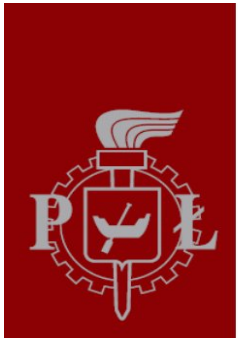


Politechnika Łódzka
Instytut Elektroniki

Programowanie Mikrokontrolerów

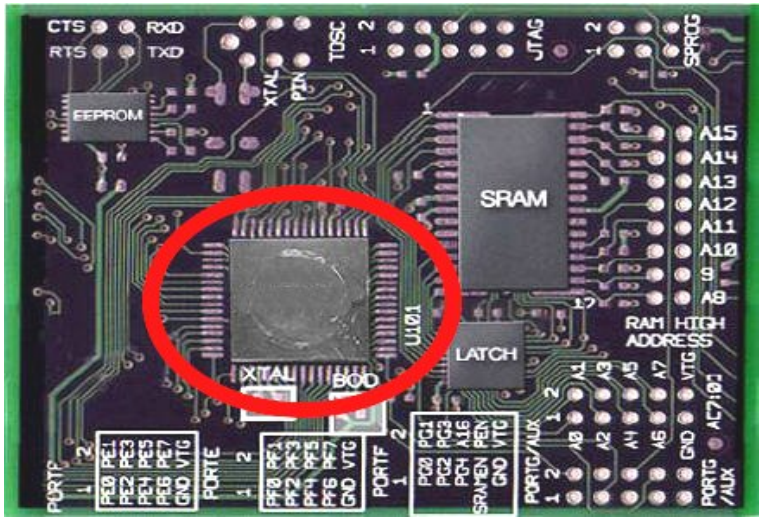
Architektura mikroprocesorów.

mgr inż. Paweł Poryzała
Zakład Elektroniki Medycznej

Zagadnienia

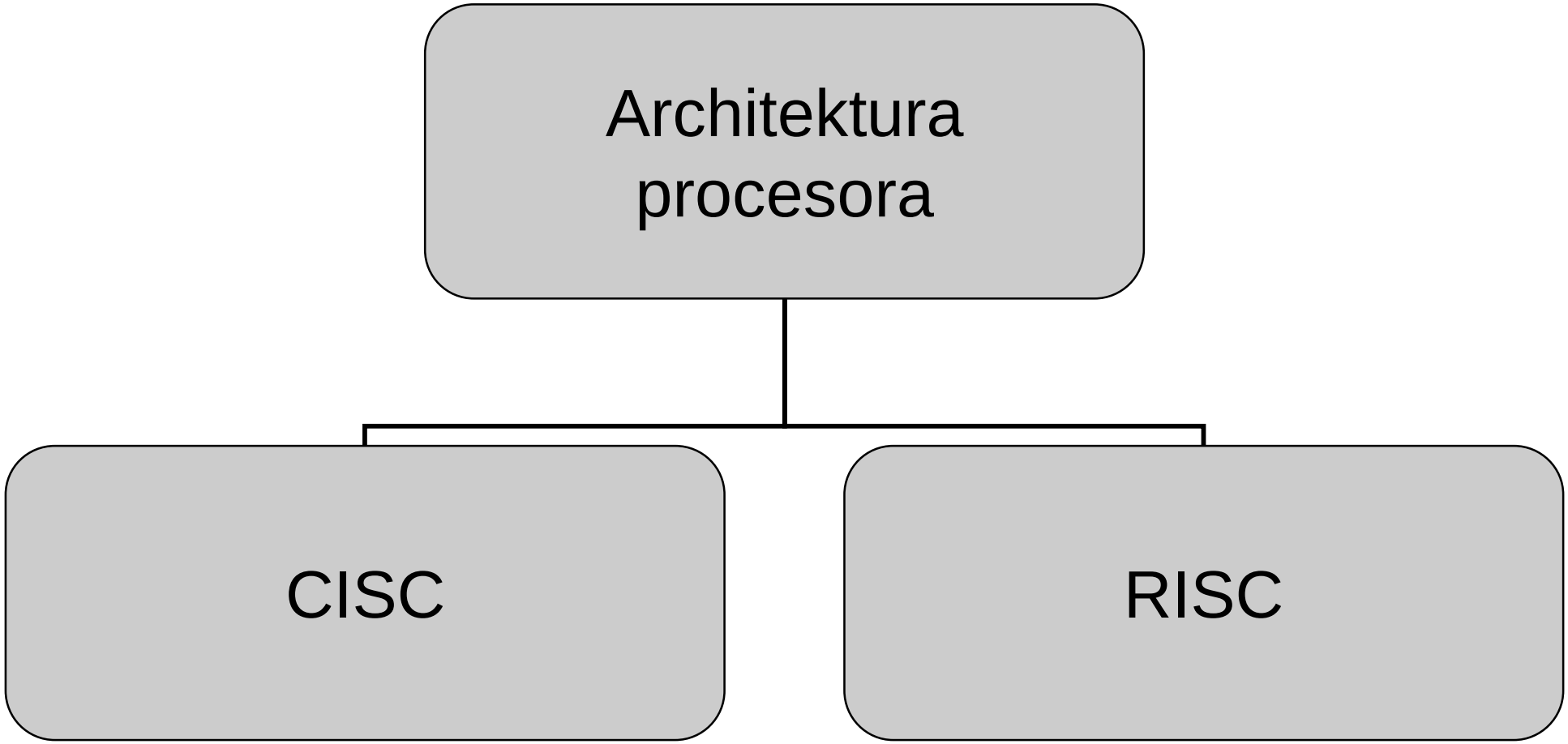
- Mikroprocesor
- Architektura mikroprocesora
  - podział: RISC / CISC
  - podział: von Neumana / harwardzka
  - podział: model programowy / mikroarchitektura
- Prosty system mikroprocesorowy
  - Układ wykonawczy
  - Układ sterujący
  - Blok rejestrów
  - Pamięci, mapy pamięci, segmentacja, jednostki MMU
- Aktualne trendy

Mikroprocesor



Cyfrowy układ scalony, wielkiej skali integracji, który pobiera, interpretuje oraz wykonuje operacje według dostarczonego ciągu rozkazów (ze zbioru instrukcji podstawowych).

Architektura mikroprocesora (1)



Complex Instruction Set Computers

Reduced Instruction Set Computers

CISC

- Główne cechy:
  - duża liczba instrukcji o dużej złożoności
  - wiele instrukcji wymaga dużej liczby cykli do wykonania,
  - występowanie złożonych, specjalistycznych rozkazów,
  - duża liczba trybów adresowania,
  - do pamięci może się odwoływać bezpośrednio duża liczba rozkazów,
  - mniejsza od RISC-ów częstotliwość taktowania procesora,
  - powolne działanie dekodera rozkazów.

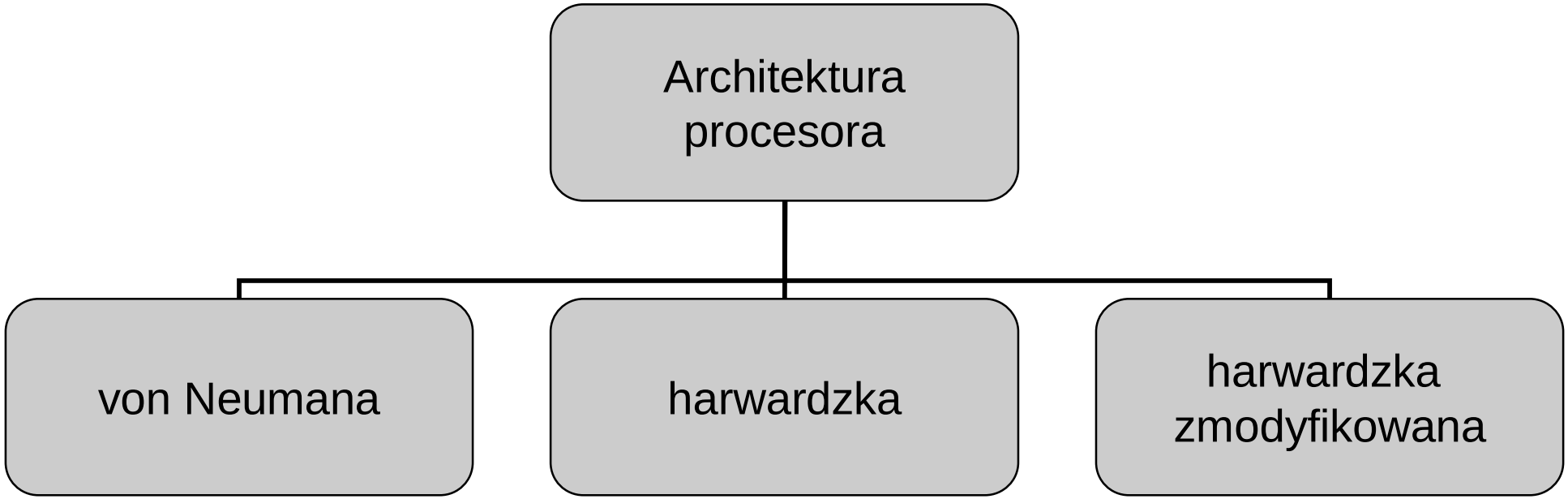
- Przykładowe układy:
  - architektura x86, 68000

RISC

- Główne cechy:
  - mniejsza liczba rozkazów, prosty dekoder rozkazów, większa częstotliwość taktowania procesora,
  - mniejsza liczba trybów adresowania, prostsze kody rozkazów,
  - ograniczenie komunikacji pomiędzy pamięcią, a procesorem,
  - do przesyłania danych pomiędzy pamięcią, a rejestrami służą dedykowane instrukcje (LOAD / STORE) .
  - większa liczba rejestrów (np. 32, 64),
  - optymalizowana na przetwarzanie potokowe, większość rozkazów wykonuje się w jednym cyklu maszynowym.

- Przykładowe układy:
  - AVR, ARM, PowerPC, MIPS.

Architektura mikroprocesora (2)



dane i instrukcje w tej samej pamięci, są one jednakowo dostępne dla procesora

pamięć danych oddzielona od pamięci rozkazów

pamięć danych oddzielona od pamięci rozkazów, wspólne magistrale dla obu pamięci

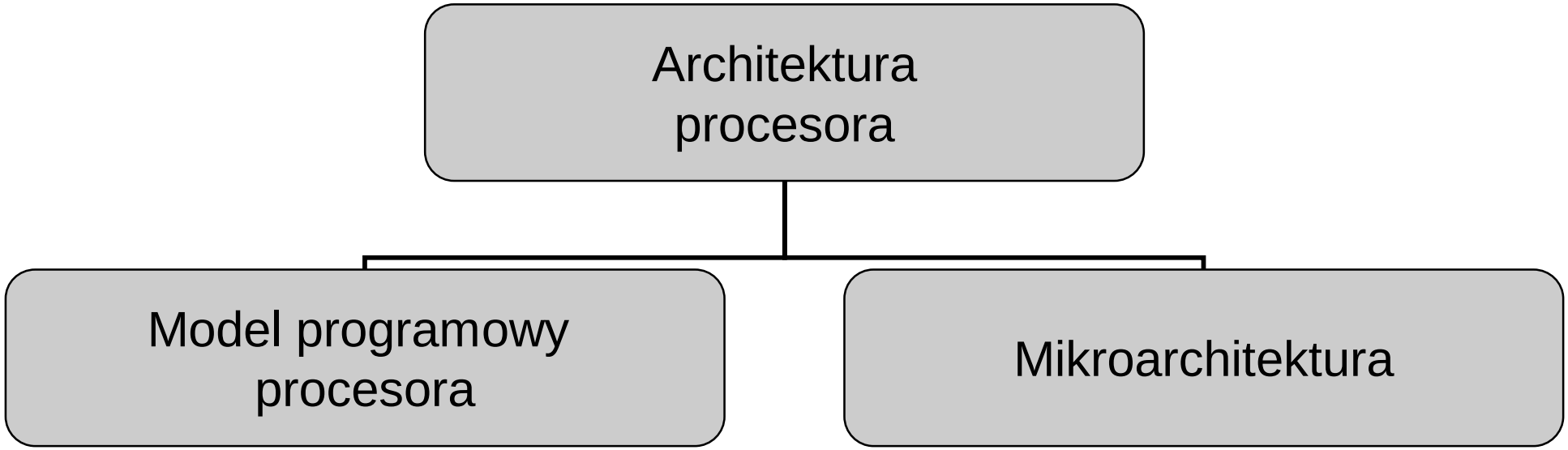
gdzie je wykorzystujemy?
słabe / mocne strony?

Architektura mikroprocesora (2)

- Architektura von Neumana:
  - rozkazy i dane przechowywane są w tej samej pamięci, są jednakowo dostępne dla procesora,
  - nie da się rozróżnić danych i rozkazów (instrukcji), są one identyfikowane są przy pomocy dostarczanego przez procesor adresu,
  - procesor ma dostęp do całej przestrzeni adresowej, dekodery adresowe zapewniają mapowanie rzeczywistych układów na komórki pamięci

- Architektura harwardzka:
  - pamięć danych oddzielona od pamięci rozkazów,
  - możliwość stosowania słowa instrukcji oraz danych o różnej długości
  - możliwość zrównoleglenia pracy (*pipeline*)
  - b. popularna w mikrokontrolerach jednoukładowych

Architektura mikroprocesora (3)



najważniejsze cechy procesora z punktu widzenia programisty

sprzętowa implementacja danego modelu programowego

jakie istnieją między nimi zależności?
co z czego wynika?

Architektura mikroprocesora (3)

- Architektura procesora określa jego najważniejsze cechy (z punktu widzenia budowy i funkcjonalności).
- 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.

- Procesory realizujące ten sam model programowy, mogą znacznie różnić się między sobą na poziomie mikro-architektury.

http://pl.wikipedia.org

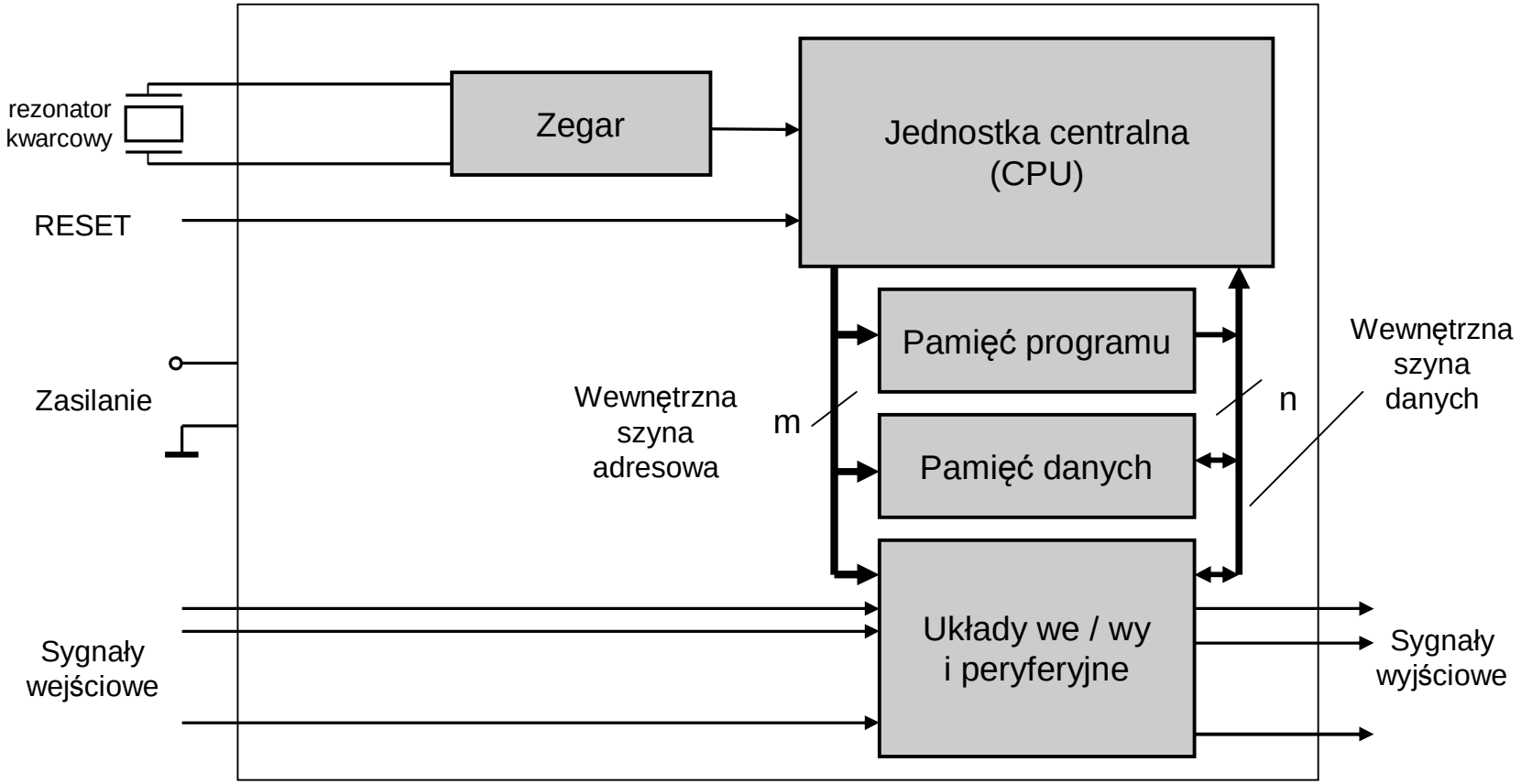
Model programowy

- cechy istotne z punktu widzenia programisty
  - lista instrukcji procesora
  - typy danych
  - dostępne tryby adresowania
  - zestaw rejestrów dostępnych dla programisty
  - zasady obsługi wyjątków i przerwań

- Dla mikrokontrolerów AVR powyższe obejmują dokumenty:
  - AVR Datasheet,
  - AVR Instruction Set:
    - http://www.atmel.com/dyn/resources/prod_documents/doc0856.pdf

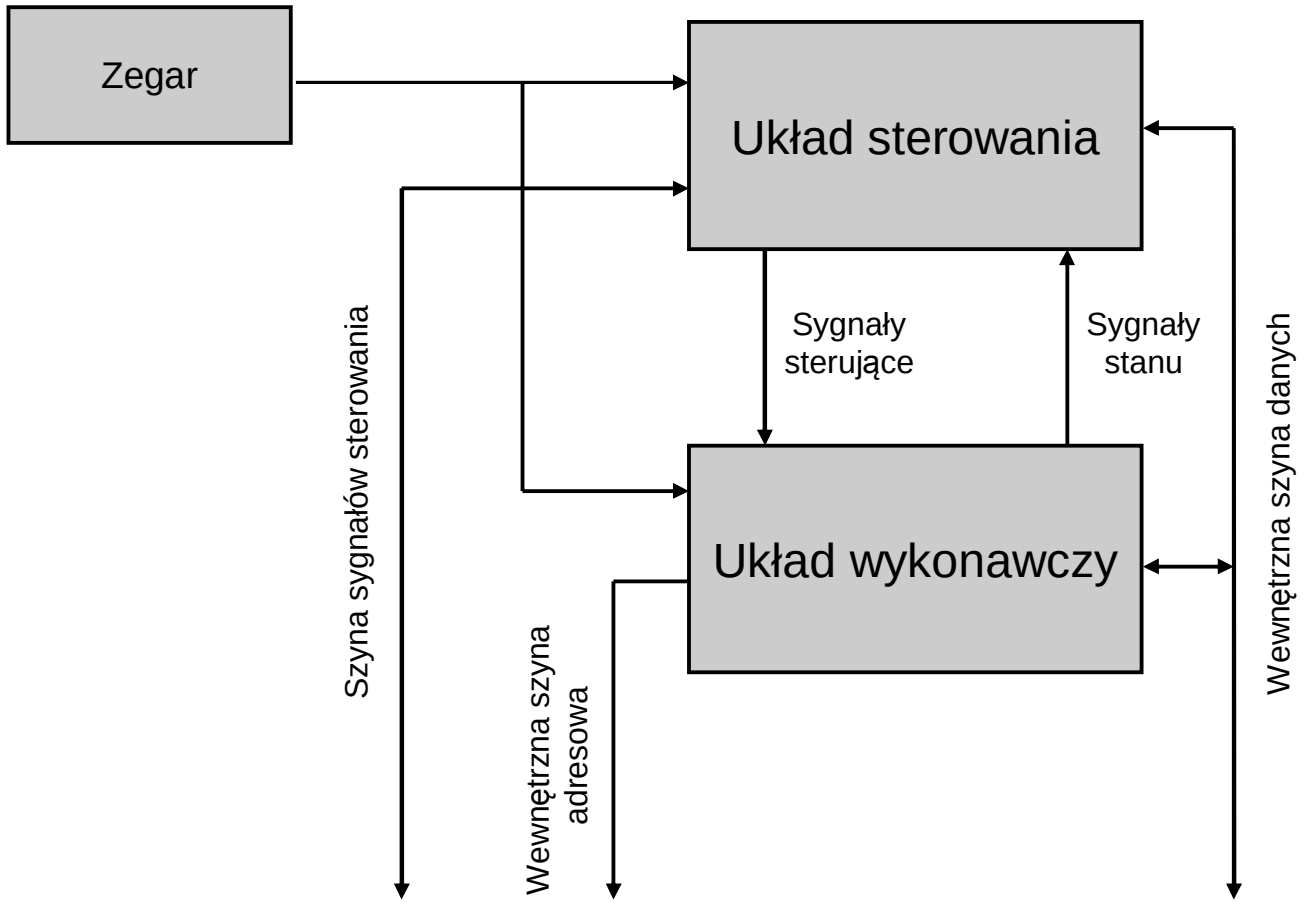
Mikroarchitektura

- Uproszczony schemat funkcjonalny:



„Mikrokontrolery: architektura, programowanie, zastosowania”, Ryszard Pełka, WKŁ 2000

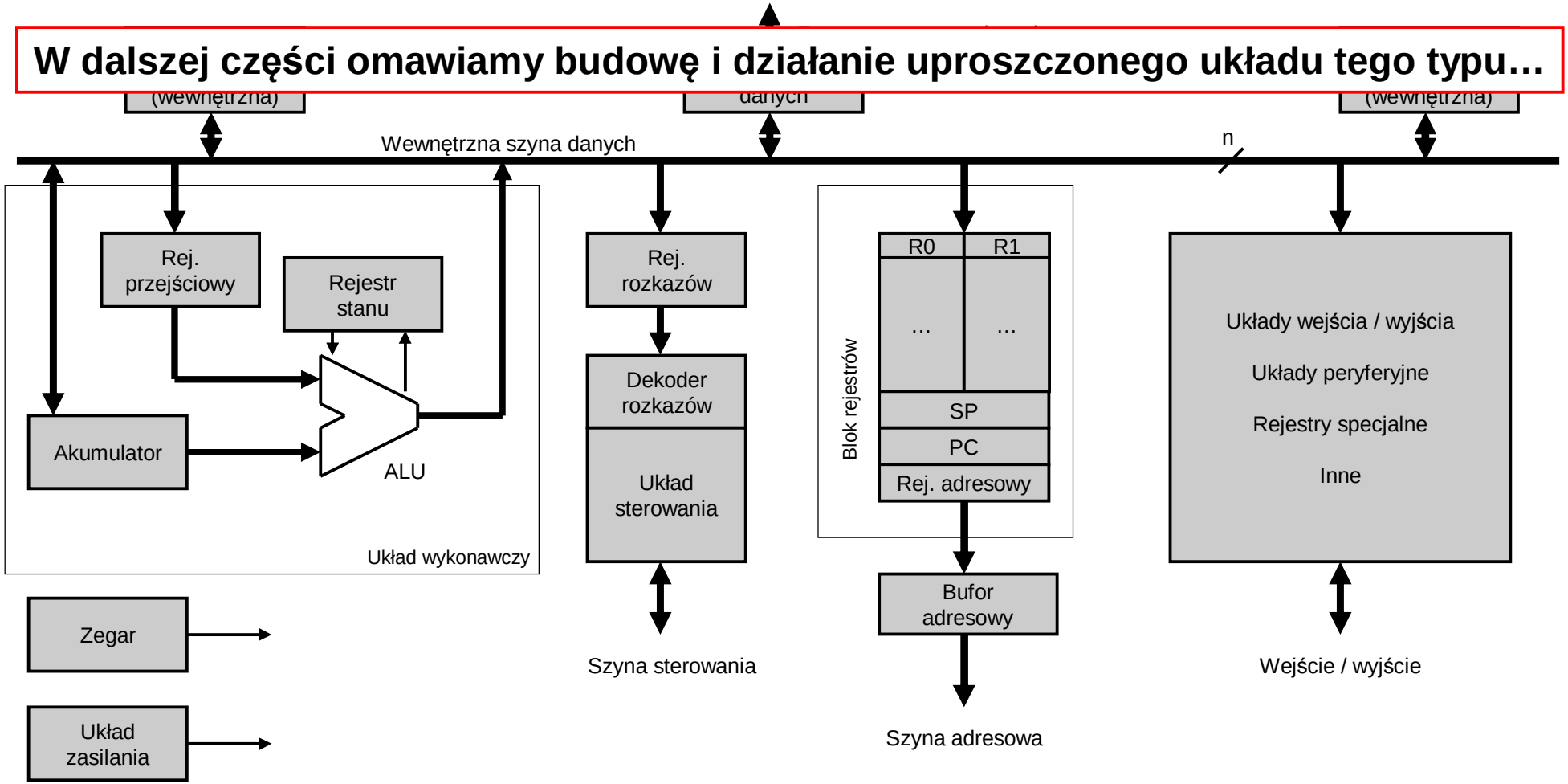
Struktura jednostki centralnej (CPU)



„Mikrokontrolery: architektura, programowanie, zastosowania”, Ryszard Pełka, WKŁ 2000

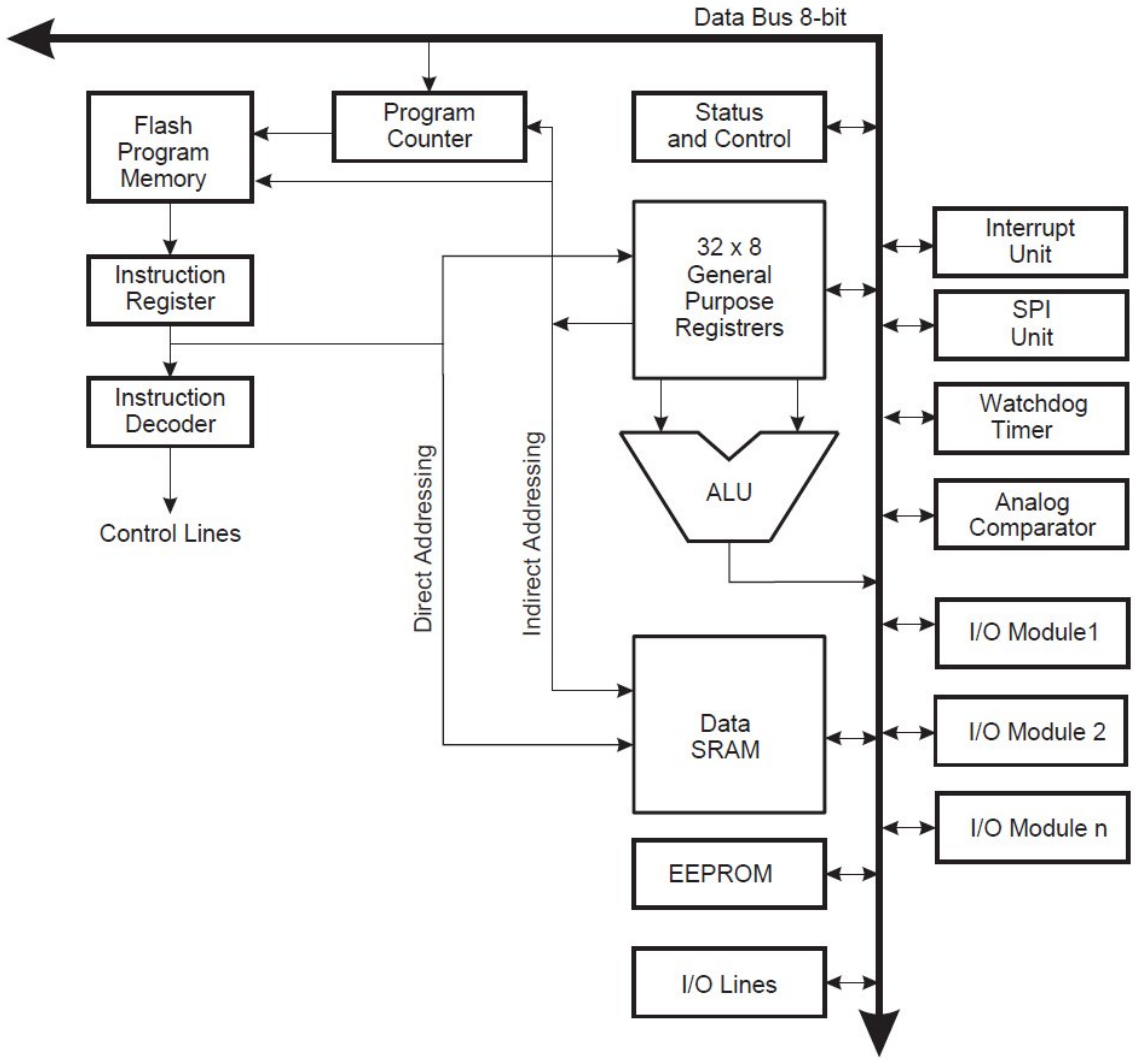
Uproszczona architektura mikrokontrolera

W dalszej części omawiamy budowę i działanie uproszczonego układu tego typu…

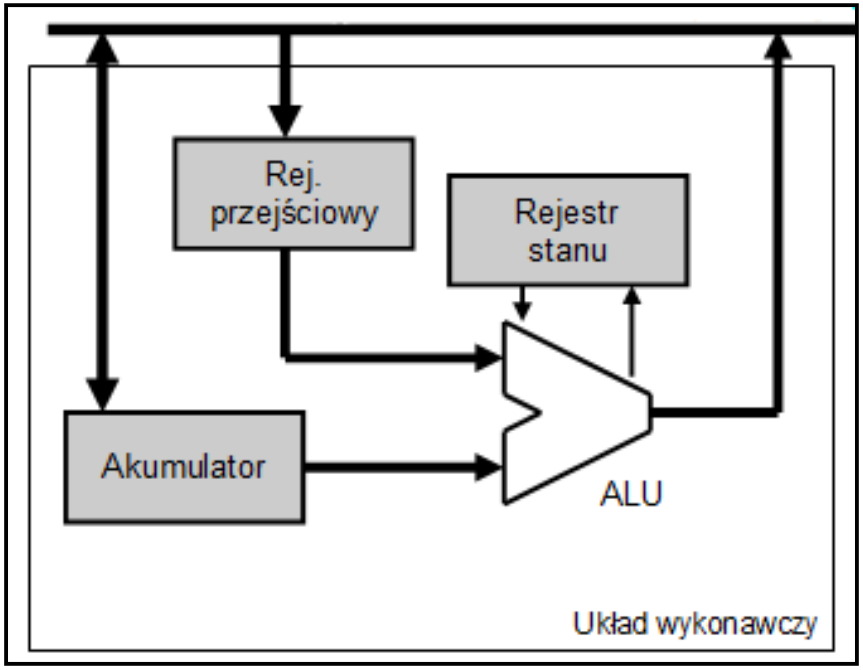


„Mikrokontrolery: architektura, programowanie, zastosowania”, Ryszard Pełka, WKŁ 2000

AVR CPU CORE


Data Bus 8-bit
Flash Program Memory
Program Counter
Status and Control
Instruction Register
Instruction Decoder
Control Lines
32 x 8 General Purpose Registrers
ALU
Direct Addressing
Indirect Addressing
Data SRAM
EEPROM
I/O Lines
Interrupt Unit
SPI Unit
Watchdog Timer
Analog Comparator
I/O Module1
I/O Module 2
I/O Module n

Układ wykonawczy



- jednostka arytmetyczno-logiczna (ALU)
- akumulator A
- rejestr przejściowy (tymczasowy)
- rejestr stanu (rejestr flag, wskaźników)

Układ wykonawczy: jednostka ALU

- Układ ALU typowo zapewnia:
  - operacje arytmetyczne (np. dodawanie),
  - operacje logiczne (np. AND, OR, XOR),
  - przesunięcia bitowe,
  - inkrementacja / dekrementacja o 1,
  - niekiedy: mnożenie / dzielenie.

- Zakres operacji ALU określony jest przez listę instrukcji konkretnego mikrokontrolera (architekturę programową),
- Bardziej złożone zadania wykonywane w sposób programowy (np. mnożenie realizowane przez sekwencję operacji dodawania).

Układ wykonawczy: rejestr wskaźników (stanu)

- Oprócz wyników operacji ALU istotne są często pewne szczególne konsekwencje:
  - wynik równy zeru,
  - przeniesienie z najbardziej znaczącego bitu ALU,
  - znak wyniku,
  - itp.

	Global Interrupt Enable	Bit Copy Storage	Half Carry Flag	Sign Bit	Two's Complement Overflow Flag	Negative Flag	Zero Flag	Carry Flag	
Bit	7	6	5	4	3	2	1	0	
	I	T	H	S	V	N	Z	C	SREG
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

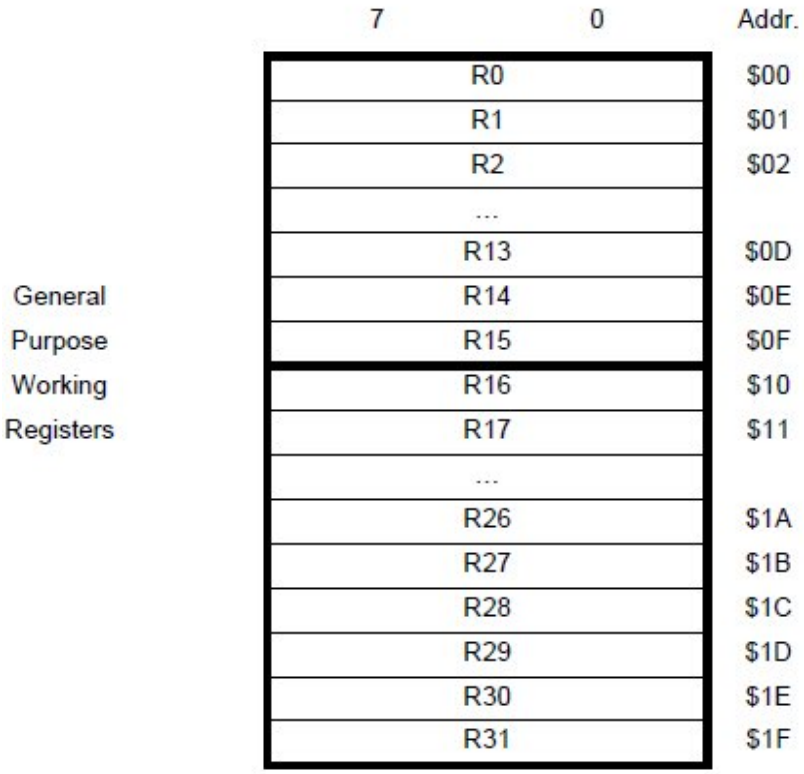
Inne układy wykonawcze

- Przykłady bardziej rozbudowanych bloków wykonawczych:
  - Kilka, równolegle pracujące ALU,
  - Dodatkowe jednostki FPU (ang. *Floating Point Units*)
  - Układy Load / Store (pobranie / zapis)
  - Układy BTB (przewidywania rozgałęzień – ang. *Branch Prediction Block*)

Blok rejestrów

- Operacje wykonywane przez ALU wymagają dostarczenia jednego / dwu operandów do rejestrów akumulatora i rej. tymczasowego.
- Wymagane częste przesłania danych z udziałem akumulatora.
- Transmisje pomiędzy akumulatorem a pamięcią danych – relatywnie wolne (przygotowanie i wysłanie adresu).
- Wartości robocze przechowywane w ramach rejestrów ogólnego przeznaczenia (niewielka pamięć z dostępem za wykorzystaniem nazw symbolicznych).
- Możliwość szybkiego odwołania do konkretnego rejestru oraz krótki kod programu.

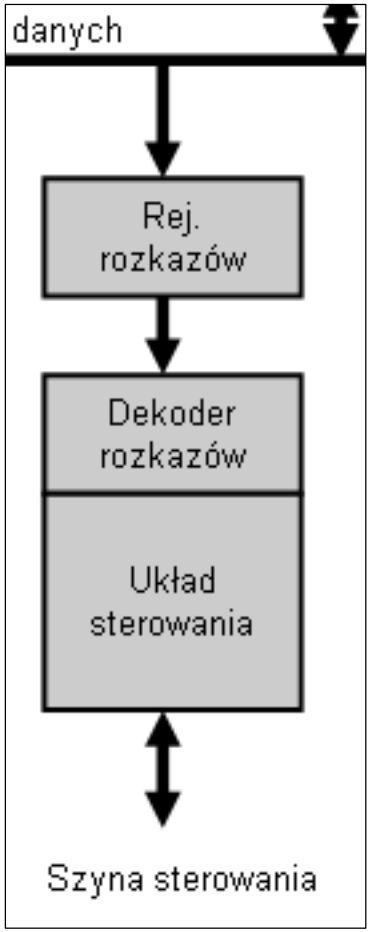
Blok rejestrów



- umieszczone wewnątrz mikroprocesora komórki pamięci o niewielkich rozmiarach (najczęściej 8/16/32 bity) służące do przechowywania tymczasowych wyników obliczeń (rejestry danych) oraz adresów lokacji w pamięci operacyjnej (rejestry adresowe)

- duża liczba rejestrów w architekturze RISC,
  - rejestry ogólnego przeznaczenia
  - rejestry specjalne

Podsystem sterowania



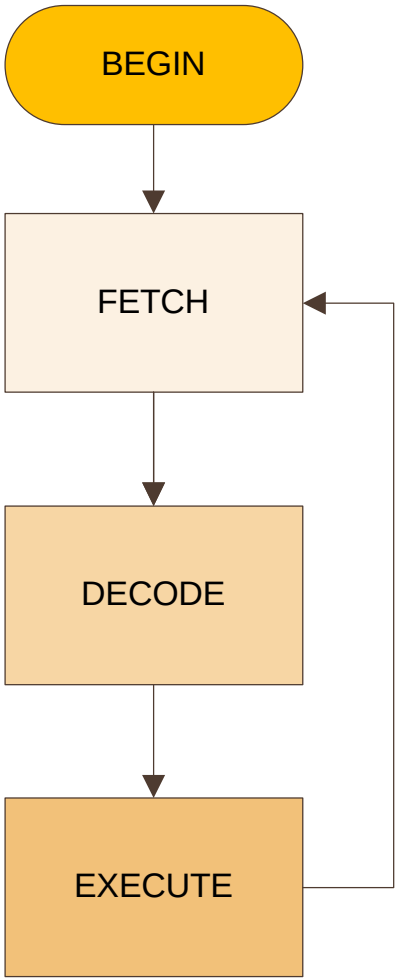
- Elementy podstawowe:
  - rejestr rozkazów (IR – ang. *Instruction Register*),
  - dekoder rozkazów,
  - licznik programu (PC – ang. *Program Counter*),
- Dodatkowo:
  - niektóre rejestry specjalne,
  - wskaźnik stosu (SP – ang. *Stack Pointer*),
  - kontroler przerwań (ang. *Interrupt Controller*).

Cykl wykonywania rozkazu

- Trój stopniowy:
  - FETCH – pobranie kodu rozkazu z pamięci programu do rejestru IR,
  - DECODE – dekodowanie kodu instrukcji,
  - EXECUTION – wykonywanie sekwencji mikrooperacji.

- Cały cykl wykonywania rozkazów nadzoruje układ sterujący.

Jak wygląda proces FETCH?



FETCH

- Podanie zawartości licznika programu (PC) przez rejestr adresowy na szynę adresową,
- Wygenerowanie przez układ sterowania sygnału odczytu z pamięci programu,
- Przesłanie kodu rozkazu z wyjścia pamięci przez szynę danych do rejestru rozkazów (IR),
- Inkrementacja licznika programu (PC).

