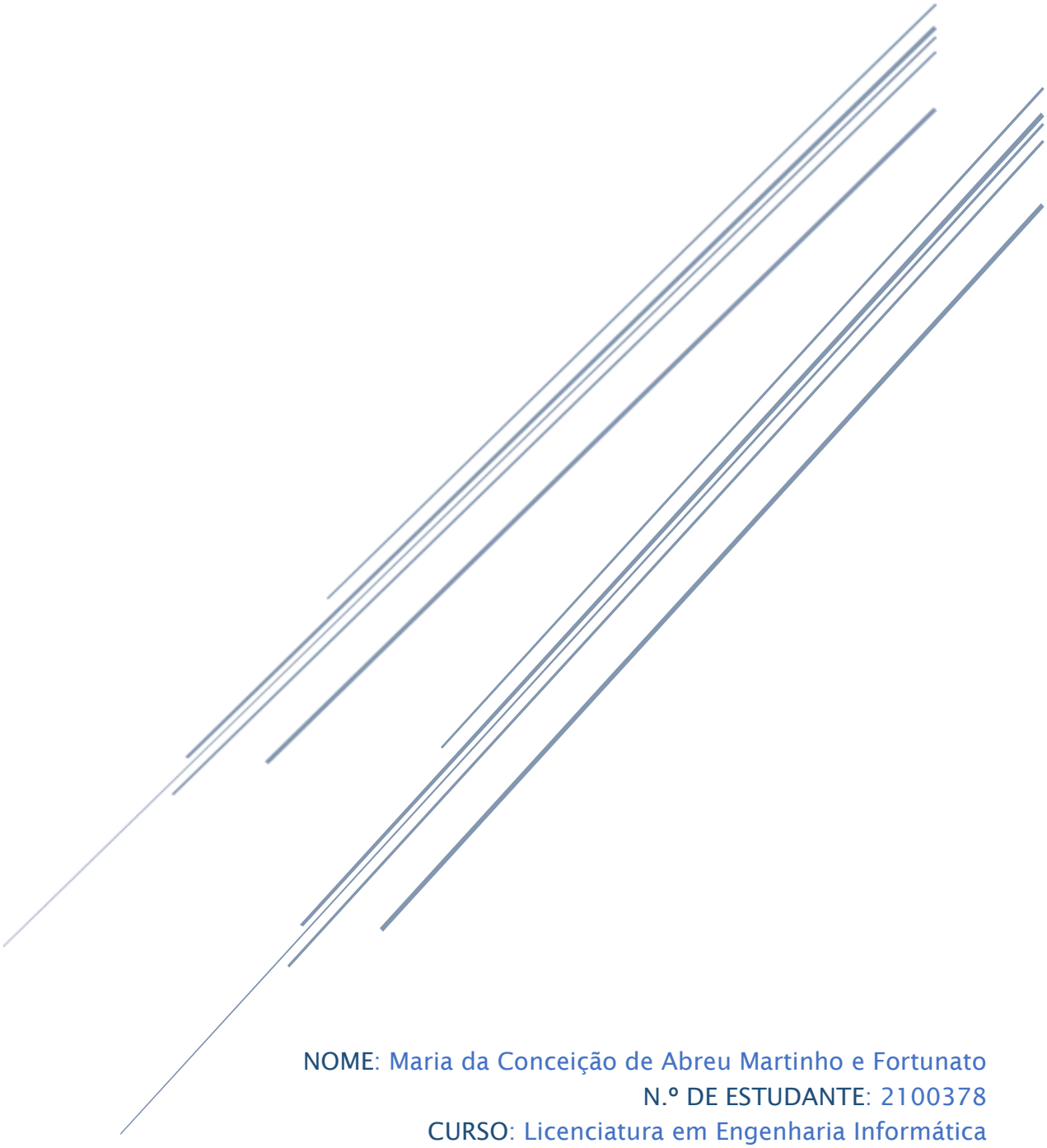
RELATÓRIO

E-folio B

UNIDADE CURRICULAR: Arquitetura de Computadores
CÓDIGO: 21010
DOCENTE: Carlos Sousa
TUTOR: Filipe Ramos



NOME: Maria da Conceição de Abreu Martinho e Fortunato
N.º DE ESTUDANTE: 2100378
CURSO: Licenciatura em Engenharia Informática
TURMA: 2025-03

Índice:

1 Estrutura global do programa

O objetivo é construir um programa em Assembly para o processador pedagógico P3, que permita o comando e controlo de um sistema de movimento linear constituído, especialmente, por: motor de passo, fuso (eixo), rolamento e base de movimento.

O programa enunciado era suposto ser desenvolvido em 4 alíneas separadas, que têm um carácter incremental. Tendo isto em conta pareceu-me melhor desenvolver o programa de forma integrada, num único programa fácil de testar e de avaliar.

Optei por uma estrutura modular que permitisse uma leitura e funcionamento mais eficiente, de acordo com as explicações que se seguem.

1.1 Origem dos dados

Os dados são armazenados em memória a partir do endereço 8000h, conforme os requisitos do software - (inscritos no enunciado do E-folio B).

Isto é feito através uso da diretiva ou pseudo-instrução ORIG 8000h que é destinada a ser tratada pelo assembler.

1.2 Origem do programa

O programa é armazenado em memória a partir do endereço 0000h, conforme os requisitos do software, inscritos no enunciado do E-folio B.

Aqui foi usada a diretiva ORIG também, agora com o endereço 0000h.

1.3 Pilha,

Embora não seja solicitado nos requisitos do software, é necessário iniciar apontador para a pilha com um endereço, por forma a permitir a chamada a rotinas e sub-rotinas, e salvaguarda do conteúdo dos registos (R1 a R7) para uso posterior.

Foi escolhido o endereço FDFFh, que tem em conta as seguintes contingências:

- O espaço de memória endereçável do P3 é de 64 k palavras (2^{16}) pelo que a memória total ocupa os endereços 0 a 65535, em decimal ou 0h a FFFFh;
- Os últimos 255 endereços (FF00h a FFFFh) estão reservados para dispositivos de entradas e saídas (I/O);
- Os anteriores 255 endereços (FE00h a FEFFh) estão reservados para a gestão de interrupções;
- A pilha tem funcionamento inverso, os endereços usados de seguida serão o ponteiro da pilha (SP) – 1.

Evitando, assim, conflito com as zonas de I/O e interrupções e garantindo espaço suficiente para as chamadas aninhadas de rotinas.

Na arquitetura do processador P3, embora possam ser usados 2 operandos, um deles tem de ser um registo, obrigatoriamente, pelo que tem de se dividir a inicialização da Pilha em 2 instruções:

1. Copia o endereço do apontador inicial para um registo acessível ao programador escolhi o registo R2 – ver linha 20 do programa;
2. Copia o endereço guardado no registo escolhido para o Registo da Pilha (registo 14 – SP) – ver linha 21 do programa.

1.4 Variáveis

As variáveis que criadas estão em linha com os requisitos do software, inscritos no enunciado do E-folio B.

POSICAO_ATUAL – Conterá a coordenada, em milímetros (mm) da posição atual da base de movimento, fica armazenada no endereço de memória **8000h**.

POSICAO_DESTINO - Conterá a coordenada, em mm da posição de destino da base de movimento, fica armazenada no endereço de memória **8001h**.

A diferença entre as duas variáveis é a distância, em mm, que a base tem de percorrer.

_SENTIDO_MOVIMENTO – Guardará o sentido do movimento que a base tem de percorrer: 0, para movimento para a esquerda e 1, para movimento para a direita. Fica armazenada no endereço de memória **8002h**.

As variáveis foram inicializadas com 0.

Estas variáveis são partilhadas entre as várias rotinas permitindo que as mesmas comuniquem o estado atual do sistema.

1.5 Constantes

Recorri às diretivas (pseudo-instruções) EQU e STR para criar constantes com o objetivo de tornar o código mais legível e facilitar qualquer revisão dos valores atribuídos inicialmente, passando a ser necessário alterar estes valores apenas uma vez.

IO_SENSOR – Em vez do endereço FFF9h, usado na rotina fornecida LE_SENSOR, foi iniciada com o referido endereço, através da já referida diretiva EQU.

O endereço FFF9h corresponde ao primeiro dos interruptores, que é usado para simular a ativação do sensor de origem. A rotina foi atualizada com a etiqueta desta constante.

IO_DIRECAO - Em vez do endereço FFF8h, usado nas rotinas fornecidas SET_DIRECAO e MOVIMENTO, foi iniciada com o referido endereço através da referida diretiva EQU.

O Endereço FFF8h corresponde ao conjunto de 16 LEDs, sendo usados os últimos 2 bits no programa. As rotinas foram atualizadas com a etiqueta desta constante

SP_INICIAL – Conterá o endereço do ponteiro da pilha (SP), foi iniciada com o endereço FDFFh através da referida diretiva EQU.

