Instytut Informatyki
ZMiTAC

LABORATORIUM SMIW

Laboratorium 20,21

Temat: Mikrokontrolery AVR

Mgr inz. Jarosław Paduch

Cel ćwiczenia:

Celem ćwiczenia jest:

1. Zapoznanie się architekturą mikrokontrolerów AVR.
2. Zapoznanie się z instalacją i używaniem narzędzi programowych (bezpłatnych) dla mikrokontrolerów AVR, t.j. AVR Studio 4.12 i WinAVR.
3. Nauka programowania w asemblerze mikrokontrolera AVR.
4. Nauka programowania w języku C dla mikrokontroler AVR.

.

Wymagania sprzętowe:

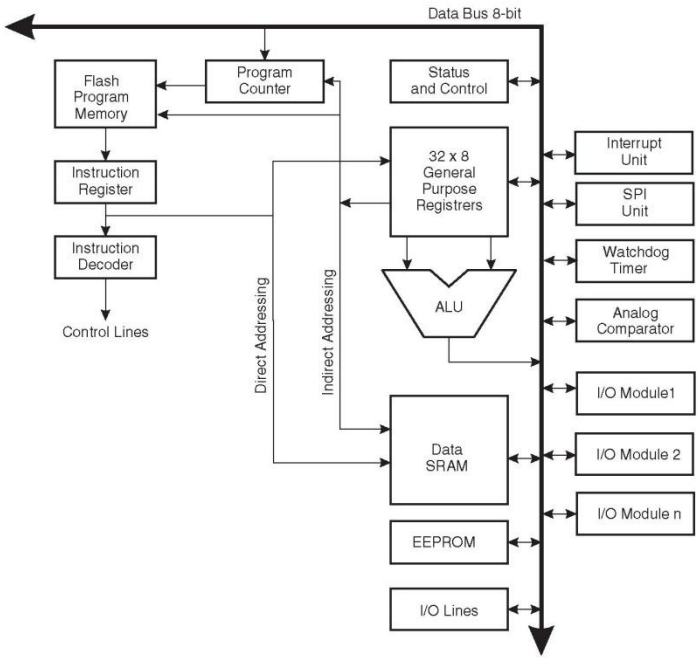
Jedno stanowisko mikrokomputer klasy IBM PC

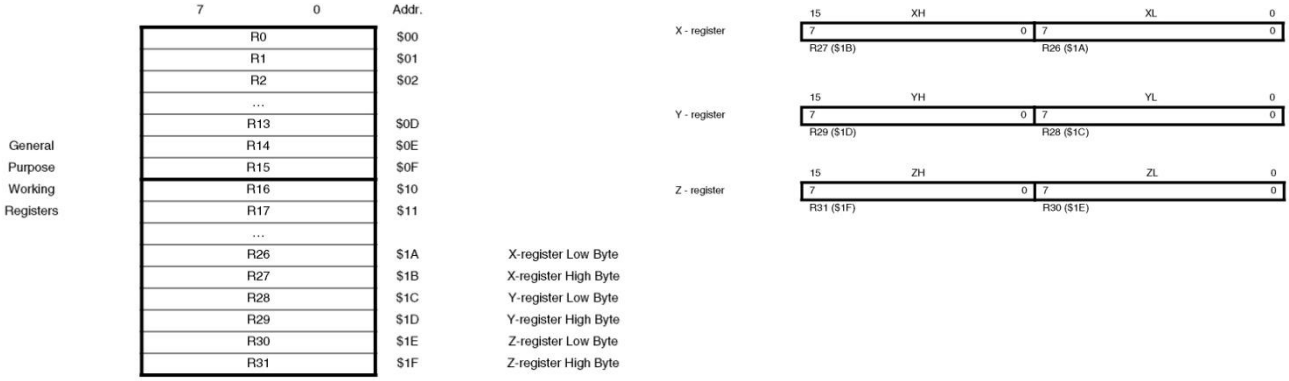
Wymagania programowe:

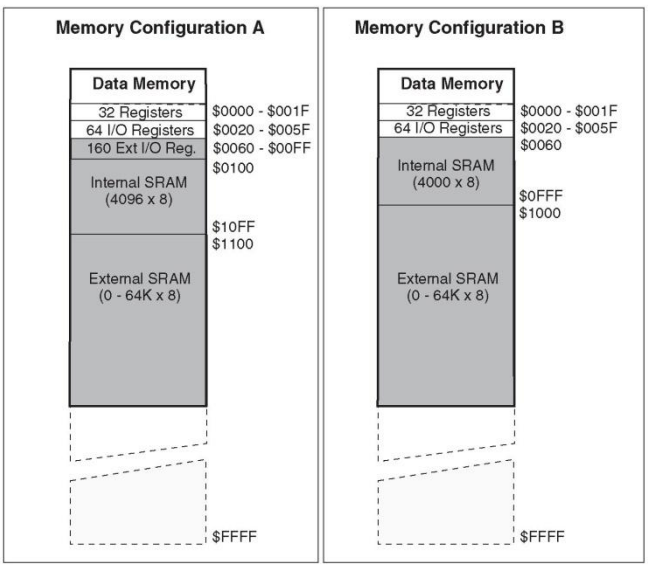
Mikrokomputer klasy IBM PC z zainstalowanym oprogramowaniem AvrStudio w wersji 4.12 oraz WinAVR 1.0 .

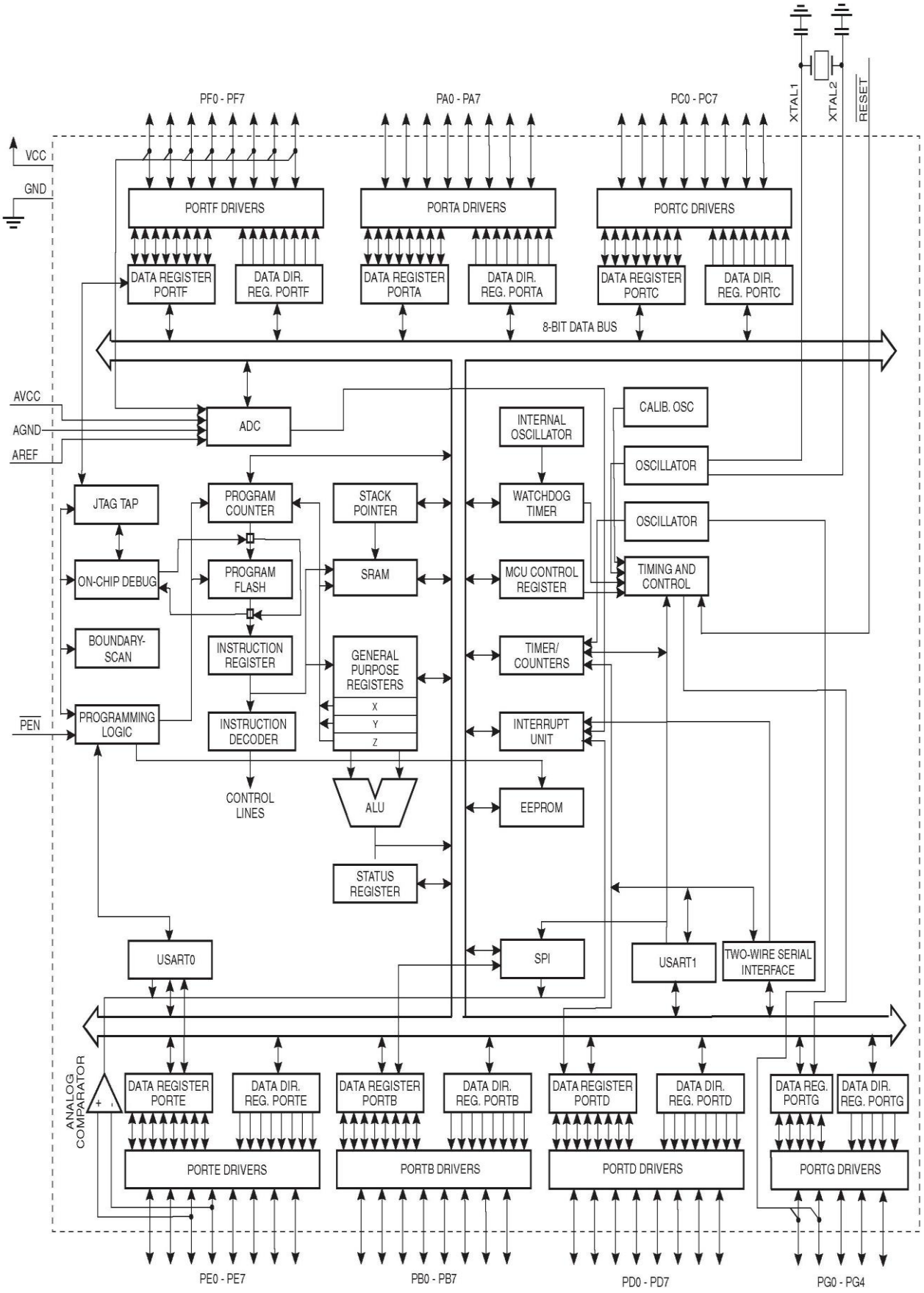
Wprowadzenie:

Mikrokontroler z serii AVR jest to 8 bitowy mikrokontroler typu RISC. Budowa jego opiera się o architekturę harwardzką, czyli w mikrokontrolerze są rozdzielone magistrale do pamięci programu (16bitów) i do pamięci danych (8bitów). Odpowiednia konfiguracja zewnętrzna mikrokontrolerów umożliwia dołączenie zewnętrznej pamięci danych o rozmiarze do 64 KB. Niestety brak jest możliwości dołożenia zewnętrznej pamięci programu. Dużą szybkość mikrokontrolera zapewnia przetwarzanie potokowe, powodujące wykonywanie większości rozkazów mieszczących się w jednym cyklu zegarowym, oraz 32 bajtowy obszar rejestrów roboczych, o natychmiastowym dostępie. Ich dodatkową zaletą jest brak ścisłego określenia akumulatora. Tę funkcję może pełnić dowolnie wybrany rejestr, spośród 32-bajtowego banku rejestrów roboczych. Zastosowanie szeregowego algorytmu programowania oraz pamięć programu typu "Flash", umożliwia programowanie i przeprogramowanie mikrokontrolera po umieszczeniu go w układzie. Konstruktorzy uwzględnili również układ Watchdog, jak i tryb pracy z obniżonym poborem mocy, które w obecnej chwili stają się standardem w budowie mikrokontrolerów. Rysunki przedstawiają architekturę jednostki centralnej; mapę pamięci i zestaw rejestrów mikrokontrolera.

