



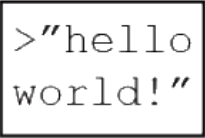


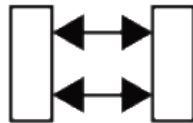
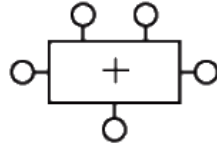



Архитектуры процессорных систем

Лекция 8. Многотактный процессор RISC-V

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

План лекции

- Классификация микроархитектур
- Кодирование инструкций RISC-V
- Синтез процессора с многотактной микроархитектурой
- Устройство управления
 - С жесткой логикой
 - С микропрограммным управлением
- Оценка производительности полученного процессора

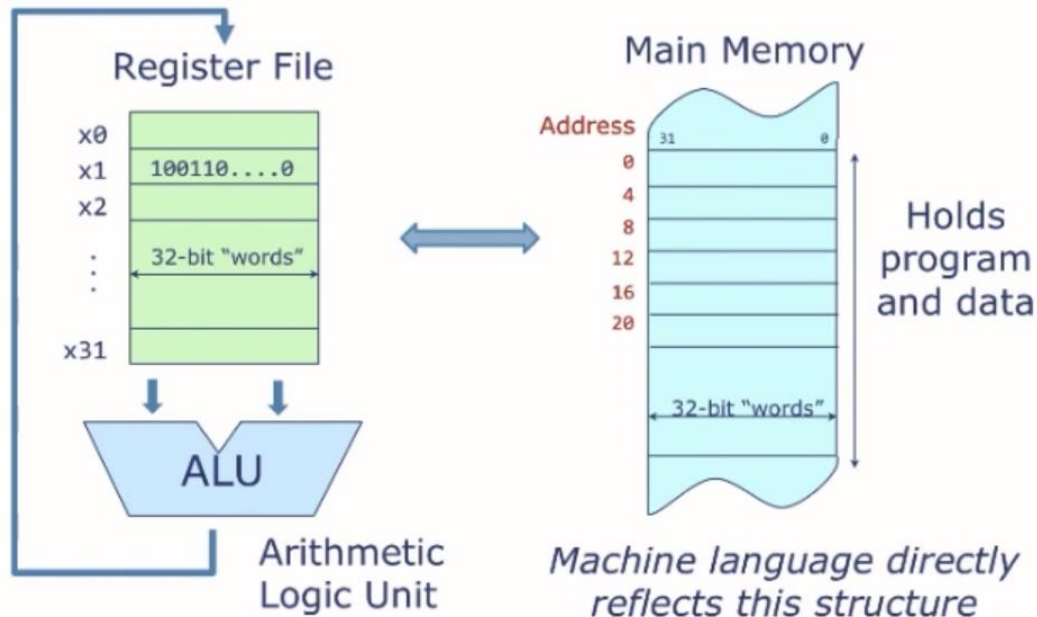
Application Software	
Operating Systems	
Architecture	
Micro-architecture	
Logic	
Digital Circuits	
Devices	
Physics	

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

Instruction Set Architecture (ISA)

- Система команд
- Средства для выполнения команд
 - Форматы данных
 - Системы регистров
 - Способы адресации
 - Модели памяти

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-immeliate `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	$rd = rs1 + rs2$ $rd = rs1 - rs2$ $rd = rs1 \wedge rs2$ $rd = rs1 \vee rs2$ $rd = rs1 \& rs2$ $rd = rs1 \ll rs2$ $rd = rs1 \gg rs2$ $rd = rs1 \ggg rs2$ $rd = (rs1 < rs2) ? 1 : 0$ $rd = (rs1 < rs2) ? 1 : 0$	R	0110011	0x0 0x0 0x4 0x6 0x7 0x1 0x5 0x5 0x2 0x3	0x00 0x20 0x00 0x00 0x00 0x00 0x00 0x20 0x00 0x00	<code>op rd, rs1, rs2</code> <code>xor x2, x5, x6</code> <code>sll x7, x11, x12</code>
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	$rd = rs1 + imm$ $rd = rs1 \wedge imm$ $rd = rs1 \vee imm$ $rd = rs1 \& imm$ $rd = rs1 \ll imm$ $rd = rs1 \gg imm$ $rd = rs1 \ggg imm$ $rd = (rs1 < imm) ? 1 : 0$ $rd = (rs1 < imm) ? 1 : 0$	I	0010011	0x0 0x4 0x6 0x7 0x1 0x5 0x5 0x2 0x3	- 0x00 0x00 0x20 -	<code>op rd, rs1, imm</code> <code>addi x6, x3, -12</code> <code>ori x3, x1, 0x8F</code>
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	-	<code>op rd, imm(rs1)</code> <code>lh x1, 8(x5)</code>
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	-	<code>op rs2, imm(rs1)</code> <code>sw x1, 0xFC(x12)</code>
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	-	<code>comp rs1, rs2, imm</code> <code>beq x8, x9, offset</code> <code>bltu x20, x21, 0xFC</code>
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	- 0x0	-	<code>jal x1, offset</code> <code>jalr x1, 0(x5)</code>
lui auipc	Load Upper Immediate Add Upper Immediate to PC	Загрузить константу в сдвинутую на 12 Сохранить счетчик команд в сумме с константой << 12	$rd = imm \ll 12$ $rd = PC + (imm \ll 12)$	U	0110111 0010111	-	-	<code>lui x3, 0xFFFFF</code> <code>auipc x2, 0x000FF</code>
ecall ebreak	Environment CALL Environment BREAK	Передача управления операционной системе Передача управления отладчику	Воспринимать как nop	I	1110011	-	-	-

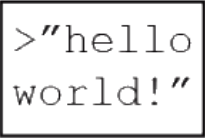



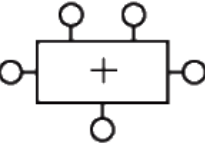



Кодирование инструкций RISC-V

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	
	imm _{11:0}		rs1	funct3	rd	op		imm _{11:0}		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	

