Systemy Mikroprocesorowe Czasu Rzeczywistego

Dariusz Makowski

Katedra Mikroelektroniki i

Technik Informatycznych

tel. 631 2648

dmakow@dmcs.pl

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

Sprawy formalne

1. Zaliczenie
2. Laboratorium i projekt z SMCR
3. Materiały do wykładu
4. „Odrabianie” wykładów

Literatura

Literatura obowiązkowa:

Phillip A. Laplante, “Real-Time Systems Design and Analysis”, 3rd Edition, May 2004, Wiley-IEEE Press

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

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

Literatura uzupełniająca:

S. R. Ball, “Embedded Microprocessor Systems” Butterworth-Heinemann 1996

Noty katalogowe procesorów Freescale (np. MPC5282)

Definicje podstawowe

- **Procesor (ang. Central Processing Unit)**

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

- **Mikroprocesor**

  układ cyfrowy wykonany jako pojedynczy układ scalony o wielkim stopniu integracji (VLSI) zdolny do wykonywania operacji cyfrowych według dostarczonych mu informacji
  np.: x86, Z80, 68k

- **Mikrokontroler**

  komputer wykonany w jednym układzie scalonym, używany do sterowania urządzeniami elektronicznymi. Oprócz jednostki centralnej CPU posiada zintegrowane pamięci oraz urządzenia peryferyjne, np.: Intel 80C51, Atmel Atmega128, Freescale MCF5282

System czasu rzeczywistego

Systemem czasu rzeczywistego (ang. real-time system) nazywamy system, który musi wykonać określone zadania w ściśle określonym czasie.

Poprawność pracy systemu czasu rzeczywistego zależy zarówno od wygenerowanych sygnałów wyjściowych jak i spełnionych zależności czasowych

System, który nie spełnia jednego lub większej liczby wymagań określonych w specyfikacji nazywany jest **systemem niesprawnym**

System czasu rzeczywistego

1. Jak szybki musi być układ przetwarzający dane ?
2. Ile sygnałów można przetwarzać jednocześnie ?
3. Obsługa sytuacji wyjątkowych ?

Przykładowe systemy sterujące:

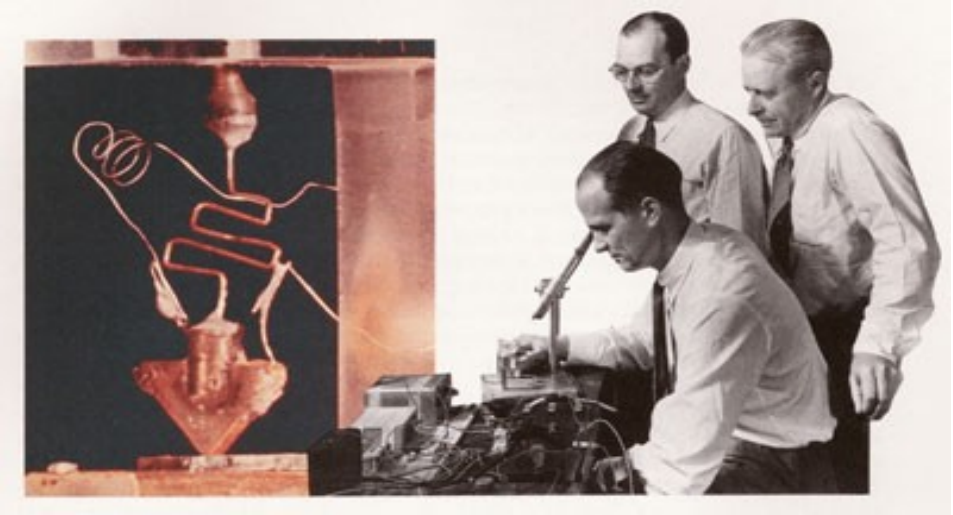
- System realizujący “internetową” sprzedaż biletów lotniczych ?
- System kontrolujący trajektorię lotu samolotu pasażerskiego ?
- System sterujący reaktorem jądrowym ?
- System sterujący rakieta ziemia-powietrze ?
- System władzy sądowniczej ?

Które z nich są systemami czasu rzeczywistego?

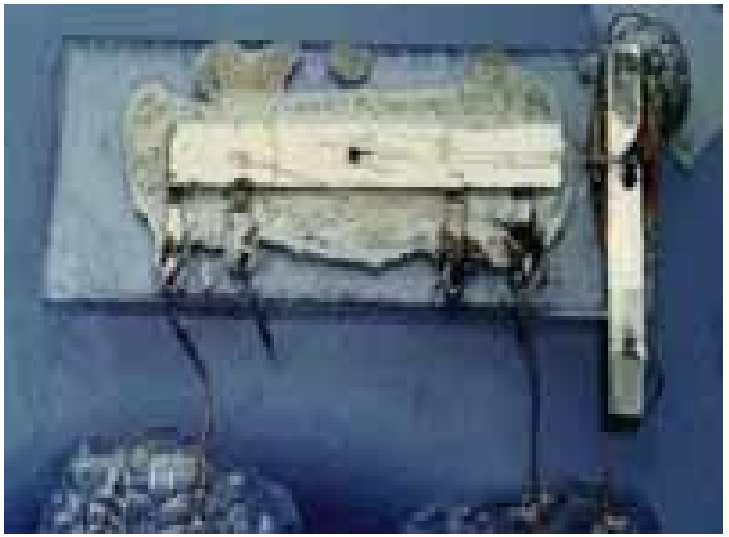
Historia mikroprocesorów (1)

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

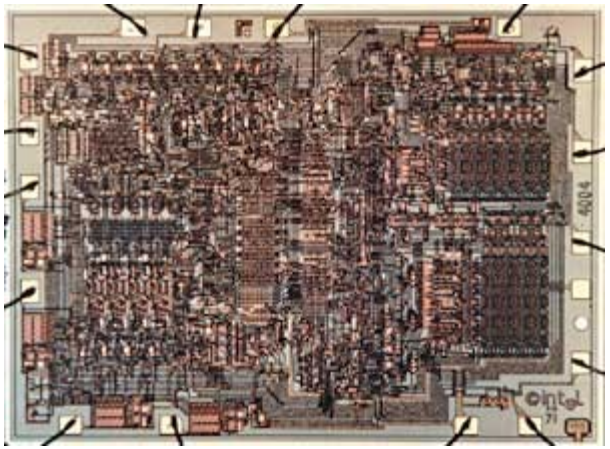
1967 – Laboratorium Fairchild oferuje pierwszą pamięć nieulotną ROM (64 bity)

1969 – Noyce i Moore opuszczają laboratorium Fairchild, powstaje niewielka firma INTEL. INTEL produkuje pamięci SRAM (64 bit).

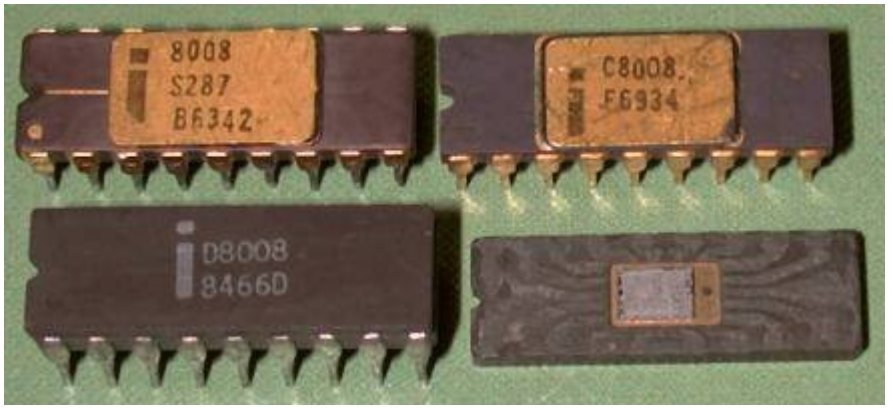
Historia mikroprocesorów (2)

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, 3200 tranzystorów. INTEL wznawia pracę nad procesorami.



Zdjęcie 4-bitowego procesora INTEL 4004



8-bitowe procesory INTEL-a

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

Historia mikroprocesorów (3)

1974 – INTEL wprowadza na rynek ulepszona wersję 8008, procesor Intel 8080. Były pracownik INTELa zakłada firmę Zilog, Motorola oferuje 8-bitowy procesor MC 6800.

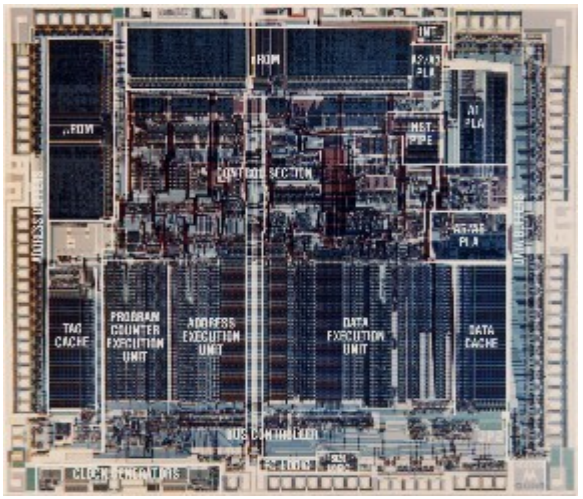
1976 – Zilog oferuję procesor Z80, INTEL pierwsza wersje procesora 8048.

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

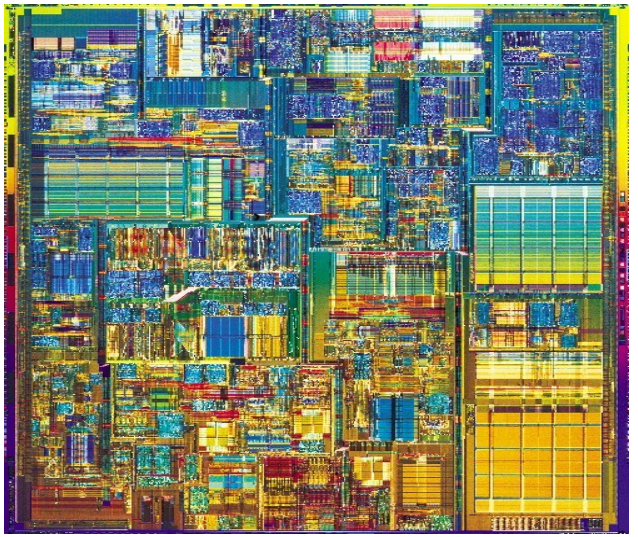
1979 – Motorola oferuje 16-bitowy procesor 68000.

1980 – Motorola wprowadza nowy 32-bitowy procesor 68020, 200.000 tranzystorów.

.....



Motorola 68020



Intel, Pentium 4 Northwood

Podział komputerów

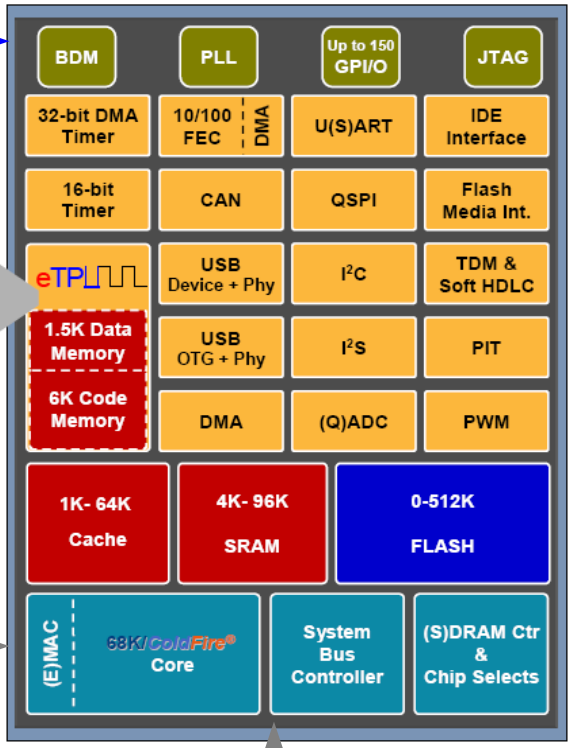
Mikrokomputer:

- **stacjonarny (desktop, Personal Computer)** – urządzenie wyposażone zwykle w silny procesor pracujący pod kontrolą systemu operacyjnego. Funkcjonalność urządzenia zależy głównie od posiadanego oprogramowania
- **wbudowany (embedded)** – komputer, maszyna, sterownik przeznaczony do realizacji ściśle określonego zadania, np. sterowanie pralką automatyczną. Komputer odpowiedzialny jest za zarządzanie urządzeniem oraz zapewnia obsługę interfejsu użytkownika.

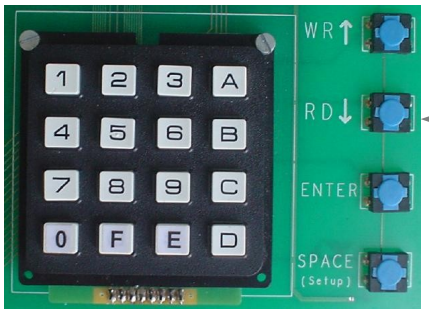
Komputer wbudowany (embedded computer)

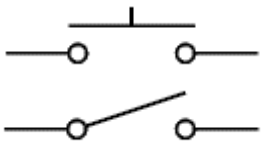
Komunikacja z komputerem zewnętrznym

Czujniki, np. czujnik temperatury, obrotów, itd



Kamera

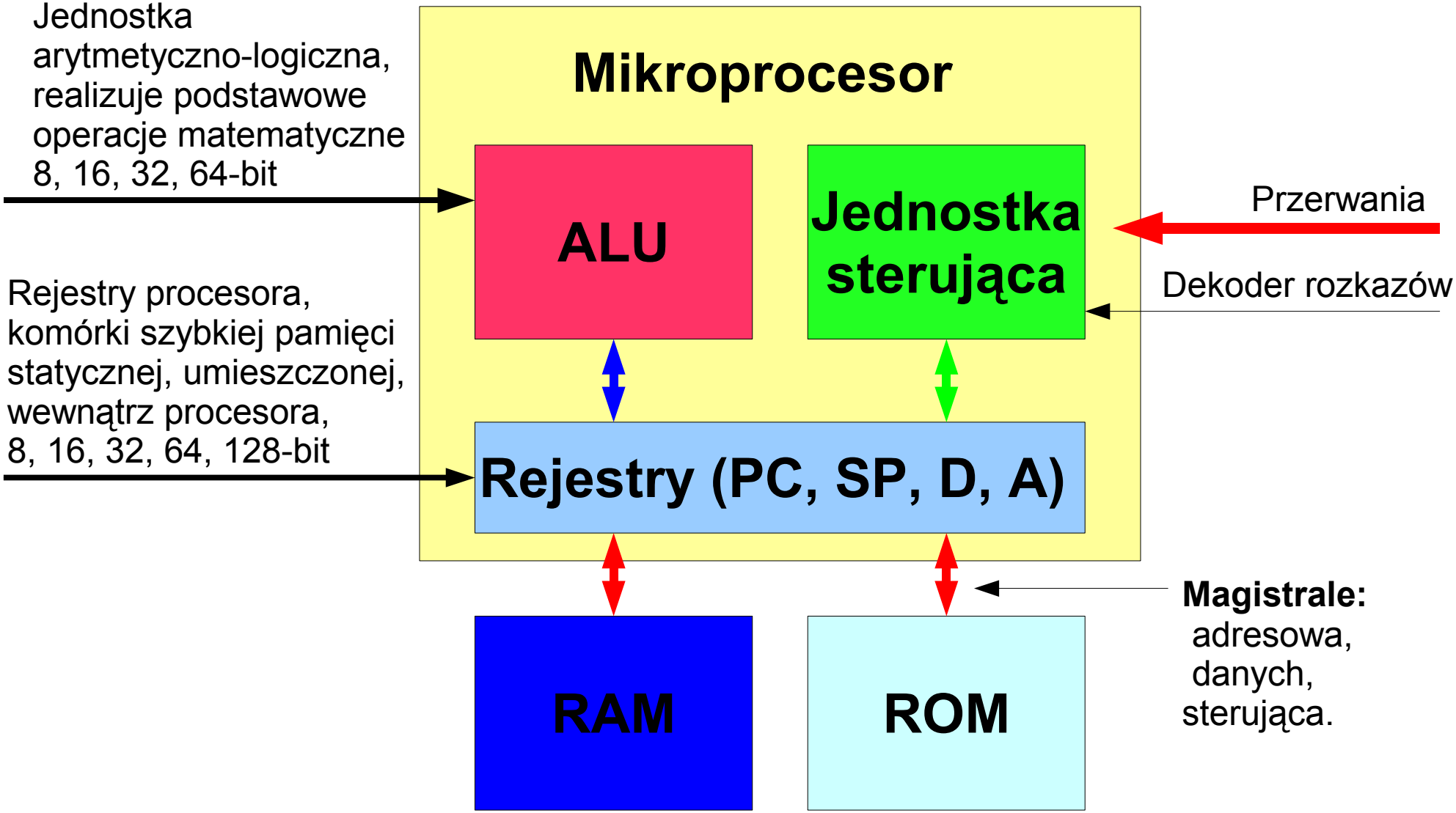




Elementy wykonawcze, np. silniki, przekaźniki

Mikroprocesor

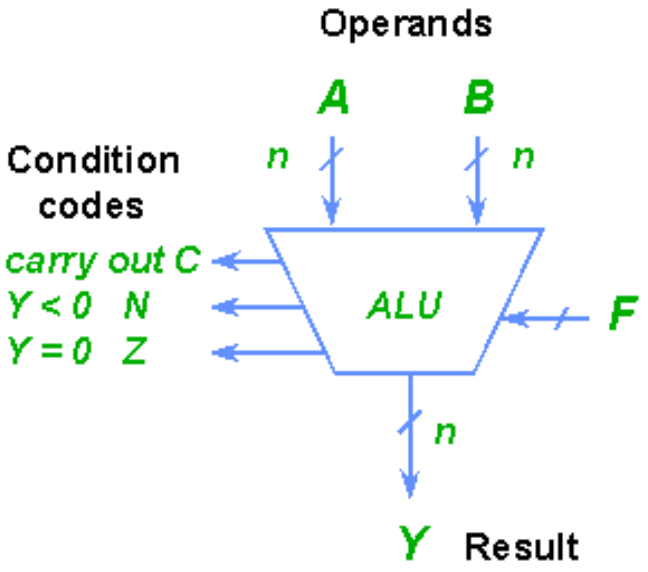
Mikroprocesor to układ cyfrowy wykonany jako pojedynczy układ scalony o wielkim stopniu integracji zdolny do wykonywania operacji cyfrowych według dostarczonych mu instrukcji.



