# Systemy cyfrowe i podstawy elektroniki

Adam Szmigielski
aszmigie@pjwstk.edu.pl
materiały: $ftp(public) : //aszmigie/SYC$

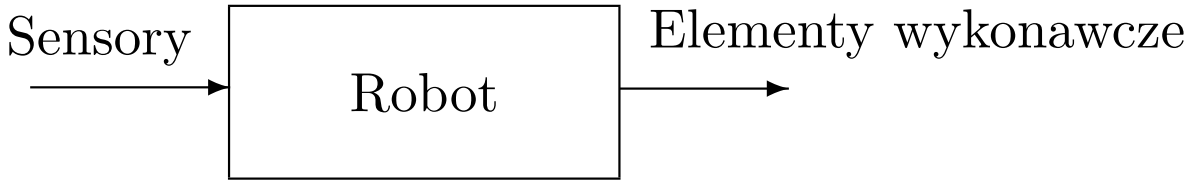
# Mikrokontrolery w systemach sterownia - wykład 13

## Obiekt sterowania


*Obiekt sterowania* – obiekt, który realizuje proces (zaplanowany).

- *Fizyczny obiekt* (proces, urządzenie) jest nieodłączną częścią problemu sterowania,
- Dla projektowania sterowania niezbędna jest wiedza o fizycznym obiekcie,
- *Wejście* - sygnał wejściowy, steruje naszym obiektem,
- *Wyjście* - sygnał wyjściowy, określa stan interesującej nas cechy obiektu.

# Sterowanie obiektem na przykładzie robota




- Sensory
  - czujniki odległości,
  - czujniki orientacji,
  - kamery,
  - inne.
- Elementy wykonawcza
  - mechanizmy jezdne,
  - chwytaki i manipulatory,
  - inne urządzenia.

## Cele sterowania

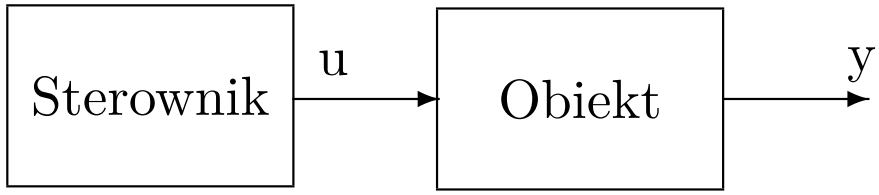
Zanim dobierzemy czujniki, elementy wykonawcze, zaprojektujemy architekturę układu regulacji musimy określić *cele* - efekty które należy osiągnąć w procesie sterowania lub po jego zakończeniu.

- *Co chcemy osiągnąć* (redukcja energii, zwiększenie zysku, ...)?
- *Jakie wielkości* należy sterować aby osiągnąć zamierzone cele?
- Jakie są wymagania (prędkość, dokładność, ...)?

## Reguła sterowania

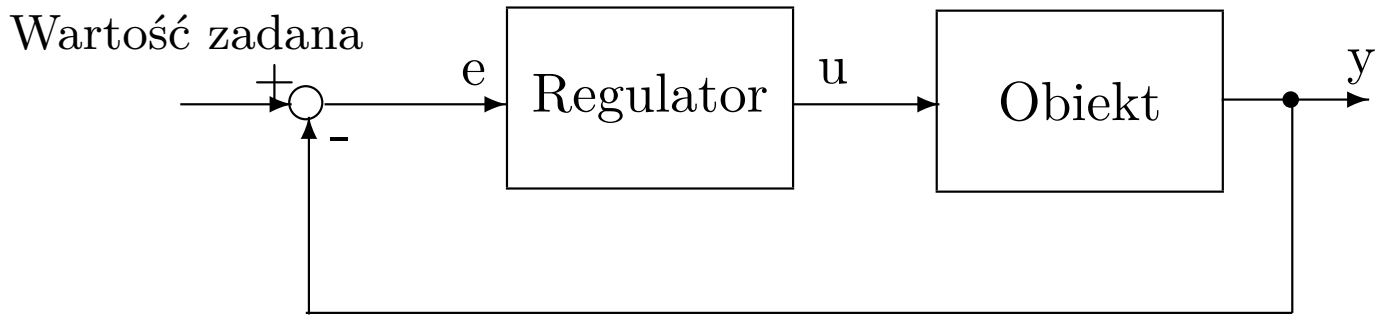
- *Zasada (algorytm)* przetwarzania informacji o stanie obiektu na sygnały sterowania elementami wykonawczymi.
- Podstawowe zasady sterowania:
  - Sterowanie w układzie otwartym,
  - Sterowanie w układzie zamkniętym ze sprzężeniem zwrotnym.

