
ADDETC – Área Departamental de Engenharia Eletrónica e Telecomunicações e de Computadores

LEIM -Licenciatura Engenharia informática e multimédia

# Computação Física

**Trabalho 2**

## Microprocessador

**Turma:** LEIM23D

**Trabalho realizado por:** Miguel Távora Nº45102
João Cunha Nº45412
Luís Farinha Nº45147

**Docente:** Carlos Carvalho

Data de entrega:10/5/2019

## Índice

**1.INTRODUÇÃO/OBJETIVOS** ..... **3**
**2.CONCEITOS UTILIZADOS NA REALIZAÇÃO DA CPU** ..... **4**
**2.1 ASSEMBLY** ..... **4**
**2.2 PORTAS TRI-STATE** ..... **4**
**2.3 CONCEITO DE ENABLE** ..... **5**
**3.ESPECIFICAÇÃO DA QUANTIDADE DE BITS DE CADA UM DOS REGISTOS** ..... **6**
**3.1 JUSTIFICAÇÃO DAS QUANTIDADES DE BITS COM BASE NAS INSTRUÇÕES** ..... **6**
**4.ADRESS BUS E DATA BUS** ..... **7**
**4.1 JUSTIFICAÇÃO DAS QUANTIDADES DE BITS COM BASE NAS INSTRUÇÕES** ..... **8**
**5.ESPECIFICAÇÃO E CODIFICAÇÃO DAS INSTRUÇÕES** ..... **9**
**5.1 ESPECIFICAÇÃO DAS INSTRUÇÕES** ..... **9**
**5.2 CODIFICAÇÃO DAS INSTRUÇÕES** ..... **9**
**6. DESENHO DO MODULO FUNCIONAL** ..... **11**
**7. MÓDULO DE CONTROLO** ..... **12**
**7.1 EPROM 128X11** ..... **13**
**8. OTIMIZAÇÃO** ..... **14**
**8.1EPROM 16X15** ..... **15**
**8.2 MODULO FUNCIONAL (OTIMIZADO)** ..... **16**
**9. SIMULAÇÃO DA ARQUITETURA E TESTES AO PROGRAMA** ..... **17**
**10.MONTAGEM DA SIMULAÇÃO DO CPU** ..... **22**
**11.CONCLUSÕES** ..... **23**
**12.BIBLIOGRAFIA** ..... **24**
**13.ANEXO** ..... **25**

## Indice de Figuras

Figure 1 - Expressão lógica da porta active-low ..... 4
Figure 2 - Arquitetura Harvard ..... 7
Figure 3 - Modulo Funcional V1 ..... 11
Figure 4 - Modulo de Controlo V1 ..... 12
Figure 5 - Modulo de Controlo auxiliar ..... 14
Figure 6 - Circuito logico do modulo de Controlo auxiliar ..... 14
Figure 7 - Modulo Funcional V2 ..... 16
Figure 8 - Esquema de montagem ..... 22
Figure 9 - Fotografia da montagem ..... 22

## Indice de Tabelas

Tabela 1 - Tabela de verdade da porta active-low ..... 4
Tabela 2 - Registo e número de bits ..... 6
Tabela 3 - Codificação das instruções ..... 10
Tabela 4 - Verificação dos sinais ativos ..... 12
Tabela 5 - EPROM128x11 do módulo de controlo ..... 13
Tabela 6 - EPROM16x15 do módulo de controlo ..... 15

# 1.Introdução/Objetivos

No âmbito da unidade curricular de Computação Física, foi nos solicitado a realização no segundo trabalho prático a construção de um microprocessador, baseado na arquitetura Harvard. Foi escolhida a arquitetura Harvard pois esta é possuídora de duas memórias diferentes e independentes em termos de barramentos, ou seja, os dados e o código estão em memórias físicas diferentes. Esta arquitetura é baseada numa arquitetura mais antiga chamada de Von Newmann, sendo que esta posssí apenas uma memória de dados e código. A arquitetura de Harvard surge com a necessidade de um processamento mais rápido e eficiente, porque permite o paralelismo de ações sobre a memória de código e dados.

O projeto possuí dois modulos distintos sendo eles o módulo funcional e o módulo de controlo. O módulo funcional é onde são executadas as ordens armazenadas na memória de código, ordens essas codificadas pelo módulo de controlo. Os códigos do módulo de controlo permite ligar multiplexers , flip-flops, registos, e memórias utilizadas no microprocessador. Para além de ligar os diversos dispositivos hardware também permite ligar os sinais que serão ativos durante cada instrução , permitindo não só estruturar o trabalho como também a leitura e escrita em registos e na memória de dados.

No decorrer do trabalho, será implemetado e testado o microprocessador com o auxilío de um botão para fazer os clock's, cumprindo desta forma as regras estipuladas no trabalho prático.

# 2.Conceitos utilizados na realização da CPU

### *2.1 Assembly*

Assembly ou linguagem de montagem é uma linguagem de baixo nível, legível por humanos para o código máquina, substituindo valores em bruto por símbolos chamados mnemónicos. A tradução do código Assembly para o código máquina é feita pelo montador (assembler) convertendo os mnemónicos nos seus opcodes. As instruções dadas no trabalho prático estão em linguagem Assembly.

### *2.2 Portas tri-state*

As portas tri-state são portas que possuem três estados possíveis “0”. “1” e “alta impedância”. A alta impedância(Z) é uma resistência infinita, ou seja, um circuito aberto onde a porta está desligada do resto do circuito a jusante da saída. A sua funcionalidade no CPU é a não realização de leitura e escrita simultânea na memória de dados.

Existem quatro variantes possíveis de portas tri-state, porém iremos usar somente a porta tri-state identidade com o control active-low.

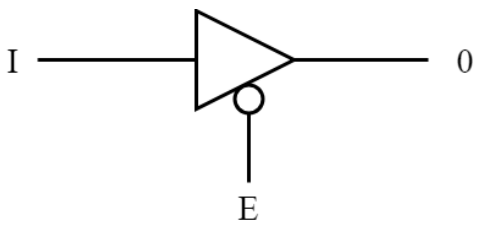



**Figure 1 - Expressão lógica da porta active-low**

| E | I | 0 |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | Z |
| 1 | 1 | Z |

**Tabela 1 - Tabela de verdade da porta active-low**

### *2.3 Conceito de Enable*

O terminal enable (E) quando ativo permite alojar na saída o valor que lhe for solicitado, podendo o valor ser (high ou low). Desta forma é possível o funcionamento do respetivo dispositivo em conjunto com outros aos quais ele se liga. Se o enable não estiver ativo, a saída fica "flutuante" não havendo ligação física, ficando o módulo inibido de realizar as suas funções. Este estado é utilizado para determinar se um dispositivo está em atividade ou não.

# 3.Especificação da quantidade de bits de cada um dos registos

O CPU(Central Processing Unit) do trabalho prático possuí quatro registos principais : registo V(value), registo R(reference), registo A(auxiliary) e o PC(Program Counter). Tambem possui registos que guardam as flags de Overflow(Ov), flag Zero(Z) e a flag de Carry(Cy). Os barramentos da CPU são todos constituídos por 8 bits, possuindo também rel5 e o end6 que são um número relativo a 5 bits e um número absoluto a 6 bits respetivamente.

| Registos | Nº de bits |
|---|---|
| V | 8 |
| A | 8 |
| R | 6 |
| PC | 6 |
| OV | 1 |
| Z | 1 |
| Cy | 1 |

**Tabela 2 - Registo e número de bits**

### *3.1 Justificação das quantidades de bits com base nas instruções*

Visto que existem 12 instruções diferentes e como o número mais próximo de 12 com base 2 é o número 4, então as instruções foram codificadas com 4 bits. Visto que $2^4 = 16$ sendo assim possível codificar todas as instruções necessárias para a realização do microprocessador.

**registoV:**
MOV V, #const8 (8bits)
MOV V, @r (8bits)
NOR V, A (8bits)
ADC V, A (8bits)
SBB V, A (8bits)

**registoA :**
MOV A,V (8bits)

**registoR :**
MOV R, #const6

**registoFlagOv:**
JOV rel5

**registoFlagZ:**
JZ rel5

**registoFlagCy:**
JNC rel5

**registoPC:**
Todas as instruções interagem com o PC (PC = PC + 1 ou PC = PC + rel5), mas a seguinte instrução define a sua quantidade de bits:
JMP end6 (PC = end6, 6bits)

## 4.Adress Bus e Data Bus

A arquitetura Harvard é constituída por três constituintes fundamentais, sendo eles a memória de dados, memória de código e a CPU(Central Processing Unit). A memória de código (code memory) é uma memória apenas de leitura e é onde estão guardadas as instruções que o programa irá cumprir, no caso de ser necessário executar outro programa é necessário regravar as instruções na memória. A memória de dados é onde são guardadas as informações provenientes da execução do programa, ou seja, as variaveís guardadas do programa (operandos e resultados). A CPU é onde o algoritmo do programa irá ser executado de forma sequêncial. Cada memória do microprocessador possuí o seu Adress Bus (AB) e o Data Bus (DB).

O Adress Bus é um apontador para um determinado endereço de memória. Na memória de código através do Adress Bus, a CPU sabe que instruções deverão ser executadas. No caso da memória de dados, a CPU sabe onde deverá ler/escrever os dados.

O Data Bus é transmitido apenas da memória de código para a CPU dando-lhe os bits das instruções a cumprir. Para a memória de dados a informação circula para os dois lados(bidirecional), representando os bits dos operandos e resultados que vão ser constantemente acedidos e alterados durante a execução do código.