Jednostka arytmetyczno-logiczna

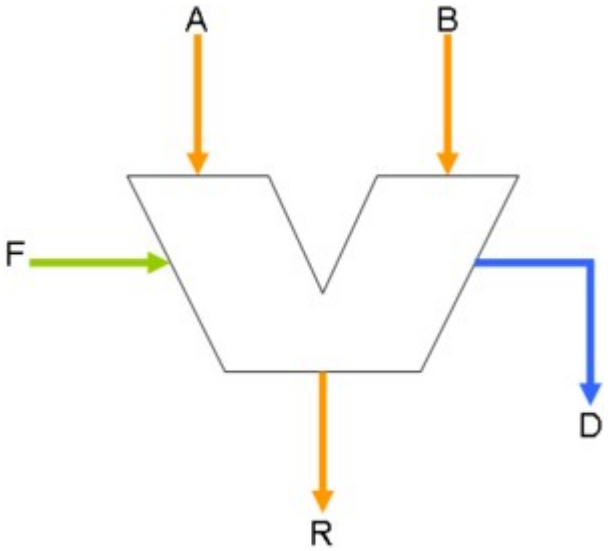
Jednostka arytmetyczno-logiczna wykorzystywana jest do wykonywania:

- 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

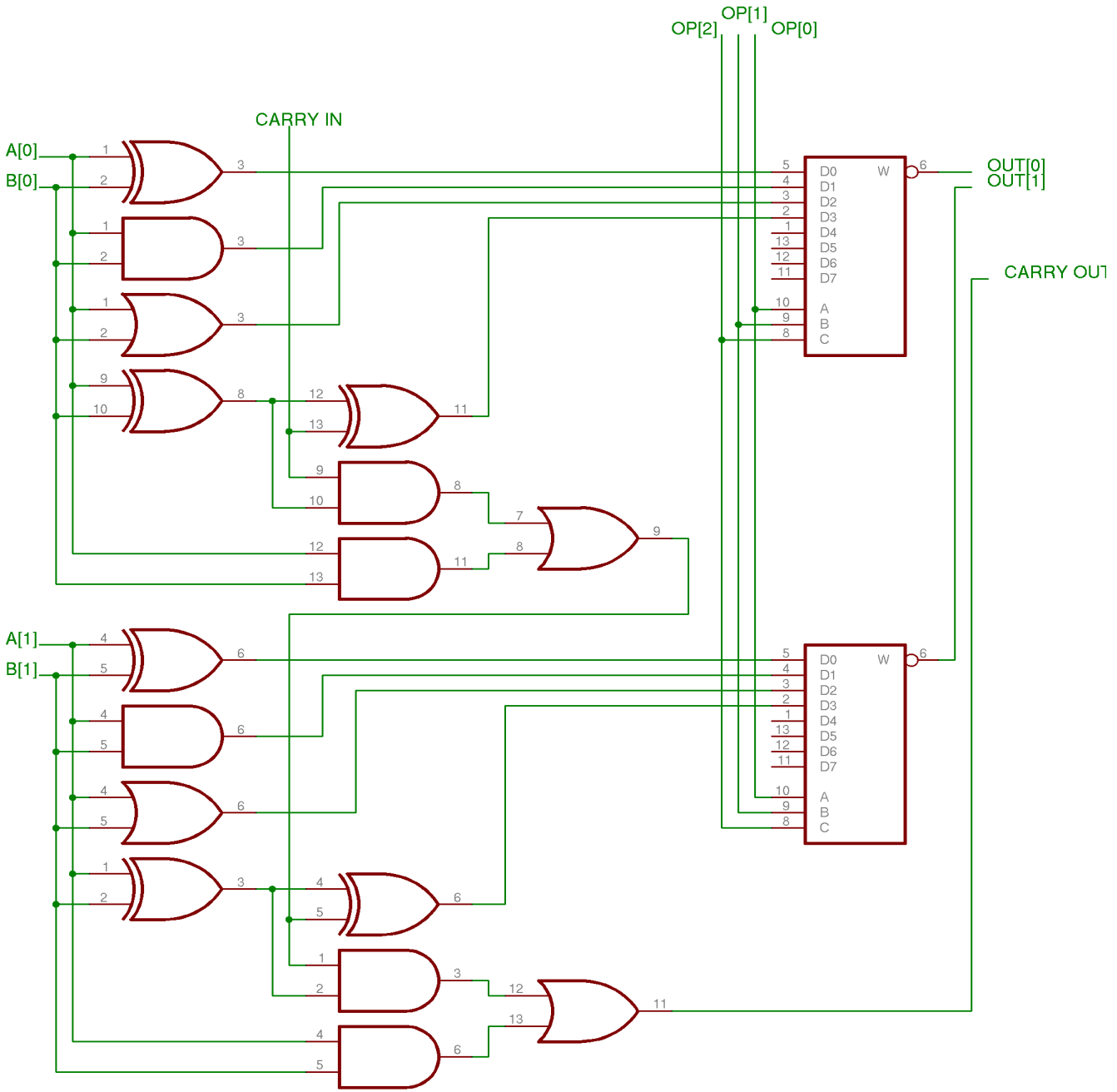


Realizowane operacje:

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

Inne możliwe operacje:

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



Architektura procesora (1)

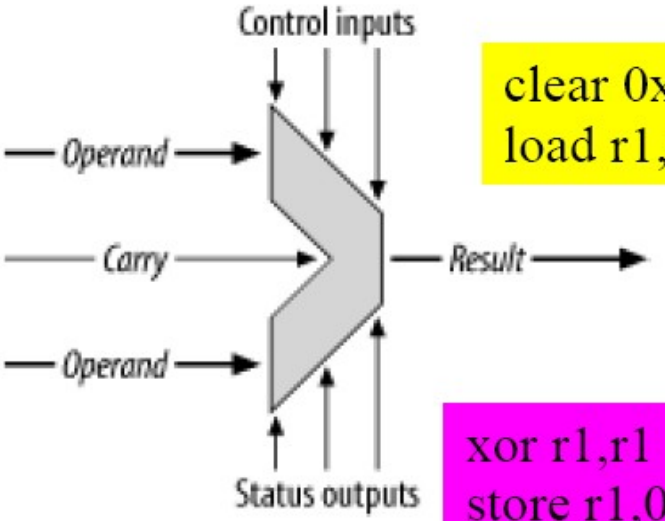
Architektura procesora określa najważniejszych z punktu widzenia budowy i funkcjonalności cechy procesora.

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.

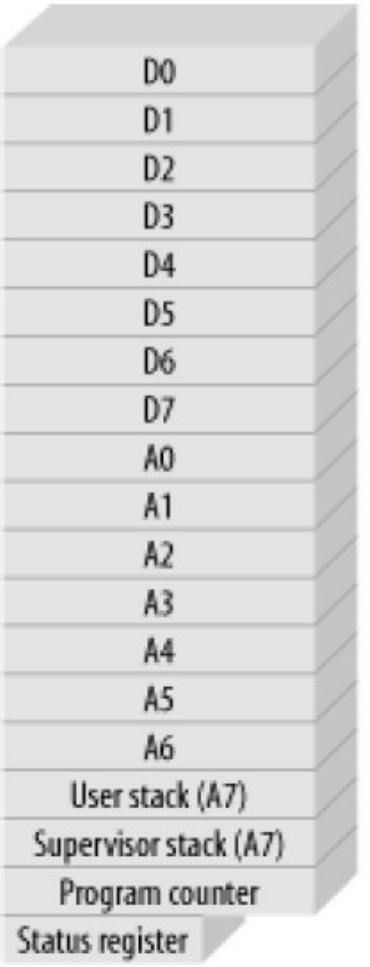
Architektura procesora (2)

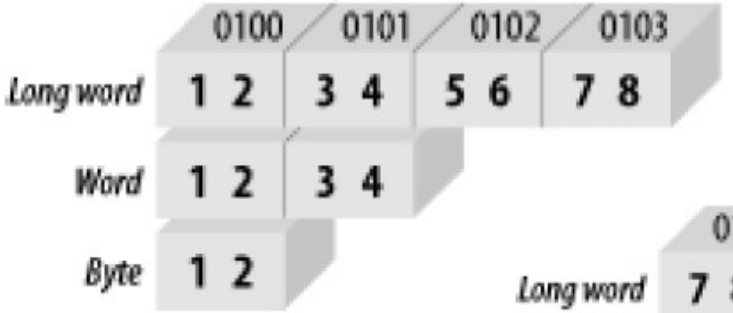
- ALU
  - lista operacji
- zestaw rejestrów
  - A, I, PC, SR, SP,...
- lista rozkazów
  - CISC, RISC
- tryby adresowania
  - R, A, I,...
- przerwania
  - hardware, software
- big/little endian

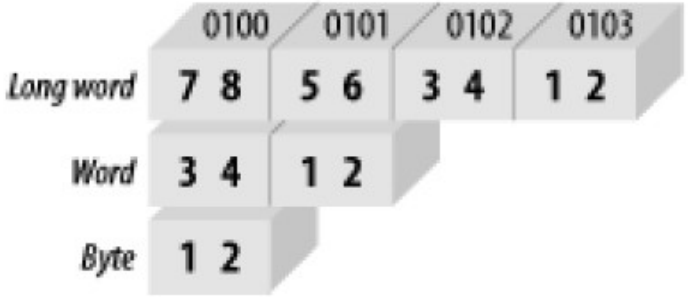



```
clear 0x1000
load r1,#5
```

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







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
- **M68000**
- PDP-11
- 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:

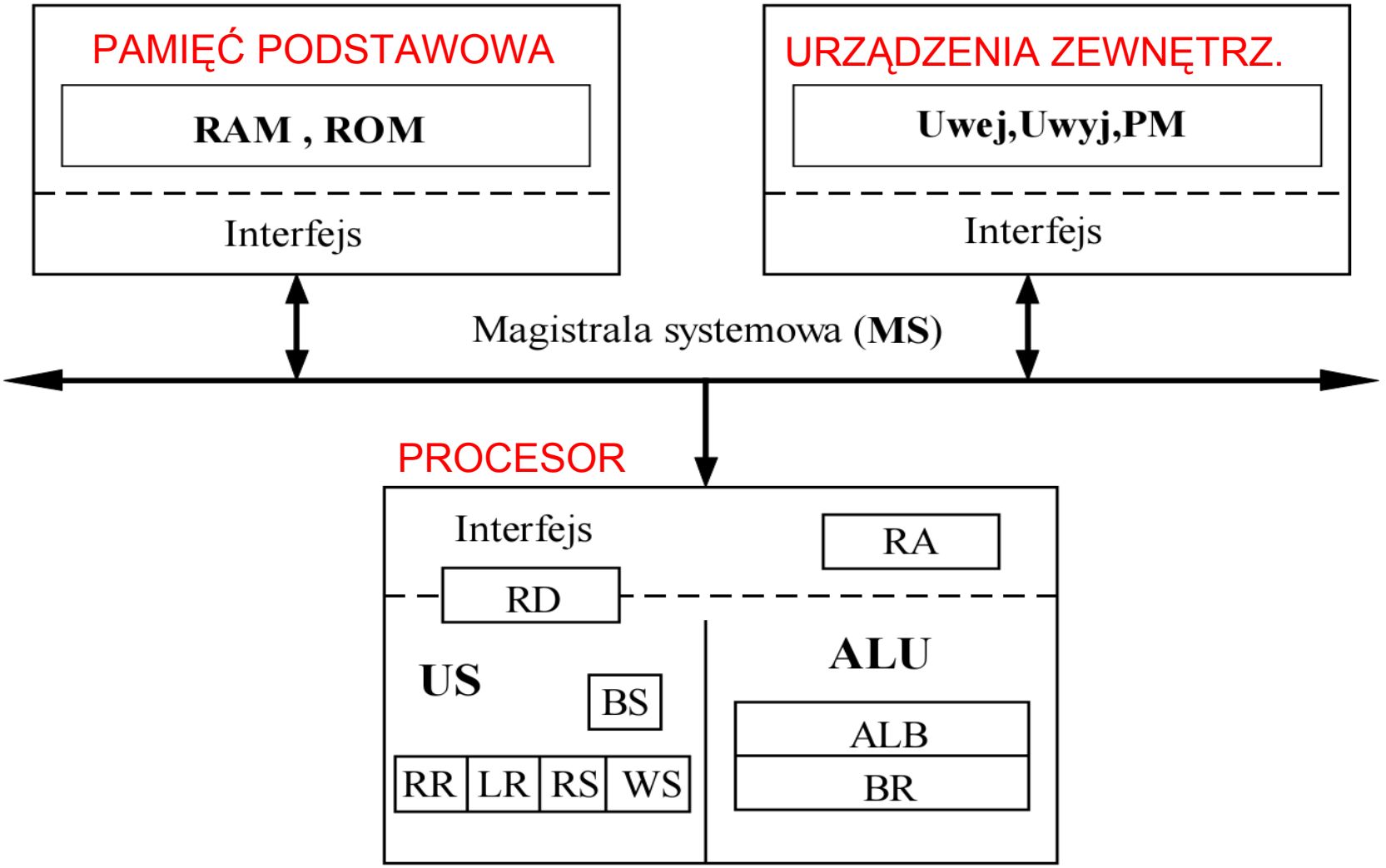
- IBM 801
- PowerPC
- MIPS
- Alpha
- ARM
- Motorola 88000
- ColdFire
- SPARC
- PA-RISC
- Atmel_AVR

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.

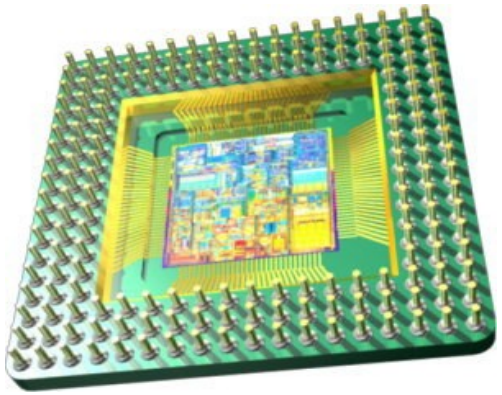
Architektura systemu komputerowego

Architektura polega na ścisłym podziale komputera na trzy podstawowe części:

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



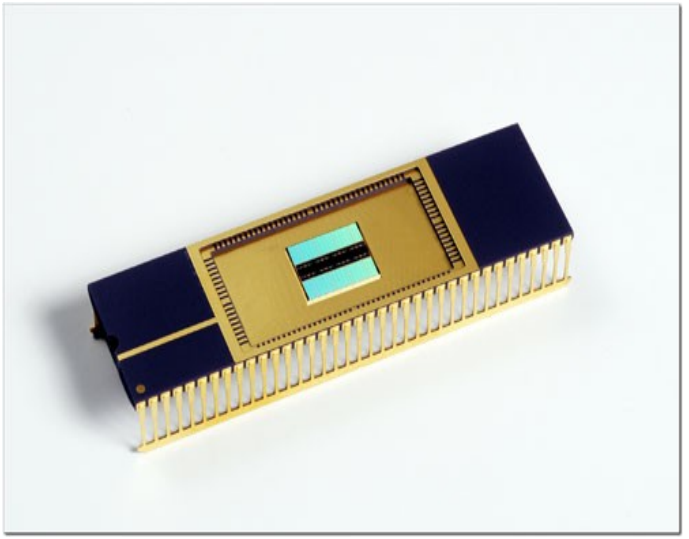
Magistrale komputera



Magistrala adresowa

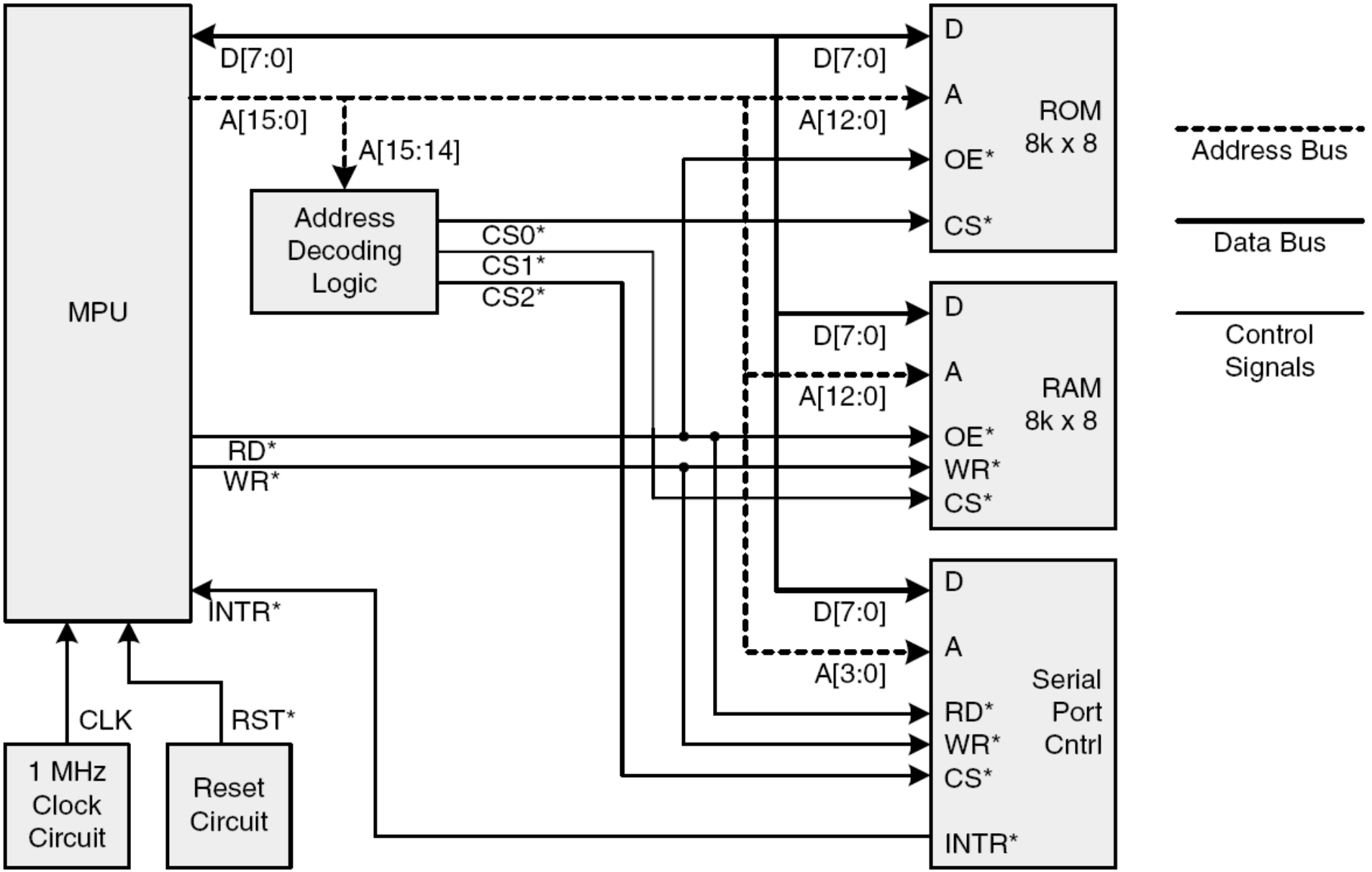
Magistrala danych

Magistrala sterująca



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

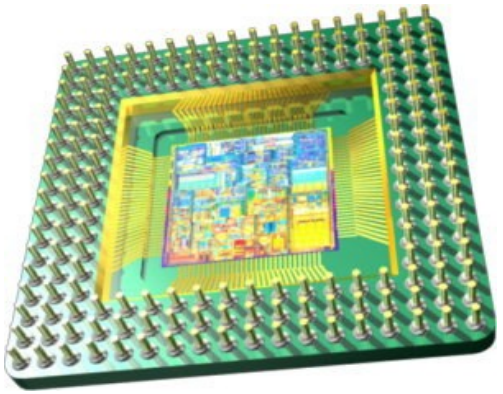
Przykładowy komputer 8-bitowy



Architektura von Neumanna

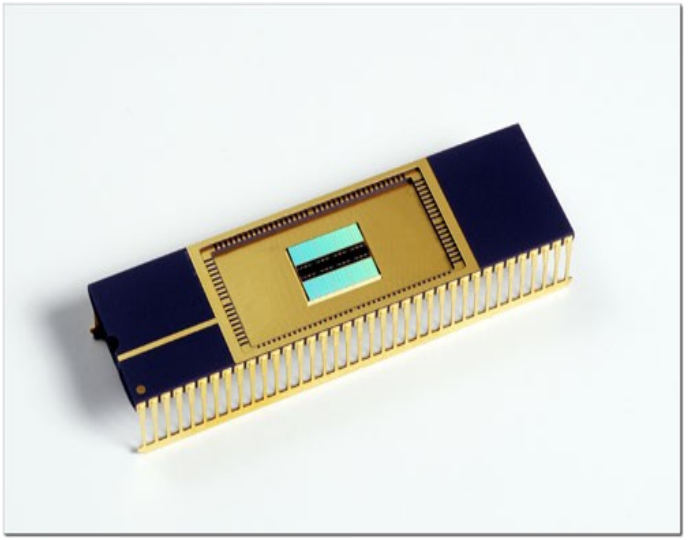
Cechy architektury von Neumanna:

