



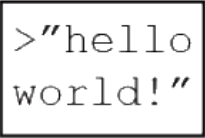


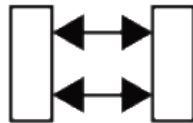
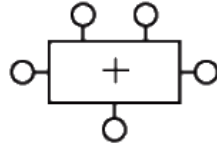



# Архитектуры процессорных систем

## Лекция 9. Конвейерный процессор RISC-V

Цикл из 16 лекций о цифровой схемотехнике, способах построения и архитектуре компьютеров

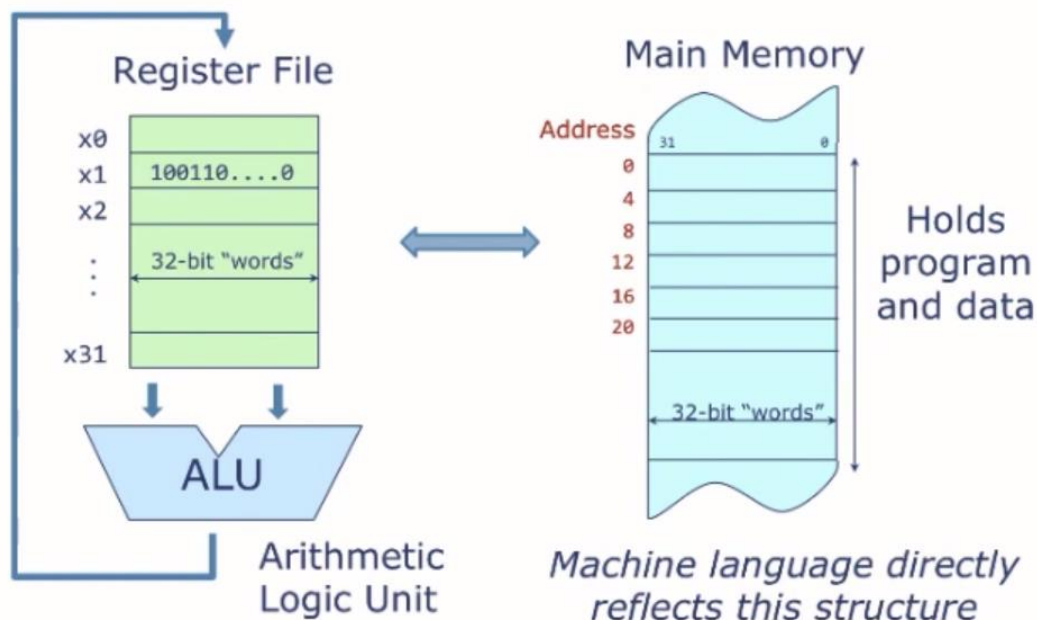
# План лекции

- Кодирование инструкций RISC-V
- Классификация микроархитектур
- Синтез процессора с конвейерной микроархитектурой
- Конфликты конвейера
- Оценка производительности полученного процессора

Application Software	
Operating Systems	
<b>Architecture</b>	
Micro-architecture	
Logic	
Digital Circuits	
Devices	
Physics	

– абстрактная модель функциональных возможностей процессора (средства, которыми может пользоваться программист / функциональная организация)

# Особенности архитектуры RISC-V



- Регистровый файл
  - 32 регистра общего назначения
  - Каждый регистр 32 бита
  - $x0 = 0$
- Память
  - Каждая ячейка памяти имеет ширину 32 бита (1 слово)
  - Память имеет побайтовую адресацию
  - Адреса соседних слов отличаются на 4
  - Адрес 32 бита
  - Может быть адресовано  $2^{32}$  байт или  $2^{30}$  слов

# RISC-V инструкции

- Вычислительные

- Register-register `op dest, src1, src2`
- Register-immediate `op dest, src1, const`

- Загрузки и сохранения

- `lw dest, offset(base)`
- `sw src, offset(base)`

- Управления

- Безусловный переход `jal label` и `jalr register`
- Условный переход `comp src1, src2, label`

Instr	Название	Функция	Описание	Формат	Opcode	Func3	Func7	Пример использования												
add sub xor or and sll srl sra slt sltu	ADDITION SUBtraction eXclusive OR OR AND Shift Left Logical Shift Right Logical Shift Right Arithmetic Set Less Then Set Less Then Unsigned	Сложение Вычитание Исключающее ИЛИ Логическое ИЛИ Логическое И Логический сдвиг влево Логический сдвиг вправо Арифметический сдвиг вправо Результат сравнения A < B Беззнаковое сравнение A < B	rd = rs1 + rs2 rd = rs1 - rs2 rd = rs1 ^ rs2 rd = rs1   rs2 rd = rs1 & rs2 rd = rs1 << rs2 rd = rs1 >> rs2 rd = rs1 >>> rs2 rd = (rs1 < rs2) ? 1 : 0 rd = (rs1 < rs2) ? 1 : 0	R	0110011	0x0 0x0 0x4 0x6 0x7 0x1 0x5 0x5 0x2 0x3	0x00 0x20 0x00 0x00 0x00 0x00 0x00 0x20 0x00 0x00	op rd, rs1, rs2  xor x2, x5, x6 sll x7, x11, x12												
addi xori ori andi slli srli srai slti sltiu	ADDITION Immediate eXclusive OR Immediate OR Immediate AND Immediate Shift Left Logical Immediate Shift Right Logical Immediate Shift Right Arithmetic Immediate Set Less Then Immediate Set Less Then Immediate Unsigned	Сложение с константой Исключающее ИЛИ с константой Логическое ИЛИ с константой Логическое И с константой Логический сдвиг влево Логический сдвиг вправо Арифметический сдвиг вправо Результат сравнения A < B Беззнаковое сравнение A < B	rd = rs1 + imm rd = rs1 ^ imm rd = rs1   imm rd = rs1 & imm rd = rs1 << imm rd = rs1 >> imm rd = rs1 >>> imm rd = (rs1 < imm) ? 1 : 0 rd = (rs1 < imm) ? 1 : 0			I	0010011		0x0 0x4 0x6 0x7 0x1 0x5 0x5 0x2 0x3	-    0x00 0x00 0x20  -	op rd, rs1, imm  addi x6, x3, -12 ori x3, x1, 0x8F									
lb lh lw lbu lbh	Load Byte Load Half Load Word Load Byte Unsigned Load Half Unsigned	Загрузить байт из памяти Загрузить полуслово из памяти Загрузить слово из памяти Загрузить беззнаковый байт из памяти Загрузить беззнаковое полуслово из памяти	rd = SE(Mem[rs1 + imm][7:0]) rd = SE(Mem[rs1 + imm][15:0]) rd = SE(Mem[rs1 + imm][31:0]) rd = Mem[rs1 + imm][7:0] rd = Mem[rs1 + imm][15:0]						I	0000011		0x0 0x1 0x2 0x4 0x5	-    	op rd, imm(rs1)  lh x1, 8(x5)						
sb sh sw	Store Byte Store Half Store Word	Сохранить байт в память Сохранить полуслово в память Сохранить слово в память	Mem[rs1 + imm][7:0] = rs2[7:0] Mem[rs1 + imm][15:0] = rs2[15:0] Mem[rs1 + imm][31:0] = rs2[31:0]									S	0100011		0x0 0x1 0x2	-  	op rs2, imm(rs1)  sw x1, 0xCF(x12)			
beq bne blt bge bltu bgeu	Branch if Equal Branch if Not Equal Branch if Less Than Branch if Greater or Equal Branch if Less Than Unsigned Branch if Greater or Equal Unsigned	Перейти, если A == B Перейти, если A != B Перейти, если A < B Перейти, если A >= B Перейти, если A < B беззнаковое Перейти, если A >= B беззнаковое	if (rs1 == rs2) PC += imm if (rs1 != rs2) PC += imm if (rs1 < rs2) PC += imm if (rs1 >= rs2) PC += imm if (rs1 < rs2) PC += imm if (rs1 >= rs2) PC += imm												B	1100011		0x0 0x1 0x4 0x5 0x6 0x7	-     	comp rs1, rs2, imm  beq x8, x9, offset bltu x20, x21, 0xFC
jal jalr	Jamp And Link Jamp And Link Register	Переход с сохранением адреса возврата Переход по регистру с сохранением адреса возврата	rd = PC + 4; PC += imm rd = PC + 4; PC = rs1															J I	1101111 1100111	
lui auipc	Load Upper Immediate Add Upper Immediate to PC	Загрузить константу в сдвинутую на 12 Сохранить счетчик команд в сумме с константой << 12	rd = imm << 12 rd = PC + (imm << 12)									U	0110111 0010111				- 	- 	lui x3, 0xFFFFF auipc x2, 0x000FF	
ecall ebreak	Environment CALL Environment BREAK	Передача управления операционной системе Передача управления отладчику	Воспринимать как nop						I	1110011			- 	- 			-			

# Кодирование инструкций RISC-V

31	27	26	25	24	20	19	15	14	12	11	7	6	0	
funct7				rs2		rs1		funct3		rd		opcode		R-type
imm[11:0]						rs1		funct3		rd		opcode		I-type
imm[11:5]				rs2		rs1		funct3		imm[4:0]		opcode		S-type
imm[12 10:5]				rs2		rs1		funct3		imm[4:1 11]		opcode		B-type
imm[31:12]										rd		opcode		U-type
imm[20 10:1 11 19:12]										rd		opcode		J-type

# Кодирование инструкций RISC-V

Assembly	Field Values						Machine Code						
	funct7	rs2	rs1	funct3	rd	op	funct7	rs2	rs1	funct3	rd	op	
add s2, s3, s4 add x18, x19, x20	0	20	19	0	18	51	0000,000	10100	10011	000	10010	011,0011	(0x01498933)
sub t0, t1, t2 sub x5, x6, x7	32	7	6	0	5	51	0100,000	00111	00110	000	00101	011,0011	(0x407302B3)
	7 bits	5 bits	5 bits	3 bits	5 bits	7 bits	7 bits	5 bits	5 bits	3 bits	5 bits	7 bits	

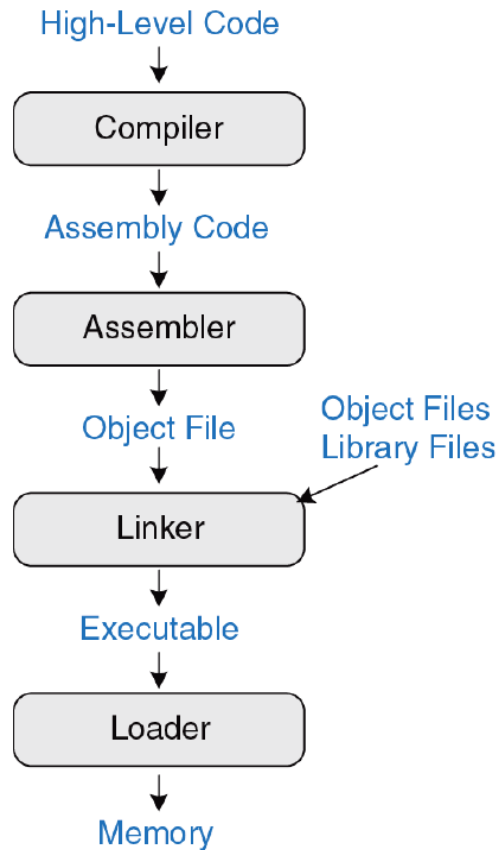
Machine Code						Field Values						Assembly	
	funct7	rs2	rs1	funct3	rd	op	funct7	rs2	rs1	funct3	rd	op	
(0x41FE83B3)	0100 000	11111	11101	000	00111	011 0011	32	31	29	0	7	51	sub x7, x29,x31 sub t2, t4, t6
	7 bits	5 bits	5 bits	3 bits	5 bits	7 bits	7 bits	5 bits	5 bits	3 bits	5 bits	7 bits	

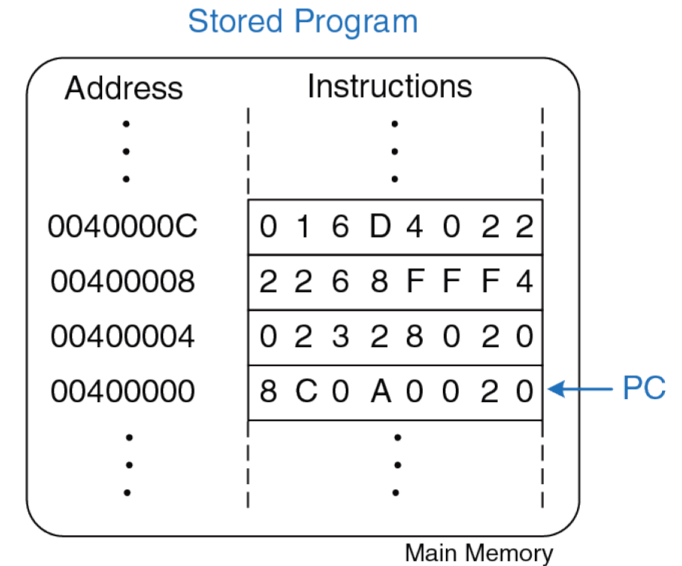
Machine Code						Field Values						Assembly	
	imm <sub>11:0</sub>	rs1	funct3	rd	op	imm <sub>11:0</sub>	rs1	funct3	rd	op			
(0xFDA48293)	1111 1101 1010	01001	000	00101	001 0011	-38	9	0	5	19	addi x5, x9, -38 addi t0, s1, -38		
	12 bits	5 bits	3 bits	5 bits	7 bits	12 bits	5 bits	3 bits	5 bits	7 bits			



# Представление программы в памяти



Assembly Code	Machine Code
lw \$t2, 32(\$0)	0x8C0A0020
add \$s0, \$s1, \$s2	0x02328020
addi \$t0, \$s3, -12	0x2268FFF4
sub \$t0, \$t3, \$t5	0x016D4022



