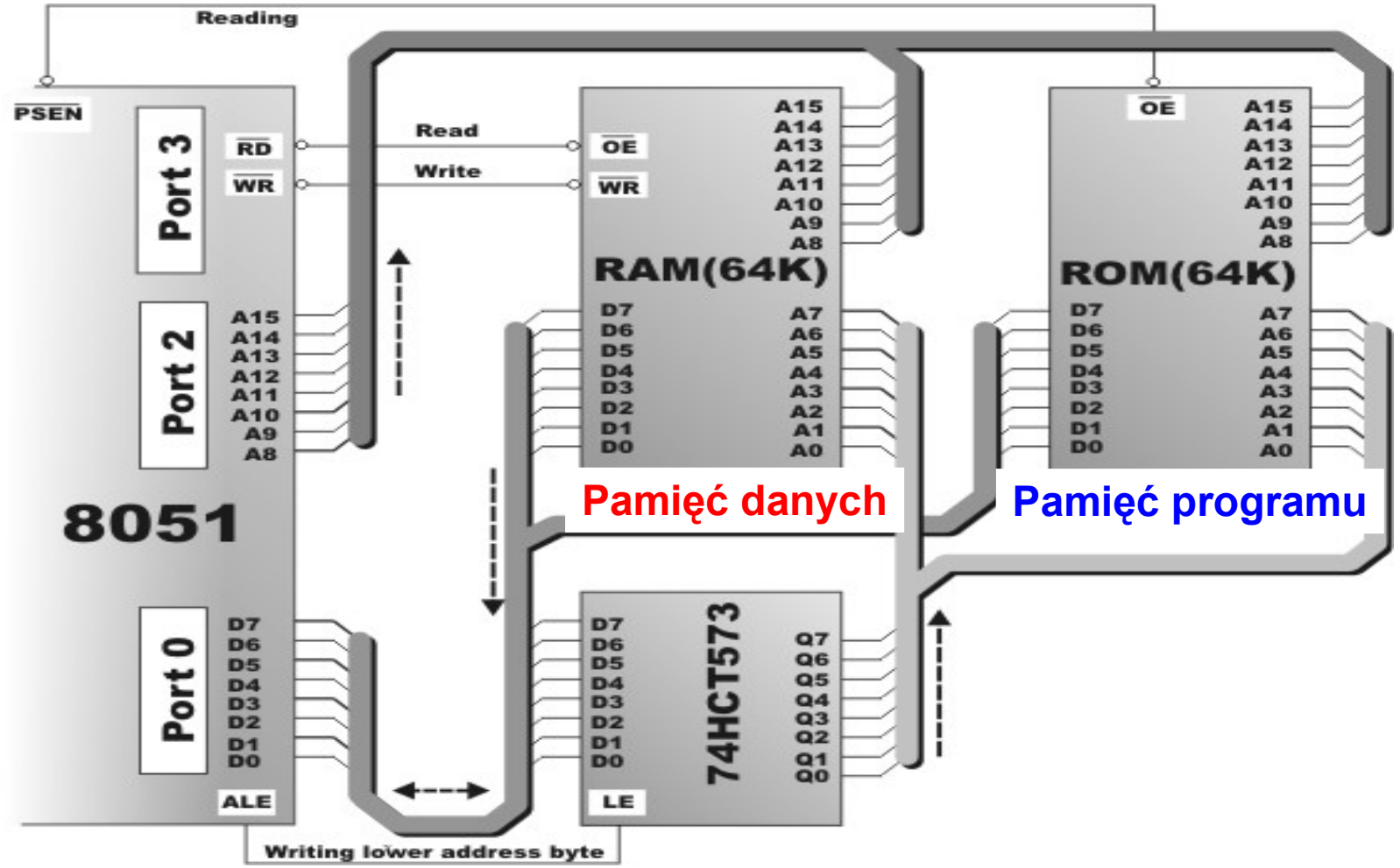
Cechy architektury **harwardzkiej:**

- rozkazy i dane przechowywane są w oddzielnych pamięciach,
- organizacja pamięci może być różna (inne długości słowa danych i rozkazów),
- możliwość pracy równoległej – jednoczesny odczyt danych z pamięci programu oraz danych,
- stosowana w mikrokontrolerach jednoukładowych.



Zmodyfikowana architektura harwardzka

Zmodyfikowana architektura harwardzka (architektura mieszana) - łączy w sobie cechy architektury harwardzkiej i architektury von Neumanna. Oddzielone zostały pamięci danych i rozkazów, lecz wykorzystują one wspólne magistrale danych i adresową. Architektura umożliwia łatwe przesyłanie danych pomiędzy rozdzielonymi pamięciami.

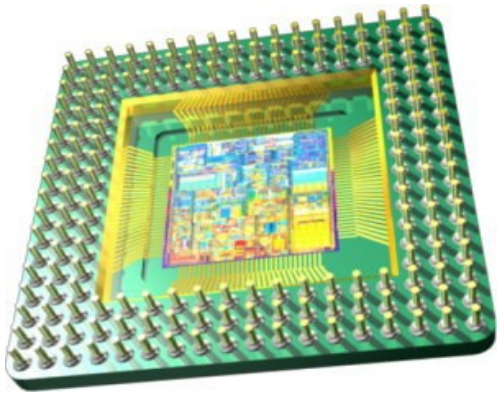


Przykład mikrokontrolera z rodziny 8051 wraz z zewnętrznymi pamięciami

Przypomnienie

- Definicja procesora,
- Architektura procesora,
- Architektura systemu komputerowego.

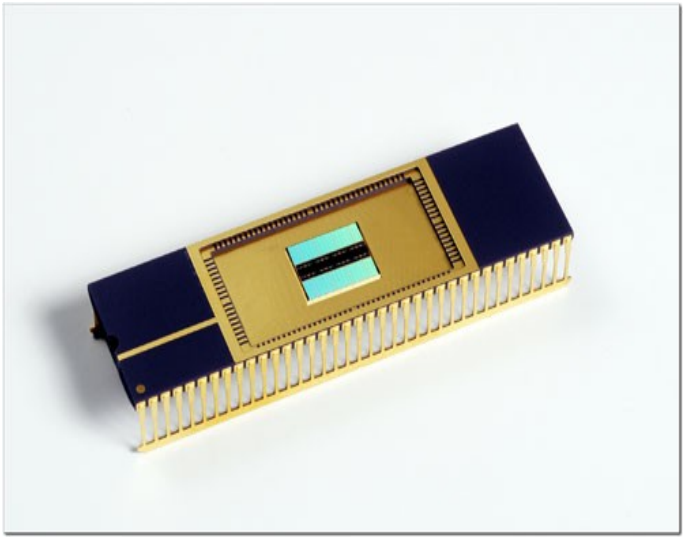
Magistrale komputera



Magistrala adresowa

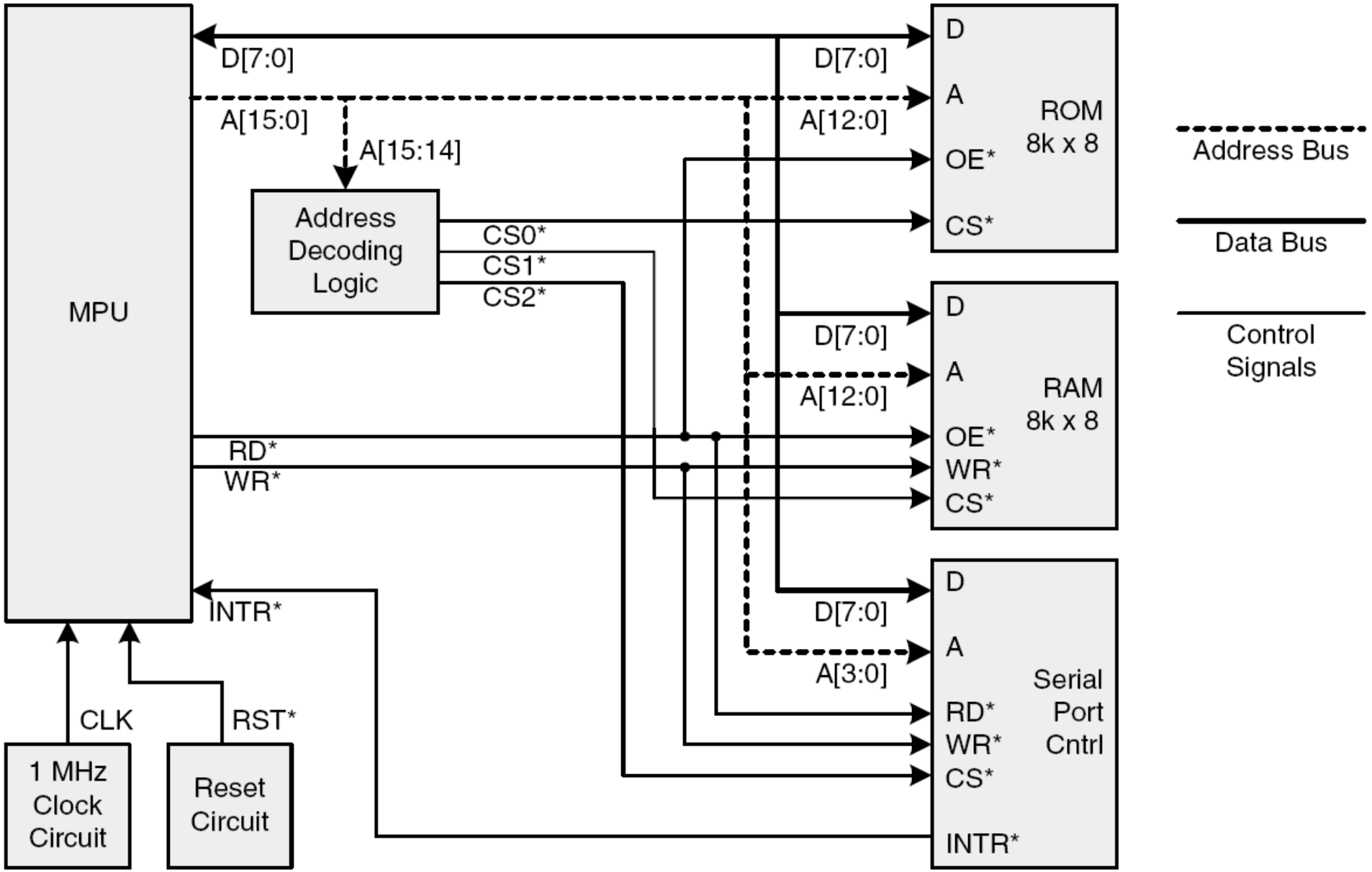
Magistrala danych

Magistrala sterująca

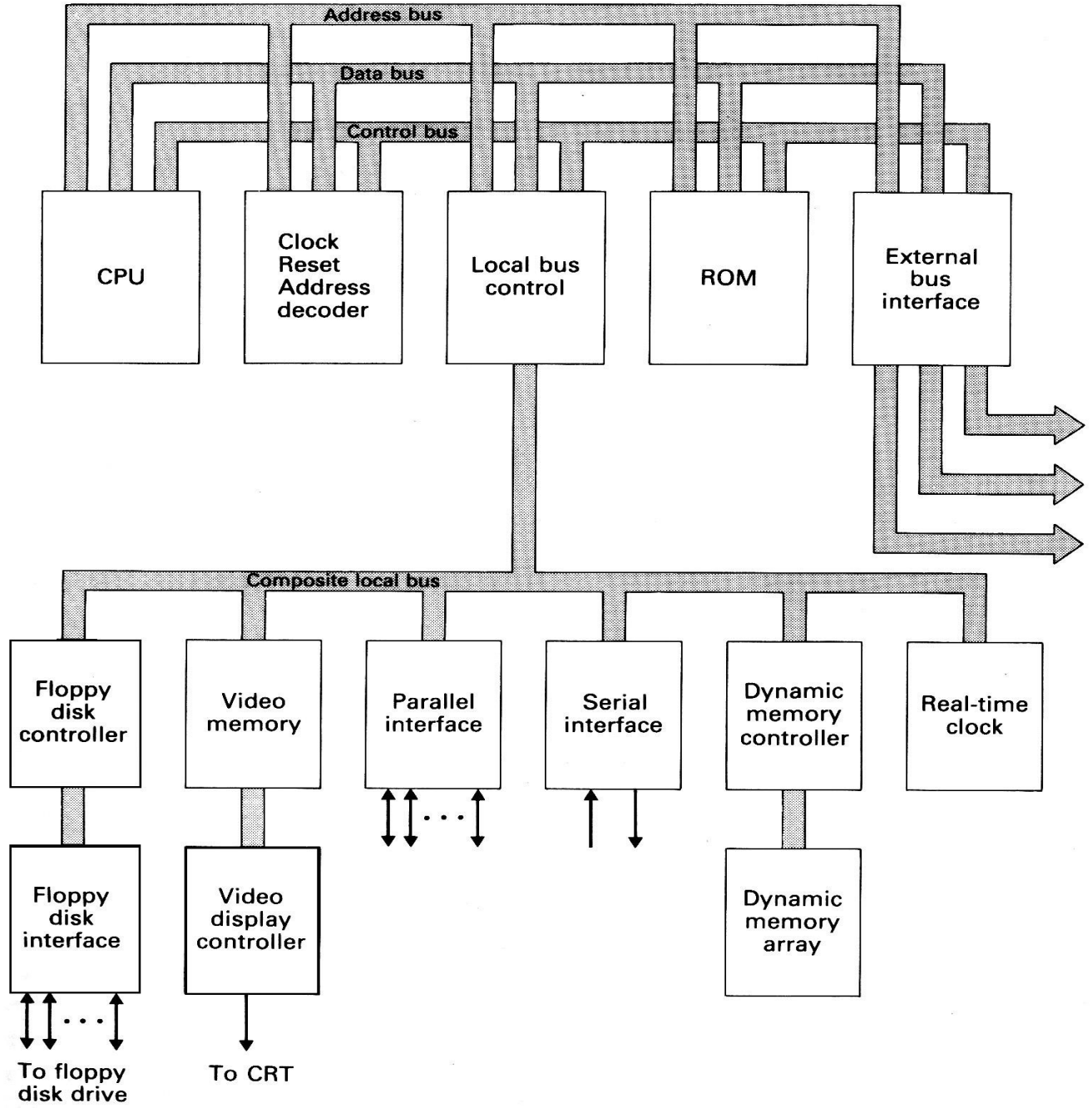


1. Rodzaj magistrali
2. Szerokość magistrali
3. Częstotliwość zegara – szybkość transmisji

Przykładowy komputer 8-bitowy

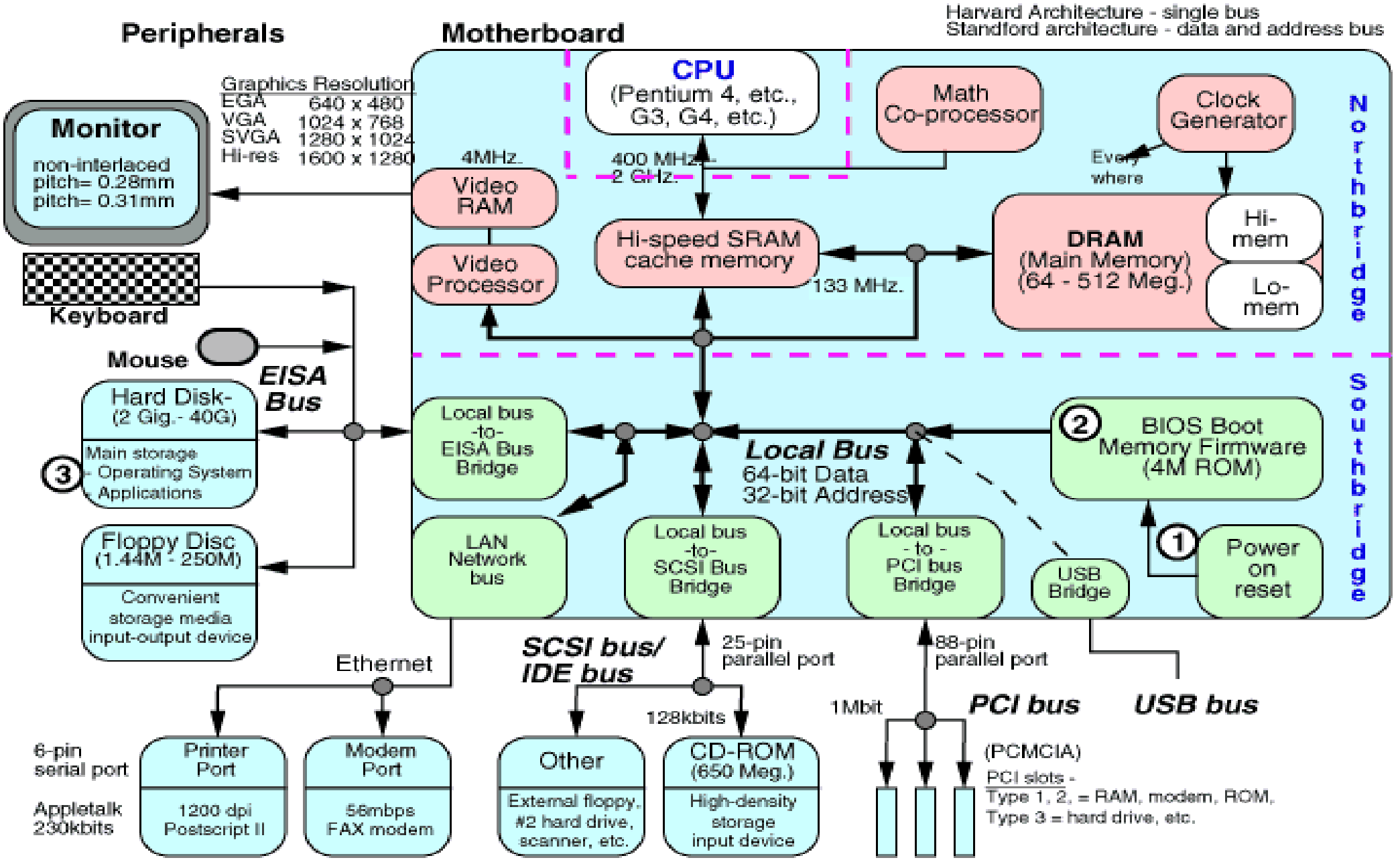


Komputer uniwersalny (1)



ISA
PCI
PCIe
AGP

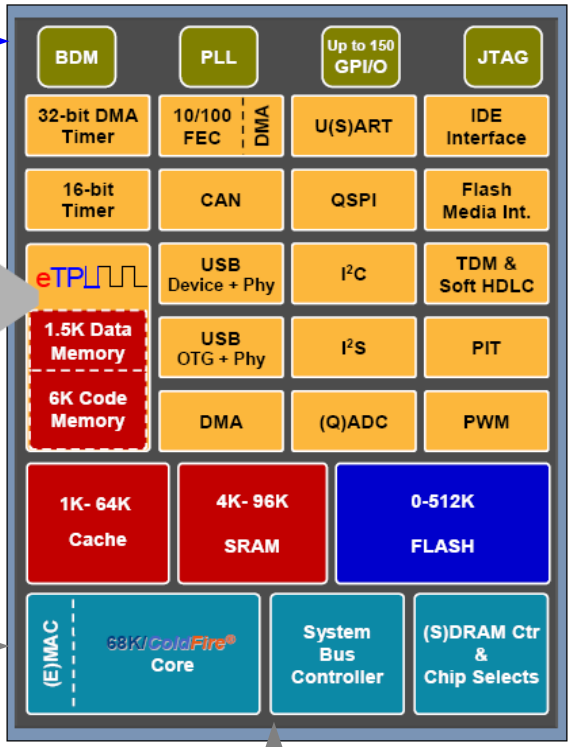
Komputer uniwersalny (2)



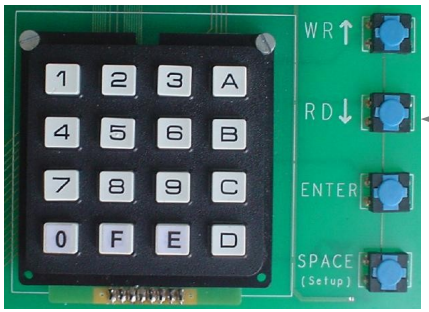
Komputer wbudowany (embedded computer)

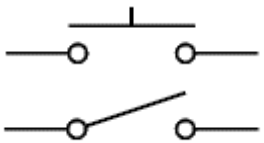
Komunikacja z komputerem zewnętrznym

Czujniki, np. czujnik temperatury, obrotów, itd



Kamera



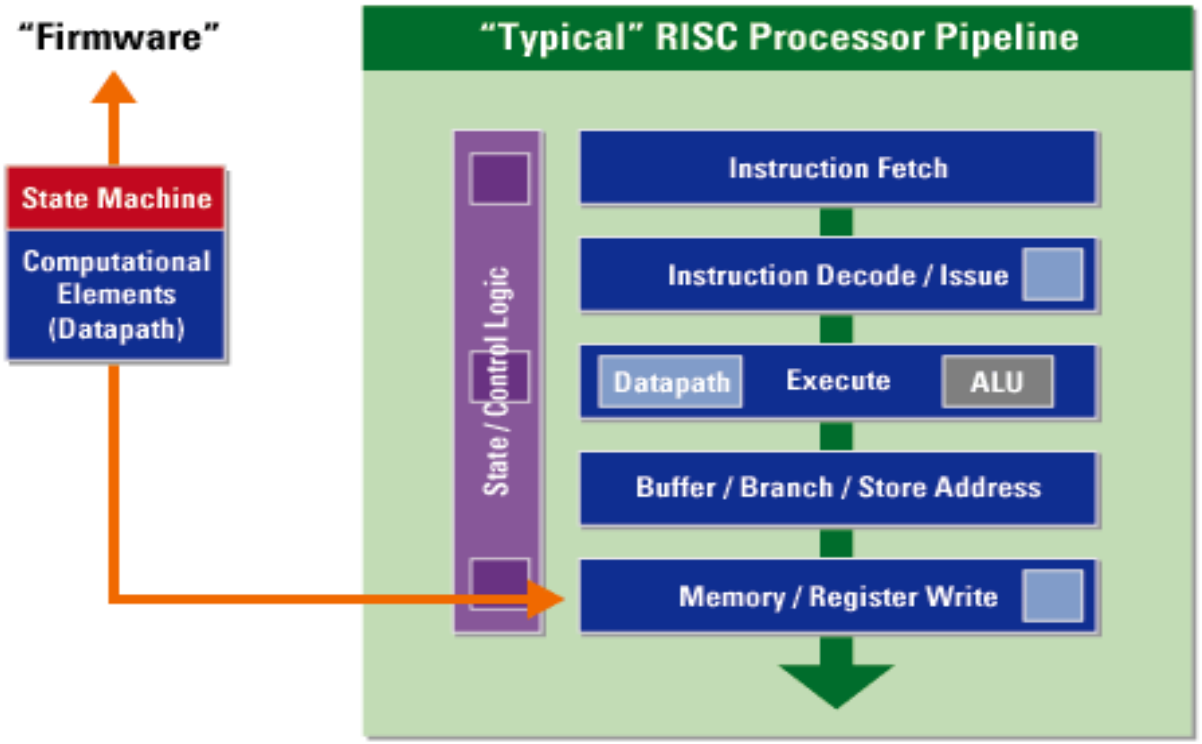


Elementy wykonawcze, np. silniki, przekaźniki

Cykle pracy procesora

Procesor pobiera rozkazy z pamięci i wykonuje je sekwencyjnie.

Wykonanie pojedynczej instrukcji zajmuje min. 5 cykli procesora (dla procesora przedstawionego na rysunku).

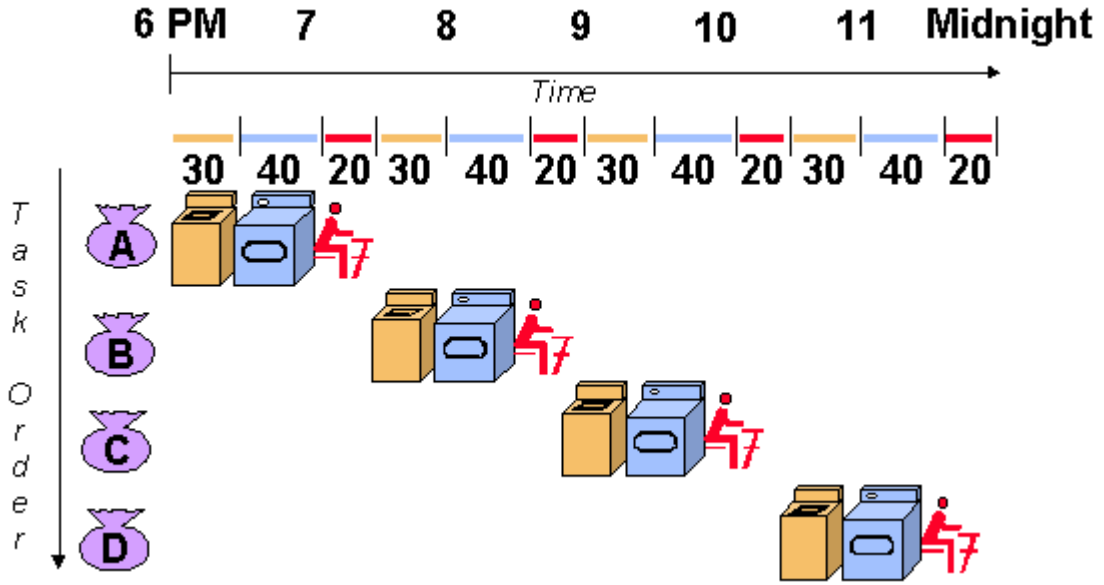


Pracę procesora można podzielić na kilka etapów:

1. Pobranie rozkazu z pamięci programu (Instruction Fetch, PC++),
2. Dekodowanie rozkazu, odczyt rejestrów (Instruction Decode),
3. Wykonanie rozkazu (Execute command - ALU),
4. Pobranie argumentów z pamięci danych (Memory Access),
5. Zapisanie wyniku operacji w pamięci (Write Back).

