KAPITAŁ LUDZKI
NARODOWA STRATEGIA SPÓJNOŚCI

UNIA EUROPEJSKA
EUROPEJSKI
FUNDUSZ SPOŁECZNY

# „Procesory ARM w systemach wbudowanych”
# „Wprowadzenie do przedmiotu”

Prezentacja jest współfinansowana przez
Unię Europejską w ramach
Europejskiego Funduszu Społecznego w projekcie pt.

*„Innowacyjna dydaktyka bez ograniczeń - zintegrowany rozwój Politechniki Łódzkiej - zarządzanie Uczelnią, nowoczesna oferta edukacyjna i wzmacniania zdolności do zatrudniania osób niepełnosprawnych”*

Prezentacja dystrybuowana jest bezpłatnie

# Dariusz Makowski

## Katedra Mikroelektroniki i Technik Informatycznych

## tel. 631 2720

## dmakow@dmcs.pl

## http://neo.dmcs.p.lodz.pl/arm/

## ul. Wólczańska 221/223, budynek B18, pokój 40


KAPITAŁ LUDZKI
NARODOWA STRATEGIA SPÓJNOŚCI

Projekt współfinansowany przez Unię Europejską
w ramach Europejskiego Funduszu Społecznego

UNIA EUROPEJSKA
EUROPEJSKI
FUNDUSZ SPOŁECZNY

# Sprawy formalne

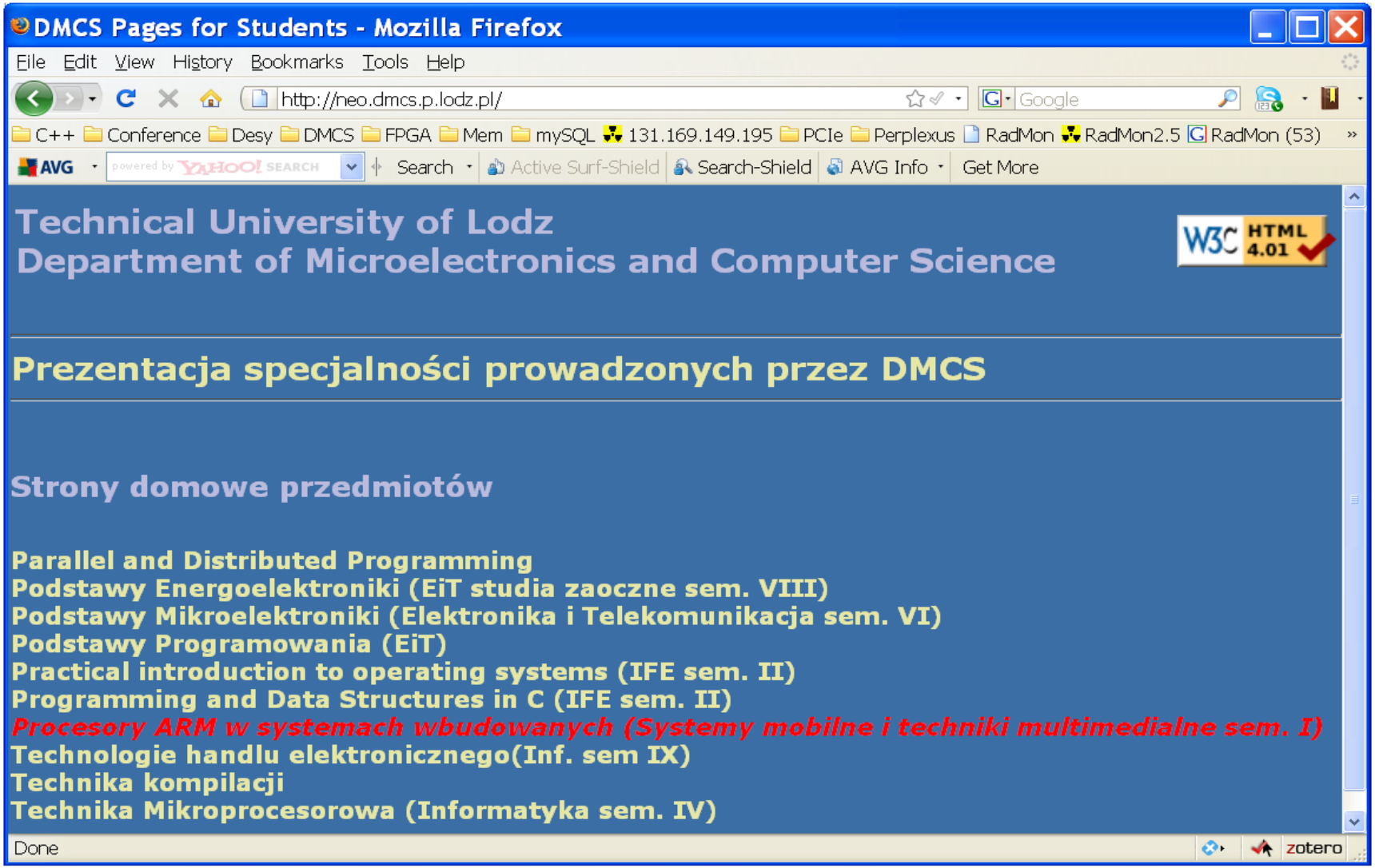
- Informacje wstępne
- Zaliczenie
- Laboratorium
- Materiały do wykładu

KAPITAŁ LUDZKI
NARODOWA STRATEGIA SPÓJNOŚCI

Projekt współfinansowany przez Unię Europejską
w ramach Europejskiego Funduszu Społecznego

UNIA EUROPEJSKA
EUROPEJSKI
FUNDUSZ SPOŁECZNY

# Strona przedmiotu – materiały do wykładu




KAPITAŁ LUDZKI
NARODOWA STRATEGIA SPÓJNOŚCI

Projekt współfinansowany przez Unię Europejską
w ramach Europejskiego Funduszu Społecznego

UNIA EUROPEJSKA
EUROPEJSKI
FUNDUSZ SPOŁECZNY

# Zakres przedmiotu

- Systemy mikroprocesorowe, systemy wbudowane
- Laboratorium
- Rodzina procesorów ARM
- Urządzenia peryferyjne
- Pamięci i dekodery adresowe
- Programy wbudowane na przykładzie procesorów ARM
- Metodyki projektowania systemów wbudowanych
- Systemy czasu rzeczywistego

KAPITAŁ LUDZKI
NARODOWA STRATEGIA SPÓJNOŚCI

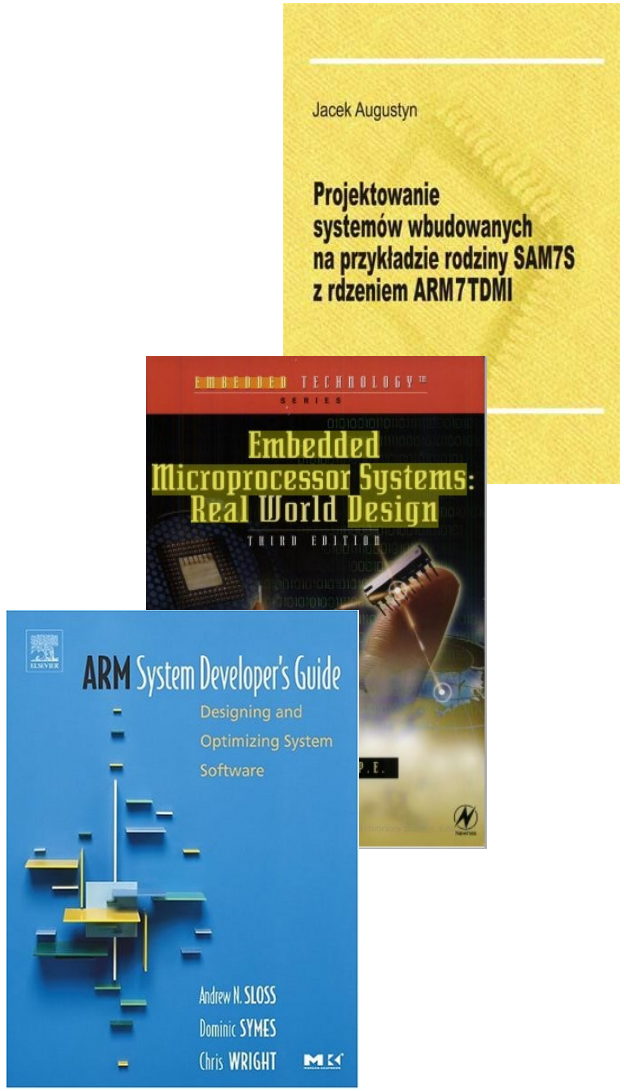
Projekt współfinansowany przez Unię Europejską
w ramach Europejskiego Funduszu Społecznego

UNIA EUROPEJSKA
EUROPEJSKI
FUNDUSZ SPOŁECZNY

# Literatura

## Literatura obowiązkowa:

- Materiały wykładowe i laboratoryjne
- J. Augustyn, “Projektowanie systemów wbudowanych na przykładzie rodziny SAM7S z rdzeniem ARM7TDMI”, IGSMiE PAN, 2007, ISBN: 978-83-60195-55-0
- A. Sloss, D. Symes, C. Wright, „ARM System Developer's Guide: Designing and Optimizing System Software”, Elsevier, 2004
- S. R. Ball, “Embedded Microprocessor Systems: Real World Design”, Elsevier Science, 2002




## Literatura uzupełniająca:

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

# Zakres przedmiotu

- Systemy mikroprocesorowe, systemy wbudowane
- Laboratorium
- Rodzina procesorów ARM
- Urządzenia peryferyjne
- Pamięci i dekodery adresowe
- Programy wbudowane na przykładzie procesorów ARM
- Metodyki projektowania systemów wbudowanych
- Interfejsy w systemach wbudowanych
- Systemy czasu rzeczywistego

KAPITAŁ LUDZKI
NARODOWA STRATEGIA SPÓJNOŚCI

Projekt współfinansowany przez Unię Europejską
w ramach Europejskiego Funduszu Społecznego

UNIA EUROPEJSKA
EUROPEJSKI
FUNDUSZ SPOŁECZNY

# Definicje podstawowe (1)

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

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

- **Mikroprocesor (ang. Microprocessor)**

  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 (ang. Microcontroller)**

  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, ARM926EJ-S

KAPITAŁ LUDZKI
NARODOWA STRATEGIA SPÓJNOŚCI

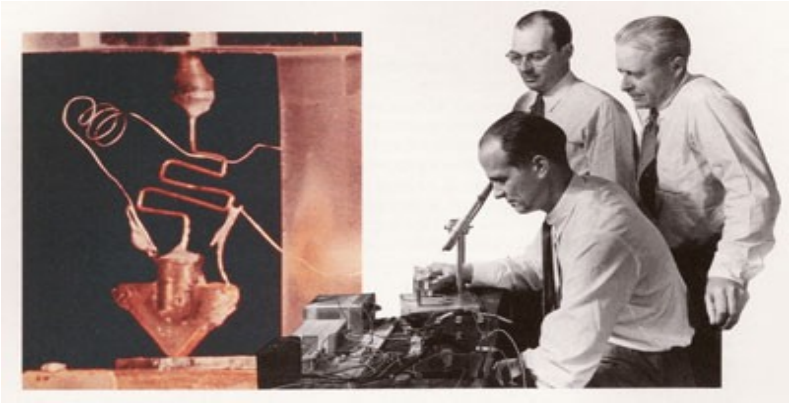
Projekt współfinansowany przez Unię Europejską
w ramach Europejskiego Funduszu Społecznego

UNIA EUROPEJSKA
EUROPEJSKI
FUNDUSZ SPOŁECZNY

# 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.

KAPITAŁ LUDZKI
NARODOWA STRATEGIA SPÓJNOŚCI

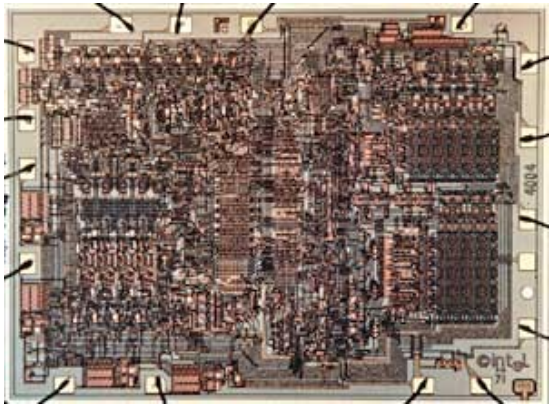
Projekt współfinansowany przez Unię Europejską
w ramach Europejskiego Funduszu Społecznego

UNIA EUROPEJSKA
EUROPEJSKI
FUNDUSZ SPOŁECZNY

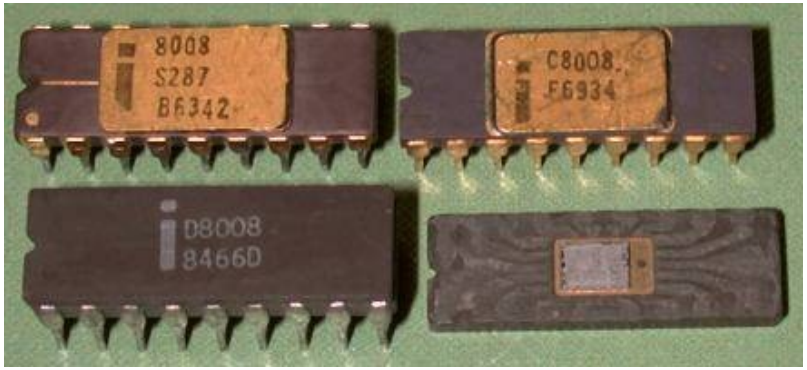
# Historia mikroprocesorów (2)

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.

# 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).

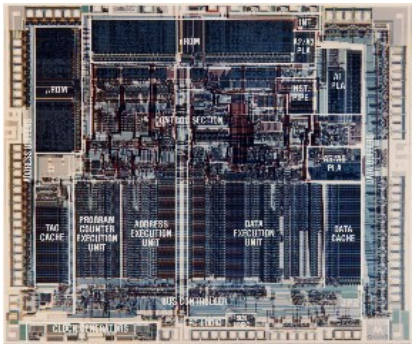
1975 – 8-bitowy procesor INTEL 6502 (technologia MOS) – najtańszy proc. na rynku.

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

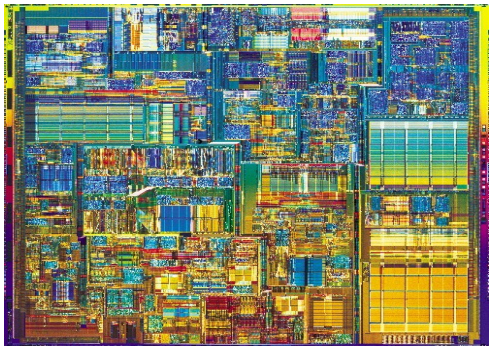
1979 – Motorola oferuje 16-bitowy procesor 68000.

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, ARM, ...

