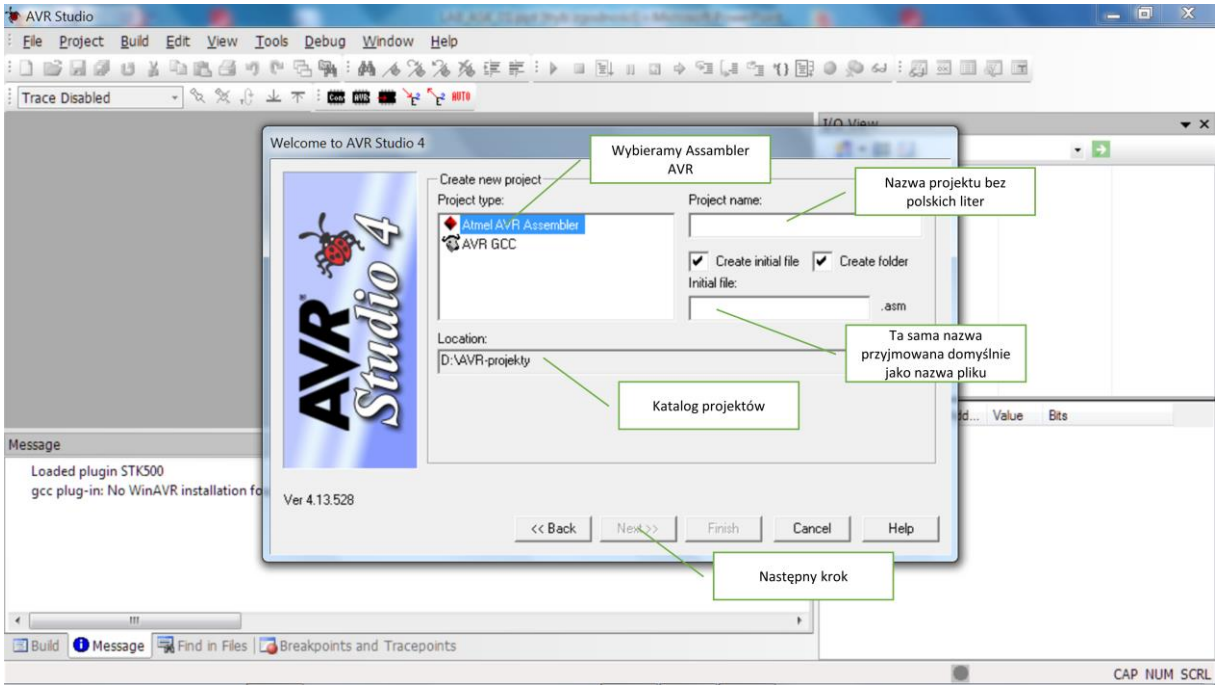
1. Zagadnienia

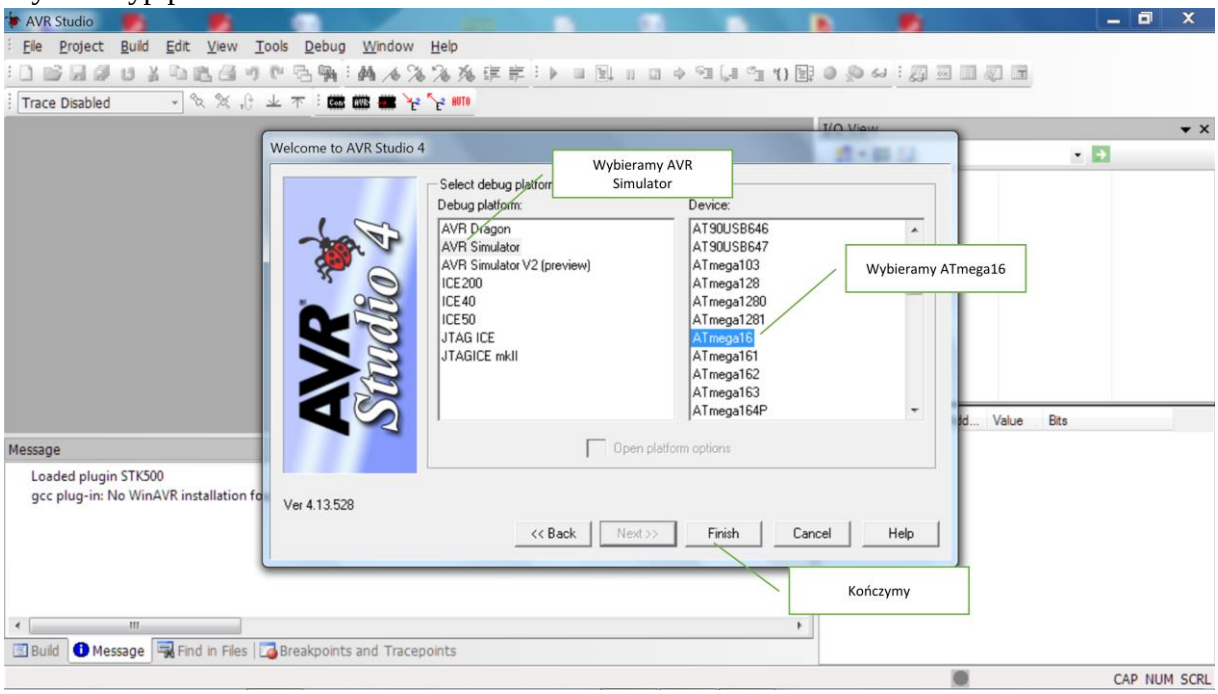
- Wprowadzenie do środowiska AVR Studio 4
- Tworzenie projektu, wprowadzanie dyrektyw
- Symulacja działania programu (Debugging)

