Programowanie mikrokontrolerów

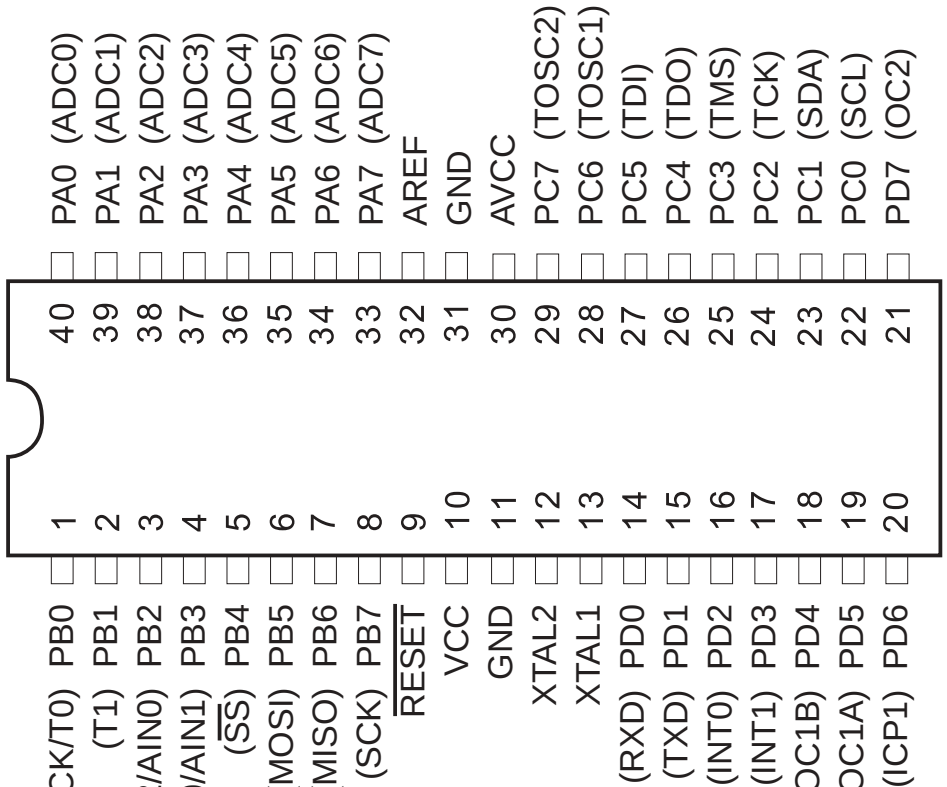
ATmega16 i ATmega32

Marcin Engel Marcin Peczarski

Instytut Informatyki Uniwersytetu Warszawskiego

1 października 2012

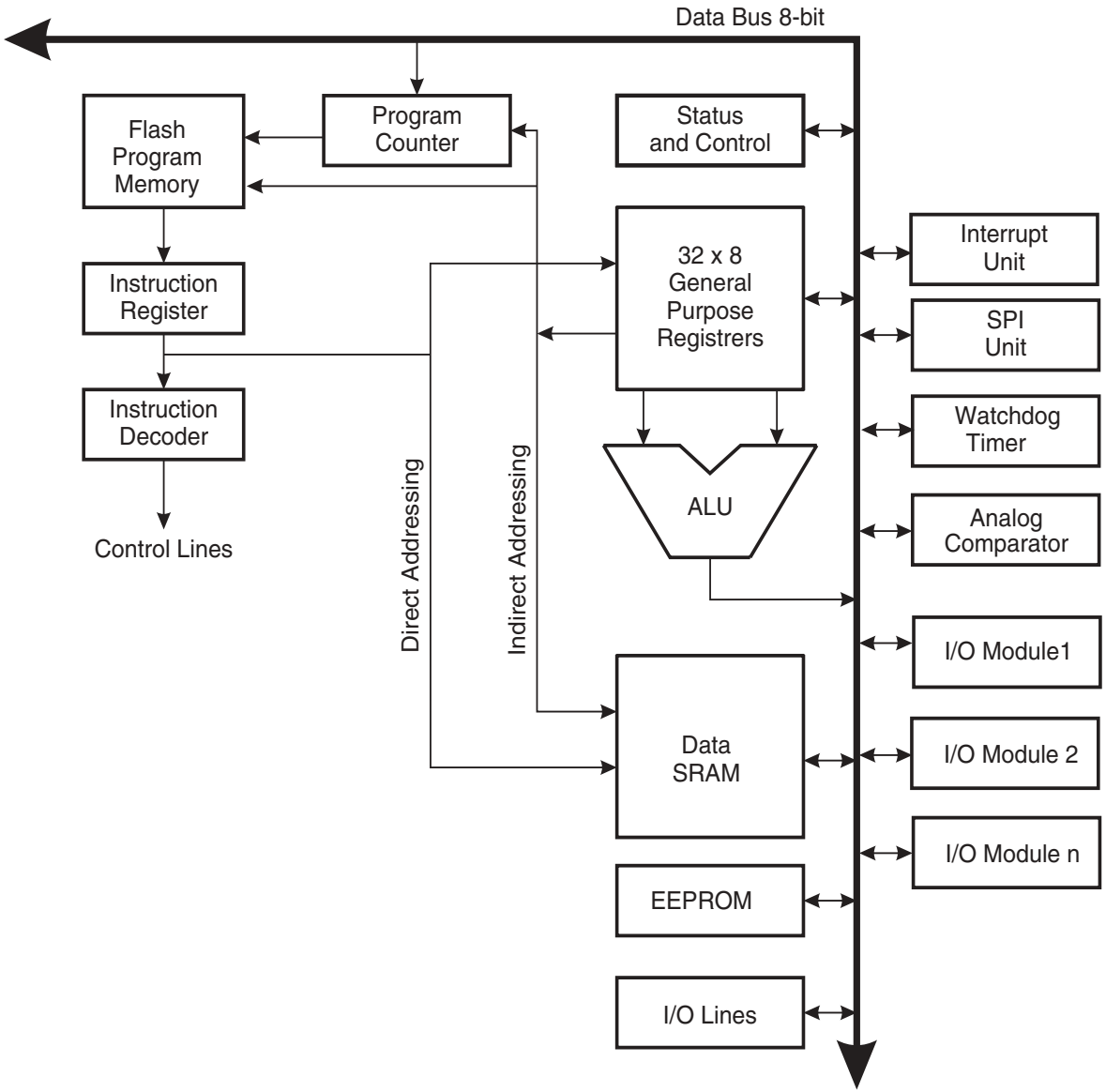
Poznajemy ATmega16 i ATmega32



Architektura mikrokontrolera ATmega16 i ATmega32

- Procesor o zredukowanym zbiorze rozkazów (RISC)
- Architektura little-endian
- Architektura harwardzka (odrębne pamięci i magistrale dla programu i danych)
- 16 KiB lub 32 KiB pamięci programu (pamięć Flash)
- 1 KiB lub 2 KiB pamięci danych (SRAM)
- 32 rejestry ogólnego przeznaczenia
- 64 rejestry wejścia-wyjścia, układy peryferyjne
- 512 lub 1024 bajtów pamięci nieulotnej (EEPROM)
- 3 liczniki
- 21 przerwań (o ustalonej kolejności obsługi)
- Interfejsy szeregowe: USART, I^2C
- 8-kanałowy, 10-bitowy przetwornik A/C

Architektura, schemat blokowy



Najprostszy układ

Podłączamy:

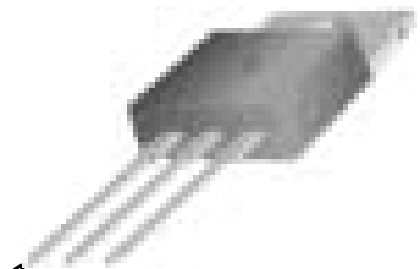
- zasilanie (wyprowadzenia VCC, AVCC, GND),
- zerowanie (wyprowadzenie $\overline{\text{RESET}}$),
- złącze programatora (wyprowadzenia MOSI, MISO, SCK, $\overline{\text{RESET}}$, VCC, GND).

Wyjaśnienia:

- VCC na schematach oznacza „+” zasilania (inne oznaczenie to VDD),
- AVCC oznacza „+” zasilania części analogowej,
- GND to „−” zasilania,
- $\overline{\text{RESET}}$ oznacza odwróconą logikę (tj. zerowanie następuje, gdy podamy niski poziom napięcia).

Zasilanie

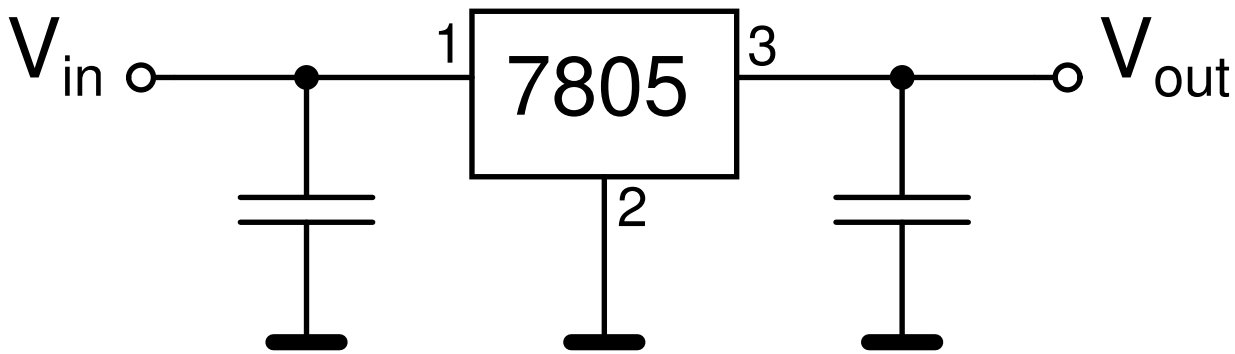
- Układy ATmega16 i ATmega32 wymagają zasilania napięciem stałym z zakresu od 4,5 do 5,5 V (wersje ATmega16L i ATmega32L od 2,7 V).
- Nowsza wersja ATmega16A i ATmega32A może być zasilana napięciem z zakresu od 2,7 V do 5,5 V.
- Zwykle stosujemy scalony stabilizator napięcia 5 V typu 7805.



1

Zasilanie, cd.

- Schemat połączeń znajduje się w nocie katalogowej.

