



UNIVERSIDADE FEDERAL DE SANTA CATARINA  
CENTRO TECNOLÓGICO - CTC  
ORGANIZAÇÃO DE COMPUTADORES I - INE5411-03208B  
PROFESSOR: MARCELO DANIEL BEREJUCK  
ALUNOS: GUSTAVO BATISTELL (12101228) E LUCAS DE S. MORO (23201136)

## RELATÓRIO - PROJETO FINAL

### MÁQUINA DE CAFÉ

Florianópolis, Dezembro de 2024.

# Proposta

A proposta deste projeto foi desenvolver um programa em Assembly para o MIPS32 no simulador *MARS 4.5* controlando uma máquina de café. A máquina foi projetada para uso em ambientes comerciais ou empresariais, oferecendo funcionalidades como preparo de três tipos de bebidas (café puro, café com leite e Mochaccino), escolha entre dois tamanhos de copos (pequeno e grande), e a opção de adicionar açúcar ao preparo.

A premissa principal foi implementar um sistema funcional que não apenas processasse as escolhas do usuário, mas também garantisse o controle interno de recursos, como doses de café, leite, chocolate e açúcar, além da liberação de água de acordo com o tamanho escolhido. Para isso, foi essencial utilizar timers via chamadas de sistema e gerenciar os estoques internos, garantindo bloqueios adequados em caso de falta de insumos. Além disso, o projeto também visou a integração com o terminal para visualização das mensagens e o teclado Digital Lab Sim para as entradas pelo usuário no programa.

## Interface alternativa e organização

Apesar da escrita e execução dos códigos deste projeto ter sido feita no simulador *MARS 4.5*, devido às suas limitações da interface com usuário e da impossibilidade de separar os blocos funcionais em arquivos distintos, foi optado por instalar a extensão *vscode-mips v0.3.1* no software *Visual Studio Code 1.95.3*, facilitando a organização e modularidade de seus blocos funcionais para posterior integração em um único arquivo. Essa abordagem alternativa para o trabalho em dupla trouxe facilidade para manipular indentação, ocultar e exibir trechos específicos, além de haver completa funcionalidade da barra lateral e das abas para navegar entre versões ou blocos/módulos funcionais.

Outro ponto importante na organização das tarefas da dupla, foi a nomenclatura utilizada para os arquivos dos códigos, com prefixo denotando separação por níveis e subníveis (conforme funcionalidade) e sufixos denotando a evolução de versões para cada arquivo, possibilitando elaborar e testar inicialmente diversas versões das implementações em separado para posteriormente serem aplicadas no código principal. Por exemplo, o arquivo principal da máquina de café como um todo, ficou no nível mais externo dessa abstração utilizada, ganhando o prefixo “0” e os sufixos de versão “v0”, “v1”, etc. assim, “0\_cafe\_v0.asm”, “0\_cafe\_v1.asm”, ... até a 11ª versão “0\_cafe\_v11.2.asm” entregue na apresentação de 05/12/24. Os prefixos crescentes foram utilizados para os arquivos de outros blocos funcionais e seus subníveis. Por exemplo “1\_FSM\_v0.asm”, “2\_display\_teclado\_DLS\_v2.asm”, “3.0\_cupom\_v6.asm” e “3.1\_Template\_Cupom\_Fiscal\_v2.txt”. Essa organização é apresentada de maneira resumida com o print de tela adaptado e apresentado na Figura 1.

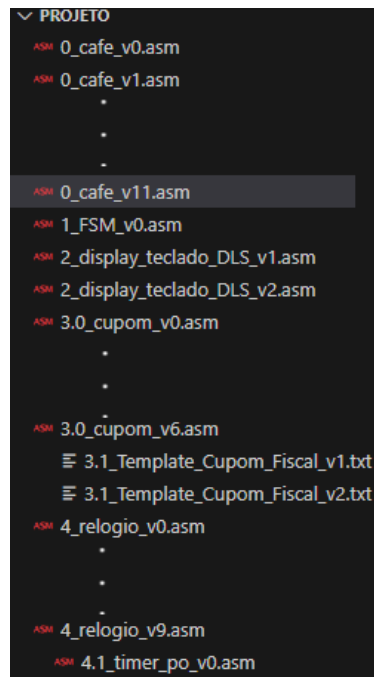


Figura 1 - Nome dos arquivos utilizados com numeração e níveis para organizar os testes dos blocos funcionais do projeto

Convém ressaltar que a ideia utilizada para essa organização de níveis e subníveis foi vagamente inspirada na estruturação de projetos de linguagens orientadas a objetos, mas não representa propriamente as camadas de abstração utilizadas de fato em sistemas embarcados.

Assim, a organização empregada resultou em 14 blocos bem definidos no código final, fáceis de expandir ou recolher, conforme apresentado na Figura 2.

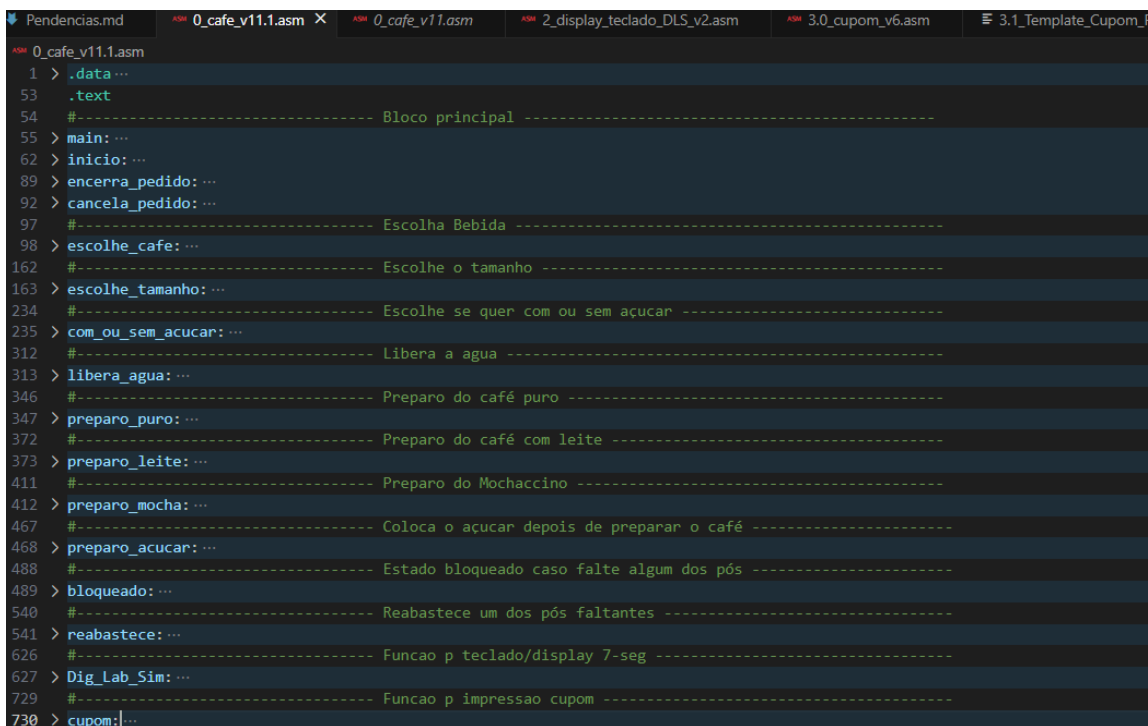


Figura 2 - Detalhe dos 14 blocos de código na interface do software VS Code.