2. Tworzenie nowego projektu

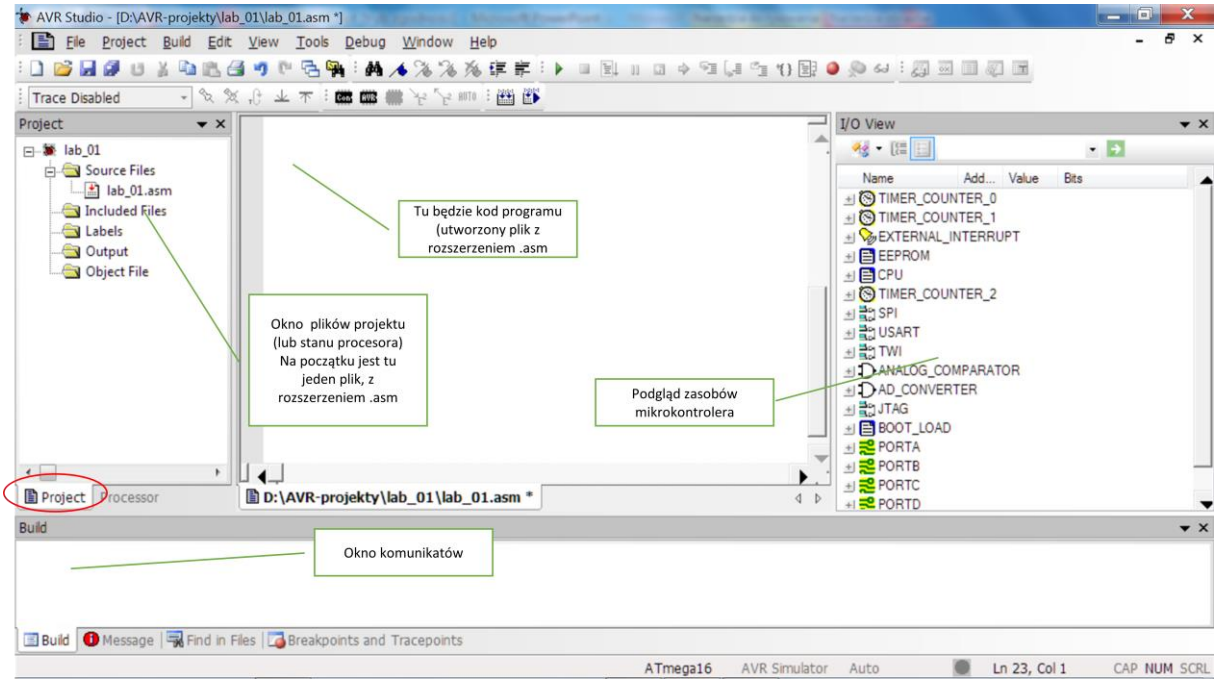
Zainstaluj środowisko AVR Studio 4. Utwórz pierwszy projekt według podanych wskazówek (**Project > Project Wizard**). Wpisz nazwę projektu (bez polskich znaków i spacji):



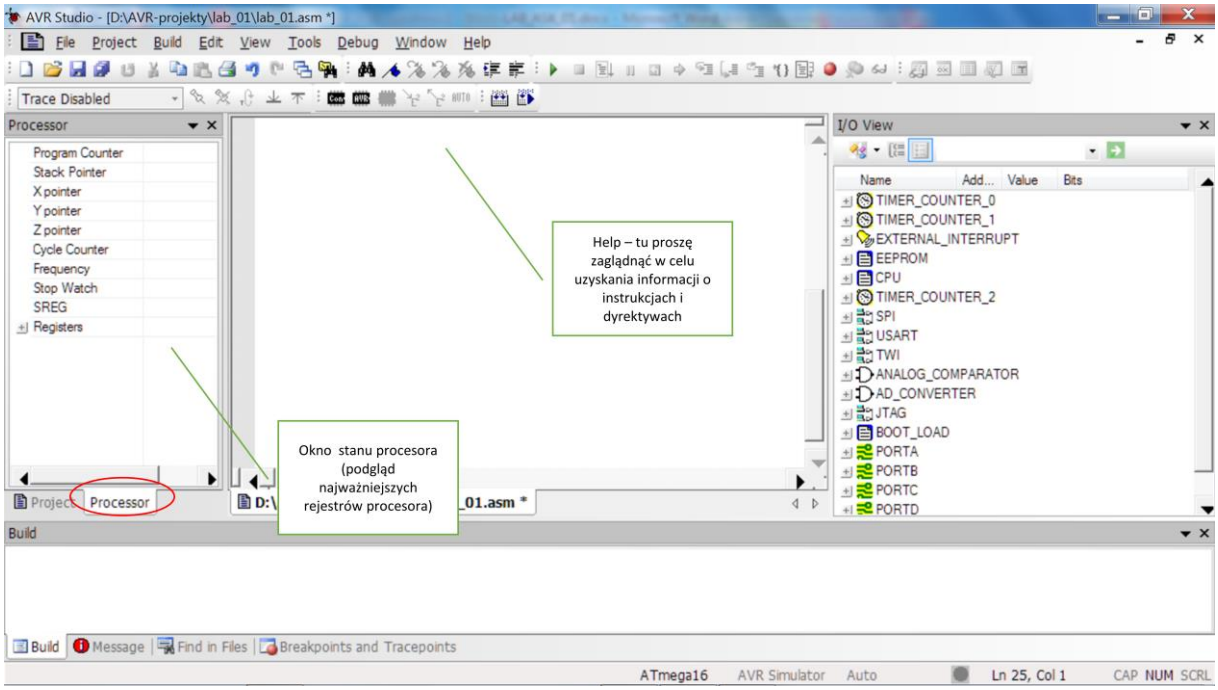
Wybierz typ procesora:



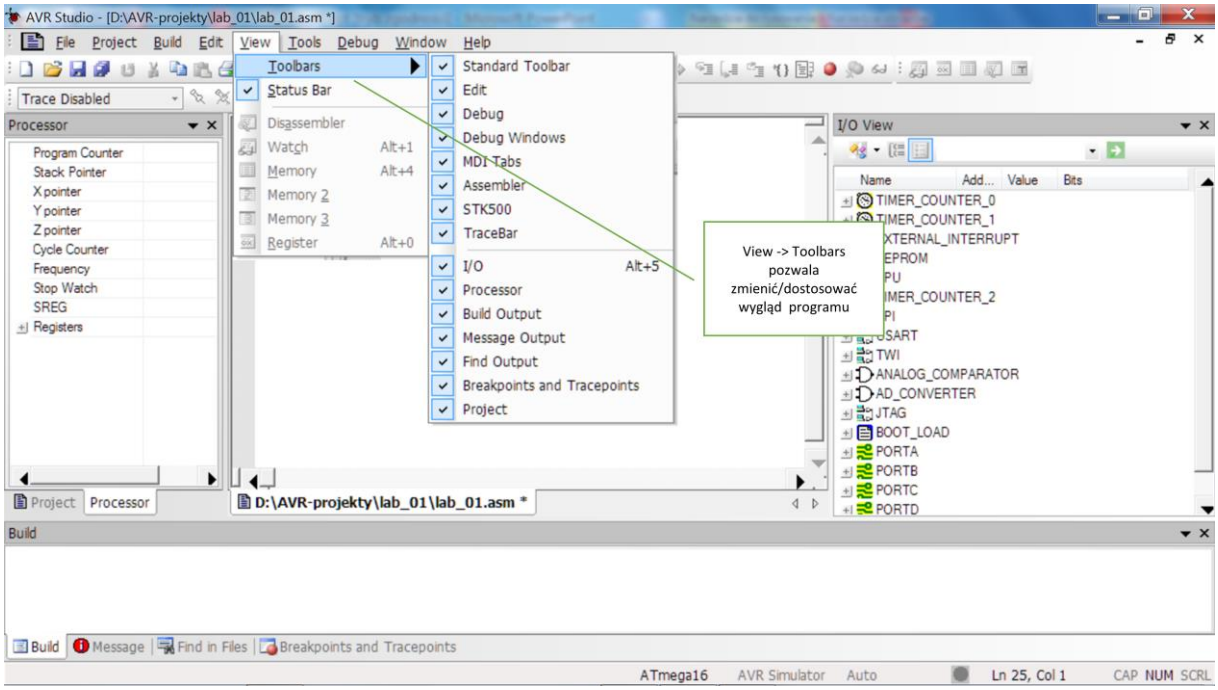
Okno programu w trybie podstawowym:



W trakcie pracy będzie można podglądać aktualny stan procesora (w trybie symulacji):



Jeżeli okno programu wygląda inaczej, można dostosować jego widok włączając/wyłączając poszczególne elementy (**View >Toolbars**):

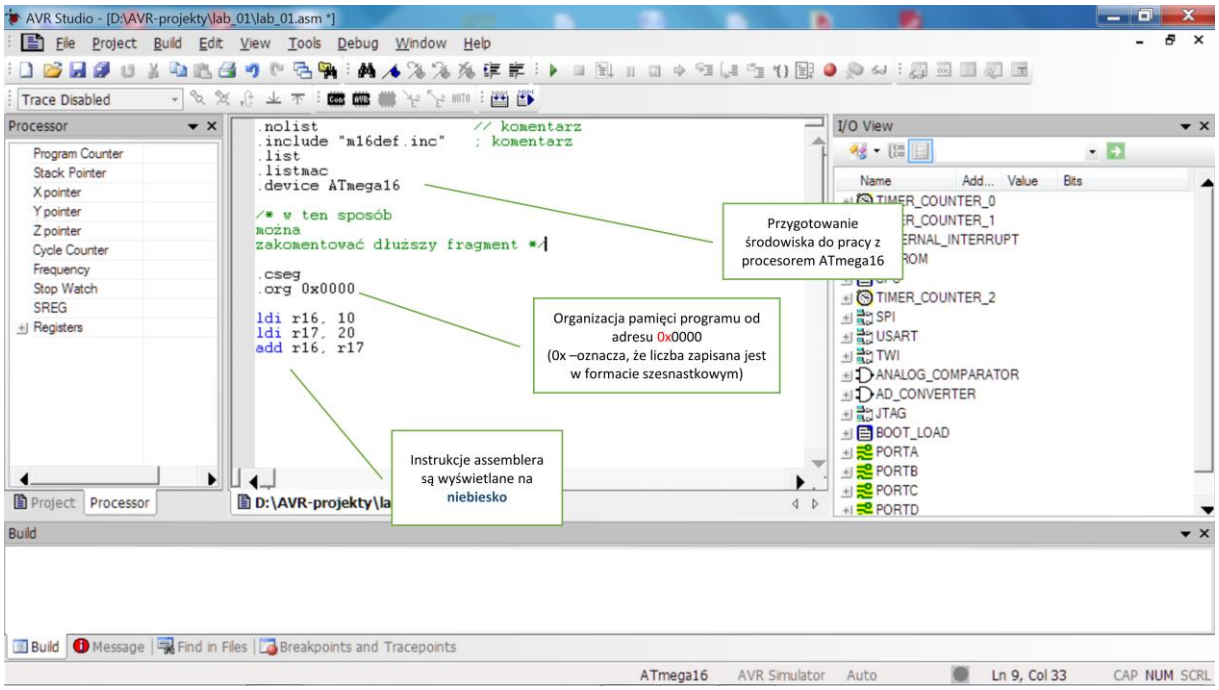


3. Pierwszy projekt w asemblerze

- Dodawanie liczb: 10 + 20 =
- Wpisz kod pierwszego programu (można używać małe i DUŻE litery: .nolist i .NOLIST lub .NoLisT - oznacz to samo)

