Relatório do Trabalho 2

Thamiris Yamate Fischer

***Resumo*—O seguinte relatório tem como objetivo apresentar a construção da arquitetura inspirada no ReduxV, além de justificar as novas instruções selecionadas.**

I. Construção do Circuito

A. Diagrama de Caixas

*1) **Datapath do processador**:* O processador foi pensado do seguinte modo:

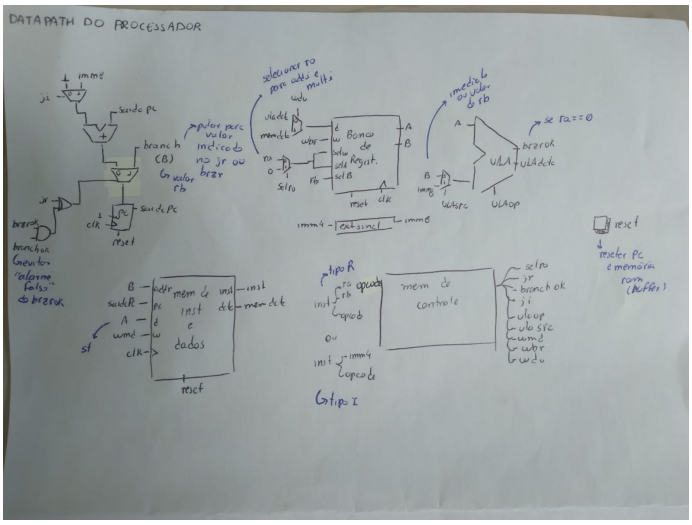

Figura 1. Datapath do Processador

- **Branch**: Tanto o ”brzr”quanto o ”jr”saltam para o valor que está em rb, já o ”ji”salta em relação ao valor atual do Pc.
- **”Zero da ULA- brzrOk**: Para identificar se é necessário fazer um salto no brzr ou não, na ULA há um comparador que indica se o valor de ra é igual à 0.
- **Seleção do r0**: As instruções ”addi”e ”multi”são executadas no r0, logo, para essas instruções o valor de ra sempre será 0.
- **Reset**: Considerando que inicialmente o processador estará indefinido, esse botão ”reseta”o valor do Pc e dos registradores.

*2) **ULA**:* Na imagem ao lado há os códigos de cada operação da ULA e como foi pensado a implementado.

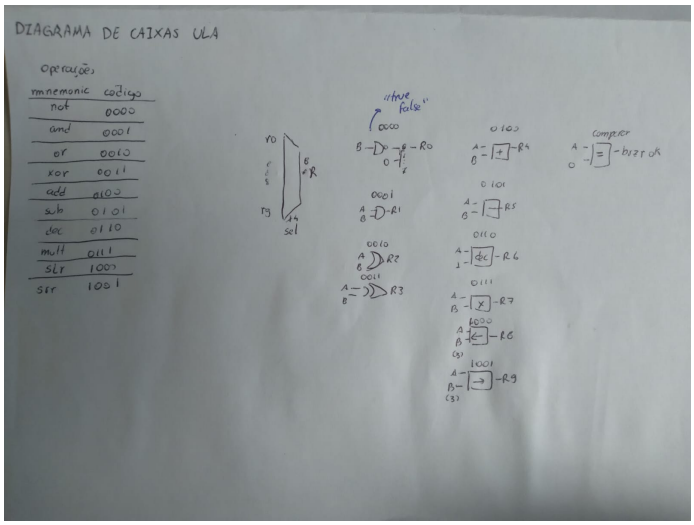

Figura 2. Diagrama de Caixas da ULA

B. Memória de Controle

Os controladores do processador são:

- **WDU** (write data or ula): Se o valor a ser escrito no registrador vem da memória de dados ou da ULA.
- **WBR** (write bank register): Ativar a escrita no banco de registradores ou não.
- **WMD** (write data memory): Ativar a escrita na memória de dados ou não.
- **ULASrc** (Ula source): Se o valor vem do rb ou do imediato.
- **ULAOp**: Qual operação será feita na ULA, de acordo com a codificação da imagem anterior.
- **ji**: Se a instrução é o jump immediate.
- **branchOk**: Evitar ”falso alarme”do brzrOk, assim evitando saltos sem ser com o brzr.
- **jr**: Se a instrução é o jump register.
- **Selr0**: Selecionar o r0 para as instruções addi e multi.

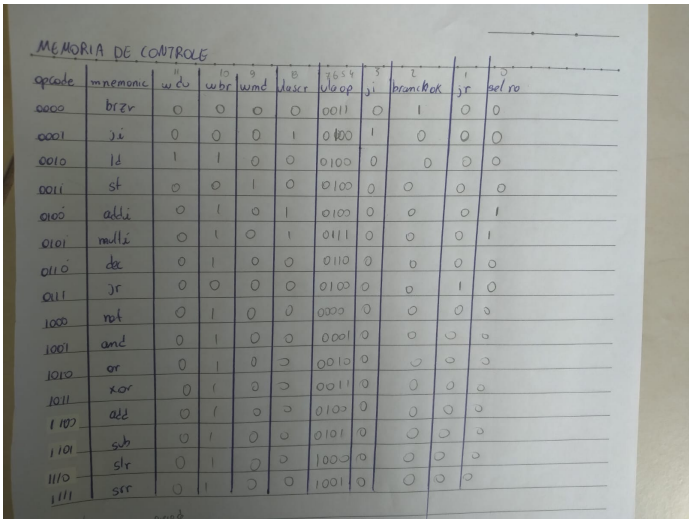

Figura 3. Memória de controle

II. Assembly

A. Novas instruções

Após analisar algumas dificuldades de implementação do assembly no trabalho anterior e refletir sobre quais instruções poderiam ajudar a tornar o código mais limpo e claro, foram desenvolvidos essas três instruções:

- **`multi`** (multiplica imediato): Com o slr é possível alcançar valores distantes sem precisar utilizar vários addi em sequência, porém, há menos controle sobre os valores da multiplicação, já que o slr multiplica apenas por 2 e suas potências. Então, a fim de evitar o uso de vários addi em sequência e para poder ter mais controle sobre os valores que desejo para r0, resolvi implementar o `multi`, em que `R[r0] = R[r0]*imm`;
- **`dec`** (decrementa 1): Ao longo do código, o registrador r1 é fixado como ”ponteiro”para os vetores e r2 como contador e valor a ser inserido nos vetores A e B. No trabalho anterior decrementar esses registradores em 1 se mostrou muito recorrente para, no r1, ”andar”pelo vetor e, no r2, para decrementar o contador e o valor dos vetores, porém para isso era necessário tomar r0 como auxiliar, atribuindo o valor 1 ou guardar essa constante em memória. Então, para evitar esses dois casos resolvi implementar o `dec ra`, em que `R[ra] = R[ra] - 1`;
- **`jr`** (jump register): No trabalho anterior, em um dos loops, foi necessário fazer diversos pequenos saltos com o ji devido à sua limitação. Como o código fica poluído desse modo, resolvi implementar o `jr rb`, em que `Pc = R[rb]`.

B. Novo código

```
    ;; Nessa versao o R nao sera inicializado com
        zeros

;; Memoria
    ;; addi 15 | #255 = 10 | Contador soma
    ;; addi 14 | #254 = r3 | Salto brzr
    ;; addi 13 | #253 = jr | Retorno

;; Registradores
    ;; r0 = puxar valores da memoria, auxiliar
    ;; r1 = ponteiro para os vetores
    ;; r2 = contador, armazenar valores dos vetores
    ;; r3 = salto brzr ou jr

;; Forcar valor zero nos Registradores
    xor r0, r0
    xor r1, r1
    xor r2, r2
    xor r3, r3

;; Armazenar valores em memoria e Calcular ponteiro
     r1
    addi 5
    multi 2
    add r3, r0 ;; r3 = 10
    multi 2
    add r2, r0 ;; r2 = 20
    dec r2 ;; r2 = 19
    multi 7 ;; r0 = 140
    add r1, r0 ;; r1 = 140
    dec r1 ;; r1 = 139 | final de B
    xor r0, r0
    addi 15
    st r3, r0 ;; #255 = 10

;; Calcular salto jr
    xor r0, r0
    addi 6
    multi 4 ;; 24
    addi -1 ;; 23
    add r3, r0 ;; r3 = 33
    xor r0, r0
    addi 13
    st r3, r0 ;; #253 = 33

;; Calcular salto r3 brzr
    xor r0, r0
    addi 3
    multi 6
    add r3, r0 ;; r3 = 50
    xor r0, r0
    addi 14
    st r3, r0 ;; #254 = 50
    xor r0, r0 ;; retorno jr
    addi 14
    ld r3, r0 ;;  #254 = 50

    xor r0, r0
        st r2, r1 ;; B[r1] = r2
        addi 5
        multi 2 ;; r0 = 10
        sub r1, r0 ;; r1 - 10
        dec r2 ;; r2 = r2 -1 | Numero par
        st r2, r1 ;; A[r1] = r2
    brzr r2, r3 ;; Loop sai aqui, quando o ultimo
        par aparece A = r2 = 0 -> r0 = 10
        dec r2 ;; r2 = r2 -1 | Numero impar
        add r1, r0 ;; Retornar para B
        dec r1  ;; Decrementar uma posicao
        xor r0, r0 ;; limpar r0
        addi 13
        ld r3, r0 ;; r3 = 33
    jr r3
    addi -5 ;; r0 = 10 -5 = 7 || r3 = 50 +7 = 57
    add r3, r0 ;; r3 = 57
    xor r0, r0
    addi 14
    st r3, r0 ;; #254 = 57
```

```
    xor r0, r0 ;;retorno brzr
        ld r2, r1 ;; r2 = val.A
        addi 5
        multi 2 ;; r0 = 10
        add r1, r0 ;; Ir para B
        ld r3, r1 ;; r3 = val.B
        add r2, r3 ;; r2 = val.A + val.B
        add r1, r0 ;; Ir para R
        st r2, r1 ;; Guardar em R
        multi 2 ;; r0 = 20
        addi -1 ;; r0 = 19
        sub r1, r0 ;; Ir para A somado em 1 pos
        xor r0, r0
        addi 14
        ld r3, r0 ;; r3 = #254
        addi 1 ;; #255
        ld r2, r0 ;; r2 = contador
        dec r2 ;; decrementa r2
        st r2, r0 ;; guarda iterador
        not r2, r2
    brzr r2, r3
ji 0 ;; halt
```

Listing 1. Assembly do T2

III. Observações

- O circuito se inicia indefinido, logo, antes de rodar o projeto é necessário apertar o botão de reset e colocar um dos códigos na memória ram.
- No logisim, há apenas uma memória de dados para os dois códigos (trabalho 1 e trabalho 2). Logo, para rodar cada um deles é necessário colocá-los na memória ram antes.
- Assim como o ji no trabalho 1, o jr e o brzr também saltam para uma instrução anterior à que realmente deve ser executada, isso é feito, pois apenas a instrução seguinte é executada.
- No circuito há displays abaixo do processador, o objetivo deles é visualizar o que o código está fazendo.