Atividade do Capítulo 4

Nome: Mateus Sousa Araújo – **Matrícula:** 374858
Nome: José Wesley Araújo – **Matrícula:** 374855

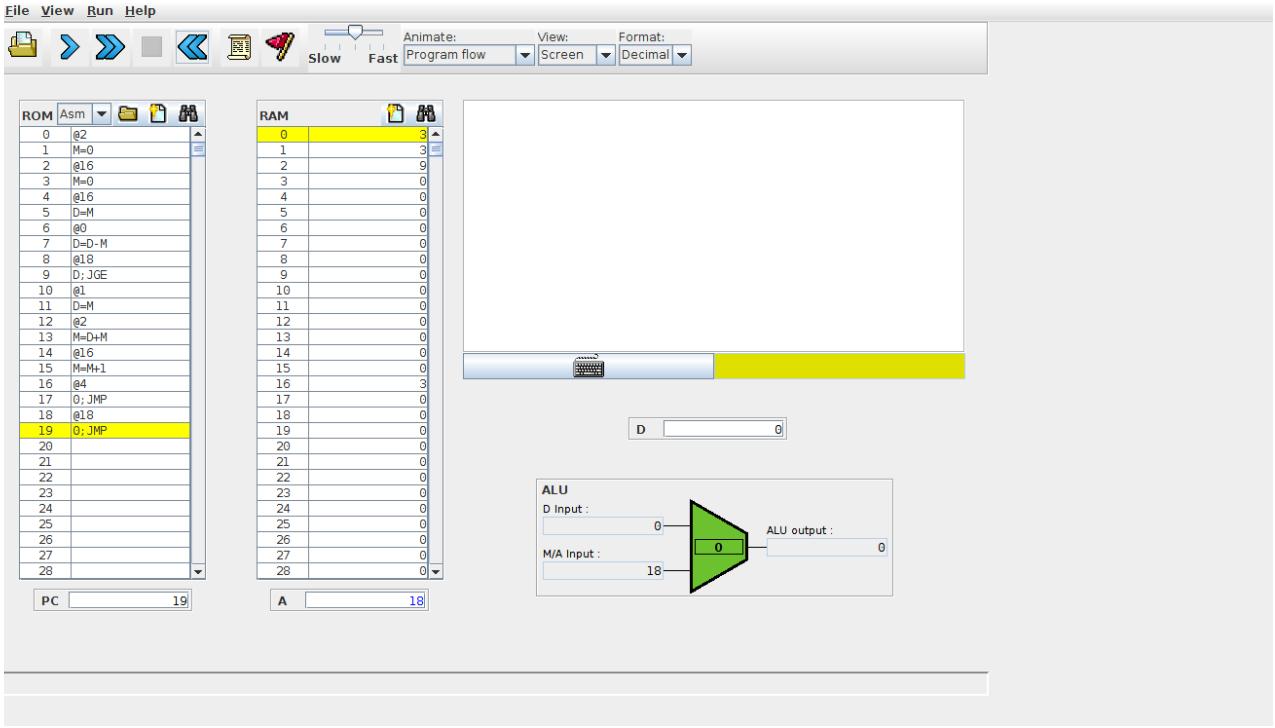
Questão 1

```
// Multiplica 2 números armazenados nos registradores R0 e R1 e coloca no registrador R2.
// R0 = RAM[0].
// R1 = RAM[1].
// R2 = RAM[2].
// A ideia é multiplicar R0 * R1 somando R1 R0 vezes. O resultado armazena a cada iteração em R2

    @2
    M=0     // Inicializa R2 com 0; (R2 = 0)    //M = RAM[2] = 0;
    @i
    M=0     // Inicializa i com 0; i = 0 (contador);

(LOOP)      // LOOP
    @i
    D=M     // D = i;
    @0
    D=D-M   // D = i-R0;
    @END
    D;JGE   // if i-R0 >= 0 goto END;

    @1
    D=M     // D = R1;
    @2
    M=D+M   // R2 = R2+R1;
    @i
    M=M+1   // incrementa i; (i = i+1);
    @LOOP
    0;JMP   // Volta pra (LOOP);

(END)       // Label para o fim do programa;
    @END
    0;JMP   // Acaba o programa com um loop infinito;
```