- rozkazy i dane przechowywane są w tej samej pamięci,
- nie da się 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

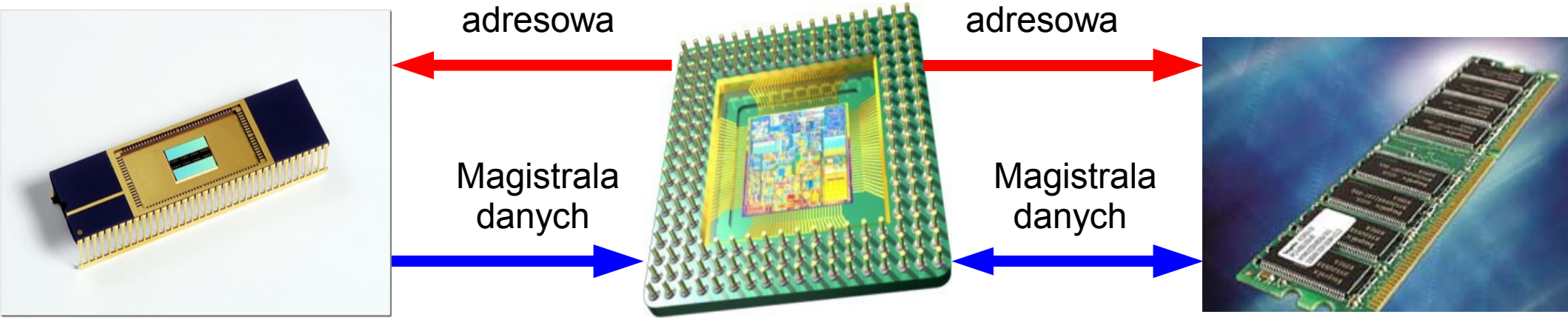


Architektura Harwardzka

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.

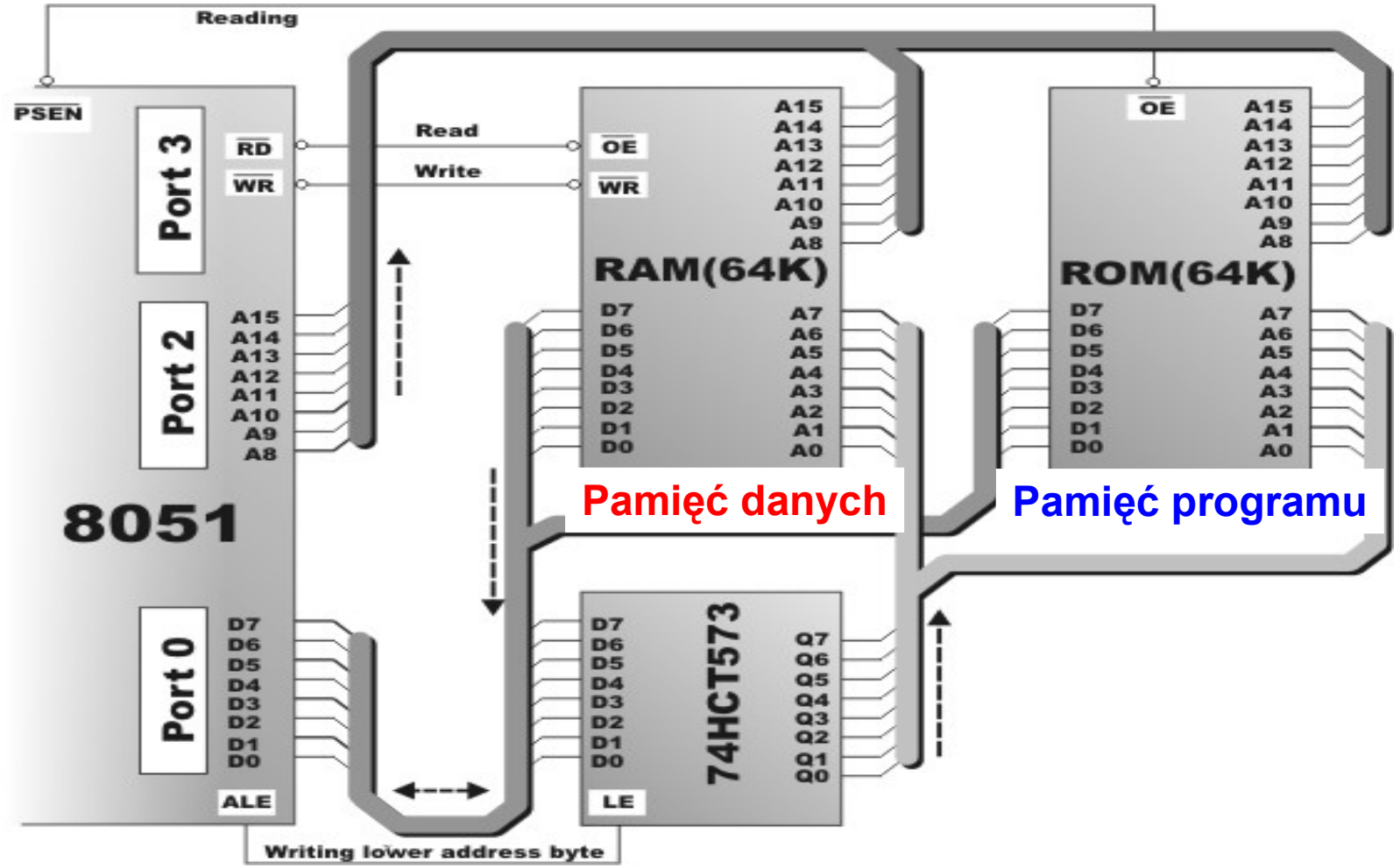
Cechy architektury Harwardzkiej:

- 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.



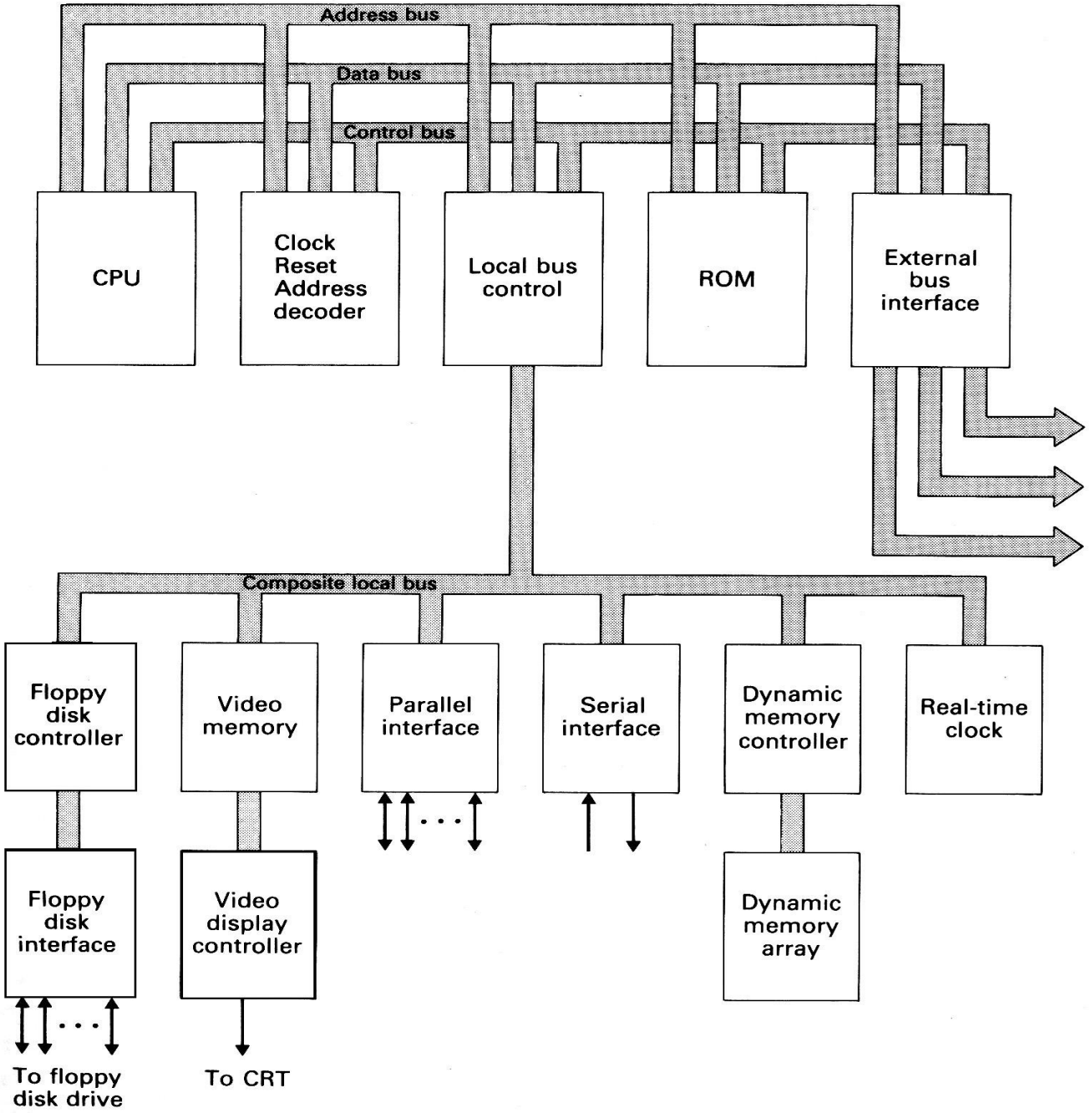
Zmodyfikowana architektura harwardzka

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.



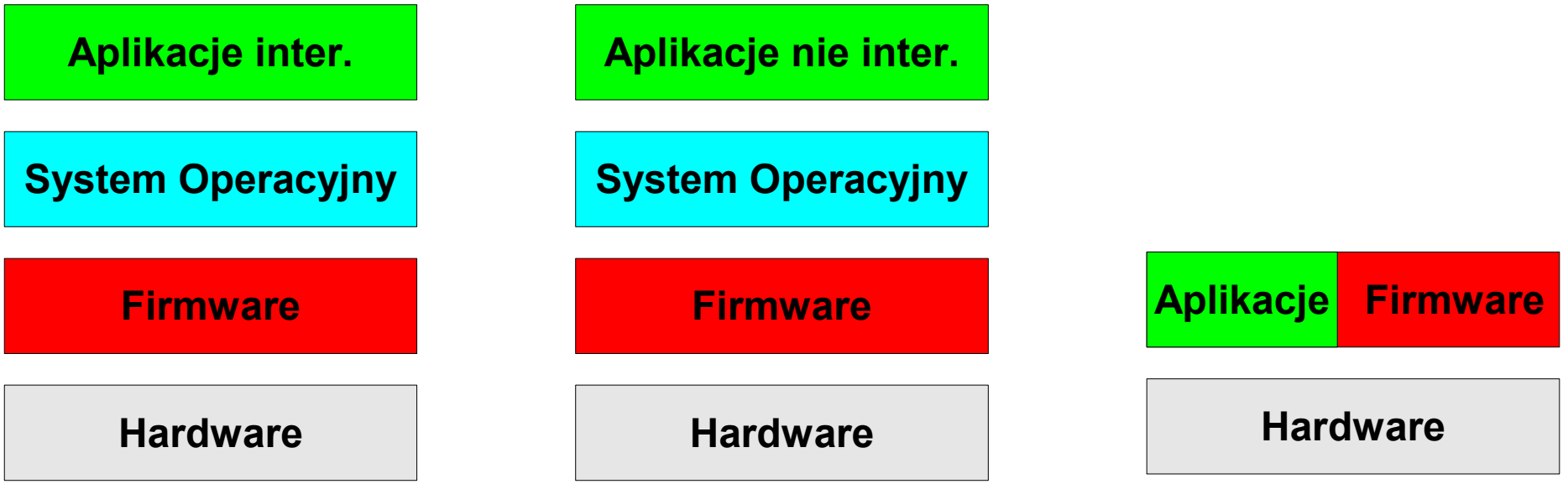
Przykład mikrokontrolera z rodziny 8051 wraz z zewnętrznymi pamięciami

Komputer uniwersalny (1)



ISA
PCI
PCIe
AGP

Oprogramowanie mikrokomputerów



Komputer uniwersalny

Złożony komputer wbudowany

Prosty komputer wbudowany

- 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).

Systemy operacyjne dla urządzeń wbudowanych (1)

Dlaczego potrzebny jest system operacyjny ?

- Rosnący poziom złożoności zadań wykonywanych przez współczesne systemy wbudowane
- Implementacje skomplikowanych algorytmów przetwarzania danych
- Programowanie współczesnych urządzeń peryferyjnych ze względu na złożoność sprzętu stało się zadaniem trudnym i czasochłonnym. Brak zunifikowanych interfejsów oraz różnorodność udostępnianych przez producentów funkcji sprawia, że kod źródłowy nie jest bezpośrednio przenośny nawet pomiędzy różnymi platformami tego samego producenta.

Systemy operacyjne dla urządzeń wbudowanych (2)

Systemy z procesorem wyposażonym w jednostkę zarządzania pamięcią (MMU)

odpowiednio zmodyfikowane odmiany systemu Linux, MontaVista Linux

Systemy z procesorem bez układu zarządzania pamięcią

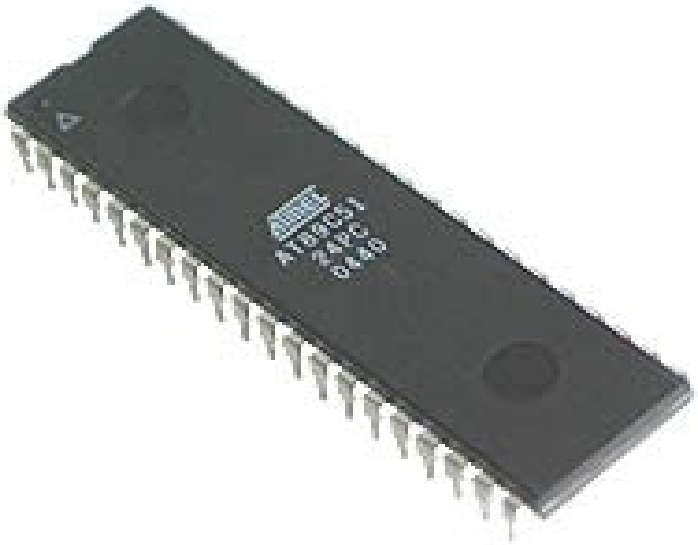
µClinux, PetaLinux, RTEMS

Systemy operacyjne dla urządzeń wbudowanych (3)

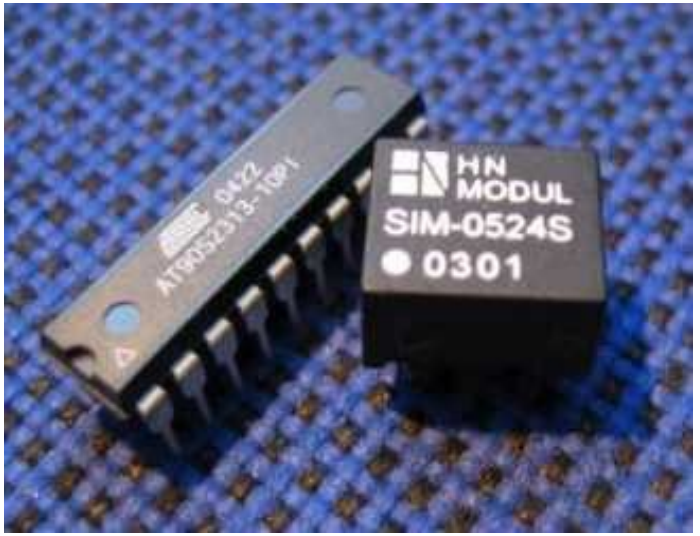
Dlaczego brak układu zarządzania pamięcią jest problemem?

