УДК 004.4 DOI: 10.26565/2519-2310-2021-2-02

ПРОГРАМУВАННЯ МІКРОКОНТРОЛЕРІВ З ВИКОРИСТАННЯМ РІШЕНЬ SIMPLICITY STUDIO, НА МОВІ ASSEMBLER Ax51

Ольга Мелкозьорова, Ірина Гальцева

Харківський національний університет імені В.Н. Каразіна, майдан Свободи 4, Харків, 61022, Україна
olha.melkozerova@karazin.ua, irina.galceva@karazin.ua

Рецензент: Володимир Хома, д.т.н., проф., Опольський політехнічний Університет, Ополе, Польща
xoma@wp.pl

Надійшла: Вересень 2021.

Анотація: *У статті наведено загальний огляд існуючих можливостей програмування на мові Assembler Ax51 мікроконтролера C8051F340 у середовищі розробки Simplicity Studio. Робота є керівництвом для практичного застосування мови Assembler. У керівництвах [1-2] дуже багато інформації, стосовно мови Assembler та опису мікроконтролерів C8051F340, але ця інформація відокремлена одна від одної. Більш того, замало пояснень, щодо практичного використання, разом зі середовищем розробки Simplicity Studio [3]. Матеріал містить пояснення, стосовно компонентів, на які поділяються команди, а саме директиви, інструкції та елементи контролю. Наведено стислий опис регістрів, директив, елементів контролю, інструкцій, роботи зі змінними. Є приклади використання, у вигляді коду, деяких логічних та арифметичних інструкцій, з поясненням результатів виконання, котрі можна побачити у скріншотах зі значеннями регістрів після виконання команд. Додано приклади роботи зі змінними, результати яких фіксуються у пам'яті мікроконтролера.*

Ключові слова*: мікроконтролер; директиви; інструкції; елементи контролю; регістри; Assembler.*

1 Вступ

Існуючи керівництва з програмування на мови *Assembler Ax51* [1-2], містять в собі досить багато відомостей, стосовно цієї мови і особливостей мікроконтролерів *mod. C8051F340,* проте всю цю інформацію досить важко сприймати, як єдине та гармонійно взаємопов'язане ціле. З метою нівелювання зазначеного змістового розриву, має сенс провести загальний огляд існуючих можливостей програмування на мові *Assembler Ax51* з використанням існуючих можливостей середовища розробки *Simplicity Studio.*

2 Основна частина

2.1 Середовище розробки

Для вивчення прийомів і можливостей програмування в статті використовувалося середовище розробки *Simplicity Studio ver. SV5.2.0.0 2021* [3] (*див. рис.1*).

Рис.1 – Використана версія від студії *Simplicity Studio*

2.2 Опис регістрів мови Assembler

*Assembler Ax*51 визначає та використовує імена регістрів. Ці імена використовуються для доступу до регістрів процесору. У таблиці 1 наведено назви деяких регістрів, які, подальше, використовуються у прикладах, та стислий опис до них. При роботі у студії *Simlicity Studio* регістри, їх стислий опис та зміст наведено у вигляді таблиці (*див. рис. 3.1, де: a –*

шістнадцятирічна система числення; б – десятинний формат). Їх можна спостерігати після запуску відповідної програми.

Таблиця 1 – Опис регістрів та їх призначення

Регістр	*Опис*
A	Акумулятор. *Використовується з багатьма операціями, такими як, помноження, поділ, перенесення «із» та «до» зовнішньої пам'яті, логічні операції.*
DPTR	Регістр **Data Pointer**. *16-бітний показник, який використовує для адресації даних у пам'яті коду.*
PC	16-бітний лічильник. *Містить адресу інструкції, яка має бути виконана.*
C	Індикатор статусу операцій, які генерують біт переносу. *Також використовується операціями, яким потрібен додатковий біт.*
AB	Пара регістрів «**A**» та «**B**». *Використовується у інструкціях MUL та DIV.*
R0-R7	Вісім 8-бітних регістрів у поточному активному банку регістрів. *Всього доступні 4 банку регістрів.*
AR0-AR7	Абсолютні адреси даних від **R0** до **R7** у поточному банку регістрів. *Абсолютні значення адрес змінюються в залежності від банку регістрів, який буде обрано у поточний момент часу.*

2.3 Компоненти мови Assembler

В цілому, програма на *Assembler* складається з однієї або декількох ***компонент*** (*statements*). Ці компоненти мають:

- *директиви* (*directives*);
- *елементи управління* (*controls*);
- *інструкції* (*instructions or mnemonics*).

