1. **Program na sumę dwóch liczb a i b**

```
__asm{

        pushf
        pusha

        mov eax, a
        add eax, b
        mov a, eax    //wynik 'a+b' pod zmienna 'a'

        popa
        popf
    }
```

2. **Program obliczający funkcję y=ax+b**

```
__asm{

        pushf
        pusha

        mov eax, a          // eax = a
        mul x;             // eax = a*x
        add eax, b        //eax = a*x +b
        mov a, eax       // wynik 'a*x+b' pod zmienna a

        popa
        popf
    }
```

3. **Program mnożący dwie liczby a i b**

```
__asm{

        pushf
        pusha

        mov eax, a;
        mul b;          // imul b to mnożenie ze znakiem
        mov a, eax      // wynik pod zmienna 'a'

        popa
        popf
    }
```

## 4. Program dzielący dwie liczby a i b

```
__asm
{
    pusha
    pushf

    mov eax, a
    mov ebx, b
    xor edx, edx;    // w edx jest reszta z dzielenia
    div ebx;         //idiv ebx to dzielenie przez liczbe ujemna
    mov a, eax;      // przeslanie do 'a' wyniku dzielenia
    mov b, edx;      // przeslanie do 'b' wyniku reszty z dzielenia

      popf
      popa

}
```

## 5. Program obliczający wartość bezwzględną z x

```
__asm{

        pushf
        pusha

        mov eax, x
        xor ebx, ebx;     //wyzerowanie rejestru ebx, forma
                          zastepcza: mov ebx, 0
        cmp ebx, eax      // (0 - x)
        jl lab           // jesli mniejsze ( less) od zera to skocz do
                        // etykiety lab
        neg eax;
        lab: mov x, eax;  // przeslanie zawartosci 'eax' do 'x'

        popa
        popf
}
```

## 6. Program na max wśród dwóch liczb a i b ( program na min analogicznie)

```
__asm{

        pushf
        pusha

        mov eax, a
        mov ebx, b

        cmp ebx, eax   // b - a
        jg lab
        mov a, eax
        jmp finish     //skok bezwarunkowy
        lab: mov a, ebx

         finish:

        popa          //wynik siedzi w zmiennej ‘a’
        popf
}
```

## 7. Program obliczający wartość funkcji:

$$f(x) = \begin{cases} |x| & x \leq 3 \\ x^2 & x > 3 \end{cases}$$

```
__asm{

      pushf
      pusha

      mov eax, x
      mov ebx, b    // ebx = b = 3
      cmp eax, ebx  //x - 3
      jle lab1   // jesli mniejsze lub rowne skocz do lab1
      mul eax
      jmp finish;

      lab1: xor ecx, ecx
        cmp ecx, eax
        jl finish
        neg eax

      finish:
      mov x, eax;            //wynik przesylamy do 'x'

      popa
      popf
}
```

## 8. Program obliczający wartość funkcji:

$$f(x) = \begin{cases} x^2 + 4 & dla\ x = 3 \\ |x| & dla\ pozostałych \\ 5x^2 - 3 & dla\ x = -8 \end{cases}$$

```
__asm{

            pushf
            pusha

            mov eax, x
            mov ebx, b      // ebx = b = 3
            mov ecx, c      // ecx = c = -8

            cmp eax, ebx     // x - b
            je etyk1
            cmp eax, ecx      // x - c
            je etyk4
            cmp eax, 0
            jge finish
            neg eax
            jmp finish

etyk1:
            mul eax
                  add eax, 4
                  jmp finish

etyk4:
            mul eax
                  imul eax, 5   // !!! mul eax, 5 raczej nie przejdzie
                  sub eax, 3
                  jmp finish

                  finish: mov x, eax


            popa
            popf
      }
```

## 9. Program obliczający wartość funkcji:

$$f(x) = \begin{cases} |x| & dla\ x < 3 \\ x + 3 & dla\ x \in [3, 10) \\ x^2 & dla\ x \geq 10 \end{cases}$$

```
__asm{

            pushf
            pusha

            mov eax, x
            mov ebx, b
            mov ecx, c

            cmp eax, ebx    // x - b
            jl etyk1

            cmp eax, ecx    // x - c
            jge etyk2

            add eax, 3
            jmp finish

etyk2:
            mul eax
            jmp finish

etyk1:
            xor edx, edx
            cmp eax, edx
            jge finish
            neg eax

            finish:
            mov x, eax;

            popa
            popf
      }
```

