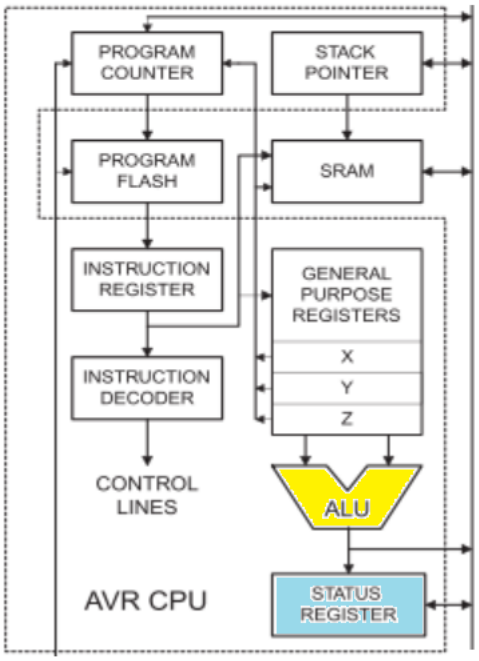
1. Zagadnienia

- Jednostka arytmetyczno-logiczna (ALU)
- Rejestr znaczników SREG
- Instrukcje warunkowe
- Algorytm programu

2. ALU (jednostka-arytmetyczno logiczna)

Jednostka arytmetyczno-logiczna ALU (*Arthmetic Logic Unit*) jest najważniejszym podzespołem mikrokontrolera, w którym wykonywane są wszystkie operacje arytmetyczne i logiczne. ATmega16 jest mikrokontrolerem 8-bitowym - to znaczy, że jednostka ALU wykonuje obliczenia na słowie 8-bitowym.



3. SREG (rejestr znaczników, rejestr flag)

W programowaniu procesorów bardzo często wykorzystujemy pojęcie rejestru. Rejestr to komórka pamięci (układu cyfrowego z przerzutnikami) przeznaczona do zapamiętania informacji). W przypadku omawianego mikrokontrolera ATmega16 najczęściej mówimy o rejestrach 8 i 16 - bitowych. Ze względu na zastosowania możemy wyróżnić:

- rejestry danych – do przechowywania danych (argumentów, wyników obliczeń)
- rejestry adresowe – do przechowywania adresów
- rejestry ogólnego zastosowania (ang. *general purpose*) - będące połączeniem dwóch powyższych typów, czyli mogące przechowywać zarówno dane, jak i adresy,
- rejestry specjalne - określające stan wykonania, wśród nich wymienić można rejestr PC (wskaźnik instrukcji), SP (wskaźnik stosu), rejestr flag (SREG),

Rejestr SREG (*Status Register* - rejestr znaczników) jest specjalnym rejestrem do którego wpisywane są informacje o przebiegu **ostatnio wykonanej** operacji w ALU (np. było przeniesienie, nadmiar, wynik zerowy itp.). Rejestr SREG składa się z 8 bitów:

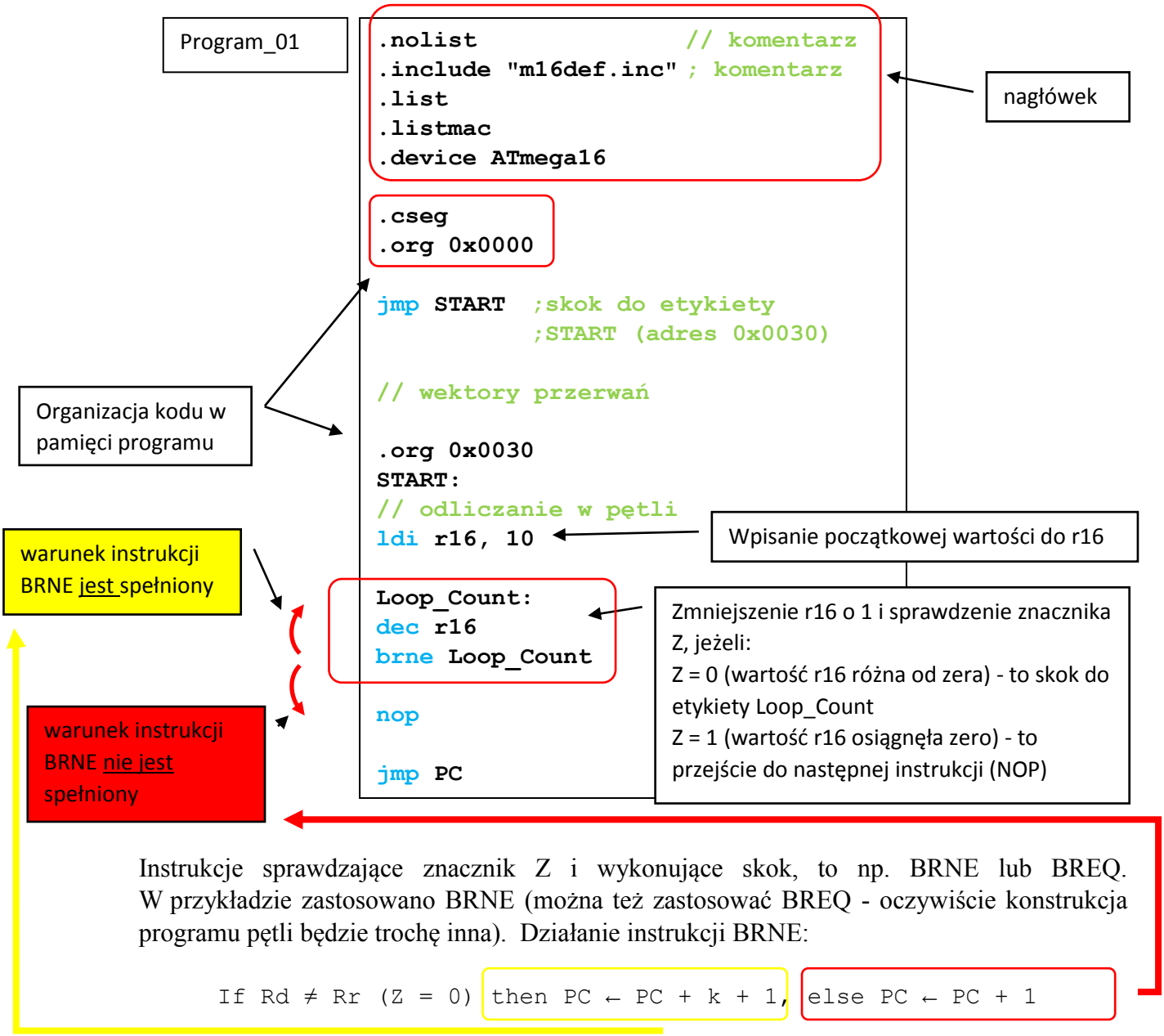
7	6	5	4	3	2	1	0
I	T	H	S	V	N	Z	C

Znaczniki C (*carry*) i Z (*zero*) były wykorzystywane już w poprzednich ćwiczeniach (inne znaczniki będą omawiane i wykorzystywane w kolejnych programach).

4. Instrukcje warunkowe, instrukcje skoku warunkowego

Informacje o stanie procesora zawarte w rejestrze SREG, są wykorzystywane w instrukcjach skoków warunkowych do tworzenia pętli programowych (warunek pozostania w pętli lub wyjścia z pętli). Mikrokontrolery AVR posiadają kilkanaście różnorodnych instrukcji skoków warunkowych (zobacz: ***Help -> Assembler Help -> Branch Instructions***)

- Utwórz nowy projekt (***Project -> Project Wizard***).
- Wpisz kod programu: Odliczanie do zera



Instrukcje sprawdzające znacznik Z i wykonujące skok, to np. BRNE lub BREQ. W przykładzie zastosowano BRNE (można też zastosować BREQ - oczywiście konstrukcja programu pętli będzie trochę inna). Działanie instrukcji BRNE:

```
If Rd ≠ Rr (Z = 0) then PC ← PC + k + 1, else PC ← PC + 1
```