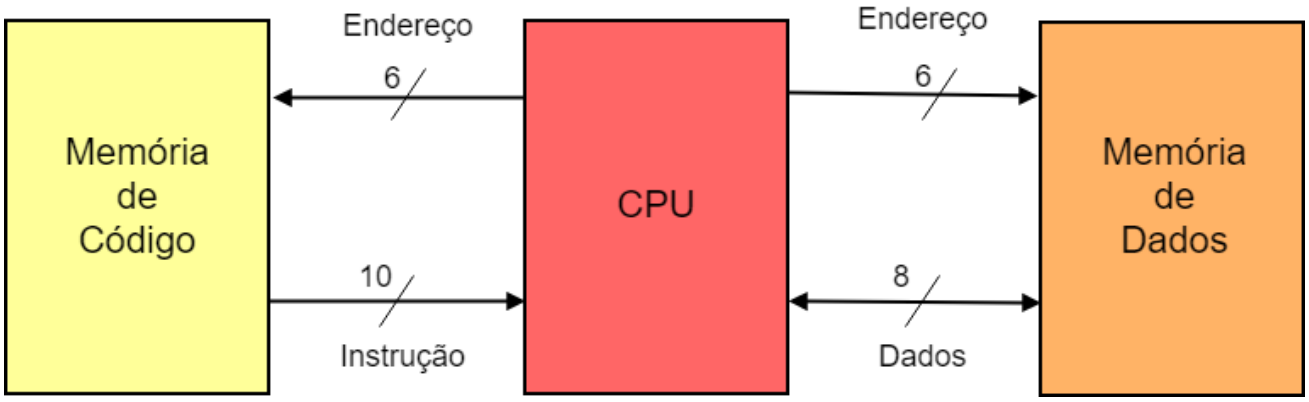



**Figure 2 - Arquitetura Harvard**

### *4.1 Justificação das quantidades de bits com base nas instruções*

O Registo do Program Counter (PC) é o registo que permite navegar nos endereços da memória de código, sendo possuidor de um Adress Bus constituído por 6 bits. Desta forma é possível criar até $2^{PC} = 2^6$ = 64 instruções diferentes. Como o registo R é o responsável por endereçar a memória de dados e este é constituído por 6 bits então é possível endereçar até $2^R = 2^6$ = 64 posições diferentes de memória.

Na memória de código o Data Bus é constituído por 10 bits, porque foram utilizados 10 bits para codificar as instruções. No caso da memória de dados como o maior número que se pode escrever é de 8 bits, então a sua Data Bus será de 8 bits (registo V).

# 5.Especificação e Codificação das instruções

### *5.1 Especificação das instruções*

As instruções também conhecidas como Insctuction Set (IS) é um conjunto de instruções em Assembly que o microprocessador utiliza para a execução dos algoritmos. As instruções possuem quatro classes distintas tendo por base as suas funcionalidades.

- Instruções de informação – são as instruções responsáveis pela leitura e escrita de registos, contadores, flags, e dispositivos externos ao CPU que partilham todos o mesmo barramento.

- Instruções aritméticas – realiza operações aritméticas normalmente realizadas na unidade aritmética e lógica (ALU – *Aritmétic Logic Unit*) pertencente á estrutura interna do CPU.

- Instruções lógicas – implementa funções lógicas comuns e geralmente realizadas na ALU.

- Instruções de controlo – permitem alterar de alguma forma a execução sequêncial das instruções.

### *5.2 Codificação das instruções*

O objetivo principal da codificação é construir as instruções com o menor número de bits possível. Contudo uma condicionante é que não podem haver códigos ambíguos, ou seja, o mesmo código não pode gerar duas instruções iguais, porque dessa forma estaria a comprometer o normal funcionamento do CPU. Como o maior número existente é constituído por 8 bits, o grupo decidiu atribuir a essa instrução 2 bits e quatro às restantes instruções, fazendo combinações entre os números para diferenciar cada instrução.

| | | | Codificação a 10bits | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Classe | Instrução | Parâmetro | D9 | D8 | D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |
| Transfêrencia de dados | MOV V, | #const8 | 0 | 0 | c7 | c6 | c5 | c4 | c3 | c2 | c1 | c0 |
| | MOV R, | #const6 | 0 | 1 | 0 | 0 | c5 | c4 | c3 | c2 | c1 | c0 |
| | MOV A, V | - | 0 | 1 | 0 | 1 | - | - | - | - | - | - |
| | MOV V, @R | - | 0 | 1 | 1 | 0 | - | - | - | - | - | - |
| | MOV @R, V | - | 0 | 1 | 1 | 1 | - | - | - | - | - | - |
| Instrução Lógica | NOR V, A | - | 1 | 0 | 0 | 0 | - | - | - | - | - | - |
| Instrução Aritmética | ADC V, A | - | 1 | 0 | 0 | 1 | - | - | - | - | - | - |
| | SBB V, A | - | 1 | 0 | 1 | 0 | - | - | - | - | - | - |
| Instrução de Controlo | JNC rel5 | rel5 | 1 | 0 | 1 | 1 | - | r4 | r3 | r2 | r1 | r0 |
| | JZ rel5 | rel5 | 1 | 1 | 0 | 0 | - | r4 | r3 | r2 | r1 | r0 |
| | JOV rel5 | rel5 | 1 | 1 | 0 | 1 | - | r4 | r3 | r2 | r1 | r0 |
| | JMP end6 | end6 | 1 | 1 | 1 | - | e5 | e4 | e3 | e2 | e1 | e0 |

**Tabela 3 - Codificação das instruções**

Para codificar as instruções o grupo teve por base os seguintes conceitos:

- Os bits D9 e D8 distinguem a primeira instrução das restantes, visto que a primeira instrução é a única que possuí o bit D9 e D8 a zero, todas as restantes possuem pelo menos um dos bits a 1.

- O bit D9 distingue as instruções de transferência de dados das restantes instruções.

- Os bits D7 e D6 diferencia dentro de cada classe de instruções, as diferentes instruções que existem.

# 6. Desenho do modulo funcional

Para realizar a simulação de um CPU, foi construído o módulo funcional do mesmo. A principal função do módulo funcional é o encaminhamento dos dados, fazendo a interligação de módulos hardware, encaminhando a informação dos dispositivos fonte para os dispositivos destino. Para isso foi utilizado o microcontrolador para simular os diferentes módulos através de diversas funções. O registo utilizado na construção do diagrama de blocos é do tipo edge-triggered com enable. Este registo atualiza a sua informação quando existe transição na sua entrada edge-triggered de clock e a entrada enable está ativa. Este tipo de registo é feito a partir de um conjunto de flip-flops tipo D edge-triggered e de um multiplexer.

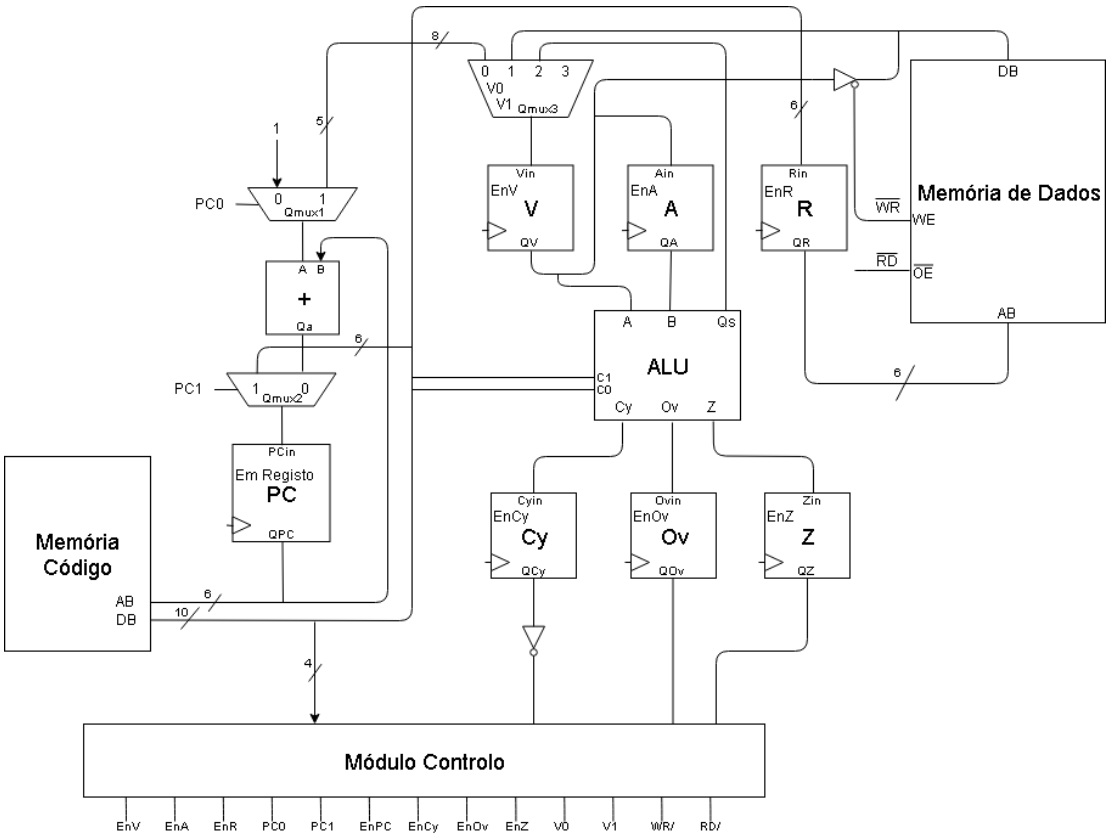



**Figure 3 - Modulo Funcional V1**

# 7. Módulo de Controlo

Para que os blocos do módulo funcional possam funcionar de forma eficiente é necessário encaminhar e ordenar a informação. Esta ordenação é possível através da construção do módulo de controlo, este módulo garante a ativação dos sinais de controlo dos dispositivos físicos envolvidos no módulo funcional.

