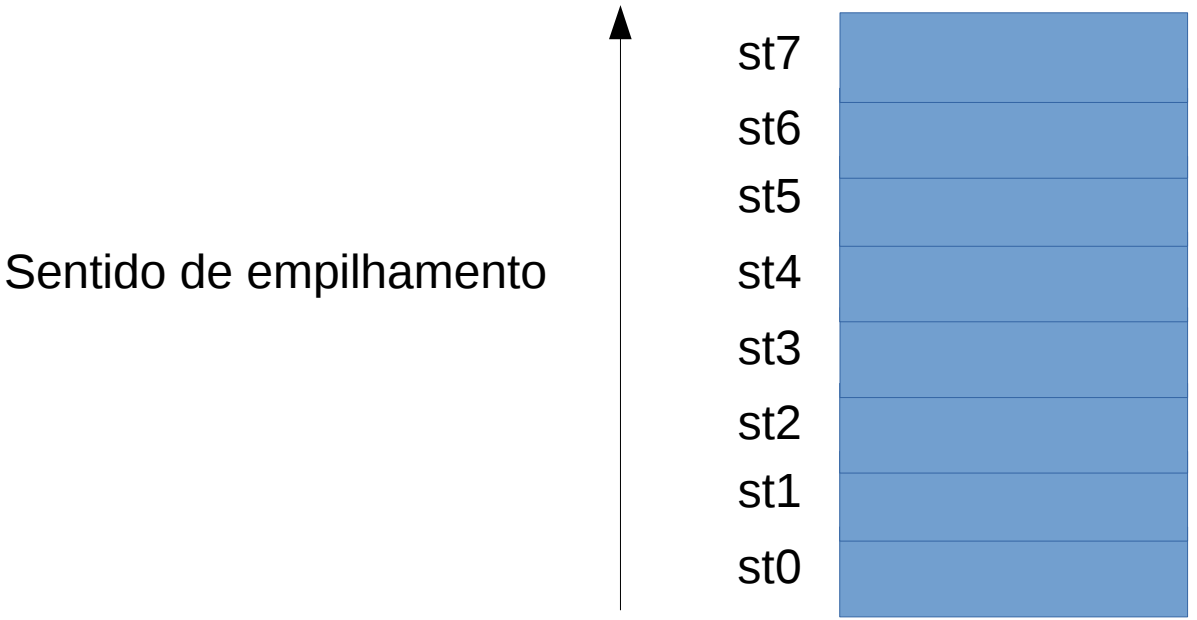
Pontos Flutuantes

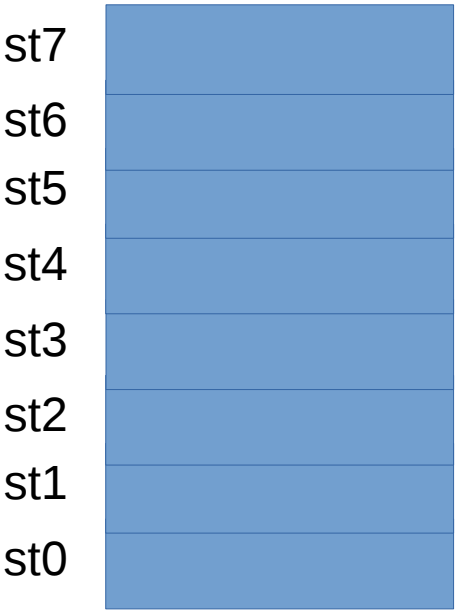
- Carregar ponto flutuante da variável para pilha
- Carregar ponto flutuante da variável para pilha
- Fazer operação
- Retirar resultado da pilha e colocar na variável

