MiM_01	Romaniak Hubert Oleniacz Michał	Informatyka niestacjonarna II rok	Semestr letni 2023/24

Zadanie 1

Kod źródłowy

```
 1. Constant        EQU 1000
 2. DataLow         DATA 20h
 3. DataHigh        DATA 21h
 4. ResultLow       DATA 30h
 5. ResultHigh      DATA 31h
 6.
 7. CSEG AT 0h
 8.       JMP start
 9.
10. CSEG AT 100h
11.       start:
12.                MOV       A, DataLow
13.                ADD       A, #LOW(Constant)
14.                MOV       ResultLow, A
15.                MOV       A, DataHigh
16.                ADDC      A, #HIGH(Constant)
17.                MOV       ResultHigh, A
18. END
```

Bez modyfikacji – dodawanie do 0

Analiza działania

1. Stworzenie stałej `Constant` i przypisanie jej wartości 1000_{10} (`0x03E8`)
2. Stworzenie wskaźnika o nazwie `DataLow` na komórkę w wewnętrznej pamięci RAM `0x20`
3. Stworzenie wskaźnika o nazwie `DataHigh` na komórkę w wewnętrznej pamięci RAM `0x21`
4. Stworzenie wskaźnika o nazwie `ResultLow` na komórkę w wewnętrznej pamięci RAM `0x30`
5. Stworzenie wskaźnika o nazwie `ResultHigh` na komórkę w wewnętrznej pamięci RAM `0x31`
6.
7. Dyrektywa oznaczająca, że wszystkie linie kodu poniżej będą przechowane w pamięci kodu zaczynając od adresu `0x0000`
8. Bezwarunkowy skok w kodzie do etykiety `start`
9.
10. Dyrektywa oznaczająca, że wszystkie linie kodu poniżej będą przechowane w pamięci kodu zaczynając od adresu `0x0100`
11. Etykieta `start`
12. Skopiowanie wartości z adresu wskazywanego przez `DataLow` (`[0x20]` → `0x00`) do rejestru A (akumulatora)
13. Dodanie mniej znaczącego bajtu stałej `Constant` (`0xE8`) do wartości w akumulatorze (`0x00`) – wynikiem jest `0xE8`
14. Skopiowanie wartości z akumulatora do adresu wskazywanego przez `ResultLow` (`0x30`)
15. Skopiowanie wartości z adresu wskazywanego przez `DataHigh` (`[0x21]` → `0x00`) do rejestru A (akumulatora)
16. Dodanie bardziej znaczącego bajtu stałej `Constant` (`0x03`) oraz wartości bitu przeniesienia (`[CY/PSW.7]` → `0`) do wartości w akumulatorze (`0x00`) – wynikiem jest `0x03`
17. Skopiowanie wartości z akumulatora do adresu wskazywanego przez `ResultHigh` (`0x31`)
18. Dyrektywa oznaczająca koniec programu

Wartości w pamięci RAM

Komórka pamięci	**0x20**	**0x21**	**0x30**	**0x31**
Przed wykonaniem programu	0x00	0x00	0x00	0x00
Po wykonaniu programu	0x00	0x00	0xE8	0x03

Wnioski

Działanie powyższego programu to dodanie do wartości przechowywanych w komórkach pamięci 0x20-0x21 (0x0000) stałej 1000_{10} (0x03E8) i zapisanie wyniku w komórkach pamięci 0x30-0x31 (0x03E8).

Należy zwrócić uwagę na sposób zapisania liczby 2-bajtowej w pamięci – jest ona zapisana od najmniej znaczącego bajtu, do najbardziej znaczącego. Ten sposób zapisu jest nazywany cienkokońcowością (little endian).

