|  |  |  |  |
| --- | --- | --- | --- |
| Vitor Mexia | 36070 |  | 1º Trabalho Prático |
|  |  |  | 18.04.2020 |
|  |  |  | Grupo 12 |
|  |  |  |  |

Objetivos

Tendo em conta o a descriçao da arqutectura do processador do enunciado, o obejctio deste trabalho é completae o projecto do processador completando as instrunções ISA, tendo em conta o esquema da ALU sendo que as operações da ALU fazem parte do OPCODE do ISA, e fazer o respectivo mapa de codificação de instruções.

Ao termos o mapa de codificação de instruções iremos obter o valor lógico das saidas do subcircuito Intruction Decoder e com isso o conteudo da ROM utilizada postriormente no Logisim.

Com o mapa de codificação do Instruction Decoder iremos codificar a sequência de instruções fornecidae carregar a mesma no logisim. Iremos depois testar as instruções demodo a validar todas as codificações feitas até aqui avaliando o que acontece no logisim em cada clock.

Codificação das instruções do ISA

1. Para completar os *opcodes* tivémos de analisar o esquema da ALU fornecido visto que as operações da ALU são defenidas pelo bit 0 e o bit 1 do respectivo *opcode*. A nossa análise da ALU concluiu que as operções da ALU são as seguintes:

|  |  |  |
| --- | --- | --- |
|  | **OP\_ALU** | |
| **Operações** | **bit 1** | **bit 0** |
| **Add** | 0 | 0 |
| **Sub** | 0 | 1 |
| **Shift** | 1 | 0 |
| **OR** | 1 | 1 |

***Tabela de codificação das instruções***

Com esta informação completámos a tabela de *opcodes* fornecida.

NOTA: A operação CMP (compare) é uma subtracção e a igualdade é fornecida pela flag Z, ou seja, a subtracção de dois registos é igual a zero.

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **ISA** | **OPCODE** | | | **AD** | | | **AA** | | | **AB** | | |
| **11** | **10** | **9** | **8** | **7** | **6** | **5** | **4** | **3** | **2** | **1** | **0** |
| **add rx, ry, rz** | 0 | 0 | 0 | rx | | | ry | | | rz | | |
| **cmp rx, ry** | 0 | 0 | 1 | - | - | - | rx | | | ry | | |
| **lsr rx, ry, imm3** | 0 | 1 | 0 | rx | | | ry | | | imm3 | | |
| **mov rx, imm 6** | 0 | 1 | 1 | rx | | | imm6 | | | | | |
| **ldr rx, [ry, imm3]** | 1 | 0 | 0 | rx | | | ry | | | imm3 | | |
| **bne rx** | 1 | 0 | 1 | - | - | - | rx | | | - | - | - |
| **b offset8** | 1 | 1 | 0 | - | offset8 | | | | | | | |
| **str rx, [ry]** | 1 | 1 | 1 | rx | | | - | - | - | ry | | |

*Tabela 1 -Tabela de codificação das instruções*

Descodificador de instruções

Analisando o diagrama de blocos do processador e do banco de registos chegámos à seguinte tabela de instruções e o respectivo contiúdo da ROM (coluna PRG).

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Instruction Decoder** | **OPCODE** | | | **Z** | **SI** | **SO** | **SD** | | **SC** | **SA** | **ER** | **EP** | **WR** | **RD** | **HEX** | **PRG** |
| **3** | **2** | **1** | **0** | **9** | **8** | **7** | **6** | **5** | **4** | **3** | **2** | **1** | **0** |
| **add rx, ry, rz** | 0 | 0 | 0 | - | 0 | 0 | 1 | 0 | 1 | - | 1 | 1 | 0 | - | 0AC | 2\*0AC |
| **cmp rx, ry** | 0 | 0 | 1 | - | 0 | 0 | - | - | 1 | - | 0 | 1 | 0 | - | 024 | 2\*024 |
| **lsr rx, ry, imm3** | 0 | 1 | 0 | - | 0 | 0 | 1 | 0 | 0 | - | 1 | 1 | 0 | - | 08C | 2\*08C |
| **mov rx, imm 6** | 0 | 1 | 1 | - | 0 | 0 | 0 | 0 | - | - | 1 | 0 | 0 | - | 008 | 2\*008 |
| **ldr rx, [ry, imm3]** | 1 | 0 | 0 | - | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | 049 | 2\*049 |
| **bne rx** | 1 | 0 | 1 | 0 | 1 | - | - | - | - | - | 0 | 0 | 0 | - | 200 | 200 |
| **bne rx** | 1 | 0 | 1 | 1 | 0 | 0 | - | - | - | - | 0 | 0 | 0 | - | 000 | 000 |
| **b offset8** | 1 | 1 | 0 | - | 0 | 1 | - | - | - | - | 0 | 0 | 0 | - | 100 | 2\*100 |
| **str rx, [ry]** | 1 | 1 | 1 | - | - | - | - | - | - | 1 | 0 | 0 | 1 | 0 | 012 | 2\*012 |