Procesor zawsze wykonuje jedną z powyższych czynności.

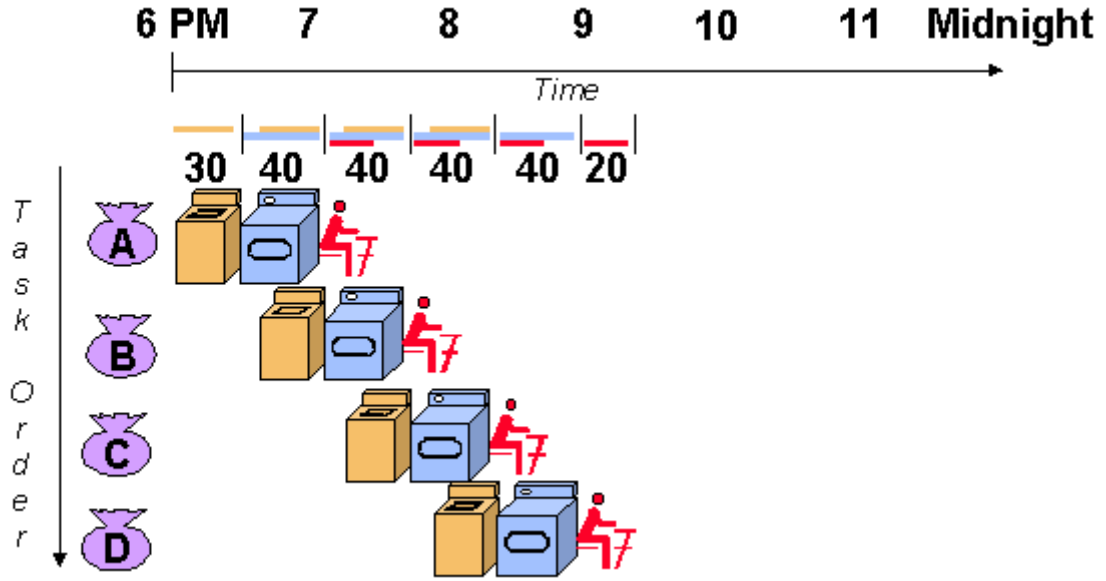
Potokowe wykonanie programu (1) – analogia do prania...



Zadanie:
Wykonanie prania, 4 wsady.

Praca wykonana sekwencyjnie.
Czas prania ~6 h.

Praca wykonana równolegle.
Czas prania ~3 h.



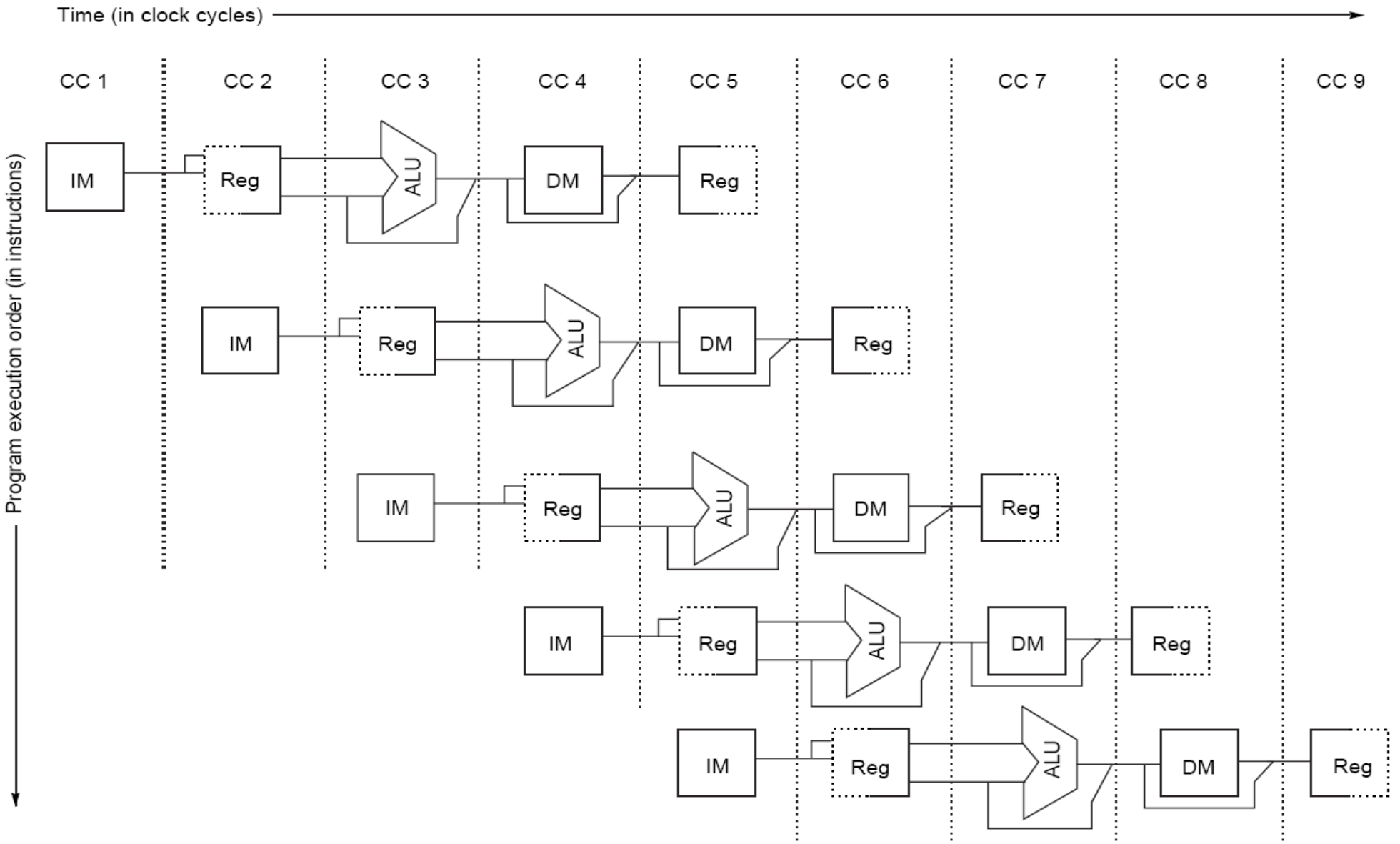
Potokowe wykonanie programu (2)

Potokowość (ang. pipelining) - technika budowy procesorów polegająca na podziale logiki procesora odpowiedzialnej za proces wykonywania programu (instrukcji procesora) na specjalizowane grupy w taki sposób, aby każda z grup wykonywała część pracy związanej z wykonaniem rozkazu. Grupy połączone są sekwencyjnie - potok (ang. pipe) i wykonują pracę równocześnie. W każdej z tych grup rozkaz jest w innym stadium wykonania. Można to porównać do taśmy produkcyjnej. W uproszczeniu, potok wykonania instrukcji procesora może wyglądać następująco:

	Clock number								
Instruction number	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
Instruction *i*	IF	ID	EX	MEM	WB				
Instruction *i* + 1		IF	ID	EX	MEM	WB			
Instruction *i* + 2			IF	ID	EX	MEM	WB		
Instruction *i* + 3				IF	ID	EX	MEM	WB	
Instruction *i* + 4					IF	ID	EX	MEM	WB

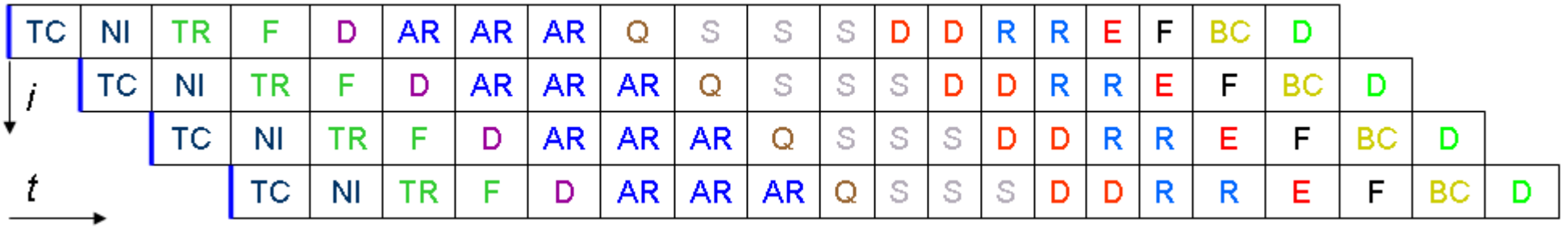
Co się stanie podczas wykonania instrukcji skoku?
Jak programista odczuwa wykorzystanie techniki potokowego wykonania programu?

Potokowe wykonanie programu (3)



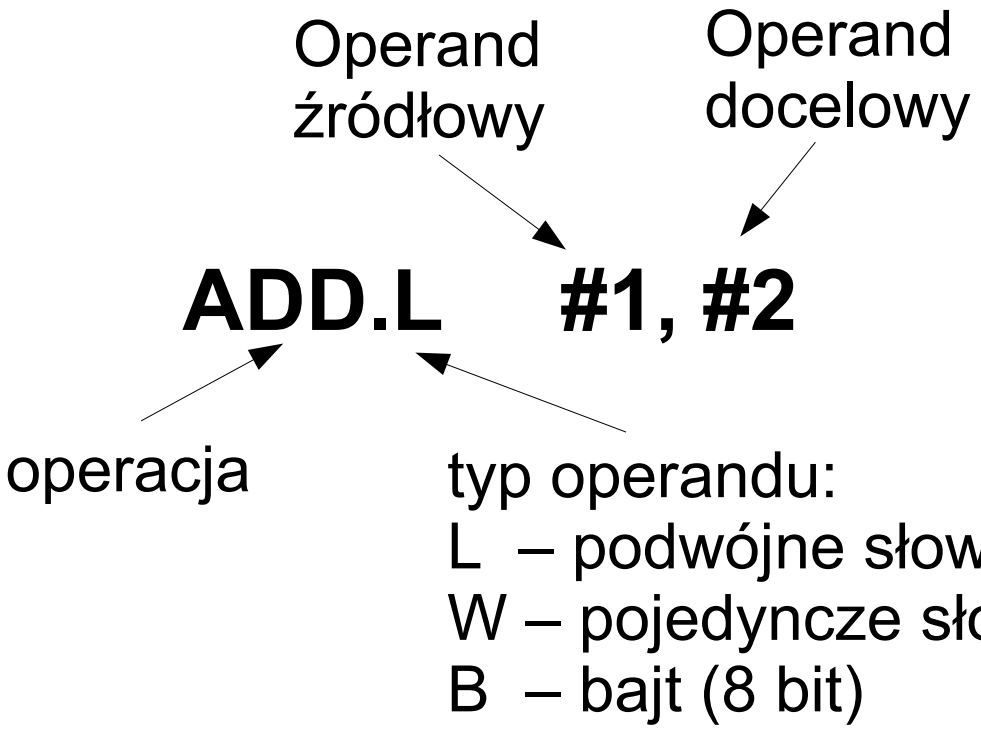
Potokowe wykonanie programu (4)

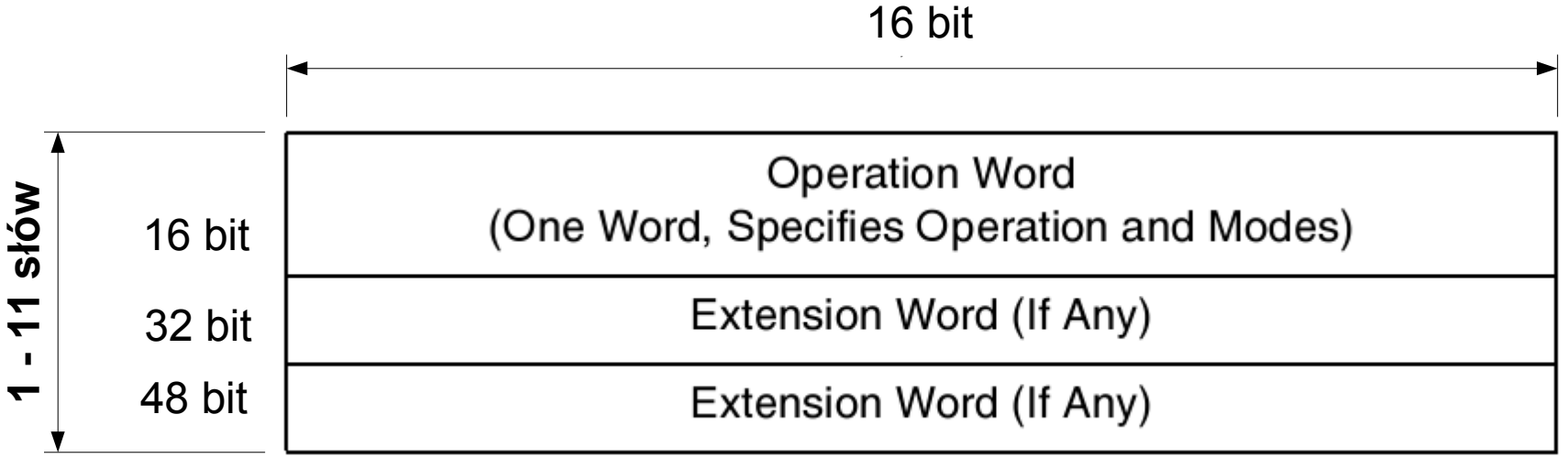
Year	CPU	MHz	Transistors	Process	Pipeline
1978	8086	4.77	29K	3000nm	2
1982	80286	8	134K	1500nm	3
1985	80386	16	275K	1000nm	3
1989	80486	25	1.2M	1000nm	5
1993	Pentium	60	3.1M	800nm	5
1995	Pentium Pro	150	7.5M	600nm	12
1997	Pentium II	233	7.5M	350nm	12
1999	Pentium III	450	9.5M	250nm	12
2000	Willamette P4	1400	42M	180nm	20
2002	Northwood P4	2000	55M	130nm	20
2004	Prescott P4	3400	125M	90nm	31
2001	Itanium	733	25M	180nm	10
2002	Itanium 2	1000	220M	180nm	8



Potok procesora Intel Pentium 4

Format instrukcji asemblera

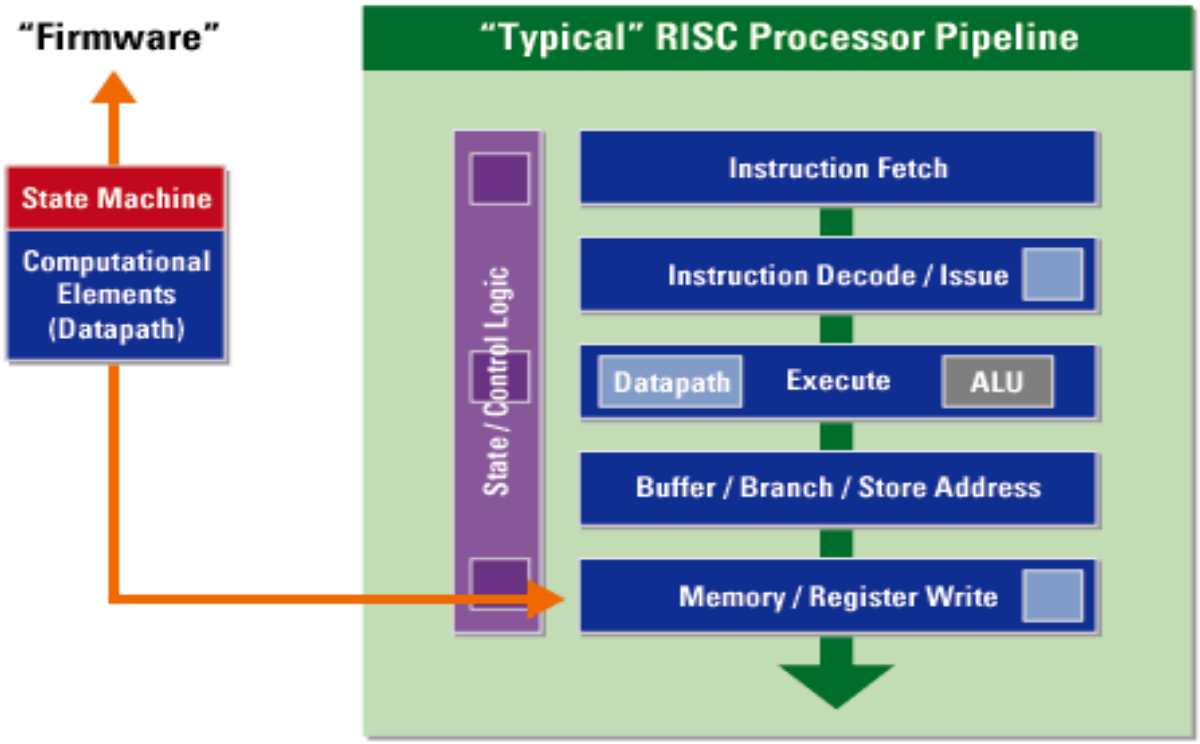




Cykle pracy procesora

ADD.L <ea>y, Dx
ADD.L D0, D1

Język maszynowy:
1 0000 D280



Pracę procesora można podzielić na kilka etapów:

1. Pobranie rozkazu z pamięci programu (Instruction Fetch),
2. Dekodowanie rozkazu (Instruction Decode),
3. Wykonanie rozkazu (Execute command),
4. Pobranie argumentów z pamięci danych (Memory Access),
5. Zapisanie wyniku operacji w pamięci (Write Back).

Wykonanie pojedynczej instrukcji (1)

Etap 1 Instruction fetch cycle (IF):

1. Wysłanie zawartości licznika rozkazów na magistralę adresową,
2. Odczyt instrukcji z pamięci programu,
3. Zwiększenie zawartości licznika rozkazów (PC = PC+4).

Etap 2 Instruction decode/register fetch cycle (ID):

1. Zdekodowanie odczytanego z pamięci rozkazu,
2. Odczytanie operandów z rejestrów (D0 - D7),

Etap 3 Execution/effective address cycle (EX):

ALU operuje na operandach przygotowanych w poprzednim cyklu. Przykładowe operacje:

a) Obliczenie adresu efektywnego danej umieszczonej w pamięci. Rezultat umieszczony w rejestrze wyjściowym ALU.

b) Wykonanie operacji matematyczno-logicznej na rejestrach procesora (rejestry D0-D7, akumulator),

c) Wykonanie operacji z użyciem danej umieszczonej w rejestrach oraz podanej w postaci natychmiastowej,

d) Wykonanie instrukcji skoku - obliczenie przez ALU nowego adresu, z którego zostanie odczytany rozkaz.

Wykonanie pojedynczej instrukcji (2)

Etap 4 Memory access/branch completion cycle (MEM):

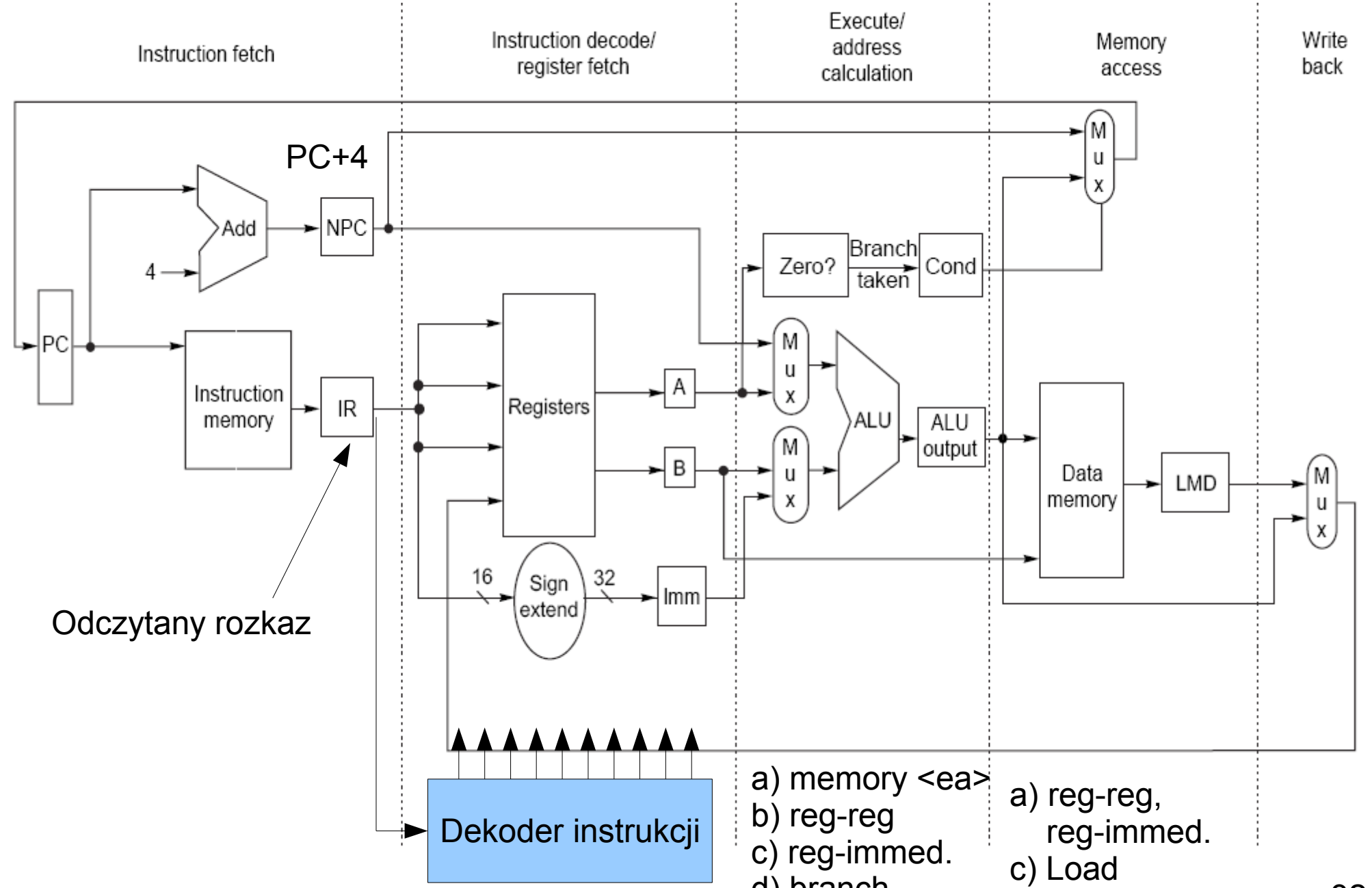
a). Odczytanie danej z pamięci danych,

b). W przypadku realizacji instrukcji skoku, adres obliczony w poprzednim cyklu wpisywany jest do licznika programów.

Etap 5 Write-back cycle(WB):

1. Zapisanie rezultatu operacji w rejestrze lub pamięci.

Datapath


Instruction fetch
Instruction decode/ register fetch
Execute/ address calculation
Memory access
Write back
PC+4
Add
4
NPC
PC
Instruction memory
IR
Registers
A
B
Imm
16
Sign extend
32
Zero?
Branch taken
Cond
M u x
ALU
ALU output
Data memory
LMD
Odczytany rozkaz
Dekoder instrukcji
a) memory <ea>
b) reg-reg
c) reg-immed.
d) branch
a) reg-reg, reg-immed.
c) Load

ADD – operacja dodawania (1)

Operation: Source + Destination → Destination

Assembler Syntax: ADD.L <ea>y,Dx
ADD.L Dy,<ea>x

Condition Codes:

X	N	Z	V	C
*	*	*	*	*

ADD.L D0,D1 => D280

Liczba	Rejestr
000	D0
001	D1
010	D2
011	D3
100	D4
101	D5
110	D6
111	D7

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	0	1	Register				Opmode		Effective Address					
										Mode			Register		

Tryby adresowania argumentu źródłowego <ea>y:

Dy, Ay, (Ay), (Ay)+, -(Ay), (d16,Ay), (d8,Ay,Xi), (xxx).W/L, #<data>, (d16,PC), (d8,PC,Xi), + złożone tryby adresowania dla 68020, 68030, 68040

Tryby adresowania argumentu docelowego <ea>x:

(Ax), (Ax)+, -(Ax), (d16,Ax), (d8,Ax,Xi), (xxx).W/L,
+ złożone tryby adresowania dla 68020, 68030, 68040

ADD – operacja dodawania (2)

Operation: Source + Destination → Destination

Assembler Syntax: ADD.L <ea>y,Dx
ADD.L Dy,<ea>x

Attributes: Size = longword

Description: Adds the source operand to the destination operand using binary addition and stores the result in the destination location. The size of the operation may only be specified as a longword. The mode of the instruction indicates which operand is the source and which is the destination as well as the operand size.

The Dx mode is used when the destination is a data register; the destination <ea>x mode is invalid for a data register.

In addition, ADDA is used when the destination is an address register. ADDI and ADDQ are used when the source is immediate data.