LIMITE_MIN – Usada para a verificação dos limites, na rotina construída para a alínea b) e reaproveitada na alínea c), foi iniciada com o valor 0 através da diretiva EQU, que corresponde à posição absoluta, em mm, mais à esquerda, que é também a posição inicial, após a calibragem inicial do sistema.

LIMITE_MAX - Usada para a verificação dos limites, na rotina construída para a alínea b) e reaproveitada na alínea c), foi iniciada com o valor 100 através da diretiva EQU, que corresponde à posição absoluta, em mm, mais à direita, que é também comprimento total do eixo.

DESTINOS – Trata-se de um vetor usado para tratar a sequência de destinos (posições absolutas, em mm) requeridos na alínea d). Foi iniciada, através da diretiva STR com os valores 30, 10, 50, 115, 50. O penúltimo valor está fora dos limites falados anteriormente, e destina-se a testar o comportamento das rotinas criadas para as para as alíneas b) e c), e o último valor destina-se a testar o comportamento das rotinas quando as posições atual e de destino coincidem.

1.6 Fluxo do programa e controlo de fluxo

O fluxo do programa é controlado a partir da rotina MAIN, que é responsável por inicializar a pilha, chamar as rotinas de calibragem do sistema (alínea a), e de processamento dos destinos (alínea d), e, no final, termina a execução.

O fluxo do programa é então como segue:

Após a definição da origem dos dados, é feita a definição e iniciação de variáveis e constantes.

De seguida o programa entra no controlador do fluxo, a rotina MAIN.

A rotina MAIN inicializa o ponteiro da pilha, e chama a rotina INICIAR_SISTEMA (Alínea a), para calibragem do sistema.

Após a correta calibragem, com a POSICAO_ATUAL situada na posição absoluta 0, a rotina MAIN chama a rotina PROCESSA_DESTINOS.

PROCESSA_DESTINOS percorre o vetor de destinos e, para cada posição, usa as rotinas das alíneas b) e c) para verificar se o movimento está dentro dos limites 0–100 mm. Se o movimento for possível, calcula o sentido (esquerda ou direita) e o número de impulsos necessários, executa o movimento e atualiza a variável POSICAO_ATUAL.

2 Fluxo e rotinas:

Irei agora detalhar, para cada alínea as rotinas criadas e o seu funcionamento.

2.1 Alínea a:

É pedida uma rotina, ou rotinas, que processe a calibragem do sistema na posição absoluta 0, uma vez que quando iniciado o sistema pode estar em qualquer posição aleatória.

2.1.1 Rotinas criadas

Para esta alínea foram criadas as rotinas INICIAR_SISTEMA, POSICAO_INICIAL as quais chamam as rotinas fornecidas SET_DIRECAO, IMPULSO (que chama a rotina DELAY) e LE_SENSOR.

2.1.2 Tarefas que executam

A rotina INICIAR_SISTEMA define o sentido de rotação para a esquerda e chama a rotina fornecida SET_DIRECAO, a qual ativa o sinal de direção do motor e escreve na memória, na variável _SENTIDO_MOVIMENTO a direção do movimento (esquerda ou direita).

Na fase de calibragem do sistema o sentido escrito na variável _SENTIDO_MOVIMENTO é esquerda, pois procuramos chegar à posição 0, que é a mais à esquerda.

A rotina POSICAO_INICIAL chama as rotinas fornecidas IMPULSO e LE_SENSOR para dar os passos ao motor, e repete até que o sensor de origem se ative, altura em que atualiza a variável POSICAO_ATUAL com o valor 0, terminando assim o processo de calibração do sistema.

2.1.3 Entradas

À entrada a rotina INICIAR_SISTEMA define o sentido de rotação para a esquerda, escrevendo 0 no registo acumulador (R1=0).

2.1.4Saídas

À saída restaura o valor do registo R1, preservado na pilha.

Garante que o valor variável _SENTIDO_MOVIMENTO = 0 (endereço de memória 8002h) – valor do sentido para a esquerda, bem como o valor da variável POSICAO_ATUAL = 0 (endereço de memória 8000h) – valor da posição Inicial.

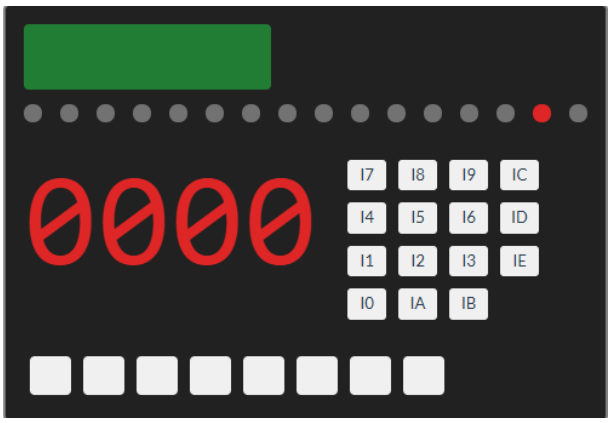
2.1.5Opções consideradas.

Não foram consideradas outras opções nesta alínea.

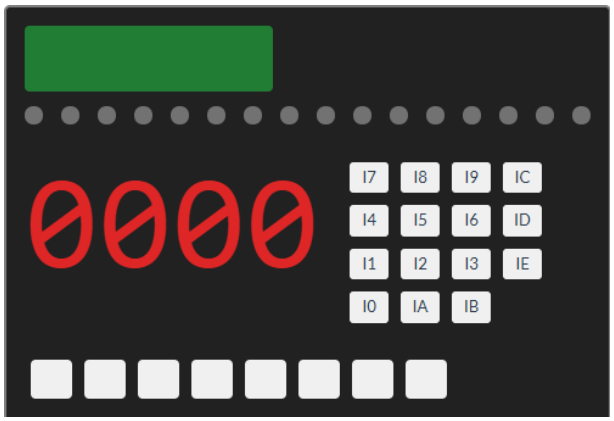
2.1.6Evidências de funcionamento

Para facilitar a recolha de evidências alterei o valor dos ciclos de DELAY de 0FFFh para 000Fh.

À entrada pela primeira vez para a rotina DELAY, a rotina IMPULSO acende pela primeira vez o segundo LED, o primeiro fica apagado, pois o movimento é para a esquerda, caso em que o valor transmitido é 0, logo o LED não acende.



O segundo LED aceso indica que foi ativado um impulso.

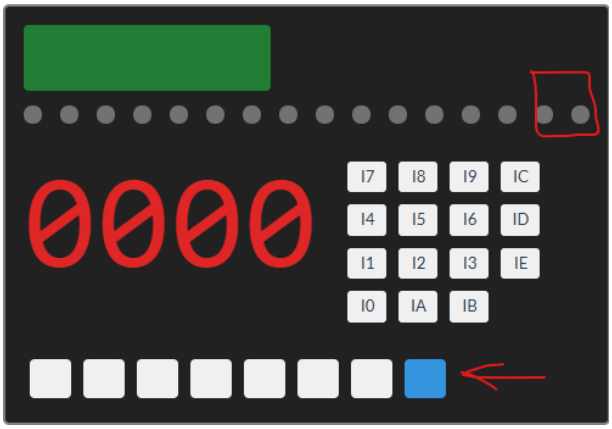


Á entrada pela segunda vez para a rotina DELAY esta, apaga o segundo LED, criando um pisca-pisca aparente do segundo LED, que simula a transmissão de impulsos para o motor até que até que o Sensor de origem seja ativado.

```
R0:  0000
R1:  0002
R2:  fdff
R3:  0000
R4:  0000
R5:  0000
R6:  0000
R7:  000f

R8:  0000
R9:  0000
R10: 0000
R11: 000f
R12: fff8
R13: 000f
SP:  fdf9
PC:  0091

CAR: 0001
SBR: 00a9
RI:  83e0
INT: 0
IAK: 0

 zcEZCNO
 0100000
```

Aparência dos registos na primeira entrada na rotina DELAY e na segunda entrada na mesma rotina.