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

Mikrokontrolery jednoukładowe



Atmel AT89C51, DIP 40

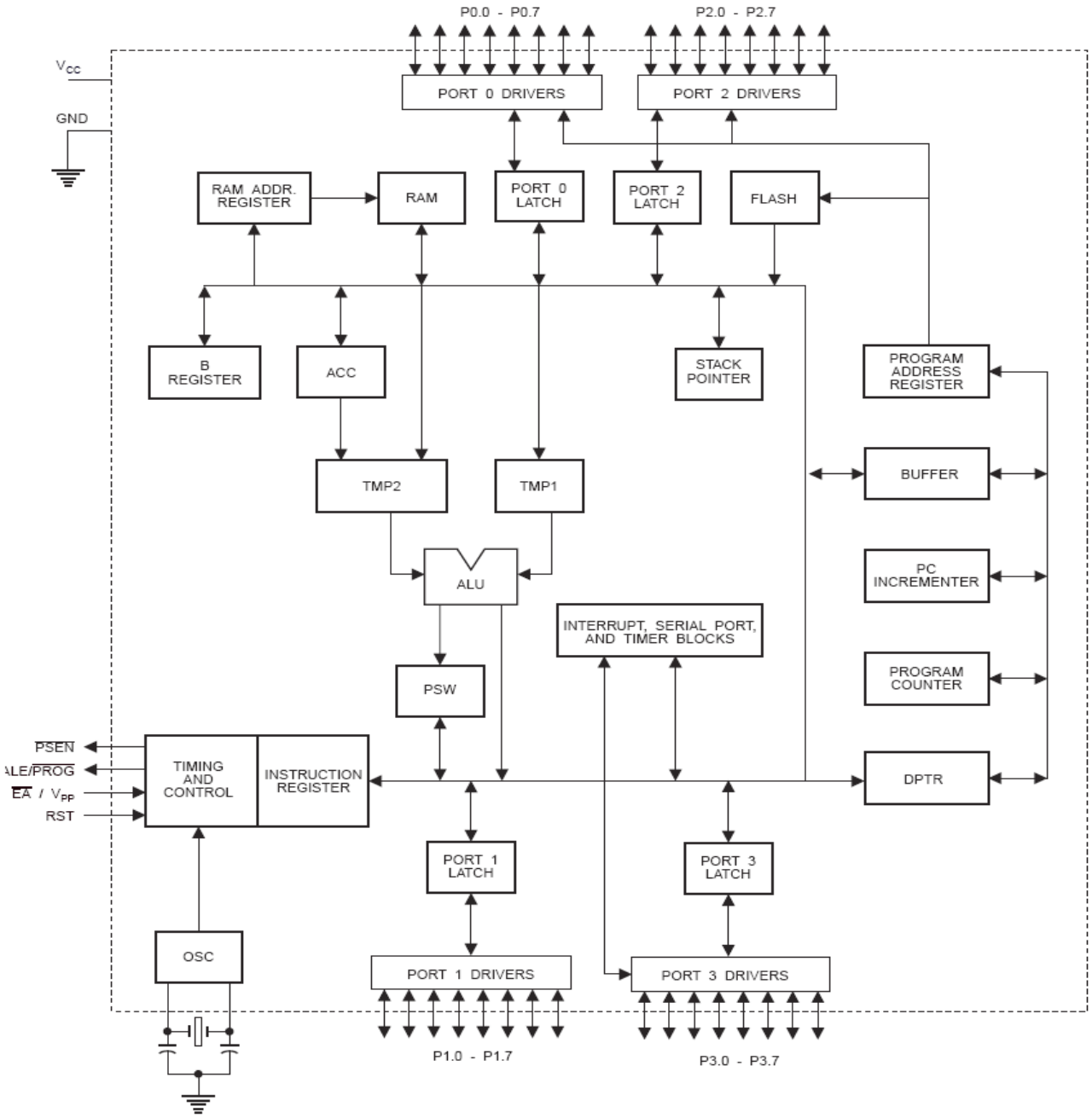


Atmel AT90S2313,
DIP 20

Mikrokontroler AT89C51

Cechy mikrokontrolera

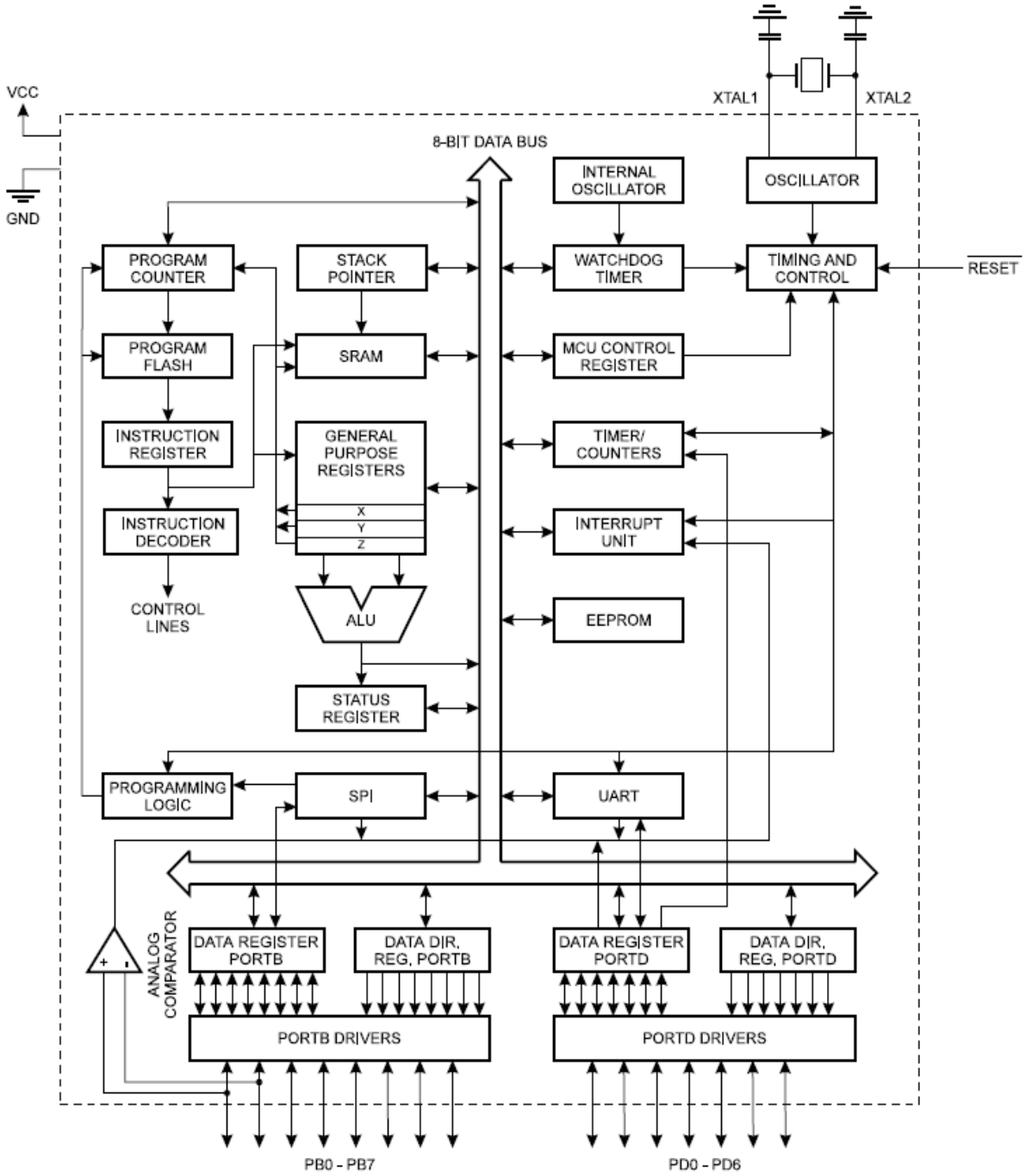
- Zgodny z architekturą '51
- Wewnętrzna pamięć FLASH 4 kB
- Wewnętrzna pamięć SRAM 128 B
- Częstotliwość taktowania 0 Hz do 24 Mhz
- UART zgodny z RS 232
- Programowalne porty I/O, 32 linie
- Dwa 16 bitowe liczniki/timery
- Sześć źrodeł przerwań



