

Licenciatura Engenharia Informática e Multimédia

Computação Física – CF

Relatório Trabalho Prático 2

Docente Carlos Carvalho

 Trabalho realizado por:

 Fábio Dias, nº 42921

Tatiana Cristão, nº 47508

Índice

Índice de Figuras

[Figura 1 - Arquitetura Harvard 5](file:///C:\Users\fabio\Desktop\ISEL\Semestre%202\CF\Trabalhos\TP2\Relatório_TP2.docx#_Toc104137907)

[Figura 2 - Módulo Funcional 7](#_Toc104137908)

Índice de Tabelas

1. Desenho do Microprocessador

Para este trabalho prático, foi-nos pedido para desenhar um microprocessador, baseando-nos na arquitetura Harvard e simulá-lo no Arduino.

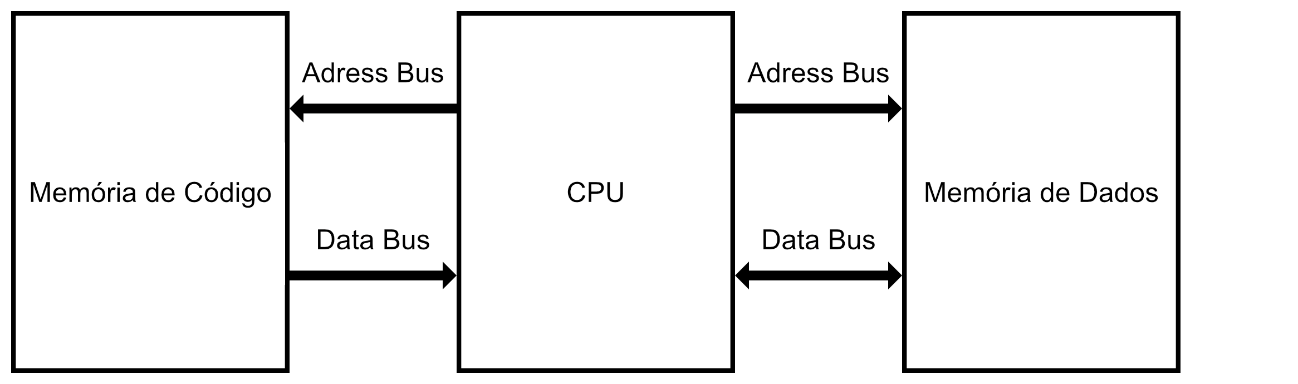
A arquitetura Harvard é composta pela Memória de Código, Memória de Dados e pelo CPU. Ambas as memórias possuem um Adress Bus e um Data Bus. O Data Bus da Memória de Código deve ser apenas de leitura e o Adress Bus deve ser apenas de escrita. O Adress Bus da Memória de Dados também deve ser apenas de escrita mas, o Data Bus da Memória de Dados pode ser de escrita e de leitura, mas apenas um destes estados está ativo . *(Ver Figura 1).*

Figura 1 - Arquitetura Harvard

A Memória de Código é onde o programa é cumprido. Este é apenas de leitura exceto quando queremos outro programa. Nesse caso, a memória tem de ser regravada.

A Memória de Dados possui as variáveis do programa.

O CPU, *Central Processing Unit*, onde o programa é cumprido.

Cada memória possui um Data Bus e um Adress Bus.

A partir do enunciado, conseguimos especificar o Registo *A*, um conjunto de Registos *Rn*, constituído por dois registos, *R0* e *R1*, o Registo de Controlo de Execução, *PC*, as *flags Carry* e *Borrow*, *Cy*, *Overflow*, *Ov*, e *Zero*, *Z*.