Program _01

```
.nolist                  // komentarz
.include "m16def.inc" ; komentarz
.list
.listmac
.device ATmega16

/* w ten sposób
można
zakomentować dłuższy fragment */

.cseg
.org 0x0000

ldi r16, 10
ldi r17, 20
add r16, r17
```



Na podstawie pomocy dostępnej w programie (**Help > Assembler Help**) wyjaśnij działanie dyrektyw:

```
.nolist
.include "m16def.inc"
.list
.listmac
.device ATmega16
```

Wyszukaj w komputerze plik „**m16def.inc**”, otwórz go za pomocą Notatnika (lub innego prostego edytora tekstowego). Standardowo umieszczony jest w katalogu Atmel, np.:

C:\Program Files (x86)\Atmel\AVR Tools\AvrAssembler\Appnotes

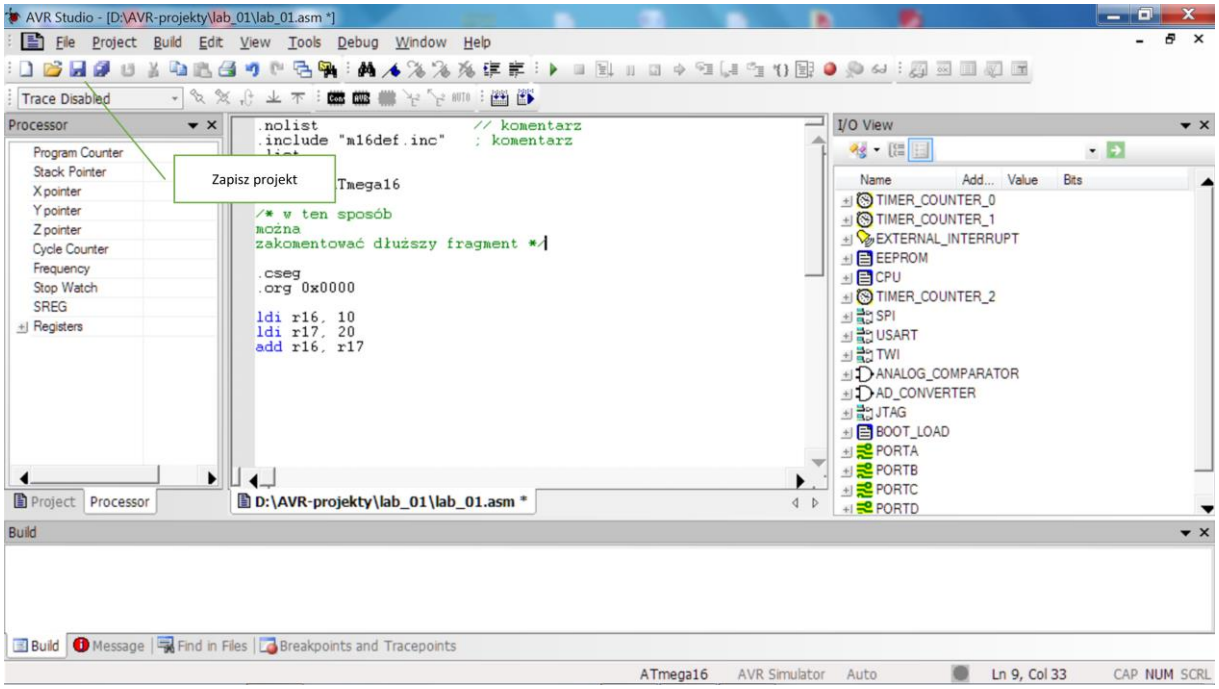
Wyszukaj fragment:

.equ RAMEND ….

- Jak działa dyrektywa .equ ?
- Jaka wartość jest przypisana do słowa RAMEND, co to oznacza?
- Po co dołączamy plik **m16def.inc** na początku każdego programu?

