Lista 2 - Soluções

Arquitetura e Organização de Computadores

Maio 2024

Qual número é armazenado como `0x49179EA0` conforme a norma IEEE 754? `62.1034E4`

Quais valores hexadecimais podem representar o número `3.462408E-11` conforme a norma IEEE 754?

$3.462408E-11 = 1.18967433003270144 * 2^{-35}$
MSb $= 0$
exp $= 127 - 35 = 92 = 0b01011100$
$16 * 0.18967433003270144 = 3.03478928052322304 \rightarrow 3$
$16 * 0.03478928052322304 = 0.55662848837156864 \rightarrow 0$
$16 * 0.55662848837156864 = 8.90605581394509824 \rightarrow 8$
$16 * 0.90605581394509824 = 14.49689302312157184 \rightarrow E$
$16 * 0.49689302312157184 = 7.95028836994514944 \rightarrow 7$
$16 * 0.95028836994514944 = 15.20461391912239104 \rightarrow F$
$0x308E7F = 0b00110000100011001111111$
concatena(MSb, exp, mantissa) $= 0b00101110000110000100011100111111$
se descarta o bit menos significativo,
$0b00101110000110000100011100111111$
$0x2E18473F$
se arredonda o bit menos significativo,
$0b00101110000110000100011101000000$
$0x2E184740$

`0x2E184740` e `0x2E18473F`

Traduza o programa abaixo, em C, para MIPS.

```
int factorial(int n) {
        if(n == 0) return 1;
        return n * factorial(n - 1);
}

void main()  {
        int num = 5;
        int fact = factorial(num);
}
```

`https://github.com/Eduardo-bat/AOC_2024/blob/main/factorial.asm`

Ao fim da execução do programa abaixo, quais os valores em `$sp` e `$s1`? Qual o menor valor armazenado, ao longo da execução deste programa, em `$sp`?

```
j main

recursive:
lw $t0, 0($sp)
addi $sp, $sp, 4

bne $t0, $0, else
addi $sp, $sp, -4
sw $t0, 0($sp)
jr $ra

else:
addi $t0, $t0, -1
addi $sp, $sp, -8
sw $t0, 0($sp)
sw $ra, 4($sp)
jal recursive
lw $t0, 0($sp)
lw $ra, 4($sp)
addi $sp, $sp, 8
addi $t0, $t0, 1
addi $sp, $sp, -4
sw $t0, 0($sp)
jr $ra

main:
addi $s0, $0, 0xABCD
addi $sp, $sp, -4
sw $s0, 0($sp)
jal recursive
lw $s1, 0($sp)
addi $sp, $sp, 4
```

https://github.com/Eduardo-bat/AOC_2024/blob/main/recursive.asm

Traduza o programa abaixo, em C, para MIPS.

```c
#include <math.h>

float fop(float a, float b) {
    if(b == 0) return NAN;
    return a*b + a/b;
}

void main() {
    float a = 40.1;
    float b = 00.4;
    float c = fop(a, b);
}
```

`https://github.com/Eduardo-bat/AOC_2024/blob/main/float.asm`

Como `PC` é tratado no estágio **Fetch**? Em quais condições seu valor não pode ser atualizado?

Em cada estágio **Fetch**, `PC'` é armazenado no registrador de `PC` e transmitido à memória de instruções. Além disso, seu valor é incrementado e o resultado é disponibilizado para o multiplexador que define o próximo `PC'` e para o registrador F/D. Quando a necessidade de um stall é detectada no estágio **Decode** de uma instrução, o **Fetch** da próxima deve ser postergado para o próximo ciclo. Neste caso, a escrita no registrador de `PC` é bloqueada. Isto mantém impede que `PC` seja incrementado por um ciclo de clock (o mesmo em que a necessidade de stall é detectada), reabilitando o fluxo normal no próximo.

Em qual estágio a tratativa de data hazards para `lw` é feita? Qual condição é verificada nesta tratativa? Quando este tipo de hazard é detectado, como é tratado?

A tratativa de data hazards para `lw` é feita no estágio **Decode**. Neste, `rs` e `rt` são transmitidos à unidade de hazards, que,
- para cada endereço, se este for igual ao endereço de destino da instrução anterior **e** a instrução anterior for `lw`, mantém a instrução neste estágio, impedindo a escrita no registrador de `PC` e no registrador F/D e forçando todos os sinais em D/E para 0.

Descreva como, no estágio **Execute**, é feita a avaliação e tratativa de hazards.

Os endereços dos registradores usados nesta instrução são transmitidos à unidade de hazards, que,
- para cada endereço, se este for igual ao endereço de destino de alguma das duas instruções anteriores,
— se a última instrução modifica o banco de registradores, seleciona o resultado desta como valor correspondente ao endereço,
— senão, se a penúltima instrução modifica o banco de registradores, seleciona o resultado desta como valor correspondente ao endereço,
- senão, seleciona o valor contido no banco como valor correspondente ao endereço.

O endereço do registrador de destino e os sinais de controle da memória são transmitidos à unidade de hazards, para permitir decisões em instruções posteriores;

Quais sinais, no estágio **Memory**, contribuem para a detecção ou tratativa de hazards? Como estes são avaliados ou utilizados?

Neste estágio, `ALUOutM` é disponibilizado para ser usado como operando no estágio **Execute** da próxima instrução e para comparação no estágio **Decode** da posterior a esta. Além disso, `WriteRegM` e `RegWriteM` são transmitidos à hazard unit para avaliação de hazards no estágio **Execute** da próxima instrução.

Quais sinais, no estágio **WriteBack**, contribuem para a detecção ou tratativa de hazards? Como estes são avaliados ou utilizados?

Neste estágio, `ALUOutW` é disponibilizado para ser usado como operando no estágio **Execute** da instrução que sucede a próxima. Além disso, `WriteRegW` e `RegWriteW` são transmitidos à hazard unit para avaliação de hazards no estágio **Execute** desta.

Em qual estágio de uma instrução `beq` os hazards pertinentes a ela são detectados? Como estes são avaliados e tratados?

Os hazards pertinentes à instrução `beq` são detectados no estágio **Decode** desta:
- se a última instrução for `lw` e esta atualizar um dos endereços fonte para a comparação, `beq` é mantida em **Decode** por dois ciclos de clock, já que a instrução anterior estará em **Execute** e o valor só estará disponível em **WriteBack**;
- senão, se a última instrução for de tipo lógica/aritmética e esta atualizar um dos endereços fonte para a comparação ou a penúltima for `lw` e esta atualizar um dos endereços fonte para a comparação, `beq` é mantida em **Decode** por um ciclo de clock;
- senão, se a penúltima instrução for lógica/aritmética e esta atualizar um dos endereços fonte para a comparação, `ALUOutM` substitui o valor armazenado no endereço coincidente;

Em um processador pipeline, o que significa, "limpar" ou "resetar" um registrador de pipeline? Como isso impede a continuação da instrução a partir do estágio em que ocorre

"Limpar" ou "resetar" um registrador de pipeline corresponde a forçar o valor 0 em todos os sinais armazenados nele. Isso impede a propagação da instrução porque desabilita a escrita em todos os componentes posteriores e acessa, restritamente, o endereço 0 do banco de registradores, habilitado somente para leitura.

