Laboratorium PTM

Programowanie w środowisku Microchip Studio – trzeci program w języku C.

Celem instrukcji jest zaznajomienie ze środowiskiem i ilustracja najprostszych programów w języku C. W szczególności zaś wykorzystanie niskopoziomowych operacji bitowych na rejestrach mikrokontrolera ATmega328P. W przykładzie wykorzystany zostanie rejestr wejścia/wyjścia (I/O) pozwalający na sterowanie stanem wyjść cyfrowych mikrokontrolera.

Zakład Systemów Informacyjno-Pomiarowych

IETiSIP, Wydział Elektryczny, PW

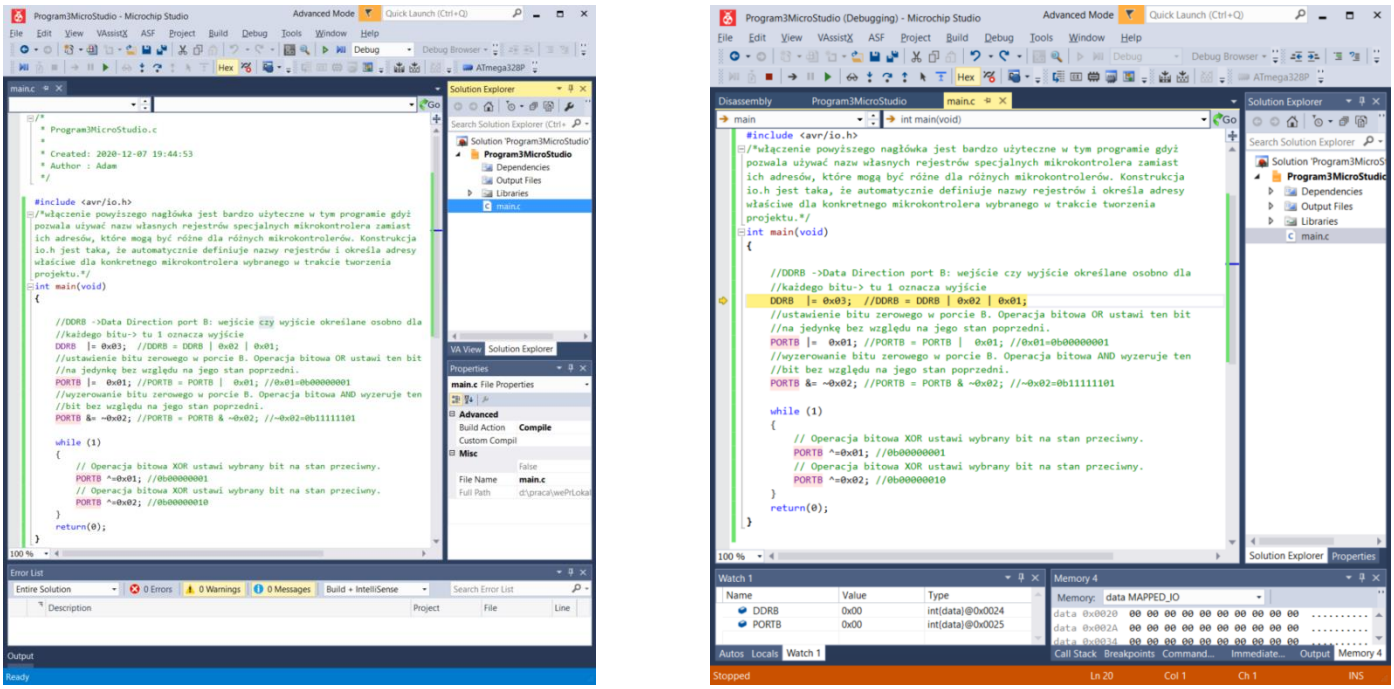
Program trzeci w języku C:

Przy analizie kodów asemblerowych przydatna jest dokumentacja mikrokontrolera ATmega328P z listą rozkazów (i przypisanymi cyklami).

