ELiAK

1. **Przedstaw w standardzie IEEE 754 liczbę $-0,75_{10}$ w pojedynczej i podwójnej precyzji.**

Pojedyncza precyzja: znak 1 bit, wykładnik 8 bitów, mantysa 23 bity.
Obliczenia:

a) $0,75_{10} = 0,11_2$;
b) przesunięcie przecinka o jedno miejsce w prawo (-1). Liczba wygląda teraz następująco $1,1_2$. Aby obliczyć wykładnik należy zsumować -1 oraz 127. $126_{10} = 01111110_2$;
c) w mantysie pomija się jedynkę, która stoi przed przecinkiem, dlatego mantysa wygląda następująco: 1000 0000 0000 0000 0000 000;
d) liczba jest ujemna, dlatego bit znaku przyjmuje wartość 1.

Ostatecznie: $-0,75_{10} = 1\ 01111110\ 10000000000000000000000_{\text{IEEE 754}}$

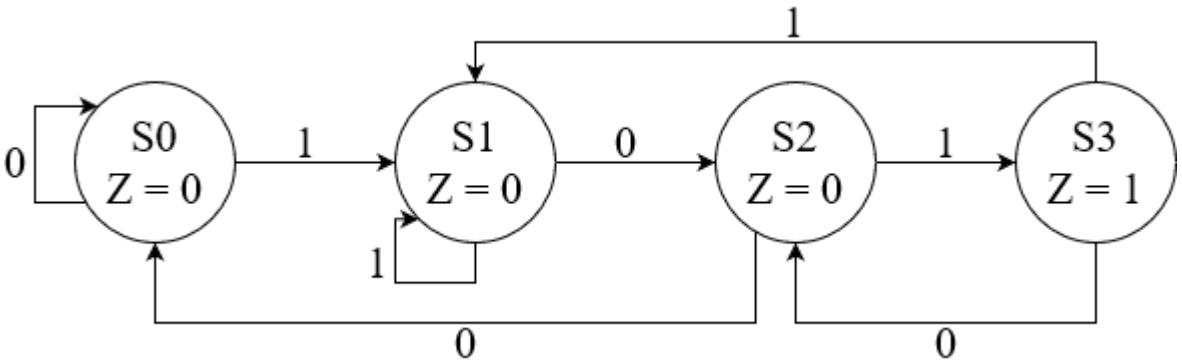
Podwójna precyzja: znak 1 bit, wykładnik 11 bitów, mantysa 52 bity.
Obliczenia wykonuje się w niemalże identyczny sposób. W obliczaniu wykładnika nie będzie się dodawać 127, lecz 1023. Stąd wykładnik wynosi: -1 + 1023 = 1022.
$1022_{10} = 01111111110_2$
Ostatecznie:
$-0,75_{10} = 1\ 01111111110\ 1000000000000000000000000000000000000000000000000000_{\text{IEEE 754}}$

2. **Oblicz wartość liczby w systemie dziesiętnym zapisanej w kodzie uzupełnień do 2, 11010_{U2}.**

Obliczenia:

a) określenie znaku liczby. Pierwszą liczbą jest 1, czyli liczba jest ujemna;
b) negacja bitów. Otrzymana liczba: 00101;
c) dodanie 1. Otrzymana liczba: 00110;
d) przeliczenie na system dziesiętny. Otrzymana liczba: -6.