DECODE

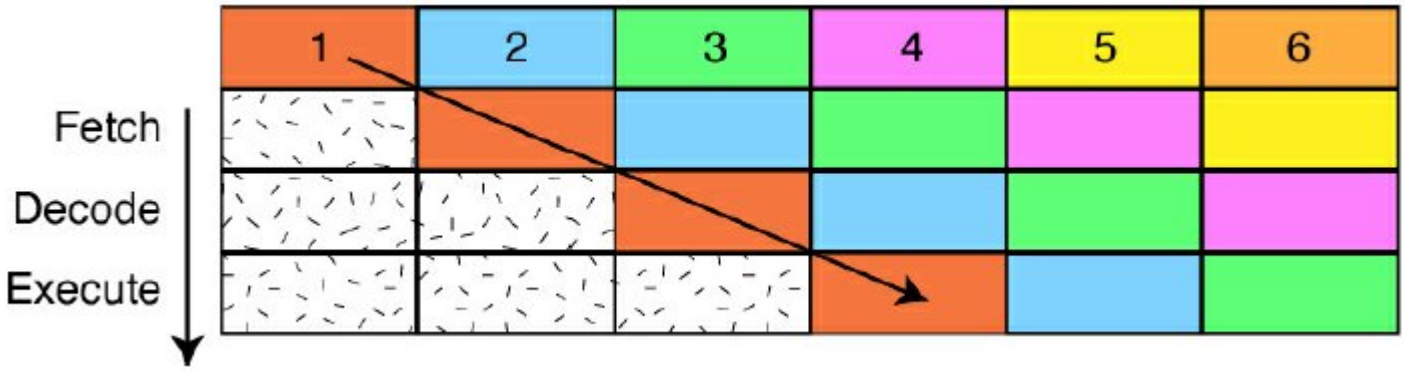
- Zawartość rejestru IR dekodowana jest przez dekoder (wykonany jako układ kombinacyjny).

EXECUTE

- Na podstawie zakodowanego stanu rejestru rozkazów układ sterowania generuje odpowiednie sekwencje – ta uzależniona jest główne liczbą odwołań do pamięci potrzebnych do wykonania instrukcji.

Rozwinięcia koncepcji cyklu wykonywania rozkazu

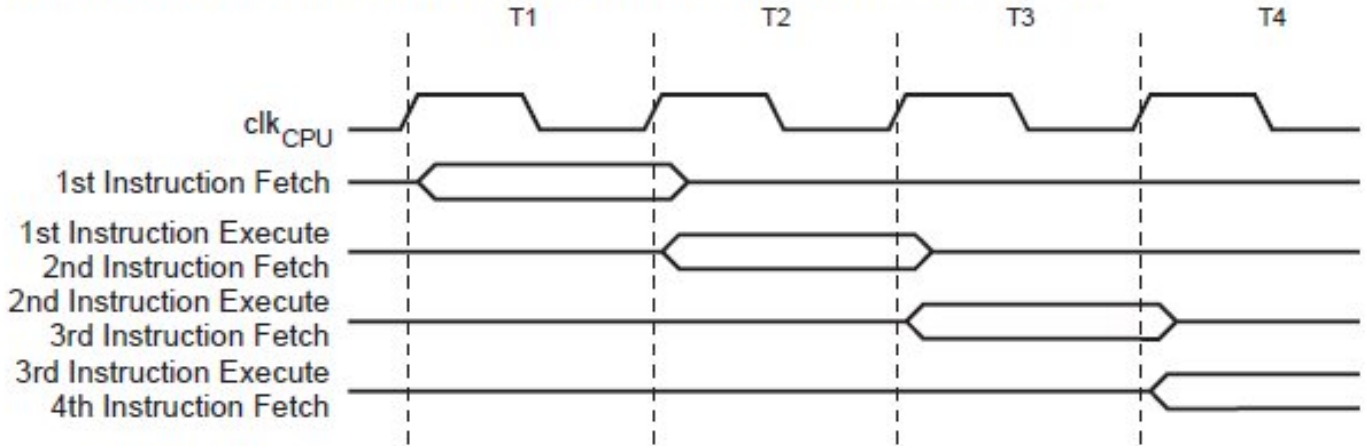
- Pipeline mechanism
  - Jednoczesne wykonywanie zdekodowanej ostatnio instrukcji oraz pobieranie kolejnej.



Przykład kompletnego cyklu - AVR

Figure 6 shows the parallel instruction fetches and instruction executions enabled by the Harvard architecture and the fast-access Register file concept. This is the basic pipelining concept to obtain up to 1 MIPS per MHz with the corresponding unique results for functions per cost, functions per clocks, and functions per power-unit.

Figure 6. The Parallel Instruction Fetches and Instruction Executions



Instrukcje programu

- Program zapisany w pamięci to ciąg zakodowanych instrukcji.
- Instrukcja programu to opis elementarnej operacji należącej do zbioru opisanego przez listę instrukcji.
- Lista instrukcji to katalog dostępnych operacji używanych przy pisaniu programu.

- Lista instrukcji wraz z opisem zachodzących między nimi współzależności oraz zbiorem reguł dotyczących składni zapisu programu tworzy opis języka asemblera.
- W asemblerze instrukcje programu odpowiadają elementarnym operacjom realizowanym przez procesor.

„Mikrokontrolery: architektura, programowanie, zastosowania”, Ryszard Pełka, WKŁ 2000

Instrukcje programu – fragment pliku *.lss

```
int main(void) {
    DDRE = 0x00;              // ustawienie całego portu E jako wejscie
  be:     12 b8               out       0x02, r1  ; 2
    PORTE = 0x00;             // wejscie trojstanowe
  c0:     13 b8               out       0x03, r1  ; 3

    DDRB = (1 << 4);          // bit 4 portu B jako wyjscie
  c2:     80 e1               ldi       r24, 0x10 ; 16
  c4:     87 bb               out       0x17, r24 ; 23

    for (;;) {
          if (SWITCH == 0) LED_OFF; else LED_ON;
  c6:     0f 99               sbic      0x01, 7   ; 1
  c8:     02 c0               rjmp      .+4       ; 0xce <main+0x10>
  ca:     c4 98               cbi       0x18, 4   ; 24
  cc:     fc cf               rjmp      .-8       ; 0xc6 <main+0x8>
  ce:     c4 9a               sbi       0x18, 4   ; 24
  d0:     fa cf               rjmp      .-12      ; 0xc6 <main+0x8>

000000d2 <_exit>:
  d2:     f8 94               cli

000000d4 <__stop_program>:
  d4:     ff cf               rjmp      .-2       ; 0xd4 <__stop_program>
```

Instrukcje programu

out 0x02, r1 opcode: 12 b8 (b8 12)

uwaga – little endian

OUT – Store Register to I/O Location

Stores data from register Rr in the Register File to I/O Space (Ports, Timers, Configuration Registers etc.).

	Syntax:	Operands:	Program Counter:
(i)	OUT A,Rr	$0 \le r \le 31, 0 \le A \le 63$	PC ← PC + 1

16-bit Opcode:

1011	1AAr	rrrr	AAAA

Words: 1 (2 bytes)
Cycles: 1

Instrukcje programu

ldi r24, 0x10 opcode: 80 e1 (e1 80)

uwaga – little endian

LDI – Load Immediate

Loads an 8 bit constant directly to register 16 to 31.

	Syntax:	Operands:	Program Counter:
(i)	LDI Rd,K	$16 \le d \le 31, 0 \le K \le 255$	$PC \leftarrow PC + 1$

16-bit Opcode:

1110	KKKK	dddd	KKKK

Words: 1 (2 bytes)
Cycles: 1

Instrukcje programu

sbic 0x01, 7 opcode: 0f 99 (99 0f)

uwaga – little endian

SBIC – Skip if Bit in I/O Register is Cleared

This instruction tests a single bit in an I/O Register and skips the next instruction if the bit is cleared. This instruction operates on the lower 32 I/O Registers – addresses 0-31.

	Syntax:	Operands:	Program Counter:
(i)	SBIC A,b	$0 \le A \le 31, 0 \le b \le 7$	PC ← PC + 1, Condition false - no skip PC ← PC + 2, Skip a one word instruction PC ← PC + 3, Skip a two word instruction

16-bit Opcode:

1001	1001	AAAA	Abbb

Words: 1 (2 bytes)

Cycles: 1 if condition is false (no skip)
2 if condition is true (skip is executed) and the instruction skipped is 1 word
3 if condition is true (skip is executed) and the instruction skipped is 2 words

Instrukcje programu

rjmp .+4 opcode: 02 c0 (c0 02)

uwaga – little endian

RJMP – Relative Jump

Relative jump to an address within PC - 2K +1 and PC + 2K (words). In the assembler, labels are used instead of relative operands. For AVR microcontrollers with Program memory not exceeding 4K words (8K bytes) this instruction can address the entire memory from every address location.

	Syntax:	Operands:	Program Counter:	Stack
(i)	RJMP k	$-2K \le k < 2K$	PC ← PC + k + 1	Unchanged

16-bit Opcode:

1100	kkkk	kkkk	kkkk

Words: 1 (2 bytes)
Cycles: 2

Instrukcje programu: podsumowanie

- 133 instrukcje
- Podział:
  - Arithmetic and Logic Instructions,
  - Branch Instructions
  - Data Transfer Instructions
  - Bit and Bit-test Instructions
  - MCU Control Instructions

Instruction Set Summary

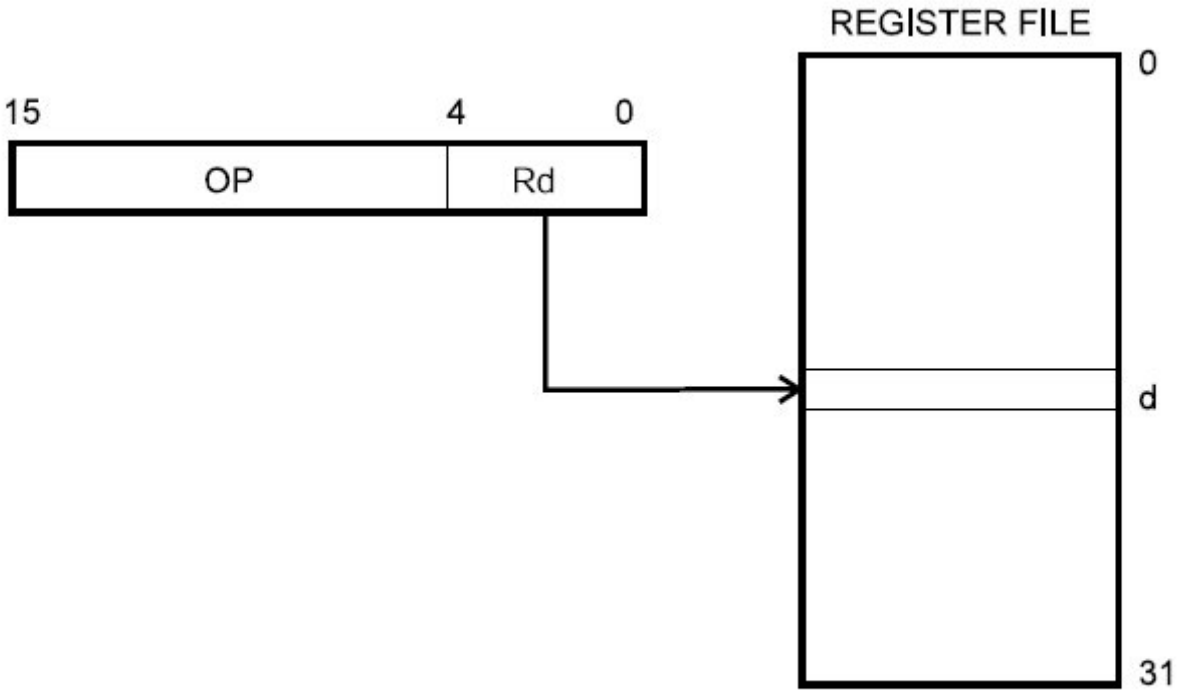
Mnemonics	Operands	Description	Operation	Flags	#Clock Note
		Arithmetic and Logic Instructions			
ADD	Rd, Rr	Add without Carry	Rd ← Rd + Rr	Z,C,N,V,S,H	1
ADC	Rd, Rr	Add with Carry	Rd ← Rd + Rr + C	Z,C,N,V,S,H	1
ADIW	Rd, K	Add Immediate to Word	Rd+1:Rd ← Rd+1:Rd + K	Z,C,N,V,S	2 (1)
SUB	Rd, Rr	Subtract without Carry	Rd ← Rd - Rr	Z,C,N,V,S,H	1
SUBI	Rd, K	Subtract Immediate	Rd ← Rd - K	Z,C,N,V,S,H	1
SBC	Rd, Rr	Subtract with Carry	Rd ← Rd - Rr - C	Z,C,N,V,S,H	1
SBCI	Rd, K	Subtract Immediate with Carry	Rd ← Rd - K - C	Z,C,N,V,S,H	1
SBIW	Rd, K	Subtract Immediate from Word	Rd+1:Rd ← Rd+1:Rd - K	Z,C,N,V,S	2 (1)
AND	Rd, Rr	Logical AND	Rd ← Rd • Rr	Z,N,V,S	1
ANDI	Rd, K	Logical AND with Immediate	Rd ← Rd • K	Z,N,V,S	1
OR	Rd, Rr	Logical OR	Rd ← Rd v Rr	Z,N,V,S	1
ORI	Rd, K	Logical OR with Immediate	Rd ← Rd v K	Z,N,V,S	1
EOR	Rd, Rr	Exclusive OR	Rd ← Rd ⊕ Rr	Z,N,V,S	1
COM	Rd	One's Complement	Rd ← $FF - Rd	Z,C,N,V,S	1
NEG	Rd	Two's Complement	Rd ← $00 - Rd	Z,C,N,V,S,H	1
SBR	Rd,K	Set Bit(s) in Register	Rd ← Rd v K	Z,N,V,S	1
CBR	Rd,K	Clear Bit(s) in Register	Rd ← Rd • ($FFh - K)	Z,N,V,S	1
INC	Rd	Increment	Rd ← Rd + 1	Z,N,V,S	1
DEC	Rd	Decrement	Rd ← Rd - 1	Z,N,V,S	1

Tryby adresowania

- Sposób na podanie pozycji operandu w pamięci (gdy dotyczy pamięci danych lub pamięci programu),
- Sposób na wywołanie funkcji (skok w ramach pamięci programu)
- Więcej trybów adresowania czyni procesor bardziej elastycznym,
- Równoważne instrukcje z różnymi trybami adresowania,
- Adresowania bezpośrednie oraz pośrednie.

Tryby adresowania

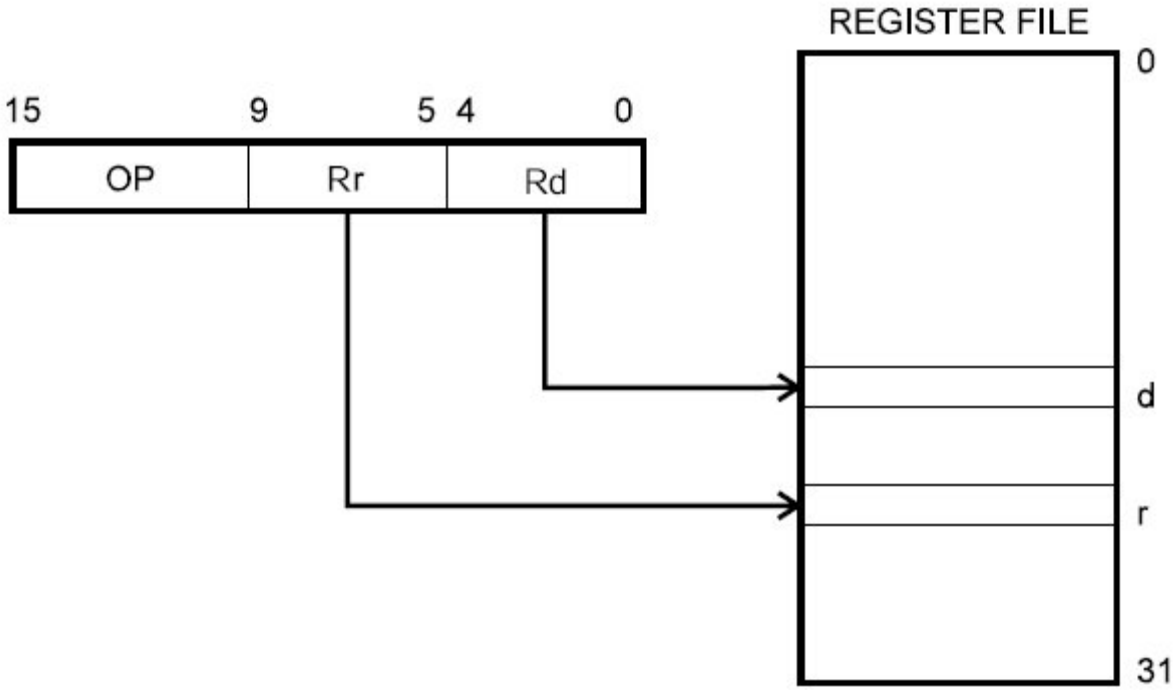
Figure 1. Direct Single Register Addressing



The operand is contained in register d (Rd).

Tryby adresowania

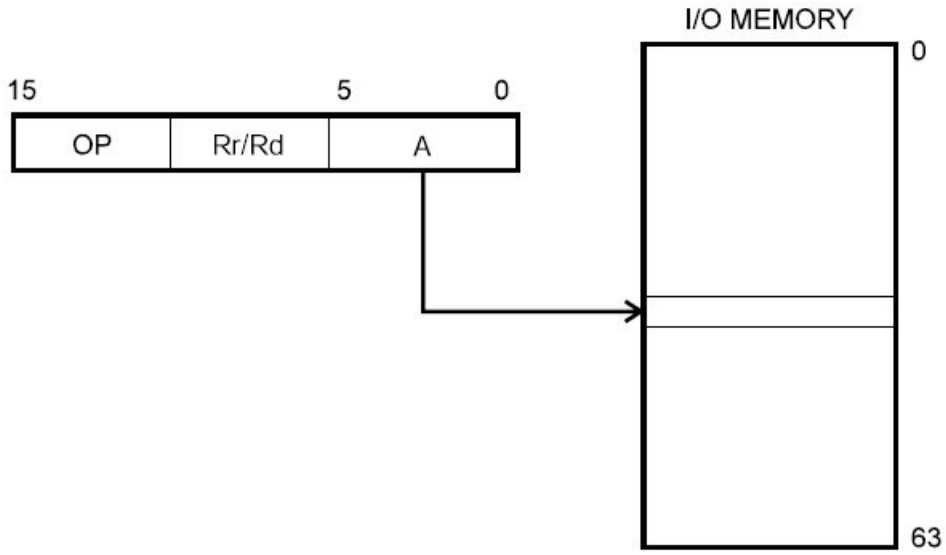
Figure 2. Direct Register Addressing, Two Registers



Operands are contained in register r (Rr) and d (Rd). The result is stored in register d (Rd).

Tryby adresowania

Figure 3. I/O Direct Addressing

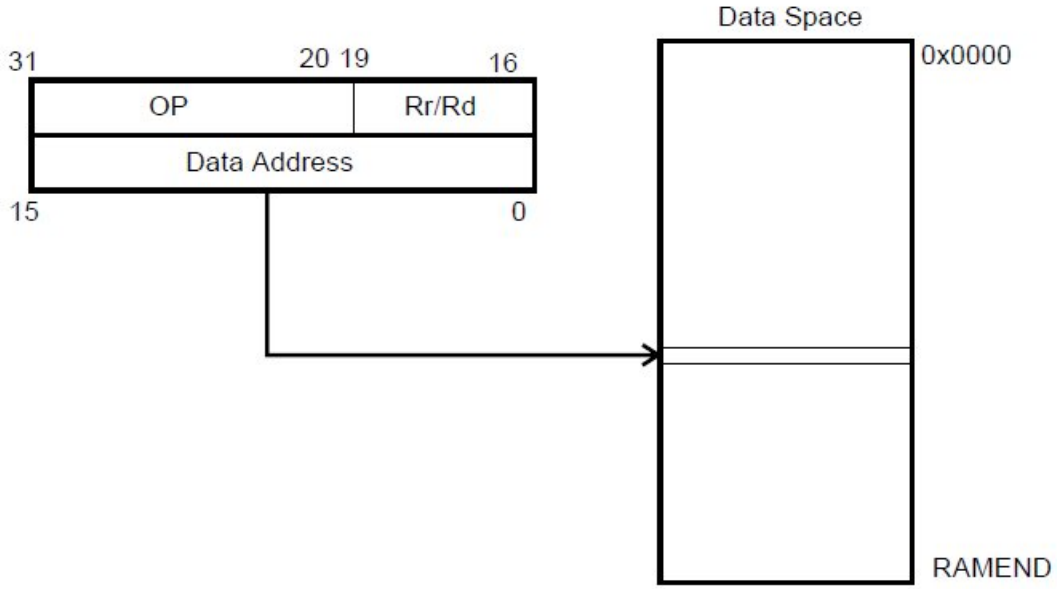


Operand address is contained in 6 bits of the instruction word. n is the destination or source register address.

Note: Some complex AVR Microcontrollers have more peripheral units than can be supported within the 64 locations reserved in the opcode for I/O direct addressing. The extended I/O memory from address 64 to 255 can only be reached by data addressing, not I/O addressing.

