

# Labolatorium 1 - Sprawozdanie

Filip Cinik 131734

28-04-2019

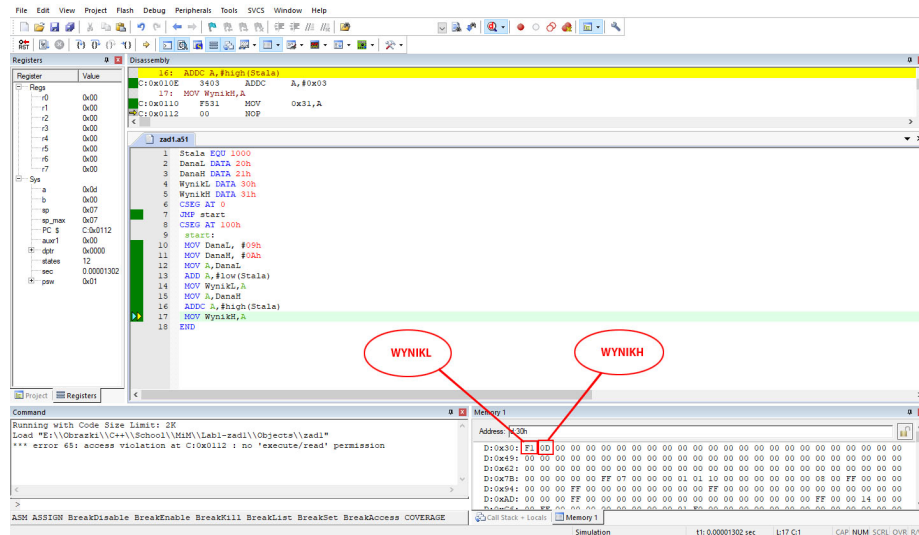
## Spis treści

|          |   |          |
|----------|---|----------|
| <b>1</b> | <b>Zadanie 1</b>                            | <b>2</b> |
| 1.1      | Dodawanie szesnastkowe . . . . .            | 2        |
| 1.2      | Przepełnienie . . . . .                     | 2        |
| 1.3      | Odejmowanie . . . . .                       | 3        |
| <b>2</b> | <b>Zadanie 3 - Generator liczb losowych</b> | <b>5</b> |
| <b>3</b> | <b>Schemat działania</b>                    | <b>5</b> |
| <b>4</b> | <b>Kod</b>                                  | <b>6</b> |

# 1 Zadanie 1

## 1.1 Dodawanie szesnastkowe

Program dodaje niższe bity stałej **Stala** do rejestru **A** a następnie dodaje wyższe bity z przeniesieniem.

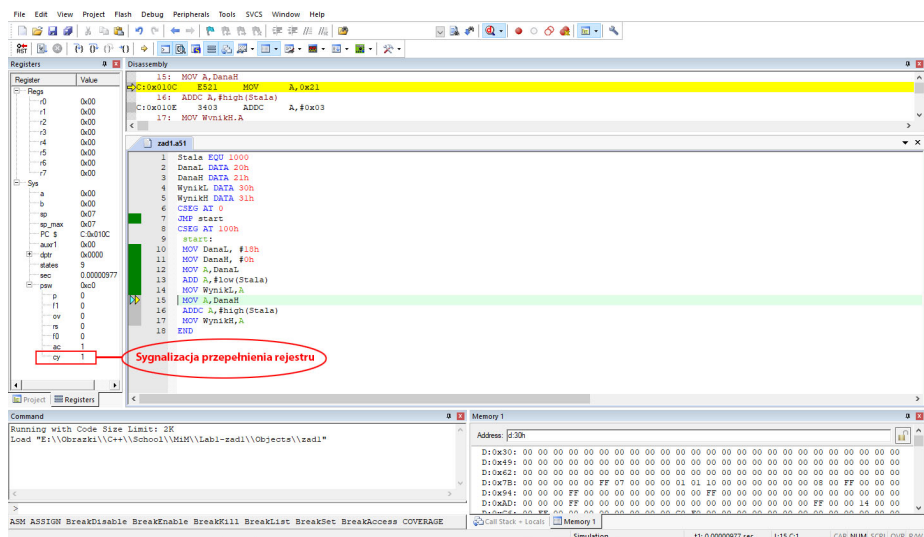


Rysunek 1: Debug programu

Do manipulacji 16 bitowymi liczbami używa się poleceń **#low** i **#high**. Dzięki temu liczbę 16 bitową, nie mieszczącą się w całości w rejestrze, można zamienić na dwie liczby 8 bitowe i manipulować nimi osobno.