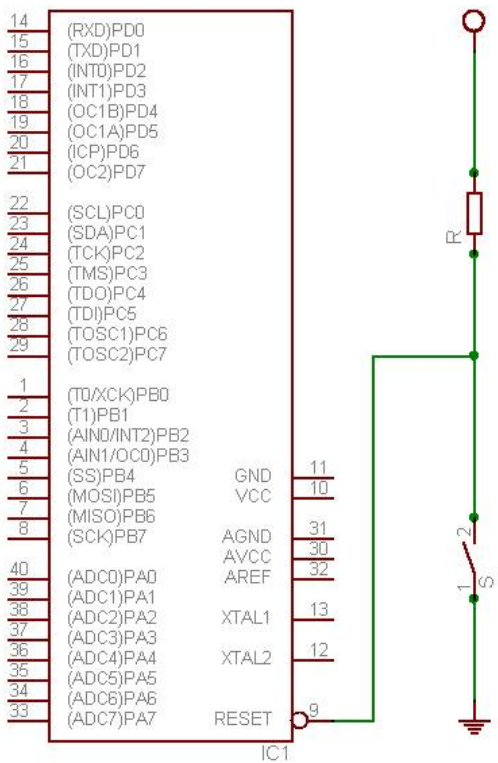


- Na płytce testowej znajduje się mostek prostowniczy i stabilizator o regulowanym napięciu wyjściowym.

Układ zerowania (ang. *reset*)

- Normalnie na nóżce $\overline{\text{RESET}}$ powinien być stan wysoki.
- Chcemy móc ręcznie wyzerować urządzenie (np. przyciskiem).
- Programator również musi móc wyzerować układ.
- Rezystor podciągający na ogół nie jest potrzebny (jest wewnętrzny).

Rozwiązanie:



Przestrzeń adresowa

Są trzy rozdzielne przestrzenie adresowe:

- pamięć danych
  - zorganizowana w bajty
  - rejestry ogólnego przeznaczenia
  - rejestry wejścia-wyjścia
  - SRAM
- nieulotna pamięć danych
  - zorganizowana w bajty
  - EEPROM
  - dostęp przez rejestry wejścia-wyjścia za pomocą odpowiedniej sekwencji rozkazów
- pamięć programu
  - zorganizowana w słowa 16-bitowe
  - przy dostępie do rozkazów adresowana słowami
  - przy dostępie do danych adresowana bajtami

Rejestry

7	0	Addr.	
R0		$00	
R1		$01	
R2		$02	
…			
R13		$0D	
R14		$0E	
R15		$0F	
R16		$10	
R17		$11	
…			
R26		$1A	X-register Low Byte
R27		$1B	X-register High Byte
R28		$1C	Y-register Low Byte
R29		$1D	Y-register High Byte
R30		$1E	Z-register Low Byte
R31		$1F	Z-register High Byte

- 32 rejestry 8-bitowe: `R0`, . . . , `R31`
- W rozkazach z adresowaniem natychmiastowym można stosować tylko rejestry `R16`, . . . , `R31`.

```
LDI R16, 123
SUBI R18, 5
```

- Przesłania wartości są możliwe między każdą parą rejestrów.

```
MOV R3, R19
```

- Operacje arytmetyczno-logiczne można wykonywać na danych w dowolnych rejestrach.

```
ADD R9, R17
```

Rejestry

7 0	Addr.	
R0	$00	
R1	$01	
R2	$02	
…		
R13	$0D	
R14	$0E	
R15	$0F	
R16	$10	
R17	$11	
…		
R26	$1A	X-register Low Byte
R27	$1B	X-register High Byte
R28	$1C	Y-register Low Byte
R29	$1D	Y-register High Byte
R30	$1E	Z-register Low Byte
R31	$1F	Z-register High Byte

- Do adresowania pośredniego pamięci danych używa się par rejestrów:
  - para rejestrów `R27:R26` to rejestr `X`,
  - para rejestrów `R29:R28` to rejestr `Y`,
  - para rejestrów `R31:R30` to rejestr `Z`.

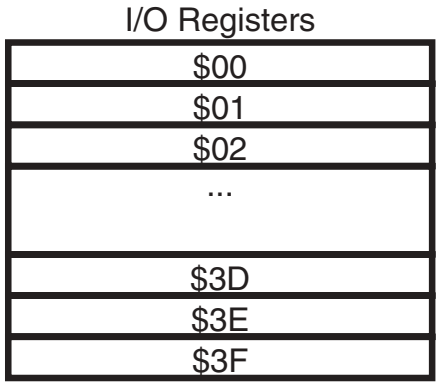
  ```
  LD R16, X
  ST Y, R17
  ```
- Rejestr `Z` można stosować do adresowania pośredniego pamięci programu (rozkazy `LPM`, `SPM`).

Rejestry

7 0	Addr.	
R0	$00	
R1	$01	
R2	$02	
...		
R13	$0D	
R14	$0E	
R15	$0F	
R16	$10	
R17	$11	
...		
R26	$1A	X-register Low Byte
R27	$1B	X-register High Byte
R28	$1C	Y-register Low Byte
R29	$1D	Y-register High Byte
R30	$1E	Z-register Low Byte
R31	$1F	Z-register High Byte

- Rejestry `R24`, . . . , `R31` mogą być używane, do operacji na danych 16-bitowych.
  `ADDW R25:R24, 3`
- Para rejestrów `R1:R0` jest domyślnym miejscem wyniku operacji mnożenia.
- Możliwe są przesłania 16-bitowe między parami rejestrów.
  `MOVW R13:R12, R17:R16`
- Rejestry są mapowane na adresy od $00 do $1F w przestrzeni adresowej danych.