Agora vou acionar o Sensor de Origem para simular a chegada à posição 0.




```
R0:  0000
R1:  0000
R2:  fdff
R3:  0000
R4:  0000
R5:  0000
R6:  0000
R7:  000f

R8:  000c
R9:  0002
R10: 0000
R11: 000f
R12: fff8
R13: 000f
SP:  fdf9
PC:  0091

CAR: 0001
SBR: 00a9
RI:  83e0
INT: 0
IAK: 0

 zcEZCNO
 0101100
```

Depois deste ciclo de impulsos, e tendo sido premido o primeiro botão de interruptores, o programa ao regressar à rotina POSICAO_INICIAL, chamar a rotina LE_SENSOR, a qual envia para o Registo R1 o valor 1 indicando que foi ativado o sensor de origem.

```
R0:  0000
R1:  0001
R2:  fdff
R3:  0000
R4:  0000
R5:  0000
R6:  0000
R7:  0000

R8:  0004
R9:  0002
R10: 0000
R11: 0001
R12: fff9
R13: 0001
SP:  fdfd
PC:  000f

CAR: 0001
SBR: 00bd
RI:  8240
INT: 0
IAK: 0

 zcEZCNO
 0100100
```

De seguida atualiza POSICAO_ATUAL = 0, indicando que foi atingida a posição mais à esquerda, terminando a execução do programa.

```
R0:  0000
R1:  0000
R2:  fdff
R3:  0000
R4:  0000
R5:  0000
R6:  0000
R7:  0000

R8:  0004
R9:  ffff
R10: 0020
R11: 0000
R12: 8000
R13: 003e
SP:  fdff
PC:  003e

CAR: 0001
SBR: 0103
RI:  e03f
INT: 0
IAK: 0

 zcEZCNO
 0000100
```

```
8000 : 0000 0000 0000 0000 0000 0000 0000 0000 ........
8008 : 0000 0000 0000 0000 0000 0000 0000 0000 ........
8010 : 0000 0000 0000 0000 0000 0000 0000 0000 ........
8018 : 0000 0000 0000 0000 0000 0000 0000 0000 ........
8020 : 0000 0000 0000 0000 0000 0000 0000 0000 ........
8028 : 0000 0000 0000 0000 0000 0000 0000 0000 ........
8030 : 0000 0000 0000 0000 0000 0000 0000 0000 ........
8038 : 0000 0000 0000 0000 0000 0000 0000 0000 ........
8040 : 0000 0000 0000 0000 0000 0000 0000 0000 ........
8048 : 0000 0000 0000 0000 0000 0000 0000 0000 ........
8050 : 0000 0000 0000 0000 0000 0000 0000 0000 ........
8058 : 0000 0000 0000 0000 0000 0000 0000 0000 ........
8060 : 0000 0000 0000 0000 0000 0000 0000 0000 ........
8068 : 0000 0000 0000 0000 0000 0000 0000 0000 ........
```

```
fd90 : 0000 0000 0000 0000 0000 0000 0000 0000 ........
fd98 : 0000 0000 0000 0000 0000 0000 0000 0000 ........
fda0 : 0000 0000 0000 0000 0000 0000 0000 0000 ........
fda8 : 0000 0000 0000 0000 0000 0000 0000 0000 ........
fdb0 : 0000 0000 0000 0000 0000 0000 0000 0000 ........
fdb8 : 0000 0000 0000 0000 0000 0000 0000 0000 ........
fdc0 : 0000 0000 0000 0000 0000 0000 0000 0000 ........
fdc8 : 0000 0000 0000 0000 0000 0000 0000 0000 ........
fdd0 : 0000 0000 0000 0000 0000 0000 0000 0000 ........
fdd8 : 0000 0000 0000 0000 0000 0000 0000 0000 ........
fde0 : 0000 0000 0000 0000 0000 0000 0000 0000 ........
fde8 : 0000 0000 0000 0000 0000 0000 0000 0000 ........
fdf0 : 0000 0000 0000 0000 0000 0000 0000 0000 ........
fdf8 : 0000 0000 0000 0032 0000 000f 0000 0005 ...2....
```

Edit Main Memory Zone | Edit Stack Memory Zone | Edit Disassemble Zone

2.2 Alínea b:

É pedida uma rotina, ou rotinas, que leia os valores guardados nas variáveis POSICAO_ATUAL e POSICAO_DESTINO e verifique se estas se encontram dentro dos limites do eixo (0 a 100 mm) e devolva em R3 1, se o movimento for possível ou 0 se o movimento for impossível.

2.2.1 Rotinas criadas

Para esta alínea foram criadas as rotinas LEITURA_POSICOES, VERIFICA_LIMITES, MOVIMENTO_IMPOSSIVEL e MOVIMENTO_POSSIVEL

2.2.2 Tarefas que executam

A rotina LEITURA_POSICOES lê os valores das variáveis POSICAO_ATUAL e POSICAO_DESTINO, chamando de seguida a rotina VERIFICA_LIMITES.

A rotina VERIFICA_LIMITES avalia se os valores da POSICAO_ATUAL e POSICAO_DESTINO estão dentro dos limites (0 a 100 mm), comparando os seus valores com as contantes LIMITE_MIN e LIMITE_MAX.

Se as variáveis POSICAO_ATUAL e POSICAO_DESTINO estiverem dentro dos limites, é devolvido em R3 o valor 1, indicando que o movimento é possível, finalizando a execução de seguida, na rotina MOVIMENTO_POSSIVEL.

Caso contrário, a rotina MOVIMENTO_IMPOSSIVEL devolve 0 em R3.

A verificação da possibilidade ou impossibilidade de realizar o percurso entre a POSICAO_ATUAL e POSICAO_DESTINO é feita do seguinte modo:

1. Verifica-se se POSICAO_ATUAL < 0 (CMP R1, LIMITE_MIN e salto BR.N), se negativo, o movimento é impossível;
2. Verifica-se se POSICAO_ATUAL > 100 (CMP R1, LIMITE_MAX e salto BR.P), se for positivo, o movimento é impossível;
3. Verifica-se se POSICAO_DESTINO < 0 (CMP R2, LIMITE_MIN e salto BR.N), se negativo, o movimento é impossível;
4. Verifica-se se POSICAO_DESTINO > 100 (CMP R2, LIMITE_MAX e salto BR.P), se for positivo, o movimento é impossível.

Passando estes testes sem que o movimento possa ser considerado impossível, então o movimento é possível, ou seja, 0 < POSICAO_ATUAL < 100 **e** 0 < POSICAO_DESTINO < 100 e é tratado pela rotina MOVIMENTO_POSSIVEL.

2.2.3 Entradas

Leitura dos valores das variáveis POSICAO_ATUAL e POSICAO_DESTINO.

2.2.4 Saídas

Se o movimento for possível a rotina MOVIMENTO_POSSIVEL devolve 1 no registo R3, caso contrário a rotina MOVIMENTO_IMPOSSIVEL devolve 0 no registo R3.

2.2.5 Opções consideradas.

Considerei a hipótese de aferir a possibilidade do movimento através da diferença entre POSICAO_DESTINO e POSICAO_ATUAL, mas isso apenas indica se o movimento deve ser feito para a direita ou para a esquerda, não servindo o propósito desta alínea.

2.2.6 Evidências de funcionamento

O comportamento das rotinas desta alínea será testado na alínea d), através da verificação dos valores das entradas e saídas (R1, R2 e R3).

2.3 Alínea c:

É pedida uma rotina, ou rotinas, que leia os valores guardados nas variáveis POSICAO_ATUAL e POSICAO_DESTINO e, caso o movimento seja possível, devolva em R3 o número de impulsos necessário para o movimento e em R4 devolva o valor do sentido do movimento (0= esquerda, 1= direita).

2.3.1 Rotinas criadas

Para esta alínea foram criadas as rotinas MOVIMENTO, ESQUERDA, DIREITA, IMPULSOS e NAO_MEXE.

Foram reaproveitadas as rotinas criadas na alínea b).

2.3.2 Tarefas que executam

A rotina MOVIMENTO, chama com a rotina LEITURA_POSICOES, criada na alínea b), cujo funcionamento está explicado.

Recebe, então, da rotina chamada pela referida rotina (VERIFICA_LIMITES) o valor 0 (movimento impossível) ou valor 1 (movimento possível),

Se o movimento for impossível não faz nada e termina a execução do programa.

Caso contrário verifica se a POSICAO_ATUAL (em R1) é superior, inferior ou igual à POSICAO_DESTINO (em R2).

Se as referidas posições forem iguais, não faz nada e devolve o número de impulsos=0 (em R3) e o sentido, esquerda ou direita a 0 (em R4) e termina a execução programa.

Se POSICAO_ATUAL < POSICAO_DESTINO, o movimento será para a direita e é chamada a rotina DIREITA. Esta rotina devolve R4=1 (movimento para a direita), calcula a distância entre as duas posições e segue para o cálculo no número de impulsos.

Caso POSICAO_ATUAL > POSICAO_DESTINO, o movimento será para a esquerda e é chamada a rotina ESQUERDA. Esta rotina devolve R4=0 (movimento para a esquerda), calcula a distância entre as duas posições e chama a rotina IMPULSOS.

A rotina IMPULSOS recebe a distância entre as duas posições e multiplica essa distância por 20 para calcular o número de impulsos necessários.

A razão por que se multiplica por 20 é que é explicado no enunciado que são necessários 2000 impulsos para percorrer 100 mm, isto quer dizer que para cada 10 mm são necessários 200 impulsos e que para percorrer 1 mm, são necessários 20 impulsos.

Em vez de usar a instrução de multiplicação (MUL) ou de usar as instruções de multiplicação e divisão (MUL e DIV) que são de tratamento mais complexo, decidi usar a instrução de deslocamento SHLA, fazendo a multiplicação em 4 fases.

A primeira fase é criar duas cópias do valor da distância, para serem trabalhadas em separado.

A segunda fase faz um deslocamento de 4 bits para a esquerda, o que equivale a multiplicar a distância por 16.

A terceira fase faz um deslocamento de 2 bits para a esquerda, o que equivale a multiplicar a distância por 4.

