

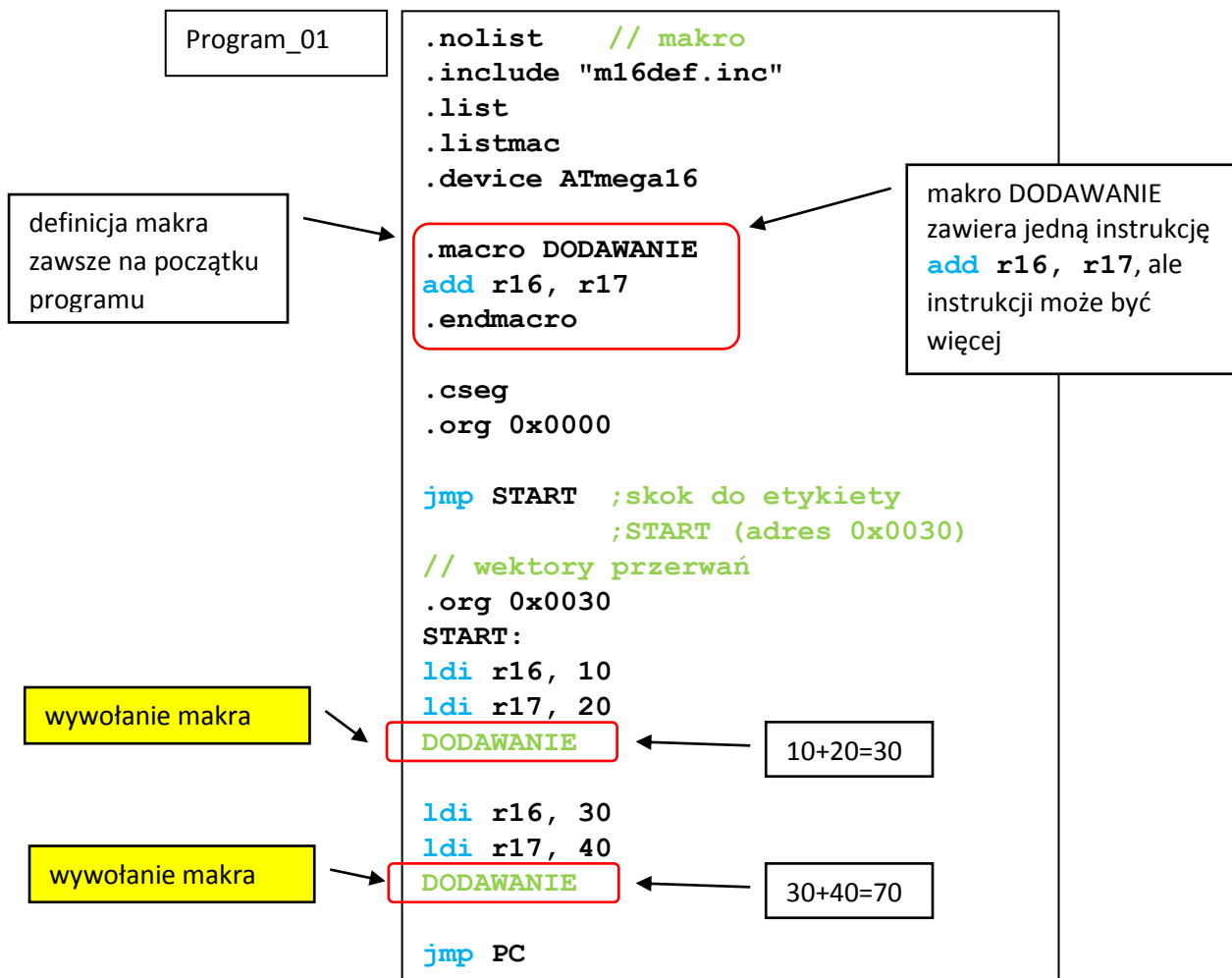
1. Zagadnienia

- Makra (bez argumentów i z argumentami)
- Podprogramy, instrukcje CALL i RET
- Stos i jego działanie
- Wskaźnik stosu

2. Tworzenie makra i wywołanie w programie

Makro - to struktura (pre)definiująca fragment programu, który może zawierać zmieniające się parametry. Wywołanie makra (jego nazwy i parametrów) w programie głównym, powoduje że translator uwzględni treść makra w miejscu jego wywołania. Makra w assemblerze definiuje się na początku programu.