# Definicje podstawowe (2)

- **Komputer (ang. Computer)**

  Urządzenie elektroniczne, maszyna cyfrowa zdolna do przetwarzania danych cyfrowych zgodnie z dostarczonym programem

- **Komputer (system) wbudowany (ang. Embedded Computer)**

  Dedykowany system komputerowy, niewielkich rozmiarów sterownik *wbudowany w urządzenie*, przeznaczony do sterowania urządzeniem mechanicznym, elektrycznym lub elektronicznym

- **Komputer osobisty (ang. Personal Computer)**

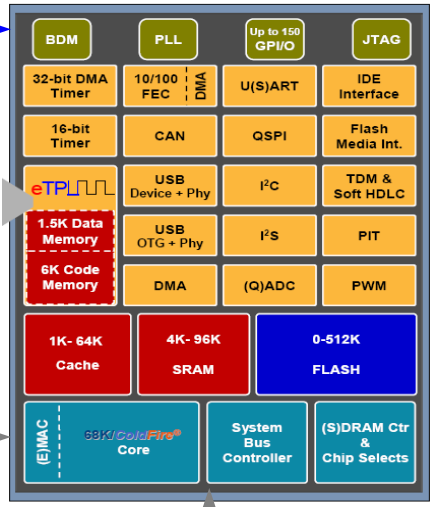
  System komputery przeznaczony do użytku osobistego, domowego lub biurowego. Komputer wyposażony w system operacyjny przeznaczony do wykonywania *aplikacji wykorzystywanych przez użytkownika*

- **Architektura komputera (ang. Computer Architecture)**
  - Sposób organizacji oraz współpracy podstawowych elementów systemu komputerowego, tj. procesora, pamięci oraz urządzeń peryferyjnych
  - Opis komputera z punktu widzenia programisty w języku niskiego poziomu (asembler). Budowa procesora, potoku wykonawczego oraz model programowy procesora

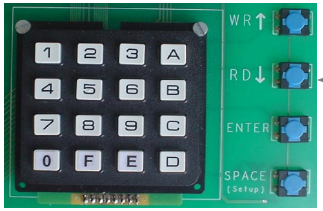
# Komputer wbudowany (embedded computer)

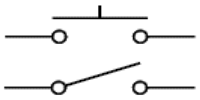
Komunikacja z komputerem zewnętrznym

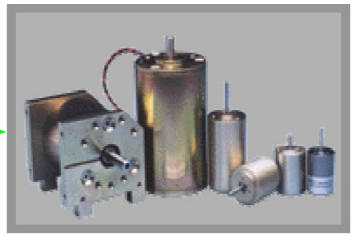
Czujniki, np. czujnik temperatury, obrotów, itd




Kamera

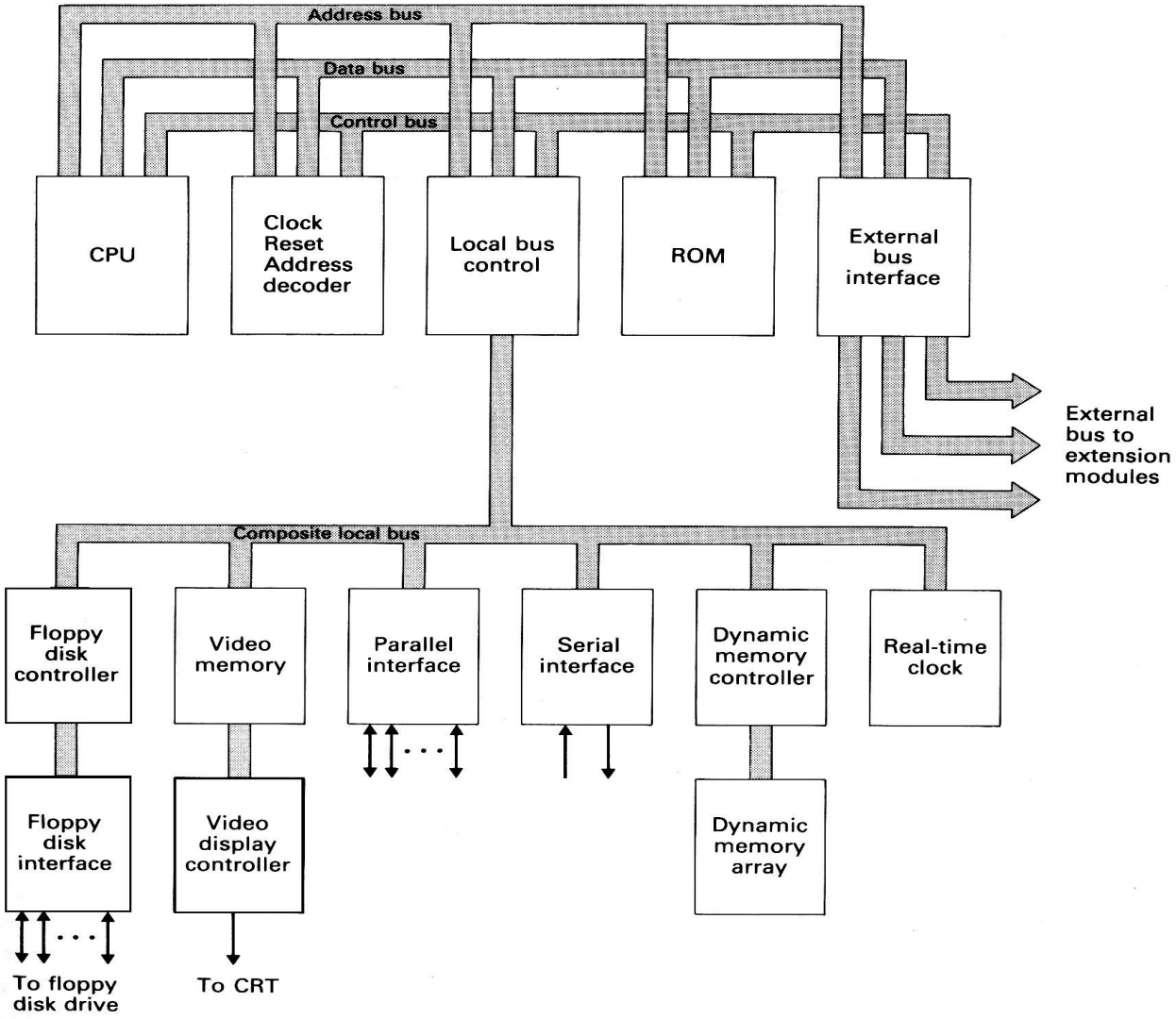


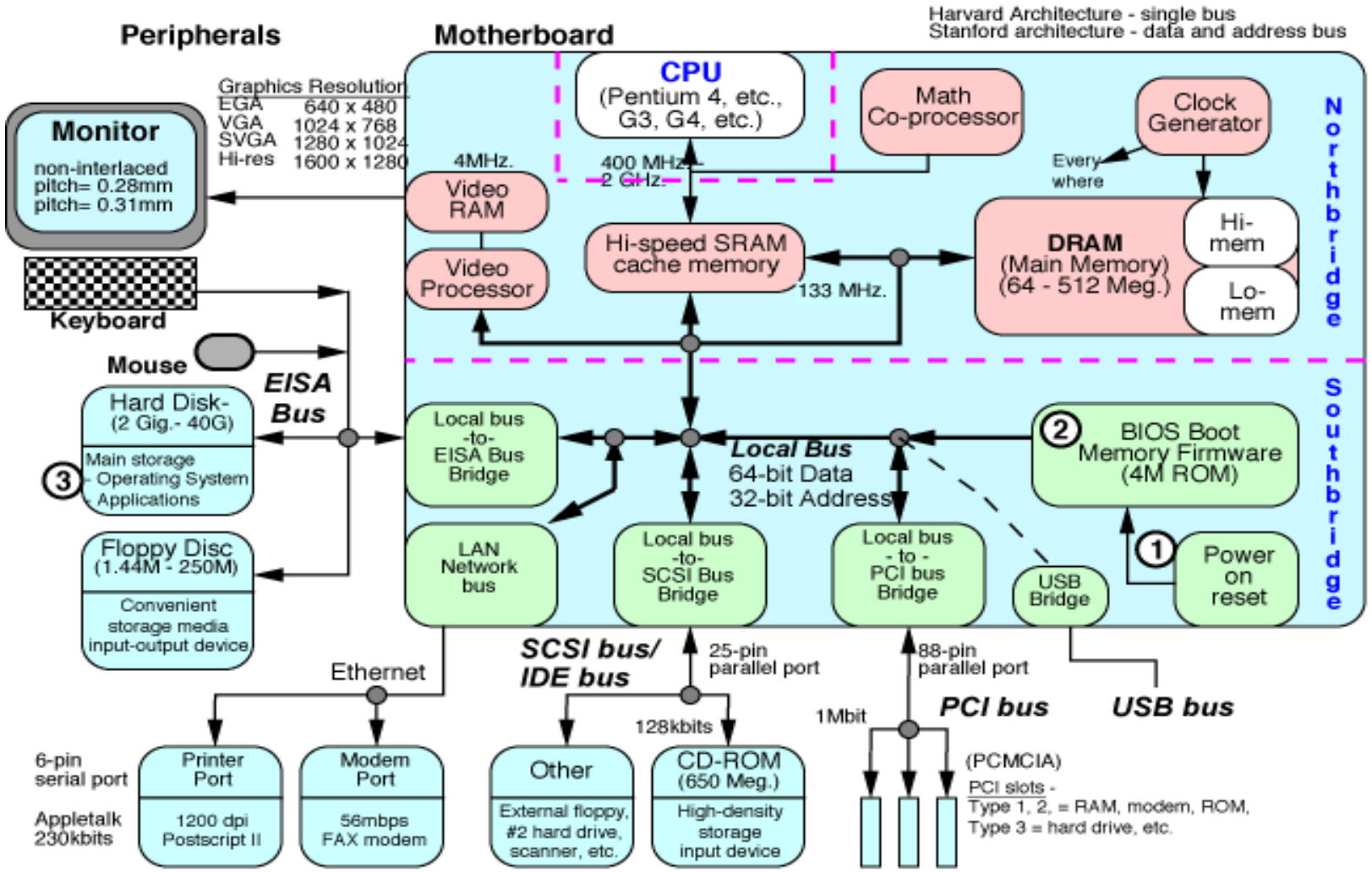



Elementy wykonawcze, np. silniki, przekaźniki

# Komputer uniwersalny



# Schemat blokowy komputera osobistego



# Definicje podstawowe (3)

- **Pamięć komputerowa (ang. Computer Memory)**
  Pamięć komputerowa to urządzenie elektroniczne lub mechaniczne służące do przechowywania danych i programów (systemu operacyjnego oraz aplikacji).
- **Urządzenia zewnętrzne, peryferyjne (ang. Peripheral Device)**
  Urządzenia elektroniczne dołączone do procesora przez magistrale systemową lub interfejs. Urządzenia zewnętrzne wykorzystywane są do realizowania specjalizowanej funkcjonalności systemu.
- **Magistrala (ang. bus)**
  Połączenie elektryczne umożliwiające przesyłanie danym pomiędzy procesorem, pamięcią i urządzeniami peryferyjnymi. Magistra systemowa zbudowane jest zwykłe z kilkudziesięciu połączeń elektrycznych (ang. Parallel Bus) lub szeregowego połączenia (ang. Serial Bus).
- **Interface (ang. Interface)**
  Urządzenie elektroniczne lub optyczne pozwalające na komunikację między dwoma innymi urządzeniami, których bezpośrednio nie da się ze sobą połączyć.

# Definicje podstawowe (4)

- **Komputer SoC (ang. System-on-Chip)**

  Układ scalony wielkiej integracji zawierający ***kompletny system elektroniczny zintegrowany z układami analogowymi, cyfrowo-analogowymi oraz radiowymi.*** Poszczególne moduły tego systemu, ze względu na ich złożoność, pochodzą zwykle od różnych dostawców, np. rdzeń procesora od jednego producenta, układy peryferyjne od innego, interfejsy od jeszcze innego, itd...

  Typowym obszarem zastosowań SoC są systemy wbudowane, a najbardziej rozpowszechnionym przedstawicielem tego rozwiązania są systemy oparte na procesorach ARM.

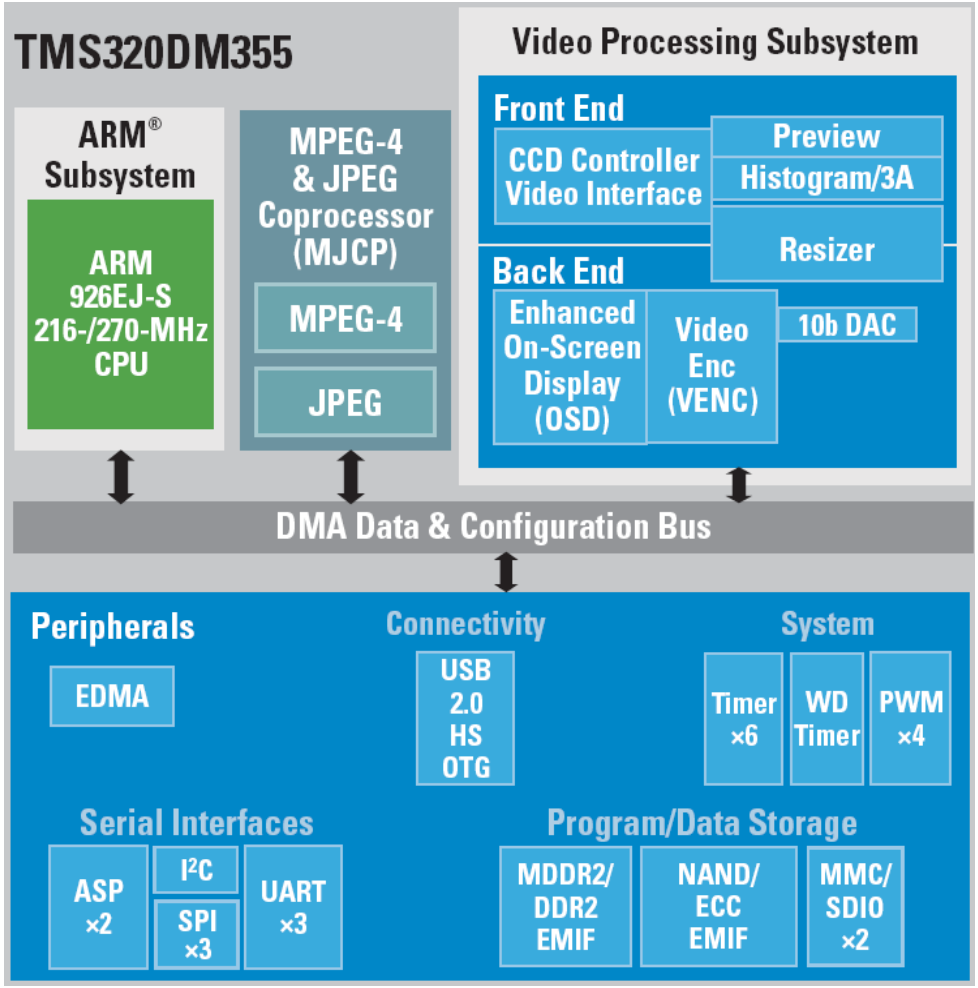
  W przypadku, gdy nie jest możliwa integracja wszystkich układów na jednym podłożu półprzewodnikowym, poszczególne moduły wykonuje się na osobnych kryształach, a całość zamyka się w jednej obudowie, SiP (ang. System-in-a-package).

  SoC różnią się od mikrokontrolerów ***znacznie wydajniejszą jednostka obliczeniową CPU*** (pozwalającej uruchamiać systemy operacyjne, np. Linux, Windows) oraz są zwykłe ***wyposażone w specjalizowane układy peryferyjne.***

# SoC - DaVinci, digital media processor

## DaVinci DM355

- SoC opracowany przez firmę Texas Instruments
- Dedykowany co-procesor do przetwarzania dźwięku i obrazu w czasie rzeczywistym
- Niski pobór energii 400 mW podczas dekodowania HD MPEG4, 1 mW w stanie czuwania (systemy przenośne)
- Bogate interfejsy i układy peryferyjne (sterownik HDD, SD/MMC, USB, Ethernet,...)

