

TRABALHO DA DISCIPLINA DE ARQUITETURA DE COMPUTADORES

IMPLEMENTAÇÃO DE UM PROCESSADOR SINGLE CYCLE COM LINGUAGEM SYSTEMVARILOG

ALUNO: PAULO RAFAEL BORGES DE OLIVEIRA (20171CECA70099)

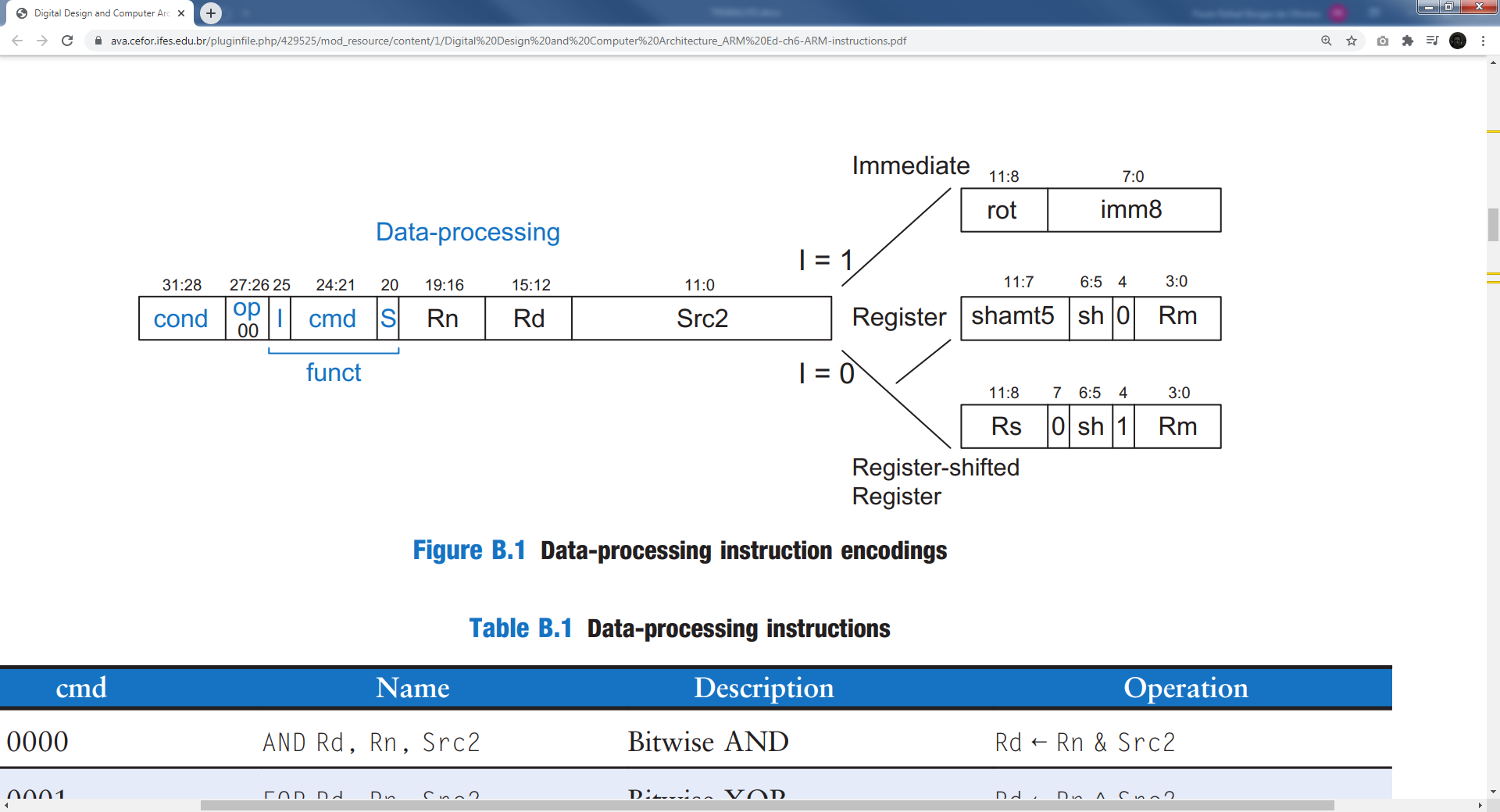
2020

**OBJETIVO**

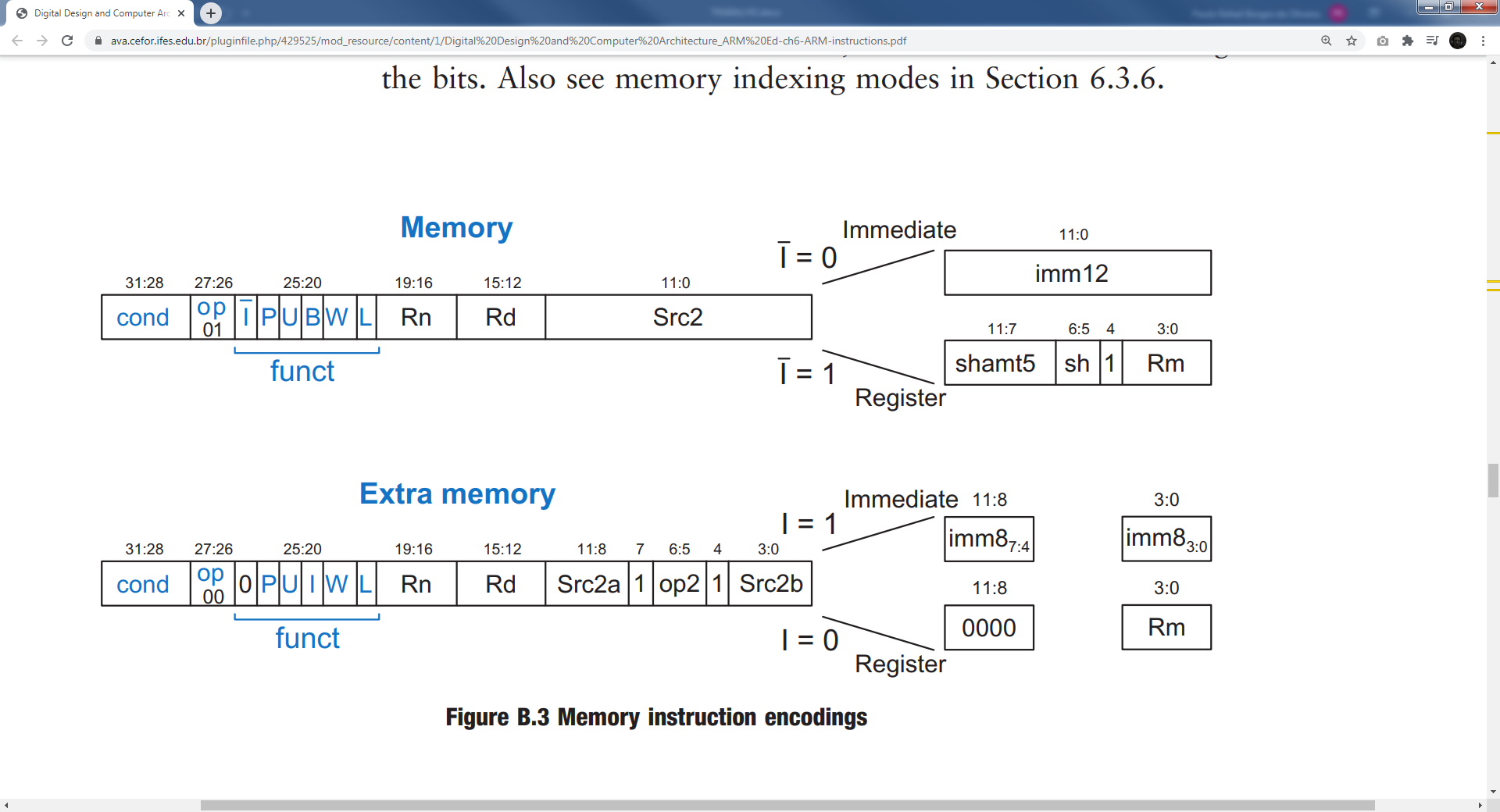
Estender instruções de um processador single cycle da arquitetura ARM com linguagem descritiva de hardware SystemVarilog e analisar via ambiente de simulação (MULTSIM) todas as portas, entradas e saídas do modelo a fim de comparar as instruções estendidas com o comportamento do hardware.