## Estrutura dos dados

No segmento de dados `.data` do programa, foram declarados os mapeamentos para o Display de 7 segmentos, mensagens de saída do programa, variáveis inteiras com o estoque de cada ingrediente em pó, o arquivo TXT template para o cupom fiscal, o arquivo TXT de saída do cupom fiscal e os campos/conteúdos a serem preenchidos no arquivo de saída de cupom fiscal, conforme apresentado na Figura 3 abaixo.

```
asm 0_cafe_v11.1.asm
1  .data
2  zero: .byte 63 #0x63 para imprimir 0 display 7 segmentos
3  um: .byte 6 #0x6 para imprimir 1 display 7 segmentos
4  dois: .byte 91 #0x91 para imprimir 2 display 7 segmentos
5  tres: .byte 79 #0x79 para imprimir 3 display 7 segmentos
6  cinco: .byte 109 #0x73 para imprimir 5 display 7 segmentos
7  g_segment: .byte 64 #0x63 = 0b 0100 0000 p acender segmento g do display
8  msg_boas_vindas: .asciiiz "Bem-vindo a maquina de cafe, escolha seu cafe (1-Puro, 2-Com leite, 3-Mochaccino):"
9  msg_tamanho: .asciiiz "Selecione o tamanho do copo: (1-Pequeno, 2-Grande)"
10 msg_grande_escolhido: .asciiiz "Voce escolheu o tamanho Grande"
11 msg_pequeno_escolhido: .asciiiz "Voce escolheu o tamanho pequeno"
12 msg_invalido: .asciiiz "Opção inválida. Tente novamente.\n"
13 msg_limite: .asciiiz "Limite de tentativas excedido. Valor extornado\n"
14 msg_puro_escolhido: .asciiiz "Voce escolheu cafe puro"
15 leite_escolhido: .asciiiz "Voce escolheu cafe COM leite"
16 mocha_escolhido: .asciiiz "Voce escolheu mochaccino"
17 msg_acucar: .asciiiz "Selecione se voce deseja acucar: 1-Sim, 0-Não"
18 msg_acucar_sim: .asciiiz "Voce escolheu COM acucar"
19 msg_acucar_nao: .asciiiz "Voce escolheu SEM acucar"
20 faltando_cafe: .asciiiz "Máquina faltando CAFÉ. Aperte 5 para reabastecer"
21 faltando_leite: .asciiiz "Máquina faltando LEITE. Aperte 5 para reabastecer"
22 faltando_chocolate: .asciiiz "Máquina faltando CHOCOLATE. Aperte 5 para reabastecer"
23 faltando_acucar: .asciiiz "Máquina faltando AÇÚCAR. Aperte 5 para reabastecer"
24 reabastecer_po: .asciiiz "Selecione o pó que voce desejar reabastecer: 0-Café, 1-Leite, 2-Chocolate, 3-Açucar"
25 po_reabastecido: .asciiiz "Pó reabastecido. Reiniciando a máquina..."
26 liberando_agua: .asciiiz "Liberando água..."
27 botando_cafe: .asciiiz "Preparando cafe"
28 botando_leite: .asciiiz "Adicionando leite"
29 botando_chocolate: .asciiiz "Adicionando chocolate"
30 botando_acucar: .asciiiz "Adicionando acucar"
31 botando_agua: .asciiiz "Adicionando agua"
32 gera_cupom: .asciiiz "Gerando Cupom Fiscal"
33 cafe_dose: .word 20 # Doses de café
34 leite_dose: .word 20 # Doses de leite
35 acucar_dose: .word 20 # Doses de acucar
36 chocolate_dose: .word 20 # Doses de chocolate
37 # Nomes dos arquivos
38 template_f: .asciiiz "Template_Cupom_Fiscal_0.txt" # Arquivo de template
39 output_f: .asciiiz "cupom_fiscal.txt" # Arquivo de saída
40 # Buffers
41 buffer: .space 1024 # Buffer para leitura do arquivo
42 in_date: .asciiiz "05/12/2024" # Argumento de teste para substituir '@'
43 in_time: .asciiiz "22:22:30" # Argumento de teste para substituir '&'
44 print_date: .asciiiz "#####" # Caractere de substituição
45 print_time: .asciiiz "#####" # Caractere de substituição
46 print_prod: .asciiiz "*****"
47 produto_1_1: .asciiiz "1.1 Cafe pequeno 1.00"
48 produto_2_1: .asciiiz "2.1 Cafe grande 1.50"
49 produto_1_2: .asciiiz "1.2 Cafe c leite P 1.50"
50 produto_2_2: .asciiiz "2.2 Cafe c leite G 2.00"
51 produto_1_3: .asciiiz "1.3 Mochaccino P 2.00"
52 produto_2_3: .asciiiz "2.3 Mochaccino G 2.50"
```

Figura 3 - Trecho de dados do código com mapeamento do display e mensagens

## Procedimentos do programa

Inicialmente no segmento `.text`, tem-se o **Bloco Principal** com **main**, **inicio**, **encerra\_pedido** e **cancela\_pedido** apresentado na Figura 4 abaixo. Esse bloco é

responsável pelo carregamento inicial dos dados na memória, pelas chamadas de procedimentos e controle geral da máquina de café, como uma camada de abstração mais externa do projeto, organizando o fluxo principal do programa, assegurando que as escolhas do usuário sejam capturadas e processadas de maneira estruturada.

Quanto ao carregamento de dados, têm-se as variáveis que armazenam o estoque de ingredientes (\$s0, \$s1, \$s2, \$s3) e o valor “0” em \$s5 utilizado como iterador de segurança para evitar *stack overflow* com saída a de loops quando usuário seleciona muitas vezes seguidas opções inválidas no teclado.