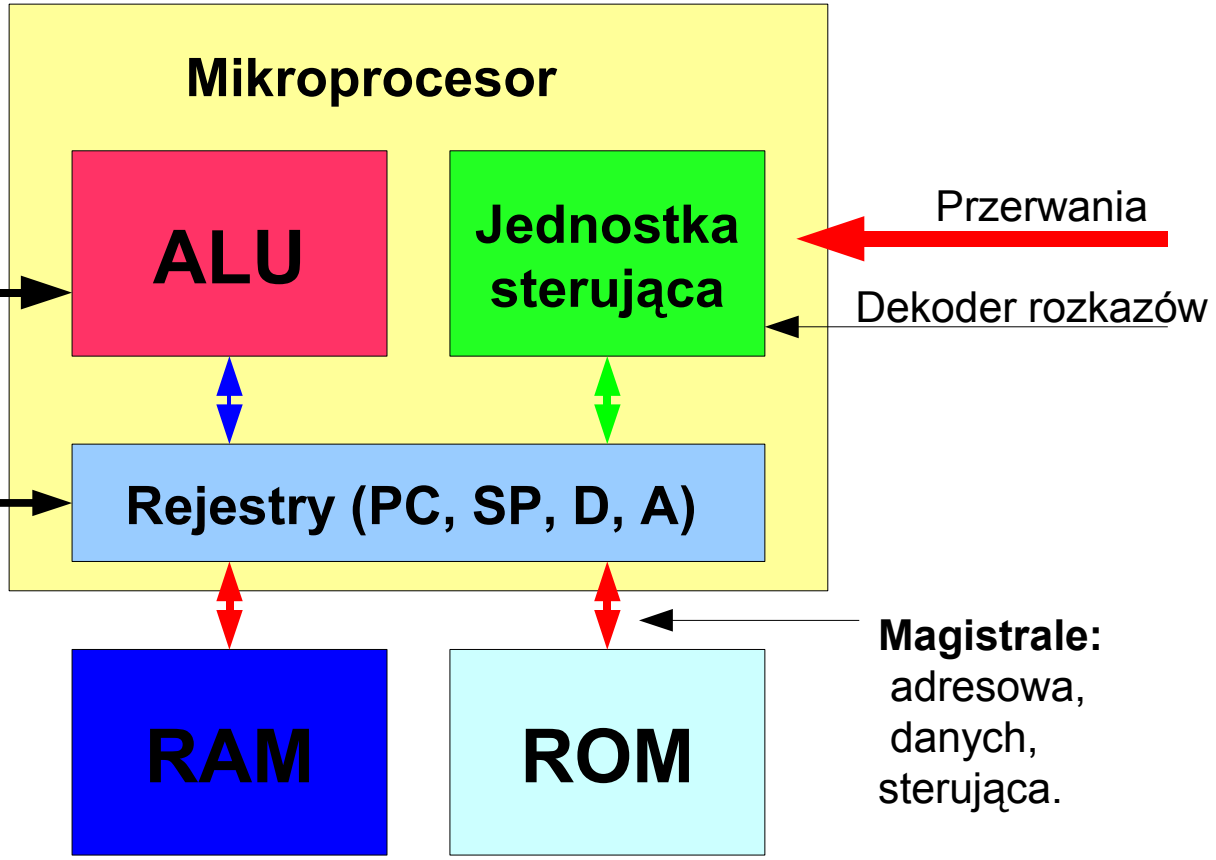



Źródło: www.arm.com

# 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 (ang. Arithmetic Logic Unit), realizuje podstawowe operacje matematyczne 8, 16, 32, 64-bit

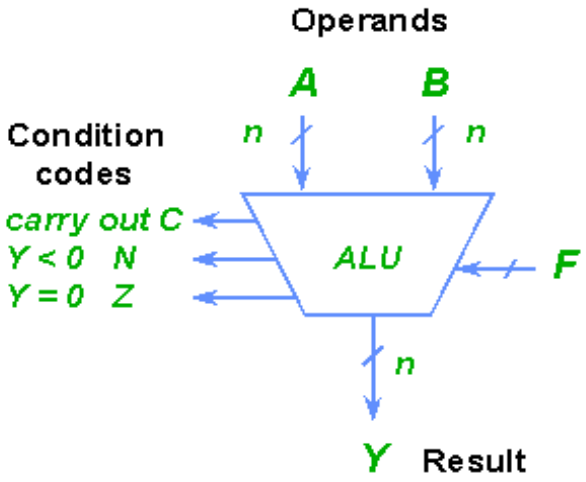



Rejestry procesora - obszar (plik) rejestrów (ang. registers file) - komórki szybkiej pamięci statycznej, umieszczonej, wewnątrz procesora, 8, 16, 32, 64, 128-bit,

# Jednostka arytmetyczno-logiczna

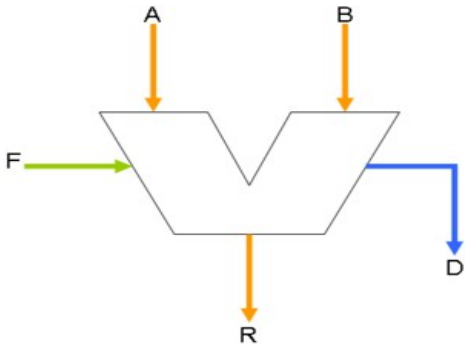
**ALU wykorzystywana jest do wykonywania:**

- 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 |

# Dwubitowa jednostka ALU

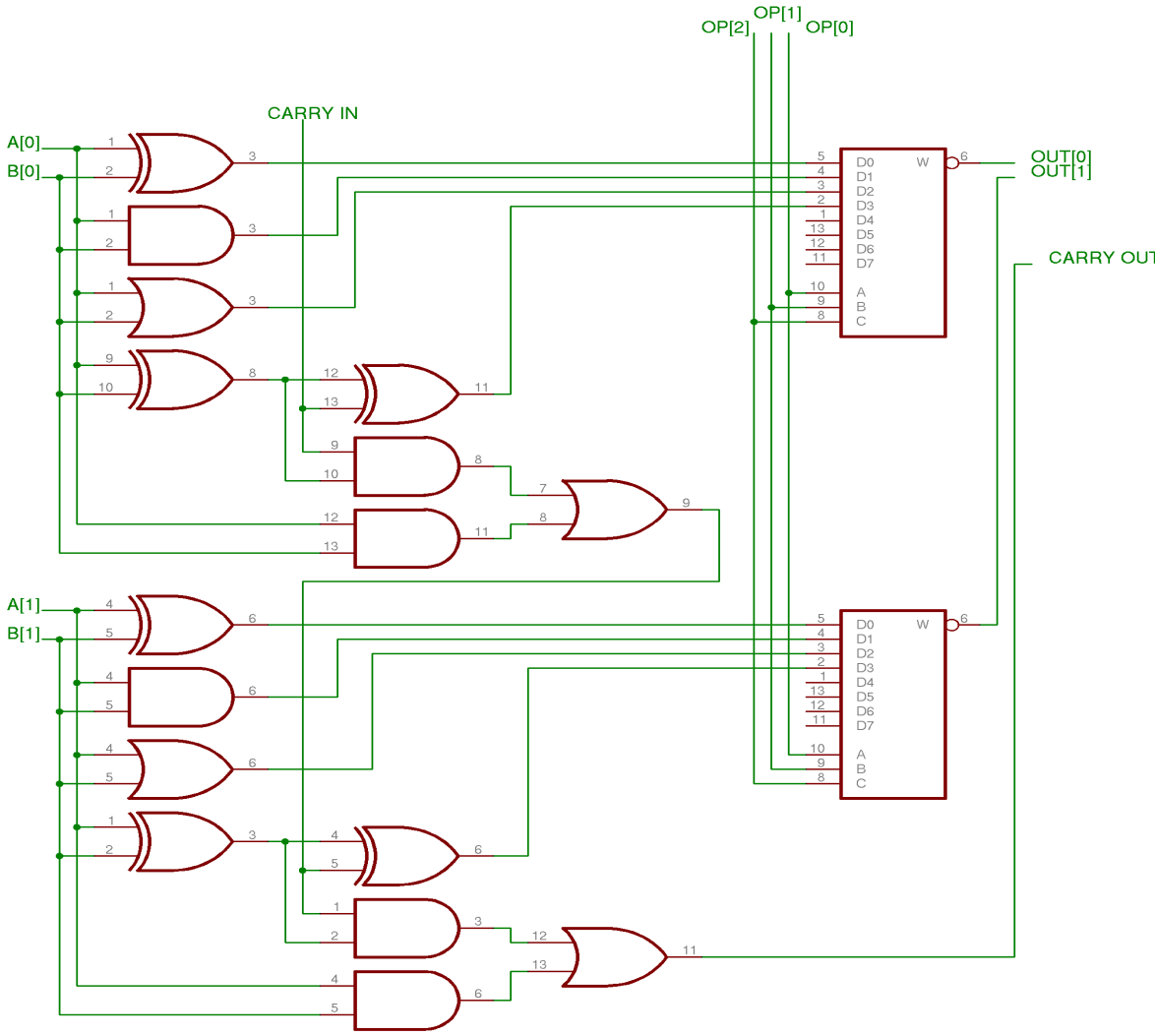



**Realizowane operacje:**

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

Inne możliwe operacje:

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




KAPITAŁ LUDZKI
NARODOWA STRATEGIA SPÓJNOŚCI

Projekt współfinansowany przez Unię Europejską
w ramach Europejskiego Funduszu Społecznego

UNIA EUROPEJSKA
EUROPEJSKI
FUNDUSZ SPOŁECZNY

# 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.

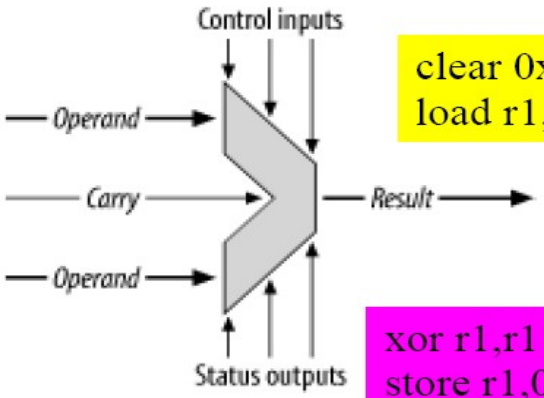
KAPITAŁ LUDZKI
NARODOWA STRATEGIA SPÓJNOŚCI

Projekt współfinansowany przez Unię Europejską
w ramach Europejskiego Funduszu Społecznego

UNIA EUROPEJSKA
EUROPEJSKI
FUNDUSZ SPOŁECZNY

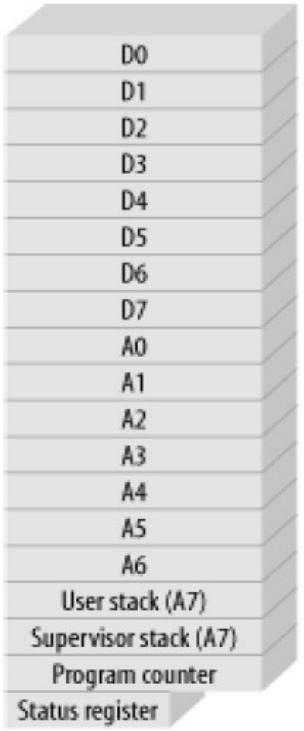
# Architektura procesora (2)

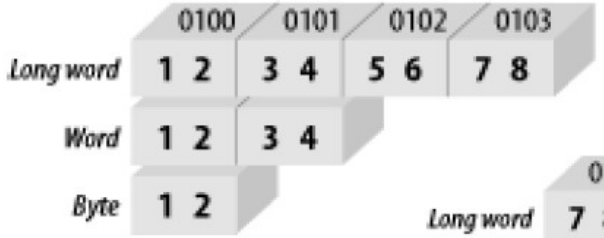
- ALU
  - lista operacji
- zestaw rejestrów
  - A, I, PC, SR, SP,...
- lista rozkazów
  - CISC, RISC
- tryby adresowania
  - R, A, I,...
- przerwania
  - hardware, software
- big/little endian

