Implementação da Arquitetura Sagui VLIW

Fernando de Barros Castro

Maio 2024

1 Introdução

O processador simplificado é feito para uma arquitetura com palavras de instrução de 32 bits (4 sub instruções de 8 bits). A contrução foi feita utilizando a técnica de pipeline com quatro estágios (IF, ID, EXE/MEM, WB) baseado no conceito de load-store. As instruções são armazenadas em uma *read-only memory* separada da RAM como modelo havard de processador.

2 IF

- Lógica do PC: O *program counter* vem da saída do MUX controlado pelos sinais *pcsrc, b, zero* e *lock*. A ideia é que se a instrução não for branch, ou seja, lock está desativado e b está em 0, o *pcsrc* é propagado e controla o próximo pc. Caso seja um branch e haja dependência de dados, ativando o sinal de lock, o sinal propagado é 11. Esse faz PC = PC e trava o pipeline (*pipeline interlock*). Ainda sendo um branch, agora com lock em 0, o sinal propagado é 00 caso *zero* esteja em 0, indicando salto não tomado. Caso contrário, se o salto for tomado, o MUX seleciona a entrada indicada por *pcsrc*.

- No operation: caso a instrução na fase de decode faça um salto, entende-se que a instrução que vem logo após o branch não está certa já que o pc que deu origem a ela é derivado do *pcsrc* da instrução anterior. Assim um nop deve ser inserido no pipeline e isso é feito com um MUX. Além disso, o reset inicial do processador também deve inserir nops em todos quatro estágios do pipeline.

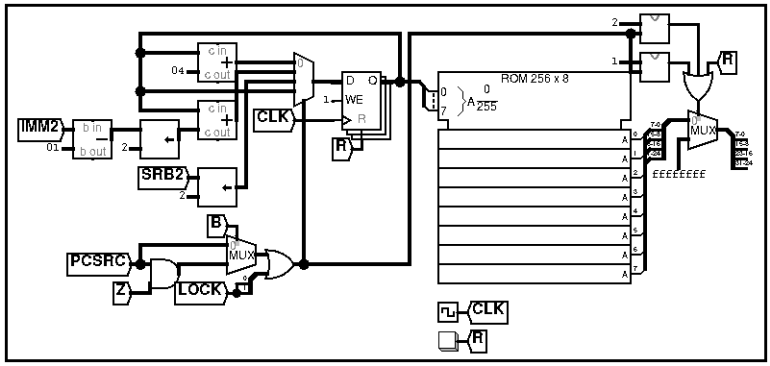

Figure 1: Instruction Fetch

3 ID

- Sinais de Controle Adotados:

  1. IMMSRC_C: Em 1 indica uso do imediato pela linha BR/JUMP.
  2. IMMSRC_D: Em 1 indica uso do imediato pela linha de MOV/LD/ST.
  3. PCSRC: Sinal que controla mux do PC. Em 00 faz PC = PC + 4; 01 faz PC = PC + IMM; 10 faz PC = R[rb] e 11 faz PC = PC.
  4. MOV: Controla se dado a ser gravado no banco de registradores vem de *movl, movh* ou memória. 00 é usado para loads, 10 para *movh* e 11 para *movl*.
  5. B: Em 1 indica que um branch está na linha de BR/JUMP e junto com *zero* e *pcsrc* decide próximo pc.
  6. REGW1, REGW2, REGW3: Em 1, indicam escrita no banco de registradores pelas linhas 1 (LD/ST/MOV), 3 (ULA) e 4 (ULA) respectivamente.
  7. MEMW: Escrita na RAM pela linha de LD/ST/MOV caso esteja em 1
  8. ULAOUT3, ULAOUT4: Mostra qual operação a ser feita na ULA nas linhas 3 e 4 respectivamente. 000/add; 001/sub; 010/and; 011/or; 100/not; 101/slr e 110/srr;

- Implementação do Controle: A implementação foi feita por meio de mapas de karnaugh já que a arquitetura VLIW facilita muito o desenvolvimento do controle com lógica combinacional. Isso se deve ao fato de que cada linha pode receber no máximo 8 opcodes direntes (incluindo nop). Dessa forma, há muitos *dont care* e o circuito fica simplificado.