A última fase é adicionar a distância multiplicada por 16 com a distância multiplicada por 4, seja 16x+4x=20x.

Por fim é escrito no registo R3 o número impulsos e termina a execução do programa.

2.3.3 Entradas

Estas rotinas recebem em R1 o valor da POSICAO_ATUAL e em R2 o valor da POSICAO_DESTINO.

Em R3, recebem 0 (se o movimento for impossível) ou 1 (se o movimento for possível).

2.3.4Saídas

Estas rotinas devolvem em R3 o número de impulsos necessários para realizar o movimento e em R4 o valor do sentido de deslocamento (0=esquerda ou 1=direita).

2.3.5Opções consideradas.

Tal como foi referido anteriormente, considerei a hipótese de usar a instrução de multiplicação (MUL) ou de usar as instruções de multiplicação e divisão (MUL e DIV), mas considerei estas de tratamento mais complexo

2.3.6Evidências de funcionamento

O comportamento das rotinas desta alínea será testado na alínea d, através da verificação dos valores das entradas e saídas (R1, R2, R3, R4 e endereços de memória 8000h a 8007h).

2.4 Alínea d:

É pedida uma rotina, ou rotinas, que inicie o sistema do eixo (feito na alínea a) mova o eixo para diversas posições absolutas (30 mm, 10 mm, 50 mm) devendo antes de cada movimento usar a rotina fornecida SET_DIRECAO.

Com o objetivo de testar as funcionalidades das alíneas anteriores foram incluídas, além das posições absolutas, referidas, a posição 115mm (fora dos limites do eixo) e a posição 50mm que deve testar, após o pedido impossível de realizar, o caso em que POSICAO_ATUAL é igual à POSICAO_DESTINO.

Para tratar as diferentes posições foi criada a constante DESTINOS (explicada no ponto 1.5 – constantes).

2.4.1Rotinas criadas

Para esta alínea foram criadas as rotinas PROCESSA_DESTINOS, SEQUENCIA_DESTINOS, PROCESSA_MOV, EXECUTA_MOV, DEC_IMPULSOS, e ATUALIZA_POS_ATUAL

2.4.2Tarefas que executam

A rotina PROCESSA_DESTINOS, apenas coloca o índice de destino a 0, num registo (R7) que fará o papel de variável iteradora, seguindo de seguida para a rotina SEQUENCIA_DESTINOS.

As rotinas PROCESSA_DESTINOS e PROCESSA_MOVIMENTO reutilizam as rotinas das alíneas b) e c) LEITURA_POSICOES, VERIFICA_LIMITES e MOVIMENTO.

A rotina SEQUENCIA_DESTINOS, depois de verificar se já foram tratados todos os destinos (caso em se termina a execução do programa), passa ao próximo destino a ser processado, chamando a rotina MOVIMENTO, que realiza as tarefas explicadas na alínea c). De seguida é chamada a rotina PROCESSA_MOV.

A rotina PROCESSA_MOV recebe as informações relativas ao número de impulsos (R3) e sentido da rotação (R4), chamando de seguida a rotina SET_DIRECAO, informando o sentido de rotação (R4).

De seguida entra-se na rotina EXECUTA_MOV verifica se já foram dados todos os impulsos, caso em que chama a rotina ATUALIZA_POS_ATUAL, caso contrário chama a rotina fornecida IMPULSO vai gerando impulsos na direção indicada por _SENTIDO_MOVIMENTO.

Após cada impulso, a rotina DEC_IMPULSOS decrementa o número de impulsos que faltam e envia de volta para a rotina EXECUTA_MOV, até que já não hajam impulsos para dar (R3=0).

Quando já não houver impulsos a realizar (R3=0), a rotina ATUALIZA_POS_ATUAL atualiza o valor da POSICAO_ATUAL com o valor da POSICAO_DESTINO, porque foi atingida a posição que se pretendia atingir.

A seguir termina a execução do programa (caso já tenha percorrido todos os destinos), ou passa ao próximo destino para processamento do movimento.

2.4.3 Entradas

As diversas rotinas desta alínea, recebem as seguintes informações:

1. Destino a tratar da conjugação da constante DESTINOS com o valor da variável iteradora (R7);
2. Número de impulsos a realizar, em R3;
3. Sentido da rotação em R4.

2.4.4 Saídas

As diversas rotinas desta alínea, enviam as seguintes informações:

1. Destino a tratar (em R2 e POSICAO_DESTINO);
2. Sentido da rotação;
3. Atualiza POSICAO_ATUAL.

2.4.5 Opções consideradas.

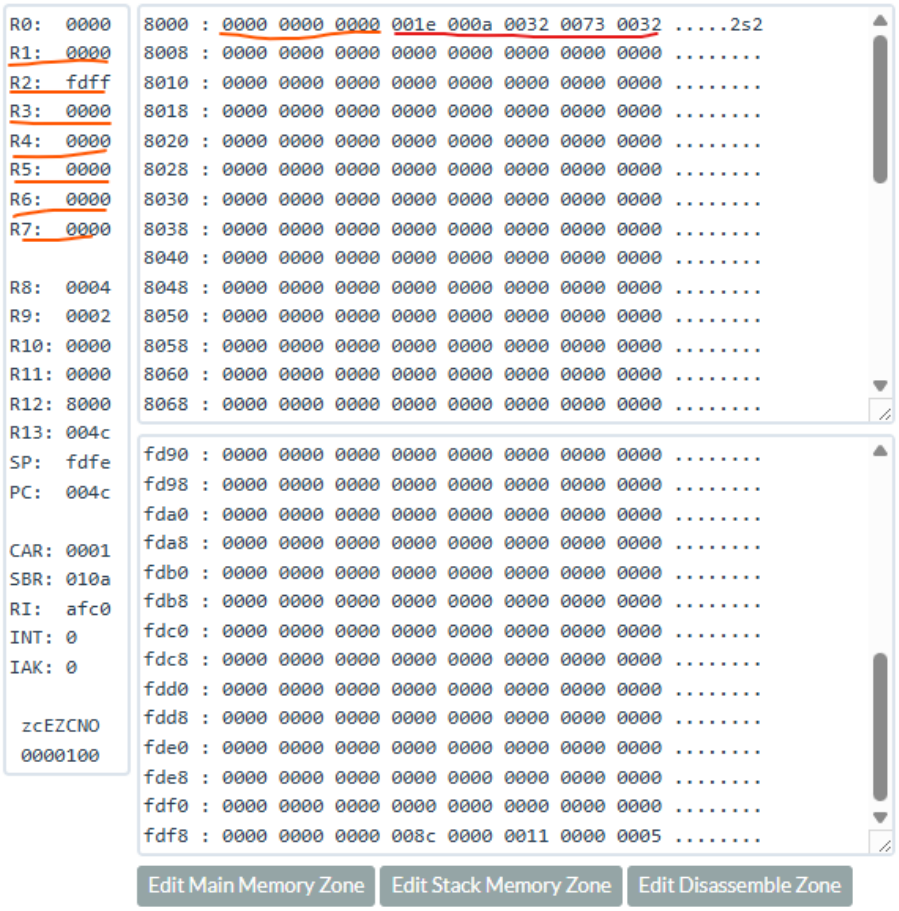
Foi estudada e considerada outra forma de tratar o vetor DESTINOS, como usar a diretiva TAB, mas essa hipótese mostrou-se desadequada ao pretendido porque não permite inicializar valores, apenas reservar posições de memória inicializadas a zero.

2.4.6 Evidências de funcionamento

Entrada na rotina PROCESSA_DESTINOS, após a calibração do sistema, onde é possível verificar os seguintes valores:

Registos:

1. R1 = R3 = R5 = R6 = R7 = 0 valores iniciais do sistema inicializado corretamente e;
2. R2 = FDFFh, valor da posição inicial do ponteiro da pilha.