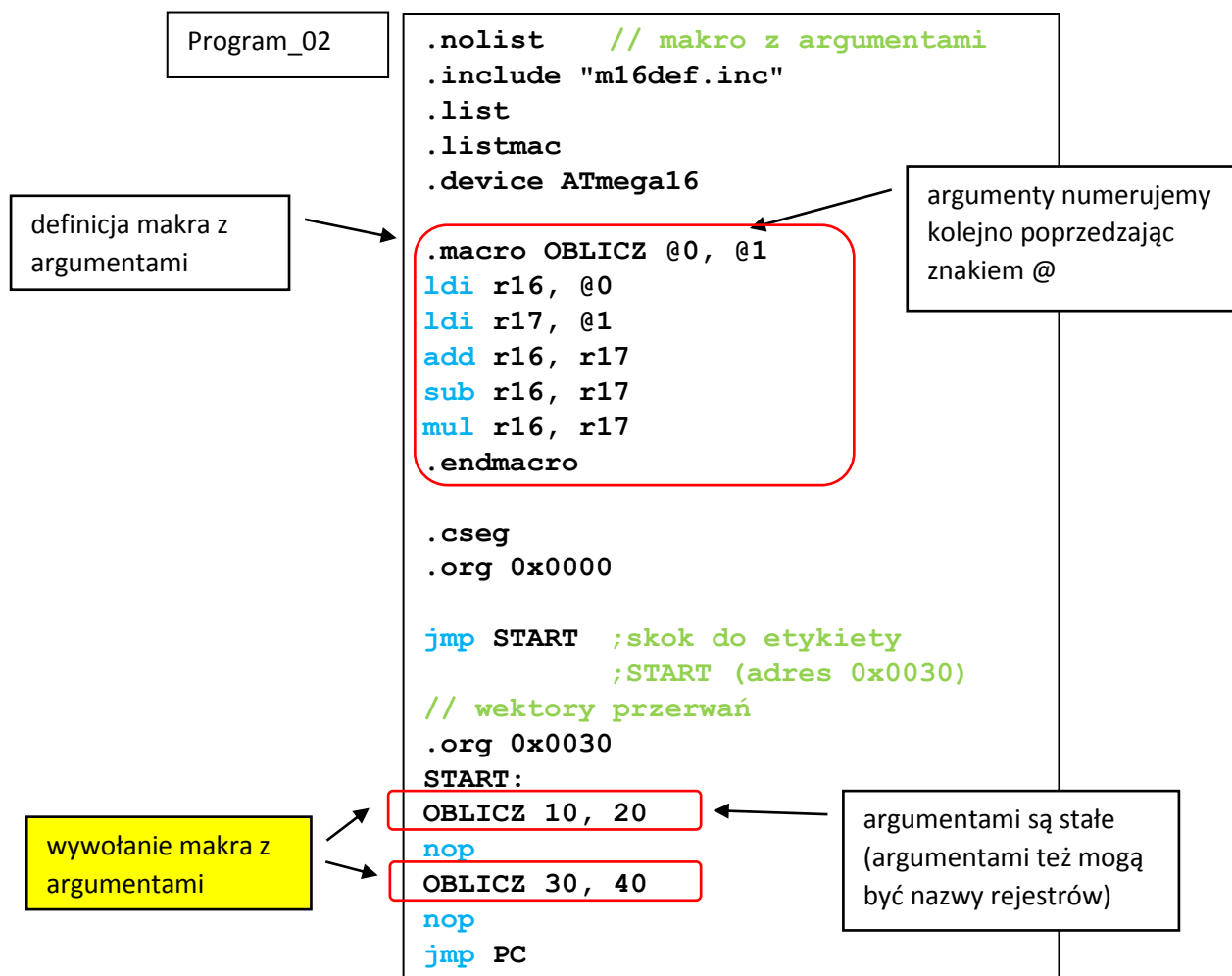
Treść makra umieszczamy pomiędzy dyrektywami .MACRO i .ENDMACRO (lub .ENDM)



Na początku zdefiniowano makro DODAWANIE, w programie głównym makro "wywołano" dwukrotnie, za każdym razem zmieniając wartości składników (w rejestrach r16 i r17). Ten typ makra nazywany jest makrem bez argumentów.

3. Makro z argumentami

Treść makra umieszczamy pomiędzy dyrektywami `.MACRO` i `.ENDMACRO` (lub `.ENDM`), po nazwie makra podajemy listę argumentów.



