Assembler jest językiem niskopoziomowym i musimy nasz kod pisać bardzo dokładnie uwzględniając każdy aspekt. Ten program wyświetli nam napis Hello World oraz zsumuje dwie liczby.

KOD PROGRAMU :

```
org 100h

start:
mov    ah,  9
mov     dx, info
int     21h
mov    eax, 1234
add    eax, 100
xor     esi, esi

petla:
mov     edx,0
mov     ebx,10
div     ebx
add     dx, 48
mov [bufor+si], dl
inc     si
cmp     eax,0
jz      wyswietl
jmp     petla

wyswietl:
mov    dl,[bufor+si-1]
mov     ah, 2
int 21h

dec si
jnz wyświetl

exit:
mov     ax, 4C00h
int     21h

info    db      "Hello World.$"

bufor:
times 40 db 0
```

Start programu:

```
start:
```

Informacja dla kompilatora o początku programu

```
org 100h
```

Na początku naszego programu określamy jego typ. W tym przypadku jest to program wykonywalny. Ta linia również mówi kompilatorowi, że kod będzie znajdował się pod adresem 100h(256 dziesiętnie) w swoim segmencie.

```
mov ah,9
```

Przypisujemy teraz do rejestru AH wartość 9, która jest 9 funkcją (wyświetli nam napis) przy wywołaniu przerwania 21h czyli przerwania programowego.

```
mov     dx, info
```

Do rejestru DX wstaw offset(adres względem początku segmentu) etykiety info. Dzięki czemu później będziemy mogli się do tego odwołać i wyświetlić odpowiedni tekst w tej zmiennej zawarty.

```
int     21h
```

Teraz wywołujemy przerwanie programowe. Ma ono zadeklarowaną wartość 9(wyświetlenie napisu).

```
mov    eax, 1234.
```

Ustawiamy dla rejestru EAX liczby 1234 .

```
add    eax, 100
```

Dodajemy do poprzednio zadeklarowanej wartości liczbę 100.

Czyli 1234+100

ASEMBLER

Funkcja petla:

```
petla:
```

Informacja dla kompilatora o tym gdzie zaczyna się funkcja.

```
mov     edx,0
```

Zerujemy rejestr EDX.

```
mov     ebx,10
```

Teraz do rejestru EBX przypisujemy wartość 10.

```
div     ebx.
```

A następnie ją dzielimy bez znaku.

```
add     dx, 48.
```

Następnie do rejestru DX, który składa się na rejestr EDX wstawiamy wartość 48

```
mov [bufor+si], dl
```

Kopiuje zawartość rejestru DL(starszy bajt rejestru DX) do [bufor+si].

```
inc     si
```

Teraz stosujemy inkrementacje czyli zwiększanie o 1 zmiennej si.

```
cmp     eax,0.
```

Porównuje działanie ale nie zapisuje nigdzie wyniku jedynie zapisuje flagi, porównuje EAX i 0 oraz ustawia odpowiednie flagi .

Teraz w zależności od tego czy spełni się warunek postawiony w poprzedniej komendzie nastąpi skok do funkcji wyświetl lub ponowne zrobienie funkcji petla.

```
jz      wyświetl
```

Skok jeżeli spełni się warunek i rejestr EAX jest równy 0.

```
jmp     petla
```

- wykona skok do etykiety petla -czyli wykona tą petle po raz kolejny, aż do spełnienia się warunku.

Funkcja wyświetl:

```
mov    dl,[bufor+si-1]
```

Kopiowanie wartości rejestru DL do bufor=si.

```
mov     ah, 2
```

Przypisanie wartości 2 do rejestru ah która odpowiada za 2 funkcje przerwania programowego 21h- funkcja 2 ma za zadanie wyświetlić znak.

```
int 21h
```

Następnie wywołujemy przerwanie programowe 21h, które ma przypisaną funkcję 2.

```
dec si
```

Zmniejszamy wartość rejestru si o 1 .

```
jnz wyswietl
```

Skok jeśli nie wyszło 0

```
exit:
```

Informacja dla kompilatora o początku funkcji „exit”

```
mov ax, 4C00h
```

Do rejestru AX przypisujemy wartość 4C00h (w systemie szestnastkowym).

```
int     21h
```

Komenda wywołuje przerwanie programowe o wartości komendy poprzedniej, która odpowiada za zakończenie działania programu.

```
info    db      "Hello World.$";
```

Dzięki tej komendzie po przerwaniu programowym ,które miało funkcje 9 wyświetli nam się napis zapisany w „”.

Znak dolara $ nie jest tam przypadkowo.

Funkcja 9 przerwania DOS wyświetla łańcuch na końcu, którego jest właśnie ten znak.

```
bufor:
```

Informacja dla programu o rozpoczęciu funkcji bufor

```
times 40 db 0
```

Deklarujemy bufor o 40 bajtach i wypełniamy go zerami