Condition Codes:

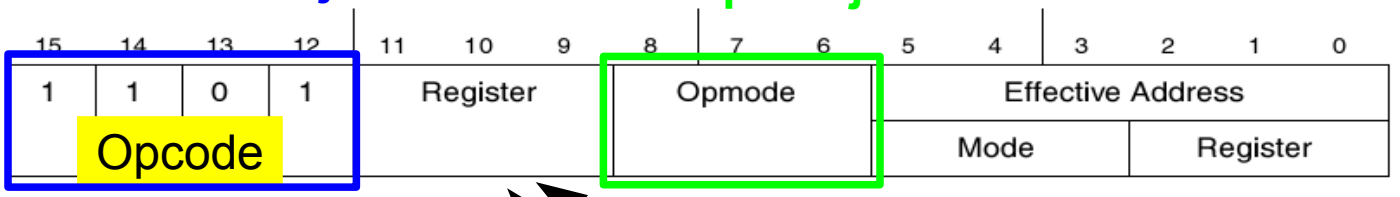
X	N	Z	V	C
*	*	*	*	*

- X Set the same as the carry bit
- N Set if the result is negative; cleared otherwise
- Z Set if the result is zero; cleared otherwise
- V Set if an overflow is generated; cleared otherwise
- C Set if an carry is generated; cleared otherwise

Instruction Format:

kod instrukcji

kod operacji



Instruction Fields:

- Register field—Specifies the data register.
- Opmode field:

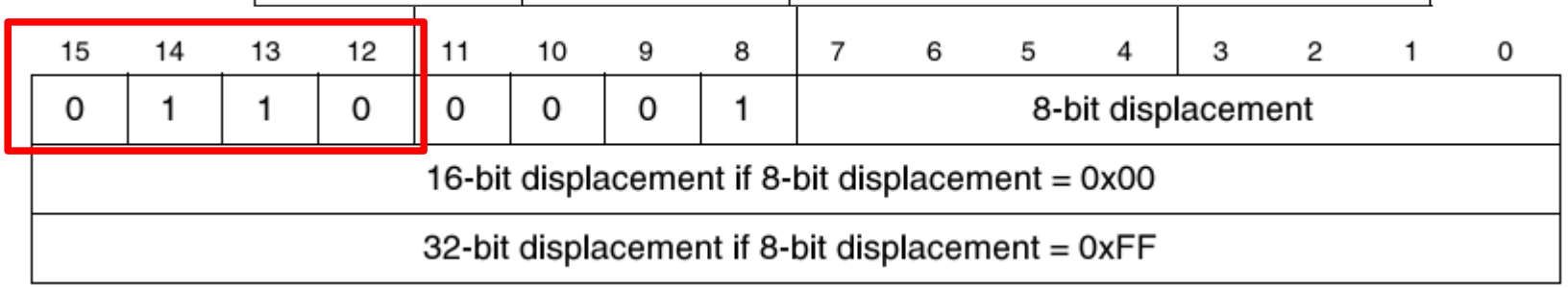
Długość instrukcji: 2-3 słów

Byte	Word	Longword	Operation
—	—	010	<ea>y + Dx → Dx
—	—	110	Dy + <ea>x → <ea>x

← kod operacji

Kod instrukcji (Opcode)

Bits 15–12	Hex	Operation
0000	0	Bit Manipulation/Immediate
0001	1	Move Byte
0010	2	Move Longword
0011	3	Move Word
0100	4	Miscellaneous
0101	5	ADDQ/SUBQ/Scc/TPF
0110	6	Bcc/BSR/BRA
0111	7	MOVEQ/MVS/MVZ
1000	8	OR/DIV
1001	9	SUB/SUBX
1010	A	MAC/EMAC instructions/MOV3Q
1011	B	CMP/EOR
1100	C	AND/MUL
1101	D	ADD/ADDX
1110	E	Shift
1111	F	Floating-Point/Debug/Cache Instructions



Format instrukcji procesora Motorola ColdFire (1)

SINGLE EFFECTIVE ADDRESS OPERATION WORD FORMAT

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
X	X	X	X	X	X	X	X	X	X	EFFECTIVE ADDRESS					
										MODE			REGISTER		

BRIEF EXTENSION WORD FORMAT

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
D/A	REGISTER			W/L	SCALE		0	DISPLACEMENT							

FULL EXTENSION WORD FORMAT

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
D/A	REGISTER			W/L	SCALE		1	BS	IS	BD SIZE		0	I/IS		
BASE DISPLACEMENT (0, 1, OR 2 WORDS)															
OUTER DISPLACEMENT (0, 1, OR 2 WORDS)															

Field	Definition
Instruction	
Mode	Addressing mode (see Table 2-3)
Register	General register number (see Table 2-3)
Extensions	
D/A	Index register type 0 = D*n* 1 = A*n*
W/L	Word/longword index size 0 = Sign-Extended Word 1 = Long Word
Scale	Scale factor 00 = 1 01 = 2 10 = 4 11 = 8 (supported only if FPU is present)

Format instrukcji procesora Motorola ColdFire (2)

Instruction Fields (continued):

- Effective Address field—Determines addressing mode
  - For the source operand <ea>y, use addressing modes listed in the following table:

Addressing Mode	Mode	Register
Dy	000	reg. number:Dy
Ay	001	reg. number:Ay
(Ay)	010	reg. number:Ay
(Ay) +	011	reg. number:Ay
– (Ay)	100	reg. number:Ay
(d_{16},Ay)	101	reg. number:Ay
(d_8,Ay,Xi)	110	reg. number:Ay

Addressing Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
#<data>	111	100
(d_{16},PC)	111	010
(d_8,PC,Xi)	111	011

  - For the destination operand <ea>x, use addressing modes listed in the following table:

Addressing Mode	Mode	Register
Dx	—	—
Ax	—	—
(Ax)	010	reg. number:Ax
(Ax) +	011	reg. number:Ax
– (Ax)	100	reg. number:Ax
(d_{16},Ax)	101	reg. number:Ax
(d_8,Ax,Xi)	110	reg. number:Ax

Addressing Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
#<data>	—	—
(d_{16},PC)	—	—
(d_8,PC,Xi)	—	—

Przykład instrukcji asemblera

ADD.L #$1000, (A0)

Umieść liczbę 0x1000 pod adresem wskazywanym przez rejestr A0

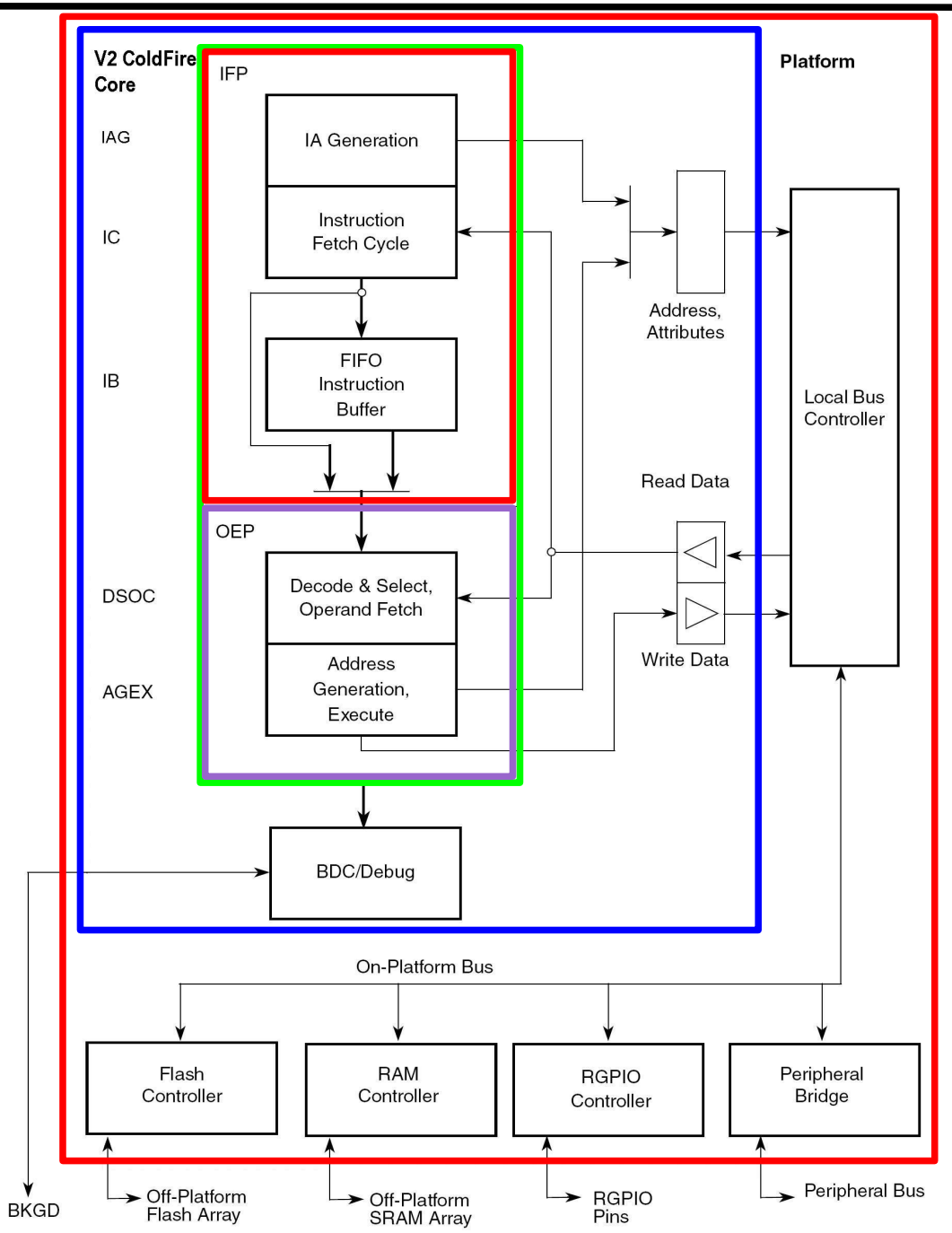
Jakie operacje musi wykonać procesor ?

1. Pobranie kodu instrukcji z pamięci,
2. Zdekodowanie pobranej instrukcji,
3. Pobranie argumentu zapisanego pod adresem wskazywanym przez A0,
4. Wykonanie operacji dodawania 0x1000+(A0),
5. Zapisanie wyniku dodawania w komórce pamięci wskazywanej przez A0.

Potok instrukcji procesora ColdFire

- IFP – jednostka pobierająca instrukcje (Instruction Fetch Pipeline)

- OEP – jednostka przetwarzająca instrukcje (Operand Execution Pipeline)



Przykłady wykonania instrukcji MOVE

- **Operacja typu rejestr-rejestr**

  MOVE.L Ry, Rx **1 cykl**

- **Operacja typu pamięć-rejestr (odczyt)**

  MOVE.L (d16,Ay), Rx **3 cykle**

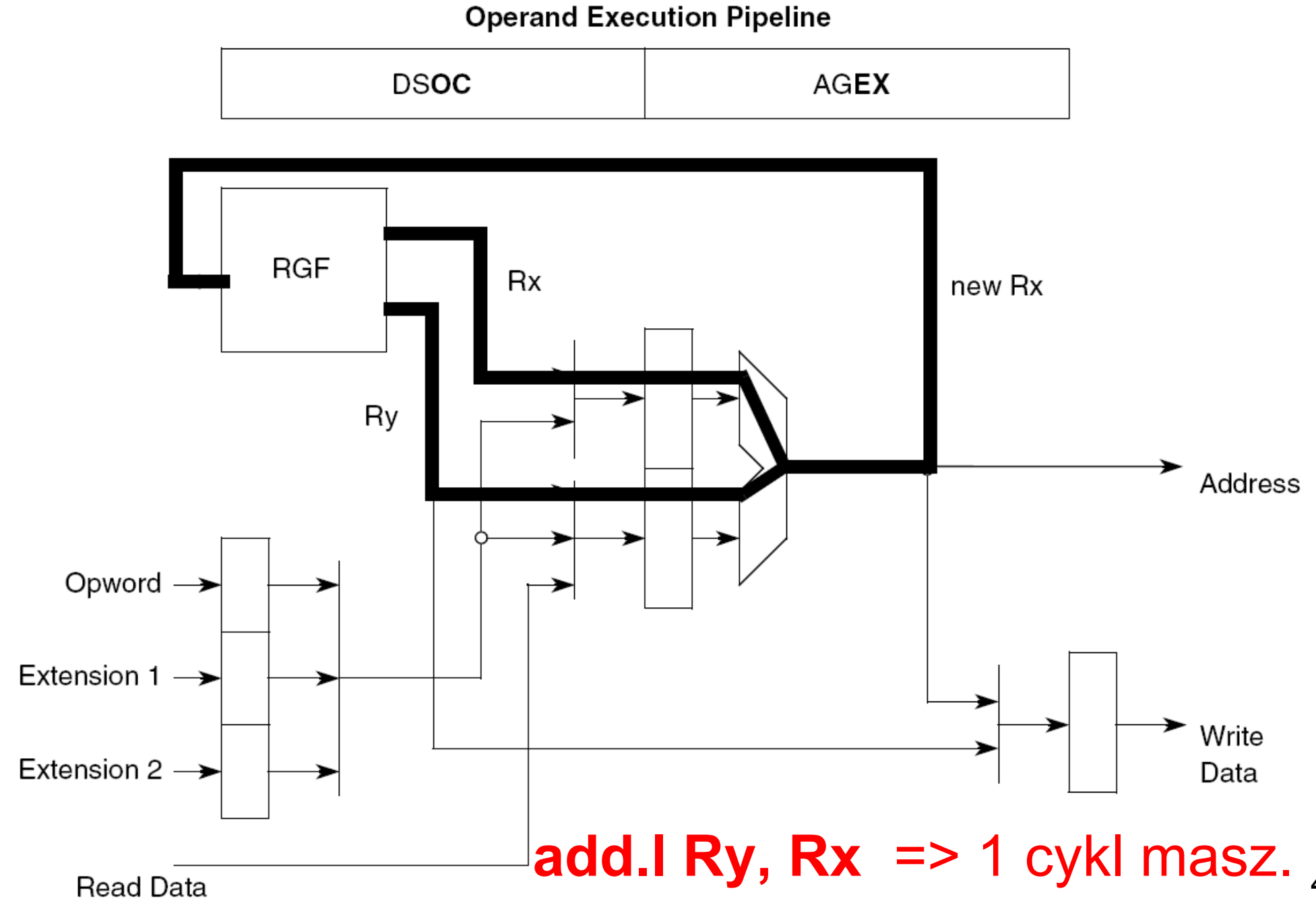
1. Obliczenie adresu efektywnego, rozpoczęcie odczytu
2. Odczytanie danej, wykonanie dodawania
3. Zapis wyniku dodawania do pamięci

- **Operacja typu rejestr-pamięć (zapis)**

  MOVE.L Ry, (d16,Ax) **3 cykle**

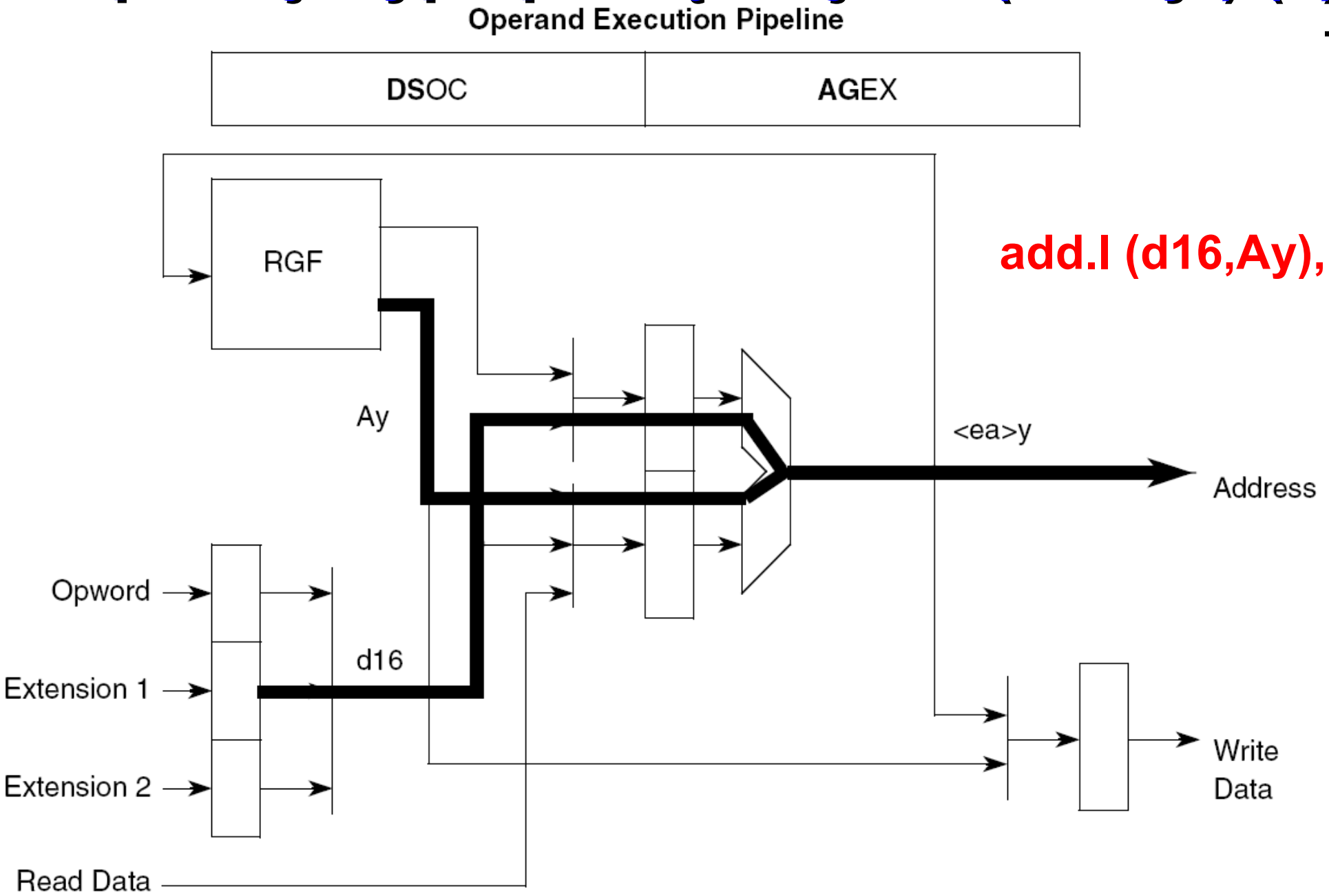
1. Obliczenie adresu efektywnego, rozpoczęcie odczytu
2. Odczytanie danej, wykonanie obliczeń
3. Zapis wyniku dodawania do pamięci

Operacja typu rejestr-rejestr



add.l Ry, Rx => 1 cykl masz.

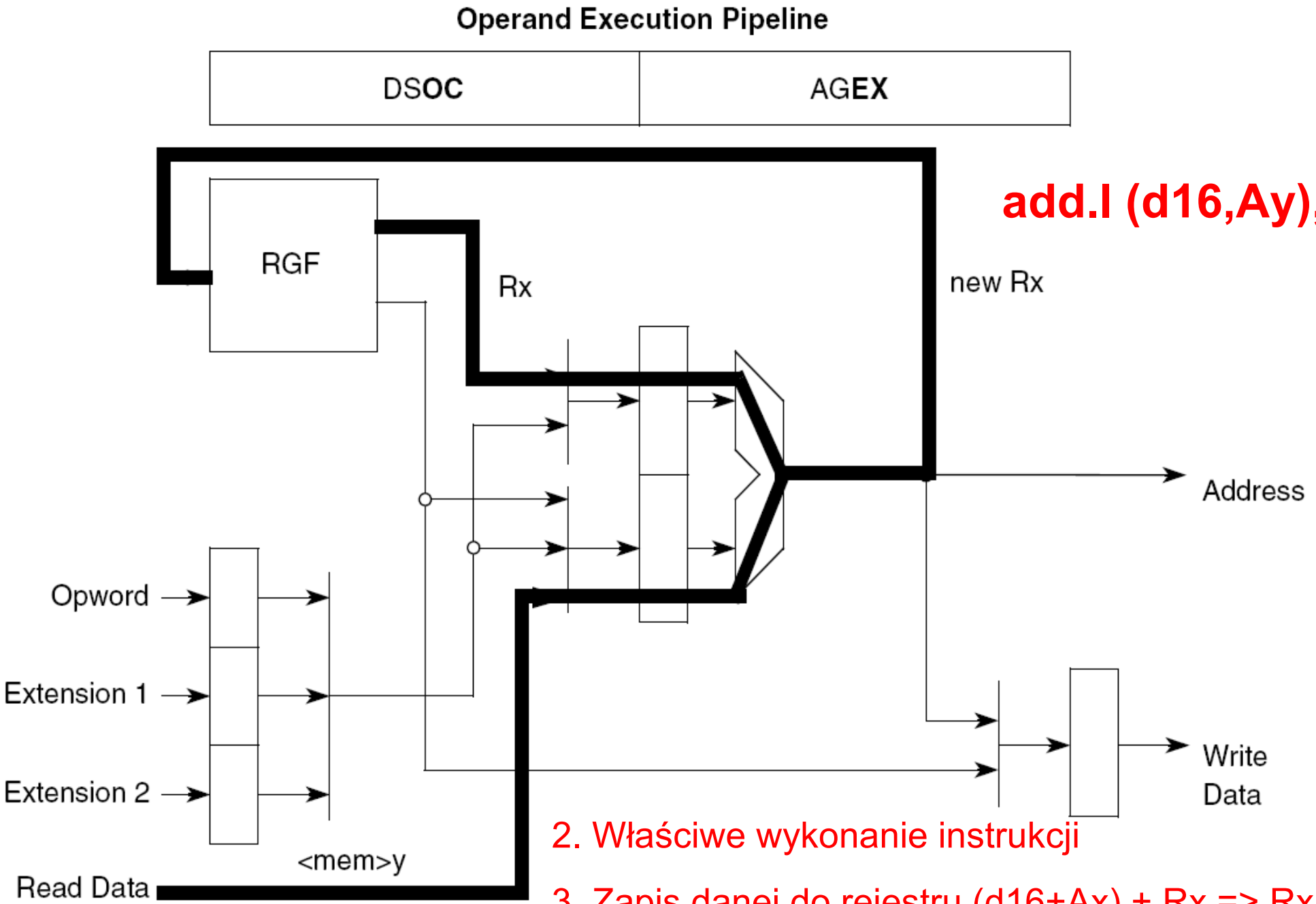
Operacja typu pamięć-rejestr (odczyt) (1)



add.l (d16,Ay), Rx

1. Obliczenie adresu efektywnego d16+Ay

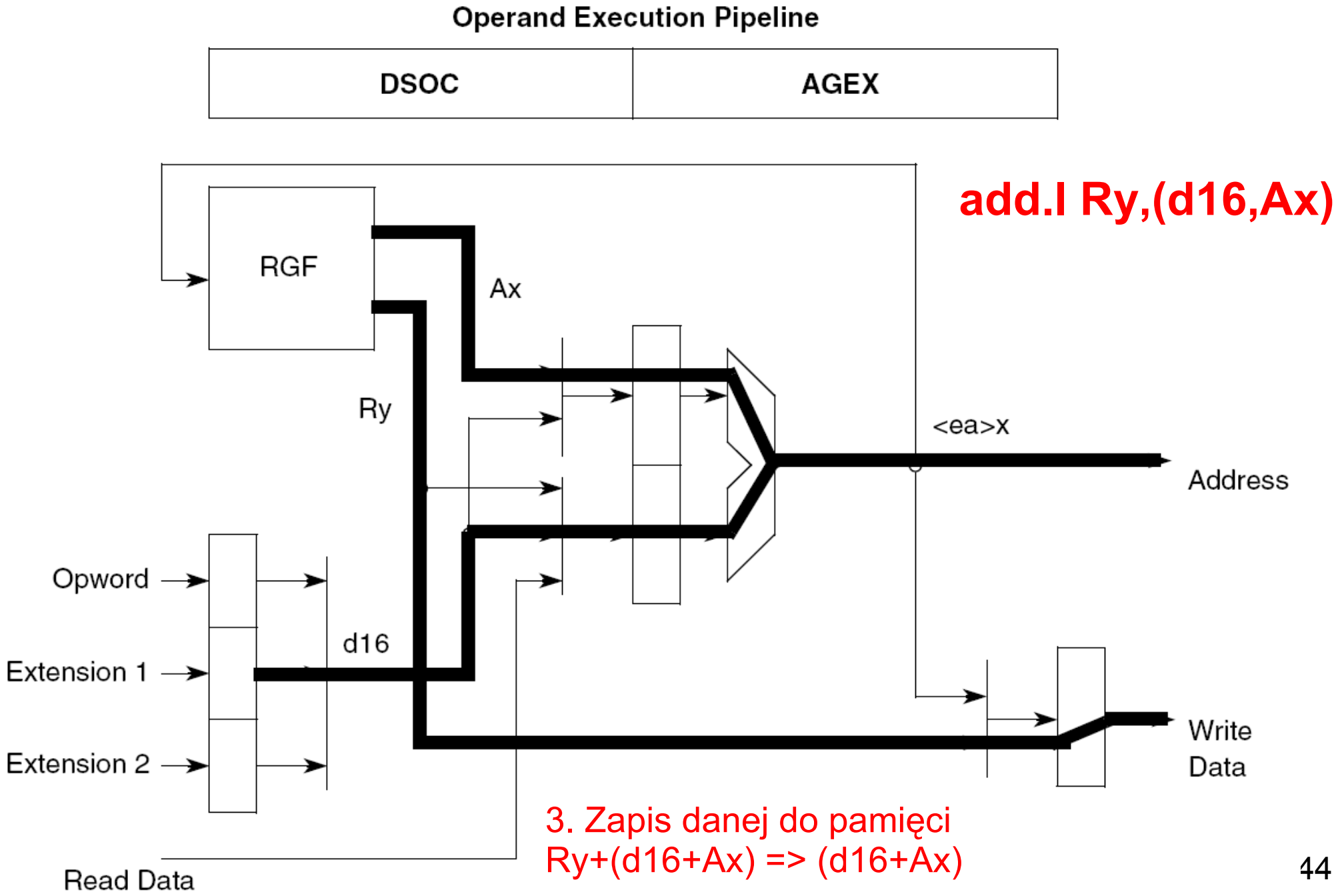
Operacja typu pamięć-rejestr (odczyt) (2)



2. Właściwe wykonanie instrukcji

3. Zapis danej do rejestru (d16+Ax) + Rx => Rx

Operacja typu rejestr-pamięć (zapis)



Czas wykonania instrukcji

Instruction Mnemonic	Operation	Execution Time
<op> Ry,Rx	register-to-register	1
mov.{b,w,l} <mem>y,Rx	8,16,32-bit load	2
mvs.{b,w} <mem>y,Rx	8,16-bit load with sign extension	2
mvz.{b,w} <mem>y,Rx	8,16-bit load with zero fill	2
mov.* Ry,<mem>x	store	1
mov.l <mem>y,<mem>x	memory-to-memory	2
<op> <mem>y,Rx	embedded-load	3
<op> Ry,<mem>x	read-modify-write	3
bsr, jsr <label>	subroutine call	3
rts	subroutine return	5
bra <label>	branch always	2
bcc <label> (forward, not taken) (forward, taken) (backward, not taken) (backward, taken)	conditional branch	 1 3 3 2

Architektura procesora (1)

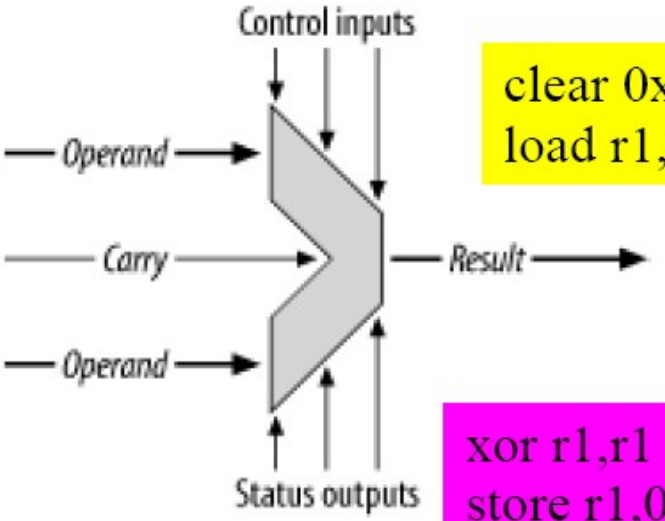
Architektura procesora określa najważniejszych z punktu widzenia budowy i funkcjonalności cechy procesora.