Quanto às chamadas de funções, têm-se a função **escolhe\_cafe**, que determina o tipo de bebida selecionada pelo usuário, a função **escolhe\_tamanho**, que define se o café será grande ou pequeno, e a função **com\_ou\_sem\_acucar**, que permite decidir o uso de açúcar e **cupom** que gera um arquivo TXT a partir de um template. Após essas etapas, o valor retornado pela função **escolhe\_tamanho** é comparado com os valores possíveis, determinando assim qual café será preparado pela máquina listados nos *branches*.

Ainda neste bloco foram implementados os labels **encerra\_pedido** e **cancela\_pedido**, importantes para controlar o fluxo de execução em um ponto único do código, deixando neste trecho aberta a possibilidade de melhoria a partir de tags ou flags para uma máquina de estados, por exemplo.

```

53 .text
54 #----- Bloco principal -----
55 main:
56     # Carrega as quantidades de doses de cada pó
57     lw     $s0,    cafe_dose
58     lw     $s1,    leite_dose
59     lw     $s2,    chocolate_dose
60     lw     $s3,    acucar_dose
61 inicio:
62     add    $s5,    $0, $0        #iterador de seguranca
63     # Printa mensagem de boas vindas e solicita selecao do tipo do café
64     la     $a0,    msg_boas_vindas
65     li     $v0,    4            #Printa mensagem para selecionar o café
66     syscall
67     # Imprime nova linha
68     li     $v0,    11           # Syscall para imprimir caractere
69     li     $a0,    10           # Caractere de nova linha (ASCII 10)
70     syscall
71     jal    escolhe_cafe        # Vai para o procedimento da escolha do tipo do café retorna em $s7
72
73     jal    escolhe_tamanho     # Vai p escolha tamanho do café, definindo qtde. de doses em $s6
74
75     jal    com_ou_sem_acucar   # Vai pra escolha se quer com ou sem açúcar e retorna em $s4
76
77     jal    cupom               #chama rotina impressao TXT
78     #Apos as escolhas, comparamos o tipo de cafe escolhidos armazenados em $t e vamos ao preparo
79     move   $t9,    $s6         # move valor de $s6 p $t9 pois $t9 decrementador iterações
80     beq    $s7,    1,          preparo_puro    # Se o valor selecionado for 1, café puro é escolhido
81     beq    $s7,    2,          preparo_leite   # Se o valor selecionado for 1, café com leite é escolhido
82     beq    $s7,    3,          preparo_mocha  # Se o valor selecionado for 1, mochaccino é escolhido
83     j      encerra_pedido     #volta para inicio
84 encerra_pedido:
85     jr     $ra
86 cancela_pedido:
87     lw     $s5,    0($sp)      # Pop do s
88     lw     $ra,    4($sp)      # Pop do return address
89     addi   $sp,    $sp,    8    # Pop na pilha de 3elementos*4=12
90     j      inicio

```

Figura 4 - Bloco principal

## Procedimento de escolha

Como mencionado, a partir do Bloco Principal por meio de instruções **jal** o fluxo de funcionamento do programa é definido.

O procedimento **escolhe\_cafe** é responsável por realizar a escolha do tipo de café (puro, com leite ou mochaccino) a ser preparado. O algoritmo segue os seguintes passos: inicialmente, o programa captura a entrada do usuário utilizando o **Digital Lab Sim** pela chamada da função **Dig\_Lab\_Sim** do bloco **Funcao p teclado/display 7-seg**. Os retornos possíveis são os inteiros 1, 2 ou 3, correspondendo às opções de café puro, com leite e mochaccino, respectivamente. Cada valor de retorno direciona para um subprocedimento específico, responsável por processar a escolha do usuário e exibir na tela uma mensagem indicando o tipo selecionado. Caso o usuário forneça uma entrada inválida, o programa exibe uma mensagem de erro e reinicia o procedimento até que uma entrada válida seja recebida.

O procedimento **escolhe\_tamanho** realiza a captura da entrada do usuário para determinar o tamanho do café (pequeno ou grande). Assim como no procedimento anterior, há um tratamento para entradas inválidas. O usuário deve selecionar 1 para café pequeno ou 2 para café grande. O valor retornado é armazenado no registrador **\$s6**, que é utilizado para definir a quantidade de doses necessárias na preparação do café. O algoritmo também faz uso de instruções **beq** para direcionar o fluxo para subprocedimentos responsáveis por exibir no console o tamanho escolhido.

Por fim, o procedimento **com\_ou\_sem\_acucar** trata da escolha do usuário em relação ao uso de açúcar no café. Após executar etapas similares às dos procedimentos anteriores, o programa solicita que o usuário selecione 0 para café sem açúcar ou 1 para café com açúcar. O valor retornado direciona para subprocedimentos que exibem a escolha no console. No entanto, há uma verificação adicional apresentada a seguir na Figura 5: no subprocedimento **com\_acucar**, é verificada a quantidade de doses de açúcar disponíveis no recipiente **\$s3** em relação à quantidade exigida para o preparo **\$s6**. Caso o estoque de açúcar seja insuficiente, ocorre um desvio para o procedimento **bloqueado**, onde a situação é tratada de maneira apropriada para a falta e reposição de ingrediente.

```
286      com_acucar:
287          # Printa a mensagem falando que o cliente escolheu com açúcar
288          la    $a0, msg_acucar_sim
289          li    $v0, 4
290          syscall
291          li    $v0, 11          # Syscall para imprimir caractere
292          li    $a0, 10          # Caractere de nova linha (ASCII 10)
293          syscall
294
295          blt    $s3, $s6, bloqueado # Se qtde. desejada açúcar $s6 for < que quantidade container $s3, vai p bloqueado
296
```

Figura 5 - Subprocedimento **com\_acucar** dentro de **com\_ou\_sem\_acucar** para bloqueio na falta de ingrediente

## Procedimentos de preparo

Os procedimentos **preparo\_puro**, **preparo\_leite** e **preparo\_mocha** são responsáveis pelo preparo dos diferentes tipos de café disponíveis. Cada um deles é implementado de forma semelhante, diferenciando-se apenas pelos ingredientes utilizados e pelas quantidades de iterações e verificações realizadas antes do preparo, conforme exemplo da Figura 6 abaixo.