Analisando as instruções da tabela do enunciado, conseguimos concluir que o Registo PC, na última instrução, toma um valor de sete bits. Esta é a dimensão máxima que PC pode ter, dado que as outras instruções de controlo de fluxo são a seis bits. Logo, o Registo PC tem uma dimensão de sete bits. Que corresponde ao Adress Bus da Memória de Código. Na primeira instrução movemos para um dos Registos R, o valor de uma constante de cinco bits, logo, os Registos Rn têm uma dimensão de cinco bits que corresponde ao Adress Bus da Memória de Dados. Isto porque o Adress Bus da Memória de Dados tem a mesma dimensão do registo que endereça a memória que, pelas instruções do enunciado, é um dos Registos Rn. Na segunda instrução, movemos para o Registo *A* o valor de um dos Registos R. Logo, também tem uma dimensão de cinco bits e este corresponde ao Data Bus da Memória de Dados. Para obtermos a dimensão do Data Bus da Memória de Código é necessário codificarmos cada uma das instruções, de forma a distingui-las e incluirmos todos os seus parâmetros.

Uma imagem com texto, palavras cruzadas, bateria, recibo

Descrição gerada automaticamente Seguindo para a codificação das instruções, como referido previamente, o objetivo é conseguimos distinguir todas as instruções, incluindo os parâmetros das instruções em causa. Conseguimos atingir este objetivo com dez bits, dos quais, quatro são de distinção. Estes são o D9 D8, D1 e D0. Assim concluímos que o Data Bus da Memória de Código tem uma dimensão de dez bits.

Tabela 1 - Codificação das Instruções

Como podemos observar na tabela, não existe nenhuma combinação repetida com os quatro bits de distinção. Para efectuarmos esta tabela, simplesmente identificamos os parâmetros, caso existam, e colocamos a sua dimensão de forma modular, como é possível identificar nas três linhas finais, por exemplo. De seguida identificamos os bits de distinção e tentamos preencher a tabela até chegarmos a valores únicos, ou seja, não repetidos. [ISTO PARECE ESTAR TÃO MAL EXPLICADO…]

De seguida partimos para o desenho do módulo funcional. Começando pelo Registo de Controlo de Execução, o Registo PC, este encontra-se a 0. No bloco somatório, vai receber o valor presente, ou seja, 0, e um valor que provém do multiplexer acima, Y1. Este é entre 1 ou um número relativo proveniente do Data Bus que é um número relativo, a 6 bits mas, dado que PC é a sete, este valor necessita que o sinal seja extendido repetindo o bit de maior peso.. primeiro mutiplexer é deparado com [DESENVOLVER CADA PEQUENO COMPONENTE.]