Trzweci program w języku C dla mikrokontrolera ATmega328p niech ma postać taką jak na rysunku 1. Jak widać struktura programu również nie jest skomplikowana. Zawiera on kilka operacji bitowych na rejestrze PORTB, który służy do sterowania wyjściami cyfrowymi na tymże porcie mikrokontrolera ATmega328P. Program rozpoczyna się włączeniem do programu nagłówka io.h:

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

Nagłówek ten zawiera informacje o adresach rejestrów i wyprowadzeń używanego mikrokontrolera. Podczas tworzenia projektu wybrany został konkretny mikrokontroler a włączony nagłówek jest tu odpowiedzialny za przypisanie właściwych danych i dodatkowo zdefiniowanie nazw (literałów znakowych) dla rejestrów i wyprowadzeń konkretnego mikrokontrolera. Dzięki temu nie trzeba się szukać w dokumentacji specyficznych danych dla wybranego układu i co więcej można wykorzystywać ustandaryzowane nazwy dla poszczególnych elementów wybranego mikrokontrolera (np.: DDRB, PORTB, PINB).



Rysunek 1. Okno z programem do ćwiczenia.

Kompilację i wygenerowanie plików binarnych można wykonać poprzez Build -> **Build Solution/Application** (F7), po ewentualnych zmianach kodu można skorzystać z opcji **Rebuild** (Ctrl+Alt+F7) a „zresetowanie" projektu można uzyskać poprzez **Clean Solution/Application**. W ćwiczeniu program można uruchamiać poprzez pasek narzędziowy i **Start Debugging** (F5) oraz **Stop Debugging** (Ctrl+Shift+F5). Identycznie można sterować wykonaniem programu wybierając odpowiednie opcje z menu **Debug**. W ćwiczeniach jednak wykorzystywać należy tryb krokowy: **Step Over** (F10). Jest to tryb pracy krokowej bez wchodzenia do funkcji/procedur (wykonywane są one w jednym kroku). Jeżeli zaistnieje potrzeba przeanalizowania działania funkcji (szczególnie samodzielnie napisanej) wtedy można użyć trybu **Step Into** (F11). Możliwość wyjścia z procedury da w tym przypadku **Step out** (Shift+F11). Powyższe opcje są standardowe dla

środowisk programistycznych. Po utworzeniu projektu i wpisaniu drugiego programu (F10) okno środowiska będzie wyglądać jak na rysunku 1 (z prawej strony). Wciskając kolejno F10 program wykonywał będzie kolejne instrukcje. Żółta strzałka wskazuje aktualną instrukcję do wykonania (wiąże się to z zawartością rejestru PC – Program Counter). W tym samym obszarze, w którym znajduje się strzałka można ustawiać pułapki (breakpoints). Wystarczy użyć lewego klawisza myszy. Pułapki są użyteczne, kiedy zaistnieje potrzeba dokładnej analizy pewnego fragmentu kodu (przy założeniu, że program jest złożony niż omawiany obecnie) z pominięciem pozostałej części.
Okno **Autos** (na dole po lewej stronie) zawiera zmienne zdefiniowane w programie wraz z informacją o położeniu (lokalizacji, adresie) zmiennej w pamięci danych. Architektura mikrokontrolerów ATmega jest architekturą Harwardzką z podziałem na pamięć danych i pamięć programu. W oknie **Memory** po prawej stronie można podglądać zawartość pamięci. Przy wyborze **IRAM** (Internal RAM) można podglądać aktualną zawartość pamięci danych. Znając adresy poszczególnych zmiennych w programie (okienko **Autos** i zmienne: i, n, a) można podglądać zmiany tychże zmiennych (w takt wykonywania programu - F10). Proszę zwrócić uwagę na kolejności umieszczania zmiennych w pamięci danych.