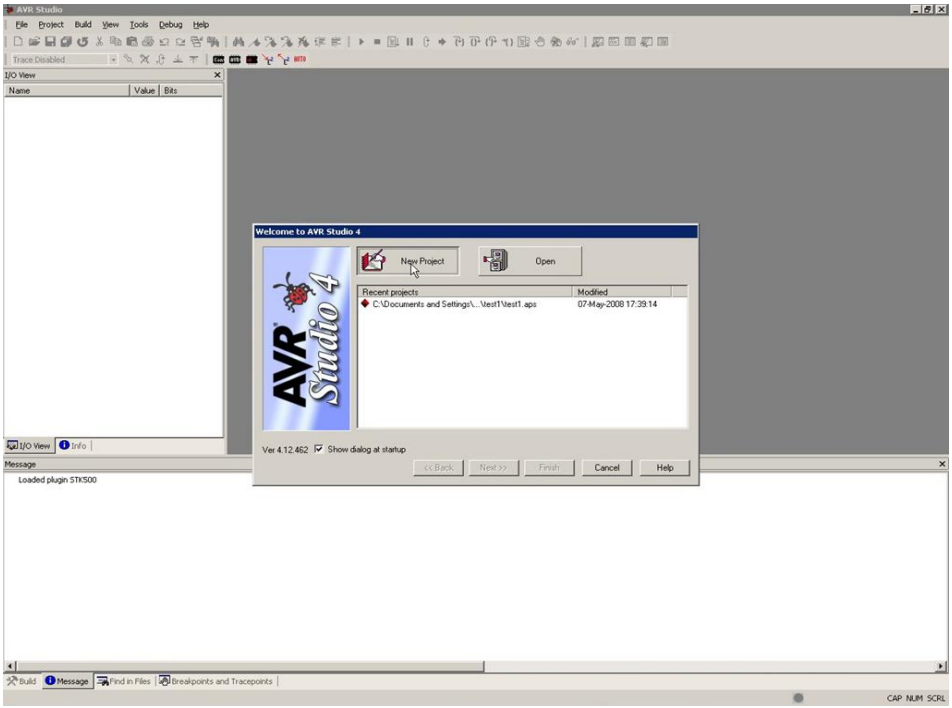

Data Bus 8-bit
Flash Program Memory
Program Counter
Status and Control
Instruction Register
Instruction Decoder
Control Lines
32 x 8 General Purpose Registrers
ALU
Direct Addressing
Indirect Addressing
Data SRAM
EEPROM
I/O Lines
Interrupt Unit
SPI Unit
Watchdog Timer
Analog Comparator
I/O Module1
I/O Module 2
I/O Module n


7 0 Addr.
R0 $00
R1 $01
R2 $02
R13 $0D
R14 $0E
R15 $0F
R16 $10
R17 $11
R26 $1A
R27 $1B
R28 $1C
R29 $1D
R30 $1E
R31 $1F
General Purpose Working Registers
X-register Low Byte
X-register High Byte
Y-register Low Byte
Y-register High Byte
Z-register Low Byte
Z-register High Byte
X - register
Y - register
Z - register
15 XH XL 0
R27 ($1B)
R26 ($1A)
R29 ($1D)
R28 ($1C)
R31 ($1F)
R30 ($1E)

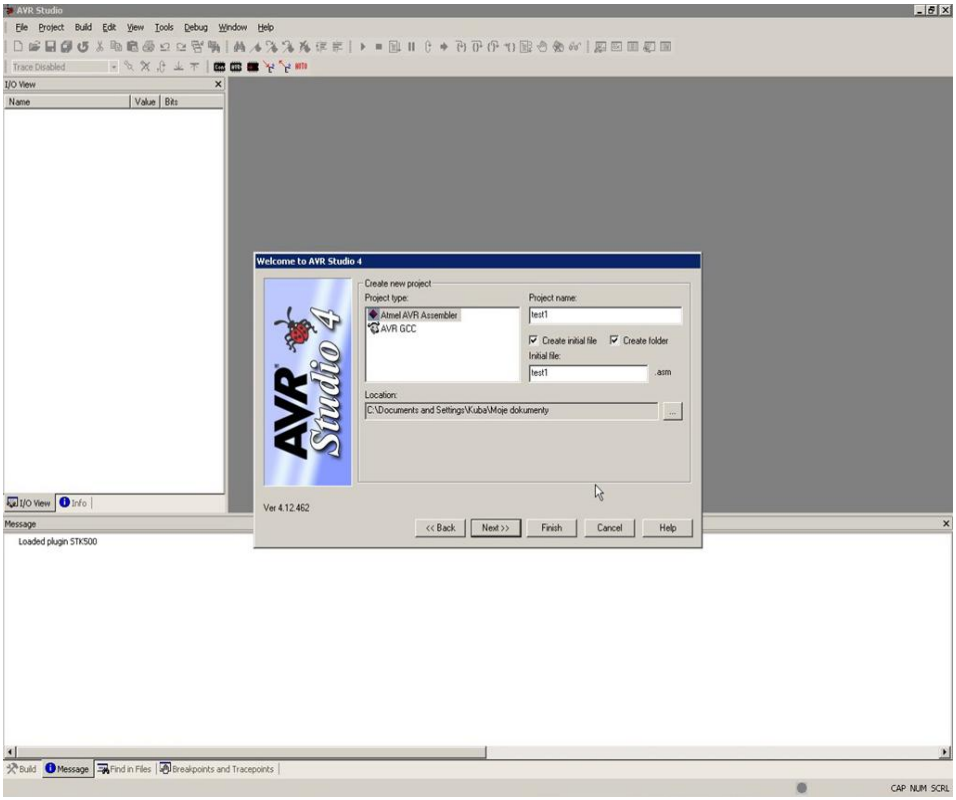

Memory Configuration A
Data Memory
32 Registers
64 I/O Registers
160 Ext I/O Reg.
Internal SRAM (4096 x 8)
External SRAM (0 - 64K x 8)
$0000 - $001F
$0020 - $005F
$0060 - $00FF
$0100
$10FF
$1100
$FFFF
Memory Configuration B
Data Memory
32 Registers
64 I/O Registers
Internal SRAM (4000 x 8)
External SRAM (0 - 64K x 8)
$0000 - $001F
$0020 - $005F
$0060
$0FFF
$1000
$FFFF


PF0 - PF7
PA0 - PA7
PC0 - PC7
XTAL1
XTAL2
RESET
VCC
GND
PORTF DRIVERS
PORTA DRIVERS
PORTC DRIVERS
DATA REGISTER PORTF
DATA DIR. REG. PORTF
DATA REGISTER PORTA
DATA DIR. REG. PORTA
DATA REGISTER PORTC
DATA DIR. REG. PORTC
8-BIT DATA BUS
AVCC
AGND
AREF
ADC
INTERNAL OSCILLATOR
CALIB. OSC
OSCILLATOR
JTAG TAP
PROGRAM COUNTER
STACK POINTER
WATCHDOG TIMER
OSCILLATOR
ON-CHIP DEBUG
PROGRAM FLASH
SRAM
MCU CONTROL REGISTER
TIMING AND CONTROL
BOUNDARY-SCAN
INSTRUCTION REGISTER
GENERAL PURPOSE REGISTERS
X
Y
Z
TIMER/ COUNTERS
PEN
PROGRAMMING LOGIC
INSTRUCTION DECODER
INTERRUPT UNIT
CONTROL LINES
ALU
EEPROM
STATUS REGISTER
USART0
SPI
USART1
TWO-WIRE SERIAL INTERFACE
ANALOG COMPARATOR
DATA REGISTER PORTE
DATA DIR. REG. PORTE
DATA REGISTER PORTB
DATA DIR. REG. PORTB
DATA REGISTER PORTD
DATA DIR. REG. PORTD
DATA REG. PORTG
DATA DIR. REG. PORTG
PORTE DRIVERS
PORTB DRIVERS
PORTD DRIVERS
PORTG DRIVERS
PE0 - PE7
PB0 - PB7
PD0 - PD7
PG0 - PG4

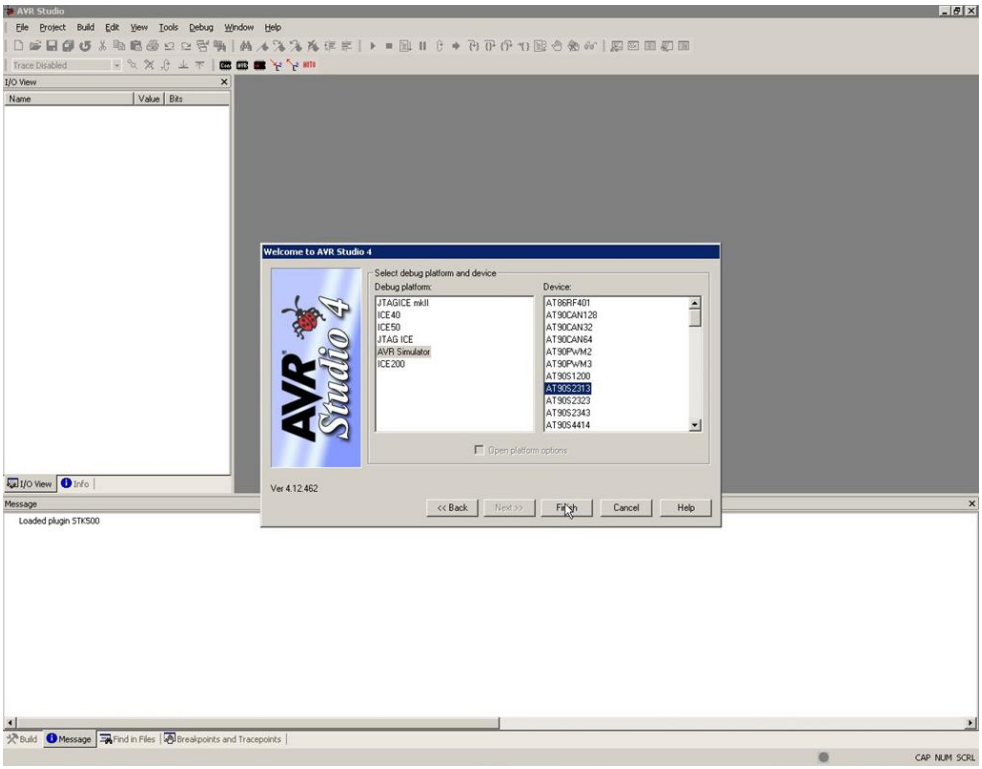
TWORZENIE PROJEKTU



Po uruchomienu programu, pojawia się następujące okienko. Aby stworzyć projekt należy kliknąć w ***„New Project”***, można również otworzyć już istniejący projekt.



Wybranie odpowiedniego projektu i jego nazwy

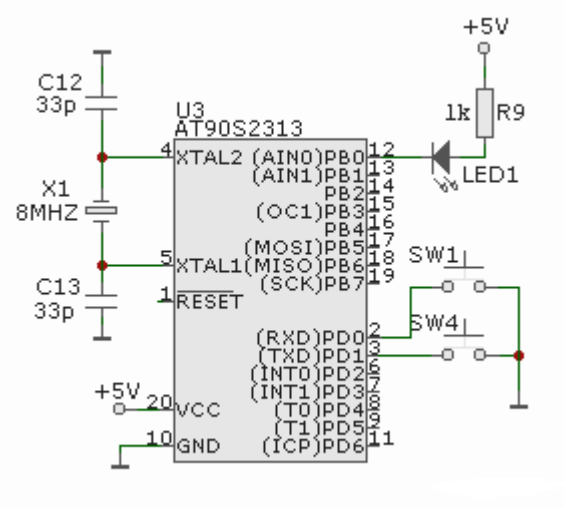


Wybór platformy do debugowania oraz urządzenia. Aby utworzyć pusty projekt należy kliknąć w ***„Finish”.***

Zobrazowanie chronologii tworzenia projektu przeprowadzimy na przykładzie zadania 1.

Zadanie 1 Zapalanie i gaszenie diody LED.

Dioda LED podłączona będzie do wyprowadzenia PB0 portu B. Do wyprowadzeń PD0 i PD1 portu D podłączone będą przełączniki, których zadaniem będzie odpowiednio włączanie i wyłączanie świecenia diody LED. Naciśnięcie przełącznika podłączonego do PD0 powinno spowodować zaświecenie diody, natomiast naciśnięcie przełącznika podłączonego do PD1 powinno spowodować zgaszenie diody.



Dioda LED1 zaświeci się gdy PB0 (pin12) będzie skonfigurowany jako wyjście i jego stan przyjmie wartość "0", to umożliwi przepływ prądu przez diodę LED1, prąd ten ograniczony jest rezystorem R9 do wartości ok. 3,5mA (zależy to również od wartości spadku napięcia na diodzie LED). Aby świecenie diody uzależnić od stanu przełączników SW1 i SW4 to PD0 (pin 2) i PD1 (pin 3) muszą być skonfigurowane jako wejścia z wejściem typu *pull-up* wymuszającym początkowy stan 1. W takim przypadku naciśnięcie jednego z przełączników spowoduje, że na odpowiednim wejściu pojawi się stan "0". Pozostałe nie wykorzystywane wyprowadzenia zarówno portu B jak i D mogą być skonfigurowane dowolnie, można je więc ustawić np. jako wyjścia.