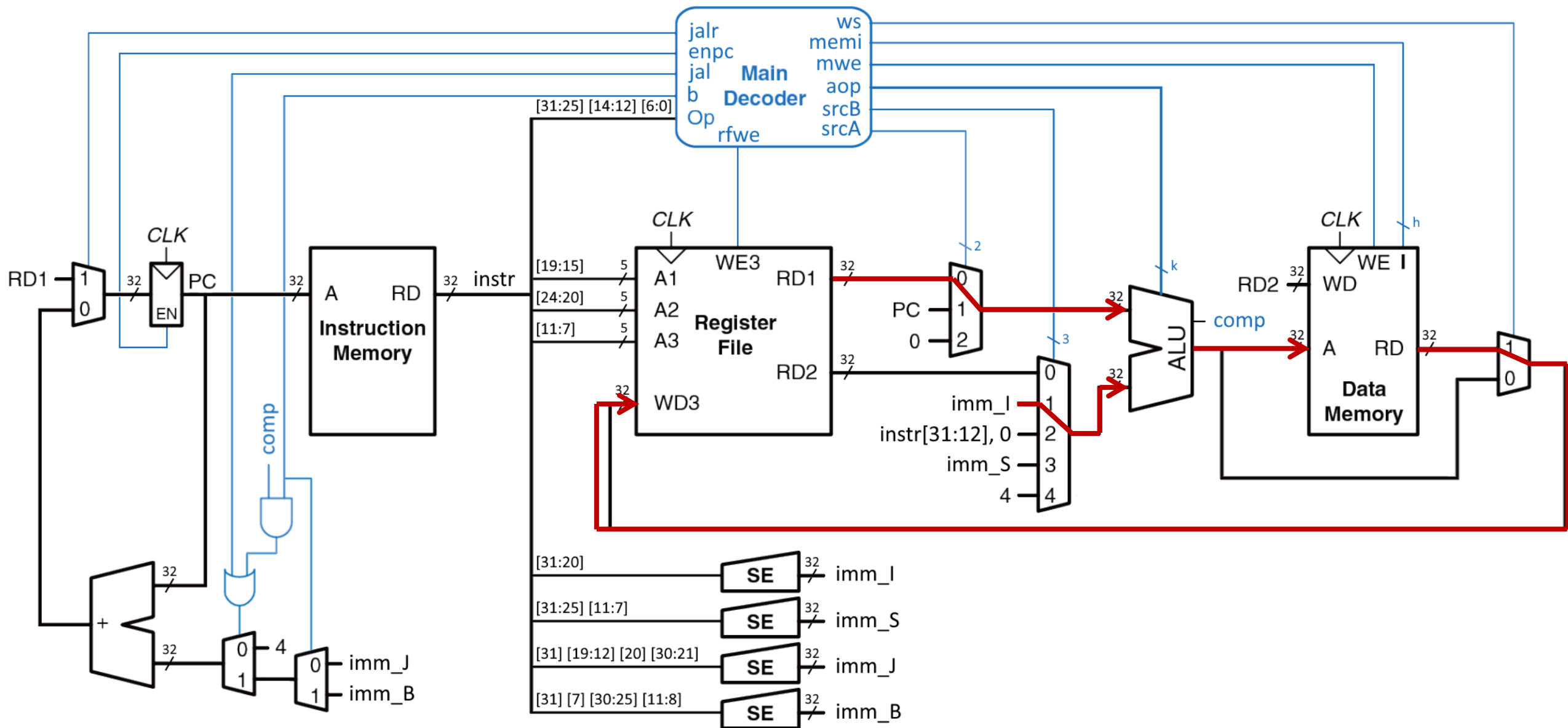
Application Software	
Operating Systems	
Architecture	
<b>Micro-architecture</b>	
Logic	
Digital Circuits	
Devices	
Physics	

- физическая модель, которая устанавливает состав, порядок и принципы взаимодействия основных функциональных частей процессора (структурная организация)

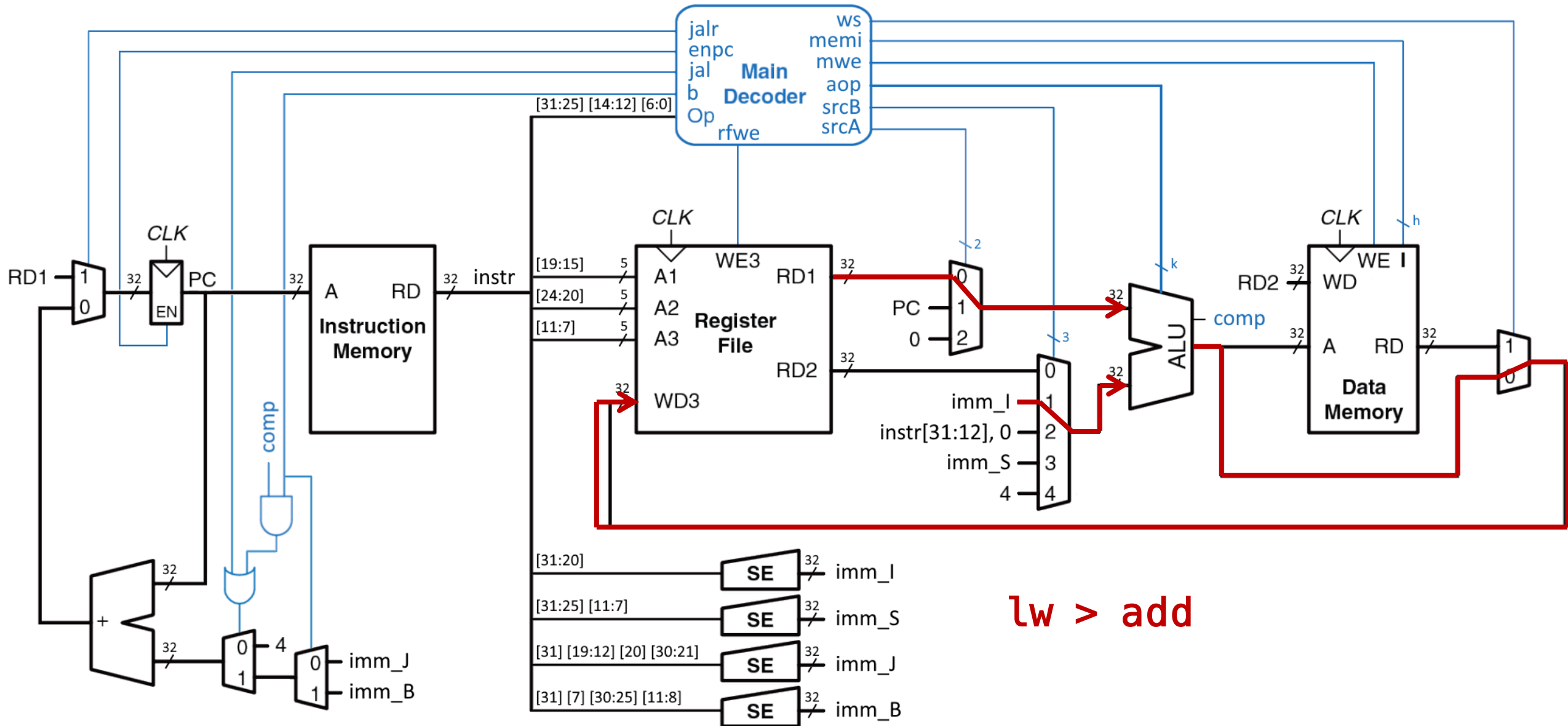
# Микроархитектуры

- Однотактная
  - выполняет одну инструкцию за один такт

lw

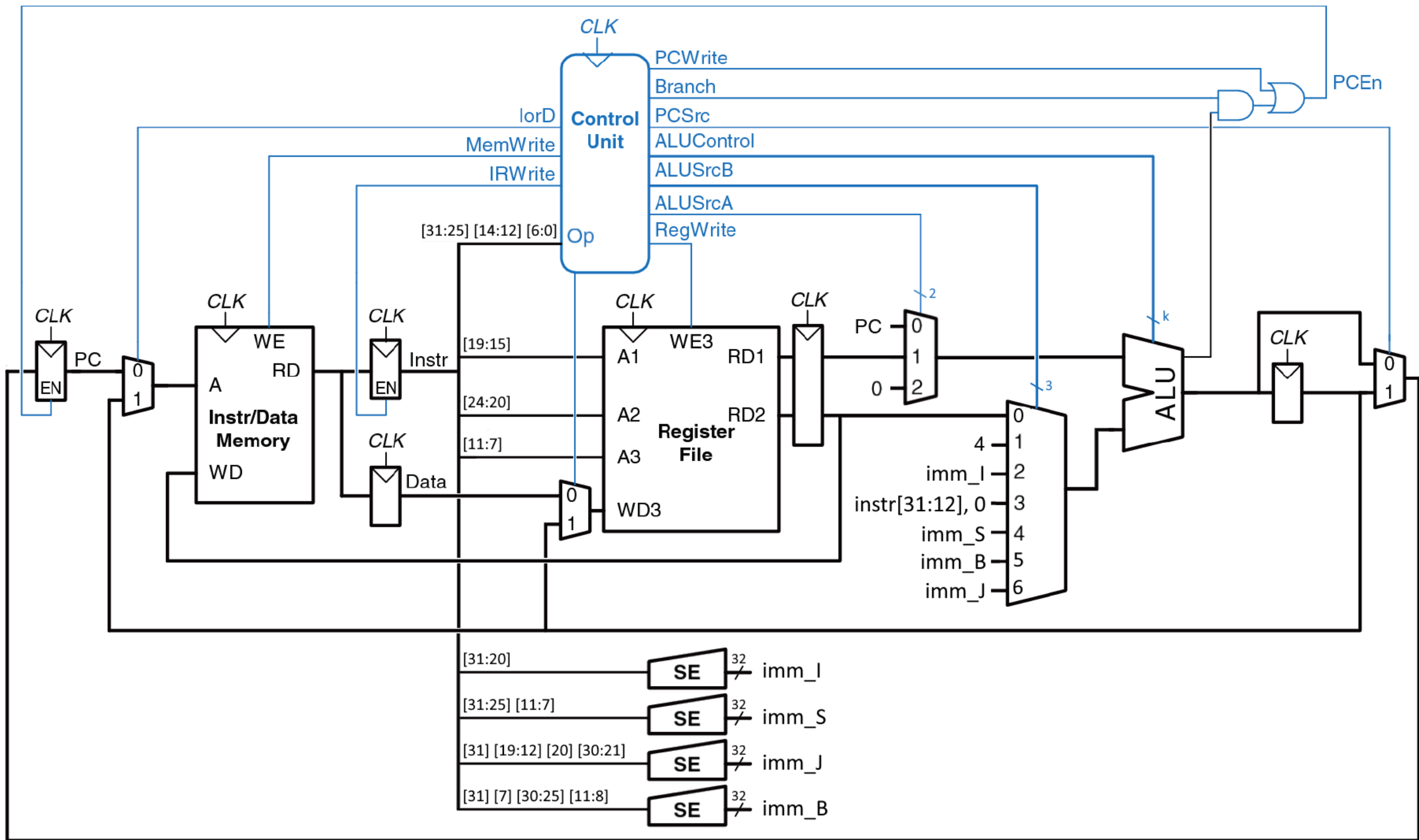


**add**



# Микроархитектуры

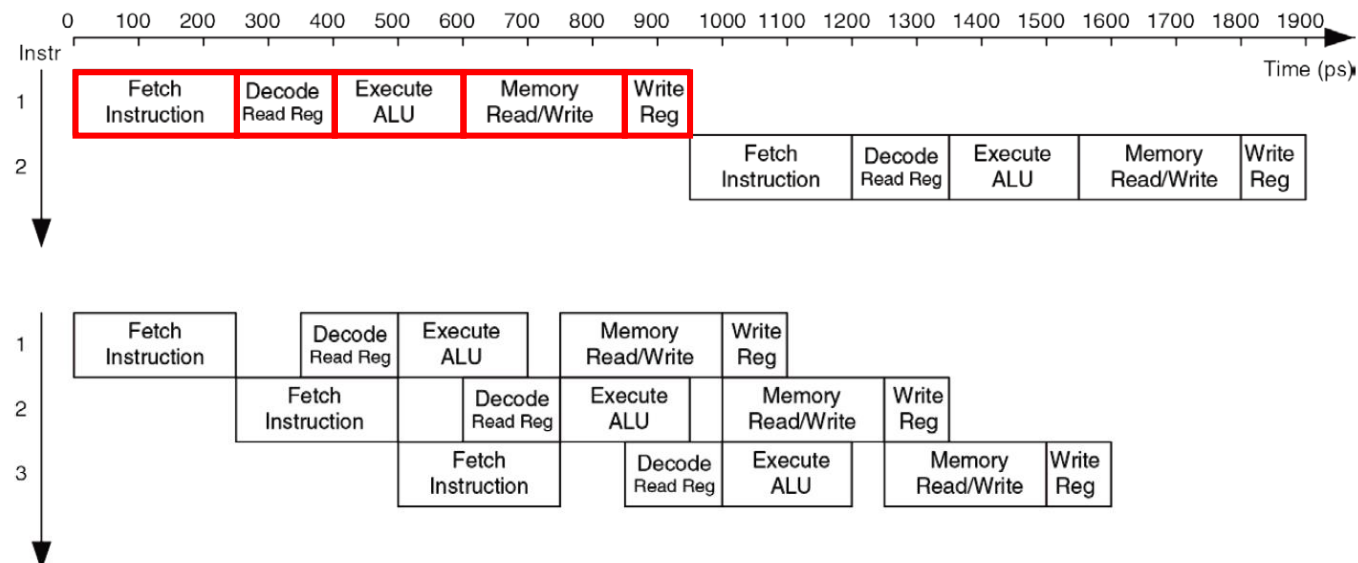
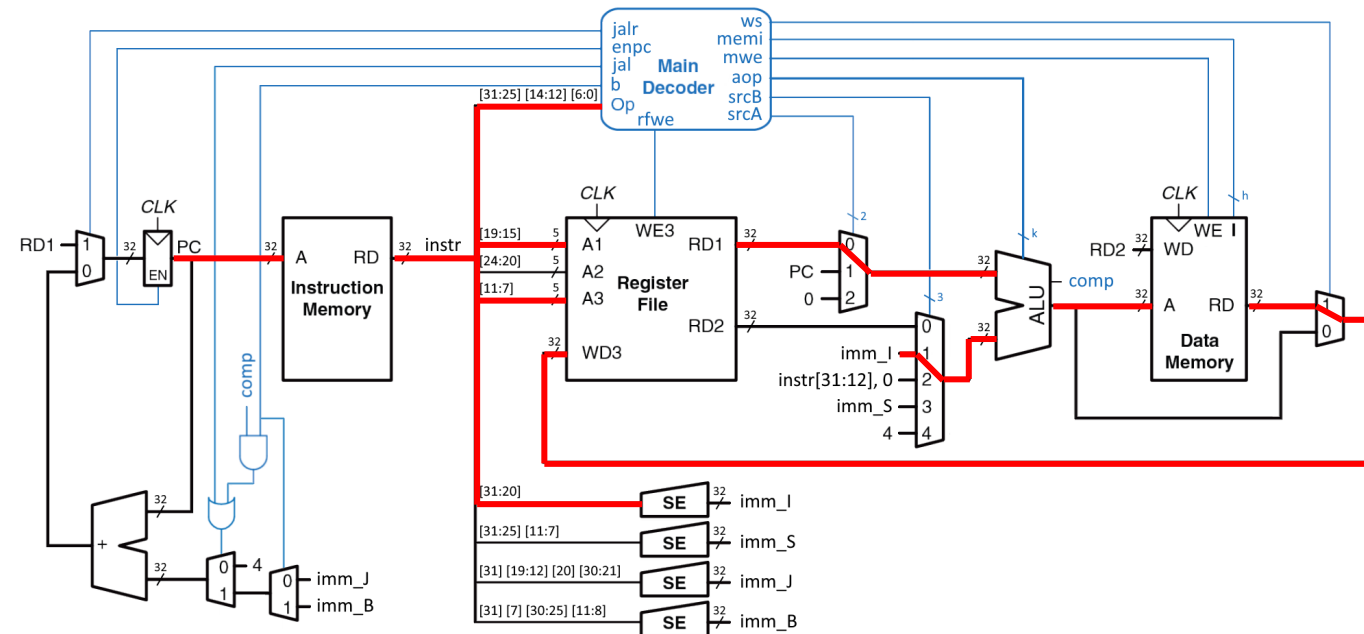
- **Однотактная**
  - выполняет одну инструкцию за один такт
- **Многотактная**
  - выполняет одну инструкцию за несколько более коротких тактов