Pamięć wejścia-wyjścia



- Zawiera 64 rejestry wejścia-wyjścia odpowiedzialne m.in. za konfigurację poszczególnych układów wejścia-wyjścia (np. rejestry `PORTx`, `DDRx`, `SP`, `SREG`).
- Adresowane od $00 do $3F.
- Adresy mapowane na adresy od $20 do $5F pamięci danych.
- Zapis odbywa się za pomocą rozkazu `OUT`, a odczyt za pomocą rozkazu `IN`.
- Na pierwszych 32 rejestrach we/wy można bezpośrednio ustawiać bity (rozkazy `SBI`, `CBI`) i je sprawdzać (rozkazy `SBIS`, `SBIC`).

Rejestr stanu

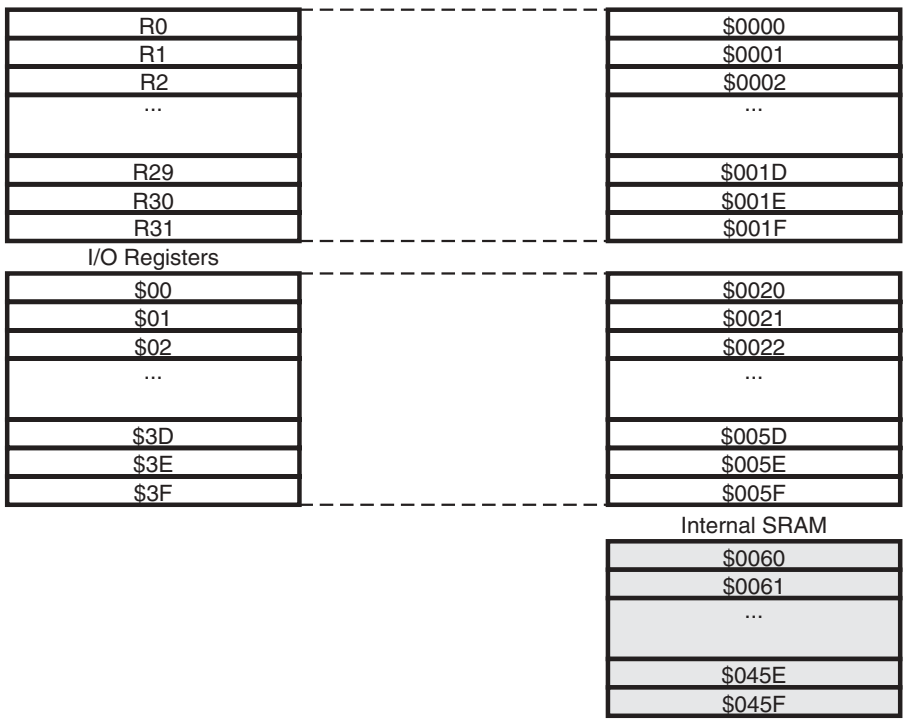
- Znajduje się w pamięci we-wy pod adresem SREG ($3F).
- Jest uaktualniany po każdej operacji arytmetyczno-logicznej.

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	

- bit I – włączone przerwania
- bit T – do przechowania dowolnego bitu (rozkazy BLD, BST)
- bit H – przeniesienie z młodszego półbajta do starszego
- bit V – nadmiar w arytmetyce U2
- bit N – najstarszy bit wyniku operacji (bit znaku w U2)
- bit S – znak rzeczywistego wyniku operacji w U2, $S = V \oplus N$
- bit Z – wynikiem operacji jest zero
- bit C – przeniesienie z najstarszego bitu

Pamięć danych – podsumowanie

- Adresowana od \$0000 do \$045F (stała `RAMEND`)
- 1 KiB o adresach od \$0060 do \$045F to pamięć ulotna SRAM
- Na adresy od \$0000 do \$005F są wirtualnie odwzorowane rejestry ogólnego przeznaczenia i pamięć wejścia-wyjścia, np.: rozkazy `OUT $10, R18` oraz `ST $30, R18` mają ten sam efekt! Ale pierwszy działa szybciej.



Stos

- Znajduje się w pamięci danych.
- Przed wykonaniem jakiejkolwiek operacji na stosie programista musi ustawić dwubajtowy wskaźnik stosu `SP` w pamięci we-wy.
- Wskaźnik stosu `SP` składa się z dwóch rejestrów `SPH` i `SPL`.
- Stos rośnie w dół pamięci (od wysokich adresów do niskich).
- `SP` pokazuje zawsze na pierwszy wolny bajt pod wierzchołkiem stosu.
- Rozkaz `PUSH` odkłada jeden bajt na stos, a `POP` zdejmuje jeden bajt ze stosu.
- Wywołanie podprogramu (`RCALL`, `CALL`) odkłada dwa bajty (adres powrotu) na stos, a powrót z niego (`RET`) zdejmuje dwa bajty.

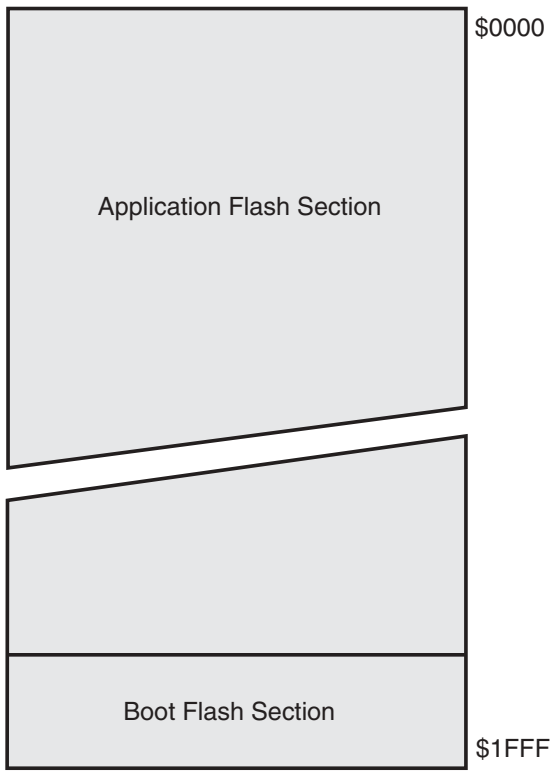
Typowe inicjowanie stosu

```
LDI R16, HIGH (RAMEND)
OUT SPH, R16
LDI R16, LOW (RAMEND)
OUT SPL, R16
```