Inicialmente, cada procedimento verifica a disponibilidade dos ingredientes nos respectivos recipientes. Caso a quantidade em estoque seja inferior à quantidade de doses requisitadas para o preparo, ocorre um desvio para o procedimento de tratamento específico **bloqueado**, que é responsável por gerenciar a situação de falta de insumos e evitar a continuidade do processo de preparo.

```
412 preparo_mocha:
413     # Verifica se a quantidade de Café, Leite e Chocolate é suficiente
414     blt $s0, $t9, bloqueado
415     blt $s1, $t9, bloqueado
416     blt $s2, $t9, bloqueado
417     jal libera_agua # Se tem ingrediente suficiente, começa preparação (Chama o procedimento que libera água)
418     mocha_cafe:
419         # Printa a mensagem "adicionando cafe"
420         la $a0, botando_cafe
421         li $v0, 4
422         syscall
423         li $a0, 1000 # Argumento para syscall 32 (milissegundos)
424         li $v0, 32 # syscall para sleep
425         syscall
426         li $v0, 11 # Syscall para imprimir caractere
427         li $a0, 10 # Caractere de nova linha (ASCII 10)
428         syscall
429         addi $s0, $s0, -1 # Cafe do container decrementa
430         addi $t9, $t9, -1 # Decrementa iterador
431         bgtz $t9, mocha_cafe # Enquanto doses faltantes de cafe nao foram colocadas, executa novamente loop
432         move $t9, $s6 # Se todas doses foram adicionadas, restaura tamanho dose de volta p $t9
```

Figura 6 - Exemplo da verificação da quantidade disponível no **preparo\_mocha**

Caso a verificação inicial não resulte em nenhuma das instruções de desvio condicional do tipo **branch if less than** (ou seja, a quantidade de doses em estoque é suficiente para preparar o café no tamanho solicitado), o procedimento **libera\_agua**, apresentado na Figura 7, é acionado.

```
312 #----- Libera a agua -----
313 libera_agua:
314     #push
315     addi $sp, $sp, -4 # Push na pilha de lelemento*4=4
316     sw $ra, 0($sp) # Push do return address
317     # Printa a mensagem "Liberando água..."
318     la $a0, liberando_agua
319     li $v0, 4
320     syscall
321     beq $s6, 2, libera_copo_grande # Caso escolhido grande, vai procedimento agua pra copo grande (10 segundos)
322     # Caso escolhido pequeno, libera agua por 5 segundos
323     li $a0, 5000 # Argumento para syscall 32 (5000 ms). Congela programa por aproximadamente 5 segundos
324     li $v0, 32 # Syscall para sleep
325     syscall
326     li $v0, 11 # Syscall para imprimir caractere
327     li $a0, 10 # Caractere de nova linha (ASCII 10)
328     syscall
329     j end_libera_agua
330     libera_copo_grande:
331         li $a0, 10000 # Argumento para syscall 32 (milissegundos)
332         li $v0, 32 # syscall para sleep
333         syscall
334         li $v0, 11 # Syscall para imprimir caractere
335         li $a0, 10 # Caractere de nova linha (ASCII 10)
336         syscall
337     end_libera_agua:
338         lw $ra, 0($sp) # Pop do return address
339         addi $sp, $sp, 4 # Pop na pilha de Selementos*4=20
340         jr $ra # Retorna a funcao chamadora
```

Figura 7 - Implementação da função **libera\_agua**

Esse procedimento apresenta uma implementação simples: ele verifica o valor armazenado no registrador **\$s6**, que representa o tamanho do café escolhido. Se o tamanho for pequeno, uma mensagem "Liberando água..." é exibida no console por 5 segundos, utilizando o comando 32 do **syscall** para congelar a execução do programa durante esse intervalo. Para o tamanho grande, o mesmo procedimento é seguido, porém o congelamento ocorre por 10 segundos, simulando o tempo adicional necessário para liberar um volume maior de água.

Após o retorno para a função chamadora, ocorre o preparo da bebida escolhida. Os procedimentos de preparo foram organizados em subprocedimentos, cada um responsável por realizar as iterações necessárias para adicionar o pó correspondente à bebida. Especificamente, o registrador **\$t9** é iterado um número de vezes igual ao valor armazenado em **\$s6** (tamanho do café), garantindo que o valor de **\$s6** permaneça inalterado durante o processo, conforme apresentado na Figura 8 abaixo.

```

372 #----- Preparo do café com leite -----
373 preparo_leite:
374 # Verifica se a quantidade de Café e Leite é suficiente
375 blt $s0, $t9, bloqueado
376 blt $s1, $t9, bloqueado
377 jal libera_agua # Se tem ingrediente o suficiente, começa a preparação (Chama o procedimento que libera água)
378 leite_cafe:
379 # Printa a mensagem "adicionando cafe"
380 la $a0, botando_cafe
381 li $v0, 4
382 syscall
383 li $v0, 11 # Syscall para imprimir caractere
384 li $a0, 10 # Caractere de nova linha (ASCII 10)
385 syscall
386 li $a0, 1000 # Argumento para syscall 32 (milissegundos)
387 li $v0, 32 # Syscall para sleep
388 syscall
389 addi $s0, $s0, -1 # Cafe do container decrementa
390 addi $t9, $t9, -1 # Decrementa iterador
391 bgtz $t9, leite_cafe # Se a quantidade de doses faltantes não for igual a 0, executa novamente o procedimento
392 move $t9, $s6 # Se todas doses foram adicionadas, restaura tamanho dose de volta p $t9
393
394 leite_leite:
395 # Printa a mensagem "adicionando leite"
396 la $a0, botando_leite
397 li $v0, 4
398 syscall
399 li $v0, 11 # Syscall para imprimir caractere
400 li $a0, 10 # Caractere de nova linha (ASCII 10)
401 syscall
402 li $a0, 1000 # Argumento para syscall 32 (milissegundos)
403 li $v0, 32 # syscall para sleep
404 syscall
405 addi $s1, $s1, -1 # leite do container decrementa
406 addi $t9, $t9, -1 # Contador decrementa
407 bgtz $t9, leite_leite # Enquanto as doses de leite faltante não for igual a 0, executa novamente o procedimento
408 move $t9, $s6 # Se todas doses foram adicionadas, restaura tamanho dose de volta p $t9
409 li $t0, 1
410 beq $s4, $t0, preparo_acucar # Se $s4 == 1, vai p "preparo_acucar"
411 j inicio # senao, acabou preparo e volta p início

```

Figura 8 - Exemplo de procedimento de preparo do café com leite no procedimento **prepara\_leite**

Essa abordagem de subprocedimentos foi adotada para evitar que o estoque de ingredientes mude durante a adição ao copo, algo que pode não refletir o funcionamento de máquinas reais de café. Assim, assegura-se que cada ingrediente selecionado seja adicionado integralmente, um de cada vez. Ao final da execução de cada subprocedimento, o valor do registrador de iteração **\$t9** é restaurado para



possibilitar iterações subsequentes nas funções seguintes, utilizando a instrução ``move $t9, $s6``.