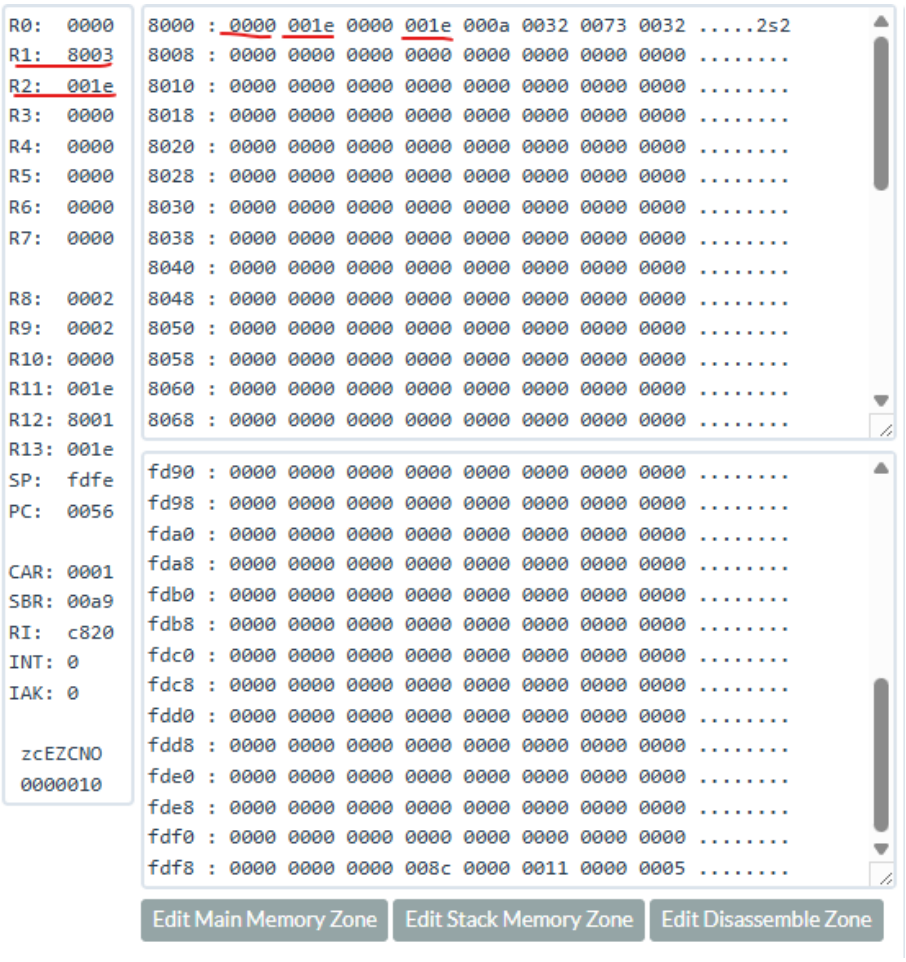
Memória:

1. 8000h (POSICAO_ATUAL) = 8001h (POSICAO_DESTINO) = 0, posições iniciais do sistema inicializado corretamente;
2. 8002h (_SENTIDO_MOVIMENTO) = 0, indicando que foi realizado um movimento para a esquerda;
3. 8003h (DESTINO[0]) = 001Eh (corresponde a 30 em notação decimal);
4. 8004h (DESTINO[1] = 000Ah (corresponde a 10 em notação decimal);
5. 8005h (DESTINO[2] = 0032h (corresponde a 50 em notação decimal);

6. 8006h (DESTINO[3] = 0073h (corresponde a 115 em notação decimal);
7. 8007h (DESTINO[4] = 0032h (corresponde a 50 em notação decimal);

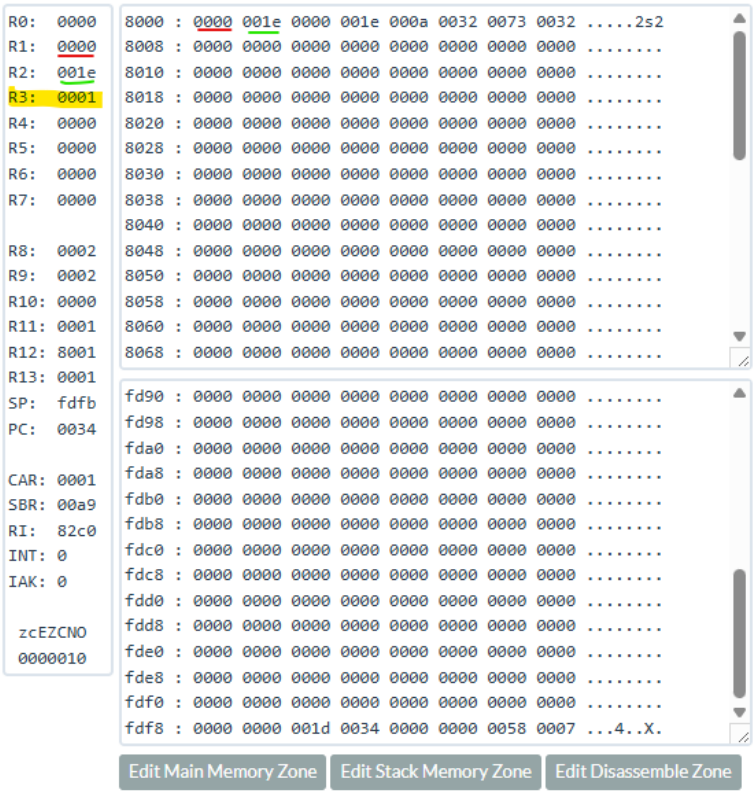
Depois de entrar na rotina SEQUENCIAS_DESTINOS:

recebeu em R1 o endereço de memória onde se encontra o primeiro destino (8003h (DESTINO[0])) e em R2 o valor do próximo destino 001Eh (corresponde a 30 em notação decimal), atualizando o valor de POSICAO_DESTINO (8001h) com o valor do próximo9 destino 001Eh.



Chamará depois a rotina MOVIMENTO que encaminhará para a rotina LEITURA_POSICOES que lerá os valores das posições atual e de destino e decidirá se pode ou não realizar o movimento.

A rotina MOVIMENTO já recebeu a indicação de que o movimento é possível (R3=1) e vai agora verificar para que sentido deve ser feito o movimento.

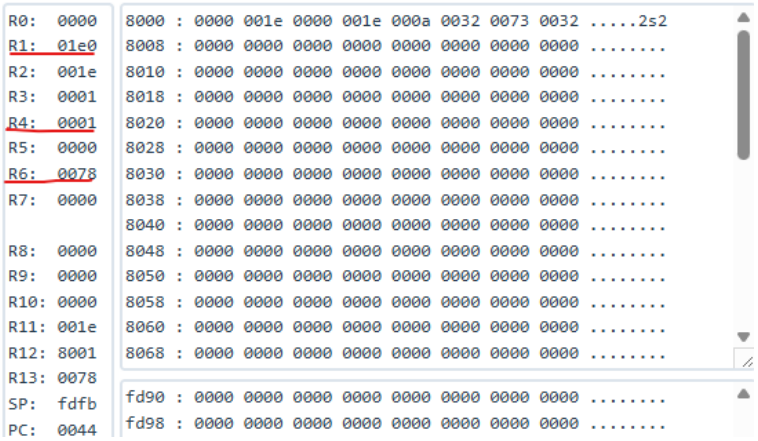


Mantém a informação da POSICAO:ATUAL (0) em R1 e a POSICAO_DESTINO (001Eh – 30d) em R2.

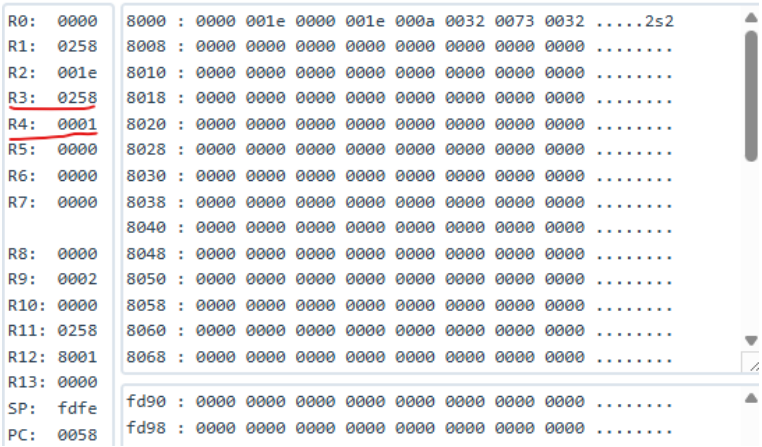
Além disso, com estes valores, comprovámos o bom funcionamento das rotinas da alínea b).

Agora irá verificar se as duas posições são iguais, caso em que não faz nada, ou se realiza um movimento para direita ou para a esquerda.

A rotina MOVIMENTO verificou que o movimento é possível: escreveu R4 = 1 e passou ao cálculo dos deslocamentos de 4 e 2 bits. Podemos verificar que R1 contém o valor 01E0h (480d = 30*16) e R6 contém o valor 0078h (120d = 30*4).

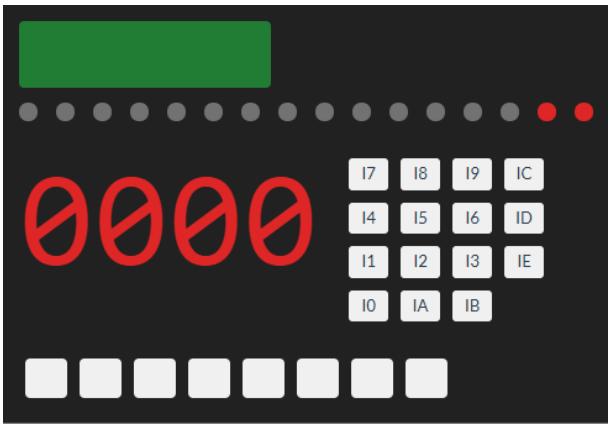


Após a saída da rotina MOVIMENTO, é possível comprovar o bom funcionamento das rotinas da alínea c) que devolvem corretamente em R3, número de impulsos = 258h (ou seja 600d) e em R4, sentido direita (R4 = 1).