Nieulotna pamięć danych

- Stanowi odrębną przestrzeń adresową.
- Dostęp do niej odbywa się za pomocą specjalnych rejestrów wejścia-wyjścia (`EEAR`, `EEDR`, `EECR`).
- Protokół dostępu do tej pamięci poznamy w dalszej części wykładu.

Pamięć programu



- Ma 16 KiB lub 32 KiB.
- Jest podzielona na dwie części z niezależną ochroną dostępu: boot loader oraz część aplikacji.
- Większość kodów rozkazów jest 2-bajtowa (1 słowo), ale są też rozkazy 4-bajtowe (2 słowa).
- Program może modyfikować sam siebie.

Tryby adresowania

- natychmiastowe: `LDI R16, 3`
- bezpośrednie: `LDS R1, 100`
- pośrednie: `LD R1, X`
- pośrednie z postinkrementacją: `LD R1, X+`
- pośrednie z predekrementacją: `LD R1, -X`
- pośrednie z przemieszczeniem: `LDD R1, Y+2`
- pośrednie pamięci programu: `LPM R1, Z`

Przerwania

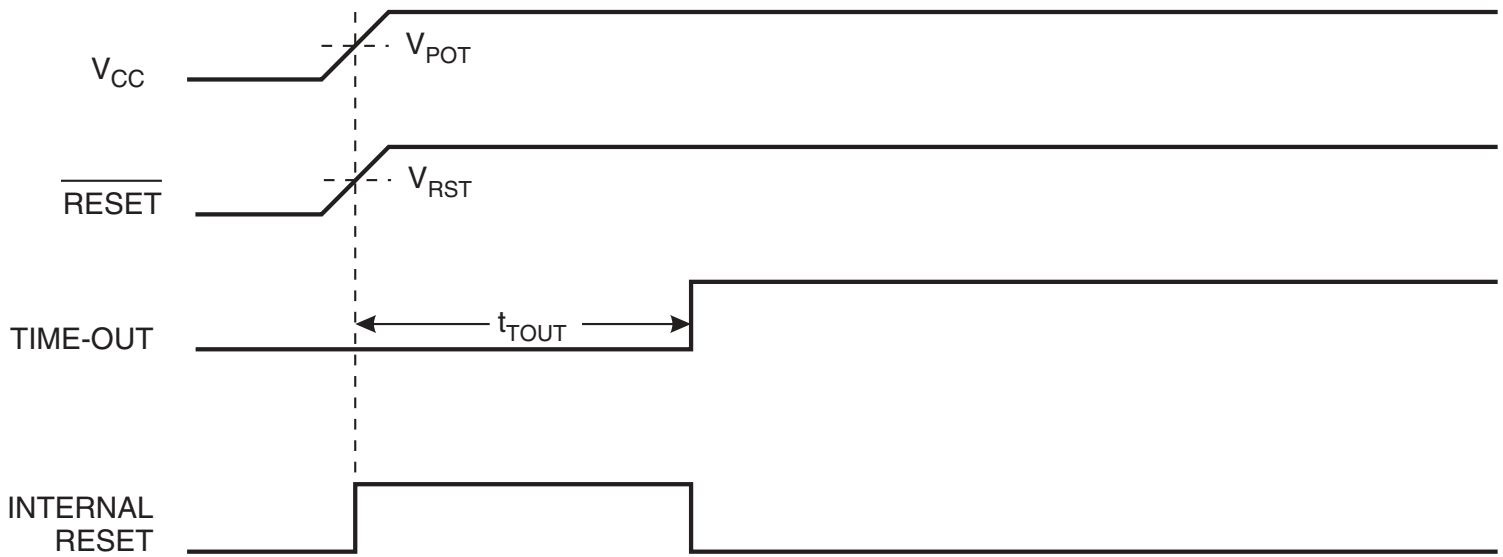
- Kod programu rozpoczyna się od wektora przerwań.
- Zgłoszenie przerwania powoduje sprzętowe:
  - odłożenie na stos adresu powrotu (ale nie rejestru stanu!),
  - zablokowanie przerwań poprzez wyzerowanie bitu I w rejestrze stanu `SREG`,
  - wykonanie rozkazu spod odpowiedniego adresu w pamięci programu.
- Program obsługi przerwania zwykle kończy się rozkazem `RETI`, który włącza przerwania.