Com a conclusão da etapa de preparo, segue-se para o procedimento de adição do açúcar **preparo\_acucar**. Este procedimento, apresentado abaixo na Figura 9, é chamado independentemente da escolha do usuário, com a diferença de que, caso o açúcar não tenha sido selecionado (ou seja, se o valor armazenado em **\$t2** for 0), o fluxo é direcionado diretamente para o término do programa. Caso contrário, a execução do procedimento prossegue.

```
467 #----- Coloca o açúcar depois de preparar o café -----
468 preparo_acucar:
469     blez    $s3,    inicio    # Se quantidade de açúcar a botar for 0, ele pula e segue adiante
470     blt     $s3,    $t9,    bloqueado    # Verifica se a quantidade de açúcar requisitado é maior que a disponível no container
471     # Imprime a mensagem "adicionando açúcar"
472     la      $a0,    botando_acucar
473     li      $v0,    4
474     syscall
475     li      $a0,    1000    # Argumento para syscall 32 (milissegundos)
476     li      $v0,    32    # syscall para sleep
477     syscall
478     li      $v0,    11    # Syscall para imprimir caractere
479     li      $a0,    10    # Caractere de nova linha (ASCII 10)
480     syscall
481     addi    $t9,    $t9,    -1
482     addi    $s3,    $s3,    -1
483
484     bgtz    $t9,    preparo_acucar    # se $t9 >0 itera, se $t9<= 0 segue
485     move    $t9,    $s6    # Se todas doses foram adicionadas, restaura tamanho dose de volta p $t9
486     j      inicio
```

Figura 9 - Bloco do procedimento que adiciona o açúcar (**preparo\_acucar**)

Durante a execução, é exibida uma mensagem indicando que o açúcar está sendo adicionado. Em seguida, o registrador de iteração **\$t9** e a quantidade de doses disponíveis no container são decrementados. Se o iterador **\$t9** alcançar o valor 0, indica que todas as iterações necessárias foram concluídas, e o programa inicia novamente.

## Procedimentos externos

Os blocos de **Função para teclado/display 7-seg** e **Função para impressão cupom** são mencionados aqui como externos pois podem representar uma abstração de *hardware* onde funcionam como *procedures* chamadas para comunicação com módulos eletrônicos externos. Por exemplo, um teclado com display ou impressora de cupons fiscais.

No bloco *Funcao p teclado/display 7-seg*, o código implementa um simulador de interação com o teclado matricial e o display de 7 segmentos do Digital Lab Sim do próprio MARS 4.5 . Inicialmente, ele configura os registradores necessários, empilha o estado dos registradores relevantes na pilha e define os endereços de memória para o teclado, display e códigos de tecla pressionada. Um loop principal realiza a leitura do teclado, verificando linha por linha se uma tecla foi pressionada. Caso uma tecla seja detectada, é processada em *processa\_tecla*. O processamento identifica se a tecla pressionada corresponde a uma entrada válida (teclas 0, 1, 2, 3 ou 5), atualizando o display e armazenando o valor associado. Caso contrário,

incrementa um contador de tentativas inválidas e emite uma mensagem de erro até que o número limite de tentativas seja atingido.

Se o limite de tentativas inválidas for excedido, o programa emite uma mensagem de aviso e volta para a chamadora. As teclas válidas são representadas por códigos específicos, e cada código leva a uma rotina que atualiza o display com o valor correspondente. No final do processamento, o programa restaura a pilha e retorna para a chamadora. A Figura 10 abaixo apresenta de forma resumida as principais partes do procedimento, o que pode ser visto com mais detalhe no arquivo `0_cafe_v11.2.asm`.

```

626 #----- Funcao p teclado/display 7-seg -----
627 Dig_Lab_Sim:
628     addi    $sp,    $sp,    -20          # Push na pilha de Selementos*4=20
629     sw      $ra,    16($sp)             # Push do return address
630     sw      $a1,    12($sp)             # Push $a1 pra qndo implementar tag de controle
631     sw      $s2,    8($sp)              # Push de $s em uso
632     sw      $s1,    4($sp)              # Push de $s em uso
633     sw      $s0,    0($sp)              # Push de $s em uso
634
635     li      $s0,    0xFFFF0010         # Endereço do display de 7 segmentos direita
636     li      $s2,    0xFFFF0012         # Endereço teclado
637     li      $s1,    0xFFFF0014         # Endereço do código da tecla pressionada
638     li      $t3,    -1                  # Inicializa $t3 com valor inválido
639     li      $t4,    0                   # Contador de tentativas
640     lb      $t7,    g_segment           # Exibe inicialmente apenas segmento G
641     sb      $t7,    0($s0)
642
643 > loop: ...
644 > processa_tecla: ...
645 > load_zero: ...
646 > load_um: ...
647 > load_dois: ...
648 > load_tres: ...
649 > load_cinco: ...
650 > atualiza_display: ...
651 > limite_tentativas: ...
652
653 fim_Dig_Lab_Sim:
654     # Pausa de 1 segundo
655     li      $a0,    1000                # 1000ms argumento p syscall 32
656     li      $v0,    32                  # syscall para sleep
657     syscall
658     lw      $s0,    0($sp)              # Pop de $s em uso
659     lw      $s1,    4($sp)              # Pop de $s em uso
660     lw      $s2,    8($sp)              # Pop de $s em uso
661     lw      $a1,    12($sp)             # Pop $a1 pra qndo implementar tag de controle
662     lw      $ra,    16($sp)             # Pop do return address
663     addi    $sp,    $sp,    20          # Pop na pilha de Selementos*4=20
664     jr      $ra

```

Figura 10 - Procedimento resumido de teclado e display.

Já o procedimento **cupom** do bloco **Função para impressão cupom** foi baseado no exemplo disponível em *Help* no simulador *MARS 4.5* e o código foi agregando funcionalidades para se tornar uma rotina que manipula um template textual gerando um cupom customizado com informações de data, hora e produto, baseando-se nos valores de entrada fornecidos nos registradores ``$s6`` e ``$s7``. Inicialmente, o código utiliza a pilha para salvar os registradores necessários. Em seguida, há validações de entrada para garantir que ``$s6`` esteja no intervalo de 1 a 2 e ``$s7`` no intervalo de 1 a 3, com tratamento de erro para entradas inválidas. Com base nos valores válidos, o código seleciona as strings de produtos e as substitui em partes específicas do arquivo `Template_Cupom_Fiscal_0.txt` para posteriormente gerar o arquivo de saída `cupom_fiscal.txt`.

A seguir, com leitura e escrita por *syscall*, o arquivo *Template\_Cupom\_Fiscal\_0.txt* é aberto, lido em um buffer, onde ocorre a substituição de marcadores (`@`, `&`, e `\*`) por informações de data, hora e o produto selecionado respectivamente. Após substituir o conteúdo do template no buffer, ele é escrito em um arquivo de saída *cupom\_fiscal.txt*. No final, o código fecha os arquivos, restaura os valores originais dos registradores a partir da pilha e retorna ao chamador, garantindo que o estado inicial seja preservado.

