**Instruction fetch:**

Nesta etapa o próximo PC é escolhido baseado na instrução atual.

Diagrama

O conteúdo gerado por IA pode estar incorreto.

Para encontrar o próximo pc primeiro a posição do pc em cada instrução de salto é calculada, depois com o sinal de controle “Jump\_type” escolho qual será usado.

Desenho com traços pretos em fundo branco

O conteúdo gerado por IA pode estar incorreto.

**Instruction decode:**

O pc entra na memória de duas portas e a seu valor é usado para selecionar e instrução a ser executada

Calendário

O conteúdo gerado por IA pode estar incorreto.

Então o valor que estava na memória é separado em 4 grupos de bits, onde os 4 primeiros sempre serão o opcode e os 4 últimos podem ser um imediato de 4 bits ou dois registradores de 2 bits cada. Caso a instrução seja do tipo I os dados serão escritos no R0.

Uma imagem contendo Diagrama

O conteúdo gerado por IA pode estar incorreto.

**Execução:**

Após ser separada a instrução passa pelo banco de registradores, onde o primeiro valor da ula ou o valor a ser guardado na memória sai pela porta “Inf\_a” e o segundo valor da ula sai pela por “inf\_b”, “inf\_b” também é usado em instruções de Branch, onde seus dados são o endereço para qual o pc deve apontar.

Diagrama

O conteúdo gerado por IA pode estar incorreto.

Decidi colocar o comparador fora da ula pois ele é usado apenas para um sinal de controle de uma instrução.

Diagrama

O conteúdo gerado por IA pode estar incorreto.

Dentro da ULA todas as operações ocorrem em paralelo, e então o resultado e decidido por um sinal de controle. Caso a operação resulte em zero o sinal zero é ativado.

Diagrama

O conteúdo gerado por IA pode estar incorreto.

**Sinais de Controle:**

Imm: sinal de 2 bits que decide a entrada da ula

Mem\_en: se ativo permite escrita na memória

Tipo\_I: Define a entrada do “ra” para r0

Reg\_en: permite a escrita no banco de registradores

Jump\_type: sinal de 2 bits que define o tipo de desvio de fluxo

Wb\_source: se 1 a escrita no banco de registradores vem da ULA, se 0 vem da memória

Ula\_op: sinal de 3 que define qual operação a ula realiza

Uma imagem contendo Diagrama

O conteúdo gerado por IA pode estar incorreto.

Conjunto de instruções (ISA)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Opcode | Tipo | Mnemonic | Nome | Operação |
| 0000 | R | brzr | Branch on Zero Register | if (R[ra] == 0) PC = R[rb] |
| 0001 | I | ji | Jump Immediate | PC = PC + Imm. |
| 0010 | R | ld | Load | R[ra] = M[ R[rb] ] |
| 0011 | R | st | Store | M[ R[rb] ] = R[ra] |
| 0100 | I | addi | Add Immediate | R[0] = R[0] + Imm |
| 0101 | I | addui | Add Unsigned Immediate | R[0] = R[0] + Imm(ext zero) |
| 0110 | I | addupi | Add Upper Immediate | R[0] = R[0] + (Imm<<4) |
| 0111 | R | bgzr | Branch on greater then Zero Register | if (R[ra] > 0) PC = R[rb] |
| 1000 | R | not | Not | R[ra] = not R[rb] |
| 1001 | R | and | And | R[ra] = R[ra] and R[rb] |
| 1010 | R | or | Or | R[ra] = R[ra] or R[rb] |
| 1011 | R | xor | Xor | R[ra] = R[ra] xor R[rb] |
| 1100 | R | add | Add | R[ra] = R[ra] + R[rb] |
| 1101 | R | sub | Sub | R[ra] = R[ra] - R[rb] |
| 1110 | R | slr | Shift Left Register | R[ra] = R[ra] << R[rb] |
| 1111 | R | srr | Shift Right Register | R[ra] = R[ra] >> R[rb] |

Motivo para a escolha das instruções:

**Add Unsigned Immediate:**

Muitas vezes no código era necessário adicionar valores maiores que 7 o que não é possível com a função addi.