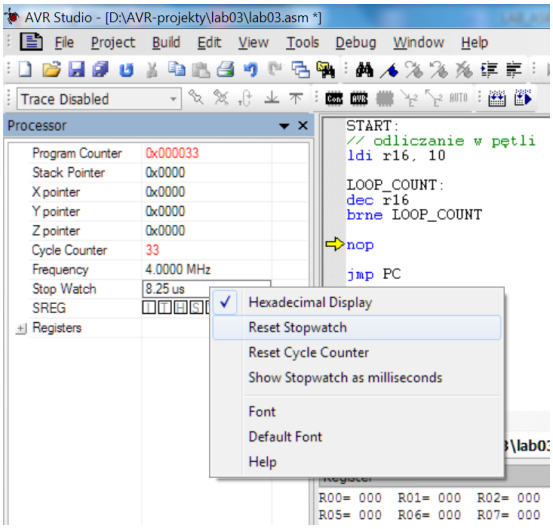
Jeżeli po wykonaniu instrukcji dekrementacji w liczniku r16 jest liczba różna od zera, to znacznik Z nie zostanie ustawiony (Z=0), jeżeli r16 osiągnął zero to znacznik Z zostanie ustawiony na 1 (Z=1).

Działanie instrukcja BREQ:

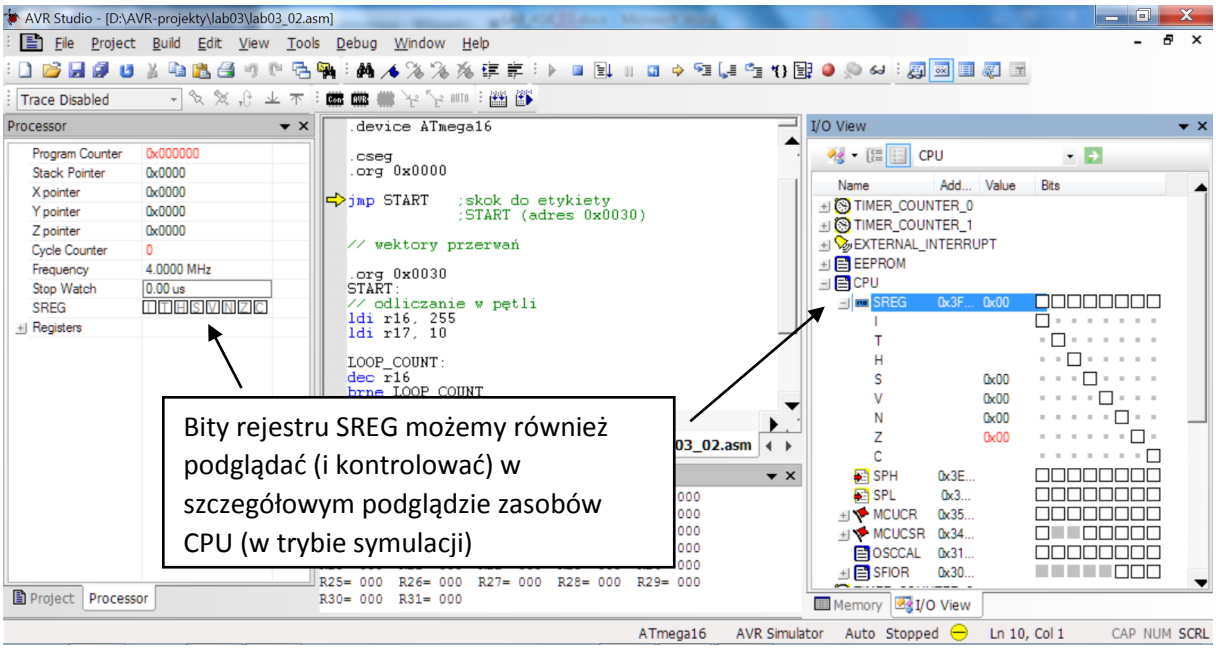
```
If Rd = Rr (Z = 1) then PC <- PC + k + 1, else PC <- PC + 1

-64 ≤ k ≤ +63
```

Liczba k (liczba dodawana do aktualnej wartości PC - zakres skoku) może przyjmować zarówno wartości dodatnie, jak i ujemne, zatem w naszym programie możemy wykonać skok do "przodu" i do "tyłu" (niestety dla instrukcji BREQ i BRNE zakres skoku jest ograniczony +/- 64 komórki, nie możemy zatem wykonać skoku w dowolne miejsce obszaru pamięci).