Na architekturę procesora składają się:

- model programowy procesora (ang. Instruction Set Architecture) - zestaw instrukcji procesora oraz inne jego cechy istotne z punktu widzenia programisty, bez względu na ich wewnętrzną realizację; stanowi granicę pomiędzy warstwą sprzętową a programową
- mikroarchitektura procesora (ang. microarchitecture) - wewnętrzna, sprzętowa implementacja danego modelu programowego, określająca sposób wykonywania operacji przez procesor, szczegółową budowę wewnętrzną procesora, itd.

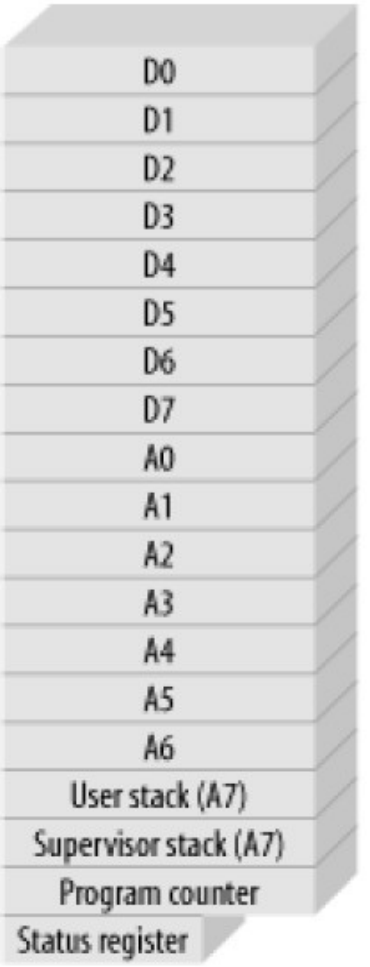
Architektura procesora (2)

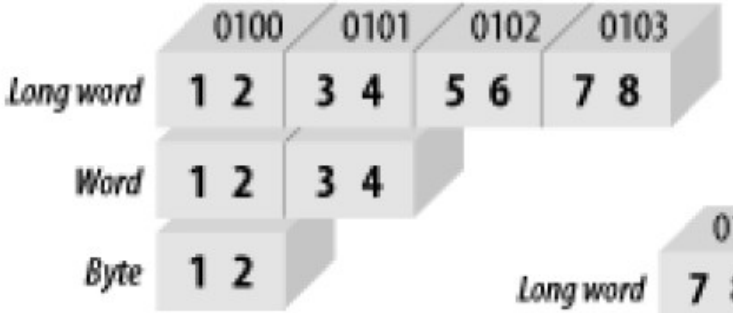
- ALU
  - lista operacji
- zestaw rejestrów
  - A, I, PC, SR, SP,...
- lista rozkazów
  - CISC, RISC
- tryby adresowania
  - R, A, I,...
- przerwania
  - hardware, software
- big/little endian

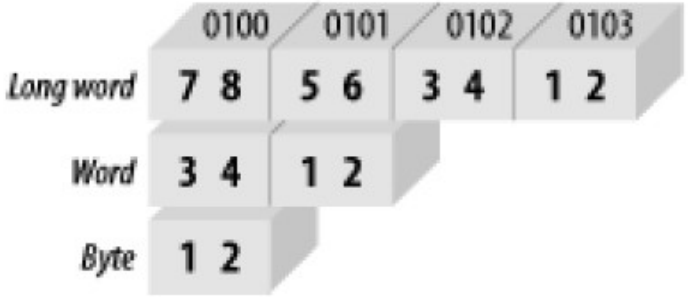



```
clear 0x1000
load r1,#5
```

```
xor r1,r1
store r1,0x1000
add r1,#5
```



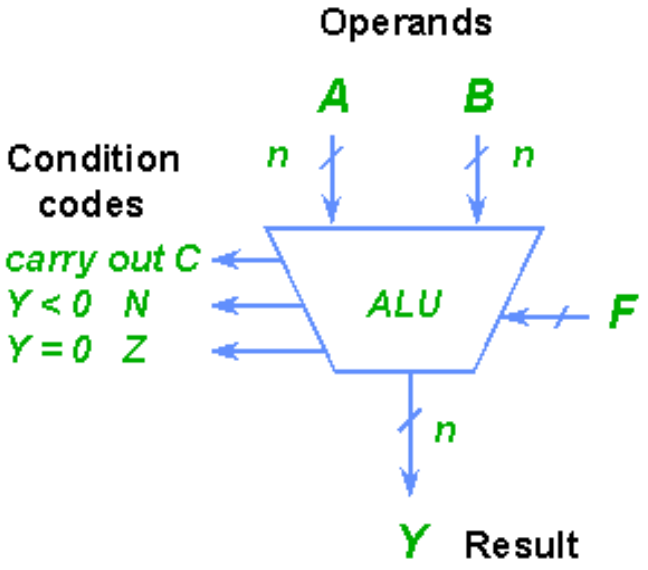




Jednostka arytmetyczno-logiczna

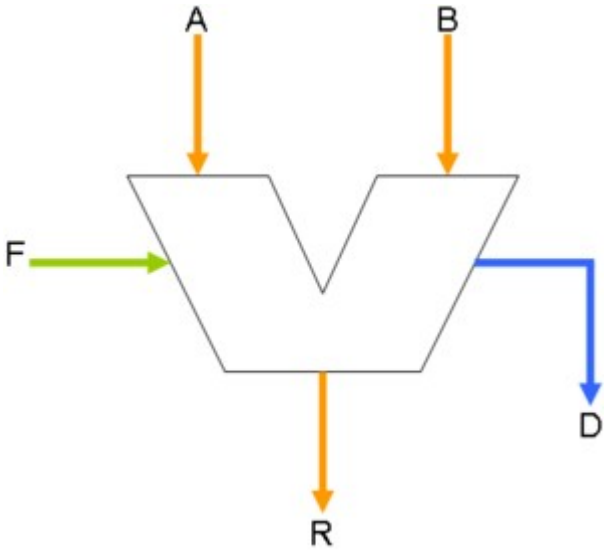
Jednostka arytmetyczno-logiczna wykorzystywana jest do wykonywania:

- operacji logicznych AND, OR, NOT, XOR,
- dodawania,
- odejmowania, negacja liczby, dodawanie z przeniesieniem, zwiększanie/zmniejszanie o 1,
- przesunięcia bitowe o stałą liczbę bitów,
- mnożenia i/lub dzielenia (dzielenie modulo).



F	Y
000	A + B
001	A - B
010	A - 1
011	A and B
100	A or B
101	A * B

Dwu-bitowa jednostka ALU

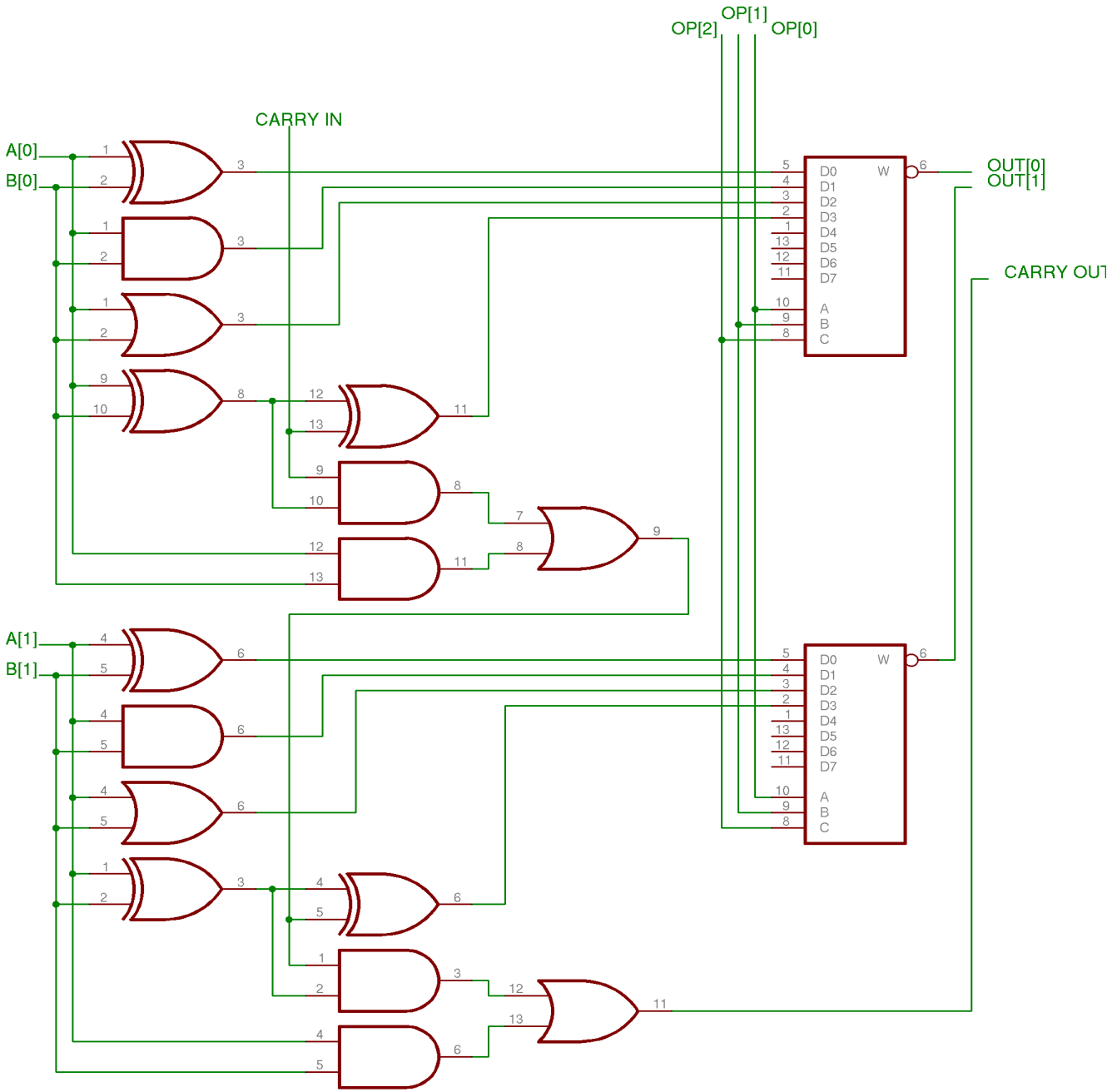


Realizowane operacje:

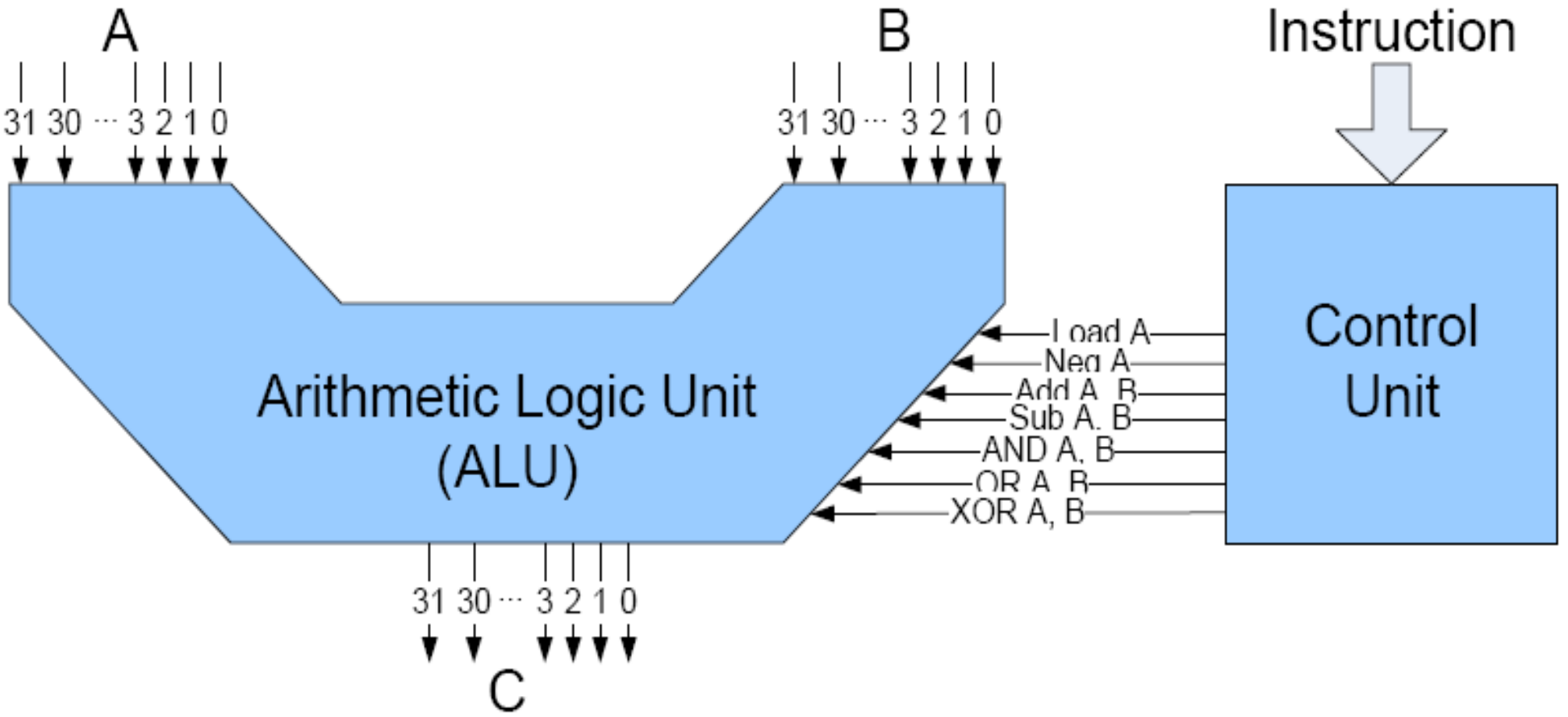
- OP = 000 → XOR
- OP = 001 → AND
- OP = 010 → OR
- OP = 011 → Addition

Inne możliwe operacje:

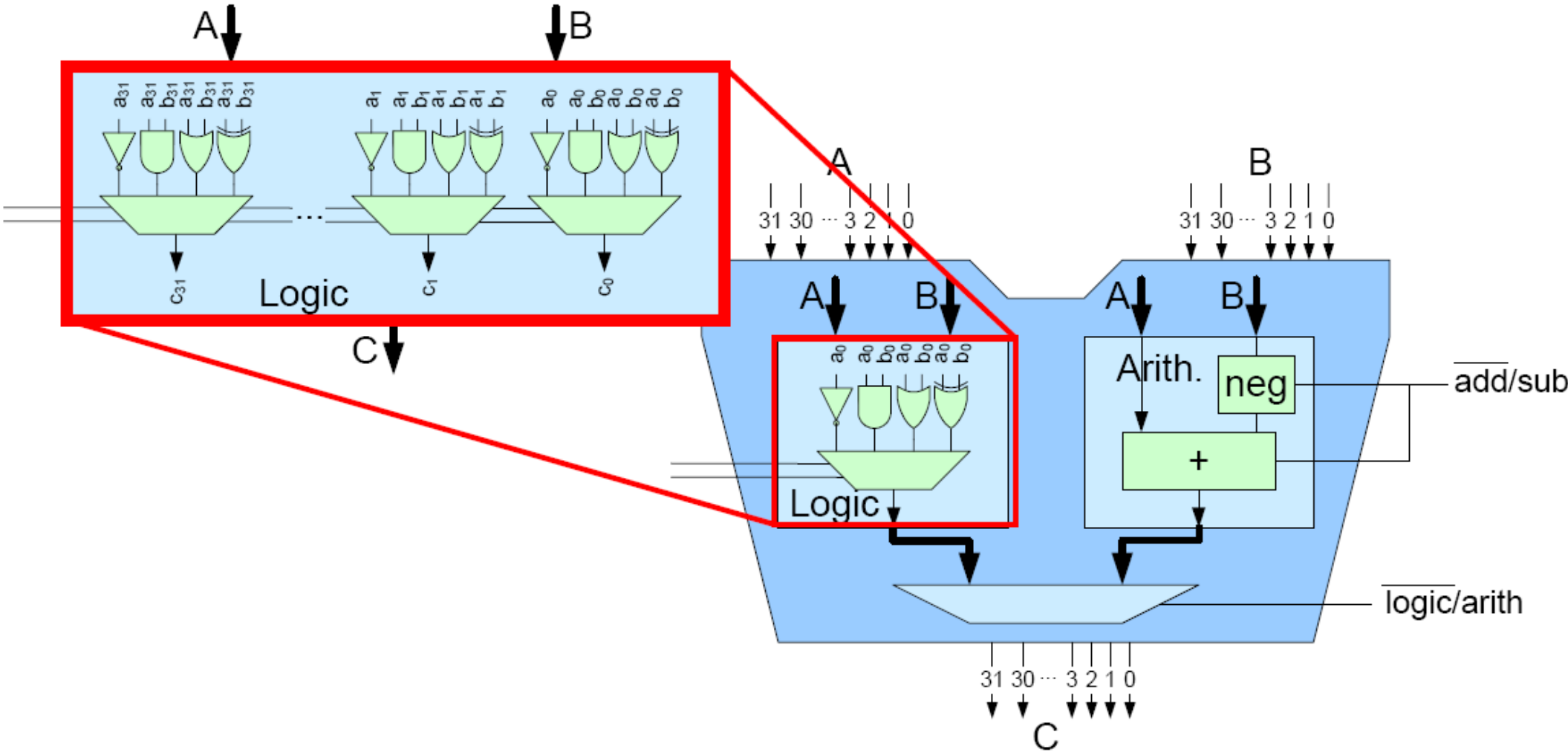
- subtraction,
- multiplication,
- division,
- NOT A,
- NOT B



ALU 32-bit (1)


A
31 30 ··· 3 2 1 0
B
31 30 ··· 3 2 1 0
Instruction
Arithmetic Logic Unit
(ALU)
Load A
Neg A
Add A, B
Sub A, B
AND A, B
OR A, B
XOR A, B
Control
Unit
31 30 ··· 3 2 1 0
C

ALU 32-bit (2)



Funkcje realizowane przez ALU

Operation select						
S_3	S_2	S_1	S_0	C_{in}	Operation	Function
0	0	0	0	0	$F = A$	Transfer A
0	0	0	0	1	$F = A + 1$	Increment A
0	0	0	1	0	$F = A + B$	Addition
0	0	0	1	1	$F = A + B + 1$	Add with carry
0	0	1	0	0	$F = A + \overline{B}$	Subtract with borrow
0	0	1	0	1	$F = A + \overline{B} + 1$	Subtraction
0	0	1	1	0	$F = A - 1$	Decrement A
0	0	1	1	1	$F = A$	Transfer A
0	1	0	0	×	$F = A \wedge B$	AND
0	1	0	1	×	$F = A \vee B$	OR
0	1	1	0	×	$F = A \oplus B$	XOR
0	1	1	1	×	$F = \overline{A}$	Complement A
1	0	×	×	×	$F = \text{shr } A$	Shift right A into F
1	1	×	×	×	$F = \text{shl } A$	Shift left A into F

ALU opisane w VHDL-u

LC-2 ALU VHDL Code

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;
use IEEE.std_logic_arith.all;
--LC-2 ALU-------------------------------------
entity LC2_ALU is
  port( A: in std_logic_vector (15 downto 0);
    B: in std_logic_vector (15 downto 0);
    S: in std_logic_vector (1 downto 0);
    O: out std_logic_vector (15 downto 0));
end LC2_ALU;

architecture bhv of LC2_ALU is
begin
  process(A, B, S)
begin

  case S is
    when "00" => O <= A+B;
    when "01" => O <= A and B;
    when "10" => O <= A;
    when "11" => O <= not A;
    when others => null;
  end case;