## 1.2 Przepelnienie

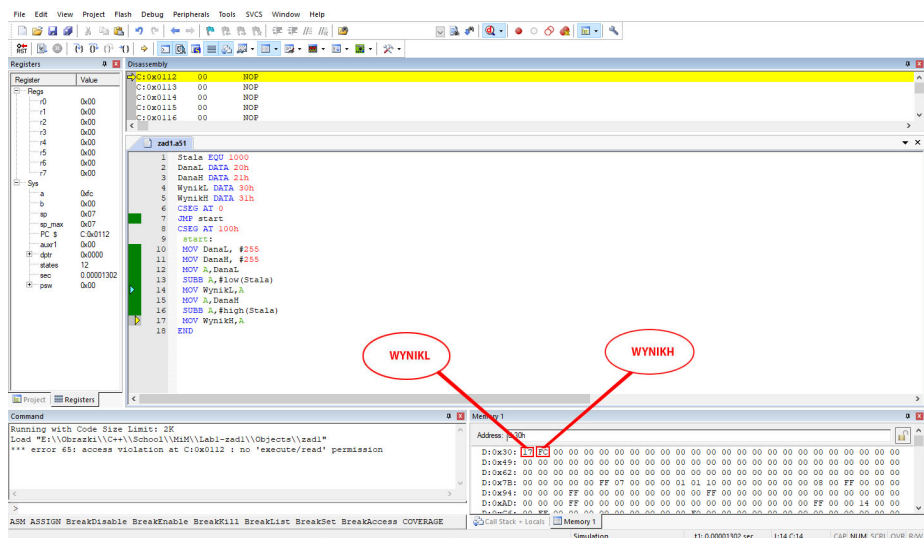
Kiedy do rejestru **A** dodane zostaną wartości większe od 255, następuje przepelnienie, który sygnalizowane jest poprzez ustawienie flagi **CY** na 1. Przepelnienie powoduje zawinięcie się liczb.



Rysunek 2: Przepełnienie rejestru A

### 1.3 Odejmowanie

Odejmowanie liczb z użyciem **SUBB**. Od 16 bitowej liczby **#FFFF** odejmujemy niższe bity stałej **Stala** i zapisujemy je do **WynikL**, a następnie odejmujemy wyższe bity stałej **Stala** i zapisujemy je do **WynikH**.



Rysunek 3: Odejmowanie liczb

Wynik odjętych liczb widać powyżej. W przypadku odjęcia większej liczby od mniejszej, również dostatniemy przepełnienie, które zostanie zasygnalizowane poprzez ustawienie flagi **CY**.

## 2 Zadanie 3 - Generator liczb losowych

Generator liczb losowych został zrealizowany poprzez użycie buforu **Buffer** przechowującego sekwencje 10 elementów w pamięci. Początkowo są używane liczby podane w treści zadania, które następnie są zastępowane przez kolejne wartości generowanej sekwencji pseudo-losowych liczb. Procedura **init** odpowiada za inicjalizację tych danych.

Wzór użyty do wygenerowania pseudo-losowej sekwencji liczb to ALFG - Addytywny Opóźniony Generator Fibonacciego (ang. *Additive Lagged Fibonacci Generator*)

$$x_n = (x_{n+j} + x_{n+k}) * \text{mod} m$$

$$j = 7, k = 10$$

Gdzie  $j$ ,  $k$  i  $n$  odpowiadają **J**, **K**, **N** w kodzie programu.

## 3 Schemat działania

Program uruchamia się w nieskończonej pętli wywołując procedurę **random**, która umieszcza nową liczbę pseudolosową w rejestrze **A** oraz zapisuje ją do buforu **Buffer**.

Procedura **random** w pierwszej kolejności upewnia się, że zmienne **J**, **K**, **N** są w zakresie 0-9 poprzez użycie reszty z dzielenia. Dzięki temu upewnia się, że nie przekroczymy zakresu buforu. Następnie umieszcza w rejestrach **R0** i **R1** wartości **J** i **K**. Odbywa się to poprzez dodanie do adresu buforu offsetu, który wskaże na pozycje danej zmiennej w buforze.