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
```

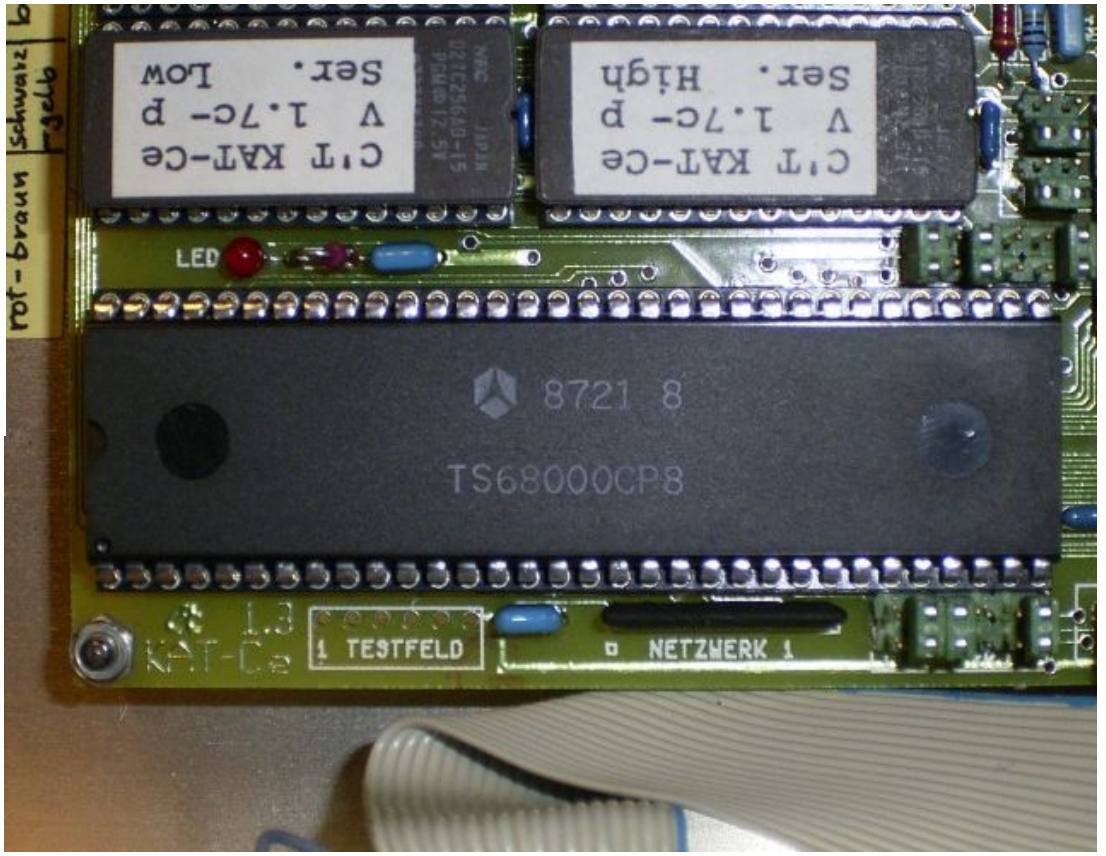
1 instrukcja = 1 clk

f = 12 Mhz => 0.16 us

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

Motorola, czy Freescale

Mikrokontroler Freescale,
MCF520x, BGA 256



Procesor Motorola, 68k
DIP 64

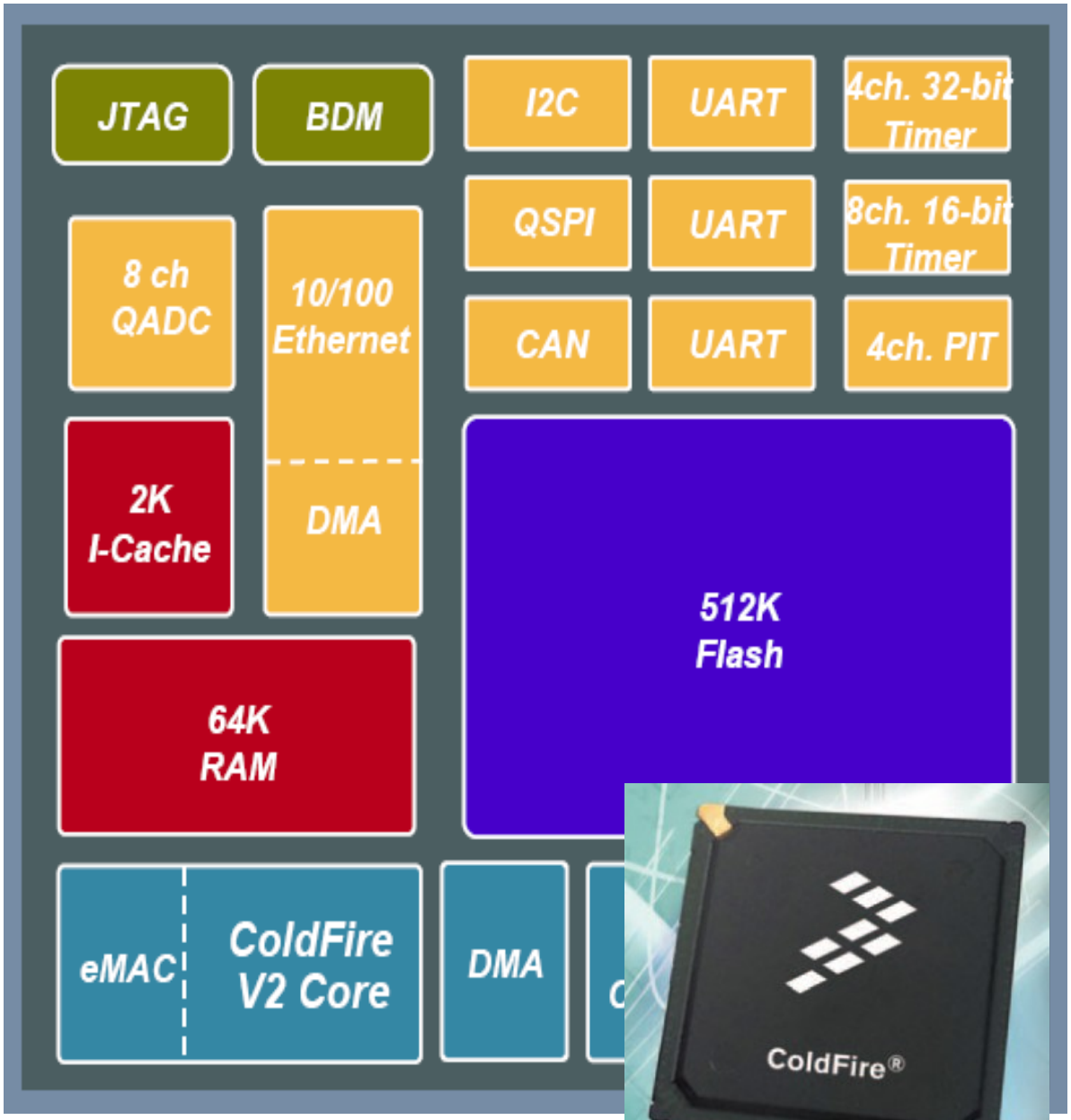
ColdFire® Embedded Controllers

MCF528x Family

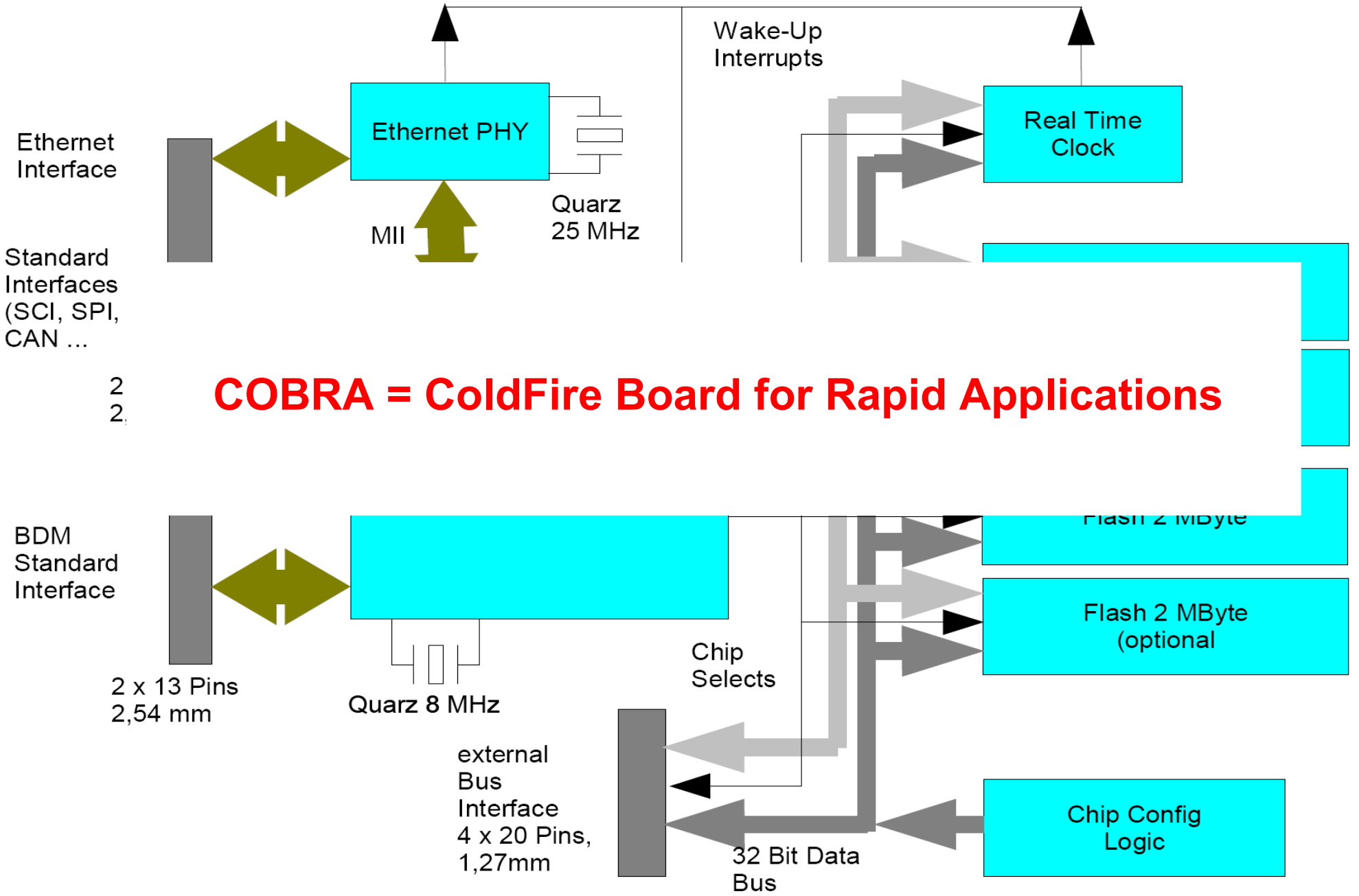
Systemy Mikroprocesorowe Czasu Rzeczywistego - laboratorium

Mikrokontroler MCF5282

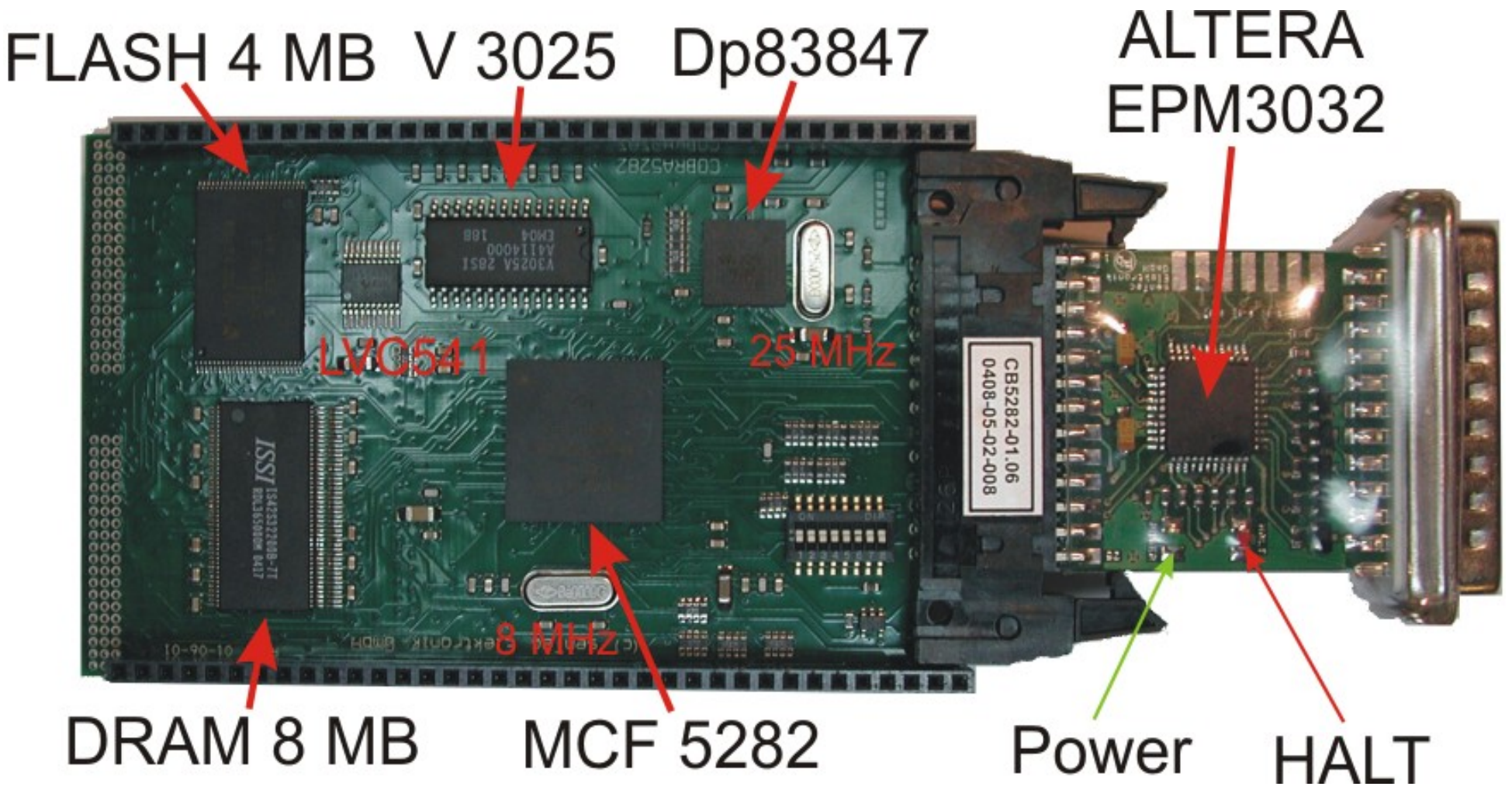
- Maksymalna częstotliwość pracy 80MHz
- 16 32-bitowych rejestrów ogólnego przeznaczenia i adresowych
- 2k pamięci cache danych lub instrukcji
- 64k pamięci RAM
- 512k pamięci Flash
- Tryby pracy z obniżonym poborem mocy (4 tryby pracy)
- Do 142 programowalnych bitowych portów I/O
- Programowalny watch-dog



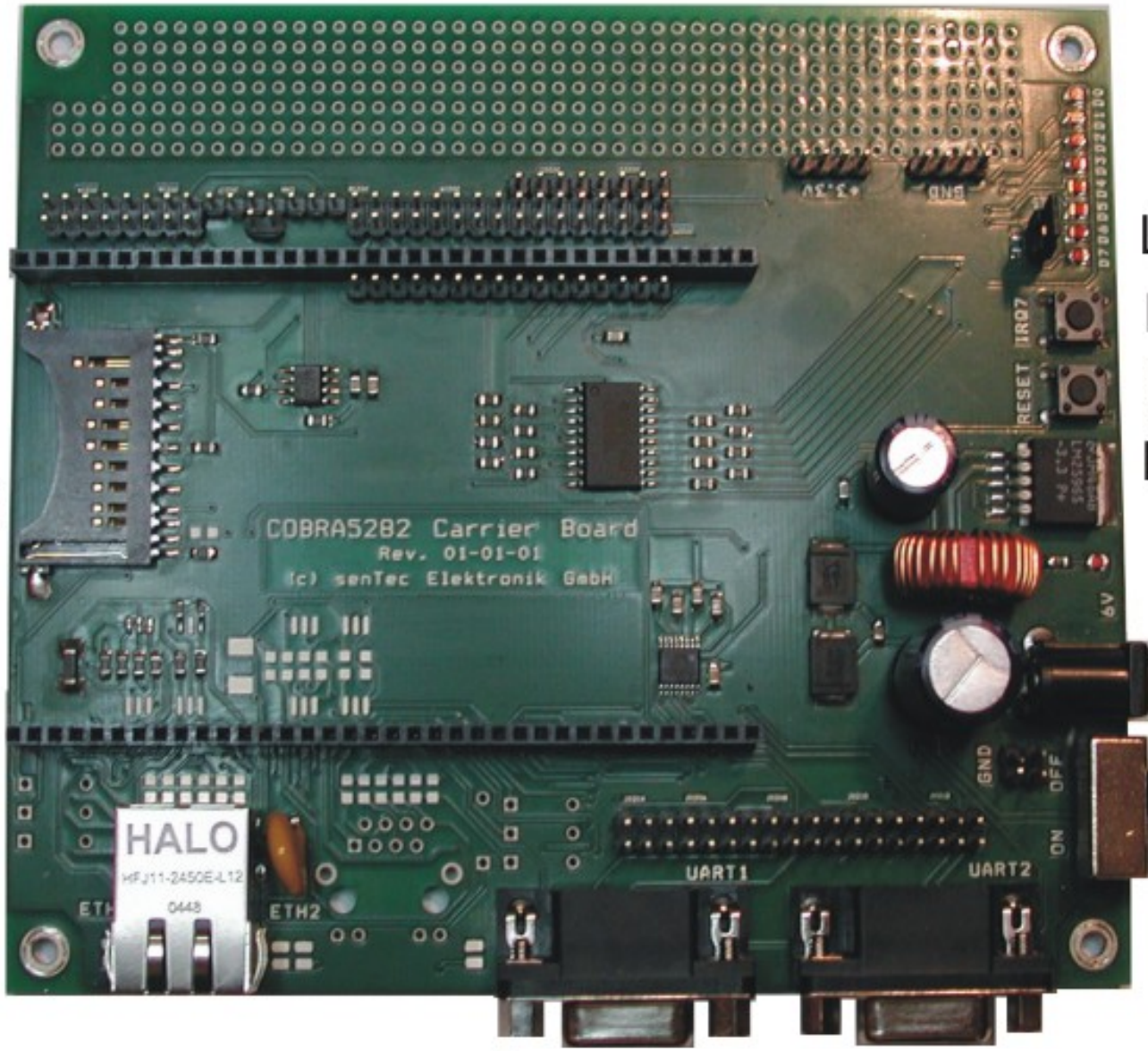
Schemat blokowy modułu COBRA



Moduł COBRA wraz z analizatorem BDM



Płytka bazowa zestawu uruchomieniowego



Gniazdo pamięci MMC/SD

8xLED

LED On/Off

IRQ7

Reset

Przetwornica +3.3 V

+5 V

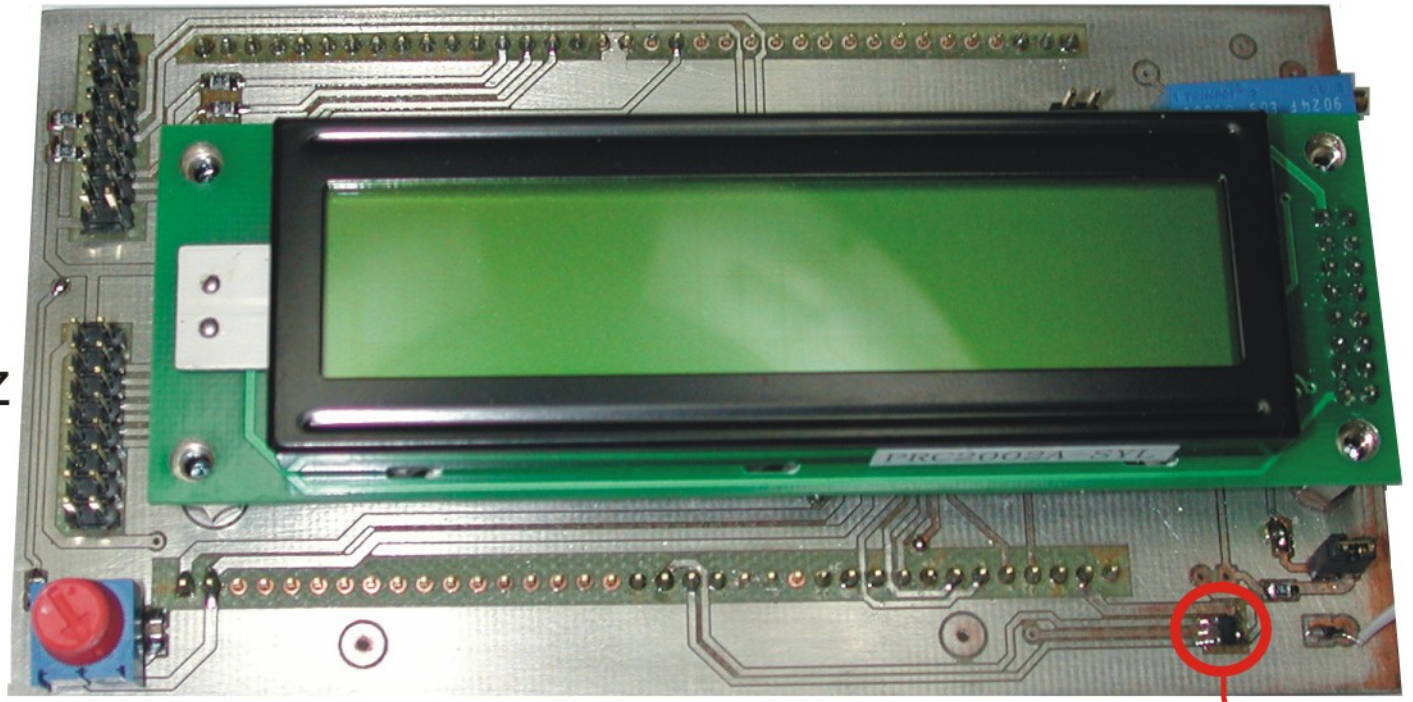
Power On/off

Ethernet 10/100

UART 0

UART 1

Płytka z dodatkowymi peryferiami



Rejestry procesora

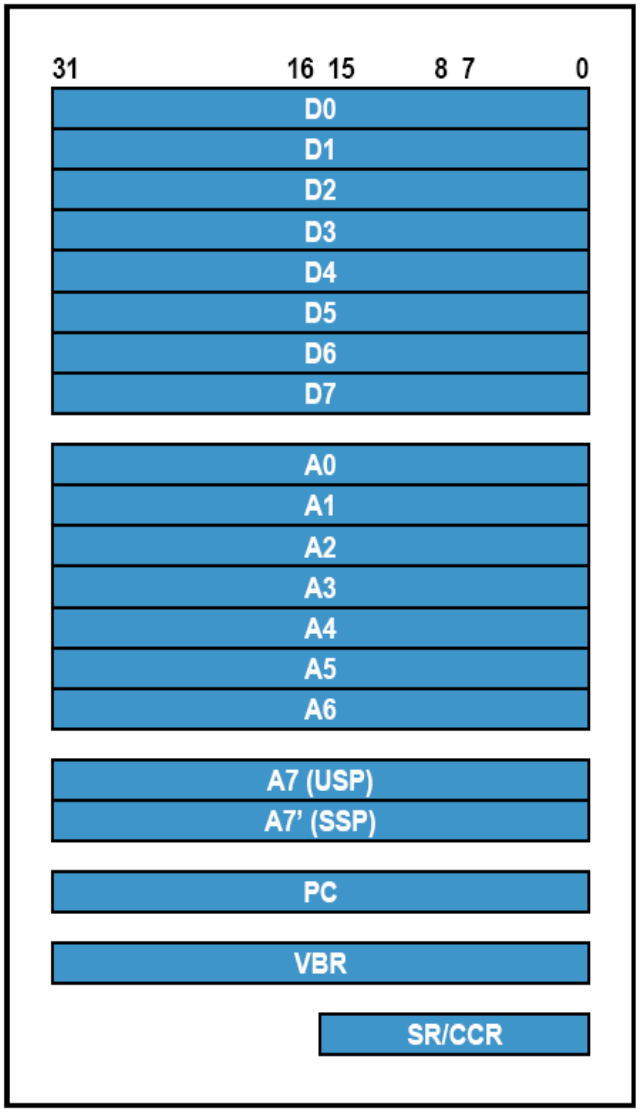
Rejestry procesora stanowią komórki wewnętrznej pamięci procesora o niewielkich rozmiarach (najczęściej 4/8/16/32/64/128 bitów) służące do przechowywania tymczasowych wyników obliczeń, adresów danych w pamięci operacyjnej, konfiguracji, itd.

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),

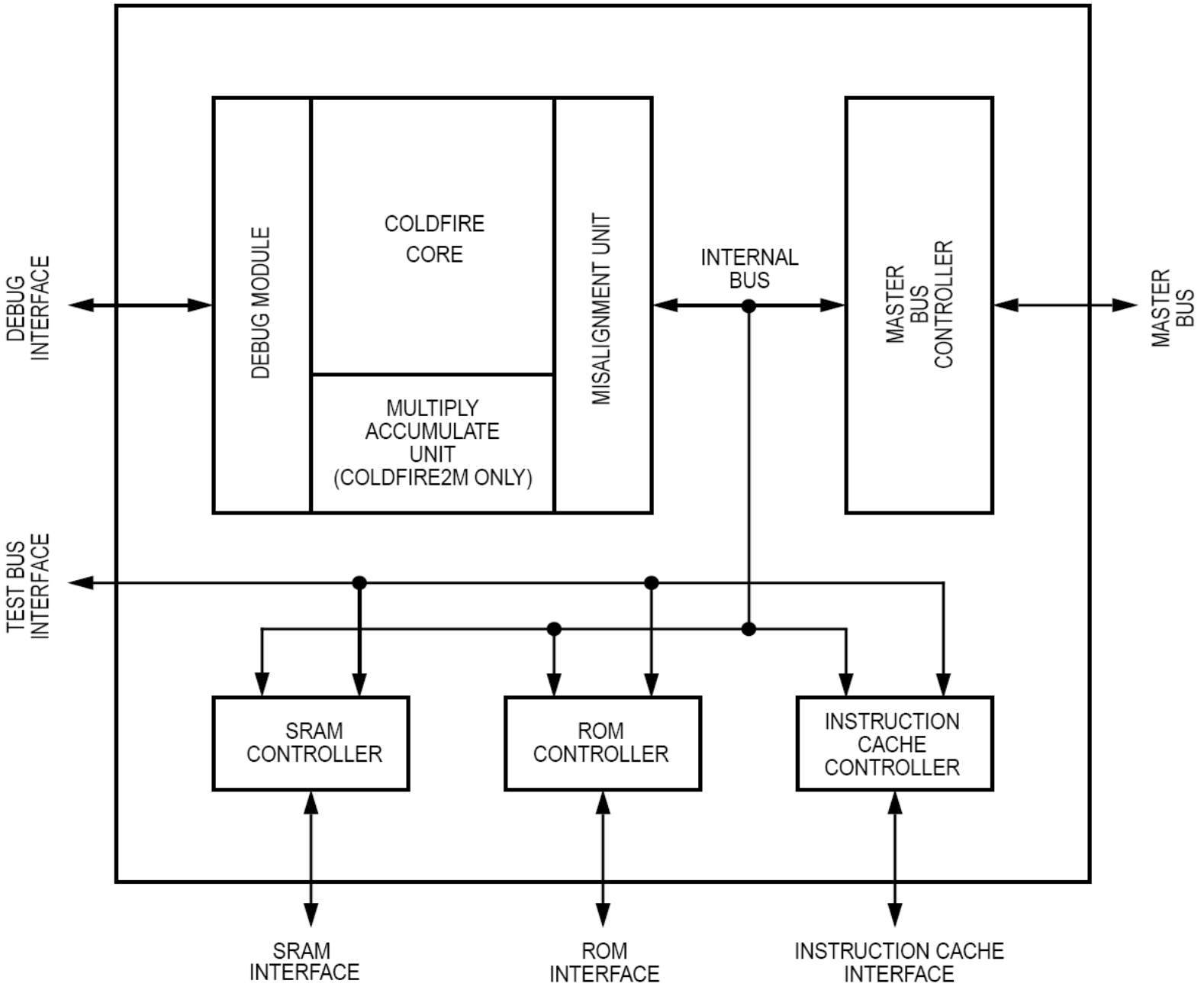


Rejestry procesora z rodziny Motorola ColdFire

Charakterystyka jądra ColdFire 2/2M

- 32-bitowa magistrala adresowa (4 GB)
- 32-bitowa magistrala danych
- Rdzeń procesora RISC o zmiennej długości rozkazów (16-, 32- oraz 48 bitowe słowa)
- Procesor posiada zoptymalizowaną pod względem wydajności listę rozkazów bazującą na procesorze rodziny Freescale (Motorola) 68k
- Lista rozkazów zoptymalizowana pod kątem języków wysokiego poziomu
- 16 rejestrów ogólnego przeznaczenia (D0 – D7, A0 – A7)
- Jednostka eMAC (Multiply Accumulate) przyśpieszająca obliczenia stałoprzecinkowe (tylko ColdFire2M)
- Tryb pracy dedykowanego użytkownika Supervisor mode umożliwiający podniesienie bezpieczeństwa pracy systemów operacyjnych
- Sterownik zewnętrznej pamięci SRAM/DRAM
- Mechanizm podręcznej pamięci Cache
- 12 trybów adresowania pamięci zgodnych z rodziną Freescale (Motorola) 68k
- Pełne wsparcie debugowania programu w czasie rzeczywistym

Architektura jądra ColdFire 2/2M

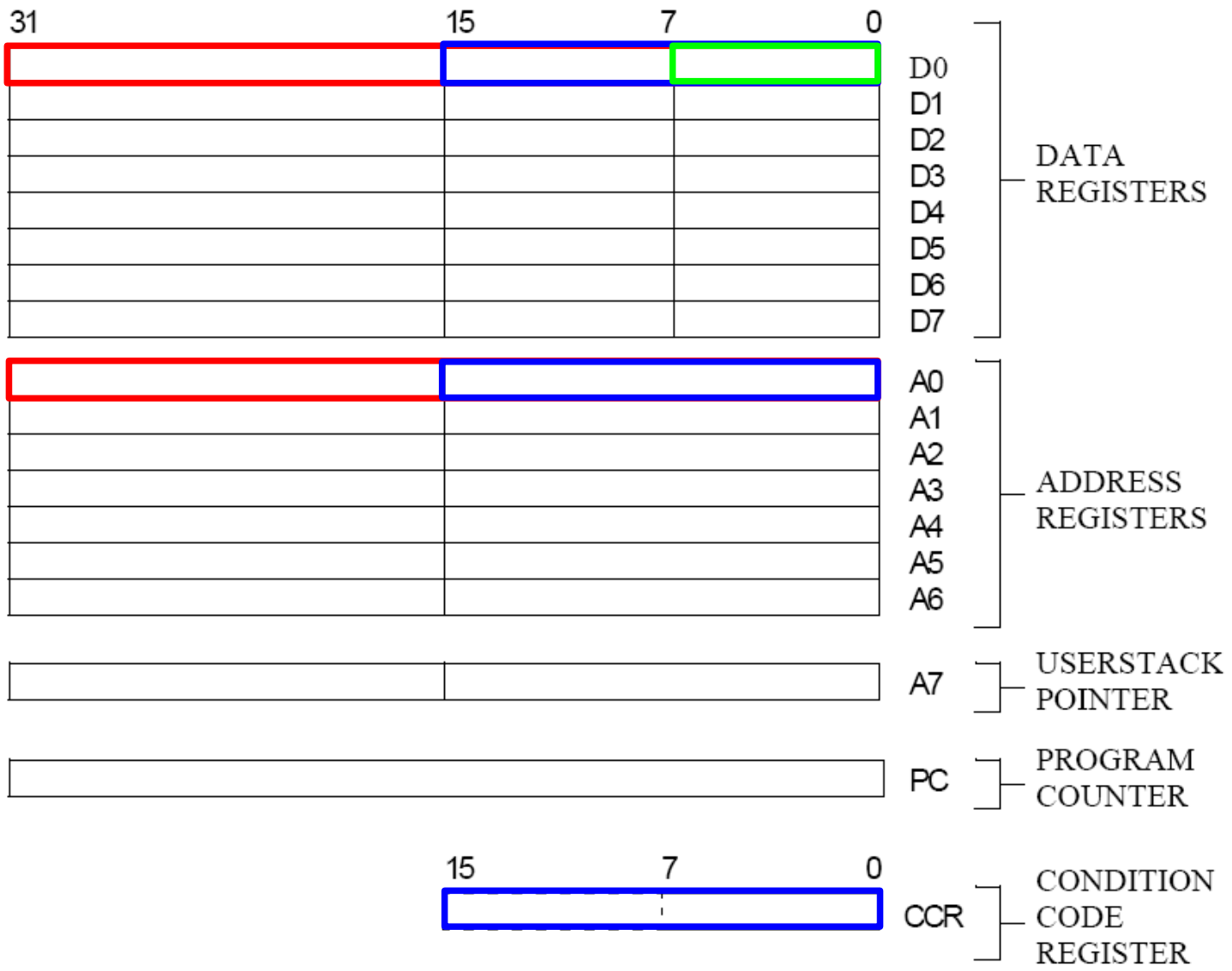


Rozszerzona lista instrukcji ColdFire V2

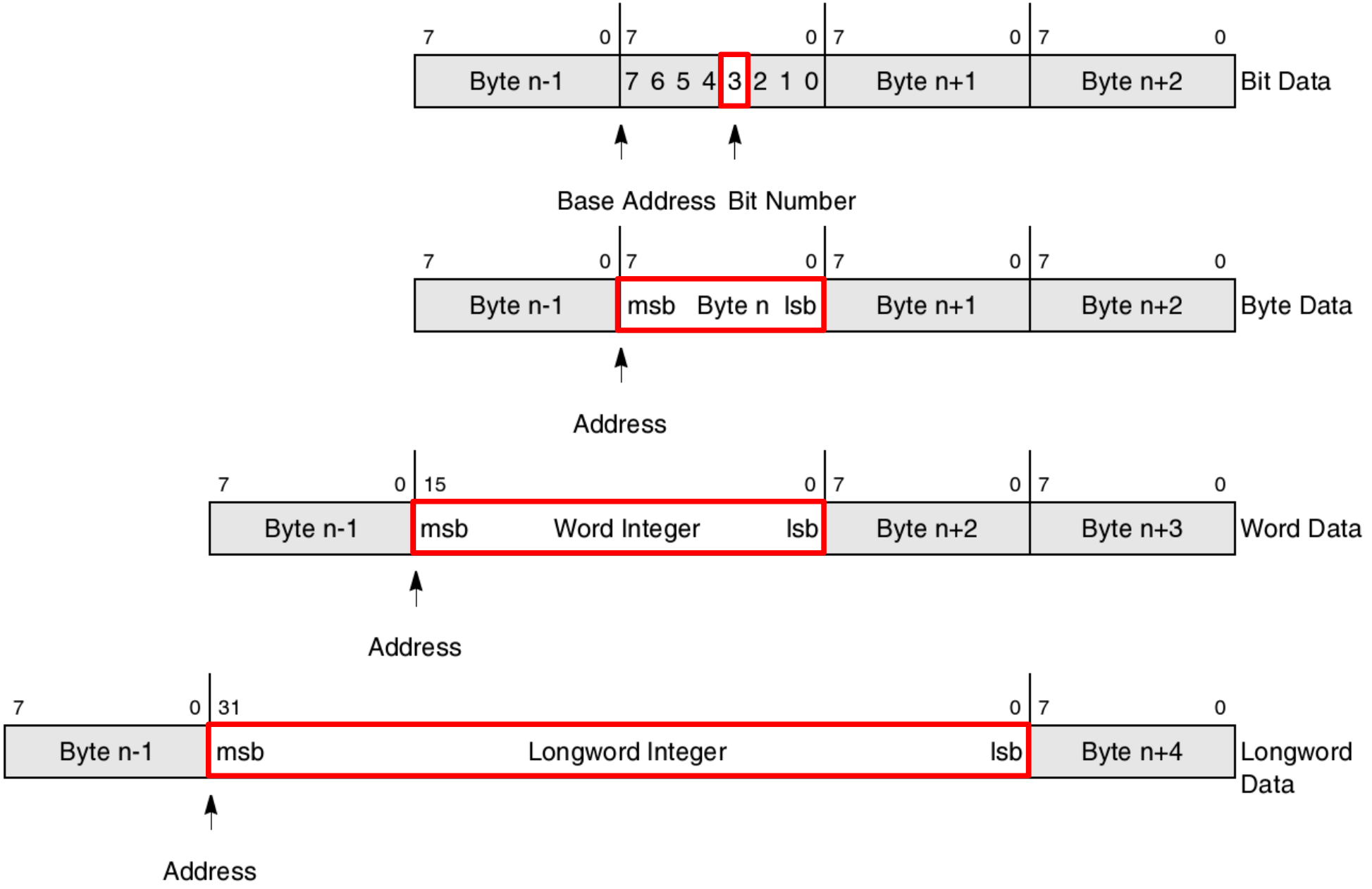
Mnemonic	Description	ISA_A	ISA_A+	ISA_B	ISA_C	FPU	MAC	EMAC	EMAC_B
ADD	Add	X	X	X	X				
ADDA	Add Address	X	X	X	X				
ADDI	Add Immediate	X	X	X	X				
ADDQ	Add Quick	X	X	X	X				
ADDX	Add with Extend	X	X	X	X				
AND	Logical AND	X	X	X	X				
ANDI	Logical AND Immediate	X	X	X	X				
ASL, ASR	Arithmetic Shift Left and Right	X	X	X	X				
Bcc.{B,W}	Branch Conditionally, Byte and Word	X	X	X	X				
BITREV	Bit Reverse		X		X				
BRA.{B,W}	Branch Always, Byte and Word	X	X	X	X				
BRA.L	Branch Always, Longword		X	X					
FF1	Find First One		X		X				
MOVE ACC to ACC	Copy Accumulator							X	X
MOVE from ACC	Move from Accumulator						X	X	X
MOVE from ACCext01	Move from Accumulator 0 and 1 Extensions							X	X
MOVE ACCext23	Move from Accumulator 2 and 3 Extensions							X	X

Model programowy procesora ColdFire

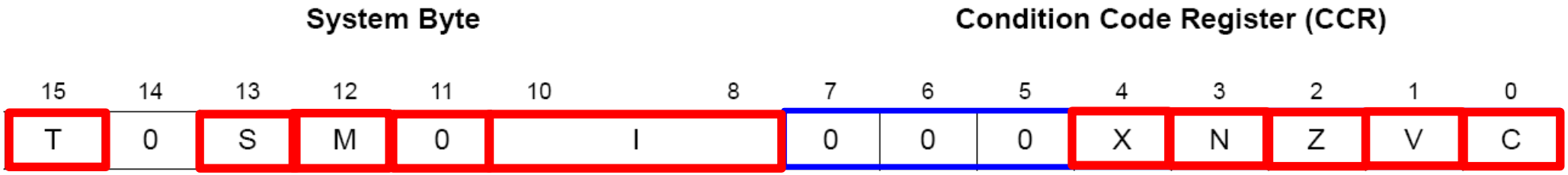
Model programowy procesora ColdFire



Model programowy procesora Motorola ColdFire



Rejestry mikrokontrolera MCF5282

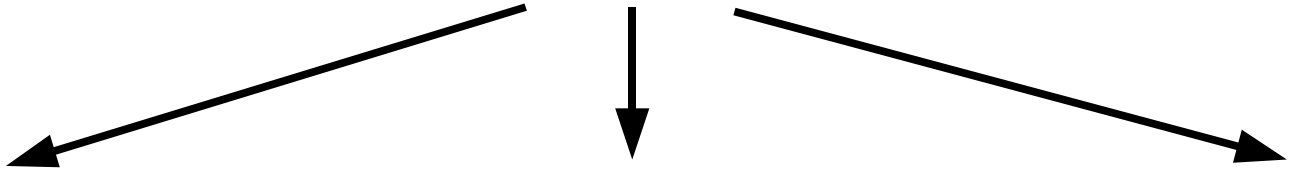


Bits	Name	Description
15	T	Trace enable. When set, the processor performs a trace exception after every instruction.
13	S	Supervisor/user state. Denotes whether the processor is in supervisor mode (S = 1) or user mode (S = 0).
12	M	Master/interrupt state. This bit is cleared by an interrupt exception, 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	Liczba ujemna w kodzie U2
2	Z	Zero condition code bit. Set if the result equals zero; otherwise cleared.
1	V	U2 – przeniesienie/pożyczka
0	C	NKB – przeniesienie/pożyczka

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				

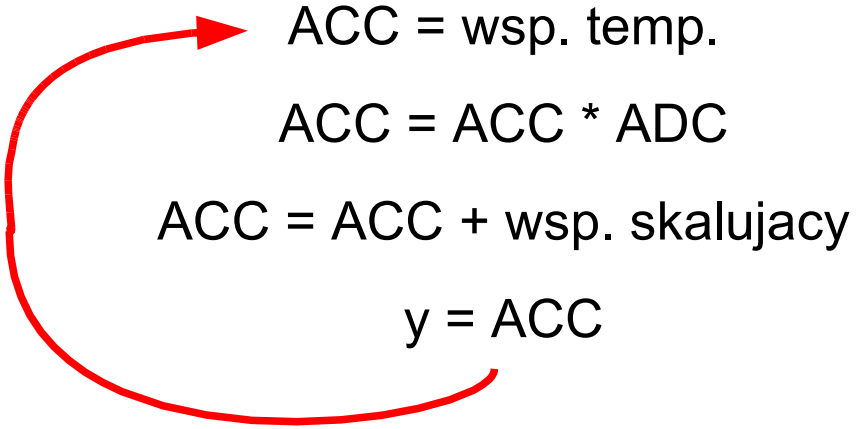
Kolejność bajtów w pamięci (5)

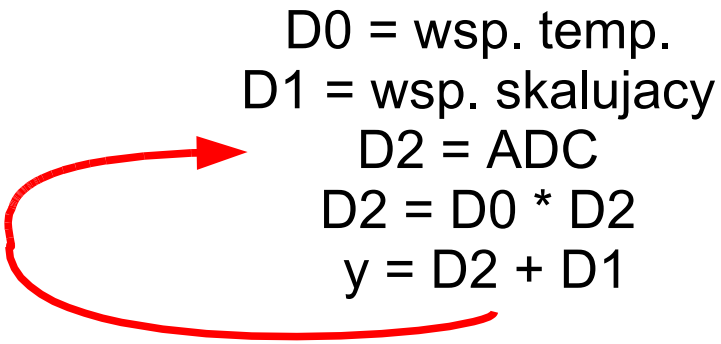
Jak rozpoznać architekturę procesora oraz rozkład bajtów w pamięci?