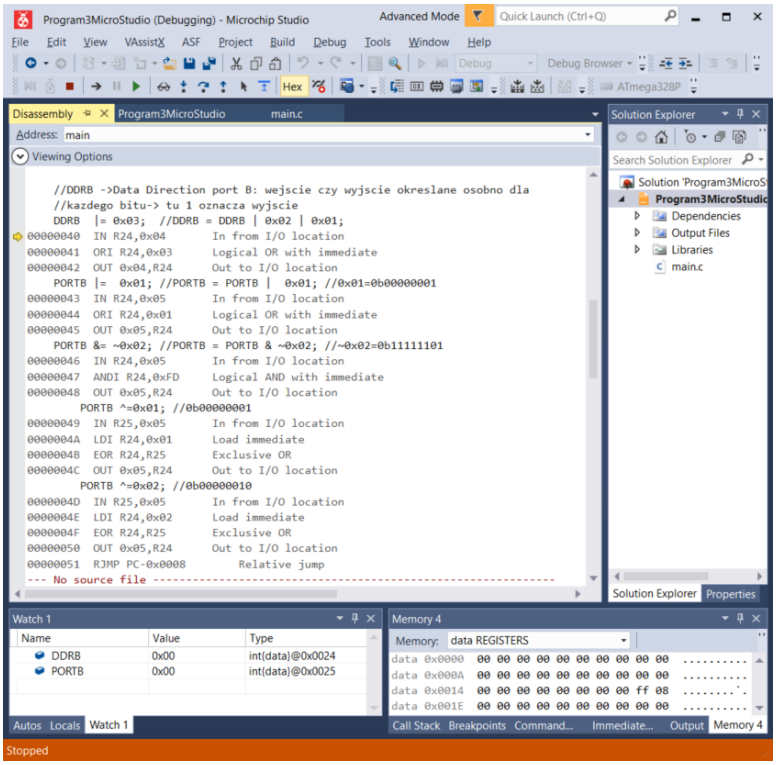
Figure 7-2. AVR CPU General Purpose Working Registers

	7	0	Addr.	
	R0		0x00	
	R1		0x01	
	R2		0x02	
	…			
	R13		0x0D	
General	R14		0x0E	
Purpose	R15		0x0F	
Working	R16		0x10	
Registers	R17		0x11	
	…			
	R26		0x1A	X-register Low Byte
	R27		0x1B	X-register High Byte
	R28		0x1C	Y-register Low Byte
	R29		0x1D	Y-register High Byte
	R30		0x1E	Z-register Low Byte
	R31		0x1F	Z-register High Byte

Rysunek 2. Rejestry ogólnego przeznaczenia mikrokontrolera ATmega328P.

UWAGA! Omawiany program został umieszczony w pamięci pod konkretnym adresem i wykorzystano w nim pewne konkretne rejestry ogólnego przeznaczenia Rxx (w programie są to rejestry R24 i R25 o adresach odpowiednio 0x18 i 0x19 – przedrostek 0x oznacza zapis heksadecymalny czyli szesnastkowy – rysunek 2). Zarówno adres początkowy programu w pamięci, jak i wykorzystane rejestry mogą się różnić w różnych kompilacjach. Dlatego podczas analizy własnych programów należy uwzględnić zarówno możliwość innego adresu początkowego zasadniczego kodu programu, inne zestawy rejestrów oraz różne położenie zmiennych w pamięci danych. Wiąże się to z koniecznością obserwacji innych obszarów pamięci.

Szczegóły wykonania programu można poznać wybierając (przy uruchomionym programie) **Debug**->**Windows**-> **Disassembly** i otrzymując w wyniku asemblerowy kod programu (rysunek 3).



Rysunek 3. Okno z kodem asemblerowym.

