Wykład 4

Środowisko programistyczne

Dostępne kompilatory

- KEIL komercyjny
- GNU licencja GPL
- ARM komercyjny
- IAR komercyjny

Porównanie kompilatorów

Dhrystone V2.1

Parameter	Compiler			
	Keil CA BETA	GNU V3.22	ARM ADS V1.2	IAR V4.11A
Execution Speed (µSeconds)	25.4	112.9	16.8	24.4
Dhrystones/sec	39,370.1	8,857.4	59,382.4	40,983.6
Total Code Size (bytes)	10,330	36,004	22,266	19,892
Stack Size (bytes)	208	852	608	356
Total Data Size (bytes)	10,256	11,912	10,256	10,269

All tests were performed under identical conditions using the Keil µVision Simulator.
The ARM device used was a Philips LPC2294 running at 60MHz in Thumb Mode.

Porównanie kompilatorów

Whetstone

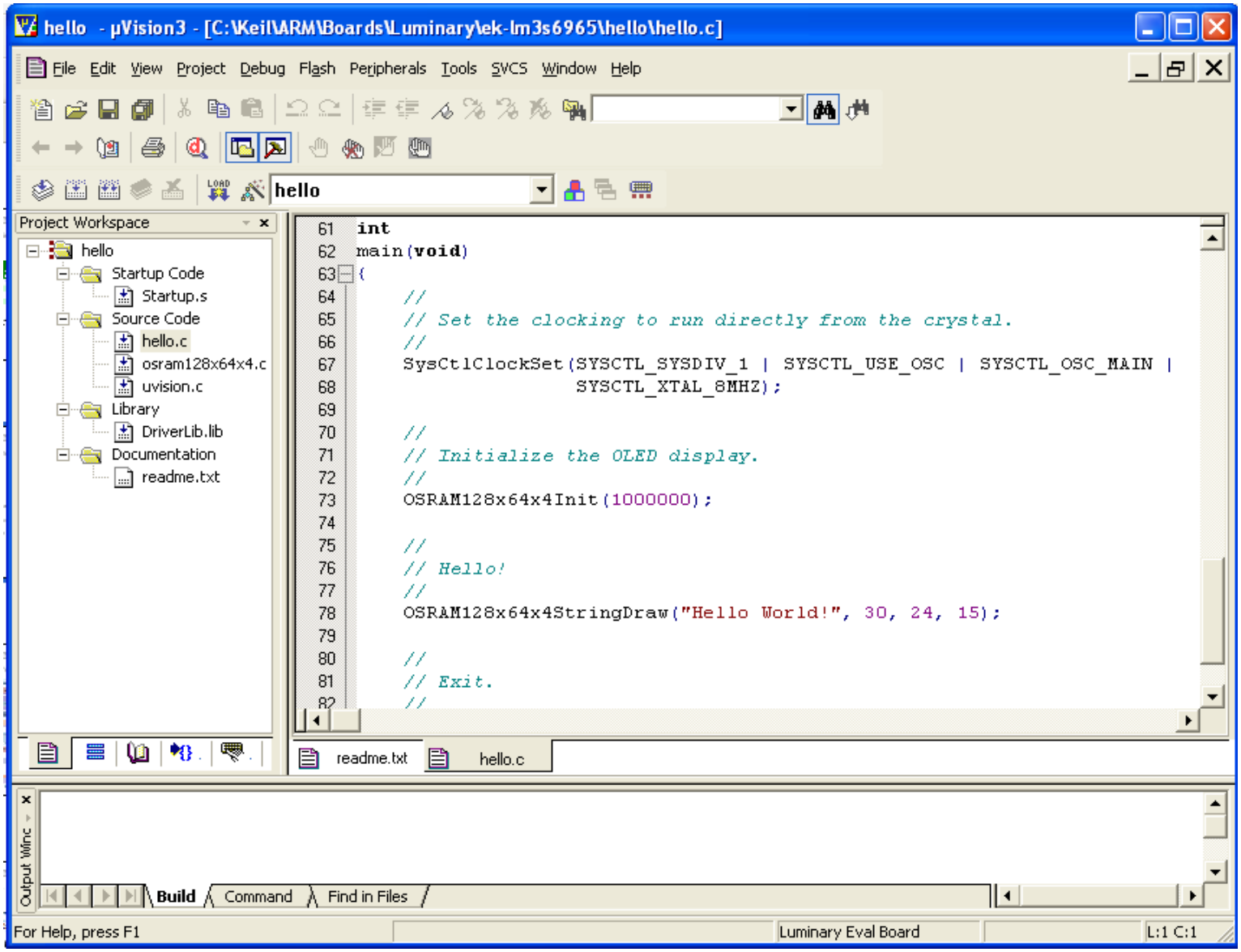
Parameter	Compiler			
	Keil CA BETA	GNU V3.22	ARM ADS V1.2	IAR V4.11A
Execution Speed (seconds)	0.195308	2.461430	0.268623	0.635633
Whetstone (KWIPS)	5,263	406	3,846	1,587
Total Code Size (bytes)	8,492	44,428	28,516	16,316
Stack Size (bytes)	212	1,552	710	488
Total Data Size (bytes)	72	1,768	76	72