- Sprawdź działanie programu w pracy krokowej, "zmierz" czas opóźnienia wprowadzanego przez pętlę (dla domyślnie ustawionej częstotliwości taktowania procesora 4MHz).
- Sprawdź jakie maksymalne opóźnienie można uzyskać w pętli (wpisz do r16 wartość 255).

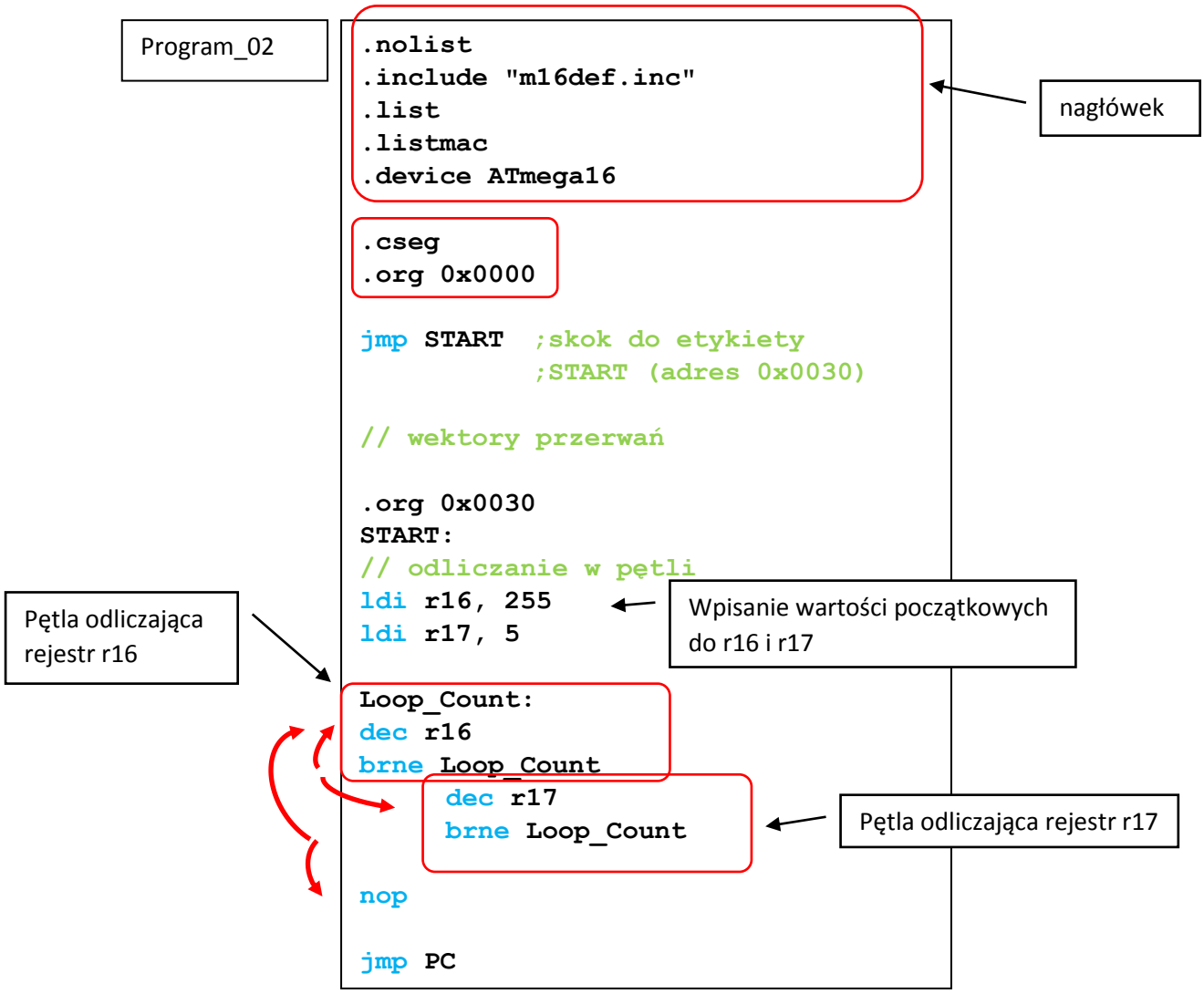


Podczas symulacji, w dowolnym momencie możemy "wyzerować" licznik wskazujący liczbę cykli zegara (Cycle Counter) oraz odmierzony czas (Stop Watch) (***Prawy-Przycisk-Myszy***). Wyzeruj wskazania na początku pętli **Loop_Count** i odczytaj po wyjściu z pętli.