Tryby adresowania

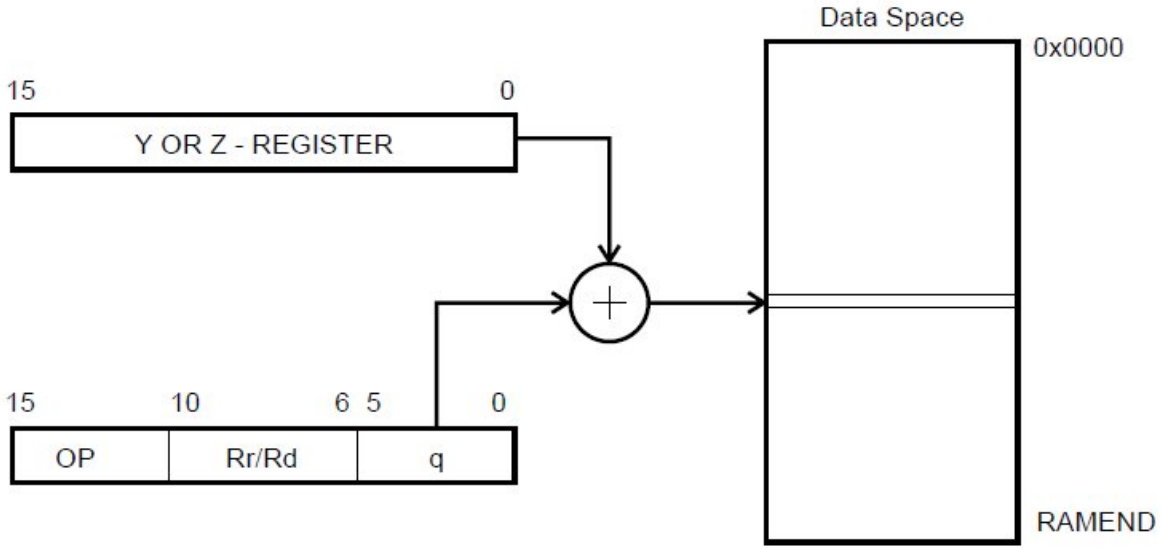
Figure 4. Direct Data Addressing



A 16-bit Data Address is contained in the 16 LSBs of a two-word instruction. Rd/Rr specify the destination or source register.

Tryby adresowania

Figure 5. Data Indirect with Displacement

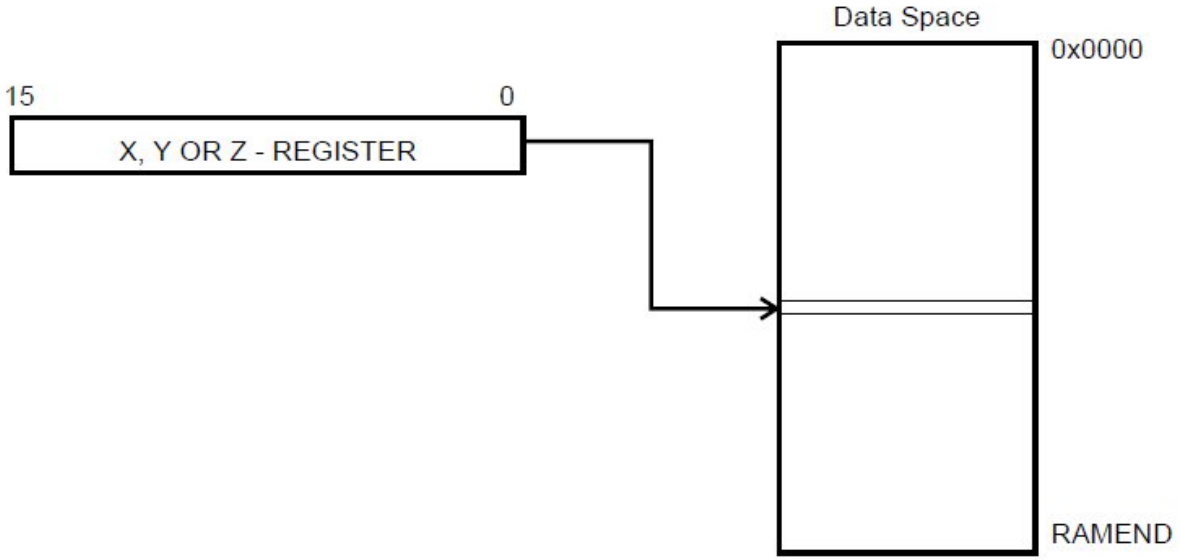


Operand address is the result of the Y- or Z-register contents added to the address contained in 6 bits of the instruction word. Rd/Rr specify the destination or source register.

Tryby adresowania

Data Indirect

Figure 6. Data Indirect Addressing

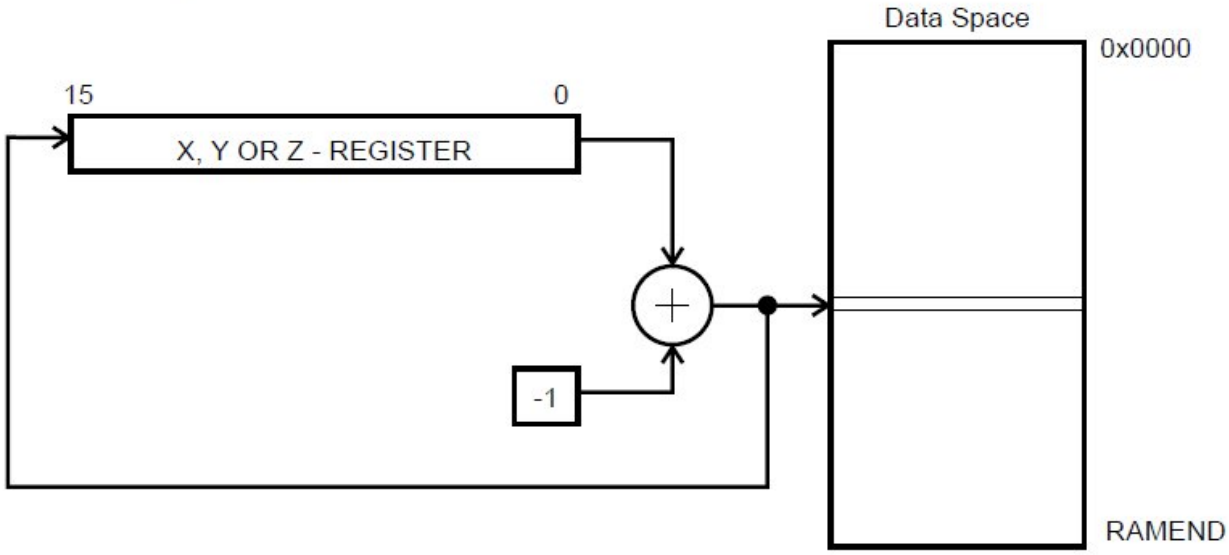


Operand address is the contents of the X-, Y-, or the Z-register. In AVR devices without SRAM, Data Indirect Addressing is called Register Indirect Addressing. Register Indirect Addressing is a subset of Data Indirect Addressing since the data space form 0 to 31 is the Register File.

Tryby adresowania

Data Indirect with Pre-decrement

Figure 7. Data Indirect Addressing with Pre-decrement

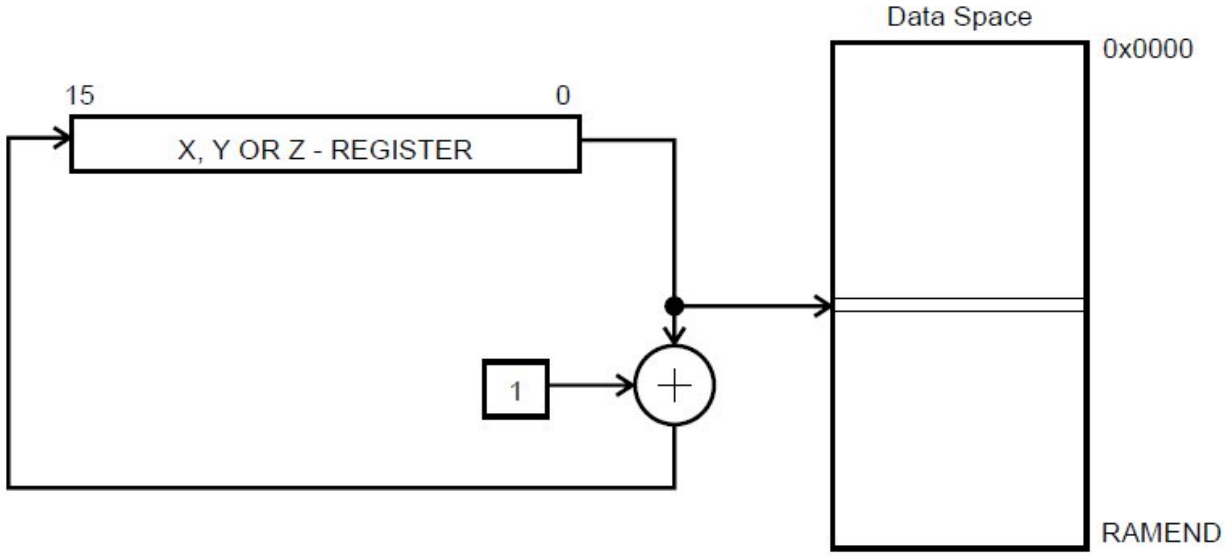


The X,- Y-, or the Z-register is decremented before the operation. Operand address is the decremented contents of the X-, Y-, or the Z-register.

Tryby adresowania

Data Indirect with Post-increment

Figure 8. Data Indirect Addressing with Post-increment

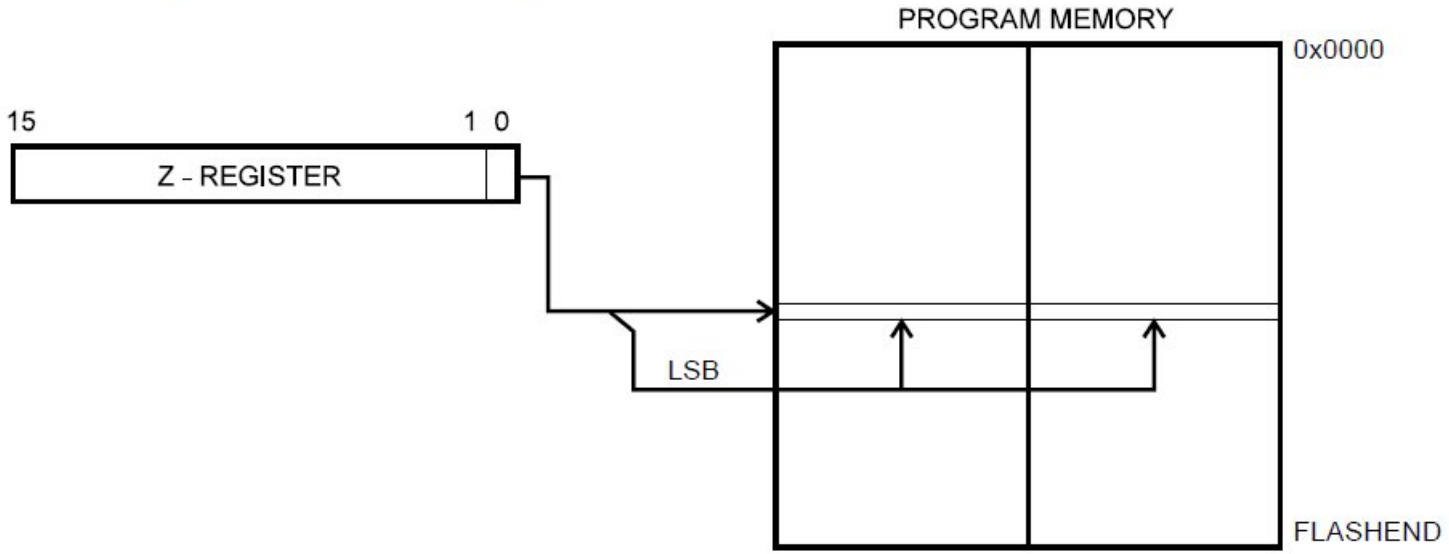


The X-, Y-, or the Z-register is incremented after the operation. Operand address is the content of the X-, Y-, or the Z-register prior to incrementing.

Tryby adresowania

Program Memory Constant Addressing using the LPM, ELPM, and SPM Instructions

Figure 9. Program Memory Constant Addressing

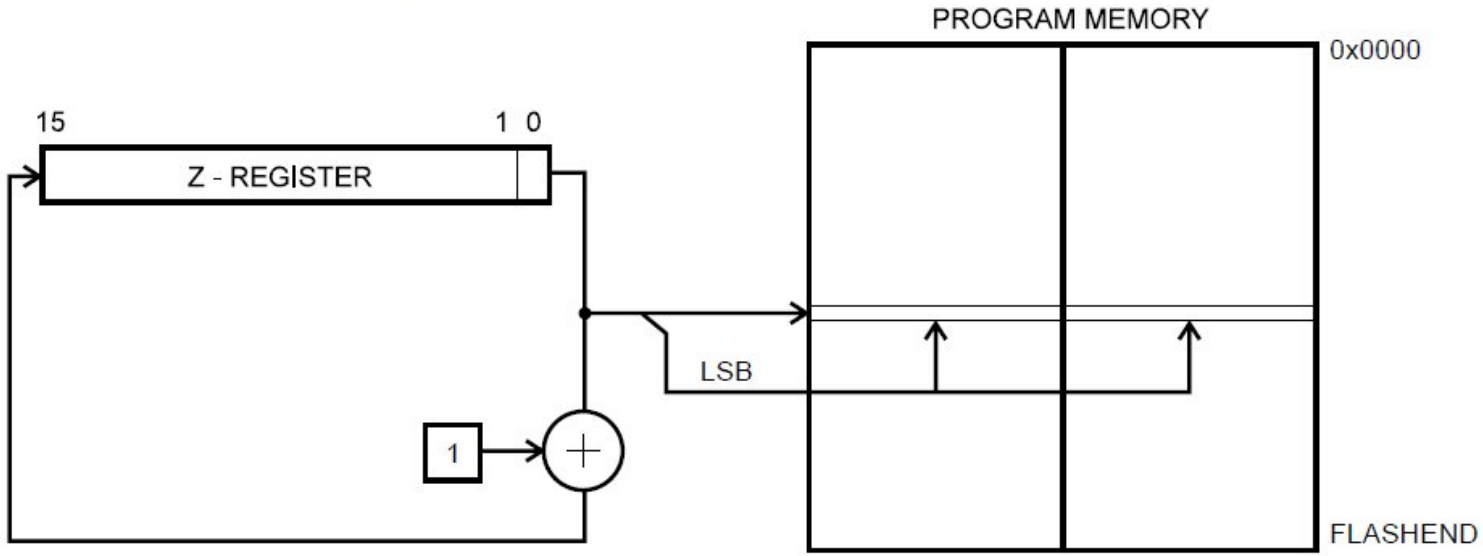


Constant byte address is specified by the Z-register contents. The 15 MSBs select word address. For LPM, the LSB selects low byte if cleared (LSB = 0) or high byte if set (LSB = 1). For SPM, the LSB should be cleared. If ELPM is used, the RAMPZ Register is used to extend the Z-register.

Tryby adresowania

Program Memory with Post-increment using the LPM Z+ and ELPM Z+ Instruction

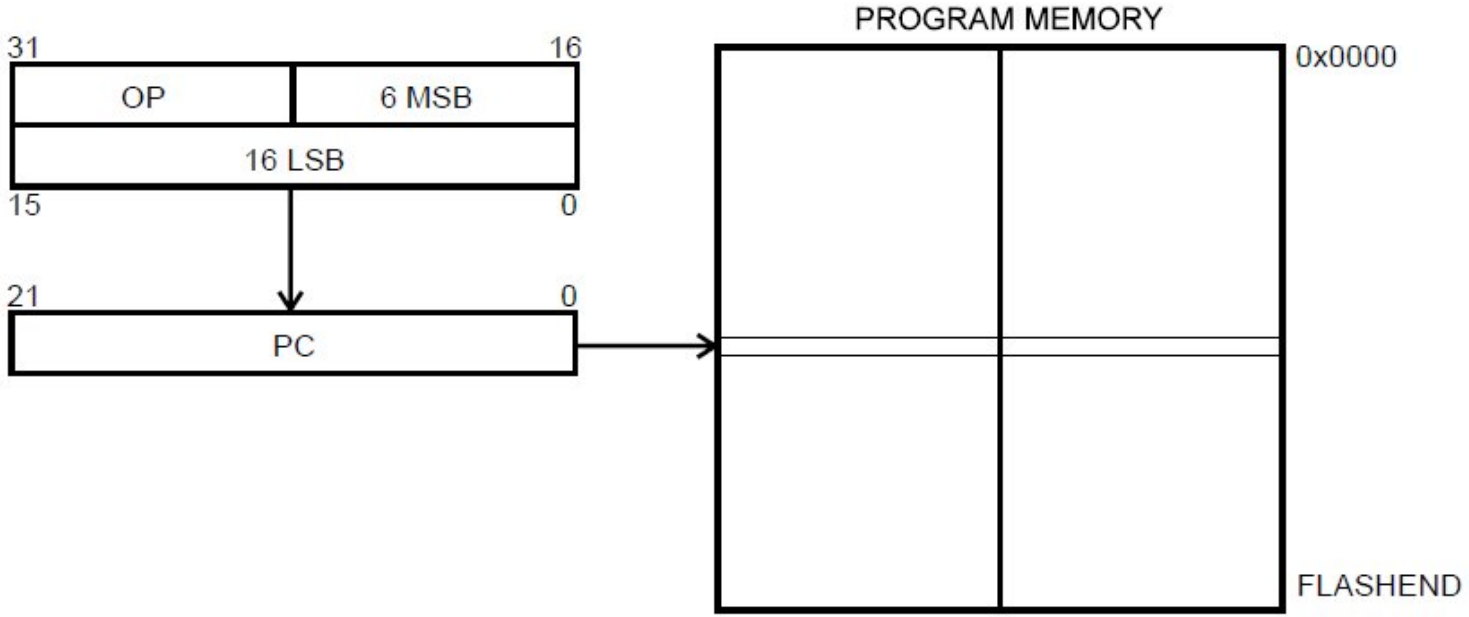
Figure 10. Program Memory Addressing with Post-increment



Constant byte address is specified by the Z-register contents. The 15 MSBs select word address. The LSB selects low byte if cleared (LSB = 0) or high byte if set (LSB = 1). If ELPM Z+ is used, the RAMPZ Register is used to extend the Z-register.

Tryby adresowania

Figure 11. Direct Program Memory Addressing

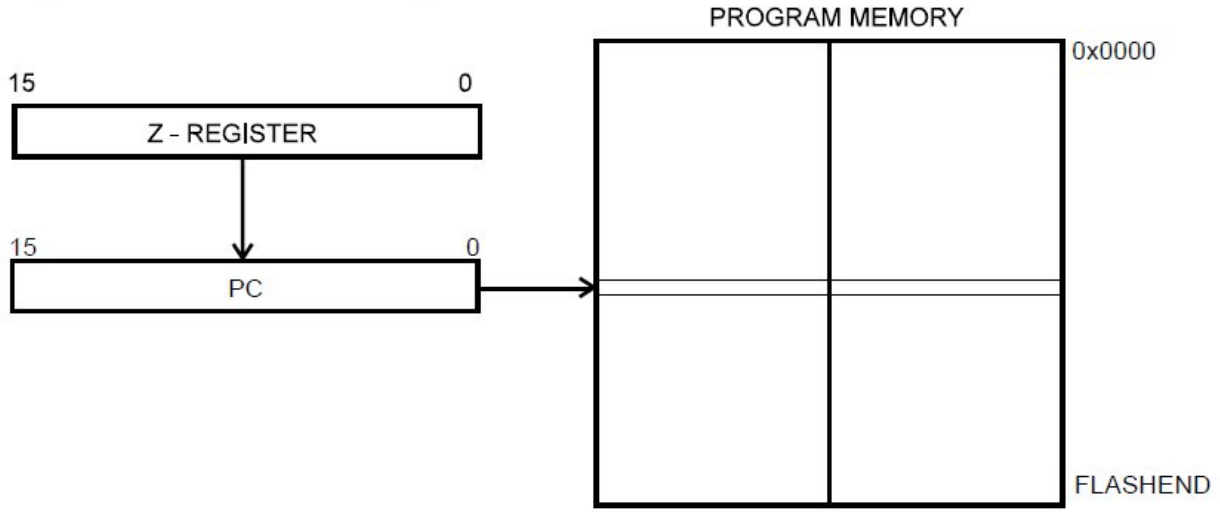


Program execution continues at the address immediate in the instruction word.