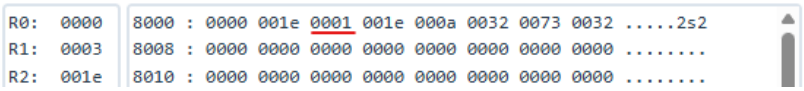


Entrará agora nas rotinas da alínea d) para processar e executar o movimento 600 impulsos para a direita.

Enquanto são gerados impulsos para atingir a posição de destino é possível verificar que o último LED assinala corretamente um movimento para a direita e o penúltimo LED vai piscando com os impulsos.

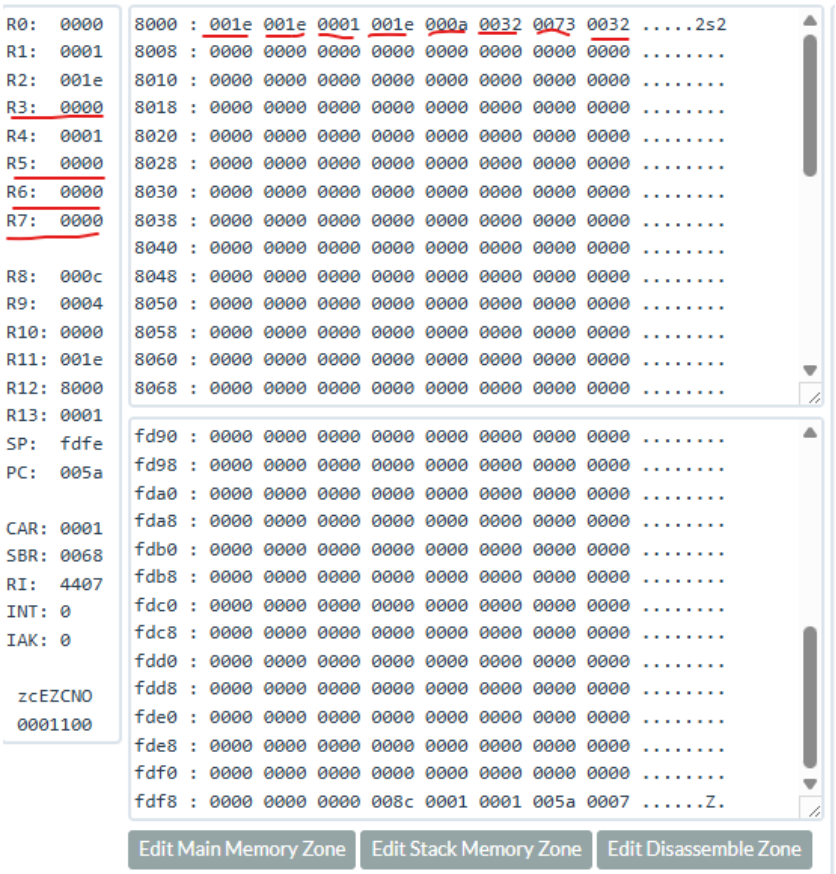


Também é possível verificar na posição de memória 8002h (_SENTIDO_MOVIMENTO) o valor 1, correspondente ao sentido para a direita

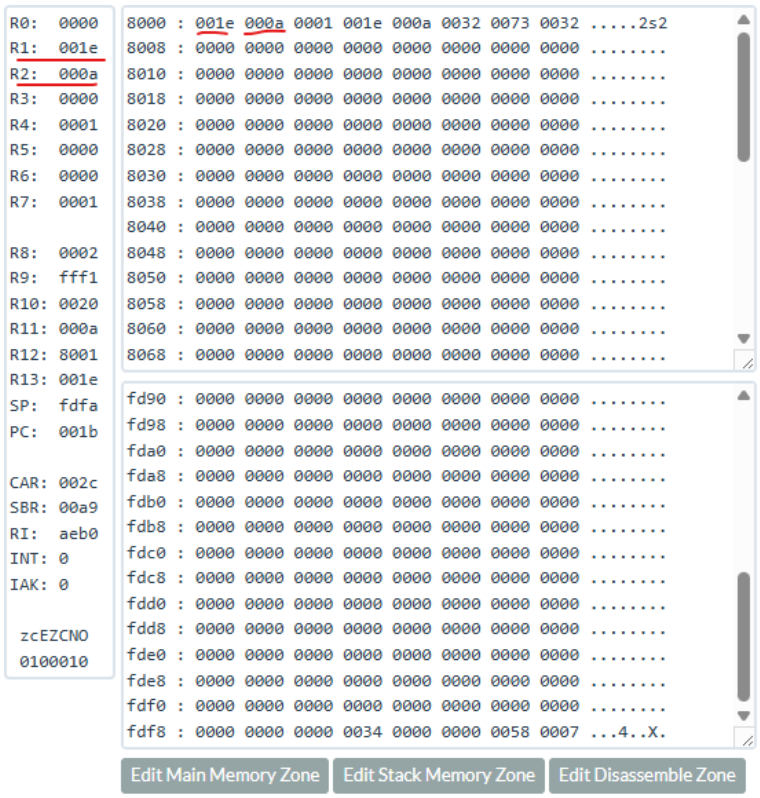


Após processar e executar o movimento para o primeiro destino (001Eh - 30 em decimal) o valor da variável POSICAO_ATUAL foi atualizado (valor 001Eh na posição de memória 8000h).

Irá agora passar para o destino seguinte.



A rotina SEQUENCIA_DESTINOS, já passou corretamente ao próximo destino e é possível ver na memória em POSICAO_ATUAL (8000h) o valor 001Eh, indicando que o eixo se encontra na posição 30, bem na POSICAO_DESTINO (8001h) o valor 000Ah, indicando que o próximo é a posição 10.



Vão agora seguir-se os passos anteriores:

1. Ler as posições atuais e de destino;

2. Verificar se o movimento possível;

3. Caso seja possível, em que direção (esquerda ou direita);

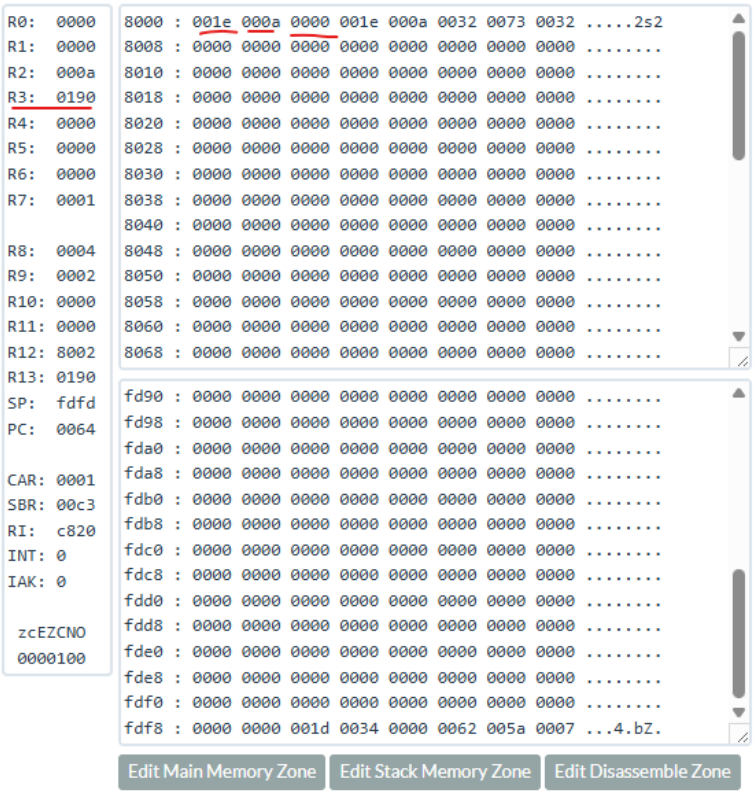
4. Calcular o número de impulsos necessários;

5. Atualiza _SENTIDO_MOVIMENTO;

6. Fazer a sucessão de impulsos necessários para ir da posição 30 à posição 10 (400 impulsos para a esquerda);

7. Atualizar a posição atual para 10 (000Ah).

Após atualizar o sentido do movimento, possível verifica em R3 o número de impulsos necessários (0190h – ou seja (30-10) x 20 = 400 impulsos) POSICAO_ATUAL = 001Eh (30d), POSICAO_DESTINO = 000Ah (10d) e sentido esquerda (0 em _SENTIDO_MOVIMENTO – 8002h).