## Układ z otwartą pętlą sterowania




- Sterownik realizuje *cel storowania* poprzez realizację *algorytmu sterowania*,
- Sterownik nie posiada żadnej informacji zwrtotnej o przebiegu sterowania,
- Brak informacji *zwrotnej* powoduje to, że sterownik jest nieodporny na błędy sterownia.

# Układ z zamkniętą pętlą sprzężenia




- Układy regulacji pracują z *ujemnym sprzężeniem* zwrotnym,
- Celem układu regulacji jest osiągnięcie poprzez wyjście *y wartości zadanej*,
- To, jak daleko jest do osiągnięcia celu regulacji określa *błąd regulacji* - $e$,
- Regulator w zależności od błędu regulacji $e$ dobiera sterowanie $u$,

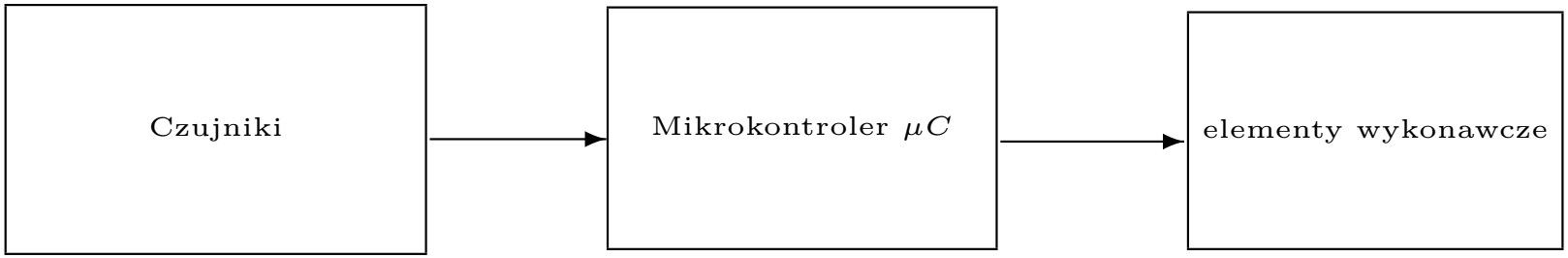
## Charakterystyka mikroprocesorowych systemów sterowania

- Wykorzystanie zaawansowanej technologii elektronicznej – zastąpienie rozwiązań analogowych i elektromechanicznych,
- Idea stabilizującego sprzężenia zwrotnego – podstawowa zasada działania układów regulacji z wykorzystaniem systemów mikroprocesorowych,
- *Sposób pracy układów* – próbkowanie stanu procesu w dyskretnych przedziałach czasu i oddziaływaniu na proces w określonych odstępach czasu, zastosowanie logiki binarnej.
- *Dokładność* – dyskretna postać sygnału odporna na szumy urządzeń pomiarowych, możliwość przesyłania na duże odległości.
- *Koszt* – rozwój technologiczny, zmniejszające się koszty

wytworzenia mikrokontrolerów.

- *Nowe algorytmy* – systemy dyskretne mogą w skończonym czasie osiągnąć wartość zadaną.
- *Elastyczność* – łatwość konfiguracji regulatorów – oprogramowanie.
- *Błędy przetwarzania* – operacje: dodawania, odejmowania, błędy pomijalne w porównaniu do układów analogowych.

# Realizacja sterowania z wykorzystaniem $\mu C$




- Programowalny $\mu C$ wykonuje zadania sterowania
- Sygnały z czujników wymagają dopasowania do $\mu C$
- Elementy wykonawcze - sygnał cyfrowy musi być przekształcony na sygnał sterujący,
- Łatwo można zmienić algorytm sterujący,
- Zazwyczaj trudno jest dodać nowe czujniki lub elementy wykonawcze

## Metody obsługi zdarzeń

- *Przerwanie* (ang. Interrupt) - zmiana sterowania, niezależnie od aktualnie wykonywanego programu, spowodowana pojawieniem się sygnału przewania. Pojawienie się przerwania powoduje wstrzymanie aktualnie wykonywanego programu i wykonanie przez kontroler procedury obsługi przerwania.
- *Zapytywanie* (ang. Polling) - aktywne, okresowe, próbkowanie (sprawdzanie) statusu urządzeń zewnętrznych przez kontroler.