Modyfikacja 1 – dodawanie bez przeniesienia

Po uruchomieniu w trybie debugowania, wartości komórek zostały ręcznie zmienione w poniższy sposób:

- [0x20] → 0x09
- [0x21] → 0x0A

Analiza działania

1. Stworzenie stałej Constant i przypisanie jej wartości 1000_{10} (0x03E8)
2. Stworzenie wskaźnika o nazwie DataLow na komórkę w wewnętrznej pamięci RAM 0x20
3. Stworzenie wskaźnika o nazwie DataHigh na komórkę w wewnętrznej pamięci RAM 0x21
4. Stworzenie wskaźnika o nazwie ResultLow na komórkę w wewnętrznej pamięci RAM 0x30
5. Stworzenie wskaźnika o nazwie ResultHigh na komórkę w wewnętrznej pamięci RAM 0x31
6.
7. Dyrektywa oznaczająca, że wszystkie linie kodu poniżej będą przechowane w pamięci kodu zaczynając od adresu 0x0000
8. Bezwarunkowy skok w kodzie do etykiety start
9.
10. Dyrektywa oznaczająca, że wszystkie linie kodu poniżej będą przechowane w pamięci kodu zaczynając od adresu 0x0100
11. Etykieta start
12. Skopiowanie wartości z adresu wskazywanego przez DataLow ([0x20] → 0x09) do rejestru A (akumulatora)
13. Dodanie mniej znaczącego bajtu stałej Constant (0xE8) do wartości w akumulatorze (0x09) – wynikiem jest 0xF1
14. Skopiowanie wartości z akumulatora do adresu wskazywanego przez ResultLow (0x30)
15. Skopiowanie wartości z adresu wskazywanego przez DataHigh ([0x21] → 0x0A) do rejestru A (akumulatora)
16. Dodanie bardziej znaczącego bajtu stałej Constant (0x03) oraz wartości bitu przeniesienia ([CY/PSW.7] → 0) do wartości w akumulatorze (0x0A) – wynikiem jest 0x0D
17. Skopiowanie wartości z akumulatora do adresu wskazywanego przez ResultHigh (0x31)
18. Dyrektywa oznaczająca koniec programu

Wartości w pamięci RAM

Komórka pamięci	**0x20**	**0x21**	**0x30**	**0x31**
Przed wykonaniem programu	0x09	0x0A	0x00	0x00
Po wykonaniu programu	0x09	0x0A	0xF1	0x0D

Wnioski

Działanie powyższego programu to dodanie do wartości przechowywanych w komórkach pamięci `0x20-0x21` (`0x0A09`) stałej 1000_{10} (`0x03E8`) i zapisanie wyniku w komórkach pamięci `0x30-0x31` (`0x0DF1`).

Modyfikacja 2 – dodawanie z przeniesieniem

Po uruchomieniu w trybie debugowania, wartości komórek zostały ręcznie zmienione w poniższy sposób:

- `[0x20] → 0x18`

Analiza działania

1. Stworzenie stałej `Constant` i przypisanie jej wartości 1000_{10} (`0x03E8`)
2. Stworzenie wskaźnika o nazwie `DataLow` na komórkę w wewnętrznej pamięci RAM `0x20`
3. Stworzenie wskaźnika o nazwie `DataHigh` na komórkę w wewnętrznej pamięci RAM `0x21`
4. Stworzenie wskaźnika o nazwie `ResultLow` na komórkę w wewnętrznej pamięci RAM `0x30`
5. Stworzenie wskaźnika o nazwie `ResultHigh` na komórkę w wewnętrznej pamięci RAM `0x31`
6.
7. Dyrektywa oznaczająca, że wszystkie linie kodu poniżej będą przechowane w pamięci kodu zaczynając od adresu `0x0000`
8. Bezwarunkowy skok w kodzie do etykiety `start`
9.
10. Dyrektywa oznaczająca, że wszystkie linie kodu poniżej będą przechowane w pamięci kodu zaczynając od adresu `0x0100`
11. Etykieta `start`
12. Skopiowanie wartości z adresu wskazywanego przez `DataLow` (`[0x20] → 0x18`) do rejestru A (akumulatora)
13. Dodanie mniej znaczącego bajtu stałej `Constant` (`0xE8`) do wartości w akumulatorze (`0x18`) – wynikiem jest `0x00`; nastąpiło przepełnienie, zatem bit przeniesienia został ustawiony (`[CY/PSW.7] → 1`)
14. Skopiowanie wartości z akumulatora do adresu wskazywanego przez `ResultLow` (`0x30`)
15. Skopiowanie wartości z adresu wskazywanego przez `DataHigh` (`[0x21] → 0x00`) do rejestru A (akumulatora)
16. Dodanie bardziej znaczącego bajtu stałej `Constant` (`0x03`) oraz wartości bitu przeniesienia (`[CY/PSW.7] → 1`) do wartości w akumulatorze (`0x00`) – wynikiem jest `0x04`; nie nastąpiło przepełnienie, zatem bit przeniesienia został zresetowany (`[CY/PSW.7] → 0`)
17. Skopiowanie wartości z akumulatora do adresu wskazywanego przez `ResultHigh` (`0x31`)
18. Dyrektywa oznaczająca koniec programu