No processador pipeline implementado no livro de referência, é possível realizar forwarding do estágio **WriteBack** para o estágio **Memory**? Em qual condição isto é necessário? Qual a justificativa para a decisão tomada na implementação apresentada pelo livro?

Não é possível. Quando uma `sw` é executada logo após uma instrução lógica/aritmética que escreve no registrador a ser armazenado na memória. Não sei a justificativa.

```
i00: addi $s0, $0, 10
i01: sw   $s0, 0($sp)
i02: addi $s0, $0, 2
i03: addi $s1, $0, 3
i04: add  $s2, $s1, $s0
i05: sub  $s1, $s2, $s0
i06: lw   $s3, 0($sp)
i07: add  $s4, $s3, $s3
i08: beq  $s4, $s3, one
i09: add  $s0, $0, $0
i10: add  $s1, $0, $0
i11: add  $s2, $0, $0
i12: add  $s3, $0, $0
i13: beq  $0, $0, zero
one:
i14: add  $s0, $0, 1
i15: add  $s1, $0, 1
i16: add  $s2, $0, 1
i17: add  $s3, $0, 1
zero:
i18: add  $s6, $s2, $s3
i19: add  $s5, $s0, $s1
i20: add  $s7, $s6, $s5
```

Considerando uma arquitetura que detecte hazards, empregue stall e previsão de desvio (assume que não é tomado), mas não empregue forwarding, indique, na tabela abaixo, em qual estágio cada instrução está em cada ciclo de clock.

inst	C0	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	C11	C12	C13	C14	C15
i0:	F	D	E	M	W											
i1:		F	D	D	D	E	M	W								
i2:					F	D	E	M	W							
i3:						F	D	E	M	W						
i4:							F	D	D	D	E	M	W			
i5:										F	D	D	D	E	M	W
inst	C12	C13	C14	C15	C16	C17	C18	C19	C20	C21	C22	C23	C24	C25	C26	C27
i6:	F	D	E	M	W											
i7:		F	D	D	D	E	M	W								
i8:					F	D	D	D	E	M	W					
i9:								F	D	E	M	W				
i10:									F	D	E	M	W			
i11:										F	D	E	M	W		
i12:											F	D	E	M	W	
inst	C23	C24	C25	C26	C27	C28	C29	C30	C31	C32	C33	C34	C35	C36	C37	C38
i13:	F	D	E	M	W											
i14:		F	-	-	-	-										
i18:			F	D	E	M	W									
i19:				F	D	E	M	W								
i20:					F	D	D	D	E	M	W					

Considerando uma arquitetura que detecte hazards e empregue forwarding, stall e previsão de desvio (assume que não é tomado) indique, na tabela abaixo, em qual estágio cada instrução está em cada ciclo de clock. Neste exercício, não se restrinja ao processador implementado no livro. Indique os recursos que achar pertinentes.

No processador que decidi usar, há forwarding WB→MEM para tratar instruções `sw` antecedidas diretamente por instruções que escrevem no registrador a ser armazenado. Uma seta vertical indica que o estágio é fonte de um forwarding. Uma seta horizontal indica que o estágio recebe forwarding.

inst	C0	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	C11	C12	C13	C14	C15
i0:	F	D	E	M	↓W											
i1:		F	D	E	→M	W										
i2:			F	D	E	M	↓W									
i3:				F	D	E	↓M	W								
i4:					F	D	⇉E	↓M	W							
i5:						F	D	→E	M	W						
i6:							F	D	E	M	↓W					
i7:								F	D	D	→E	↓M	W			
i8:										F	D	→D	E	M	W	
i9:												F	D	E	M	W
inst	C12	C13	C14	C15	C16	C17	C18	C19	C20	C21	C22	C23	C24	C25	C26	C27
i10:	F	D	E	M	W											
i11:		F	D	E	M	W										
i12:			F	D	E	M	W									
i13:				F	D	E	M	W								
i14:					F	-	-	-	-							
i18:						F	D	E	M	↓W						
i19:							F	D	E	↓M	W					
i20:								F	D	⇉E	M	W				

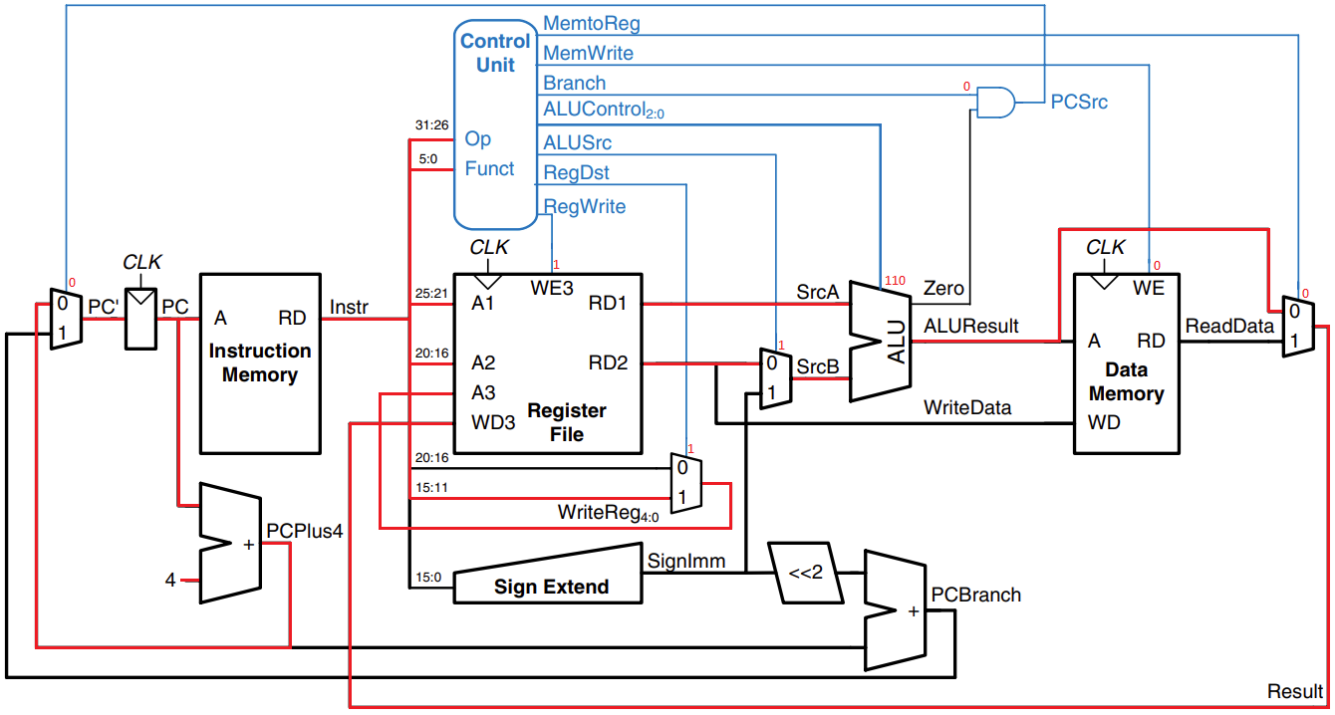
Reorganize o programa a seguir a fim de evitar hazards sem alterar seu estado final.

```
or   $t2, $t1, $t0
nor  $t9, $t8, $t7
addi $s2, $s1, 1
add  $s2, $s1, $s0
sub  $s4, $s3, $s2
xor  $t2, $t1, $t0
sw   $t2, 0($sp)
lw   $t9, 4($sp)
subi $t0, $t0, 1
and  $t7, $t8, $t9
```

