

Projeto Final ESC 2020-1

1 Informações

- Projetos para grupos **de até 3** integrantes (podendo ser individual);
- Nos algoritmos, considere todas as variáveis com identificadores em maiúsculo como de interesse (veja as linhas de *define*) e em minúsculo como variáveis auxiliares.
- Estude o fluxo de controle do programa para entender quando o mesmo está finalizado.
- As funções foram definidas para auxiliar no entendimento do algoritmo em questão; os alunos deverão implementar “funções” em ASM ao definir regiões de memória ROM para cada trecho de código que sugerir uma função, com auxílio de *labels*. Por exemplo:

```
...    // main
@R1
D=M
@FOO
D;JEQ // foo call, only if D = 0
...
(END_FOO) // foo returns from here
...
(END)
@END
0;JMP // end of main
...
(FOO)    // foo begin
...
@END_FOO
0;JMP // foo return, end of foo
```

- Podem ser criadas “funções” auxiliares tantas quanto forem necessárias.
- Como referência, segue parte da Tabela 4.5 do livro:

Mnemônico	Condição de <i>jump</i>
JGT	$out > 0$
JEQ	$out = 0$
JGE	$out \geq 0$
JLT	$out < 0$
JNE	$out \neq 0$
JLE	$out \leq 0$
JMP	incondicional

Note que *out* é a saída da ALU, observada na primeira parte da instrução de *jump*. Por exemplo, “0;JMP” instrui a ALU para fornecer 0 na saída *out*, deixando a instrução “0;JEQ” com o mesmo comportamento.

2 Projeto #1 - Multiplicação em ASM

2.1 Algoritmo

O Algoritmo 1 refere-se ao algoritmo de multiplicação por adições repetidas. Os alunos podem implementar outro algoritmo de sua preferência, se assim desejarem¹.

Algoritmo 1: Multiplicação por adição repetida

```
1  #define B (Multiplier) as R0 (RAM[0])
2  #define C (Multiplicand) as R1 (RAM[1])
3  #define A (Product) as R2 (RAM[2]), as in A=B*C
4
5  // main *****
6  if B = 0 or C = 0 then
7      A := 0
8  else
9      if B < 0 and C < 0 then
10         B := -B; C := -C
11         multiply()
12         B := -B; C := -C
13     else
14         if B < 0 then
15             B := -B
16             multiply()
17             B := -B; A := -A
18         else
19             if C < 0 then
20                 C := -C
21                 multiply()
22                 C := -C; A := -A
23             else
24                 multiply()
25             endif
26         endif
27     endif
28 endif
29 end
30
31 // support *****
32 function multiply():
33     i := B - 1; A := C
34     while i > 0 do
35         i := i - 1; A := A + C
36     endwhile
37 endfunction
```

¹https://en.wikipedia.org/wiki/Multiplication_algorithm

2.2 Detalhes

- Implementar o Algoritmo 1 da multiplicação em ASM (arquivo `mult.asm`) para o computador HACK estudado nesse curso.
- Usar o Assembler implementado para gerar o código de máquina (arquivo `mult.hack`);
- Usar o *CPUEmulator* para verificar se o código está correto.

Tabela 1: Casos de teste		
Multiplicador RAM[0]	Multiplicando RAM[1]	Produto RAM[2]
3	4	12
42	7	294
0	0	0
1	0	0
0	3	0
-26	1	-26
26	-1	-26
7	-2	-14
-7	-2	14

3 Projeto #2 - Divisão em ASM

3.1 Algoritmo

O Algoritmo 2 refere-se ao algoritmo de divisão por subtração repetida e sempre produz **resto não negativo**. Apesar de simples, é exponencialmente mais lento do que outros algoritmos, sendo útil se o quociente é pequeno.

Algoritmo 2: Divisão por subtração repetida

```
1  #define N (Numerator) as R0 (RAM[0])
2  #define D (Denominator) as R1 (RAM[1])
3  #define Q (Quotient) as R2 (RAM[2])
4  #define R (Rest) as R3 (RAM[3])
5
6  // main *****
7  if D = 0 then
8      division_by_zero()
9  else
10     if N = 0 then
11         Q := 0; R := 0
12     else
13         divide()
14     endif
15 endif
16 end
17
18 // support *****
19 function divide():
20     if D < 0 then
21         D := -D
22         divide()
23         D := -D; Q := -Q
24     else
25         if N < 0 then
26             N := -N
27             divide()
28             N := -N
29             if R = 0 then
30                 Q := -Q
31             else
32                 Q := -Q - 1; R := D - R
33             endif
34         else // At this point, N >= 0 and D > 0
35             divide_unsigned()
36         endif
37     endif
38 endfunction
39
40 function divide_unsigned():
41     Q := 0; R := N
42     while R >= D do
43         Q := Q + 1; R := R - D
44     endwhile
45 endfunction
46
47 function division_by_zero():
48     Q := 0
49     R := 0x7FFF // or 32767, max. 16-bits positive integer
50 endfunction
```

3.2 Detalhes

- Note que 0x7FFF ou 32767 é o máximo inteiro positivo de um número representado com 16 bits.
- Implementar o Algoritmo 2 da divisão inteira (com quociente e resto) em ASM (arquivo `div.asm`) para o computador HACK estudado nesse curso.
- Usar o Assembler implementado para gerar o código de máquina (arquivo `div.hack`);
- Usar o *CPUEmulator* para verificar se o código está correto.

Tabela 2: Casos de teste

Numerador RAM[0]	Denominador RAM[1]	Quociente RAM[2]	Resto RAM[3]
42	7	6	0
3	4	0	3
26	7	3	5
0	0	0	32767
1	0	0	32767
0	3	0	0
0	-3	0	0
-42	7	-6	0
-7	2	-4	1
7	-2	-3	1
-7	-2	4	1