Wartości w pamięci RAM

Komórka pamięci	**`0x20`**	**`0x21`**	**`0x30`**	**`0x31`**
Przed wykonaniem programu	`0x18`	`0x00`	`0x00`	`0x00`
Po wykonaniu programu	`0x18`	`0x00`	`0x00`	`0x04`

Wnioski

Działanie powyższego programu to dodanie do wartości przechowywanych w komórkach pamięci `0x20-0x21` (`0x0018`) stałej 1000_{10} (`0x03E8`) i zapisanie wyniku w komórkach pamięci `0x30-0x31` (`0x0400`).

Nastąpiło tutaj dodawanie z przeniesieniem; po dodaniu mniej znaczących bajtów nastąpiło przepełnienie – oznacza to, że wynik dodawania nie zmieścił się na jednym bajcie, więc została

ustawiona flaga przeniesienia (CY/PSW.7). Podczas dodawania bardziej znaczących bajtów wartość flagi przeniesienia (1) również została dodana. Uwzględnienie bitu przeniesienia podczas dodawania jest zapewnione poprzez zamianę dyrektywy ADD na ADDC podczas drugiego dodawania.

Modyfikacja 3 – odejmowanie z pożyczeniem

Instrukcje ADD oraz ADDC zostały zamienione na instrukcję SUBB.

Po uruchomieniu w trybie debugowania, wartości komórek zostały ręcznie zmienione w poniższy sposób:

- `[0x20] → 0xD9`
- `[0x21] → 0x06`

Analiza działania

1. Stworzenie stałej `Constant` i przypisanie jej wartości 1000_{10} (`0x03E8`)
2. Stworzenie wskaźnika o nazwie `DataLow` na komórkę w wewnętrznej pamięci RAM `0x20`
3. Stworzenie wskaźnika o nazwie `DataHigh` na komórkę w wewnętrznej pamięci RAM `0x21`
4. Stworzenie wskaźnika o nazwie `ResultLow` na komórkę w wewnętrznej pamięci RAM `0x30`
5. Stworzenie wskaźnika o nazwie `ResultHigh` na komórkę w wewnętrznej pamięci RAM `0x31`
6.
7. Dyrektywa oznaczająca, że wszystkie linie kodu poniżej będą przechowane w pamięci kodu zaczynając od adresu `0x0000`
8. Bezwarunkowy skok w kodzie do etykiety `start`
9.
10. Dyrektywa oznaczająca, że wszystkie linie kodu poniżej będą przechowane w pamięci kodu zaczynając od adresu `0x0100`
11. Etykieta `start`
12. Skopiowanie wartości z adresu wskazywanego przez `DataLow` (`[0x20] → 0xD9`) do rejestru A (akumulatora)
13. Odjęcie mniej znaczącego bajtu stałej `Constant` (`0xE8`) oraz bitu pożyczenia (`[CY/PSW.7] → 0`) od wartości w akumulatorze (`0xD9`) – wynikiem jest `0xF1`; nastąpiło niedopełnienie, zatem bit pożyczenia został ustawiony (`[CY/PSW.7] → 1`)
14. Skopiowanie wartości z akumulatora do adresu wskazywanego przez `ResultLow` (`0x30`)
15. Skopiowanie wartości z adresu wskazywanego przez `DataHigh` (`[0x21] → 0x06`) do rejestru A (akumulatora)
16. Odjęcie bardziej znaczącego bajtu stałej `Constant` (`0x03`) oraz wartości bitu pożyczenia (`[CY/PSW.7] → 1`) do wartości w akumulatorze (`0x06`) – wynikiem jest `0x02`; nie nastąpiło niedopełnienie, zatem bit pożyczenia został zresetowany (`[CY/PSW.7] → 0`)
17. Skopiowanie wartości z akumulatora do adresu wskazywanego przez `ResultHigh` (`0x31`)
18. Dyrektywa oznaczająca koniec programu