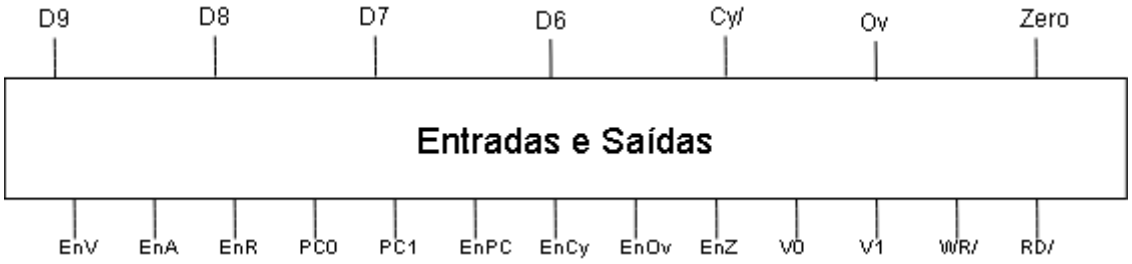



**Figure 4 - Modulo de Controlo V1**

Para a construção do módulo de Controlo foram utilizados os bits de valor fixo, utilizados aquando da codificação das instruções. Desta forma é possível diferenciar as instruções existentes, também é possível ativar durante a execução do programa as flags de Cy, Z e Ov caso o resultado assim o dite e também os sinais ativos durante a execução do programa.

| Instruções | Parâmetros | D9 | D8 | D7 | D6 | Ov | Z | Cy | Sinais ativos |
|---|---|---|---|---|---|---|---|---|---|
| MOV V, | #const8 | 0 | 0 | - | - | - | - | - | EnV, V0 , V1 |
| MOV R, | #const6 | 0 | 1 | 0 | 0 | - | - | - | EnR |
| MOV A, V | - | 0 | 1 | 0 | 1 | - | - | - | EnA |
| MOV V, @R | - | 0 | 1 | 1 | 0 | - | - | - | EnV, V0, RD/ |
| MOV @R, V | - | 0 | 1 | 1 | 1 | - | - | - | WR/ |
| NOR V, A | - | 1 | 0 | 0 | 0 | - | - | - | EnV, EnCy, EnOv, EnZ |
| ADC V, A | - | 1 | 0 | 0 | 1 | - | - | - | EnV, EnCy, EnOv, EnZ |
| SBB V, A | - | 1 | 0 | 1 | 0 | - | - | - | EnV, EnCy, EnOv, EnZ |
| JNC rel5 | rel5 | 1 | 0 | 1 | 1 | - | - | 0 | PC0 |
| JNC rel5 | rel5 | 1 | 0 | 1 | 1 | - | - | 1 | - |
| JZ rel5 | rel5 | 1 | 1 | 0 | 0 | - | 0 | - | - |
| JZ rel5 | rel5 | 1 | 1 | 0 | 0 | - | 1 | - | PC0 |
| JOV rel5 | rel5 | 1 | 1 | 0 | 1 | 0 | - | - | - |
| JOV rel5 | rel5 | 1 | 1 | 0 | 1 | 1 | - | - | PC0 |
| JMP end6 | end6 | 1 | 1 | 1 | - | - | - | - | PC1 |

**Tabela 4 - Verificação dos sinais ativos**

### *7.1 EPROM 128x11*

A EPROM (“Erasable Programmable Read-Only Memory”) é uma memória apagável somente de leitura do qual é possível relacionar cada uma das instruções com os sinais do módulo de controlo. Os sinais do módulo de controlo devem ser ativos para que as instruções possam ser cumpridas com sucesso.

Para cada instrução são utilizados 4 bits de valores constantes que permitem a diferenciação das instruções, para além dos valores constantes temos ainda 3 bits para flags Cy, Z e Ov obtendo desta forma uma EPROM de $2^7$= 128 endereços. Cada endereço de memória tem os seus respetivos dados de ativação de sinais, obtidos através das combinações de 11bits. Sempre que é executada uma instrução o módulo de controlo ativa os sinais previamente definidos.

| Instrução | Parâmetros | D9 | D8 | D7 | D6 | Ov | Z | Cy | Endereço | EnV | EnA | EnR | PC0 | PC1 | EnPC | EnOv | EnZ | EnCy | V0 | V1 | WR/ | RD/ | Dados |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| MOV V, | #const8 | 0 | 0 | - | - | - | - | - | [0x00 - 0x1F] | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0x1083 |
| MOV R, | #const6 | 0 | 1 | 0 | 0 | - | - | - | [0x20 - 0x27] | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0x483 |
| MOV A, V | - | 0 | 1 | 0 | 1 | - | - | - | [0x28 - 0x2F] | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0x883 |
| MOV V, @R | - | 0 | 1 | 1 | 0 | - | - | - | [0x30 - 0x37] | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0x108A |
| MOV @R, V | - | 0 | 1 | 1 | 1 | - | - | - | [0x38 - 0x3F] | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0x81 |
| NOR V, A | - | 1 | 0 | 0 | 0 | - | - | - | [0x40 - 0x47] | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 0x1087 |
| ADC V, A | - | 1 | 0 | 0 | 1 | - | - | - | [0x48 - 0x4F] | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 0x10F7 |
| SBB V, A | - | 1 | 0 | 1 | 0 | - | - | - | [0x50 - 0x57] | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 0x10F7 |
| JNC rel5 | rel5 | 1 | 0 | 1 | 1 | - | - | 0 | [0x58, 0x5A, 0x5C, 0x5E] | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0x83 |
| JNC rel5 | rel5 | 1 | 0 | 1 | 1 | - | - | 1 | [0x59, 0x5B, 0x5D, 0x5F] | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0x283 |
| JZ rel5 | rel5 | 1 | 1 | 0 | 0 | - | 0 | - | [0x60, 0x61, 0x64, 0x65} | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0x83 |
| JZ rel5 | rel5 | 1 | 1 | 0 | 0 | - | 1 | - | [0x62,0x63, 0x66, 0x67] | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0x283 |
| JOV rel5 | rel5 | 1 | 1 | 0 | 1 | 0 | - | - | [0x68, 0x69, 0x6A, 0x6B] | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0x83 |
| JOV rel5 | rel5 | 1 | 1 | 0 | 1 | 1 | - | - | [0x6C, 0x6D, 0x6E,0x6F] | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0x283 |
| JUMP end6 | end6 | 1 | 1 | 1 | - | - | - | - | [0x70 - 0x7F] | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0x183 |

**Tabela 5 - EPROM128x11 do módulo de controlo**

## 8. Otimização

Para reduzir a dimensão da EPROM de $2^7$(128) para $2^4$ (16), verificou-se que as várias flags de JNC, JZ e JOV apenas controlam o PC0, podendo assim ser substituídas no endereçamento. Conseguindo assim obter uma expressão com as saídas do módulo de controlo que indicam qual das instruções está a ser realizada e a saída dos registos das flags:

$$PC0 = JNC.\overline{Cy} + JZ.Z + JOV.Ov$$

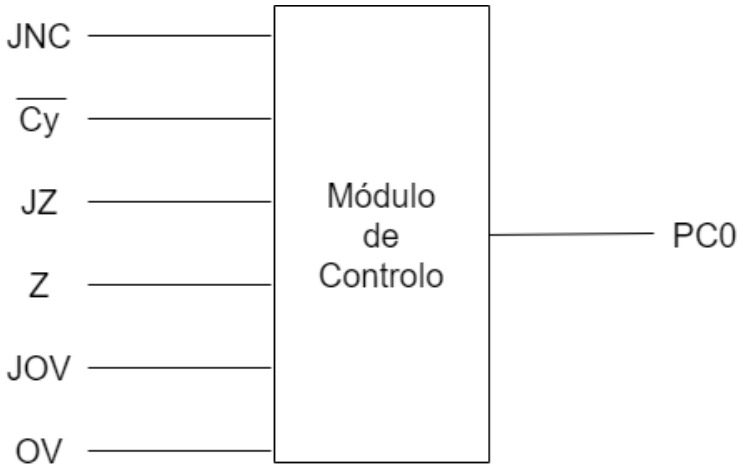



**Figure 5 - Modulo de Controlo auxiliar**

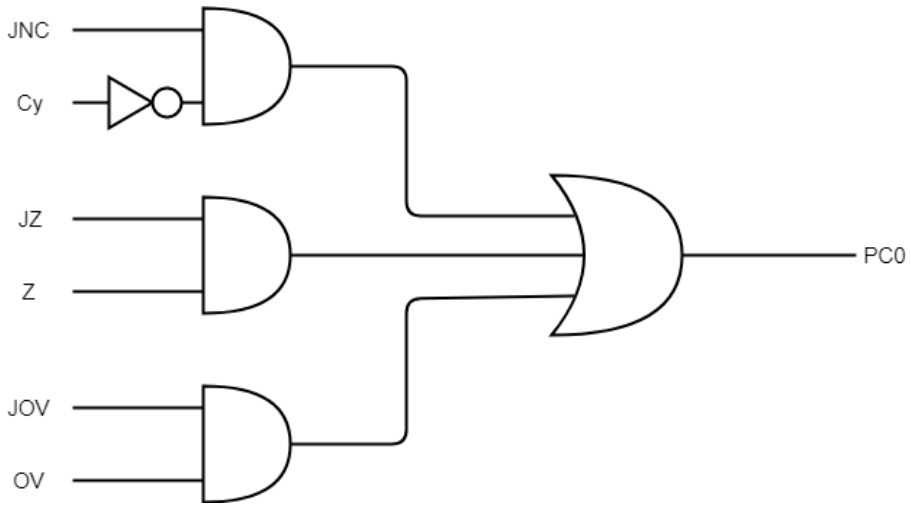



**Figure 6 - Circuito logico do modulo de Controlo auxiliar**

### *8.1EPROM 16x15*

| Instruções | Parâmetros | D9 | D8 | D7 | D6 | Endereço | EnV | EnA | EnR | EnPC | EnOv | EnZ | EnCy | V0 | V1 | WR/ | RD/ | JMP | JOV | JZ | JNC | Dados |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| MOV V, | #const8 | 0 | 0 | - | - | [0x00 - 0x03] | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0x4830 |
| MOV A, | #const6 | 0 | 1 | 0 | 0 | [ 0x04] | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0x1830 |
| MOV A, V | - | 0 | 1 | 0 | 1 | [0x05] | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0x1430 |
| MOV V,@R | - | 0 | 1 | 1 | 0 | [0x06] | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0x48A0 |
| MOV @R, V | - | 0 | 1 | 1 | 1 | [0x07] | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0xA70 |
| NOR V, A | - | 1 | 0 | 0 | 0 | [0x08] | 1 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0x4A70 |
| ADC V, A | - | 1 | 0 | 0 | 1 | [0x09] | 1 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0x4F70 |
| SUBB V, A | - | 1 | 0 | 1 | 0 | [0xA] | 1 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0x4F70 |
| JNC rel5 | rel5 | 1 | 0 | 1 | 1 | [0xB] | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 1 | 0x831 |
| JZ rel5 | rel5 | 1 | 1 | 0 | 0 | [0xC] | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 0x832 |
| JOV rel5 | rel5 | 1 | 1 | 0 | 1 | [0xD] | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 0x834 |
| JUMP end6 | end6 | 1 | 1 | 1 | - | [0xE - 0xF] | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 0x838 |

**Tabela 6 - EPROM16x15 do módulo de controlo**

Comparando a tabela EPROM otimizada de 16x15(240bits) com a tabela realmente utilizada de 128x13 (1664bits), o grupo conseguiu assim reduzir o tamanho da EPROM para cerca de $\frac{1}{6}$ do tamanho. A tabela reduzida foi construída após a conclusão da implementação no Arduíno, pelo que não foi de facto utilizada no trabalho prático.

### *8.2 Modulo funcional (otimizado)*

A implementação em blocos hardware (módulo funcional) seria da seguinte maneira:

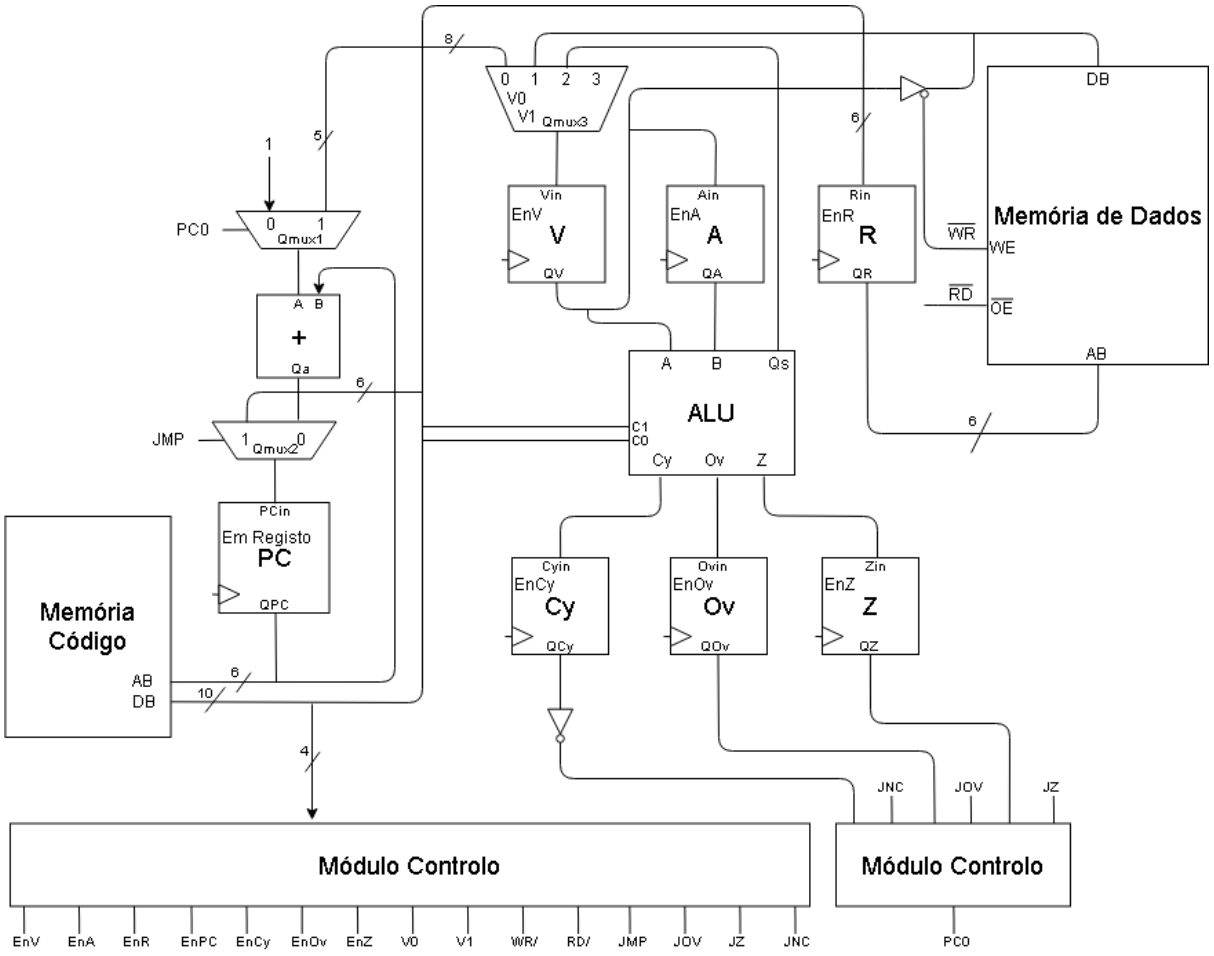



**Figure 7 - Modulo Funcional V2**

# 9. Simulação da arquitetura e testes ao programa

Foram feitos vários testes à simulação Arduíno:

```
425 void teste0(){//som, sub resultado = (8+6 = 14; 24-14 = 10)
426   InstructionMemory[0] =  0x6;   // move V,6
427   InstructionMemory[1] =  0x140; // mov A,V
428   InstructionMemory[2] =  0x8;   // move V, 8
429   InstructionMemory[3] =  0x240; // V = V + A
430   InstructionMemory[4] =  0x140; // move A,V
431   InstructionMemory[5] =  0x18;  // move V,24
432   InstructionMemory[6] =  0x280; // V = 24 - 14
433 }
```

Verificação passo a passo:

Mov V, 6

```
Processador
PC: 0
V: 6
A: 0
R: 0
Codigo: 0
RAM
1 Posicao: 0
2 Posicao: 0
Flag Ov: 0
Flag Zero: 0
Flag Cy: 0
```

Mov A,V

```
Processador
PC: 1
V: 6
A: 6
R: 0
Codigo: 101
RAM
1 Posicao: 0
2 Posicao: 0
Flag Ov: 0
Flag Zero: 0
Flag Cy: 0
```

Mov V, 8

```
Processador
PC: 2
V: 8
A: 6
R: 0
Codigo: 0
RAM
1 Posicao: 0
2 Posicao: 0
Flag Ov: 0
Flag Zero: 0
Flag Cy: 0
```

V = V + A

```
Processador
PC: 3
V: 14
A: 6
R: 0
Codigo: 1001
RAM
1 Posicao: 0
2 Posicao: 0
Flag Ov: 0
Flag Zero: 0
Flag Cy: 0
```

Mov A,V

```
Processador
PC: 4
V: 14
A: 14
R: 0
Codigo: 101
RAM
1 Posicao: 0
2 Posicao: 0
Flag Ov: 0
Flag Zero: 0
Flag Cy: 0
```

Mov V, 24

```
Processador
PC: 5
V: 24
A: 14
R: 0
Codigo: 0
RAM
1 Posicao: 0
2 Posicao: 0
Flag Ov: 0
Flag Zero: 0
Flag Cy: 0
```

V = 24 - 14

```
Processador
PC: 6
V: 10
A: 14
R: 0
Codigo: 1010
RAM
1 Posicao: 0
2 Posicao: 0
Flag Ov: 0
Flag Zero: 0
Flag Cy: 1
```

```
435 void testel(){//nor resultado = 17(0b10001) e teste flag z
436   InstructionMemory[0] =  0xAA;  // mov V, 0b10101010
437   InstructionMemory[1] =  0x140; // mov A,V
438   InstructionMemory[2] =  0xCC;  // mov V, 0b11001100
439   InstructionMemory[3] =  0x200; // V = V nor A
440   InstructionMemory[4] =  0x0;   // mov V, 0
441   InstructionMemory[5] =  0x140; // mov A,V
442   InstructionMemory[6] =  0xFF;  // mov V, 0b11111111
443   InstructionMemory[7] =  0x200; // V = V nor A
444   InstructionMemory[8] =  0x308; // ( JZ, rel5 = 8)
445 }
```

