Trabalho Prático 2

Filipe de Araujo Mendes - 2021031920 - flipeara@ufmg.br
Thaís Ferreira da Silva - 2021092571 - thaisfds@ufmg.br

Universidade Federal de Minas Gerais (UFMG)
Belo Horizonte - MG - Brasil

1 Introdução

O problema proposto para esse trabalho prático foi a implementação de quatro operações vistas em sala de aula utilizando da linguagem de Descrição de Hardware Verilog, sendo elas multiplicação, divisão, and com imediato e branch beq.

Para a realização desta atividade foi necessário utilizar dos conhecimento adquiridos em sala de aula para alterar um código em python no Google Colab disponibilizado pelo professor contendo o caminho de dados de um processador com pipeline. Além disso, implementamos novos casos de teste e geramos formas de onda para analisar melhor todo o funcionamento do processador apos as modificações realizadas no trabalho.

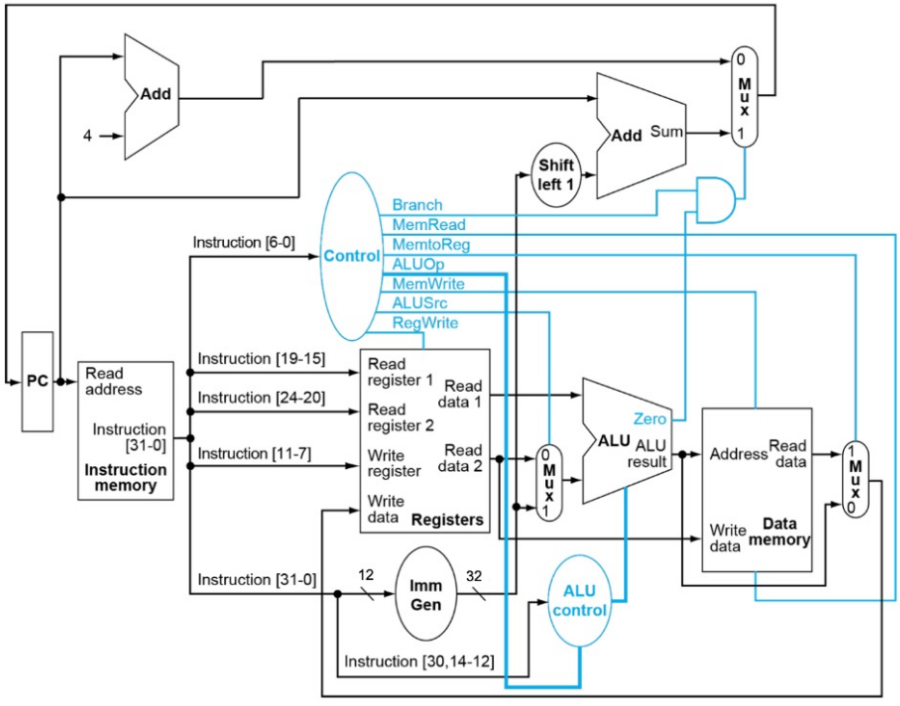

Figura 1: Caminho de dados do processador com pipeline

2 Decisões do Projeto

Tendo como base os slides das aulas e os guias construidos pelos monitores da disciplina, implementamos todas as 4 operações solicitadas nesse segundo trabalho prático de OC1. Segue abaixo uma melhor explicação das modificações realizadas no código para tornar as operações mul, div, andi e beq funcionais no código fornecido para o trabalho:

2.1 mul Rd, Rs1, Rs2 e div Rd, Rs1, Rs2

Na implementação da operação multiplicação e divisão foi explicado que seria necessário a utilização de um bit a mais do f7 totalizando 5 bits. O bit extra utilizado é o f7[0] que é o bit menos significativo e o responsável por definir se a operação é uma multiplicação/divisão ou uma das outras operações já existentes. Dessa forma para adicionar esse novo bit foi necessário modificar o Pipeline Bar e ALU Control and ALU.