Pontos Flutuantes



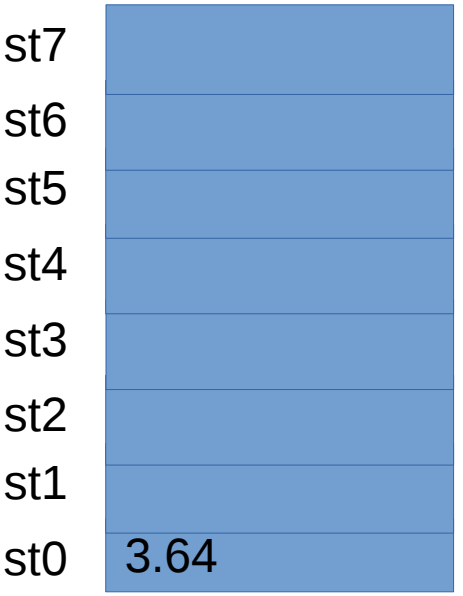
Exemplo

```
SECTION .data
a dq  3.64
b dq  6.25
c dq  0


SECTION .text

fld a
fld b
fadd
fstp c
```



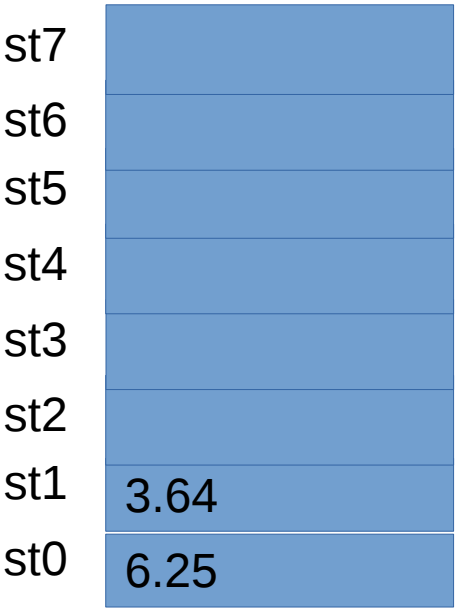
Exemplo

```
SECTION .data
a dq  3.64
b dq  6.25
c dq  0


SECTION .text

fld a
fld b
fadd
fstp c
```



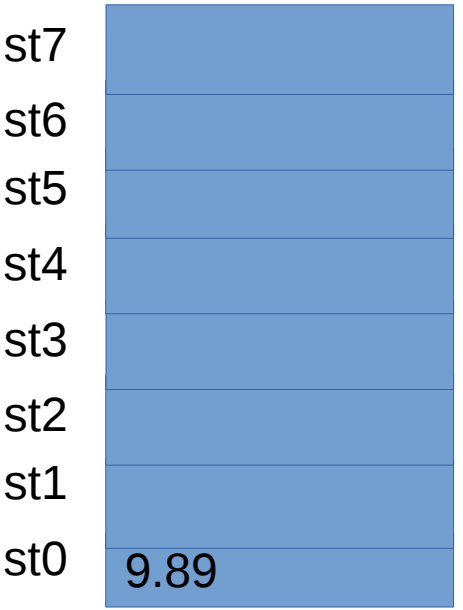
Exemplo

```
SECTION .data
a dq  3.64
b dq  6.25
c dq  0


SECTION .text

fld a
fld b
fadd
fstp c
```



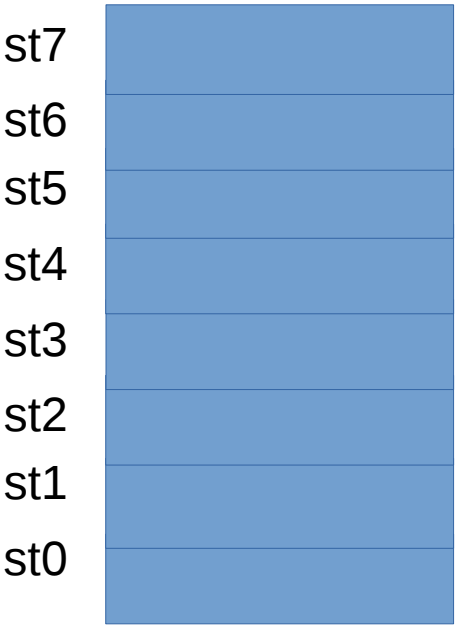
Exemplo

```
SECTION .data
a dq  3.64
b dq  6.25
c dq  0


SECTION .text

fld a
fld b
fadd
fstp c
```