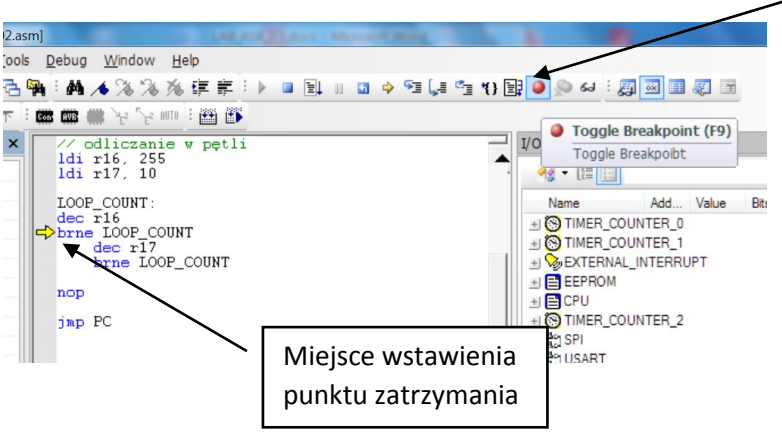
5. Pętla w pętli

Wykorzystując do odliczania rejestr 8-bitowy oraz pojedynczą pętlę, możemy generować opóźnienia rzędu mikrosekund, aby wydłużyć opóźnienia możemy zastosować pętle w pętli.



W przykładzie po odliczeniu do zera rejestru r16, program przechodzi do instrukcji DEC r17, zmniejsza zawartość rejestru r17, sprawdza warunek i jeżeli nie doliczył do zera to wraca do Loop_Count, czyli zaczyna odliczać r16. Mamy zatem zagnieżdżenie pętli. Całkowity czas opóźnienia będzie zatem wielokrotnością (liczba w r17) i czasu wykonania podstawowe pętli odliczającej r16.

- Sprawdź działanie programu, "zmierz" czas opóźnienia wprowadzanego przez pętlę dla wartości r16 =255, r17=10).
- Sprawdź jakie maksymalne opóźnienie można uzyskać w pętli (wpisz do r16 i do r17 wartość 255).



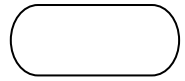
Analizując działanie pętli w pracy krokowej warto zastosować punkty zatrzymania (Breakpoint).

Breakpoint zostanie wstawiony w aktualnie wskazywanym w programie w miejscu.

Po wstawieniu punktów zatrzymania możemy uruchomić symulację w trybie pracy ciągłej używając poleceń: ***Debug -> Auto-Step (Alt+F5)*** lub ***Debug -> Run (F5).*** Przerwanie pracy ciągłej: ***Debug -> Break (Ctrl+F5)*** lub ***Debug -> Reset (Shift+F5).***

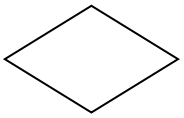
6. Algorytm programu

Działanie programu (algorytm) bardzo często przedstawiamy w postaci graficznej. Podstawowe symbole to:

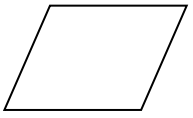


początek (lub koniec) algorytmu
w każdym algorytmie musi się znaleźć dokładnie jedna taka figura z napisem "Start" oznaczająca początek algorytmu oraz dokładnie jedna figura z napisem "Stop" oznaczająca koniec algorytmu.