- **Pipeline Bar (Decode -> Exec):** Para que o novo modelo fosse capaz de se adaptar ao f7[0] foi necessário adicionar o func7[0] no registrador onde já havia o func7[5] e func3 inicialmente, dessa forma podemos guardar esse dado extra necessário para as operações mul e div.
- **ALU Control and ALU:** Em relação as modificações na ALU, adicionamos as operações mul (4'd3: out <= a * b;) e div (4'd4: out <= a / b;) no case (ctl), pois elas não estavam definidas no processador. Além disso, atualizamos o tamanho de funct para 5 bits alterando a func para input wire [4:0] funct e os valores no ALU CONTROL para que suportasse o novo bit adicionado.

2.2 andi Rd, Rs1, Rs2

Essa foi a operação mais simples de se implementar pois bastou uma única modificação no ALU CONTROL para que a operação andi passa-se a funcionar. Dessa forma, adicionamos no case(funct[2:0]) a linha 3'd7: _functi = 4'd0; responsável pela nova função com imediato.

2.3 beq, Rs1, Rs2, label

Por fim temos a operação de desvio caso o registrador 1 seja igual ao registrador 2 que já fui previamente implementada pelos elaboradores do trabalho. Desta forma foi necessário que adicionássemos a operação na lógica do módulo de controle do RISCV no Control Unit através do opcode, branch_eq, ImmGen e aluop como definido a seguir:

- OPCODE: 1100011
  Opcode é o código da operação beq
- BRANCH_EQ: 1
  O 1 simboliza que é uma operação branch e o 0 que é uma outra operação
- IMMGEN : 19inst[31],inst[31],inst[7],inst[30:25],inst[11:8],1'b0;
  Esse é o gerador de imediato necessário que foi dado pelo monitor
- ALUOP: 1
  O 1 sinaliza que é pra realizar uma operação de subtração entre os registradores, se a subtração for igual a 0 o desvio é realizado.

3 Testes Realizados e Waveform

3.1 Problema 1: MUL

Para o primeiro teste realizamos uma multiplicação simples de 7*8 que deve ter como resultado o número 56 armazenado no registrado x6. Através da Figura 2 abaixo é possível perceber um passo da operação em execução onde no ciclo 7 a operação de 7*8 é calculada na ALU e possui como resultado o número que será posteriormente escrito na memória. Além disso, na Figura 3 podemos analisar mais precisamente o que está ocorrendo no processador como o valor de func7 como sendo 1, a realização da multiplicação no alursit, e a bolha gerada no mul através do opfunc73 pela falta de encaminhamento.