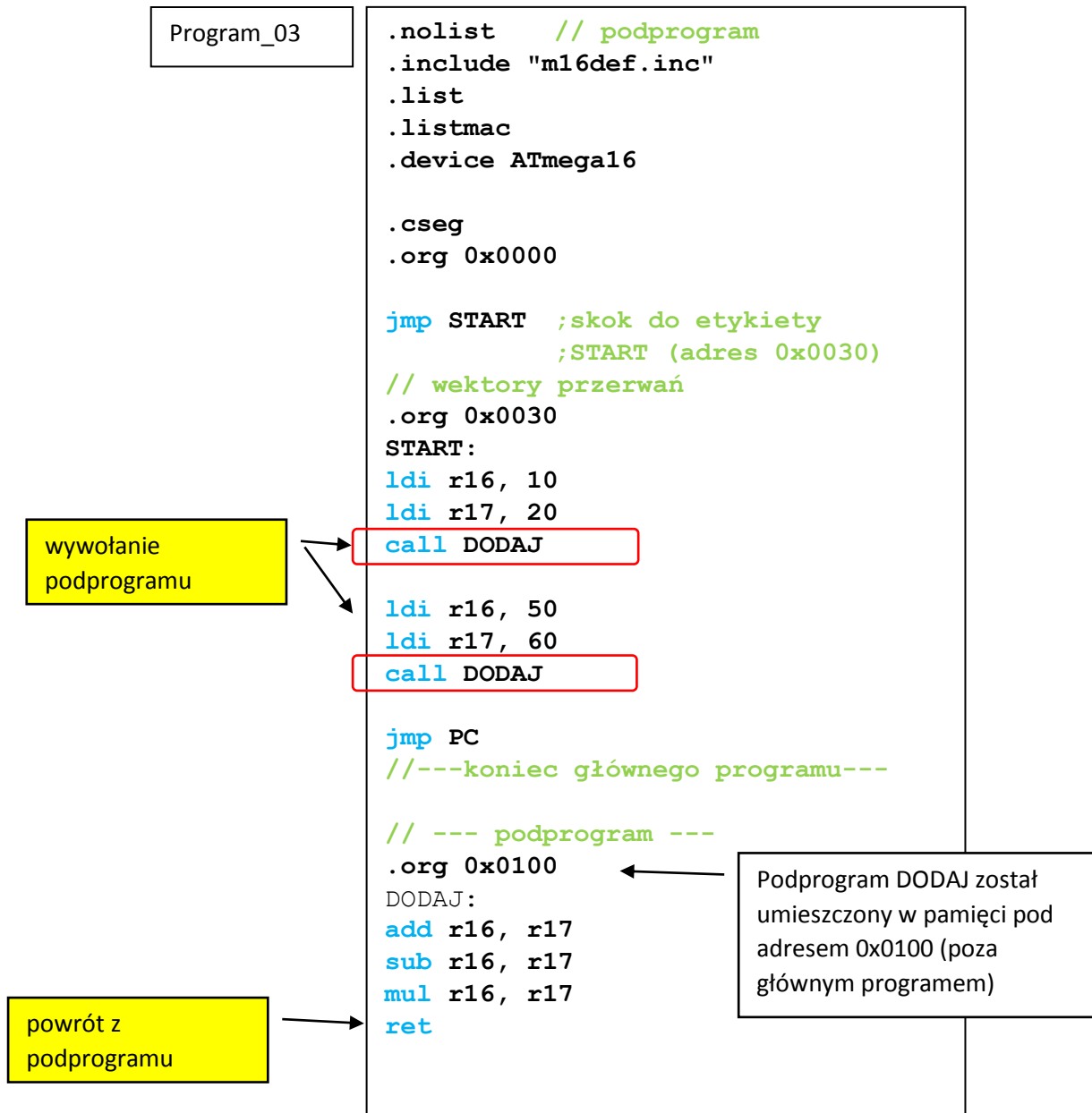
Aby wywołać zdefiniowane wcześniej makro, podajemy jego nazwę i parametry (oddzielone przecinkami).

Uruchom programy w pracy krokowej. Podczas symulacji włącz podgląd deasemblacji (**View > Disassembler**). Obserwuj jaki jest efekt wstawianie makra do kodu programu.