All tests were performed under identical conditions using the Keil µVision Simulator.
The ARM device used was a Philips LPC2294 running at 60MHz in Thumb Mode.

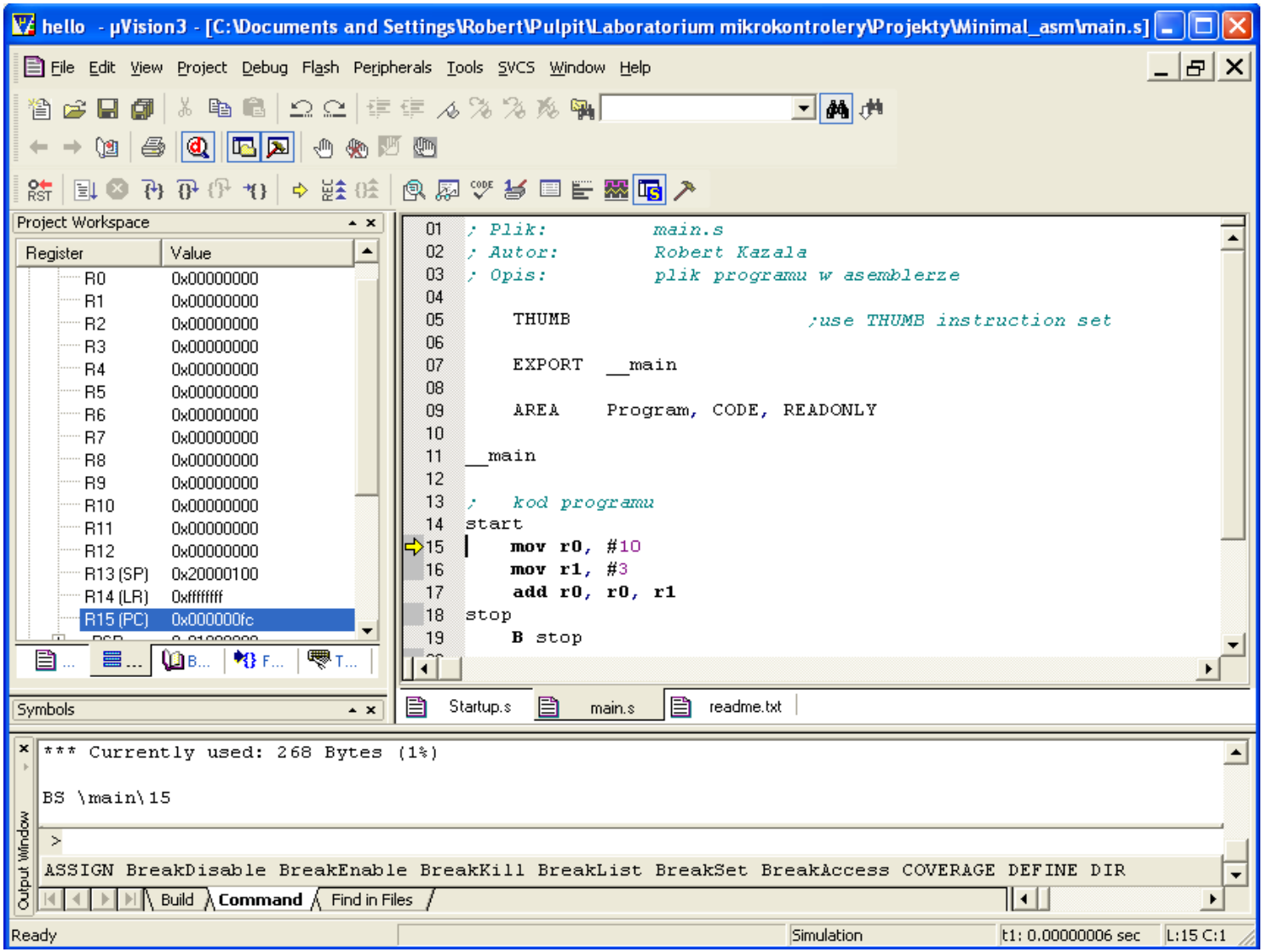
Środowisko programistyczne Keil uVision

- Graficzny edytor
- Kompilator i linker
- Symulator rdzenia procesora
- Symulator otoczenia procesora
- Wbudowane wsparcie dla różnych rodzin mikrokontrolerów
- Graficzny debuger