Tryby adresowania

Indirect Program Addressing, IJMP and ICALL

Figure 12. Indirect Program Memory Addressing

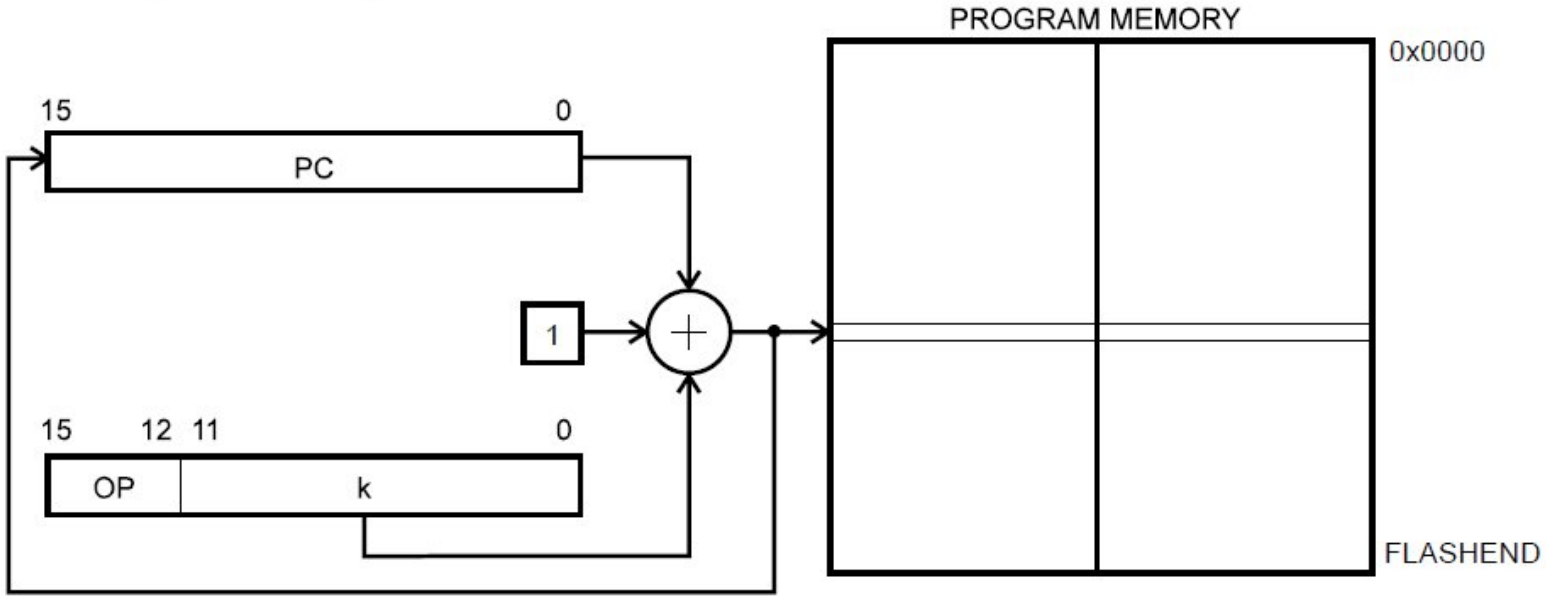


Program execution continues at address contained by the Z-register (i.e., the PC is loaded with the contents of the Z-register).

Tryby adresowania

Relative Program Addressing, RJMP and RCALL

Figure 13. Relative Program Memory Addressing

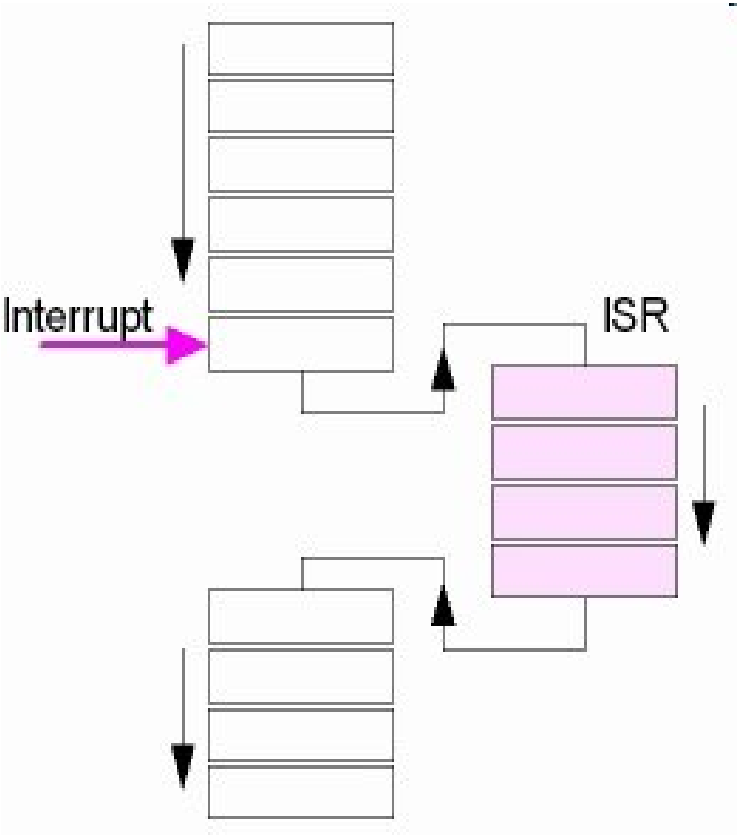


Program execution continues at address PC + k + 1. The relative address k is from -2048 to 2047.

Tryby adresowania: podsumowanie

- 13 trybów adresowania
- Najważniejsze:
  - rejestrowe (wewnętrzne bądź implikowane) – *register, inherent, implied*
  - natychmiastowe – *immediate* (tu: I/O direct addressing) – operand ukryty w opcode
  - bezpośrednie – *direct*
  - pośrednie – *indirect*
  - indeksowe – *indexed*
  - względne – *relative*

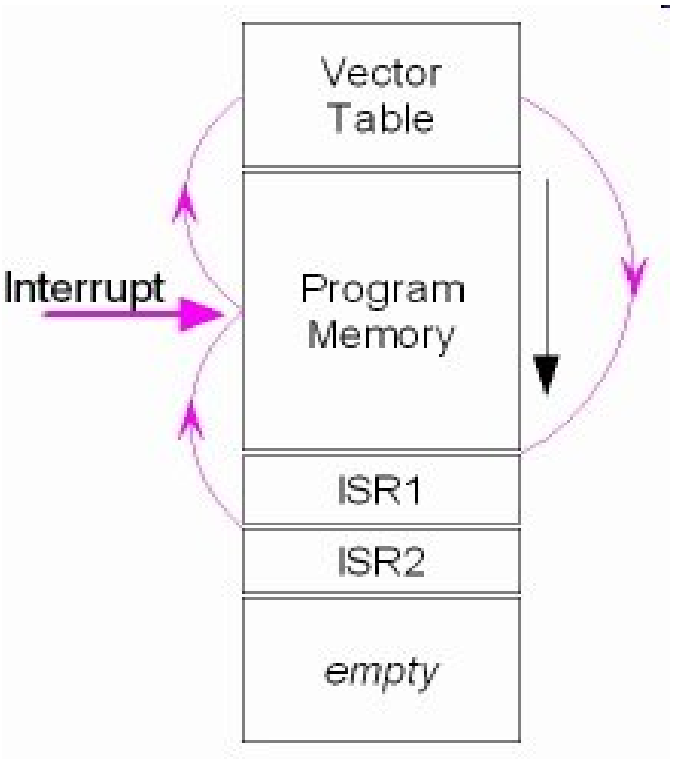
Przerwania i sytuacje wyjątkowe



- Program zapisany w pamięci mikrokontrolera nie jest jedynym czynnikiem determinującym jego pracę.
- Współpraca z różnymi urządzeniami zewnętrznymi (oraz wbudowanymi peryferiami), wymagającymi obsługi w określonych sytuacjach.
- Maksymalny dopuszczalny czas oczekiwania na obsługę – oznacza to konieczność wprowadzenia mechanizmu przerwania programu wykonywanego przez MCU.

„Mikrokontrolery: architektura, programowanie, zastosowania”, Ryszard Pełka, WKŁ 2000

Przerwania i sytuacje wyjątkowe



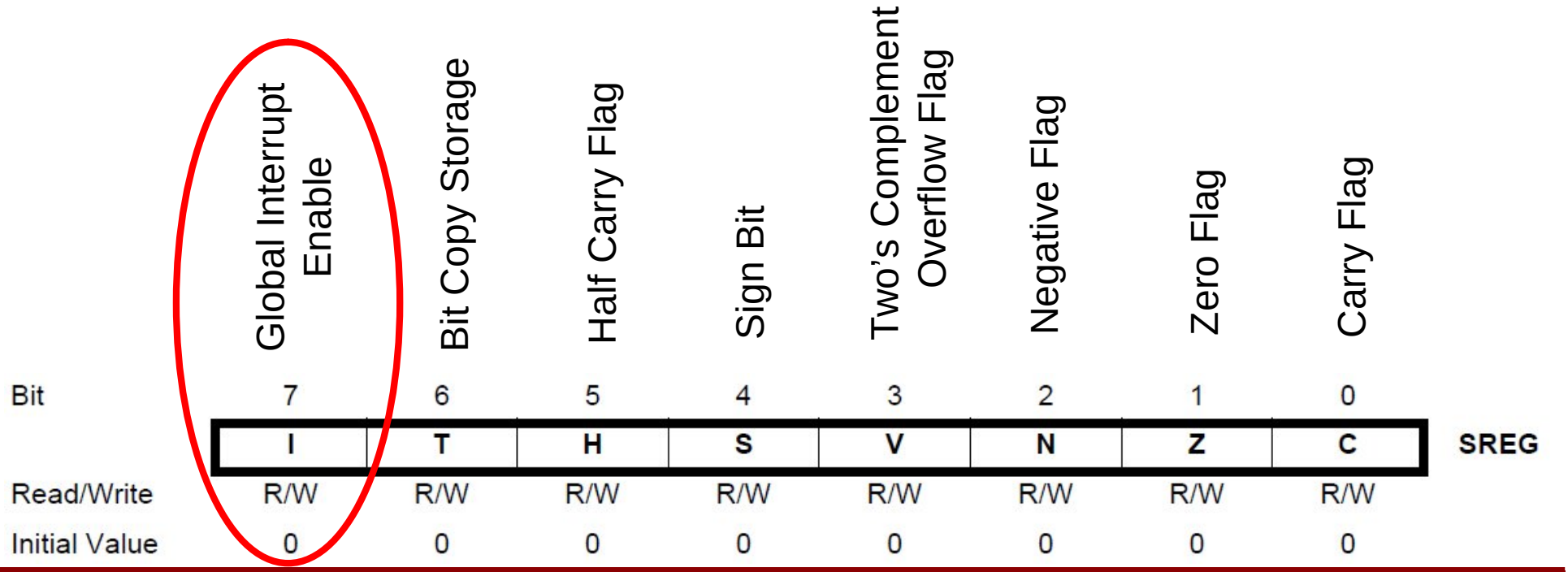
- Na sygnał przerwania z urządzenia zewnętrznego mikrokontroler przerywa aktualnie wykonywany program i przechodzi do procedury obsługi przerwania (wcześniej zapamiętuje stan mikrokontrolera oraz adres instrukcji, od której wznowić wykonywanie programu po zakończeniu obsługi przerwania),
- Po jej zakończeniu wraca do poprzednio realizowanych czynności.

Przerwania i sytuacje wyjątkowe

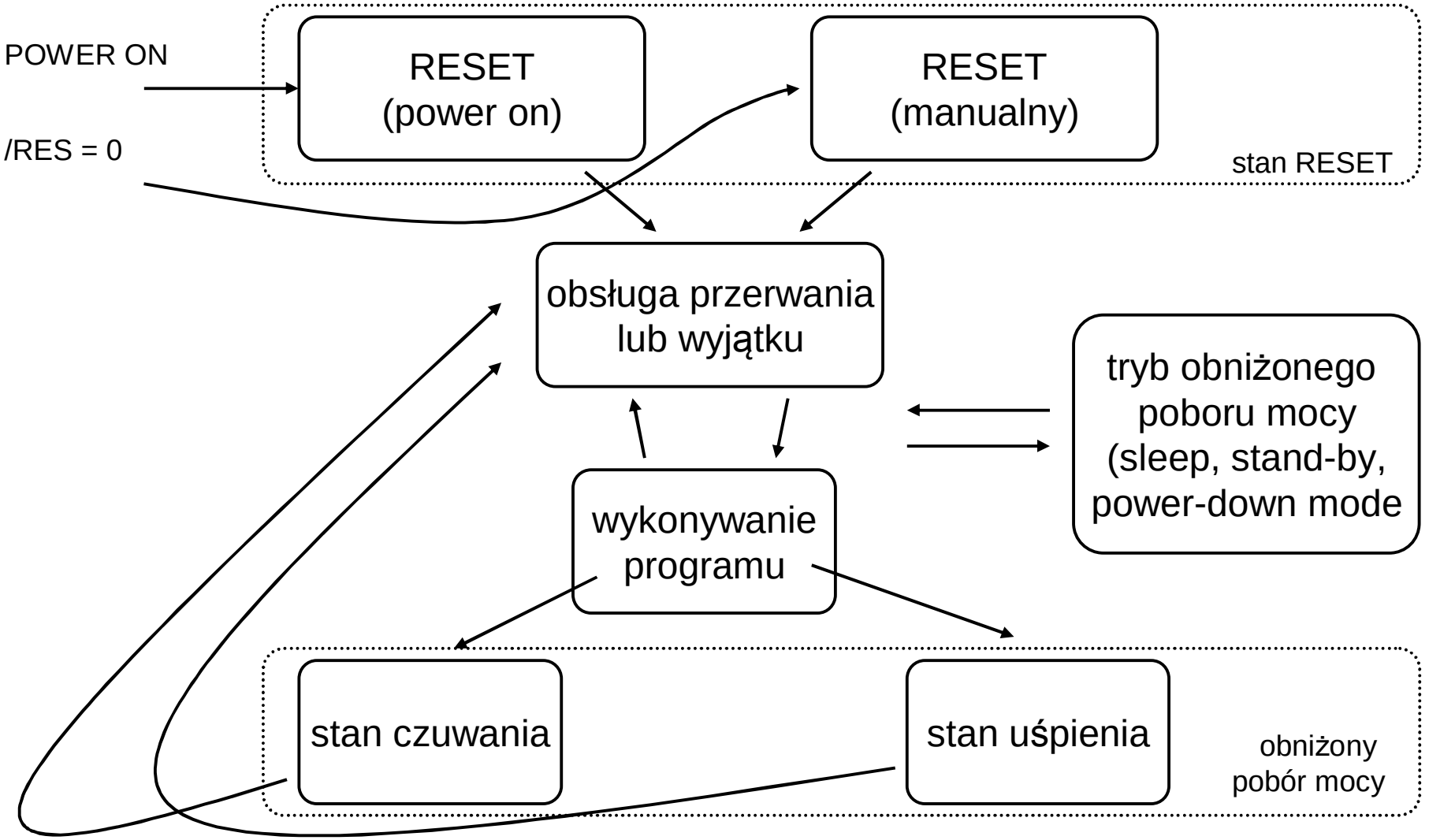
- Generalnie przerwania to cztery kategorie zdarzeń:
  - sytuacje wyjątkowe (ang. *exception interrupts*), o największym znaczeniu, niemaskowalne
  - maskowalne przerwania sprzętowe (ang. *event interrupts*) peryferiów wbudowanych oraz zewnętrznych
  - przerwania programowe (ang. *software interrupts*), wywoływane przez specjalne instrukcje w programie (np. instrukcja SWI w procesorach ARM7TDMI-S)
  - pułapki (ang. *traps*) – wykorzystywane głównie przy *debuggowaniu* aplikacji użytkownika

Przerwania i sytuacje wyjątkowe

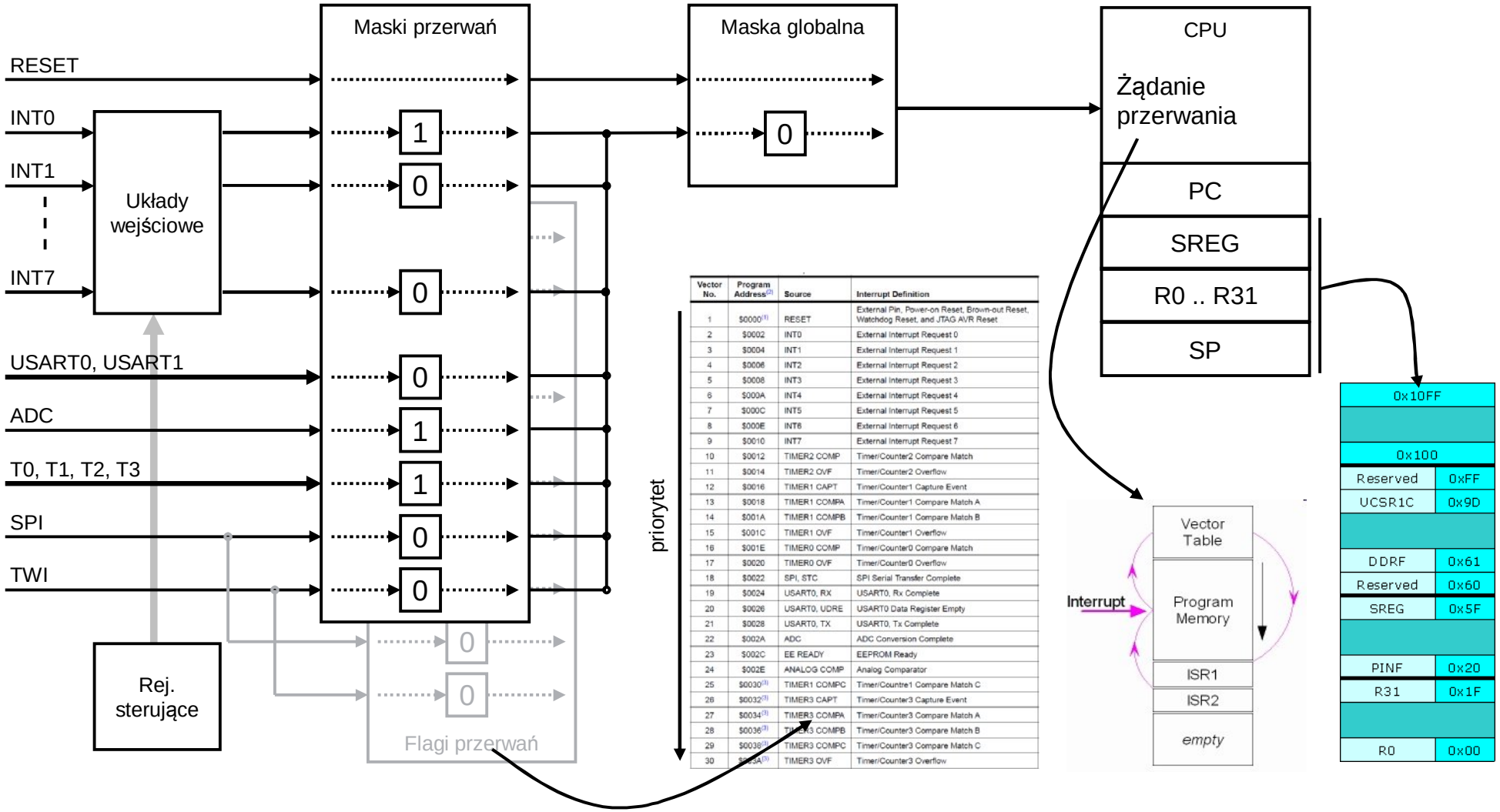
- W rejestrze stanu lub w rejestrach SFR znajdują się wydzielone bity sterujące maskowaniem oraz priorytetami przerwań.