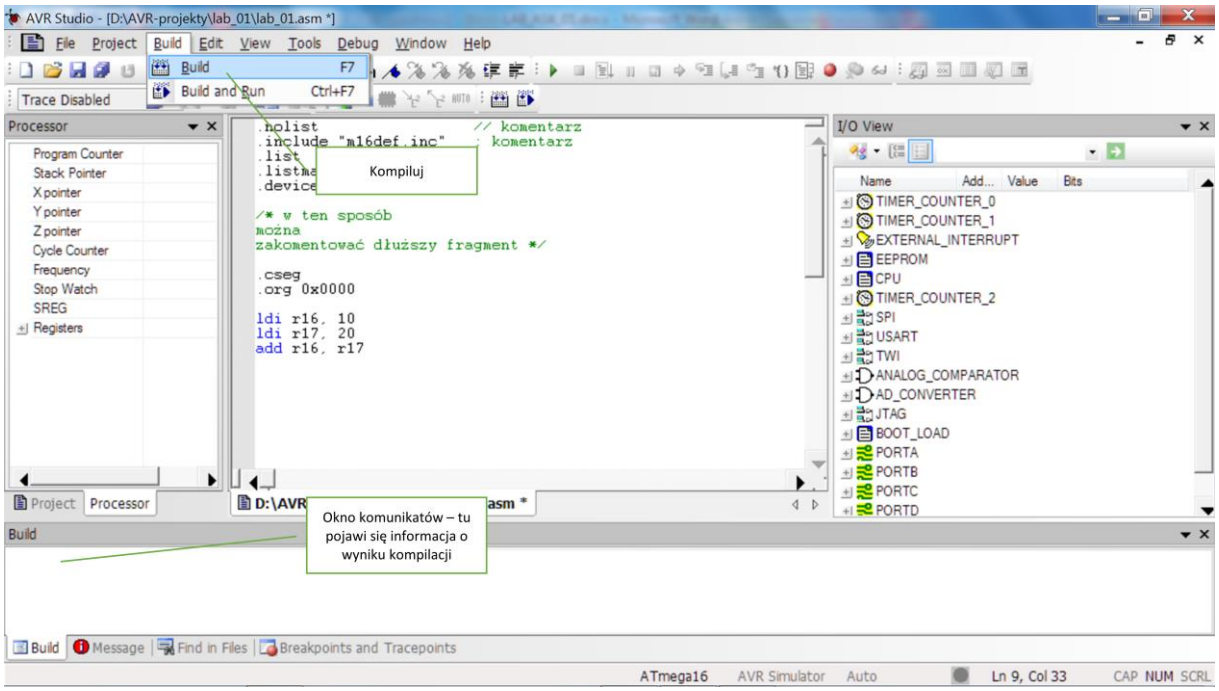
(znajomość wartości RAMEND będzie przydatna w kolejnych ćwiczeniach)

Zapisz projekt (**File -> Save as/Save**):



4. Kompilacja projektu

Uruchom kompilację (**Build -> Build** lub **F7**):



W oknie komunikatów otrzymujemy informację o wyniku kompilacji, np:

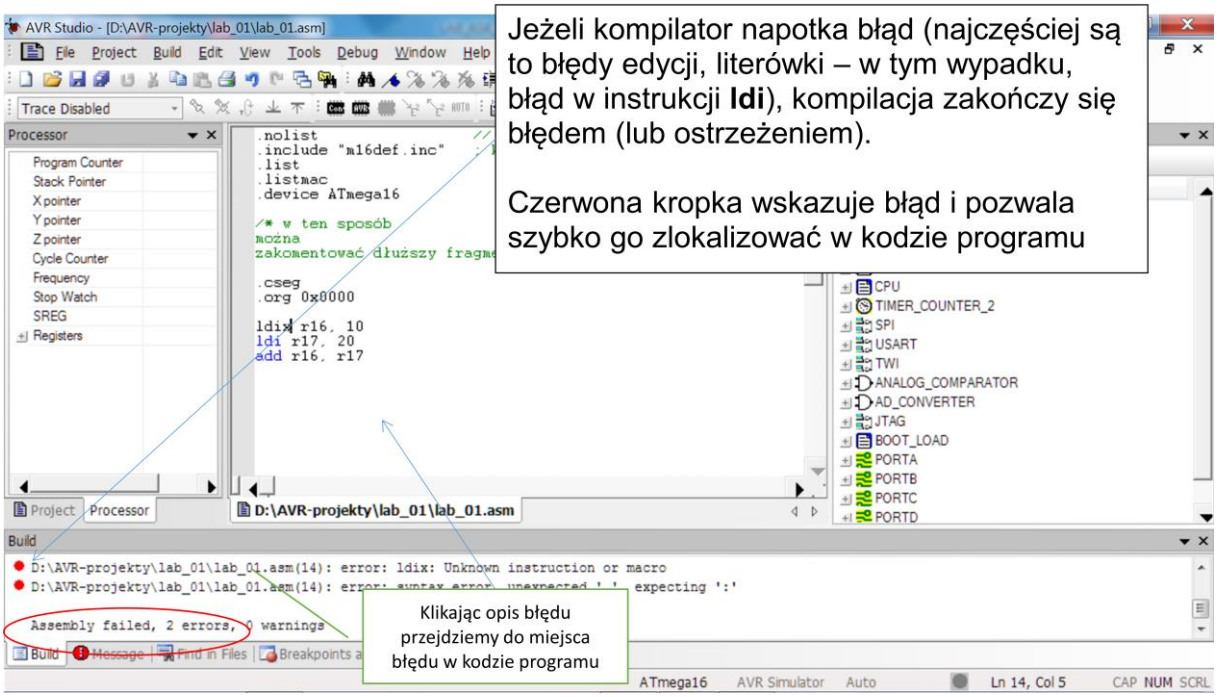
```
AVRASM: AVR macro assembler 2.1.12 (build 87 Feb 28 2007 07:31:13)
Copyright (C) 1995-2006 ATMEL Corporation

D:\AVR-projekty\lab_01\lab_01.asm(2): Including file 'C:\Program Files
(x86)\Atmel\AVR Tools\AvrAssembler2\Appnotes\m16def.inc'
D:\AVR-projekty\lab_01\lab_01.asm(16): No EEPROM data, deleting
D:\AVR-projekty\lab_01\lab_01.eep

ATmega16 memory use summary [bytes]:
Segment   Begin    End      Code  Data  Used   Size   Use%
---------------------------------------------------------------
[.cseg] 0x000000 0x000006     6     0     6  16384   0.0%
[.dseg] 0x000060 0x000060     0     0     0   1024   0.0%
[.eseg] 0x000000 0x000000     0     0     0    512   0.0%

Assembly complete, 0 errors. 0 warnings
```