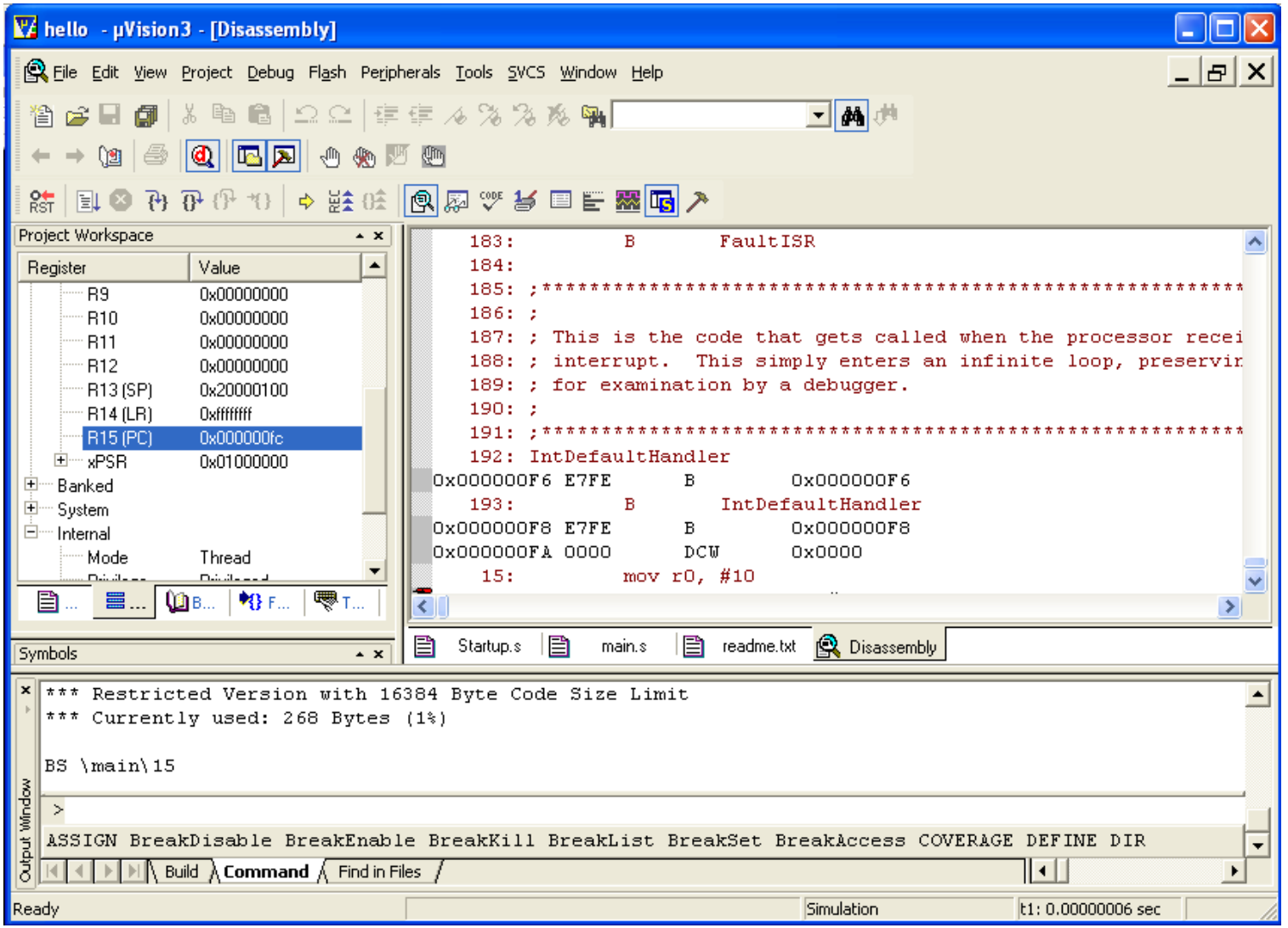
Widok edytora



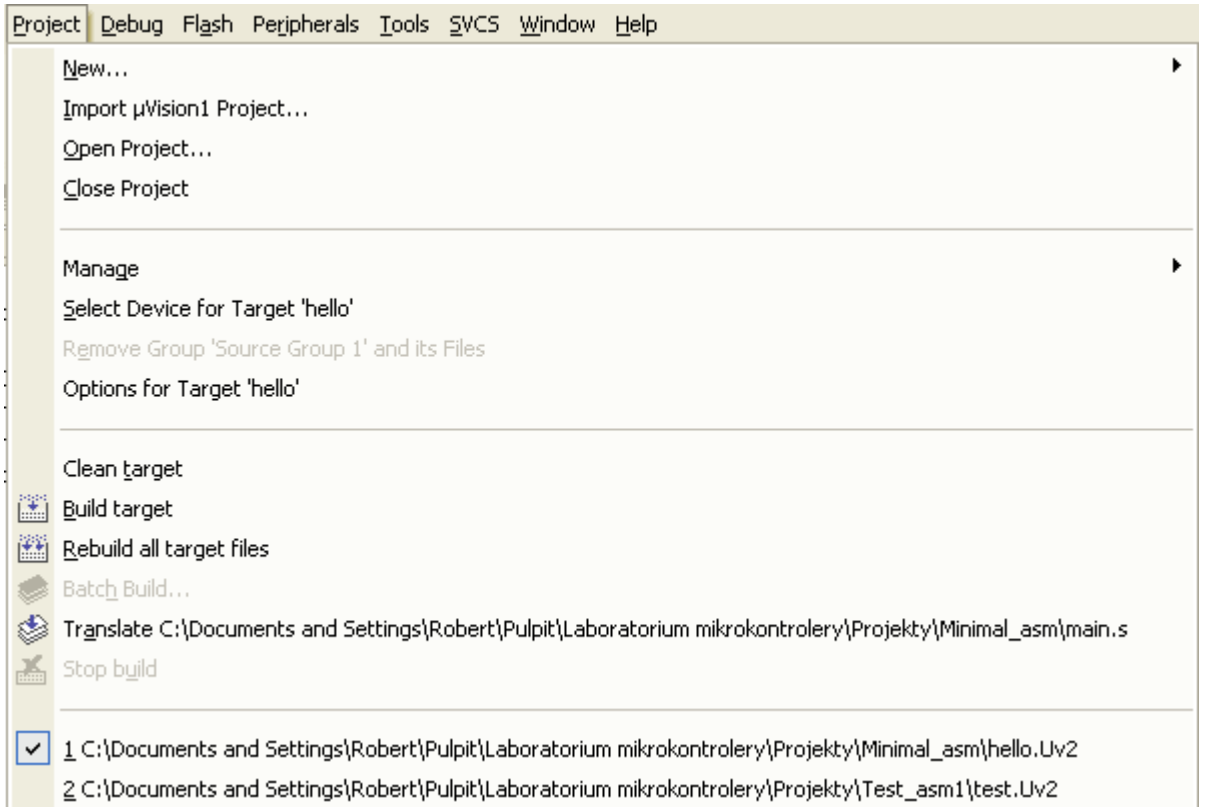
Widok debuggera



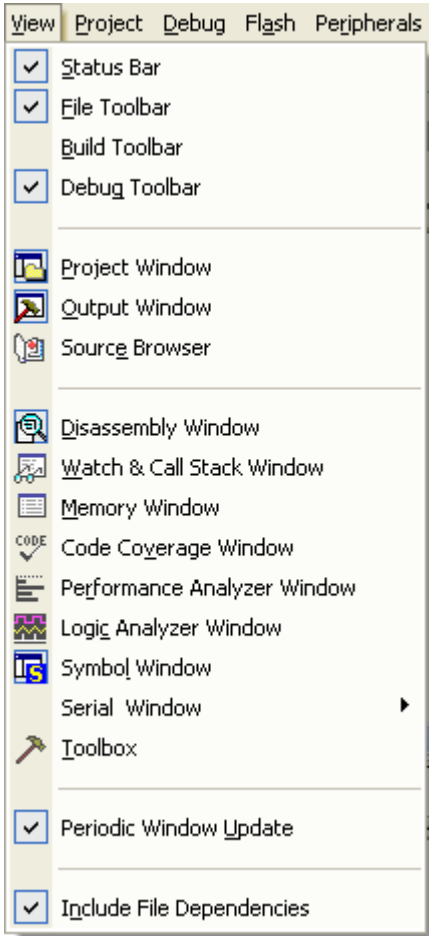
Widok disasemblera