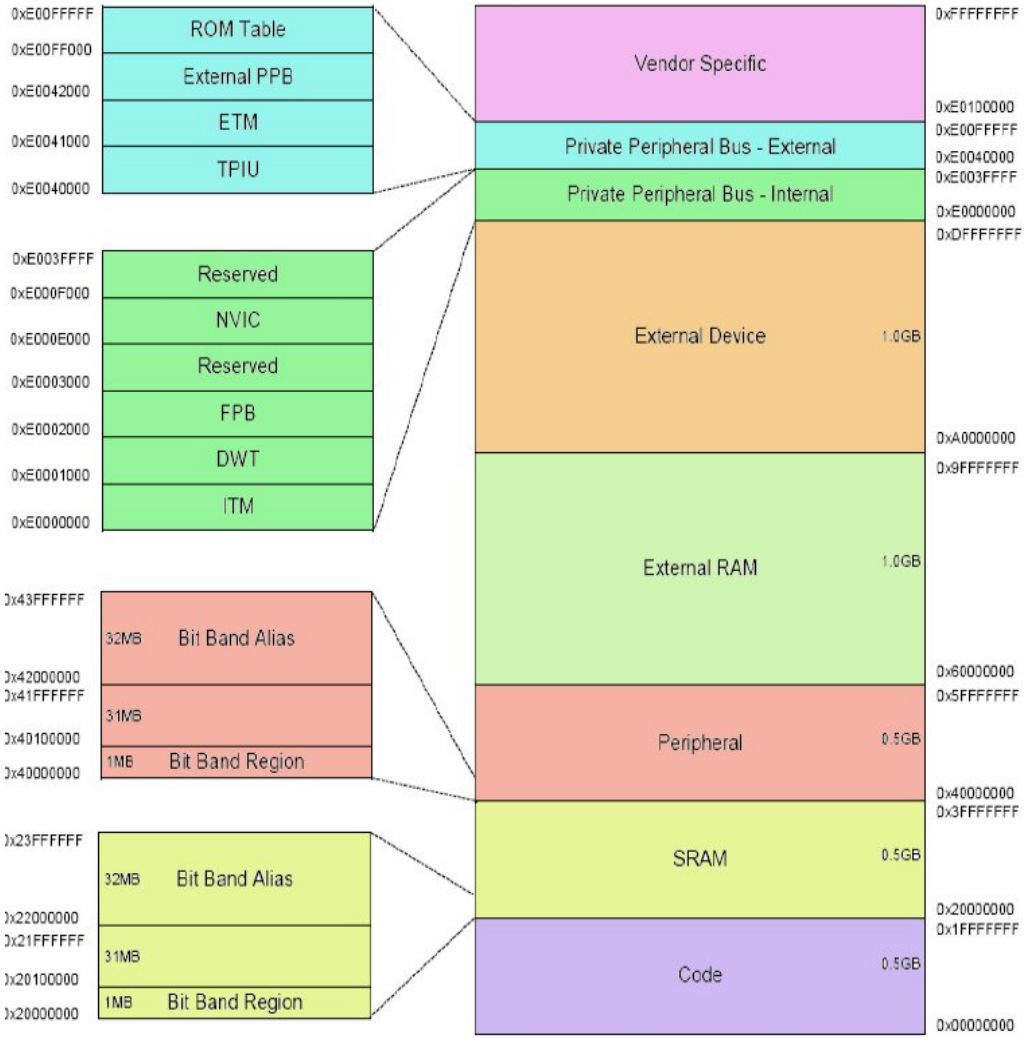
Układ sterowania: przerwania



Układ sterowania: przerwania



Mapy pamięci

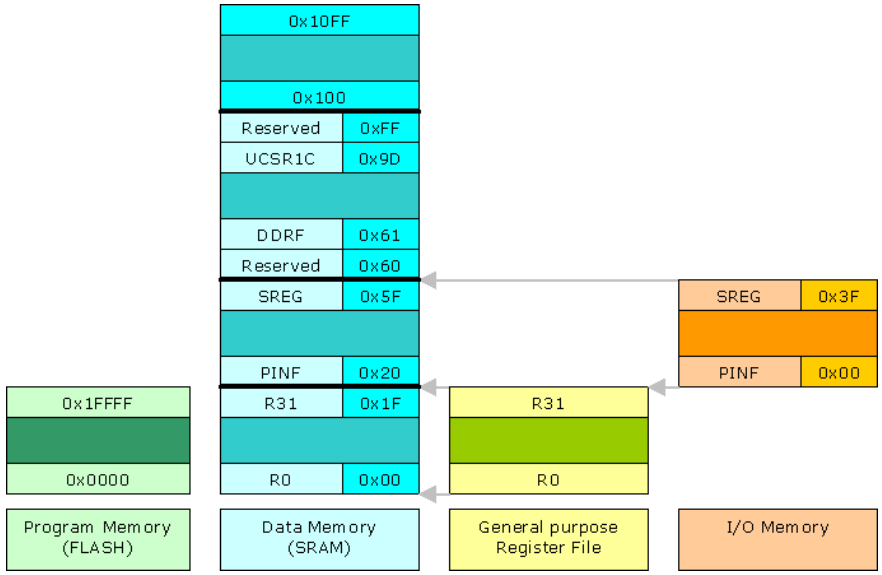


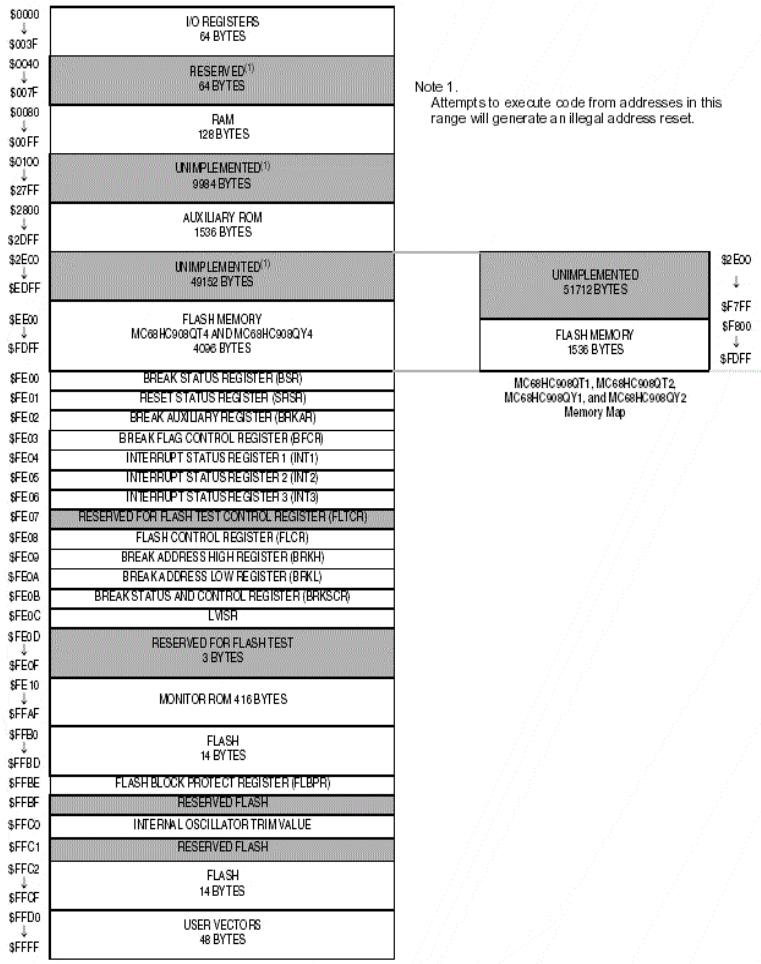
- sposób rozmieszczenia poszczególnych pamięci w przestrzeni adresowej,
- usytuowanie rejestrów uniwersalnych, adresów procedur obsługi przerwań, rejestrów specjalnych oraz rejestrów wejścia / wyjścia.

<= Cortex-M3 Memory Map

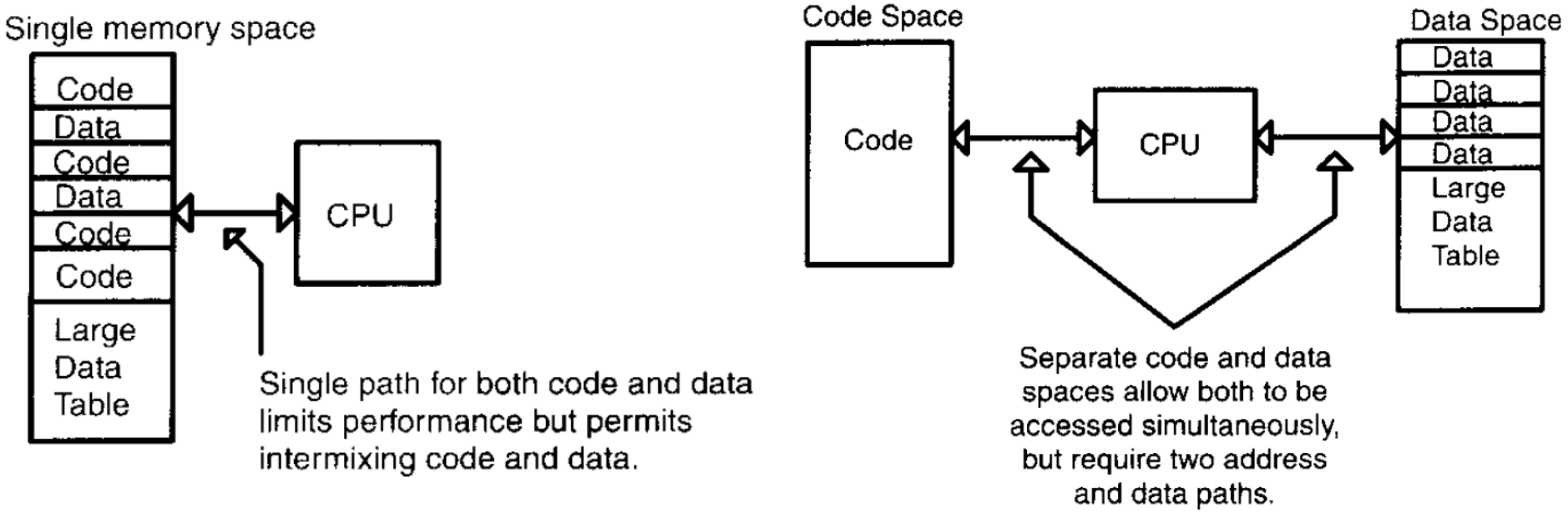
Mapy pamięci

- rozmieszczenie poszczególnych pamięci w przestrzeni adresowej jednostki centralnej (CPU)
  - jednolita przestrzeń adresowa (arch. von Neumanna)
  - niejednolita przestrzeń adresowa (arch. harwardzka)





Arch. Von Neumana oraz harwardzka



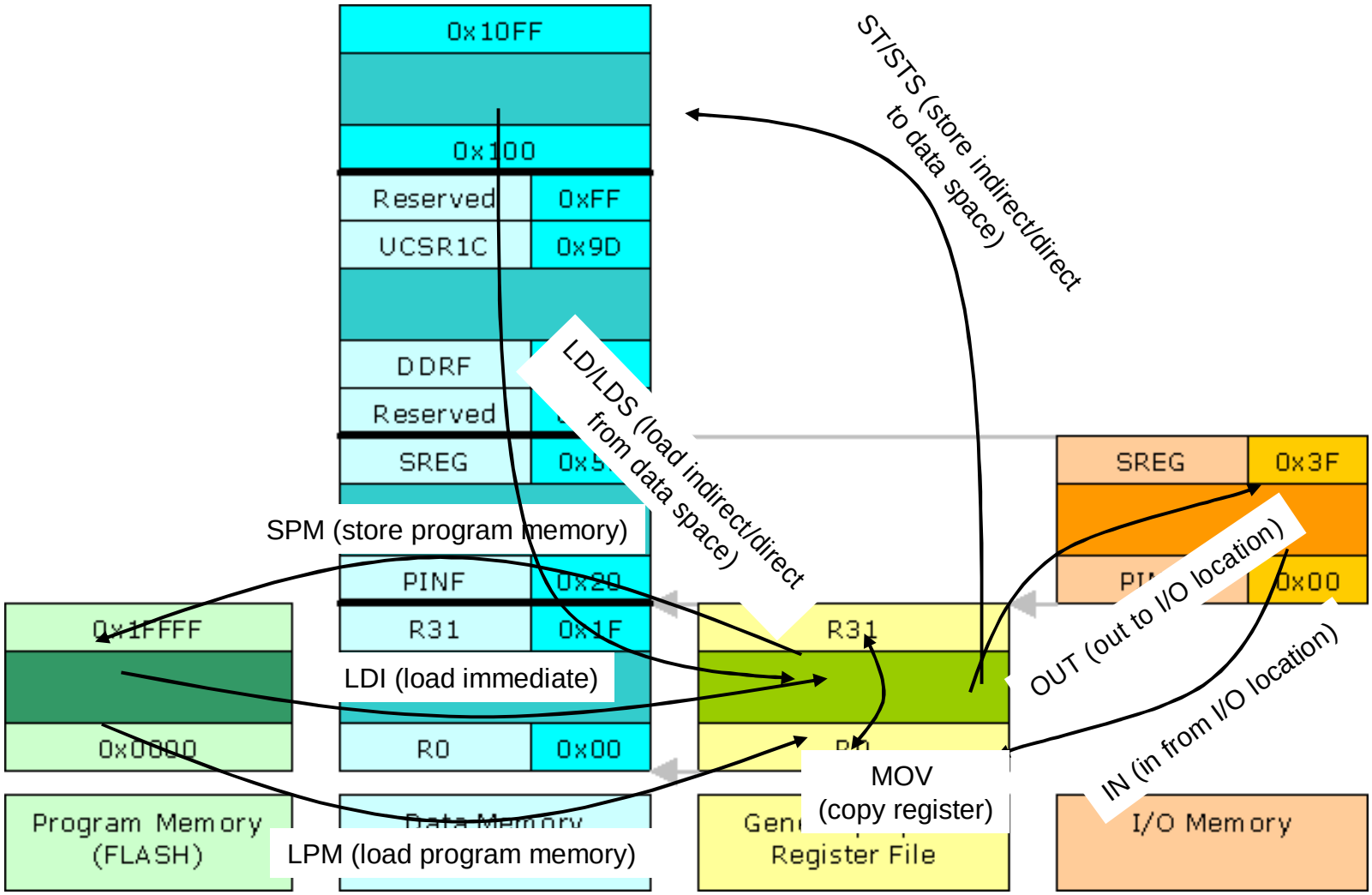
Jednolita przestrzeń adresowa (arch. Von-Neumanna)

- Czysto umowny podział na pamięć programu, danych oraz obszar I/O
- Zalety:
  - Prostota oraz przejrzystość,
  - Ułatwione programowanie (dostęp do danych, programu, urządzeń I/O odbywa się przy użyciu zunifikowanych rozkazów wykorzystujących te same tryby adresowania),
- Wady:
  - Powolna realizacja cyklu rozkazowego (wspólna szyna do pobierania danych i rozkazów).

Niejednolita przestrzeń adresowa (arch. Harwardzka)

- Dwie oddzielne szyny dla danych i rozkazów – całkowita separacja obszarów pamięci.
- Zalety:
  - Dwie oddzielne szyny dla danych i rozkazów umożliwiają tzw. *pre-fetch* (w trakcie pobierania argumentów wykonywanej właśnie instrukcji można równocześnie zacząć pobieranie następnego słowa rozkazowego),
- Wady:
  - Niejednoznaczność adresów (konieczność stosowania odmiennych rozkazów przy dostępie do poszczególnych obszarów pamięci lub zróżnicowanych trybów adresowania).

Niejednolita przestrzeń adresowa (arch. Harwardzka)



Pamięci

- pamięć RAM
  - statyczne
  - dynamiczne
- pamięć ROM
- pamięć OTP (PROM)
- pamięć EPROM
- pamięć EEPROM
- pamięć FLASH

ROM (Read Only Memory)

- programowana przez producenta w trakcie procesu produkcyjnego, tylko do odczytu,
- używana w formie pamięci programu,
- po przygotowaniu najtańsza dla większych serii.

Prędkość odczytu	Prędkość zapisu	Ilość cykli zapisu
duża	---	0

PROM (Programmable ROM)

- domyślnie pusta pamięć, jednokrotnie programowalna po zakupie z pomocą programatora,
- używana w formie pamięci programu,
- używana w mniejszych seriach, bez opłat za przygotowanie,
- droższa niż ROM.

np. seria mikrokontrolerów PIC z literą „C” jak PIC12C5xx – aktualnie stopniowo wycofywane z produkcji

Prędkość odczytu	Prędkość zapisu	Ilość cykli zapisu
duża	sekundy	1

EPROM (Erasable Programmable ROM)

- domyślnie pusta pamięć, wielokrotnie programowalna po zakupie z pomocą programatora,
- nie można „nadpisać” danych, należy je wcze
- kasowana przed każdy pomocą światła UV (ki wydanie OTP nie posia okienka,
- używana w formie pa

Intel 8048 (MCS-48) w wersji z pamięcią EPROM kasowaną światłem UV

Prędkość odczytu	Prędkość zapisu	Ilość cykli zapisu
duża	sekundy	wiele

EEPROM (Electrically Erasable Programmable ROM)

- wielokrotnie programowana, kasowana elektrycznie,
- używana w systemach mikroprocesorowych w formie pamięci nieulotnej (po wyłączeniu zasilania), zwykle o bardzo ograniczonej pojemności,
- pozwala kasować pojedyncze komórki pamięci,
- ze względu na powolne cykle zapisu i odczy na jako pamięć programu.

Prędkość odczytu	Prędkość zapisu	Ilość cykli zapisu
niewielka	niewielka	100,000

Flash EEPROM

- ograniczona liczba cykli zapisu (typowo około 10,000),
- kasowana blokami danych (od 64B do nawet 4kB), nie ma możliwości skasowania lub „nadpisania” pojedynczego bajtu,
- proces kasowania jest powolny (nawet pojedyncze milisekundy na zapis bloku danych), proces odczytu b. szybki

Prędkość odczytu	Prędkość zapisu	Ilość cykli zapisu
duża	niewielka	10,000

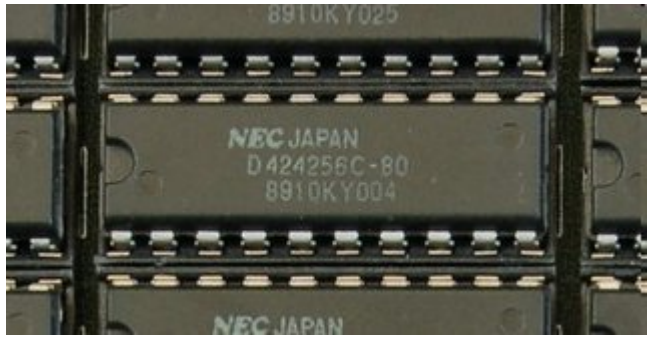
SRAM

- pamięci statyczne o swobodnym dostępie,
- przechowuje dane do czasu wyłączenia zasilania
- używana do przechowywania danych,
- niekiedy w celu zwiększenia prędkości wykonywania programu kopiowany jest do niej kod wykonywalny lub jego części (procedury) i z niej uruchamiane,

Prędkość odczytu	Prędkość zapisu	Ilość cykli zapisu
b. duża	b. duża	nieskończona

DRAM

- pamięci dynamiczne o swobodnym dostępie,
- wymagają okresowego odświeżania zawartości, za odświeżanie zwykle odpowiada specjalizowany układ kontrolera pamięci bądź sam procesor,
- zużywają mniej energii,
- niższe niż w przypadku SRAM koszty produkcji, większe pojemności,