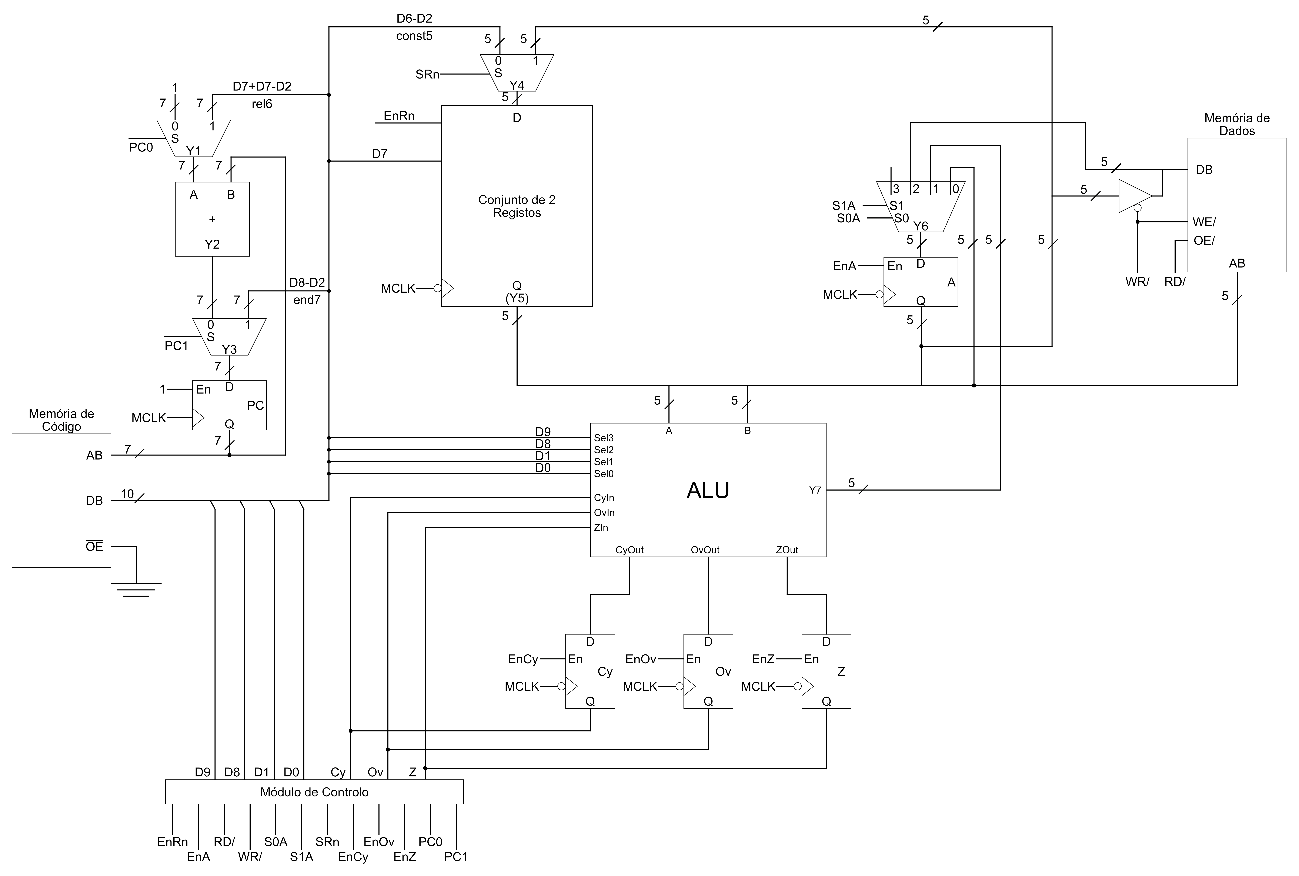


Figura 2 - Módulo Funcional

Observando cada instrução, podemos obter uma especificação das entradas do módulo de controlo, assim como as suas saídas [REFERENCIAR IMAGEM E ENUMERÁ-LAS E DETALHA-LÁS]

[DESENHAR E DESCREVER O PC0 SER UM CONJUNTO DE PORTAS LÓGICAS. AQUELAS APRESENTADAS PELO ENGENHEIRO]

[TAMBÉM TENHO DE IMPLEMENTAR ISSO NO AFETARSINAIS DO ARDUINO]

[MODULO FUNCIONAL ERRADO. CADA FLAG (CY OV Z) TEM O SEU PROPRIO ENABLE]

[MODULO FUNCIONAL ERRADO. ENPC0 NÃO EXISTE. EXISTE APENAS PC0. ASSIM COMO ENPC1 NÃO EXISTE. EXISTE PC1]

De seguida, partimos para a tabela de verdade do Módulo de Controlo, onde identificamos as saídas ativas em cada instrução. [REFERENCIAR IMAGEM]

Finalmente, adicionamos a ROM e definimos a gama de endereços para cada instrução. [REFERENCIAR IMAGEM E COMO OBTEMOS A GAMA]

[TENHO DE REFERENCIAR QUE O CLK É DIVIDIDO EM DOIS]

2. Implementação do Microprocessador

Dado que o CLK é manual, com auxílio a um botão, definimos o pino do botão e, como foi feito no trabalho anterior, definimos duas variáveis de controlo de tempo para evitar o *debounce*. Este será definido como *INPUT\_PULLUP*. [REFEERENCIAR IMAGEM]

Com base no desenho do microprocessador, estabelecemos que possuímos uma memória de Código, *ROM*, e uma Memória de Dados, *RAM*, sendo estas simuladas a partir de array’s. Possuem, respectivamente, X e Y índices. [EXPLICAR PORQUE – Acho que é explicado em cima mas temos de ver]

