

Trabalho Prático 2

**Computação Física**

Curso de Licenciatura Informática e Multimédia (LEIM)

Ano Letivo 2017/2018

**Data: 15/05/2018**

**Turma: LEIM 23D**

**Docente:**

Eng. Jorge Pais

**Grupo:** 6 (?)

**Alunos:**

Luis Fonseca (A45125)

Gabriel Diaz(A45133)

Philipp Al-Badavi(A45138)

**Índice**

1.Introdução………………………………………………………………...3

2.Registos da Memória de Dados e Código………………………………..4

3.Especificação das Instruções……………………………………………..5

4.Codificação das Instruções……….………………………………………6

5.Módulo Funcional………………………………………………………..7

6.Tabela do Módulo Funcional…………………………………………….8

7. EPROM 64x14…………………………………………………………..9

8.Codigo Arduíno…………………………………………………………11

9.Conclusão……………………………………………………………….12

10.Bibliografia…………………………………………………………….13

**Introdução**

Com a tabela de instruções que foi proposta, coube ao nosso grupo realizar um CPU, através da construção de um módulo funcional, um modulo de controlo e uma tabela EPROM no qual, depois de feitas, implementar no Arduíno para obter as instruções que eram pedidas.

Ao longo do relatório vão ser apresentados os passos que foram realizados, para a solução encontrada ao problema.

**Especificação dos registos de uso geral e dos barramentos de endereço e dados para as memórias**

Olhando para a tabela com as instruções dadas podemos retirar algumas informações, que nos vai ser útil para a realização do nosso módulo funcional, assim como para a tabela do código de instruções.

Rn = 6 bits

A = 8 bits

C = Flag Carry (1 bit)

Z = Flag Zero (1 bit)

rel5 = relativo a 5 bits

PC = registo de 6 bit

Na Memória de Dados:

DB = 8bits e o AB = 6bits

Na Memória de Código:

DB = 8bits e o AB = 6bits

**Especificação de instruções**

Nas primeiras instruções temos os dois primeiros registos, V e R no qual o registo V vai ter 8 bits já para o registo R vai ter 6 bits. Logo na terceira instrução podemos verificar que o registo A vai ter o mesmo número de bits que o registo V que neste caso também vai ser 8 bits. Logo de seguida temos as operações aritméticas, no qual os registos que aqui vão ser usados são o A e o V. Chegando últimas instruções temos os “jumps” no qual, JC e JZ correspondem respetivamente ao “jump carry” e ao “jump zero”, ao chegar a essa funcionalidade o PC (program counter) faz uma incrementação do valor relativo a 5 bits, tanto para o “carry” como para o zero, e o programa acaba quando o PC chega aos 6bits.

|  |  |
| --- | --- |
| Instrução | Funcionalidade |
| Mov V, #const8 | V = const8 |
| Mov R, #const6 | R = const6 |
| Mov A, V | A=V |
| Mov V, @R | V = M(R) |
| Mov @R, V | M(R) = V |
| NAND V, A | V = (V.A)\ |
| ADD V, A | V = V + A |
| SUBB V, A | V = V - A |
| DIV V, A | V = V / A ; A = V % A |
| JC rel5 | Se(Cy) PC += rel5 |
| JZ rel5 | Se(Z) PC += rel5 |
| JMP end6 | PC = end6 |

