Laboratorium PTM / 5 - ATmega

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

Celem instrukcji jest zaznajomienie ze środowiskiem i ilustracja prostych programów w języku C. W szczególności zaś zaprezentowanie prostego algorytmu sortującego tablice zmiennych zawierających losowe lub zakłócone dane (np. z przetwornika ADC) w oparciu o mikrokontroler ATmega328P. W przykładzie zostanie wykorzystany algorytm sortowania bąbelkowego.

Zakład Systemów Informacyjno-Pomiarowych

IETiSIP, Wydział Elektryczny, PW

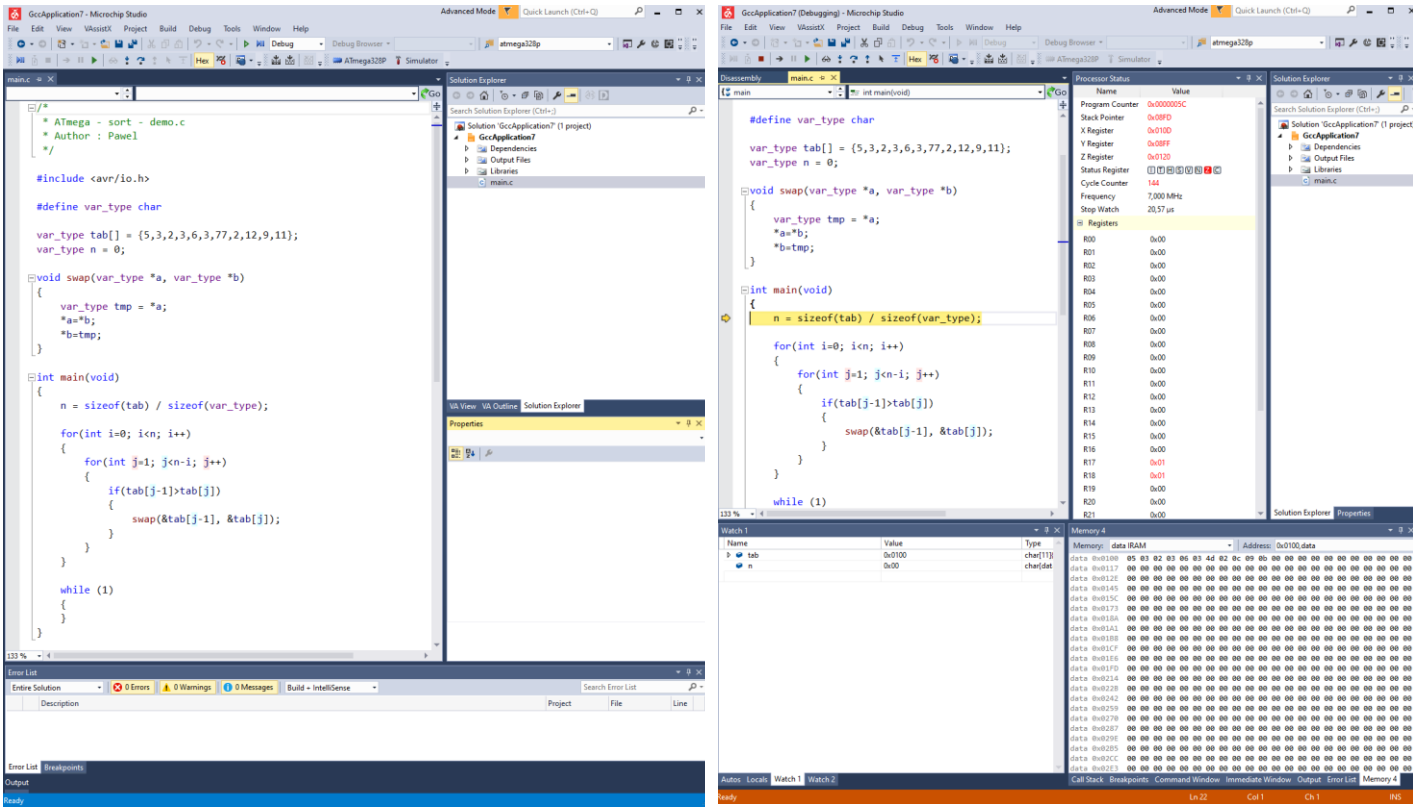
Piąty program w języku C:

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

Piąty program w języku C dla mikrokontrolera ATmega328p niech ma postać taką jak na rysunku 1. Jak widać struktura programu również nie jest skomplikowana. Zawiera bowiem kod wykonujący procedurę sortowania bąbelkowego, która jest jednym z najprostszych w implementacji algorytmów porządkujących dane. Złożoność tego sposobu sortowania jest rzędu $O(n^2)$. Oznacza to, że sortowanie bąbelkowe nadaje się do sortowania niezbyt dużych zbiorów. Nazwa wzięła się stąd, że dane podczas sortowania - tak jak bąbelki w napoju gazowanym - przemieszczają się ku „górze" i układają się w odpowiednim porządku w tablicy.

Działanie możemy opisać następująco: w każdym przejściu pętli wewnętrznej porównywane są ze sobą dwie sąsiednie wartości tablicy i jeśli spełniony jest warunek, są one zamieniane miejscami. W jednym cyklu pętli wewnętrznej, największa liczba w zbiorze będzie się przemieszczała na ostatnią pozycję. W ten sposób otrzymujemy podzbiór częściowo już posortowany. Czynności te powtarzamy dla zbioru pominiętego o elementy już poukładane. W przykładowym kodzie tabela jest zainicjowana wartościami losowymi, została ona zdefiniowana na początku programu jako zmienna globalna **tab[]**. Użytkownik ma możliwość (i powinność) wprowadzenia własnych danych do tabeli w dowolnej ilości z tym zastrzeżeniem, że zwróci uwagę na typ bazowy tabeli i typ bazowy zmiennej wyliczającej długość. Typ zmiennych (w tablicy) podajemy przez następującą definicję:

#define var_type char



Rysunek 1. Okno z programem do ćwiczenia.

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).

Definicja:

```
#define var_type char
```

Umożliwia zmianę typu zmiennej na której pracuje algorytm. Zmienna **n** przechowuje wyliczoną długość zainicjalizowanej tabeli **tab[]**, obie w tym przypadku będą typu **char**. Wprowadza to pewnie ograniczenie co do maksymalnej długości tabeli wejściowej.

Innym, lepszym nawet rozwiązaniem jest stworzenie własnego synonimu typu za pomocą poniższej instrukcji:

```
typedef char var_type;
```

Początkowe wartości zmiennych **tab[]** i **n** zostały wprowadzone ręcznie. Zmienna tabelaryczna przechowuje dane zebrane przykładowo przez mikrokontroler w ramach np. procesu pomiarowego. Zmienna **n** będzie dynamicznie określać długość tej tabeli. Pokazuje to linia:

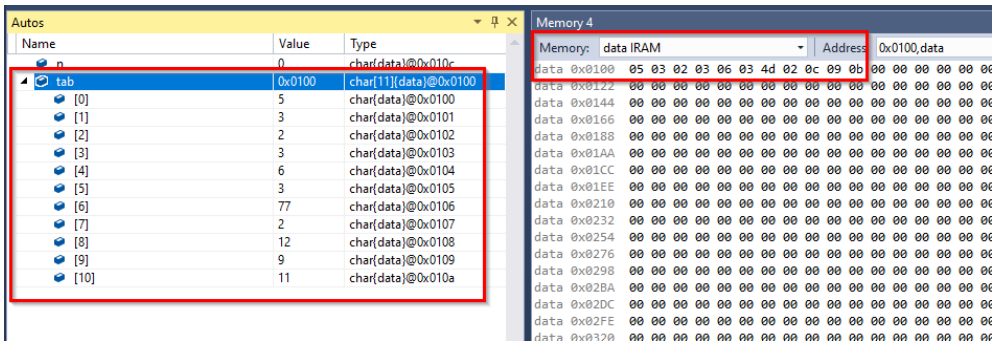
```
n = sizeof(tab) / sizeof(var_type);
```

Wynik działania funkcji **sizeof()** podawany jest w bajtach. W przypadku zmiennych typu char funkcja ta poda bezpośrednią długość tabeli, tj. 11 bajtów czyli pozycji w tabeli. W przypadku gdy zmienimy typ bazowy na int, każda pozycja w tabeli będzie już 2-bajtowa i funkcja zwróci 22 bajty – co będzie błędnym wskazaniem ilości pozycji w tabeli. Dlatego też, wyliczenie uzupełniono o dzielenie przez długość bazową wykorzystywanej zmiennej, gdzie 22 bajty podzielone przez 2 bajty daje ponownie 11 pozycji tabeli. Co jest już wartością prawidłową, możliwą do wykorzystania w dalszej części algorytmu.