Exemplo

```
SECTION .data
a dq  3.64
b dq  6.25
c dq  9.89


SECTION .text

fld a
fld b
fadd
fstp c
```



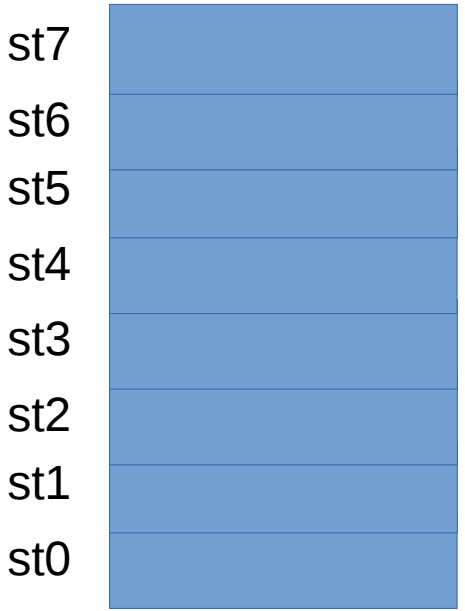
Exemplo

Computar z = (x-1) * (y+3.5), onde x é 1.5 e y é 2.5

```
        SECTION .data

msg     db      "soma = %e",0x0a,0x00
x       dd      1.5
y       dd      2.5
z       dd      0
temp    dq      0
```

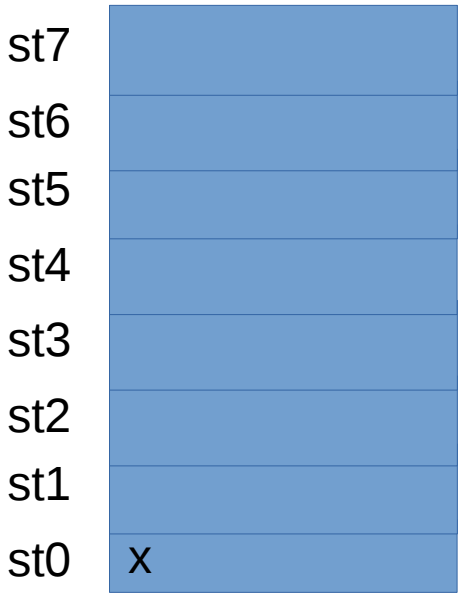
Computar z = (x-1) * (y+3.5), onde x é 1.5 e y é 2.5

```
main:
    fld   dword [x]       ; st0 <- x
    fld1                  ; st0 <- 1 st1 <- x
    fsub                  ; st0 <- x-1

    fld   dword [y]           ; st0 <- y st1 <- x-1
    fld   dword [esp]         ; st0 <- 3.5 st1 <- y st2 <- x-1
    fadd                      ; st0 <- y+3.5 st1 <- x-1
    fmul                      ; st0 <- (x-1) * (y+3.5)
    fst   dword [z]           ; armazena a soma em z

    fld   dword [z]          ; transforma z em 64 bits
    fstp  qword [temp]       ; armazena z como palavra de 64 bits
```