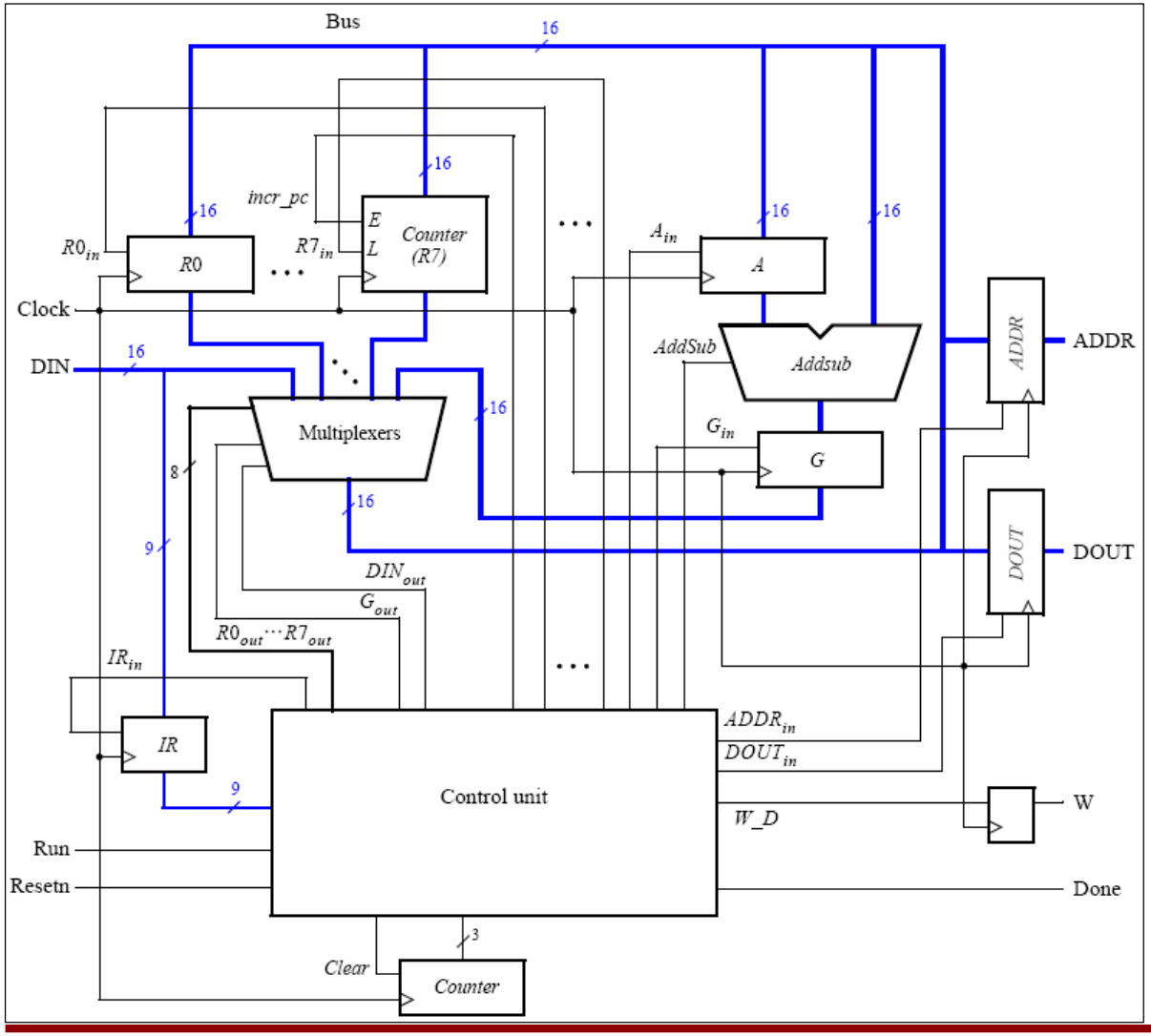
Pozostałe

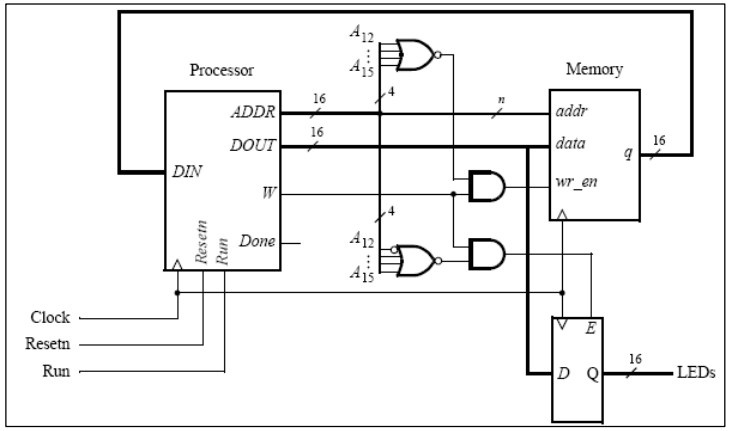
- Układy wejścia / wyjścia
  - porty I/O
- Układy peryferyjne
  - kontrolery interfejsów szeregowych / równoległych
  - przetworniki
  - układy liczników / timerów
- Rejestry specjalne
  - kontrola trybów oszczędzania energii
  - tryby pracy pamięci zewnętrznej

Własny procesor?

- Stworzyć model programowy
- Opracować oraz zadecydować o kluczowych założeniach architektury
- Zaprojektować
- Zweryfikować założenia (CPLD, FPGA, TTL)

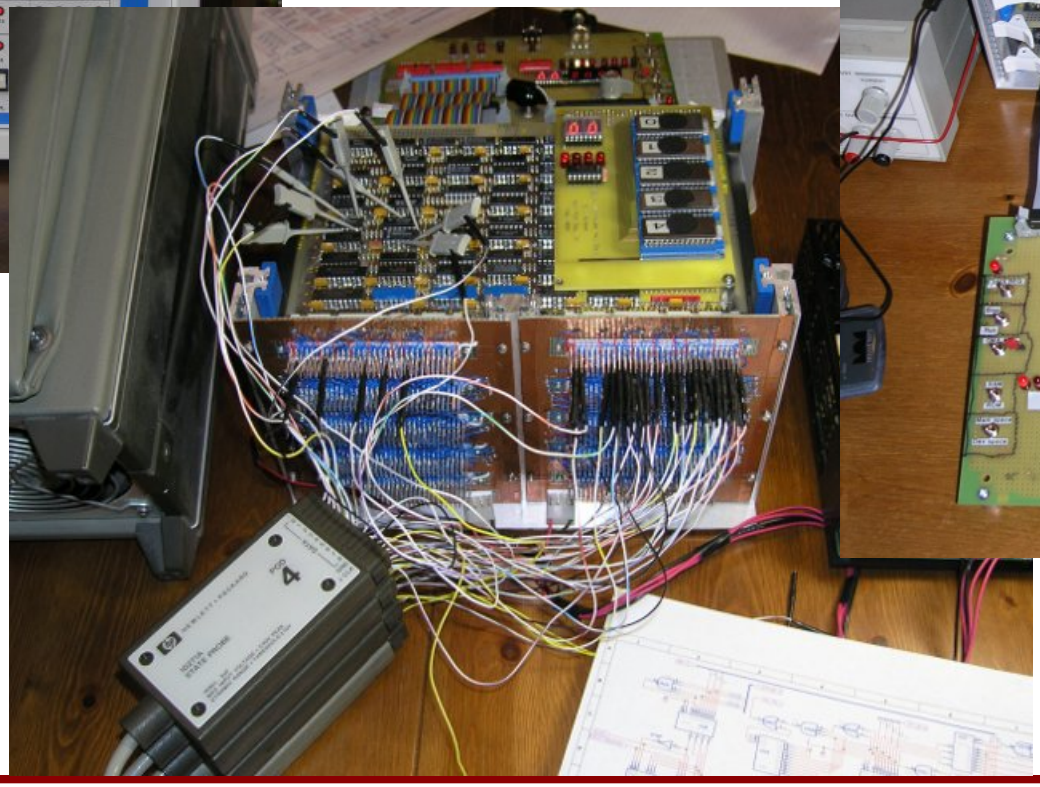
Własny procesor?

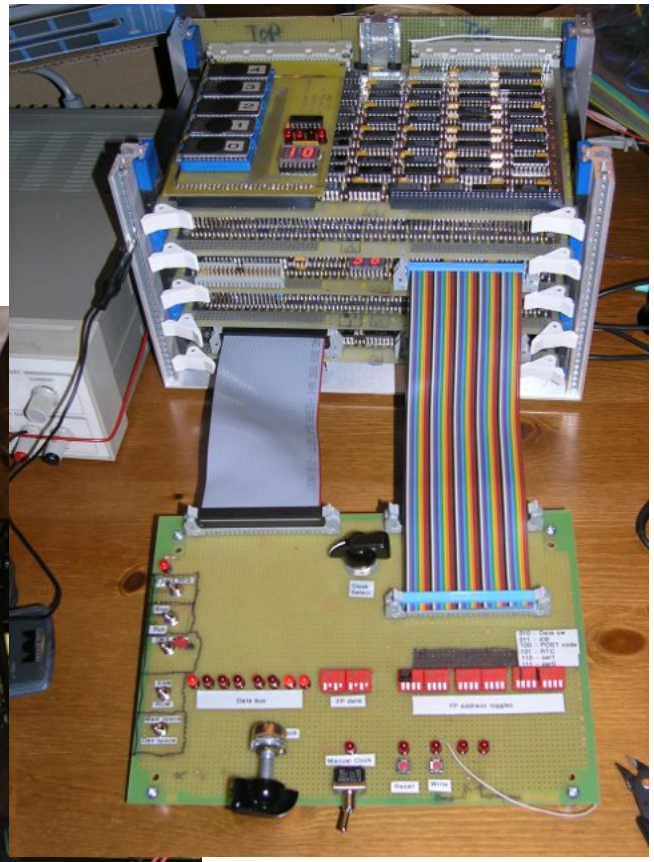





```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_signed.all;

ENTITY proc IS PORT (
  DIN : IN STD_LOGIC_VECTOR(15 DOWNTO 0);
  DOUT : OUT STD_LOGIC_VECTOR(15 DOWNTO 0);
  ADDR : OUT STD_LOGIC_VECTOR(15 DOWNTO 0);

  W : OUT STD_LOGIC;

  Resetn, Clock, Run : IN STD_LOGIC;
  Done : BUFFER STD_LOGIC;
  BusWires : BUFFER
             STD_LOGIC_VECTOR(15 DOWNTO 0));
END proc;

... ~1000 linii VHDL
```

Własny procesor?





http://www.homebrewcpu.com

Aktualne trendy

- System On Chip - układ scalony zawierający kompletny system elektroniczny, w tym układy cyfrowe, analogowe (w tym radiowe) oraz cyfrowo-analogowe.
- Opracowanie i sprzedaż modelu programowego, architektury – firma ARM.
  - aktualnie jeden z najczęściej stosowanych procesorów na świecie
- Soft CPU

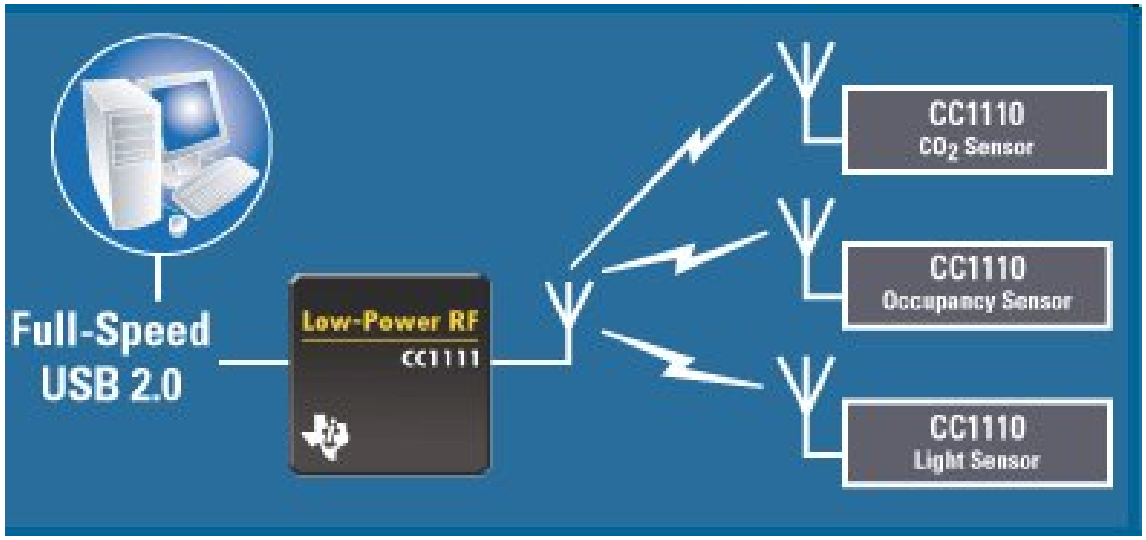
System-on-Chip (SoC)

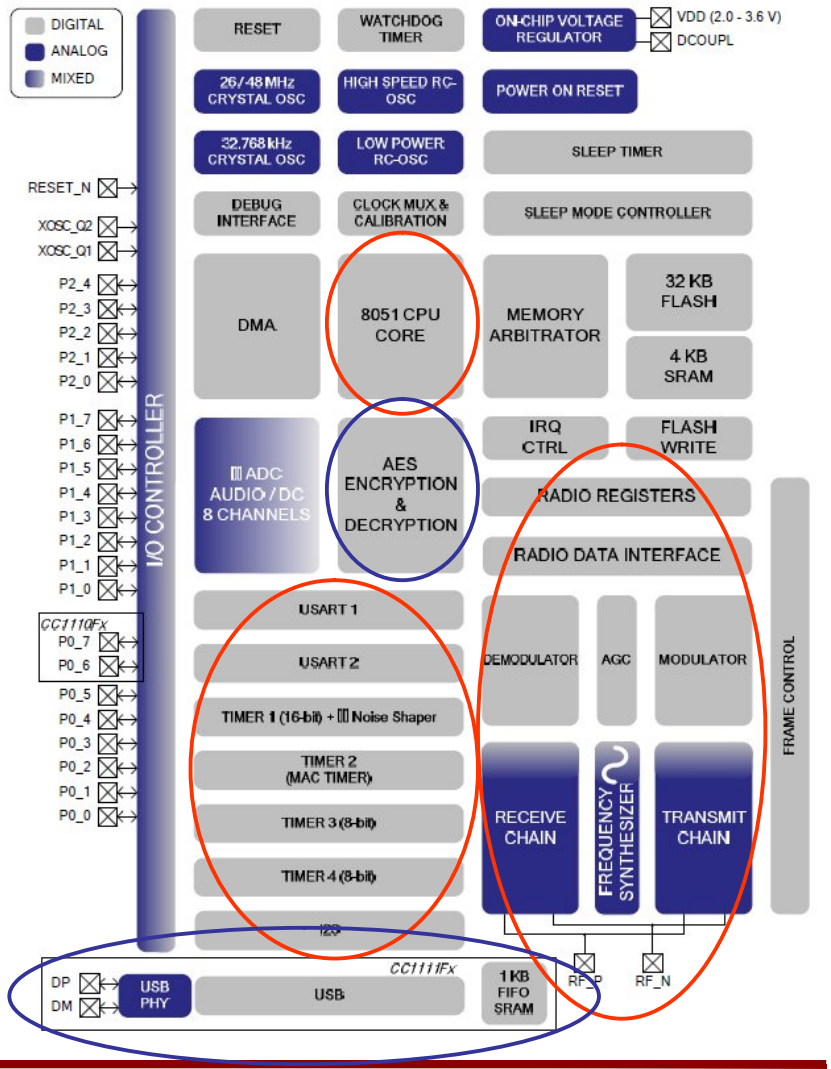
- **SoC** *(ang. System-on-a-chip)* – układ scalony zawierający kompletny system elektroniczny. Poszczególne moduły tego systemu, ze względu na ich złożoność, pochodzą zwykle od różnych dostawców. Przykładowo jednostka centralna pochodzi od jednego dostawcy, a porty komunikacji szeregowej od innego (np. ARM).

http://pl.wikipedia.org

System On Chip

- CC1110 – Radio + MCU + Flash + AES
- CC1111 – Radio + MCU + Flash + AES + USB



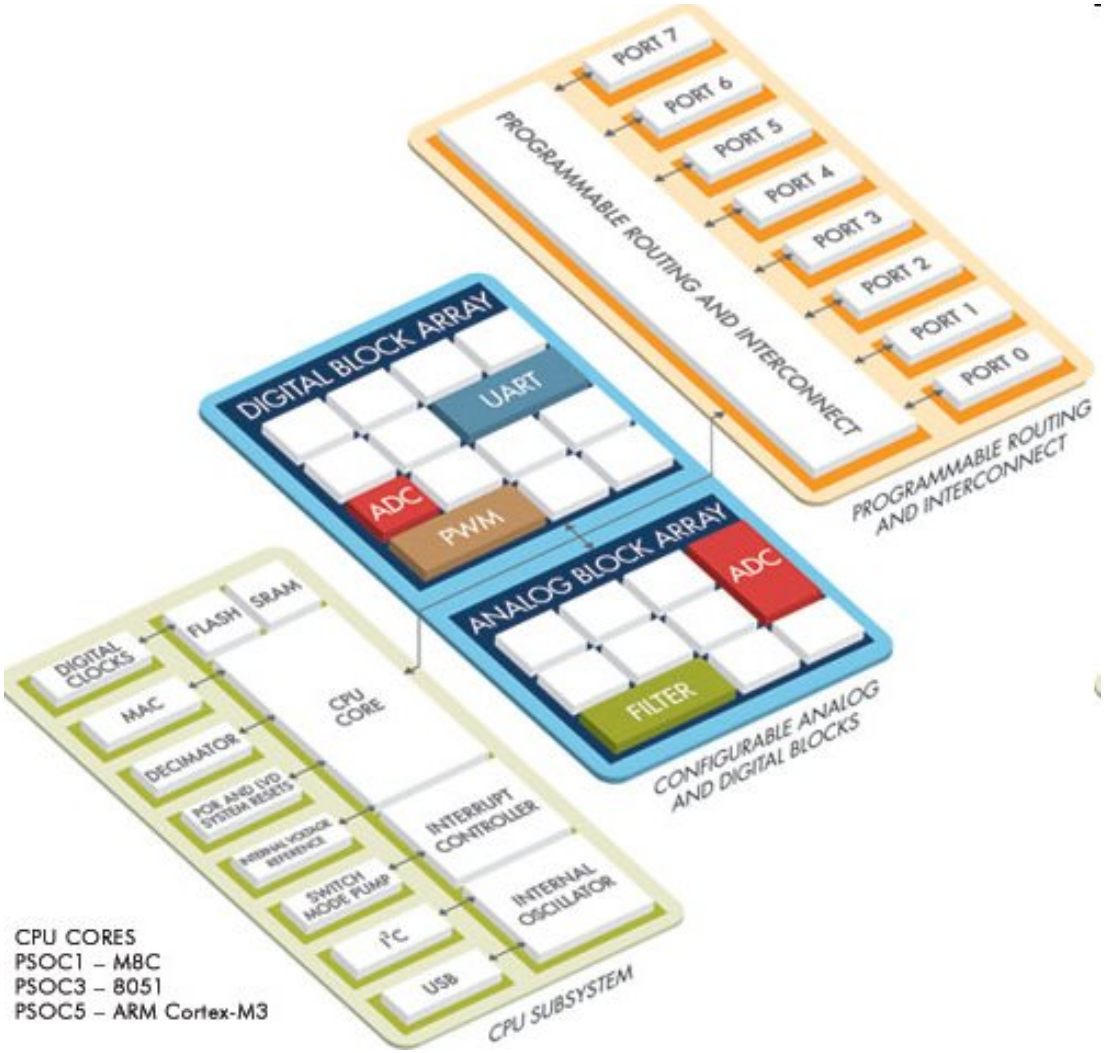


Programmable System On Chip (PSoC)

- produkt f. Cypress Semiconductors

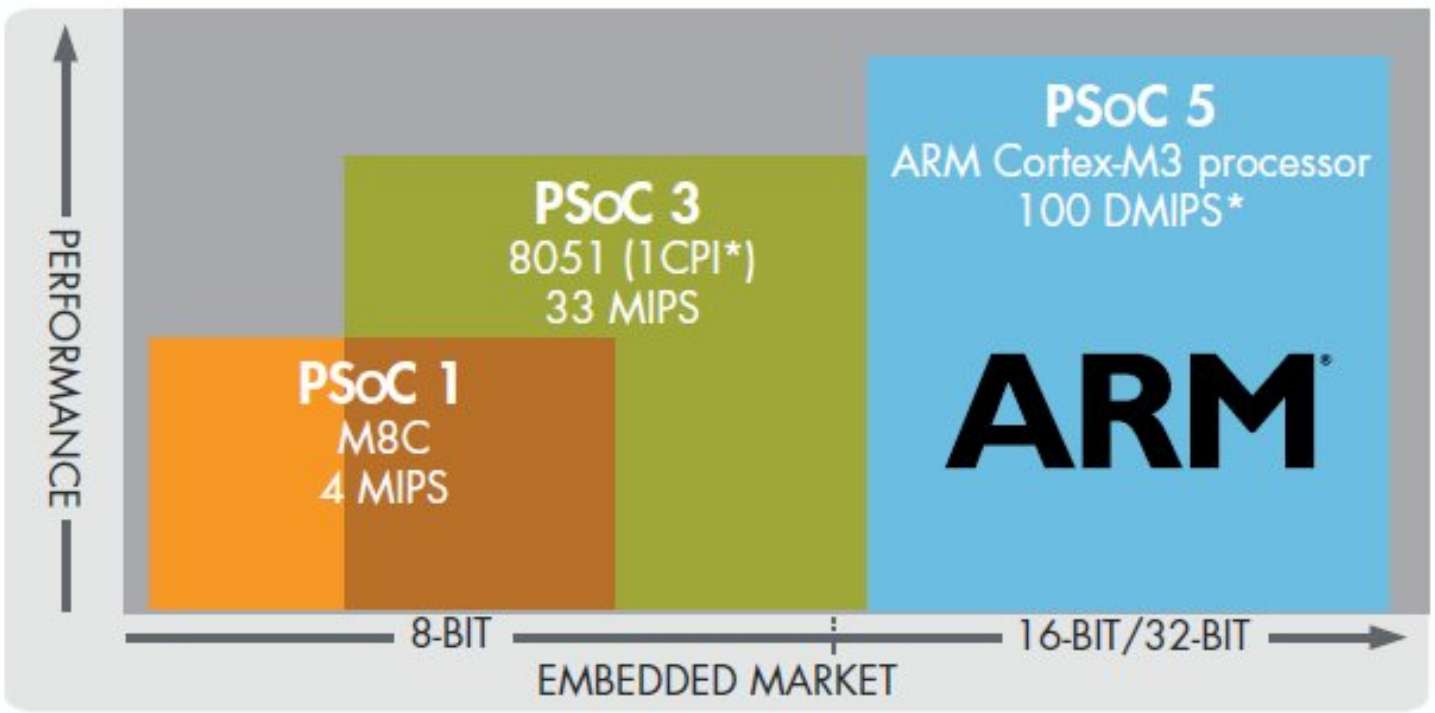
- CPU:
  - PSOC1: M8C (8-bit, arch. harwardzka),
  - PSOC3: 8051
  - PSOC5: ARM Cortex-M3
- programowane w systemie (ISP)
- zintegrowane peryferia cyfrowe oraz analogowe ułożone w konfigurowalnej matrycy połączeń (*mixed-signal array*)