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.

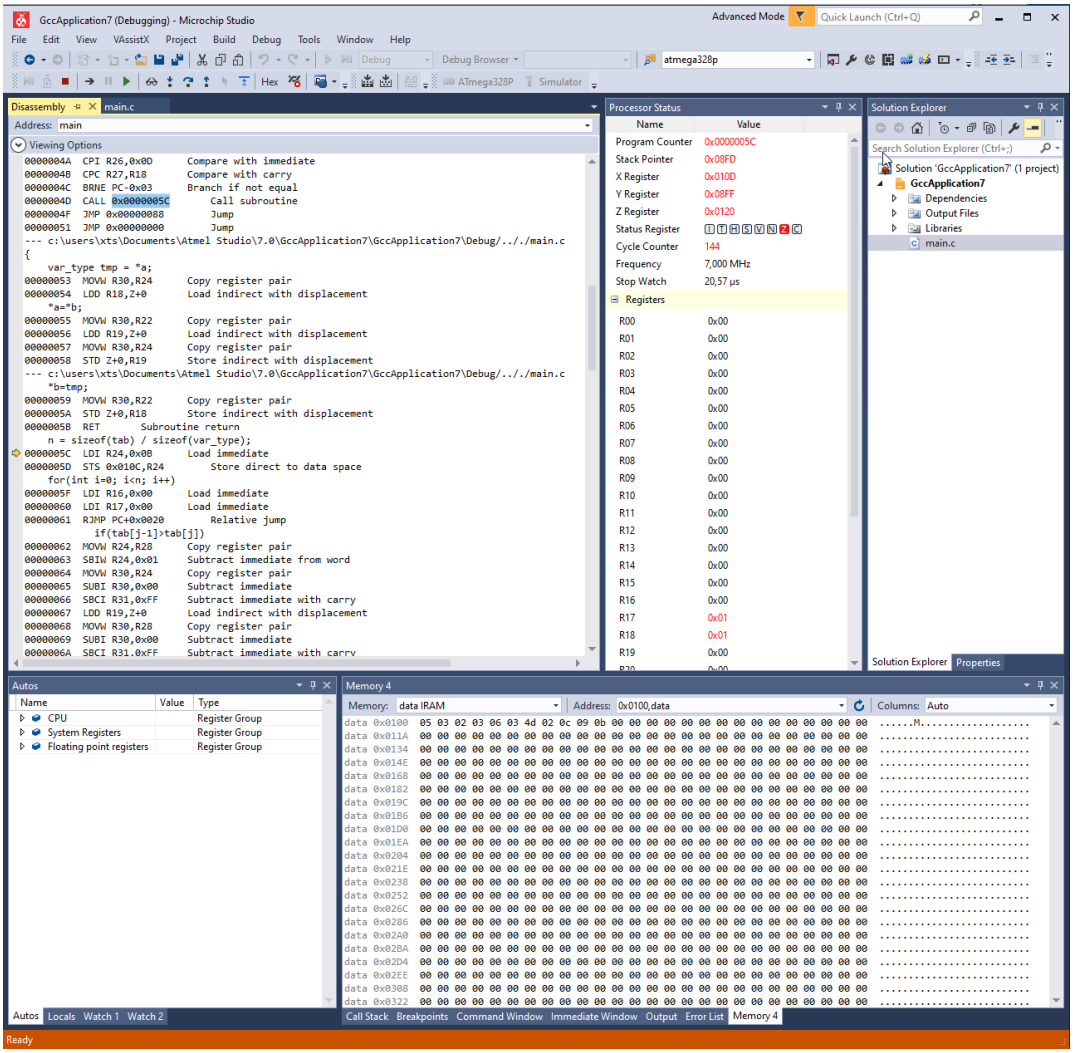
Okno **Autos** (na dole po lewej stronie) zawiera zmienne zdefiniowane w programie wraz z informacją o położeniu (lokalizacji, adresie) zmiennej w pamięci danych. Architektura mikrokontrolerów ATmega jest architekturą Harwardzką z podziałem na pamięć danych i pamięć programu. W oknie **Memory** po prawej 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: tab[], n) można podglądać zmiany tychże zmiennych (w takt wykonywania programu - F10). Proszę zwrócić uwagę na możliwość wyświetlenia zawartości tabeli w formie hierarchicznej oraz jej reprezentacji w pamięci IRAM – rysunek 2.