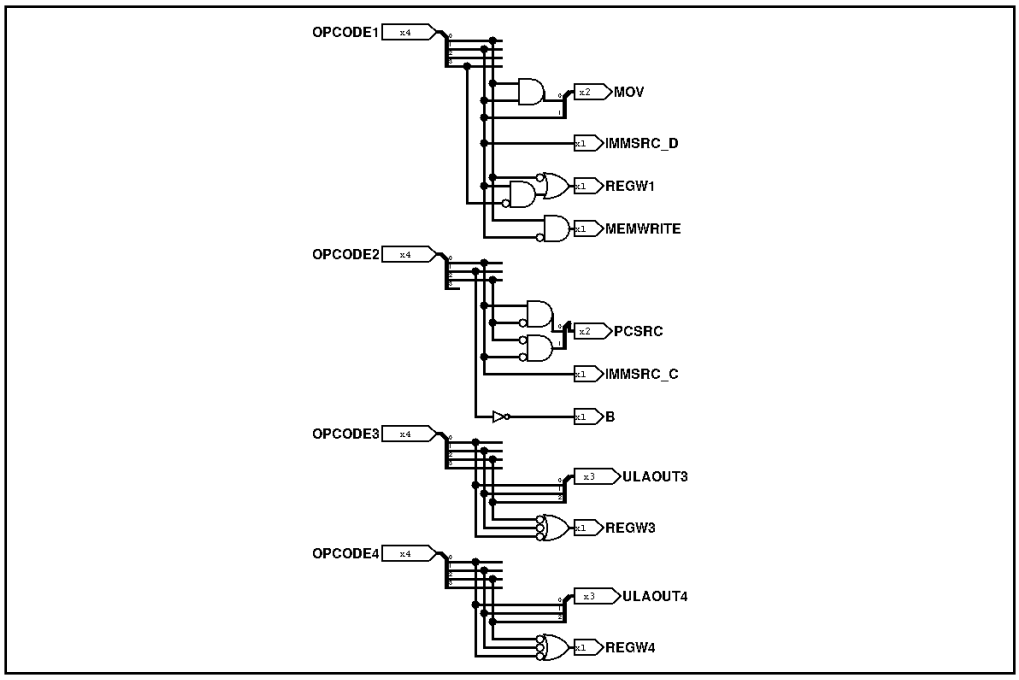

Figure 2: Controle

- Implementação do Banco de Registradores:

1. Escrita: Os dados a serem escritos vem da fase de *Write Back* na forma das entradas RAW1, RAW3 e RAW4 indicando endereço da escrita; WB1, WB3 e WB4 sendo o sinal de *regwrite* dessas instruções e DATA1, DATA3 e DATA4 os dados a serem escritos. A escrita é feita na borda de descida para resolver dependência estrutural (duas instruções usando mesma estrutura), de forma que o dado certo é escrito na borda de descida e pronto para ir para próximo estágio na próxima subida do clock.
2. Leitura: A leitura é realizada com quatro entradas de endereço de RA e mais quatro de RB. As instruções que usam imediato usam o sinal de *IMMSRC* para definir RA como 0.
3. Contador: Essa estrutura é usada para marcar dependências dos registradores, ou seja, 'informar' para a instrução atual que algum registrador tem dado atrasado que ainda não foi alterado por uma instrução anterior. Essa informação é usada para implementar *pipeline interlock* para as instruções do tipo branch e jump que são processadas logo no segundo estágio.