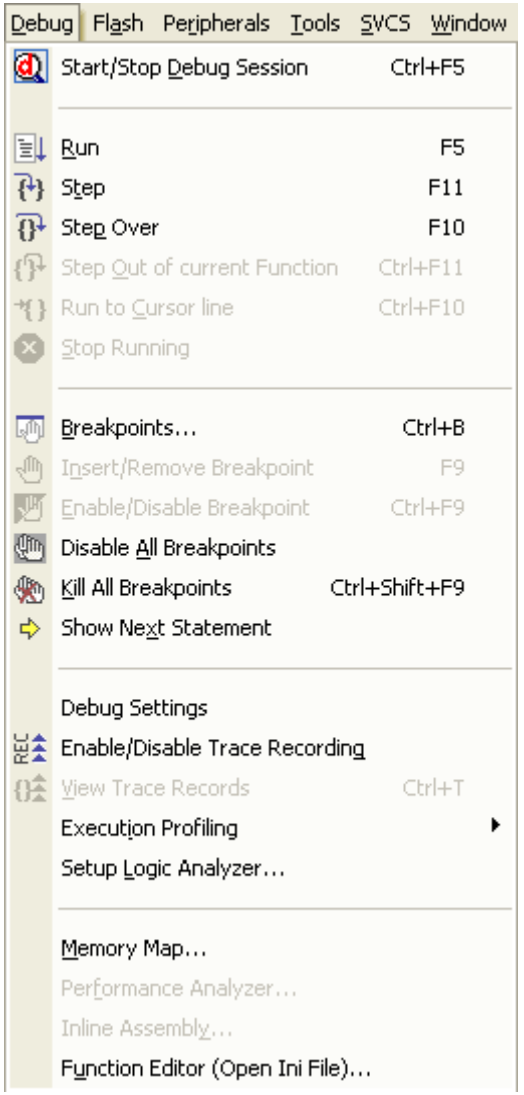
Menu project



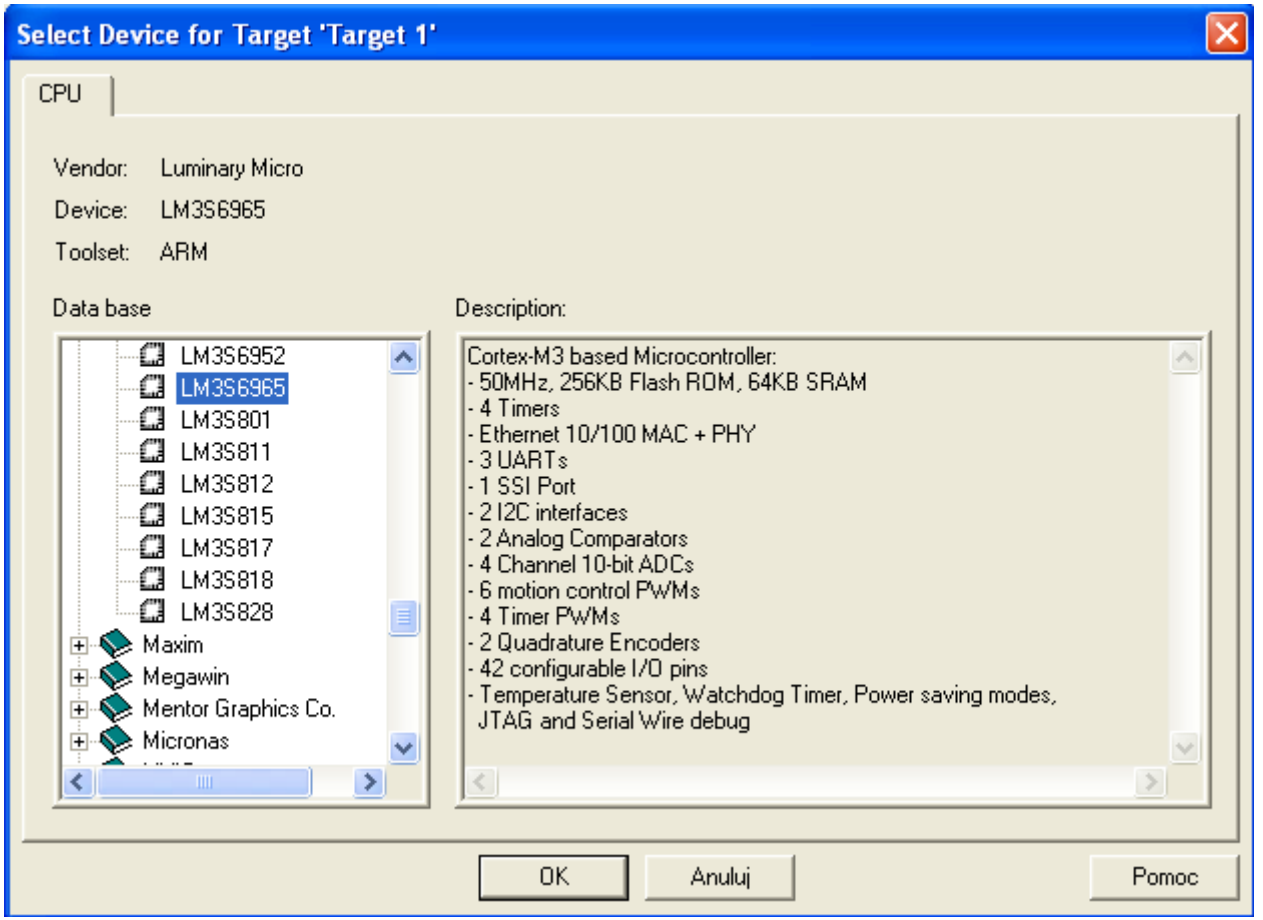
Menu view

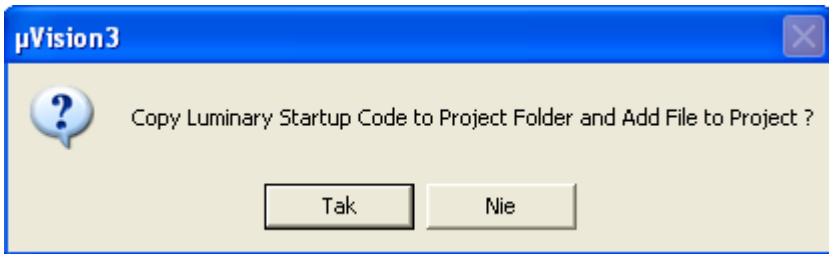

View
Project
Debug
Flash
Peripherals
Status Bar
File Toolbar
Build Toolbar
Debug Toolbar
Project Window
Output Window
Source Browser
Disassembly Window
Watch & Call Stack Window
Memory Window
Code Coverage Window
Performance Analyzer Window
Logic Analyzer Window
Symbol Window
Serial Window
Toolbox
Periodic Window Update
Include File Dependencies