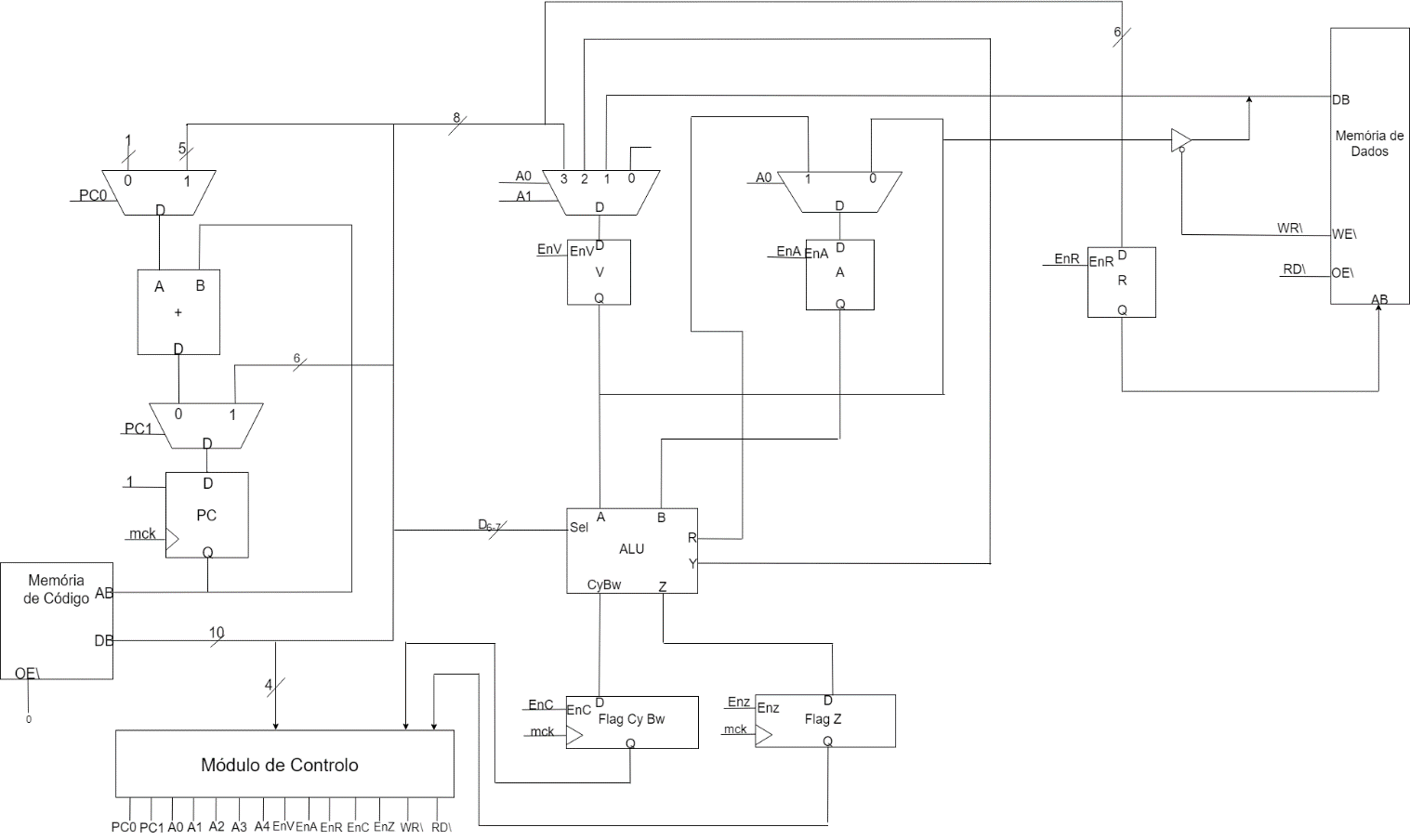
**Codificação de instruções**

Com a tabela da instrução e funcionalidade fornecida, foi feita uma tabela com uma codificação a 10bits, no qual os bits D9, D8, D7 e D6 vão ser usados para distinguir as instruções.

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Codificação a 10bits | | | | | | | | | | | |
|  | Parâmetro | D9 | D8 | D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |
| MOV V,#const8 | Const8 | 0 | 1 | C7 | C6 | C5 | C4 | C3 | C2 | C1 | C0 |
| MOV R,#const6 | Const6 | 1 | 0 | 0 | 0 | C5 | C4 | C3 | C2 | C1 | C0 |
| MOV A,V | - | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| MOV V,@R | - | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Mov @R,V | - | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| NAND V,A | - | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| ADD V,A | - | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| SUBB V,A | - | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| DIV V,A | - | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| JC rel5 | Se(Cy) += rel5 | 0 | 0 | 1 | 0 | 0 | R4 | R3 | R2 | R1 | R0 |
| JZ rel5 | Se(Z) += rel5 | 0 | 0 | 0 | 1 | 0 | R4 | R3 | R2 | R1 | R0 |
| JMC end6 | PC = end6 | 0 | 0 | 0 | 0 | E5 | E4 | E3 | E2 | E1 | E0 |

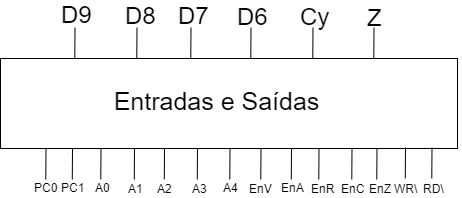
**Módulo Funcional**

Depois da codificação das instruções feita, passou se então a construção do módulo funcional, recorrendo a multiplexers do tipo 2x1 e 4x2, as saídas desses mux vão estar nos registos V e A. As saídas desses mesmo vão para a ALU no qual, vão ser realizadas as operações aritméticas. Umas das saídas que estão presentes na ALU são o Cy e o Z, no qual servem também para realizar essas operações, e estes vão ser do tipo JK Edge-Triggered, sendo que têm 1 bit de registo.



**Módulo de Controlo e tabela com Sinais Ativos**

Feito o módulo funcional, passou-se á construção do módulo de controlo, no qual vai ter como entrada, os bits D9, D8, D7, D6, o Cy e o Z. Já para as saídas temos os dois Program Counter, usados como entrada de seleção do mux 2x1, os bits de saída A0, A1, A2, A3 e A4, os Enables dos registos V, A, R e do Cy e do Z, e o WR\(write) no qual quando ativa (zero lógico) indica que o dispositivo está a ser lido pelo CPU e o RD(read) que quando ativa (zero lógico) indica que o dispositivo está a ser gravado pelo CPU. Com isto foi feita uma tabela com estes bits de entrada, olhando para o módulo funcional e seguindo as instruções, foi feita uma análise dos sinais ativos que iam ser usados na implementação do CPU no Arduíno.