Essa procedure para o cupom fiscal foi a mais extensa implementada e poderia ainda chamar outro procedimento que, por exemplo, capturasse a data e hora do sistema ou de um *RTC (Real Time Clock)*, para substituir tais campos por dados válidos. Mesmo assim, sua implementação atual já merece ser vista diretamente no arquivo *0\_cafe\_v11.2.asm* e uma amostra de sua estrutura é apresentada na Figura 11 abaixo.

```

730  cupom:
731      addi $sp, $sp, -20          # Reservar espaço na pilha
732      sw   $ra, 16($sp)          # Salvar $ra
733      sw   $s7, 12($sp)         # Salvar $s
734      sw   $s6, 8($sp)          # Salvar $s
735      sw   $s1, 4($sp)          # Salvar $s
736      sw   $s0, 0($sp)          # Salvar $s
737
738
739 >  # Validação de entrada segura...
750 >  # Selecionar produto com base em $s6 e $s7...
761 >  check_1: ...
770 >  check_2: ...
779 >  use_prod_1_1: ...
782 >  use_prod_1_2: ...
785 >  use_prod_1_3: ...
788 >  use_prod_2_1: ...
791 >  use_prod_2_2: ...
794 >  use_prod_2_3: ...
798 >  end_check: ...
804 >  replace_prod: ...
818 >  done_replace: ...
841 >  read_loop: ...
855 >  replace_loop: ...
877 >  replace_date_section: ...
881 >  replace_time_section: ...
885 >  replace_prod_section: ...
889 >  perform_date_replace: ...
903 >  next_char_date: ...
906 >  perform_time_replace: ...
920 >  next_char_time: ...
923 >  perform_prod_replace: ...
937 >  next_char_prod: ...
940 >  next_char: ...
945 >  write_buffer: ...
955 >  end_read: ...
975 >  invalid_input: ...
980 >  erro_arquivo: ...
985  erro_leitura:
986      # Tratamento para erro de leitura de arquivo
987      li   $v0, 10              # Código para encerrar programa
988      syscall

```

Figura 11 - Implementação resumida da função **cupom**

## Interrupções do programa

No programa da máquina de café, o fluxo de execução pode ser interrompido em decorrência de ações do usuário ou de verificações internas realizadas pelo sistema. Uma das possíveis interrupções no fluxo de execução ocorre em razão do excesso de entradas inválidas no procedimento **Dig\_Lab\_Sim** ou nos testes de limite de tentativas de escolha. Nesse caso, se o usuário selecionar uma opção inválida por três vezes consecutivas, o programa interrompe sua execução. Como resultado, uma mensagem informando sobre o ocorrido é exibida na tela, indicando o motivo da interrupção do fluxo, e o programa retorna para a chamadora, garantindo um comportamento robusto diante de entradas incorretas. Tal implementação pode ser vista de maneira resumida no **loop** e em **limite\_tentativas** de **Dig\_Lab\_Sim** da Figura 12 abaixo.

```
627 Dig_Lab_Sim:
643     loop:
644         bge    $t4, 3,      limite_tentativas # P dar limite selecoes invalidas e cancelar compra
645         li     $t0, 0x01    # Seleciona linha 1
646         sb     $t0, 0($s2)  # Escreve em $s2 para selecionar a linha
647         lbu    $t1, 0($s1)  # Carrega tecla pressionada da linha 1
648         bne    $t1, $zero, processa_tecla
649         li     $t0, 0x02    # Seleciona linha 2
650         sb     $t0, 0($s2)  # Escreve em $s2 para selecionar a linha
651         lbu    $t1, 0($s1)  # Carrega tecla pressionada da linha 1
652         bne    $t1, $zero, processa_tecla
653         li     $t0, 0x04    # Seleciona linha 3
654         sb     $t0, 0($s2)  # Escreve em $s2 para selecionar a linha
655         lbu    $t1, 0($s1)  # Carrega tecla pressionada da linha 1
656         bne    $t1, $zero, processa_tecla
657         li     $t0, 0x08    # Seleciona linha 4
658         sb     $t0, 0($s2)  # Escreve em $s2 para selecionar a linha
659         lbu    $t1, 0($s1)  # Carrega tecla pressionada da linha 1
660         bne    $t1, $zero, processa_tecla
661         j      loop        # Repete o loop
662
663 > processa_tecla: ...
683 > load_zero: ...
688 > load_um: ...
693 > load_dois: ...
698 > load_tres: ...
703 > load_cinco: ...
708 > atualiza_display: ...
712     limite_tentativas:
713         # Imprime mensagem de limite de tentativas
714         li     $v0, 4
715         la     $a0, msg_limite
716         syscall
717     fim_Dig_Lab_Sim:
718         # Pausa de 1 segundo
719         li     $a0, 1000    # 1000ms argumento p syscall 32
720         li     $v0, 32     # syscall para sleep
721         syscall
722         lw     $s0, 0($sp)  # Pop de $s em uso
723         lw     $s1, 4($sp)  # Pop de $s em uso
724         lw     $s2, 8($sp)  # Pop de $s em uso
725         lw     $a1, 12($sp) # Pop $a1 pra qndo implementar tag de controle
726         lw     $ra, 16($sp) # Pop do return address
727         addi   $sp, $sp, 20 # Pop na pilha de 5elementos*4=20
728         jr     $ra
```

Figura 12 - Resumo da implementação de testes de entradas inválidas em **Dig\_Lab\_Sim**

Outra interrupção possível é a falta de doses de um determinado pó requisitado. Isso ocorre se a verificação no início de todos os procedimentos de preparo for verdadeira, como apresentado no exemplo da Figura 13, resultando em um salto para o **bloqueado**. Nesse procedimento, o programa avisa sobre a falta do pó e entra em um loop até que o usuário selecione o número 5. A seleção leva a **reabastece**, onde o pó requisitado será reabastecido. A verificação se a quantidade de doses requisitada é suficiente ocorre da seguinte forma: após o salto para o procedimento de preparo, é feita uma verificação para cada pó requisitado, comparando a quantidade de pó armazenada no recipiente com a quantidade de doses requisitadas para o preparo do café (em \$s6). Dessa forma, se o usuário solicitar um café grande e houver apenas uma dose de café disponível no recipiente, o preparo será bloqueado. Por outro lado, se o usuário pedir uma dose pequena, o preparo será realizado.

```

412 preparo_mocha:
413     # Verifica se a quantidade de Café, Leite e Chocolate é suficiente
414     blt    $s0,    $t9,    bloqueado
415     blt    $s1,    $t9,    bloqueado
416     blt    $s2,    $t9,    bloqueado
417     jal    libera_agua # Se tem ingrediente suficiente, começa a preparação (Chama o procedimento que libera água)
418     mocha_cafe:
419         # Printa a mensagem "adicionando cafe"
420         la    $a0,    botando_cafe
421         li    $v0,    4
422         syscall
423         li    $a0,    1000          # Argumento para syscall 32 (milissegundos)