Além de auxiliar a próxima instrução, pois para a soma dos bits mais significativos com os bits menos significativos e preciso que estes sejam sem sinal, para não interferir com a parte superior do número.

**Add Upper Immediate:**

Sem essa instrução para encontrar qualquer número acima de 15 era preciso executar o seguinte código:

addi x ; x = 4 bits mais significativos do valor

add ra,r0

xor r0, r0

addi 4

sll ra, r0

xor r0, r0

addi y ; x = 4 bits menos significativos do valor

add ra, r0 ; ra contém o valor de 8 bits

Com a função de addupi esse código se torna:

addupi x ; x = 4 bits mais significativos do valor

addui y ; x = 4 bits menos significativos do valor

add ra, r0 ; ra contém o valor de 8 bits

**Branch on greater then Zero Register:**

Usado para reduzir o loop de soma, devido ao jump ser apenas usando imediato não é possível voltar para o início do loop com um jump, e devido a quantidade limitada de registradores não é viável usar 2 Branch. Com essa instrução é possível fazer loops sem utilizar jumps de retorno, o loop para de ser executado quando um decrementador chega a zero.

|  |  |  |
| --- | --- | --- |
| Opcode | Hexa | Sinal de Controle |
| 0000 | 0 | 10000100001 |
| 0001 | 1 | 01010010000 |
| 0010 | 2 | 00001000000 |
| 0011 | 3 | 00100000000 |
| 0100 | 4 | 01011001000 |
| 0101 | 5 | 10011001000 |
| 0110 | 6 | 11011001000 |
| 0111 | 7 | 10000110001 |
| 1000 | 8 | 00001001111 |
| 1001 | 9 | 00001001100 |
| 1010 | a | 00001001101 |
| 1011 | b | 00001001110 |
| 1100 | c | 00001001000 |
| 1101 | d | 00001001001 |
| 1110 | e | 00001001010 |
| 1111 | f | 00001001011 |

Novo Código:

xor r0, r0

xor r1, r1

xor r2, r2

xor r3, r3

addupi 3

addui 13

add r1,r0

;r1 = primeiro endereço vago após o código

xor r0, r0

addui 10

add r3, r0

;r3 = 10 (contador)

addi 3

add r2, r0

;r2 = endereço de início do loopR

xor r0, r0

;loopR Inicializa R[ ] com 0

st r0, r1

addi 1

sub r3, r0

add r1, r0

xor r0, r0

bgzr r3, r2

;fimloopR

addi 1

add r3, r0 ; r3 = 1

addui 8

add r1, r0

addi 4

add r2, r0

;r2 = endereço de início do loopA

addi 7 ; r0=20

;loopA Carrega A[ ] do fim para o início com todos os pares entre 0 e 18

sub r0, r3

sub r0, r3

st r0, r1

sub r1, r3

bgzr r0, r2

;fimloopA

addui 9

add r2, r0 ; r2 = 36

addui 11 ; r0 = 20

add r1, r0

;loopB Carrega B[ ] do fim para o início com todos os ímpares entre 1 e 19

sub r0, r3

st r0, r1

sub r0, r3

sub r1, r3

bgzr r0, r2

;fimloopB

xor r3 r3

xor r0, r0

addui 9

add r2, r0 ; r2 = 47

;r2 = endereço de início do loopSoma

;loopSoma Carrega R[ ] com a soma de A[ ] e B[ ] do fim para o início

ld r3, r1 ; carrega A[i]

xor r0, r0

addui 10

add r1, r0

ld r0, r1 ; carrega B[i]

add r3, r0 ; r3 = A[i] + B[i]

xor r0, r0

addui 10

sub r1, r0

sub r1, r0

st r3, r1 ; R[i] = r3

add r1, r0 ; r1 = A[i]

ld r3, r1

xor r0, r0

addi 1

sub r1, r0

;r1 = A[i-1]

bgzr r3, r2

Total de linhas: 61