Wartości w pamięci RAM

Komórka pamięci	**`0x20`**	**`0x21`**	**`0x30`**	**`0x31`**
Przed wykonaniem programu	`0xD9`	`0x06`	`0x00`	`0x00`
Po wykonaniu programu	`0xD9`	`0x06`	`0xF1`	`0x02`

Wnioski

Działanie powyższego programu to odjęcie do wartości przechowywanych w komórkach pamięci `0x20-0x21` (`0x06D9`) stałej 1000_{10} (`0x03E8`) i zapisanie wyniku w komórkach pamięci `0x30-0x31` (`0x02F1`).

Nastąpiło tutaj odejmowanie z pożyczeniem; po odjęciu mniej znaczących bajtów nastąpiło niedopełnienie – oznacza to, że wynik dodawania był ujemny, więc została ustawiona flaga pożyczenia (CY/PSW.7). Podczas odejmowania bardziej znaczących bajtów wartość flagi pożyczenia (1) również została odjęta. Uwzględnienie bitu pożyczenia podczas odejmowania jest zapewnione poprzez użycie dyrektywy SUBB.

Zadanie 2

Niezmodyfikowany program

Kod źródłowy

```
 1. MAIN SEGMENT CODE
 2.
 3. Source          DATA 20h
 4. Destination     DATA 30h
 5. DataAmount      EQU 16
 6.
 7. CSEG AT 0h
 8.       JMP start
 9.
10. RSEG MAIN
11.       start:
12.                MOV       R0, #Source
13.                MOV       R1, #Destination
14.                MOV       R3, #DataAmount
15.       loop:
16.                MOV       A, @R0
17.                MOV       @R1, A
18.                INC       R0
19.                INC       R1
20.                DJNZ      R3, loop
21. END
```

Po uruchomieniu w trybie debugowania, wartości komórek zostały ręcznie zmienione w poniższy sposób:

- [0x20] → 0x00
- [0x21] → 0x01
- [0x22] → 0x02
- [0x23] → 0x03
- [0x24] → 0x04
- [0x25] → 0x05
- [0x26] → 0x06
- [0x27] → 0x07
- [0x28] → 0x08
- [0x29] → 0x09
- [0x2A] → 0x0A
- [0x2B] → 0x0B
- [0x2C] → 0x0C
- [0x2D] → 0x0D
- [0x2E] → 0x0E
- [0x2F] → 0x0F

Analiza działania

1. Deklaracja segmentu typu CODE o nazwie MAIN
2.
3. Stworzenie wskaźnika o nazwie Source na komórkę w wewnętrznej pamięci RAM 0x20