Mov V, 0xAA

```
Processador
PC: 0
V: 170
A: 0
R: 0
Codigo: 0
RAM
1 Posicao: 0
2 Posicao: 0
Flag Ov: 0
Flag Zero: 0
Flag Cy: 0
```

→ Mov A,V

```
Processador
PC: 1
V: 170
A: 170
R: 0
Codigo: 101
RAM
1 Posicao: 0
2 Posicao: 0
Flag Ov: 0
Flag Zero: 0
Flag Cy: 0
```

→ Mov V, 0xCC

```
Processador
PC: 2
V: 204
A: 170
R: 0
Codigo: 0
RAM
1 Posicao: 0
2 Posicao: 0
Flag Ov: 0
Flag Zero: 0
Flag Cy: 0
```

→ V NOR A

```
Processador
PC: 3
V: 17
A: 170
R: 0
Codigo: 1000
RAM
1 Posicao: 0
2 Posicao: 0
Flag Ov: 0
Flag Zero: 0
Flag Cy: 0
```

→ Mov V, 0

```
Processador
PC: 4
V: 0
A: 170
R: 0
Codigo: 0
RAM
1 Posicao: 0
2 Posicao: 0
Flag Ov: 0
Flag Zero: 0
Flag Cy: 0
```

Mov A,V

```
Processador
PC: 5
V: 0
A: 0
R: 0
Codigo: 101
RAM
1 Posicao: 0
2 Posicao: 0
Flag Ov: 0
Flag Zero: 0
Flag Cy: 0
```

→ Mov V, 0xFF

```
Processador
PC: 6
V: 255
A: 0
R: 0
Codigo: 0
RAM
1 Posicao: 0
2 Posicao: 0
Flag Ov: 0
Flag Zero: 0
Flag Cy: 0
```

→ V NOR A

```
Processador
PC: 7
V: 0
A: 0
R: 0
Codigo: 1000
RAM
1 Posicao: 0
2 Posicao: 0
Flag Ov: 0
Flag Zero: 1
Flag Cy: 0
```

→ JZ

```
Processador
PC: 8
V: 0
A: 0
R: 0
Codigo: 1100
RAM
1 Posicao: 0
2 Posicao: 0
Flag Ov: 0
Flag Zero: 1
Flag Cy: 0
```

→

```
Processador
PC: 16
V: 0
A: 0
R: 0
Codigo: 0
RAM
1 Posicao: 0
2 Posicao: 0
Flag Ov: 0
Flag Zero: 1
Flag Cy: 0
```

```
447 void teste2(){//Data memory write and write
448   InstructionMemory[0] =  0x8;   // mov V, 8
449   InstructionMemory[1] =  0x1C0; // mov @r, V
450   InstructionMemory[2] =  0x10;  // mov V,10
451   InstructionMemory[3] =  0x101; // mov R, 1
452   InstructionMemory[4] =  0x1C0; // mov @r, V
453   InstructionMemory[5] =  0x100; // mov R,0
454   InstructionMemory[6] =  0x180; // move V,@r
455 }
```

Mov V, 8

```
Processador
PC: 0
V: 8
A: 0
R: 0
Codigo: 0
RAM
1 Posicao: 0
2 Posicao: 0
Flag Ov: 0
Flag Zero: 0
Flag Cy: 0
```

Mov @r, V

```
Processador
PC: 1
V: 8
A: 0
R: 0
Codigo: 111
RAM
1 Posicao: 8
2 Posicao: 0
Flag Ov: 0
Flag Zero: 0
Flag Cy: 0
```

Mov V,10

```
Processador
PC: 2
V: 16
A: 0
R: 0
Codigo: 0
RAM
1 Posicao: 8
2 Posicao: 0
Flag Ov: 0
Flag Zero: 0
Flag Cy: 0
```

Mov R, 1

```
Processador
PC: 3
V: 16
A: 0
R: 1
Codigo: 100
RAM
1 Posicao: 8
2 Posicao: 0
Flag Ov: 0
Flag Zero: 0
Flag Cy: 0
```

Mov @r,

```
Processador
PC: 4
V: 16
A: 0
R: 1
Codigo: 111
RAM
1 Posicao: 8
2 Posicao: 16
Flag Ov: 0
Flag Zero: 0
Flag Cy: 0
```

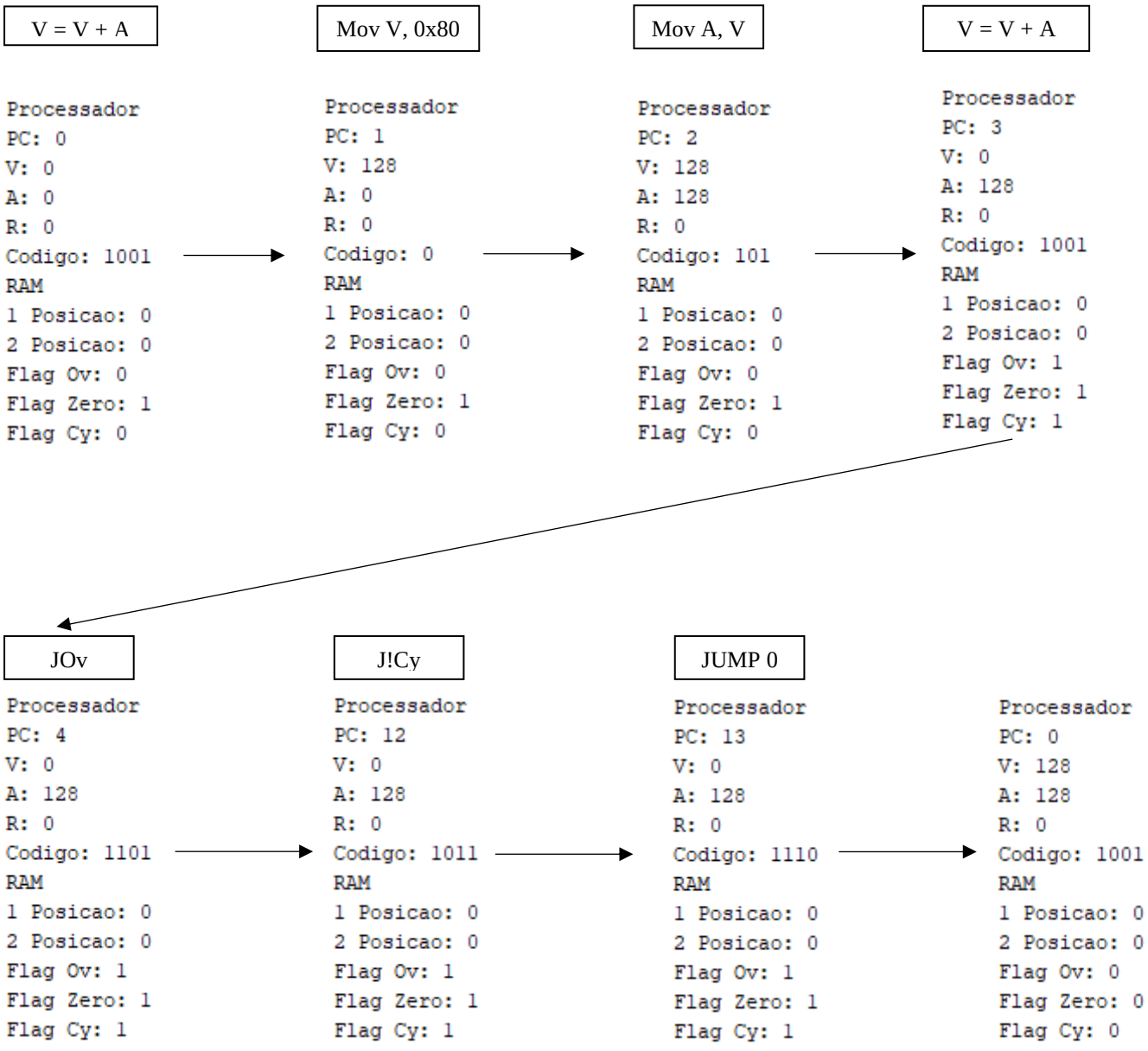
Mov R, 0

```
Processador
PC: 5
V: 16
A: 0
R: 0
Codigo: 100
RAM
1 Posicao: 8
2 Posicao: 16
Flag Ov: 0
Flag Zero: 0
Flag Cy: 0
```