Директиви показують, як упорядкувати інструкції мови *Assembler*. Також директиви забезпечують шлях, щоб визначити константи програми та простір для змінних.

У мові *Assembler Ax51* є декілька директив, які дозволяють:

- визначити символьні значення;
- визначити ініціалізацію зберігання;
- контролювати розташування коду.

Директиви не можна переплутати з інструкціями, так як вони не виробляють виконуваний код, за винятком директив **DB**, **DW**, **DD**. Ці три директиви дозволяють:

- змінювати стан мови;
- визначати символи користувача;
- додавати інформацію до об'єктних файлів.

В якості приклада, в таблиці 2 наведено опис деяких характерних ***директив*** *Assembler*.

Таблиця 2 – Опис директив

Директива	Формат	Опис
BIT	symbol BIT bit_address	*Визначає бітову адресу в бітовому просторі даних*
BSEG	BSEG [absolute address]	*Визначає абсолютний сегмент з бітовим адресним простором*
CODE	symbol CODE code_address	*Призначає символьне ім'я спеціальної адреси у кодовому просторі*
DATA	symbol DATA data_address	*Призначає символьне ім'я до спеціальної адреси даних*
DB	[label:] DB expression [,expression]…	*Генерує список байтових значень*

Елементи управління керують операціями.

Умовні елементи управління (наприклад, **IF**, **ELSE**, **ENDIF** та **ELSEIF**) забезпечують набір операторів, які можна використовувати, щоб *увімкнути* або *вимкнути* частину програми з виконання. Наприклад, після виконання коду програми, котрий наведено нижче, маємо наступний результат: R0 105, R1 100.

```
VAR1 SET 105;
  MOV R0, #VAR1; load variable into the R0
  IF VAR1<200
    MOV R1, #100; load 100 into the R0
  ENDIF
```

Інструкції визначають програмний код, Assembler переводить інструкції у машинний код та зберігає їх у об'єктні файли. Інструкція має наступний вигляд:

[label:] mnemonic [operand] [, operand] [, operand] [; comment] ,

де, мітка (label) – місце (*адреса*) інструкції в програмі.

Операнди мови *Assembler* можуть бути із різними типами адресації (див Табл. 3).

Таблиця 3 – Операнди *Assembler*

Операнд	Опис
Immediate data	*Символи або константи, які використовуються як чисельне значення.*
Direct bit address	*Символи або константи, які є посиланням на бітовий адрес.*
Program addresses	*Символи або константи, які є посиланням на кодовий адрес.*
Indirect addresses	*Непряме посилання до пам'яті, опціонально зі зміщенням.*
Special assembler symbol	*Назва регістрів.*

Приклад програми з використанням інструкцій на мові програмування *Assembler*:

```
  mov R6, #01;
  mov R5, #01;
Loop0:  djnz  R5, Loop1;
  mov R0, #01;
Loop1:  djnz  R6, Loop0;
  mov R6, #01;
Loop2:  mov   R4, #90;
```

Після цього регістри, які задіяні у програмі, мають містить наступне значення:
R0 1, R4 90, R5 0, R6 1.