```

Figura 13 - Exemplo de verificação dos ingredientes em **preparo\_mocha**

Dentro do procedimento **bloqueado**, apresentado na Figura 14 abaixo, há subprocedimentos responsáveis por tratar e imprimir na tela o pó faltante. Inicialmente, é verificado qual ingrediente está faltando, comparando a quantidade de doses armazenadas no recipiente com a quantidade necessária. Assim que o pó faltante é identificado, ocorre um salto para o subprocedimento que informará o que falta para o preparo do café. Em seguida, há um salto incondicional para o subprocedimento **bloqueado\_aguarda**, onde o programa executará um loop até que o usuário pressione a tecla de reabastecimento. Caso a tecla selecionada seja inválida, o programa emitirá uma mensagem no console e o loop continuará.

```

489 bloqueado:
490     blt    $s1,    $s6,    falta_leite
491     blt    $s2,    $s6,    falta_chocolate
492     blt    $s3,    $s6,    falta_acucar
493 >     falta_cafe: ...
501 >     falta_leite: ...
509 >     falta_chocolate: ...
517 >     falta_acucar: ...
524     bloqueado_aguarda:
525     # addi    $sp,    $sp,    -4          # Push na pilha de 1elemento*4=4
526     # sw     $ra,    0($sp)          # Push do return address
527
528     jal    Dig_Lab_Sim
529     move    $t1,    $v1
530     beq     $t1,    5,    reabastece    # Se o valor 5 for digitado, reabastece
531
532     # Printa msg_invalido caso usuario selecione uma opcao invalida
533     la    $a0,    msg_invalido
534     li    $v0,    4
535     syscall
536     li    $v0,    11          # Syscall para imprimir caractere
537     li    $a0,    10          # Caractere de nova linha (ASCII 10)
538     syscall
539     j      bloqueado_aguarda    # Senao volta o loop

```

Figura 14 - Implementação do subprocedimento **bloqueado\_aguarda**

Na última etapa, após a tecla necessária ser selecionada, ocorre o reabastecimento do pó escolhido. O processo segue da seguinte forma: primeiramente, são impressas na tela as opções disponíveis para o reabastecimento. Em seguida, o programa passa por uma série de verificações (**beq**) comparando ao valor retornado por **Dig\_Lab\_Sim**. Dependendo do valor, ocorre desvio para o subprocedimento correspondente. Caso um número inválido seja selecionado, o programa avisa no console e reinicia o procedimento. Após a escolha do pó a ser reabastecido, o recipiente é totalmente carregado (20 doses) usando a instrução **li**, que carrega o valor 20 em registrador. Em seguida, ocorre a instrução **sw**, que armazena o valor carregado na memória. Após concluir todas as etapas necessárias, a pilha é esvaziada e o programa realiza um salto incondicional de volta para o main, onde o programa será reexecutado. Essa implementação é apresentada de maneira resumida na Figura 15 abaixo.

```

541 reabastece:
542     la    $a0, reabastecer_po
543     li    $v0, 4
544     syscall
545
546     li    $v0, 11          # Syscall para imprimir caractere
547     li    $a0, 10          # Caractere de nova linha (ASCII 10)
548     syscall
549
550     #addi  $sp, $sp, -4      # Push na pilha de 1elemento*4=4
551     #sw    $ra, 0($sp)      # Push do return address
552
553     jal    Dig_Lab_Sim
554     move   $t1, $v1
555
556     beq    $t1, 0, reabastece_cafe
557     beq    $t1, 1, reabastece_leite
558     beq    $t1, 2, reabastece_chocolate
559     beq    $t1, 3, reabastece_acucar
560     # Printa msg_invalido caso usuario selecione uma opcao invalida
561     la    $a0, msg_invalido
562     li    $v0, 4
563     syscall
564     li    $v0, 11          # Syscall para imprimir caractere
565     li    $a0, 10          # Caractere de nova linha (ASCII 10)
566     syscall
567     j      reabastece
568
569 > reabastece_cafe: ...
583 > reabastece_leite: ...
596 > reabastece_chocolate: ...
609 > reabastece_acucar: ...
622     lw    $ra, 16($sp)      # Restaura o return address de $ra
623     addi   $sp, $sp, 20      # Ajusta o ponteiro da pilha para limpar o espaço
624     j      main

```

Figura 15 - Bloco de reabastecimento apresentado de maneira resumida

De maneira mais detalhada, é apresentada na Figura 16 abaixo a implementação da reposição de ingredientes. Para mais detalhes, convém verificar esses blocos de bloqueio e abastecimento no arquivo *0\_cafe\_v11.2.asm*.

```


596 reabastece_chocolate:
597     la    $a0, po_reabastecido
598     li    $v0, 4
599     syscall
600     li    $v0, 11          # Syscall para imprimir caractere
601     li    $a0, 10          # Caractere de nova linha (ASCII 10)
602     syscall
603     li    $t1, 20000000
604     sw    $t1, chocolate_dose
605     # # Esvaziar a pilha antes de voltar ao main
606     # lw    $ra, 16($sp)    # Restaura o return address de $ra
607     # addi  $sp, $sp, 20    # Ajusta o ponteiro da pilha para limpar o espaço
608     j     main             # TEM QUE ESVAZIAR A PILHA ANTES

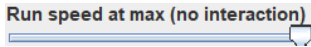
```

Figura 16 - Exemplo de reabastecimento do container do chocolate

## Instruções de uso

Para o uso do código da máquina, de maneira que ocorra o funcionamento correto do programa, evitando interrupções desnecessárias, convém que:

- O executável do simulador MARS 4.5 esteja no mesmo diretório do código fonte;
- O arquivo *Template\_Cupom\_Fiscal\_0.txt*, que será usado como base para o cupom fiscal, também deve estar no mesmo diretório do executável do simulador MARS 4.5;
- O código *0\_cafe\_v11.2.asm* pode estar em pasta diferente do executável desde que seja devidamente indicado o seu caminho quando for aberto;
- Com o executável do simulador MARS 4.5 rodando, abrir em *Tools* o *Digital Lab Simulator*, e clicar em *Connect to MIPS*;
- Clicar no botão  *Assemble the current file and clear breakpoints*;

- Escolher a velocidade de execução *Run speed at max*  para a execução normal no programa;

Obs.: Caso o simulador trave ou ocorram bugs na impressão no terminal ou seleção de botões, pode ser necessário utilizar velocidade diferente, por exemplo *Run speed 30 inst/sec*, mas pode ser tedioso esperar a execução do bloco de cupom fiscal.