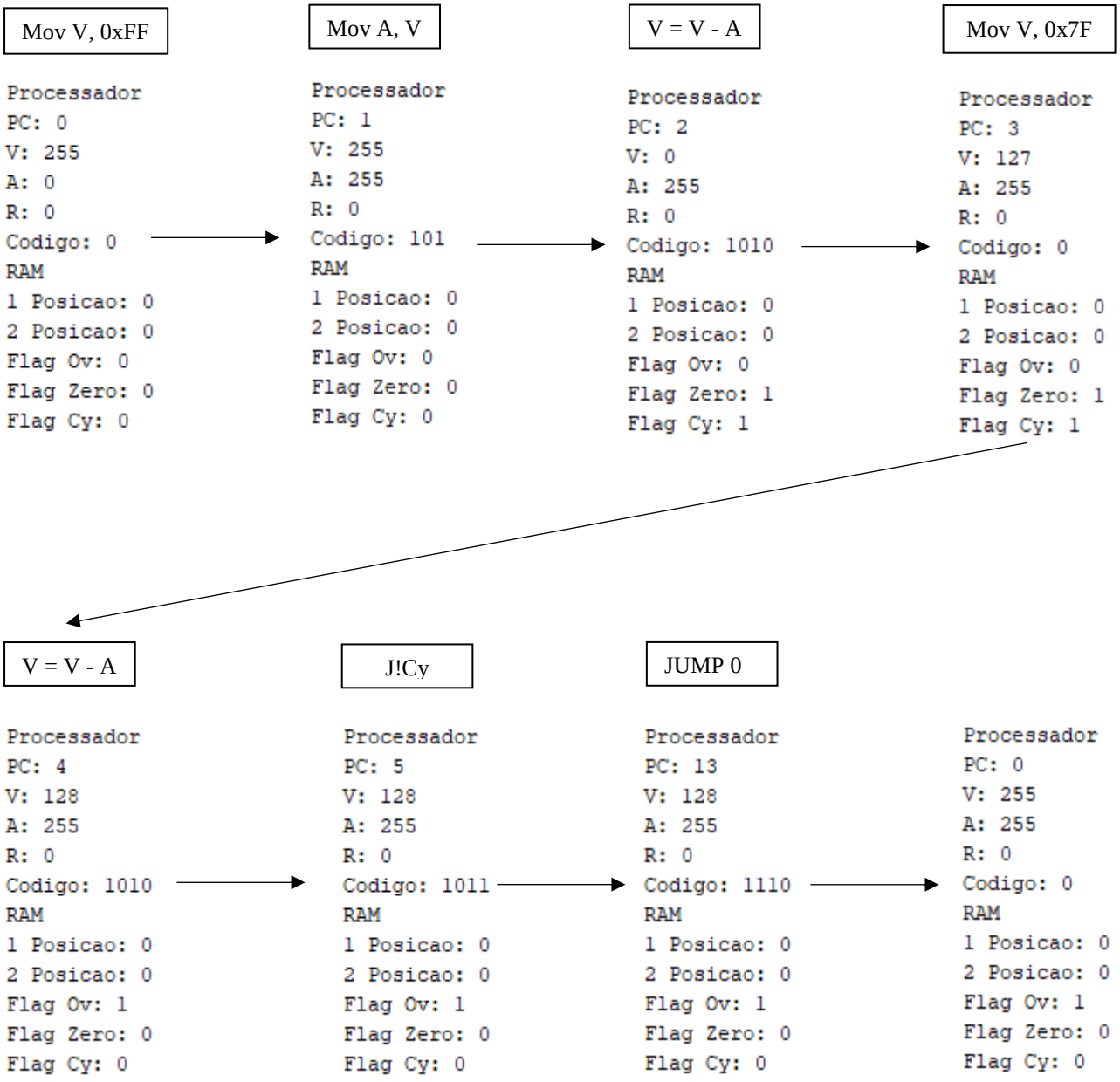
Mov V, @r

```
Processador
PC: 6
V: 8
A: 0
R: 0
Codigo: 110
RAM
1 Posicao: 8
2 Posicao: 16
Flag Ov: 0
Flag Zero: 0
Flag Cy: 0
```

```
457 void teste3(){//soma flags test z, cy e ov
458   InstructionMemory[0] =  0x240; // V = V + A (0+0)
459   InstructionMemory[1] =  0x80;  // V = 0b10000000
460   InstructionMemory[2] =  0x140; // mov A,V
461   InstructionMemory[3] =  0x240; // V = V + A
462   InstructionMemory[4] =  0x348; // (jov, rel5=8)
463   InstructionMemory[12] = 0x2C8; // (j!Cy, rel5=8)
464   InstructionMemory[13] = 0x380; // (jump 0)
465 }
```



```
467 void teste4(){//sub, flags test z,cy, ov
468   InstructionMemory[0] =  0xFF;  // V = 0bllllllll
469   InstructionMemory[1] =  0x140; // mov A,V
470   InstructionMemory[2] =  0x280; // V = V - A
471   InstructionMemory[3] =  0x7F;  // V = 0bllllllll
472   InstructionMemory[4] =  0x280; // V = V - A
473   InstructionMemory[5] =  0x2C8; //(j!Cy, rel5 = 8)
474   InstructionMemory[13] =  0x380; //(jump 0)
475 }
```



# 10.Montagem da simulação do CPU

Esquema de montagem:

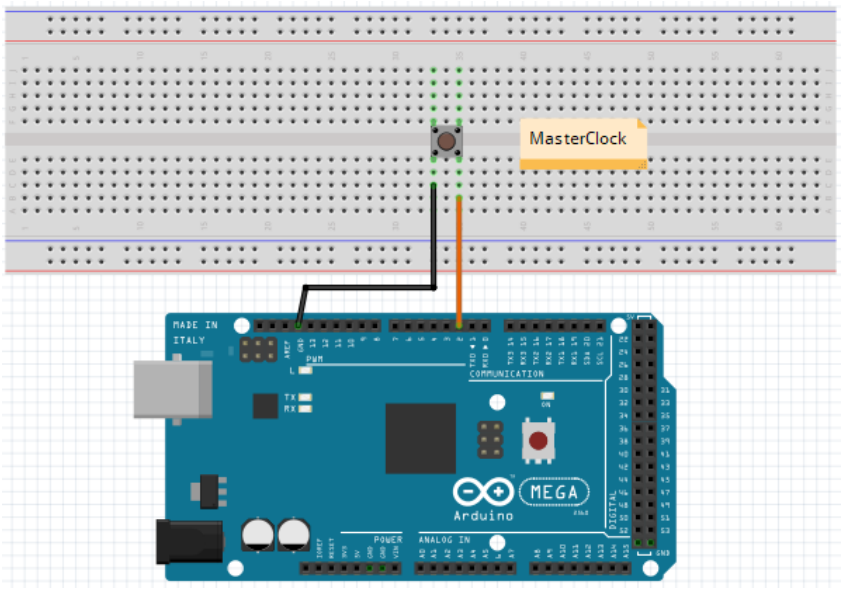



**Figure 8 - Esquema de montagem**

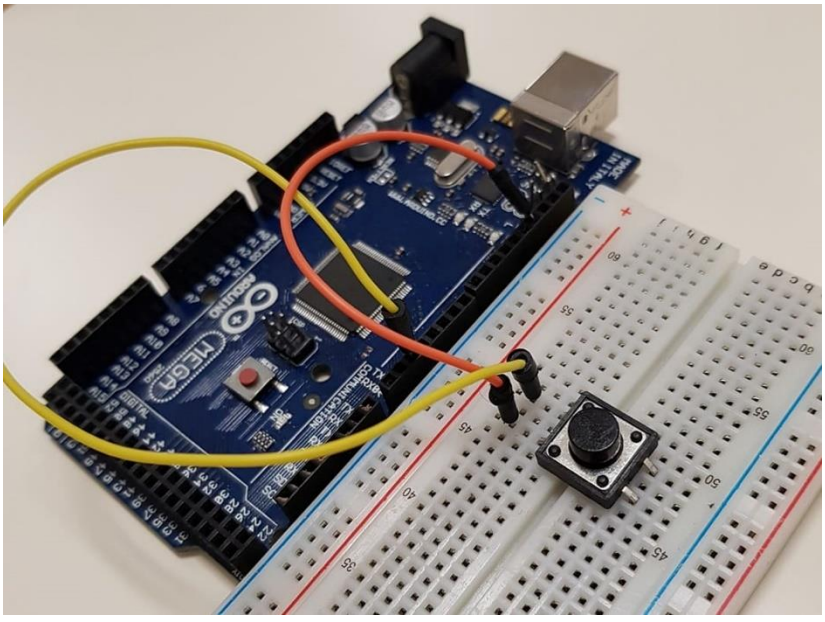


**Figure 9 - Fotografia da montagem**

# 11.Conclusões

Durante a realização do trabalho prático o grupo foi capaz de adquirir conhecimentos como:

- Diferenças entre a arquitetura de Harvard face à arquitetura de Von Newmann.
- Especificação dos bits utilizados pelos registos V, A, R e Program Counter.
- Utilização de portas tri-state e funcionalidade do Enable.
- Especificação dos barramentos das memórias de código e memória de dados (Adress Bus e Data Bus).
- Classificação das Instruções da memória de código em linguagem Assembly.
- Desenho do módulo funcional que mostra o encaminhamento dos dados perante as diferentes instruções.
- Desenho do módulo de controlo a encaminhar a informação no módulo de controlo.
- Construção da tabela EPROM para cada sinal e os sinais ativos.
- Otimizar os barramentos do modulo de controlo de forma a utilizar o menor número de bits possível.
- Montagem e simulação do programa no microcontrolador.
- Realização de testes ao microcontrolador.

Podemos concluir que o objetivo de simular a arquitetura de um CPU utilizando a arquitetura Harvard foi bem realizado, tendo sido testados todas as instruções do enunciado.