3. **Zaprojektować synchroniczny układ cyfrowy, który w ciągu zero-jedynkowym, przyłożonym na wejście X, rys. 1, ma wykryć sekwencję bitów (101), co sygnalizuje impulsem na wyjściu, Z=1. Po wykryciu sekwencji układ nie jest resetowany. Stany na wejściu X mogą się zmieniać jedynie między impulsami taktującymi (zegara).**



4. Zaprojektuj blok logiczny zamieniający sumator na jednostkę arytmetyczno-logiczną.

NIESKOŃCZONE

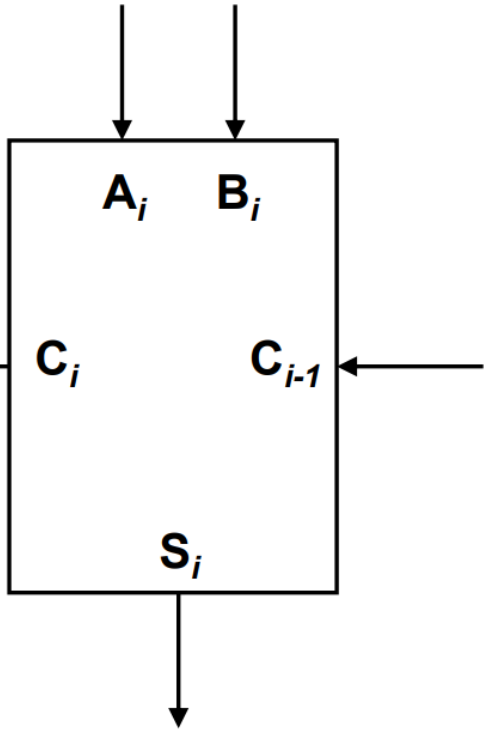
Sumator pełny (Full Adder - FA)
(jednobitowy)

A_i, B_i – wejścia *i*-tch bitów dodawanych argumentów

C_{i-1} – wej. przeniesienia z poprzedniej pozycji

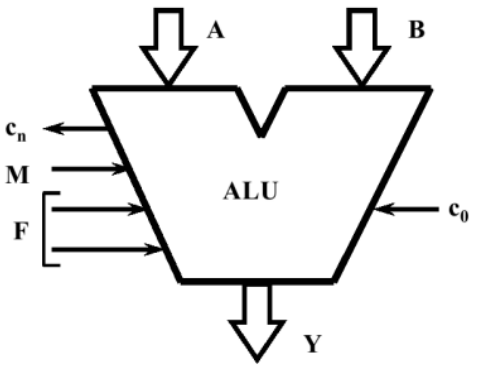
C_i - wyj. przeniesienia do następnej pozycji

S_i – *i*-ty bit wyniku (sumy)



Jednostki arytmetyczno-logiczne (ALU)

- ALU (ang. *Arithmetic Logic Unit*) - jest uniwersalnym układem cyfrowym przeznaczonym do wykonywania operacji arytmetycznych i logicznych
- ALU jest podstawowym blokiem centralnej jednostki obliczeniowej komputera
- Typowe ALU ma dwa wejścia odpowiadające parze argumentów i jedno wyjście na wynik, a operacje jakie prowadzi to:
  - AND, OR, NOT, XOR,
  - sumowanie słów logicznych
  - przesunięcia bitowe o jeden bit, stałą liczbę bitów, czasem też o zmienną liczbę
  - często też: odejmowanie, negacja liczby, dodawanie z przeniesieniem, zwiększanie/zmniejszanie o 1 mnożenie dzielenie/modulo



5. Określ sieć bramek AND i OR realizującą funkcję f(a, b, c, d)=∑m(1, 5, 6, 10, 13, 14).

Kroki potrzebne do rozwiązania zadania:

a) lista mintermów:

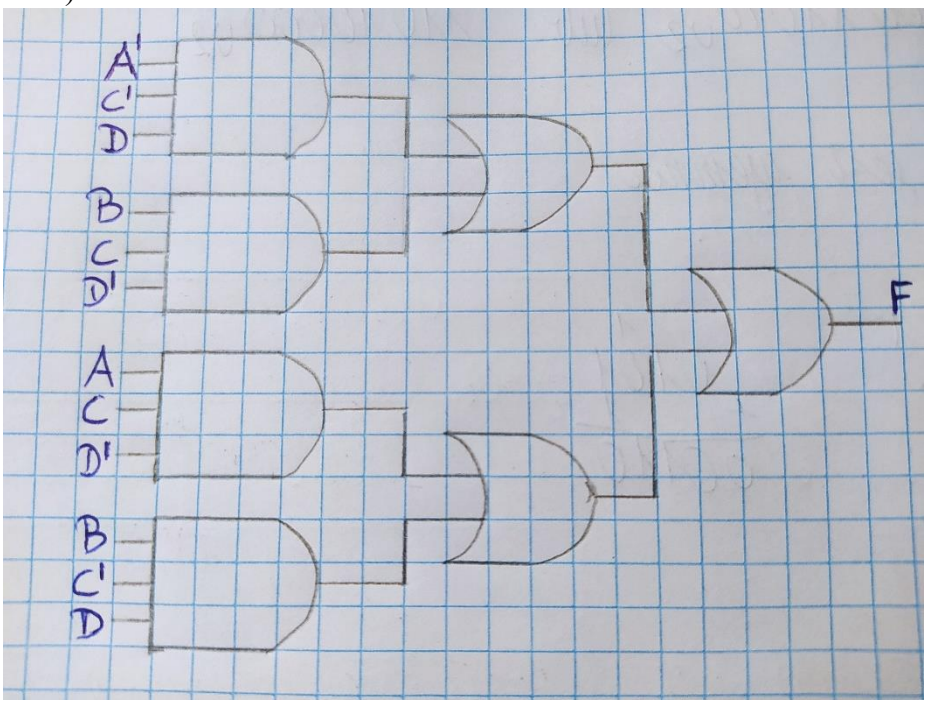
Minterm	a	b	c	d	Pozycja
m_1	0	0	0	1	(00, 01)
m_5	0	1	0	1	(01, 01)
m_6	0	1	1	0	(01, 10)
m_{10}	1	0	1	0	(10, 10)
m_{13}	1	1	0	1	(11, 01)
m_{14}	1	1	1	0	(11, 10)

b) metoda Karnaugha

ab/cd	00	01	11	10
00	0	1	0	0
01	0	1	0	1
11	0	1	0	1
10	0	0	0	1

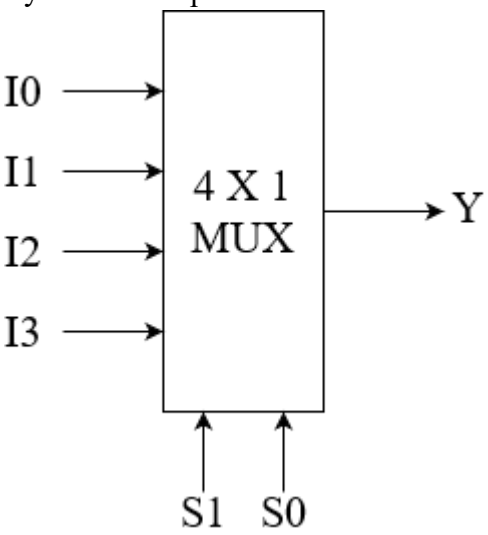
f(a, b, c, d) = a'c'd + bcd' + acd' + bc'd

c) sieć bramek AND i OR



6. **Zaprojektuj multiplekser 4-wejściowy.**

Rysunek multipleksera:



n = 4 – liczba wejść
m = $\log_2$ n – liczba sygnałów sterujących
m = $\log_2$ 4 = 2

Tabela:

S_1	S_0	Y
0	0	I_0
0	1	I_1
1	0	I_2
1	1	I_3

Równanie:
$Y = \overline{S_1}\,\overline{S_0} I_0 + \overline{S_1} S_0 I_1 + S_1 \overline{S_0} I_2 + S_1 S_0 I_3$

7. **Zrealizuj następujące funkcje wyjściowe bazując na ROM (8x4). Y0(A, B, C)=∑m(0, 2, 5); Y1(A, B, C)=∑m(1, 3, 4); Y2(A, B, C)=∑m(2, 4, 7); Y3(A, B, C)=∑m(0, 1, 3, 4, 5).**