- Devido à “aderência” das teclas do *Digital Lab Sim*, para funcionamento correto do programa em execução, o usuário deve pressionar uma vez na tecla que deseja selecionar e imediatamente pressionar novamente para “soltá-la”, dessa forma, evitando que ela se mantenha fixada para não escolher uma opção indesejada nas próximas etapas de execução;
- O usuário deve se orientar pelas instruções dadas no terminal, para que assim possa saber qual tecla selecionar para o preparo do café desejado;
- O estoque de cada produto no arquivo *0\_cafe\_v11.2.asm* foi setado para 1, ou seja, logo no primeiro preparo o usuário já pode testar a funcionalidade de bloqueio e reposição de ingredientes que os colocam em 20;
- Ao final de cada preparação, o usuário pode encontrar o arquivo de cupom fiscal *cupom\_fiscal.txt* no mesmo diretório onde roda o simulador *MARS 4.5*.



## Conclusões

Apesar do sucesso no cumprimento dos objetivos principais, algumas melhorias foram identificadas e podem ser implementadas para aprimorar o sistema. Dentre elas, a inclusão de um bloco de relógio que, utilizando timestamp via *syscall*, pode registrar data e hora no cupom fiscal de maneira mais fidedigna. Essa funcionalidade extra até começou a ser implementada pela dupla e testada via terminal e alguns exemplos da funcionalidade podem ser encontrados nos arquivos *4\_relogio\_v6.asm* e *4\_relogio\_v7.asm*.

Também foi considerado o controle de execução por uma máquina de estados ao invés do fluxo de utilização, pensada e esquematizada para a organização das etapas conforme a Figura 17 abaixo, que apesar de não ter sido efetivamente implementada serviu como guia para que nenhuma funcionalidade especificada fosse esquecida.

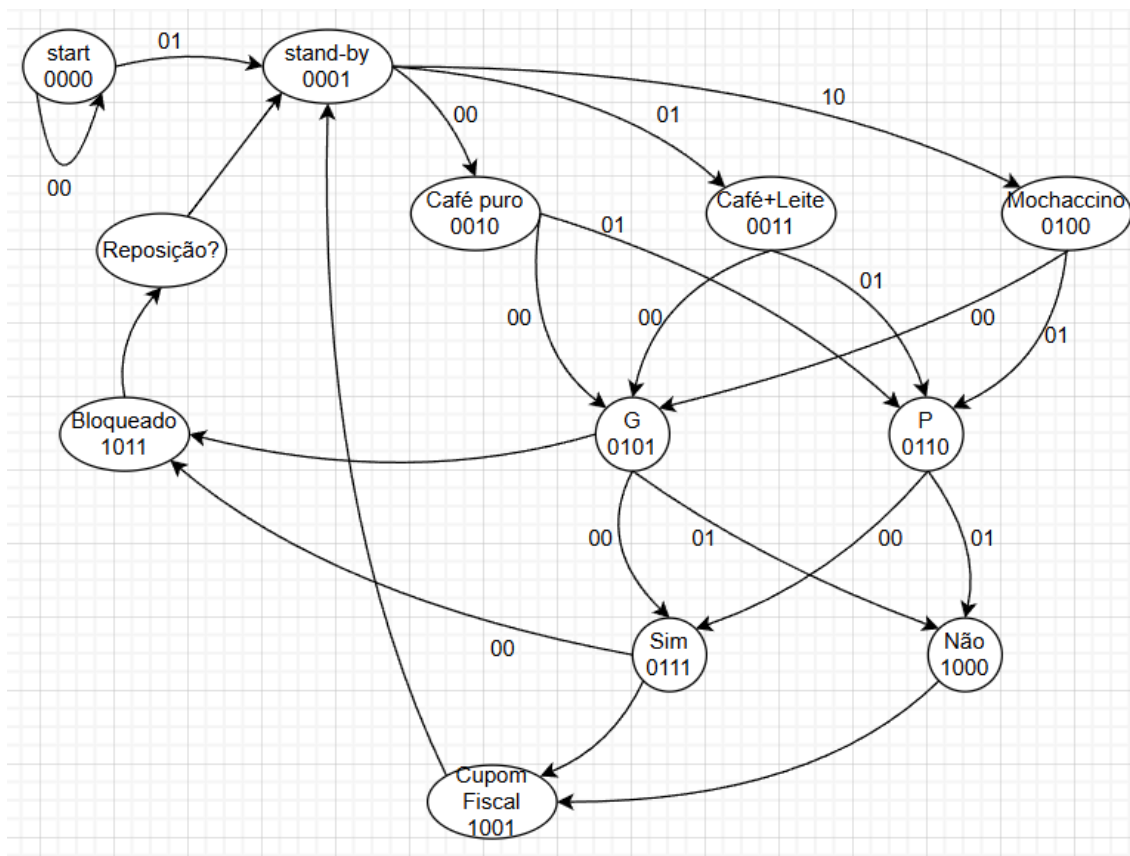


Figura 17 - Máquina de estados base

Além disso, uma outra possível melhoria é o aproveitamento do Display de 7 segmentos como uma tela auxiliar durante a execução, podendo indicar as etapas da execução ou o estado atual da máquina, por exemplo S para seleção, P para preparo, etc.



No decorrer do projeto a dupla se deparou com as dificuldades de organizar os blocos funcionais e por isso a divisão das tarefas, o que foi facilitado pela interface gráfica do *VS Code*. Outra dificuldade foi o cumprimento mútuo dos cuidados com a convenção de chamadas, evitando sobrescrever registradores. Neste mesmo âmbito de chamadas, procedimentos e subrotinas, as rotinas do tipo folha exigiram cuidados extras para restauração de contexto. A principal lição que elas deixaram, e são úteis tanto para Assembly quanto para linguagens de alto nível foi: uma boa prática de programação deve estipular apenas uma única localização dentro de um loop ou bloco funcional para a saída/retorno da rotina, ou seja, exclusivamente no início ou no final.

Por fim, atingiu-se o objetivo do projeto em simular um sistema funcional e didático para o aprendizado de conceitos em organização de computadores, programação em Assembly ou outras linguagens de mais alto nível. As melhorias sugeridas, caso implementadas, expandiriam ainda mais a funcionalidade e a aplicação prática do sistema, tornando-o mais próximo de uma solução real para máquinas de café comerciais.