Wektor przerwań ATmega16

Vector No.	Program Address	Source	Interrupt Definition
1	$000	RESET	External Pin, Power-on Reset, Brown-out Reset, Watchdog Reset, and JTAG AVR Reset
2	$002	INT0	External Interrupt Request 0
3	$004	INT1	External Interrupt Request 1
4	$006	TIMER2 COMP	Timer/Counter2 Compare Match
5	$008	TIMER2 OVF	Timer/Counter2 Overflow
6	$00A	TIMER1 CAPT	Timer/Counter1 Capture Event
7	$00C	TIMER1 COMPA	Timer/Counter1 Compare Match A
8	$00E	TIMER1 COMPB	Timer/Counter1 Compare Match B
9	$010	TIMER1 OVF	Timer/Counter1 Overflow
10	$012	TIMER0 OVF	Timer/Counter0 Overflow
11	$014	SPI, STC	Serial Transfer Complete
12	$016	USART, RXC	USART, Rx Complete
13	$018	USART, UDRE	USART Data Register Empty
14	$01A	USART, TXC	USART, Tx Complete
15	$01C	ADC	ADC Conversion Complete
16	$01E	EE_RDY	EEPROM Ready
17	$020	ANA_COMP	Analog Comparator
18	$022	TWI	Two-wire Serial Interface
19	$024	INT2	External Interrupt Request 2
20	$026	TIMER0 COMP	Timer/Counter0 Compare Match
21	$028	SPM_RDY	Store Program Memory Ready

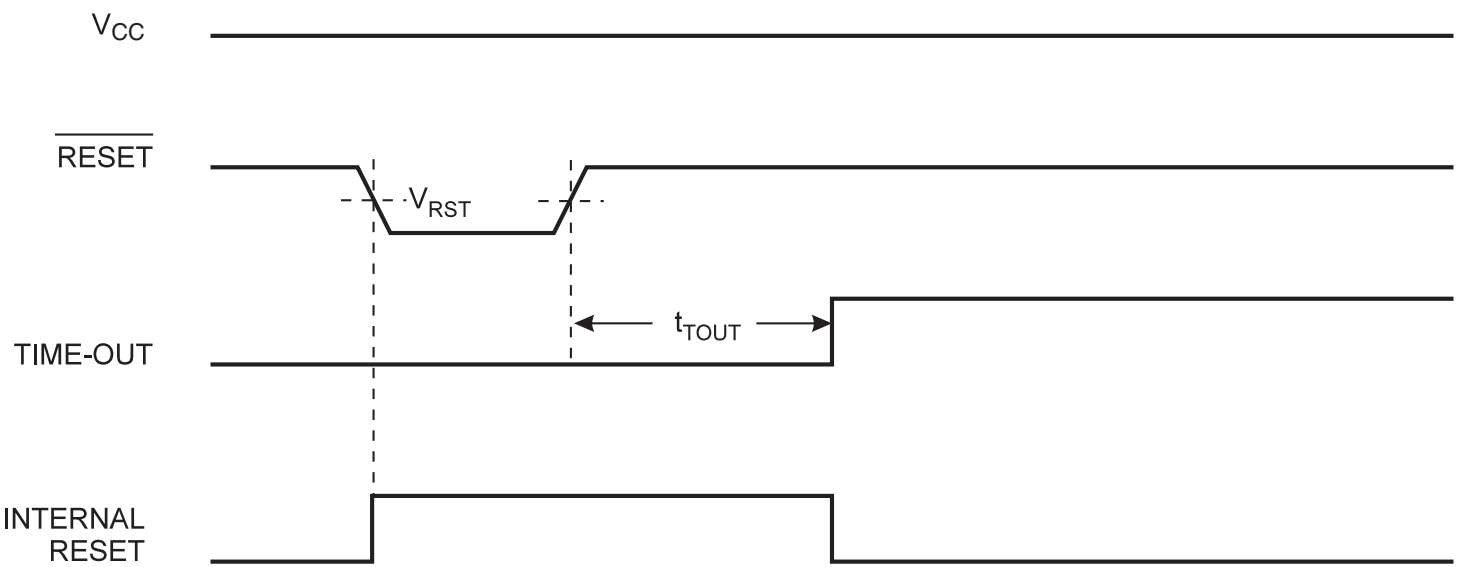
Kiedy pojawia się przerwanie RESET

Po włączeniu zasilania (ang. *power-on reset*)