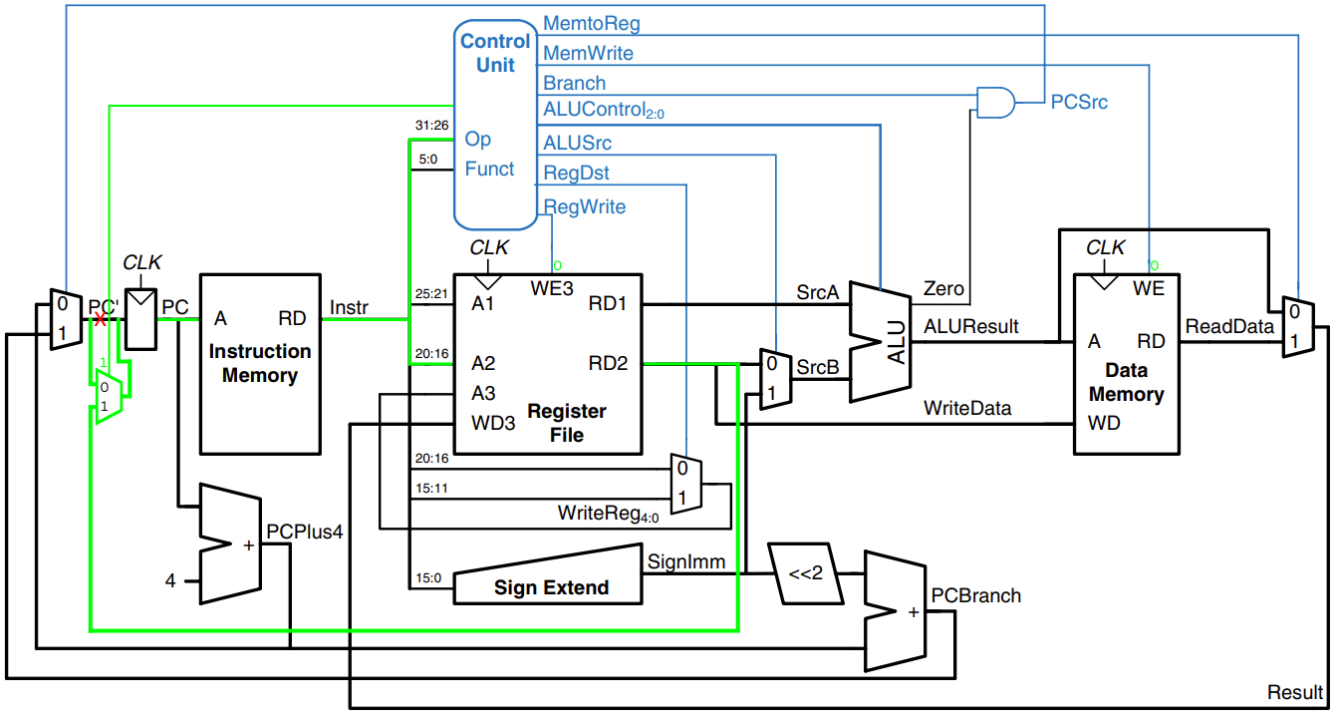
```
nor  $t9, $t8, $t7
lw   $t9, 4($sp)
addi $s2, $s1, 1
add  $s2, $s1, $s0
or   $t2, $t1, $t0
xor  $t2, $t1, $t0
subi $t0, $t0, 1
and  $t7, $t8, $t9
sub  $s4, $s3, $s2
sw   $t2, 0($sp)
```

Mostre como as seguintes instruções são executadas em um processador de ciclo único. Para isso, indique quais sinais são transferidos entre quais componentes.