**MODELOS DA ARQUITETURA ARM**

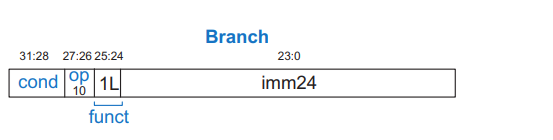
Para instruções do tipo **DATA-PROCESSING** temos a seguinte organização do frame:



Para instruções do tipo **MEMORY** temos a seguinte organização do frame:



Para instruções do tipo **BRANCH** temos a seguinte organização do frame:



**IMPLEMENTAÇÃO**

Serão implementas 4 instruções dentre as possibilidades: TST, CMP, LSL, MOV, EOR e LDRB. Optou-se pela implementação das instruções TST, CMP, LSL e EOR.

Inicialmente foram incluídas as instruções já implementadas a fim de verificar o funcionamento no simulador, sendo elas:

//SUB R4, R15, R15 E04F400F

//ADD R5, R4, #0 E2845000

//ADD R3, R4, #10 E284300A

//STR R3, [R5, #4] E5853004

//LDR R4, [R5, #4] E5954004

//ADD R5, R5, #5 E2855005

//AND R6, R4, R5 E0046005

//ORR R7, R4, R5 E1847005

//SUB R8, R4, R5 E0448005

Após, foi implementada as instruções conforme o seguinte:

**1) INSTRUÇÃO EOR**

EOR [CMD: 0001, S=0] : Bitwise XOR – RD <- RN ^ Src2

EOR RD, RN, SRC2

A título de exemplo foi usado o seguinte comando:

//EOR R9, R4, R5 E0249005

Que possui representação em hexadecimal E0249005, sendo sua forma binária o seguinte:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| COND | | | | OP | | FUNCT | | | | | | RN | | | | RD | | | | SRC2 | | | | | | | | | | | |
| I | CMD | | | | S |  | | | | IMM8 | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 09 | 08 | 07 | 06 | 05 | 04 | 03 | 02 | 01 | 00 |
| 1 | 1 | 1 | 0 | 0 | 0 | 0 | **0** | **0** | **0** | **1** | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 |

Considerando que todos os elementos do frame já estão implementados (COND, OP, I, S, RN, RD e SRC2), torna-se necessário apenas a alteração no decoder para que a instrução 0001 seja reconhecida e envie comando para a ULA realizar a operação correspondente.

**2) INSTRUÇÃO TST**

TST [CMD: 1000, S=1] : Seta flags (N, Z, C) baseadas no RN & Src2 e não atualiza o registrador

TST RD, RN, SRC2

A título de exemplo foi usado o seguinte comando:

//TST R4, R5 E1140005

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| COND | | | | OP | | FUNCT | | | | | | RN | | | | RD | | | | SRC2 | | | | | | | | | | | |
| I | CMD | | | | S |  | | | | IMM8 | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 09 | 08 | 07 | 06 | 05 | 04 | 03 | 02 | 01 | 00 |
| 1 | 1 | 1 | 0 | 0 | 0 | 0 | **1** | **0** | **0** | **0** | **1** | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 |

**3) INSTRUÇÃO CMP**

CMP [CMD: 1010, S=1] : Seta flags baseadas no RN - Src2

CMP RN, SRC2

A título de exemplo foi usado o seguinte comando:

//CMP R4, R5 E1540005

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| COND | | | | OP | | FUNCT | | | | | | RN | | | | RD | | | | SRC2 | | | | | | | | | | | |
| I | CMD | | | | S |  | | | | IMM8 | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 09 | 08 | 07 | 06 | 05 | 04 | 03 | 02 | 01 | 00 |
| 1 | 1 | 1 | 0 | 0 | 0 | 0 | **1** | **0** | **1** | **0** | **1** | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 |

**4) INSTRUÇÃO LSL**

LSL [CMD: 1101 I:0 and sh=00]– Logical Shift Left – RD <- RM << SRC2

LSL RD, RM RS/SHAMT5

A título de exemplo foi usado o seguinte comando:

//E1A0C384 - LSL R12, R4, #7

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| COND | | | | OP | | FUNCT | | | | | | RN | | | | RD | | | | SRC2 | | | | | | | | | | | |
| I | CMD | | | | S | SHAMT5 | | | | | SH | | 0 | RM | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 09 | 08 | 07 | 06 | 05 | 04 | 03 | 02 | 01 | 00 |
| 1 | 1 | 1 | 0 | 0 | 0 | 0 | **1** | **1** | **0** | **1** | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |

A implementação gerou os seguintes resultados:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Ciclo | Reset | PC | Instrução (Hex) | Instrução Assembly | SrcA | SrcB | Branch | AluResult | Flags[3:0] | CondEx | WriteData | MemWrite | ReadData |
| ... | 1 | 0 | E04F400F | SUB R4, R15, R15 | 8 | 8 | 0 | 0 | 0000 | 1 | 8 | 0 | X |
| 0 | 0 | 0 | E04F400F | SUB R4, R15, R15 | 8 | 8 | 0 | 0 | 0000 | 1 | 8 | 0 | X |
| 1 | 0 | 4 | E2845000 | ADD R5, R4, #0 | 0 | 0 | 0 | 0 | 0000 | 1 | X | 0 | X |
| 2 | 0 | 8 | E284300A | ADD R3, R4, #10 | 0 | 10 | 0 | 10 | 0000 | 1 | X | 0 | X |
| 3 | 0 | 12 | E5853004 | STR R3, [R5, #4] | 0 | 4 | 0 | 4 | 0000 | 1 | 10 | 1 | X |
| 4 | 0 | 16 | E5954004 | LDR R4, [R5, #4] | 0 | 4 | 0 | 4 | 0000 | 1 | 0 | 0 | 10 |
| 5 | 0 | 20 | E2855005 | ADD R5, R5, #5 | 0 | 5 | 0 | 5 | 0000 | 1 | 0 | 0 | 10 |
| 6 | 0 | 24 | E0046005 | AND R6, R4, R5 | 10 | 5 | 0 | 0 | 0000 | 1 | 5 | 0 | X |
| 7 | 0 | 28 | E1847005 | ORR R7, R4, R5 | 10 | 5 | 0 | 15 | 0000 | 1 | 5 | 0 | X |
| 8 | 0 | 32 | E0448005 | SUB R8, R4, R5 | 10 | 5 | 0 | 5 | 0000 | 1 | 5 | 0 | 10 |
| 9 | 0 | 36 | E0249005 | EOR R9, R4, R5 | 10 | 5 | 0 | 15 | 0000 | 1 | 5 | 0 | X |
| 10 | 0 | 40 | E1140005 | TST R4, R5 | 10 | 5 | 0 | 0 | 0000 | 1 | 5 | 0 | X |
| 11 | 0 | 44 | E1540005 | CMP R4, R5 | 10 | 5 | 0 | 5 | 0100 | 1 | 5 | 0 | 10 |
| 12 | 0 | 48 | E1A0C384 | LSL R12, R4, #7 | 7 | 10 | 0 | 1280 | 0010 | 1 | 10 | 0 | X |

Avaliando os resultados verifica-se que as implementações mostraram resultados coerentes com o esperado.

O esquemático completo consta conforme abaixo

Diagrama, Desenho técnico

Descrição gerada automaticamente

ARM = CONTROL + DATAPATH

DATA MEMORY

INSTRUCTION MEMORY

CONTROL

DATAPATH