4. Podprogramy

Podprogram - jest wydzieloną częścią programu, do której możemy się odwoływać wielokrotnie. W odróżnieniu od makra, treść podprogramu w kodzie programu umieszczona jest raz (treść makra umieszczana jest w programie za każdym razem, gdy odwołujemy się do makra).

Wywołanie podprogramu CALL, powrót z podprogramu RET



Instrukcja CALL "wywołuje" podprogram DODAJ, instrukcja RET wraca do programu głównego. Uruchom program w pracy krokowej i zastanów się dlaczego program nie działa prawidłowo?

5. Stos i jego znaczenie, instrukcje CALL i RET

Napisany program nie będzie działał prawidłowo, ponieważ w naszym programie nie zdefiniowano stosu. Stos jest wydzieloną pamięcią danych w której zapisywane są informacje o adresach powrotu przy wykonywaniu skoków do podprogramu.

Jak działa instrukcja CALL i RET ?

CALL	k	Call Subroutine	STACK = PC+2, PC = k
----------------------	-------------------	-----------------	----------------------

Wywołanie instrukcji CALL powoduje podstawienie pod PC (Program Counter) wartości przypisanej do nazwy wywoływanego podprogramu. W naszym przypadku jest to nazwa DODAJ pod adresem 0x0100, czyli do PC zostanie wpisana wartość 0x0100. Działanie programu zostanie "przekierowane" do adresu 0x0100, czyli do naszego podprogramu. Jednocześnie na stos zostanie zapisany adres powrotu (a dokładnie adres kolejnej instrukcji w programie głównym po CALL). Wskaźnik stosu SP (Stack Pointer) zostaje zmniejszony i wskazuje kolejną komórkę w pamięci danych (procesor jest gotowy do zapisania kolejnej wartości na stos, np. w przypadku wykonanie kolejnego skoku do podprogramu).

RET	None	Subroutine Return	PC = STACK
---------------------	------	-------------------	------------

Ostatnią instrukcją podprogramu jest RET. Do PC zapisywana jest wartość zapisana wcześniej (odłożona) na stosie (PC <- STACK), czyli wracamy do programu głównego (a ściślej do następnej instrukcji po wywołaniu CALL), następnie zwiększany jest wskaźnik stosu (zdejmujemy wartość ze stosu).

Definicja stosu w programie (fragment):

Program_04

```
// ~ poprzednia część programu

.cseg
.org 0x0000

jmp START ;skok do etykiety
           ;START (adres 0x0030)
// wektory przerwań
.org 0x0030
START:
ldi r16, 0x5F
out SPL, r16
ldi r17, 0x04
out SPH, r17

// ~ dalsza część programu
```

Możemy również napisać:

```
ldi r16, LOW(RAMEND)
out SPL, r16
ldi r17, HIGH(RAMEND)
out SPH, r17
```