# *Zapytywanie* (ang. Polling)

- Technika *polling* jest najczęściej używana w kontekście obsługi urządzeń wejścia/ wyjścia,
- W *polling-u* komputer centralny cyklicznie sprawdza stan urządzenia zewnętrznego w oczekiwaniu na gotowość tego urządzenia - czeka na gotowość,
- *Polling* znajduje zastosowanie w sytuacjach, gdy komputer łączy się z zewnętrznymi urządzeniami w celu zebrania (odświeżenia) danych, przy czym współpraca ta odbywa się w trybie *off-line*,
- *Polling* może być wykorzystany do wymiany informacji z urządzeniami zewnętrznymi, w sytuacji gdy z jakiś względów urządzenia te nie mogą rozpocząć komunikacji,
- W systemach obsługujących jedno zadanie *polling* może również mieć zastosowanie. Większość czasu procesora byłaby wówczas tracona na sprawdzanie gotowości urządzenia,
- W systemach, które wymagają wykonania **wielu zadań** *polling* **jest mało efektywny** w stosunku do przerwań.

## Rodzaje przerwań

1. *Sprzętowe:*
   - *Zewnętrzne* sygnał przerwania pochodzi z zewnętrznego źródła. Przerwania te służą do komunikacji z urządzeniami zewnętrznymi.
   - *Wewnętrzne* - pochodzące od timera
   - *Wewnętrzne wyjątki* - (ang. exceptions) – zgłaszane przez procesor dla sygnalizowania sytuacji wyjątkowych (np. dzielenie przez zero)
2. *Programowe:* z kodu programu wywoływana jest procedura obsługi przerwania (do komunikacji z systemem operacyjnym).

## Wektory przerwań

- *Wektor przerwań* jest adresem początku obsługi przerwania,
- *Wektor przerwań*, w momencie wystąpienia przerwania, wpisywany jest do *licznika rozkazów* - rejestr PC, a zawartość rejestru PC jest kładziona na *stos*,
- Adresy procedur obsługi przerwań zapisane są w *tablicy wektorów przerwań*,
- Przechowuje ona adresy poszczególnych procedur obsługi przerwań,

| Vector No. | Address | Source | Interrupt Definition |
|---|---|---|---|
| 1 | 0x000 | RESET | External Pin, Power-on, Brown-out, and Watchdog |
| 2 | 0x001 | INT0 | External Interrupt Request 0 |
| 3 | 0x002 | INT1 | External Interrupt Request 1 |
| 4 | 0x003 | PCINT0 | Pin Change Interrupt Request 0 |
| 5 | 0x004 | PCINT1 | Pin Change Interrupt Request 1 |
| 6 | 0x005 | PCINT2 | Pin Change Interrupt Request 2 |
| 7 | 0x006 | WDT | Watchdog Time-out Interrupt |
| 8 | 0x007 | TIMER2 COMPA | Timer/Counter2 Compare Match A |
| 9 | 0x008 | TIMER2 COMPB | Timer/Counter2 Compare Match B |
| 10 | 0x009 | TIMER2 OVF | Timer/Counter2 Overflow |
| 11 | 0x00A | TIMER1 CAPT | Timer/Counter1 Capture Event |
| 12 | 0x00B | TIMER1 COMPA | Timer/Counter1 Compare Match A |
| 13 | 0x00C | TIMER1 COMPB | Timer/Coutner1 Compare Match B |
| 14 | 0x00D | TIMER1 OVF | Timer/Counter1 Overflow |
| 15 | 0x00E | TIMER0 COMPA | Timer/Counter0 Compare Match A |
| 16 | 0x00F | TIMER0 COMPB | Timer/Counter0 Compare Match B |
| 17 | 0x010 | TIMER0 OVF | Timer/Counter0 Overflow |
| 18 | 0x011 | SPI, STC SPI | Serial Transfer Complete |
| 19 | 0x012 | USART, RX | USART Rx Complete |
| 20 | 0x013 | USART, UDRE | USART, Data Register Empty |
| 21 | 0x014 | USART, TX | USART, Tx Complete |
| inne | ... | ... | ... |

## Przerwania maskowalne i niemaskowalne

- *Przerwania maskowalne* które można blokować i odblokować programowo,
- *Przerwania niemaskowalne* - przerwania, których nie można zablokowć programowo. Są to przerwania, których wystąpienie każdorazowo powoduje bezwarunkowy skok do funkcji obsługi tego przerwania, np. reset

## Obsługa przerwania

- Procedura obsługi przerwania - ciąg rozkazów realizujących pożądaną reakcję na przerwanie,
- *Program główny* - sekwencja działań (rozkazów) mikroprocesora realizowanych gdy nie ma przerwań,
- Obsługa przerwania nie może wprowadzać żadnych zmian w programie głównym.