end process;
end bhv;
```

Wykonywanie specjalizowanych obliczeń

1. Projekt skomplikowanej jednostki ALU, która potrafi obliczyć pierwiastek kwadratowy w 1 cyklu zegarowym,
2. Projekt sekwencyjnej jednostki ALU, która potrafi obliczyć pierwiastek kwadratowy w kilku cyklach zegarowych,
3. Projekt prostej jednostki ALU, która współpracuje z drogim, skomplikowanym koprocesorem,
4. Programowa emulacja pomocniczej jednostki obliczeniowej (koprocesora). Oprogramowanie sprawdza, czy w systemie jest obecny koprocesor, jeżeli nie pierwiastek kwadratowy jest obliczany programowo,
5. Brak wyspecjalizowanej jednostki obliczeniowe zdolnej do wyliczenia pierwiastka. Obliczenia wykonywane są programowo – wykorzystanie biblioteki.

Rejestry procesora

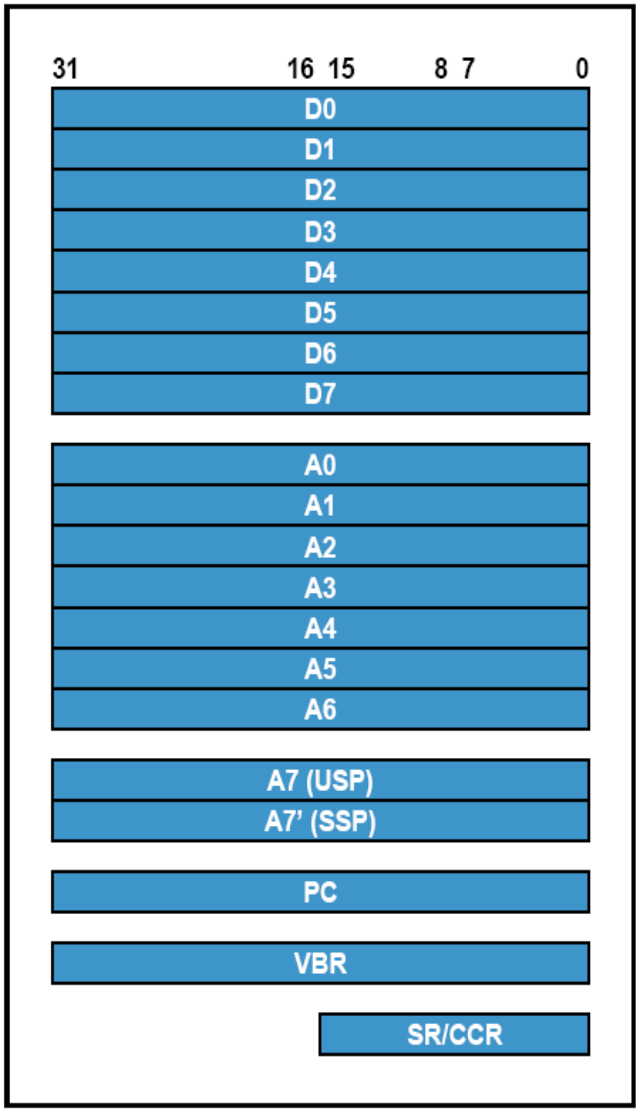
Rejestry procesora stanowią komórki wewnętrznej pamięci procesora o niewielkich rozmiarach (najczęściej 4/8/16/32/64/128 bitów) służące do przechowywania tymczasowych wyników obliczeń, adresów danych w pamięci operacyjnej, konfiguracji, itd.

Cechy rejestrów procesora:

- stanowią najwyższy szczebel w hierarchii pamięci (najszybszy rodzaj pamięci komputera),
- Realizowane w postaci przerzutników dwustanowych,
- Liczba rejestrów zależy od zastosowania procesora.

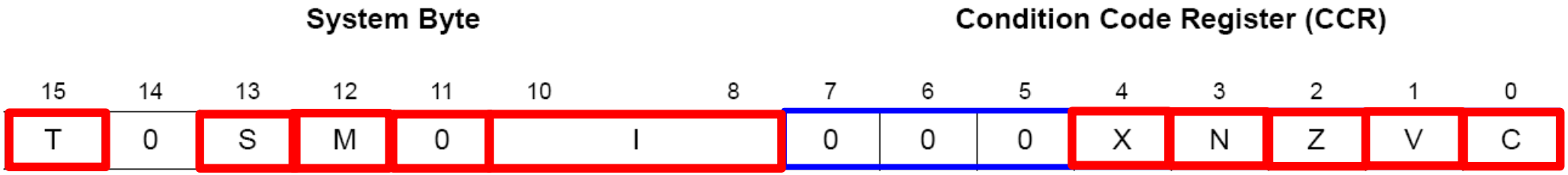
Rejestry dzielimy na:

- rejestry danych - do przechowywania danych np. argumentów i wyników obliczeń,
- rejestry adresowe - do przechowywania adresów (wskaźnik stosu, wskaźnik programu, rejestry segmentowe),
- rejestry ogólnego zastosowania (ang. general purpose), przechowują zarówno dane, jak i adresy,
- rejestry zmiennoprzecinkowe - do przechowywania i wykonywania obliczeń na liczbach zmiennoprzecinkowych (koprocesor FPU),



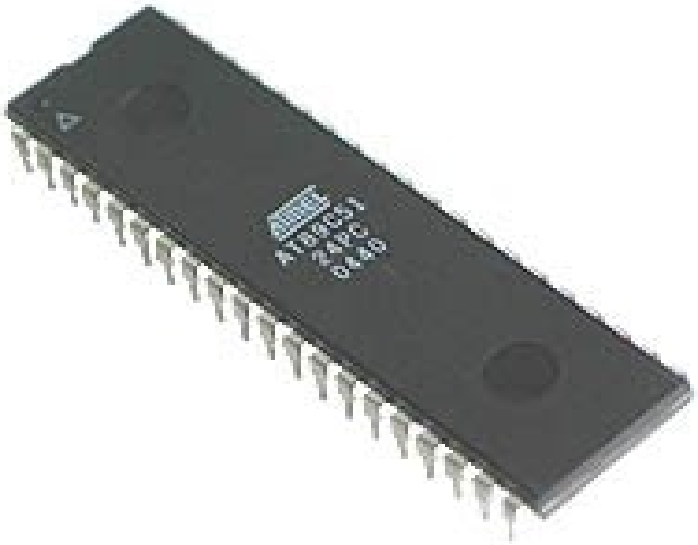
Rejestry procesora z rodziny Motorola 68k

Rejestr statusowy (Status Register)

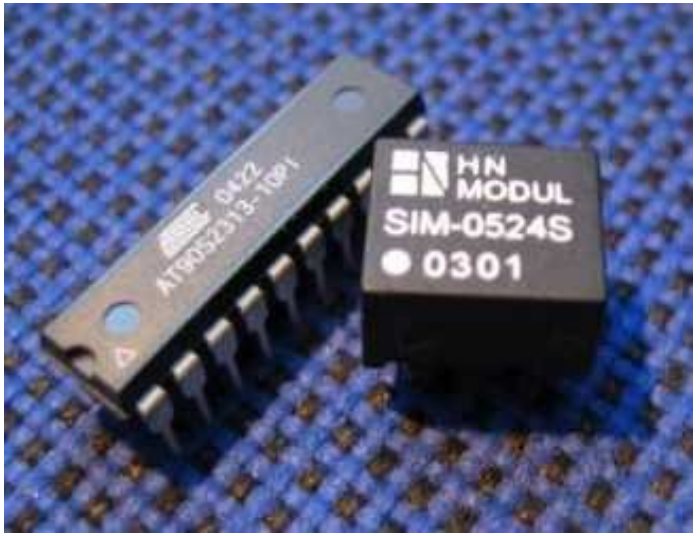


Bits	Name	Description
15	T	Trace enable. When set, the processor performs a trace exception after every instruction.
13	S	Supervisor/user state. Denotes whether the processor is in supervisor mode (S = 1) or user mode (S = 0).
12	M	Master/interrupt state. This bit is cleared by an interrupt exception, and can be set by software during execution of the RTE or move to SR instructions.
10–8	I	Interrupt level mask. Defines the current interrupt level. Interrupt requests are inhibited for all priority levels less than or equal to the current level, except the edge-sensitive level 7 request, which cannot be masked.
4	X	Extend condition code bit.
3	N	Negative condition code bit. Set if the most significant bit of the result is set; otherwise cleared.
2	Z	Zero condition code bit. Set if the result equals zero; otherwise cleared.
1	V	Overflow condition code bit. Set if an arithmetic overflow occurs implying that result cannot be represented in the operand size; otherwise cleared.
0	C	Carry condition code bit. Set if a carry out of the operand msb occurs for an addition, or if a borrow occurs in a subtraction; otherwise cleared Set to the value of the C bit for arithmetic operations; otherwise not affected.

Mikrokontrolery jednoukładowe



Atmel AT89C51, DIP 40

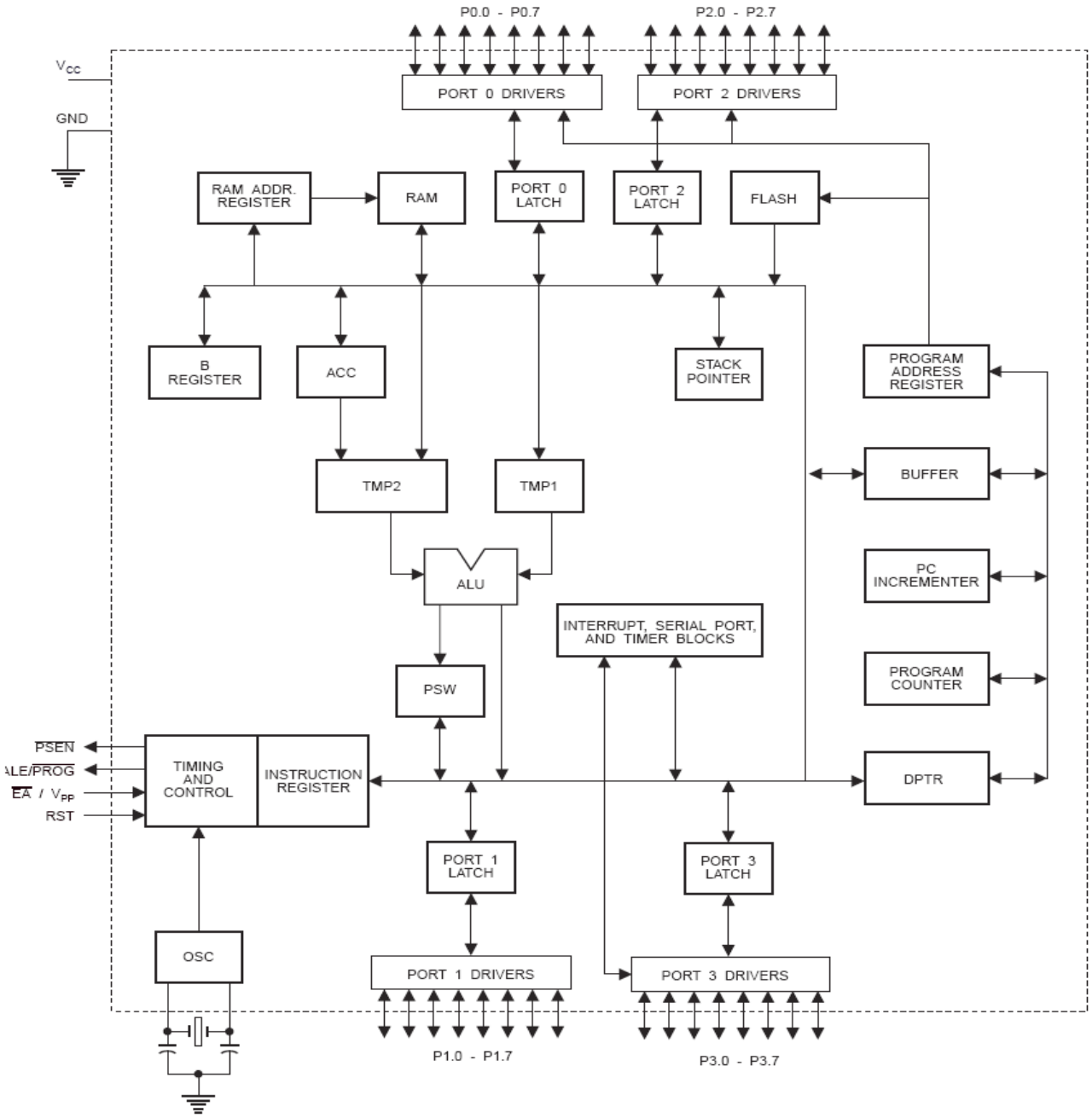


Atmel AT90S2313,
DIP 20

Mikrokontroler AT89C51

Cechy mikrokontrolera

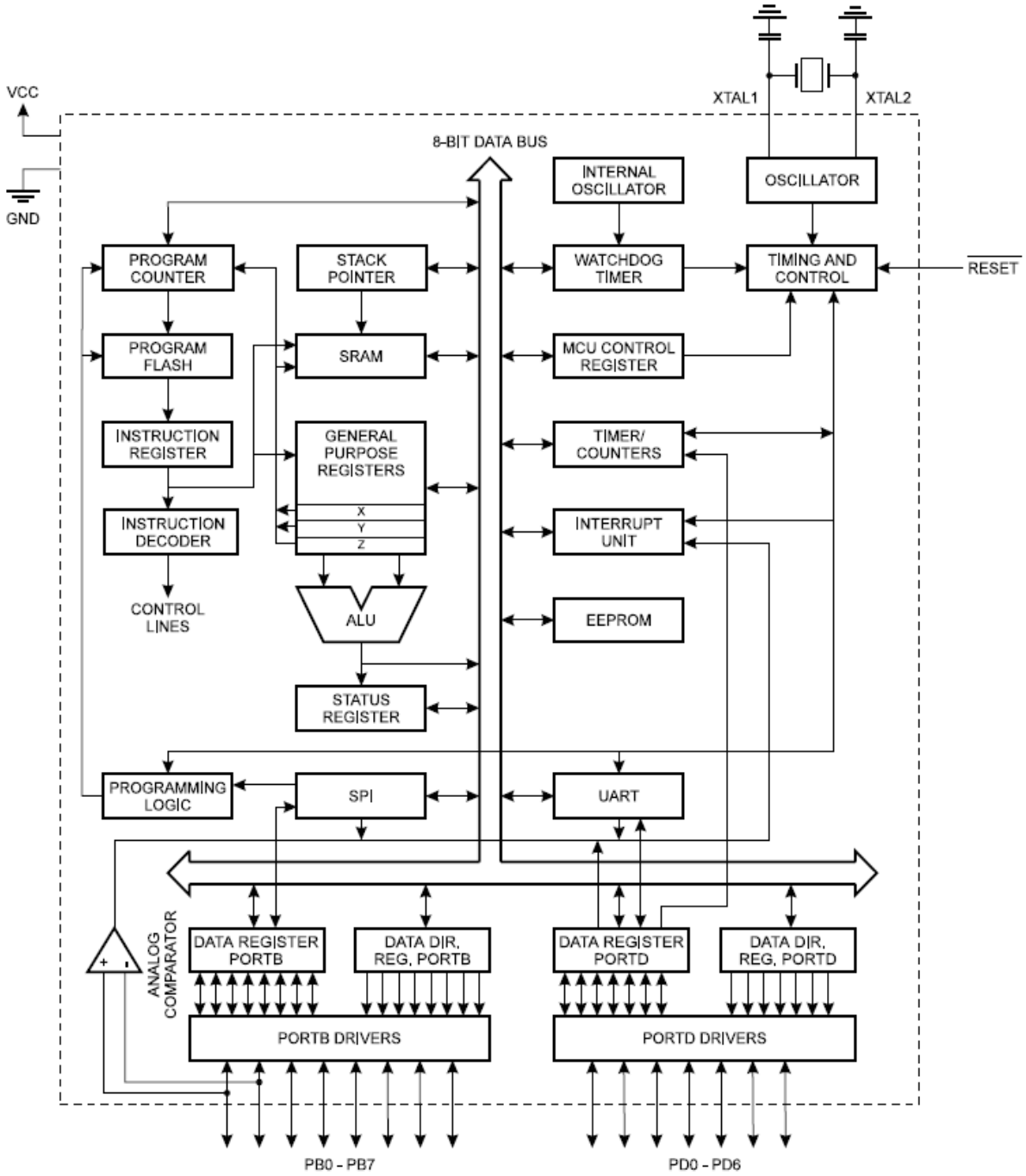
- Zgodny z architekturą '51
- Wewnętrzna pamięć FLASH 4 kB
- Wewnętrzna pamięć SRAM 128 B
- Częstotliwość taktowania 0 Hz do 24 Mhz
- UART zgodny z RS 232
- Programowalne porty I/O, 32 linie
- Dwa 16 bitowe liczniki/timery
- Sześć źrodeł przerwań



Mikrokontroler AT90S2313

Cechy mikrokontrolera

- Zgodny z architekturą AVR/RISC
- Wewnętrzna pamięć
  FLASH 2 kB / 128 B EEPROM
- Wewnętrzna pamięć
  SRAM 128 B
- Częstotliwość taktowania
  0 Hz do 10 Mhz
- UART zgodny z RS 232, SPI, Watchdog, komparator
- Programowalne porty I/O, 15 linii (AT90S8535 32-linie)
- Dwa 8/16 bitowe liczniki/timery
- Jedenaście źrodeł przerwań



Porównanie mikrokontrolerów

Zadanie:

Zrealizować operację sumowania dwóch 16-bit liczb

R1/R0 + R3/R2 => R1/R0

```
;Program procesora rodziny '51
MOV   A, R0
ADD   A, R2
MOV   R0, A
MOV   A, R1
ADDC  A, R3
MOV   R1, A
```

1 instrukcja = 12 clk

f = 12 Mhz => 6 us

