Universidade Federal de Minas Gerais
Departamento de Ciência da Computação
Belo Horizonte - MG - Brasil

Organização de Computadores 1

Trabalho Prático 2

Flávio Marcílio de Oliveira

Introdução

Este relatório tem como objetivo apresentar as alterações feitas no caminho de dados fornecido no arquivo .ipynb disponibilizado juntamente com as instruções e os problemas para serem implementados.

Para o desenvolvimento deste trabalho foram utilizadas como referências bibliográficas o manual RISC-V Assembly Language Programer Manual Part I, desenvolvido por SHAKTI Development Team e The RISC-V Instruction Set Manual Volume I: Unprivileged ISA

Problema 1: ANDI - Bitwise of Immediate

AND Immediate (ANDI) realiza operação binária entre o conteúdo do registrador (rs1) e os dados imediatos (imm) e armazena no registrador (rd).

Sintaxe

andi rd, rs1, imm

Onde,

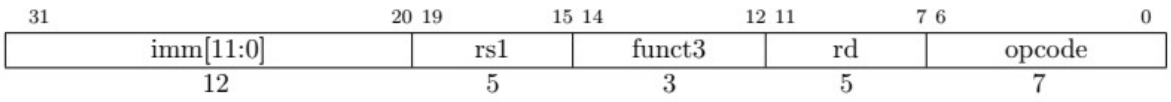
rd registrador de destino
rs1 registrador de origem 1
imm dados imediatos

Descrição

Um ANDI bit a bit é uma operação binária que usa dois padrões de bits de igual comprimento e executa a operação lógica inclusiva AND imediata em cada bit. Os registradores de origem e destino podem ser qualquer um dos 31 do banco de registradores. O registrador x0 pode ser usado apenas como registrador de origem, mas não como registrador de destino. 32 bits de resultado são escritos no registrador de destino.

Implementação

A instrução em linguagem de máquina apresenta a seguinte codificação:



onde,

opcode: 0010011
funct3: 111

Para realizar essa operação o módulo ***control*** foi modificado para quando o opcode for 0010011 o aluop seja 11 e alusrc seja 1 para controlar o mux que libera como segunda entrada da ALU o imediato estendido (também feito no Control), conforme abaixo:

```
    case (opcode)
      ...//outras operações
       7'b0010011: begin /* I-Type: addi, slli, slti, sltiu, xori, srli,
srai, ori, andi */
          // Problema 1 - Alteração necessária para permitir ALUOp = 11
          aluop[1:0]  <= 2'b11;
          alusrc <= 1'b1;
          ImmGen <= {{20{inst[31]}},inst[31:20]};
        end
      ...//outras operações
      endcase
```

Outra alteração foi realizada no módulo ***alu_control*** para quando o aluop for 11 seja avaliado o funct3, conforme abaixo:

```
   case(aluop)
      ...//outras operações
      // Problema 1 - Alteração para realizar operações com imediato
       2'b11: aluctl = _funct; /* addi, andi, ori, slli, slti, sltui,
xori, srli, srai */
      default: aluctl = 0;
    endcase
```

Assim, caso o funct3 seja 111 será mandado um sinal para a ALU definido por 0000

```
    case(funct[3:0])
      ...//outras operações
      // Problema 1 - Alterações necessárias para implementar andi
      4'b0111:  _funct = 4'b0000;   /* and, andi */
      ...//outras operações
      default: _funct = 4'bxxxx;//al
    endcase
```

E, finalmente, a ALU ao receber o sinal 0000 realiza a operação AND, como mostrado abaixo:

```
    case (ctl)
      4'b0000: out <= a & b;      /* and, andi */
      ...//outras operações
      default: out <= {32{1'bx}};
    endcase
```

Problema 2: SRLI - Shift Right Logical Immediate

SRLI executa um deslocamento lógico para direita no valor do registrador (rs1) pela quantidade de deslocamento informado pelo imediato (imm) e armazena no registrador (rd).

Sintaxe

srli rd, rs1, imm

Onde,

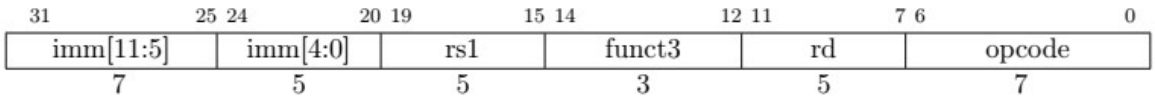
rd registrador de destino
rs1 registrador de origem 1
imm dados imediatos

Descrição

Um Shift Right Logical Immediate (SRLI) move cada bit para a direita. O bit mais significativo é substituído por um bit zero e o bit menos significativo é descartado.