## 10. Program sumujący liczby od 0 do n włącznie:

```
__asm{

            pushf
            pusha

            mov ecx, n    // n - dokad zsuwmowac liczby
            xor eax, eax         // eax = 0
            xor ebx, ebx          // ebx = 0

skacz:
            cmp ebx, ecx    // 0 - n
            jg koniec    // skocz do konca jesli ebx przewyzsza "n"
            add eax, ebx
            inc ebx
            jmp skacz

koniec:
            mov n, eax      // wynik z eax przesylamy do 'n'

            popa
            popf
      }
```

## 11. Program sumujący nieparzyste liczby od 1 do n włącznie:

```
__asm{

            pushf
            pusha

            mov ecx, n;      // n = dokad chcemy zsumowac
            xor eax, eax;

petla:
            add eax, ecx;
            dec ecx
            dec ecx             // ecx - skok o 2 w dol
            cmp ecx, 0
            jg petla
            mov n, eax

            popa
            popf
      }
```

## 12. Program sumujący pierwsze n liczb nieparzystych:

```
__asm{

            pushf
            pusha

            xor edx, edx
            mov eax, n
            xor ecx, ecx
            inc ecx        // ecx =1

            cmp eax, 0    //sprawdzamy czy n=0 czasem?
            je finish

petla:
                  add edx, ecx
                  dec eax
                  inc ecx
                  inc ecx
                  cmp eax, 0
                  jg petla

                  finish: mov n, edx

            popa
            popf
      }
```

## 13. Program sumujący:

$$\sum_{i=1}^{n} (i)^2$$

```
__asm{

            pushf
            pusha

            mov ecx, n;       // n = dokad chcemy zsumowac
            xor ebx, ebx;     //wyzerowanie rejestru gdzie bedzie suma

petla:
            mov eax, ecx;
            mul ecx
            add ebx, eax;

            loop petla;  // loop obniza ecx o 1 i o ile ecx nie
                         // osiagnelo  wartosc 0 skacze de 'petla'
            mov n, ebx

            popa
            popf
      }
```

## 14. Program obliczający silnie z ‘n’:

```
__asm{

            pushf
            pusha

            mov ecx, n
            xor eax, eax
            inc eax        //eax =1

petla:
            cmp ecx, 0 //sprawdzanie czy ecx =0, bo zaraz mnozymy przez ecx
            je finish
            mul ecx
            loop petla

finish:
            mov n, eax;

            popa
            popf

}
```

## 15. Program sumujący jednowymiarową tablicę liczb

```
__asm{

        pushf
        pusha

        xor ebx, ebx; //tu bedzie suma
        mov ecx, n   //rozmiar tablicy
        mov esi, tab   // pod rejestr esi(rejestr indeksowy) adres
                       // tablicy

petla:
        add ebx, [esi+4*ecx-4]   // zaczynamy dodawac
        loop petla // loop - zmniejsza ecx o 1 i dopoki nie osiagnie 0,
                   // siedzi w 'petla'

        mov n, ebx  // przeslanie wyniku pod zmienna n

        popa
        popf

        }
```

## 16. Program sumujący parzyste indeksy jednowymiarowej tablicy

```
__asm{

        pushf
        pusha

        xor ebx, ebx;

        xor ecx, ecx
        inc ecx     //ecx=1
        mov esi, tab

petla:
        add ebx, [esi+4*ecx-4]  // tab[0] + (obieg petli) tab[2] +...

        inc ecx
        cmp ecx, n
        jge finish
        inc ecx
        jmp petla

finish:
        mov n, ebx


        popa
        popf

    }
```

