

# Trabalho de Arquitetura de Computadores

Integrantes:

Levy Nascimento Oliveira - 541800

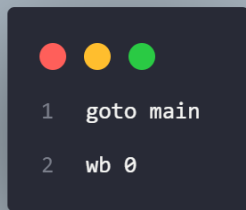
Anna Beatriz Vieira Sousa - 538758

# Instruções Gerais

```
1 instructions = ['add', 'sub', 'goto', 'mov', 'jz', 'halt', 'wb', 'ww', 'mult', 'div', 'mod', 'pot', 'pull', 'fat', 'inc', 'dec', 'zero']
2 instruction_set = {'add': 0x02,
3                   'sub': 0x06,
4                   'mult': 0x11,
5                   'div': 0x1D,
6                   'mod': 0x31,
7                   'pull': 0x35,
8                   'fat': 0x25,
9                   'inc': 0x2F,
10                  'dec': 0x30,
11                  'zero': 0x38,
12                  'mov': 0x0A,
13                  'goto': 0x0D,
14                  'jz': 0x0F,
15                  'halt': 0xFF}
```

Nesta parte conseguimos ver as instruções básicas implementadas no firmware, as quais podem ser acessadas por seus endereços em hexadecimal ao serem chamadas no arquivo assembly. Algumas dessas instruções, nós acabamos não utilizando, mas pensando em casos gerais, elas seriam úteis para arquiteturas maiores e mais complexas.

Iremos explicar agora a utilização e o modo como algumas funções são chamadas no assembly:



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

**wb:** escreve um byte nos dados do programa, o qual é utilizado para alinhar as words.

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.



```
1 a ww 5
2 b ww 3
3 c ww 1
4 d ww 2
```

**ww:** escreve uma word nos dados do programa com o valor indicado no parâmetro, seu endereço corresponde a um nome(letra) declarada antes do comando.



```
1 goto main
```

**goto address:** faz o programa ir até o endereço passado como parâmetro e executar a partir desse ponto. É muito útil para saltos e loops que ocorrem ao longo do código, em que uma sequência de operações são realizadas logo após entrar no endereço passado como parâmetro. A instrução main será executada logo em seguida



```
1 pull y, a
```

**pull:** atribuir a x o valor da word no endereço de memória passado como parâmetro, nesse caso o valor que está presente em a no momento do chamado da função.



```
1 mov y, c
```

**mov:** atribui ao endereço de memória passado como parâmetro o valor atual de x. Neste caso, c está recebendo o valor de y.



```
1 add x, a  
2 sub x, b
```

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

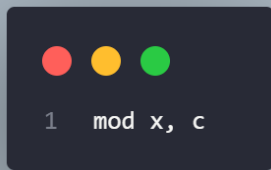
Nesse caso,  $x \leftarrow x + a$  e  $x \leftarrow x - b$



```
1 mult x, a  
2 div y, d
```

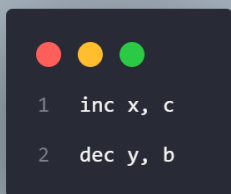
**mul, div:** o comando mul x faz o registrador x receber o produto entre seu valor atual e o valor da word presente no endereço passado como parâmetro, o comando div y realiza a divisão inteira do valor do registrador e da word presente no endereço passado como parâmetro.

Nesse exemplo,  $x \leftarrow x * a$  e  $y \leftarrow y / d$



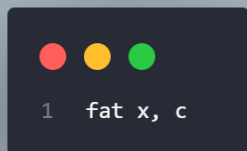
**mod:** o comando mod realiza a operação de resto da divisão do registrador pela word contida no endereço passado como parâmetro.

Nesse exemplo, a função irá armazenar em x o valor do resto da divisão de x por c

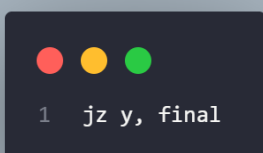


**inc:** o comando inc recebe um valor como parâmetro, porém este não é utilizado para realizar nenhuma operação envolvendo. O resultado é a incrementação do valor do registrador em 1.

**dec:** o comando dec recebe um valor como parâmetro, porém este não é utilizado para realizar nenhuma operação envolvendo. O resultado é o decremento do valor do registrador em 1.



**fat:** o comando fact recebe parâmetros, porém não o utiliza na operação e faz o registrador receber o valor do fatorial de x.



**jz:** O comando jz x realiza um salto para o endereço do comando passado como parâmetro caso o valor do registrador x seja zero

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

**halt:** encerra o programa.

**Observação:** Vale ressaltar que as funções foram implementadas também para o registrador y, de modo a otimizar a arquitetura, visto que diminui os passos e o tempo de execução.

