Logo

Description automatically generated

**Licenciatura em Engenharia Informática e de Computadores**

Relatório do 2º Trabalho

\*

Estudo do funcionamento de um processador

Trabalho realizado por:

Romário Dias Nº 50083

Iliano Santos Nº 50096

Turma: 26D

Docente: Tiago Dias

Arquitetura de Computadores

2022 / 2023 verão

21 de abril de 2023

**ÍNDICE**

[1 Introdução 3](#_Toc133064908)

[2 Descrição do processador em estudo 3](#_Toc133064909)

[3 Análise da microarquitetura do processador 3](#_Toc133064910)

[4 Codificação das instruções do processador 4](#_Toc133064911)

[5 Projeto do descodificador de instruções (*Instruction Decoder*) 5](#_Toc133064912)

[6 Codificação de programas em linguagem máquina 6](#_Toc133064913)

[7 Conclusões 8](#_Toc133064914)

# Introdução

O objetivo principal deste trabalho é compreender e demonstrar o funcionamento de um processador de um computador (neste trabalho foi estudado a codificação de um processador a 8 bits com instruções codificadas a 9 bits) onde abordamos os seguintes tópicos:

-Codificação de um conjunto de instruções;

-Funcionamento de uma microarquitetura;

-Codificação de programas em linguagem máquina.

# Descrição do processador em estudo

O processador considerado é de 8 bits, espaço de endereçamento para código e dados com 256 endereços; oito registos de uso geral e um registo de estado, denominado CPSR, possuí uma arquitetura de ciclo único, um bloco *Instruction Memory*, um *Register File*, uma ALU para realizar operações entre registos e uma unidade *Data Memory*.

Uma imagem com diagrama

Descrição gerada automaticamente

Figura 1: microarquitetura do processador

# Análise da microarquitetura do processador

**1 "A microarquitetura do processador é do tipo Harvard."**

É dito que a arquitetura deste processador é do ciclo único, o que implica que a sua microarquitetura é do tipo Harvard, ou seja, os barramentos de acesso (endereço, dados e controlo) são independentes, possuí espaços de memória independentes para código e dados, como demonstra a figura 1.

**2** O bloco **Ext**, ou seja, “Extensão de Sinal", é um bloco de um circuito que pode ser usado em processadores para estender um valor de bits ou palavras de bits, com ou sem sinal para uma palavra de bits, este que terá um tamanho de bits superior ao número de bits definido na entrada do bloco. Neste caso, se o sinal de controlo **SE** for 1 numa instrução de controlo do tipo **bne**, queremos estender uma palavra de 6 para 8 bits, correspondente à label. Enquanto, se o sinal de controlo **SE** for 0 numa instrução do **mov** queremos estender uma palavra de 3 para 8 bits, correspondente a uma constante imm3.

Por exemplo, na **figura 1** é usado para fazer a extensão de 6 para 8 bits dos valores codificados.

**3** A instrução **bne label** utiliza o modo de endereçamento imediato, enquanto a instrução **b rn** utiliza o modo de endereçamento por registo. Por sua a instrução de salto condicional **bne label**, permite mudar o fluxo de execução do programa tendo em conta um valor de deslocamento designado pelo offset, correspondente à label, o que quer dizer que utilizamos uma referência para determinar o fluxo de execução do programa, o que não se verifica na instrução **b rn** onde fazemos sempre um salto incondicional baseado no endereço definido por um registo e tendo em conta deixámos que poder utilizar esse registo para uma outra operação, pois neste caso temos uma escassez de registos de r0 a r7, onde r7 é o *stack pointer*.

# Codificação das instruções do processador

**1** A partir da codificação da instrução **add** dada pelo enunciado, definimos a codificação para as restantes instruções, sabendo de antemão que o campo *opcode* deverá corresponder aos 3 bits menos significativos, e o restante a 6 bits depende da instrução identificada pelo *opcode*. Sabendo que temos 8 registos (r0-r7), portanto, utilizaremos 3 bits para identificar o registo, e a *label* a 6 bits, pois precisamos que esta represente endereços na vizinhança de +/- 32 endereços.

* **Mapa de codificação**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Instruções | b8 | b7 | b6 | b5 | b4 | b3 | b2 | b1 | b0 |
| **Add rd, rn** | **rd** | | | **rn** | | | **opcode** | | |
| b rn | - | | | rn | | | opcode | | |
| bne label | label | | | | | | opcode | | |
| cmp rn, rm | rn | | | rm | | | opcode | | |
| ldr rd, [rn] | rd | | | rm | | | opcode | | |
| mov rd, #imm3 | rd | | | #imm3 | | | opcode | | |
| push rn | rn | | | - | | | opcode | | |
| str rd, [rn] | rd | | | rn | | | opcode | | |

\* Os símbolos ‘-‘ representam *dont’t care*, ou seja, esses valores não interessam na codificação, pois não têm significado na instrução a que referem.