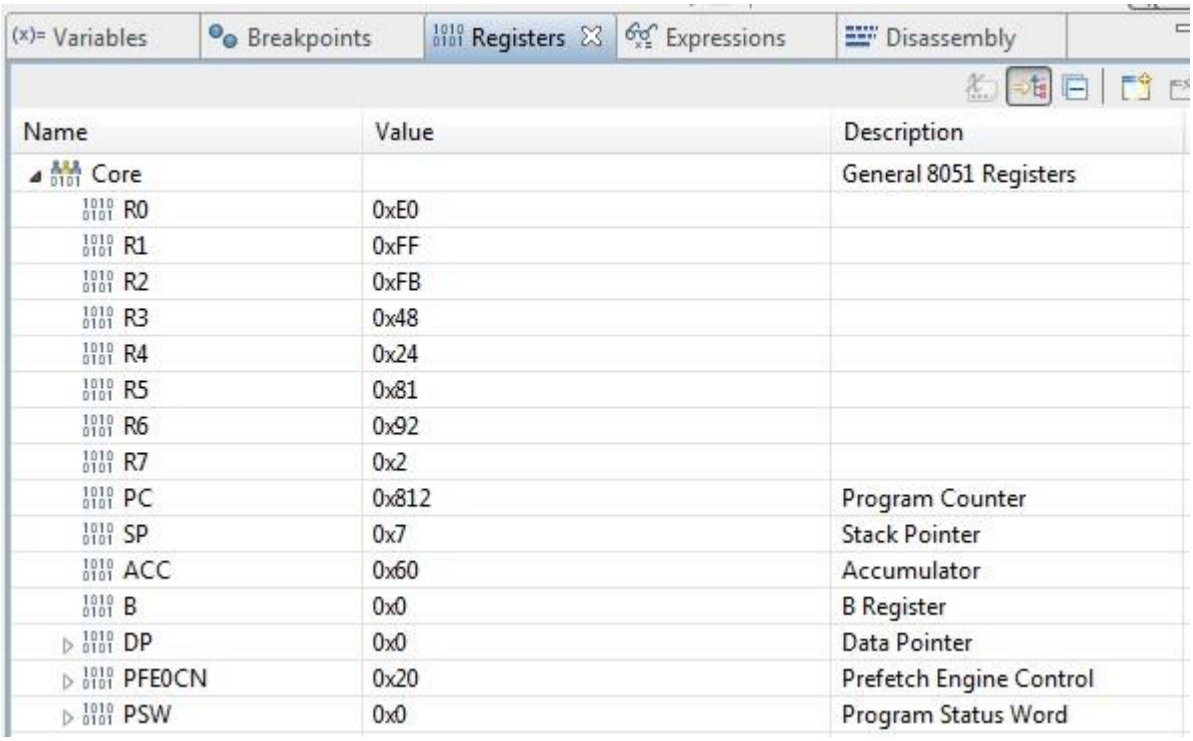
Операнди із прямою адресацією – це чисельні вирази, які кодуються, як частини інструкції. Такі значення використовуються у інструкціях, щоб змінити зміст регістрів або розташування пам'яті. Перед довільним виразом повинен бути символ «#», який використовується, як операнд з прямою адресацією. Приклад використання прямої адресації наведено нижче, а поруч з кодом наведено відповідні коментарі.

У значенні регістра **A** (*ACC accumulator*) можна побачити значення 60 (*див. рис 2а, де дані наведено в шістнадцятирічній системі числення*) або 96 (*рис. 2б, де дані наведено в десятинній системі числення*). 224= 11100000b, 128=10000000b.
$11100000 \oplus 10000000 = 01100000$ (*тут* $\oplus$ - *операція XOR*). $01100000=60h=96_{10}$.

