assembly xUFC

manual de instruções

- Comandos:

wb: escreve um byte nos dados do programa, pode ser útil para alinhar as words.

```
1   goto main
2   wb 0
```

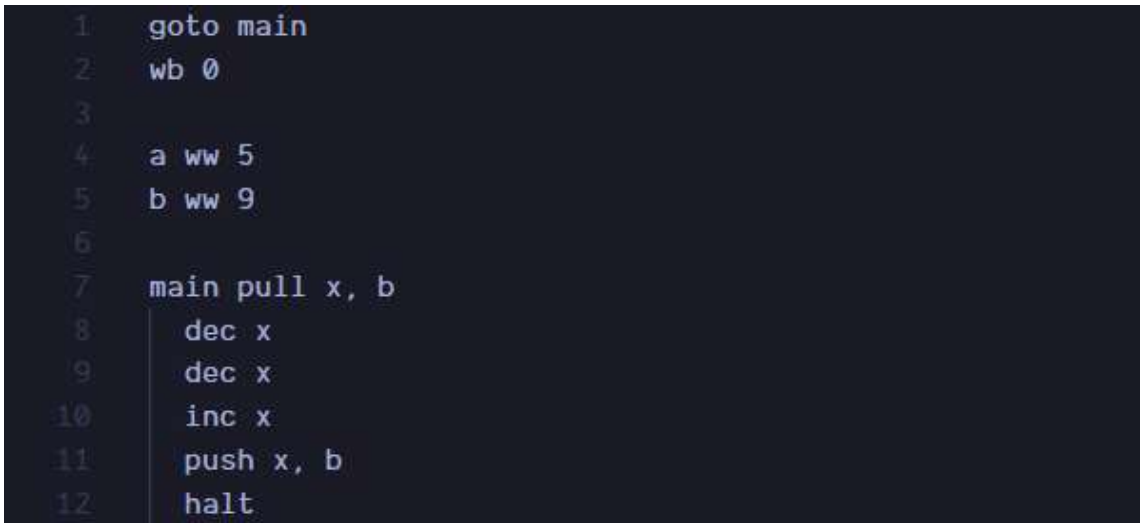
Na linha 2, é escrito um byte de valor 0 para fazer com que as próximas words sejam escritas nos endereços corretos, visto que o comando "goto main", junto com o byte 0 ocupam 3 bytes de memória, sendo necessário mais um byte para preencher a word 1.

ww: escreve uma word nos dados do programa com o valor indicado no parâmetro, seu endereço pode ser atribuído a um nome declarado antes do comando.

```
1   goto main
2   wb 0
3
4   a ww 5
5   b ww 9
```

Na linha 4, será escrita uma word de valor 5 e seu endereço será atribuído ao nome "a", que poderá ser utilizado como parâmetro em outros comandos.

goto addr: faz o programa ir até o endereço passado como parâmetro e executar a partir desse ponto.



```
1   goto main
2   wb 0
3
4   a ww 5
5   b ww 9
6
7   main pull x, b
8     dec x
9     dec x
10    inc x
11    push x, b
12    halt
```

Na linha 7, antes de declararmos o comando "pull x, b" nomeamos essa linha com o nome "main", para, então, utilizarmos esse mesmo nome para nos referirmos a essa linha. Note que, na linha 1, utilizamos o comando "goto main", para fazer o programa saltar

para a linha 7, que nomeamos como "main" anteriormente.

pull x: atribui a x o valor da word no endereço de memória passado como parâmetro.

```
1  goto main
2  wb 0
3
4  a ww 5
5
6  main pull x, a
7    halt
```

Na linha 6, x terá o valor 5.

push x: atribui ao endereço de memória passado como parâmetro o valor atual de x.

```
1  goto main
2  wb 0
3
4  a ww 5
5  b ww 9
6
7  main pull x, a
8    push x, b
9    halt
```

Na linha 8, o valor da word b será 5.

add x e **sub x**: o comando add x, adicionar ao registrador x o valor da word no endereço passado como parâmetro, em contrapartida, o comando sub x, faz a subtração.

```
1   goto main
2   wb 0
3
4   a ww 5
5   b ww 9
6
7   main pull x, a
8     add x, b
9     push x, b
10    halt
```

Na linha 8, é somado ao registrador x o valor 9, contido na variável b.

mul x, **div x** e **mod x**: o comando mul x faz o registrador x receber o produto entre seu valor atual e o valor da word presente no endereço passado como parâmetro, o comando div x realiza a divisão inteira e o comando mod x, realiza a operação de resto da divisão.

```
1   goto main
2   wb 0
3
4   a ww 5
5   b ww 9
6
7   main pull x, b
8     mul x, a
9     push x, b
10    halt
```

Na linha 8, o comando mul x atribui ao registrador x o resultado da multiplicação 5 x 9, ou seja, o produto dos valores contidos no registrador b (atribuído a x através do comando pull x) e no registrador a.

inc x e dec x: o comando inc x não recebe nenhum parâmetro, e incrementa o valor do registrador x em 1, o comando dec x decrementa em 1.

```
goto main
wb 0

a ww 5
b ww 9

main pull x, b
  dec x
  dec x
  inc x
  push x, b
  halt
```

Nas linhas 8, 9 e 10, respectivamente, decrementamos duas vezes o valor 1 de x e, em seguida, incrementamos 1 ao valor de x.

sl x e **sr x**: ambos os comandos não recebem parâmetros e realizam o shift de bits do valor presente no registrador x, o comando sr (shift right) realiza o shift para a direita, equivalente à divisão inteira por 2, o comando sl (shift left) para a esquerda, equivalente à multiplicação por 2.

```
goto main
wb 0

a ww 5
b ww 9

main pull x, b
  sr x
  push x, b
  halt
```

Na linha 8, realizamos o shift de bits do valor atual de x para a direita, ou seja, x recebe o valor da divisão inteira de x por 2.

fact x: o comando fact x não recebe parâmetros e faz x receber o valor do fatorial de x.

```
goto main
wb 0

a ww 5
b ww 9

main pull x, b
  fact x
  push x, b
  halt
```

Na linha 8, x recebe o valor do fatorial de x, ou seja, o valor do fatorial de 9.