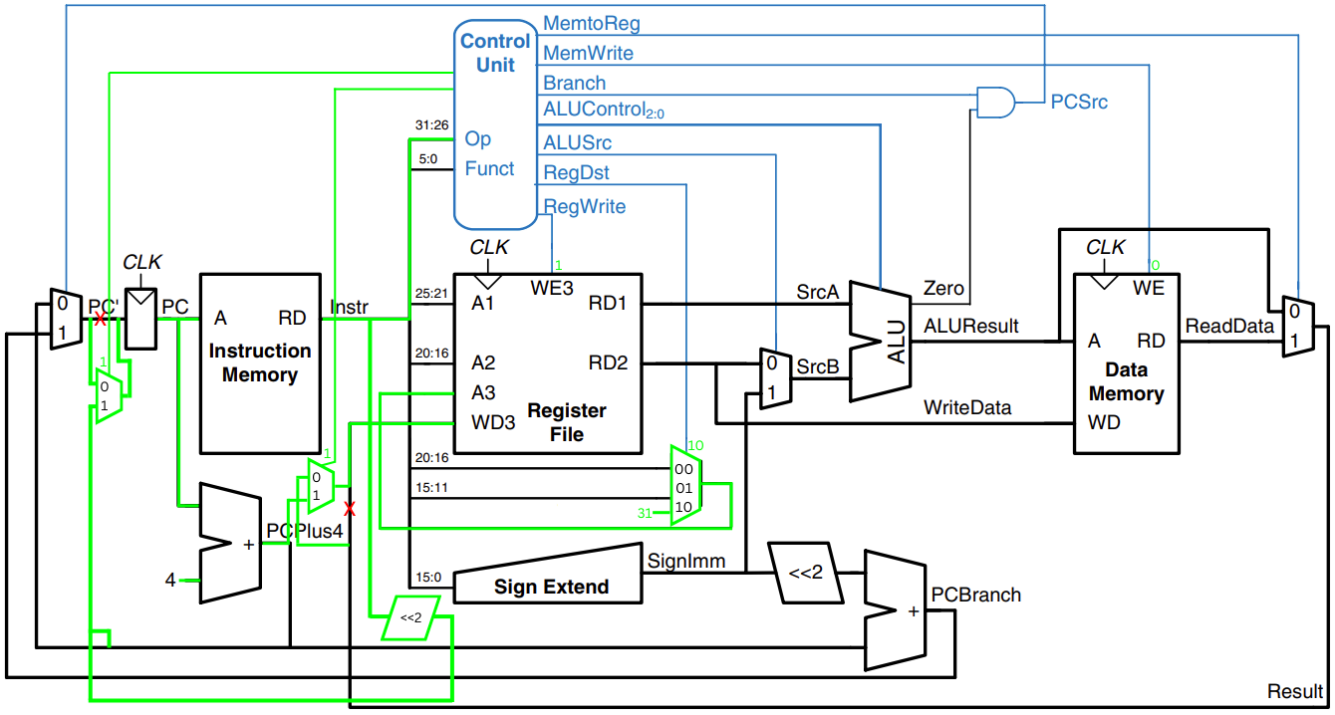
1. sub


Control Unit
MemtoReg
MemWrite
Branch
ALUControl2:0
ALUSrc
RegDst
RegWrite
Op
Funct
31:26
5:0
PCSrc
CLK
PC'
PC
A
RD
Instruction Memory
Instr
25:21
20:16
A1
A2
A3
WD3
WE3
RD1
RD2
Register File
SrcA
SrcB
ALU
Zero
ALUResult
WriteData
WE
Data Memory
WD
ReadData
15:11
WriteReg4:0
PCPlus4
4
15:0
Sign Extend
SignImm
<<2
PCBranch
Result

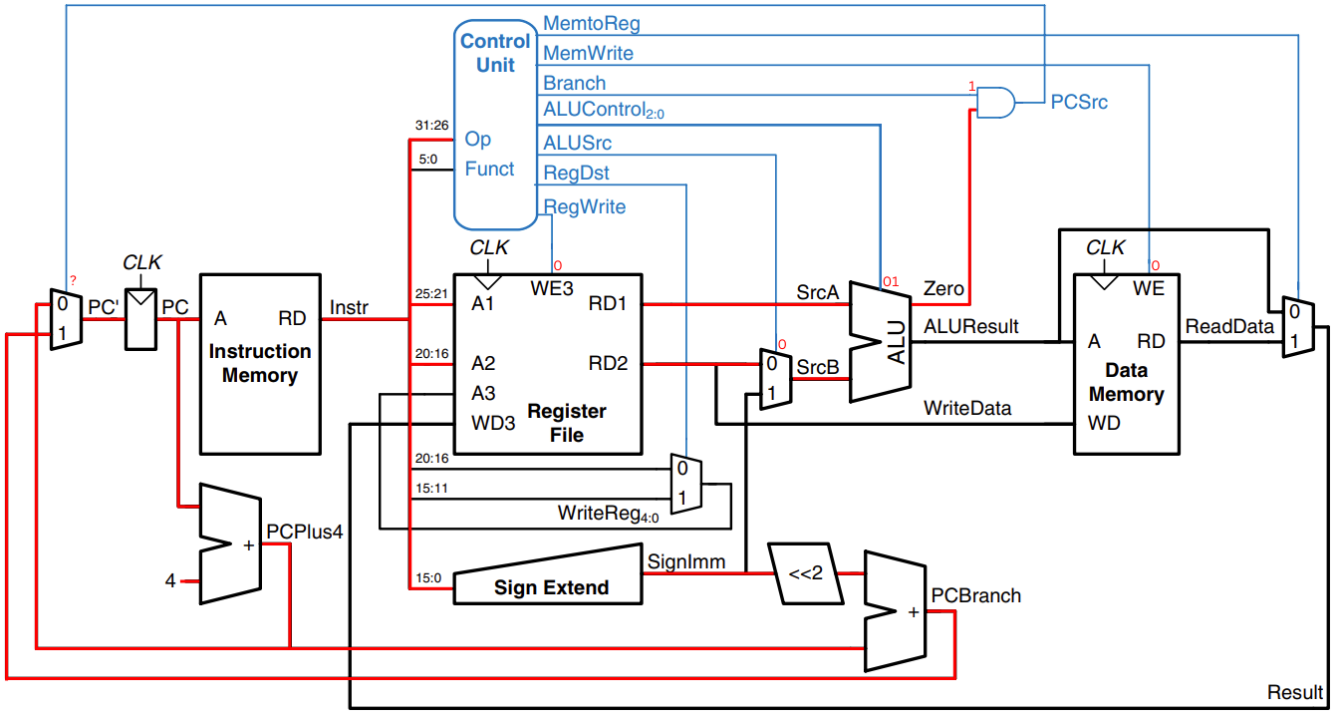
2. jr


Control Unit
MemtoReg
MemWrite
Branch
ALUControl2:0
ALUSrc
RegDst
RegWrite
Op
Funct
31:26
5:0
PCSrc
CLK
PC'
PC
A
RD
Instruction Memory
Instr
25:21
20:16
A1
A2
A3
WD3
WE3
RD1
RD2
Register File
SrcA
SrcB
ALU
Zero
ALUResult
WriteData
WE
Data Memory
WD
ReadData
15:11
WriteReg4:0
PCPlus4
4
15:0
Sign Extend
SignImm
<<2
PCBranch
Result