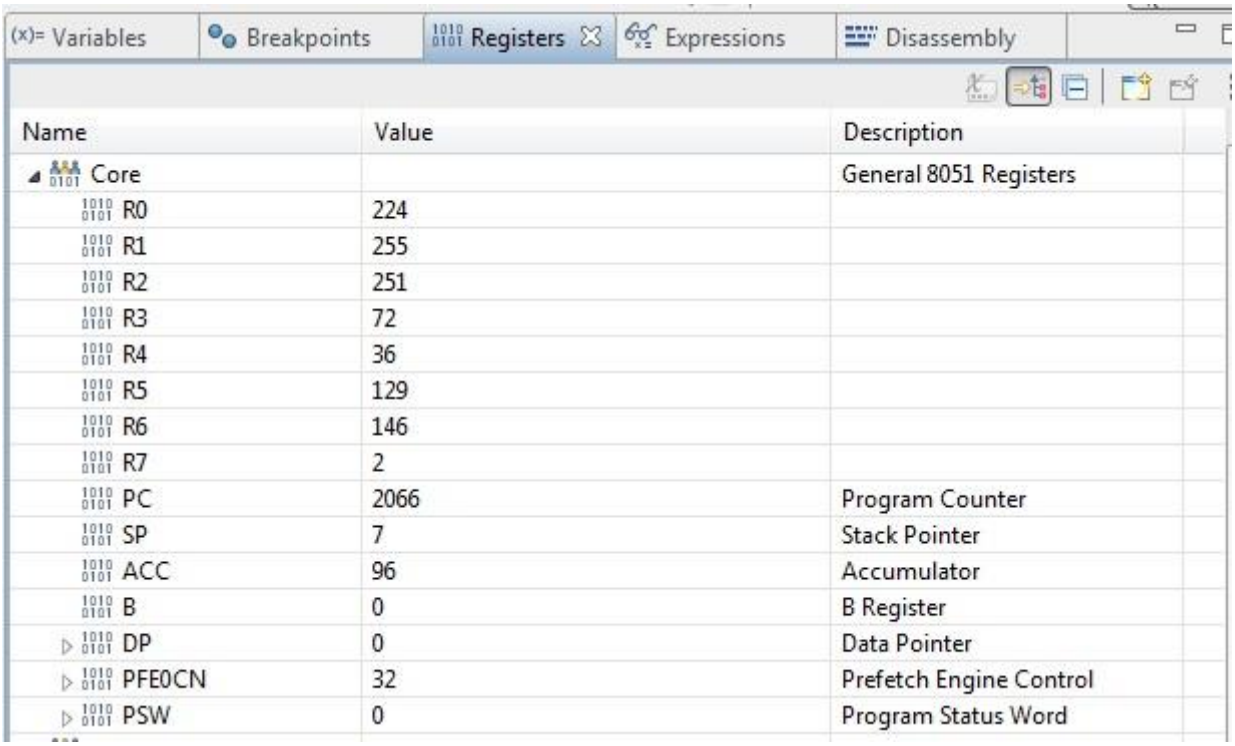
Остання команда (*нижній рядок, «load E0 into R0»*) завантажує значення 0E0H на регістр **R0** (див. рис. 2а) або 224 (на рис.2б).

```
  MOV A,#224; load E0 into the accumulator
```

```
XRL A, #128;  XOR the accumulator with 128
MOV R0, #0E0H;  load E0 into R0
```



а) – значення регістрів у шістнадцятирічному форматі



б) – значення регістрів у десятинному форматі

Рис. 2 – Регістри у середовищі розробки *Simlicity Studio*

Ще один приклад, це застосування інструкції «**MUL**» – множення.

Для цієї операції можна використовувати лише регістри **A** (тобто акумулятор) та **B**. З іншими регістрами така інструкція працювати не буде!

```
mov A, #9;
mov B, #2;
mul AB
```

Після виконання відповідні регістри мають значення:

```
ACC     18      Accumulator   ,
B       0       B Register    .
```

Якщо перемножити числа 204 та 2, то виникає переповнення регістра **A**, тому його значення буде 408-2^8=152 (408_{10}=110011000_2, 10011000_2 - 152_{10}). Натомість у регістрі **PSW** значення **OV** буде дорівнювати 1.

```
mov A, #204;
mov B, #2;
mul AB;
```

Значення регістрів мають відповідні значення:

ACC 152 *Accumulator*
OV 1 - SET (*An overflow occurred*) *Overflow Flag*

Аналогічно є інструкція поділу **DIV**:

```
mov A, #18;
mov B, #2;
div AB;
```

Після виконання, відповідні регістри мають наступне значення:

ACC 9 Accumulator ,
B 0 B Register .

Символьний опис ***директив*** дозволяє створювати символи, які можуть використовуватися у відповідних регістрах, числах та адресах.

