**Problemas na decodificação e execução das instruções**

|  |  |  |  |
| --- | --- | --- | --- |
| **#** | **Instruções** | **Dúvida** |  |
| 1 | SLT, SLTU | Como tratar os valores dos registrados rs1 e rs2 com sinal e sem sinal |  |
| 2 | SRL | Como tratar os valores dos registrados rs1 e rs2?   * Logical right shift on the value in register rs1 by the shift amount held in the lower 5 bits of register rs2 | https://open4tech.com/logical-vs-arithmetic-shift/ |
| 3 | SRA | Como tratar os valores dos registrados rs1 e rs2?   * Performs arithmetic right shift on the value in register rs1 by the shift amount held in the lower 5 bits of register rs2 | https://open4tech.com/logical-vs-arithmetic-shift/ |
| 4 | MULH | Como tratar os valores dos registrados rs1 e rs2?   * performs an XLEN-bit × XLEN-bit multiplication of signed rs1 by signed rs2 and places the upper XLEN bits in the destination register. | * Faz multiplicação normal * Depois faz shift rigth de 32 bits pegando a parte mais significativa e armazena no RD |
| 5 | MULHSU | Como tratar os valores dos registrados rs1 e rs2?   * performs an XLEN-bit × XLEN-bit multiplication of signed rs1 by unsigned rs2 and places the upper XLEN bits in the destination register. | * Rs1 com sinal * Rs2 sem sinal * Faz multiplicação normal * Depois faz shift rigth de 32 bits pegando a parte mais significativa e armazena no RD |
| 6 | MULHU | Como tratar os valores dos registrados rs1 e rs2?   * performs an XLEN-bit × XLEN-bit multiplication of unsigned rs1 by unsigned rs2 and places the upper XLEN bits in the destination register. | * Rs1 sem sinal * Rs2 sem sinal * Faz multiplicação normal * Depois faz shift rigth de 32 bits pegando a parte mais significativa e armazena no RD |
| 7 | REM | Como tratar os valores dos registrados rs1 e rs2?   * perform an XLEN bits by XLEN bits signed integer reminder of rs1 by rs2. |  |
| 8 | REMU | Como tratar os valores dos registrados rs1 e rs2?   * perform an XLEN bits by XLEN bits unsigned integer reminder of rs1 by rs2. |  |
| 9 | JALR | Validar implementação |  |
| 10 | SLTI | Como tratar os valores dos registrados rs1 e rs2?   * Place the value 1 in register rd if register rs1 is less than the signextended immediate when both are treated as signed numbers, else 0 is written to rd. | O load já resolve a questão do sinal |
| 11 | SLTI | Como tratar os valores dos registrados rs1 e rs2?  Place the value 1 in register rd if register rs1 is less than the signextended immediate when both are treated as signed numbers, else 0 is written to rd. |  |
| 12 | SLLI | Validar o shift rigth com o campo shamt   * Performs logical left shift on the value in register rs1 by the shift amount held in the lower 5 bits of the immediate | Pegar o imediatro de 12 bits, completer até 32 bits  Se bit12 = 1 , completar de 0 a 20 com digito 1 (python) à esquerda  Se bit12 = 0 , completar de 0 a 20 com digito 0 (python) à esquerda  <https://open4tech.com/logical-vs-arithmetic-shift/> |
| 13 | SLRI | Validar o shift rigth com o campo shamt   * Performs logical right shift on the value in register rs1 by the shift amount held in the lower 5 bits of the immediate | Pegar o imediatro de 12 bits, completer até 32 bits  Se bit12 = 1 , completar de 0 a 20 com digito 1 (python) à esquerda  Se bit12 = 0 , completar de 0 a 20 com digito 0 (python) à esquerda  https://open4tech.com/logical-vs-arithmetic-shift/ |