Przeliczenie cyfr z systemu dziesiętnego na dwójkowy:
$0_{10} = 0_2$
$2_{10} = 10_2$
$5_{10} = 101_2$
$1_{10} = 1_2$
$3_{10} = 11_2$
$4_{10} = 100_2$
$7_{10} = 111_2$

Tablica prawdy ROM (8x4) dla zdefiniowanych funkcji wyjściowych:

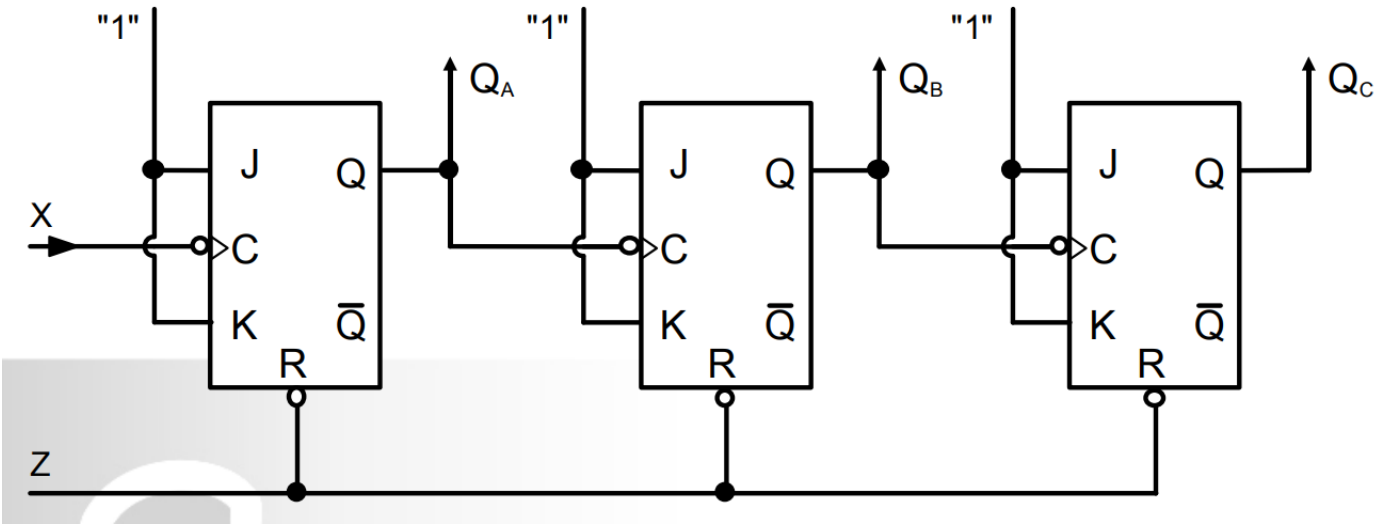
Nr adresu	A	B	C	Y_0	Y_1	Y_2	Y_3
0	0	0	0	1	0	0	1
1	0	0	1	0	1	0	1
2	0	1	0	1	0	1	0
3	0	1	1	0	1	0	1
4	1	0	0	0	1	1	1
5	1	0	1	1	0	0	1
6	1	1	0	0	0	0	0
7	1	1	1	0	0	1	0

Y przyjmuje wartość 1, jeśli zawiera w sobie liczbę odpowiadającą adresowi.