A primeira atividade pedia pra realizar a multiplicação de dois números armazenados em 2 registradores R0 e R1 e em seguida o resultado dessa operação fosse armazenado em um outro registrador R2. O programa deveria ser feito utilizando as instruções do Assembly do Hack. A ideia é que para realizar uma multiplicação devemos realizar uma soma do valor armazenado em R1 R0 vezes. Uma sucessiva sequência de somas. Iniciamos a inicialização do R2 (RAM[2]) em 0 fazendo @2 e em seguida atribuindo 0 para a posição de memória RAM[2] (M = 0). Fizemos a criação de uma variável de incremento “i” para realizar a contagem. Inicializamos a mesma em 0 (M = 0). Em seguida fizemos um laço com label LOOP. Primeiramente armazenamos a variável de incrementação “i” no registrador de destino “D”. Em seguida fizemos D = D – M. Essa última instrução vai nos auxiliar quantas vezes iremos somar R1 baseado na informação contida dentro do Registrador R0. A cada incremento no LOOP a nossa variável “i” vai sendo incrementada em 1 e diminuindo o valor de “i” pelo valor contido em R0.

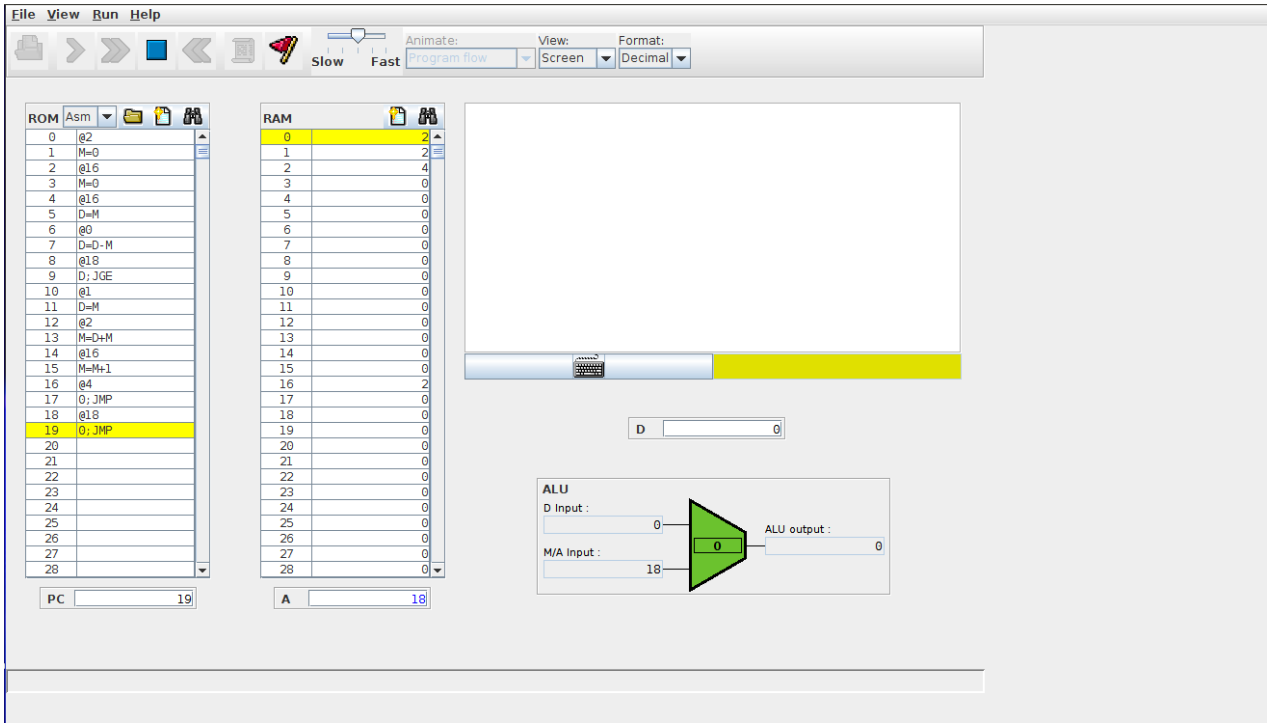
Vai chegar um momento que o Registrador D irá ser zero, fazendo com que a partir de agora o nosso programa seja encerrado com uma condição de JGE. Se o conteúdo dentro do registrador D for zero, então pule para a label de END onde encerrará o programa em um loop infinito. Depois fizemos com que o registrador R1 (RAM[1]) fosse colocado no registrador de destino D, realizando @1, D = M. Em seguida fizemos com que o registrador R2 recebesse a soma de R2 com R1, ou seja, M = D + M. Essa última instrução seria a mesma coisa que R2 = R2 + R1. Já que inicializamos o conteúdo de R2 em zero, então a cada incremento o R2 receberá a soma de apenas R0 vezes o conteúdo armazenado dentro de R1. Depois incrementamos a variável “i” fazendo M = M + 1. Depois fizemos um JMP condicional voltando para a estrutura de repetição LOOP.