* **Tabela com os sinais de entrada do modulo de controlo e os sinais ativos.**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Instrução | D9 | D8 | D7 | D6 | Cy | Z | Sinais Activos |
| MOV V,#const8 | 0 | 1 | - | - | - | - | EnV,A1,A2 |
|  |  |  |  |  |  |  |  |
| MOV R,#const6 | 1 | 0 | 0 | 0 | - | - | EnR |
| MOV A,V | 1 | 0 | 0 | 1 | - | - | EnA |
| MOV V,@R | 1 | 0 | 1 | 0 | - | - | EnV,RD\,A1 |
| MOV @R,V | 1 | 0 | 1 | 1 | - | - | WR\ |
| NAND V,A | 1 | 1 | 0 | 0 | - | - | EnV |
| ADD V,A | 1 | 1 | 0 | 1 | - | - | EnV,EnC,A2 |
| SUBB V,A | 1 | 1 | 1 | 0 | - | - | EnV,EnC,A2 |
| DIV V,A | 1 | 1 | 1 | 1 | - | - | EnV,EnC,A2 |
| JC rel5 | 0 | 0 | 0 | 0 | 0 | - |  |
| JC rel5 | 0 | 0 | 0 | 0 | 1 | - | PC0 |
| JZ rel5 | 0 | 0 | 0 | 1 | - | 0 |  |
| JZ rel5 | 0 | 0 | 0 | 1 | - | 1 | PC0 |
| JMC end6 | 0 | 0 | 1 | 0 | - | - | PC1 |

**Tabela EPROM 64x14**

Com a tabela feita dos sinais ativos das diferentes instruções, prossegui-se para o último passo antes da implementação do CPU no Arduíno, calcular o valor da data e do address. O cálculo do address foi feita á custa da codificação feita nos sinais de entrada, já a data foi feita através dos sinais de saída, depois de calculados os valores foram passados para hexadecimal.

Nota: Devido ao comprimento da tabela EPROM que ia ser feita, teve de ser divida em duas partes, uma para a entrada e ser feito o cálculo do address, outra para as saídas e o cálculo da data.

* **Tabela com entradas e o valor do address**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | D9 | D8 | D7 | D6 | Cy | Z | Address |
| JC | 0 | 0 | 0 | 0 | 1 | - | [2,3] |
| JZ | 0 | 0 | 0 | 1 | - | 1 | [5,7] |
| JMP | 0 | 0 | 1 | 0 | - | - | [8,9, a,b] |
|  | 0 | 0 | 1 | 1 | - | - | [c,d,e,f] |
| MOV V,#const8 | 0 | 1 | - | - | - | - | [10,1f] |
| MOV R,#const6 | 1 | 0 | 0 | 0 | - | - | [20,23] |
| MOV A,V | 1 | 0 | 0 | 1 | - | - | [24,27] |
| MOV V,@R | 1 | 0 | 1 | 0 | - | - | [28,26] |
| MOV @R,V | 1 | 0 | 1 | 1 | - | - | [2c,2f] |
| NAND V,A | 1 | 1 | 0 | 0 | - | - | [30,33] |
| ADD V,A | 1 | 1 | 0 | 1 | - | - | [34,37] |
| SUBB V,A | 1 | 1 | 1 | 0 | - | - | [38,36 |
| DIV V,A | 1 | 1 | 1 | 1 | - | - | [3c,3f] |

* **Tabela com as saídas e o valor da data**

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Enc | EnZ | WR\ | RD\ | EnN | EnR | EnA | PC0 | PC1 | A0 | A1 | A2 | Data |
| 1 | 0 | 1 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | B10h |
| 0 | 1 | 1 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 710h |
| 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 308h |
| 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 300h |
| 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 383h |
| 0 | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 340h |
| 0 | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 320h |
| 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 182h |
| 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 200h |
| 0 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 381h |
| 1 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | B81h |
| 1 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | B81h |
| 0 | 1 | 1 | 1 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 1 | 7A5h |

No Arduíno…

**Conclusão**

Com a realização do CPU concluímos que é um dispositivo capaz de ler instruções e realizar operações aritméticas, lógicas e controla a entrada e a saída de dados. Através da realização de vários passos conseguimos cumprir o objetivo de implementar o CPU em Arduíno.

**Bibliografia**

Pais, Eng. Jorge, PowerPoint Computação Física, pp.36-50

Carvalho, Eng. Carlos, PowerPoint Computação Física, pp. 200-207