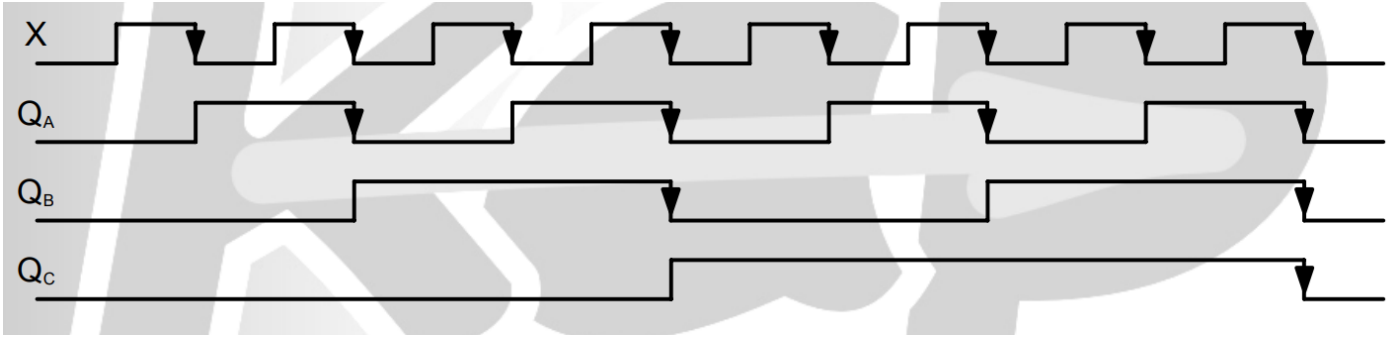
8. Zaprojektuj licznik szeregowy zliczający w przód do 8 na przerzutnikach JK.

Licznikiem nazywamy sekwencyjny układ cyfrowy służący do zliczania i pamiętania liczby impulsów podawanych w określonym przedziale czasu na jego wejście zliczające.

Schemat:



Przebiegi czasowe:



Zliczane impulsy są wprowadzone na wejście zegarowe (x) pierwszego przerzutnika. Wejścia zegarowe kolejnych przerzutników są zwarte z wyjściami Q poprzednich przerzutników. Podanie zera logicznego na wejścia zerujące (R) przerzutników (z=0) umożliwia asynchroniczne wyzerowanie licznika w dowolnej chwili, w czasie jego pracy.

9. **Przeanalizować, czy w układzie kombinacyjnym opisanym funkcją przełączającą $y = x1x2' + x2x3$ wystąpi hazard statyczny.**

Tabela:

x_1	x_2	x_3	y
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	1

Mapa Karnaugha:

x_2x_3/ x_1	00	01	11	10
0	0	0	1	0
1	1	1	1	0

Analiza możliwego hazardu:
Hazard statyczny na mapie Karnaugha występuje tam, gdzie sąsiednie komórki różniące się jednym bitem nie są objęte tą samą grupą. Eliminuje się go poprzez dodanie dodatkowej grupy. Jeśli obok siebie istnieje grupa A oraz grupa B i można ich część połączyć dodatkową grupą to lepiej to zrobić. Nie można zapominać o tym, że siatki Karnaugha się zwijają.
W tym przypadku zmiana wektora wejść z 111 na 101 może spowodować przejściowo poziom niski na wyjściu. Hazard statyczny występuje w tym układzie.
Wzór funkcji, który uwzględnia eliminację hazardu: **$y = x1x2' + x2x3 + x1x3$**

10. Zaprojektuj detektor błędu dla kodu binarnego 6-3-1-1 liczb dziesiętnych. Wyjście F jest 1, gdy na wejściach A, B, C, D jest błąd.

Kod 6-3-1-1 wykorzystuje 4 bity (A, B, C, D), w których wartości mają następujące wagi:
- bit A ma wagę 6,
- bit B ma wagę 3,
- bit C ma wagę 1,
- bit D ma wagę 1.