MULTIPLICAÇÃO
addi x5, x0, 7
addi x6, x0, 8
mul x7, x5, x6

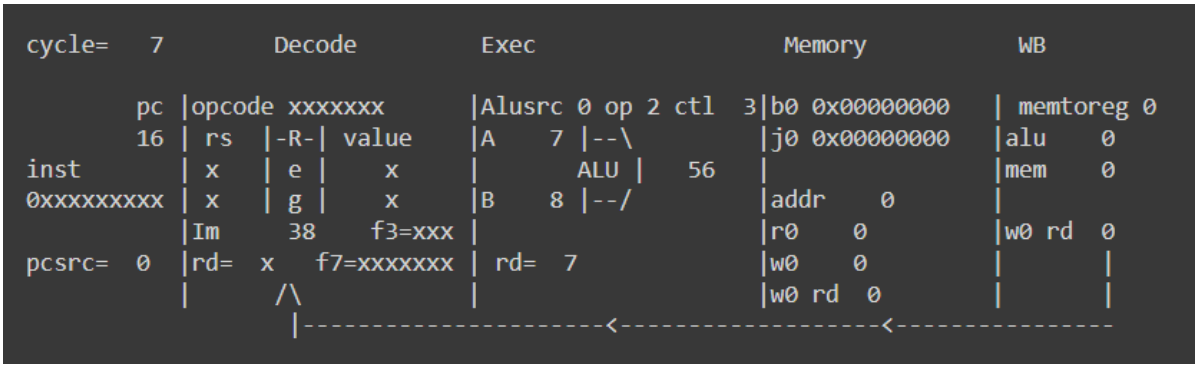

Figura 2: Execução da multiplicação na ALU no 7° ciclo

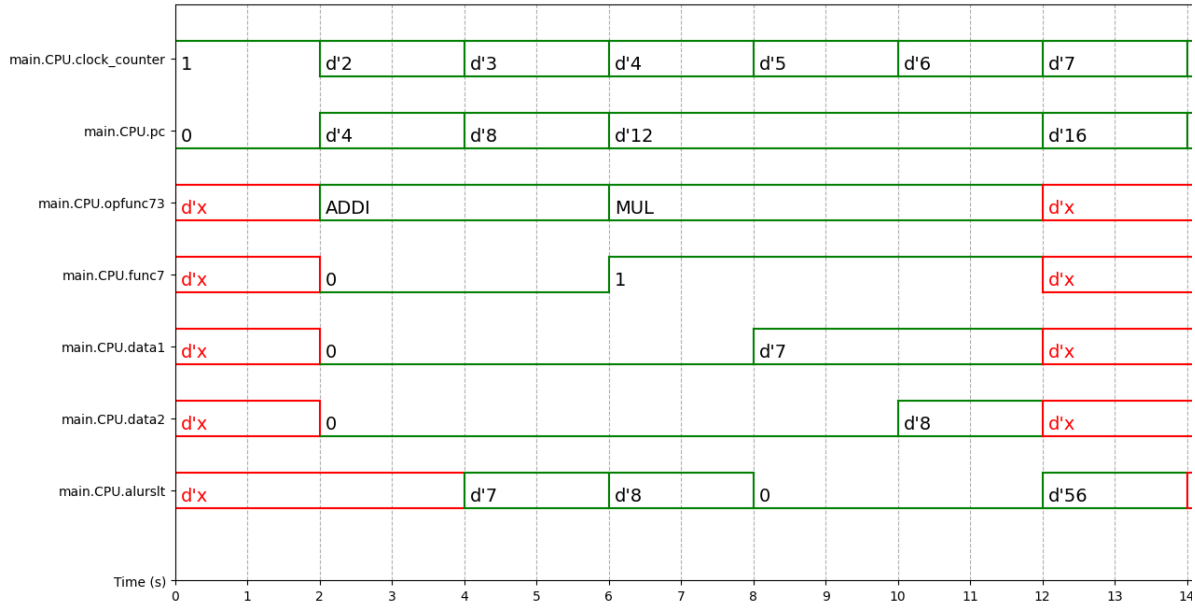

Figura 3: Waveform MUL

3.2 Problema 2: DIV

Semelhantemente a operação de multiplicação no ciclo 7 é possível ver a operação de 20/5 ocorrendo na ALU, o que gera como resultado o número 4 que deve ser armazenado no registrador x6. Além disso, no waveform podemos analisar a operação div ocorrendo, a utilização do novo valor adicionado (func7[0]), e a bolha formada ao longo do processo.