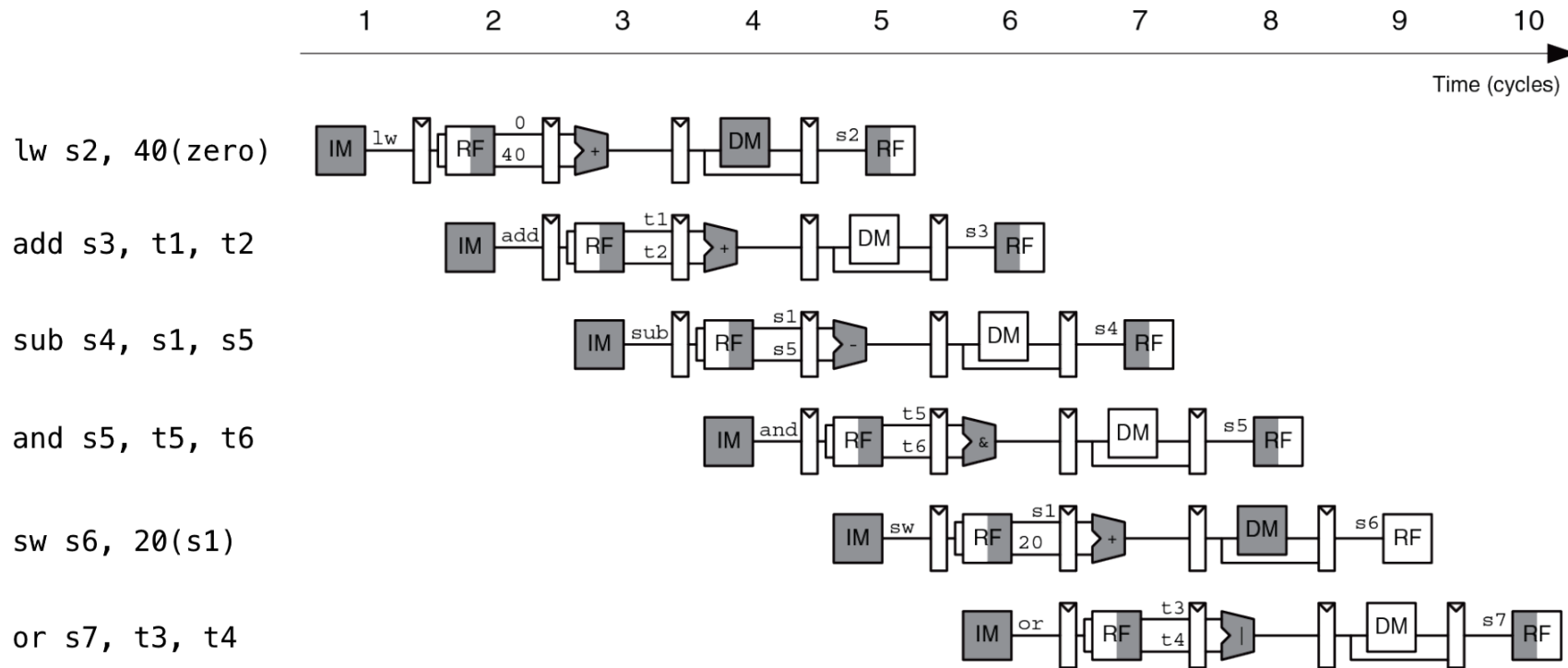
# Микроархитектуры

- **Однотактная**
  - выполняет одну инструкцию за один такт
- **Многотактная**
  - выполняет одну инструкцию за несколько более коротких тактов
- **Конвейерная**
  - результат применения принципа конвейерной обработки к однотактной микроархитектуре

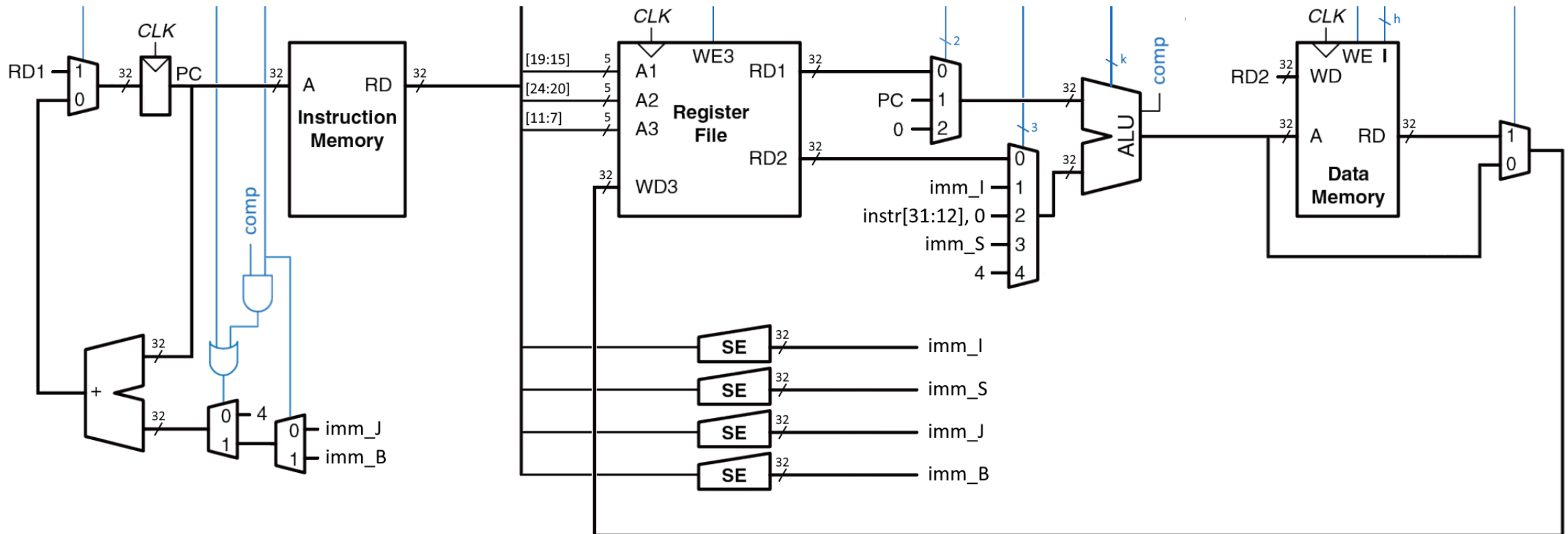




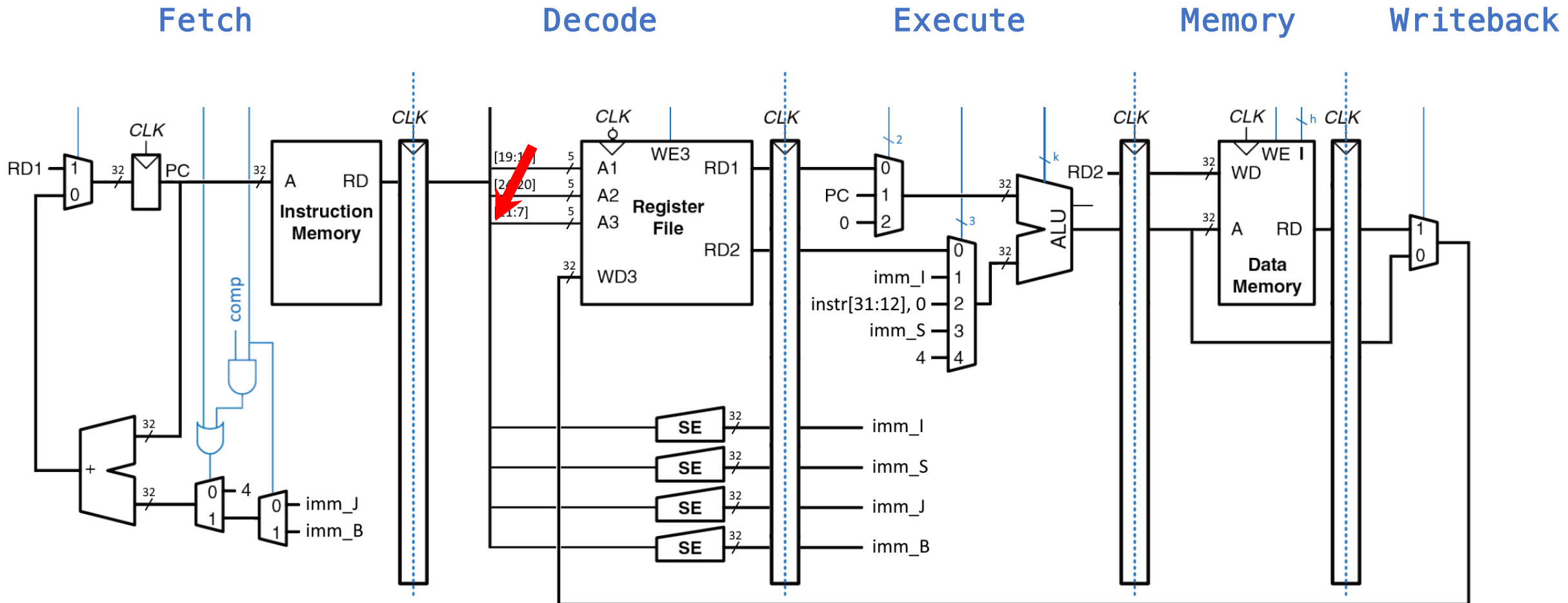
# Конвейер команд



# Однотактный тракт данных

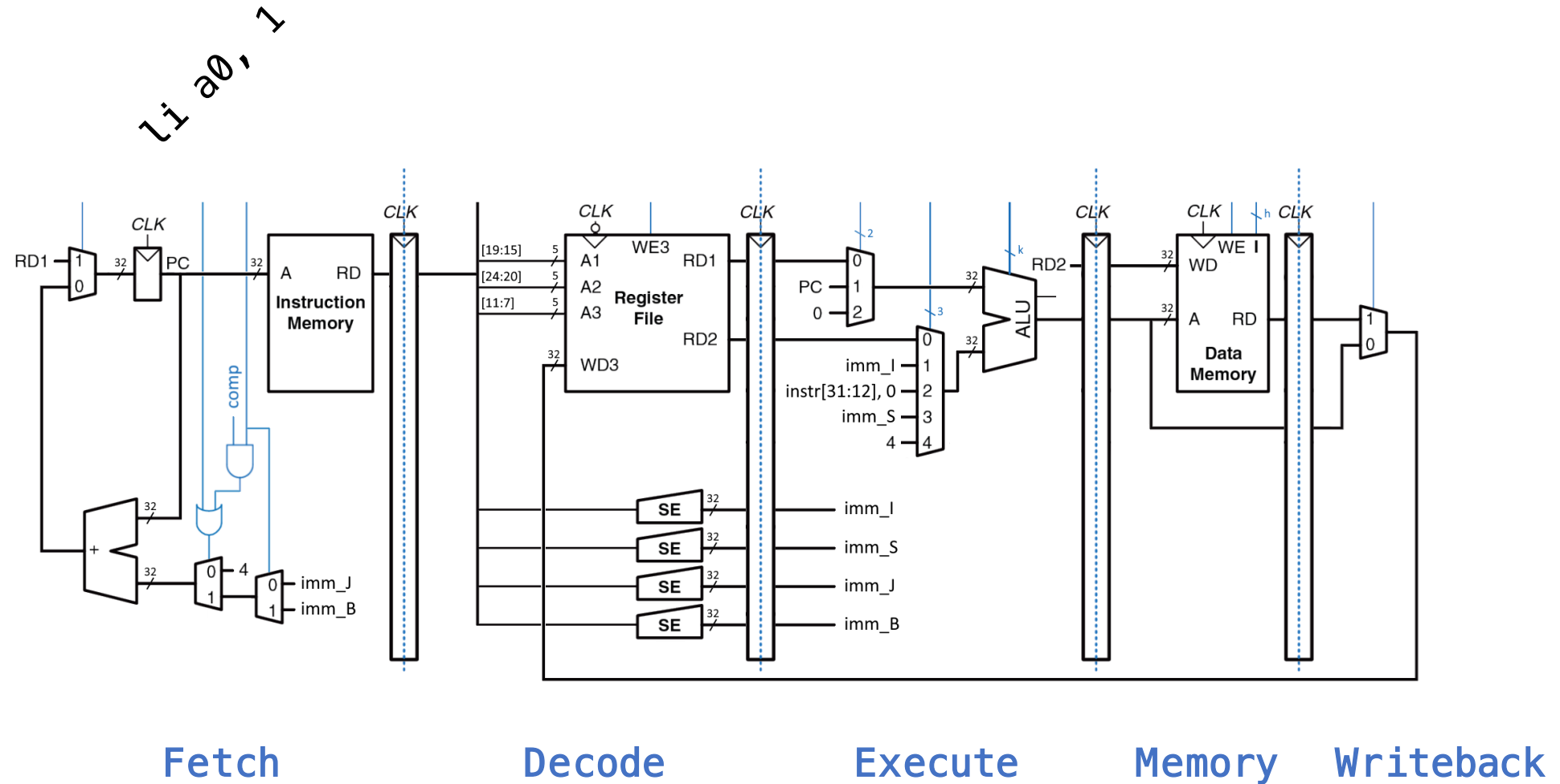


# Конвейерный тракт данных



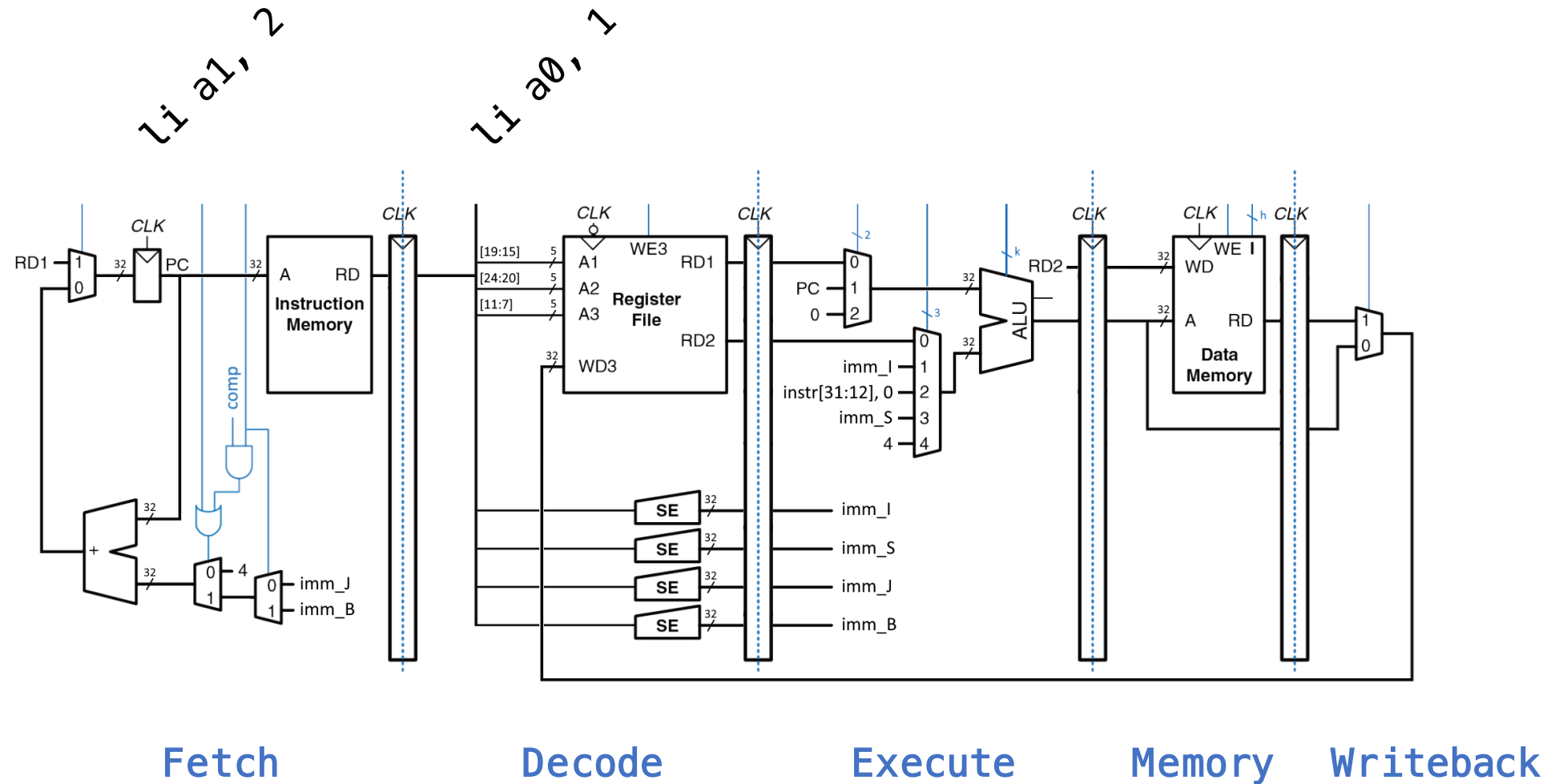
# Пример

**pc** → `li a0, 1`  
`li a1, 2`  
`addi s1, s2, -8`  
`add t1, t2, t3`  
`sw a1, 4(sp)`  
`jal ra, sum`  
`lw a1, 4(sp)`  
`jal ra, sum`  
`lw ra, 0(sp)`  
`addi sp, sp, 8`