Kod ten ma odpowiadać liczbie dziesiętnej w zakresie 0 – 9.
Przykład: 1001 = 6 * 1 + 3 * 0 + 1 * 0 + 1 * 1 = 6 + 1 = 7

A	B	C	D	Liczba dziesiętna	Funkcje
0	0	0	0	0	$F_0 = \overline{A} * \overline{B} * \overline{C} * \overline{D}$
0	0	0	1	1	$F_1 = \overline{A} * \overline{B} * \overline{C} * D$
0	0	1	1	2	$F_2 = \overline{A} * \overline{B} * C * D$
0	1	0	0	3	$F_3 = \overline{A} * B * \overline{C} * \overline{D}$
0	1	0	1	4	$F_4 = \overline{A} * B * \overline{C} * D$
0	1	1	1	5	$F_5 = \overline{A} * B * C * D$
1	0	0	0	6	$F_6 = A * \overline{B} * \overline{C} * \overline{D}$
1	0	0	1	7	$F_7 = A * \overline{B} * \overline{C} * D$
1	0	1	1	8	$F_8 = A* \overline{B} * C * D$
1	1	0	0	9	$F_9 = A * B* \overline{C} * \overline{D}$

Detektor błędu:
$F = \overline{F_0 + F_1 + F_2 + F_3 + F_4 + F_5 + F_6 + F_7 + F_8 + F_9}$

Architektura systemów komputerowych

3. Napisz program realizujący kopiowanie danych zdefiniowanych w postaci tablic i realizowanych w pętli. Omów wykorzystanie rejestrów procesora. Odnieś się do sposobu realizacji programu w zależności od jego rozmiaru (modelu). Jaka jest optymalność tego programu. Jak rozumiesz pojęcie adresu i wskaźnika.

Kod programu:

```
.model tiny
.stack 100h             ; Ustawienie rozmiaru stosu na 256 bajtów
.code
org 100h                ; Ustawienie początku kodu na adresie 100h (format COM)

start:
    ; Inicjalizacja segmentu danych
    mov ax, @data         ; Załaduj adres segmentu danych do rejestru AX
    mov ds, ax            ; Ustaw DS na segment danych
    mov es, ax            ; Ustaw ES na segment danych (potrzebne do kopiowania)

    ; Wyświetlenie zawartości tablicy źródłowej przed kopiowaniem
    mov dx, offset new_line    ; Ustaw DX na adres tekstu nowej linii
    mov ah, 9                  ; Funkcja DOS: wyświetl tekst
    int 21h                    ; Wywołanie DOS do wyświetlenia nowej linii

    mov dx, offset source_label ; Ustaw DX na adres etykiety źródłowej
    mov ah, 9                  ; Funkcja DOS: wyświetl tekst
    int 21h                    ; Wywołanie DOS do wyświetlenia etykiety źródłowej

    mov dx, offset source       ; Ustaw DX na adres tablicy źródłowej
    mov ah, 9                  ; Funkcja DOS: wyświetl tekst
    int 21h                    ; Wywołanie DOS do wyświetlenia zawartości tablicy źródłowej

    ; Wyświetlenie zawartości tablicy docelowej przed kopiowaniem
    mov dx, offset new_line    ; Ustaw DX na adres tekstu nowej linii
    mov ah, 9                  ; Funkcja DOS: wyświetl tekst
    int 21h                    ; Wywołanie DOS do wyświetlenia nowej linii

    mov dx, offset dest_label_before ; Ustaw DX na adres etykiety docelowej przed kopiowaniem
    mov ah, 9                  ; Funkcja DOS: wyświetl tekst
    int 21h                    ; Wywołanie DOS do wyświetlenia etykiety docelowej przed kopiowaniem

    mov dx, offset destination  ; Ustaw DX na adres tablicy docelowej przed kopiowaniem
    mov ah, 9                  ; Funkcja DOS: wyświetl tekst
    int 21h                    ; Wywołanie DOS do wyświetlenia zawartości tablicy docelowej przed kopiowaniem