DIVISÃO
addi x5, x0, 20
addi x6, x0, 5
div x7, x5, x6

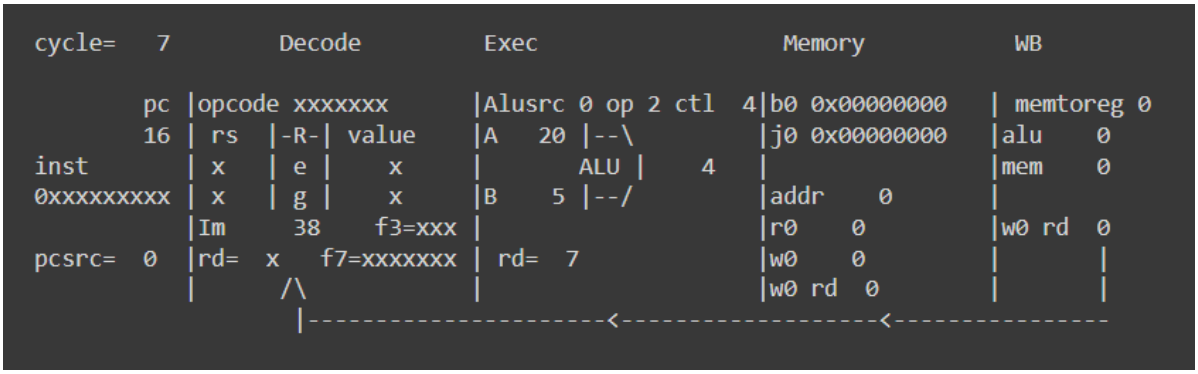

Figura 4: Execução da divisão na ALU no 7° ciclo

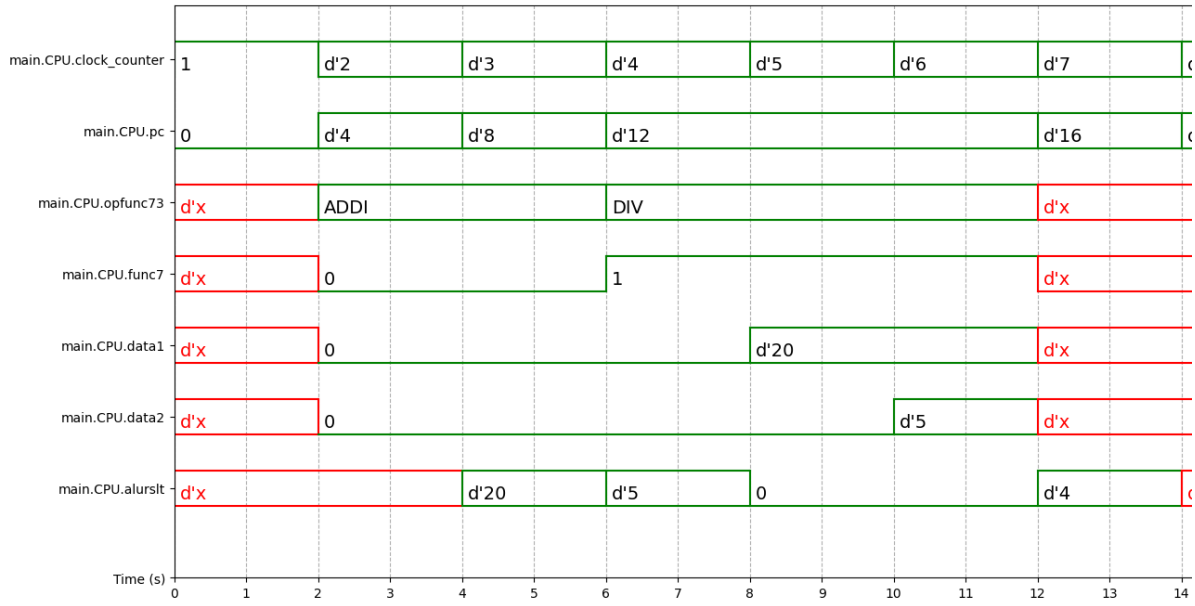

Figura 5: Waveform DIV

3.3 Problema 3: ANDI

No teste da operação and com imediato adicionamos o número 13 ao registrador x5, e depois realizamos a operação andi entre o registrador x5 e 7 adicionando o seu resultado no registrador x6. Novamente o resultado da operação 13&7 pode ser observado no ciclo 7 na imagem abaixo.

1101 (bin) -> 13 (dec)
0110 (bin) -> 07 (dec)

0101 (bin) -> 05 (dec)

Através da figura 7 podemos perceber o imediato 13 (ImmGen) sendo salvo em um registrador (data1), e logo depois sendo comparado com o imediato 7 (ImmGen) na operação de AND que tem como resultado 5 (alurslt).