```
;Program procesora rodziny AVR
ADD   R0, R2
ADDC  R1, R3
```

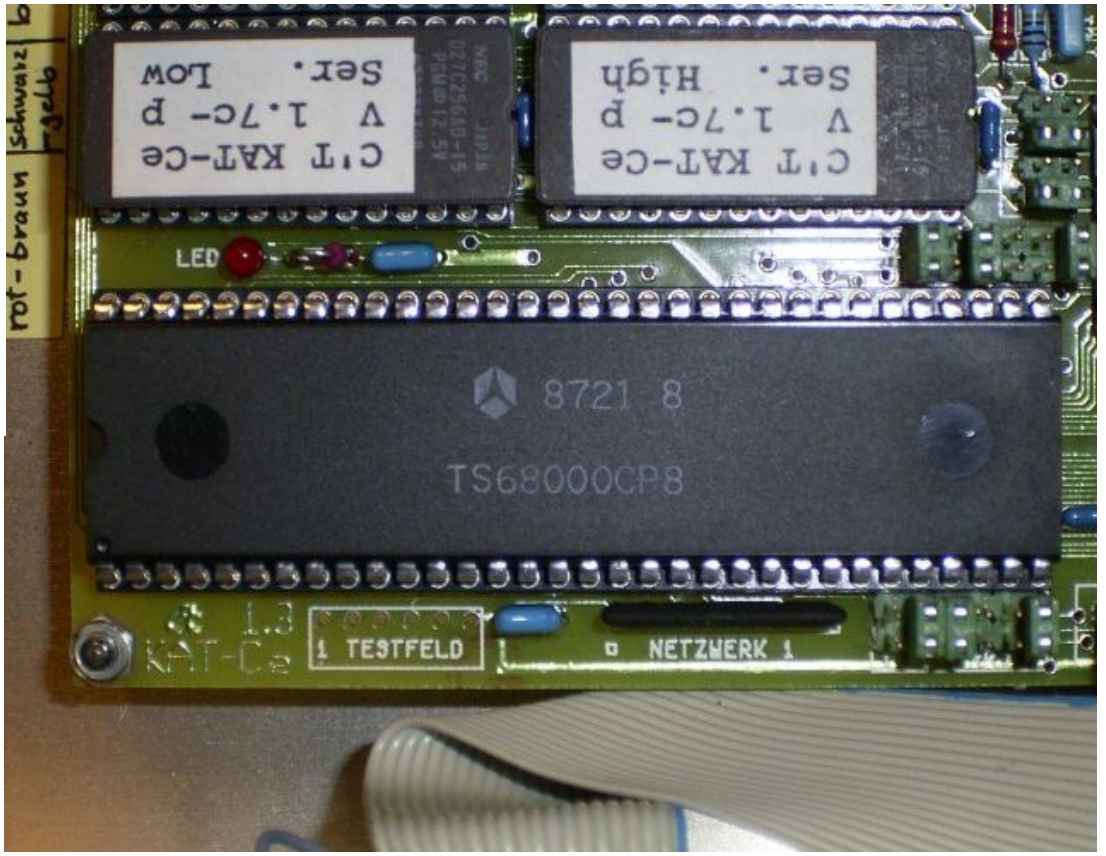
ADDC A, R3 A <= A + R3 + C

1 instrukcja = 1 clk

f = 12 Mhz => 0.16 us

Motorola, czy Freescale

Mikrokontroler Freescale,
MCF520x, BGA 256



Procesor Motorola, 68k
DIP 64

Systemy operacyjne dla urządzeń wbudowanych

Definicje podstawowe

- ***Procesor***

  urządzenie cyfrowe, sekwencyjne zdolne pobierać dane z pamięci, interpretować je i wykonywać jako rozkazy

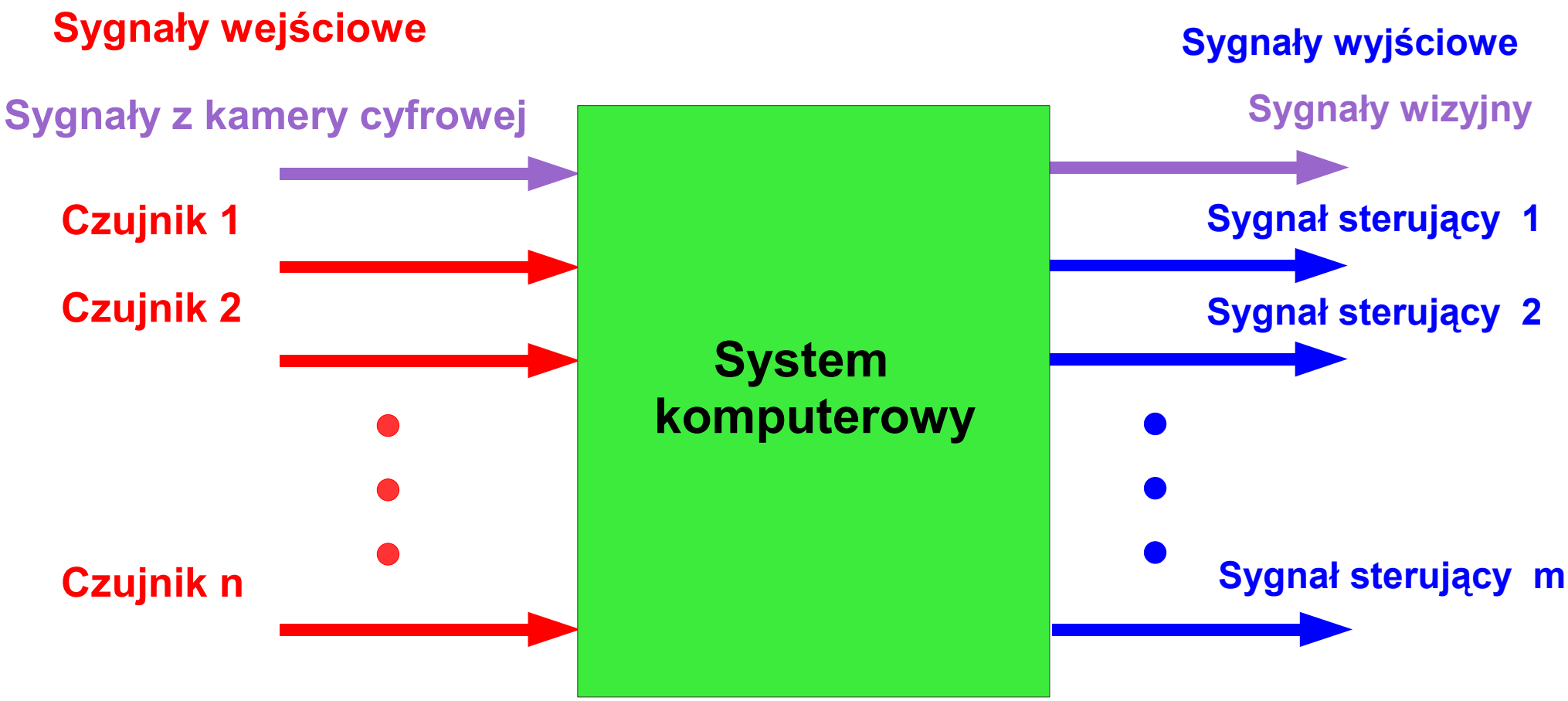
- ***Mikroprocesor***

  układ cyfrowy wykonany jako pojedynczy układ scalony o wielkim stopniu integracji zdolny do wykonywania operacji cyfrowych według dostarczonych mu informacji
  np.: x86, Z80, itp...

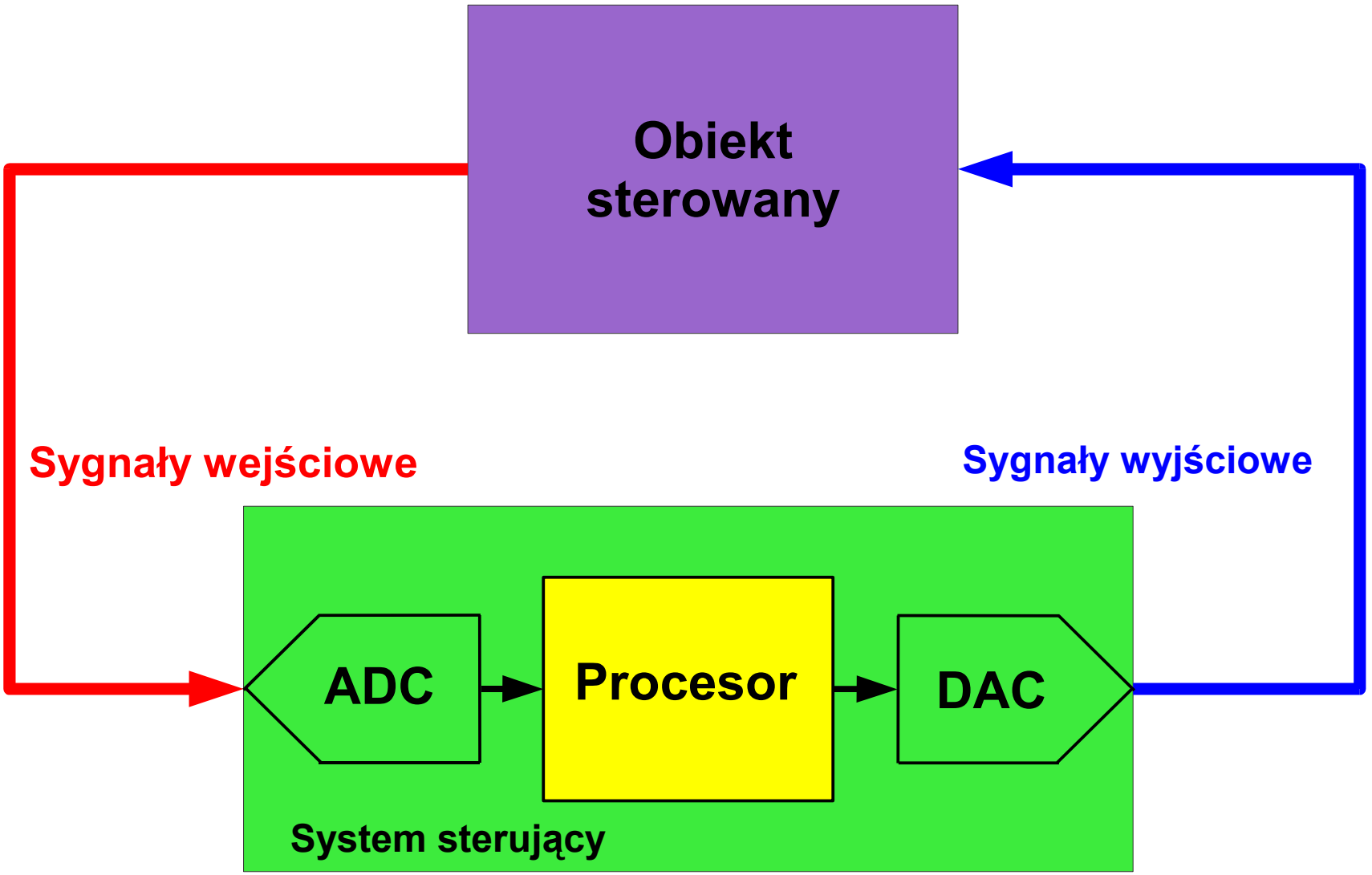
- ***Mikrokontroler***

  komputer wykonany w jednym układzie scalonym, używany do sterowania urządzeniami elektronicznymi. Oprócz jednostki centralnej CPU posiada zintegrowane urządzenia peryferyjne (pamięci, układy wejścia–wyjścia, zegary, itp...). Przykłady mikrokontrolerów: Atmel AVR, MCF5282, itp...

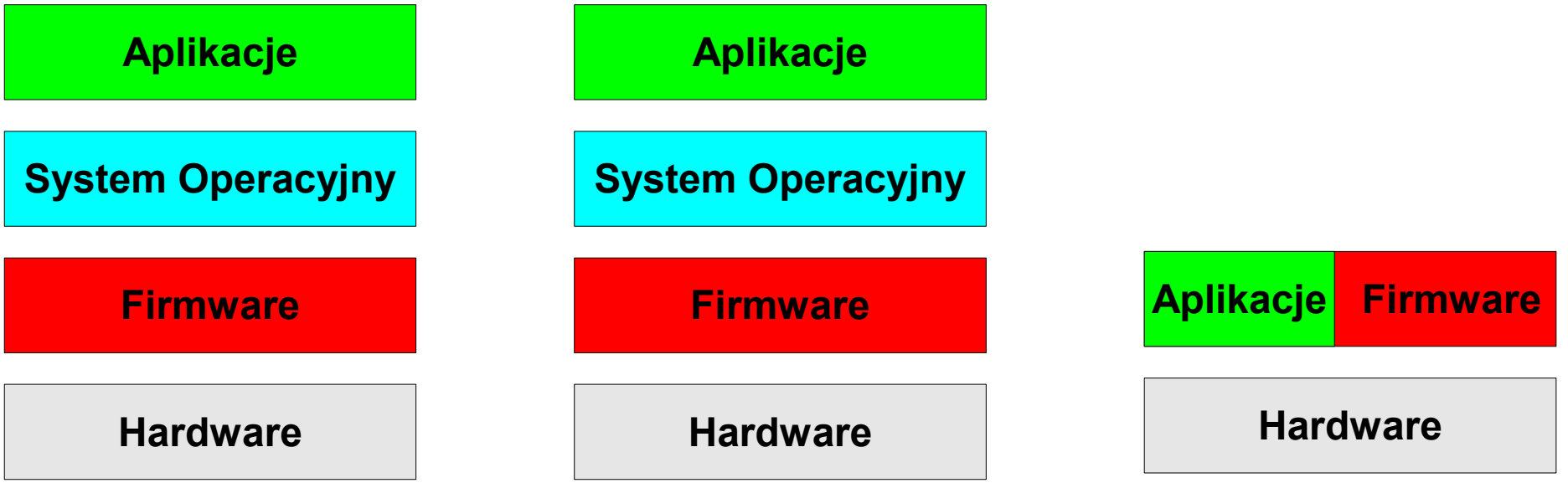
System mikroprocesorowy



System sterujący



Oprogramowanie mikrokomputerów



Komputery osobiste, uniwersalne:

- języki wysokiego poziomu (Asembler, C/C++, Pascal, Java, Basic...)

Komputery wbudowane, sterowniki:

- język niskiego poziomu Asembler,
- języki wysokiego poziomu (C/C++, Basic, Ada).

Język C dla urządzeń wbudowanych

- Kompilator GCC
  - Zapewnia wsparcie dla wielu rodzin procesorów

    (avr-gcc, m68k-elf-gcc, powerpc-gcc)
  - Projekt na licencji GNU GPL
  - Umożliwia stosowanie standardowych bibliotek niezależnie od architektury docelowej
  - Zapewnia możliwość dodatkowego konfigurowania programu w zależności od architektury docelowej
    - __attribute__ ((signal)
    - __attribute__ ((naked)
    - __attribute__ ((section (".noinit")))
    - __attribute__ ((section (".eeprom")));

Język C a systemy wbudowane

- Definicje
  - Build – typ mikroprocesora na którym kompilujemy gcc
  - Host – typ mikroprocesora dla którego kompilujemy gcc
  - Target – typ mikroprocesora na którym będzie uruchamiana aplikacja kompilowana
- Rodzaje kompilacji gcc i oprogramowania

System operacyjny

System Operacyjny OS (Operating System) - oprogramowanie, które zarządza sprzętem oraz aplikacjami komputera (procesora). Podstawą wszystkich systemów operacyjnych jest wykonywanie podstawowych zadań takich jak: zarządzanie pamięcią, przydział czasu procesora, obsługa urządzeń, ustalanie połączeń sieciowych oraz zarządzanie plikami.

Możemy wyróżnić trzy główne elementy systemu operacyjnego:

- jądro systemu wykonujące ww. zadania,
- powłoka - specjalny program komunikujący użytkownika z systemem operacyjnym,
- system plików - sposób zapisu struktury danych na nośniku.

System operacyjny czasu rzeczywistego

- Systemem czasu rzeczywistego określa się taki system, którego wynik przetwarzania zależy nie tylko od jego logicznej poprawności, ale również od czasu, w jakim został osiągnięty
- System czasu rzeczywistego odpowiada w sposób przewidywalny na bodźce zewnętrzne napływające w sposób nieprzewidywalny
- Poprawność pracy systemu czasu rzeczywistego zależy zarówno od wygenerowanych sygnałów wyjściowych jak i spełnionych zależności czasowych
- Z pojęciem „czasu rzeczywistego” wiąże się wiele nadużyć. Terminu tego używa się potocznie dla określenia obliczeń wykonywanych bardzo szybko, co nie zawsze jest prawdą.

System czasu rzeczywistego

- **Czas reakcji systemu** – przedział czasu potrzebny systemowi operacyjnemu na wypracowanie decyzji (sygnału wyjściowego) w odpowiedzi na zewnętrzny bodziec (sygnał wejściowy)
- **Czas reakcji systemu** może wahać się w granicach od ułamków sekund (np.: system akwizycji danych z kamery) do kilkudziesięciu godzin (np.: system sterowania poziomem wody w zbiorniku retencyjnym)
- Charakterystyka różnych zadań aplikacji musi być znana *a priori*
- Systemy czasu rzeczywistego, w szczególności systemy dynamiczne, muszą być szybkie, przewidywalne, niezawodne i adoptowalne

Podział systemów czasu rzeczywistego

- **Systemy o ostrych ograniczeniach czasowych** (ang. *hard real-time*) – przekroczenie terminu powoduje katastrofalne skutki (zagrożenie życia lub zdrowia ludzi, uszkodzenie lub zniszczenie urządzenia). Nie jest istotna wielkość przekroczenia terminu, a jedynie sam fakt jego przekroczenia
- **Systemy o miękkich lub łagodnych ograniczeniach czasowych** (ang. *soft real-time*) - gdy przekroczenie terminu powoduje negatywne skutki. Skutki są tym poważniejsze, im bardziej termin został przekroczony
- **Systemy o mocnych ograniczeniach czasowych** (ang. *firm real-time*) - gdy fakt przekroczenia terminu powoduje całkowitą nieprzydatność wypracowanego przez system wyniku. Fakt niespełnienia wymagań czasowych nie stanowi jednak zagrożenia dla ludzi lub urządzenia (bazy danych czasu rzeczywistego)

Dlaczego Linux nie jest systemem czasu rzeczywistego

- Zastosowany algorytm szeregowania z podziałem czasu
- Niska rozdzielczość zegara systemowego
- Nie wywłaszczalne jądro (nie dotyczy wersji > 2.6)
- Wyłączanie obsługi przerwań w sekcjach krytycznych
- Zastosowanie pamięci wirtualnej
- Optymalizacja wykorzystania zasobów sprzętowych

Po co stosować system operacyjny dla urządzeń wbudowanych?

- Złożoność implementowanych algorytmów
- Złożoność procesów sterowania
- Przenośność aplikacji

Podział systemów operacyjnych dla urządzeń wbudowanych

- System operacyjny dla procesora (mikrokontrolera) z układem zarządzania pamięcią
  - **Linux, MontaVista Linux**
- System operacyjny dla procesora (mikrokontrolera) bez układu zarządzania pamięcią
  - **µClinux, PetaLinux, RTEMS**

Dlaczego brak układu zarządzania pamięcią jest problemem?

- **Brak obsługi pamięci wirtualnej**
  - Brak sprzętowej ochrony pamięci
  - Brak możliwości dynamicznego zmieniania ilości pamięci przydzielonej danemu procesowi
  - Brak obsługi obszaru wymiany
  - Fragmentacja pamięci przy dynamicznej alokacji

System operacyjny czasu rzeczywistego RTEMS

RTEMS
Real-Time Executive for Multiprocessor Systems

(Real Time Executive for Missile Systems)

- System operacyjny czasu rzeczywistego RTOS (Real Time Operating System) rozwijany jako projekt Open Source na licencji GPL.
- RTEMS został opracowany jako wydajny system operacyjny dla urządzeń wbudowanych.
- Dostępne są implementacje RTEMS, tzw. BSP (Board Support Packages), dla wielu procesorów: ARM, ColdFire, MC68000, Intel i960, Intel i386, MIPS, LEON, itd...

Cechy systemu operacyjnego RTEMS

- **Zgodność ze standardami**
  - POSIX 1003.1b API wraz z wątkami
  - TCP/IP wraz z gniazdami BSD
  - uITRON 3.0 API
- **Podstawowe cechy jądra systemu**
  - Wielozadaniowość
  - Wywłaszczanie bazujące na priorytetach i zdarzeniach
  - Komunikacja i synchronizacja międzyprocesowa
  - Dynamiczna alokacja pamięci
  - Możliwość pełnej konfiguracji systemu
- **Wsparcie dla języków skryptowych**
  - Python

Cechy systemu operacyjnego RTEMS

- **Stos TCP/IP**
  - TCP, UDP
  - ICMP, DHCP, RARP
  - RPC, CORBA
  - TFTP, FTPD, HTTPD
- **Wsparcie dla systemów plików**
  - In-Memory Filesystem (IMFS)
  - TFTP Client Filesystem
  - FTP Client Filesystem
  - FAT Filesystem (IDE and CompactFlash)

Procesory wspierane przez RTEMS (1)

Architecture	4.6	4.7	4.8	CVS
Altera NIOS II	No	No	Yes	Yes
ADI Blackfin	No	No	Yes	Yes
ARM with many CPU models	Yes	Yes	Yes	Yes
Atmel AVR	No	Partial	Partial	Partial
AMD A29K	Yes	No	No	No
HP PA-RISC	Yes	No	No	No
Intel/AMD x86 (i386 and above)	Yes	Yes	Yes	Yes
Intel i960	Yes	No	No	No
MIPS including multiple ISA levels	Yes	Yes	Yes	Yes

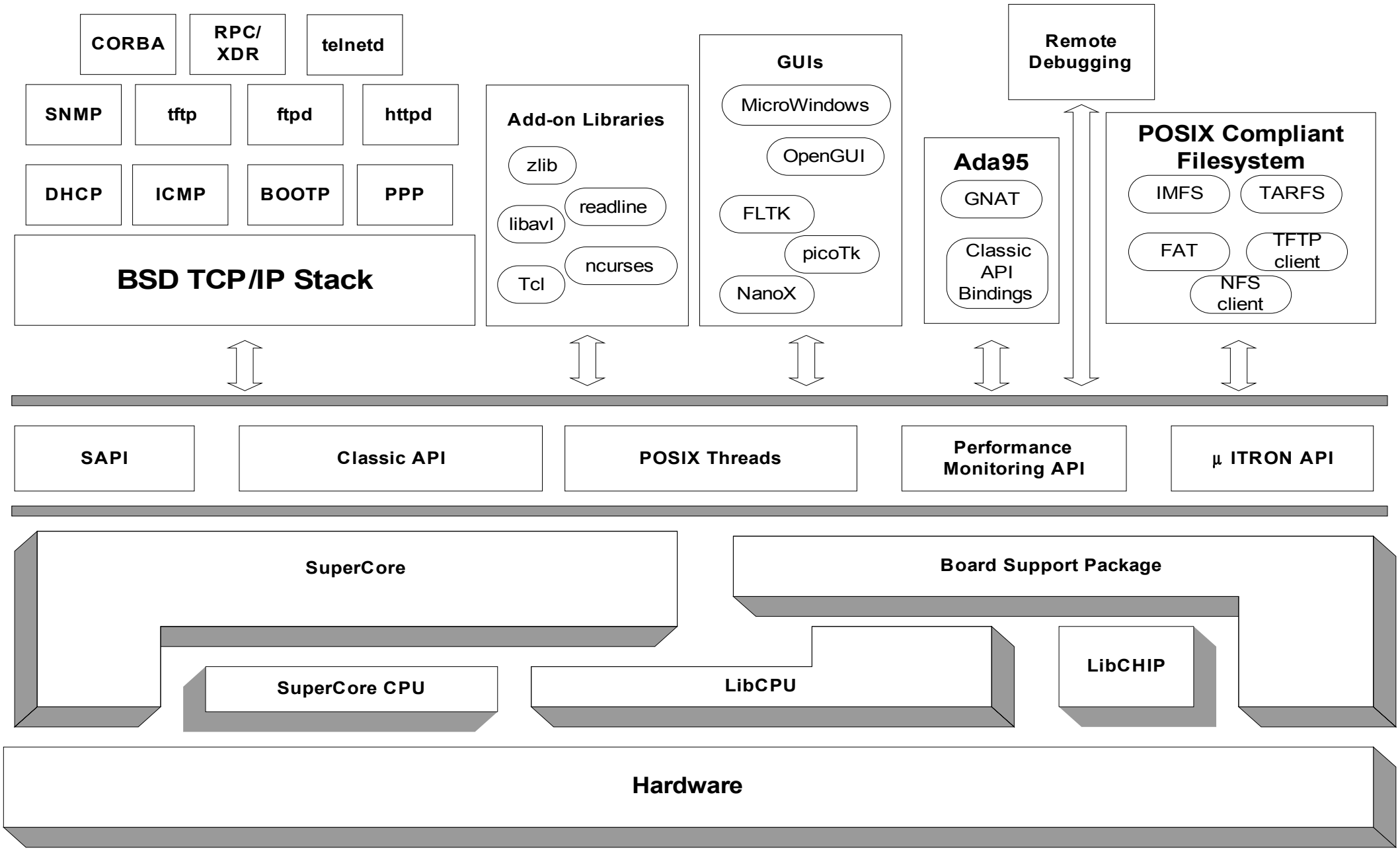
Procesory wspierane przez RTEMS (2)

Architecture	4.6	4.7	4.8	CVS
Freescale MC68xxx	Yes	Yes	Yes	Yes
Freescale MC683xx	Yes	Yes	Yes	Yes
Freescale Coldfire	Yes	Yes	Yes	Yes
OpenCores OR32	Yes	No	No	No
PowerPC including 4xx, 5xx, 6xx, 7xx, 8xx, 52xx, and 74xx	Yes	Yes	Yes	Yes
Reneas H8/300	Yes	Yes	Yes	Yes
Reneas SuperH including SH1, SH2, SH3, and SH4	Yes	Yes	Yes	Yes
SPARC including ERC32 and LEON	Yes	Yes	Yes	Yes
TI C3x/C4x	Yes	No	No	Yes

Procesory wspierane przez RTEMS (2)

- System RTEMS wspiera **trzy** rodzaje API (ang. Application Programming Interface)
  - Natywne API systemu operacyjnego (Classic API),
    - **rtems_status_code** rtems_semaphore_create(**rtems_name** name, **uint32_t** count, **rtems_attribute** attribute_set, **rtems_task_priority** priority_ceiling, **rtems_id** *id);
  - API zgodne z standardem POSIX (z pewnymi ograniczeniami)
    - **int** sem_init(**sem_t** *sem, **int** pshared, **unsigned** int value);
  - API zgodne z standardem ITRON
    - **ER** cre_sem(**ID** semid, **T_CSEM** *pk_csem);

Architektura systemu operacyjnego RTEMS



Porównanie czasu obsługi przerwań oraz przełączenia kontekstu

MVME5500	Interrupt latency (usec) Maximum (Average)	Context Switching(usec) Maximum (Average)
Idle System		
RTEMS	**5.04 (3.45)**	**6.80 (0.96)**
VxWorks	**6.10 (1.58)**	**9.65 (0.91)**
Loaded System		
RTEMS	**8.17 (3.74)**	**17.48 (1.69)**
VxWorks	**13.90 (1.68)**	**20.80 (1.90)**

RTEMS - Podsumowanie

- W przypadku pracy bez obciążenia zarówno RTEMS jak i VxWorks wykazują podobne opóźnienia czasowe
- W przypadku pracy z obciążeniem RTEMS wykazuje niewiele większą stabilność opóźnień niż system WxVorks
- Jeżeli chodzi o wydajność i stabilność RTEMS jest systemem porównywalnym do komercyjnych systemów czasu rzeczywistego
- RTEMS jest systemem elastycznym, niezawodnym oraz odpowiednim nawet do zastosowań o ostrych wymaganiach czasowych
- RTEMS jest nieustannie rozwijany co potwierdza stale zwiększająca się lista obsługiwanych procesorów
- RTEMS nie jest tak popularnym systemem jak Linux, jednakże gwarantuje wysoką wydajność i stałość opóźnień czasowych nawet w przypadku bardzo obciążonego systemu. Tego typu parametry są mniej przewidywalne w przypadku systemu Linux
- Dostęp do implementacji RTEMS dla danej platformy sprzętowej jest gwarantowany nawet jeżeli nie jest ona wspierana w kolejnych wersjach systemu

Obsługa sytuacji wyjątkowych
(Exception Processing)

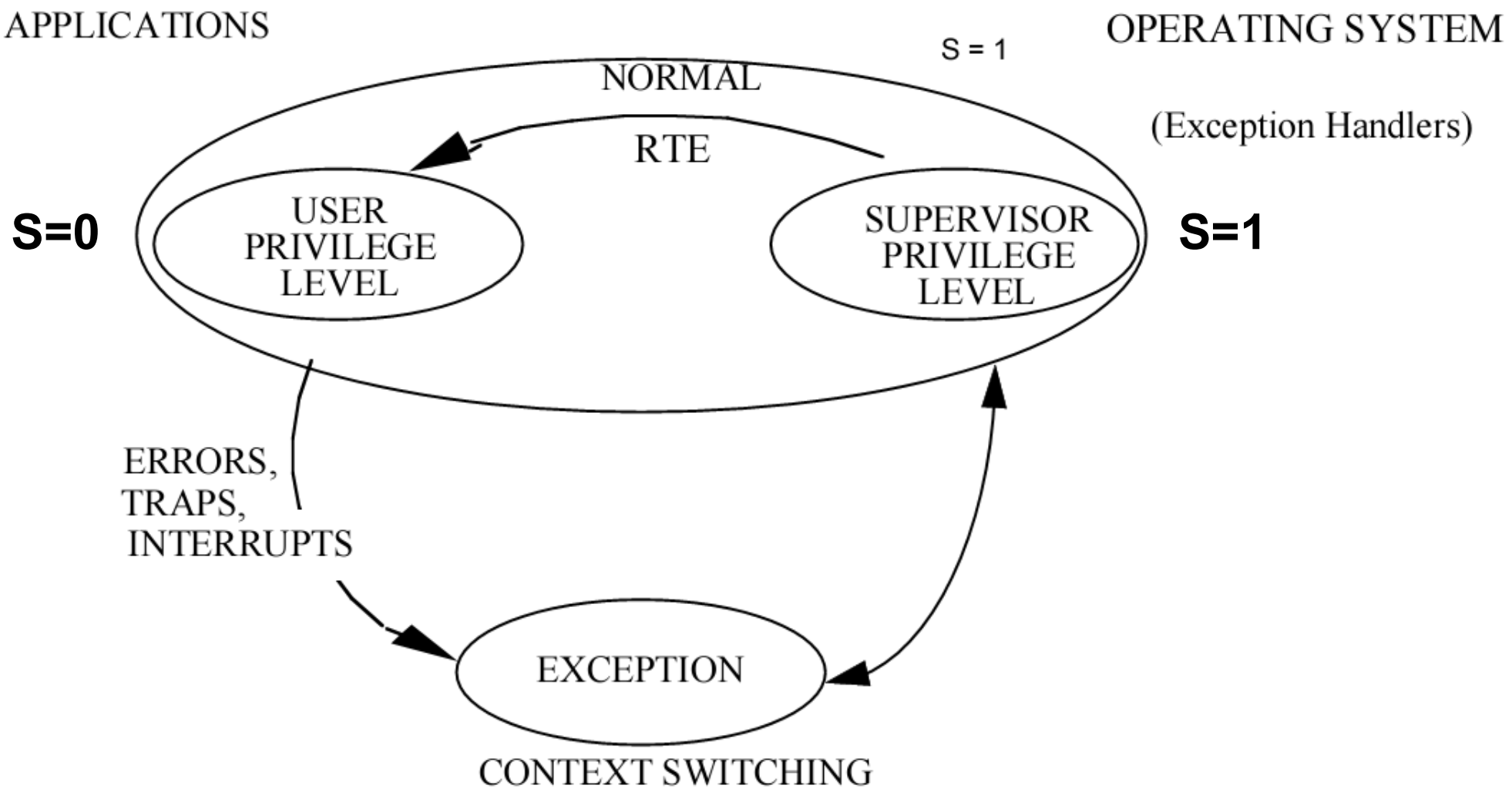
Przerwania

Przerwanie (ang. interrupt) – sygnał powodujący zmianę przepływu sterowania. Pojawienie się przerwania powoduje wstrzymanie aktualnie wykonywanego programu i wykonanie przez procesor kodu procedury obsługi przerwania (ang. interrupt handler).

Przerwania dzielą się na dwie grupy:

1. Sprzętowe:
   - 1.1. Zewnętrzne – sygnał przerwania pochodzi z zewnętrznego układu,
   - 1.2. Wewnętrzne – zgłaszane przez procesor w celu sygnalizacji sytuacji wyjątkowych, tzw. wyjątek (ang. exception),
     - a) faults (niepowodzenie) – sytuacje, w których aktualnie wykonywana instrukcja powoduje błąd,
     - b) traps (pułapki) – sytuacja, która nie jest błędem, jej wystąpienie ma na celu wykonanie określonego kodu,
     - c) aborts – błędy, których nie można naprawić,
   - 1.3 Niemaskowalne – przerwania, które nie da się wyłączyć,
2. Programowe – z kodu programu wywoływana jest procedura obsługi przerwania.

Obsługa sytuacji wyjątkowych



Obsługa sytuacji wyjątkowych

Wykonanie niedozwolonej operacji przez procesor w danym stanie uprzywilejowania powoduje wygenerowanie wyjątku.

Obsługa wyjątku obejmuje wszystkie operacje od momentu wykrycia błędu do pobrania pierwszej instrukcji obsługującej sytuację wyjątkową.

1. a) Wykonanie kopii SR,
   b) Przejście do trybu superużytkownika (S<=1; T<=0; M<=0),
   c) Ustawienie maski IRQ na poziomie zgłaszanego przerwania.
2. Określenie wektora obsługiwanego wyjątku (przerwania).
3. Odłożenie kopi obecnego kontekstu na wierzchołku stosu superużytkownika (ramka zdarzenia wyjątkowego).
4. Obliczenie adresu pierwszej instrukcji procedury obsługującej dany wyjątek (przerwanie).

VBR(31..20) + 4 x numer_wektora

Detekcja przerwań zostaje wstrzymana na jedną instrukcję przed wykonaniem procedury obsługującej dany wyjątek.

Daje to możliwość wyłączenia przerwań i poprawnego obsłużenia danego wyjątku, np. wykonanie instrukcji **STLDSR.**

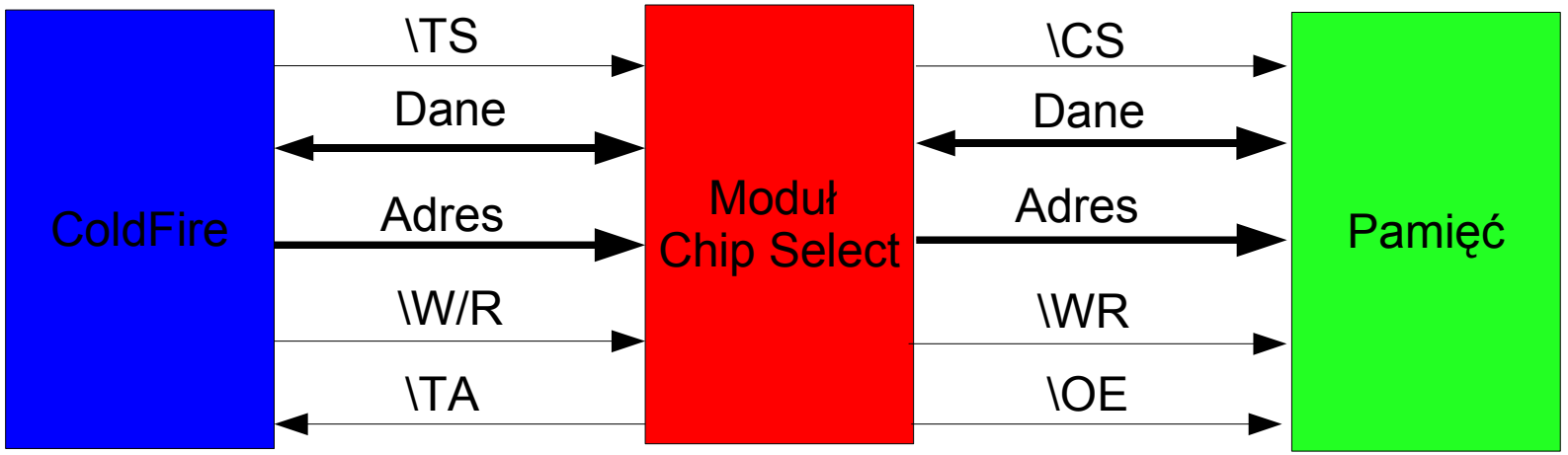
Tablica wektorów wyjątków i przerwań

Vector Number(S)	Vector Offset (Hex)	Stacked Program Counter	Assignment
0	0x000	—	Initial stack pointer
1	0x004	—	Initial program counter
2	0x008	Fault	Access error
3	0x00C	Fault	Address error
4	0x010	Fault	Illegal instruction
5	0x014	Fault	Divide by zero
6–7	0x018–0x01C	—	Reserved
8	0x020	Fault	Privilege violation
9	0x024	Next	Trace
10	0x028	Fault	Unimplemented line-a opcode
11	0x02C	Fault	Unimplemented line-f opcode
12	0x030	Next	Debug interrupt
13	0x034	—	Reserved
14	0x038	Fault	Format error
15–23	0x03C–0x05C	—	Reserved
24	0x060	Next	Spurious interrupt
25–31	0x064-0x07C	—	Reserved
32–47	0x080–0x0BC	Next	Trap # 0-15 instructions
48–63	0x0C0–0x0FC	—	Reserved
64–255	0x100–0x3FC	Next	User-defined interrupts

Wyjątek dostępu do pamięci (1)

- Adres nie istniejącego urządzenia lub pamięci (adres nie jest przypisany do żadnego rejestru konfiguracyjnego CSCR0-CSCR6)
- Próba zapisu do pamięci zabezpieczonej przed zapisem lub tylko do odczytu
- Próba dostępu do pamięci superużytkownika z poziomu zwykłego użytkownika

generują wyjątek **Access Error**



Dzielenie przez zero

Destination/Source → Destination

DIVS.W <ea>y,Dx 32-bit Dx/16-bit <ea>y Æ (16r:16q) in Dx
DIVS.L <ea>y,Dx 32-bit Dx/32-bit <ea>y Æ 32q in Dx
where q indicates the quotient, and r indicates the remainder

An attempt to divide by zero results in a divide-by-zero exception and no registers are affected. The resulting exception stack frame points to the offending divide opcode. If overflow is detected, the destination register is unaffected. An overflow occurs if the quotient is larger than a 16-bit (.W) or 32-bit (.L) signed integer.

Dzielenie liczby ze znakiem podanej jako operand przeznaczenia przez liczbę podaną jako operand źródłowy. Wynik dzielenia umieszczony jest pod adresem wskazywanym przez operand przeznaczenia.

Jeżeli operand źródłowy jest równy 0 generowany jest wyjątek **Division by Zero**.

Naruszenie uprawnień superużytkownika

Próba wykonania instrukcji wymagających uprawnień superużytkownika przez użytkownika generuje wyjątek **Privilege Violation**

OPCODE	OPERAND SIZE	ADDRESSING MODE
*HALT	UNSIZED	CONFIGURABLE
MOVE SR	W	Dn
MOVEC	L	Rn,Rc
RTE	UNSIZED	-
STOP	UNSIZED	-
WDEBUG	L	<EA>
*PULSE	UNSIZED	-

TRAP

Wykonanie polecenia **Trap** powoduje wygenerowanie wyjątku o numerze podanym w postaci operandu.

TRAP #<vector>	$1 \rightarrow$ S-Bit of SR $SP - 4 \rightarrow SP$; nextPC $\rightarrow$ (SP); $SP - 2 \rightarrow SP$; $SR \rightarrow$ (SP); $SP - 2 \rightarrow SP$; Format/Offset $\rightarrow$ (SP); $(VBR + 0x80 + 4*n) \rightarrow PC$ where n is the TRAP vector number
TRAPcc TRAPcc.W # < data > TRAPcc.L # < data >	
TRAPV	

Adres wiersza w tablicy wyjątków obsługującego dany wyjątek można policzyć zgodnie z równaniem:

0x80 + 4 * #<vector>

Moduł procesora 68k umożliwia obsłużenie do 16 zdefiniowanych zdarzeń

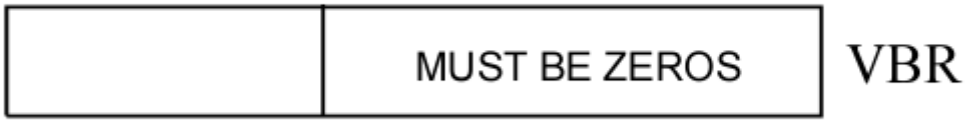
Obsługa nieznanego przerwania

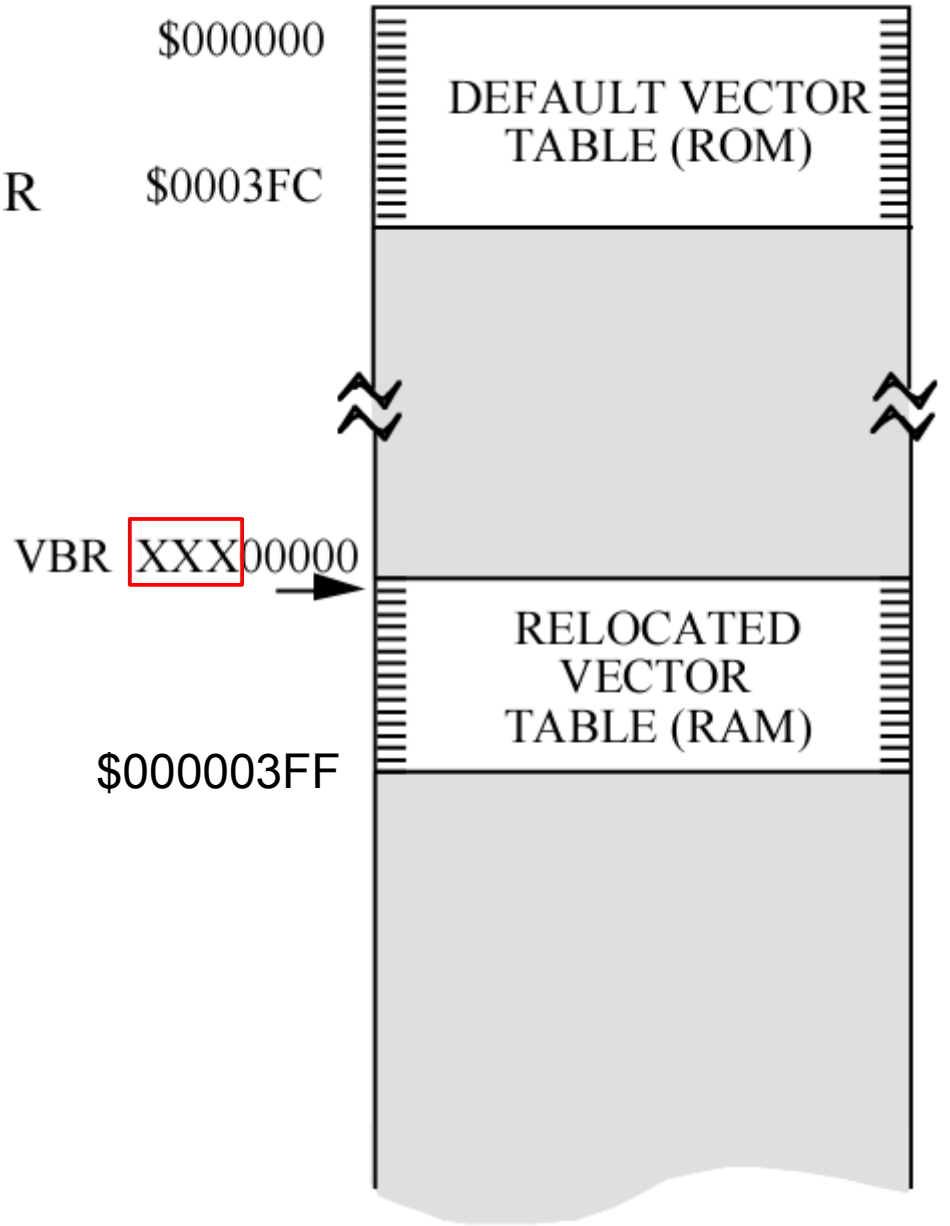
Jeżeli praca procesora zostanie przerwania (w wyniku zgłoszenia przerwania przez sterownik przerwań), jednak sterownik przerwań nie jest w stanie określić źródła przerwania o odpowiednio wysokiej masce następuje wygenerowanie wyjątku **Spurious Interrupt**.

Podwójny wyjątek

Jeżeli podczas obsługi wyjątku zostanie wygenerowany kolejny wyjątek (**fault-on-fault**) procesor natychmiast przerywa pracę. Procesor pozostaje w stanie bezczynności, do momentu wygenerowania sygnału reset.

Tablica adresów wyjątków oraz przerwań



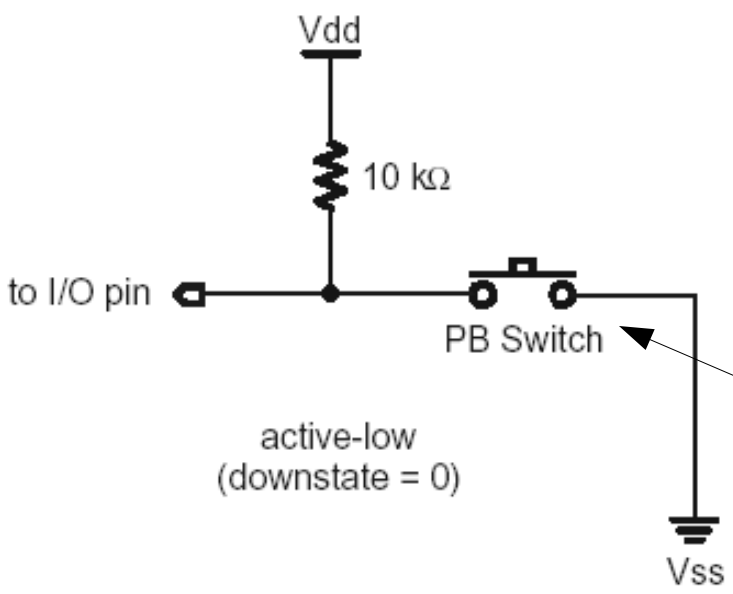


Tablica musi zostać umieszczona na 1 MB granicy.

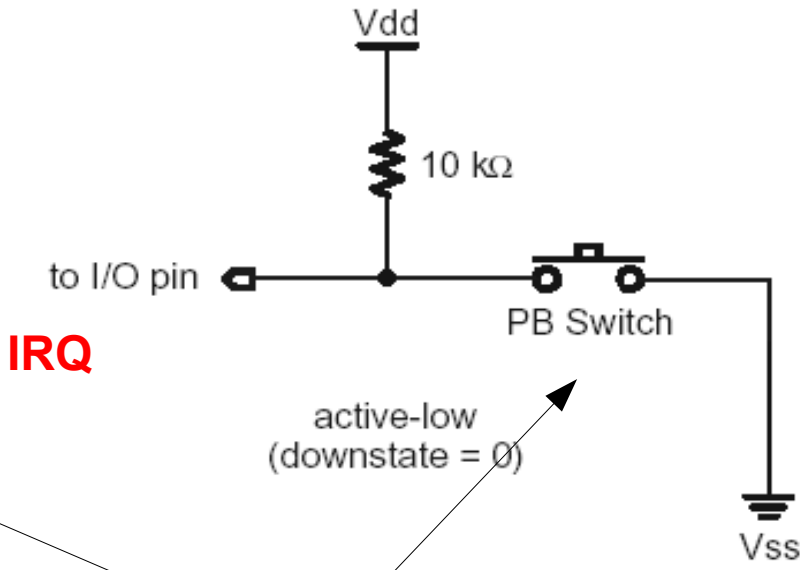
VBR (reset) = 0x0000.0000

Przerwania

Polling loop



Interrupt