## Procedura obsługi przerwania

1. Rozpoznanie przyczyny przerwania (realizacja może być sprzętowa),
2. Skasowanie przyczyny przerwania (realizacja może być sprzętowa),
3. Zablokowanie przerwania,
4. Składowanie na stosie rejestrów roboczych,
5. Właściwa obsługa przerwania,
6. Odtworzenie rejestrów roboczych ze stosu,
7. Odblokowanie przerwania,
8. Powrót do zawieszonego programu.

## Stos

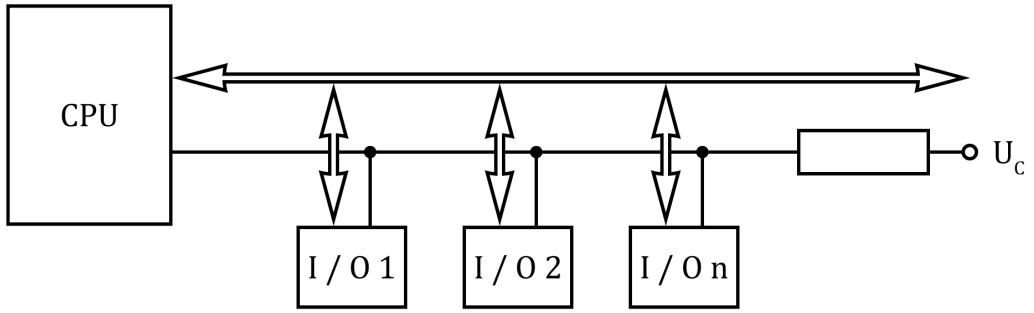
- W momencie wywołania przerwania adres odkładany jest na stos,
- Wskaźnik stosu powinien być ustawiony na miejsce gdzie znajduje się stos.

## Priorytet przerwań

- *Priorytet przerwań* - zróżnicowanie co do ważności (pilności) zadań realizowanych przez system mikroprocesorowy,
- W szczególności zadaniami tymi mogą być procedury obsługi przerwań różnicując ich pilność dokonuje się określenia priorytetów poszczególnych przerwań,
- W przypadku AVR system obsługi przerwań jest płaski (brak hierarchii). Wszystkie przerwania są jednakowo ważne.

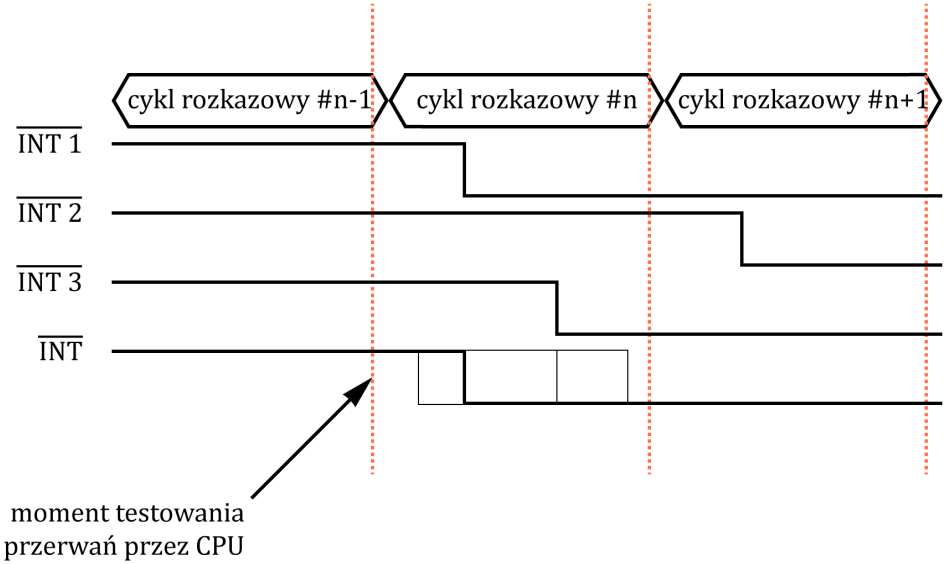
# Realizacja systemu przerwań

- *sprzętowo* - o wyborze przerwania decyduje *kontroler przerwań.*
- *programowo* - poprzez wspólną procedurę obsługi przerwań. Jest on arbitrem systemu przerwań (rozpoznaje źródła aktualnych przerwań i decyduje o kolejności ich obsługi)