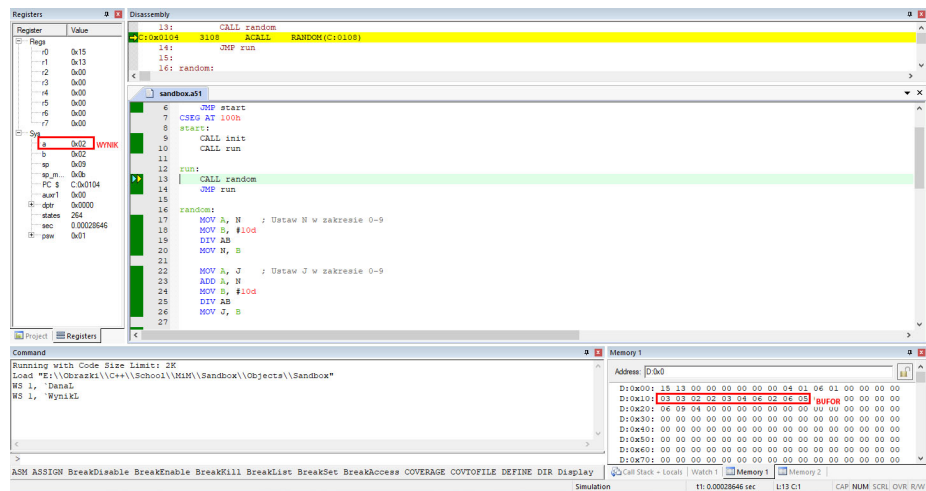
Wartości z rejestrów **R0** i **R1** są sumowane aby otrzymać pierwszą część równania  $(x_{n+j} + x_{n+k})$ . Następnie liczone jest modulo poprzez umieszczenie wyniku dodawania w rejestrze **A** oraz liczby **4** w rejestrze **B** co daje nam drugą część równania  $\text{mod} m$ .

Ostatnim krokiem jest znalezienie pozycji zmiennej **N** w buforze poprzez, tak jak poprzednio, dodanie do adresu buforu wartości **N**, która jest jakby aktualnym indeksem naszej tablicy **Buffer**.

Nowa liczba pseudo-losowa jest przeniesiona do rejestru **A**, a wartości **J**, **K**, **N** są inkrementowane o 1.

| t | j | k | n | Sekwencja |          |          |   |   |   |   |   |   |   |
|---|---|---|---|-----------|----------|----------|---|---|---|---|---|---|---|
| 0 | 6 | 9 | 0 | 4         | 5        | 3        | 3 | 3 | 4 | 6 | 2 | 6 | 5 |
| 1 | 8 | 1 | 1 | <b>3</b>  | 5        | 3        | 3 | 3 | 4 | 6 | 2 | 6 | 5 |
| 2 | 1 | 4 | 2 | 3         | <b>3</b> | 3        | 3 | 3 | 4 | 6 | 2 | 6 | 5 |
| 3 | 5 | 8 | 3 | 3         | 3        | <b>2</b> | 3 | 3 | 4 | 6 | 2 | 6 | 5 |
| 4 | 0 | 3 | 4 | 3         | 3        | <b>2</b> | 3 | 4 | 6 | 2 | 6 | 5 |   |

Powyżej zademonstrowano 5 pierwszych cykli generacji liczb. **J** i **K** zawijają się po przekroczeniu rozmiaru buforu.



Rysunek 4: Najważniejsze elementy pamięci

## 4 Kod

Buffer DATA 10h

J DATA 20h

K DATA 21h

N DATA 22h

CSEG AT 0

JMP start

CSEG AT 100h

start:

CALL init

CALL run

run:

CALL random

JMP run

random:

MOV A, N ; Ustaw N w zakresie 0-9

MOV B, #10d

DIV AB

MOV N, B

MOV A, J ; Ustaw J w zakresie 0-9

ADD A, N

MOV B, #10d

DIV AB

MOV J, B

```

MOV A, K ; Ustaw K w zakresie 0-9
ADD A, N
MOV B, #10d
DIV AB
MOV K, B

MOV A, #Buffer ; Wez wartosc Buffer + offset
ADD A, J
MOV R0, A

MOV A, #Buffer ; Wez wartosc Buffer + offset
ADD A, K
MOV R1, A

MOV A, @R0 ; Buffer + J
ADD A, @R1 ; A + (Buffer + K)
MOV B, #4d
DIV AB ; Ustaw nowa liczbe w zakresie 0-3

MOV A, #Buffer
ADD A, N
MOV R1, A
MOV @R1, B

MOV A, B ; Ustaw nowa wartosc w pozycji Buffer(n)

INC J
INC K
INC N
RET

init:
MOV J, #6d
MOV K, #9d
MOV Buffer + 0, #4d
MOV Buffer + 1, #5d
MOV Buffer + 2, #3d
MOV Buffer + 3, #3d
MOV Buffer + 4, #3d
MOV Buffer + 5, #4d
MOV Buffer + 6, #6d
MOV Buffer + 7, #2d
MOV Buffer + 8, #6d
MOV Buffer + 9, #5d
RET

END

```

---