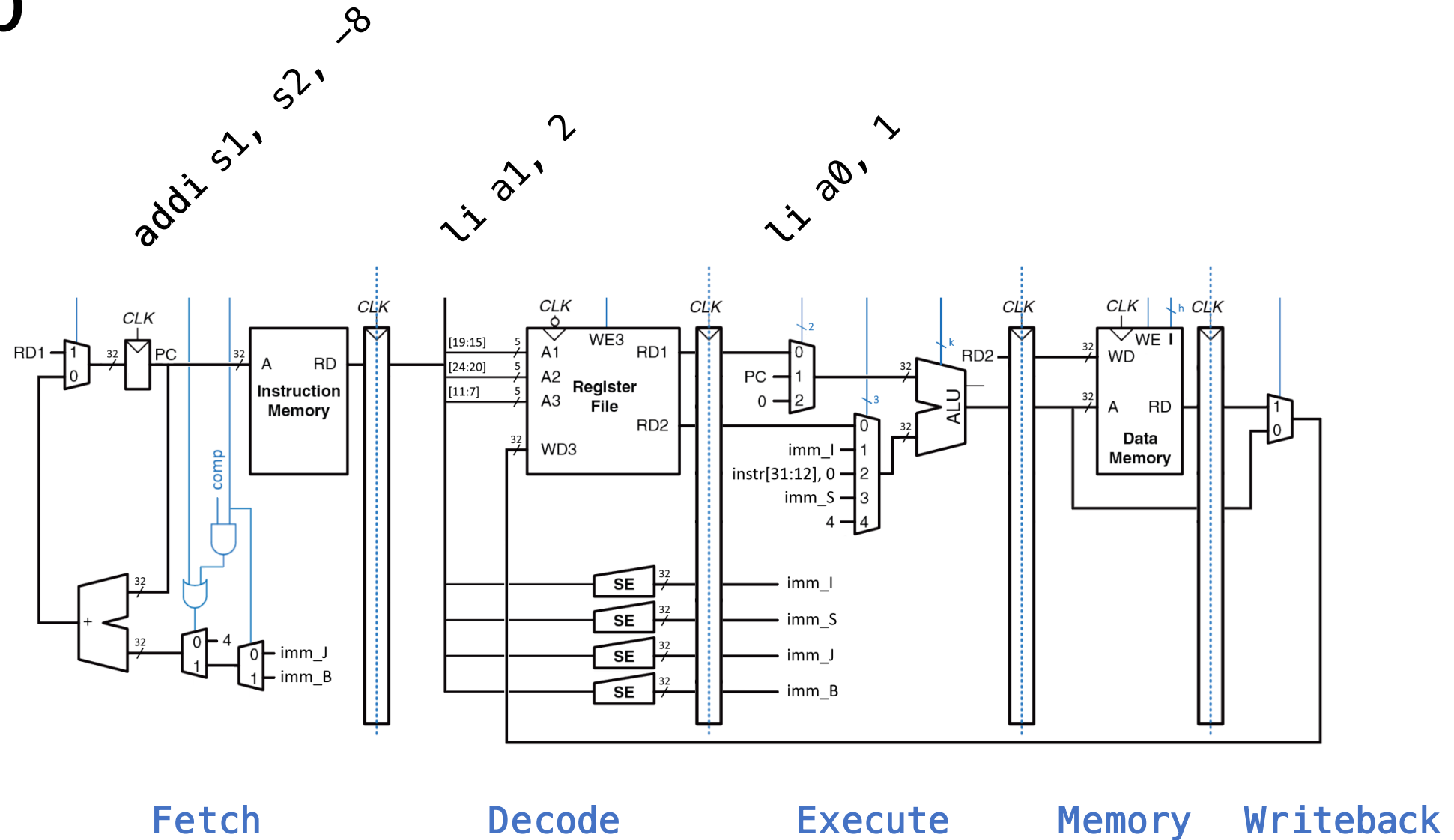
# Пример

```
li a0, 1
pc → li a1, 2
addi s1, s2, -8
add t1, t2, t3
sw a1, 4(sp)
jal ra, sum
lw a1, 4(sp)
jal ra, sum
lw ra, 0(sp)
addi sp, sp, 8
```



# Пример

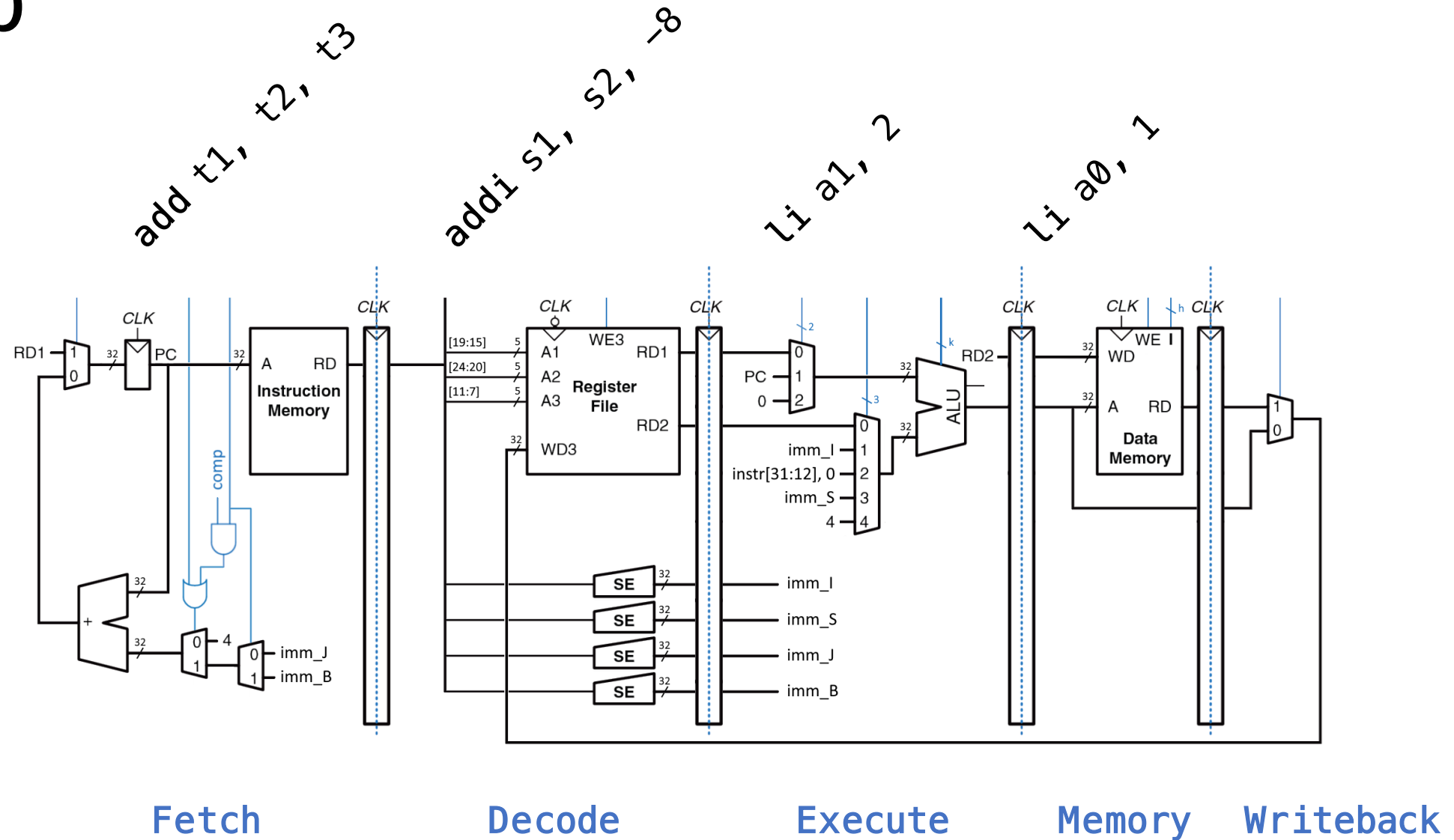
**pc** → `li a0, 1`  
`li a1, 2`  
`addi s1, s2, -8`  
`add t1, t2, t3`  
`sw a1, 4(sp)`  
`jal ra, sum`  
`lw a1, 4(sp)`  
`jal ra, sum`  
`lw ra, 0(sp)`  
`addi sp, sp, 8`



# Пример

pc

```
li a0, 1  
li a1, 2  
addi s1, s2, -8  
add t1, t2, t3  
sw a1, 4(sp)  
jal ra, sum  
lw a1, 4(sp)  
jal ra, sum  
lw ra, 0(sp)  
addi sp, sp, 8
```

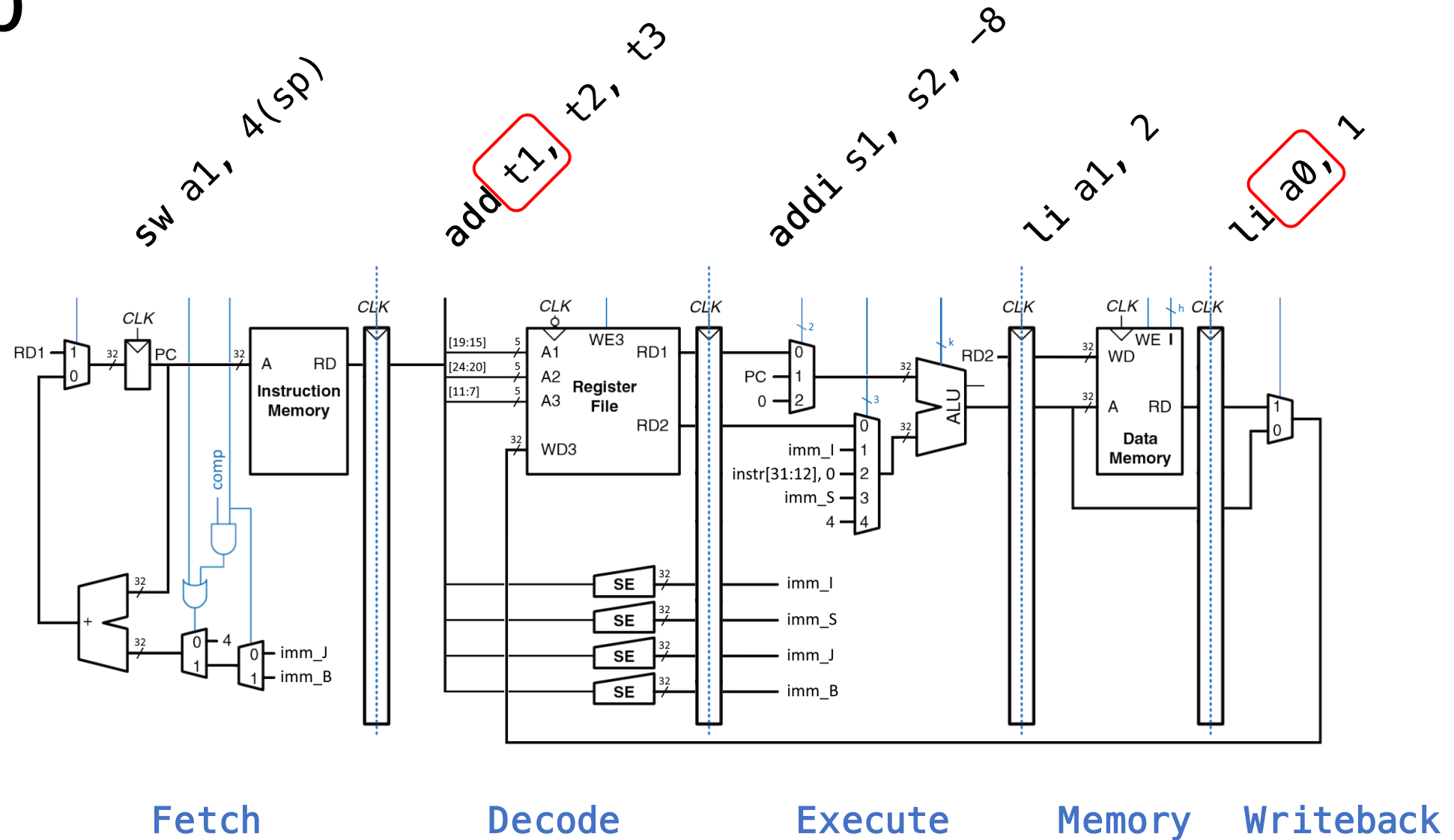




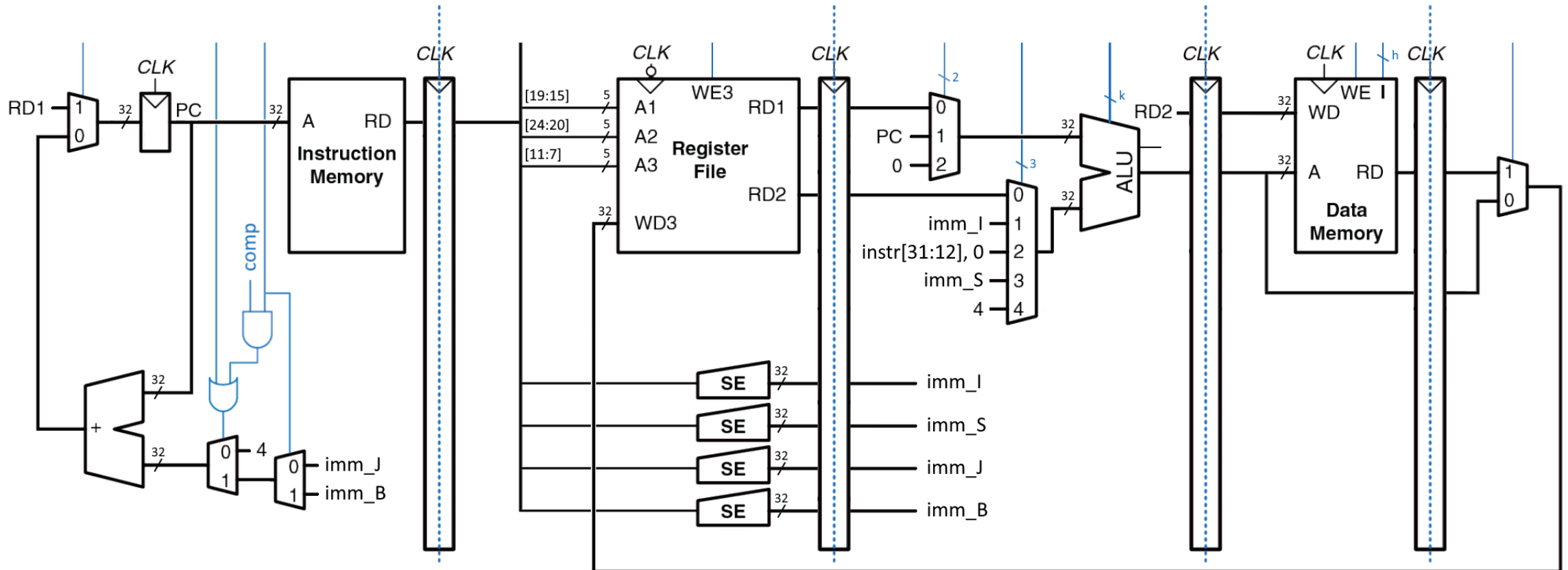
# Пример

```
li a0, 1
li a1, 2
addi s1, s2, -8
add t1, t2, t3
jal ra, sum
lw a1, 4(sp)
jal ra, sum
lw ra, 0(sp)
addi sp, sp, 8
```