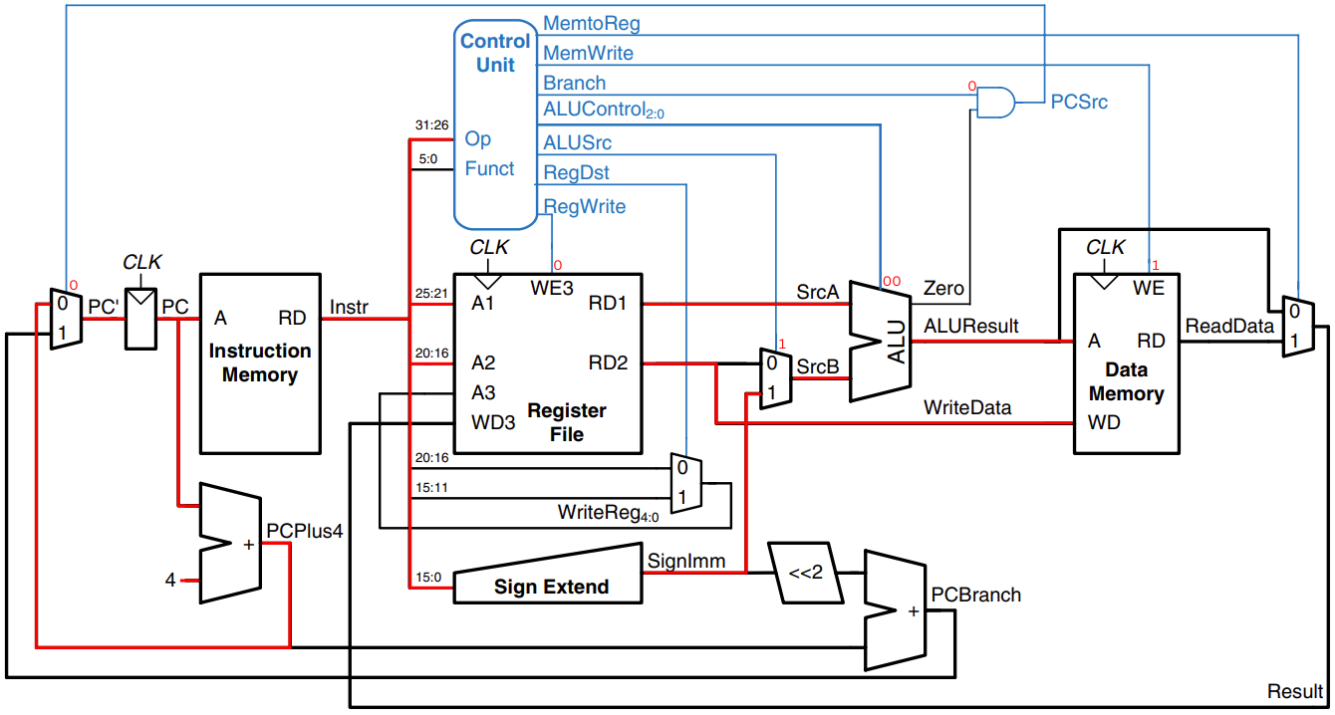
3. jal


Control Unit
MemtoReg
MemWrite
Branch
ALUControl2:0
ALUSrc
RegDst
RegWrite
Op
Funct
31:26
5:0
PCSrc
CLK
PC'
PC
A
RD
Instruction Memory
Instr
25:21
20:16
15:11
15:0
A1
A2
A3
WD3
WE3
RD1
RD2
Register File
SrcA
SrcB
ALU
Zero
ALUResult
WriteData
WE
WD
Data Memory
ReadData
PCPlus4
4
Sign Extend
SignImm
<<2
PCBranch
Result

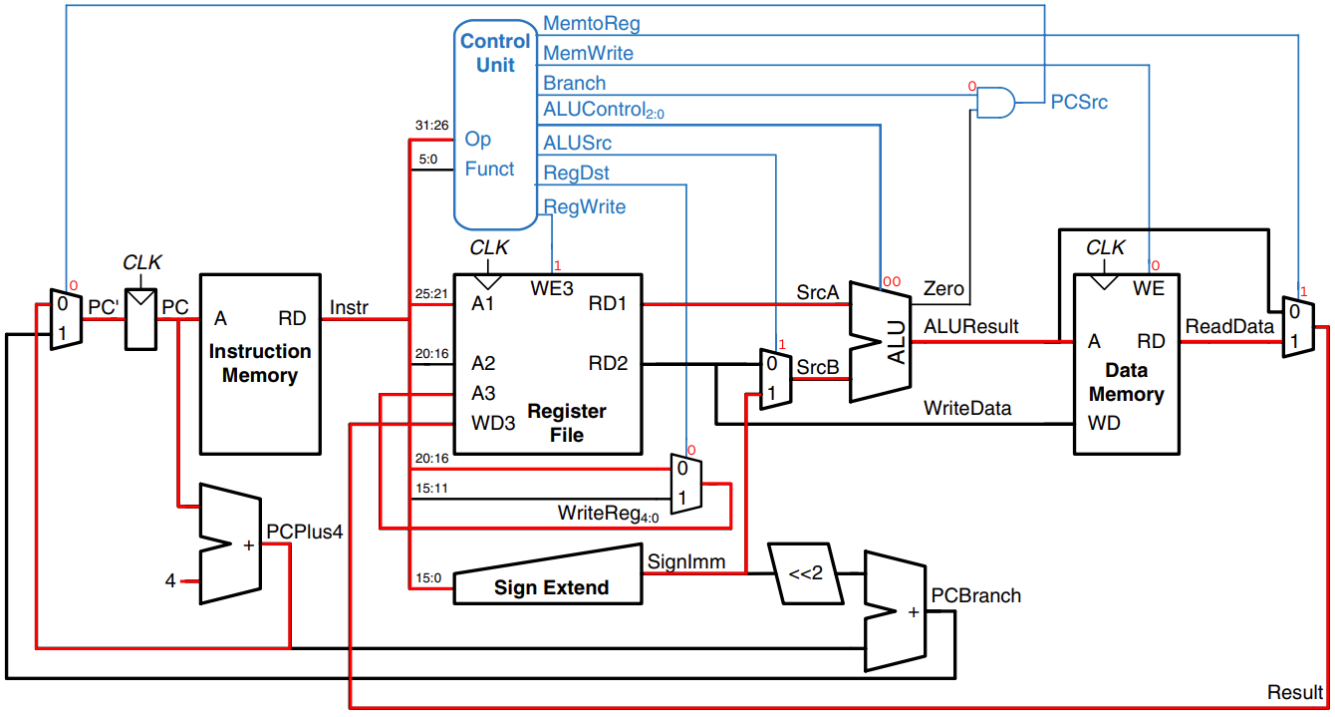
4. beq


Control Unit
MemtoReg
MemWrite
Branch
ALUControl2:0
ALUSrc
RegDst
RegWrite
Op
Funct
31:26
5:0
PCSrc
CLK
PC'
PC
A
RD
Instruction Memory
Instr
25:21
20:16
15:11
15:0
A1
A2
A3
WD3
WE3
RD1
RD2
Register File
SrcA
SrcB
ALU
Zero
ALUResult
WriteData
WE
WD
Data Memory
ReadData
PCPlus4
4
WriteReg4:0
Sign Extend
SignImm
<<2
PCBranch
Result