Program w asemblerze:

```
;W programie wykorzystywane są dwa rejestry ogólnego przeznaczenia R24 i R25
;Program umieszczony jest w pamięci programu począwszy od adresu 0x00000040
;Proszę zwrócić uwagę, że operacje na rejestrach specjalnych (DDRB i PORTB)
;odbywają się pośrednio poprzez operacje na rejestrach ogólnego przeznaczenia
;R24 i R25)
//Program w C-> DDRB  |= 0x03;//DDRB = DDRB |  0x02 | 0x01;
;Do rejestru R24 ładuje się zawartość rejestru DDRB (Rysunek A.1)
00000040  IN R24,0x04      ;In from I/O location - In Port
;Zawartość rejestru R24 sumowana jest logicznie z wartością 0x03 - jedynki na
;dwóch najmłodszych bitach
00000041  ORI R24,0x03     ;Logical OR with immediate
;Aktualizacja zawartości rejestru DDRB - dwa najmłodsze bity portu B zostały
;ustawione jako wyjścia
00000042  OUT 0x04,R24     ;Out to I/O location - Out Port

//Program w C-> PORTB |=  0x01; //PORTB = PORTB |  0x01; //0b00000001
;Do rejestru R24 ładuje się zawartość rejestru PORTB (Rysunek A.1)
00000043  IN R24,0x05      ;In from I/O location  - In Port
;Najmłodszy bit rejestru R24 ustawiany jest na jedynkę
00000044  ORI R24,0x01     ;Logical OR with immediate
;Aktualizacja zawartości rejestru PORTB
00000045  OUT 0x05,R24     ;Out to I/O location - Out Port

//Program w C-> PORTB &= ~0x02; //PORTB = PORTB & ~0x02; //0b11111101
;Ponownie do rejestru R24 ładuje się zawartość rejestru PORTB (Rysunek A.1)
00000046  IN R24,0x05      ;In from I/O location  - In Port
;Tym razem zerowany jest bit o wadze jeden: 0xFD = 0b11111101. Jedynki na
;pozostałych bitach gwarantują, że pozostałe bity pozostaną niezmienione.
00000047  ANDI R24,0xFD    ;Logical AND with immediate
```

```
;Aktualizacja nowej zawartości rejestru PORTB
00000048  OUT 0x05,R24      ;Out to I/O location - Out Port

;Jesteśmy w pętli while.  W asemblerze nie jest to widoczne ale wskazuje na to
;instrukcja z linii 0x0051
//Program w C-> PORTB ^=0x01; //0b00000001
;Ponownie do rejestru R25 ładuje się zawartość rejestru PORTB  (Rysunek A.1)
00000049  IN R25,0x05       ;In from I/O location  - In Port
;Do rejestru R24 ładuje się wartość 1 (stała) - bit o wadze 0 ma wartość 1,
pozostałe 0.
;W tym momencie warto zauważyć, że do załadowania rejestru ogólnego
;przeznaczenia zawartością rejestru I/O używana jest instrukcja IN
;(analogicznie z instrukcją OUT). Natomiast do ustawienia rejestru na wartość
;stałą służy instrukcja LDI.
0000004A  LDI R24,0x01      ;Load immediate
;Zmiana najmniej znaczącego bitu (waga 0) rejestru R24 na przeciwny. Zera na
;pozostałych bitach stałej gwarantują, że reszta bitów R24 pozostanie
;niezmieniona. Wynik w R24.
0000004B  EOR R24,R25       ;Exclusive OR
;Aktualizacja nowej zawartości rejestru PORTB
0000004C  OUT 0x05,R24      ;Out to I/O location - Out Port
//Program w C-> PORTB ^=0x02; //00000010
;Ponownie do rejestru R25 ładuje się zawartość rejestru PORTB  (Rysunek A.1)
0000004D  IN R25,0x05       ;In from I/O location
;Do rejestru R24 ładuje się wartość 2 (stała) - bit o wadze 1 ma wartość 1,
pozostałe 0.
0000004E  LDI R24,0x02      ;Load immediate
;Zmiana bitu o wadze 1 rejestru R24 na przeciwny. Zera na pozostałych bitach
;stałej gwarantują, że reszta bitów R24 pozostanie niezmieniona. Wynik w R24.
0000004F  EOR R24,R25       ;Exclusive OR
;Aktualizacja nowej zawartości rejestru PORTB
00000050  OUT 0x05,R24      ;Out to I/O location
;Skok bezwarunkowy pod adres PC-0x0008 = 0x0051 - 0x0008 = 0x0049.
00000051  RJMP PC-0x0008    ;Relative jump
```

Zadania do samodzielnego wykonania:
Odpowiednie bity ustawić należy jako wyjścia w rejestrze DDRB. Program należy wykonywać w pętli, gdzie w pojedynczej iteracji dokonywana jest pojedyncza zmiana (mały wyjątek stanowią tu światła na skrzyżowaniu gdzie czasami trzeba zmienić ustawienie dwóch linii/świateł).