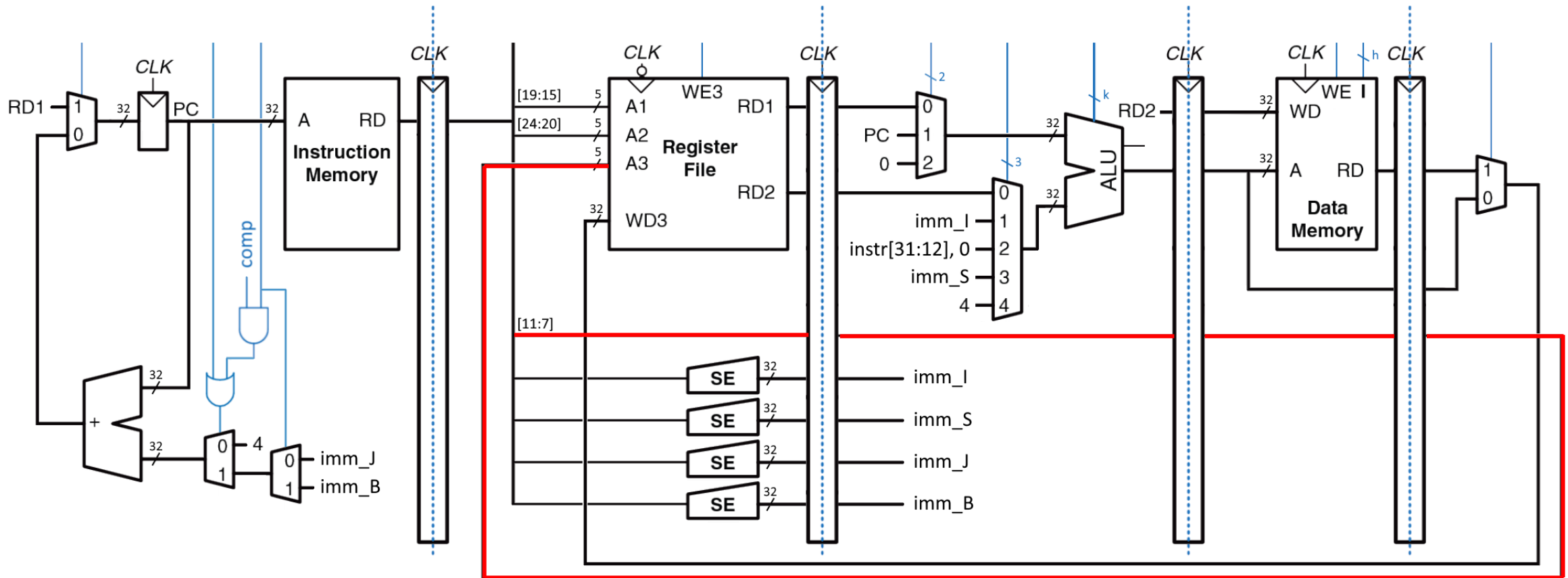
pc →

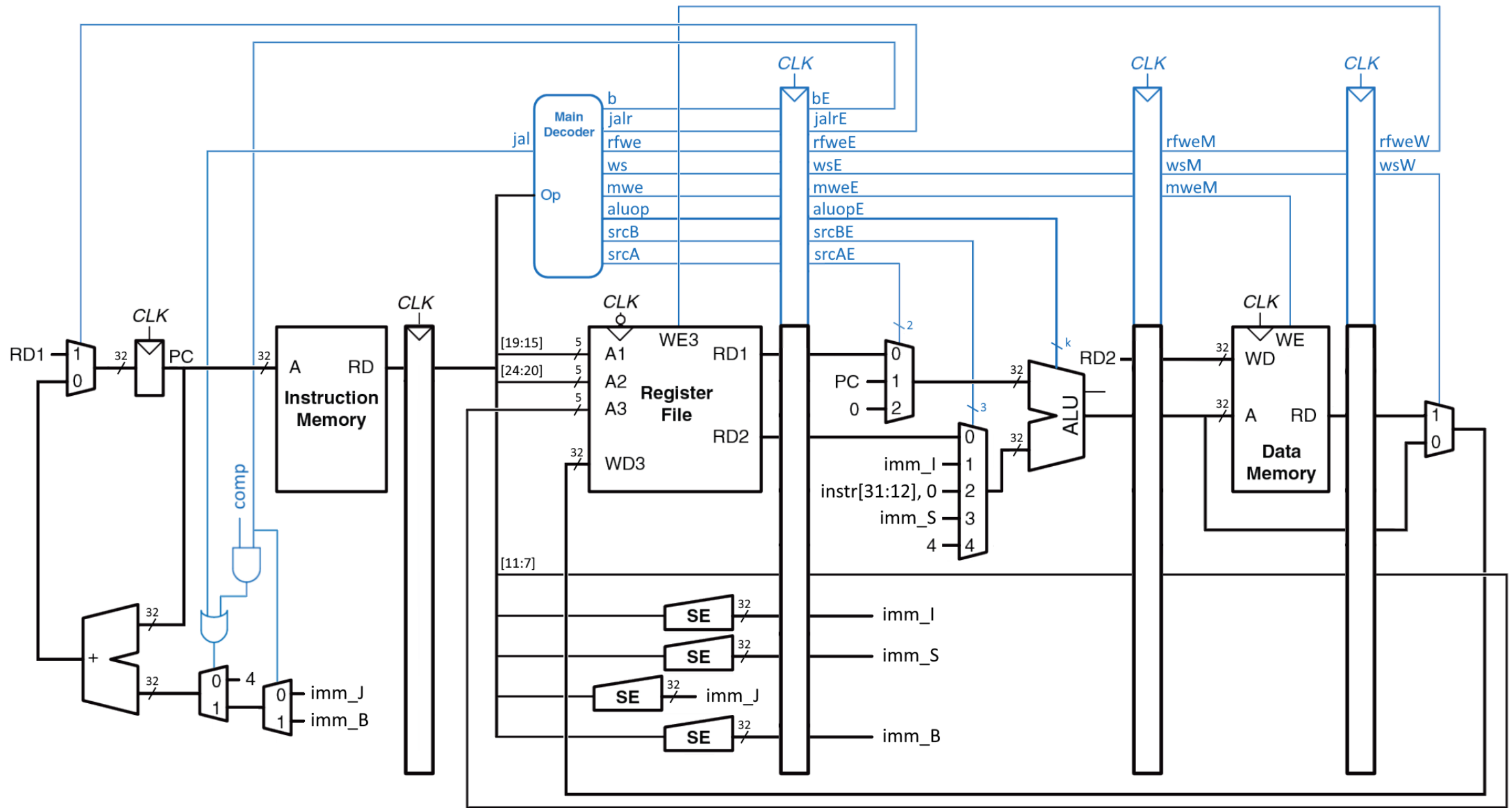


# Конвейерный тракт данных (–)

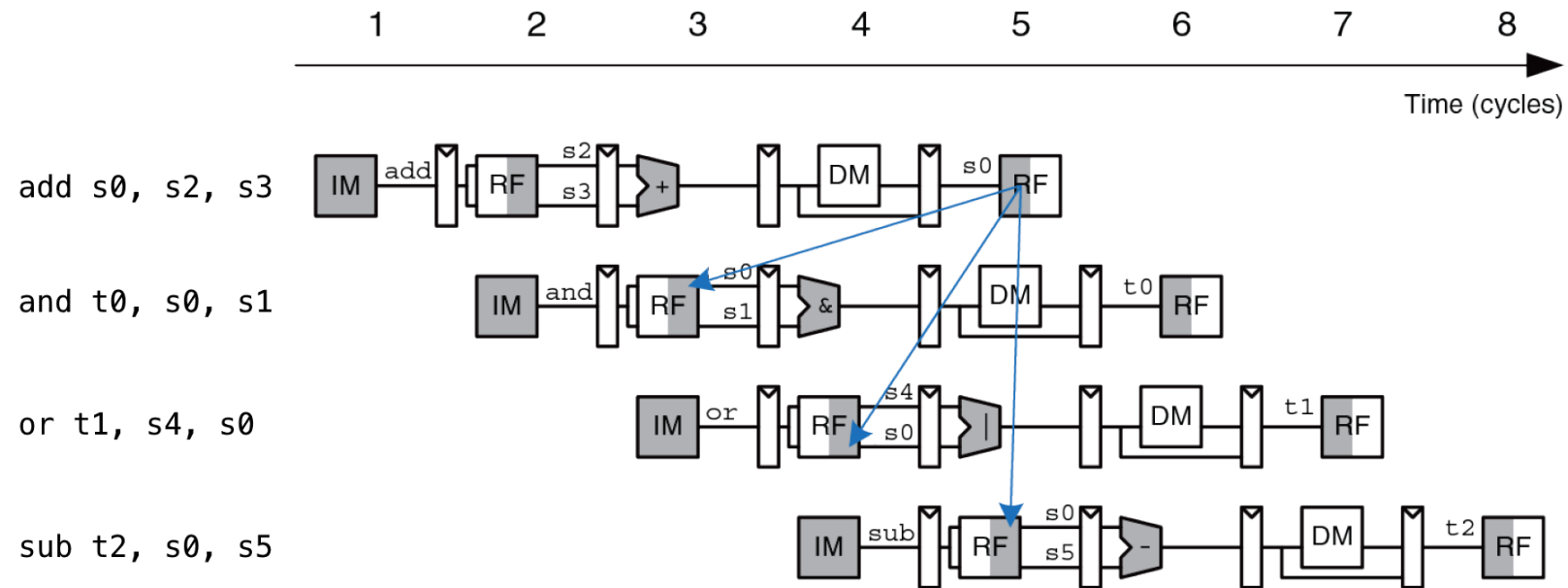


# Конвейерный тракт данных (+)

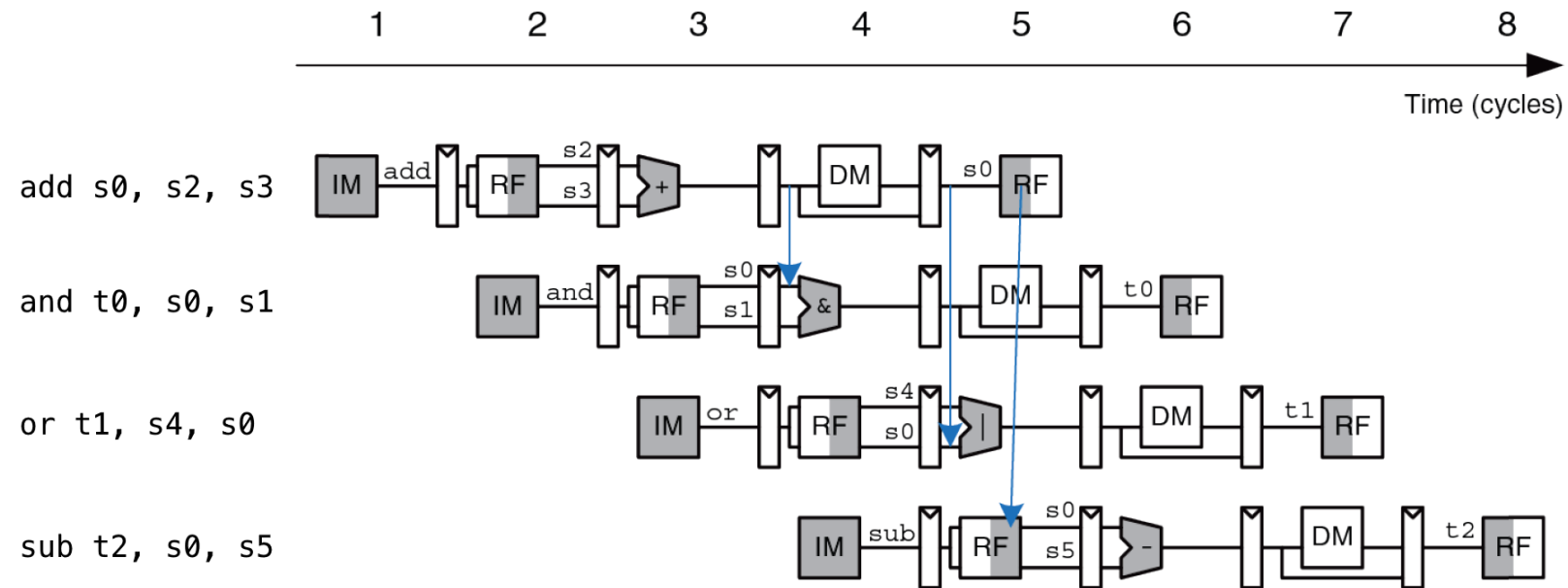


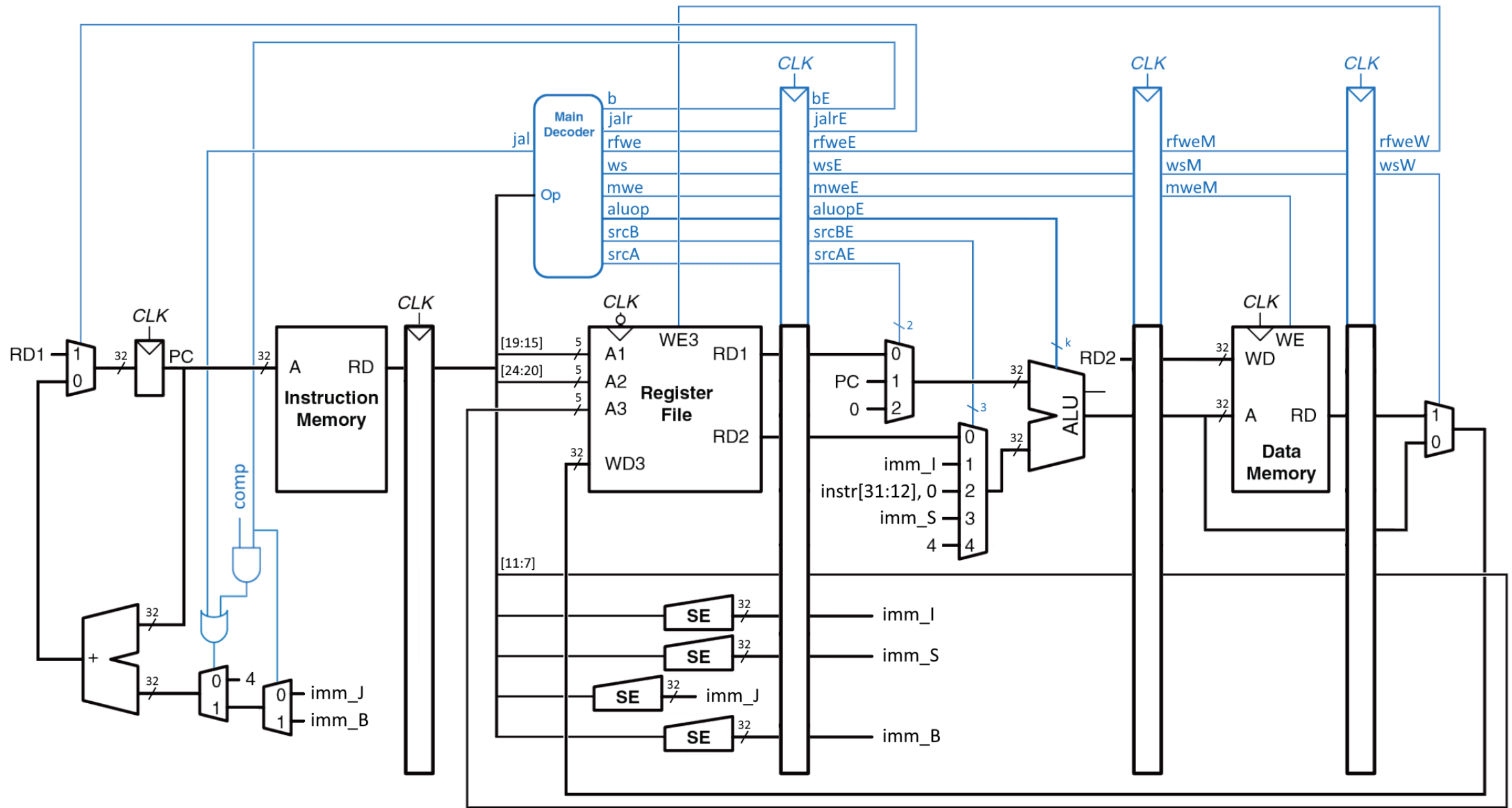


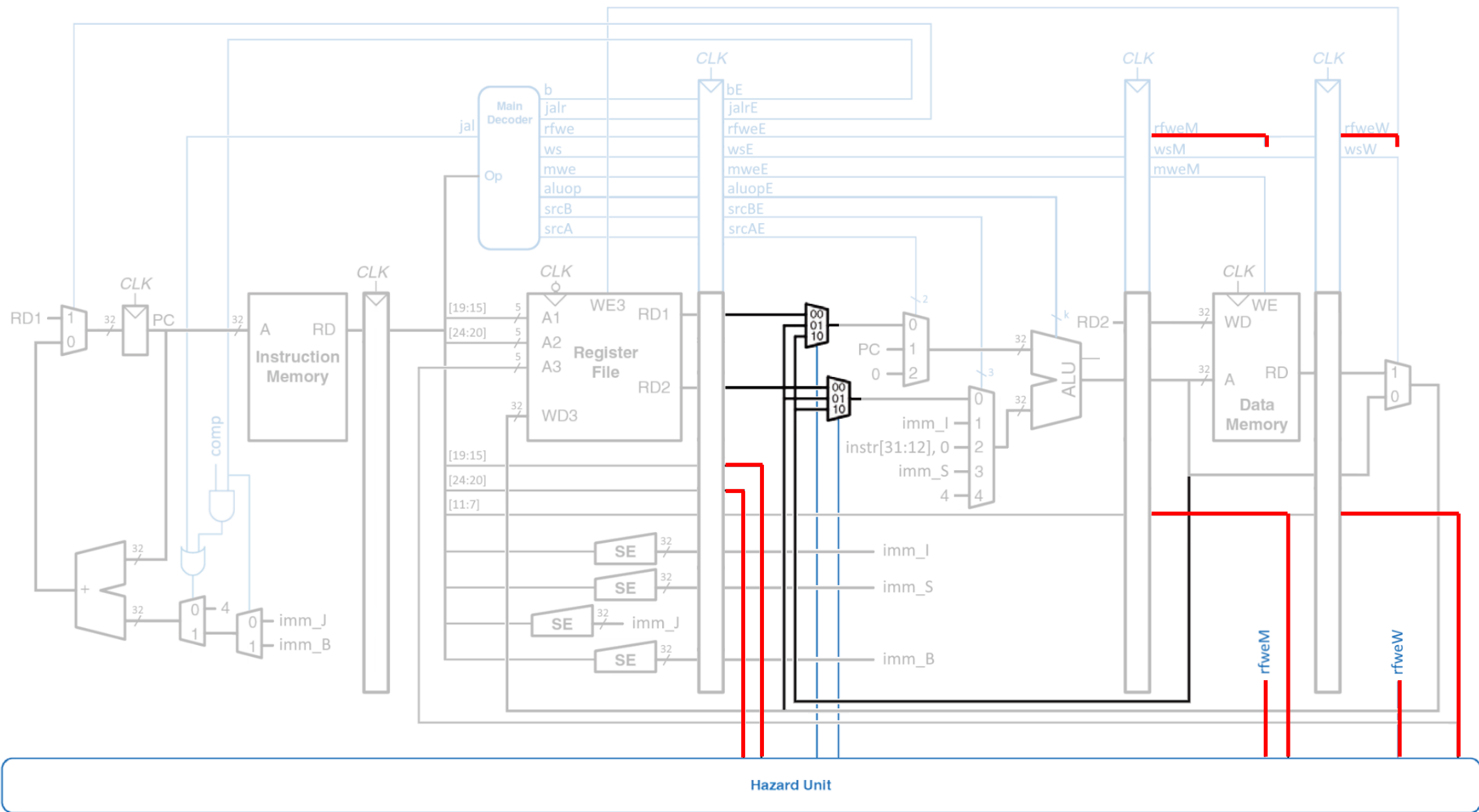
# Конфликт по данным



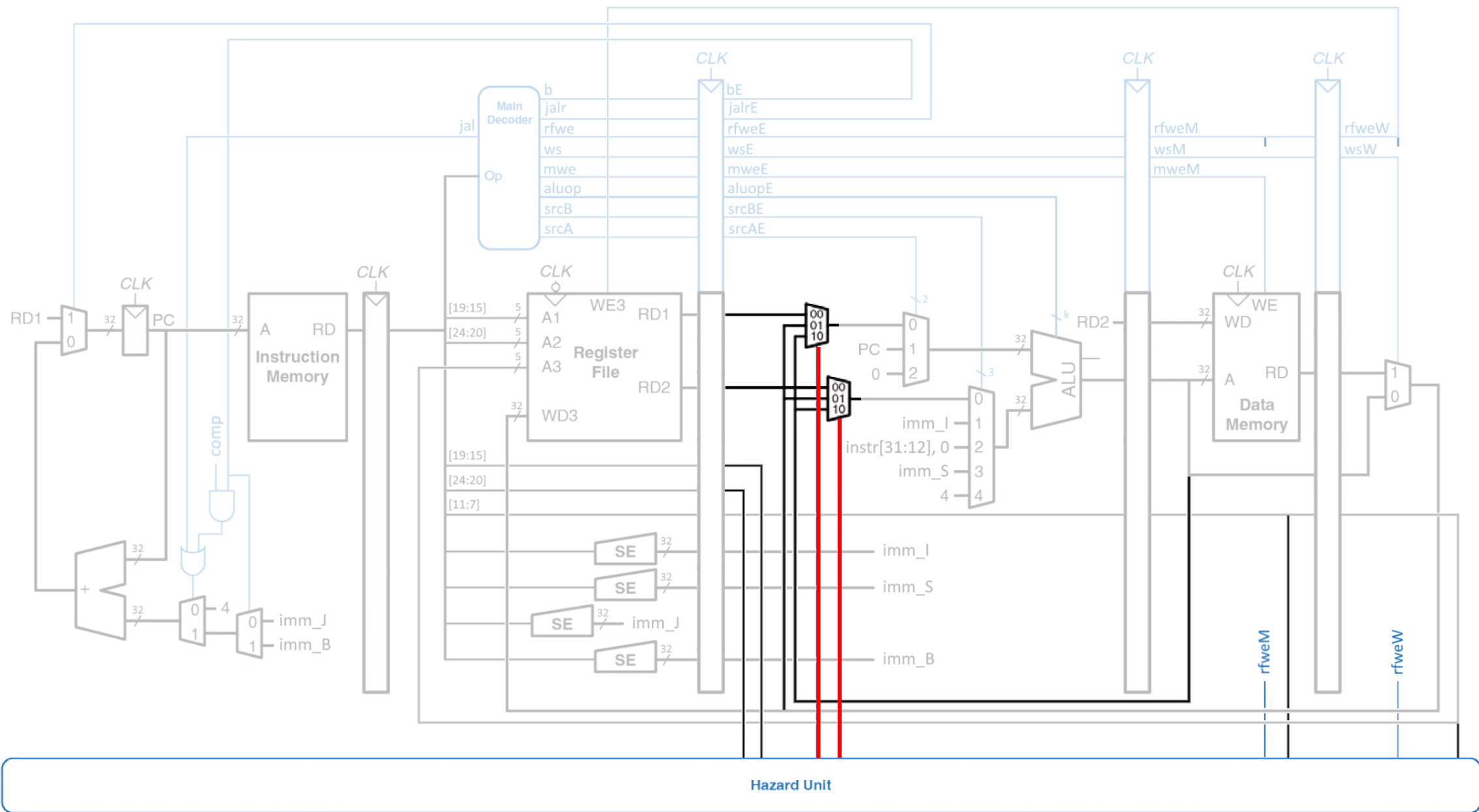
# Конфликт по данным

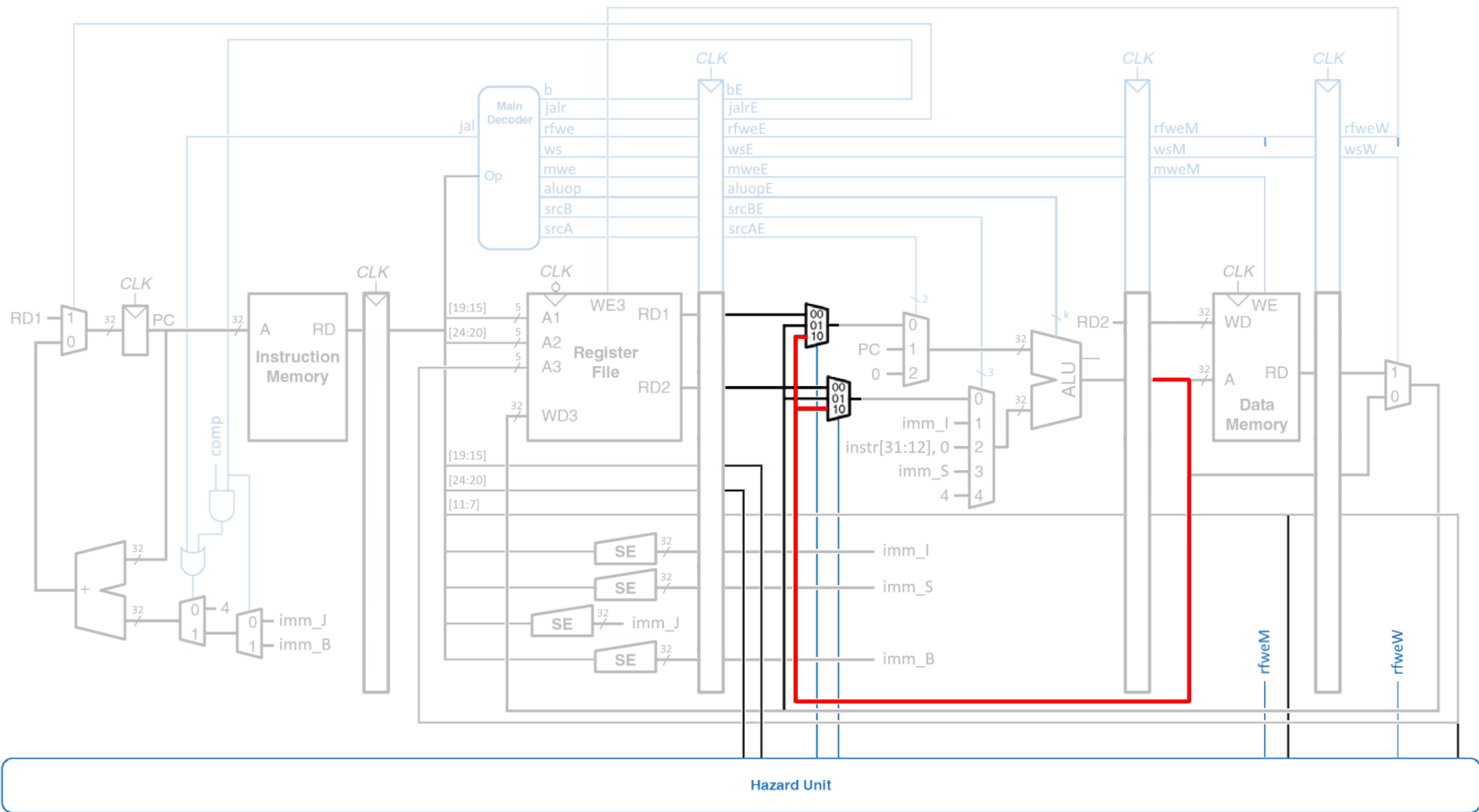


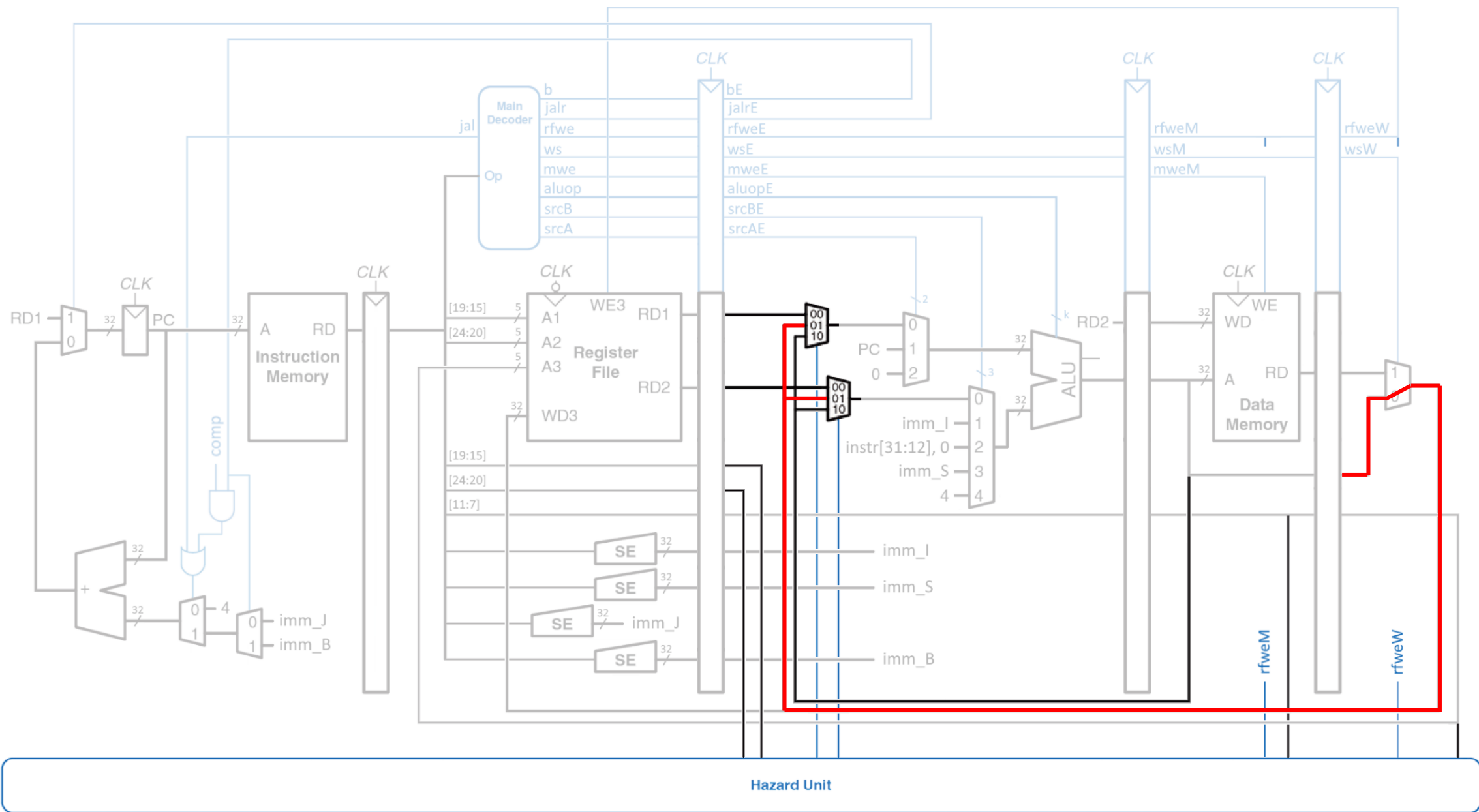


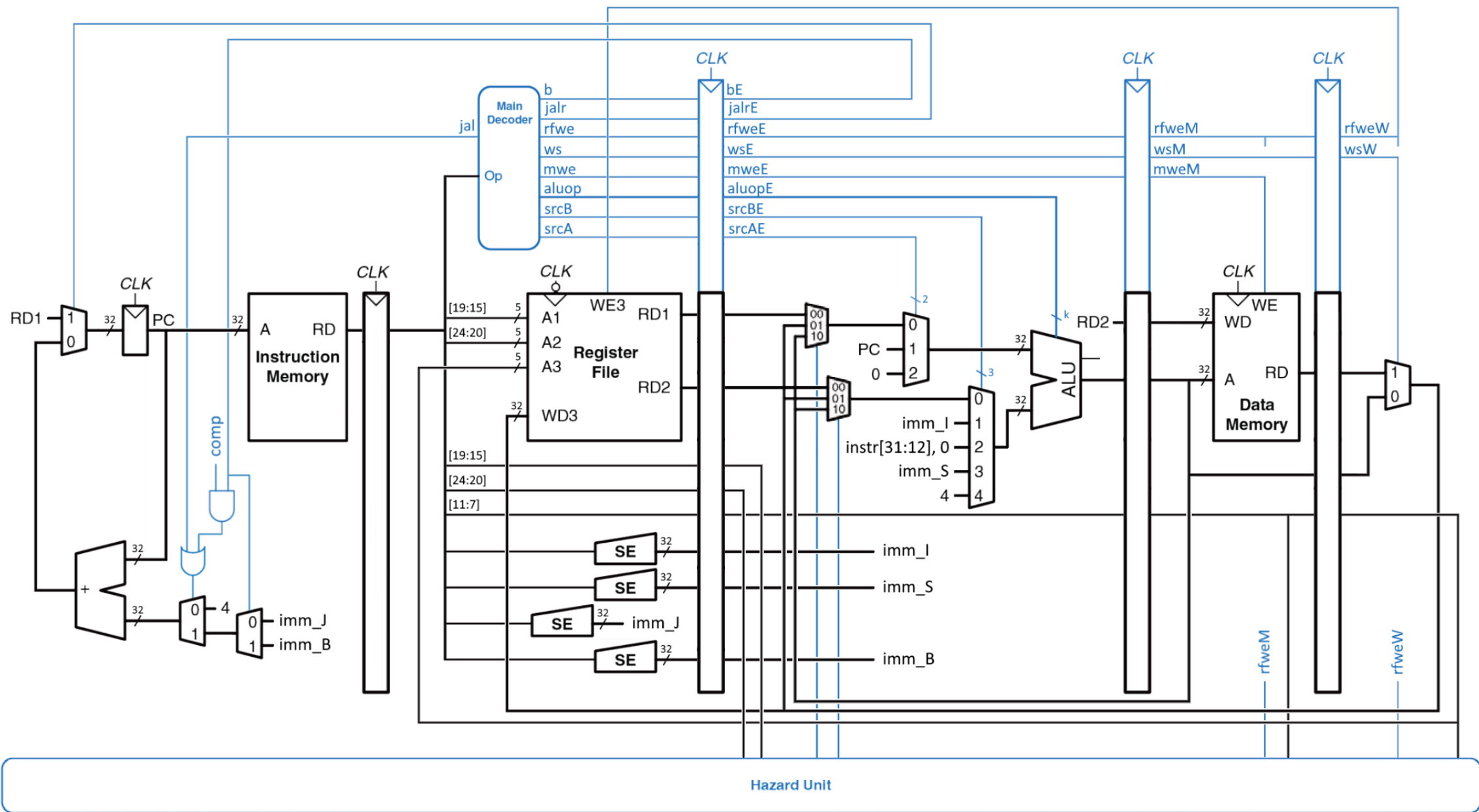




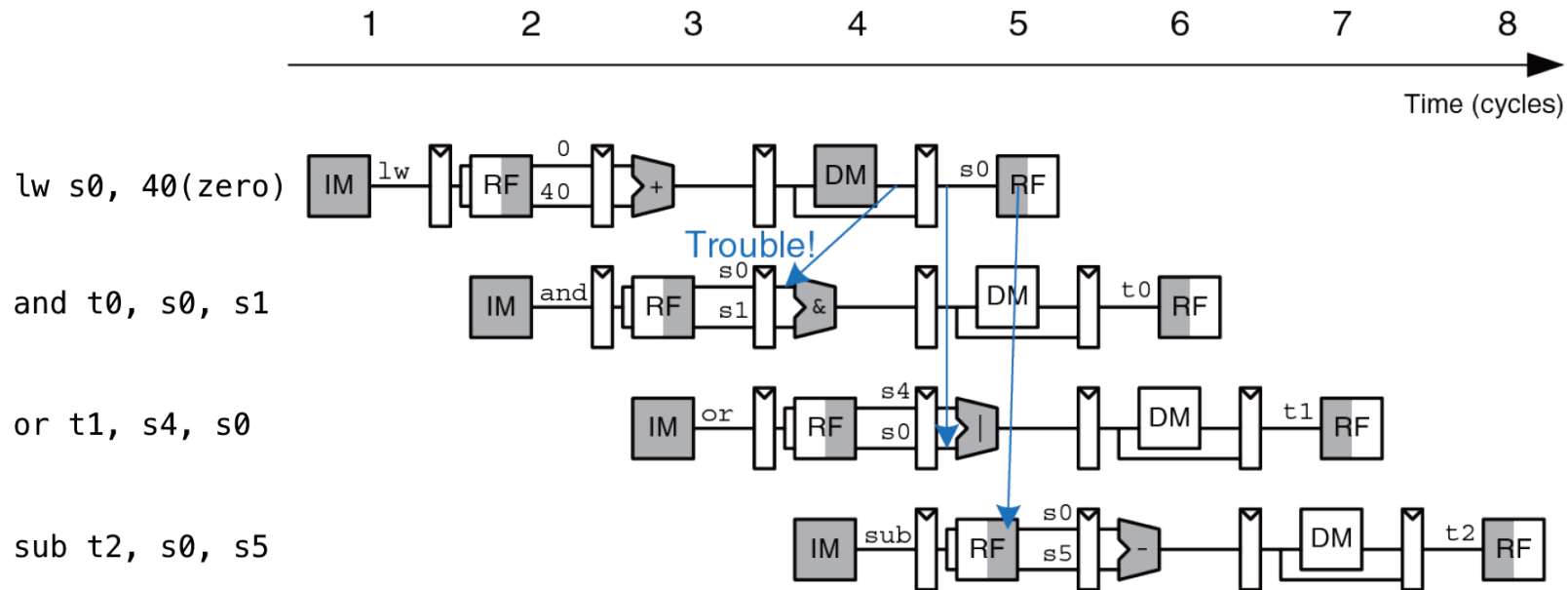




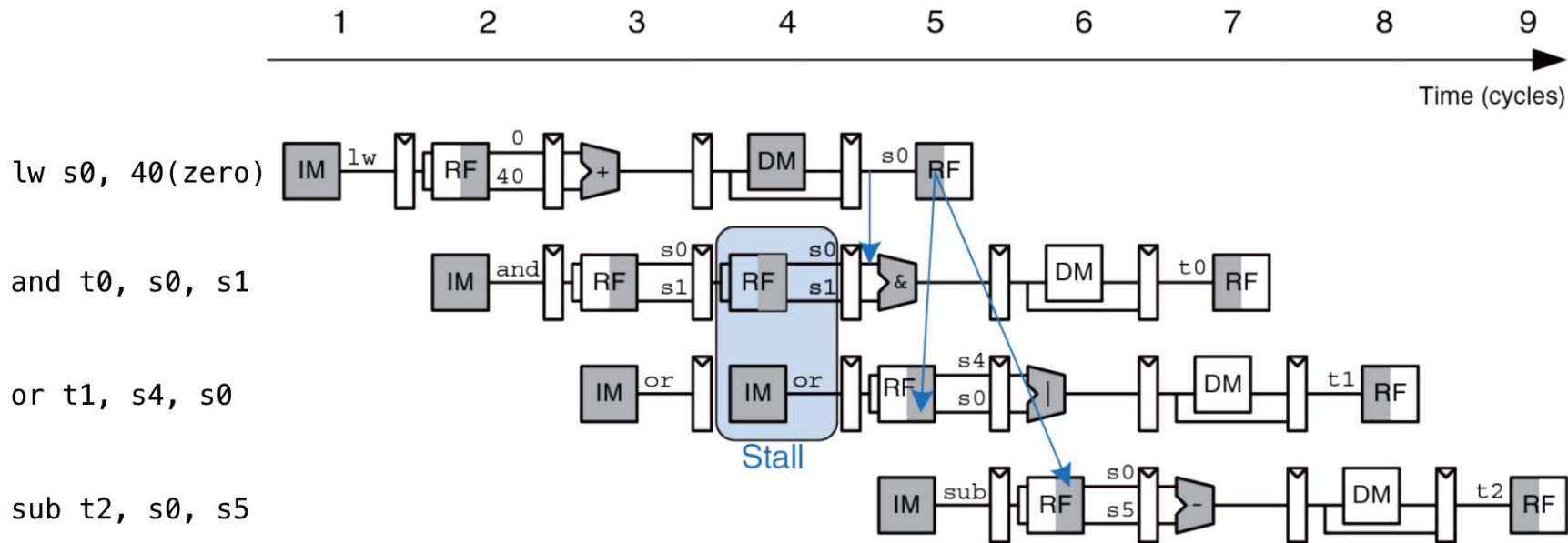


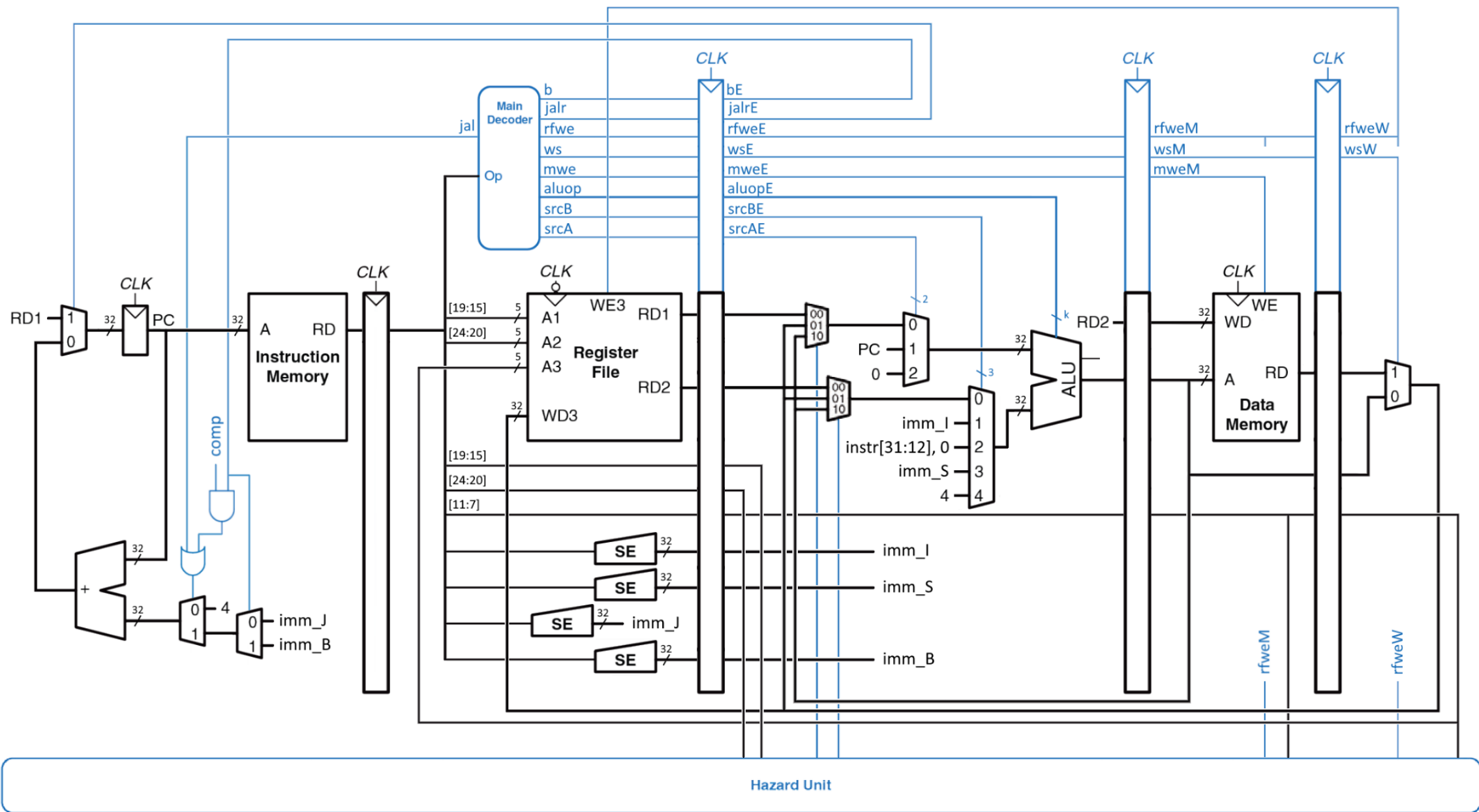