Na simulação usando o CPU Emulator precisamos fixar 2 números nas primeiras posições de memória, dessa forma representando os registradores R0 e R1. Fizemos vários testes observando os conteúdos alterados a cada instrução. Abaixo colocamos 3 prints com três exemplos diferentes de multiplicação.



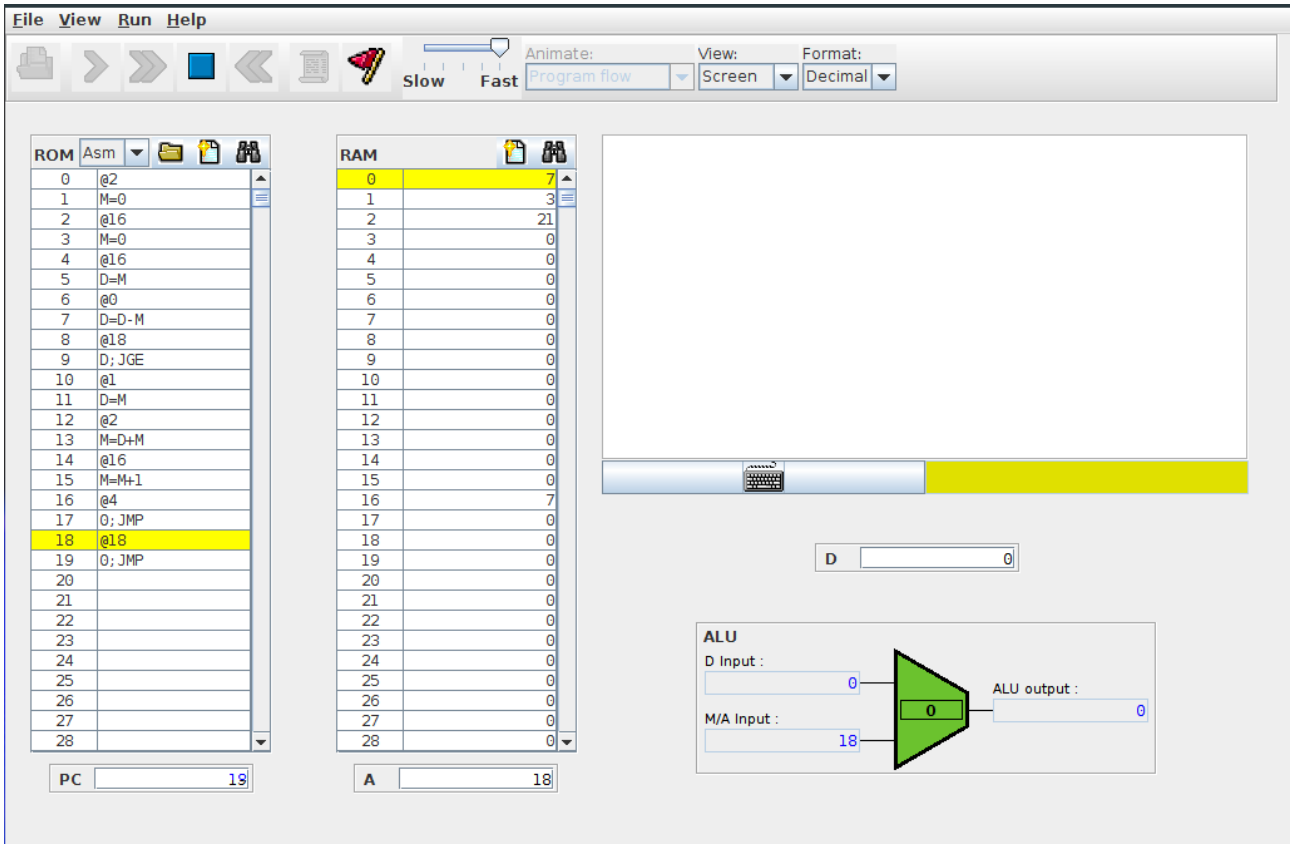
Simulação 1 – Exemplo 1