Informacje o dołączonych plikach

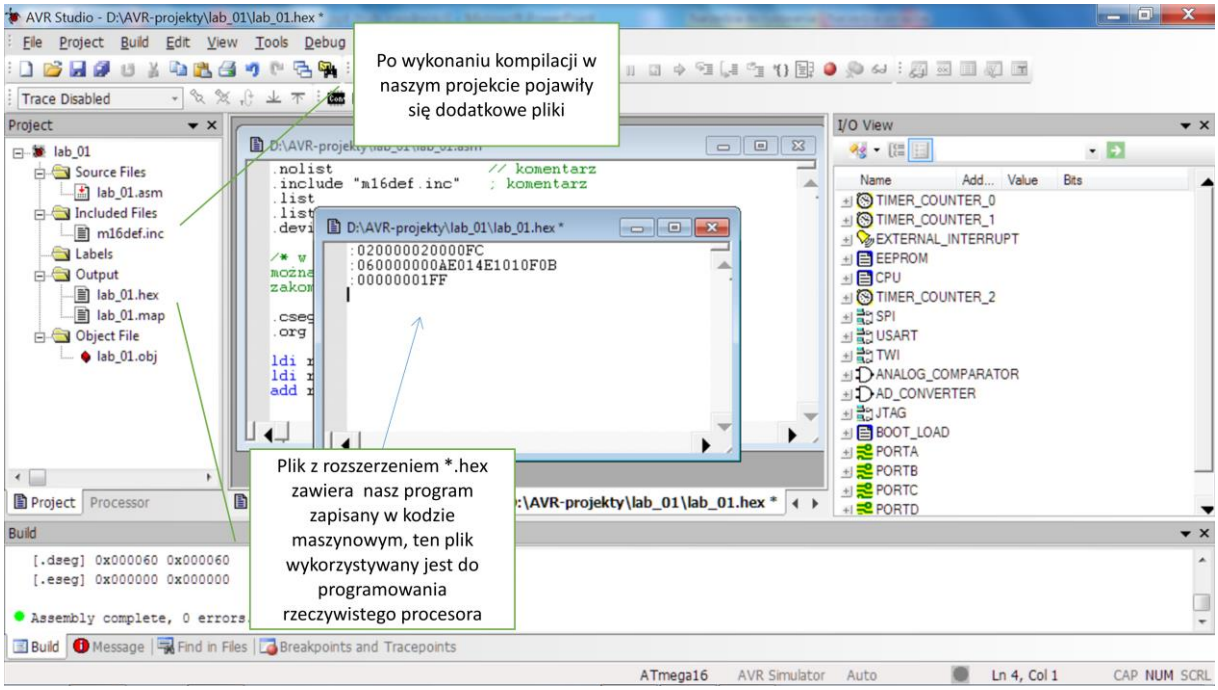
Informacje o wykorzystanych zasobach pamięci, w tym wypadku program zajmuje 6 bajtów z dostępnych ogółem 16384 bajtów pamięci programu

Brak błędów lub ostrzeżeń

W przypadku błędów (lub ostrzeżeń), dostajemy informację o typie i miejscu wystąpienia błędu w kodzie:



Widok po poprawnej kompilacji:



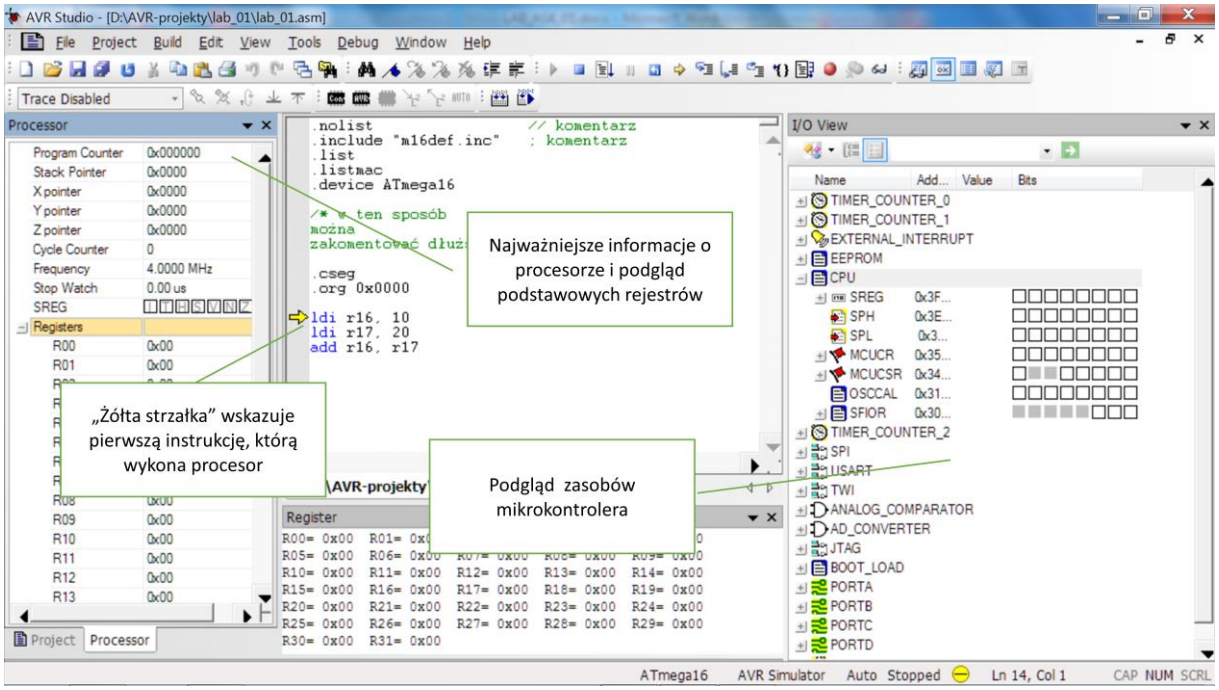
Otwórz plik *.hex i zobacz jego zawartość:

```
:020000020000FC
:060000000AE014E1010F0B
:00000001FF
```