```
#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 ......?  */

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

   else
      return BIG_ENDIAN;
}
```

Przykład użycia rejestrów danych

y = wsp. temp. * ADC + wsp. skalujący



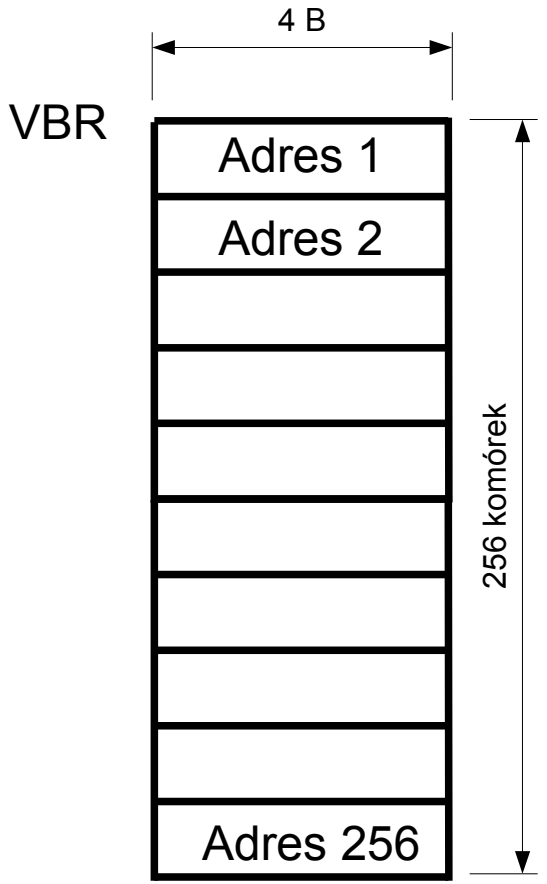


Przykład użycia rejestrów adresowych (1)

Zapisanie tablicy adresów przerwań

VBR = 0x500.000

A1 = VBR
(A1) = adres procedury przerwania
A1 = A1 + 1



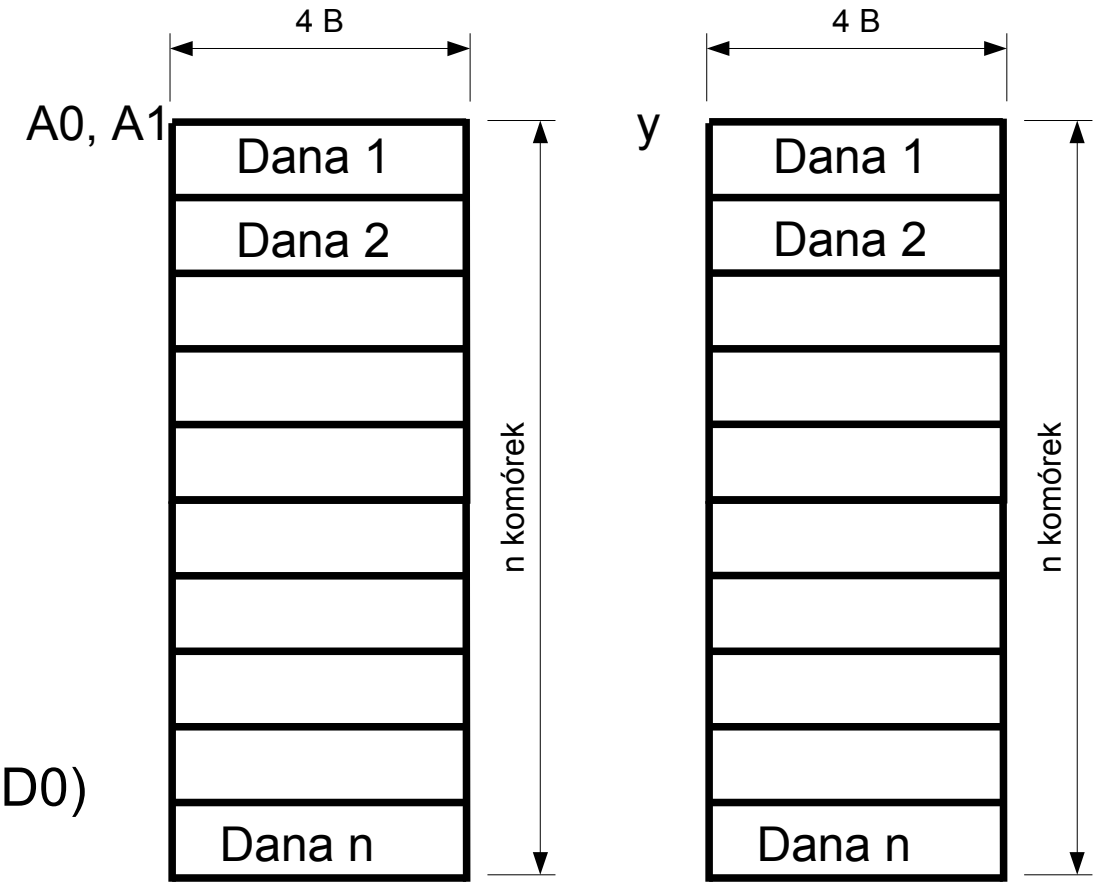
Przykład użycia rejestrów adresowych (2)

y = A0 + A1

A = adres 2
ACC = (A)
A = adres 1
ACC = ACC + (A)
A = adres 3
(A) = ACC

A1 = adres 1
A2 = adres 2
A3 = adres 3
D0 = 0

(A3, D0) = (A1, D0) + (A2, D0)



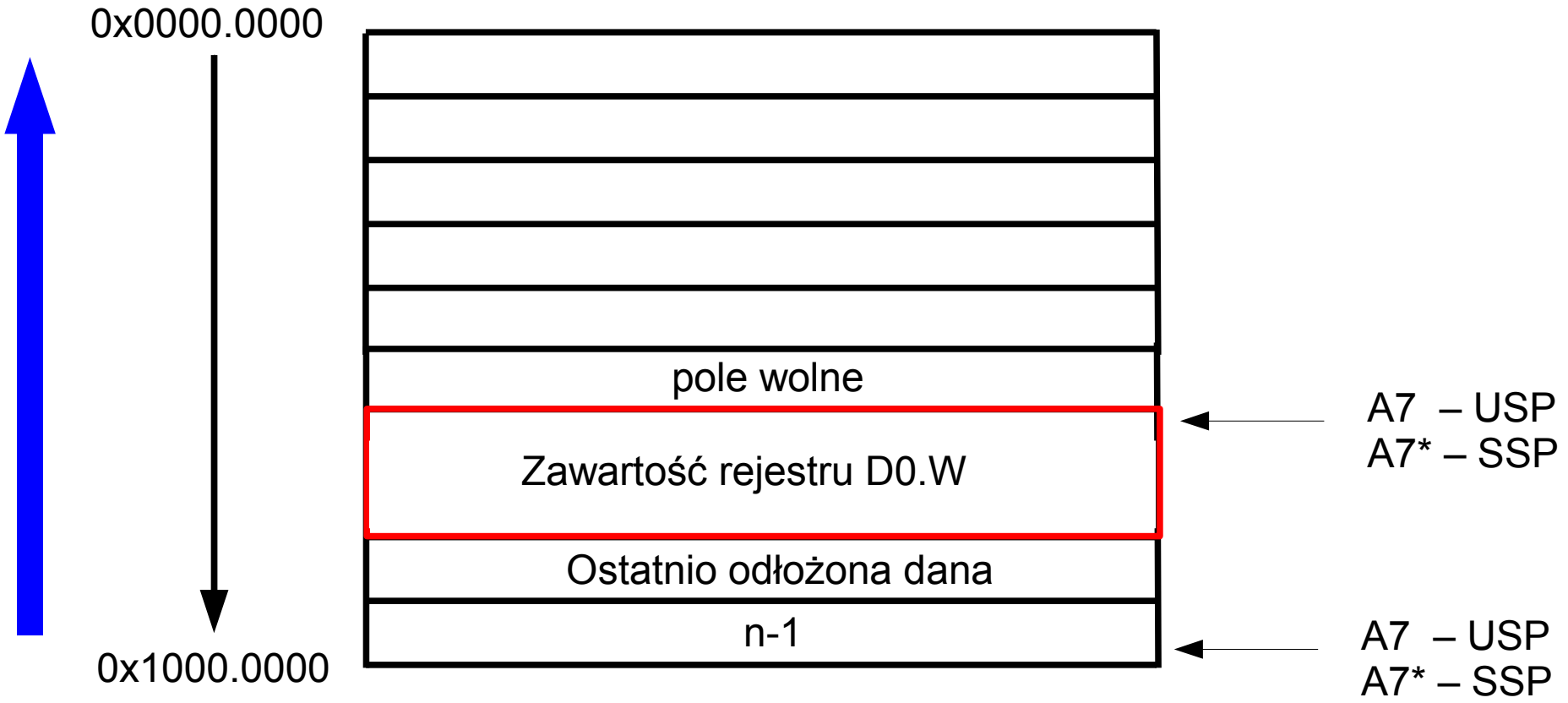