# 12.Bibliografia

[1] Folhas de Computação Física, Jorge Pais, 2018/2019

[2] Folhas de Computação Física, Carlos Carvalho, 2019

Links:

pt.wikipedia.org/wiki/Assembly

pt.wikipedia.org/wiki/EPROM

# 13.ANEXO

```
#define DATA_LENGTHT 256
#define INSTRUNCTION_LENGTHT 256

int DataMemory[DATA_LENGTHT];//Array da memória de dados
int InstructionMemory[INSTRUNCTION_LENGTHT];//Array da memória de Código

unsigned int ep[128]; //Tabela EPROM
int fromMemory;

////////////

bool EnV;          //Enable do Registo V
bool EnA;          //Enable do Registo A
bool EnR;          //Enable do Registo R
bool P0;          //seletor do multiplexer onde seleciona se é jump ou se nao
bool P1;          //seletor do multiplexer onde seleciona se é jump ou se nao
bool EnPc;         //Enable da Flag carry
bool EnOv;         //Enable da Flag overflow
bool EnZ;          //Enable da Flag zero
bool EnCy;         //Enable da Flag carry
bool V0;           //seletor do multiplexer
bool V1;           //seletor do multiplexer
bool WR;           //Write Enable
bool RD;           //Read Enable
//////////////
int instrucao = 0;
byte soma, sub, nor, S, Z, Ov, Cy;
byte registoPC = 0;
byte registoV = 0;
byte registoR = 0;
byte registoA = 0;
bool JNC = 0;

bool JZ = 0;
bool JOV = 0;
bool JMP = 0;
bool registoFlagCy = 0;
bool registoFlagZ = 0;
bool registoFlagOV = 0;
bool alu1 = 0;
bool alu0 = 0;
bool Cy1bit = 0;
bool Cy1bit2 = 0;

bool somaCy, subCy;
bool somaOv, subOv;
bool norZ, somaZ, subZ;

void setup() {
  Serial.begin(9600);
  pinMode (2, INPUT_PULLUP);
  createEPROM();
  createDataMemory();
  teste4();
  reset();

}

void loop() {
  printValores();
  instrucao = InstructionMemory[registoPC];
  controlModule(instrucao>>6, registoFlagOV,registoFlagZ, !registoFlagCy );
  alu1 = ((instrucao>>7)&0b1);
  alu0 = ((instrucao>>6)&0b1);
  alu(registoV,registoA, alu1,alu0);
```

```
  dataMemory(WR,RD, registoR, registoV);

  if(clock1(2)){
    execRegistos();
    imprimirRegistos();
    execPC();
    delay(400);
  }

}
void reset(){
  RegistoPC(0,0,0);
  RegistoV(0,0,0);
  RegistoA(0,0,0);
  RegistoR(0,0,0);
  RegistoFlagCy(0,0,0);
  RegistoFlagZ(0,0,0);
  RegistoFlagOV(0,0,0);
}

boolean clock1 (int btn){
  if (digitalRead (btn) == HIGH)
    return false;
  else
    return true;
}
void execPC(){
  byte a = mux2x1(1,instrucao&0b11111,P0);
  a = somador(a,registoPC);
  a = mux2x1(a,instrucao&0b111111,P1);
  RegistoPC(a,registoPC,1);

}

void execRegistos(){
  byte v = mux4x1(instrucao&0b11111111, fromMemory, S,0,V1,V0);
  RegistoV(v,registoV,EnV);
  RegistoA(registoV,registoA,EnA);
  RegistoR(instrucao&0b111111,registoR,EnR);
}


void controlModule(byte MEMcodigo, byte QOv, byte QZ, byte QCy){
  //Serial.println(ep[MEMcodigo<<3|QOv<<2|QZ<<1|QCy],BIN);
  readControlModule(MEMcodigo<<3|QOv<<2|QZ<<1|QCy);

}

void RegistoPC(byte in, byte out, bool enable){//8bits
  registoPC = (enable)?in:out;
}

void RegistoV(byte in, byte out, bool enable){//8bits
  registoV = (enable)?in:out;
}

void RegistoR(byte in, byte out, bool enable){//8bits
  registoR = (enable)?in:out;
}
void RegistoA(byte in, byte out, bool enable){//8bits
  registoA = (enable)?in:out;
}
void RegistoFlagCy(bool in, bool out, bool enable){
  registoFlagCy = (enable)?in:out;
```

```
}
void RegistoFlagZ(bool in,bool out, bool enable){
  registoFlagZ = (enable)?in:out;
}
void RegistoFlagOV(bool in,bool out, bool enable){
  registoFlagOV = (enable)?in:out;
}

void alu(byte a, byte b, bool C1, bool C0){
  bool A[] = {0,0,0,0,0,0,0,0};
  bool B[] = {0,0,0,0,0,0,0,0};
  A[7] = mux2x1(0,1,(a&(1<<7)));
  A[6] = mux2x1(0,1,(a&(1<<6)));
  A[5] = mux2x1(0,1,(a&(1<<5)));
  A[4] = mux2x1(0,1,(a&(1<<4)));
  A[3] = mux2x1(0,1,(a&(1<<3)));
  A[2] = mux2x1(0,1,(a&(1<<2)));
  A[1] = mux2x1(0,1,(a&(1<<1)));
  A[0] = mux2x1(0,1,(a&(1<<0)));

  B[7] = mux2x1(0,1,(b&(1<<7)));
  B[6] = mux2x1(0,1,(b&(1<<6)));
  B[5] = mux2x1(0,1,(b&(1<<5)));
  B[4] = mux2x1(0,1,(b&(1<<4)));
  B[3] = mux2x1(0,1,(b&(1<<3)));
  B[2] = mux2x1(0,1,(b&(1<<2)));
  B[1] = mux2x1(0,1,(b&(1<<1)));
  B[0] = mux2x1(0,1,(b&(1<<0)));

  nor8bits(A,B);

  somador8bits(A,B);
  sub8bits(A,B);

  S = mux4x1(nor,soma,sub,0, C1,C0);
  RegistoFlagZ (mux4x1(norZ,somaZ,subZ,0, C1,C0),registoFlagZ,EnZ);
  RegistoFlagOV( mux4x1(0,somaOv,subOv,0, C1,C0),registoFlagOV,EnOv);
  RegistoFlagCy( mux4x1(0,somaCy,subCy,0, C1,C0),registoFlagCy,EnCy);


//  Serial.println("saida");
//  Serial.println(a,BIN);
//  Serial.println(b,BIN);
//  Serial.println(S,BIN);



}

void readControlModule(int posicao){
  bool exitM[13];
  int count = 0;
  for(int mask = 0x1000; mask; mask >>= 1){      //Esta máscara serve para imprimir os zeros
    if(mask  & ep[posicao]){                    //que se encontram antes do primeiro um,
      exitM [count]= 1;                         //porque se o número for 000010010 o arduino
      //Serial.print(1);
    }                                           //imprime 10010 e não 000010010.
    else{
      exitM [count]= 0;
      //Serial.print(0);
    }
    count ++;
```

```
  }
  //Serial.println();
  EnV = exitM[0];
  EnA = exitM[1];
  EnR = exitM[2];
  P0 = exitM[3];
  P1 = exitM[4];
  EnPc = exitM[5];
  EnOv = exitM[6];
  EnZ = exitM[7];
  EnCy = exitM[8];
  V0 = exitM[9];
  V1 = exitM[10];
  WR = exitM[11];
  RD = exitM[12];


}

void createEPROM(){
  for(int i = 0; i <128;i++){
    if (i < 32)//mov v, #const8
      ep[i] = 0x1083;
    else if (i < 40)//mov r, #const6
      ep[i] = 0x483;
    else if (i < 48)//mov a, v
      ep[i] = 0x883;
    else if (i < 56)//mov v,@r
      ep[i] = 0x108A;
    else if (i < 64)//mov @r, v
      ep[i] = 0x81;

    else if (i < 72)//nor
      ep[i] = 0x10A7;
    else if (i < 80){//add
      ep[i] = 0x10F7;
    }
    else if (i < 88)//sub
      ep[i] = 0x10F7;
    else if (i==88||i==90||i==92||i==94)//JNC
      ep[i] = 0x83;
    else if(i==89||i==91||i==93||i==95)//jnc
      ep[i] = 0x283;
    else if(i==96||i==97||i==100||i==101)
      ep[i] = 0x83;
    else if(i==98||i==99||i==102||i==103)
      ep[i] = 0x283;
    else if (i < 108)//j0v
      ep[i] = 0x83;
    else if (i < 112)//jov
      ep[i] = 0x283;
    else if (i < 128)//jmp
      ep[i] = 0x183;
  }

}

void dataMemory (byte EnWrite, byte EnRead, byte QR, byte QV){
  if(~EnWrite&EnRead){          //write
    DataMemory[QR] = QV;
  }
  if(EnWrite&~EnRead){          //read
```

```
    fromMemory = DataMemory[QR];
  }
}

byte somador(byte a, byte b){
  bool S[] = {0,0,0,0,0,0};
  bool A[] = {0,0,0,0,0,0};
  bool B[] = {0,0,0,0,0,0};
  A[5] = mux2x1(0,1,(a&(1<<5)));
  A[4] = mux2x1(0,1,(a&(1<<4)));
  A[3] = mux2x1(0,1,(a&(1<<3)));
  A[2] = mux2x1(0,1,(a&(1<<2)));
  A[1] = mux2x1(0,1,(a&(1<<1)));
  A[0] = mux2x1(0,1,(a&(1<<0)));

  B[5] = mux2x1(0,1,(b&(1<<5)));
  B[4] = mux2x1(0,1,(b&(1<<4)));
  B[3] = mux2x1(0,1,(b&(1<<3)));
  B[2] = mux2x1(0,1,(b&(1<<2)));
  B[1] = mux2x1(0,1,(b&(1<<1)));
  B[0] = mux2x1(0,1,(b&(1<<0)));
  Cy1bit2 = 0;
  S[0] = somador1bit(A[0],B[0],0);
  S[1] = somador1bit(A[1],B[1],Cy1bit);