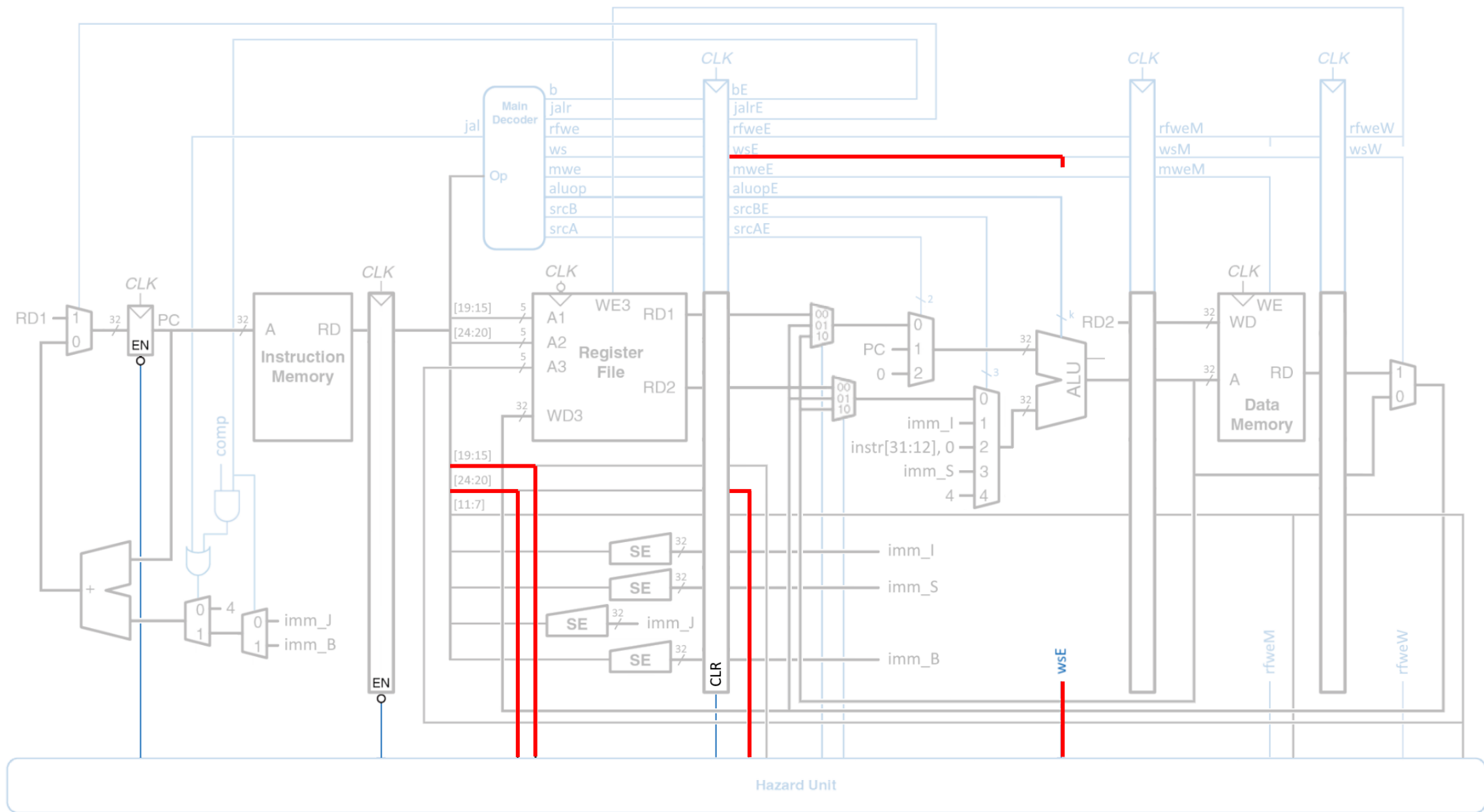
# Конфликт по данным



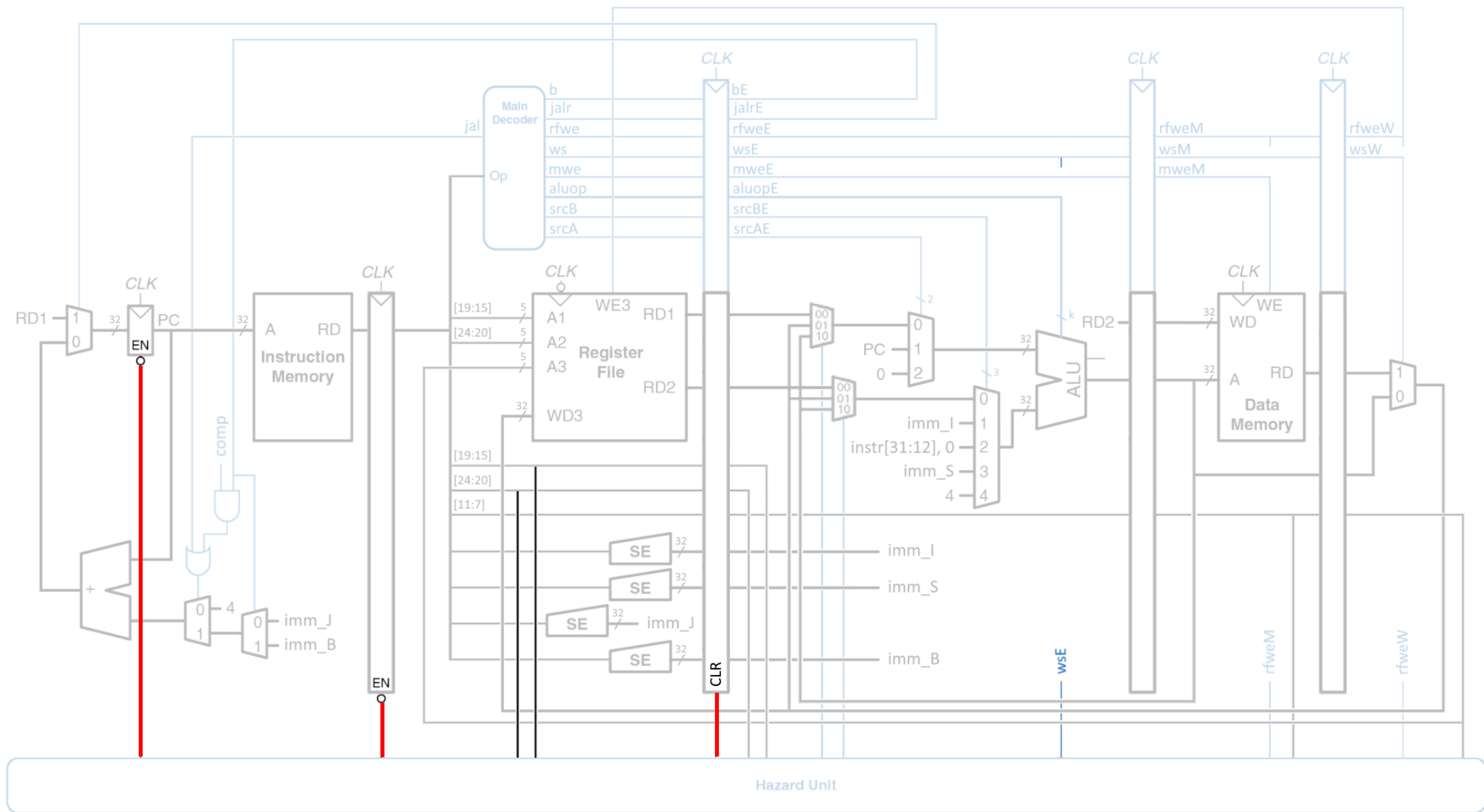
# Конфликт по данным

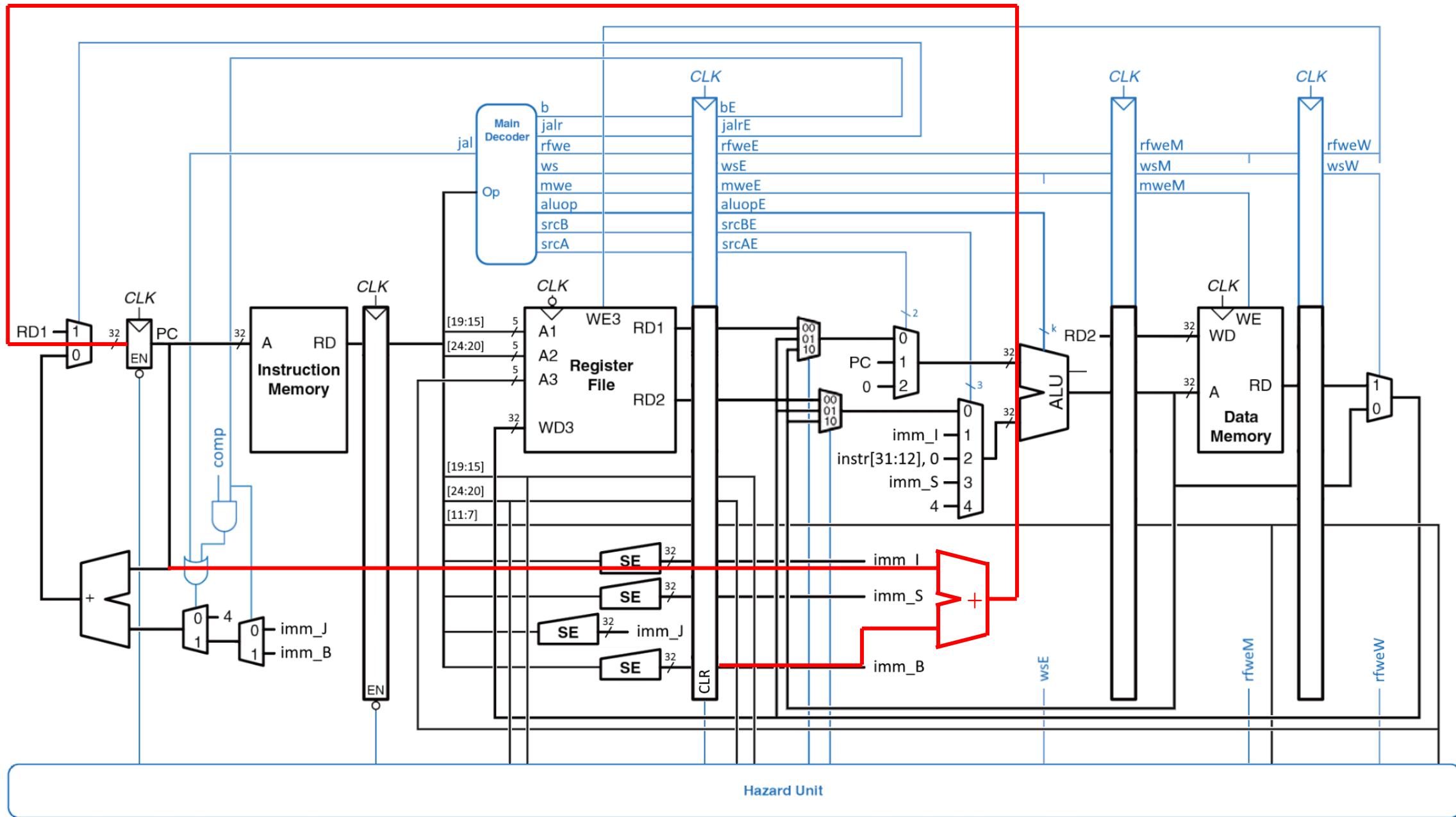












# Конфликты конвейера

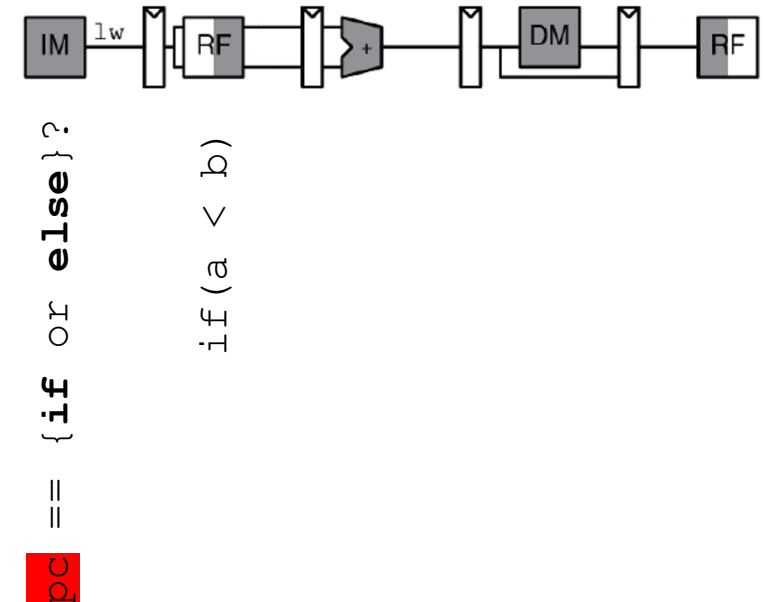
Структурные

```
mul s1, s1  
mul t2, t4  
mul t1, t3
```

По данным

```
add s3, t1, t2  
sub s4, s3, s1  
or s5, s3, t2
```

По управлению



# Производительность конвейера

- Состояние **ожидания** – конвейер пропускает один или несколько тактов из-за того, что не готовы операнды
- Состояние **простоя** – конвейер пропускает один или несколько тактов потому, что данный этап конвейера не используется в данной команде