Menu debuggera

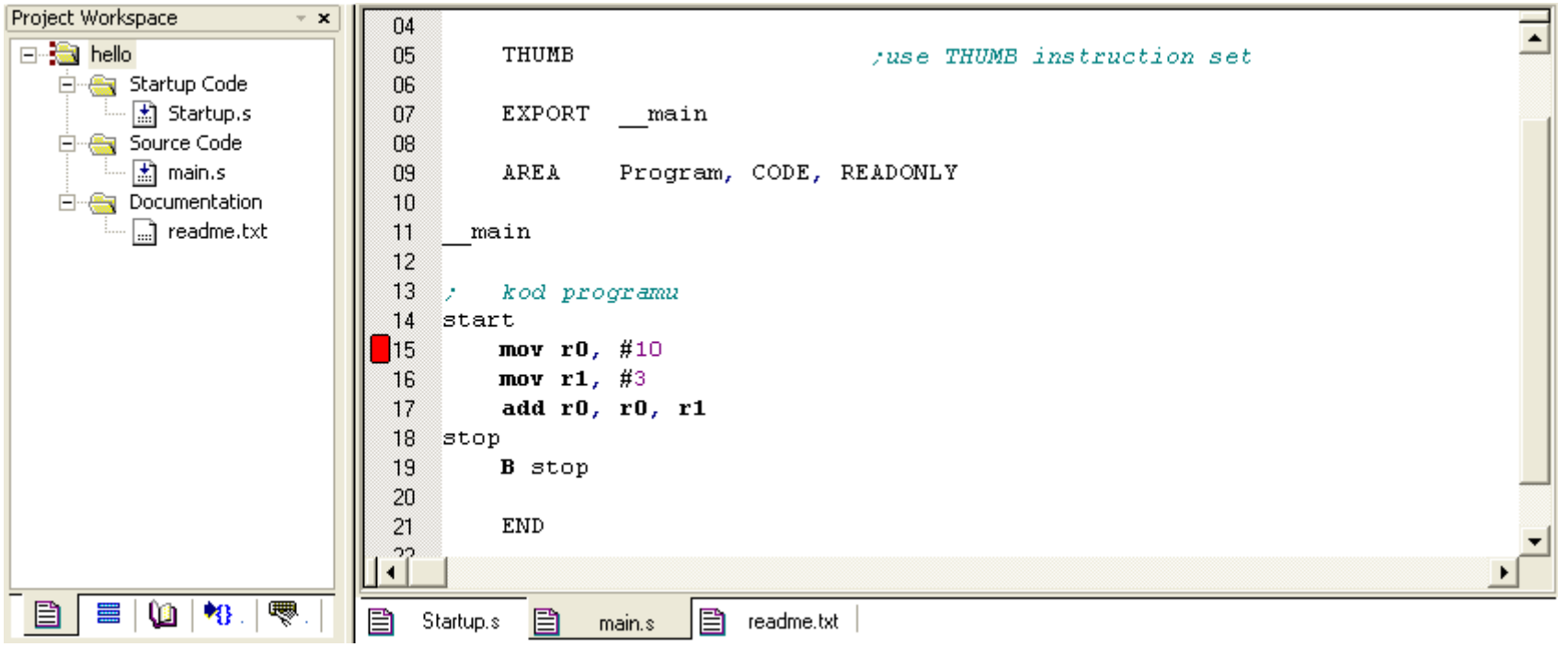


Wybór mikrokontrolera





Struktura projektu



Struktura modułów asemblera

- Moduły napisane w języku asemblera są przetwarzane przez ASM asembler do postaci kodu obiektowego.

- Pliki mogą zawierać programy lub funkcje. Pojedyncza sekcja kodu wystarczy do utworzenia aplikacji.