Acima colocamos os valores de R0 (RAM[0]) em 3 e R1 (RAM[1]) em 3. O resultado deve ser armazenado em R2 (RAM[2]). O resultado como esperado, deve dar 9. O que está corretamente repreentado com a simulação de teste acima.



Simulação 2 – Exemplo 2

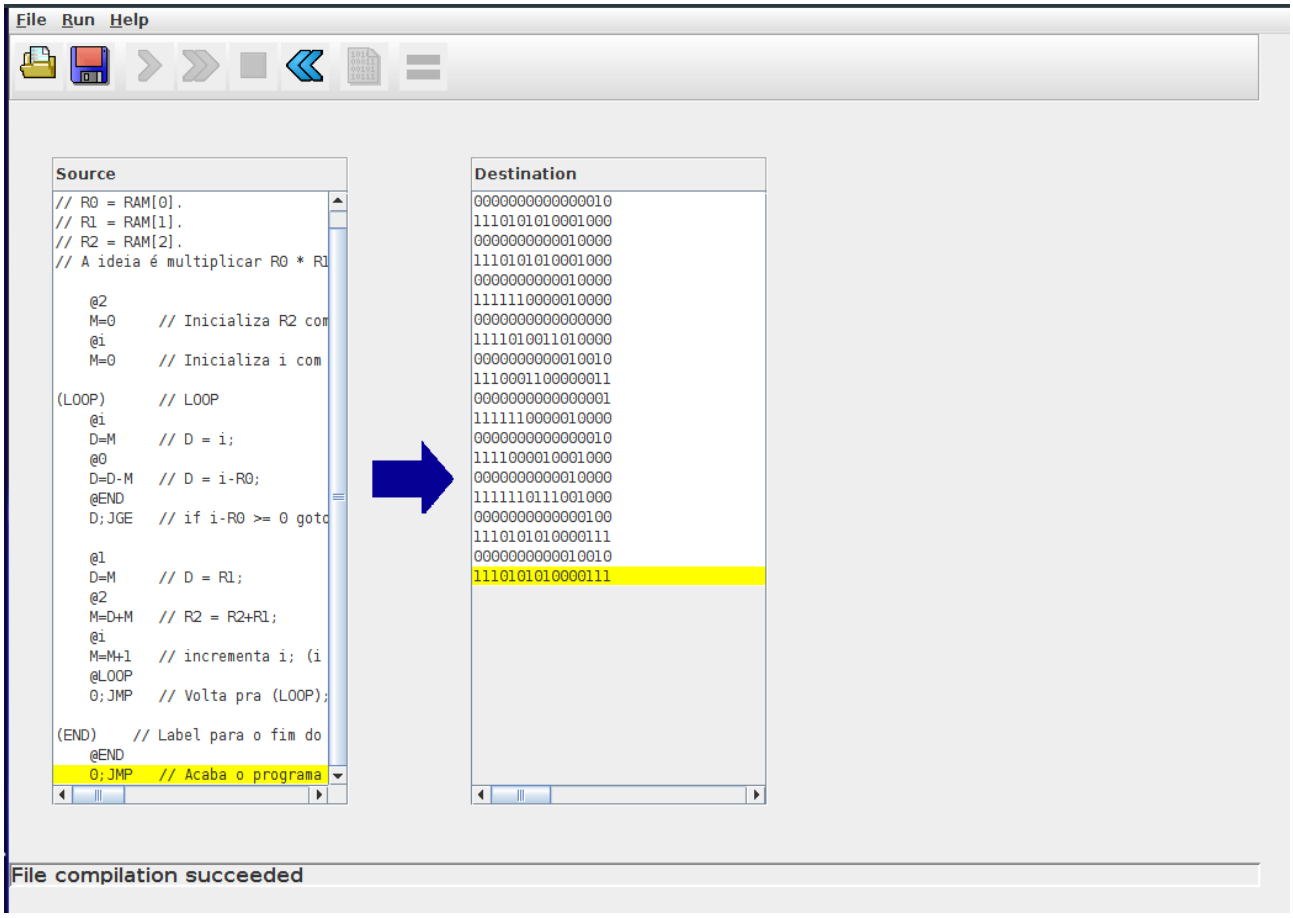
Acima colocamos os valores de 2 dentro de R0 e R1. O resultado esperado é 2, como mostra na simulação acima.