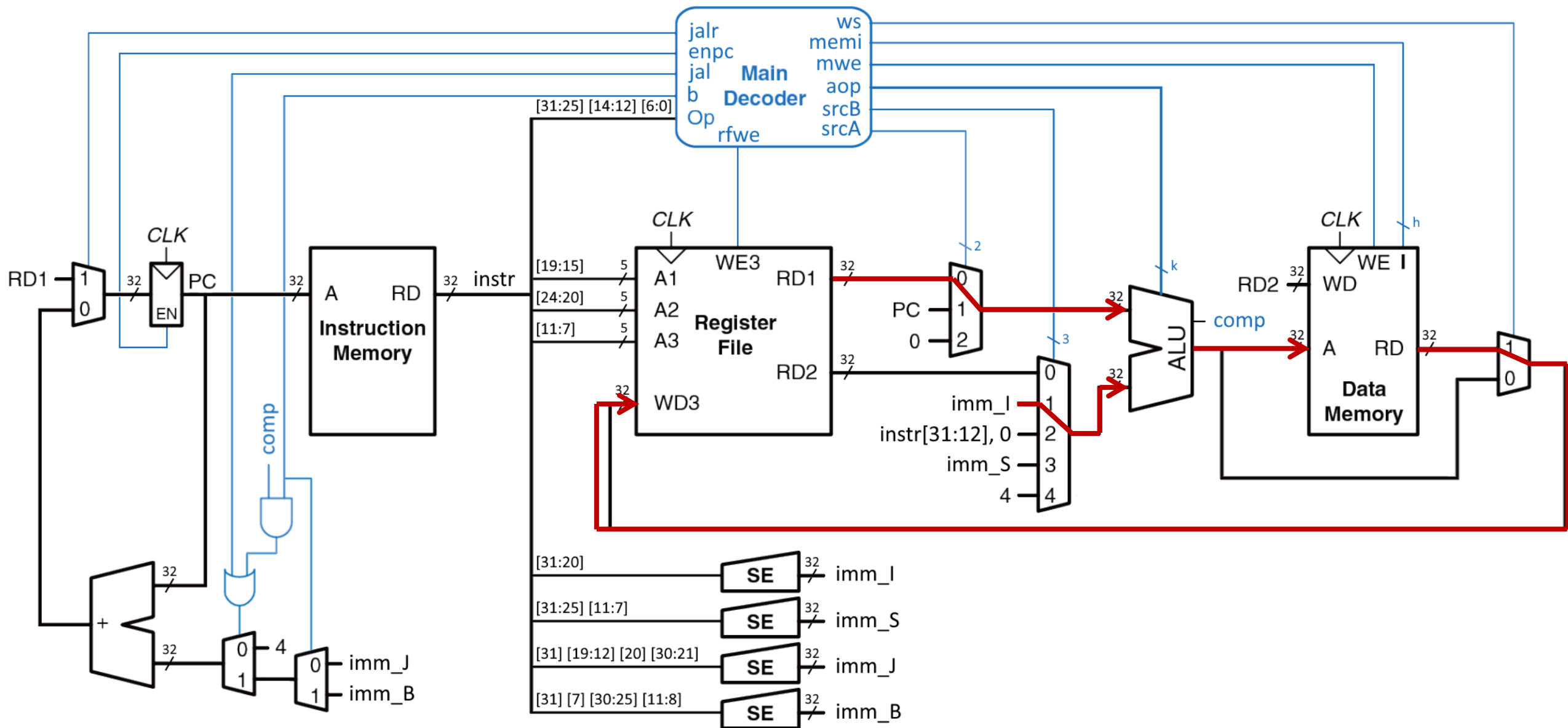
Application Software	
Operating Systems	
Architecture	
Micro-architecture	
Logic	
Digital Circuits	
Devices	
Physics	