```
                LEA         KEY_STATUS, A0
                LEA         KE_VALUE, A1
TEST_LOOP       BTST        #0, (A0)
                BNE         TEST_LOOP
                MOVE.B      (A1), D1
```

Sygnał asynchroniczny

Sterownik przerwań (2)

Sterownik przerwań rozpoczyna cykl obsługi wyjątku jeżeli zgłoszone przerwanie ma poziom wyższy od poziomu ustawionego w rejestrze SR. Powyższa reguła nie dotyczy przerwań o poziomie równym 7.

Przerwania o poziomie równym 7 obsługiwane są zawsze – przerwania niemaskowalne.

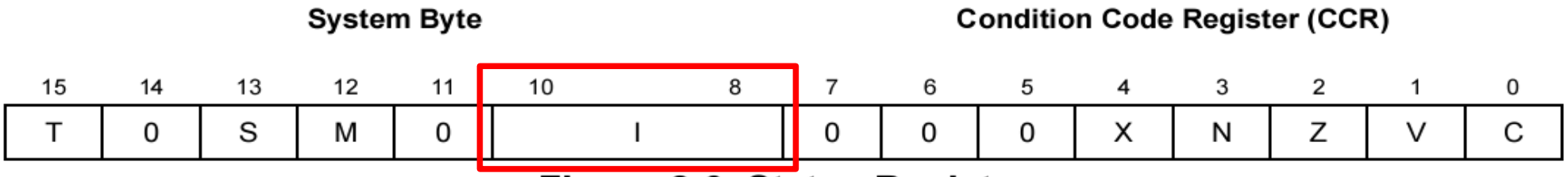
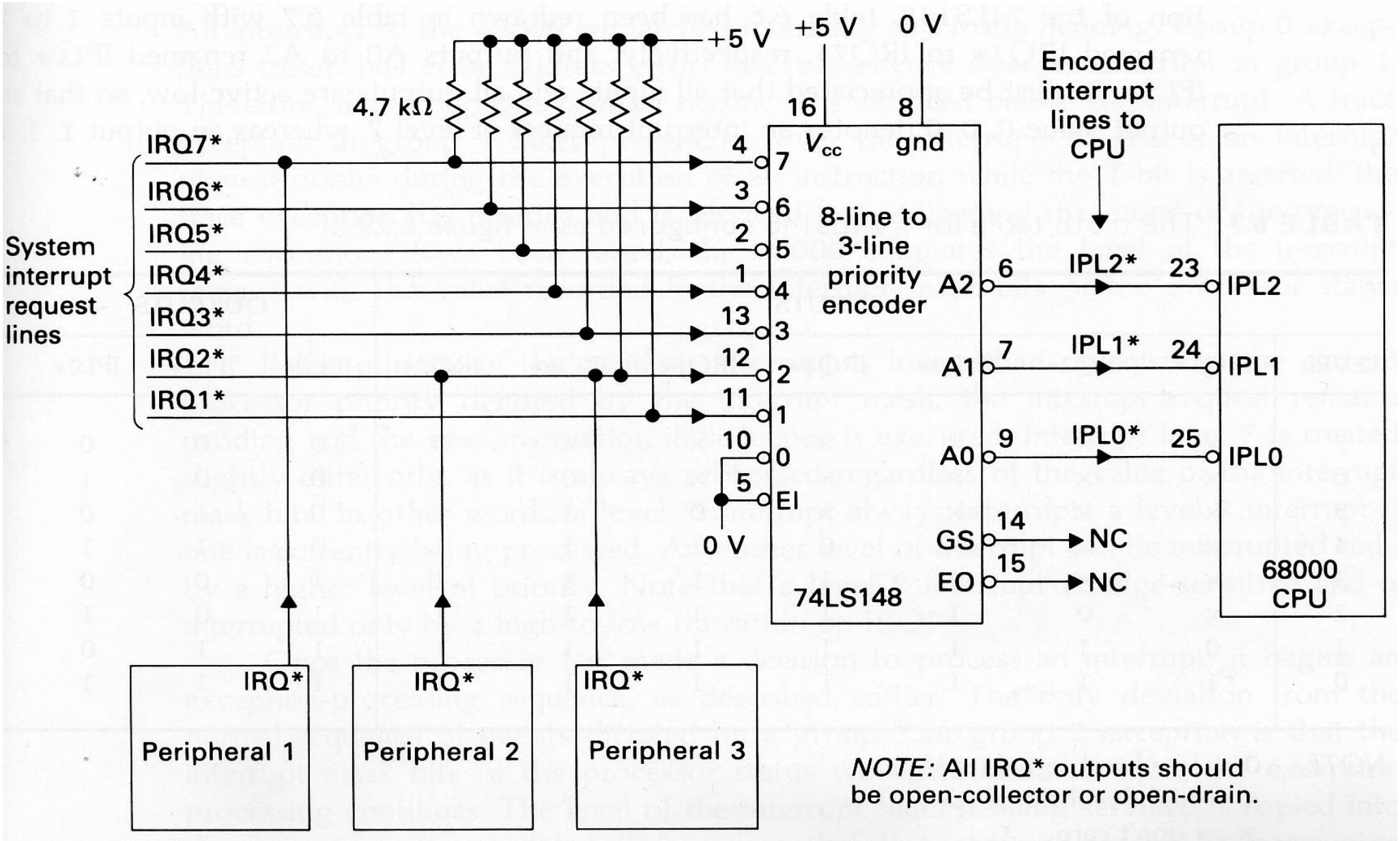


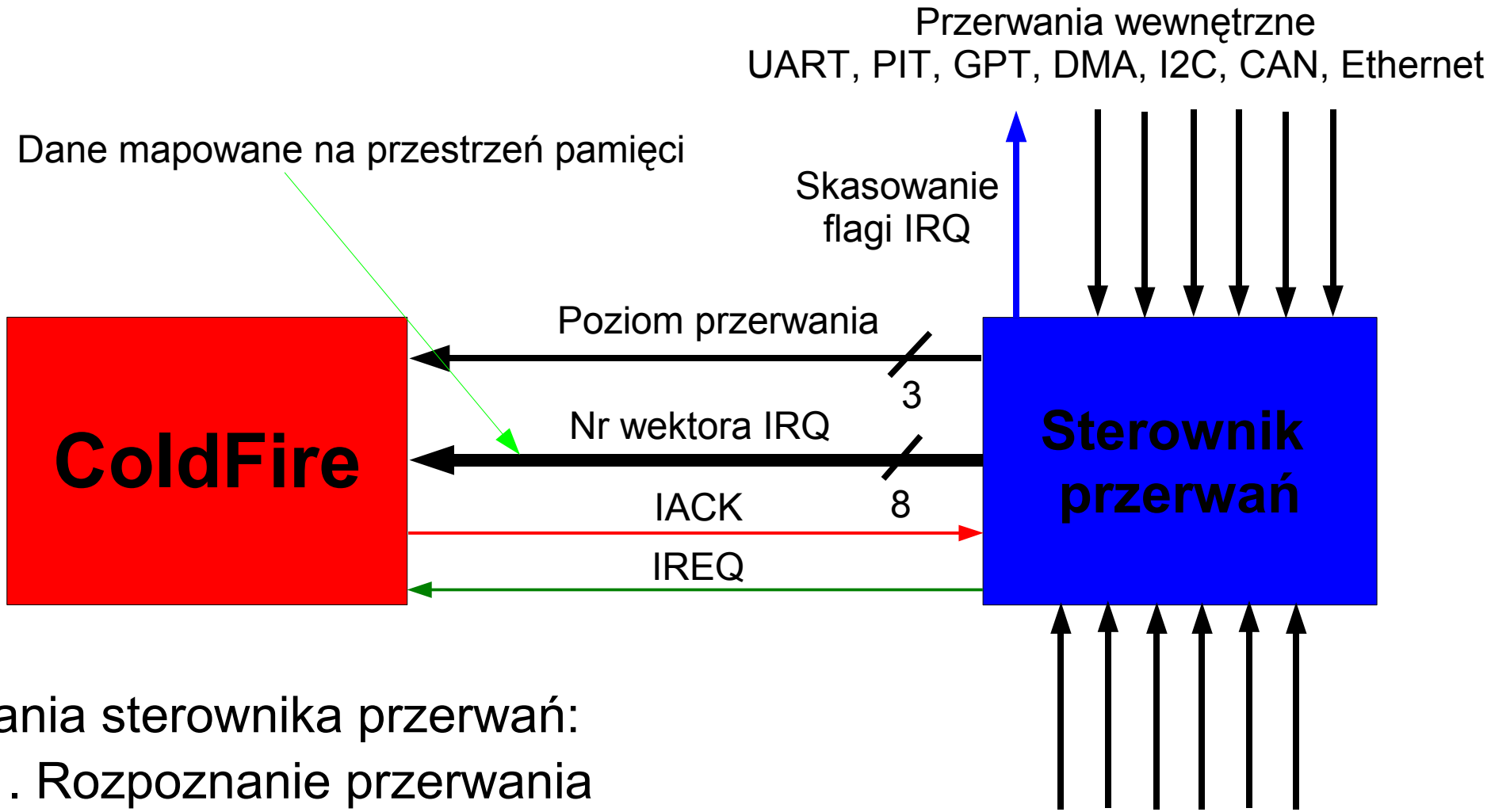
Figure 2-6. Status Register

I > poziom IRQ
IRQ=7 obsługiwane zawsze

Przerwania zewnętrzne



Współpraca sterownika przerwań z procesorem



Zadania sterownika przerwań:

1. Rozpoznanie przerwania
2. Wyznaczenie przerwania o najwyższej masce i priorytecie
3. Obliczenie numeru wektora przerwania

Obsługa przerwania

1. Przejście do trybu superużytkownika (S=1, T=0)
2. Odczytanie wektora aktywnego przerwania
3. Odłożenie na stos kopii rejestru SR, numeru przerwania oraz adresu powrotu
4. Obliczenie przesunięcia w tablicy przerwań oraz odczytanie adresu procedury obsługującej przerwanie
5. Obniżenie maski przerwań w rejestrze SR do poziomu obsługiwanego przerwania
6. Skok do procedury obsługującej przerwanie

Sterownik przerwań

Rejestr stanu **SR**

15	14	13	12	11	10 … 8	7	6	5	4	3	2	1	0
T	0	S	M	0	I	0	0	0	X	N	Z	V	C

Maska przerwań – poziom obsługiwanego przerwania

Priorytety przerwań

ICR[2:0]	Priority	Interrupt Sources
111	7 (Highest)	8-63
110	6	8-63
101	5	8-63
100	4	8-63
—	Fixed Midpoint Priority	1-7
011	3	8-63
010	2	8-63
001	1	8-63
000	0 (Lowest)	8-63

przerwania zewnętrzne

Źródła przerwań sterownika INTC0 (1)

Source	Module	Flag	Source Description	Flag Clearing Mechanism
1	EPORT	EPF1	Edge port flag 1	Write EPF1 = 1
2		EPF2	Edge port flag 2	Write EPF2 = 1
3		EPF3	Edge port flag 3	Write EPF3 = 1
4		EPF4	Edge port flag 4	Write EPF4 = 1
5		EPF5	Edge port flag 5	Write EPF5 = 1
6		EPF6	Edge port flag 6	Write EPF6 = 1
7		EPF7	Edge port flag 7	Write EPF7 = 1
8	SCM	SWT1	Software watchdog timeout	Cleared when service complete
9	DMA	DONE	DMA Channel 0 transfer complete	Write DONE = 1
10		DONE	DMA Channel 1 transfer complete	Write DONE = 1
11		DONE	DMA Channel 2 transfer complete	Write DONE = 1
12		DONE	DMA Channel 3 transfer complete	Write DONE = 1
13	UART0	Multiple	UART0 interrupt	Cleared when service complete
14	UART1	Multiple	UART1 interrupt	Cleared when service complete
15	UART2	Multiple	UART2 interrupt	Cleared when service complete
17	I^2C	IIF	I^2C interrupt	Write IIF = 0
18	QSPI	Multiple	QSPI interrupt	See QIR description
19	DTIM0	CAP/REF	DTIM0 capture/reference event	Write CAP = 1 or REF = 1
20	DTIM1	CAP/REF	DTIM1 capture/reference event	Write CAP = 1 or REF = 1
21	DTIM2	CAP/REF	DTIM2 capture/reference event	Write CAP = 1 or REF = 1
22	DTIM3	CAP/REF	DTIM3 capture/reference event	Write CAP = 1 or REF = 1
23	FEC	X_INTF	Transmit frame interrupt	Write X_INTF = 1
24		X_INTB	Transmit buffer interrupt	Write X_INTB = 1
34		BABT	Babbling transmit error	Write BABT = 1
35		BABR	Babbling receive error	Write BABR = 1
36	PMM	LVDF	LVD	Write LVDF = 1

Język niskiego poziomu - asembler

Rodzaje instrukcji procesora

- Arytmetyczne
- Transferu danych
- Sterujące przepływem programu
- Logiczne oraz arytmetyczne przesunięcia
- Operacje na bitach
- Specjalne
- Diagnostyczne
- Sterujące pamięcią podręczną

Rodzaje instrukcji procesora Motorola 68k

DATA

CLR	MOVE
EXT	MOVEQ
EXTB	SWAP

SHIFT

ASL	LSR
ASR	LSL

CONTROL

BRA	*RTE
BSR	RTS
JSR	TRAP
JMP	TRAPF
NOP	

ARITHMETIC

ADD	MUL
CMP	NEG
TST	SUB

MULTIPLE PRECISION

ADDX	SUBX
NEGX	

CONDITIONAL

Bcc	Scc

LOGICAL

AND	NOT
EOR	OR

BIT MANIPULATION

BCHG	BSET
BCLR	BTST

SPECIAL INSTRUCTIONS

ILLEGAL	LINK	MOVEM
LEA	UNLK	*MOVEC
PEA	*STOP	MAC

DEBUG

*PULSE
WDDATA
*WDEBUG
! HALT

Kod instrukcji (Opcode)

Bits 15–12	Hex	Operation
0000	0	Bit Manipulation/Immediate
0001	1	Move Byte
0010	2	Move Longword
0011	3	Move Word
0100	4	Miscellaneous
0101	5	ADDQ/SUBQ/Scc/TPF
0110	6	Bcc/BSR/BRA
0111	7	MOVEQ/MVS/MVZ
1000	8	OR/DIV
1001	9	SUB/SUBX
1010	A	MAC/EMAC instructions/MOV3Q
1011	B	CMP/EOR
1100	C	AND/MUL
1101	D	ADD/ADDX
1110	E	Shift
1111	F	Floating-Point/Debug/Cache Instructions

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	0	0	0	0	1	8-bit displacement							
16-bit displacement if 8-bit displacement = 0x00															
32-bit displacement if 8-bit displacement = 0xFF															

Instrukcje arytmetyczne

Instruction	Operand Syntax	Operand Size	Operation	ISA, (E)MAC
ADD ADDA	Dy,<ea>x <ea>y,Dx <ea>y,Ax	L L L	Source + Destination → Destination	ISA_A
ADDI ADDQ	#<data>,Dx #<data>,<ea>x	L L	Immediate Data + Destination → Destination	ISA_A
ADDX	Dy,Dx	L	Source + Destination + CCR[X] → Destination	ISA_A
CLR	<ea>x	B, W, L	0 → Destination	ISA_A
CMP CMPA	<ea>y,Dx <ea>y,Ax	B, W, L W, L	Destination – Source → CCR	ISA_A[1]
CMPI	#<data>,Dx	B, W, L	Destination – Immediate Data → CCR	ISA_A[1]
DIVS/DIVU	<ea>y,Dx	W, L	Destination / Source → Destination (Signed or Unsigned)	ISA_A
EXT EXTB	Dx Dx Dx	B → W W → L B → L	Sign-Extended Destination → Destination	ISA_A
MAC	Ry,RxSF,ACCx Ry,RxSF,<ea>y,Rw,ACCx	W, L W, L	ACCx + (Ry * Rx){<<|>>}SF → ACCx ACCx + (Ry * Rx){<<|>>}SF → ACCx; (<ea>y(&MASK)) → Rw	ISA_A
MSAC	Ry,RxSF,ACCx Ry,RxSF,<ea>y,Rw,ACCx	W, L W, L	ACCx - (Ry * Rx){<<|>>}SF → ACCx ACCx - (Ry * Rx){<<|>>}SF → ACCx; (<ea>y(&MASK)) → Rw	ISA_A
MULS/MULU	<ea>y,Dx	W * W → L L * L → L	Source * Destination → Destination (Signed or Unsigned)	ISA_A
NEG	Dx	L	0 – Destination → Destination	ISA_A
NEGX	Dx	L	0 – Destination – CCR[X] → Destination	ISA_A
REMS/REMU	<ea>y,Dw:Dx	L	Destination / Source → Remainder (Signed or Unsigned)	ISA_A
SUB SUBA	<ea>y,Dx Dy,<ea>x <ea>y,Ax	L L L	Destination - Source → Destination	ISA_A
SUBI SUBQ	#<data>,Dx #<data>,<ea>x	L L	Destination – Immediate Data → Destination	ISA_A
SUBX	Dy,Dx	L	Destination – Source – CCR[X] → Destination	ISA_A

ADD

Operation: Source + Destination → Destination

Assembler Syntax: ADD.L <ea>y,Dx
ADD.L Dy,<ea>x

Condition Codes:

X	N	Z	V	C
*	*	*	*	*

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	0	1	Register				Opmode		Effective Address					
										Mode		Register			

Tryby adresowania argumentu źródłowego <ea>y:

Dy, Ay, (Ay), (Ay)+, -(Ay), (d16,Ay), (d8,Ay,Xi), (xxx).W/L, #<data>, (d16,PC), (d8,PC,Xi)

Tryby adresowania argumentu docelowego <ea>x:

(Ax), (Ax)+, -(Ax), (d16,Ax), (d8,Ax,Xi), (xxx).W/L

ADDA.L <ea>y, Ax

Operation: Source + Destination → Destination

Assembler Syntax: ADDA.L <ea>y,Ax

Condition Codes: Not affected

Tryby adresowania argumentu źródłowego <ea>y:

Dy, Ay, (Ay), (Ay)+, -(Ay), (d16,Ay), (d8,Ay,Xi), (xxx).W/L, #<data>, (d16,PC), (d8,PC,Xi)