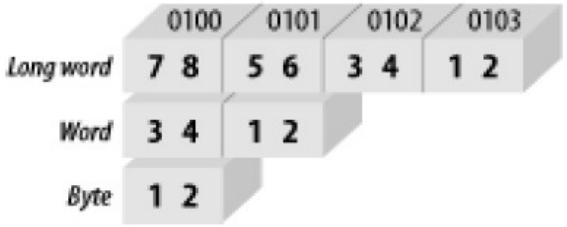



```
clear 0x1000
load r1,#5
```

```
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:**

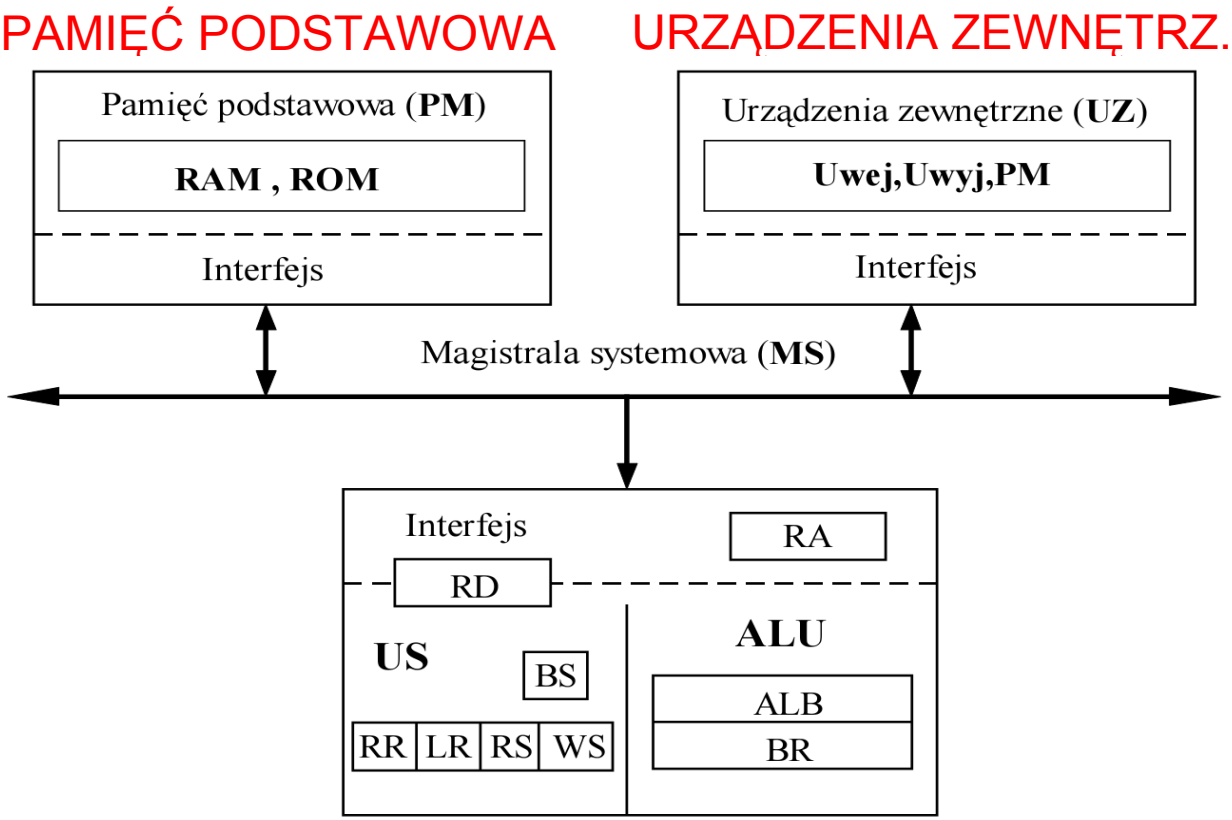
- IBM 801
- PowerPC
- MIPS
- Alpha
- ARM
- Motorola 88000
- ColdFire
- SPARC
- PA-RISC
- Atmel_AVR

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.

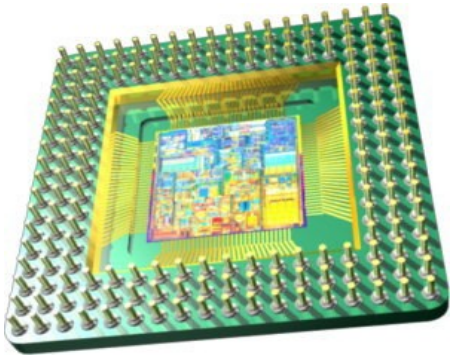
# Architektura systemu komputerowego

Architektura polega na ścisłym podziale komputera na trzy podstawowe części:

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



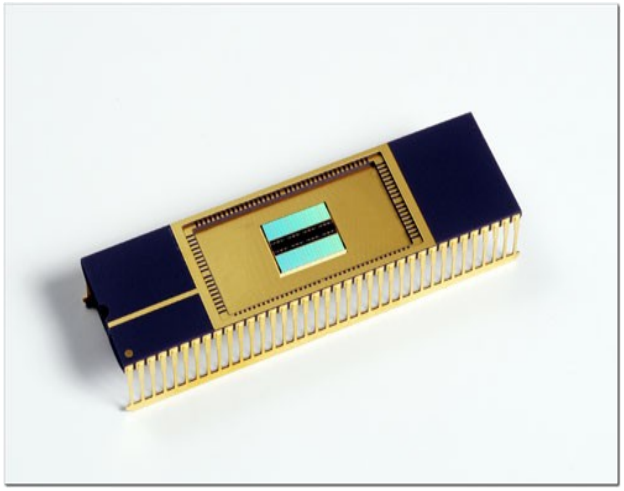
# Magistrale komputera



Magistrala adresowa

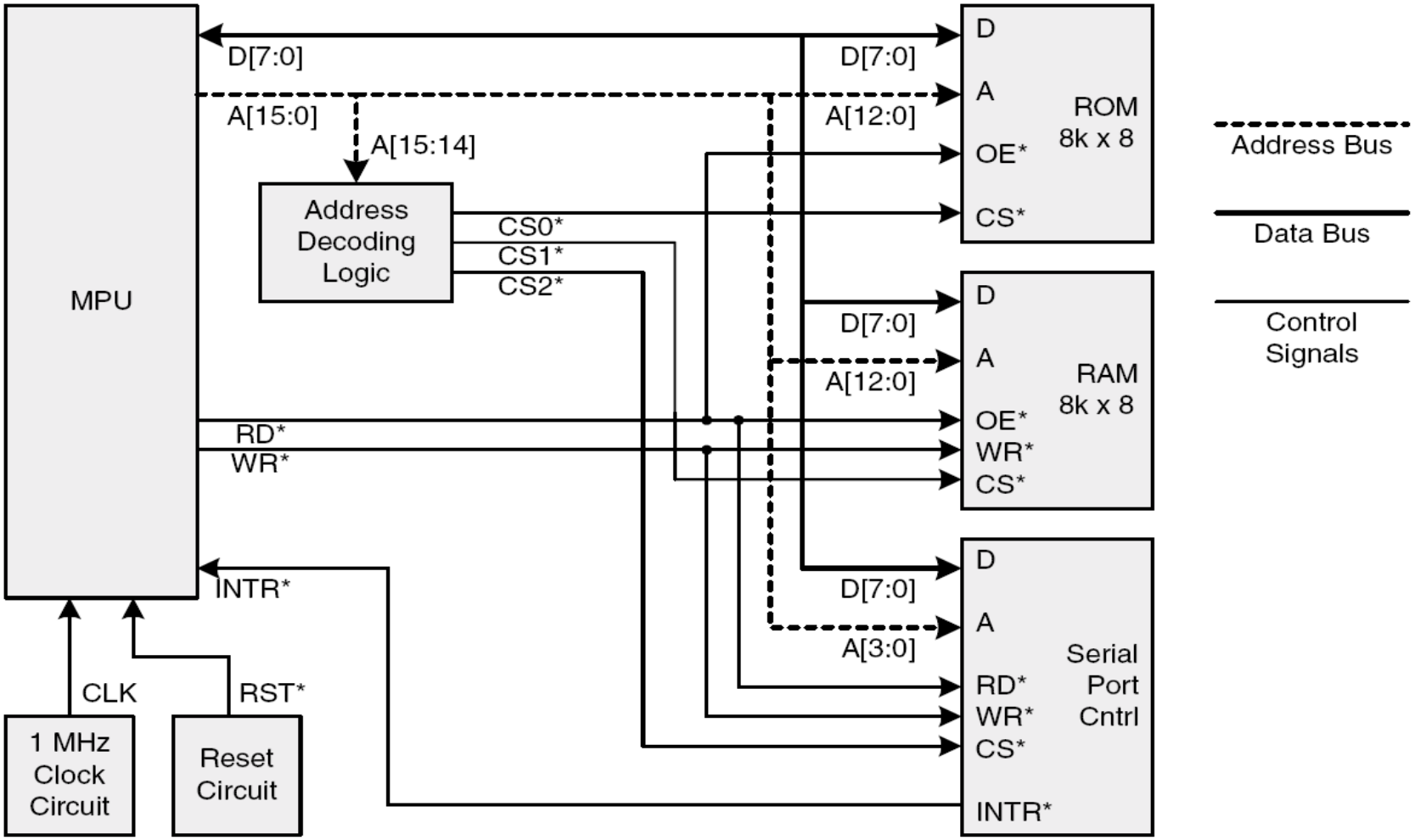
Magistrala danych

Magistrala sterująca



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

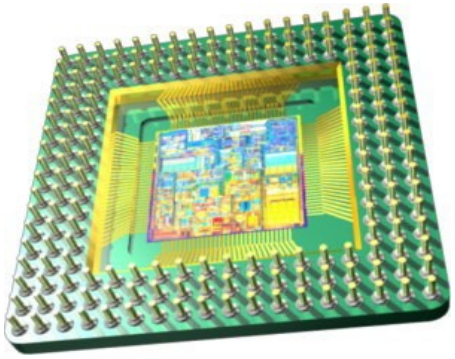
# Przykładowy komputer 8-bitowy



# Architektura von Neumanna

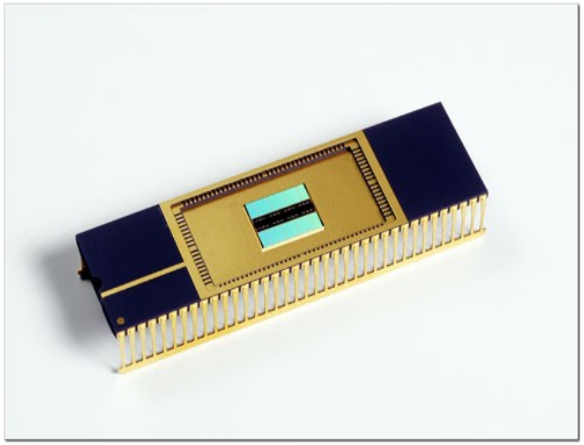
## Cechy architektury von Neumanna:

- 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

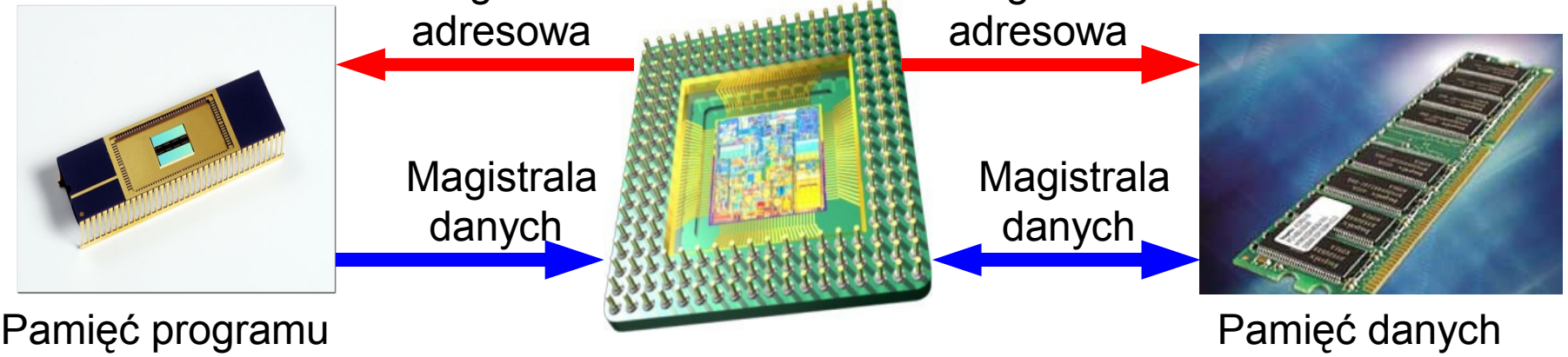


# Architektura Harwardzka

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:**

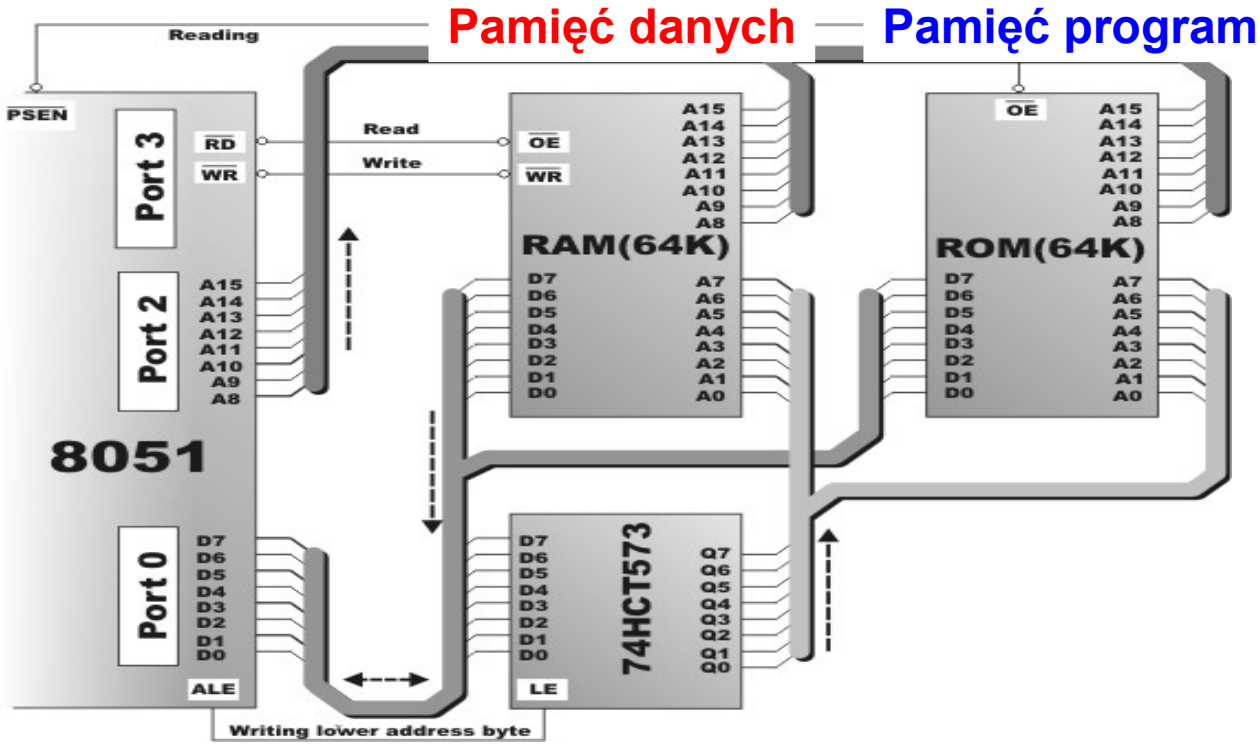
- rozkazy i dane przechowywane są w oddzielnych pamięciach,
- organizacja pamięci może być różna (inne długości słowa danych i rozkazów),
- możliwość pracy równoległej – jednoczesny odczyt danych z pamięci programu oraz danych,
- stosowana w mikrokontrolerach jednoukładowych.



# 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**

# Rejestry procesora

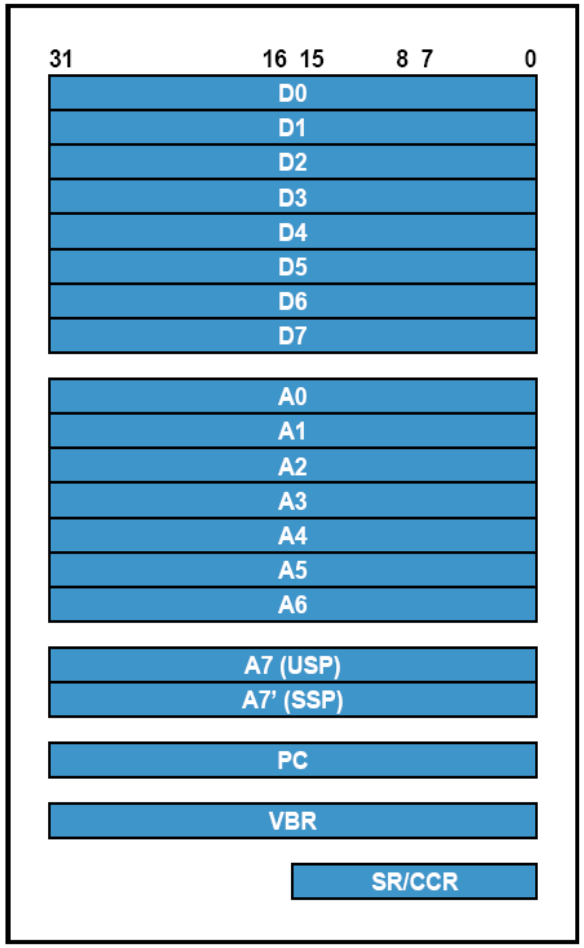
Rejestry procesora stanowią komórki wewnętrznej pamięci procesora o niewielkich rozmiarach (najczęściej 4/8/16/32/64/128 bitów) służące do przechowywania tymczasowych wyników obliczeń, adresów danych w pamięci operacyjnej, konfiguracji, itd...

## Cechy rejestrów procesora:

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

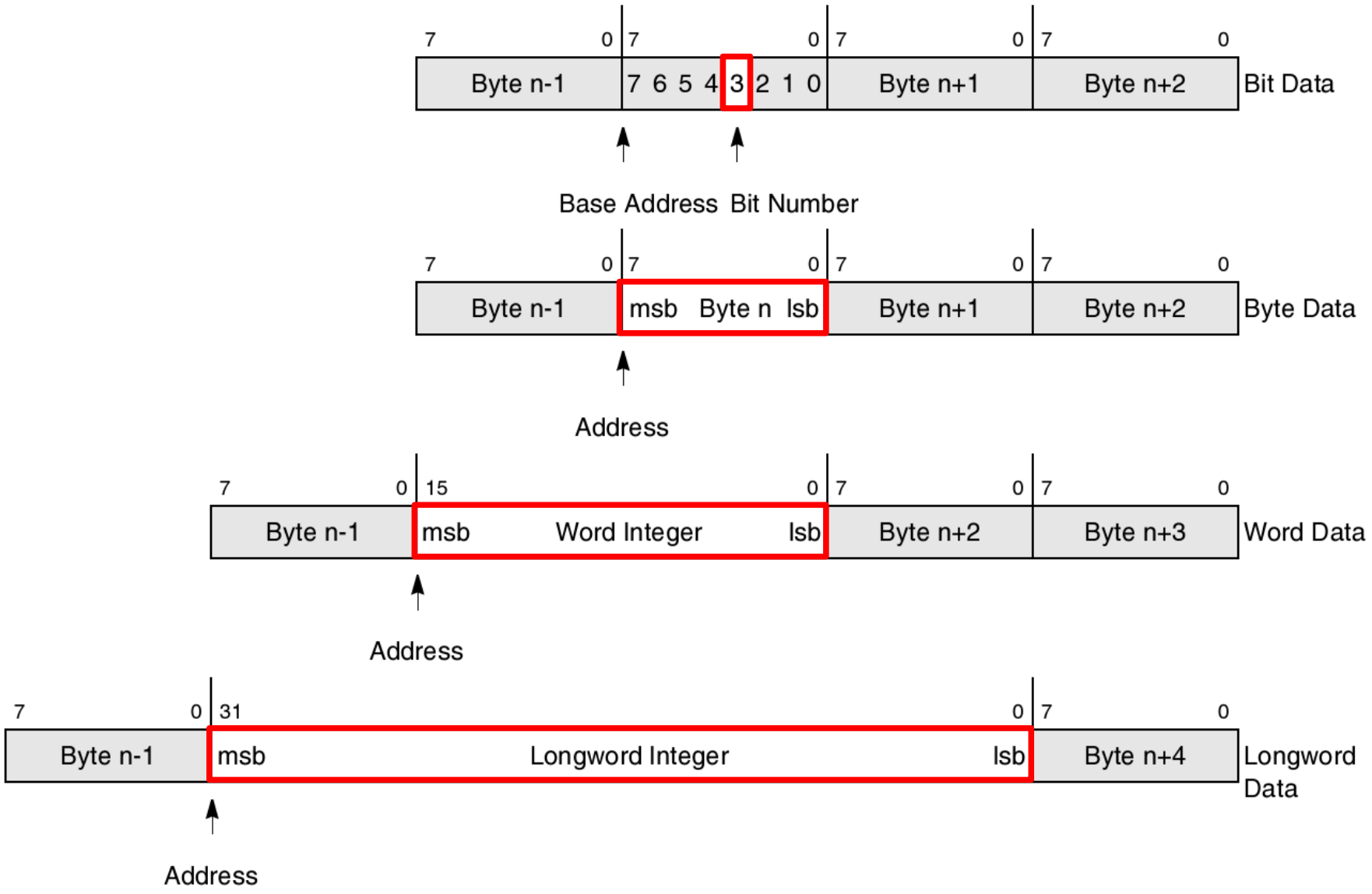
## Rejestry dzielimy na:

- 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

# Model programowy procesora ColdFire



# 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

KAPITAŁ LUDZKI
NARODOWA STRATEGIA SPÓJNOŚCI

Projekt współfinansowany przez Unię Europejską
w ramach Europejskiego Funduszu Społecznego

UNIA EUROPEJSKA
EUROPEJSKI
FUNDUSZ SPOŁECZNY

# 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 |

Podwójne słowo (DW):
**0x1234.5678**

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

KAPITAŁ LUDZKI
NARODOWA STRATEGIA SPÓJNOŚCI

Projekt współfinansowany przez Unię Europejską
w ramach Europejskiego Funduszu Społecznego

UNIA EUROPEJSKA
EUROPEJSKI
FUNDUSZ SPOŁECZNY

# 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

| | 31 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

| | 7 0 | 15 8 | 23 16 | 31 24 |
|---|---|---|---|---|
| 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;
}
```