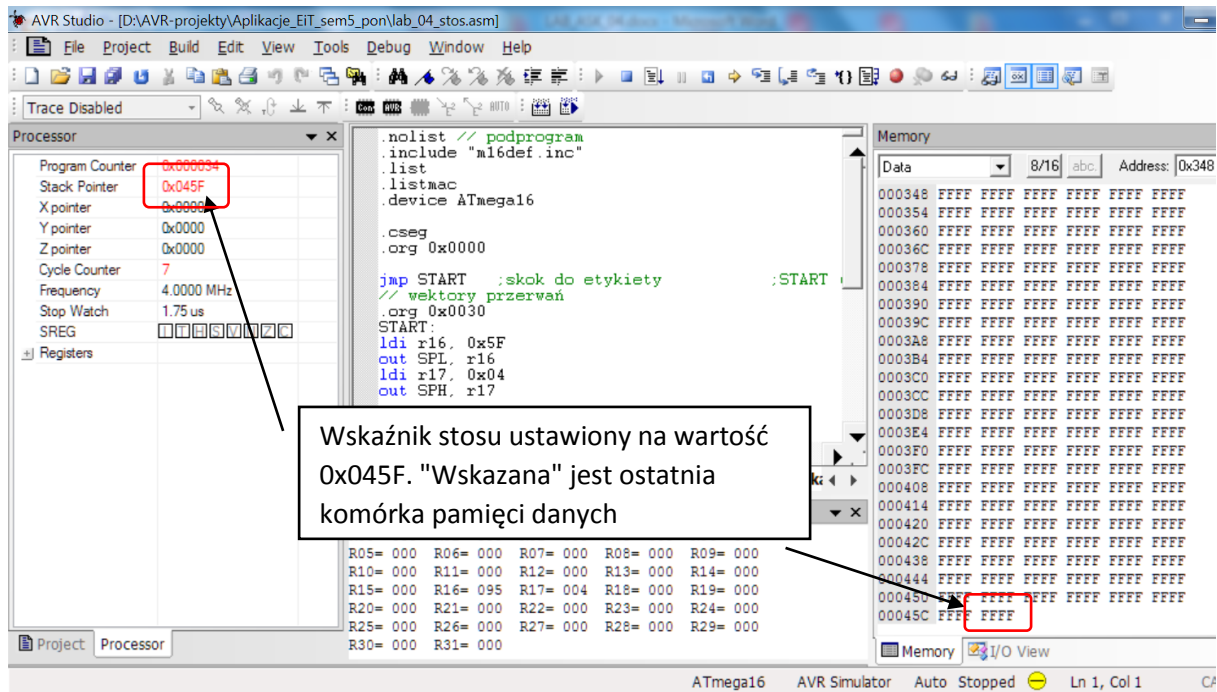
definicja stosu

→

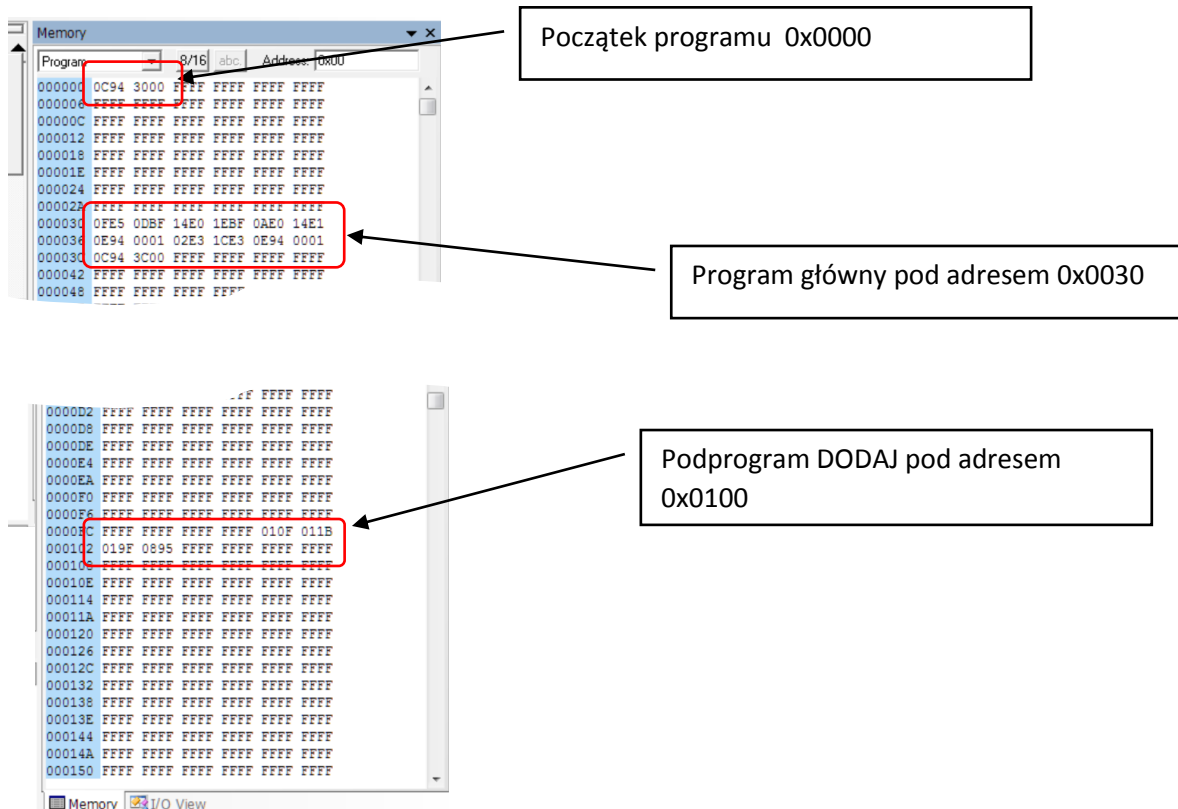
```
ldi r16, 0x5F
out SPL, r16
ldi r17, 0x04
out SPH, r17
```

Stos jest najważniejszą strukturą w procesorze lokalizowaną przez programistę. Umieszczamy go (zazwyczaj) na końcu pamięci danych i budujemy ...w górę.