Teremos também as saídas do Módulo de Controlo, num array de dezasseis bits, designado por *ROM\_MC*. Este é inicializado e preenchido manualmente. [REFERENCIAR IMAGEM]

Adicionalmente, definimos os registos. Temos de definir duas variáveis para cada um. Uma entrada, *D*, e uma saída, *Q*, que só é afetada quando existir *enable*. Como são apenas afetadas na transição ascendente ou descendente do CLK, as saídas têm de ser *volatile*. [REFERENCIAR A IMAGEM]

De seguida, definimos as variáveis que correspondem às saídas do Módulo de Controlo. Estas só possuem um bit, logo serão *boolean*’s. [REFERENCIAR IMAGEM]

Como foi apresentado previamente no Módulo Funcional, existem saídas designadas por *Y*. No código, também as vamos definir de forma a facilitar a organização do módulo combinatório. [REFERENCIAR IMAGEM]

Dado que o Bloco de Registos vai possuir dois registos com *enable*’s próprios, vamos ter de definir estas duas variáveis. [REFERENCIAR IMAGEM]

Na implementação, foi possível observar que iriamos ter de obter o conteúdo da Memória de Código da instrução atual. De forma a facilitar o seu acesso, foi definida uma variável global que é afetada com essa instrução. [REFERENCIAR IMAGEM]

Começaremos por definir todas as estruturas que vamos precisar, com base no desenho do Módulo Funcional. Vamos precisar de Multiplexer’s, tanto dois para um como quatro para dois, Registos com Enable, para *bytes* e para *boolean*’s, e também um Demultiplexer. [REFERENCIAR IMAGENS DE CADA UM DESTES COMPONENTES]

Vamos depender de duas transições do CLK, da ascendente e da descendente. Desta forma, foram criados dois métodos. De forma a manter coerência, quando um dos CLK’s é invocado, via uma interrupção, esta interrupção é redefinida para o CLK contrário e o seu modo também. [REFERENCIAR IMAGEM]

Passamos para o método de *AfetarSinais*, esta pega no código actual da Memória de Código, isola e agrega os bits de distinção, que servirá de índice para os sinais de saída do Módulo de Controlo, *ROM\_MC*, afeta cada um dos sinais. [REFERENCIAR IMAGEM]

Temos também o método simulador do Bloco de Registos, *BlocoDeRegistos*. Obtemos o valor do selector do registo, pelo sétimo bit da instrução actual, obtemos os valores dos *enable*’s dos registos, ER0 e ER1 e afetamos os registos e, de seguida, com passamos ambos os valores por um MUX\_2x1, cujo selector é o sétimo bit da instrução actual.

De seguida, implementá-los o bloco do ALU, que, devido à nossa codificação, recebe quatro selectores, *S\_0*, *S\_1*, *S\_2* e *S\_3*, recebe dois bytes, *A* e *B*, recebe os valores das *flags*, *CyIn*, *OvIn* e *ZIn*, e, por referência, as variáveis correspondentes a *CyOut*, *OvOut*, *ZOut* e *yOut*.

É também implementado o tri-state, *TriState*, que grava o que está no registo A, na Memória de Dados. Confirma-se que o sinal *RD*, está activo, ou seja a zero, e, nesse caso, procede a escrita.

Para simular instruções e valores guardados, foram criadas duas funções que preenchem a *ROM* e a *RAM*, com auxílio a função *random*.

Finalmente, chegamos ao correr do programa. Na função *setup* iniciamos a comunicação com o arduíno, via o método *Serial.begin()*, definimos o pino do botão que servirá de CLK como *INPUT\_PULLUP* no método *pinMode*. Chamamos as funções de preenchimento da Memória de Código e da Memória de Dados, adicionamos o interrupt e ligamo-la.

Chegando ao *loop*, invocamos toda a lógica para as funções combinatórias. Ou seja, tudo menos os registos. [DETALHAR]

De forma a analisarmos os resultados, foi necessário criar métodos de *Debug.* O utilizador pode interagir com a consola, permitindo observar cada componente deste microprocessador [MELHORAR O PARÁGRAFO E DETALHAR]