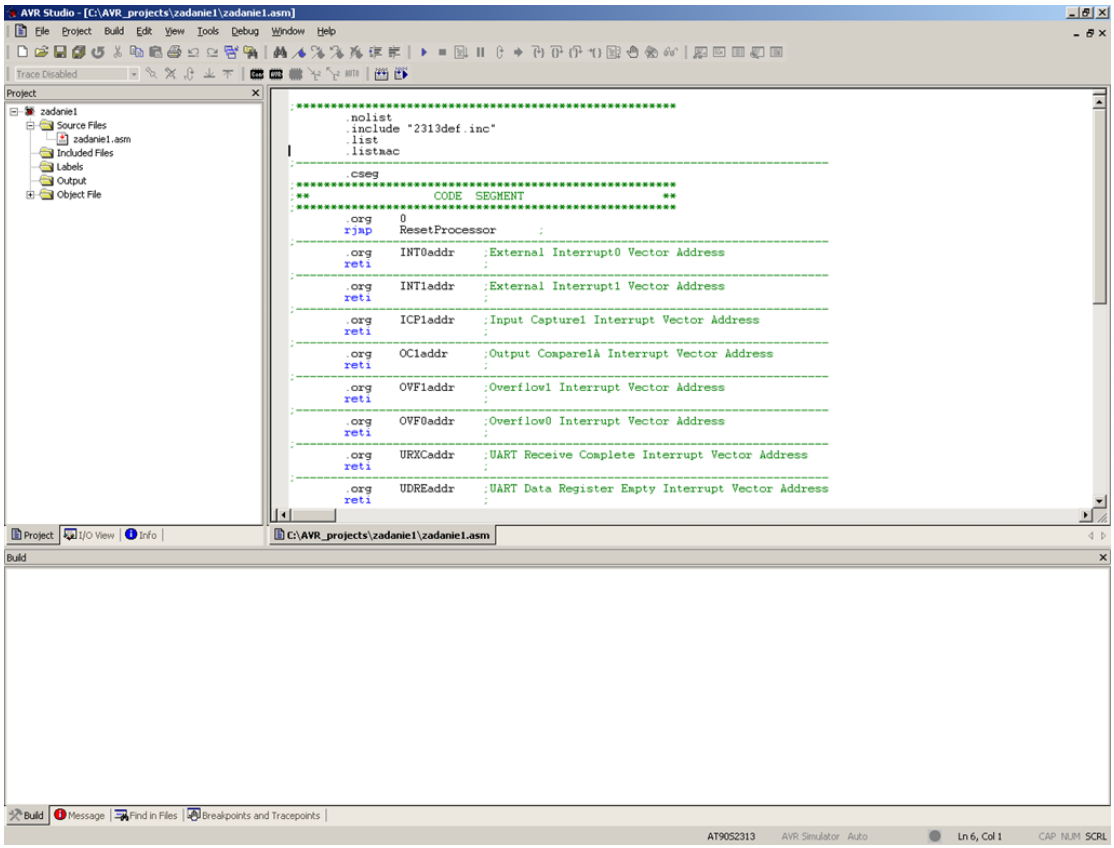
Mikrokontroler realizując swój program zawarty w pamięci programu (pamięci FLASH) operuje na zasobach zawartych wewnątrz (czasami na zewnątrz – zewnętrzna pamięć RAM, zewnętrzne porty i kontrolery) układu scalonego. Do tych zasobów zaliczamy pamięć statyczną SRAM, pamięć nieulotną EEPROM, zbiór rejestrów roboczych od R0 do R31 oraz zbiór rejestrów w przestrzeni I/O. W najprostszym wariancie *"łączność ze światem zewnętrznym"* realizowana jest poprzez dostępne porty. Zbiór możliwych do użycia portów jest zależny od modelu mikrokontrolera.

KOD ZRÓDŁOWY:

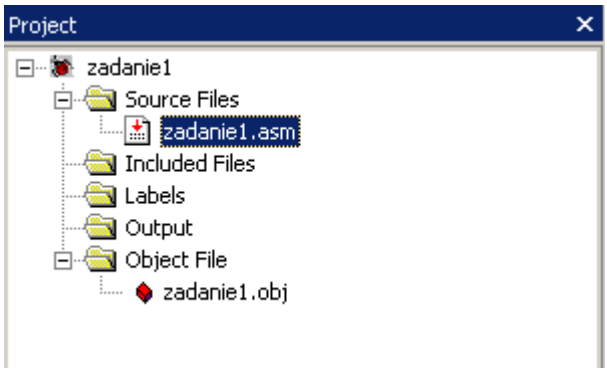
```
;***************************************************************
; Pierwszy program - zapalanie i gaszenie diody LED
;***************************************************************
; Dioda LED podlaczona bedzie do wyprowadzenia PB0 portu B.
; Do wyprowadzen PD0 i PD1 portu D podlaczone beda przelaczniki,
; ktorych zadaniem bedzie odpowiednio wlaczanie i wylaczanie
; swiecenia diody LED. Nacisniecie przelacznika podlaczonego do
; PD0 powinno spowodowac zaswiecenie diody, natomiast nacisniecie
; przelacznika podlaczonego do PD1 powinno spowodowac zgaszenie
; diody.
;***************************************************************
;*****************************************************
            .nolist
            .include "2313def.inc"
            .list
            .listmac

            .cseg

            .org   0
            rjmp   ResetProcessor      ;
            .org   INT0addr      ;External Interrupt0 Vector Address
            reti                 ;
            .org   INT1addr      ;External Interrupt1 Vector Address
            reti                 ;
            .org   ICP1addr      ;Input Capture1 Interrupt Vector Address
            reti                 ;
            .org   OC1addr       ;Output Compare1A Interrupt Vector Address
            reti                 ;
            .org   OVF1addr      ;Overflow1 Interrupt Vector Address
            reti                 ;
            .org   OVF0addr      ;Overflow0 Interrupt Vector Address
            reti                 ;
```

```
            .org   URXCaddr     ;UART Receive Complete Interrupt Vector Address
            reti                ;
            .org   UDREaddr     ;UART Data Register Empty Interrupt Vector Address
            reti                ;
            .org   UTXCaddr     ;UART Transmit Complete Interrupt Vector Address
            reti                ;
            .org   ACIaddr             ;Analog Comparator Interrupt Vector Address
            reti                ;

ResetProcessor      :                   ;
            cli                         ;
            ldi    r16,LOW(RAMEND)      ;
            out    SPL,r16                    ;
            ldi    r16,0x00             ;
            out    DDRD,r16             ; PORTD - jako wejsciowy
            ldi    r16,0xFF             ;
            out    PORTD,r16            ; PORTD - wejscia PULL-UP
            ldi    r16,0xFF             ;
            out    PORTB,r16            ; PORTB - jako wyjscie
            out    DDRB,r16             ; PORTB - wyjscie w stanie wysokim
Main_0      :                           ; poczatek petli
            in     r16,PIND             ;  czy jest przycisnięty przycisk
            andi   r16,0x03             ;  na pinie PD0
            cpi    r16,0x02             ;
            breq   Main_1               ;  tak: skok do Main_1
            cpi    r16,0x01             ;  czy na pinie PD1
            breq   Main_2               ;  tak: skok do Main_2
            rjmp   Main_0               ;  powrot do petli
Main_1      :                           ;
            cbi    PORTB,0              ;  PORTB.0 = 0 ==> LED swieci
            rjmp   Main_0               ;
Main_2      :                           ;
            sbi    PORTB,0                  ;  PORTB.0 = 1 ==> LED nie swieci
            rjmp   Main_0               ;
                                        ;  koniec petli
;-------------------------------------------------------------------------------
.exit
```

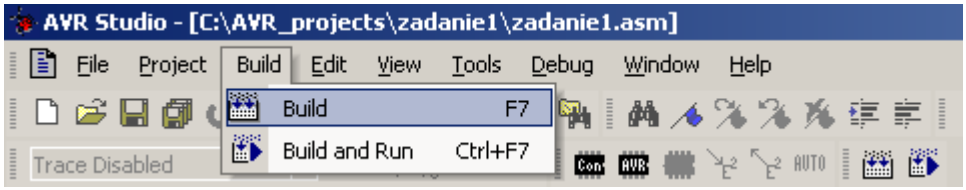
Aby skompilować taki kod w AVRStudio należy stworzyć nowy projekt



Po wpisaniu kodu w okno edycji kodu.

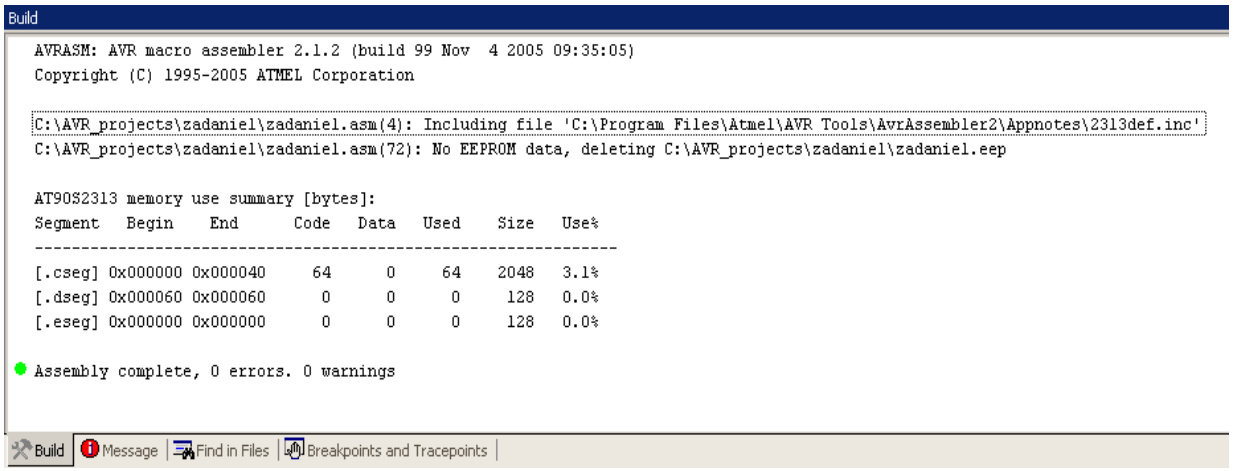


W początkowej fazie pisania programu nasz projekt składa się jedynie z pliku z kodem źródłowym oraz plik *.obj.

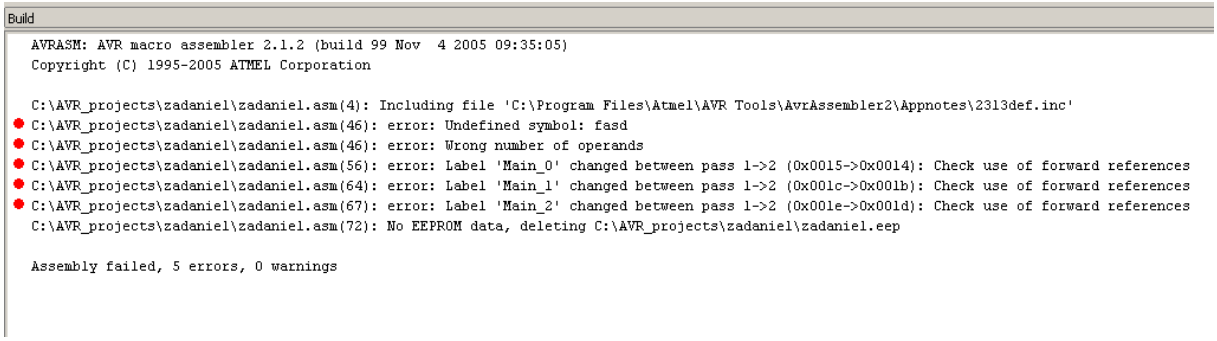


Aby zasemblować kod należy wybrać z menu głównego Build->Build. Opcjonalnie można użyć skrótu F7.

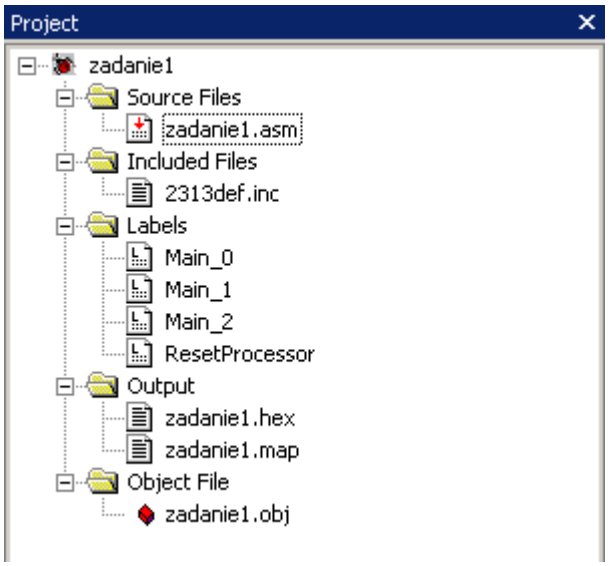
Można również użyć ikony na pasku narzędzi.



Po poprawnej asemblacji w oknie Build wyświetli się log z procesu. Jeżeli wszystko się udało zostaniemy i tym poinformowani stosownym komunikatem.