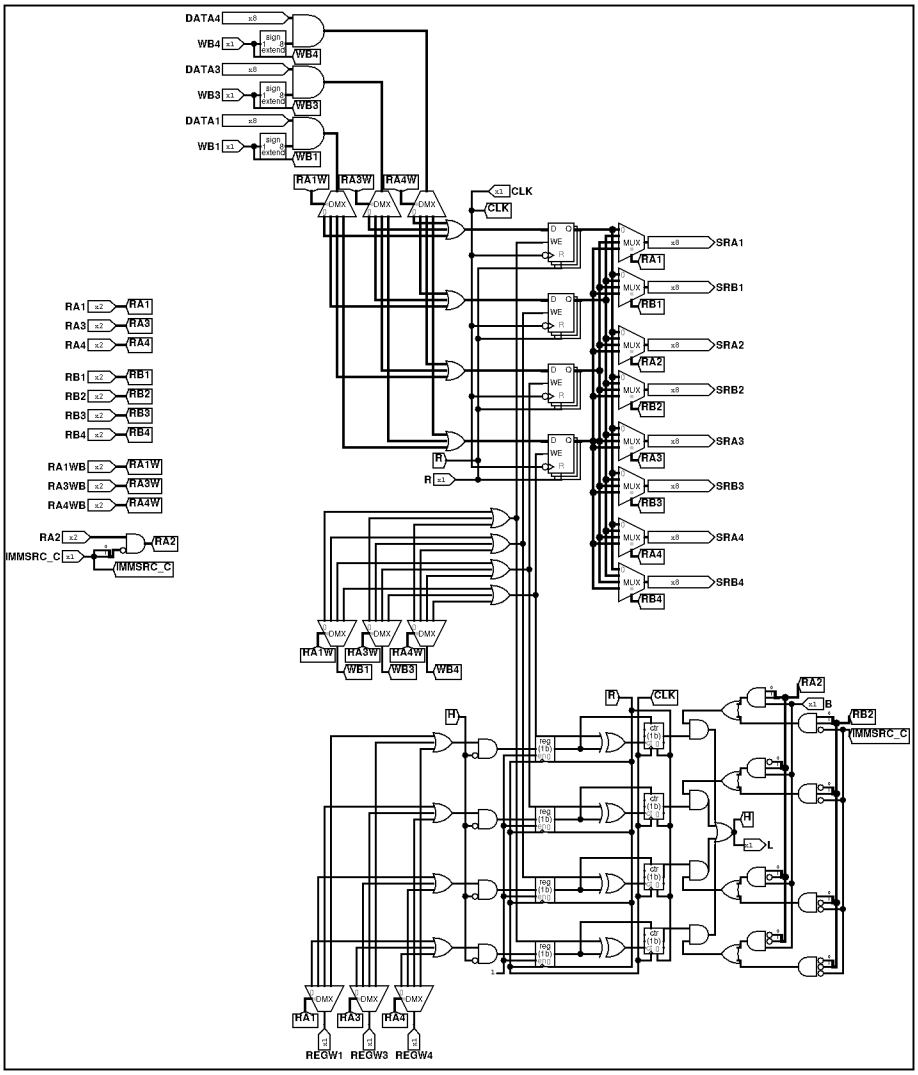

Figure 3: Banco de Registradores

4 Execution/MEM Read

- Detalhes da ULA: É importante perceber que tanto shift a esquerda quanto a direita, o deslocamento máximo é de 7 bits. Assim, o circuito utilizado que implementa essa lógica faz com que um bit mais significativo que o terceiro ($> 2^2$) ativo em R[rb] faz a saída ser zero.

- Adiantamento de Dados: As operações da ULA, de *move* ou que acessam memória podem ter depêndencia com a instrução imediatamente anterior que ainda não escreveu no banco de registradores. Dessa forma, os três RA devem ser comparados com os quatro RA e RB da instrução anterior a barreira, totalizando 18 comparações de igualdade. Se algum RA após a barreira for igual a um RA ou RB anterior e a instrução após não é NOP, então o dado modificado é propagado e um bit de DepRA leva 1 para o MUX que seleciona o RA ou RB mais atualizado.