```
...
MOVE.L    0(%A1,%D0*1), %D1
MOVE.L    (%A1), %D1
MOVE.L    %D1, (%A1)+
...
```

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.

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

Struktura stosu (2)

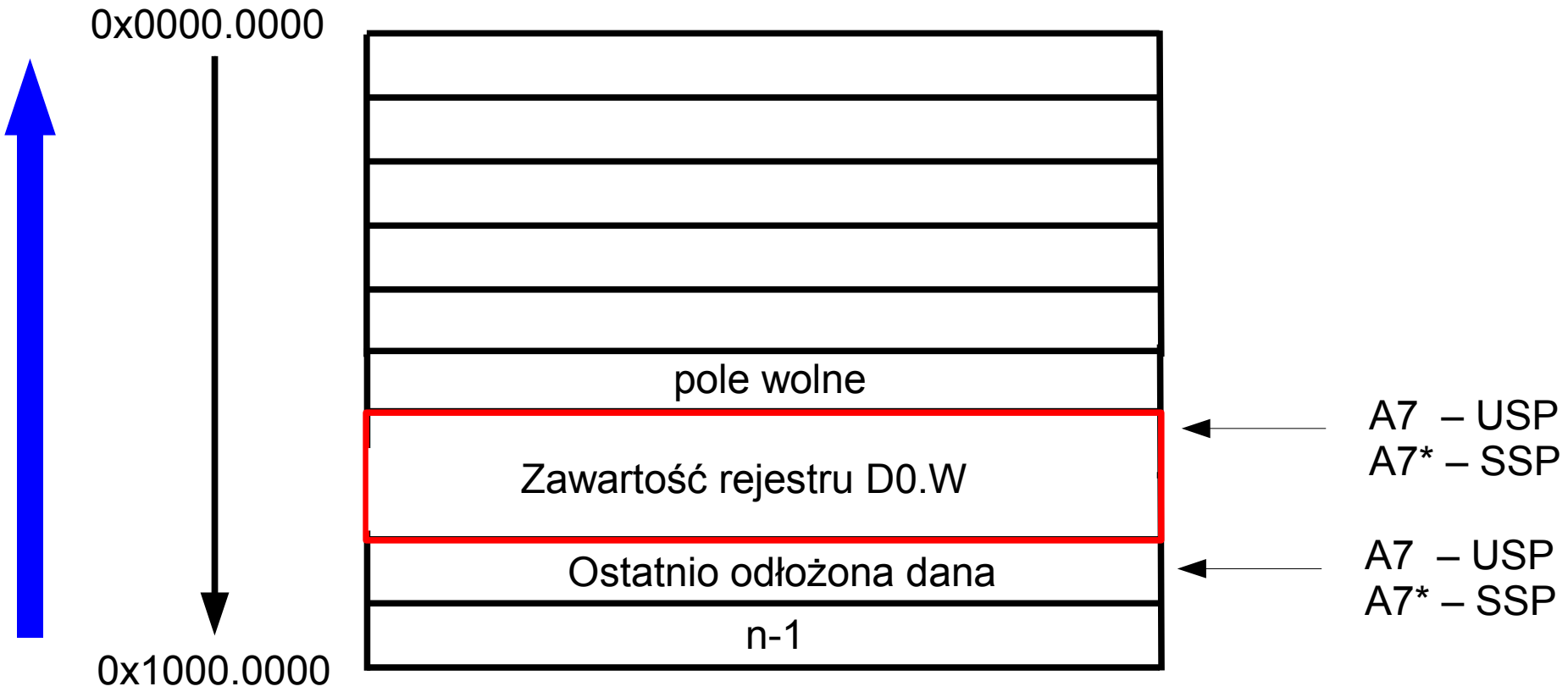


%A7 - wskaźnik stosu

```
MOVE.W    %D0, -(%A7)     | zmniejszenie A7 o 2, odłożenie zawartości D0.W na stos,

MOVE.W    (%A7)+, %D0     | zdjęcie słowa ze stosu i zapisanie D0.W, po przesłaniu danej
                          | rejestr A7 jest zwiększany o 2
```

Struktura stosu (2)

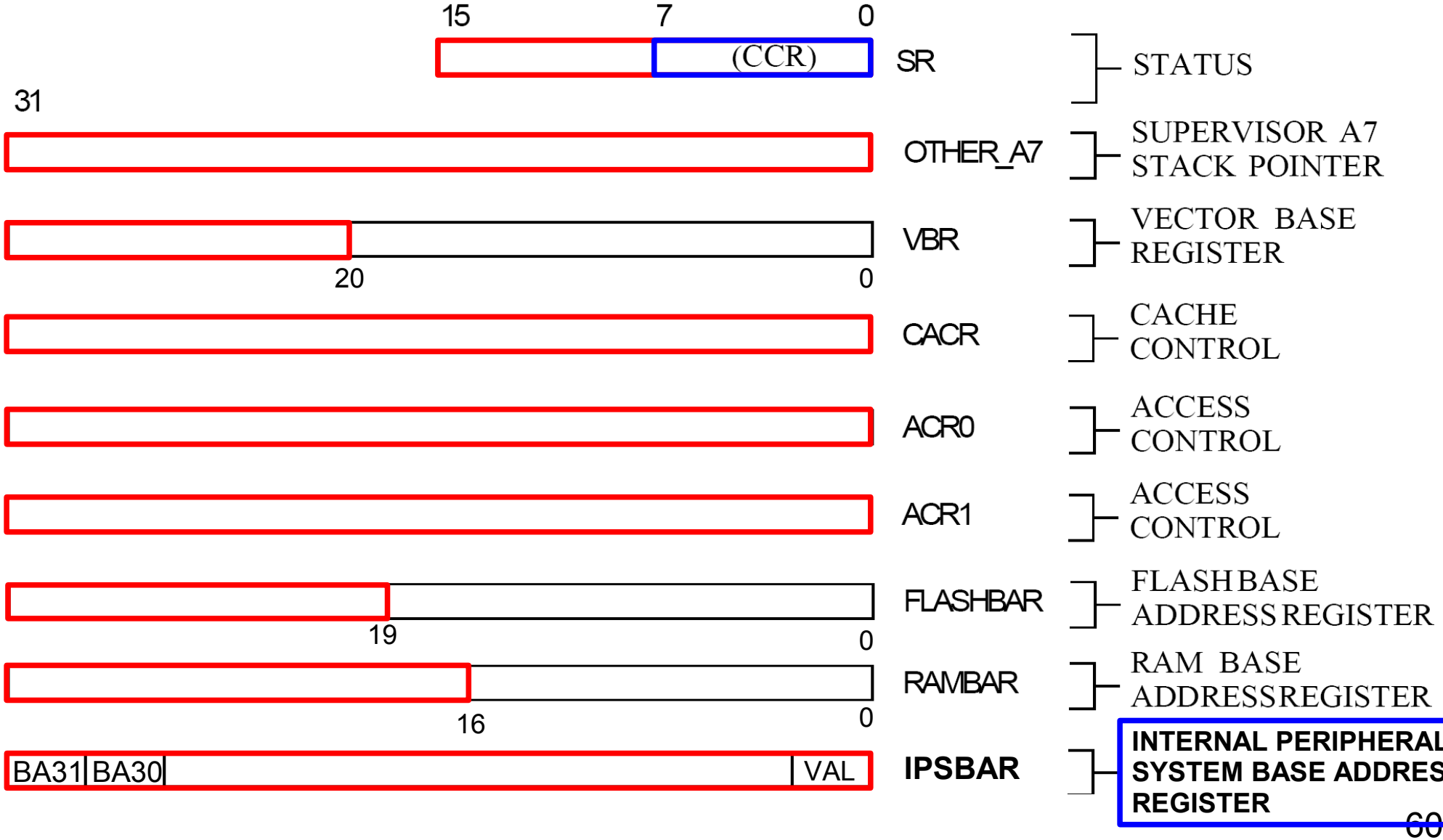
A7 - wskaźnik stosu



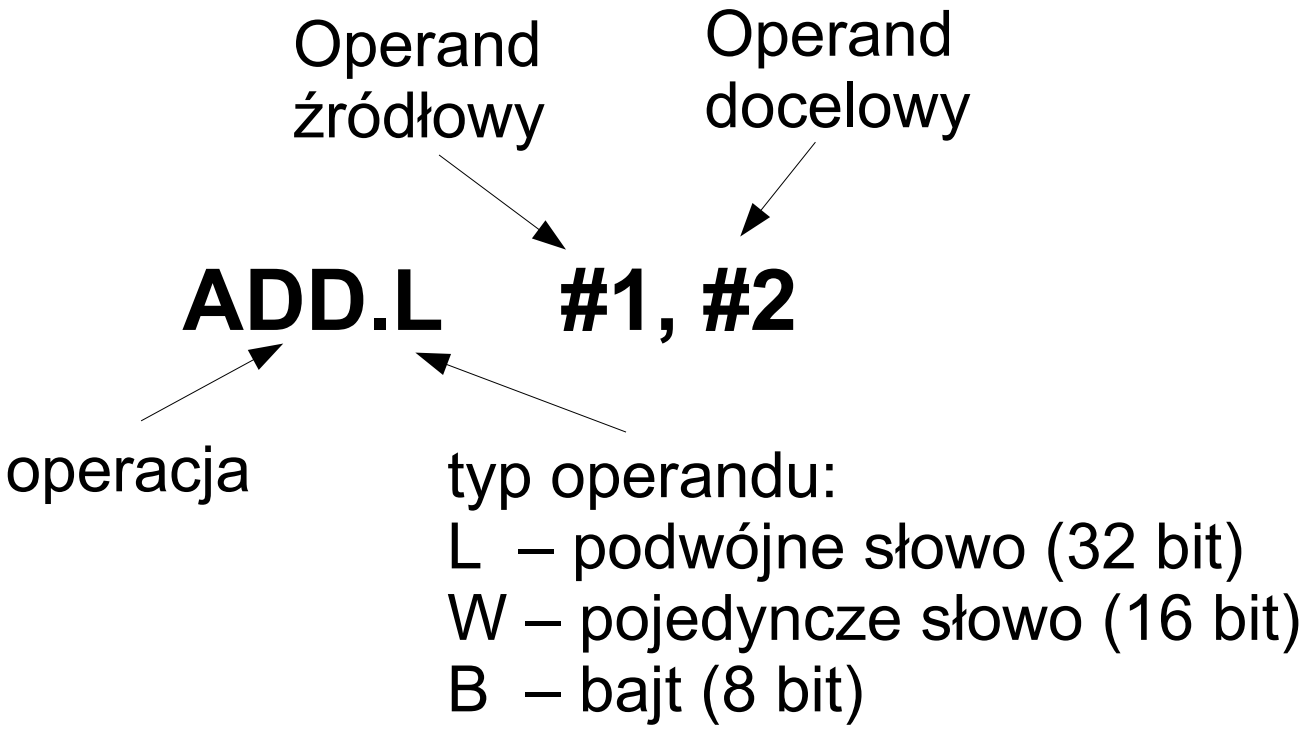

```
MOVE.W    %D0, -(%A7)    | zmniejszenie A7 o 2, odłożenie zawartości D0.W na stos,