```

```
37
38      ; Ustawienie wskaźników tablic
39      lea si, source              ; Załaduj adres tablicy źródłowej do SI
40      lea di, destination         ; Załaduj adres tablicy docelowej do DI
41      mov cx, source_length       ; Ustaw CX na długość tablicy źródłowej (ilość bajtów do skopiowania)
42
43      ; Kopiowanie danych
44  copy_loop:
45      lodsb                     ; Załaduj bajt z DS:SI do AL (bajt z tablicy źródłowej)
46      stosb                     ; Zapisz bajt z AL do ES:DI (bajt do tablicy docelowej)
47      loop copy_loop            ; Powtarzaj pętlę, dopóki CX > 0 (dekrementuj CX i powtarzaj pętlę)
48
49      ; Wyświetlenie zawartości tablicy docelowej po kopiowaniu
50      mov dx, offset new_line     ; Ustaw DX na adres tekstu nowej linii
51      mov ah, 9                   ; Funkcja DOS: wyświetl tekst
52      int 21h                     ; Wywołanie DOS do wyświetlenia nowej linii
53
54      mov dx, offset dest_label_after ; Ustaw DX na adres etykiety docelowej po kopiowaniu
55      mov ah, 9                   ; Funkcja DOS: wyświetl tekst
56      int 21h                     ; Wywołanie DOS do wyświetlenia etykiety docelowej po kopiowaniu
57
58      mov dx, offset destination  ; Ustaw DX na adres tablicy docelowej po kopiowaniu
59      mov ah, 9                   ; Funkcja DOS: wyświetl tekst
60      int 21h                     ; Wywołanie DOS do wyświetlenia zawartości tablicy docelowej po kopiowaniu
61
62      ; Zakończenie programu
63      mov ax, 4C00h               ; Ustaw AX na kod zakończenia programu (4C00h)
64      int 21h                     ; Wywołanie DOS do zakończenia programu i powrotu do DOS
```

```
65
66  .data
67      new_line db 0Dh, 0Ah, '$'     ; Tekst nowej linii (CR+LF) zakończony znakiem $
68      source db 'Kopiowanie$', 0    ; Tablica źródłowa z danymi do skopiowania, zakończona znakiem $
69      destination db 16 dup('$')    ; Pusta tablica docelowa wypełniona znakami $, zarezerwuj 16 bajtów
70      source_length equ ($ - source - 1) ; Obliczenie długości tablicy źródłowej, bez znaku kończącego $
71
72      ; Napisy informacyjne
73      source_label db 'Zawartosc zrodlowa: $' ; Etykieta dla zawartości źródłowej
74      dest_label_before db 'Zawartosc docelowa przed kopiowaniem: $' ; Etykieta dla zawartości docelowej przed kopiowaniem
75      dest_label_after db 'Zawartosc docelowa po kopiowaniu: $' ; Etykieta dla zawartości docelowej po kopiowaniu
76
77  end start
78
```

Działanie programu:

```
D:\>D:\test

Zawartosc zrodlowa: Kopiowanie
Zawartosc docelowa przed kopiowaniem:
Zawartosc docelowa po kopiowaniu: Kopiowanie
D:\>[_
```

Wykorzystanie rejestrów:
- **AX** do operacji systemowych i przerwań,
- **DS/ES** do zarządzania segmentami danych,
- **SI/DI** do indeksowania tablic,
- **CX** jako licznik pętli,
- **DX** do przechowywania adresów tekstów,
- **AL** do manipulacji pojedynczymi bajtami.

Sposób realizacji programu w zależności od jego rozmiaru (modelu):
W modelu tiny wszystko odbywa się w jednym segmencie, co maksymalnie upraszcza obsługę pamięci, ale jednocześnie ogranicza rozmiar programu.