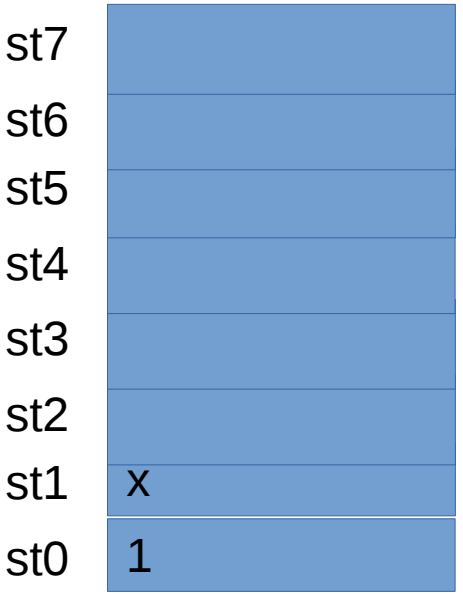
Computar z = (x-1) * (y+3.5), onde x é 1.5 e y é 2.5

```
main:
    fld   dword [x]       ; st0 <- x
    fld1                  ; st0 <- 1 st1 <- x
    fsub                  ; st0 <- x-1

    fld   dword [y]           ; st0 <- y st1 <- x-1
    fld   dword [esp]         ; st0 <- 3.5 st1 <- y st2 <- x-1
    fadd                      ; st0 <- y+3.5 st1 <- x-1
    fmul                      ; st0 <- (x-1) * (y+3.5)
    fst   dword [z]           ; armazena a soma em z

    fld   dword [z]         ; transforma z em 64 bits
    fstp  qword [temp]      ; armazena z como palavra de 64 bits
```



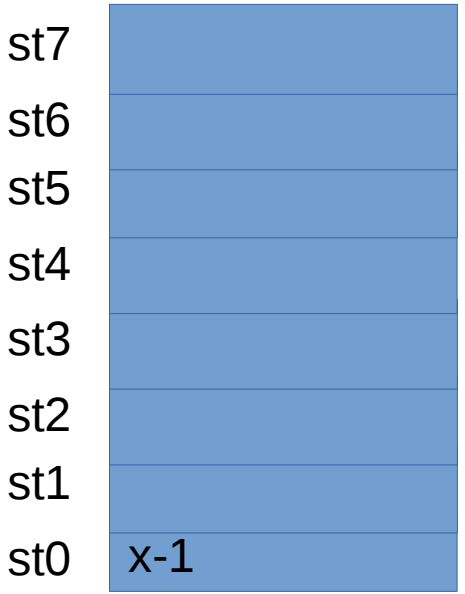
Computar z = (x-1) * (y+3.5), onde x é 1.5 e y é 2.5

```
main:
        fld   dword [x]        ; st0 <- x
        fld1                   ; st0 <- 1 st1 <- x
        fsub                   ; st0 <- x-1

        fld   dword [y]            ; st0 <- y st1 <- x-1
        fld   dword [esp]          ; st0 <- 3.5 st1 <- y st2 <- x-1
        fadd                       ; st0 <- y+3.5 st1 <- x-1
        fmul                       ; st0 <- (x-1) * (y+3.5)
        fst   dword [z]            ; armazena a soma em z

        fld   dword [z]          ; transforma z em 64 bits
        fstp  qword [temp]       ; armazena z como palavra de 64 bits
```



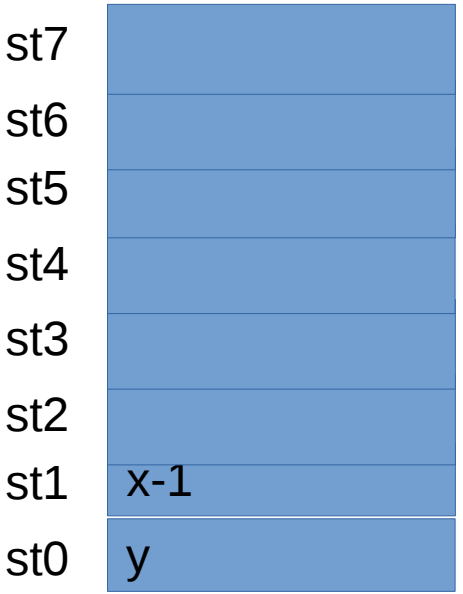
Computar z = (x-1) * (y+3.5), onde x é 1.5 e y é 2.5

```
main:
        fld   dword [x]       ; st0 <- x
        fld1                  ; st0 <- 1 st1 <- x
        fsub                  ; st0 <- x-1

        fld   dword [y]           ; st0 <- y st1 <- x-1
        fld   dword [esp]         ; st0 <- 3.5 st1 <- y st2 <- x-1
        fadd                      ; st0 <- y+3.5 st1 <- x-1
        fmul                      ; st0 <- (x-1) * (y+3.5)
        fst   dword [z]           ; armazena a soma em z

        fld   dword [z]         ; transforma z em 64 bits
        fstp  qword [temp]      ; armazena z como palavra de 64 bits
```



