Organização de Computadores B (2022/1)

Trabalho 1 - Implementação de Instruções MIPS

Grupo: Henrique Utzig, João P. Cosme, Matheus Stein de Aguiar e Victor Machado Gonçalves.

**Descrição das Instruções:**

|  |  |  |  |
| --- | --- | --- | --- |
| **ADDI – Add Immediate** | | | |
| TIPO I | | | |
|  | | | |
| ADDI  0 0 1 0 0 0 | rs | rt | immediate | |
| 6 | 5 | 5 | 16 | |

A operação é simples. Queremos ler registro rs a partir do banco de registradores e somar com o valor imediato que é estendido com sinal para 32 bits. A soma em si é efetuada por um ALU. Simplificadamente:

|  |
| --- |
| GPR[rt] ← GPR[rs] + || |

Para deixar mais claro a notação, irei explicar alguns trechos:

|  |  |
| --- | --- |
| GPR[ra] | Ler o registro ra do banco de registradores |
|  | Obter os bits de a (inclusive) da variável abc, podemos omitir y, levando somente a 1 bit. |
|  | Copiar sequência de bits da variável e unir sequencialmente vezes |
|  | Unir sequência de bits e , preservando a ordem dada |

|  |
| --- |
| **BLTZ – Branch On Less Than Zero** |
| TIPO I |
|  |

|  |  |  |  |
| --- | --- | --- | --- |
| REGIMM  0 0 0 0 0 1 | rs | BLTZ  0 0 0 0 0 | offset |
| 6 | 5 | 5 | 16 |

Temos uma instrução Branch. Isso implica que há a possibilidade de aplicar o que o manual chama de “delayed slot”. Supondo que estamos no monociclo, podemos desconsiderar a menção desse conceito. Funcionalmente, iremos pular para PC+offset se o bit 31(supondo um vetor de 31...0), ou seja, queremos o último bit (bit de sinal) do registrador rs, assim se for 1(implica que é negativo, logo, menor que zero), pulamos.

Seguindo uma ordem aproximada queremos:

1. Offset é sempre de 4 em 4 bits, portanto, não é necessário incluir os dois primeiros zeros. Assim, queremos incluir dois zeros no início (equiv. a multiplicar por 4). Temos 18 bits.
2. Estender por sinal o offset por 14 bits levando a
3. Com o tamanho adequado, somar com o PC.
4. Verificar se o bit 31 de rs está ativado, se sim, pular

Repare, temos quase um BEQ, em vez de compararmos dois registros por igualdade, devemos apenas verificar um bit do registro rs.

|  |
| --- |
| target ← || ||  condition ← ()  if condition then  PC ← PC + target  endif |

|  |
| --- |
| **MULT – Multiply** |
| TIPO R |
|  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| SPECIAL  0 0 0 0 0 0 | rs | rt | 0  0 0 0 0 0 0 0 0 0 0 | MULT  0 1 1 0 0 0 |
| 6 | 5 | 5 | 10 | 6 |

A primeira operação R. De alto nível, seu funcionamento é bem simples, não é necessária uma explicação, só devemos reparar que, como estamos em 32 bits, o valor retornado pela multiplicação é de 64 bits. Como o nosso MIPS é de 32 bits, devemos criar dois registradores LO e HI, como proposto no manual.

|  |
| --- |
| t ←  LO ←  HI ← |

|  |
| --- |
| **JR – Jump Register** |
| TIPO R |
|  |

|  |  |  |  |
| --- | --- | --- | --- |
| SPECIAL  0 0 0 0 0 0 | rs | 0  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 | JR  0 0 1 0 0 0 |
| 6 | 5 | 15 | 6 |

Por último, temos uma instrução de pulo. Repare que a instrução é do tipo R. Temos uma pista que podemos utilizar o ALU de alguma forma para efetuar a operação. OBS: somar com zero e corrigir bits de controle

|  |
| --- |
| temp ←  PC ← |

**MIPS – Monociclo**

*ADDI* - Não é necessário realizar nenhuma modificação estrutural, apenas no bloco de controle. A instrução ADDI é equivalente ao código 003 do bloco de comando, onde as únicas flags ativas são o AluSrc(precisa controlar a ULA pelo OpCode e não pelo funct visto que não é uma instrução R) e o RegWrite (precisa escrever no registrador), visto que não mexe na memória (MemWrite, MemRead), não tem pulo (JMP, Branch), o registrador destino é o rt (RegDst = 0), a operação realizada pela ula é a soma (AluOp1 = 0, AluOp2 = 0) e como iremos usar o valor que vem da ula e não da memória (MemToReg = 0).

BLTZAL -

MULT -

JR -

**MIPS – Multiciclo**

ADDI -

BLTZAL -

MULT -

JR -

**MIPS – Pipeline**

ADDI -

BLTZAL -

MULT -

JR -