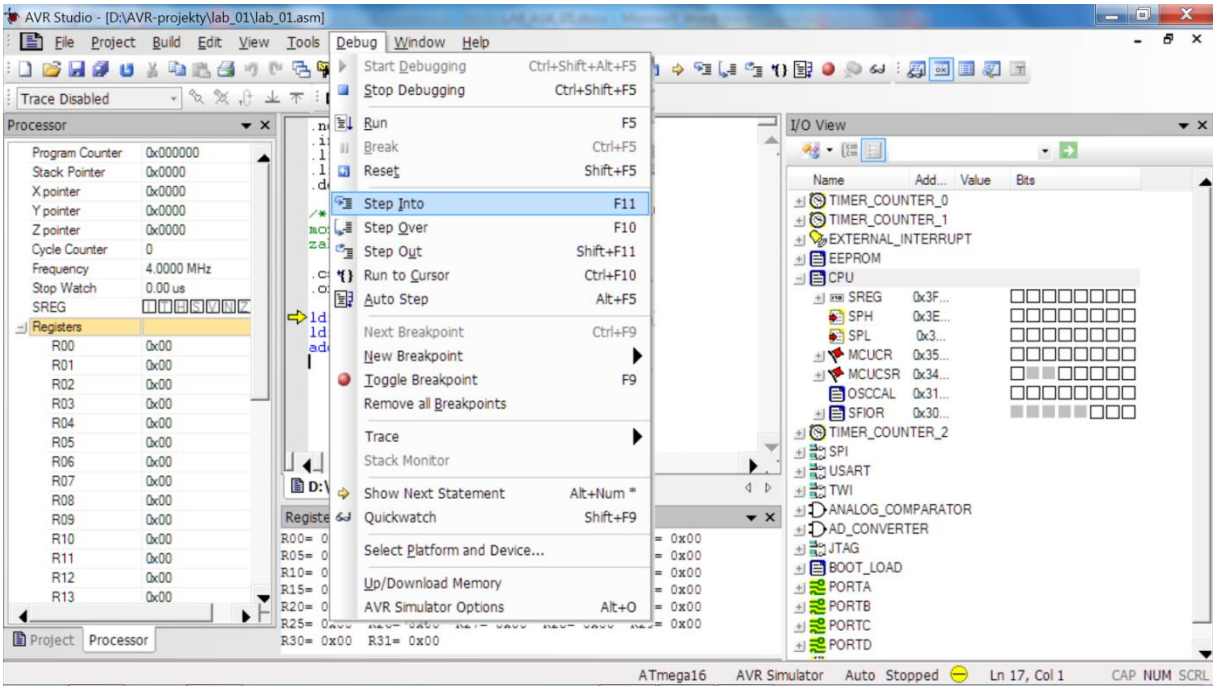
Jest to cały nasz program zapisany w postaci 0 i 1 (kod maszynowy). Taka postać programu wykorzystywana jest do programowania procesora (np. w programatorze).

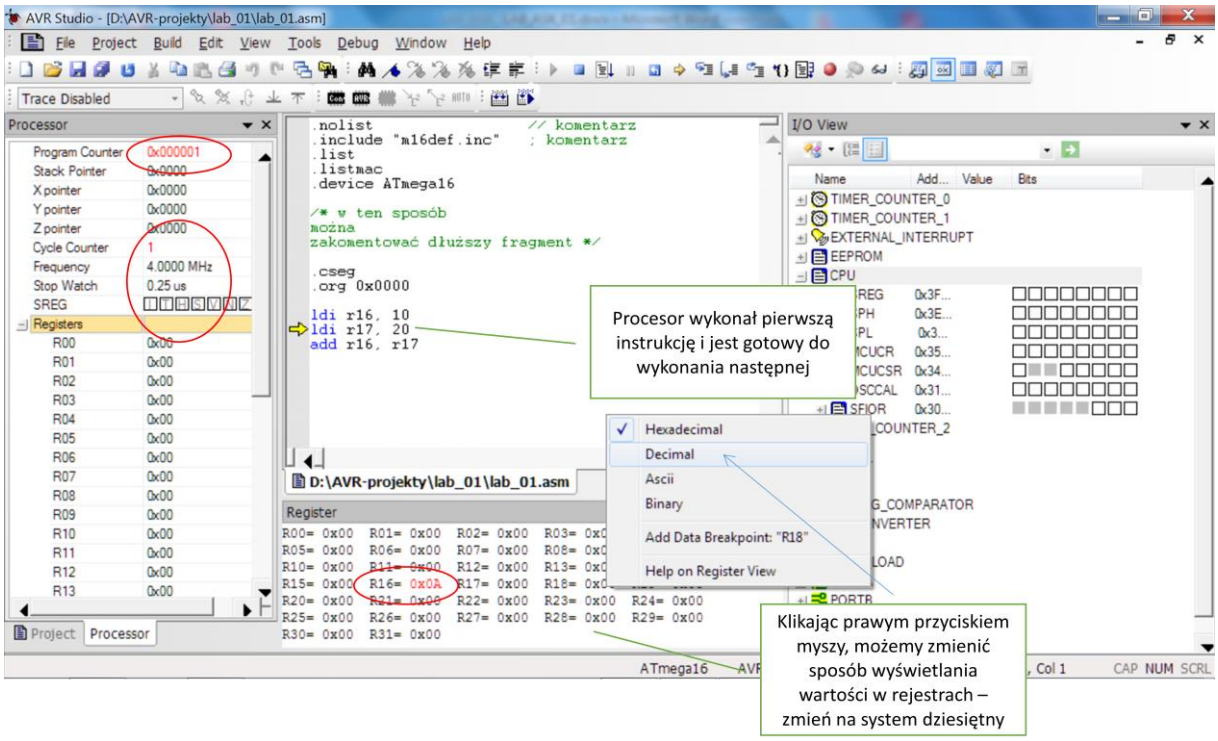
5. Uruchomienie programu w trybie symulacji