W przypadku błędów w kodzie programu, w okienku Build pojawią się informacje na ich temat pomagając
w lokalizacji błędów. Po dwukrotnym kliknięciu na daną pozycję w liście błędów kursor zostanie automatycznie przeniesiony w miejsce wystąpienie błędu.

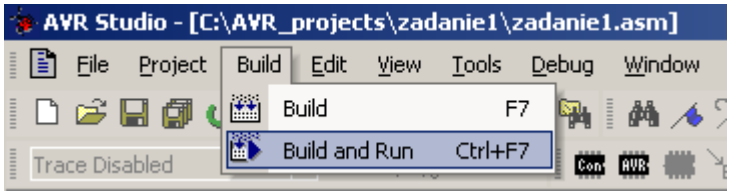


Po zbudowaniu programu do naszego projektu zostaną dodane dodatkowe pozycje:

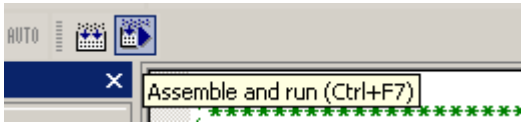
lista zaincludowanych plików

lista etykiet w programie

lista plików wynikowych powstałych w skutek asemblacji.



Aby zasemblować i uruchomić program należy wybrać menu Build-> Build and Run. Jak poprzednio można posłużyć się skrótem klawiszowym Ctrl+F7



Program można również zasemblować i uruchomić klikając w odpowiednią pozycję w pasku narzędzi.

URUCHAMIANIE PROGRAMU

Proces uruchamiania programu zostanie pokazany z użyciem programu przykładowego nr 2.

Zadanie 2

Zadaniem programu jest zapalanie po naciśnięciu przycisku diody LED na określony czas trwania . Przycisk SW1 jest przyłączony do portu D (pin PD0), dioda LED jest podłączone do portu B (pin PB0). Mikrokontroler będzie po naciśnięciu przycisku zapalać diodę LED na określony czas, po upływie którego dioda zostanie zgaszona.
Drganie zestyków nie jest eliminowane drogą programową ani sprzętową, gdyż jego wpływ na działanie programu jest nieznaczący. Dioda LED1 zaświeci się gdy PB0 (pin12 AT90S2313) będzie skonfigurowany jako wyjście i jego stan przyjmie wartość "0", to umożliwi przepływ prądu przez diodę LED. Aby zaświecenie diody uzależnić od stanu przełącznika SW1 to PD0 (pin 2) musi być skonfigurowany jako wejście z wejściem typu pull-up wymuszającym początkowy stan 1. W takim przypadku naciśnięcie przełącznika spowoduje, że na wejściu PD0 pojawi się stan "0". Pozostałe nie wykorzystywane wyprowadzenia zarówno portu B jak i D mogą być skonfigurowane dowolnie, można je więc ustawić np. jako wyjścia.

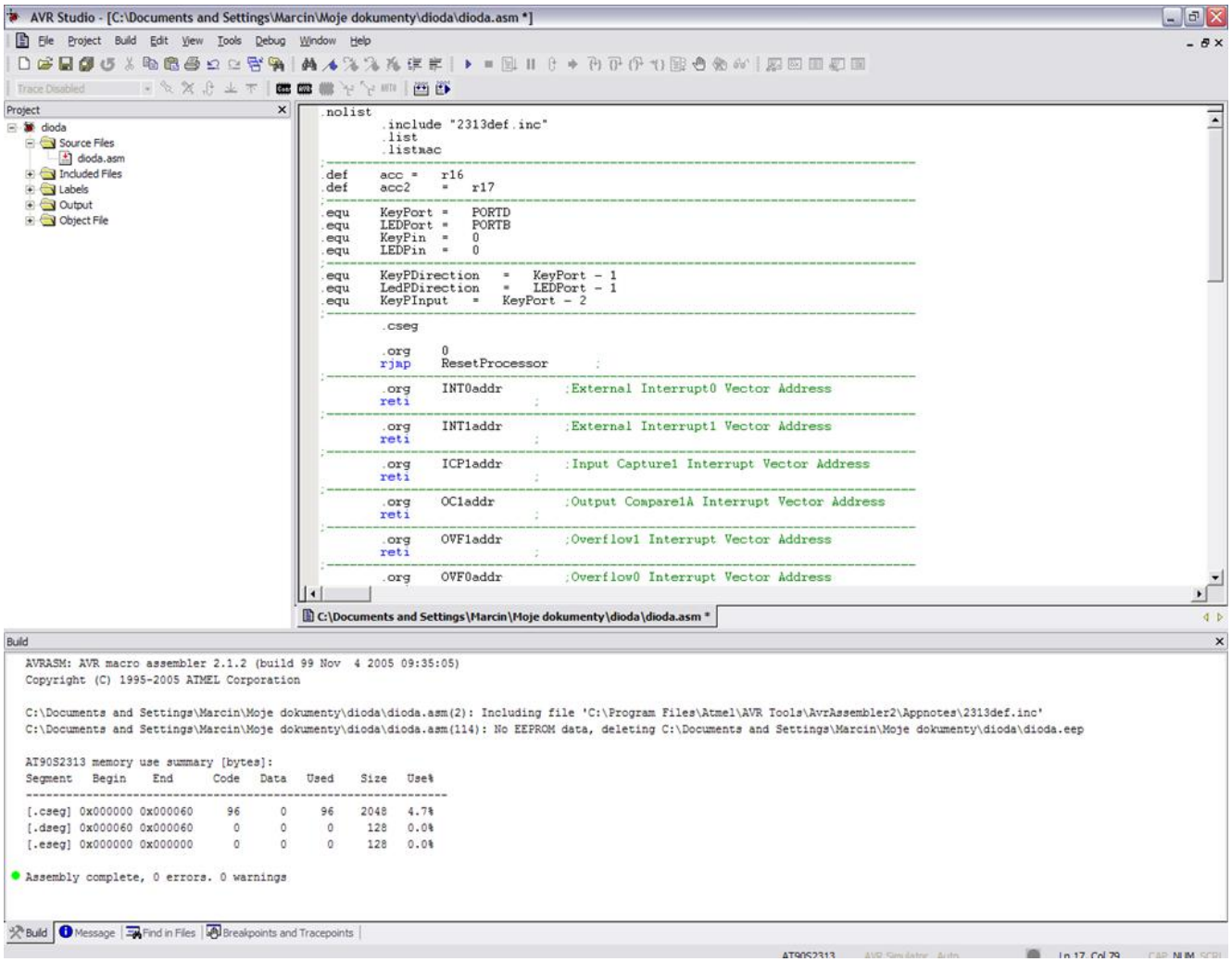
Kod programu:

```
;---------------------------------------------------------------------------
;Zadaniem programu jest zapalanie po nacisnieciu przycisku diody LED
;na okreslony czas trwania. Przycisk SW1 jest przylaczony
;do portu D (pin PD0), dioda LED jest podlaczone do portu B (pin PB0).
;Mikrokontroler bedzie po nacisnieciu przycisku zapalac diode LED
;na okreslony czas, po uplywie ktorego dioda zostanie zgaszona.
;---------------------------------------------------------------------------

.nolist
.include "2313def.inc"
.list
.listmac
;---------------------------------------------------------------------------
.def   acc    =      r16
.def   acc2   =      r17
;---------------------------------------------------------------------------
.equ   KeyPort       =      PORTD
.equ   LEDPort       =      PORTB
.equ   KeyPin =      0
.equ   LEDPin =      0
;---------------------------------------------------------------------------
.equ   KeyPDirection=      KeyPort - 1
.equ   LedPDirection=      LEDPort - 1
.equ   KeyPInput     =      KeyPort - 2
;---------------------------------------------------------------------------
.cseg

.org   0
rjmp   ResetProcessor            ;
;---------------------------------------------------------------------------
```

```
.org   INT0addr           ;External Interrupt0 Vector Address
reti                      ;
;-------------------------------------------------------------------------------
.org   INT1addr           ;External Interrupt1 Vector Address
reti                      ;
;-------------------------------------------------------------------------------
.org   ICP1addr           ;Input Capture1 Interrupt Vector Address
reti                      ;
;-------------------------------------------------------------------------------
.org   OC1addr                  ;Output Compare1A Interrupt Vector Address
reti                      ;
;-------------------------------------------------------------------------------
.org   OVF1addr           ;Overflow1 Interrupt Vector Address
reti                      ;
;-------------------------------------------------------------------------------
.org   OVF0addr     ;Overflow0 Interrupt Vector Address
reti                      ;
;-------------------------------------------------------------------------------
.org   URXCaddr     ;UART Receive Complete Interrupt Vector Address
reti                      ;
;-------------------------------------------------------------------------------
.org   UDREaddr     ;UART Data Register Empty Interrupt Vector Address
reti                      ;
;-------------------------------------------------------------------------------
.org   UTXCaddr     ;UART Transmit Complete Interrupt Vector Address
reti                      ;
;-------------------------------------------------------------------------------
.org   ACIaddr            ;Analog Comparator Interrupt Vector Address
reti                      ;
;-------------------------------------------------------------------------------
Delay:                                ;odczekanie pewnego czasu
;****************                     ;
            ldi    acc2,0             ;
            ldi    acc,0              ;
Del_0       :                         ;
            inc    acc                ;
            brne   Del_0              ;
            inc    acc2               ;
            brne   Del_0              ;
            ret                       ;
;-------------------------------------------------------------------------------
ResetProcessor      :                       ;
            cli                       ;
            ldi    acc,LOW(RAMEND)            ;
            out    SPL,acc                    ;
            cbi    KeyPDirection,KeyPin       ; KeyPin - jako wejscia z PULL=UP
            sbi    KeyPort,KeyPin             ;
            sbi    LedPDirection,LEDPin       ; LEDPin - jako wyjscie
            sbi    LEDPort,LEDPin             ; LEDPin - wyjscie w stanie wysokim
Main_0      :                         ; poczatek petli
            in     acc,KeyPInput      ;  acc = stan portu KeyPort
            andi   acc,1<<KeyPin      ;  pozostawienie w acc stanu
                                      ;  jednego wybranego bitu
                                      ;  w wyniku opeacji AND rejest acc
                                      ;  bedzie zawieral na pozycji KeyPin
                                      ;  bit=1 jezeli nie jest nacisniety
                                      ;  przycisk oraz caly rejestr bedzie
                                      ;  wyzerowany jezeli przycisk jest
                                      ;  nacisniety
                                      ;  wskaznik Z=1 jezeli acc=0 => przycisk nacisniety
                                      ;   wskaznik Z=0 jezeli acc <> 0 => przycisk nie ;
                                      ;nacisniety
            brne   Main_0             ;  skok do Main_0 (Z=0)
            cbi    LEDPort,LEDPin           ;  LEDPin = 0 ==> LED swieci
            rcall  Delay              ;  odczekanie pewnego czasu
            rcall  Delay              ;  odczekanie pewnego czasu
            rcall  Delay              ;  odczekanie pewnego czasu
            rcall  Delay              ;  odczekanie pewnego czasu
```

```
                rcall  Delay                ;  odczekanie pewnego czasu
                rcall  Delay                ;  odczekanie pewnego czasu
                rcall  Delay                ;  odczekanie pewnego czasu
                rcall  Delay                ;  odczekanie pewnego czasu
                rcall  Delay                ;  odczekanie pewnego czasu
                rcall  Delay                ;  odczekanie pewnego czasu
                rcall  Delay                ;  odczekanie pewnego czasu
                rcall  Delay                ;  odczekanie pewnego czasu
                rcall  Delay                ;  odczekanie pewnego czasu
                rcall  Delay                ;  odczekanie pewnego czasu
                sbi    LEDPort,LEDPin       ;  LEDPin = 1 ==> LED nie swieci
Main_1          :                           ;
                in     acc,KeyPInput        ;  analogicznie jak wyzej odczekanie
                andi   acc,1<<KeyPin        ;  na puszczenie klawisza
                breq   Main_1               ;
                rjmp   Main_0               ;
                                            ; koniec petli
;-----------------------------------------------------------------------------
.exit
```

Okno główne programu:



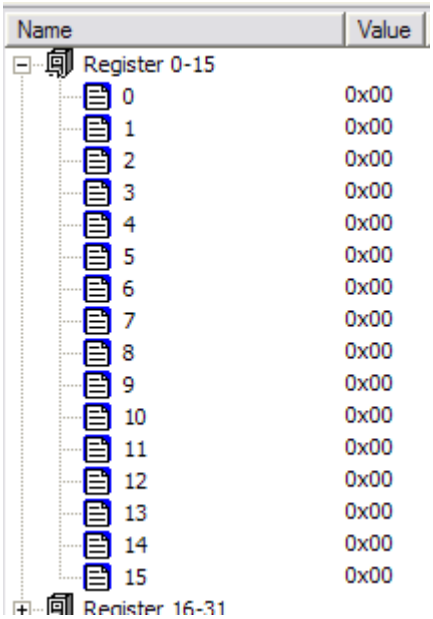
Po pomyślnej asemblacji programu można przystąpić do debugowania. AVR Studio posiada bardzo wygodny i łatwy w obsłudze debugger.

