

1. Asembler mikrokontrolerów AVR - wprowadzenie do programowania

Imię nazwisko, indeks :
Marcel Garczyk, 147935
Paweł Hatka,

Data wykonania : 13.10.2022r.

Grupa : T-2, czwartek 15:10-16:40

1. Program napisany w czasie zajęć wraz z opisem poszczególnych linii.

```
.nolist
#include "m16def.inc" // file with mnemonic names like RAMEND = 0x045F
.list
.listmac
.device ATmega16 // microprocessor name
.cseg
.org 0x0000 // starting address
// Commands below are executed in one clock cycle.
ldi R16, 255 //Store value 255 in register R16.
ldi R17, 1 //Store value 1 in register R17
add R16, R17 //Add values in registers R16 and R17. The result will be store in the register on the left side (R16)
//Jmp is execute in two cycles - this command informs us that the program has ended.
jmp PC
>E1 :nop
nop
jmp E1
```

2. Pytania i odpowiedzi z instrukcji:

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

.nolist :

Dyrektywa .nolist wyłącza generowanie listy programów, która jest kombinacją kodu źródłowego, adresów, kodów operacyjnych.

.include :

Jest to dyrektywa służąca do umożliwienia Assemblerowi dostępu do danego pliku i informacji w nim zawartych.

.list :

Dyrektywa ta rozpoczyna generowanie listy instrukcji, która domyślnie jest włączona, lecz poprzez użycie dyrektywy .nolist od wybranego momentu można przywrócić tą opcję.

.listmac :

Dyrektywa ta mówi assemblerowi, że kiedy wywoływane jest makro, rozwinięcie makra ma być pokazane w wygenerowanym pliku listy.

.device ATmega16

Dyrektywa definiuje dla jakiego rzeczywistego urządzenia należy zweryfikować dalsze instrukcje - w tym przypadku dla ATmega16.

- Gdzie znajduje się wynik dodawania?
Wynik dodawania zapisuje się w pierwszym argumencie instrukcji add, czyli w R16. W przypadku przekroczenia wartości 255 jako wyniku operacji, będzie tam wartość $x - 256$. Czyli np. jak w R16 wpiszemy 255, a w R17 wpiszemy 1, to $R16 + R17 = 256$, $256 - 256 = 0$
- Co wskazuje rejestr Program Counter?
Licznik programu jest rejestrem mikroprocesora zawierającym adres następnej instrukcji do wykonania z pamięci. Zmieniając wartość zapisaną w tym rejestrze możemy zaimplementować skoki, a zatem instrukcje warunkowe, pętle i podprogramy.
- Co wskazuje Cycle Counter?
Cycle counter informuje nas ile sumarycznie cykli zegara potrzebuje mikroprocesor, aby wykonać zadane instrukcje. Przykładowo instrukcja LDI zajmuje 1 cykl zegara.
- Dlaczego po wykonaniu ostatniej instrukcji Cycle Counter osiąga tak dużą wartość?
Wykonanie instrukcji JMP zajmuje 3 cykle zegara, wykonany zostaje skok to PC i potem ten sam skok jeszcze raz, jeszcze raz i tak dalej, co prowadzi do szybkiego naliczania się Cycle counter'a.
- W ilu cyklach zegara powinien zostać wykonany cały program (zobacz - w opisie instrukcji Help > Assembler Help - ile taktów zegara potrzebuje każda instrukcja)?

Jeżeli policzymy tylko instrukcje LDI, LDI oraz ADD to wykonanie programu zajmie 3 cykle zegara, każda z tych instrukcji wykonuje się w 1 cyklu.

Możemy zastanowić się co się stanie jeżeli policzymy pozostałe instrukcje:

JMP PC zajmuje 3 cykle (program zapętli się tu i będzie naliczał cykle w nieskończoność).

Ewentualnie usunięcie JMP PC i pozostawienie struktury:

```
E1:  nop
      nop
      jmp E1
```

Spowoduje, doliczenie 5 cykli (2xNOP (1 cykl) + JMP do E1 (3cykle)) i potem ciągłe doliczanie tych 5 cykli aż do wyłączenia zasilania.