# Dostęp do rejestrów z poziomu języka C

```
#define BASEREG          volatile unsigned int*    0x4000.0000
#define REGPOINTER       volatile unsigned char*   0x4000.1000
#define OFFSET           0x10

        /* rejestry 8-bitowe */
*REGPOINTER = 0x55;
*(REGPOINTER+OFFSET) = 79;
*REGPOINTER = 010;
*REGPOINTER = 010;
        /* rejestry 8-bitowe */
*(volatile unsigned char*) BASEREG = 0x55;
*((volatile unsigned char*) BASEREG + OFFSET) = 0xFF;
        /* rejestry 32-bitowe */
*BASEREG = 0x55;
 *(BASEREG + OFFSET) =0xFE;
```

# Liczby binarne w C – makro umożliwiające operacje na licz. binarnych

```
/* change a numeric literal into a hex constant  (fill up with leading zeroes up to 32 bits)  8-bit
      constants with maximum value 0x11111111, always fits in unsigned long */

#define HEX__(n) 0x##n##LU

/* Convert 8-bit binary data into compatible with C digit */

#define B8__(x) ((x&0x0000000FLU)?1:0)          \
              +((x&0x000000F0LU)?2:0)           \
              +((x&0x00000F00LU)?4:0)           \
              +((x&0x0000F000LU)?8:0)           \
              +((x&0x000F0000LU)?16:0)          \
              +((x&0x00F00000LU)?32:0)          \
              +((x&0x0F000000LU)?64:0)          \
              +((x&0xF0000000LU)?128:0)

/* for upto 8-bit binary constants */

#define B8(d) ((unsigned char)B8__(HEX__(d)))

/* for upto 32-bit binary constants, MSB first */

#define B32(dmsb,db2,db3,dlsb) (((unsigned long)B8(dmsb)<<24)  \
                              + ((unsigned long)B8(db2)<<16) \
                              + ((unsigned long)B8(db3)<<8) + B8(dlsb))
```

# Liczby binarne w C – użycie makra

```
/* How to use
    B8(01111101)                          data equal to      125   or      0x7D
    B16(10111011,01110101)                data equal to    47989         or      0xBB75
    B32(10100100,10101111,01010,01001)    data equal to 2762934793 or 0xA4AF0A09
*/
```

# Tablica jako plik rejestrów

```
/* Plik rejestowy */
        unsigned int Registers[REGFILESIZE];
        unsigned int* BASEREGISTER = Registers;
        #define PORTNE 0x42
/* Funkcje biblioteki operującej na pliku rejestru */
PrintRegisters();
InitRegisters8b();
InitRegistersBE();
InitRegistersLE();
WriteReg (BASEREGISTER, PORTNE,  49);
WriteReg (BASEREGISTER, PORTNE, 061);
WriteReg (BASEREGISTER, PORTNE, 0x31);
WriteReg (BASEREGISTER, PORTNE, B8(00110001));
PrintReg(BASEREGISTER, PORTNE);
DataFromReg = ReadReg(BASEREGISTER, PORTNE);
```

**gcc -static** Registers.c **-L. -lreg** -o Register

# http://neo.dmcs.p.lodz.pl/arm/index.html

**Materiały do wykładu (materiały będą zamieszczane sukcesywnie, wymagana autoryzacja)**

**1. Systemy wbudowane - wprowadzenie**

* Przykładowa implementacja rejestrów procesora z poziomu języka C oraz biblioteka libreg (gcc 4.1.2)

KAPITAŁ LUDZKI
NARODOWA STRATEGIA SPÓJNOŚCI

Projekt współfinansowany przez Unię Europejską
w ramach Europejskiego Funduszu Społecznego

UNIA EUROPEJSKA
EUROPEJSKI
FUNDUSZ SPOŁECZNY

# Operowanie bitami rejestrów (1)

volatile unsigned char* PORTA=0x4010.000A;

```
*PORTA    = 0x1;
*PORTA    = 7;
*PORTA    = 010;
*PORTA    = *PORTA  |  0x2;
*PORTA   |= 0x1 | 0x2 | 0x8 ;
*PORTA  &=  ~(0x2 | 0x4);
*PORTA   ^=    (0x1 | 0x2);
*PORTA   ^=     0x3;
If (*PORTA & (0x1 | 0x4)) == 0  {...}
while (*PORTA != 0x6)   {...}
do {...} while (*PORTA & 0x4)
```

# Operowanie bitami rejestrów (2)

```
#define PB0   0x1
#define PB1   0x2
#define PB2   1<<2
#define PB3   1<<3

volatile unsigned char* PORTA=0x4010.000A;

*PORTA   |= PB1 | PB2;
*PORTA &=  ~(PB1 | PB2);
*PORTA  ^=  (PB1 | PB2);
If (*PORTA & (PB1 | PB2)) == 0

enum {PB0=1<<0, PB1=1<<2, PB2=1<<3, PB3=1<<3};
```

# Operowanie bitami rejestrów (3)

```
volatile unsigned char* PORTA=0x4010.000A;

/* makro generujące maskę bitową */
#define BIT(x)      (1 << (x))
*PORTA  |=  BIT(0);
*PORTA &=~BIT(1);
*PORTA  ^= BIT(2);

/* makra ustawiające lub zerujące bity */
#define SETBIT(P, B)        (P)   |=   BIT(B)
#define CLRBIT(P, B)        (P)  &= ~BIT(B)

SETBIT(*PORTA, 7);
CLRBIT(*PORTA, 2);
```

# Operacje łączenia rejestrów

```
int main(void) {
        unsigned char reg1=0x15, reg2=0x55;
        unsigned char = reg3, reg4;
        unsigned int tmp;
        /* operacja „łączenia” rejestrów */
        tmp = reg1;
        tmp = tmp<<8 | reg2;

        /* operacja „rozłączenia” rejestrów */
        reg3 = tmp>>8;              /* uwaga na liczby ze znakiem */
        reg4 = tmp & 0xFF;

}
```

KAPITAŁ LUDZKI
NARODOWA STRATEGIA SPÓJNOŚCI

Projekt współfinansowany przez Unię Europejską
w ramach Europejskiego Funduszu Społecznego

UNIA EUROPEJSKA
EUROPEJSKI
FUNDUSZ SPOŁECZNY

# Rejestry odwzorowane w strukturze

```
typedef volatile unsigned int AT91_REG;       // Hardware register definition
typedef struct _AT91S_PIO {
        AT91_REG        PIO_PER;                // PIO Enable Register, 32-bit register
        AT91_REG        PIO_PDR;                // PIO Disable Register
        AT91_REG        PIO_PSR;                // PIO Status Register
        AT91_REG        Reserved1[1];           //
        AT91_REG        PIO_IFER;               // Input Filter Enable Register
        AT91_REG        PIO_IFDR;               // Input Filter Disable Register
        AT91_REG        PIO_IFSR;               // Input Filter Status Register
        AT91_REG        Reserved2[1];           //
} AT91S_PIO, *AT91PS_PIO;
        /* blok rejestrów portów I/O PIOA...PIOE */
#define AT91C_BASE_PIOA     (AT91PS_PIO)    0xFFFFF200     // (PIOA) Base Address
        /* maska zerowego bitu portu PA */
#define AT91C_PIO_PA0       (1 <<  0)       // Pin Controlled by PA0
```

**Jak ustawić bity 0 i 19 rejestru PIO_PER ?**

```
AT91C_BASE_PIOA->PIO_PER = AT91C_PIO_PA0 | AT91C_PIO_PA19;
```

# Pola bitowe - odwzorowanie rejestru w strukturze

```
Struct Port_4bit {
unsigned Bit_0          :    1;
unsigned Bit_1          :    1;
unsigned Bit_2          :    1;
unsigned Bit_3          :    1;
unsigned Bit_Filler     :    4;
};
#define PORTC (*(Port_4bit*)0x4010.0002)
int i = PORTC.Bit_0;      /* read data */
PORTC.Bit_2 = 1;          /* write data */

Port_4bit* PortTC = (Port_4bit*) 0x4010.000F;
int i = PortTC->Bit_0;
PortTC->Bit_0 = 1;
```

- Pola bitowe umożliwiają upakowanie danych – wykorzystanie pojedynczych bitów
- Wzrost złożoności kodu potrzebnego do obsługi „rejestrów”
- Pola bitowe mogą zostać rozmieszczone w pamięci w sposób zależny od kompilatora i architektury procesora
- Nie można stosować makra ***offsetof*** podającego położenie danej w strukturze
- Nie można używać makra ***sizeof*** do wyliczenia rozmiaru pola bitowego
- Nie można definiować tablic na polach bitowych

# Unie – dostęp do rejestrów pełniących różne funkcje

```
extern volatile union {
        struct {
                unsigned EID16      :1;
                unsigned EID17      :1;
                unsigned            :1;
                unsigned EXIDE      :1;
                unsigned            :1;
                unsigned SID0       :1;
                unsigned SID1       :1;
                unsigned SID2       :1;
        };
        struct {
                unsigned            :3;
                unsigned EXIDEN     :1;
        };
} RXF3SIDLbits_;
```

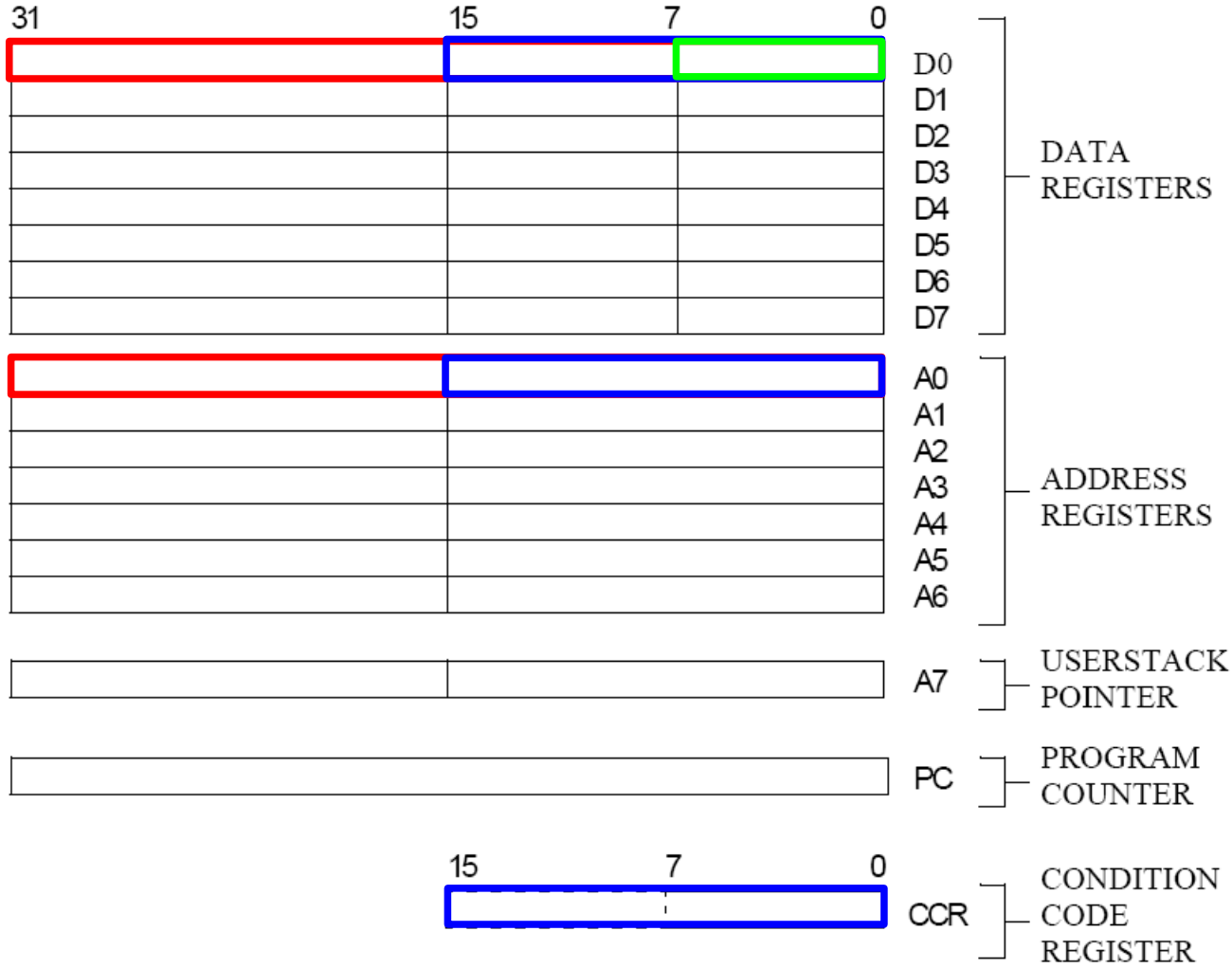
Struktury znajduja się pod tym samym adrese:

```
#define RXF3SIDLbits
    (*(Port_RXF3SIDLbits_*)0x4010.0000)
```