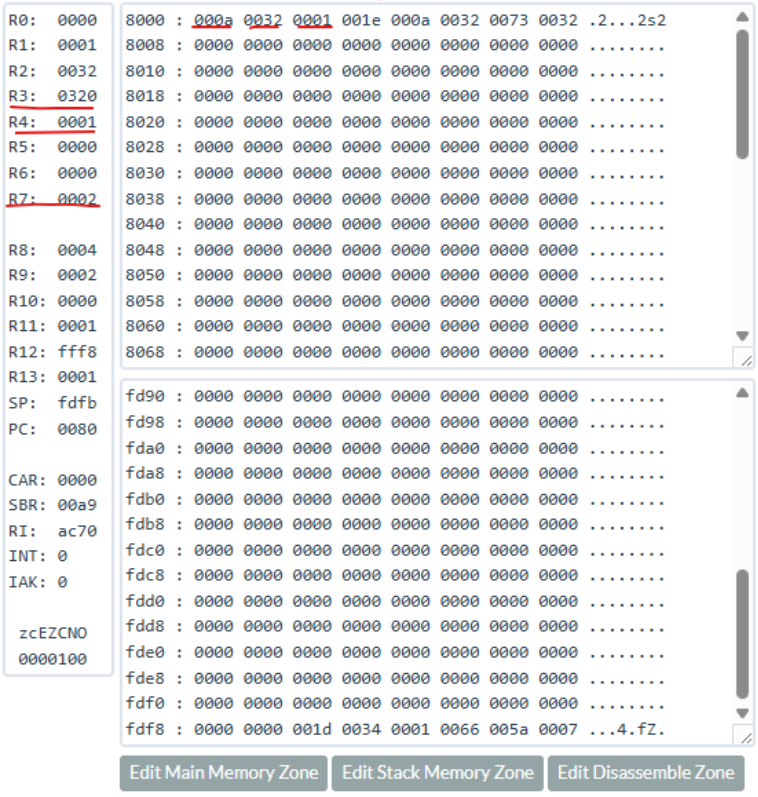
Após selecionar o novo destino, é possível ver que foram calculados com sucesso:

POSICAO_ATUAL = 000Ah (10d) – na posição de memória 800h;

POSICAO DESTINO = 0032h (50d) – na posição de memória 8001h;

_SENTIDO_MOVIMENTO para a direita – na posição de memória 8002h, porque 50 > 10;

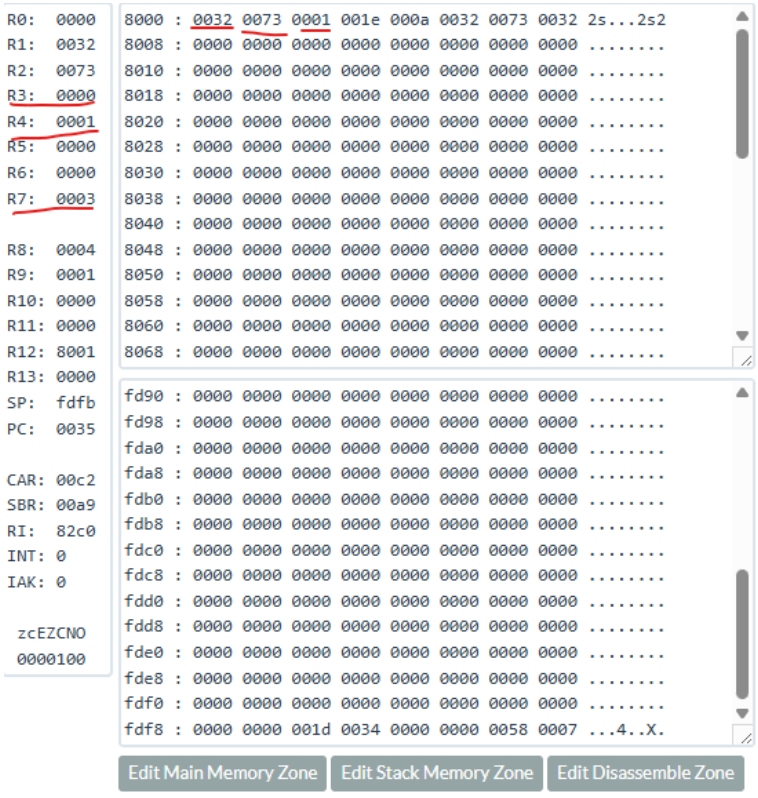
R3 = 0320h (800 d = (50-10) x 20);



R4 = 1 (sentido de rotação para a direita):

R7 = 2, significando que iremos agora executar o movimento para DESTINO[2]. gravado na posição de memória 8005h.

Iremos de seguida verificar o que acontece ao tentarmos processar um destino fora dos limites

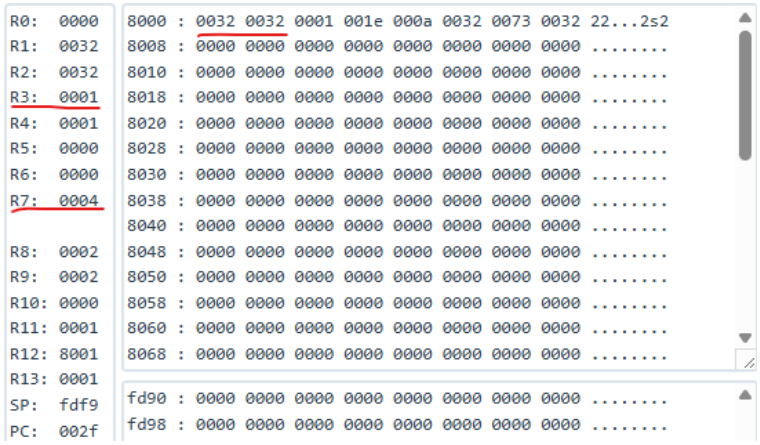


Após passar ao destino seguinte, na posição de memória 8001h, é possível ver o valor 0073h (115 d) e em R7 está o valor 3, significando que iremos tratar o DESTINO[3].

A rotina MOVIMENTO já informou que o movimento é impossível (R3 = 0), também informou que o movimento seria para a direita (R4 = _SENTIDOMOVIMENTO (8002h) = 1).

É ainda possível verificar que POSICAO_ATUAL tem o valor 0032h (50d) última posição legal encontrada.

De seguida passará ao processamento do próximo destino, no qual testaremos o que acontece quando a posição atual é igual à posição de destino.

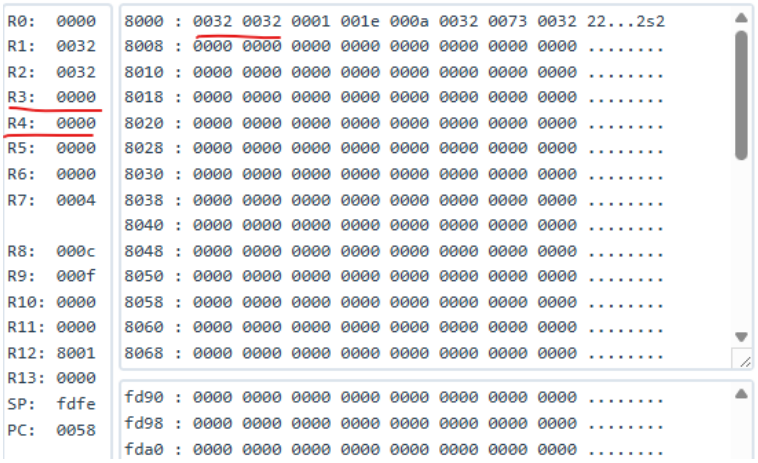


Após selecionar o último destino, podemos verificar que POSICAO_DESTINO apresenta o valor 0032h (50d) o registo R7 = 4, significando que iremos tratar o DESTINO[4] (último destino, posição de memória 8007h) e que o movimento foi classificado como possível, por se encontrar dentro dos limites (R3=1).

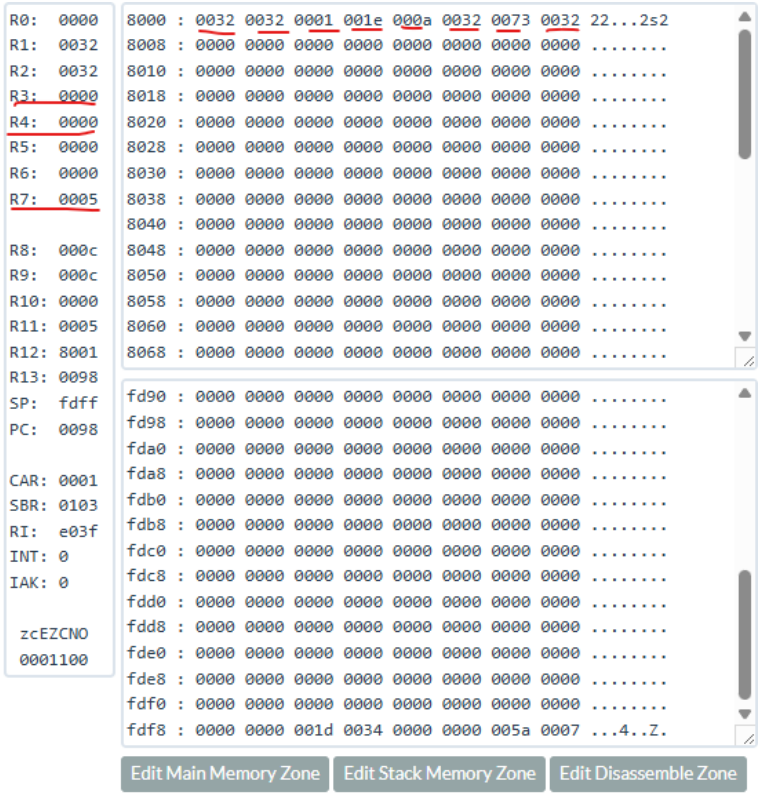
Vai agora entrar na rotina PROCESSA_MOV e verificar que o movimento não pode ser realizado porque POSICAO_ATUAL = POSICAO_DESTINO, terminando o processamento do movimento e devolvendo 0 em R3 (0 impulsos) e R4 = 0

A rotina PROCESSA_MOV verificará que não há nada a fazer e tentará passar ao próximo destino.