Simulação 3 – Exemplo 3

Acima fixamos o R0 em 7 e o R1 em 3. O resultado esperado é 21 e esse resultado é armazenado na posição 2 de memória RAM[2] – R2.

Abaixo utilizamos o simulador do Assembly HACK para a geração das instruções em binário de cada instrução do código de multiplicação. Abaixo segue o resultado:



Geração das instruções em Assembly HACK

Questão 2

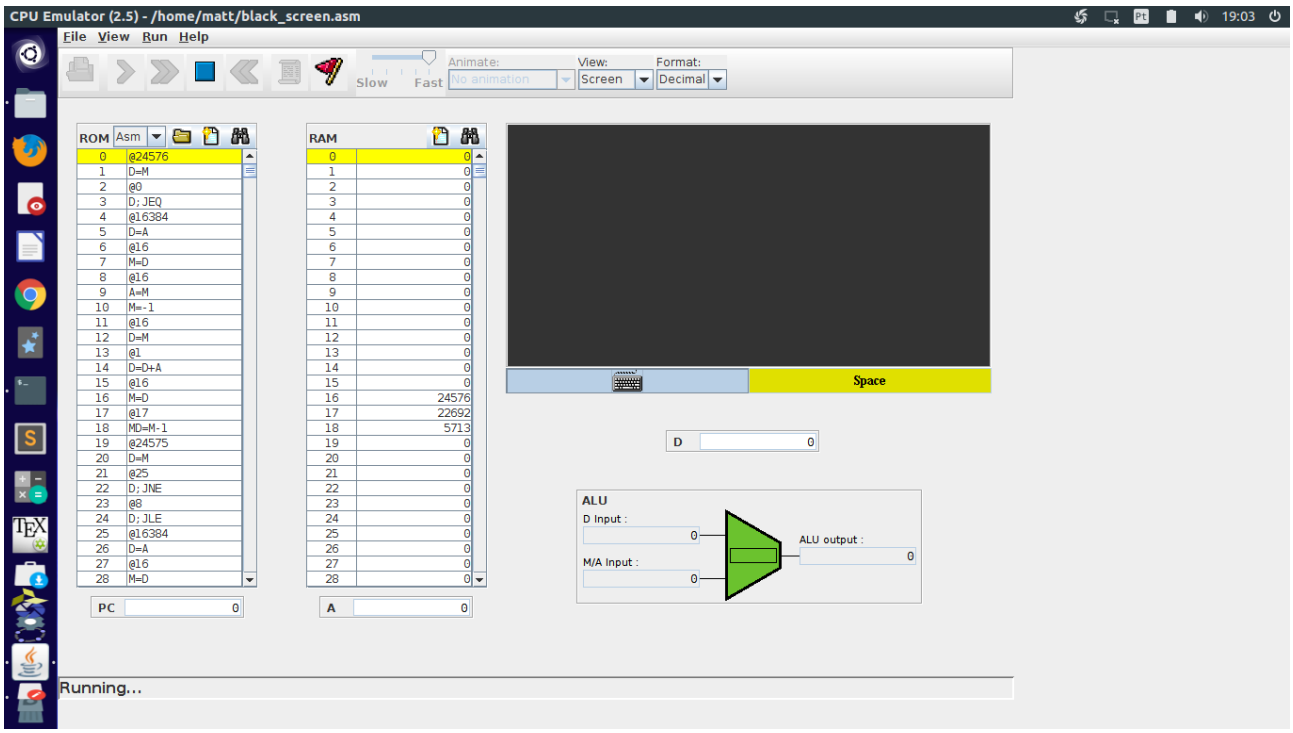
Na 2ª questão foi pedido pra fazer um código utilizando o assembly do Hack em que se o usuário apertasse alguma tecla a tela de output do simulador de CPU ficava preta. Quando o usuário soltar, a tela fica branca. Utilizamos um exemplo muito parecido que baixamos no pacote que contém os aplicativos de simulação no site da Nand to Tetris. O exemplo 6 é de um exemplo que é desenhado um retângulo na tela do simulador. Pegamos a parte do código lá demonstrado e fizemos duas sessões, uma para a tela preta e outra para a tela branca. O código está dividido em 2 partes. Temos um LOOP para o preenchimento da tela na cor preta um segundo LOOP para o preenchimento da tela na cor branca. No começo nós fizemos um loop para verificar se há alguma tecla sendo apertada no momento.