| 14 | SRAI | * Performs arithmetic right shift on the value in register rs1 by the shift amount held in the lower 5 bits of the immediate | Pegar o imediatro de 12 bits, completer até 32 bits  Se bit12 = 1 , completar de 0 a 20 com digito 1 (python) à esquerda  Se bit12 = 0 , completar de 0 a 20 com digito 0 (python) à esquerda   * https://open4tech.com/logical-vs-arithmetic-shift/ |
| 15 | LB | Como tratar o intervalo [7:0] do valor obtido a partir da memória, bem como a operação “sext”?   * # Implementation: x[rd] = sext(M[x[rs1] + sext(offset)][7:0]) | Recupera a primeira posição de memória apontada pelo rs1 + offset: M[1] |
| 16 | LH | Como tratar o intervalo [15:0] do valor obtido a partir da memória, bem como a operação “sext”?   * # Implementation: x[rd] = sext(M[x[rs1] + sext(offset)][15:0]) | Recupera a primeira posição de memória apontada pelo rs1 + offset: M[2]+M[1] |
| 17 | LW | Como tratar o intervalo [31:0] do valor obtido a partir da memória, bem como a operação “sext”?   * # Implementation: x[rd] = sext(M[x[rs1] + sext(offset)][31:0]) | Recupera a primeira posição de memória apontada pelo rs1 + offset: M[4]+M[3]+M[2]+M[1] |
| 18 | LBU | Como tratar o intervalo [7:0] do valor obtido a partir da memória, bem como a operação “sext”?   * # Implementation: x[rd] = sext(M[x[rs1] + sext(offset)][7:0]) | NÃO PODE FAZER O COMPLEMENTO A DOIS PORQUE É UNSINGED (NÃO PODE CHAMAR adjust\_bin\_immediate\_n\_bits NA DESMONTAGEM DO TUIPO I)  Recupera a primeira posição de memória apontada pelo rs1 + offset: M[1] |
| 19 | LHU | Como tratar o intervalo [15:0] do valor obtido a partir da memória, bem como a operação “sext”?   * # Implementation: x[rd] = sext(M[x[rs1] + sext(offset)][15:0]) | NÃO PODE FAZER O COMPLEMENTO A DOIS PORQUE É UNSINGED (NÃO PODE CHAMAR adjust\_bin\_immediate\_n\_bits NA DESMONTAGEM DO TUIPO I)  Recupera a primeira posição de memória apontada pelo rs1 + offset: M[2]+M[1] |
| 20 | CSRRW, CSRRWI, CSRRS, CSRRSI, CSRRC, CSRRCI | Como utilizar o vetor CSRs nas instruções? | Nada a fazer |
| 21 | SB | Como tratar o intervalo [7:0] do valor obtido a partir da memória, bem como a operação “sext”?   * # Implementation: M[x[rs1] + sext(offset)] = x[rs2][7:0] | Armazena o valor apontado pelo rs2 na memoria |
| 22 | SH | Como tratar o intervalo [15:0] do valor obtido a partir da memória, bem como a operação “sext”?   * # Implementation: M[x[rs1] + sext(offset)] = x[rs2][15:0]) | Divide o valor apontado pelo rs2, divide em 2 partes de 8 bits e armazena em dois endereços de memória consecutivos.  M[1] = rs2[0:7]  M[2] = rs2[8:15] |
| 23 | SW | Como tratar o intervalo [31:0] do valor obtido a partir da memória, bem como a operação “sext”?   * # Implementation: M[x[rs1] + sext(offset)] = x[rs2][31:0]) | Divide o valor apontado pelo rs2, divide em 4 partes de 8 bits e armazena em quatro endereços de memória consecutivos.  M[1] = rs2[0:7]  M[2] = rs2[8:15]  M[3] = rs2[16:23]  M[4] = rs2[24:31] |
| 24 | BEQ, BNE | Validar implementação | Correto com sinal |
| 25 | BLT, BGE | Como fazer o teste utilizando comparação com sinal? | Correto com sinal |
| 26 | BLTU, BGEU | Como fazer o teste utilizando **comparação sem sinal?** | If rs1 < 0 :  Aux\_rs1 = rs1 + (1<<32)  If rs2 < 0 :  Aux\_rs2 = rs2 + (1<<32)  Depois faz o teste de aux\_rts1 e aux\_rs2 |
| 27 | DIV, DIVU | Verificar falha nos valores de memória endereçados por rs1 e rs2 (zerados) |  |

**Usar para todas as portas lógicas ORI, XORI, ANDI, etc**

Pegar o imediatro de 12 bits, completer até 32 bits

Se bit12 = 1 , completar de 0 a 20 com digito 1 (python) à esquerda

Se bit12 = 0 , completar de 0 a 20 com digito 0 (python) à esquerda