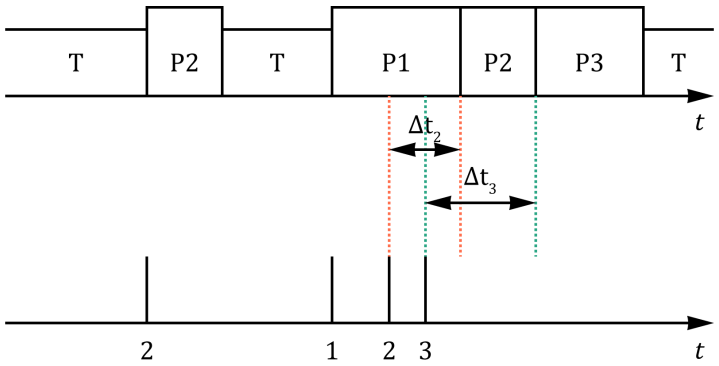
# Asynchroniczność przerwań

- Przerwania z różnych źródeł pojawiają się w dowolnych, niezależnych od siebie, chwilach czasu,
- z punktu widzenie procesora przerwania 1 i 3 wystąpiły jednocześnie.



# System przerwań bez priorytetów

- Opóźnienia $\{\Delta t_2, \Delta t_3\}$ w reakcji na obsługę przerwań,
- Możliwość zgubienia przerwania podczas tych opóźnień,
- Maksymalny czas oczekiwania na obsługę przerwania może być równy sumie czasów obsługi pozostałych przerwań w systemie,



# System przerwań z priorytetami

- Istnieje hierarchia ważności przerwań,
- Opóźnienia $\{\Delta t_1, \Delta t_2\}$ w reakcji na obsługę przerwań,
- Przerwania o niższych priorytetach mogą dłużej czekać na obsługę (w skrajnych przypadkach mogą być nie obsłużone),

## Rodzaje przerwań

- Przerwania zegarowe - odmierzanie czasu,
- Przerwania od urządzeń zewnętrznych - nieregularne,
- Przerwania od układów kontrolujących pracę systemu - o najwyższym priorytecie. Sygnalizują stan pracy jak
  - zanik zasilania,
  - błąd/wyjątek procesora,
  - inne

## Inne przerwania

1. SPI, STC - Serial Transfer Complete,
2. USART, RXC USART - Rx Complete,
3. USART, TXC USART, Tx Complete,
4. USART, UDRE USART Data Register Empty,
5. ADC ADC Conversion Complete,
6. - inne, opis w dokumentacji.

# Struktura programu obsługi przerwań Arduino - tylko przerwania zewnętrzne

```
const byte interruptPin = 2;
volatile byte state = LOW;

void setup() {
  pinMode(ledPin, OUTPUT);
  pinMode(interruptPin, INPUT_PULLUP);
  attachInterrupt(digitalPinToInterrupt(interruptPin), blink, CHANGE);
}

void loop() {
  digitalWrite(ledPin, state);
}

void blink() {
  state = !state;
}
```

## Zadania na ćwiczenia

1. Wykorzystujšc przerwanie *int0* napisz program, który wyświetla liczbę wciśnięć przycisku na wyświetlaczu LCD,
2. Napisz program realizujący poprzednie zadanie z wykorzystaniem mechanizmu zpytań,
3. Zrealizuj generator o częstotliwości $f = 1Hz$ i współczynniku wypełnienia $\omega = \frac{1}{2}$. Po naciśnięciu przycisku generowana częstotliwość powinna zwiększać się o 10%, a informacja o wartości częstotliwości powinna zostać wysłana na łącze szeregowe. Odebranie z łącza szeregowego informacji przywraca wyjściową częstotliwość. Generowany sygnał podawany jest na pin 13 (z diodą).