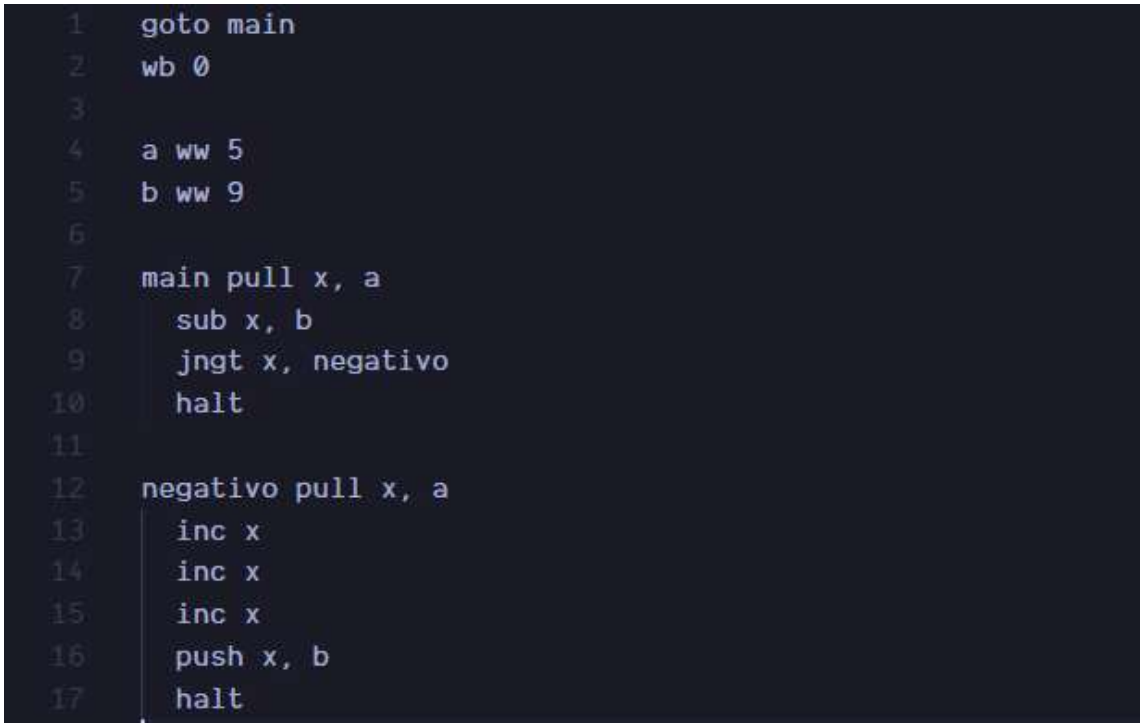
flg x e **clg x**: ambos os comandos não recebem parâmetros e realizam a operação do logaritmo de x na base 2. O comando flg x faz x receber o valor do piso do logaritmo de x na base 2, enquanto clg x faz x receber o teto do logaritmo de x na base 2.

```
1   goto main
2   wb 0
3
4   a ww 5
5   b ww 9
6
7   main pull x, b
8     clg x
9     push x, b
10    halt
```

Na linha 8, x recebe o valor do teto do logaritmo de 9 na base 2.

jz x e **jngt x**: ambos os comandos não recebem parâmetros e realizam saltos condicionais. O comando jz x realiza um salto para o endereço do comando passado como parâmetro caso o valor do registrador x seja zero, já o

comando jngt x realiza o salto para o endereço do parâmetro caso o valor do registrador x seja negativo.



```
1   goto main
2   wb 0
3
4   a ww 5
5   b ww 9
6
7   main pull x, a
8     sub x, b
9     jngt x, negativo
10    halt
11
12  negativo pull x, a
13    inc x
14    inc x
15    inc x
16    push x, b
17    halt
```

Na linha 9, caso o valor do registrador x seja negativo, ele realizará um salto para a linha 12.

zero x: Esse comando não recebe parâmetros e atribui a x o valor 0.

```
1  goto main
2  wb 0
3
4  a ww 5
5
6  main pull x, a
7    zero x
8    push x, a
9    halt
```

Na linha 7, o valor de x é zerado.

pow2 x: Esse comando não recebe parâmetros e faz x receber o quadrado do seu valor atual.

```
1  goto main
2  wb 0
3
4  a ww 5
5
6  main pull x, a
7    pow2 x
8    push x, a
9    halt
```

Na linha 7, x recebe o quadrado do seu valor atual.

halt: encerra o programa.

```
1  goto main
2  wb 0
3
4  a ww 5
5
6  main pull x, a
7    pow2 x
8    push x, a
9    halt
```

Na linha 9, o programa é encerrado.

Feito por:

Gustavo Valença e Breno Macêdo

Obrigado por ler ☺