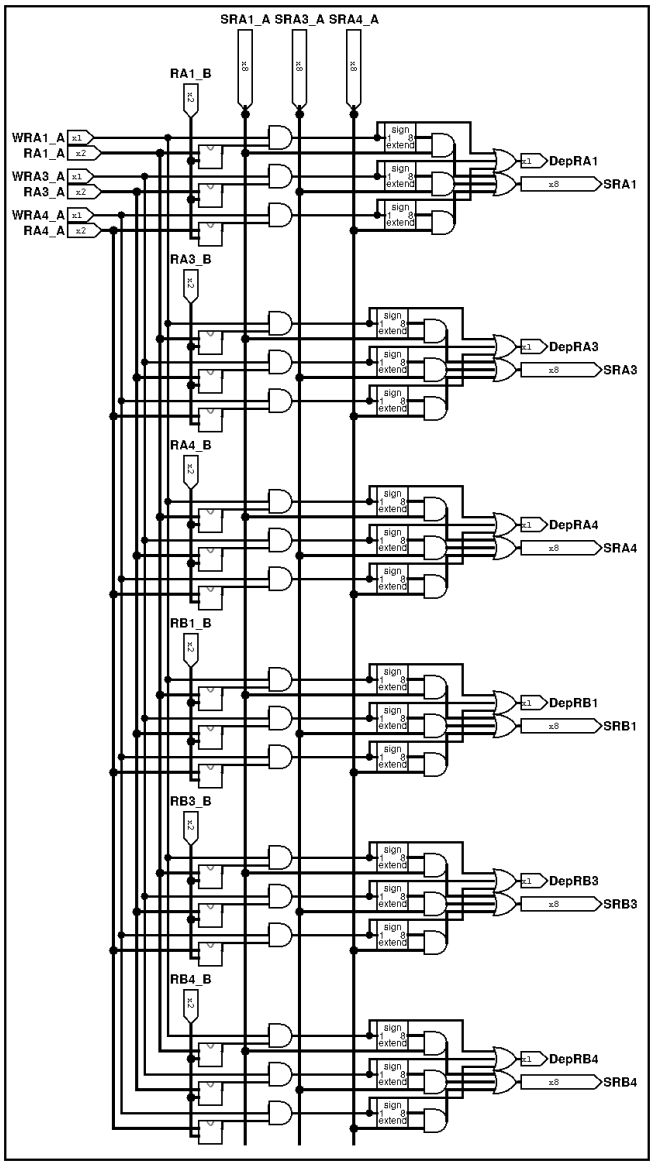

Figure 4: Banco de Registradores

5 Assembly para Teste das Instruções

```
movh 0001     // R[0] = 16
nop
sub r2, r2    // R[2] = 0
sub r3, r3;   // R[3] = 0
nop
ji 0001       // SALTA PARA PROXIMA INSTRUCAO
nop
nop;
movl 0000     // R[0] = 16
brzi 0000     // SALTO NAO TOMADO
not r1, r0    // R[1] = !R[0]
add r3, r0;   // R[3] = 16
st r2, r3     // MEM[16] = 0
nop
and r0, r1    // R[0] = 0
or r1, r0;    // R[1] = 255
movl 0001     // R[0] = 1
nop
nop
nop;
ld r0, r3    // R[0] = 0
nop
slr r1, r3   // R[1] DEVE RECEBER 0
srr r3, r0;  // R[3] = 8
movl 1010    // R[0] = 10
brzr r2, r3  // SALTO TOMADO PARA INSTRUCAO 8
nop
nop;
nop;         // INSTRUCAO VAZIA
nop;         // INSTRUCAO VAZIA
nop
jr r0        // SALTA PARA PROXIMA INSTRUCAO
nop
nop;
```