- Pliki przetwarzane są do formatu ELF i następnie łączone są w linkerze do postaci kodu wynikowego.

- Zasady rozmieszczania kodu i danych definiowane są na poziomie kodu asemblera i zapisywane w plikach źródłowych.

- Każdy moduł przetworzony do formatu ELF jest niezależny, ma swoją nazwę i zawiera kod lub dane, które będą rozmieszczone przez linker w pamięci mikrokontrolera.

- Sekcje kodu są najczęściej sekcjami z atrybutem tylko do odczytu, sekcje danych umożliwiają zapis i odczyt.

Struktura modułów asemblera

- Linia programu asemblera ma następującą postać

{etykieta} {instrukcja|dyrektywa|pseudo-instrukcja} {;komentarz}

- Uwaga! Jeżeli w kodzie nie ma etykiety instrukcje muszą być poprzedzone spacją lub tabulatorem.

- Instrukcje i dyrektywy mogą być pisane małymi lub dużymi literami. Nie mogą być jednak mieszane.

- Linie, które są za długie mogą być dzielone przy wykorzystaniu znaku \ (backslash) umieszczanego na końcu linii. Znak ten nie może być oddzielony od poprzedzającej go instrukcji za pomocą spacji lub tabulatora. W takim przypadku jest on pomijany.

Etykiety i komentarze

- Etykiety reprezentują adresy. Adresy wskazywane przez etykiety są obliczane w czasie asemblacji kodu. Etykiety mogą być definiowane jednokrotnie.

- Etykiety lokalne rozpoczynają się liczbą od 0-99. W przeciwieństwie do etykiet mogą być definiowane wiele razy. Jeżeli asembler znajdzie skok do takiej etykiety to wykonuje skok do najbliższej etykiety o wskazanej nazwie. Widoczność etykiet lokalnych jest ograniczona do jednego obszaru AREA.

- Pierwszy średnik umieszczony w linii programu oznacza rozpoczęcie komentarza. Koniec linii jest zakończeniem komentarza. Dopuszczalne są linie tylko z samym komentarzem.

Przykładowy program

```
   AREA Program, CODE, READONLY
                  ; Nazwa bloku Program

   ENTRY          ; Oznaczenie pierwszej instrukcji
start
   MOV r0, #10       ; Ustawienie parametrów
   MOV r1, #3
   ADD r0, r0, r1       ; r0 = r0 + r1

stop  B stop          ; Pętla kończąca przykład

   END          ; Znak końca pliku
```

Dyrektywa AREA

- W pliku źródłowym dyrektywa AREA oznacza rozpoczęcie sekcji.
- Ta dyrektywa umożliwia podanie nazwy sekcji oraz jej atrybutów.
- Atrybuty umieszczane są po nazwie sakcji i oddzielone przecinkami.
- Nazwa może być dowolnym zestawem znaków.
- W przypadku znaków rozpoczynających spoza alfabetu wymaga otoczenia kreskami np. |1_ObszarDanych|.
- W przykładowym kodzie zdefiniowana jest jedna sekcja o nazwie Program, która zawiera kod i jest oznaczona jako tylko do odczytu READONLY.

Dyrektywa ENTRY i END

- Dyrektywa ENTRY oznacza pierwszą instrukcję, która będzie wykonywana.

- W pliku zdefiniowano dodatkowo dwie etykiety start i stop, które nie są wymagane. Pomiędzy etykietami zapisany jest zasadniczy kod sekcji.

- Po wykonaniu kodu wykonywana jest instrukcja skoku do etykiety stop.

- Dyrektywa END informuje asembler o zakończeniu pliku źródłowego.

Dyrektywy asemblera

Oprócz przedstawionych dyrektyw występują także inne, które można podzielić na:

- definiujące nazwy symboliczne (GBLA, GBLL,GBLS, LCLA, LCLL, LCLS),
- definiujące obszary danych (DCB, DCD, DCQ, SPACE)
- dyrektywy sterujące asemblerem (MACRO, WHILE, IF, INCLUDE)
- dyrektywy raportujące (ASSERT, INFO, OPT, TTL, SUBT).