AND COM IMEDIATO
addi x5, x0, 13
andi x6, x5, 7

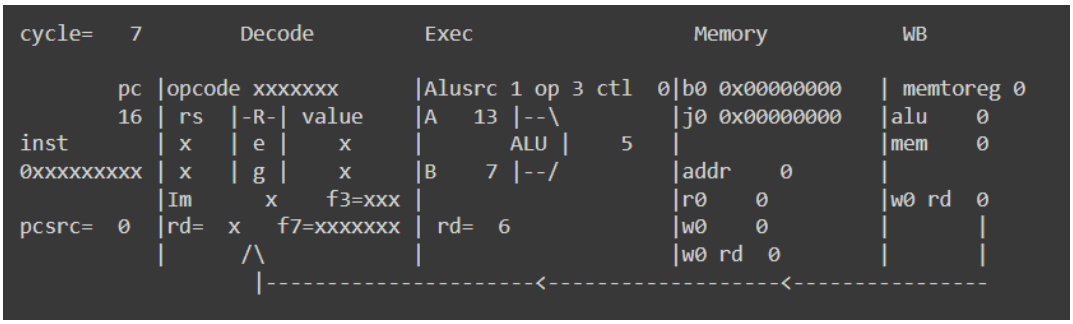

Figura 6: Execução do ANDI no 7° ciclo

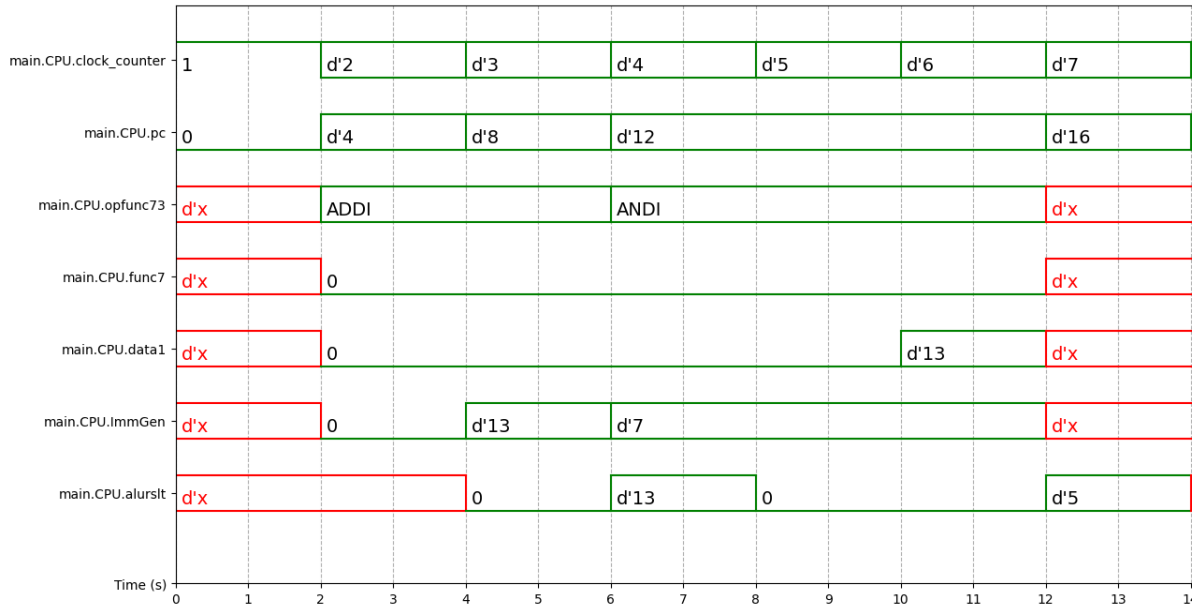

Figura 7: Waveform ANDI

3.4 Problema 4: BEQ

Por fim o teste de desvio condicional onde podemos analisar o desvio ocorrendo através da soma final realizada. Neste caso definimos x5 e x6 como sendo 5 e colocamos aluop como 1 (observado no ciclo de clock 8 na waveform), para indicar que deve ser feita uma subtração dos dois valores. Se os valores forem iguais, a subtração resulta em 0, fazendo com que um desvio de 8 bits seja tomado, a partir do PC da instrução BEQ, que é 12. Assim, o PC é alterado para 12 + 8 = 20 e todo o trabalho que estava sendo feito até o desvio ser tomado é descartado, como o cálculo da ALU no clock 9, que resultou em 10. No ciclo 10 de clock, a instrução de PC 20 é lida e todos os outros estágios da pipeline são descartados, evidenciado por não haver instrução no estado ID. Então, a execução é retomada e a soma 5 + 7 é feita, gravando o número 12 no x5 ao final (observe que se o desvio não tivesse sido tomado, o resultado seria 17).

BRANCH BEQ

```
04:    addi x5, x0, 5
08:    addi x6, x0, 5
12:    beq x5, x6, 8
16:    addi x5, x5, 5
20:    addi x5, x5, 7
```