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

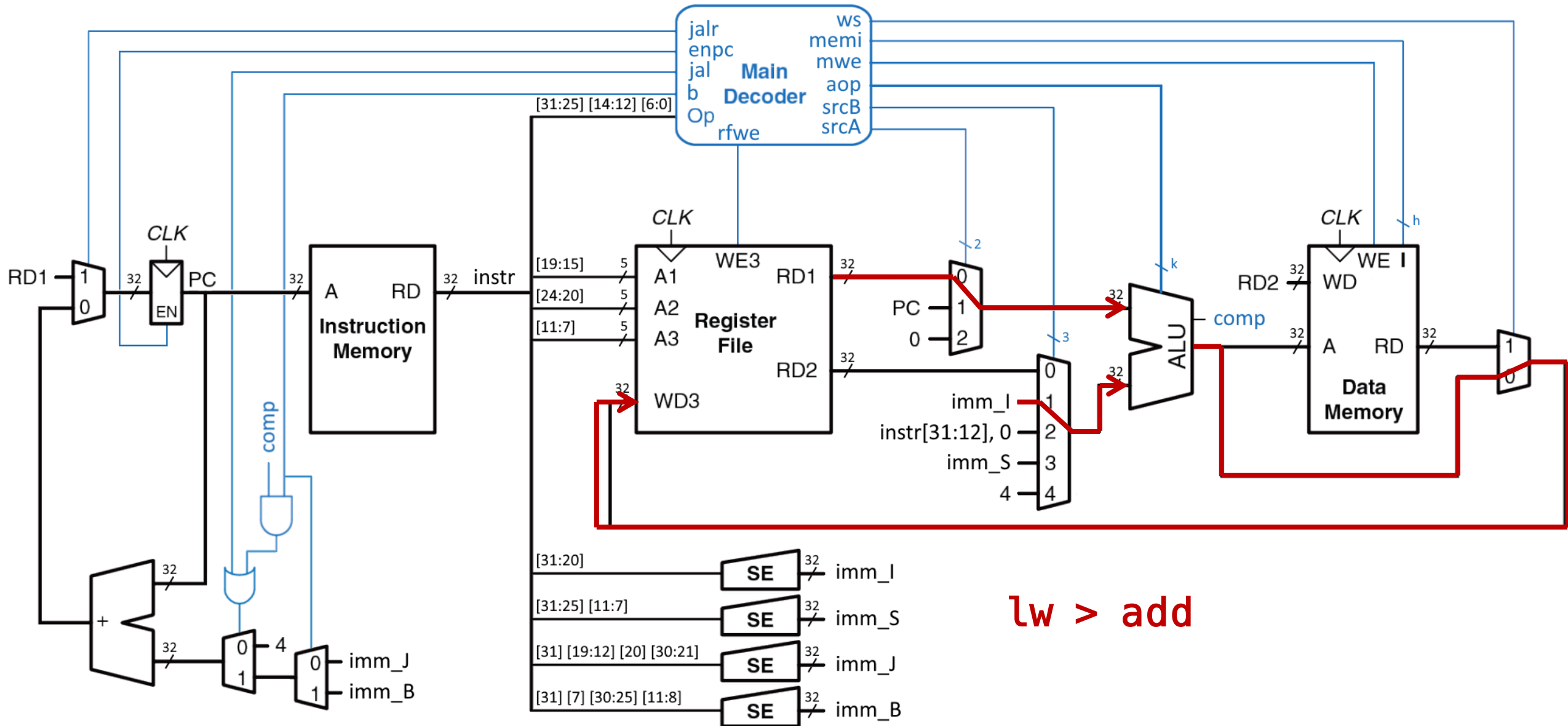
Микроархитектуры

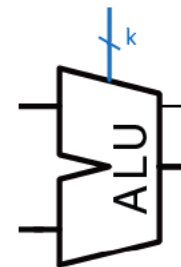
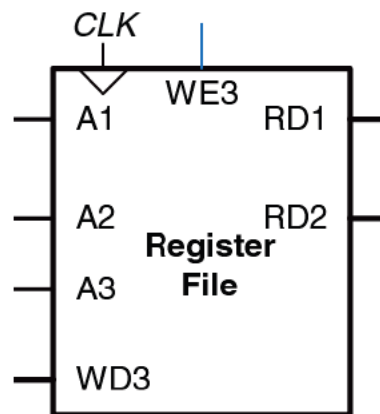
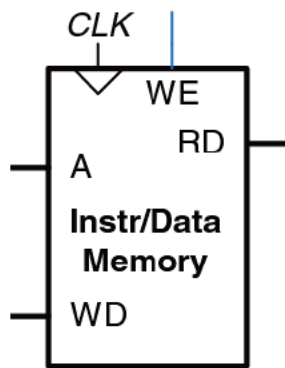
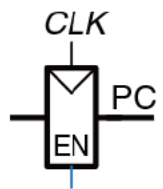
- **Однотактная**
 - выполняет одну инструкцию за один такт
- **Многотактная**
 - выполняет одну инструкцию за несколько более коротких тактов
- **Конвейерная**
 - результат применения принципа конвейерной обработки к однотактной микроархитектуре

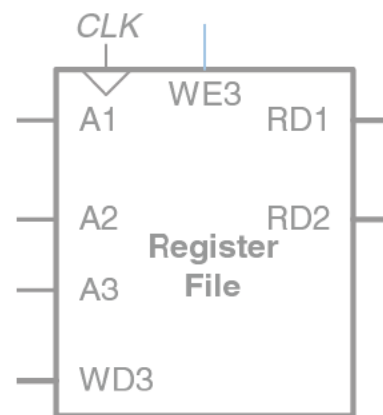
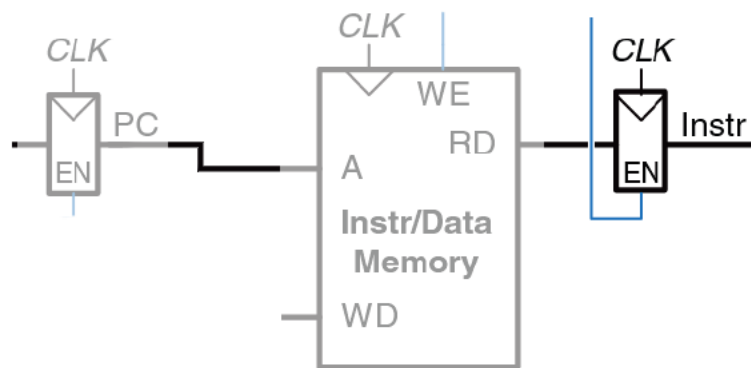
lw