Debugowanie rozpoczynamy przyciskiem 

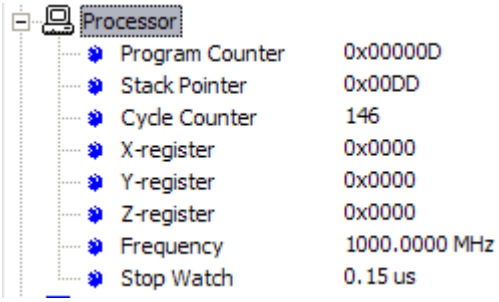
Debugger zatrzymuje się na pierwszej instrukcji: 

```
.org    0
rjmp    ResetProcessor
```

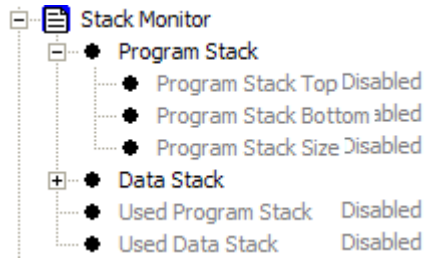
W procesie debugowania bardzo przydatny jest I/O View , który pozwala na sprawdzenie stanów poszczególnych rejestrów procesora w dowolnym momencie:



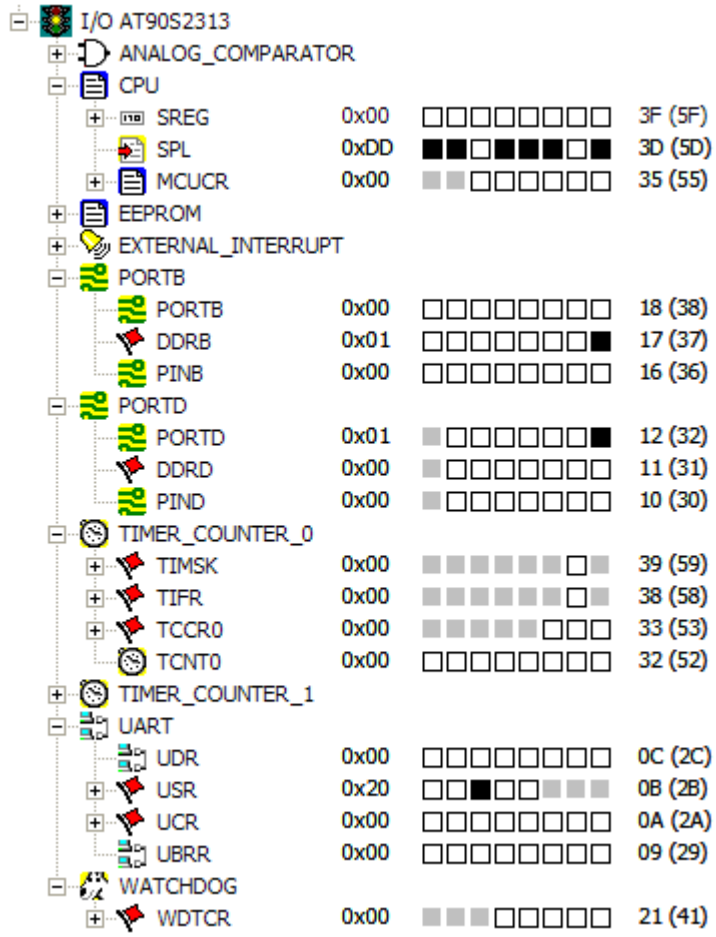
Możemy również sprawdzić stan procesora poprzez podejrzenie licznika rozkazów, czy wskaźnika stosu:



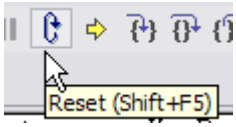
Dysponujemy także podglądam stosu:



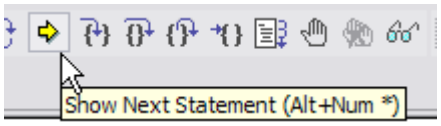
Najbardziej przydatnym elementem debugger jest podgląd portów wejścia / wyjścia. Pozwala on po wstrzymaniu wykonania programu, obejrzeć zawartość lub dowolnie zmienić zawartość rejestrów (np. portu A lub D).



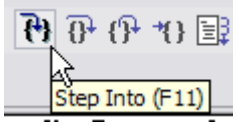
Do manipulowania procesem debugowania służą następujące przyciski:



Reset służy do przerwania programu i uruchomienia go od nowa w dowolnym momencie.



Przycisk Show Next Statement służy do przeniesienia kursora do aktualnie wykonywanej instrukcji programu.



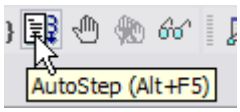
Step Into, Step Over, Step Out oraz Run To Cursor służą kolejno do:

-przejścia do następnej instrukcji wraz z ewentualnym wejściem do podprogramu

-przejścia do następnej instrukcji bez wchodzenia do podprogramu

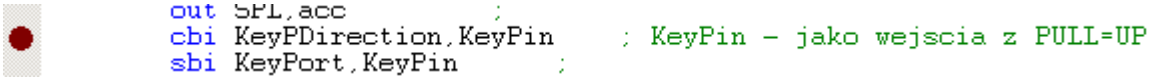
-wyjścia z podprogramu

-uruchomienia programu i zatrzymaniu go w miejscu, w którym znajduje się kursor

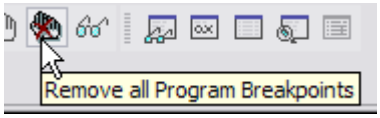


Auto Step automatycznie przechodzi do kolejnych instrukcji aż do zakończenia programu

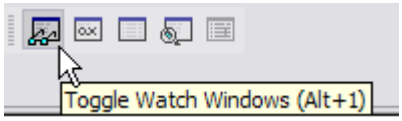
Toggle Breakpoint umożliwia wstawienie BreakPointa:

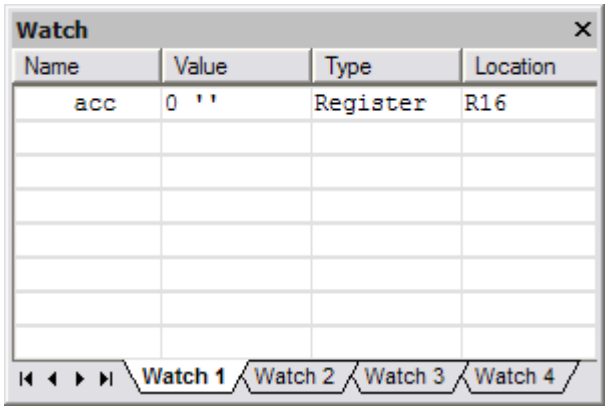


Mamy możliwość usunięcia wszystkich breakpointów w programie:

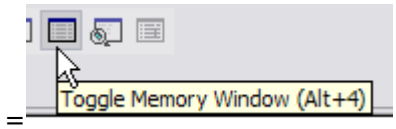


Przyciskiem Toggle Watch Windows włączamy okno umożliwiające śledzenie wartości poszczególnych zmiennych, których nazwy wpiszemy do niego.

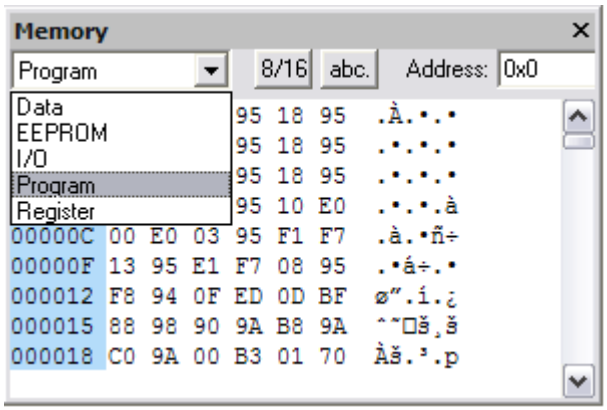




Przydatną opcją jest również memory window umożliwiające podgląd zawartości pamięci (Programu, ROM, I/O itp.)