## 17. Program sumujący elementy tablicy, które są parzyste

```
__asm{

            pushf
            pusha

            mov ecx , n
            mov esi , tab
            xor ebx, ebx;

petla:
            mov eax , [ esi + 4 * ecx - 4 ]
            xor edx , edx                    //  wyzerowac edx - bo zaraz
                                bedzie dzielenie, a w edx reszta z dzielenia


            div mod2    //zadeklarowana globalnie zmienna int mod2=2

            xor eax , eax
            cmp edx , eax


            jne skok
            add ebx , [ esi + 4 * ecx - 4 ]

skok:

            loop petla;

            mov n , ebx            //przeslanie wyniku pod n

            popa
            popf

      }
```

## 18. Program szukający max z tablicy

```
__asm{

            pushf
            pusha

            mov ecx, n   //rozmiar tablicy
            mov esi, tab
            mov ebx, [esi+4*ecx-4]
            dec ecx

petla:
      cmp ebx, [esi+4*ecx-4]
      jg niezamieniaj     // jl niezamieniaj – program na min z tablicy
      mov ebx, [esi+4*ecx-4]

niezamieniaj:
      loop petla

            mov n, ebx;    //przeslanie wyniku


            popa
            popf

      }
```

## 19. Program realizujący tab[0]-tab[1]-tab[2]-tab[3]-…

```
   __asm{

            pushf
            pusha

            mov esi, tab
            mov ecx, n
            mov eax, [esi]

            dec ecx; //ecx = n-1

petla:
            sub eax, [esi+ecx*4] //sub - odejmowanie
            loop petla

            mov n, eax

            popa
            popf

      }
```

## 20. Program mnożący skalarnie dwa wektory

```
__asm{

        pushf
        pusha

        mov ecx, n  //rozmiar wektorow ( oczywiscie taki sam)
        mov esi, tab1
        mov edi, tab2
        xor ebx, ebx

lab:
        mov eax, [esi+4*ecx-4]
        mul [edi+4*ecx-4]
        add ebx, eax
        loop lab

        mov n, ebx    // wynik bedzie w 'n'

        popa
        popf

}
```

## 21. Program sumujący wszystkie elementy macierzy

```
__asm{

        pushf
        pusha

        xor ebx, ebx
        mov esi, tab
        mov ecx, m   //ecx= ilosc wierszy macierzy

lab1:
        push ecx      // na stos dajemy ilosc wierszy
        mov edi, [esi+4*ecx-4]
        mov ecx, n  //ecx= ilosc kolumn macierzy

lab2:
        add ebx, [edi+4*ecx-4]
        loop lab2

              pop ecx   // wazna linijka :)

              loop lab1
              mov n, ebx   //wynik siedzi w 'n'

        popa
        popf
}
```

**Uwaga! Aby policzyć sumę głównej przekątnej musimy mieć macierz kwadratową: jest to macierz, która ma taką samą ilość wierszy i kolumn. Macierz kwadratowa o wymiarach m x n – to my wiemy, że m=n**

## 22. Sumowanie głównej przekątnej macierzy

```
__asm{

        pushf
        pusha

        xor ebx, ebx
        mov esi, tab
        mov ecx, m //ecx = ilosc wierszy macierzy = kolumn macierzy

lab1:
        mov edi, [esi+4*ecx - 4]
        add ebx, [edi+4*ecx-4]
        loop lab1

              mov m, ebx;      //wynik siedzi w 'm'

        popa
        popf

}
```

## 23. Sumowanie ‘odwrotnej’ przekątnej macierzy

```
__asm{

        pushf
        pusha

        xor ebx, ebx
        mov esi, tab
        mov ecx, m //ecx = ilosc wierszy macierzy = kolumn macierzy

        xor edx, edx;
        inc edx;     // edx=1
lab1:
        mov edi, [esi+4*ecx - 4]
        add ebx, [edi+4*edx-4]
        inc edx;
        loop lab1

              mov m, ebx;      //wynik siedzi w 'm'

        popa
        popf

}
```

## 24. Ile razy zmienna „wyn” wystąpiła w jednowymiarowej tablicy ?

```
__asm{

        pushf
        pusha

        mov ebx, wyn        // przeslanie zmiennej ‘wyn’ do ebx
        xor eax, eax         // wyzerowanie licznika, gdzie bedzie wynik
        mov esi, tab
        mov ecx, n

petla:
        cmp [esi+4*ecx -4], ebx;
        jne lab
        inc eax

lab:
        loop petla

        mov n, eax;   // wynik siedzi w 'n'

        popa
        popf

}
```