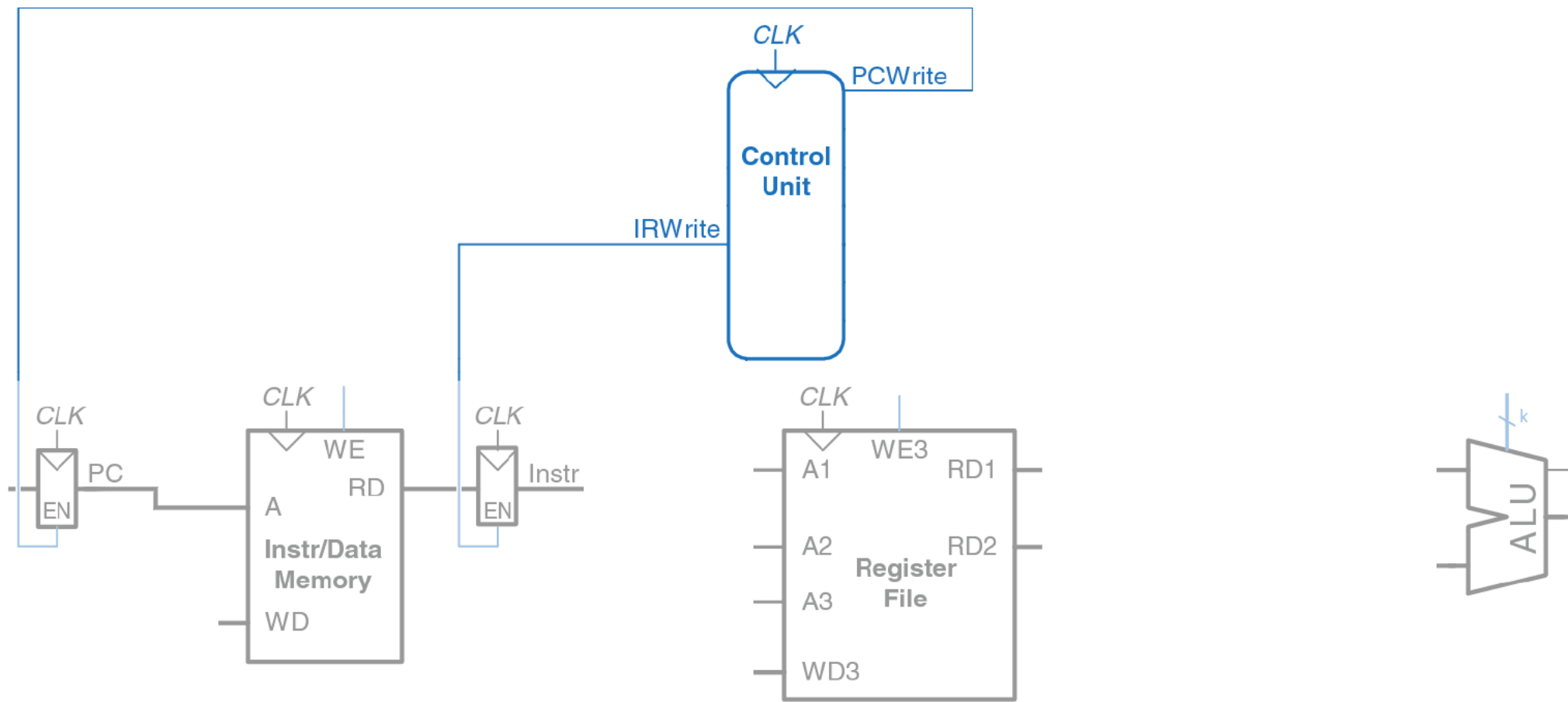


add





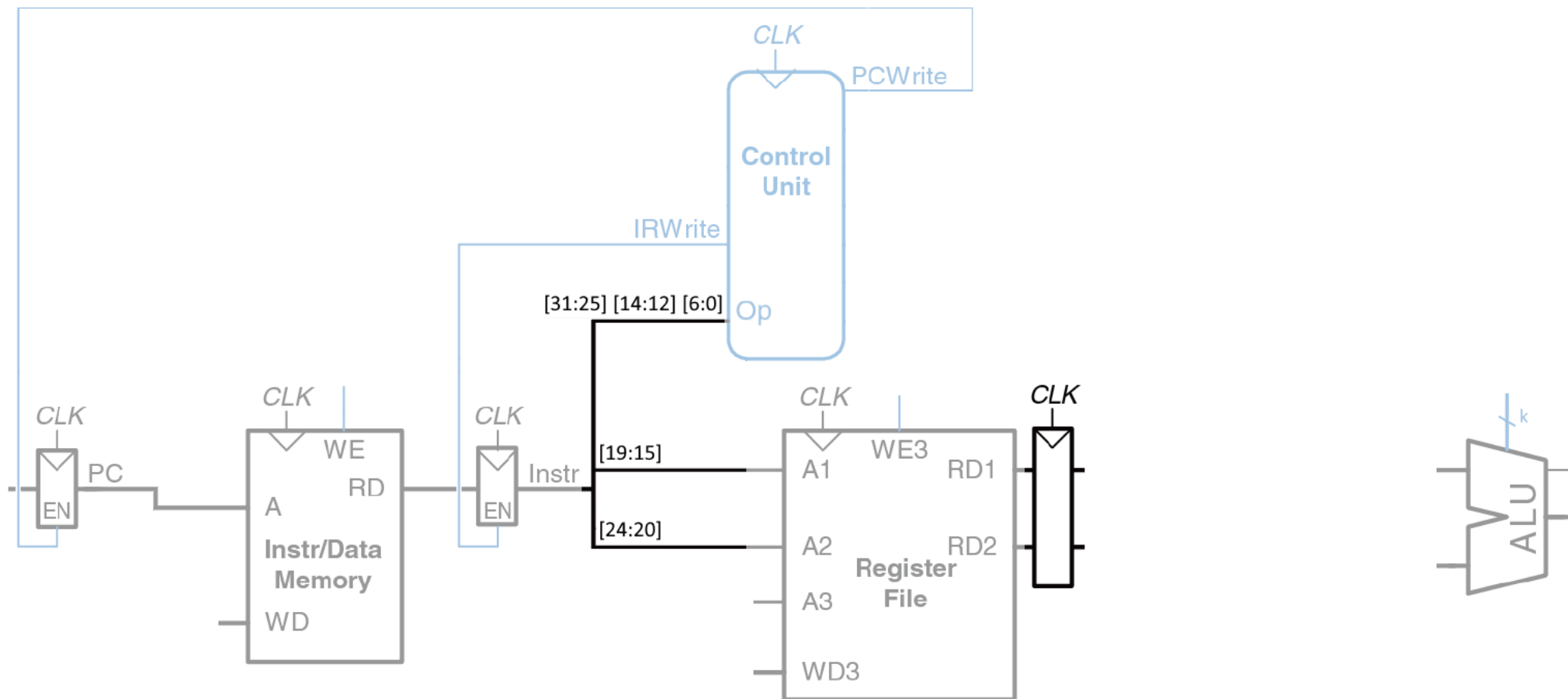




add xN, xM, xK

$rd = rs1 + rs2$

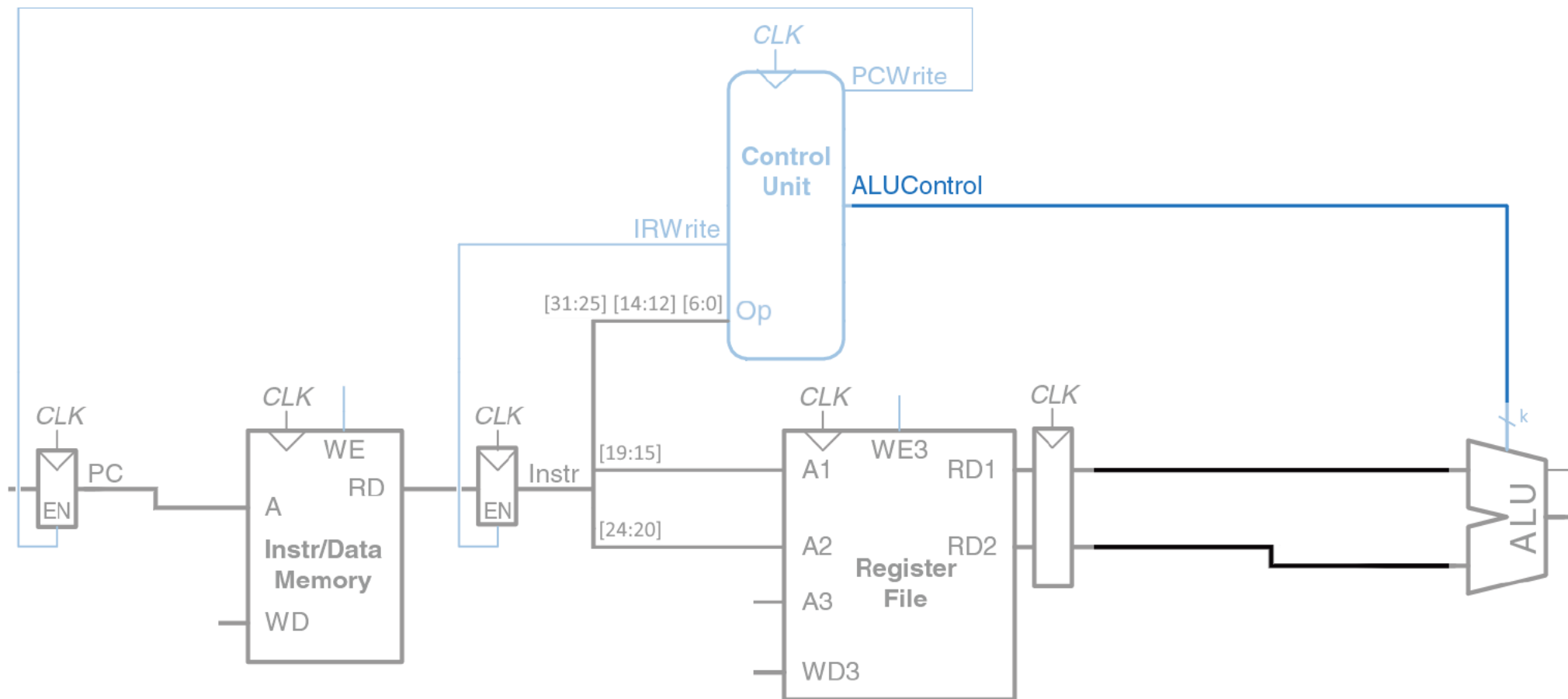
31	27	26	25	24	20	19	15	14	12	11	7	6	0	
funct7				rs2		rs1		funct3		rd		opcode		R-type
0000000				rs2		rs1		000		rd		0110011		ADD