Como este é último destino terminará a execução do programa.

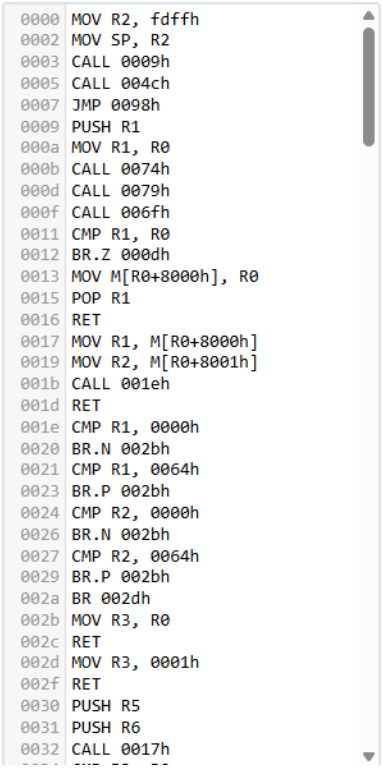


Confirma-se que o programa termina com os valores esperados em R3, R4, R7 e nas posições de memória 8000h a 8007h.

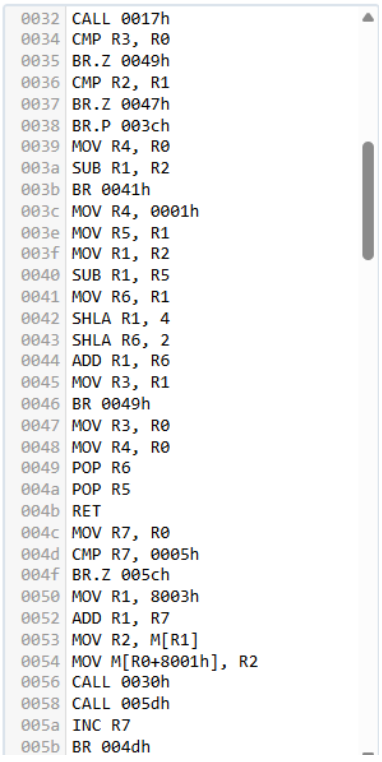


Programa, escrito na Disassemble Zone do Simulador P3, no site https://p3js.goncalomb.com/,

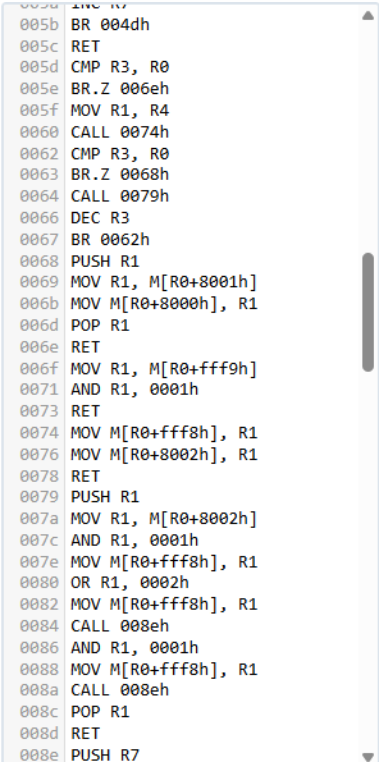
https://p3js.goncalomb.com/



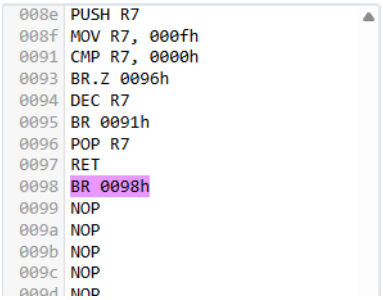
```
0000 MOV R2, fdffh
0002 MOV SP, R2
0003 CALL 0009h
0005 CALL 004ch
0007 JMP 0098h
0009 PUSH R1
000a MOV R1, R0
000b CALL 0074h
000d CALL 0079h
000f CALL 006fh
0011 CMP R1, R0
0012 BR.Z 000dh
0013 MOV M[R0+8000h], R0
0015 POP R1
0016 RET
0017 MOV R1, M[R0+8000h]
0019 MOV R2, M[R0+8001h]
001b CALL 001eh
001d RET
001e CMP R1, 0000h
0020 BR.N 002bh
0021 CMP R1, 0064h
0023 BR.P 002bh
0024 CMP R2, 0000h
0026 BR.N 002bh
0027 CMP R2, 0064h
0029 BR.P 002bh
002a BR 002dh
002b MOV R3, R0
002c RET
002d MOV R3, 0001h
002f RET
0030 PUSH R5
0031 PUSH R6
0032 CALL 0017h
```



```
0032 CALL 0017h
0034 CMP R3, R0
0035 BR.Z 0049h
0036 CMP R2, R1
0037 BR.Z 0047h
0038 BR.P 003ch
0039 MOV R4, R0
003a SUB R1, R2
003b BR 0041h
003c MOV R4, 0001h
003e MOV R5, R1
003f MOV R1, R2
0040 SUB R1, R5
0041 MOV R6, R1
0042 SHLA R1, 4
0043 SHLA R6, 2
0044 ADD R1, R6
0045 MOV R3, R1
0046 BR 0049h
0047 MOV R3, R0
0048 MOV R4, R0
0049 POP R6
004a POP R5
004b RET
004c MOV R7, R0
004d CMP R7, 0005h
004f BR.Z 005ch
0050 MOV R1, 8003h
0052 ADD R1, R7
0053 MOV R2, M[R1]
0054 MOV M[R0+8001h], R2
0056 CALL 0030h
0058 CALL 005dh
005a INC R7
005b BR 004dh
```



```
005b BR 004dh
005c RET
005d CMP R3, R0
005e BR.Z 006eh
005f MOV R1, R4
0060 CALL 0074h
0062 CMP R3, R0
0063 BR.Z 0068h
0064 CALL 0079h
0066 DEC R3
0067 BR 0062h
0068 PUSH R1
0069 MOV R1, M[R0+8001h]
006b MOV M[R0+8000h], R1
006d POP R1
006e RET
006f MOV R1, M[R0+fff9h]
0071 AND R1, 0001h
0073 RET
0074 MOV M[R0+fff8h], R1
0076 MOV M[R0+8002h], R1
0078 RET
0079 PUSH R1
007a MOV R1, M[R0+8002h]
007c AND R1, 0001h
007e MOV M[R0+fff8h], R1
0080 OR R1, 0002h
0082 MOV M[R0+fff8h], R1
0084 CALL 008eh
0086 AND R1, 0001h
0088 MOV M[R0+fff8h], R1
008a CALL 008eh
008c POP R1
008d RET
008e PUSH R7
```



```
008e PUSH R7
008f MOV R7, 000fh
0091 CMP R7, 0000h
0093 BR.Z 0096h
0094 DEC R7
0095 BR 0091h
0096 POP R7
0097 RET
0098 BR 0098h
0099 NOP
009a NOP
009b NOP
009c NOP
```

Lista de referências e etiquetas do programa

Labels / References 37

POSICAO_ATUAL	8000
POSICAO_DESTINO	8001
_SENTIDO_MOVIMENTO	8002
IO_DIRECAO	fff8
IO_SENSOR	fff9
SP_INICIAL	fdff
LIMITE_MIN	0000
LIMITE_MAX	0064
DESTINOS	8003
MAIN	0000
INICIAR_SISTEMA	0009
POSICAO_INICIAL	000d
LEITURA_POSICOES	0017
VERIFICA_LIMITES	001e
MOVIMENTO_IMPOSSIVEL	002b
MOVIMENTO_POSSIVEL	002d
MOVIMENTO	0030
ESQUERDA	0039
DIREITA	003c
IMPULSOS	0041
NAO_MEXE	0047
FIM_MOVIMENTO	0049
PROCESSA_DESTINOS	004c
SEQUENCIA_DESTINOS	004d
FIM_SEQ_DESTINOS	005c
PROCESSA_MOV	005d
EXECUTA_MOV	0062
DEC_IMPULSOS	0066
ATUALIZA_POS_ATUAL	0068
FIM_PROC_MOV	006e
LE_SENSOR	006f
SET_DIRECAO	0074
IMPULSO	0079
DELAY	008e
CICLO_DELAY	0091
FIM_DELAY	0096
FIM	0098