Dostęp do danych umieszczonych w strukturze:

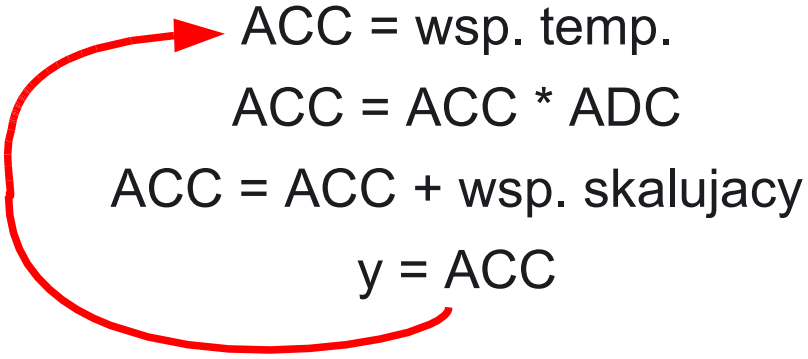
```
/* dostęp do rejestru z pierwszej struktury */
    RXF3SIDLbits.EID16 = 1;

/* dostęp do rejestru z drugiej struktury */
    RXF3SIDLbits.EXIDEN = 0;
```

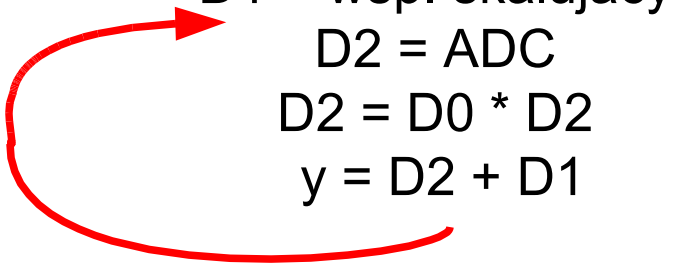
# Model programowy procesora ColdFire



# Przykład użycia rejestrów danych

y = wsp. temp. * ADC + wsp. skalujacy



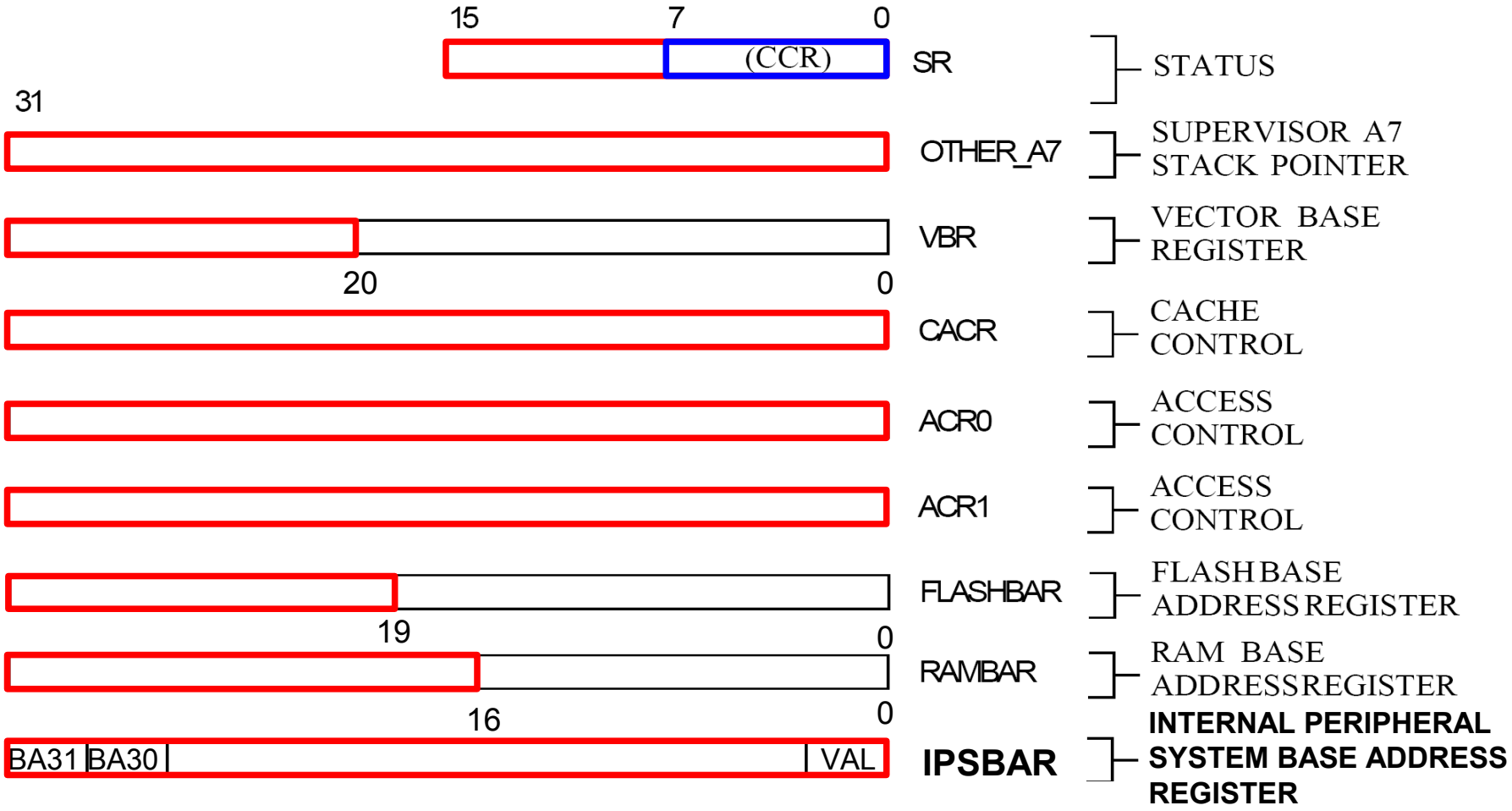

KAPITAŁ LUDZKI
NARODOWA STRATEGIA SPÓJNOŚCI

Projekt współfinansowany przez Unię Europejską
w ramach Europejskiego Funduszu Społecznego

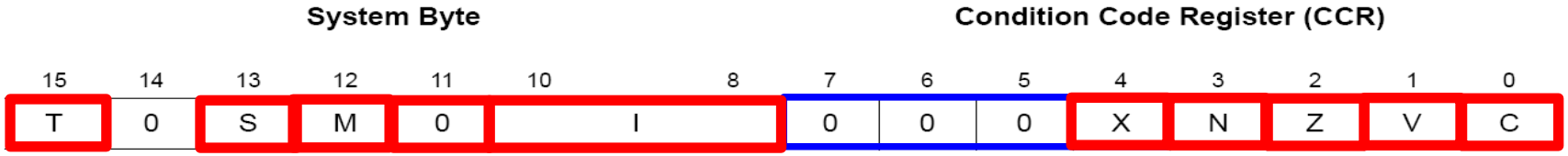
UNIA EUROPEJSKA
EUROPEJSKI
FUNDUSZ SPOŁECZNY

# Model programowy procesora ColdFire

## Rejestry dostępne w trybie superużytkownika



# 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 | 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<br>Set to the value of the C bit for arithmetic operations; otherwise not affected. |

# 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 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

# 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

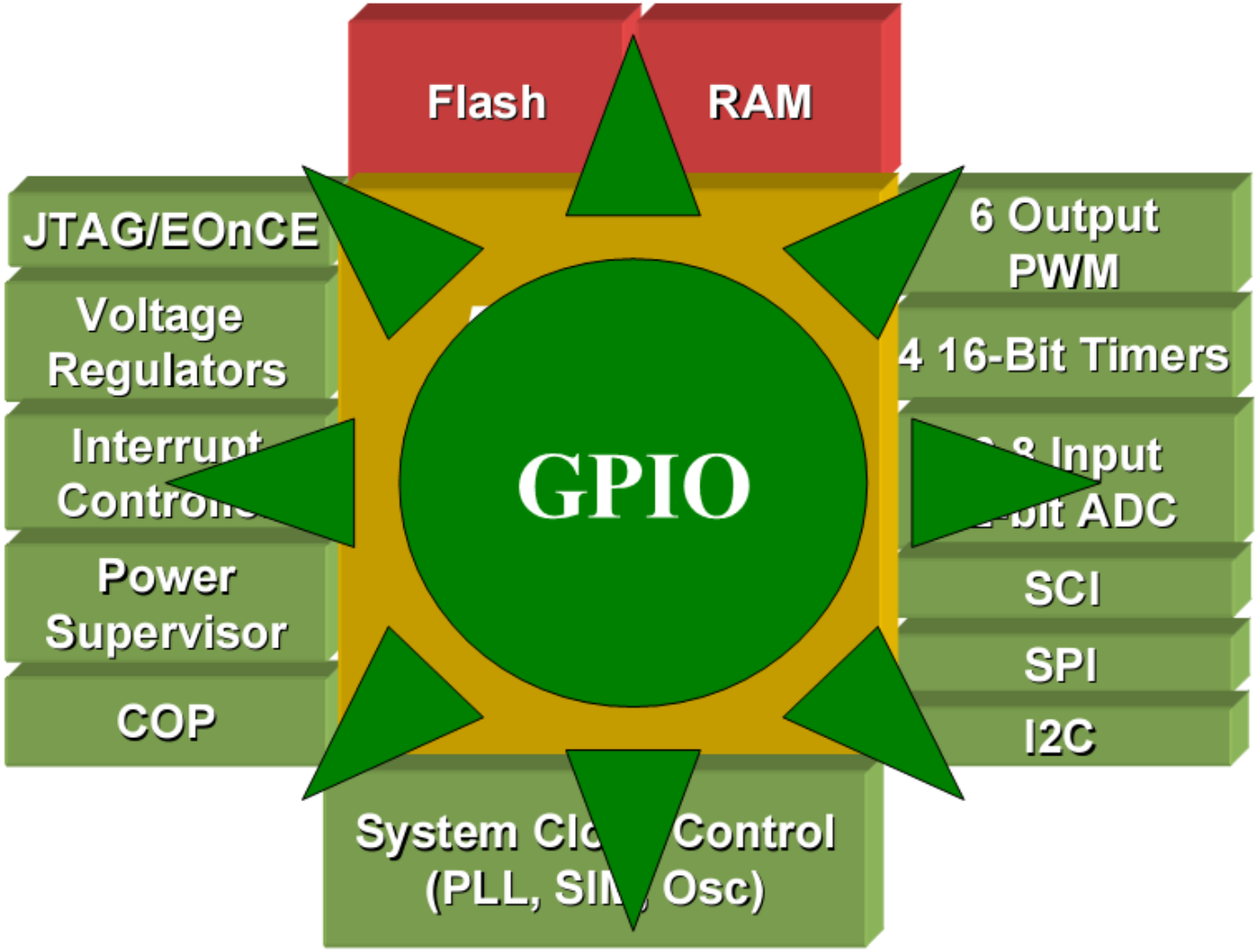
# Moduł portów wejścia-wyjścia (General Purpose I/O module)

KAPITAŁ LUDZKI
NARODOWA STRATEGIA SPÓJNOŚCI

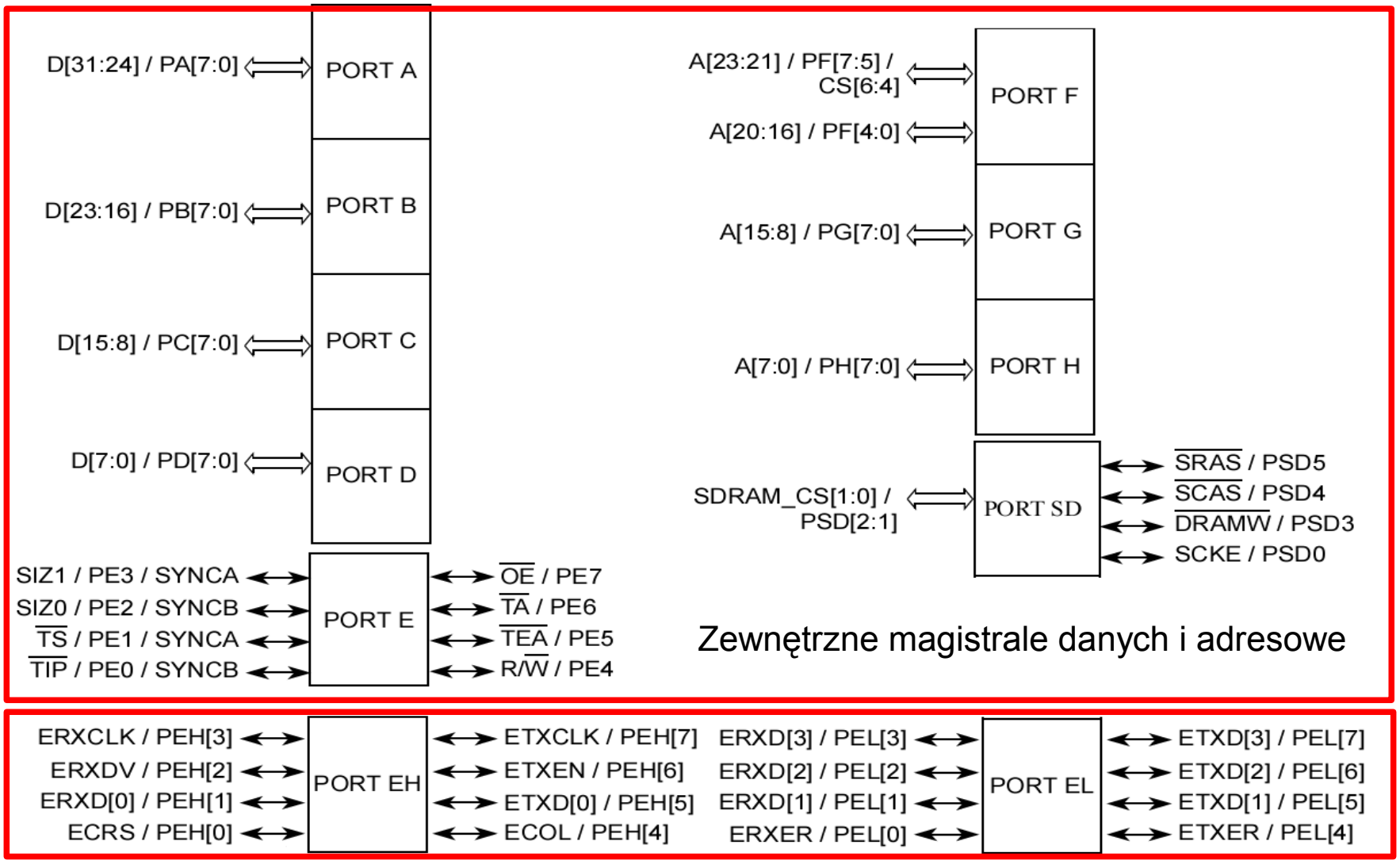
Projekt współfinansowany przez Unię Europejską
w ramach Europejskiego Funduszu Społecznego