6 Assembly para Teste de Soma de Dois Vetores

```
// R[0] = 11, R[1] = 23, R[2] = 31, R[3] = 1
movh 0001
nop
sub r1, r1
sub r2, r2;
movl 0111
nop
sub r3, r3
nop;
movl 1111
nop
add r1, r0
nop;
movh 0000
nop
add r2, r0
nop;
movl 0001
nop
nop
nop;
movl 1011
nop
add r3, f0
nop;
// CARREGA VETORES A E B
st r2, r1
nop
sub r2, r3
nop;
st r0, r0
brzi 0010
nop
nop;
nop
ji 1110
sub r0, r3
sub r1, r3;
// R[0] = 11, R[1] = 24, R[2] = 0
movl 1011
nop
slr r1, r3
sub r2, r2;
// ZERA O VETOR R
st r2, r1
brzi 0010
add r1, r3
nop;
nop
ji 1111
sub r0, r3
```

```
nop;
// R[0] = 11, R[1] = 12, R[2] = R[3] = 23
movh 0001
nop
sub r2, r2
sub r3, r3;
movl 0111
nop
sub r1, r1
nop;
movh 0000
nop
add r2, r0
add r3, r0;
movl 1100
nop
nop
nop;
movl 1011
nop
add r1, r0
nop;
// SOMA VETORES A E B EM R
ld r2, r2
nop
nop
nop;
nop
nop
add r0, r2
sub r2, r2;
nop
ji 0010
nop
nop;
nop
ji 1101
nop
nop;
nop
nop
add r2, r3
add r3, r1;
st r0, r3
nop
sub r0, r0
sub r3, r3;
movl 0001
nop
nop
nop;
nop
nop
add r3, r0
```

```
sub r0, r0;
nop
nop
add r0, r2
sub r2, r3;
nop
nop
sub r0, r1
nop;
nop
brzi 0010
sub r0, r3
sub r3, r3;
nop
ji 1000
add r3, r2
nop;
```

7 Design Final do Processador

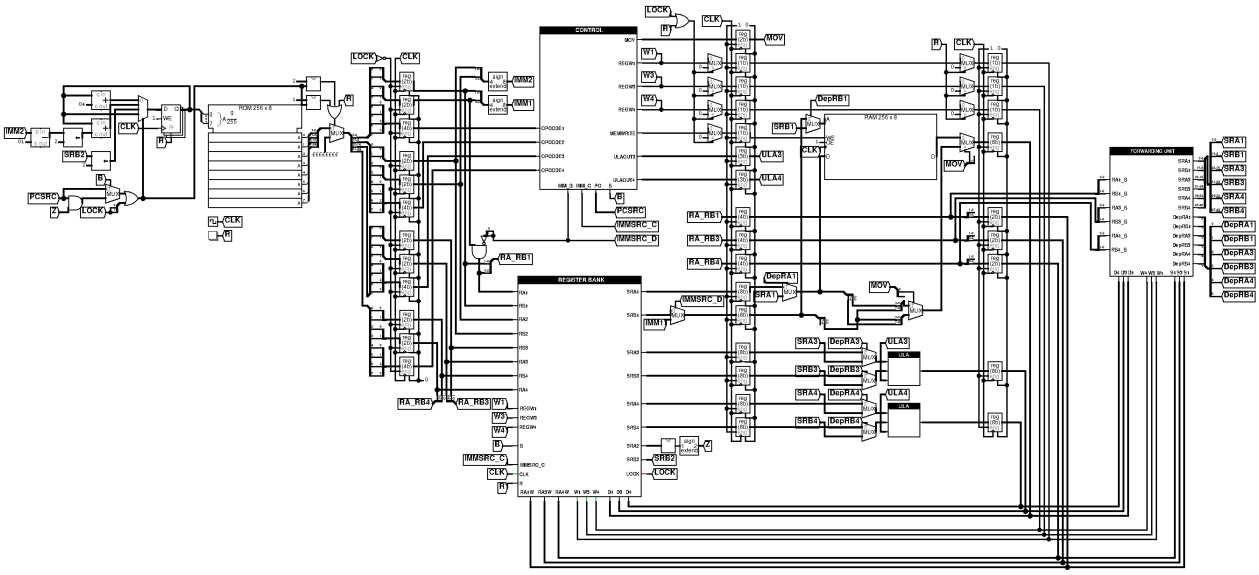

Figure 5: Datapath