# Manual de uso dos assembly

## Questão 2:

### Divisão e resto

Nome do arquivo asm: divmod.asm

Nome do arquivo binário: divmod.bin

Código para rodar o programa:

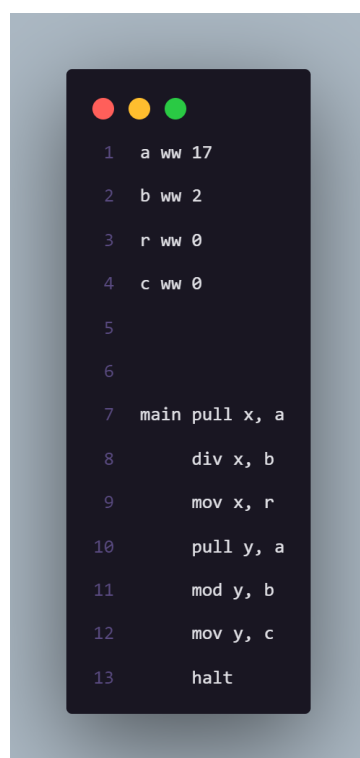
- `python assembler.py divmod.asm divmod.bin`
- `python computer.py divmod.bin`

**Endereço de memória “a”:** dividendo, valor que deve ser dividido.

**Endereço de memória “b”:** divisor, valor que dividirá a word presente no endereço de memória “a”

**Endereço de memória “r”:** endereço de memória que será armazenado o resultado da divisão inteira de “a” por “b”

**Endereço de memória “c”:** endereço de memória que será armazenado o resultado do resto da divisão de “a” por “b”



O valor de “a” é passado para o registrador x e, logo em seguida, ocorre a divisão do valor presente no registrador x pelo valor presente em “b” e é armazenado em x. Após a divisão, o valor presente em “x” é copiado para “r”.

Após realizar a divisão e salvar o resultado inteiro em “r”, o valor de “a” é copiado para y, e logo após ocorre a divisão de y por b e o resto é salvo em y. Após realizar a operação de resto, o valor é copiado para “c” e o programa encerra.

## Questão 3:

### CSW

Nome do arquivo asm: csw.asm

Nome do arquivo binário: csw.bin

Código para rodar o programa:

- python assembler.py csw.asm csw.bin
- python computer.py csw.bin

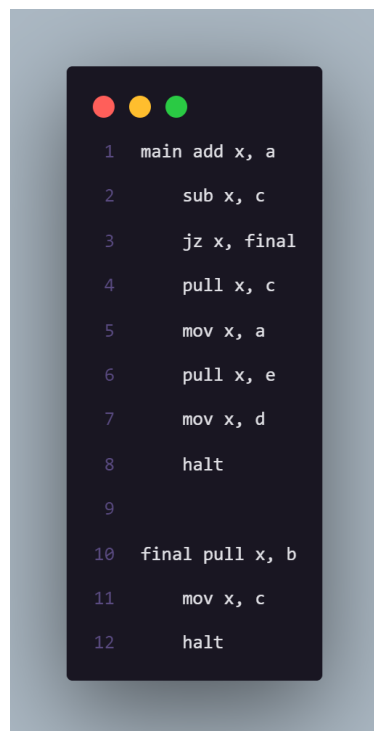
**Endereço de memória “a”:** valor “a” que será comparado.

**Endereço de memória “b”:** valor “b” que será modificado

**Endereço de memória “c”:** valor “c” que será comparado

**Endereço de memória “d”:** endereço de memória que será armazenado o retorno da função

**Endereço de memória “e”:** endereço de memória que será armazenado um valor auxiliar importante para a operação



O valor “a” é adicionado ao registrador e logo em seguida o registrador será subtraído. Caso o valor da subtração seja 0, o programa irá para o final, o que significa que o valor presente em “c” e em “a” são iguais, o que acarreta no carregamento do valor presente em “b” para “c” e o retorno de 0 para d. Caso, após a subtração de “a” por “c”, o valor presente no registrador x ainda não seja 0, significa que o valor de “c” e “a” são diferentes, o que acarreta no carregamento do valor presente em “c” para “a”.

## Questão 4:



# Fatorial

Nome do arquivo asm: fatorial.asm

Nome do arquivo binário: fatorial.bin

Código para rodar o programa:

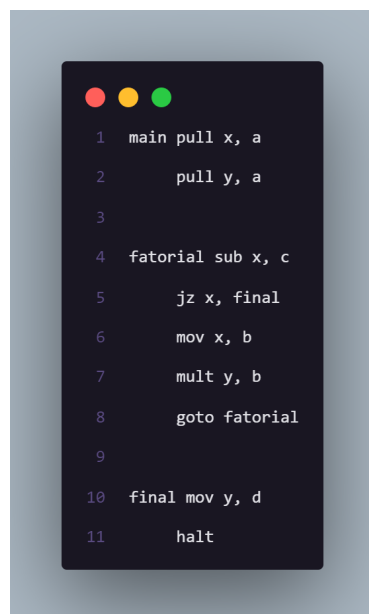
- `python assembler.py fatorial.asm fatorial.bin`
- `python computer.py fatorial.bin`

**Endereço de memória “a”:** valor “a” o qual será realizado o fatorial.

**Endereço de memória “b”:** auxiliar que será utilizado para calcular o fatorial de “a”.

**Endereço de memória “c”:** auxiliar que será utilizado para calcular o fatorial de “a”.

**Endereço de memória “d”:** endereço de memória que será lido no final do programa e onde será armazenado o valor do fatorial.



O valor que será calculado o fatorial é armazenado em x e em y. No fatorial, o valor de x é decrementado em “c” a cada multiplicação. A cada multiplicação, é verificado se o valor de x já é 0 e, se for, ele finaliza e põe o resultado presente em y em d. A função fatorial é recursiva e se repete até zerar o valor presente em x.