**2** Sabendo que o campo *opcode* para a instrução add é 100, definimos uma codificação para as restantes instruções, tendo em conta o funcionamento da ULA, nomeadamente nas operações que ela realiza.

O raciocínio para a resolução deste problema é descrito por este esquema:

Uma imagem com logótipo

Descrição gerada automaticamente

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Instruções | b8 | b7 | b6 | b5 | b4 | b3 | b2 | b1 | b0 |
| **add rd, rn** | **rd** | | | **rn** | | | **1** | **0** | **0** |
| b rn | - | | | rn | | | 1 | 1 | 0 |
| bne label | label | | | | | | 1 | 1 | 1 |
| cmp rn, rm | rn | | | rm | | | 1 | 0 | 1 |
| ldr rd, [rn] | rd | | | rn | | | 0 | 0 | 0 |
| mov rd, #imm3 | rd | | | #imm3 | | | 0 | 1 | 1 |
| push rn | rn | | | - | | | 0 | 1 | 0 |
| str rd, [rn] | rd | | | rn | | | 0 | 0 | 1 |

# Projeto do descodificador de instruções (*Instruction Decoder*)

**1**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Instruções | opcode | | | Z | SO | SI | SS | SE | SD | EP | ER | nRD | nWR |
| **add rd, rn** | **1** | **0** | **0** | - | 0 | 0 | 00 | - | 01 | 1 | 1 | 1 | 1 |
| b rn | 0 | 0 | 0 | - | 0 | 1 | 00 | - | 00 | 0 | 0 | 1 | 1 |
| bne label | 0 | 0 | 1 | 0 | 1 | 0 | 00 | 1 | 00 | 0 | 0 | 1 | 1 |
| 1 | 0 | 0 | 00 | - | 00 | 0 | 0 | 1 | 1 |
| cmp rn, rm | 1 | 0 | 1 | - | 0 | 0 | 00 | - | 01 | 1 | 0 | 1 | 1 |
| ldr rd, [rn] | 0 | 1 | 1 | - | 0 | 0 | 00 | - | 10 | 0 | 1 | 0 | 1 |
| mov rd, #imm3 | 1 | 1 | 0 | - | 0 | 0 | 00 | 0 | 00 | 0 | 1 | 1 | 1 |
| push rn | 0 | 1 | 0 | - | 0 | 0 | 11 | - | 10 | 0 | 0 | 1 | 0 |
| str rd, [rn] | 0 | 1 | 1 | - | 0 | 0 | 00 | - | 10 | 0 | 0 | 1 | 0 |

**2** Implementação de uma memória de com porto de endereços dados pelo opcode e a flag Z e o porto de dados que representa os sinais SO, SI, SS, SE, SD, EP, ER, nRD e nWR.

Uma imagem com diagrama

Descrição gerada automaticamente

**3** A capacidade da memória ROM é de (24x 11) 176 bits.

# Codificação de programas em linguagem máquina

Uma imagem com mesa

Descrição gerada automaticamente

Figura 2: Troço de código em linguagem *assembly* do processador

**1** O troço de código carrega valores iniciais para registos a 8 bits e implementa um ciclo onde ao registo r1 adiciona o valor da posição de memória referenciada por r0, incrementa um valor a r0 e repete o ciclo até que seu valor seja 4. Quando r0 = #4, o programa escreve no endereço 0x04, dado por r2, a soma dos valores guardados em posições de memória inferiores a 0x04, e em seguida, faz um salto infinito para a própria instrução, pois ao realizar a operação 6+6 = 12 = 0x0C, este é o endereço da última instrução do programa.

|  |  |  |
| --- | --- | --- |
| Instrução | Endereço | Código máquina |
| mov r0, # 0 | 0x00 | 000 |
| mov r1, # 0 | 0x01 | 040 |
| mov r2, # 4 | 0x02 | 0A0 |
| ldr r3, [ r0 ] | 0x03 | 0C1 |
| add r1, r3 | 0x04 | 05C |
| mov r4, # 1 | 0x05 | 108 |
| add r0, r4 | 0x06 | 024 |
| cmp r0, r2 | 0x07 | 015 |
| bne loop | 0x08 | 1D7 |
| str r1, [ r2 ] | 0x09 | 052 |
| mov r5, # 6 | 0x0A | 170 |
| add r5, r5 | 0x0B | 16C |
| b r5 | 0x0C | 02E |

**2**

# Conclusões

O trabalho realizado consistiu no estudo do funcionamento de um processador a 8 bits, onde adquirimos mais experiência na codificação, consolidando a matéria neste capítulo. Vale salientar que é importante levar em conta a arquitetura do processador em todos os aspetos, como por exemplo, o número de ciclos, os modos de endereçamento, os registos, a capacidade da memória, este que é o cérebro do computador e contribui para o seu melhor desempenho. Importante também dizer que arranjar formas de trabalhar com programas de maior legibilidade e compreensão pelos humanos é uma tarefa que deve ser procurada de forma a proporcionar uma maior produtividade para quem trabalha com processadores no seu dia-a-dia.