Символи визначаються директивами, та не можуть перевизначитися. Виняток складає лише директива **SET**. Так, наприклад, у наступному коді:

```
LIMIT EQU 1200
VALUE EQU LIMIT -200+'A'
ACCU EQU A
COUNT EQU R5
COUNTER SET R1
TEMP SET COUNTER
TEMP SET VALUE*VALUE
```

- змінна «**TEMP**» перевизначається, як «**COUNTER**» або «**VALUE*VALUE».** - З іншими директивами такого робити не можна!

2.4 Приклад роботи зі змінними

Робота зі списками змінних слід виконувати тільки за допомогою директиви типу **DB**. Спостереження цих змінних в області пам'яті мікроконтролера – важлива задача, та перший етап у подальшій роботі з цими змінними. Нижче наведено приклад, який дозволяє формувати списки змінних та розташовувати їх у певній області пам'яті. Результат виконання цих процедур, представлено нижче, на рис. 3.

```
; Змінні знаходяться у пам'яті із адресою від 40H до 4BH
    ll          DATA    40H   ;
    lh          DATA    41H   ;
    U12l        DATA    42H   ;
    U12h        DATA    43H   ;
    U15l        DATA    44H   ;
```

```
        U15h        DATA        45H   ;
        U33l        DATA        46H   ;
        U33h        DATA        47H   ;
        U5l         DATA        48H   ;
        U5h         DATA        49H   ;
        Tc          DATA        4AH   ;
        X1          DATA        4BH   ;       допоміжна константа
; значення констант
Table:              DB    90    ;
                    DB    50    ;
                    DB    25    ;
                    DB    25    ;
                    DB    25    ;
                    DB    25    ;
                    DB    25    ;
                    DB    25    ;
                    DB    125   ;
                    DB    25    ;
                    DB    65    ;

mov DPTR, #Table
mov         X1,#0
mov    R1, #40H
mov         R0,#40h
M1:
        mov    A, #0
        movc   A, @A+DPTR
        mov    @R0,A
        inc DPTR
        inc    X1
        inc    R0
cjne    R0,#4BH,M1
```

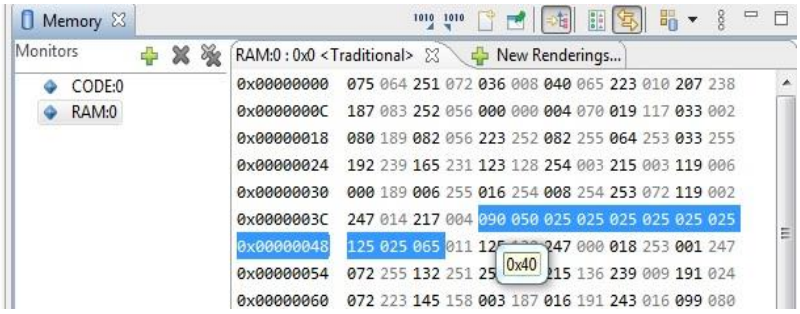


Рис. 3 – Моніторинг пам’яті зі змінними
(ACC 65 Accumulator)

3 Висновки

Робота містить стислий опис використання мови *Assembler ax51* для цілей програмування мікроконтролера C8051F340 у середовищі розробки Simplicity Studio. Звернено увагу на той факт, що всі три джерела [1-3] інформації (*керівництво по Assembler Ax51 та описи мікроконтролера та середовища Simplicity Studio*), змістовно відокремлені один від одного, в наслідок чого, важко орієнтуватися та оцінювати результат роботи команд.

В роботі розглянуто декілька прикладів, які демонструють результати і пояснюють їх роботу з точки зору тестування. Також, запропоновані пояснення, стосовно компонентів, на які поділяються команди, а саме директиви, інструкції та елементи контролю.

Наведено стислий опис регістрів, директив, елементів контролю, інструкцій та роботи зі змінними. Надано приклади використання деяких логічних та арифметичних інструкцій (у вигляді коду), з поясненням результатів виконання, які можна побачити у скріншотах зі значеннями регістрів після виконання цих команд. Додано приклади роботи зі змінними, результати яких фіксуються у пам’яті мікроконтролера.