UNIA EUROPEJSKA
EUROPEJSKI
FUNDUSZ SPOŁECZNY

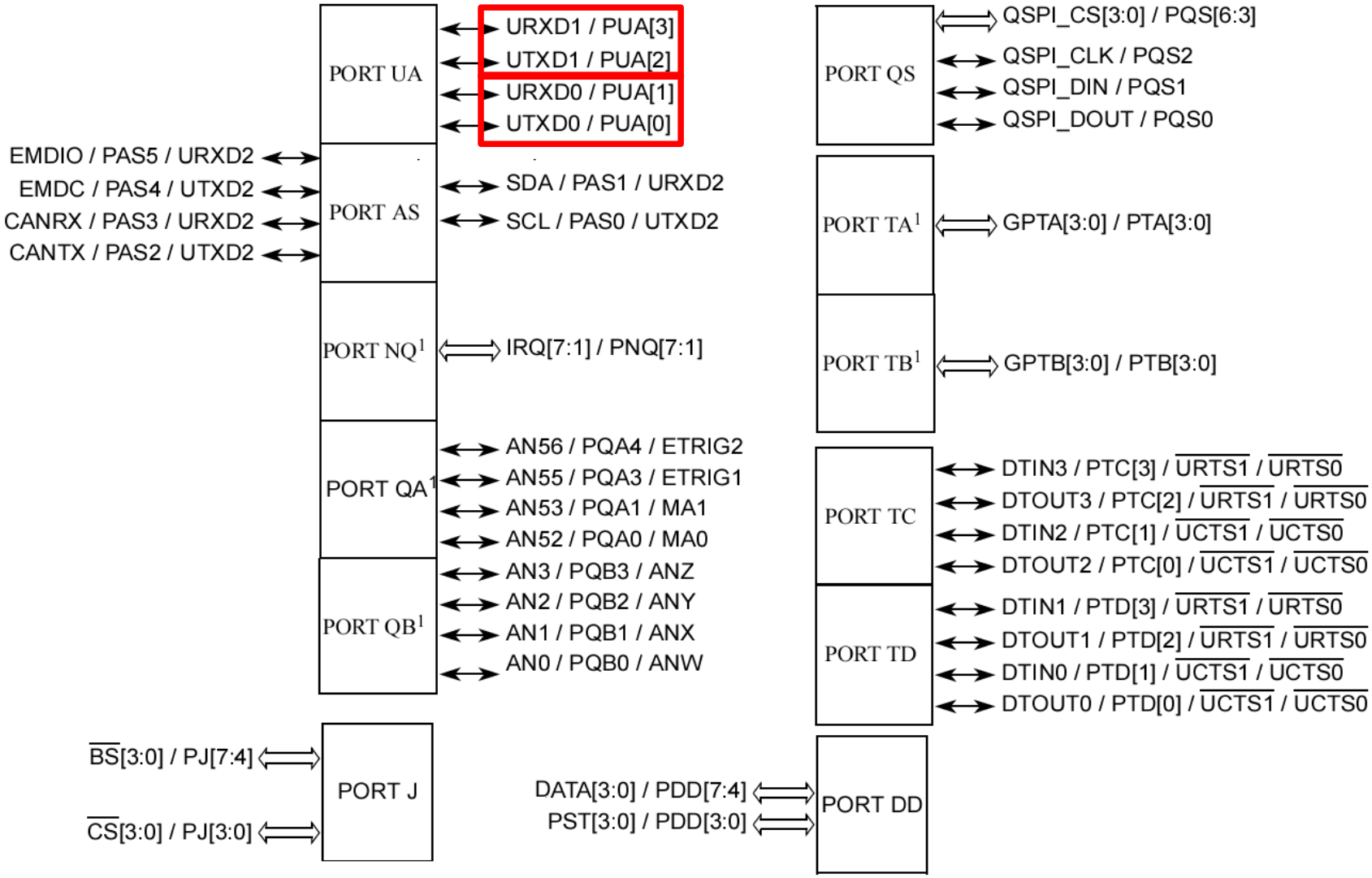
# Moduł portów I/O (1)



# Moduł portów I/O (2)



# Moduł portów I/O (3)



# Rejestry sterujące

**PnPAR** - rejestr kontrolujący przeznaczenie portu

**DDRn** - Rejestr kontrolujący kierunek sygnałów portu I/O

**PORTn** - rejestr kontrolujący stan wyprowadzeń wyjściowych

**PORTnP** - rejestr odwzorowujący stan wyprowadzenia I/O

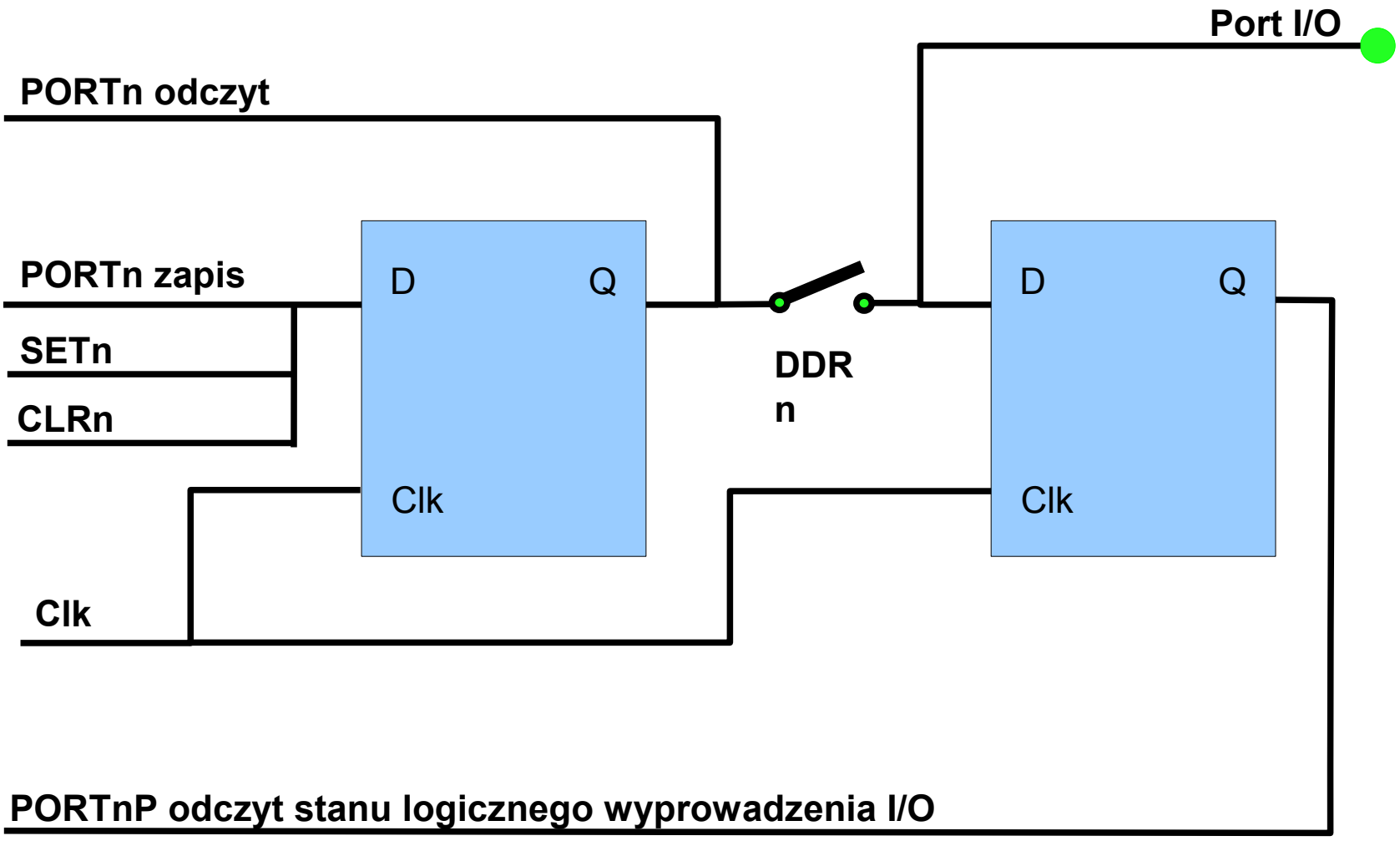
**SETn/CLRn** - Rejestr służący do ustawiania/zerowania przerzutnika wyjściowego

KAPITAŁ LUDZKI
NARODOWA STRATEGIA SPÓJNOŚCI

Projekt współfinansowany przez Unię Europejską
w ramach Europejskiego Funduszu Społecznego

UNIA EUROPEJSKA
EUROPEJSKI
FUNDUSZ SPOŁECZNY

# Schemat blokowy portu I/O



# Przykładowe rejestry sterujące modułu I/O

| | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Field | PORT*n*7 | PORT*n*6 | PORT*n*5 | PORT*n*4 | PORT*n*3 | PORT*n*2 | PORT*n*1 | PORT*n*0 |
| Reset | 1111_1111 | | | | | | | |
| R/W: | R/W | | | | | | | |
| Address | IPSBAR + 0x10_0000 (PORTA), 0x10_0001 (PORTB), 0x10_0002 (PORTC), 0x10_0003 (PORTD), 0x10_0004 (PORTE), 0x10_0005 (PORTF), 0x10_0006 (PORTG), 0x10_0007 (PORTH), 0x10_0008 (PORTJ), 0x10_0009 (PORTDD), 0x10_000A (PORTEH), 0x10_000B (PORTEL) | | | | | | | |

**Figure 26-2. Port Output Data Registers (8-bit)**

| | 7 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|
| Field | — | | PORT*n*3 | PORT*n*2 | PORT*n*1 | PORT*n*0 |
| Reset | 0000_1111 | | | | | |
| R/W: | R | | R/W | | | |
| Address | IPSBAR + 0x10_000F (PORTTC), 0x10_0010 (PORTTD), 0x10_0011 (PORTUA) | | | | | |

**Figure 26-5. Port Output Data Registers (4-bit)**

# Port przerwań zewnętrznych (EPORT)

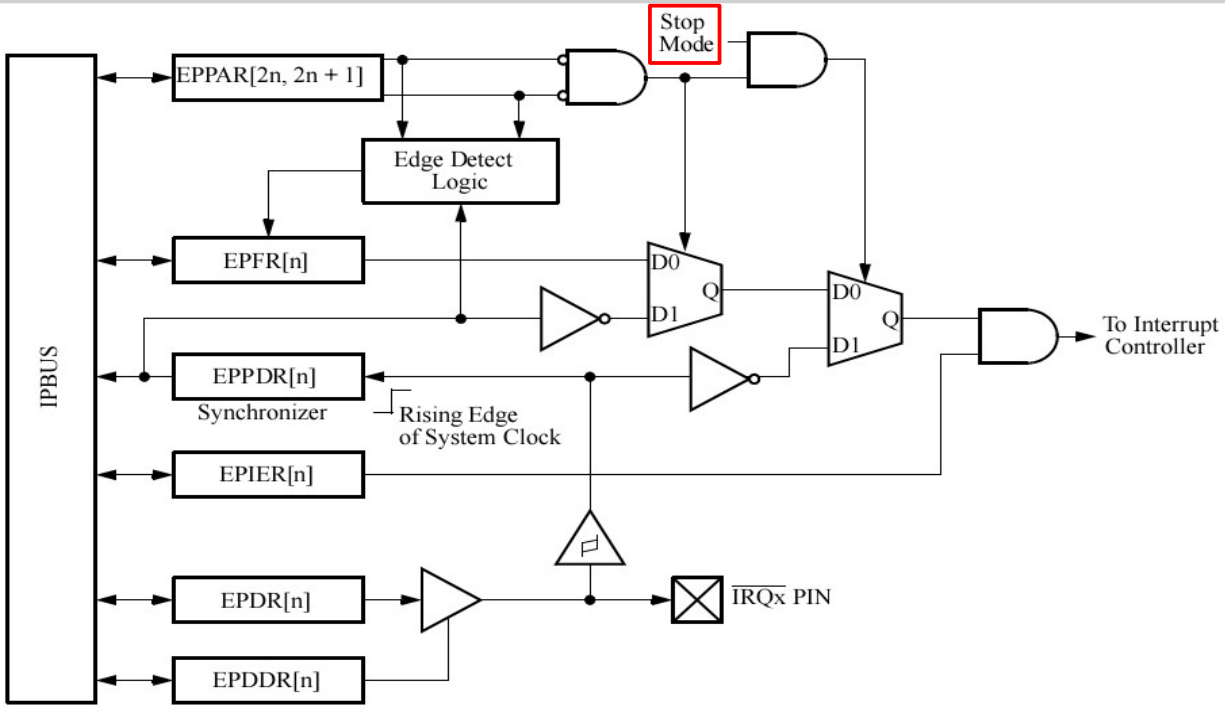



**Table 11-2. Edge Port Module Memory Map**

| IPSBAR Offset | Bits 15–8 | Bits 7–0 | Access [1] |
|---|---|---|---|
| 0x0013_0000 | EPORT Pin Assignment Register (EPPAR) | | S |
| 0x0013_0002 | EPORT Data Direction Register (EPDDR) | EPORT Interrupt Enable Register (EPIER) | S |
| 0x0013_0004 | EPORT Data Register (EPDR) | EPORT Pin Data Register (EPPDR) | S/U |
| 0x0013_0006 | EPORT Flag Register (EPFR) | Reserved [2] | S/U |

# Zakresy napięć wejściowych

## Table 33-1. Absolute Maximum Ratings

| Rating | Symbol | Value | Unit |
|---|---|---|---|
| Supply Voltage | $V_{DD}$ | – 0.3 to +4.0 | V |
| Clock Synthesizer Supply Voltage | $V_{DDPLL}$ | – 0.3 to +4.0 | V |
| RAM Memory Standby Supply Voltage | $V_{STBY}$ | – 0.3 to + 4.0 | V |
| Flash Memory Supply Voltage | $V_{DDF}$ | – 0.3 to +4.0 | V |
| Flash Memory Program / Erase Supply Voltage | $V_{PP}$ | – 0.3 to + 6.0 | V |
| Analog Supply Voltage | $V_{DDA}$ | – 0.3 to +6.0 | V |
| Analog Reference Supply Voltage | $V_{RH}$ | – 0.3 to +6.0 | V |
| Analog ESD Protection Voltage | $V_{DDH}$ | – 0.3 to +6.0 | V |
| Digital Input Voltage | $V_{IN}$ | – 0.3 to + 6.0 | V |
| Analog Input Voltage | $V_{AIN}$ | – 0.3 to + 6.0 | V |
| EXTAL pin voltage | $V_{EXTAL}$ | 0 to 3.3 | V |
| XTAL pin voltage | $V_{XTAL}$ | 0 to 3.3 | V |

# Zakres przedmiotu

- Systemy mikroprocesorowe, systemy wbudowane
- Laboratorium
- Rodzina procesorów ARM
- Urządzenia peryferyjne
- Pamięci i dekodery adresowe
- Programy wbudowane na przykładzie procesorów ARM
- Metodyki projektowania systemów wbudowanych
- Systemy czasu rzeczywistego

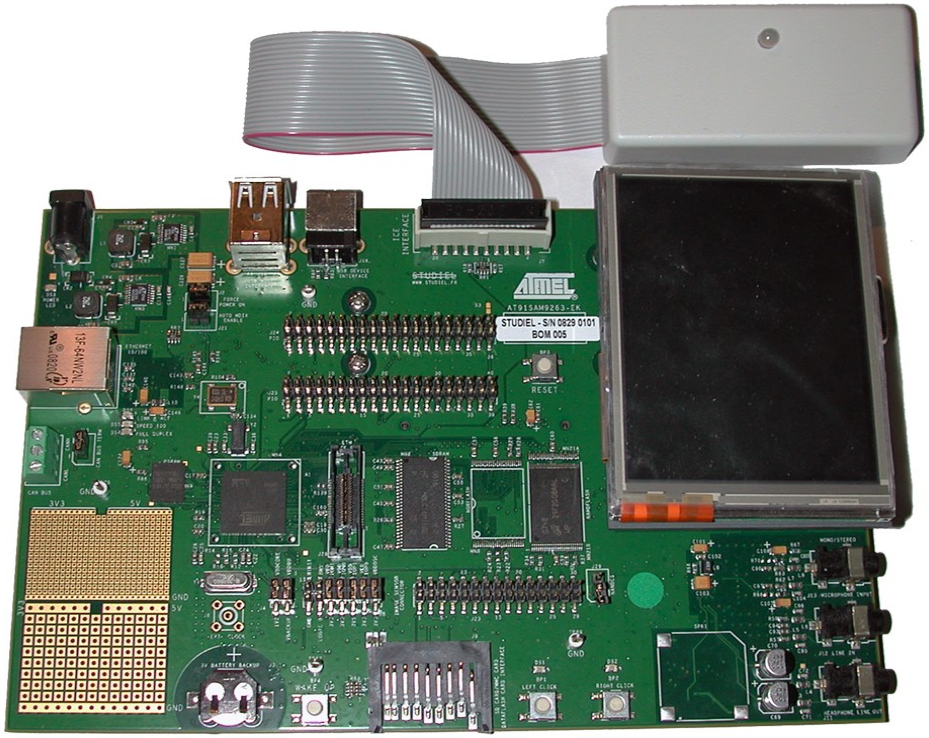
KAPITAŁ LUDZKI
NARODOWA STRATEGIA SPÓJNOŚCI