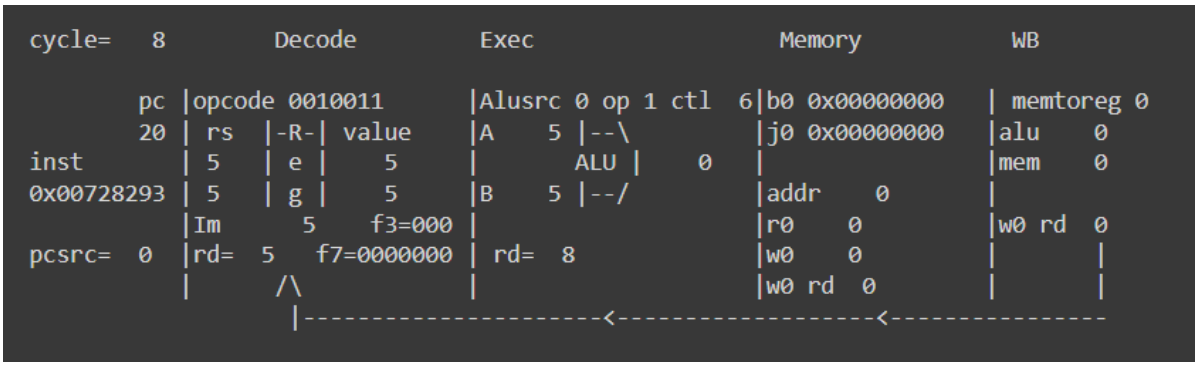

Figura 8: Verificação dos valores dos registrador x5 com o x6

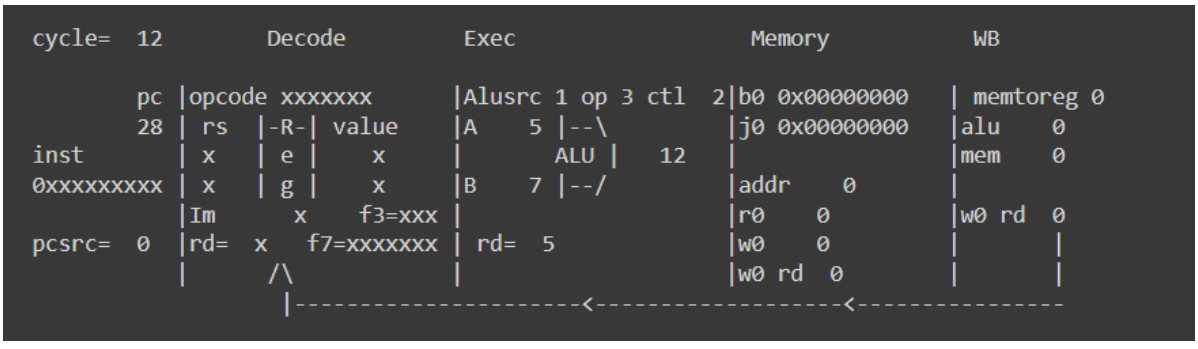

Figura 9: Execução da soma na ALU após o desvio

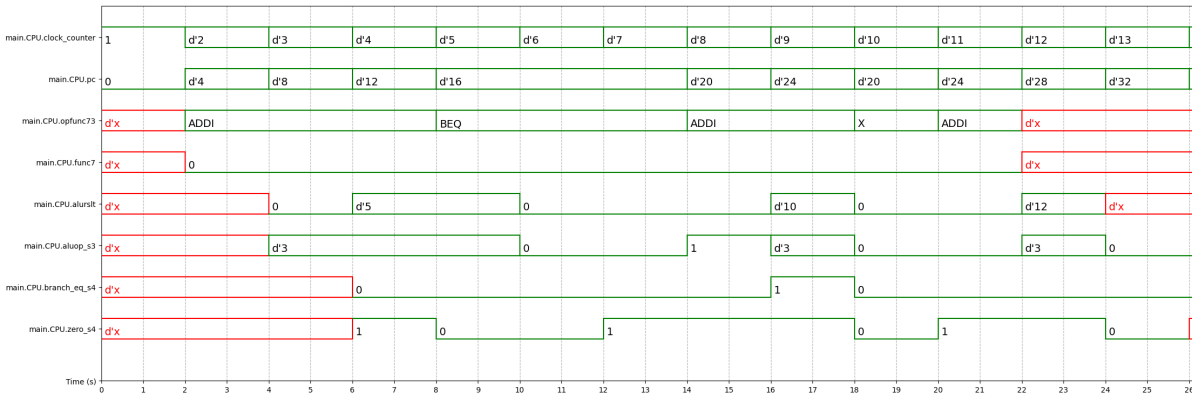

Figura 10: Waveform BEQ