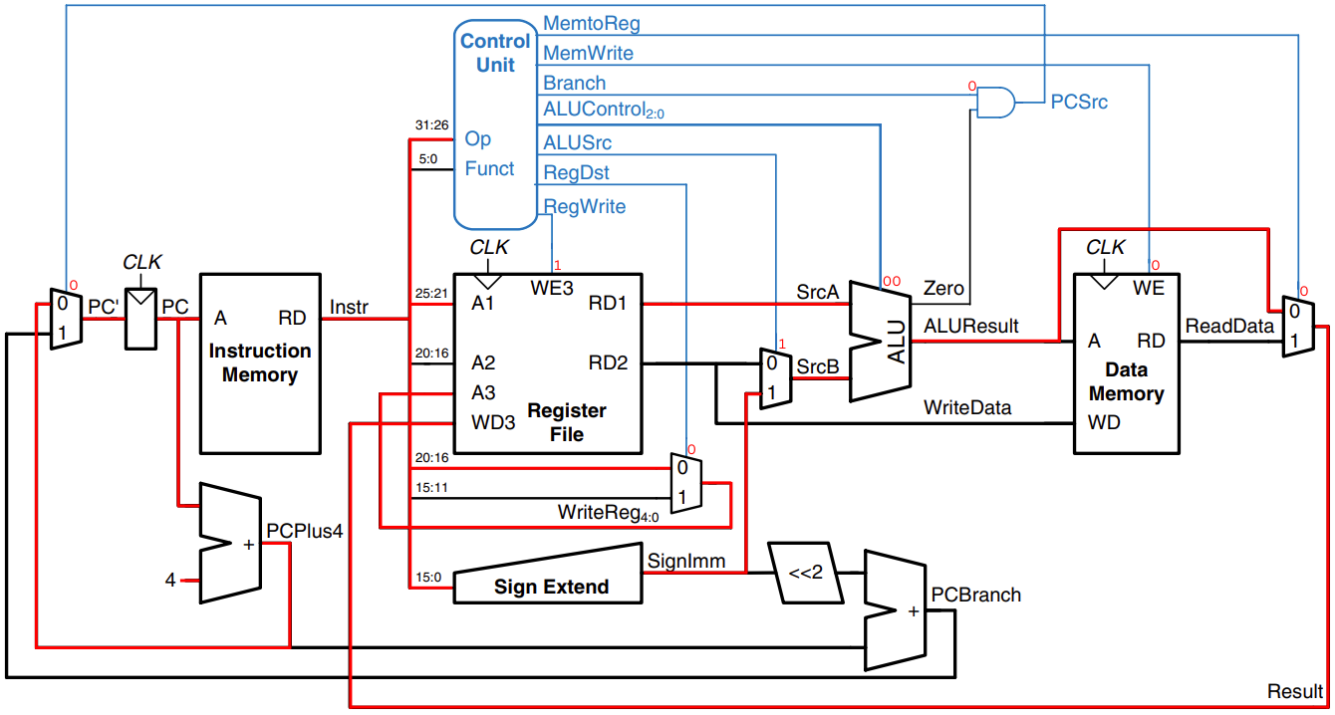
5. sw


Control Unit
MemtoReg
MemWrite
Branch
ALUControl2:0
ALUSrc
RegDst
RegWrite
Op
Funct
31:26
5:0
PCSrc
CLK
PC'
PC
A
RD
Instruction Memory
Instr
25:21
20:16
A1
A2
A3
WD3
WE3
RD1
RD2
Register File
SrcA
SrcB
ALU
Zero
ALUResult
WriteData
WE
Data Memory
WD
ReadData
15:11
WriteReg4:0
PCPlus4
4
15:0
Sign Extend
SignImm
<<2
PCBranch
Result

6. lw


Control Unit
MemtoReg
MemWrite
Branch
ALUControl2:0
ALUSrc
RegDst
RegWrite
Op
Funct
31:26
5:0
PCSrc
CLK
PC'
PC
A
RD
Instruction Memory
Instr
25:21
20:16
A1
A2
A3
WD3
WE3
RD1
RD2
Register File
SrcA
SrcB
ALU
Zero
ALUResult
WriteData
WE
Data Memory
WD
ReadData
15:11
WriteReg4:0
PCPlus4
4
15:0
Sign Extend
SignImm
<<2
PCBranch
Result

7. addi



Descreva, temporalmente, como as seguintes instruções são executadas em um processador multiciclo. Para isso, indique quais sinais são transferidos entre quais componentes.

1. sub

$$1\ \mathrm{PC_{reg}} \xrightarrow{\mathrm{PC}} \mathrm{Adr_{multiplexer}} \xrightarrow{\mathrm{Adr}} \mathrm{Mem} \xrightarrow{\mathrm{Instr/Data}} \mathrm{InstReg},\ \mathrm{CtrlUnit} \begin{array}{l} \xrightarrow{\mathrm{PC_{write}}} \mathrm{PCEn_{OR}} \xrightarrow{\mathrm{PCEn}} \mathrm{PC_{reg}} \\ \xrightarrow{\mathrm{PCSrc}} \mathrm{PCSrc_{multiplexer}} \\ \xrightarrow{\mathrm{IR_{write}}} \mathrm{InstReg} \\ \xrightarrow{\mathrm{IorD}} \mathrm{Adr_{multiplexer}} \\ \xrightarrow{\mathrm{ALUSrcA}} \mathrm{SrcA_{multiplexer}} \\ \xrightarrow{\mathrm{ALUSrcB}} \mathrm{SrcB_{multiplexer}} \\ \xrightarrow{\mathrm{PCSrc}} \mathrm{PCSrc_{multiplexer}} \end{array},$$

$$\mathrm{PC_{reg}} \xrightarrow{\mathrm{PC}} \mathrm{SrcA_{multiplexer}} \xrightarrow{\mathrm{SrcA}} \mathrm{ALU} \xrightarrow{\mathrm{PC+4}} \mathrm{PCSrc_{multiplexer}} \xrightarrow{\mathrm{PC+4}} \mathrm{PC_{reg}}$$
$$4 \xrightarrow{4} \mathrm{SrcB_{multiplexer}} \xrightarrow{\mathrm{SrcB}}$$

$$2\ \mathrm{InstReg} \xrightarrow{\mathrm{Funct\ \&\ Op}} \mathrm{CtrlUnit}$$
$$\xrightarrow{\mathrm{rt\ \&\ rs}} \mathrm{RegFile} \xrightarrow{\mathrm{rd1\ \&\ rd2}} \mathrm{RegFileData_{reg}}$$

$$3\ \mathrm{RegFileData_{reg}} \begin{array}{l} \xrightarrow{\mathrm{A}} \mathrm{SrcA_{multiplexer}} \\ \xrightarrow{\mathrm{B}} \mathrm{SrcB_{multiplexer}} \end{array} \begin{array}{l} \xrightarrow{\mathrm{SrcA}} \\ \xrightarrow{\mathrm{SrcB}} \end{array} \mathrm{ALU} \xrightarrow{\mathrm{ALURes}} \mathrm{ALUOut_{reg}},\ \mathrm{CtrlUnit} \begin{array}{l} \xrightarrow{\mathrm{ALUCtrl}} \mathrm{ALU} \\ \xrightarrow{\mathrm{ALUSrcA}} \mathrm{SrcA_{multiplexer}} \\ \xrightarrow{\mathrm{ALUSrcB}} \mathrm{SrcB_{multiplexer}} \end{array}$$

$$4\ \mathrm{ALUOut_{reg}} \xrightarrow{\mathrm{ALUOut}} \mathrm{WD3_{multiplexer}} \xrightarrow{\mathrm{ALUOut}} \mathrm{WD3_{RegFile}},\ \mathrm{CtrlUnit} \begin{array}{l} \xrightarrow{\mathrm{RegDst}} \mathrm{RegDst_{multiplexer}} \\ \xrightarrow{\mathrm{MemtoReg}} \mathrm{WD3_{multiplexer}} \\ \xrightarrow{\mathrm{RegWrite}} \mathrm{RegFile} \end{array}$$