add xN, xM, xK

rd = rs1 + rs2

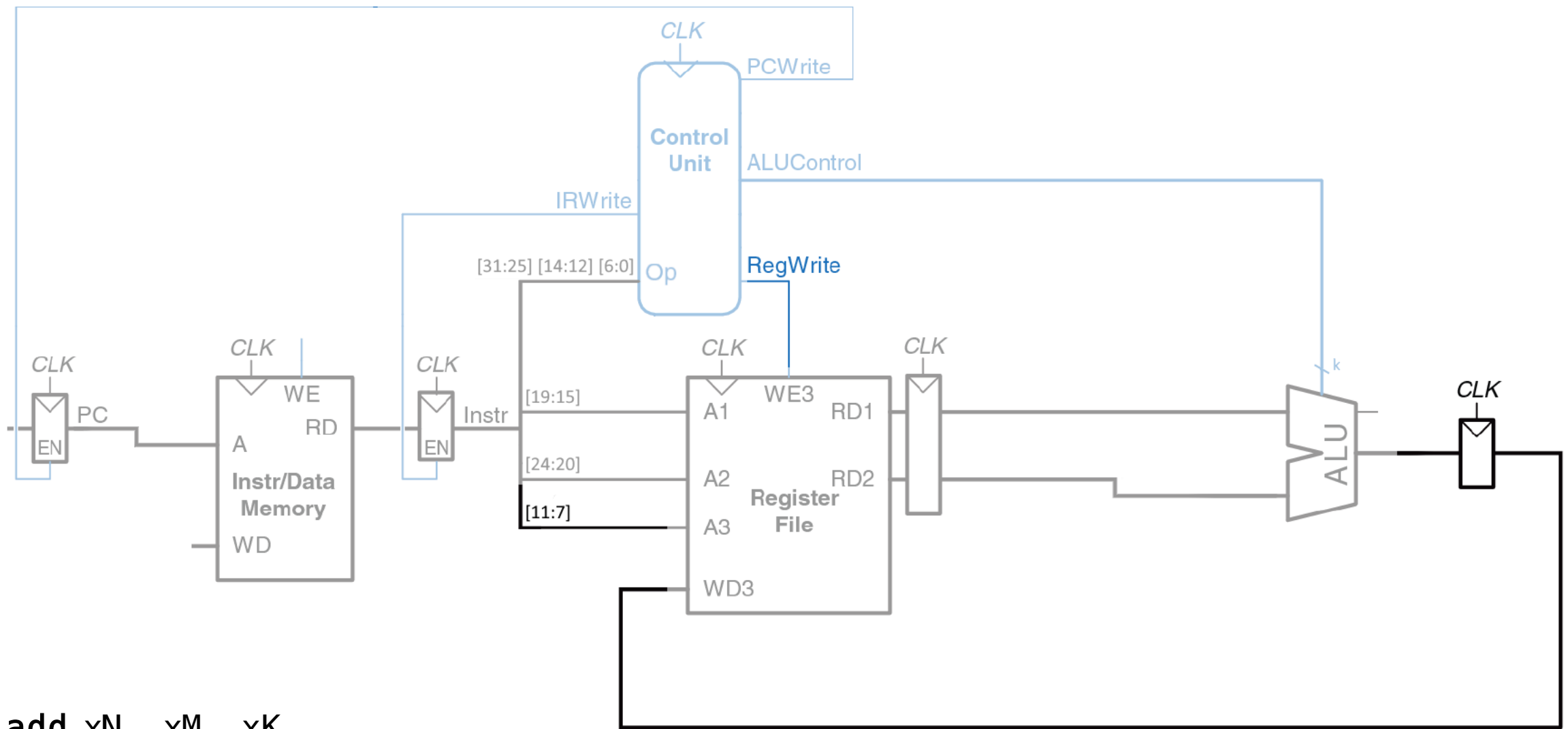
31	27	26	25	24	20	19	15	14	12	11	7	6	0	
funct7				rs2		rs1		funct3		rd		opcode		R-type
0000000				rs2		rs1		000		rd		0110011		ADD



add xN, xM, xK

rd = rs1 + rs2

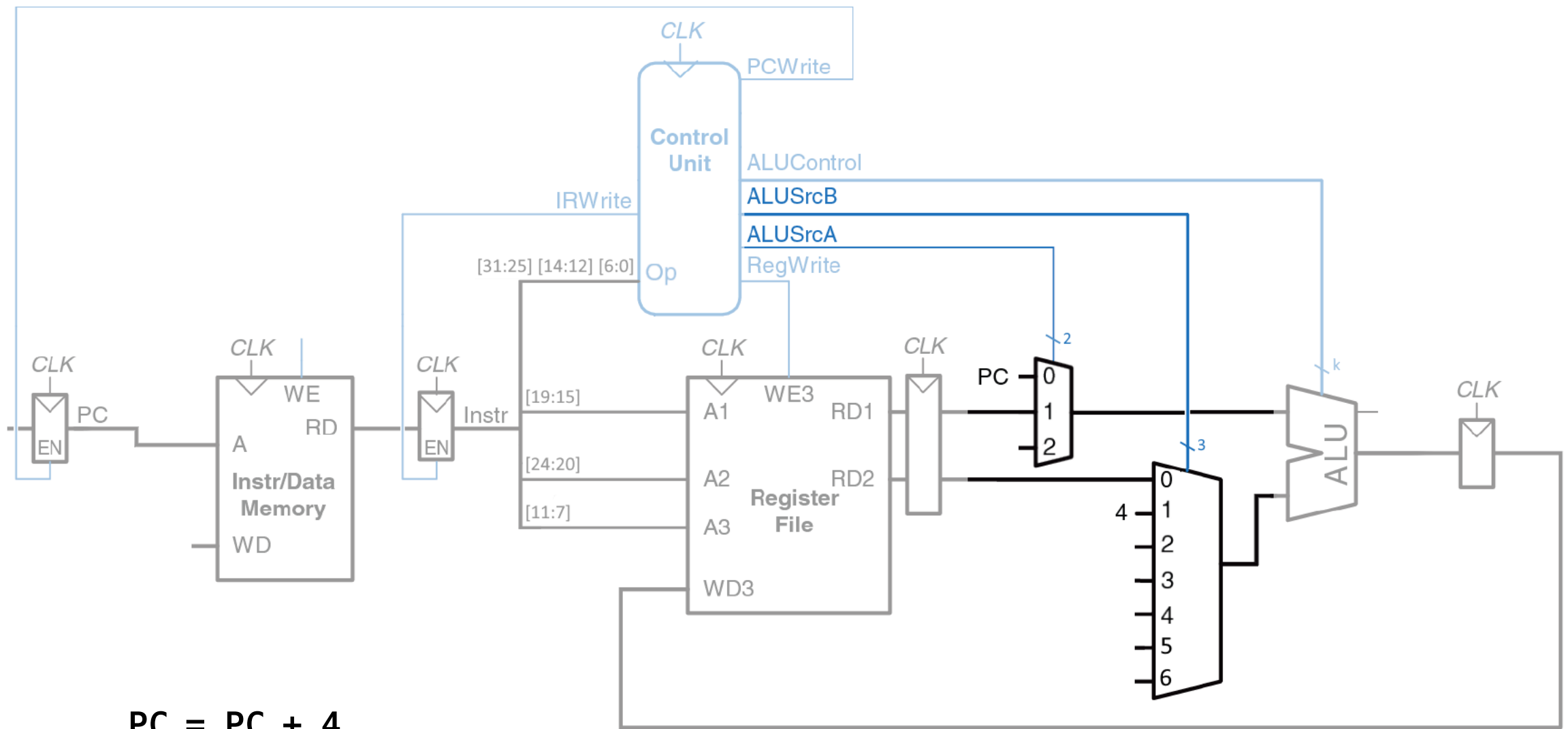
31	27	26	25	24	20	19	15	14	12	11	7	6	0	
funct7				rs2		rs1		funct3		rd		opcode		R-type
0000000				rs2		rs1		000		rd		0110011		ADD