1. stan początkowy PORTB same zera
   - kolejno pojawiające się jedynki począwszy od najmniej znaczącego bitu (LSB) do najbardziej znaczącego bitu (MSB)
2. stan początkowy PORTB same zera
   - kolejno pojawiające się jedynki począwszy od najbardziej znaczącego bitu (MSB) do najmniej znaczącego bitu (LSB)
3. stan początkowy PORTB same jedynki
   - kolejno pojawiające się zera począwszy od najmniej znaczącego bitu (LSB) do najbardziej znaczącego bitu (MSB)
4. stan początkowy PORTB same jedynki
   - kolejno pojawiające się zera począwszy od najbardziej znaczącego bitu (MSB) do najmniej znaczącego bitu (LSB)
5. stan początkowy PORTB same zera
   - wędrująca jedynka w kierunku począwszy od najmniej znaczącego bitu (LSB) do najbardziej znaczącego bitu (MSB)
6. stan początkowy PORTB same zera

- wędrująca jedynka w kierunku począwszy od najbardziej znaczącego bitu (MSB) do najmniej znaczącego bitu (LSB)

7. stan początkowy PORTB same jedynki
   - wędrujące zero w kierunku począwszy od najmniej znaczącego bitu (LSB) do najbardziej znaczącego bitu (MSB)
8. stan początkowy PORTB same jedynki
   - wędrujące zero w kierunku począwszy od najbardziej znaczącego bitu (MSB) do najmniej znaczącego bitu (LSB)
9. stan początkowy PORTB same zera
   - sekwencja sygnalizacji świetlnej na najmłodszych 3 bitach
10. stan początkowy PORTB same zera
    - sekwencja sygnalizacji świetlnej na najstarszych 3 bitach

Dodatek A:

ATmega48A/PA/88A/PA/168A/PA/328/P

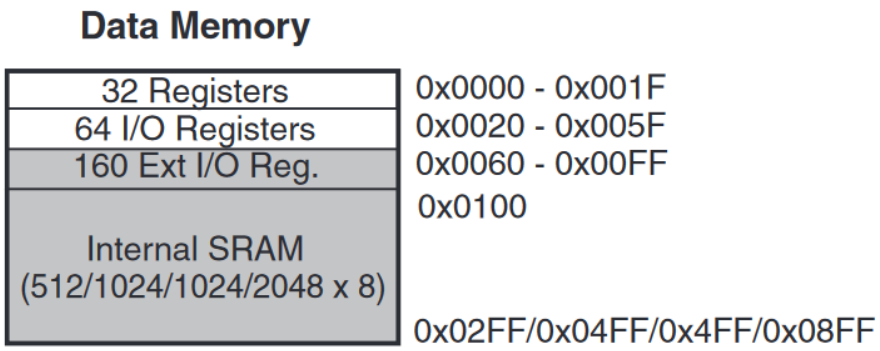
Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Page
0x15 (0x35)	TIFR0	–	–	–	–	–	OCF0B	OCF0A	TOV0	
0x14 (0x34)	Reserved	–	–	–	–	–	–	–	–	
0x13 (0x33)	Reserved	–	–	–	–	–	–	–	–	
0x12 (0x32)	Reserved	–	–	–	–	–	–	–	–	
0x11 (0x31)	Reserved	–	–	–	–	–	–	–	–	
0x10 (0x30)	Reserved	–	–	–	–	–	–	–	–	
0x0F (0x2F)	Reserved	–	–	–	–	–	–	–	–	
0x0E (0x2E)	Reserved	–	–	–	–	–	–	–	–	
0x0D (0x2D)	Reserved	–	–	–	–	–	–	–	–	
0x0C (0x2C)	Reserved	–	–	–	–	–	–	–	–	
0x0B (0x2B)	PORTD	PORTD7	PORTD6	PORTD5	PORTD4	PORTD3	PORTD2	PORTD1	PORTD0	101
0x0A (0x2A)	DDRD	DDD7	DDD6	DDD5	DDD4	DDD3	DDD2	DDD1	DDD0	101
0x09 (0x29)	PIND	PIND7	PIND6	PIND5	PIND4	PIND3	PIND2	PIND1	PIND0	101
0x08 (0x28)	PORTC	–	PORTC6	PORTC5	PORTC4	PORTC3	PORTC2	PORTC1	PORTC0	100
0x07 (0x27)	DDRC	–	DDC6	DDC5	DDC4	DDC3	DDC2	DDC1	DDC0	100
0x06 (0x26)	PINC	–	PINC6	PINC5	PINC4	PINC3	PINC2	PINC1	PINC0	101
0x05 (0x25)	PORTB	PORTB7	PORTB6	PORTB5	PORTB4	PORTB3	PORTB2	PORTB1	PORTB0	100
0x04 (0x24)	DDRB	DDB7	DDB6	DDB5	DDB4	DDB3	DDB2	DDB1	DDB0	100
0x03 (0x23)	PINB	PINB7	PINB6	PINB5	PINB4	PINB3	PINB2	PINB1	PINB0	100
0x02 (0x22)	Reserved	–	–	–	–	–	–	–	–	
0x01 (0x21)	Reserved	–	–	–	–	–	–	–	–	
0x0 (0x20)	Reserved	–	–	–	–	–	–	–	–	