## 25. Ile razy zmienna „wyn” wystąpiła w dwuwymiarowej tablicy (macierzy) ?

```
__asm{

            pushf
            pusha

            mov ebx, wyn
            xor eax, eax //wyzerowanie licznika gdzie bedzie wynik
            mov esi, tab
            mov ecx, m // ecx = ilosc wierszy macierzy

lab1:
            push ecx  // na stos ilosc wierszy macierzy
            mov edi, [esi+4*ecx-4]
            mov ecx, n   // ecx = ilosc kolumn macierzy

lab2:
            cmp [edi+4*ecx - 4], ebx
            jne lab3
            inc eax

lab3:
            loop lab2
            pop ecx            //sciagnij ze stosu ilosc wierszy macierzy
            loop lab1

            mov n, eax  // wynik siedzi pod 'n'


            popa
            popf

      }
```

## 26. Suma dwóch macierzy

```
      __asm{

            pushf
            pusha

            mov ecx, m  //ecx = ilosc wierszy macierzy

lab2:
            mov esi, tab1   //esi=adres 1. macierzy
            mov edi, tab2     //edi=adres 2. macierzy
            mov ebx, tab3   // ebx = adres macierzy wynikowej

            mov esi, [esi+4*ecx-4]
            mov edi, [edi+4*ecx-4]
            mov ebx, [ebx+4*ecx-4]

            push ecx
           mov ecx, n   //ecx = ilosc kolumn
lab1:
      mov eax, [esi+4*ecx-4]
      add eax, [edi+4*ecx-4]
      mov [ebx+4*ecx-4], eax

      loop lab1;

      pop ecx
      loop lab2


            popa
            popf

      }
```

## 27. Mnozenie macierzy razy wektor

**Uwaga: Aby pomnożyć macierz razy wektor ilość kolumn macierzy musi być równa wymiarowi wektora. To znaczy mnożymy macierz o wymiarach ‘m x n’ przez wektor o wymiarze ‘n’. Wynikiem jest wektor o wymiarze m.**

```
__asm{

            mov ecx,m;                          // ecx = ilosc wierszy
macierzy
lab1:
            mov edi,wek;        //adres wektora
            xor ebx, ebx;
            mov esi, tab;            //adres macierzy
            mov esi, [esi+4*ecx-4];   //pokazuje na ostatni wiersz macierzy
//**
            push ecx;              //na stos ecx, czyli ilosc wierszy
macierzy
            mov ecx,w;          // pod ecx wymiar wektora to jest to samo co
ilosc kolumn macierzy
lab2:
            mov eax, [edi+4*ecx-4];   //ostatni element wektora
            mul [esi+4*ecx-4];     //**
            add ebx, eax;
            loop lab2;
            pop ecx; //sciagamy ecx ze stosu po to aby w odpowiednie
miejsce wpisac nasza sume spod ebx
            mov edi, wwyn;       //wwyn to wektor wynikowy zadeklarowany
przed __asm
            mov [edi+4*ecx-4], ebx;
            loop lab1;

}
```

## 28. Szukanie elementu minimalnego z macierzy

```
      __asm{

            mov esi, tab;
            mov ecx, m;   //ilosc wierszy
            mov ebx, n;    //ilosc kolumn
            mov eax, [esi+4*ecx - 4];   // pokazanie ostatniego wiersza
            mov eax, [eax+4*ebx -4];    //pokazanie ostatniego elementu
ostatniego wiersza

lab1:
            push ecx;         //ilosc wierszy na stos, na samym poczatku
np. ecx=3 i to lezy na stosie
            mov edi, [esi+4*ecx - 4];
            mov ecx, n;

lab2:
            cmp eax, [edi+4*ecx - 4];
            jle pomin;
            mov eax, [edi+4*ecx - 4];

            pomin:
            loop lab2;
            pop ecx;
            loop lab1

                  mov wynik, eax;      //wynik siedzi pod 'wynik'
      }
```