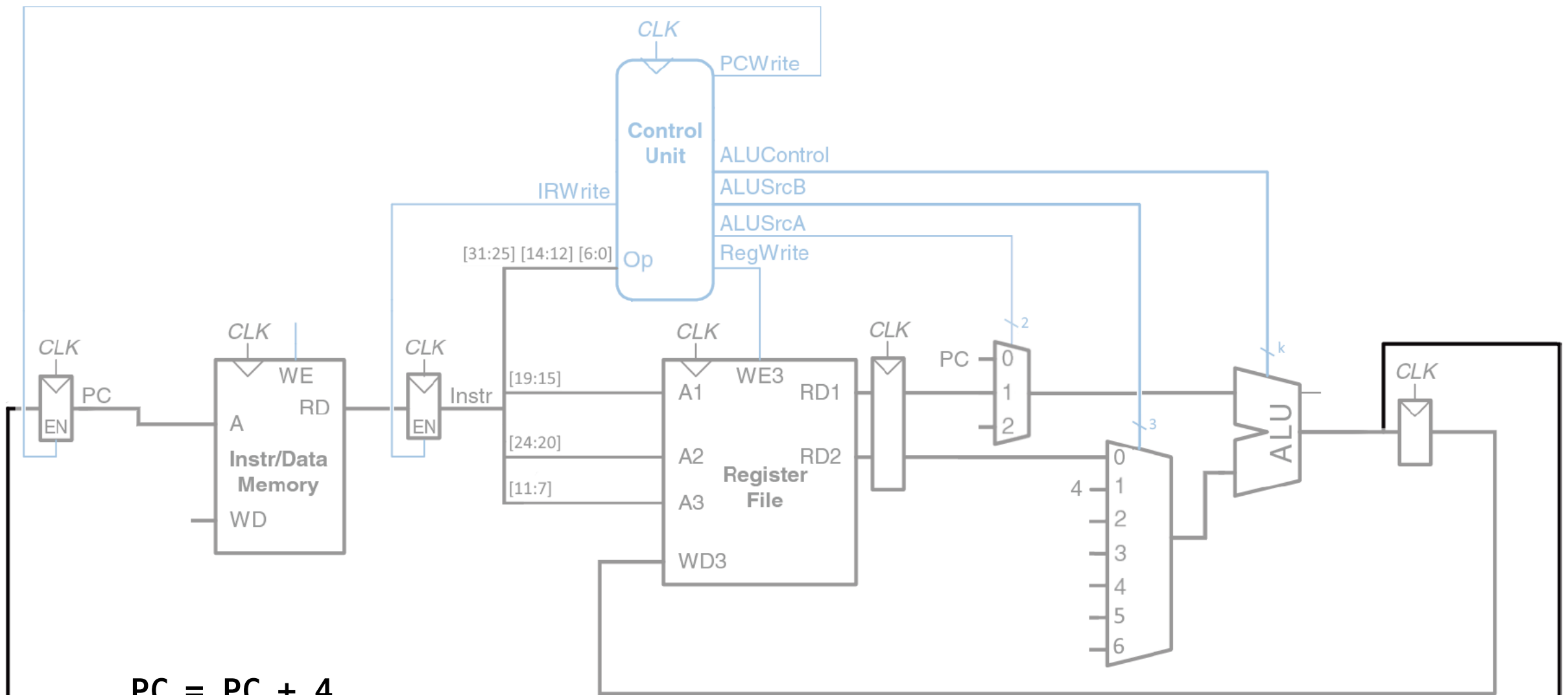
add xN, xM, xK

rd = rs1 + rs2

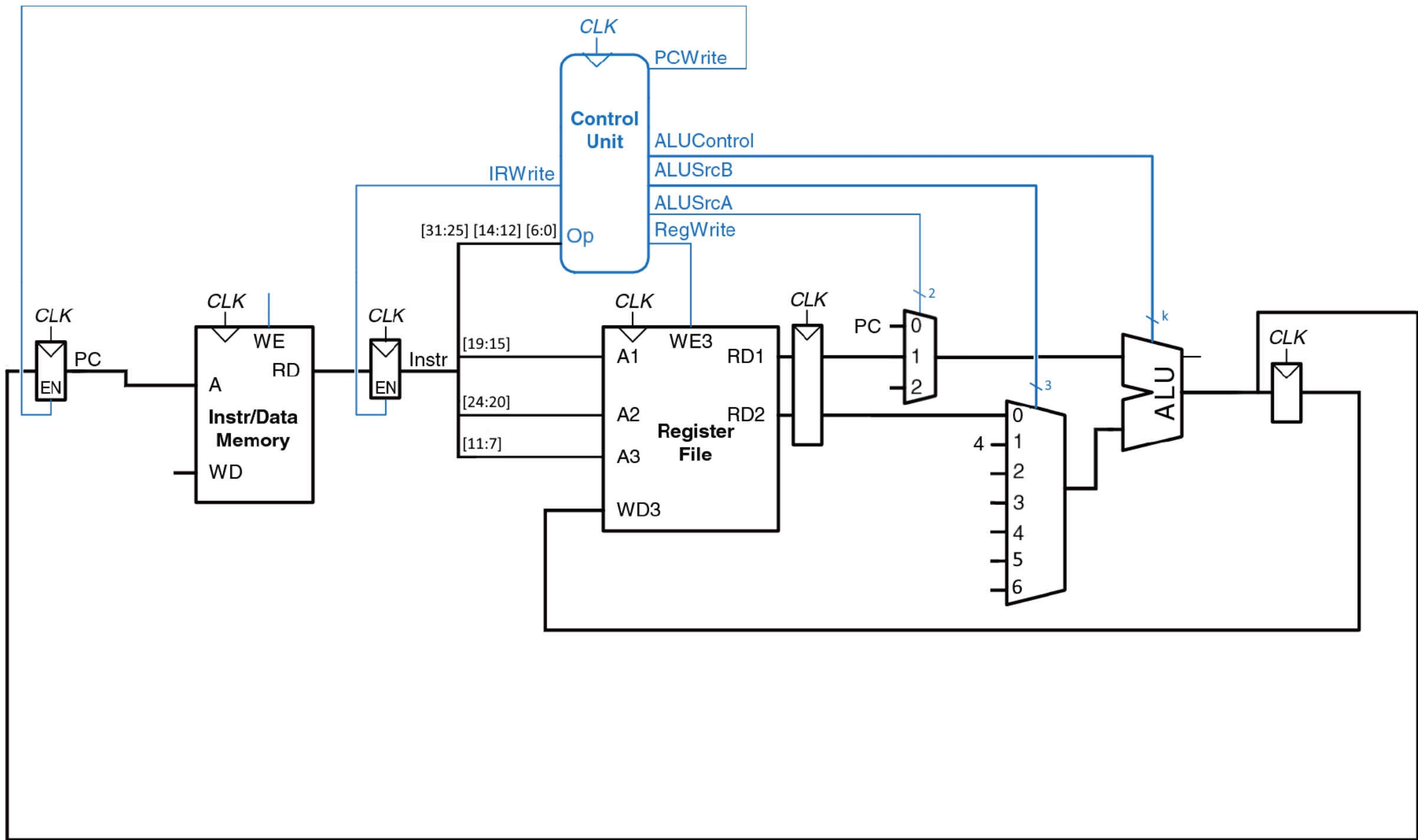
31	27	26	25	24	20	19	15	14	12	11	7	6	0	
funct7				rs2		rs1		funct3		rd		opcode		R-type
0000000				rs2		rs1		000		rd		0110011		ADD