Посилання

[1] Macro assembler and utilities for 8051 and variants/ [Електронний ресурс] – Режим доступу: https://web.engr.uky.edu/~jel/course/587/datasheets/A51.pdf - 24.08.2021.
[2] Опис мікроконтролера C8051F340/ [Електронний ресурс] – Режим доступу: https://www.silabs.com/documents/public/data-sheets/C8051F34x.pdf - 24.08.2021.
[3] Simplicity studio software/ [Електронний ресурс] – Режим доступу: https://www.silabs.com/developers/simplicity-studio - 24.08.2021.

Рецензент: Владимир Хома, д.т.н., проф., Опольский Политехнический Университет, Ополе, Польша.
E-mail: xoma@wp.pl

Поступила: Сентябрь 2021.

Авторы:
Ольга Мелкозерова, к.т.н., доцент кафедры Безопасности информационных систем и технологий, ХНУ имени В.Н. Каразина, Харьков, Украина.
E-mail: olha.melkozerova@karazin.ua
Ирина Гальцева, ст. преподаватель кафедры Безопасности информационных систем и технологий, ХНУ имени В.Н. Каразина, Харьков, Украина.
E-mail: irina.galceva@karazin.ua

Программирование микроконтроллеров с использованием решений Simplicity Studio, на языке Assembler Ax51.

Аннотация. В статье приведен обзор существующих возможностей программирования на языке Assembler Ax51 микроконтроллера C8051F340 в среде разработке Simplicity Studio. Статья является руководством для практического применения языка Assembler. В руководствах [1-2] довольно много информации, относительно языка Assembler и описания микроконтроллеров C8051F340, однако эта информация отделена друг от друга. Более того, мало объяснений по практическому использованию, вместе со средой разработки Simplicity Studio [3]. Материал содержит объяснение относительно компонентов, на которые делятся команды, а именно директивы, инструкции и элементы контроля. Представлено краткое описание регистров, директив, элементов контроля, инструкций, работы с переменными. Есть примеры использования, в виде кода, некоторых логических и арифметических инструкций с объяснением результатов выполнения, которые можно увидеть в скриншотах со значениями регистров после выполнения команд. Добавлены примеры работы с переменными, результаты которых фиксируются в памяти микроконтроллера.

Ключевые слова: микроконтроллер; директивы; инструкции; элементы контроля; регистры; Assembler.

Reviewer: Volodymyr Khoma, Dr. of Sciences (Eng.), Full Prof., The Opole University of Technology, Opole, Poland.
E-mail: xoma@wp.pl

Received: September 2021.

Authors:
Olha Melkozerova, Ph.D., Associate Professor Department of Security of Information Systems and Technologies, V.N. Karazin Kharkiv National University, Ukraine.
E-mail: olha.melkozerova@karazin.ua
Irina Galceva, Senior Lecturer, Department of Security of Information Systems and Technologies, V.N. Karazin Kharkiv National University, Ukraine.
E-mail: irina.galceva@karazin.ua

Programming of microcontrollers using Simplicity Studio solutions of language Assembler Ax51.

Abstract. The article provides an overview of the existing programming capabilities in the Assembler Ax51 language of the C8051F340 microcontroller in the Simplicity Studio development environment. The article is a guide for the practical application of the Assembler language. Guides [1-2] have a lot of information about the Assembler language and the description of the C8051F340 microcontrollers, but this information is separated from each other. Moreover, there was little explanation for its practical use with the Simplicity Studio development environment [3]. The paper provides explanations of the components into which the teams are divided, namely directives, instructions and controls. A brief description of registers, directives, control elements, instructions, work with variables is given. There are examples of using some logical and arithmetic instructions in the form of code with an explanation of the results of execution, which can be seen in screenshots with the values of the registers after the execution of commands. Added examples of working with variables, the results of which are stored in the memory of the microcontroller.

Key words: Microcontroller; Directives; Instructions; Controls; Registers; Assembler.