Rysunek 2. Możliwość obserwacji elementów składowych tabeli danych w pamięci MCU.

UWAGA! Omawiany program został umieszczony w pamięci pod konkretnym adresem iwykorzystano w nim pewne konkretne rejestry ogólnego przeznaczenia Rxx. Zarówno adres początkowy programu wpamięci, jak i wykorzystane rejestry mogą się różnić w różnych kompilacjach. Dlatego podczasanalizy własnych programów należy uwzględnić zarówno możliwość innego adresu początkowegozasadniczego kodu programu, inne zestawy rejestrów oraz różne położenie zmiennych w pamięcidanych. 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.

W języku asemblera również istnieje możliwość wykonywania programu w trybie krokowym. Jednakże w tym przypadku wykonywanie programu odbywa się instrukcja po instrukcji asemblerowej (NIE zaś instrukcja po instrukcji w języku C). Wartość prezentowana, jako pierwsza z lewej strony do adres w pamięci programu, czyli zawartość rejestru PC – Program Counter. Wygenerowany kod asemblerowy pochodzi z szablonu/schematu środowiska programistycznego. Fragment, który będzie w obszarze zainteresowania znajduje się poniżej komentarza z informacją o pliku źródłowym, w którym się znajduje tj. ---... /main.c. Komentarze dodane przez środowisko wyjaśniają dokładnie znaczenie poszczególnych instrukcji asemblerowych.

Początek programu użytkownika w asemblerze wraz z funkcją składową:

```
;Kod kompilatora poprzez poniższy CALL wywołuje program użytkownika, czyli
;funcję( void main() ). Omija on w wywołaniu funkcję swap().
0000004D  CALL 0x0000005C         Call subroutine
0000004F  JMP 0x00000088          Jump
00000051  JMP 0x00000000          Jump
;Funkcja swap()
{
```

```
       var_type tmp = *a;
00000053  MOVW R30,R24           Copy register pair
00000054  LDD R18,Z+0            Load indirect with displacement
       *a=*b;
00000055  MOVW R30,R22           Copy register pair
00000056  LDD R19,Z+0            Load indirect with displacement
00000057  MOVW R30,R24           Copy register pair
00000058  STD Z+0,R19            Store indirect with displacement
       *b=tmp;
00000059  MOVW R30,R22           Copy register pair
0000005A  STD Z+0,R18            Store indirect with displacement
0000005B  RET              Subroutine return
;Funkcja main()
;      n = sizeof(tab) / sizeof(var_type);
0000005C  LDI R24,0x0B           Load immediate
0000005D  STS 0x010C,R24         Store direct to data space
```

Wywołanie z algorytmu funkcji użytkownika w asemblerze :

```
;      swap(&tab[j-1], &tab[j]);
0000006E  MOVW R22,R30           Copy register pair
0000006F  SUBI R24,0x00          Subtract immediate
00000070  SBCI R25,0xFF          Subtract immediate with carry
00000071  CALL 0x00000053        Call subroutine
```

Zadania do wykonania:

1. Proszę zmienić definicję zmiennej bazowej **var_type** na **int** a następnie **float** i ocenić w jaki sposób rozbudowie uległa struktura kodu odpowiedzialnego za realizację funkcji swap i generalnie całego algorytmu.
2. Czy program wykonuje się poprawnie dla różnych zawartości tablic?
3. Proszę przetestować działanie kilku zestawów losowych, dla tablicy posortowanej ale przeciwnie do wymaganego porządku (dwa testy). W tym przypadku proszę użyć tablicy zainicjowanej inaczej niż 1,2,3… i odpowiednio np.: 10,9,8…
4. Czy można skrócić czas działania algorytmu (chodzi o pętlę zewnętrzną, która zawsze wykonuje się n razy) ?
5. Proszę wprowadzić tablicę 100-elementową oraz ustawić zegar na 1MHz, a następnie wykonać kod aby sprawdzić w jakim czasie MCU posortuje tablicę zmiennych typu **char**, **int** oraz **float**. Czy MCU jest optymalnym wyborem jeśli chodzi o pracę na zmiennych **float** ?