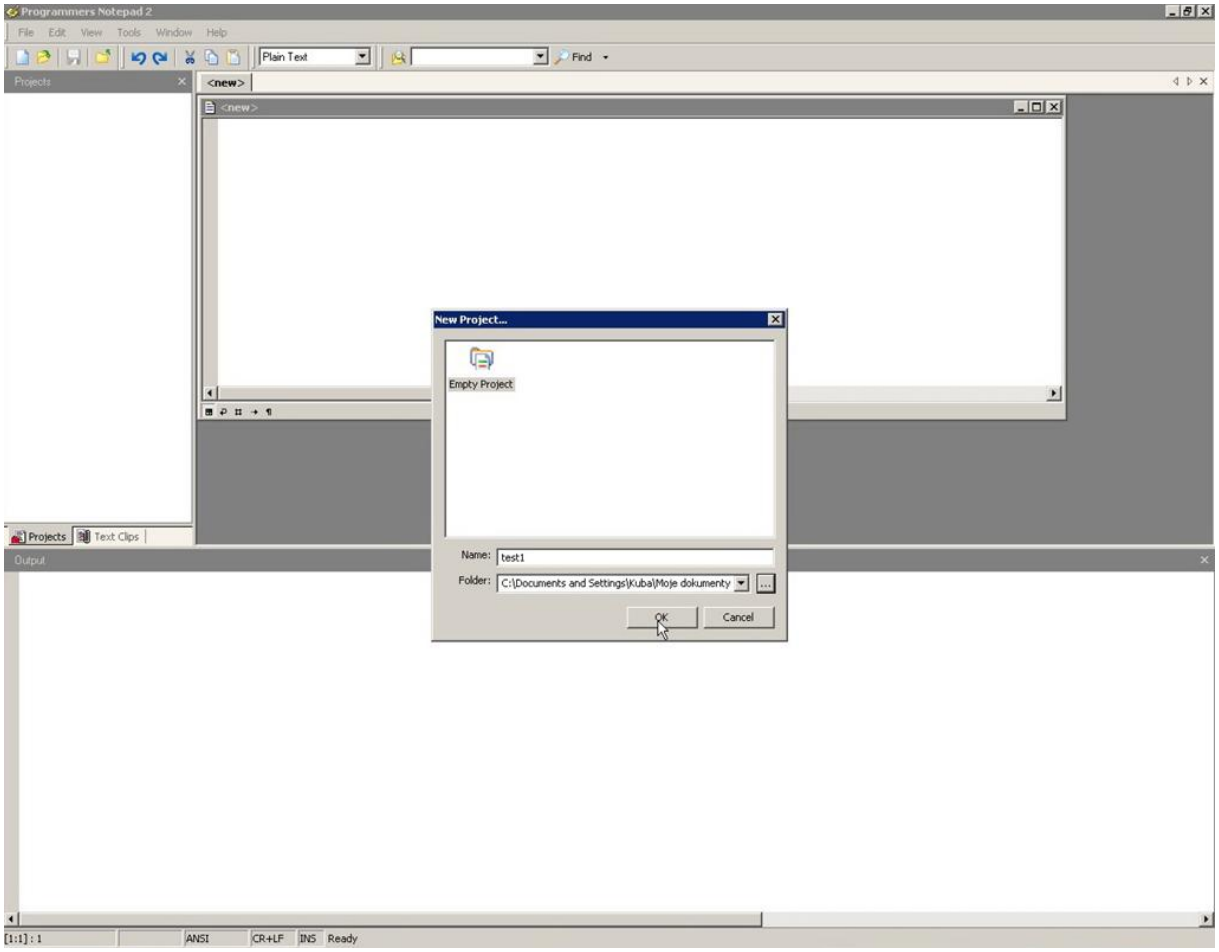


=

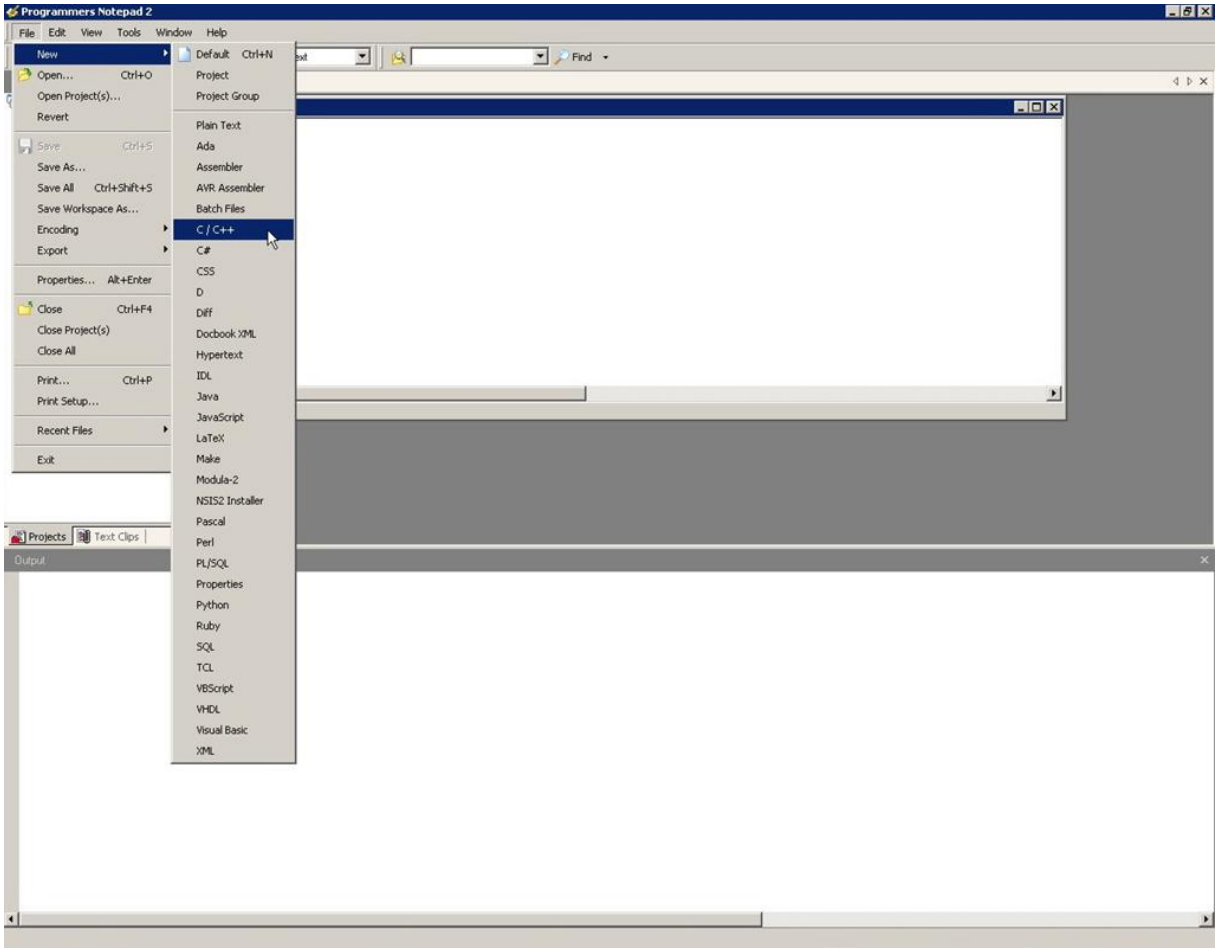


TWORZENIE PROJEKTU

Po pomyślnej instalacji możemy sprawdzić zainstalowane składniki. Jednym z nich jest Programmers Notepad, który jest edytorem, w którym będziemy pisać i kompilować nasze programy. Jego wygląd jest przedstawiony na poniższym zdjęciu:

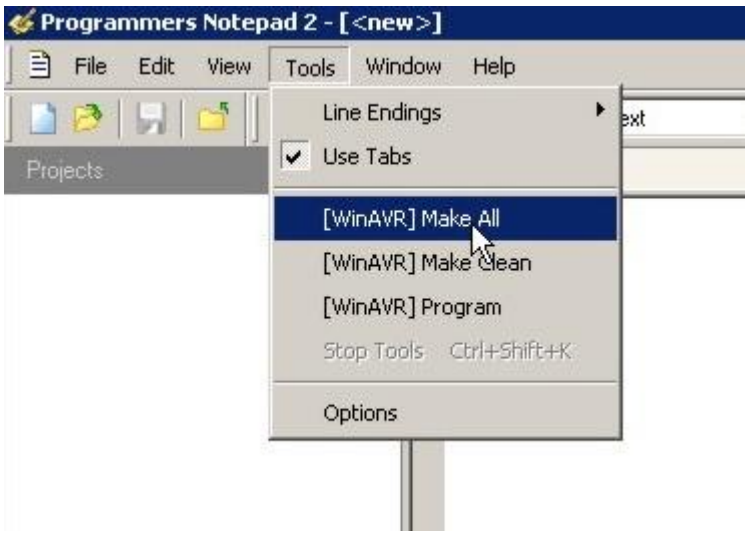


Aby utworzyć nowy projekt, należy kliknąć w: ***„File->New->Project”*** powinno pojawić się powyższe okienko. Trzeba wpisać nawę projektu i folder w którym będzie się on znajdował. Następnie należy kliknąć w ***„OK”.***



Wybranie kolorowania składni dla języka C/C++.

Jeżeli masz jakieś podstawy języka C to nie powinno być problemów w pisaniu programów i ich kompilacją. Po napisaniu przykładowego kompilujemy go wybierając z menu ***„Tools >MakeAll”.***



Musimy pamiętać, aby w katalogu w którym zapisujemy nasz plik z kodem źródłowym (rozszerzenie *.c) umieścić plik makefile. Jeżeli w kodzie nie było błędów i plik makefile został skonfigurowany prawidłowo w dolnym okienku ***„Output"*** pojawi się informacja, że kompilacja przebiegła prawidłowo, oraz inne dodatkowe informacje. Jeżeli kompilacja przebiegła pomyślnie, i w pliku makefile wybraliśmy odpowiedni programator to możemy bezpośrednio z programu zaprogramować nasz mikrokontroler wybierając z menu ***„Tools >Program".*** Automatycznie zostaje wykasowana cała zawartość pamięci Flash i EEPROM i wgrany nasz plik *.hex wygenerowany podczas kompilacji.

MAKEFILE

Kompilacja programu w języku C składa się z kilku faz. Pierwszą z nich jest wygenerowanie tzw. pliku pośredniego (object file), zazwyczaj z rozszerzeniem ".o". Następnie pliki pośrednie modułów i głównego programu są łączone za pomocą konsolidatora (linker) w plik wykonywalny (.elf). Dla prostych programów te dwie operacje mogą być wykonane w jednym kroku. Jednak plik .elf nie nadaje się do bezpośredniego zaprogramowania mikrokontrolera (na dzień dzisiejszy nie są znane programatory mikrokontrolerów "rozumiejące" ten format plików) dlatego należy jeszcze z niego "wydobyć" dane w formacie obsługiwanym przez popularne programatory np. Intel HEX. *Make* jest programem podejmującym decyzję, które części dużego programu muszą zostać zrekompilowane i wywołującym polecenia służące do tego. Aby korzystać z programu *make* potrzebujemy pliku zawierającego informacje o tym jak należy postępować w przypadku zmian w plikach źródłowych i zależnościach między nimi. Domyślnie ten plik nosi nazwę *makefile*. Gdy zmieni się zawartość któregokolwiek pliku źródłowego musi on zostać zrekompilowany, jeżeli zmieni się zawartość któregoś z plików nagłówkowych bezpiecznie jest zrekompilować wszystkie źródła zawierające ten plik. Kiedy którykolwiek z plików wynikowych (ang. object files; np. .o) się zmieni wtedy trzeba ponownie skonsolidować całość. Korzystanie z *make* sprowadza się wiec do stworzenia pliku *makefile*, który pokieruje procesem kompilacji naszego programu.

Najczęściej używane opcje programu make:

```
-d  włącza tryb szczegółowego śledzenia

-f  plik_sterujacy umożliwia stosowanie innych niż standardowe nazw plików sterujących

-i  powoduje ignorowanie błędów kompilacji (stosować z ostrożnością!)
```

```
-n  powoduje wypisanie poleceń na ekran zamiast ich wykonania

-p  powoduje wypisanie makrodefinicji i reguł transformacji

-s  wyłącza wypisywanie treści polecenia przed jego wykonaniem
```

Opcje można ze sobą łączyć. Np.: polecenie *{make -np}* powoduje wypisanie wszystkich reguł i makrodefinicji oraz ciągu poleceń jakie powinne być wykonane, aby uzyskać żądany cel. Jest to pomocne w sytuacji, gdy chcemy sprawdzić poprawność definicji zawartych w pliku sterującym bez uruchamiania długotrwałej kompilacji wielu plików.

Plik sterujący (makefile)

Plik sterujący zawiera definicje relacji zależności, które mówią w jaki sposób i z jakich elementów należy stworzyć cel (program, bibliotekę, lub plik obiektowy) i wskazują pliki, których zmiany implikują wykonanie powtórnej kompilacji poszczególnych celów. Plik sterujący może również zawierać zdefiniowane przez programistę reguły transformacji. W pliku makefile znakiem komentarza jest znak # (hash) umieszczony na początku linii.

PROJEKTY W C

Zadanie 3

```
#include <avr/io.h>
#define PORTK PORTB
#define PINK PINB
#define DDRK DDRB
int test()
{
	if(bit_is_clear(PINK,0)) return 0;
	if(bit_is_clear(PINK,1)) return 1;
	if(bit_is_clear(PINK,2)) return 2;
	if(bit_is_clear(PINK,3)) return 3;
	return 5;
}

int main(void)
{
	DDRK=0xF0;
	while(1)
	{
		PORTK=0xF0;
		switch(test())
		{
		case 0: PORTK&= ~0x10; break;
		case 1: PORTK&= ~0x20; break;
		case 2: PORTK&= ~0x40; break;
		case 3: PORTK&= ~0x80; break;
		default: PORTK&= ~0x00; break;
		}
	}
}
```

Powyższy program prezentuje klawiaturę zbudowaną na jednym z portów poprzez przypięcie do jego 4 najmłodszych bitów przycisków, które po wciśnięciu zwierają do masy, a do 4 starszych

bitów diod podpiętych przez opornik do zasilania. Na samym początku następuje definicja portu dla klawiatury, czyni to kod podatniejszym na zmiany, jeśli zechcemy przenieść klawiaturę na inny port np. D wystarczy zmienić zapis na początku programu. Następnie zdefiniowana jest funkcja test która testuje poszczególne piny portu klawiatury i zwraca odpowiednią liczbę w zależności od tego który został naciśnięty. W funkcji main ustawiamy wartość rejestru DDRK na 0xF0 co definiuje nam końcówki 4-7 jako wyjście a 0-3 jako wejścia. Po tej operacji rozpoczyna się pętla programu. Realizuje ona operację zapalania poszczególnych diod w zależności od wartości zwróconej przez funkcję test. Następuje to poprzez przypisanie odpowiedniej wartości do portu klawiatury. Podczas działania programu tylko jedna dioda na raz może świecić, dzieje się tak przez funkcję test która po znalezieniu naciśniętego przycisku zwraca jego numer i kończy swe działanie. Najwyższy priorytet mają bity najmłodsze.