W ATmega16 ostatnia komórka pamięci danych to 0x045F lub RAMEND. Instrukcja LDI r16, LOW(RAMEND) wydziela młodszą część adresu 0x045F (czyli 5F) i umieszcza w rejestrze r16, instrukcja OUT SPL, r16 zapisuje wartość r16 do młodszej części wskaźnika stosu SP (SPL). Instrukcja LDI r17, HIGH(RAMEND) wydziela starszą część adresu 0x045F (czyli 04) i umieszcza w rejestrze r17, instrukcja OUT SPH, r17 zapisuje wartość r17 do starszej części wskaźnika stosu SP (SPH).



Wskaźnik stosu ustawiony na wartość 0x045F. "Wskazana" jest ostatnia komórka pamięci danych



Początek programu 0x0000

Program główny pod adresem 0x0030

Podprogram DODAJ pod adresem 0x0100

Rozmieszczenie instrukcji programu i podprogramu można przedstawić obrazując mapę pamięci programu:

.org 0x0000	0x0000	jmp 0x0030
	0x0001	
	0x0002	
	0x0003	
.org 0x0030	0x0030	ldi r16, LOW(RAMEND)
	0x0031	out SPL,r16
	0x0032	ldi r17, HIGH(RAMEND)
	0x0033	out SPH,r17
	0x0034	ldi r16, 10
	0x0035	ldi r17, 20
	0x0036	call DODAJ
	0x0037	
	0x0038	ldi r16, 10
	0x0039	ldi r17, 20
	0x003A	call DODAJ
	0x003B	
	0x003C	jmp PC
	0x003D	
.org 0x0100	0x0100	add r16, r17
	0x0101	sub r16, r17
	0x0102	mul r16, r17
	0x0103	ret

Podprogram możemy wywoływać wielokrotnie

Uruchom program w pracy krokowej. Obserwuj, jak działa odkładanie adresu na stosie, obserwuj wskaźnik stosu SP i zawartość stosu (ostatnie komórki pamięci danych).

6. Zadania do samodzielnej realizacji

1. Analiza instrukcji PUSH i POP

PUSH	Rr	Push register on Stack	STACK = Rr
POP	Rd	Pop register from Stack	Rd = STACK

- Zapoznaj się z działaniem instrukcji PUSH i POP (**Help > Assembler Help**).
- Napisz program zapisujący na stosie liczby od 1 do 10, a następnie zdejmujący te liczby ze stosu.

2. Programy

- **Zadanie 1:** Napisz program zawierający makro FUNKCJA z argumentem, obliczające wartość funkcji x^2+2x+1 (zakładamy, że nie przekraczamy zakresu liczb 8-bitowych).
- **Zadanie 2:** Napisz program zawierający makro z argumentem wykonujące dzielenie przez 10 i obliczanie reszty (argumentem jest dzielona liczba).
- **Zadanie 3:** Napisz program zawierający podprogramy odmierzenia czasu 100 μ s, 1ms, 1s (wykorzystaj kody programów utworzonych w ćwiczeniu 3, uwzględnij czas trwania instrukcji CALL i RET).
- **Zadanie 4:** Narysuj algorytmy do w/w programów.
- **Zadanie 5:** Zadanie podane przez prowadzącego.

3. Zadania dodatkowe (dla chętnych)

- **Zadanie A:** Napisz program zawierający makro z argumentami, obliczające wartość wielomianu x^2+ax+b (argumentami są współczynniki a,b oraz x), zakładamy, że argumenty są liczbami 8-bitowymi, a wynik może być liczbą 16- lub 32-bitową).
- **Zadanie B:** Napisz podprogram dzielenia liczby 16-bitowej przez liczbę 8-bitową (wskazówka: dzielenie wykonaj na zasadzie wielokrotnego odejmowania, efekt końcowy to uzyskanie wyniku dzielenia i reszty).
- **Zadanie C:** Zadanie podane przez prowadzącego.

7. Sprawozdanie

- W sprawozdaniu należy umieścić algorytmy oraz kody wykonanych programów z odpowiednim wyjaśnieniem działania zastosowanych dyrektyw i instrukcji.
- Na podstawie noty katalogowej ATmega16 opisać przeznaczenie rejestru Stack Pointer. Pod jakim adresem i w której pamięci umieszczony jest ten rejestr? (strona 12 oraz 331).