2. jr

3.
CtrlUnit -> {PCSrc} -> PCSrcMult;

CtrlUnit -> {PCWrite} -> PCreg;
RegFileReg -> {A} -> PCSrcMult -> {A} -> PCReg;

3. jal

2.
CtrlUnit -> {PCSrc} -> PCSrcMult;
CtrlUnit -> {PCWrite} -> PCreg;
{PC[31:28] concat Imm < < 2} -> PCSrcMult;
CtrlUnit -> {WriteSrc} -> WD3Mult;
{31} -> WD3Mult -> RegFile

4. beq (interessante: o endereço de desvio é computado em Decode e a igualdade verificada no execute)

$$1\ \mathrm{PC_{reg}} \xrightarrow{\mathrm{PC}} \mathrm{Adr_{multiplexer}} \xrightarrow{\mathrm{Adr}} \mathrm{Mem} \xrightarrow{\mathrm{Instr/Data}} \mathrm{InstReg},\ \mathrm{CtrlUnit} \begin{array}{l} \xrightarrow{\mathrm{PC_{write}}} \mathrm{PCEn_{OR}} \xrightarrow{\mathrm{PCEn}} \mathrm{PC_{reg}} \\ \xrightarrow{\mathrm{PCSrc}} \mathrm{PCSrc_{multiplexer}} \\ \xrightarrow{\mathrm{IR_{write}}} \mathrm{InstReg} \\ \xrightarrow{\mathrm{IorD}} \mathrm{Adr_{multiplexer}} \\ \xrightarrow{\mathrm{ALUSrcA}} \mathrm{SrcA_{multiplexer}} \\ \xrightarrow{\mathrm{ALUSrcB}} \mathrm{SrcB_{multiplexer}} \\ \xrightarrow{\mathrm{PCSrc}} \mathrm{PCSrc_{multiplexer}} \end{array},$$

$$\begin{array}{r} \mathrm{PC_{reg}} \xrightarrow{\mathrm{PC}} \mathrm{SrcA_{multiplexer}} \\ 4 \xrightarrow{4} \mathrm{SrcB_{multiplexer}} \end{array} \xrightarrow{\begin{array}{c}\mathrm{SrcA}\\ \mathrm{SrcB}\end{array}} \mathrm{ALU} \xrightarrow{\mathrm{PC+4}} \mathrm{PCSrc_{multiplexer}} \xrightarrow{\mathrm{PC+4}} \mathrm{PC_{reg}}$$

$$2\ \mathrm{InstReg} \begin{array}{l} \xrightarrow{\mathrm{Funct\ \&\ Op}} \mathrm{CtrlUnit} \\ \xrightarrow{\mathrm{rt\ \&\ rs}} \mathrm{RegFile} \xrightarrow{\mathrm{rd1\ \&\ rd2}} \mathrm{RegFileData_{reg}} \\ \xrightarrow{\mathrm{Imm}} \mathrm{ImmExt} \xrightarrow{\mathrm{SignImm}} \mathrm{ImmShift} \xrightarrow{\mathrm{ShiftImm}} \mathrm{SrcB_{multiplexer}} \end{array}, \mathrm{PC_{reg}} \xrightarrow{\mathrm{PC}} \mathrm{SrcA_{multiplexer}},$$

$$\mathrm{CtrlUnit} \begin{array}{l} \xrightarrow{\mathrm{ALUCtrl}} \mathrm{ALU} \\ \xrightarrow{\mathrm{ALUSrcA}} \mathrm{SrcA_{multiplexer}}, \\ \xrightarrow{\mathrm{ALUSrcB}} \mathrm{SrcB_{multiplexer}} \end{array} \begin{array}{r} \mathrm{SrcA_{multiplexer}} \xrightarrow{\mathrm{PC}} \\ \mathrm{SrcB_{multiplexer}} \xrightarrow{\mathrm{ShiftImm}} \end{array} \mathrm{ALU} \xrightarrow{\mathrm{PCBranch}} \mathrm{PCSrc_{multiplexer}}$$

$$3\ \mathrm{RegFileData_{reg}} \begin{array}{l} \xrightarrow{\mathrm{A}} \mathrm{SrcA_{multiplexer}} \\ \xrightarrow{\mathrm{B}} \mathrm{SrcB_{multiplexer}} \end{array} \xrightarrow{\begin{array}{c}\mathrm{SrcA}\\ \mathrm{SrcB}\end{array}} \mathrm{ALU} \xrightarrow{\mathrm{Zero}} \mathrm{Branch_{AND}},\ \mathrm{CtrlUnit} \begin{array}{l} \xrightarrow{\mathrm{Branch}} \mathrm{Branch_{AND}} \\ \xrightarrow{\mathrm{PCSrc}} \mathrm{PCSrc_{multiplexer}} \\ \xrightarrow{\mathrm{ALUCtrl}} \mathrm{ALU} \\ \xrightarrow{\mathrm{ALUSrcA}} \mathrm{SrcA_{multiplexer}} \\ \xrightarrow{\mathrm{ALUSrcB}} \mathrm{SrcB_{multiplexer}} \end{array},$$

$$\begin{array}{r} \mathrm{PCSrc_{multiplexer}} \xrightarrow{\mathrm{PCBranch}} \\ \mathrm{Branch_{AND}} \xrightarrow{\mathrm{Branch}} \mathrm{Branch_{OR}} \end{array} \xrightarrow{\mathrm{PCEn}} \mathrm{PC_{reg}}$$

5. sw

$$1\ \mathrm{PC_{reg}} \xrightarrow{\mathrm{PC}} \mathrm{Adr_{multiplexer}} \xrightarrow{\mathrm{Adr}} \mathrm{Mem} \xrightarrow{\mathrm{Instr/Data}} \mathrm{InstReg},\ \mathrm{CtrlUnit} \begin{array}{l} \xrightarrow{\mathrm{PC_{write}}} \mathrm{PCEn_{OR}} \xrightarrow{\mathrm{PCEn}} \mathrm{PC_{reg}} \\ \xrightarrow{\mathrm{PCSrc}} \mathrm{PCSrc_{multiplexer}} \\ \xrightarrow{\mathrm{IR_{write}}} \mathrm{InstReg} \\ \xrightarrow{\mathrm{IorD}} \mathrm{Adr_{multiplexer}} \\ \xrightarrow{\mathrm{ALUSrcA}} \mathrm{SrcA_{multiplexer}} \\ \xrightarrow{\mathrm{ALUSrcB}} \mathrm{SrcB_{multiplexer}} \\ \xrightarrow{\mathrm{PCSrc}} \mathrm{PCSrc_{multiplexer}} \end{array},$$