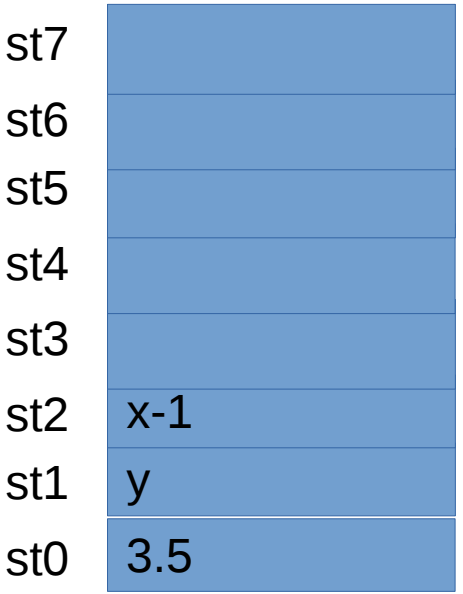
Computar z = (x-1) * (y+3.5), onde x é 1.5 e y é 2.5

```
main:
        fld   dword [x]       ; st0 <- x
        fld1                  ; st0 <- 1 st1 <- x
        fsub                  ; st0 <- x-1

        fld   dword [y]           ; st0 <- y st1 <- x-1
        fld   dword [esp]         ; st0 <- 3.5 st1 <- y st2 <- x-1
        fadd                      ; st0 <- y+3.5 st1 <- x-1
        fmul                      ; st0 <- (x-1) * (y+3.5)
        fst   dword [z]           ; armazena a soma em z

        fld   dword [z]         ; transforma z em 64 bits
        fstp  qword [temp]      ; armazena z como palavra de 64 bits
```



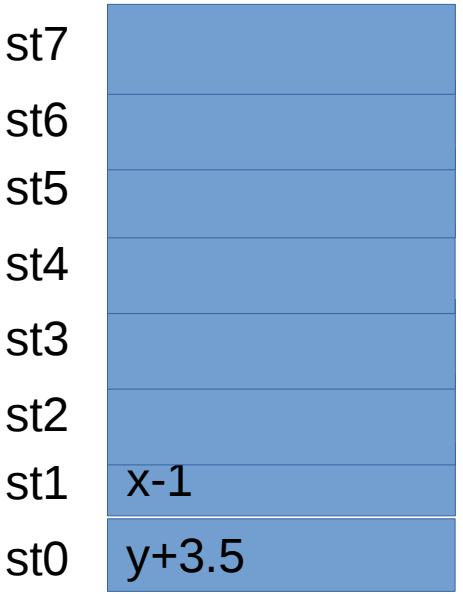
Computar z = (x-1) * (y+3.5), onde x é 1.5 e y é 2.5

```
main:
        fld   dword [x]       ; st0 <- x
        fld1                  ; st0 <- 1 st1 <- x
        fsub                  ; st0 <- x-1

        fld   dword [y]           ; st0 <- y st1 <- x-1
        fld   dword [esp]         ; st0 <- 3.5 st1 <- y st2 <- x-1
        fadd                      ; st0 <- y+3.5 st1 <- x-1
        fmul                      ; st0 <- (x-1) * (y+3.5)
        fst   dword [z]           ; armazena a soma em z

        fld   dword [z]         ; transforma z em 64 bits
        fstp  qword [temp]      ; armazena z como palavra de 64 bits
```