$$N_{\text{конв}} = N_{\text{идеальный}} + N_{\text{стр}} + N_{\text{дан}} + N_{\text{упр}}$$

# Оценка производительности

$$Execution\ Time = (\# instructions) \left( \frac{cycles}{instruction} \right) \left( \frac{seconds}{cycle} \right)$$

1.15



# Оценка производительности

## Однотактный

$$T_{c1} = 30 + 2(250) + 150 + 200 + 25 + 20 = 925 \text{ пс.}$$

$$T_1 = (100 \times 10^9 \text{ команд}) (1 \text{ такт/команду}) (925 \times 10^{-12} \text{ с/такт}) = 92,5 \text{ с.}$$

## Многотактный

$$T_c = t_{pcq} + t_{mux} + \max(t_{ALU} + t_{mux}, t_{mem}) + t_{setup}$$

$$T_{c2} = 30 + 25 + 250 + 20 = 325 \text{ пс.}$$

$$T_2 = (100 \times 10^9 \text{ команд})(4,12 \text{ тактов/команду}) (325 \times 10^{-12} \text{ с/такт}) = 133,9 \text{ с.}$$

# Оценка производительности

## Конвейерный

$$T_c = \max \left( \begin{array}{l} t_{pcq} + t_{mem} + t_{setup} \\ 2(t_{RFread} + t_{mux} + t_{eq} + t_{AND} + T_{mux} + t_{setup}) \\ t_{pcq} + t_{mux} + t_{mux} + t_{ALU} + t_{setup} \\ t_{pcq} + t_{memwrite} + t_{setup} \\ (t_{pcq} + t_{mux} + t_{RFwrite}) \end{array} \right) \left\{ \begin{array}{l} \text{Fetch} \\ \text{Decode} \\ \text{Execute} \\ \text{Memory} \\ \text{Writeback} \end{array} \right.$$

$$T_{c3} = 550 \text{ пс.}$$

$$T_3 = (100 \times 10^9 \text{ команд})(1,15 \text{ тактов/команду})(550 \times 10^{-12} \text{ с/такт}) = 63,3 \text{ с.}$$