4. Stworzenie wskaźnika o nazwie `Destination` na komórkę w wewnętrznej pamięci RAM `0x30`
5. Stworzenie stałej `DataAmount` i przypisanie jej wartości 16_{10} (`0x0010`)
6.
7. Dyrektywa oznaczająca, że wszystkie linie kodu poniżej będą przechowane w pamięci kodu zaczynając od adresu `0x0000`
8. Bezwarunkowy skok w kodzie do etykiety `start`
9.
10. Dyrektywa oznaczająca, że wszystkie linie poniżej są liniami kodu (tak jak został zadeklarowany segment o nazwie `MAIN`) i będą przechowane w pamięci kodu w miejscu najbardziej optymalnym, wybranym przez asembler
11. Etykieta `start`
12. Wstawienie wartości bezpośredniej `Source` (`0x20`) do rejestru `R0`
13. Wstawienie wartości bezpośredniej `Destination` (`0x30`) do rejestru `R1`
14. Wstawienie wartości bezpośredniej `DataAmount` (`0x10`) do rejestru `R3`
15. Etykieta `loop`
16. Skopiowanie wartości z adresu wskazywanego przez `R0` (w pierwszej iteracji: `[0x20]` → `0x00`; w ostatniej iteracji: `[0x2F]` → `0x0F`) do akumulatora
17. Skopiowanie wartości z akumulatora (w pierwszej iteracji: `0x00`; w ostatniej iteracji: `0x0F`) do adresu wskazywanego przez `R1` (w pierwszej iteracji: `0x30`; w ostatniej iteracji: `0x3F`)
18. Inkrementacja wartości w `R0` (w pierwszej iteracji: z `0x20` do `0x21`; w ostatniej iteracji: z `0x2F` do `0x30`)
19. Inkrementacja wartości w `R1` (w pierwszej iteracji: z `0x30` do `0x31`; w ostatniej iteracji: z `0x3F` do `0x40`)
20. Dekrementacja wartości w `R3` (w pierwszej iteracji: z `0x10` do `0x0F`; w ostatniej iteracji: z `0x01` do `0x00`); jeżeli wartość w `R3` nie jest równa 0, skok do etykiety `loop` (w pierwszej iteracji: `0x0F` ≠ `0`, skok zostaje wykonany; w ostatniej iteracji: `0x00` = `0`, skok nie zostaje wykonany)
21. Dyrektywa oznaczająca koniec programu

Wartości w pamięci RAM

Komórka pamięci	**0x30**	**0x31**	**0x32**	**0x33**	**0x34**	**0x35**	**0x36**	**0x37**
Przed wykonaniem programu	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00
Po wykonaniu programu	0x00	0x01	0x02	0x03	0x04	0x05	0x06	0x07

Komórka pamięci	**0x38**	**0x39**	**0x3A**	**0x3B**	**0x3C**	**0x3D**	**0x3E**	**0x3F**
Przed wykonaniem programu	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00
Po wykonaniu programu	0x08	0x09	0x0A	0x0B	0x0C	0x0D	0x0E	0x0F

Wnioski

Działanie powyższego programu to skopiowanie wartości przechowywanych w komórkach pamięci `0x20-0x2F` do komórek pamięci `0x30-0x3F`.

Warto zauważyć, że do wykonania poniższego zadania została użyta pętla. Do rejestru `R3` została wczytana ilość iteracji pętli, która pod koniec każdego wykonania była dekrementowana. Gdy wartość przechowywana w tym rejestrze osiągnęła wartość `0`, pętla została przerwana.

Dodatkowo, przy przepisywaniu elementów, wartości w rejestrach `R0` i `R1` są inkrementowane tak, aby adresy wskazywane przez nie mogły się przesuwać do kolejnych elementów (przypomina to arytmetykę na wskaźnikach w języku programowania C).

Zmodyfikowany program – kopiowanie wartości z pamięci programu do RAM

Kod źródłowy

```
 1. MAIN SEGMENT CODE
 2.
 3. Destination     DATA 20h
 4. DataAmount      EQU 5
 5.
 6. CSEG AT 0h
 7.       JMP start
 8.
 9. RSEG MAIN
10.       start:
11.                 MOV       DPTR, #hardcoded_data
12.                 MOV       R0, #Destination
13.                 MOV       R1, #DataAmount
14.       loop:
15.                 MOV       A, #0h
16.                 MOVC      A, @A+DPTR
17.                 MOV       @R0, A
18.                 INC       DPTR
19.                 INC       R0
20.                 DJNZ      R1, loop
21.
22.                 SJMP      $
23.
24.       hardcoded_data:
25.                 DB        11, 21, 4, 18, 8
26. END
```