Jeżeli program nie zawiera błędów, możemy uruchomić tryb symulacji (**Debug > Start Debugging**), widok podstawowego okna trochę się zmienia, podobnie jak poprzednio możemy dostosować widok do naszych oczekiwań (**View > Toolbars**):



W trybie symulatora możemy uruchomić tryb pracy krokowej (**Debug > Step Into** lub **F11**):





Wykonaj kolejne kroki i sprawdź poprawność wykonania programu dodawania.

- Gdzie znajduje się wynik dodawania? Co wskazuje rejestr Program Counter?
- Co wskazuje Cycle Counter? Dlaczego po wykonaniu ostatniej instrukcji Cycle Counter osiąga tak dużą wartość?
- W ilu cyklach zegara powinien zostać wykonany cały program (zobacz - w opisie instrukcji **Help > Assembler Help** - ile taktów zegara potrzebuje każda instrukcja)?

6. Programowe "zatrzymanie programu"

Wyjdź z trybu symulacji (**Debug > Stop Debugging**), uzupełnij program instrukcjami:

Program_ 02

```
ldi r16, 10
ldi r17, 20
add r16, r17
nop

jmp PC
```

- Jak działa instrukcja **nop** ?
- Jakie jest ogólne działanie instrukcja **jmp** ?
- Co zrobi procesor po wykonaniu instrukcji **jmp PC** ?

Nie możemy zatrzymać działania (taktowania) procesora (procesor nie przestaje pracować, tzn. ciągle wykonuje kolejne instrukcje), ale możemy "zatrzymać" w miejscu nasz program. Jednym ze sposobów jest zastosowanie instrukcji **jmp PC**. Przeanalizuj w pracy krokowej działanie programu obserwując Program Counter i Cycle Counter.

7. Zadania do samodzielnej realizacji

1. Podgląd pamięci programu

- W opisie działania instrukcji LDI (**Help > Assembler Help**) znajdź binarny kod instrukcji (16-bit Optcode).
- W trybie symulatora otwórz podgląd pamięci programu (**View > Memory**), wybierz pamięć programu (Program). W pamięci programu znajdź (zapisaną szesnastkowo) instrukcję LDI r16, 10.
- Podobnie zlokalizuj pozostałe instrukcje: LDI, ADD, NOP, JMP.
- Otwórz plik wynikowy *.hex i również zlokalizuj te instrukcje.

2. Status znacznika (bitu) C w rejestrze SREG

- Zmień argumenty instrukcji ADD, wybierz dwie liczby tak, aby suma przekroczyła 255 (np: 100 + 200 =).
- W pracy krokowej obserwuj działanie programu i zachowanie znacznika C w SREG.
- Co wskazuje znacznik (bit C)?

3. Programy

- **Zadanie 1**: W kodzie programu liczby możemy zapisywać w różnych postaciach: dziesiętnie (10), szesnastkowo (0x0A) lub binarnie (0b00001001). Napisz program, który doda wartość 0xAB oraz 0x1C, a wynik (w postaci dziesiętnej) będzie znajdował się w rejestrze r20.
- **Zadanie 2**: zadanie podane przez prowadzącego.

4. Zadania dodatkowe (dla chętnych)

- **Zadanie A**: Napisz program, który zapisze daną liczbę **N** w rejestrze r0, następnie doda do niej daną liczbę **M** i wynik zapisze w rejestrze następnym (r1), następnie do wartości z rejestru r1 doda liczbę **M** i wynik umieści w kolejnym rejestrze (r2), następnie do wartości zapisanej w rejestrze r2 doda liczbę **M** i wynik umieści w rejestrze r3 ... itd., aż do rejestru r31 (czyli w kolejnych rejestrach zapisujemy kolejne wartości pewnego ciągu arytmetycznego). Wartości **N** i **M** podaje prowadzący.
- **Zadanie B**: zadanie podane przez prowadzącego.

8. Sprawozdanie

- W sprawozdaniu należy umieścić kody programów z odpowiednim wyjaśnieniem działania zastosowanych dyrektyw i instrukcji.
- Podobnie opisać realizację zadań z punktu "Zadania do samodzielnej realizacji"
- Zamieścić odpowiedzi na pytania pojawiające się w instrukcji.
- Na podstawie noty katalogowej ATmega16 opisać, gdzie znajdują się i do czego służą rejestry r0-r31 (strona 10, **General Purpose Register File**).