Note:
1. For compatibility with future devices, reserved bits should be written to zero if accessed. Reserved I/O memory addresses should never be written.
2. I/O Registers within the address range 0x00 - 0x1F are directly bit-accessible using the SBI and CBI instructions. In these registers, the value of single bits can be checked by using the SBIS and SBIC instructions.
3. Some of the Status Flags are cleared by writing a logical one to them. Note that, unlike most other AVRs, the CBI and SBI instructions will only operate on the specified bit, and can therefore be used on registers containing such Status Flags. The CBI and SBI instructions work with registers 0x00 to 0x1F only.
4. When using the I/O specific commands IN and OUT, the I/O addresses 0x00 - 0x3F must be used. When addressing I/O Registers as data space using LD and ST instructions, 0x20 must be added to these addresses. The ATmega48A/PA/88A/PA/168A/PA/328/P is a complex microcontroller with more peripheral units than can be supported within the 64 location reserved in Opcode for the IN and OUT instructions. For the Extended I/O space from 0x60 - 0xFF in SRAM, only the ST/STS/STD and LD/LDS/LDD instructions can be used.
5. Only valid for ATmega88A/88PA/168A/168PA/328/328P.
6. BODS and BODSE only available for picoPower devices ATmega48PA/88PA/168PA/328P

Rysunek A.1: Rejestry specjalne (I/O Registers) mikrokontrolera ATmega328P wraz z ich adresami.

Proszę zwrócić uwagę (rysunek A.1), że przy dostępie bezpośrednim (np.: instrukcje IN, OUT) podaje się adresy względem początku bloku rejestrów specjalnych (I/O Registers). Stąd przy dostępie do rejestru PORTB należy użyć adresu 0x04. Jeżeli natomiast do tego samego rejestru chcemy się odwołać za pomocą instrukcji LD czy SD należy użyć adresowania jednolitego wynikającego z organizacji pamięci danych (Rysunek A.2). Stąd w tym przypadku adres dla ww. rejestru PORTB wynosi 0x04+0x20 = 0x24.

Figure 8-3. Data Memory Map



Rysunek A.2: Organizacja adresowa pamięci danych mikrokontrolera ATmega328P.

Adresy rejestrów dla mikrokontrolera ATmega328P:
DDRB: 0x04 (0x24)
PORTB: 0x05 (0x25)
PINB: 0x03 (0x03) Port B Input
R24: 0x18
R25: 0x19
IRAM – Internal RAM
RS jest na porcie D
I2C jest na porcie C
SPI jest na porcie B