*Tabela 2 -Tabela do descodificador de instruções*

NOTA: Assumimos neste caso que os *don’t cares*  tomam valor zero.

Ficheiro a carregar na ROM do Descodificador



*FICHEIRO 1 – Ficheiro de carregamento na ROM do Descodificador*

Teste da arquitetura

Analisando a sequência de instruções, concluimos que estamos a fazer um ciclo DO-WHILE onde a condição de paragem é termos r0 = 4. A r0 é incrementado 1 (valor de r4) a cada ciclo . Dentro do ciclo, adicionamos a r1 os conteudos de memória da posição r0 (00h, 01h, 02h, 03h), guardados em r3 no inicio de cada ciclo. Ao sair do ciclo, r0 = 4, o valor de r1 é shiftado 2 vezes para a direita. Como estamos a trabalhar com numeros naturais, cada shift para a direita resulta de uma divisão por 2 descartando o resto, logo temos (r1/2)/2. De seguida guardamos r1 na posição 04h (r2) da RAM finalizando o programa.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  | **OPCODE** | | | **AD** | | | **AA** | | | **AB** | | |  |
|  |  |  |  | **11** | **10** | **9** | **8** | **7** | **6** | **5** | **4** | **3** | **2** | **1** | **0** |  |
| **Code Address** | **MNEMONICS** | | | **INSTRUCTION** | | | | | | | | | | | | |
| **LABEL** | **OPCODE** | **OPERANDS** | **BIN** | | | | | | | | | | | | **HEX** |
| 00 | r0 = 0 | **mov** | **r0, 0** | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 600 |
| 01 | r1 = 0 | **mov** | **r1, 0** | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 640 |
| 02 | r2 = 4 | **mov** | **r2, 4** | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 684 |
| 03 | r4 = 1 | **mov** | **r4, 1** | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 701 |
| 04 | DO:  r3 = RAM[r0] | **ldr** | **r3, [r0, 0]** | 1 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 8C0 |
| 05 | r1 = r1 + r3 | **add** | **r1, r1, r3** | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 1 | 1 | 04B |
| 06 | r0 = r0 + 1 | **add** | **r0, r0, r4** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 004 |
| 07 | While:  r0 != 4 | **cmp** | **r0, r2** | 0 | 0 | 1 | - | - | - | 0 | 0 | 0 | 0 | 1 | 0 | 202 |
| 08 | r0 == 4 ? | **bne** | **r2** | 1 | 0 | 1 | - | - | - | 0 | 1 | 0 | - | - | - | A10 |
| 09 | Then: | **lsr** | **r1, r1, 2** | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 1 | 0 | 44A |
| 0A | R1\_Final = r1 | **str** | **r1, [r2]** | 1 | 1 | 1 | - | - | - | 0 | 0 | 1 | 0 | 1 | 0 | E0A |
| 0B |  | **b** | **0** | 1 | 1 | 0 | - | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | C00 |

*Tabela 3 - Codificação das instruções do programa*

NOTA: Assumimos neste caso que os *don’t cares*  tomam valor zero.