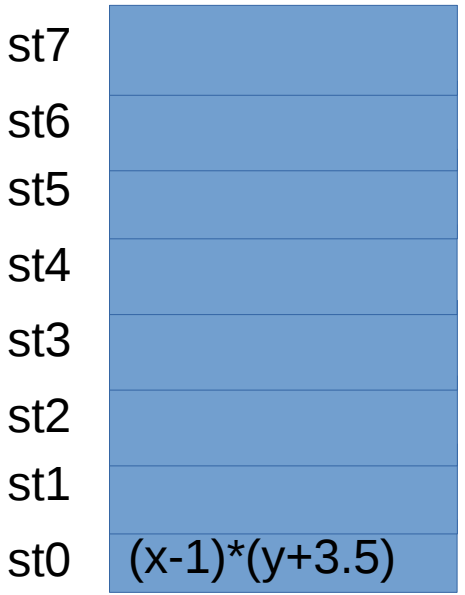
Computar z = (x-1) * (y+3.5), onde x é 1.5 e y é 2.5

```
main:
        fld   dword [x]       ; st0 <- x
        fld1                  ; st0 <- 1 st1 <- x
        fsub                  ; st0 <- x-1

        fld   dword [y]           ; st0 <- y st1 <- x-1
        fld   dword [esp]         ; st0 <- 3.5 st1 <- y st2 <- x-1
        fadd                      ; st0 <- y+3.5 st1 <- x-1
        fmul                      ; st0 <- (x-1) * (y+3.5)
        fst   dword [z]           ; armazena a soma em z

        fld   dword [z]         ; transforma z em 64 bits
        fstp  qword [temp]      ; armazena z como palavra de 64 bits
```



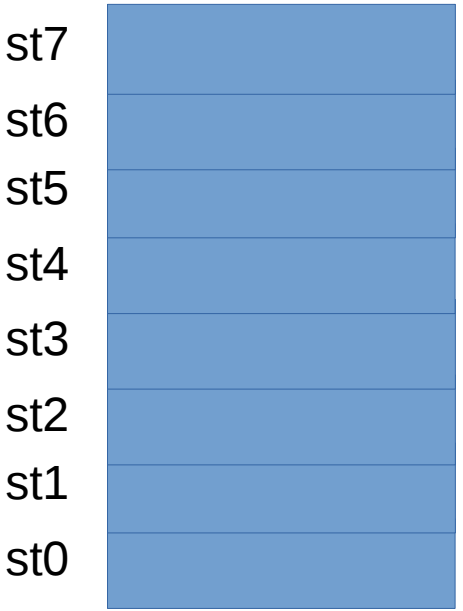
Computar z = (x-1) * (y+3.5), onde x é 1.5 e y é 2.5

```
main:
    fld   dword [x]       ; st0 <- x
    fld1                  ; st0 <- 1 st1 <- x
    fsub                  ; st0 <- x-1

    fld   dword [y]           ; st0 <- y st1 <- x-1
    fld   dword [esp]         ; st0 <- 3.5 st1 <- y st2 <- x-1
    fadd                      ; st0 <- y+3.5 st1 <- x-1
    fmul                      ; st0 <- (x-1) *(y+3.5)
    fst   dword [z]           ; armazena a soma em z

    fld   dword [z]         ; transforma z em 64 bits
    fstp  qword [temp]      ; armazena z como palavra de 64 bits
```