MOVE.W    (%A7)+, %D0    | zdjęcie słowa ze stosu i zapisanie D0.W, po przesłaniu danej
                         | rejestr A7 jest zwiększany o 2
```

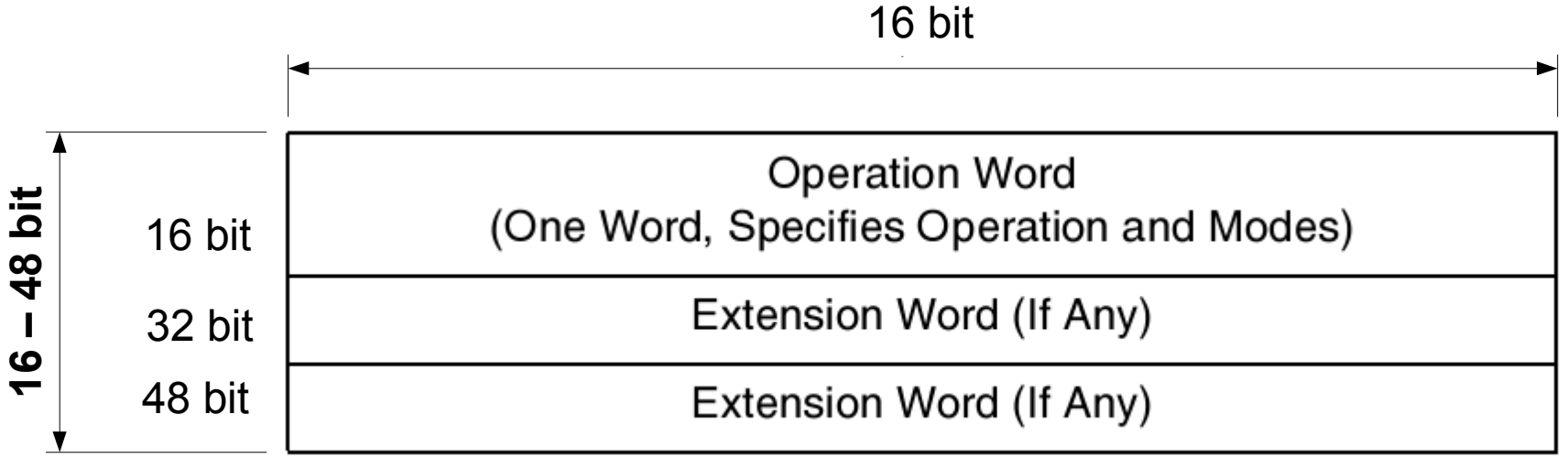
Model programowy procesora ColdFire

Rejestry dostępne w trybie superużytkownika



Format instrukcji asemblera





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.

Instrukcja dodawania

ADD **Add** **ADD**

First appeared in ISA_A

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

ADD – operacja dodawania

Operation: Source + Destination → Destination

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

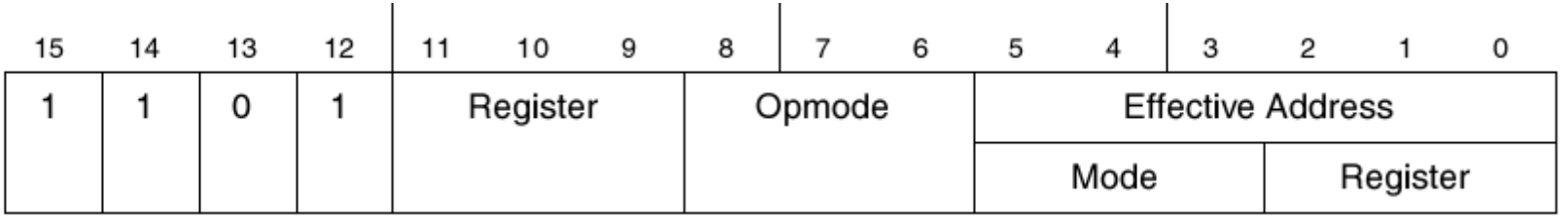
Condition Codes:

X	N	Z	V	C
*	*	*	*	*

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

ADD.L D0,D1 => D280

Instruction Format:



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

Instrukcja porównująca argumenty

CMP **Compare** **CMP**

First appeared in ISA_A
.B and .W First appeared in ISA_B

Operation: Destination – Source → cc

Assembler Syntax: CMP.sz <ea>y,Dx

Attributes: Size = byte, word, longword (byte, word supported starting with ISA_B)

Description: Subtracts the source operand from the destination operand in the data register and sets condition codes according to the result; the data register is unchanged. The operation size may be a byte, word, or longword. CMPA is used when the destination is an address register; CMPI is used when the source is immediate data.

Condition Codes:

X	N	Z	V	C
—	*	*	*	*

X Not affected
N Set if the result is negative; cleared otherwise
Z Set if the result is zero; cleared otherwise
V Set if an overflow occurs; cleared otherwise
C Set if a borrow occurs; cleared otherwise

Instrukcja skoku warunkowego

Bcc **Branch Conditionally** **Bcc**

First appeared in ISA_A

.L First appeared in ISA_B

Operation: If Condition True
Then PC + $d_n \rightarrow$ PC

Assembler Syntax: Bcc.sz <label>

Attributes: Size = byte, word, longword (longword supported starting with ISA_B)

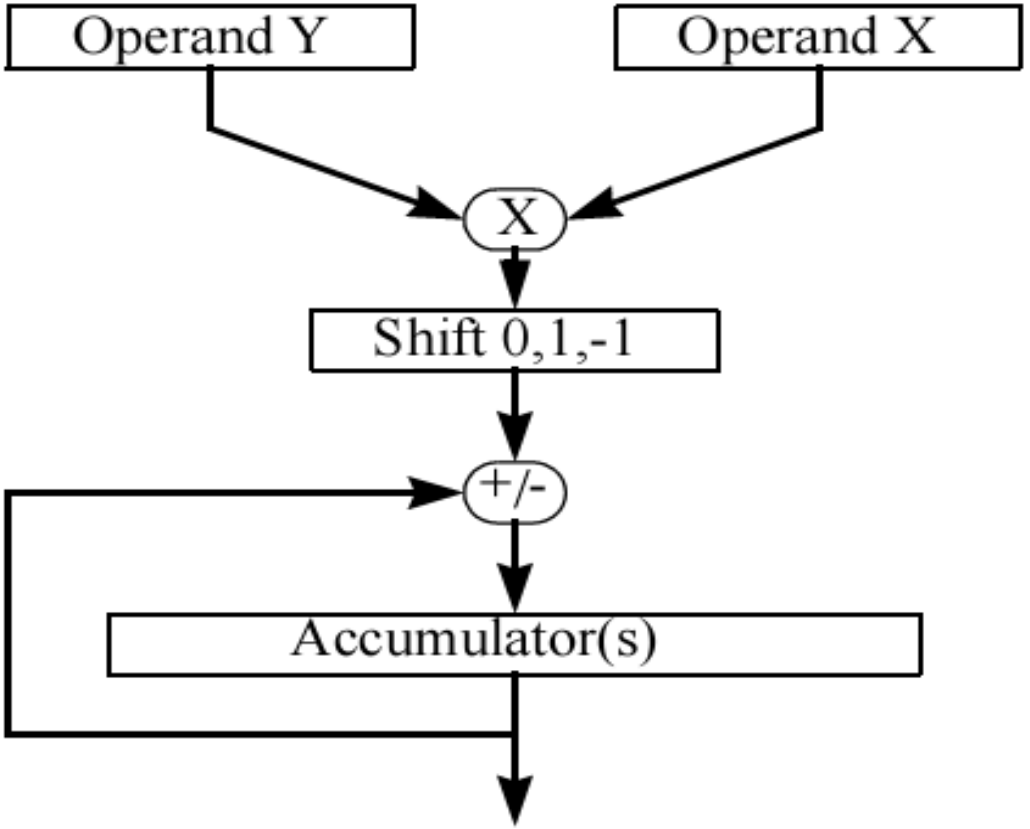
Description: If the condition is true, execution continues at (PC) + displacement. Branches can be forward, with a positive displacement, or backward, with a negative displacement. PC holds the address of the instruction word for the Bcc instruction, plus two.
Condition code specifies one of the following tests, where C, N, V, and Z stand for the condition code bits CCR[C], CCR[N], CCR[V] and CCR[Z], respectively:

Code	Condition	Encod-ing	Test	Code	Condition	Encod-ing	Test
CC(HS)	Carry clear	0100	$\overline{C}$	LS	Lower or same	0011	C I Z
CS(LO)	Carry set	0101	C	LT	Less than	1101	N & $\overline{V}$ I $\overline{N}$ & V
EQ	Equal	0111	Z	MI	Minus	1011	N
GE	Greater or equal	1100	N & V I $\overline{N}$ & $\overline{V}$	NE	Not equal	0110	$\overline{Z}$
GT	Greater than	1110	N & V & $\overline{Z}$ I $\overline{N}$ & $\overline{V}$ & $\overline{Z}$	PL	Plus	1010	$\overline{N}$
HI	High	0010	$\overline{C}$ & $\overline{Z}$	VC	Overflow clear	1000	$\overline{V}$
LE	Less or equal	1111	Z I N & $\overline{V}$ I $\overline{N}$ & V	VS	Overflow set	1001	V

Condition Codes: Not affected

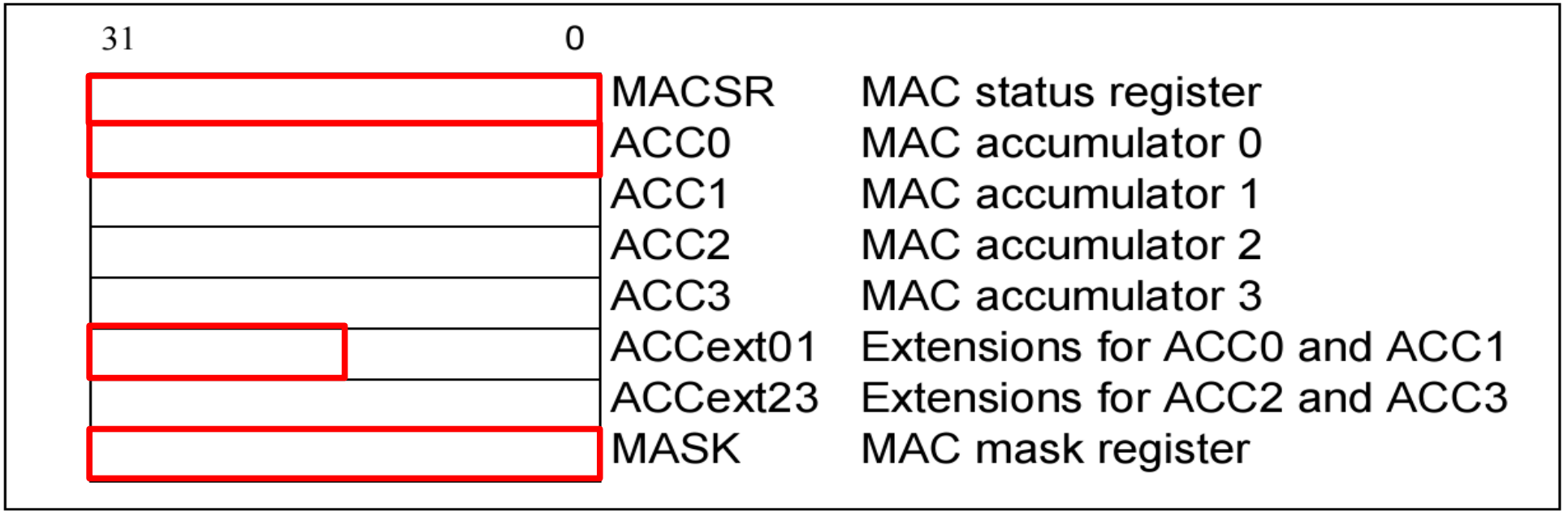
Moduł arytmetyczny EMAC (1)

Enhanced Multiply-ACcumulate Unit



$$y(i) = \sum_{k=1}^{N-1} a(k)y(i-k) + \sum_{k=0}^{N-1} b(k)x(i-k)$$

Moduł arytmetyczny EMAC (2)

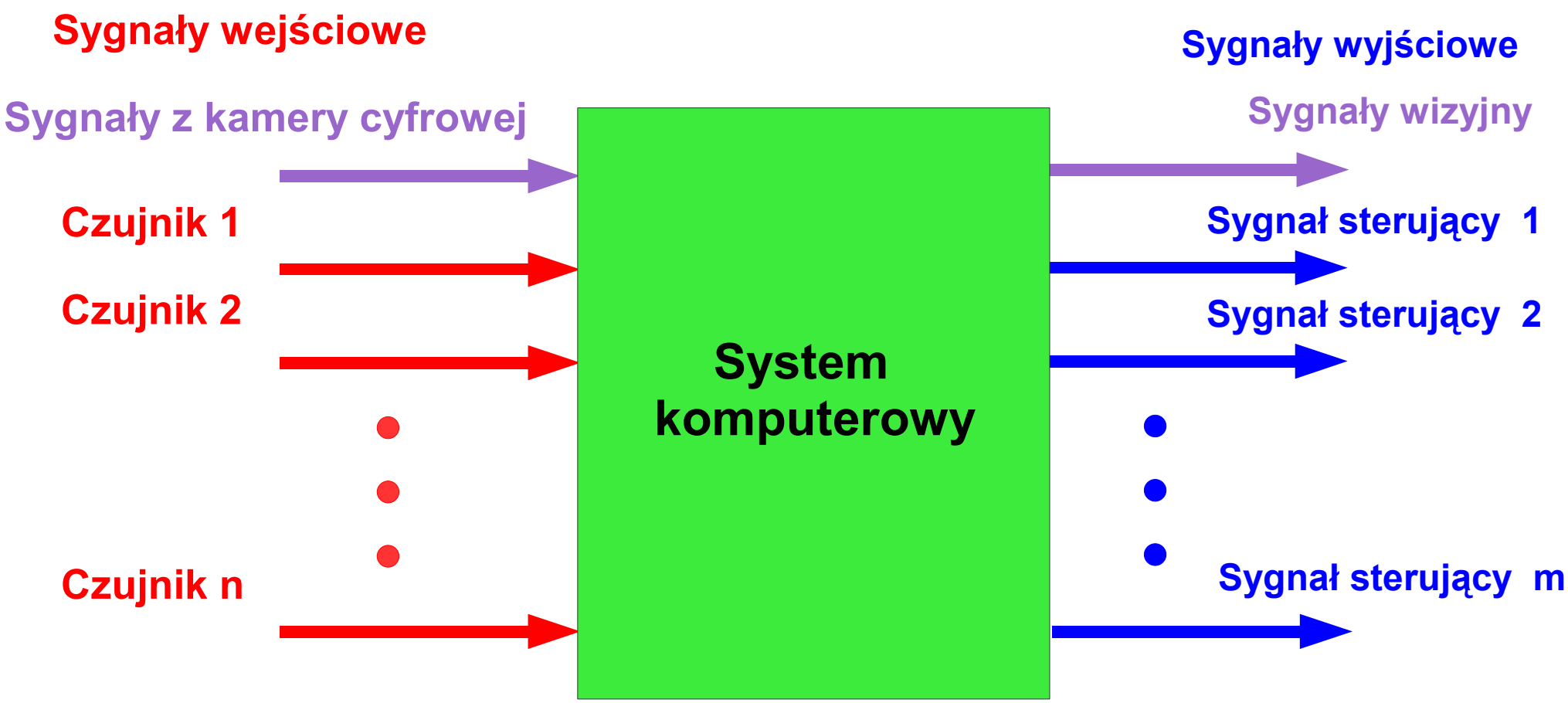


ColdFire® Embedded Controllers

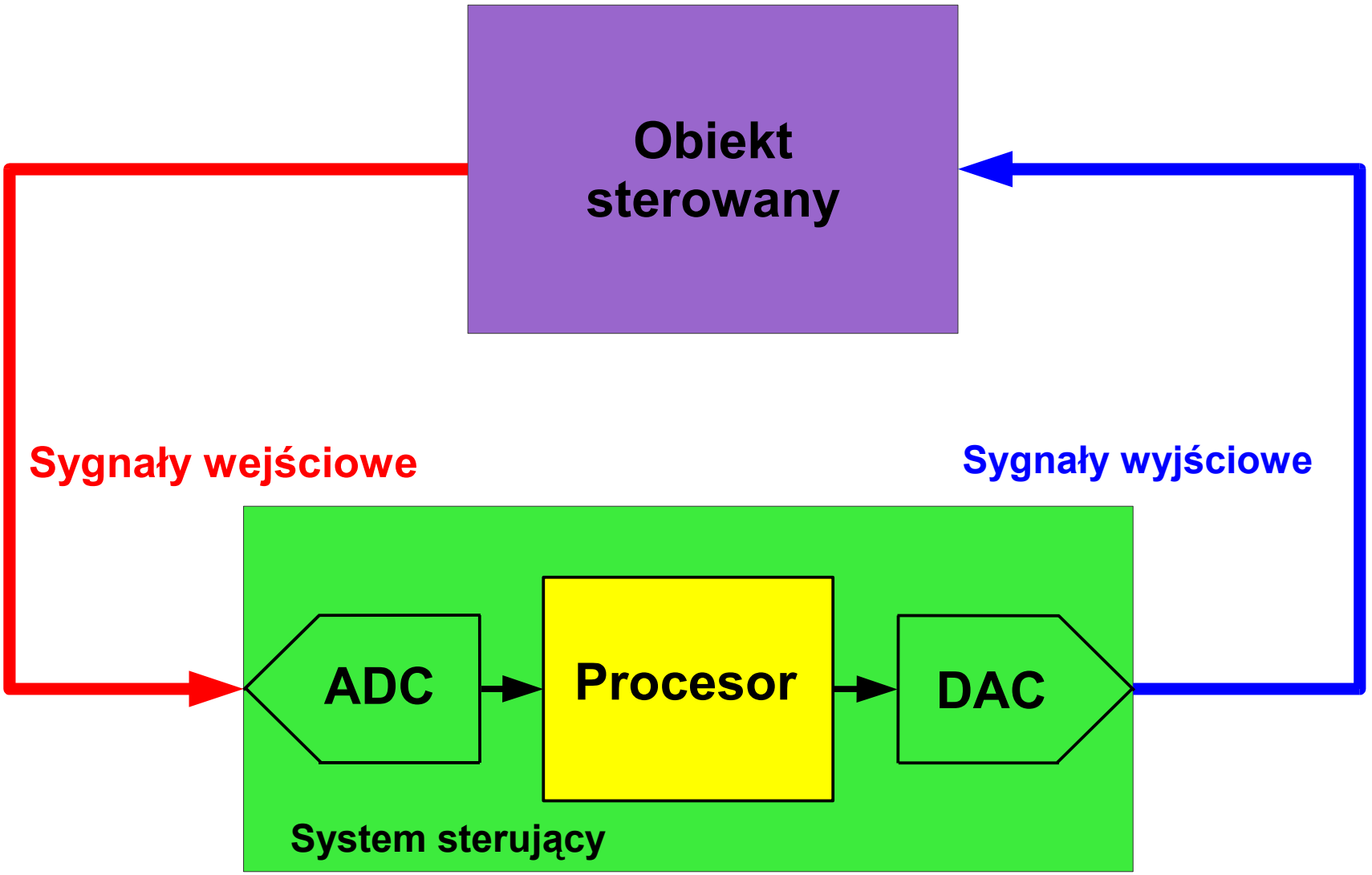
MCF528x Family

Systemy Mikroprocesorowe Czasu Rzeczywistego

System mikroprocesorowy



System sterujący



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

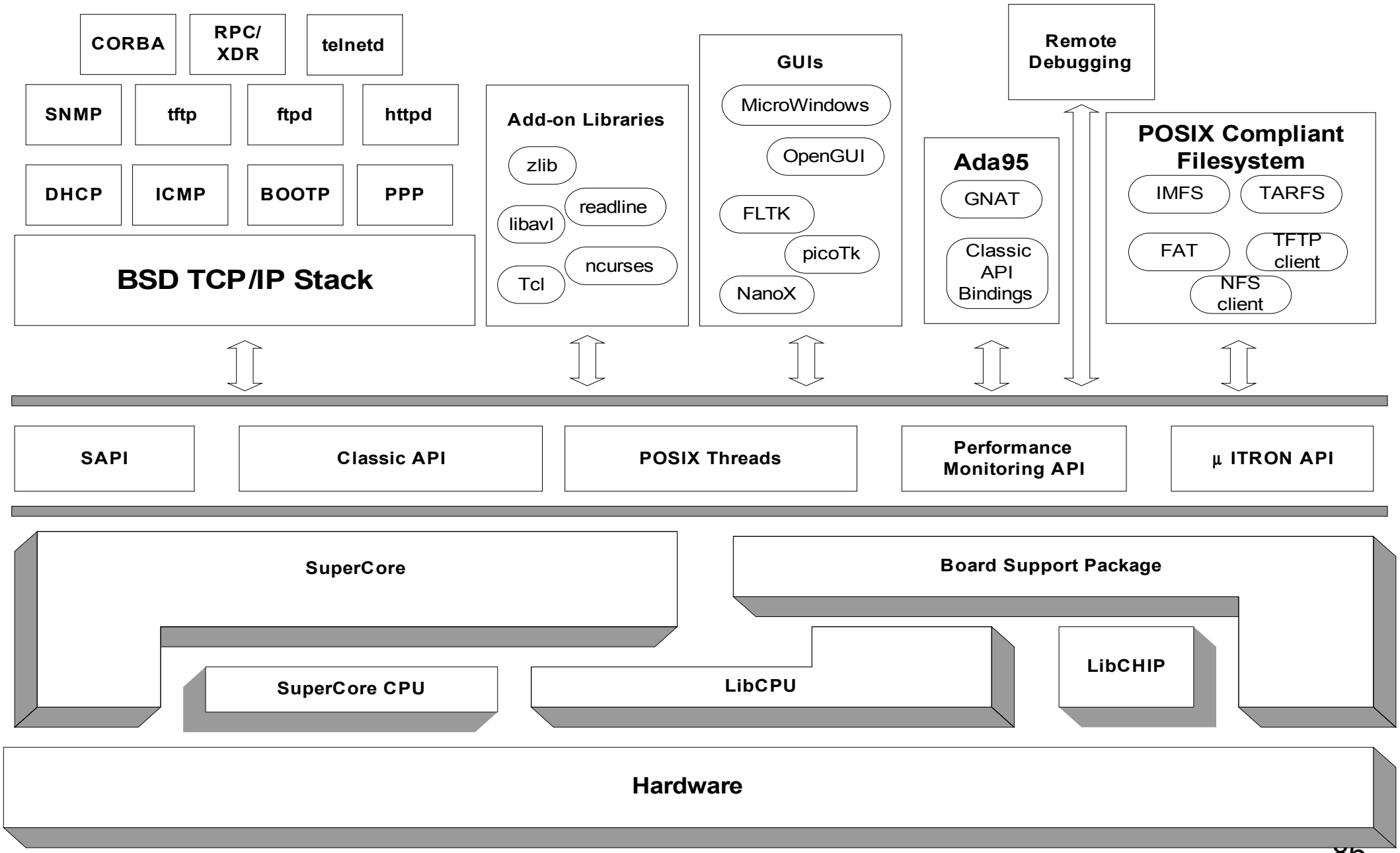
Procesory wspierane przez RTEMS (2)

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