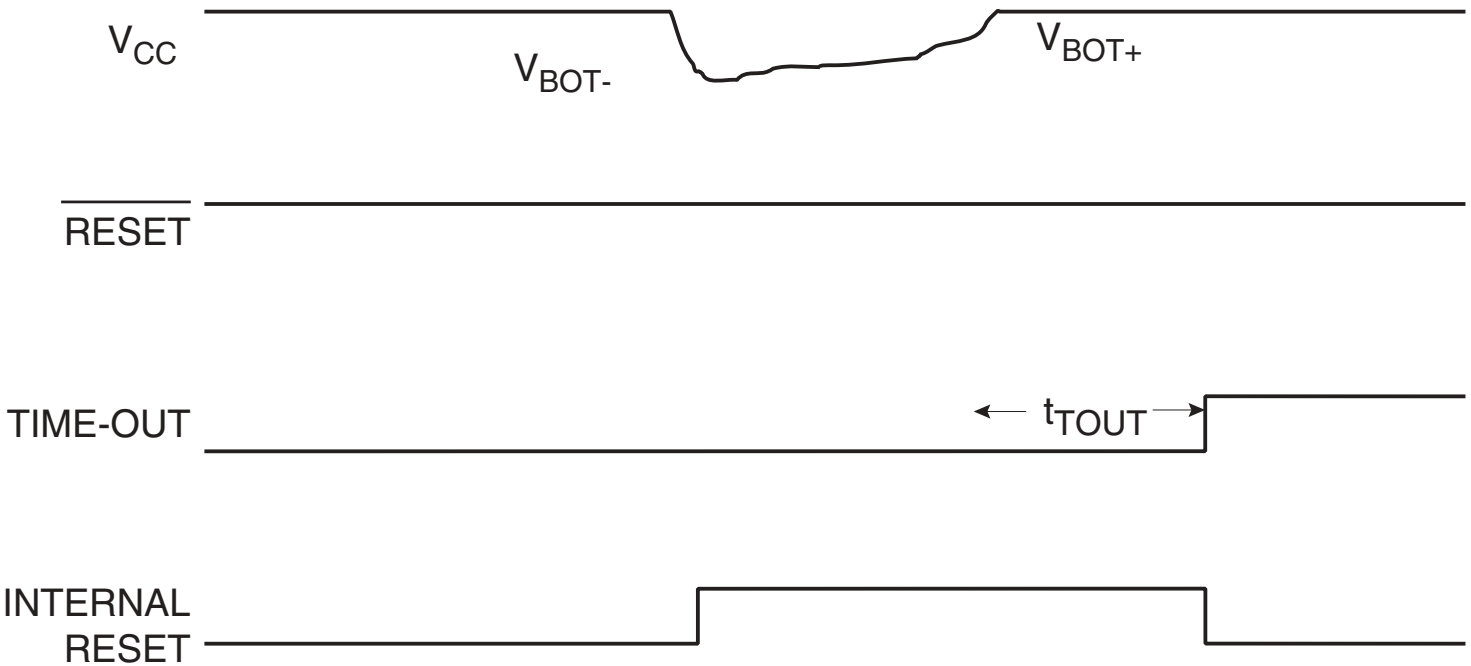
Kiedy pojawia się przerwanie RESET

Na skutek zewnętrznego wyzerowania



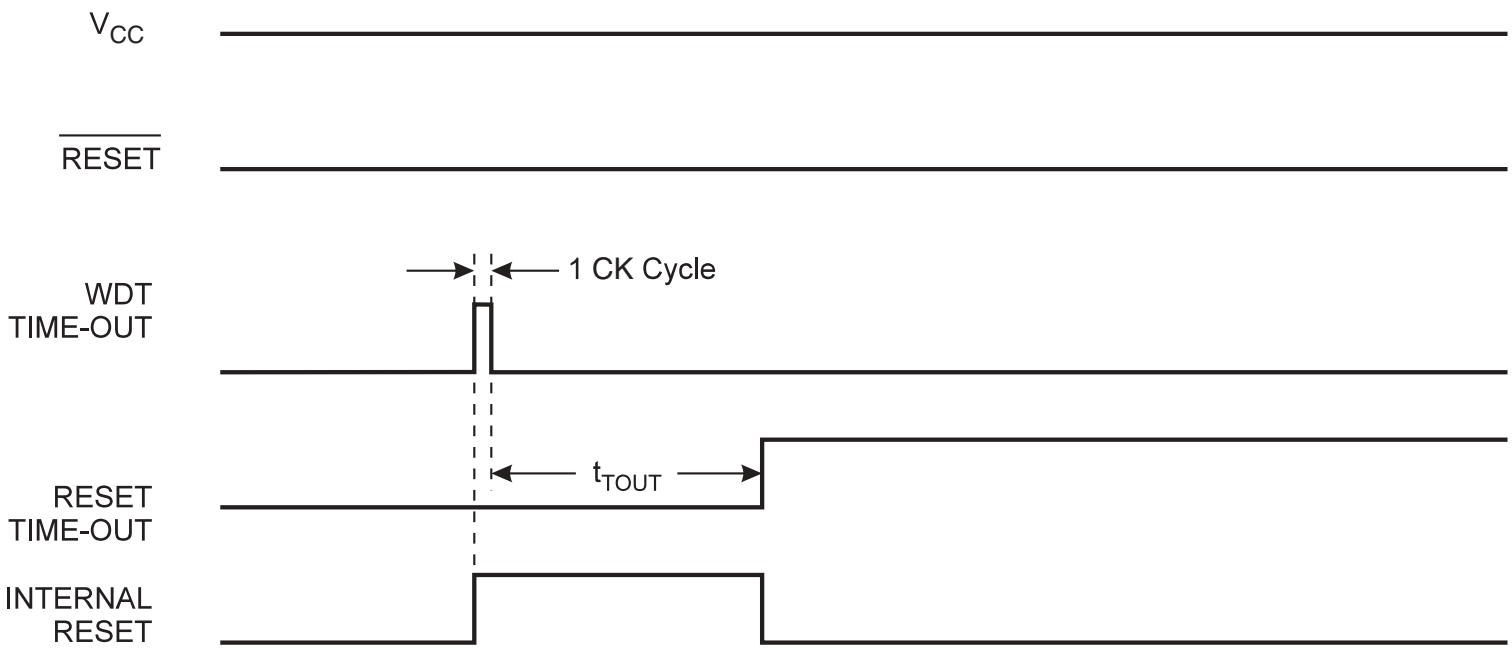
Kiedy pojawia się przerwanie RESET

Przy spadku napięcia zasilania (ang. *brown-out detection*)



Kiedy pojawia się przerwanie RESET

Przy braku wyzerowania układu strażnika (ang. *watchdog*)



Co się dzieje po wyzerowaniu?