$$\begin{array}{r} \mathrm{PC_{reg}} \xrightarrow{\mathrm{PC}} \mathrm{SrcA_{multiplexer}} \\ 4 \xrightarrow{4} \mathrm{SrcB_{multiplexer}} \end{array} \xrightarrow{\begin{array}{c}\mathrm{SrcA}\\ \mathrm{SrcB}\end{array}} \mathrm{ALU} \xrightarrow{\mathrm{PC+4}} \mathrm{PCSrc_{multiplexer}} \xrightarrow{\mathrm{PC+4}} \mathrm{PC_{reg}}$$

$$2\ \text{InstReg} \xrightarrow{\text{Funct \& Op}} \text{CtrlUnit}$$
$$\xrightarrow{\text{rt \& rs}} \text{RegFile} \xrightarrow{\text{rd1 \& rd2}} \text{RegFileData}_{\text{reg}}$$
$$\xrightarrow{\text{Imm}} \text{ImmExt} \xrightarrow{\text{SignImm}} \text{SrcB}_{\text{multiplexer}}$$

$$3\ \begin{matrix}\text{RegFileData}_{\text{reg}} \xrightarrow{\text{A}} \text{SrcA}_{\text{multiplexer}} \\ \text{SrcB}_{\text{multiplexer}}\end{matrix} \begin{matrix}\xrightarrow{\text{SrcA}} \\ \xrightarrow{\text{SrcB}}\end{matrix} \text{ALU} \xrightarrow{\text{ALURes}} \text{ALUOut}_{\text{reg}},\ \text{CtrlUnit} \begin{matrix}\xrightarrow{\text{ALUCtrl}} \text{ALU} \\ \xrightarrow{\text{ALUSrcA}} \text{SrcA}_{\text{multiplexer}} \\ \xrightarrow{\text{ALUSrcB}} \text{SrcB}_{\text{multiplexer}}\end{matrix}$$

$$4\ \begin{matrix}\text{ALU}_{\text{reg}} \xrightarrow{\text{ALUOut}} \text{Adr}_{\text{multiplexer}} \\ \text{RegFileData}_{\text{reg}}\end{matrix} \begin{matrix}\xrightarrow{\text{DAdr}} \\ \xrightarrow{\text{B}}\end{matrix} \text{Mem},\ \text{CtrlUnit} \begin{matrix}\xrightarrow{\text{IorD}} \text{Adr}_{\text{multiplexer}} \\ \xrightarrow{\text{MemWrite}} \text{Mem}\end{matrix}$$

6. lw

$$1\ \text{PC}_{\text{reg}} \xrightarrow{\text{PC}} \text{Adr}_{\text{multiplexer}} \xrightarrow{\text{Adr}} \text{Mem} \xrightarrow{\text{Instr/Data}} \text{InstReg},\ \text{CtrlUnit} \begin{matrix}\xrightarrow{\text{PC}_{\text{write}}} \text{PCEn}_{\text{OR}} \xrightarrow{\text{PCEn}} \text{PC}_{\text{reg}} \\ \xrightarrow{\text{PCSrc}} \text{PCSrc}_{\text{multiplexer}} \\ \xrightarrow{\text{IR}_{\text{write}}} \text{InstReg} \\ \xrightarrow{\text{IorD}} \text{Adr}_{\text{multiplexer}} \\ \xrightarrow{\text{ALUSrcA}} \text{SrcA}_{\text{multiplexer}} \\ \xrightarrow{\text{ALUSrcB}} \text{SrcB}_{\text{multiplexer}} \\ \xrightarrow{\text{PCSrc}} \text{PCSrc}_{\text{multiplexer}}\end{matrix},$$

$$\begin{matrix}\text{PC}_{\text{reg}} \xrightarrow{\text{PC}} \text{SrcA}_{\text{multiplexer}} \\ 4 \xrightarrow{4} \text{SrcB}_{\text{multiplexer}}\end{matrix} \begin{matrix}\xrightarrow{\text{SrcA}} \\ \xrightarrow{\text{SrcB}}\end{matrix} \text{ALU} \xrightarrow{\text{PC+4}} \text{PCSrc}_{\text{multiplexer}} \xrightarrow{\text{PC+4}} \text{PC}_{\text{reg}}$$

$$2\ \text{InstReg} \xrightarrow{\text{Funct \& Op}} \text{CtrlUnit}$$
$$\xrightarrow{\text{rt \& rs}} \text{RegFile} \xrightarrow{\text{rd1 \& rd2}} \text{RegFileData}_{\text{reg}}$$
$$\xrightarrow{\text{Imm}} \text{ImmExt} \xrightarrow{\text{SignImm}} \text{SrcB}_{\text{multiplexer}}$$

$$3\ \begin{matrix}\text{RegFileData}_{\text{reg}} \xrightarrow{\text{A}} \text{SrcA}_{\text{multiplexer}} \\ \text{SrcB}_{\text{multiplexer}}\end{matrix} \begin{matrix}\xrightarrow{\text{SrcA}} \\ \xrightarrow{\text{SrcB}}\end{matrix} \text{ALU} \xrightarrow{\text{ALURes}} \text{ALUOut}_{\text{reg}},\ \text{CtrlUnit} \begin{matrix}\xrightarrow{\text{ALUCtrl}} \text{ALU} \\ \xrightarrow{\text{ALUSrcA}} \text{SrcA}_{\text{multiplexer}} \\ \xrightarrow{\text{ALUSrcB}} \text{SrcB}_{\text{multiplexer}}\end{matrix}$$

$$4\ \text{ALU}_{\text{reg}} \xrightarrow{\text{ALUOut}} \text{Adr}_{\text{multiplexer}} \xrightarrow{\text{DAdr}} \text{Mem} \xrightarrow{\text{Data}} \text{DataReg},\ \text{CtrlUnit} \begin{matrix}\xrightarrow{\text{IorD}} \text{Adr}_{\text{multiplexer}} \\ \xrightarrow{\text{MemWrite}} \text{Mem} \\ \xrightarrow{\text{IRWrite}} \text{InstReg}\end{matrix}$$

$$5\ \begin{matrix}\text{InstReg} \xrightarrow{\text{rt}} \text{RegDst}_{\text{multiplexer}} \\ \text{DataReg} \xrightarrow{\text{Data}} \text{MemtoReg}_{\text{multiplexer}}\end{matrix} \begin{matrix}\xrightarrow{\text{rt}} \\ \xrightarrow{\text{Data}}\end{matrix} \text{RegFile},\ \text{CtrlUnit} \begin{matrix}\xrightarrow{\text{RegDest}} \text{RegDst}_{\text{multiplexer}} \\ \xrightarrow{\text{MemtoReg}} \text{MemtoReg}_{\text{multiplexer}} \\ \xrightarrow{\text{RegWrite}} \text{RegFile}\end{matrix}$$

7. addi

2. InstReg -> {Imm} -> ImmExt -> {SignExt} -> SrcBMult;
3. CtrlUnit -> {ALUSrcB} -> SrcBMult;