Ficheiro a carregar na ROM do Programa



*FICHEIRO 2 – Ficheiro de carregamento na ROM do Programa*

Registo de ocorrências

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **r0** | **r1** | **r2** | **r3** | **r4** | **r5** | **r6** | **r7** | **PC** | **PSR** |
| Clock 1 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 000 | 0 |
| Clock 2 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 001 | 0 |
| Clock 3 | 00 | 00 | 04 | 00 | 00 | 00 | 00 | 00 | 002 | 0 |
| Clock 4 | 00 | 00 | 04 | 00 | 01 | 00 | 00 | 00 | 003 | 0 |
| Clock 5 | 00 | 00 | 04 | 01 | 01 | 00 | 00 | 00 | 004 | 0 |
| Clock 6 | 00 | 01 | 04 | 01 | 01 | 00 | 00 | 00 | 005 | 0 |
| Clock 7 | 01 | 01 | 04 | 01 | 01 | 00 | 00 | 00 | 006 | 0 |
| Clock 8 | 01 | 01 | 04 | 01 | 01 | 00 | 00 | 00 | 007 | 0 |
| Clock 9 | 01 | 01 | 04 | 01 | 01 | 00 | 00 | 00 | 008 | 0 |
| Clock 10 | 01 | 01 | 04 | 10 | 01 | 00 | 00 | 00 | 004 | 0 |
| Clock 11 | 01 | 11 | 04 | 10 | 01 | 00 | 00 | 00 | 005 | 0 |
| Clock 12 | 02 | 11 | 04 | 10 | 01 | 00 | 00 | 00 | 006 | 0 |
| Clock 13 | 02 | 11 | 04 | 10 | 01 | 00 | 00 | 00 | 007 | 0 |
| Clock 14 | 02 | 11 | 04 | 10 | 01 | 00 | 00 | 00 | 008 | 0 |
| Clock 15 | 02 | 11 | 04 | 01 | 01 | 00 | 00 | 00 | 004 | 0 |
| Clock 16 | 02 | 12 | 04 | 01 | 01 | 00 | 00 | 00 | 005 | 0 |
| Clock 17 | 03 | 12 | 04 | 01 | 01 | 00 | 00 | 00 | 006 | 0 |
| Clock 18 | 03 | 12 | 04 | 01 | 01 | 00 | 00 | 00 | 007 | 0 |
| Clock 19 | 03 | 12 | 04 | 01 | 01 | 00 | 00 | 00 | 008 | 0 |
| Clock 20 | 03 | 12 | 04 | 10 | 01 | 00 | 00 | 00 | 004 | 0 |
| Clock 21 | 03 | 22 | 04 | 10 | 01 | 00 | 00 | 00 | 005 | 0 |
| Clock 22 | 04 | 22 | 04 | 10 | 01 | 00 | 00 | 00 | 006 | 0 |
| Clock 23 | 04 | 22 | 04 | 10 | 01 | 00 | 00 | 00 | 007 | 1 |
| Clock 24 | 04 | 22 | 04 | 10 | 01 | 00 | 00 | 00 | 008 | 1 |
| Clock 25 | 04 | 08 | 04 | 10 | 01 | 00 | 00 | 00 | 009 | 0 |
| Clock 26 | 04 | 08 | 04 | 10 | 01 | 00 | 00 | 00 | 010 | 0 |
| Clock 27 | 04 | 08 | 04 | 10 | 01 | 00 | 00 | 00 | 011 | 0 |

*TABELA 4 – Registo de ocorrências no processador*

|  |  |
| --- | --- |
| **RAM** | |
| **ADDR** | **Value (h)** |
| 0 | 01 |
| 1 | 10 |
| 2 | 01 |
| 3 | 10 |
| 4 | R1\_Final |
| … | … |

***Valores aliatórios colocados em memória para o teste das operações***

Conclusões

Conlui-se que com este processador conseguimos fazer um conjunto de operações interessante como somas, divisoes e subtracções assim como comparações ou mesmo OR bit a bit assim com fazer loops. A utilização dos bits de menor peso do *opcode* para a selecção de operações da ALU permite-nos ter instruções mais pequenas no descodificador de instruções (neste caso 2 bits mais pequenas). O facto de a ALU fornecer uma flag Z também simplifica em termos de numero deoperções para realizar um compare.