Zadanie 4

```
// Wyświetlenie prostej animacji przy pomocy 8 diod LED podłączonych do

// portu B procesora


#include <avr/io.h>                        // dostęp do rejestrów

int main( void )
{
  DDRB=0xFF;                               // użyj wszystkich linii PB jako wyjścia
  PORTB=0xF8;

  TCNT0 = 0;                              // wartość początkowa zegara
  TCCR0 = _BV(CS00)|_BV(CS02);   // czestotliwosc dzielimy przez 1024

  int anim=0x07;                  //00000111 zapalamy 3 ostatnie diody
  while(1)
  {
    while(bit_is_set(PORTB,7))    //ruch w lewo
       {
              while(TCNT0!=0xFF);
              anim*=2;
              PORTB= ~anim;


       }
    while(bit_is_set(PORTB,0))    //ruch w prawo
       {
              while(TCNT0!=0xFF);
              anim/=2;
              PORTB= ~anim;
       }
  }
}
```

Program wyświetla prostą animację trzech przesuwających się punktów, do wyświetlania użyjemy portu B. Na początek za pomocą portu DDRB ustawiamy wszystkie linie na wyjścia a potem przypisujemy początkową wartość do portu B. Inicjalizujemy również zegar, który posłuży nam do opóźniania animacji, za pomocą portu TCNT0. Ustawiamy częstotliwość z jaką będziemy dzielić taktowanie procesora portem TCCR0. Następnie deklarujemy zmienną pomocniczą anim i nadajemy jej początkową wartość. W pętli programu znajdują się dwie inne pętle realizujące

przesuwanie animacji w lewo oraz w prawo. Dopóki animacja nie dotarła do końca linii czekają na przepełnienie zegara zmieniają zmienną pomocniczą i wyświetlają ją. Następnie przechodzą do ruchu w przeciwną stronę.

Zadanie 5

```
#include <avr/io.h>

#include <avr/interrupt.h>
#include <avr/eeprom.h>

uint8_t zmienna __attribute__((section(".eeprom"))) = 0;
uint8_t wartosc;

SIGNAL (SIG_INTERRUPT0)
{
  wartosc = PINB;
  eeprom_write_byte(&zmienna,wartosc);
}

SIGNAL (SIG_INTERRUPT1)
{
  wartosc=eeprom_read_byte(&zmienna);
  PORTD = wartosc;
}

int main(void)                // program główny
{
  DDRD = 0xFF;
  DDRB = 0x00;

  GIMSK = _BV(INT0)|_BV(INT1);
  MCUCR = _BV(ISC01)|_BV(ISC11);

  sei();                // włącz obsługę przerwań

  while(1);                // pętla nieskończona
}
```

Powyższy program ilustruje użycie przerwań oraz dostępu do pamięci eeprom. Załóżmy że mamy dwa urządzenia. Pierwsze urządzenie podpięte do portu B oraz końcówki int0 generuje daną i wysyła przerwanie gdy dana ma zostać zapisana w pamięci. Urządzenie drugie przypięte jest do portu D i końcówki int1 wysyła przerwanie gdy chce odczytać daną z pamięci. Wracając do programu… po załączeniu odpowiednich plików potrzebnych do skompilowania projektu, następuje deklaracja dwóch zmiennych typu int 8 bitowych, pierwsza dostępna w sekcji eepromu druga pomocnicza. Następnie widzimy definicję dwóch procedur wykonywanych gdy pojawi się przerwanie na danej końcówce. Dla int0 odczytujemy wartość z portu B i zapisujemy ją do pamięci eeprom. Dla int1 odczytujemy wartość z eepromu i wysyłamy ją na portD. W procedurze main ustawiamy odpowiednio wyjścia portów; port D jako wyjście, port B jako wejście. Następnie za pomocą rejestru GIMSK włączamy obsługę przerwań int0 oraz int1, a przy pomocy MCUCR

ustawiamy generowanie przerwań opadającym zboczem. Po włączeniu obsługi przerwań za pomocą procedury sei następuje nieskończona pętla programu oczekująca na ich nadejście.

Zadanie 6

```
#include <avr/io.h>
#include <avr/interrupt.h>
#include <avr/signal.h>

uint8_t led;

SIGNAL (SIG_OVERFLOW1)
{
  PORTB = ~led++;
  TCNT1 = 0xFF00;
}

int main(void)
{
  DDRB = 0xFF;
  TIMSK = _BV(TOIE1);
  TCNT1 = 0xFF00;
  TCCR1A = 0x00;
  TCCR1B = _BV(CS10)|_BV(CS12);

  sei();
  while(1);
}
```

Powyższy prosty program ilustruje użycie zegara, będziemy wyświetlać ilość przepełnień zegara na diodach podpiętych do portu B. W tym celu deklarujemy 8 bitową zmienną typu **int** która będzie przechowywać nam ilość przepełnień a następnie definiujemy procedurę obsługującą przerwanie przepełnienia. Zwiększa ona naszą zmienną neguje ją by dało się wyświetlać, oraz ustawia początkowy stan zegara (bo nie chce nam się tak długo czekać ;). W procedurze main ustawiamy linie portu B jako wyjścia a następnie zajmujemy się rejestrami zegara. Za pomocą TIMSK włączamy obsługę przerwań zegara, TCNT1 ustawia nam wartość początkową, trzeba jeszcze włączyć zegar w tryb czasomierza za pomocą rejestru TCCR1A i ustawić ilość taktów która będzie powodować inkrementacje licznika. Po włączeniu przerwań za pomocą procedury sei następuje pętla programu.

Treść ćwiczenia:

1. Napisanie prostego programu zadanego przez prowadzącego w asemblerze i jego uruchomienie w symulatorze AVR Studio.
2. Napisanie prostego programu w C i jego uruchomienie w symulatorze.
3. Porównanie obydwu programów.

BIBLIOGRAFIA

1. Jarosław Doliński - "Mikrokontrolery AVR w praktyce"
2. Andrzej Pawluczuk - "Sztuka programowania mikrokontrolerów. AVR - podstawy"
3. Piotr Górecki - "Mikrokontrolery dla początkujących"
4. http://www.itee.uq.edu.au/~cse/_atmel/AVR_Studio_Tutorial/
5. http://winavr.scienceprog.com/avr-gcc-tutorial/
6. http://imakeprojects.com/Projects/avr-tutorial/
7. http://www.atmel.com/products/avr/
8. http://www.avrfreaks.net/
9. http://pl.wikipedia.org/wiki/Atmel_AVR
10. http://www.elportal.pl/ea/asm_avr.html
11. Jakub Jankowski, Marcin Kania, Mariusz Macheta, Łukasz Strzelecki – Opracowanie na temat mikrokontrolery AVR”

Załącznik

1.Lista rozkazów AVR

Mnemonik	Operandy	Opis	Operacja	Rejestr statusu SREG								Liczba	
				I	T	H	S	V	N	Z	C	cykli	słów
Operacje arytmetyczne i logiczne													
ADD	Rd, Rs	Dodaj zawartość dwóch rejestrów	Rd<-Rd+Rs			↕	↕	↕	↕	↕	↕	1	1
ADC	Rd, Rs	Dodaj zawartość dwóch rejestrów z C	Rd<-Rd+Rs+C			↕	↕	↕	↕	↕	↕	1	1
ADIW	RR, K6	Dodaj bezpośrednio stałą do słowa	RRh: RR1 <-RRh:RRl+K6				↕	↕	↕	↕	↕	2	1
SUB	Rd, Rs	Odejmij zawartość dwóch rejestrów	Rd<-Rd-Rs			↕	↕	↕	↕	↕	↕	1	1
SUBI	Rh, K8	Odejmij stałą od rejestru	Rh<-Rh-K8			↕	↕	↕	↕	↕	↕	1	1
SBIW	RR, K6	Odejmij bezpośrednio stałą do słowa	RRh: RR1 <-RRh:RRl-K6				↕	↕	↕	↕	↕	1	1
SBC	Rd, Rs	Odejmij zawartość dwóch rejestrów z C	Rd<-Rd-Rs-C			↕	↕	↕	↕	↕	↕	1	1
SBCI	Rh, K8	Odejmij stałą wraz z C od rejestru	Rh<-Rh-K8-C			↕	↕	↕	↕	↕	↕	1	1
AND	Rd, Rs	Iloczyn logiczny rejestrów	Rd<-RdARs				↕	0	↕	↕		1	1
ANDI	Rh, K8	Iloczyn logiczny rejestru ze stałą	Rh<-RhAK8				↕	0	↕	↕		1	1
OR	Rd, Rs	Suma logiczna rejestrów	Rd<-RdvRs				↕	0	↕	↕		1	1
ORI	Rh, K8	Suma logiczna rejestru ze stałą	Rh<-RhvK8				↕	0	↕	↕		1	1
EOR	Rd, Rs	Suma modulo 2 dwóch rejestrów	Rd<-Rd0Rs				↕	0	↕	↕		1	1
COM	Rd	Uzupełnienie do jedności (Ul)	Rd<-$FF-Rd				↕	0	↕	↕	1	1	1

NEG	Rd	Uzupełnienie do dwóch (U2)	Rd<-$00-Rd			↕	↕	↕	↕	↕	↕	1	1
SBR	Rh, K8	Ustaw bit(y) w rejestrze	Rh<-RhvK8				↕	0	↕	↕		1	1
CBR	Rh, K8	Skasuj bit(y) w rejestrze	Rh<-RhA($FF-K8)				↕	0	↕	↕		1	1
INC	Rd	Inkrementuj rejestr	Rd<-Rd+1				↕	2	↕	↕		1	1
DEC	Rd	Dekrementuj rejestr	Rd<-Rd-1				↕	2	↕	↕		1	1
TST	Rd	Sprawdź czy zero lub minus	RdARd				↕	0	↕	↕		1	1
CLR	Rd	Zeruj rejestr	Rd<-Rd0Rd				0	0	0	1		1	1
SER	Rh	Ustaw rejestr	Rh<-$FF									1	1

Operacje skoków

RJMP	al2	Skok względny	PC<-PC+al2+l									2	1
IJMP		Skok względny określony zawartością Z	PC<-Z									2	1
RCALL	al2	Względne wywołanie podprogramu	PC<-PC+al2+l; (SP)<-PC+1									3,4	1
ICALL		Pośrednie wywołanie podprogramu	PC<-Z; (SP)<-PC+1									3,4	1
RET		Powrót z podprogramu	PC<-(SP)									4,5	1
RETI		Powrót z przerwania	PC<-(SP)	1								4,5	1
CPSE	Rd, Rs	Porównaj i skok, jeśli równe	(Rd=Rs) -> PC<-PC+(2/3)									1,2,3	1
CP	Rd, Rs	Porównaj rejestry	Rd-Rs			↕	↕	↕	↕	↕	↕	1	1
CPC	Rd, Rs	Porównaj rejestry wraz z C	Rd-Rs-C			↕	↕	↕	↕	↕	↕	1	1
CPI	Rh, K8	Porównaj rejestr ze stałą	Rh-K8			↕	↕	↕	↕	↕	↕	1	1
SBRC	Rs, b	Pomiń, gdy bit w rejestrze wyzerowany	(Rs.b=0) → PC<-PC+l ↪ PC<-PC+2									1,2,3	1
SBRS	Rs, b	Pomiń, gdy bit w rejestrze ustawiony	(Rs.b=l) → PC<-PC+1 ↪ PC<-PC+2									1,2,3	1
SBIC	PI, b	Pomiń, gdy bit w rejestrze IO wyzerowany	(Pl.b=0) → PC<-PC+l ↪ PC<-PC+2									1,2,3	1
SBIS	PI, b	Pomiń, gdy bit w rejestrze IO ustawiony	(Pl.b=l) → PC<-PC+1 ↪ PC<-PC+2									1,2,3	1
BRBS	b,k7	Skok, gdy flaga w SREG ustawiona	(SREG.b=l)→PC<-PC+k7+l ↪ PC<-PC+l									1,2	1
BRBC	b,k7	Skok, gdy flaga w SREG skasowana	(SREG.b=0)→ PC <-PC+k7+1 ↪ PC <-PC+1									1,2	1
BREQ	k7	Skok względny, gdy równe	(Z=l)→PC<-PC+k7+l ↪ PC<-PC+l									1,2	1