Programmable System On Chip (PSoC)

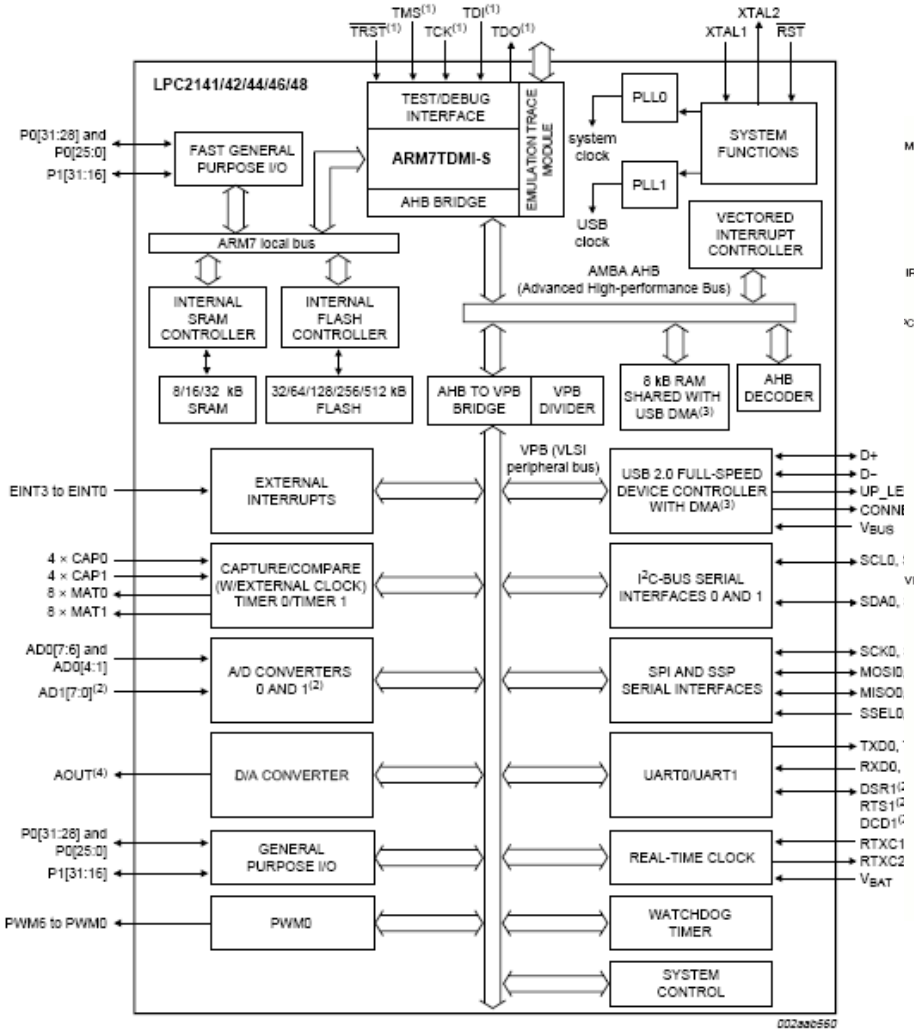


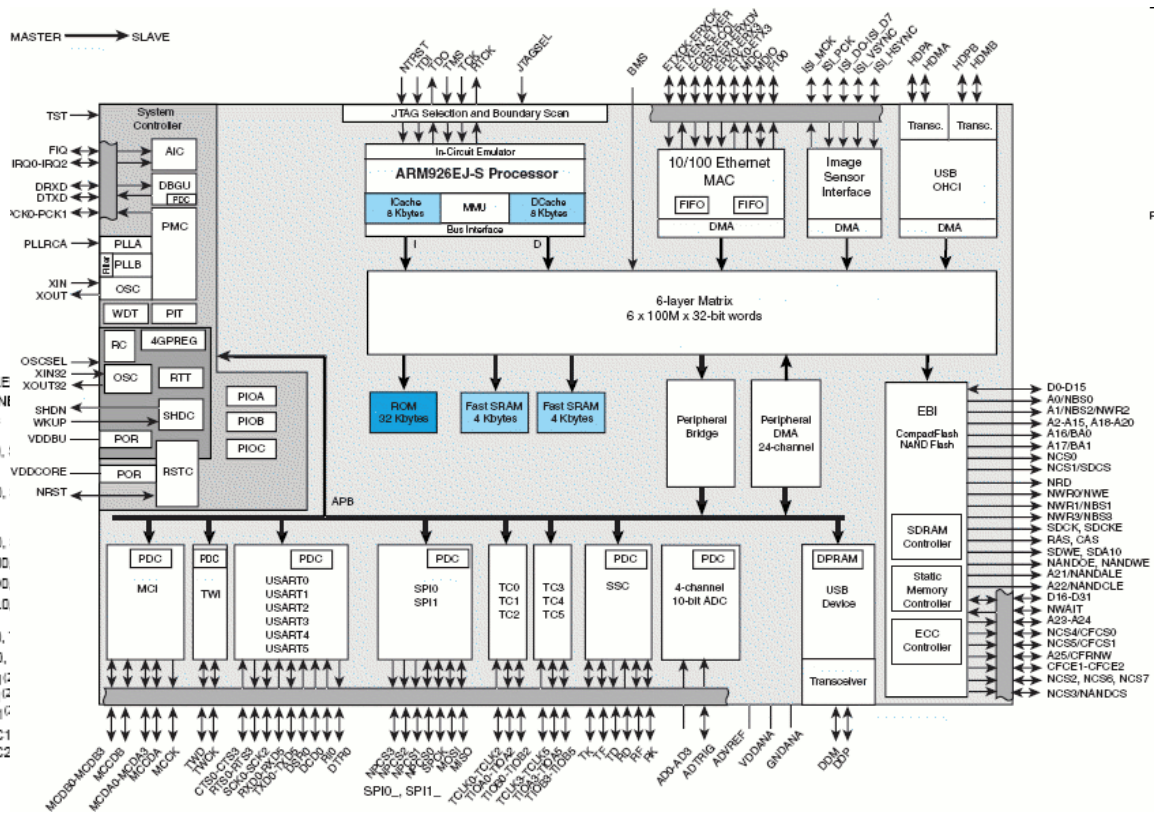
Programmable System On Chip (PSoC)

FEATURE	PSoC 1	PSoC 3	PSoC 5
INTERFACE	SPI, UART, GPIO FS-USB, I²C	PSoC 1, plus: CAN, I²S	Same as PSoC 3
INPUTS	Sensors: CapSen touchscreen,		
OUTPUTS	LED contr motor con analog bu		
PROCESSING	M8 24 MH		

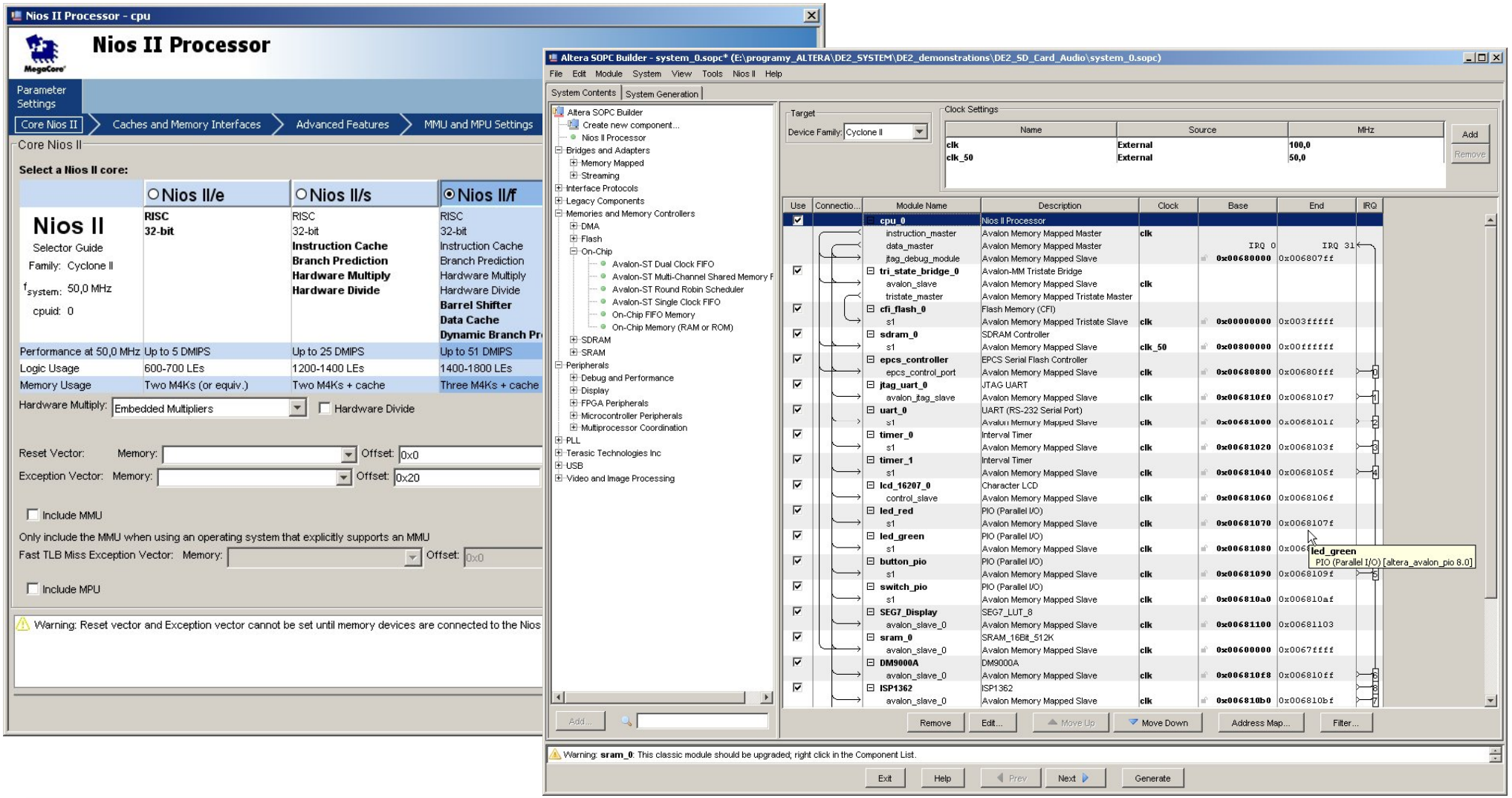


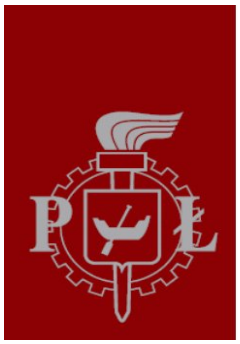
ARM


LPC2141/42/44/46/48
TEST/DEBUG INTERFACE
ARM7TDMI-S
AHB BRIDGE
EMULATION TRACE MODULE
FAST GENERAL PURPOSE I/O
ARM7 local bus
INTERNAL SRAM CONTROLLER
INTERNAL FLASH CONTROLLER
8/16/32 kB SRAM
32/64/128/256/512 kB FLASH
AHB TO VPB BRIDGE
VPB DIVIDER
AHB DECODER
8 kB RAM SHARED WITH USB DMA(3)
AMBA AHB (Advanced High-performance Bus)
VECTORED INTERRUPT CONTROLLER
SYSTEM FUNCTIONS
PLL0
PLL1
system clock
USB clock
VPB (VLSI peripheral bus)
EXTERNAL INTERRUPTS
CAPTURE/COMPARE (W/EXTERNAL CLOCK) TIMER 0/TIMER 1
A/D CONVERTERS 0 AND 1(2)
D/A CONVERTER
GENERAL PURPOSE I/O
PWM0
USB 2.0 FULL-SPEED DEVICE CONTROLLER WITH DMA(3)
I2C-BUS SERIAL INTERFACES 0 AND 1
SPI AND SSP SERIAL INTERFACES
UART0/UART1
REAL-TIME CLOCK
WATCHDOG TIMER
SYSTEM CONTROL
TMS(1) TDI(1) TRST(1) TCK(1) TDO(1)
XTAL2 XTAL1 RST
P0[31:28] and P0[25:0]
P1[31:16]
EINT3 to EINT0
4 × CAP0
4 × CAP1
8 × MAT0
8 × MAT1
AD0[7:6] and AD0[4:1]
AD1[7:0](2)
AOUT(4)
PWM6 to PWM0
D+
D−
VBUS
SCL0,
SDA0,
SCK0,
MOSI0
MISO0
SSEL0
TXD0,
RXD0,
RTXC1
RTXC2
VBAT
002aac960


MASTER → SLAVE
JTAG Selection and Boundary Scan
In-Circuit Emulator
ARM926EJ-S Processor
ICache 8 Kbytes
MMU
DCache 8 Kbytes
Bus Interface
10/100 Ethernet MAC
Image Sensor Interface
USB OHCI
6-layer Matrix
6 x 100M x 32-bit words
ROM 32 Kbytes
Fast SRAM 4 Kbytes
Fast SRAM 4 Kbytes
Peripheral Bridge
Peripheral DMA 24-channel
EBI
CompactFlash NAND Flash
SDRAM Controller
Static Memory Controller
ECC Controller
System Controller
AIC
DBGU
PMC
PLLA
PLLB
OSC
WDT
PIT
RC
4GPREG
RTT
SHDC
POR
RSTC
PIOA
PIOB
PIOC
APB
PDC
MCI
TWI
USART0 USART1 USART2 USART3 USART4 USART5
SPI0 SPI1
TC0 TC1 TC2
TC3 TC4 TC5
SSC
4-channel 10-bit ADC
DPRAM
USB Device
Transceiver
SDRAM Controller
Static Memory Controller
ECC Controller
D0-D15
A0/NBS0
A1/NBS2/NWR2
A2-A15, A18-A20
A16/BA0
A17/BA1
NCS0
NCS1/SDCS
NRD
NWR0/NWE
NWR1/NBS1
NWR3/NBS3
SDCK, SDCKE
RAS, CAS
SDWE, SDA10
NANDOE, NANDWE
A21/NANDALE
A22/NANDCLE
D16-D31
NWAIT
A23-A24
NCS4/CFCS0
NCS5/CFCS1
A25/CFRNW
CFCE1-CFCE2
NCS2, NCS6, NCS7
NCS3/NANDCS

Soft CPU


Nios II Processor - cpu
Nios II Processor
Parameter Settings
Core Nios II
Caches and Memory Interfaces
Advanced Features
MMU and MPU Settings
Core Nios II
Select a Nios II core:
Nios II/e
Nios II/s
Nios II/f
Nios II
Selector Guide
Family: Cyclone II
f system: 50,0 MHz
cpuid: 0
RISC
32-bit
Instruction Cache
Branch Prediction
Hardware Multiply
Hardware Divide
Barrel Shifter
Data Cache
Dynamic Branch Pr
Performance at 50,0 MHz
Up to 5 DMIPS
Up to 25 DMIPS
Up to 51 DMIPS
Logic Usage
600-700 LEs
1200-1400 LEs
1400-1800 LEs
Memory Usage
Two M4Ks (or equiv.)
Two M4Ks + cache
Three M4Ks + cache
Hardware Multiply:
Embedded Multipliers
Hardware Divide
Reset Vector:
Memory:
Offset:
0x0
Exception Vector:
Offset:
0x20
Include MMU
Only include the MMU when using an operating system that explicitly supports an MMU
Fast TLB Miss Exception Vector:
Include MPU
Warning: Reset vector and Exception vector cannot be set until memory devices are connected to the Nios
Altera SOPC Builder - system_0.sopc* (E:\programy_ALTERA\DE2_SYSTEM\DE2_demonstrations\DE2_SD_Card_Audio\system_0.sopc)
File Edit Module System View Tools Nios II Help
System Contents
System Generation
Altera SOPC Builder
Create new component...
Nios II Processor
Bridges and Adapters
Memory Mapped
Streaming
Interface Protocols
Legacy Components
Memories and Memory Controllers
DMA
Flash
On-Chip
Avalon-ST Dual Clock FIFO
Avalon-ST Multi-Channel Shared Memory F
Avalon-ST Round Robin Scheduler
Avalon-ST Single Clock FIFO
On-Chip FIFO Memory
On-Chip Memory (RAM or ROM)
SDRAM
SRAM
Peripherals
Debug and Performance
Display
FPGA Peripherals
Microcontroller Peripherals
Multiprocessor Coordination
PLL
Terasic Technologies Inc
USB
Video and Image Processing
Target
Device Family: Cyclone II
Clock Settings
Name
Source
MHz
clk
External
100,0
clk_50
External
50,0
Add
Remove
Use
Connectio...
Module Name
Description
Clock
Base
End
IRQ
cpu_0
Nios II Processor
instruction_master
Avalon Memory Mapped Master
clk
data_master
Avalon Memory Mapped Master
IRQ 0
IRQ 31
jtag_debug_module
Avalon Memory Mapped Slave
0x00680000
0x006807ff
tri_state_bridge_0
Avalon-MM Tristate Bridge
avalon_slave
Avalon Memory Mapped Slave
clk
tristate_master
Avalon Memory Mapped Tristate Master
cfi_flash_0
Flash Memory (CFI)
s1
Avalon Memory Mapped Tristate Slave
clk
0x00000000
0x003fffff
sdram_0
SDRAM Controller
s1
Avalon Memory Mapped Slave
clk_50
0x00800000
0x00ffffff
epcs_controller
EPCS Serial Flash Controller
epcs_control_port
Avalon Memory Mapped Slave
clk
0x00680800
0x00680fff
jtag_uart_0
JTAG UART
avalon_jtag_slave
Avalon Memory Mapped Slave
clk
0x006810f0
0x006810f7
uart_0
UART (RS-232 Serial Port)
s1
Avalon Memory Mapped Slave
clk
0x00681000
0x0068101f
timer_0
Interval Timer
s1
Avalon Memory Mapped Slave
clk
0x00681020
0x0068103f
timer_1
Interval Timer
s1
Avalon Memory Mapped Slave
clk
0x00681040
0x0068105f
lcd_16207_0
Character LCD
control_slave
Avalon Memory Mapped Slave
clk
0x00681060
0x0068106f
led_red
PIO (Parallel I/O)
s1
Avalon Memory Mapped Slave
clk
0x00681070
0x0068107f
led_green
PIO (Parallel I/O)
s1
Avalon Memory Mapped Slave
clk
0x00681080
button_pio
PIO (Parallel I/O)
s1
Avalon Memory Mapped Slave
clk
0x00681090
0x0068109f
switch_pio
PIO (Parallel I/O)
s1
Avalon Memory Mapped Slave
clk
0x006810a0
0x006810af
SEG7_Display
SEG7_LUT_8
avalon_slave_0
Avalon Memory Mapped Slave
clk
0x00681100
0x00681103
sram_0
SRAM_16Bit_512K
avalon_slave_0
Avalon Memory Mapped Slave
clk
0x00600000
0x0067ffff
DM9000A
DM9000A
avalon_slave_0
Avalon Memory Mapped Slave
clk
0x006810f8
0x006810ff
ISP1362
ISP1362
avalon_slave_0
Avalon Memory Mapped Slave
clk
0x006810b0
0x006810bf
led_green
PIO (Parallel I/O) [altera_avalon_pio 8.0]
Add...
Remove
Edit...
Move Up
Move Down
Address Map...
Filter...
Warning: sram_0: This classic module should be upgraded; right click in the Component List.
Exit
Help
Prev
Next
Generate



Politechnika Łódzka

Instytut Elektroniki

Dziękuję za uwagę.