   W tym przypadku może okazać się pomocne wypełnienie tablicy wartościami losowymi, w tym celu można wykorzystać poniższy kod. Zawiera on podstawowe wywołanie funkcji „losującej” wraz z zainicjowaniem tj. wykorzystaniem „zarodka” do pracy generatora. Należy pamiętać o włączeniu stosownych plików nagłówkowych.

```
#include <avr/io.h>
#include <stdlib.h>
#include <time.h>
int a;
int main(void)
{
   srand(time(NULL));
   a=rand();
   /*
   if(( a < maksimum) && (a > minimum))
```

```
        {
                Jeżeli chcemy losować wartości z określonego zakresu
                można przepisywać dane do tablicy tylko wtedy kiedy
                spełniony jest warunek
                tab[i] = a;
        } */
}
```

6. Proszę zmienić kierunek sortowania na odwrotny.

Dodatek A: Wskaźniki

Programując w języku C, często spotkasz się ze wskaźnikami. Jak sama nazwa wskazuje, jest to zmienna, która musi na coś wskazywać – w naszym przypadku, jest to adres do zmiennej. Używanie wskaźników jest często wymagane dla działania kodu, ale także można tym sposobem poprawić jego wydajność. Wyobraź sobie, że masz niebieskie pudełko i skierowaną na nie czerwoną strzałkę – pudełko pełni w tym wypadku rolę zmiennej, a strzałka jest jej wskaźnikiem. Za pomocą strzałki, możemy wyciągnąć adres zmiennej która nas interesuje, ale także możemy znaleźć dane, które w tym pudełku pod tym adresem są przechowywane. Deklaracja wskaźnika nie różni się dużo od deklaracji zwykłej zmiennej. Najpierw określamy typ, a potem nazwę poprzedzoną symbolem gwiazdki „*". Poniżej znajduje się przykład deklaracji wskaźnika:

```
int zm;
int *wsk_zm;
```

Na chwile obecną, mamy zadeklarowaną zmienną o nazwie zm i wskaźnik o nazwie wsk_zm, jednak póki co nie przechowuje on żadnego adresu – jest to jedynie deklaracja. Aby przypisać adres zmiennej do wskaźnika, używamy następującej konstrukcji:

```
wsk_zm = &zm;
```

Możemy teraz za pomocą wskaźnika przypisać do zmiennej, na której adres wskazuje wstawić konkretną wartość w następujący sposób:

```
*wsk_zm = 150;  // w zmiennej zm mamy 150;
```

Gdzie w programowaniu możemy w praktyce zastosować wskaźniki? Na przykład przy tablicach. Tablica, to nic innego jak kolejno następujące po sobie komórki pamięci:

```
Tab[0] = 1;

Tab[1] = 2;

Tab[2] = 6;
```

Zapis wsk_zm = &tab[1] podaje adres w pamięci pozycji 1 tabeli. *wsk_zm = 8; nadpisuje w tym przypadku wartość 2 wartością 8. Przykład takiego odwołania znajdą Państwo w funkcji swap() która za jedyne zadanie ma zamienić miejscami w tabeli 2 wartości wskazane przez funkcję nadrzędną.

Dodatek B: Sortowanie bąbelkowe

i=1	i=2	i=3	i=4	i=5	i=6	i=7	i=8
42	04	04	04	04	04	04	04
59	42	11	11	11	11	11	11
11	59	42	16	16	16	16	16
41	11	59	42	41	41	41	41
97	41	16	59	42	42	42	42
16	97	41	41	59	59	59	59
04	16	97	63	63	63	63	63
63	63	63	97	97	97	97	97