  S[2] = somador1bit(A[2],B[2],Cy1bit);
  S[3] = somador1bit(A[3],B[3],Cy1bit);
  S[4] = somador1bit(A[4],B[4],Cy1bit);
  S[5] = somador1bit(A[5],B[5],Cy1bit);
  return S[5]<<5 | S[4]<<4 | S[3]<<3 | S[2]<<2 | S[1]<<1 | S[0];

}
bool somador1bit2(bool A,bool B,bool CarryIn){
  bool S = CarryIn^A^B;
  Cy1bit2 = A&&B || CarryIn&&(A^B);
  return S;

}

void somador8bits(bool A[], bool B[]){
  bool S[] = {0,0,0,0,0,0,0,0};
  Cy1bit = 0;
  S[0] = somador1bit(A[0],B[0],0);
  S[1] = somador1bit(A[1],B[1],Cy1bit);
  S[2] = somador1bit(A[2],B[2],Cy1bit);
  S[3] = somador1bit(A[3],B[3],Cy1bit);
  S[4] = somador1bit(A[4],B[4],Cy1bit);
  S[5] = somador1bit(A[5],B[5],Cy1bit);
  S[6] = somador1bit(A[6],B[6],Cy1bit);
  S[7] = somador1bit(A[7],B[7],Cy1bit);
  soma = S[7]<<7 | S[6]<<6 | S[5]<<5 | S[4]<<4 | S[3]<<3 | S[2]<<2 | S[1]<<1 | S[0];
  somaOv = Ov1bit(A[7],B[7], S[7],Cy1bit);
  somaZ = mux2x1(1,0,soma);
  somaCy = Cy1bit;
}
void sub8bits(bool A[], bool B[]){
  bool S[] = {0,0,0,0,0,0,0,0};
  Cy1bit = 1;
  S[0] = somador1bit(A[0],!B[0],Cy1bit);
  S[1] = somador1bit(A[1],!B[1],Cy1bit);
  S[2] = somador1bit(A[2],!B[2],Cy1bit);
  S[3] = somador1bit(A[3],!B[3],Cy1bit);
  S[4] = somador1bit(A[4],!B[4],Cy1bit);
```

```
  S[5] = somador1bit(A[5],!B[5],Cy1bit);
  S[6] = somador1bit(A[6],!B[6],Cy1bit);
  S[7] = somador1bit(A[7],!B[7],Cy1bit);
  sub = S[7]<<7 | S[6]<<6 | S[5]<<5 | S[4]<<4 | S[3]<<3 | S[2]<<2 | S[1]<<1 | S[0];
  subOv = Ov1bit(A[7],!B[7], S[7],Cy1bit);
  subZ =  mux2x1(1,0,sub);
  subCy = Cy1bit;
}
bool somador1bit(bool A,bool B,bool CarryIn){
  bool S = CarryIn^A^B;
  Cy1bit = A&&B || CarryIn&&(A^B);
  return S;

}
bool Ov1bit(bool A, bool B, bool S, bool Cy){
  return A^B^S^Cy;
}
void nor8bits(bool A[], bool B[]){
  bool S[] = {0,0,0,0,0,0,0,0};
  S[7] = NOR1bit(A[7], B[7]);
  S[6] = NOR1bit(A[6], B[6]);
  S[5] = NOR1bit(A[5], B[5]);
  S[4] = NOR1bit(A[4], B[4]);
  S[3] = NOR1bit(A[3], B[3]);
  S[2] = NOR1bit(A[2], B[2]);
  S[1] = NOR1bit(A[1], B[1]);
  S[0] = NOR1bit(A[0], B[0]);
  nor = (S[7]<<7 | S[6]<<6 | S[5]<<5 | S[4]<<4 | S[3]<<3 | S[2]<<2 | S[1]<<1 | S[0]);
  norZ =  mux2x1(1,0,nor);

}
bool NOR1bit(bool A, bool B){
  return (!A&&!B);
}

void printBits(byte myByte){
 for(byte mask = 0x80; mask; mask >>= 1){
   if(mask  & myByte)
       Serial.print('1');
   else
       Serial.print('0');
 }

}
void createDataMemory(){
  for(int i= 0; i <DATA_LENGTHT; i++){
    DataMemory[i]= 0x0;
  }
}

void printValores(){
  if(Serial.available()>0){
    String readInput = Serial.readString();
    if(readInput=="PC" || readInput == "pc"){
      Serial.println(registoPC);
    }
    if(readInput=="V" || readInput == "v"){
      Serial.println(registoV);
    }
    if(readInput=="R" || readInput == "r"){
      Serial.println(registoR);
    }
    if(readInput=="A" || readInput == "a"){
```

```
      Serial.println(registoA);
    }
    if(readInput=="JMP" || readInput == "jmp"){
      Serial.println(JMP);
    }
    if(readInput=="JNC" || readInput == "jnc" || readInput=="JZ" || readInput == "jz" || readInput=="JOV" || readInput == "jov"){
      Serial.println(JNC);
    }
    if(readInput=="Z" || readInput == "z" || readInput == "flagz"){
      Serial.println(registoFlagZ);
    }
    if(readInput=="Cy" || readInput == "cy" || readInput == "flagcy"){
      Serial.println(registoFlagCy);
    }
    if(readInput=="sub"){
      Serial.println(sub);
    }
    if(readInput=="subZ"){
      Serial.println(subZ);
    }
    if(readInput=="subCy"){
      Serial.println(subCy);
    }
    if(readInput=="subOv"){
      Serial.println(subOv);
    }
    if(readInput=="soma"){
      Serial.println(soma);
    }
    if(readInput=="somaZ"){
       Serial.println(somaZ);
     }
     if(readInput=="somaCy"){
       Serial.println(somaCy);
     }
     if(readInput=="somaOv"){
       Serial.println(somaOv);
     }
     if(readInput=="nor"){
       Serial.println(nor);
     }
     if(readInput=="norz"){
       Serial.println(norZ);
     }

  }
}

void teste0(){//som, sub resultado = (8+6 = 14; 24-14 = 10)
  InstructionMemory[0] =  0x6;   // move V,6
  InstructionMemory[1] =  0x140; // mov A,V
  InstructionMemory[2] =  0x8;   // move V, 8
  InstructionMemory[3] =  0x240; // V = V + A
  InstructionMemory[4] =  0x140; // move A,V
  InstructionMemory[5] =  0x18;  // move V,24
  InstructionMemory[6] =  0x280; // V = 24 - 14
}

void teste1(){//nor resultado = 17(0b10001) e teste flag z
  InstructionMemory[0] =  0xAA;  // mov V, 0b10101010
```

```
  InstructionMemory[1] =  0x140; // mov A,V
  InstructionMemory[2] =  0xCC;  // mov V, 0b11001100
  InstructionMemory[3] =  0x200; // V = V nor A
  InstructionMemory[4] =  0x0;   // mov V, 0
  InstructionMemory[5] =  0x140; // mov A,V
  InstructionMemory[6] =  0xFF;  // mov V, 0b11111111
  InstructionMemory[7] =  0x200; // V = V nor A
  InstructionMemory[8] =  0x308; // ( JZ, rel5 = 8)
}

void teste2(){//Data memory write and write
  InstructionMemory[0] =  0x8;   // mov V, 8
  InstructionMemory[1] =  0x1C0; // mov @r, V
  InstructionMemory[2] =  0x10;  // mov V,10
  InstructionMemory[3] =  0x101; // mov R, 1
  InstructionMemory[4] =  0x1C0; // mov @r, V
  InstructionMemory[5] =  0x100; // mov R,0
  InstructionMemory[6] =  0x180; // move V,@r
}

void teste3(){//soma flags test z, cy e ov
  InstructionMemory[0] =  0x240; // V = V + A (0+0)
  InstructionMemory[1] =  0x80;  // V = 0b10000000
  InstructionMemory[2] =  0x140; // mov A,V
  InstructionMemory[3] =  0x240; // V = V + A
  InstructionMemory[4] =  0x348; // (jov, rel5=8)
  InstructionMemory[12] = 0x2C8; // (j!Cy, rel5=8)
  InstructionMemory[13] = 0x380; // (jump 0)
}
void teste4(){//sub, flags test z,cy, ov
  InstructionMemory[0] =  0xFF;  // V = 0b11111111
  InstructionMemory[1] =  0x140; // mov A,V
  InstructionMemory[2] =  0x280; // V = V - A
  InstructionMemory[3] =  0x7F;  // V = 0b1111111
  InstructionMemory[4] =  0x280; // V = V - A
  InstructionMemory[5] =  0x2C8; //(j!Cy, rel5 = 8)
  InstructionMemory[13] =  0x380; //(jump 0)
}

byte mux2x1(byte A, byte B, bool C){
  return (!C)? A:B;
}
byte mux4x1(byte A, byte B, byte D, byte E, bool C1, bool C0){
  int mux1 = mux2x1(A,B,C0);
  int mux2 = mux2x1(D,E,C0);
  return mux2x1(mux1,mux2, C1);
}
void imprimirRegistos(){
  Serial.println("Processador");
  Serial.print("PC: ");
  Serial.println(registoPC);
  Serial.print("V: ");
  Serial.println(registoV);
  Serial.print("A: ");
  Serial.println(registoA);
  Serial.print("R: ");
  Serial.println(registoR);
  Serial.print("Codigo: ");
  if((instrucao>>8)==0){
```

```
      Serial.println("0");
    }else{
      Serial.println(instrucao>>6,BIN);
    }

    Serial.println("RAM");
    Serial.print("1 Posicao: ");
    Serial.println(DataMemory[0]);
    Serial.print("2 Posicao: ");
    Serial.println(DataMemory[1]);
    Serial.print("Flag Ov: ");
    Serial.println(registoFlagOV);
    Serial.print("Flag Zero: ");
    Serial.println(registoFlagZ);
    Serial.print("Flag Cy: ");
    Serial.println(registoFlagCy);
    Serial.println();

}
```