- Jak działa instrukcja NOP ?
Instrukcja NOP - inaczej no operate, program przez 1 takt nie wykonuje żadnych działań następnie przechodzi dalej.
- Jak jest ogólne działanie instrukcja JMP?
Instrukcja JMP służy do przejścia do miejsca wskazanego przez użytkownika np. JMP PC - skok do program counter.
- Co zrobi procesor po wykonaniu instrukcji JMP PC ? :
Po wykonaniu tej instrukcji program w kółko będzie wykonywał instrukcja JMP PC- program będzie pozostawał w tej samej linii (pętla nieskończona).

3. Zadanie do wykonania samodzielnie:

I. Zadanie 1

- Kod programu analizowany w zadaniu:

```
.nolist
#include "m16def.inc" // file with mnemonic names like RAMEND = 0x045F
.list
.listmac
.device ATmega16 // microprocessor name
.cseg
.org 0x0000 // starting address
// Commands below are executed in one clock cycle.
ldi R16, 10 //Store value 255 in register R16.
ldi R17, 20 //Store value 1 in register R17
add R16, R17 //Add values in registers R16 and R17. The result will be store in the register on the left side (R16)
//Jmp is execute in two cycles - this command informs us that the program has ended.
jmp PC
```

- W opisie działania instrukcji LDI (Help > Assembler Help) znajdź binarny kod instrukcji (16-bit Optcode)

1110 KKKK dddd KKKK

- W trybie symulatora otwórz podgląd pamięci programu (View > Memory), wybierz pamięć programu (Program). W pamięci programu znajdź (zapisaną szesnastkowo) instrukcję LDI r16, 10.

Będzie to 0A E0 lub po zamianie bitów E0 0A, czyli dwójkowo 1110 0000 0000 1010, Gdzie pierwsze 4 bity to kod operacji, kolejne to górne 4 bity liczby ośmiobitowej, następne to numer rejestru gdzie R16 ma numer 0, a R31 numer 15, ostatnie to dolne 4 bity liczby.

- Podobnie zlokalizuj pozostałe instrukcje: LDI, ADD, NOP, JMP
 - a) LDI - E1 14
 - b) ADD – 0F 01 – opcode: 0000 11rd dddd rrrr
 - c) NOP – 00 00 – opcode: 0000 0000 0000 0000
 - d) JMP – 94 0C 00 03 – opcode: 1001 010k kkkk 110k kkkk kkkk kkkk
- Otwórz plik wynikowy *.hex i również zlokalizuj te instrukcje.

```
:0200000020000FC
:0A0000000FEF14E1010F0C94030050
:000000001FF
```

II. Zadanie 2:

- Zmień argumenty instrukcji ADD, wybierz dwie liczby tak, aby suma przekroczyła 255 (np: 100 + 200 =)

```
.nolist
#include "m16def.inc" // file with mnemonic names like RAMPEND = 0x045F
.list
.listmac
.device ATmega16 // microprocessor name
.cseg
.org 0x0000 // starting address
// Commands below are executed in one clock cycle.
ldi R16, 255 //Store value 255 in register R16.
ldi R17, 1 //Store value 1 in register R17
add R16, R17 //Add values in registers R16 and R17. The result will be store in the register on the left side (R16)
//Jmp is execute in two cycles - this command informs us that the program has ended.
jmp PC
```

- W pracy krokowej obserwuj działanie programu i zachowanie znacznika C w SREG.

The screenshots illustrate the execution of the assembly program in AVR Studio. The top window shows the initial state: Program Counter (PC) is 0x000001, and the SREG register shows the Carry flag (C) is 0. The middle window shows the execution of the 'add R16, R17' instruction, where the PC has advanced to 0x000002, and the SREG register shows the Carry flag (C) is 1, indicating a carry-out from the addition of 255 and 1. The bottom window shows the execution of the 'jmp PC' instruction, where the PC has advanced to 0x000003, and the SREG register shows the Carry flag (C) is 0.

- Co wskazuje znacznik (bit C)?
Jest to bit przeniesienia wskazuje, że przekroczona została wartość 255 podczas wykonywania instrukcji ADD.