Analiza działania

1. Deklaracja segmentu typu CODE o nazwie MAIN
2.
3. Stworzenie wskaźnika o nazwie Destination na komórkę w wewnętrznej pamięci RAM 0x20
4. Stworzenie stałej DataAmount i przypisanie jej wartości 5_{10} (0x0005)
5.
6. Dyrektywa oznaczająca, że wszystkie linie kodu poniżej będą przechowane w pamięci kodu zaczynając od adresu 0x0000
7. Bezwarunkowy skok w kodzie do etykiety start
8.
9. Dyrektywa oznaczająca, że wszystkie linie poniżej są liniami kodu (tak jak został zadeklarowany segment o nazwie MAIN) i będą przechowane w pamięci kodu w miejscu najbardziej optymalnym, wybranym przez asembler
10. Etykieta start
11. Wstawienie adresu etykiety hadrcoded_data jako wartość bezpośrednia do rejestru DPTR
12. Wstawienie wartości bezpośredniej Destination (0x20) do rejestru R0
13. Wstawienie wartości bezpośredniej DataAmount (0x05) do rejestru R1
14. Etykieta loop
15. Wstawienie wartości bezpośredniej 0x00 do akumulatora
16. Skopiowanie wartości z adresu w segmencie kodu wskazywanego przez sumę wartości w akumulatorze i rejestrze DPTR (w pierwszej iteracji: [0x00 + 0x??] = [0x??] → 0x0B; w ostatniej iteracji: [0x00 + 0x??] = [0x??] → 0x08) do akumulatora
17. Skopiowanie wartości z akumulatora (w pierwszej iteracji: 0x0B; w ostatniej iteracji: 0x08) do adresu wskazywanego przez R0 (w pierwszej iteracji: 0x20; w ostatniej iteracji: 0x24)
18. Inkrementacja wartości w DPTR (w pierwszej iteracji: z 0x?? do 0x?? + 1; w ostatniej iteracji: z 0x?? + 4 do 0x?? + 5)

19. Inkrementacja wartości w R0 (w pierwszej iteracji: z 0x20 do 0x21; w ostatniej iteracji: z 0x24 do 0x25)
20. Dekrementacja wartości w R1 (w pierwszej iteracji: z 0x05 do 0x04; w ostatniej iteracji: z 0x01 do 0x00); jeżeli wartość w R1 nie jest równa 0, skok do etykiety loop (w pierwszej iteracji: 0x04 ≠ 0, skok zostaje wykonany; w ostatniej iteracji: 0x00 = 0, skok nie zostaje wykonany)
21.
22. Skok do bieżącej linii kodu; nieskończona pętla oznaczająca zakończenie programu
23.
24. Etykieta hadrcoded_data
25. Inicjalizacja w segmencie kodu, rozpoczynając od miejsca wskazywanego przez etykietę hadrcoded_data, wartości 11 (0x0B), 21 (0x15), 4 (0x04), 18 (0x12) i 8 (0x08)
26. Dyrektywa oznaczająca koniec programu

Wartości w pamięci RAM

Komórka pamięci	**0x20**	**0x21**	**0x22**	**0x23**	**0x24**
Przed wykonaniem programu	0x00	0x00	0x00	0x00	0x00
Po wykonaniu programu	0x0B	0x15	0x04	0x12	0x08

Wnioski

Działanie powyższego programu to skopiowanie wartości przechowywanych w komórkach pamięci w segmencie kodu (0x??-0x?? + 4 – wartości nie są znane na etapie kompilacji, ponieważ asembler automatycznie zaalokował w pamięci najbardziej optymalne miejsce dla kodu) do komórek pamięci 0x20-0x24.

W pętli wykonującej kopiowanie została użyta instrukcja MOVC – wykonuje ona kopiowanie z pamięci z segmentu kodu do akumulatora.

Do inicjalizacji wartości w pamięci kodu została użyta dyrektywa DB.

Zadanie 3

Stworzyć generator liczb pseudolosowych oparty o opóźniony ciąg Fibonacciego z operacją dodawania dla liczb $j = 7$, $k = 10$, $m = 4$ korzystając z liczb startowych $4, 5, 3, 3, 3, 4, 6, 2, 6, 5$ według wzoru:

$$S_n \equiv S_{n-j} + S_{n-k} \ (mod\ m) \qquad 0 < j < k$$

Kod źródłowy

```
 1. MAIN SEGMENT CODE
 2. 
 3. J               EQU 7
 4. K               EQU 10
 5. M               EQU 4
 6. BufferStart     EQU 20h
 7. 
 8. BufferEnd       EQU BufferStart+K
 9. J_COMP          EQU K-J
10. Buffer          DATA BufferStart
11. 
12. CSEG AT 0h
13.       JMP start
14. 
15. RSEG MAIN
16. start:
17.                MOV       DPTR, #seed
18.                MOV       R0, #Buffer
19.                MOV       R1, #K
```

```
20.       start_loop:
21.                           CLR       A
22.                           MOVC      A, @A+DPTR
23.                           MOV       @R0, A
24.                           INC       DPTR
25.                           INC       R0
26.                 DJNZ      R1, start_loop
27.                 MOV       R0, #Buffer
28. main_loop:
29.                 CLR       C
30.                 MOV       A, R0
31.                 SUBB      A, #Buffer
32.                 CJNE      A, #J, a_not_equal_j
33.                 JMP       a_equal_or_greater_than_j
34.       a_not_equal_j:
35.                 JC        a_less_than_j
36.       a_equal_or_greater_than_j:
37.                 MOV       A, R0
38.                 SUBB      A, #J
39.                 MOV       R1, A
40.                 MOV       A, @R1
41.                 JMP       after_compare_shift
42.       a_less_than_j:
43.                 MOV       A, R0
44.                 ADD       A, #J_COMP
45.                 MOV       R1, A
46.                 MOV       A, @R1
47.       after_compare_shift:
48.                 ADD       A, @R0
49.       modulo_loop:
50.                           CJNE      A, #M, a_not_equal_m
51.                           JMP       a_equal_or_greater_than_m
52.                 a_not_equal_m:
53.                           JC        a_less_than_m
54.                 a_equal_or_greater_than_m:
55.                           CLR       C
56.                           SUBB      A, #M
57.                           JMP       modulo_loop
58.       a_less_than_m:
59.                 MOV       @R0, A
60.                 INC       R0
61.                 CJNE      R0, #BufferEnd, main_loop
62.                 MOV       R0, #Buffer
63.                 JMP       main_loop
64. seed:
65.       DB 4, 5, 3, 3, 3, 4, 6, 2, 6, 5
66. END
```

Analiza działania

Linie 3-6

Inicjalizacja parametrów generatora oraz położenia w pamięci bufora cyklicznego.

Linie 8-10

Inicjalizacja pomocniczych stałych i zmiennych.

Linie 17-26

Kopiowanie wartości początkowych z pamięci programu do pamięci RAM.

Linia 27

Ustawienie R0 na początku bufora.

Linie 29-31

Odnalezienie obecnej pozycji R0 w stosunku do początku bufora.

Linie 32-46
Odnalezienie i wpisanie do akumulatora wartości S_{n-j}.

Linia 48
Dodanie do wartości S_{n-j} wartości S_{n-k} w akumulatorze.

Linie 49-57
Obliczenie modulo wyrażenia $S_{n-j} + S_{n-k}$.

Linia 59
Wpisanie do bufora cyklicznego w pamięci RAM nowo wylosowanej wartości.

Linie 60-63
Inkrementacja wskaźnika R0, ustawienie go na początku bufora jeśli wyszedł za ostatni indeks w buforze cyklicznym, powrót na początek głównej pętli programu (do linii 28).

Wnioski
Przedstawiony powyżej program przeprowadza generację liczb pseudolosowych opartą o opóźniony ciąg Fibonacciego z operacją dodawania dla dowolnych wartości j, k i m oraz dla dowolnych wartości początkowych.

Jak można zauważyć, w asemblerze można zastosować połączenie prostych dyrektyw aby zbudować bardziej złożone działanie. Głównymi instrukcjami które na to pozwalają są różne odmiany instrukcji JMP – pozwalają one na tworzenie najważniejszych sposobów kontroli przebiegu programu: instrukcji warunkowych oraz pętli.

Dzięki łączeniu ze sobą tych instrukcji możliwe jest wykonanie bardziej skomplikowanych operacji, takich jak modulo, inkrementacja wskaźnika do bufora cyklicznego czy pobieranie danych z bufora cyklicznego z pewnym offsetem.