Projekt współfinansowany przez Unię Europejską
w ramach Europejskiego Funduszu Społecznego

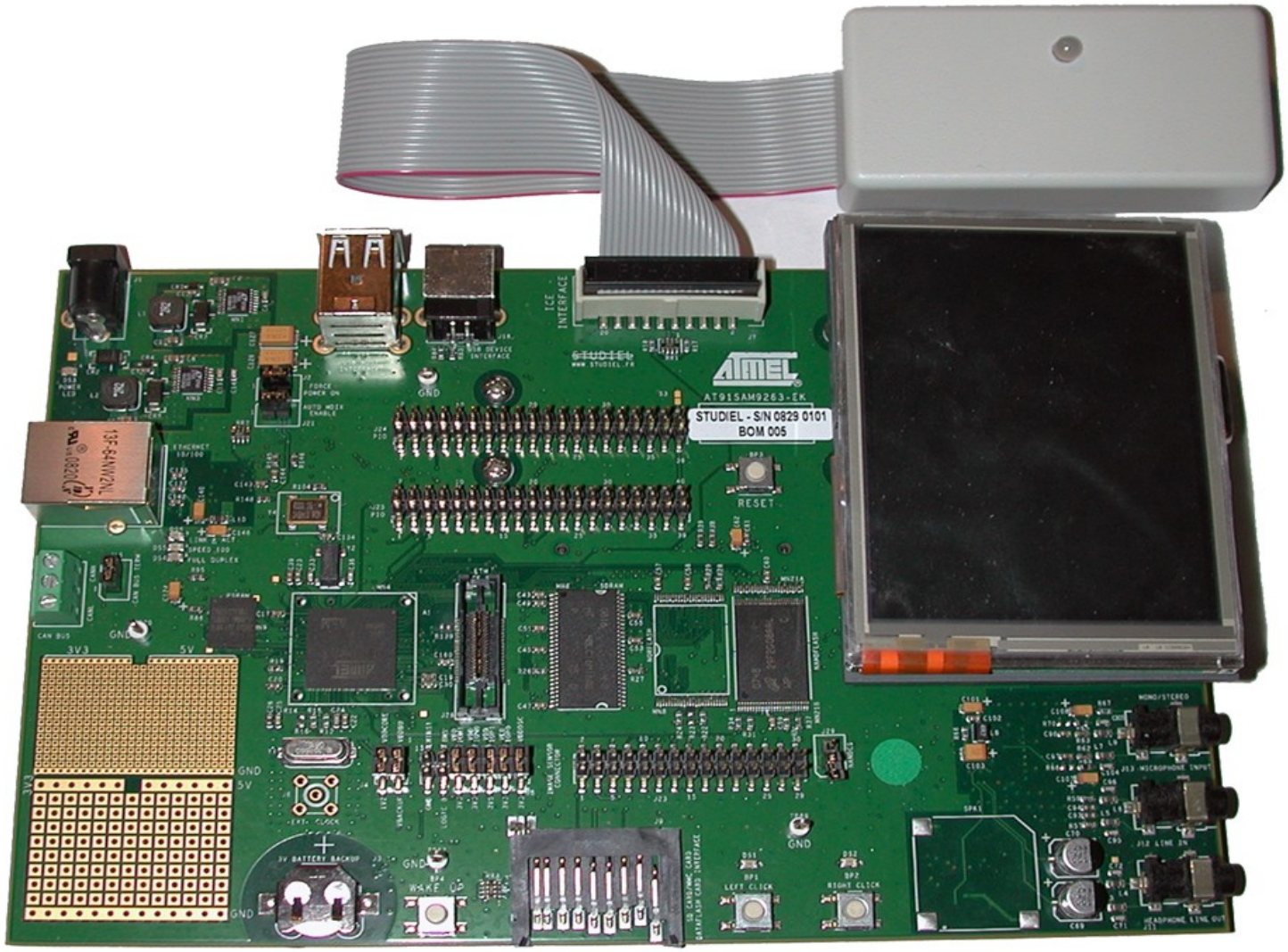
UNIA EUROPEJSKA
EUROPEJSKI
FUNDUSZ SPOŁECZNY

# Zestaw ewaluacyjny firmy MSC (1)

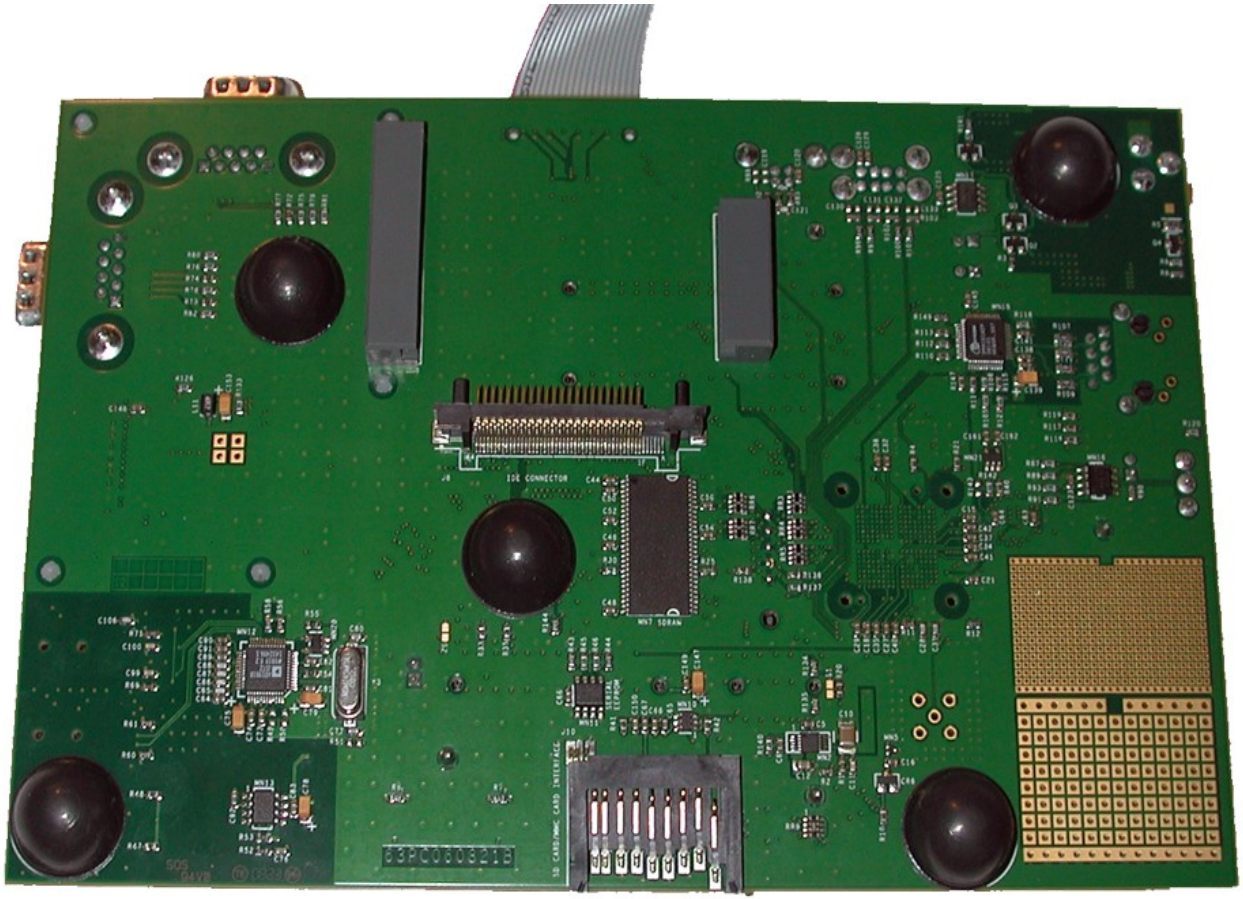
- **Procesor z rdzeniem ARM9TDMI firmy ATMEL:** AT91SAM9263
- **Dostępne pamięci:** 64MB SDRAM, 256MB NAND FLASH, 4 MB DataFlash, FlashCard slots
- **Dostępne interfejsy:** Ethernet 100-base TX, USB FS device, 2 x USB FS Host, CAN 2.0B, EIA RS232,
- **Wyświetlacz:** 3.5" 1/4 VGA TFT LCD z ekranem dotykowym
- **Kodek audio:** AC97 Audio DAC
- **Debug intefece:** JTAG
- **Interfejs do programowania:** JTAG, Free Atmel SAM-BA tools
- **Złącze:** SD/SDIO/MMC card slot
- **Oznaczenie producenta:** AT91SAM9263-STARTUP-PAKET



# Zestaw ewaluacyjny firmy MSC (2)



# Zestaw ewaluacyjny firmy MSC (3)



# Narzędzia GNU

**GNU ARM toolchain –** narzędzia dla procesorów z rodziny ARM dostępne w ramach licencji GNU GPL (General Public License).

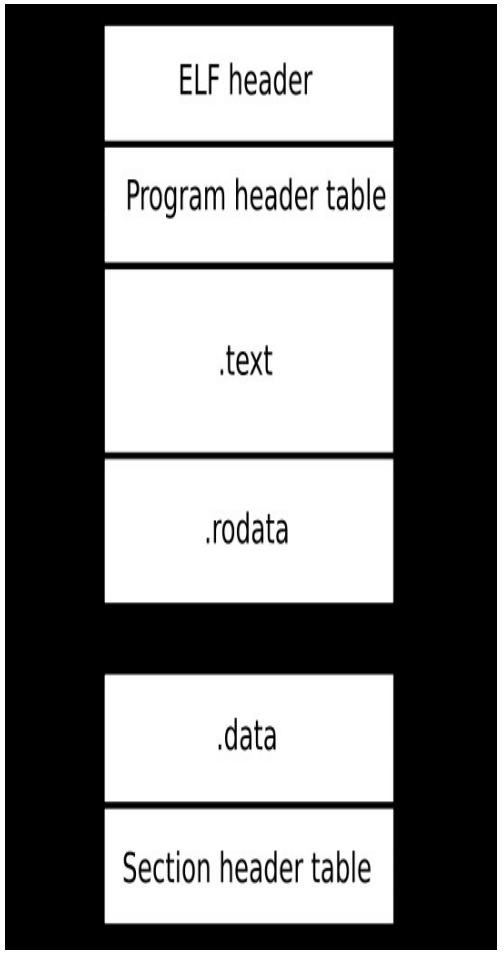
Obecnie dostępne narzędzia (***http://www.gnuarm.org/***):

- GCC-4.3 toolchain (Linux):
  - Kompilator: gcc-4.3.2
  - Przydatne narzędzia: binutils-2.19
  - Biblioteki C/C++: newlib-1.16
  - Debuger zgodny z GDB: insight-6.8
- Cygwin (Windows):
  - binutils-2.19, gcc-4.3.2-c-c++, newlib-1.16.0, insight-6.8, setup.exe
- Debuger JTAG z interfejsem USB (informacje dostępne w Katedrze DMCS)

# COFF vs ELF

- **COFF (Common Object File Format) –** standard plików wykonywalnych, relokowalnych i bibliotek dynamicznych opracowany na potrzeby systemów operacyjnych bazujących na systemie Unix. COFF miał zastąpić standard plików **a.out**. Wykorzystywany na różnych systemach, również Windows. Obecnie standard COFF wypierany jest przez pliki ELF.

- **ELF (Executable and Linkable Format) –** standard plików wykonywalnych, relokowalnych, bibliotek dynamicznych i zrzutów pamięci wykorzystywany na różnych komputerach i systemach operacyjnych, np.: rodzina x86, PowerPC, OpenVMS, BeOS, konsole PlayStation Portable, PlayStation 2, PlayStation 3, Wii, Nintendo DS, GP2X, AmigaOS 4 oraz Symbian OS v9.
- Przydatne narzędzia:
  - readelf
  - elfdump
  - objdump




Źródło: wikipedia

# Wnętrza plików...

**file** at91sam9263_getting_started_sdram.elf

at91sam9263_getting_started_sdram.elf: ELF 32-bit LSB executable, ARM, version 1, statically linked, not stripped

**file** main.o

main.o: ELF 32-bit LSB relocatable, ARM, version 1, not stripped

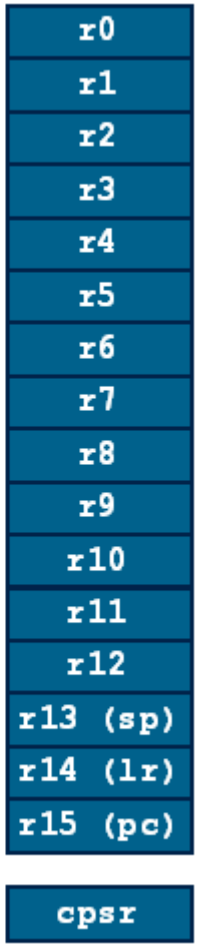
**arm-elf-objdump** -d at91sam9263_getting_started_sdram.elf | less

```
at91sam9263_getting_started_sdram.elf:     file format elf32-littlearm
Disassembly of section .text:
20000000 <_stext>:
20000000:      e59ff09c        ldr     pc, [pc, #156]    ; 200000a4 <_lp_data+0xc>
20000004 <_low_level_init>:
20000004:      e59f209c        ldr     r2, [pc, #156]    ; 200000a8 <_lp_data+0x10>
20000008:      e8920003        ldm     r2, {r0, r1}
2000000c:      e1a0d001        mov     sp, r1
20000010:      e1a0e00f        mov     lr, pc
20000014:      e12fff10        bx      r0
```

# Rejestry procesora a GDB

## (gdb) info r

```
r0             0x2              0x2
r1             0x20000ba4       0x20000ba4
r2             0x57b            0x57b
r3             0x270f           0x270f
r4             0x300069         0x300069
r5             0x3122dc         0x3122dc
r6             0x1000           0x1000
r7             0x800bc004       0x800bc004
r8             0x3122c4         0x3122c4
r9             0x407c81a4       0x407c81a4
r10            0x441029ab       0x441029ab
r11            0x313f2c         0x313f2c
r12            0x313f30         0x313f30
sp             0x313f18         0x313f18
lr             0x20000a7c       0x20000a7c
pc             0x20000474       0x20000474 <delay+60>
fps            0x0              0x0
cpsr           0x80000053       0x80000053
```



KAPITAŁ LUDZKI
NARODOWA STRATEGIA SPÓJNOŚCI

UNIA EUROPEJSKA
EUROPEJSKI
FUNDUSZ SPOŁECZNY

# „Procesory ARM w systemach wbudowanych”
# „Wprowadzenie do przedmiotu”

Prezentacja jest współfinansowana przez
Unię Europejską w ramach
Europejskiego Funduszu Społecznego w projekcie pt.

*„Innowacyjna dydaktyka bez ograniczeń - zintegrowany rozwój Politechniki Łódzkiej - zarządzanie Uczelnią, nowoczesna oferta edukacyjna i wzmacniania zdolności do zatrudniania osób niepełnosprawnych”*

Prezentacja dystrybuowana jest bezpłatnie