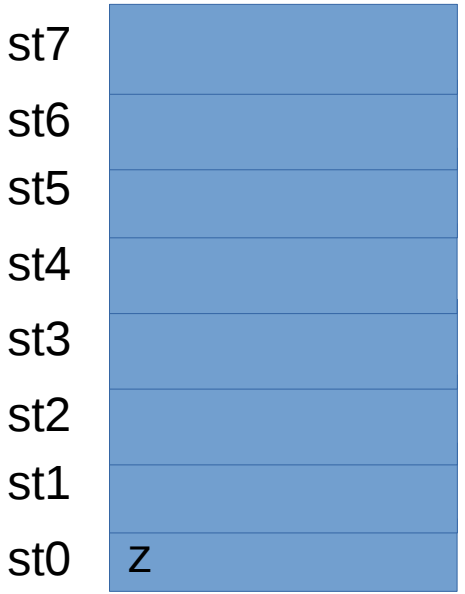
Computar z = (x-1) * (y+3.5), onde x é 1.5 e y é 2.5

```
main:
        fld   dword [x]       ; st0 <- x
        fld1                  ; st0 <- 1 st1 <- x
        fsub                  ; st0 <- x-1

        fld   dword [y]           ; st0 <- y st1 <- x-1
        fld   dword [esp]         ; st0 <- 3.5 st1 <- y st2 <- x-1
        fadd                      ; st0 <- y+3.5 st1 <- x-1
        fmul                      ; st0 <- (x-1) * (y+3.5)
        fst   dword [z]           ; armazena a soma em z

        fld   dword [z]         ; transforma z em 64 bits
        fstp  qword [temp]      ; armazena z como palavra de 64 bits
```



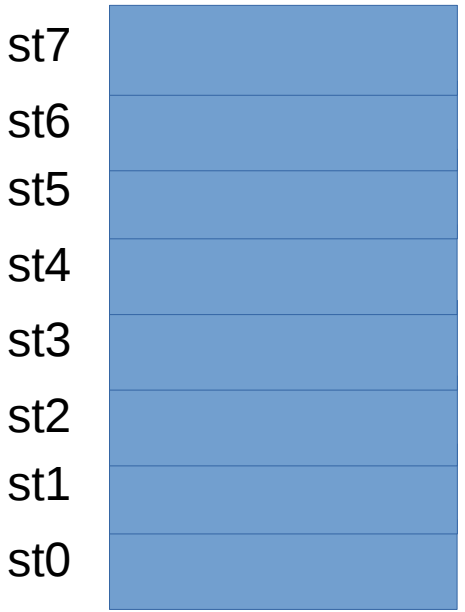
Computar z = (x-1) * (y+3.5), onde x é 1.5 e y é 2.5

```
main:
        fld   dword [x]       ; st0 <- x
        fld1                  ; st0 <- 1 st1 <- x
        fsub                  ; st0 <- x-1

        fld   dword [y]           ; st0 <- y st1 <- x-1
        fld   dword [esp]         ; st0 <- 3.5 st1 <- y st2 <- x-1
        fadd                      ; st0 <- y+3.5 st1 <- x-1
        fmul                      ; st0 <- (x-1) * (y+3.5)
        fst   dword [z]           ; armazena a soma em z

        fld   dword [z]         ; transforma z em 64 bits
        fstp  qword [temp]    ; armazena z como palavra de 64 bits
```



Computar z = (x-1) * (y+3.5), onde x é 1.5 e y é 2.5

```
main:
        fld   dword [x]       ; st0 <- x
        fld1                  ; st0 <- 1 st1 <- x
        fsub                  ; st0 <- x-1

        fld   dword [y]           ; st0 <- y st1 <- x-1
        fld   dword [esp]         ; st0 <- 3.5 st1 <- y st2 <- x-1
        fadd                      ; st0 <- y+3.5 st1 <- x-1
        fmul                      ; st0 <- (x-1) * (y+3.5)
        fst   dword [z]           ; armazena a soma em z

        fld   dword [z]         ; transforma z em 64 bits
        fstp  qword [temp]      ; armazena z como palavra de 64 bits
```



Diferenças entre código 32 e 64 bits

```
printf(“soma = %e\n”, temp);
```

```
msg     db      "sum = %e",0x0a,0x00
```

```
push      dword [temp+4]
push      dword [temp]
 push    dword msg
 call    printf
 add     esp, 12
```

```
mov  rdi, msg
 movq xmm0 , qword [temp]
 mov  rax,1
 call printf
```