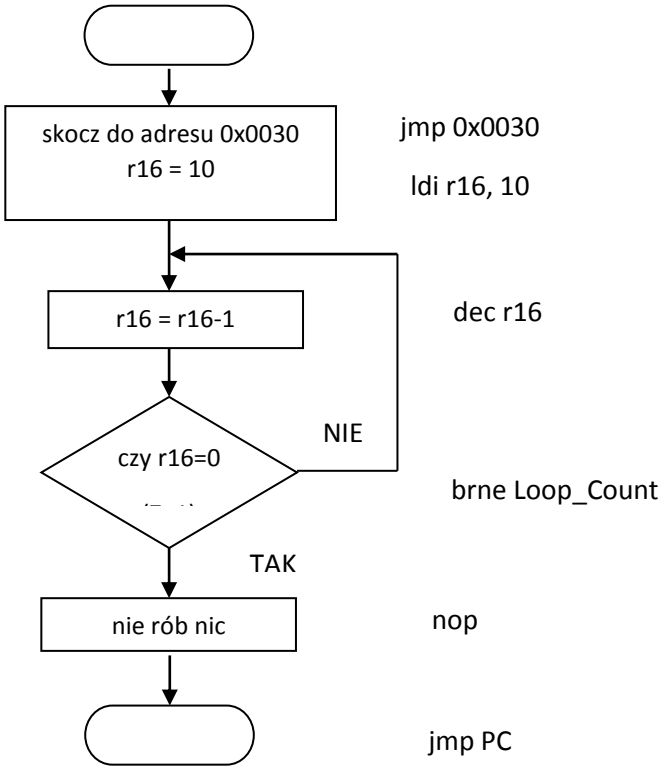
process
w jej obrębie umieszczamy wszelkie obliczenia lub podstawienia. Proces ma dokładnie jedną strzałkę wchodzącą i dokładnie jedną strzałkę wychodzącą



blok decyzyjny
umieszcza się w nim jakiś warunek



odczytu lub zapisu danych



Algorytm programu Program_01

7. Zadania do samodzielnej realizacji

1. Analiza instrukcji BRBC i BRBS

BRBC	s,k	Branch if Status flag cleared	if(SREG(s)==0) PC = PC + k + 1
BRBS	s,k	Branch if Status flag set	if(SREG(s)==1) PC = PC + k + 1

- Zapoznaj się z działaniem instrukcji BRBC i BRBS (***Help > Assembler Help***).
- Zaproponuj modyfikację programu Program_01 zastępując instrukcję BRNE instrukcją BRBC lub BRBS.

2. Programy
- **Zadanie 1**: W przykładzie Program_02 dodaj kolejny rejestr (r18) i kolejną zagnieżdżoną pętlę. "Zmierz" czas opóźnienia dla r16, r17, r18 = 255 (Uwaga! - symulacja może potrwać nawet kilka minut, w zależności od szybkości twojego komputera).
- **Zadanie 2**: Napisz program w którym będą umieszczone pętle odmierzające opóźnienie odpowiednio: 100 us, 1ms, 1s.
- **Zadanie 3**: Napisz program, który wykona odejmowanie liczb 8-bitowych, jeżeli wynik odejmowania będzie ujemny, zamień odjemnik z odjemną i wykonaj jeszcze raz odejmowanie.
- **Zadanie 4:** Narysuj algorytmy do w/w programów.
- **Zadanie 5**: Zadanie podane przez prowadzącego.

3. Zadania dodatkowe (dla chętnych)
- **Zadanie A**: Zapoznaj się z działaniem instrukcji CPSE (***Help > Assembler Help***).

CPSE	Rd,Rr	Compare, Skip if equal	if (Rd ==Rr) PC = PC 2 or 3

  Zaproponuj modyfikację programu Program_01 zastępując instrukcję BRNE instrukcją CPSE.
- **Zadanie B**: Napisz program zliczający w rejestrze r20 wartości od 0 do 100 (zliczanie "w górę" w pętli, po osiągnięciu 100 - zaczyna od 0).
- **Zadanie C**: Napisz program dzielenia danej liczby (całkowitej, 8-bitowej) przez 10. Dzielenie wykonaj na zasadzie wielokrotnego odejmowania, efekt końcowy to uzyskanie wyniku dzielenia i reszty (operujemy na liczbach całkowitych).
- **Zadanie D**: Zadanie podane przez prowadzącego.

8. Sprawozdanie

- W sprawozdaniu należy umieścić algorytmy oraz kody wykonanych programów z odpowiednim wyjaśnieniem działania zastosowanych dyrektyw i instrukcji.
- Na podstawie noty katalogowej ATmega16 opisać znaczenie bitów rejestru SREG. Pod jakim adresem i w której pamięci umieszczony jest ten rejestr? (strona 9 oraz 331).