```
ADD.L   D0, D1
ADD.L   D0, D0
ADD.L   D0, A0

ADD.L  A0, D0

ADDA.L  D0, A0
ADDA.L  A0, A0

ADDA.L  A0, D0        Error: operands mismatch -- statement `adda.l %A0,%D0' ignored
```

```
1 001a      D3F0 1CEC           ADDA.L      -20(A0, D1*4), A1
2 001e      D3FC 0000  0003     ADDA.L      #$03, A1
```

ADDI.L #<data>, Dx

Operation: Immediate Data + Destination → Destination

Condition Codes:

X	N	Z	V	C
*	*	*	*	*

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	1	1	0	1	0	0	0	0	Register, Dx		
Upper Word of Immediate Data															
Lower Word of Immediate Data															

1 001a 0681 0000 0009 ADDI.L #$09, D1

2 0020 **0681 0000 0003** **ADDI.L #$03, D1**

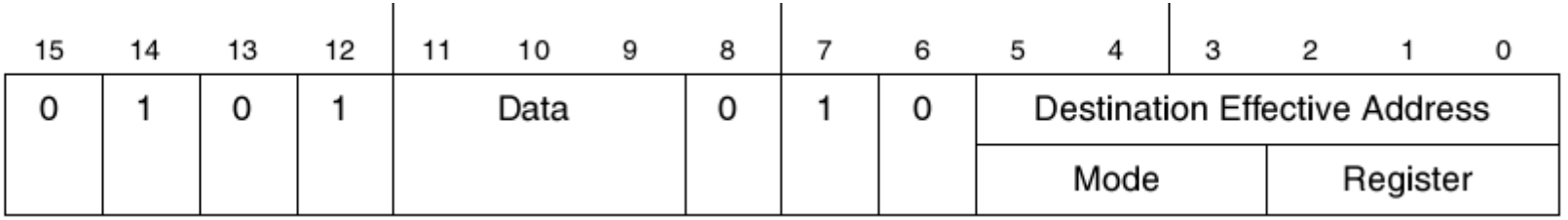
3 ???? ADDI.L #1, A0

Error: operands mismatch -- statement `addi.l #1,A0' ignored

ADDQ.L #<data>, <ea>x

Operation: Immediate Data + Destination → Destination

Instruction Format:



1 - 8

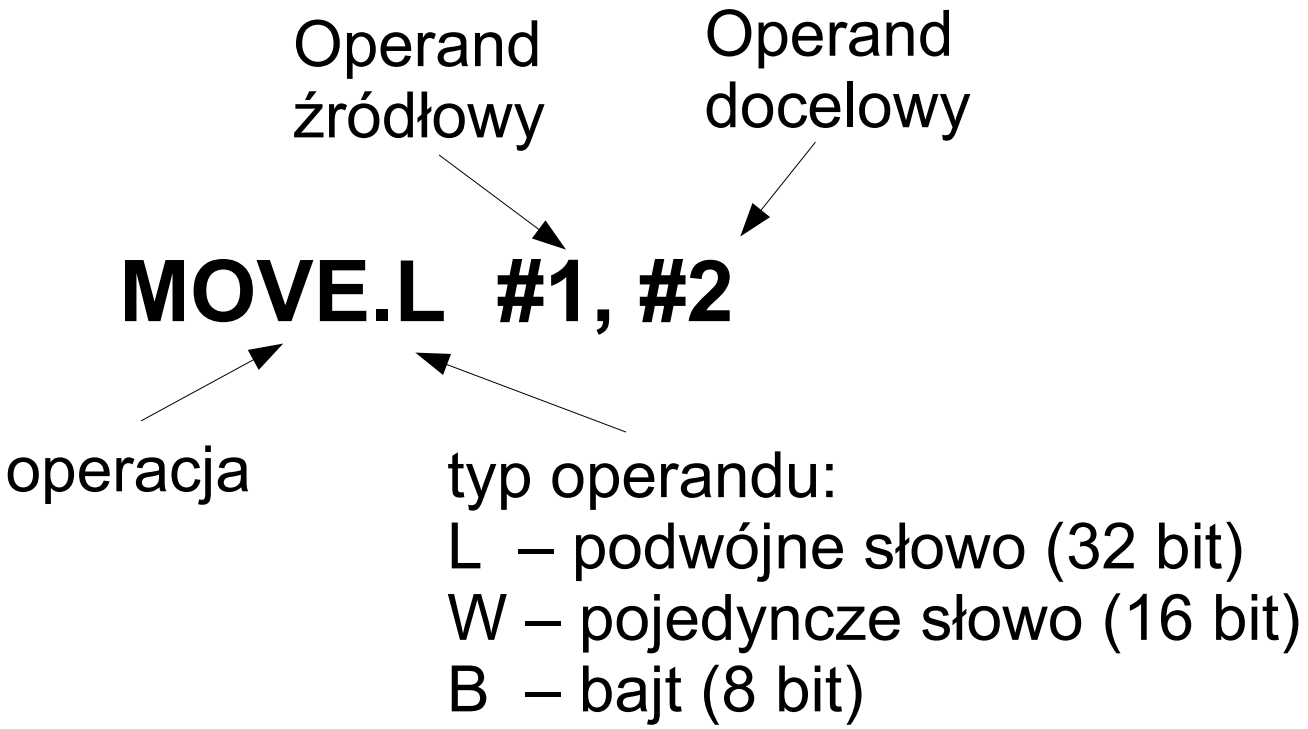
Tryby adresowania argumentu docelowego <ea>x:

Dx, Ax, (Ax), (Ax)+, -(Ax), (d16,Ax), (d8,Ax,Xi), (xxx).W/L

```
1 001a     0681 0000 0001      ADDI.L  #$01, D1
2 0020     0681 FFFF FFFF      ADDI.L  #$FFFFFFFF, D1
3 0026     5682                ADDQ.L  #$3, D2

4 001a     5681                ADD.L   #$3, D1  =>  ADDQ
5 001a     0681 0000 0009      ADD.L   #9, D1   =>  ADDI
6 001c     D2B8 0003           ADD.L    3, D1   =>  ADD
```

```
ADDQ.L    #9, D3     gp.asm:30: Error: operands mismatch --
                     statement `addq.l #9,D3' ignored

ADDQ.L    #0, D2     gp.asm:31: Error: operands mismatch --
                     statement `addq.l #0,D2' ignored
```

Tryby adresowania



MOVE.W	$00A1.FFFF, D0	przesłanie zawartości komórki pamięci o adresie $00A1.FFFF do rejestru D0
MOVE.W	(A0), (A7)+	przesłanie zawartości komórki pamięci o adresie zawartym w A0 do komórki o adresie A7, postinkrementacja adresu A7

Podstawowe tryby adresowania procesora Motorola 68k

Addressing Modes	Syntax	Mode Field	Reg. Field	
Register Direct				
Data	Dn	000	reg. no.	move.l $1000,D0
Address	An	001	reg. no.	move.l $1000,A0
Register Indirect				
Address	(An)	010	reg. no.	move.l D0,(A1)
Address with Postincrement	(An)+	011	reg. no.	move.l D1,(A2)+
Address with Predecrement	–(An)	100	reg. no.	move.l D2,-(A3)
Address with Displacement	(d_{16},An)	101	reg. no.	move.l D7,50(A7)
Address Register Indirect with Scaled Index and 8-Bit Displacement	(d_8,An,Xi*SF)	110	reg. no.	move.l D7,50(A7,D6*2)
Program Counter Indirect with Displacement	(d_{16},PC)	111	010	move.l D7,50(PC)
Program Counter Indirect with Scaled Index and 8-Bit Displacement	(d_8,PC,Xi*SF)	111	011	move.l D7,50(PC,D6*2)
Absolute Data Addressing				
Short	(xxx).W	111	000	move.l $1234,D0
Long	(xxx).L	111	001	move.l $1234FFFF,D0
Immediate	#<xxx>	111	100	move.l #$1234,D0

Tryby adresowania dla instrukcji ADD

Operation: Source + Destination → Destination

Assembler Syntax: ADD.L <ea>y,Dx
ADD.L Dy,<ea>x

Condition Codes:

X	N	Z	V	C
*	*	*	*	*

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	0	1	Register				Opmode				Effective Address			
										Mode			Register		

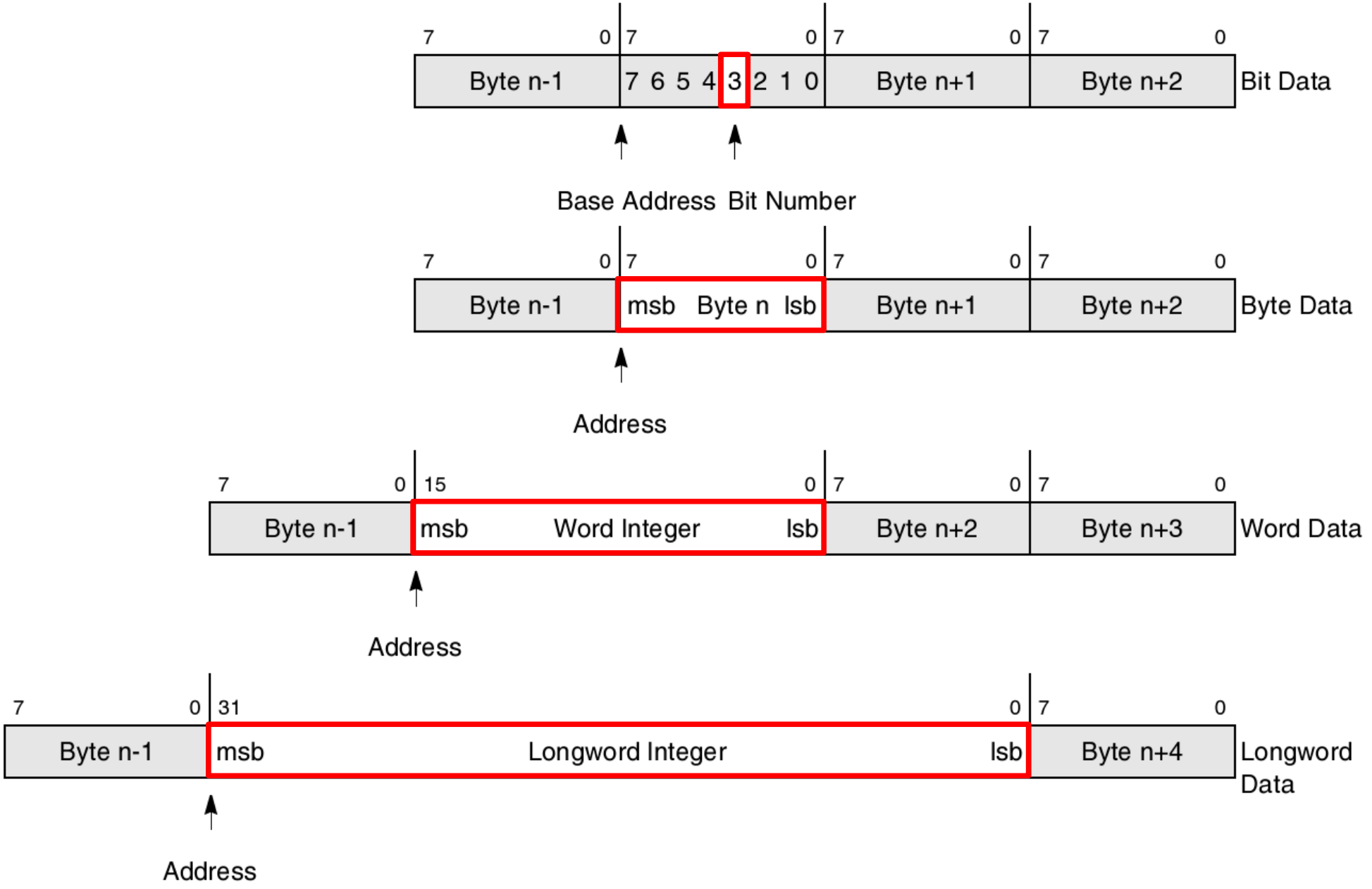
Tryby adresowania argumentu źródłowego <ea>y:

Dy, Ay, (Ay), (Ay)+, -(Ay), (d16,Ay), (d8,Ay,Xi), (xxx).W/L, #<data>, (d16,PC), (d8,PC,Xi)

Tryby adresowania argumentu docelowego <ea>x:

(Ax), (Ax)+, -(Ax), (d16,Ax), (d8,Ax,Xi), (xxx).W/L

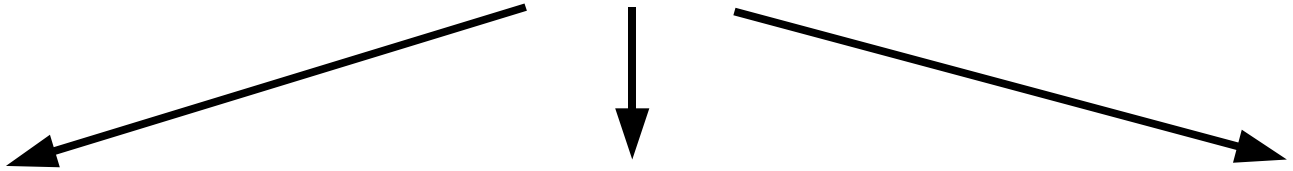
Model programowy procesora Motorola 68k



Kolejność bajtów w pamięci (1)

Bajt – najmniejsza adresowalna jednostka pamięci komputerowej

Endianess



Big-endian

...pod najmłodszym adresem umieszczony jest najstarszy bajt

podobnie jak w języku polskim, angielskim

Motorola, SPARC, ARM

middle-endian

liczby zmiennoprzecinkowe podwójnej precyzji

VAX and ARM

Little-endian

...pod najstarszym adresem umieszczony jest najstarszy bajt

podobnie jak w językach arabskich, hebrajski

Intel x86, 6502 VAX

Bi-Endian

ARM, PowerPC (za wyjątkiem PPC970/G5), DEC Alpha, MIPS, PA-RISC oraz IA64

Kolejność bajtów w pamięci (2)

Architektura 8-bitowa

	7 ... 0
0x0000.0000	Byte 1
0x0000.0001	Byte 2
0x0000.0002	Byte 3
0x0000.0003	Byte 4
0x0000.0004	Byte 5

	7 ... 0
0x0000.0000	0x12
0x0000.0001	0x34
0x0000.0002	0x56
0x0000.0003	0x78
0x0000.0004	0x90

Kolejność bajtów w pamięci (3)

Byte 4 ... Byte 1
MSB LSB

Big-endian

0x0000.0000	Byte 4	Byte 3	Byte 2	Byte 1
0x0000.0004	Byte 8	Byte 7	Byte 6	Byte 5
0x0000.0008	Byte 12	...	...	...
0x0000.000C				
0x0000.0010				

Little-endian

0x0000.0000	Byte 1	Byte 2	Byte 3	Byte 4
0x0000.0004	Byte 5	Byte 6	Byte 7	Byte 8
0x0000.0008	Byte 9	...	...	...
0x0000.000C				
0x0000.0010				

Kolejność bajtów w pamięci (4)

Podwójne słowo (DW): **0x1234.5678**

Big-endian

	32 24	23 16	15 8	7 0
0x0000.0000	0x12	0x34	0x56	0x78
0x0000.0004	Byte 5	Byte 6	Byte 7	Byte 8
0x0000.0008	Byte 9	...	...	...
0x0000.000C				
0x0000.0010				

Little-endian

	32 24	23 16	15 8	7 0
0x0000.0000	0x78	0x56	0x34	0x12
0x0000.0004	Byte 8	Byte 7	Byte 6	Byte 5
0x0000.0008	Byte 12	...	...	...
0x0000.000C				
0x0000.0010				

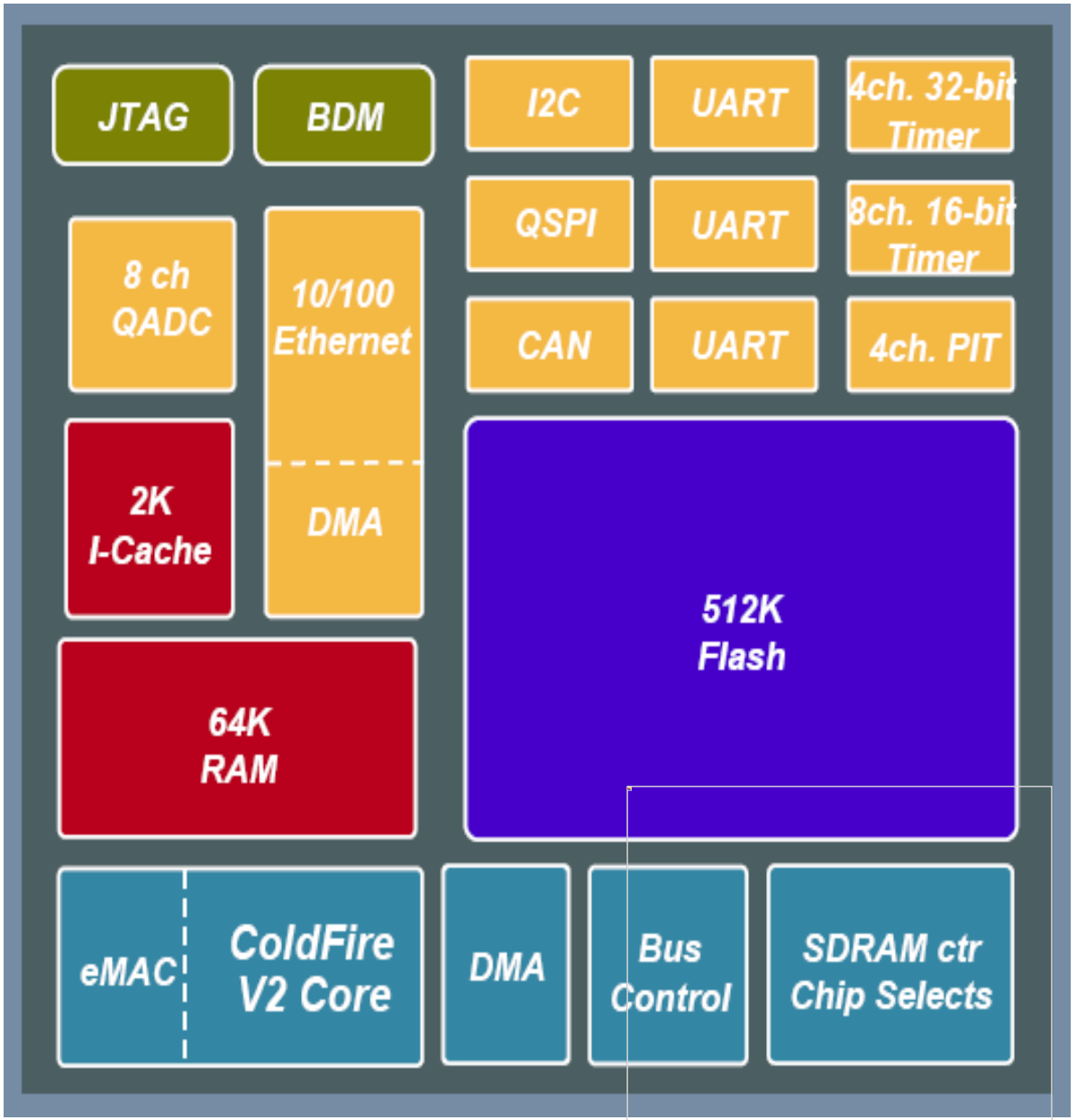
Kolejność bajtów w pamięci (5)

Jak rozpoznać architekturę procesora oraz rozkład bajtów w pamięci?

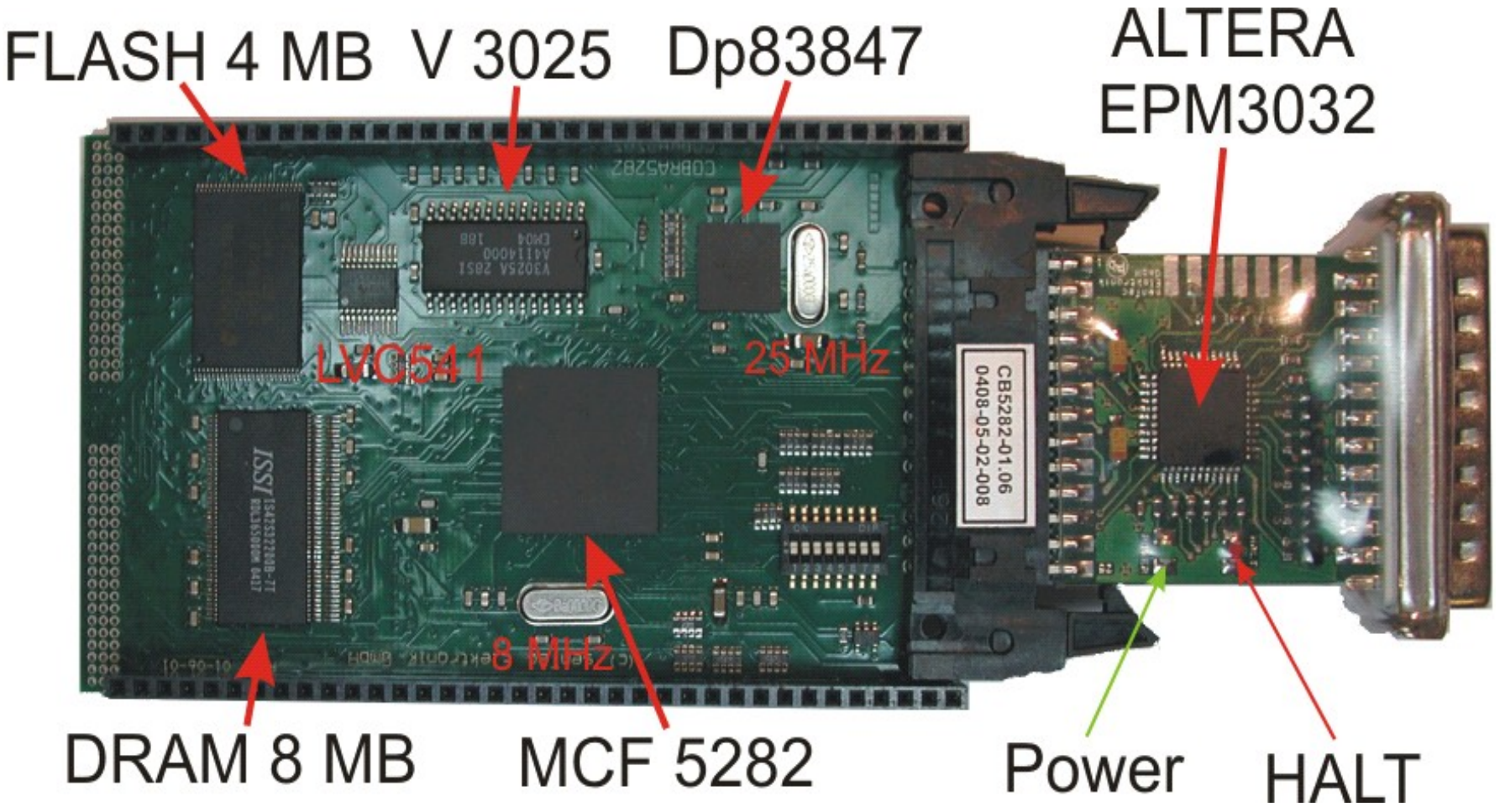
```
#define LITTLE_ENDIAN   0
#define BIG_ENDIAN      1

int machineEndianness()
{
   long int i = 1;                          /* 32 bit = 0x0000.0001 */
   const char *p = (const char *) &i;       /* wskaźnik do stałej typu char */

   if (p[0] == 1)  /* Lowest address contains the least significant byte */
      return LITTLE_ENDIAN;

   else
      return BIG_ENDIAN;
}
```

Mikrokontroler MCF5282

- Statyczny rdzeń procesora zgodny z rodziną ColdFire 2M
- Wydajność procesora:
  76 MIPS @ 80 MHz
- Zasilanie 3,3V rdzeń oraz wyprowadzenia I/O
  (5V tolerant I/O)
- Procesor wykonany w technologii 0,25 µm TSMC
- Tryby pracy z obniżonym poborem mocy (4 tryby pracy)
- Do 142 programowalnych bitowych portów I/O
- Zintegrowany generator z pętlą synchronizacji fazowej PLL
- Programowalny watch-dog



Moduł COBRA wraz z analizatorem BDM



Sprawdzian zaliczeniowy
został przesunięty z dn.
29.04.2008
na 06.05.2008
godz. 14.15-15.00