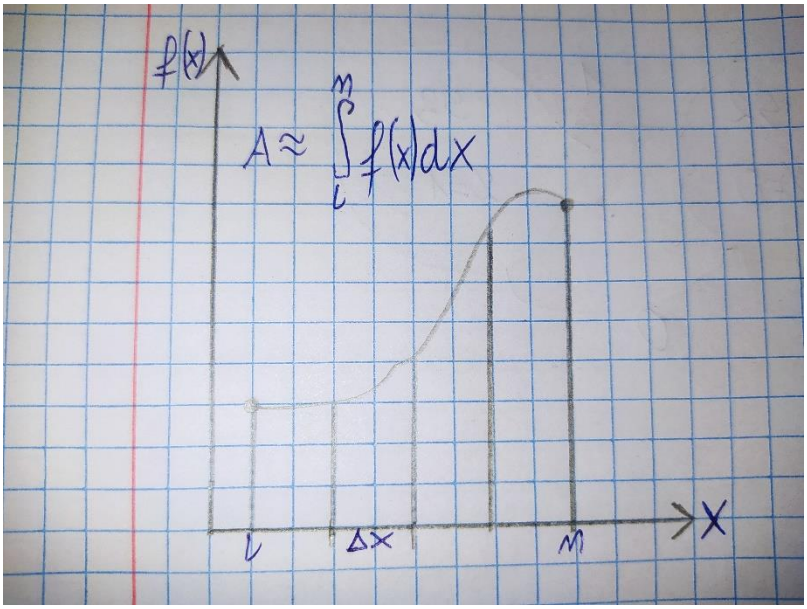
Optymalność programu:
Program jest **optymalny pod względem użycia pamięci**, ponieważ zajmuje mało miejsca i nie wymaga przełączania segmentów. Pętla z użyciem rejestru **CX** (LOOP copy_loop) jest wydajna, ponieważ operacja LOOP automatycznie zmniejsza wartość CX i sprawdza, czy należy kontynuować pętlę. Dzięki temu unika się dodatkowych operacji porównania.

Pojęcie adresu i wskaźnika:
Adres to miejsce w pamięci, które jest identyfikowane przez unikalny numer. W kodzie asemblera adresy są używane do lokalizowania danych i instrukcji w pamięci.
Wskaźnik to zmienna lub rejestr, który przechowuje adres pamięci. Wskaźnik wskazuje na miejsce w pamięci, a operacje mogą być wykonywane na danych znajdujących się pod tym adresem.

Metody numeryczne

10. Opisz i narysuj stosowne ilustracje z oznaczeniami dla metody trapezów i Simpsona przy podziale przedziału całkowania na podprzedziały.

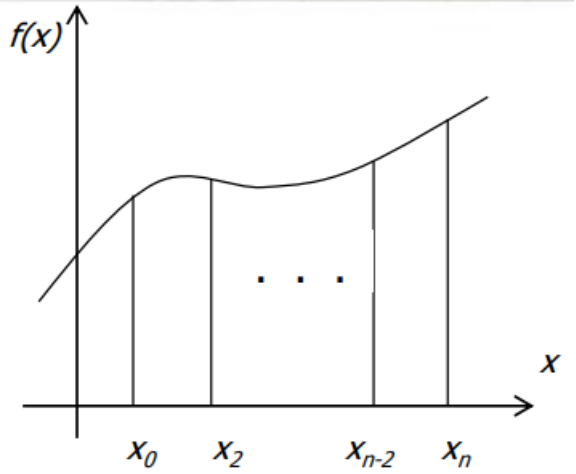
a) metoda trapezów



Ogólny wzór na pole trapezu:
$P = \frac{1}{2} * (a+b) * h$
Dla wzoru metody trapezów:
$h = \Delta x$
$a = f(x_k)$
$b = f(x_{k+1})$

$$P = \frac{1}{2}\Delta x[(f(x_0) + f(x_n)) + 2\sum_{k=1}^{n-1} f(x_k)]$$

b) metoda Simpsona



$$\int_a^b f(x)dx = \frac{b-a}{3n}[f(x_0) + 4\sum_{\substack{i=1 \\ i=np}}^{n-1} f(x_i) + 2\sum_{\substack{i=2 \\ i=p}}^{n-2} f(x_i) + f(x_n)]$$

i = np – „i” ma być nieparzyste

i = p – „i” ma być parzyste