BRNĘ	k7	Skok względny, gdy różne	(Z=0)➔PC<-PC+k7+l ↳ PC<-PC+l										1,2	1
BRCS	k7	Skok względny, gdy C=l	(C=l)➔PC<-PC+k7+l ↳ PC<-PC+l										1,2	1
BRCC	k7	Skok względny, gdy C=0	(C=0)➔PC<-PC+k7+l↳ PC<-PC+l										1,2	1
BRSH	k7	Skok względny, gdy większy lub równy	(C=0)➔PC<-PC+k7+l ↳ PC<-PC+l										1,2	1
BRLO	k7	Skok względny, gdy mniejszy (1)	(C=l)➔PC<-PC+k7+l ↳ PC<-PC+l										1,2	1
BRMI	k7	Skok względny, gdy ujemny	(N=l)➔PC<-PC+k7+l ↳ PC<-PC+l										1,2	1
BRPL	k7	Skok względny, gdy dodatni	(N=0)➔PC<-PC+k7+l ↳ PC<-PC+l										1,2	1
BRGE	k7	Skok względny, gdy większy lub równy (2)	(S=0)➔PC<-PC+k7+l ↳ PC<-PC+l										1,2	1
BRLT	k7	Skok względny, gdy mniejszy od zera (2)	(S=l)➔PC<-PC+k7+l ↳ PC<-PC+l										1,2	1
BRHS	k7	Skok względny, gdy H=l	(H=l)➔PC<-PC+k7+l ↳ PC<-PC+l										1,2	1
BRHC	k7	Skok względny, gdy H=0	(H=0)➔PC<-PC+k7+l ↳ PC<-PC+l										1,2	1
BRTS	k7	Skok względny, gdy T=l	(T=l)➔PC<-PC+k7+l ↳ PC<-PC+l										1,2	1
BRTC	k7	Skok względny, gdy T=0	(T=0)➔PC<-PC+k7+l ↳ PC<-PC+l										1,2	1
BRVS	k7	Skok względny, gdy V=l	(V=l) ➔PC<-PC+k7+l ↳ PC<-PC+l										1,2	1
BRVC	k7	Skok względny, gdy V=0	(V=0) ➔PC<-PC+k7+l ↳ PC<-PC+l										1,2	1
BRIE	k7	Skok względny, gdy 1=1	(I=l) ➔ PC<-PC+k7+l ↳ PC<-PC+l										1,2	1
BRID	k7	Skok względny, gdy 1=0	(I=0) ➔ PC<-PC+k7+l ↳ PC<-PC+l										1,2	1

Mnemonik	Operandy	Opis	Operacja	Rejestr statusu SREG								Liczba	
				I	T	H	S	V	N	Z	C	cykli	słów

Operacje bitowe

LSL	Rd	Przesuń logicznie w lewo Rd	C <- Rd <- 0			↕	↕	↕	↕	↕	↕	1	1
LSR	Rd	Przesuń logicznie w prawo Rd	0 -> Rd -> C				↕	↕	0	↕	↕	1	1
ROL	Rd	Obróć w lewo z przeniesieniem Rd	↱Rd<-C↵			↕	↕	↕	↕	↕	↕	1	1
ROR	Rd	Obróć w prawo z przeniesieniem Rd	↦C->Rd↴				↕	↕	↕	↕	↕	1	1
ASR	Rd	Przesuń arytmetycznie w prawo Rd	↻Rd->			↓	↕	↕	↕	↕	↕	1	1
SWAP	Rd	Zamień tetrady w rejestrze Rd	Rd[3:0]⇆ Rd[7:4]									1	1

BSET	b	Ustaw znacznik w SREG	SREG.b<-l	↕	↕	↕	↕	↕	↕	↕	↕	1	1
BCLR	b	Zeruj znacznik w SREG	SREG.b<-0	↕	↕	↕	↕	↕	↕	↕	↕	1	1
SBI	PI, b	Ustaw bit w rejestrze IO	P.b<-1									2	1
CBI	PI, b	Zeruj bit w rejestrze IO	P.b<-0									2	1
BST	Rs, b	Zachowaj bit rejestru Rs w znaczniku T	T<-Rs.b		↕							1	1
BLD	Rd, b	Ładuj znacznik T do bitu rejestru Rd	Rd.b<-T									1	1
SEC		Ustaw znacznik przeniesienia C	C<-1								1	1	1
CLC		Zeruj znacznik przeniesienia C	C<-0								0	1	1
SEN		Ustaw znacznik wartości ujemnej N	N<-1						1			1	1
CLN		Zeruj znacznik wartości ujemnej N	N<-0						0			1	1
SEZ		Ustaw znacznik zera Z	Z<-1							1		1	1
CLZ		Zeruj znacznik zera Z	z<-0							0		1	1
SEI		Odblokuj przerwania	I<-1	1								1	1
CLI		Zablokuj przerwania	I<-0	0								1	1
SES		Ustaw znacznik znaku S	S<-1				1					1	1
CLS		Zeruj znacznik znaku S	S<-0				0					1	1
SEV		Ustaw znacznik pożyczki V	V<-1					1				1	1
CLV		Zeruj znacznik pożyczki V	V<-0					0				1	1
SET		Ustaw znacznik T	T<-1		1							1	1
CLT		Zeruj znacznik T	T<-0		0							1	1

SEH		Ustaw znacznik przeniesienia H	H<-1			1						1	1
CLH		Zeruj znacznik przeniesienia H	H<-0			0						1	1

Inne rozkazy

NOP		Nic nie rób										1	1
SLEEP		Przejdź w tryb uśpienia										1	1
WDR		Zeruj licznik Watchdog										1	1

Rozkazy przesyłania danych

MOV	Rd, Rs	Kopiuj zawartość Rs do Rd	Rd<-Rs									1	1
LDI	Rh, K8	Ładuj rejestr stałą bezpośrednią	Rh<-K8									1	1
LDS	Rd, Al 6	Ładuj rejestr bezpośrednio daną z SRAM	Rd<-(A16)									2	2
LD	Rd, X	Ładuj rejestr pośrednio daną z SRAM	Rd<-(X)									2	1
LD	Rd, X+	Ładuj rejestr pośrednio daną z SRAM	Rd<-(X); X<-X+l									2	1
LD	Rd,-X	Ładuj rejestr pośrednio daną z SRAM	X<-X-l;Rd<-(X)									2	1
LD	Rd, Y	Ładuj rejestr pośrednio daną z SRAM	Rd<-(Y)									2	1
LD	Rd, Y+	Ładuj rejestr pośrednio daną z SRAM	Rd<-(Y); Y<-Y+l									2	1
LD	Rd,-Y	Ładuj rejestr pośrednio daną z SRAM	Y<-Y-l;Rd<-(Y)									2	1
LDD	Rd, Y+K6	Ładuj rejestr pośrednio daną z SRAM	Rd<-(Y+K6)									2	1
LD	Rd, Z	Ładuj rejestr pośrednio daną z SRAM	Rd<-(Z)									2	1
LD	Rd, Z+	Ładuj rejestr pośrednio daną z SRAM	Rd<-(Z); Z<-Z+l									2	1
LD	Rd,-Z	Ładuj rejestr pośrednio daną z SRAM	Z<-Z-l; Rd<-(Z)									2	1
LDD	Rd, Z+K6	Ładuj rejestr pośrednio daną z SRAM	Rd<-(Z+K6)									2	1
STS	A16, Rs	Zachowaj bezpośrednio rejestr w SRAM	(A16)<-Rs									2	2
ST	X, Rs	Zachowaj pośrednio rejestr w SRAM	(X)<-Rs									2	1
ST	X+, Rs	Zachowaj pośrednio rejestr w SRAM	(X)<-Rs; X<-X+l									2	1
ST	-X, Rs	Zachowaj pośrednio rejestr w SRAM	X<-X-l;(X)<-Rs									2	1
ST	Y, Rs	Zachowaj pośrednio rejestr w SRAM	(Y)<-Rs									2	1

ST	Y+, Rs	Zachowaj pośrednio rejestr w SRAM	(Y)<-Rs; Y<-Y+l									2	1
ST	-Y, Rs	Zachowaj pośrednio rejestr w SRAM	Y<-Y-1;(Y)<-Rs									2	1
STD	Y+K6, Rs	Zachowaj pośrednio rejestr w SRAM	(Y+K6)<-Rs									2	1
ST	Z, Rs	Zachowaj pośrednio rejestr w SRAM	(Z)<-Rs									2	1
ST	Z+, Rs	Zachowaj pośrednio rejestr w SRAM	(Z)<-Rs; Z<-Z+l									2	1
ST	-Z, Rs	Zachowaj pośrednio rejestr w SRAM	Z<-Z-l;(Z)<-Rs									2	1
STD	Z+K6, Rs	Zachowaj pośrednio rejestr w SRAM	(Z+K6)<-Rs									2	1
LPM		Ładuj bajt pamięci programu do RO	R0<-FLASH(Z)									3	1
IN	Rd, P	Odczyt rejestru IO	Rd<-P									2	1
OUT	P, Rs	Zapis rejestru IO	P<-Rs									2	1
PUSH	Rs	Odłóż rejestr na stos	(SP)<-Rs									2	1
POP	Rd	Pobierz rejestr ze stosu	Rd<-(SP)									2	1

2.Template programu w asemblerze.

```
; code ex ample for lab 20
.nolist ;quartz assumption 4Mhz
.include "m128def.inc"
.list
.ESEG ; EEPROM memory segment

.DSEG ; SRAM memory.segment
.ORG 0x100; may be omitted this is default value
RAMTAB: .BYTE xlengthxx ; Destination table (xlengthx bytes).

.CSEG ; CODE Program memory. Remember that it is "word" address space
.org 0x0000
jmp RESET  ; Reset Handler

; Interrupts vector table / use only when needed
jmp EXT_INT0        ; IRQ0 Handler
jmp EXT_INT1        ; IRQ1 Handler
jmp EXT_INT2        ; IRQ2 Handler
jmp EXT_INT3        ; IRQ3 Handler
jmp EXT_INT4        ; IRQ4 Handler
jmp EXT_INT5        ; IRQ5 Handler
jmp EXT_INT6        ; IRQ6 Handler
jmp EXT_INT7        ; IRQ7 Handler
jmp TIM2_COMP       ; Timer2 Compare Handler
jmp TIM2_OVF  ;Timer2 Overflow Handler
jmp TIM1_CAPT ;Timer1 Capture Handler
jmp TIM1_C0MPA;Timer1 CompareA Handler
jmp TIM1_C0MPB;Timer1 CompareB Handler
jmp TIM1_0VF  ;Timer1 Overflow Handler
jmp TIM0_COMP ;Timer0 Compare Handler
jmp TIM0_OVF  ;Timer0 Overflow Handler
jmp SPI_STC   ;SPI Transfer Complete Handler
jmp USART0_RXC;USART0 RX Complete Handler
```

```
jmp USART0_DRE;USART0,UDR Empty Handler
jmp USART0_TXC;USART0 TX Complete Handler
jmp ADC       ;ADC Conversion Complete Handler
jmp EE_RDY    ;EEPROM Ready Handler
jmp ANA_COMP  ;Analog Comparator Handler
jmp TIM1_C0MPC;Timer1 CompareC Handler
jmp TIM3_CAPT ;Timer3 Capture Handler
jmp TIM3_COMPA;Timer3 CompareA Handler
jmp TIM3_COMPB; Timer3 CompareB Handler
jmp TIM3_COMPC;Timer3 CompareC Handler
jmp TIM3_OVF  ;Timer3 Overflow Handler
jmp USART1_RXC;USART1 RX Complete Handler
jmp USART1_DRE;USART1,UDR Empty Handler
jmp USART1_TXC;USART1 TX Complete Handler
jmp TWI       ;Two-wire Serial Interface Interrupt Handler
jmp SPM_RDY   ;SPM Ready Handler

RESET:        ldi r16, high(RAMEND); Main program start
      out   SPH,r16   ; Set stack pointer to top of RAM
      ldi   r16, low(RAMEND)
      out   SPL,r16
      cli ; Disable all interrupts
;
; place here code related to initialization of ports and interrupts
;;    <instr>  xxx
; End of port  initialization
      sei         ; Enable interrupts
;
; Main program code place here
;     <instr>  xxx
; First  load initial values of index registers
;  Z, X, Y
;
;-----------------------------------------------------------
```

```
; Ending loop
;-------------------------------------------------------
End:
rjmp END
; place here test values
; Test with value 0x8000 also
;
ROMTAB: .db 0x01, 0x00 , 0xffff
.EXIT
```

3.Template programu w C.

```
//  Code example for Lab 21
//
//
#include <avr/interrupt.h>
#include <avr/eeprom.h>
#include <avr/io.h>
#include <avr/iom16.h>

// ************** zmienne globalne **************
#define TABLE_LENGTH ????????set proper value ?????!!!    // remember to set proper value here

volatile unsigned char tab_ram[TABLE_LENGTH]; // Table in RAM
static unsigned char tab_rom[] PROGMEM = {0x20,0x15, 0x10, 0x43, 0x20, 0x02, 0x00};

// ************** main **************
void main(void) {

// ----------------------------------------------------
//  I/O configuration
// for instancje port A
```

```
// PORTA=0x00;
// DDRA=0xFF; // output
//enable interrupts
sei();
// place main code here


// end of programm
}
```