Usamos a instrução @screen para isso. Caso haja alguma tecla pressionada, o preenchimento da tela começará no pixel 1, que equivale a 1ª posição de memória. Fizemos essa definição para começar os preenchimentos a partir do pixel 1. No loop para o preenchimento da tela presta colocamos o A com valor de M e depois colocamos M valendo -1, que vai corresponder ao número para deixar a tela preta a cada iteração. Em seguida fizemos com que fosse percorrido todas as linhas por todos os pixels fazendo a incrementação no registrador D. Em seguida fizemos uma verificação para o último endereço de memória que vai equivaler ao último pixel. Para o preenchimento de tela branca fizemos exatamente o mesmo processo, com diferença de que o registrador M vai receber 1 para a tela ficar branca. Fizemos tambem a verificação do ultimo endereço e várias condições iguais ao do começo do código para verificar se há ou não alguma tecla apertada. Os códigos e as simulações estão abaixo junto com as gerações das intruções pelo

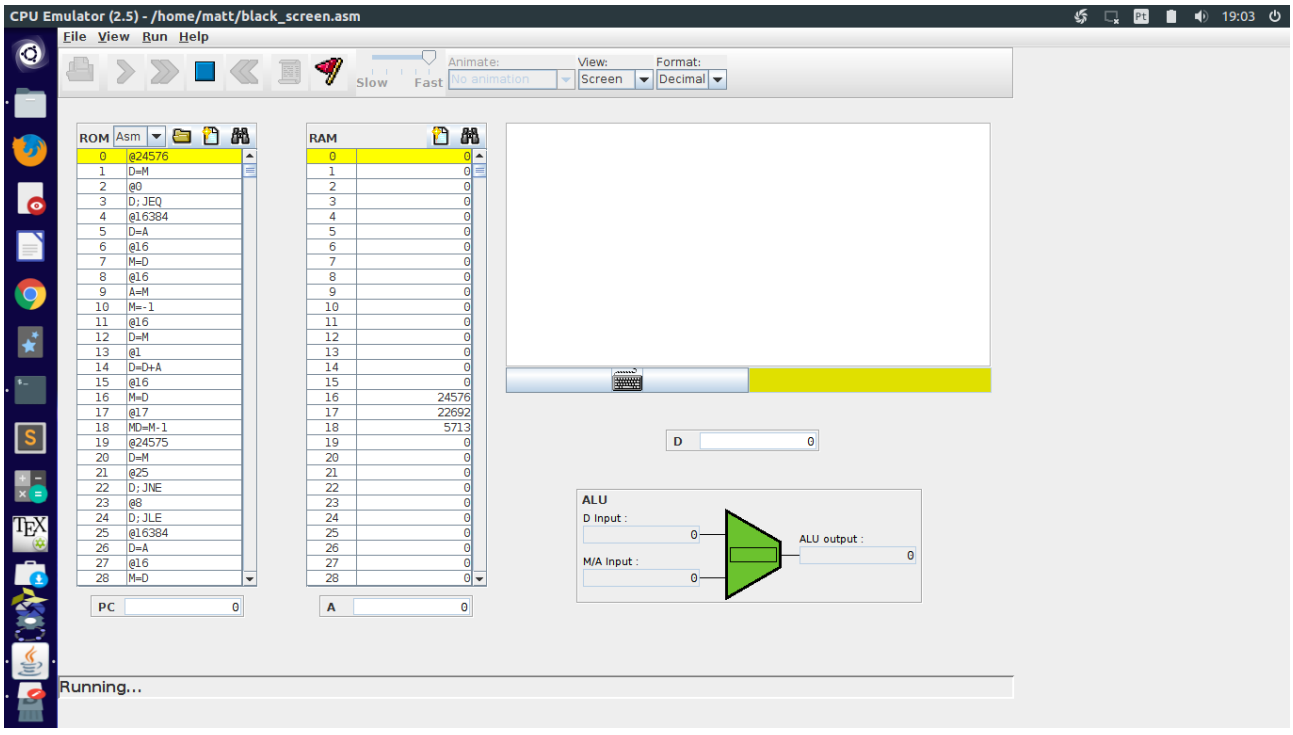
assembler. Não deu muito bem pra tirar um print da tela piscando, mas vamos mandar todos os códigos referentes ao exercício 1 e 2.

```
1         @24576          // Laço que verifica se há alguma tecla pressionada
2         D=M             // D recebe o endereço @24567
3         @0              // Posição 0 de memória
4         D;JEQ           // Pula se o valor de D for zero
5
6         @SCREEN         // Endereço para o primeiro pixel;
7         D=A             // D = SCREEN;
8         @address
9         M=D             // Recebe endereço. M = addreess;
10 (LOOP)                 // Label para laço de tela branca
11        @address
12        A=M             // A = endereço
13        M=-1            // Tela fica preta
14        @address
15        D=M             // D = M