Implementação

A instrução em Linguagem de Máquina tem a seguinte codificação:



onde,

opcode: 0010011
funct3: 101

Para realizar essa operação, a mesma alteração realizada no módulo **control** para o problema 1 será utilizada, pois também se trata de uma operação com imediato.

Como o aluop é 11 e depende do funct3, foi incluído a operação com funct3 = 101 no módulo ***alu_control***, como mostrado abaixo:

```
case(funct[3:0])
  ...//outras operações
  // Problema 2 - Alterações necessárias para implementar srli
  4'b0101:  _funct = 4'b1000;   /* srl, srli */
  ...//outras operações
  default: _funct = 4'bxxxx;//al
endcase
```

Assim, quando o funct3 for igual a 101 será mandado o sinal 1000 para a ALU. E, finalmente, a ALU ao receber o sinal 1000 realiza a operação SRLI, como mostrado abaixo:

```
    case (ctl)
      ...//outras operações
      // Problema 2 - Inclusão necessária para implementar a operação
srli
      4'b1000: out <= a >> b;      /* srl, srli */
      ...//outras operações
      default: out <= {32{1'bx}};
    endcase
```

Problema 3: J - Jump

Jump (J) é uma pseudo-instrução que usa Jump and Link (JAL) e define o registrador de destino como zero para descartar o endereço de retorno.

Sintaxe

j label

Onde,

j Saltar

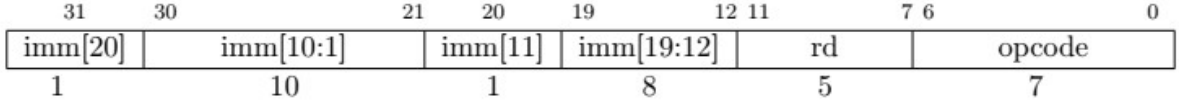
label Uma string que aponta para uma instrução

Descrição

J é uma instrução de salto incondicional simples (tipo UJ) usada para saltar para qualquer lugar na memória de código. Esta instrução se traduz em jal x0, label, que define o endereço de retorno como zero, descartando o endereço de retorno.

Implementação

A instrução em Linguagem de Máquina tem a seguinte codificação:



onde,

opcode: 1101111

rd: 00000

Para realizar essa operação, foi utilizado o mesmo processo utilizado quando é uma operação de branch com algumas modificações. Para isso, o módulo ***control*** foi modificado para quando o opcode for 1101111 disparar o aluop seja 01 e o sinal branch_eq seja 1. O imediato também foi estendido e shiftado para a esquerda, conforme mostrado abaixo:

```
   case (opcode)
     ...//outras operações
      // Problema 3: Alteração para implementar o jump
      7'b1101111: begin /* J-Type: j */
        jump <= 1'b1;
        aluop[1:0]   <= 2'b01;
ImmGen   <=   {{11{inst[31]}},inst[20],inst[10:1],inst[11],inst[19:12],
1'b0};
        branch_eq  <= 1'b1;
      end
    endcase
```

Problema 4: BGT - Branch if Greater Than

A instrução Branch if Greater Than (BGT) desloca o contador do programa para o local especificado se o valor em um registrador for maior que o de outro.

Sintaxe

bgt rs1, rs2, label

Tradução

blt rs2, rs1, etiqueta

Onde,

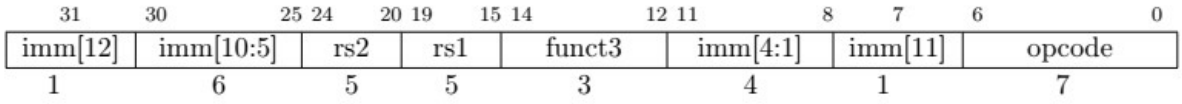
rs1 registrador de origem 1
rs2 registrador de origem 2
label Endereço para SALTAR para

Descrição

O BGT é uma instrução de comparação assinada que se traduz em BLT. Nesta instrução, é examinado se o conteúdo de rs2 é menor que o conteúdo do registrador rs1. Se a condição for satisfeita, o contador de programa desvia para o local especificado.

Implementação

A instrução em Linguagem de Máquina tem a seguinte codificação:



onde,

opcode: 1100011

funct3: 100

Para realizar essa operação foram utilizadas as implementações existentes da instrução BLT, não sendo necessária nenhuma modificação adicional, uma vez que na linguagem de máquina o código já é traduzido para corresponder a uma operação blt.