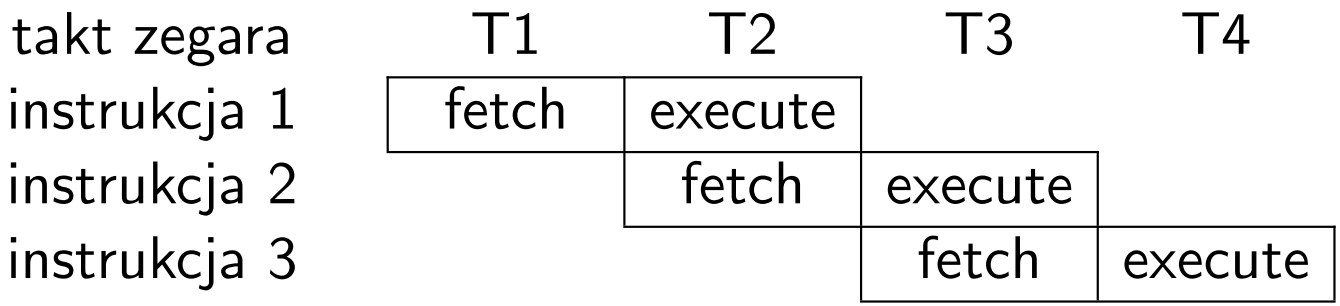
- Ustawienie rejestrów wejścia-wyjścia na wartości początkowe
- Rozpoczęcie wykonania programu od instrukcji znajdującej się pod adresem 0 (przerwanie 0)

Pierwszy program

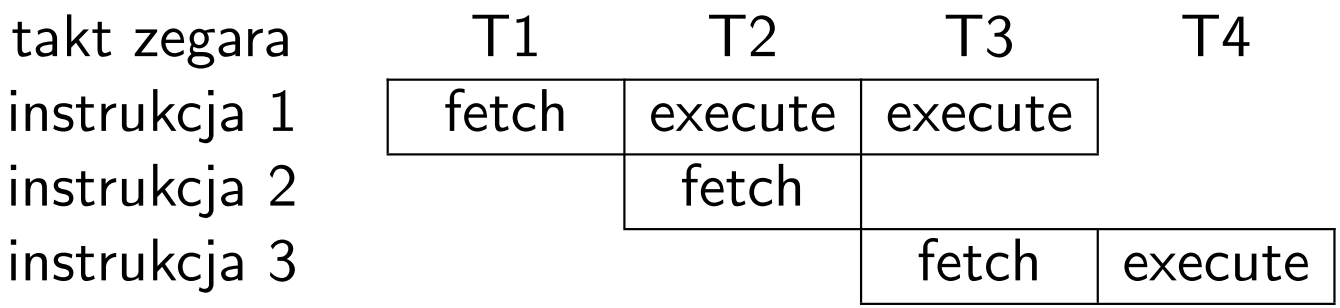
```
.CSEG                ; segment kodu
.ORG 0               ; przerwanie o adresie 0 = reset
   JMP START

.ORG 42              ; pierwszy adres za wektorem przerwań
START:
   LDI R16, 0b00001111
   OUT DDRA, R16          ; nogi PA0..PA3 jako wyjścia
   LDI R16, 0b00001011
   OUT PORTA, R16         ; nogi PA0,PA1,PA3 stan wysoki
                          ; noga PA2 stan niski
PETLA:
   RJMP PETLA             ; główna pętla programu
```

Cykl wykonania instrukcji

- Gdy nie ma skoków.

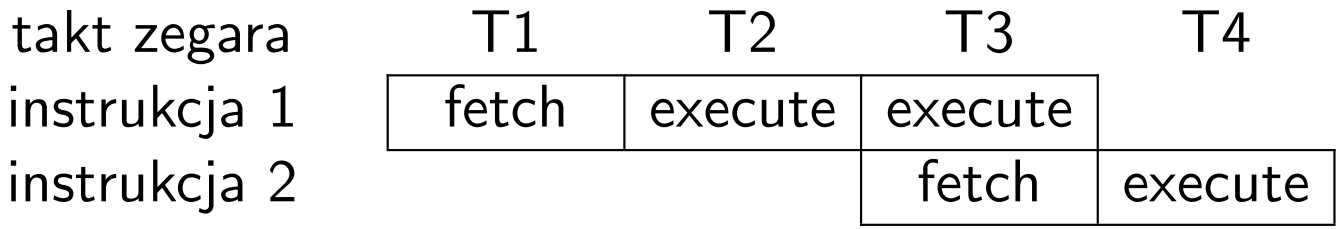


- Gdy instrukcja 1 wykona skok do instrukcji 3.



Cykl wykonania instrukcji, cd.

- Gdy instrukcja 1 potrzebuje więcej taktów do wykonania.



- Dokumentacja podaje liczbę taktów execute.
- Z wyjątkiem dwóch instrukcji (`SPM`, `BREAK`) wszystkie pozostałe potrzebują maksymalnie cztery takty execute.
- Większość instrukcji potrzebuje jeden lub dwa takty execute.
- Czas wykonywania programu jest w pełni przewidywalny – brak spekulatywnego wykonywania instrukcji.

Jednostka arytmetyczno-logiczna ALU

- Rozkazy arytmetyczne 8-bitowe wykonuje w jednym cyklu zegara.
- Rozkazy arytmetyczne 16-bitowe i rozkazy mnożenia wykonuje w dwóch cyklach zegara.
- Wspiera operacje na liczbach bez znaku, ze znakiem (notacja uzupełnieniowa do dwóch) oraz ułamkowych (stałoprzecinkowych).
- Rejestr stanu (`SREG`) jest uaktualniany po każdej operacji ALU.