16        @1              // primeira posição de memória
17        D=D+A           // Incrementa em 1 para passar por todos os endereços
18        @address
19        M=D             // Recebe o endereço
20        @counter        // variável contador
21        MD=M-1
22        @24575          // Faz a verificação do último pixel equivalente ao endereço de memória
23        D=M             // D = endereço (@24575)
24        @BRANCO
25        D;JNE           // Pula se o D for igual a 0
26        @LOOP
27        D;JLE           // Pula se o D for igual a 0
28
29 (BRANCO)
30        @SCREEN         //Endereço para o primeiro pixel
31        D=A             // D recebe A, endereço do pixel
32        @address
33        M=D             // M recebe endereço
34 (LOOP2)                // Loop de laço para tela preta
35        @address
36        A=M             // Iguala A com o valor armazenado em M
37        M=0             // Inicializa o M em 0
38        @address
39        D=M             // D recebe M
40        @1              // Posição 1 de memória
41
42        D=D+A           // Incrementa o endereço pra varrer a tela
43        @address
44        M=D             // M recebe o valor do incremento anterior
45        @counter
46        MD=M-1          // MD recebe o decremento de conta
47
48        @24576          //Verifica se n há nenhuma tecla pressionada
49        D=M             // M recebe o endereço
50        @0              // posição 0 de memoria
51        D;JNE           // Pula se o conteúdo de D for 0
52
53        @24575          // Faz verificação para último pixel
54        D=M             // D recebe o endereço
55        @0              // posição 0 de memória
56        D;JEQ           // Pula se o conteúdo de D for 0
57        @LOOP2          // Retorna para o 2º laço
58        D;JLE           // Pula se o conteúdo de D for 0
```

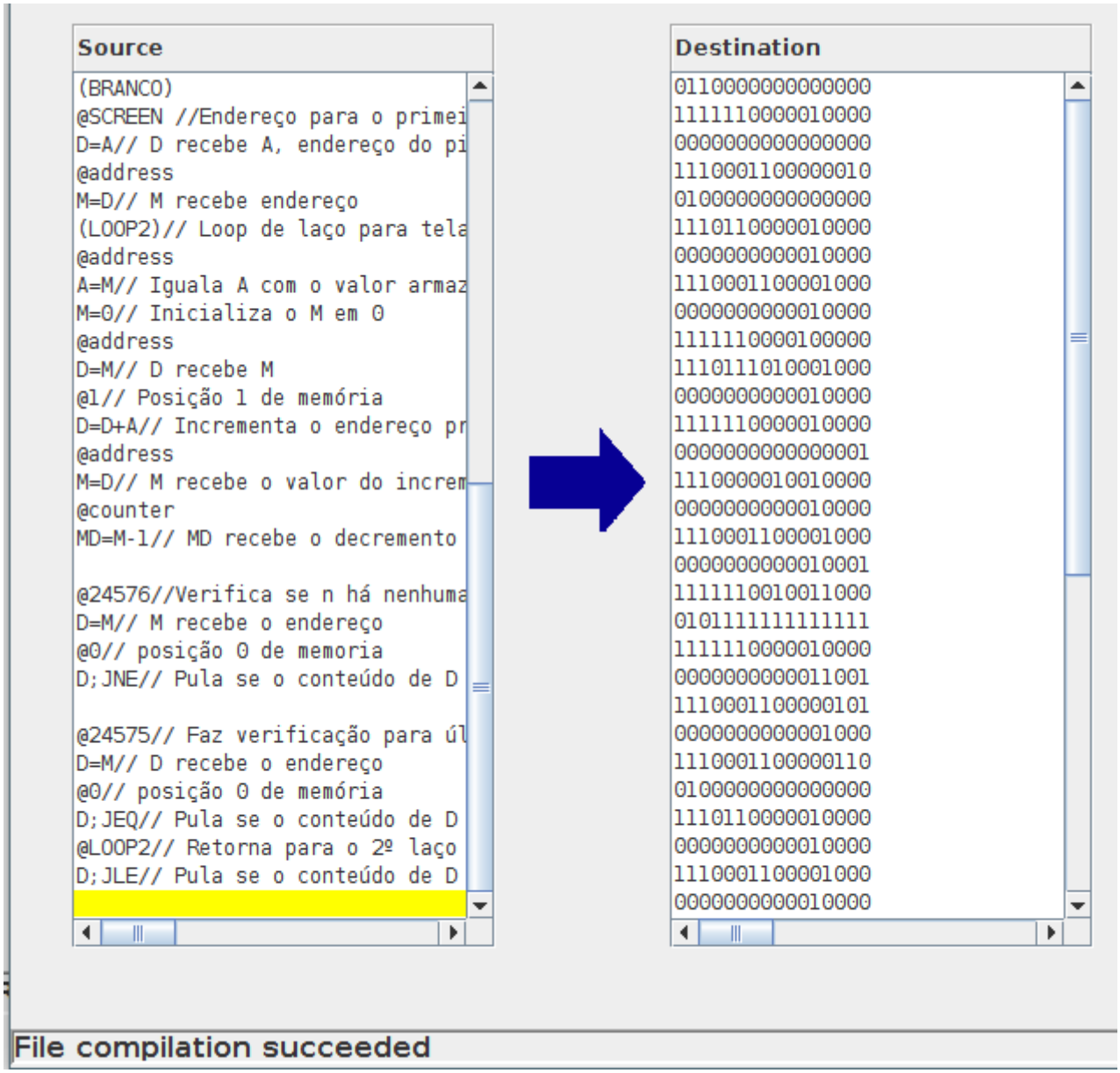
Código para o Black_White_screen



Tela preta com tecla pressionada



Tela branca com nenhuma tecla pressionada



Geração das instruções