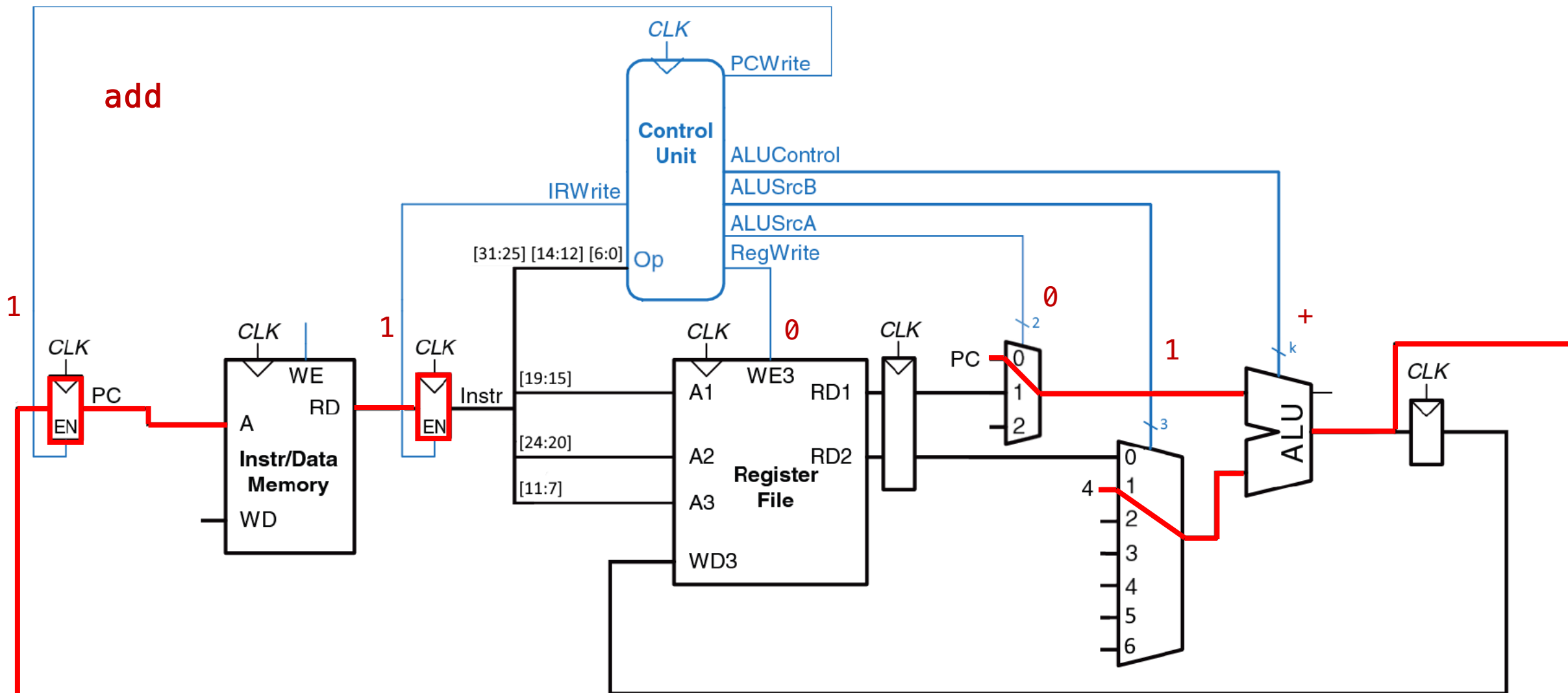
$$PC = PC + 4$$



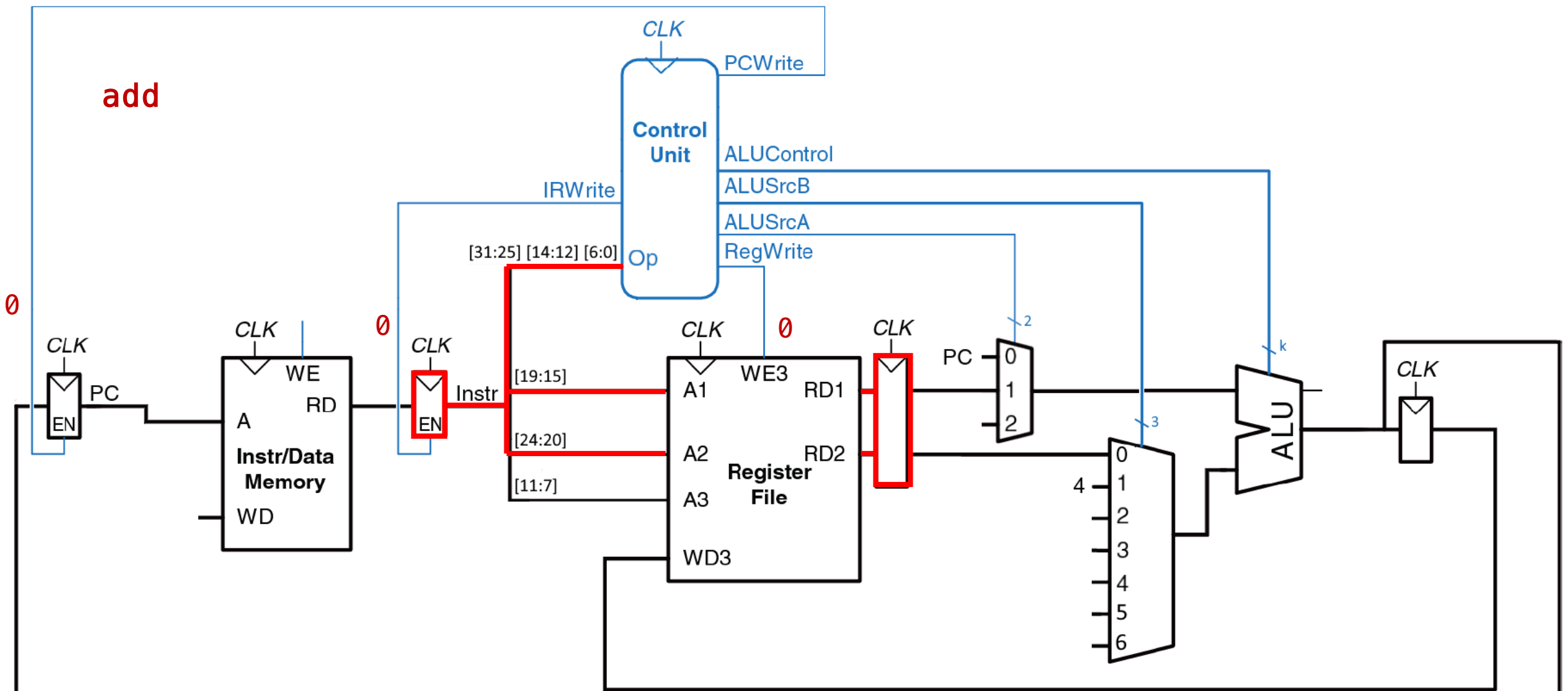
$$PC = PC + 4$$



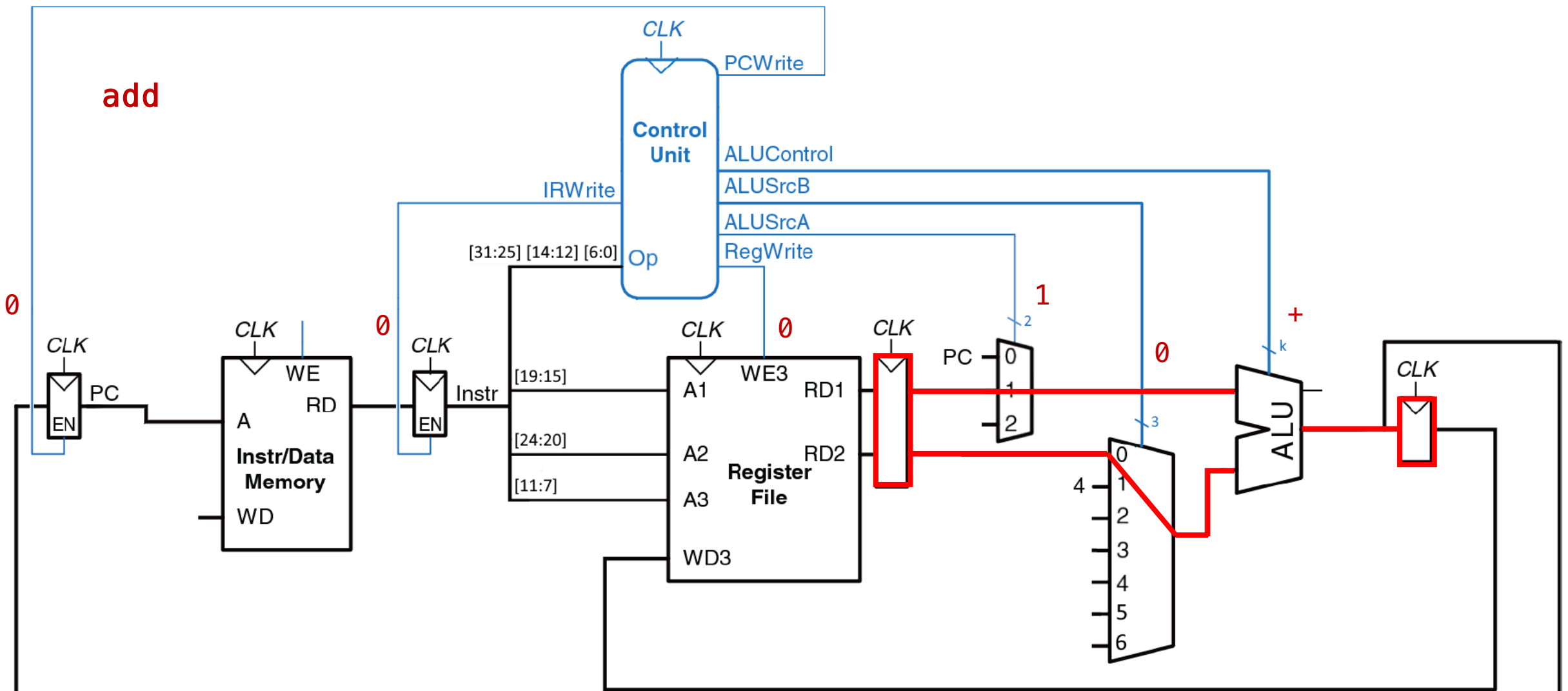
add



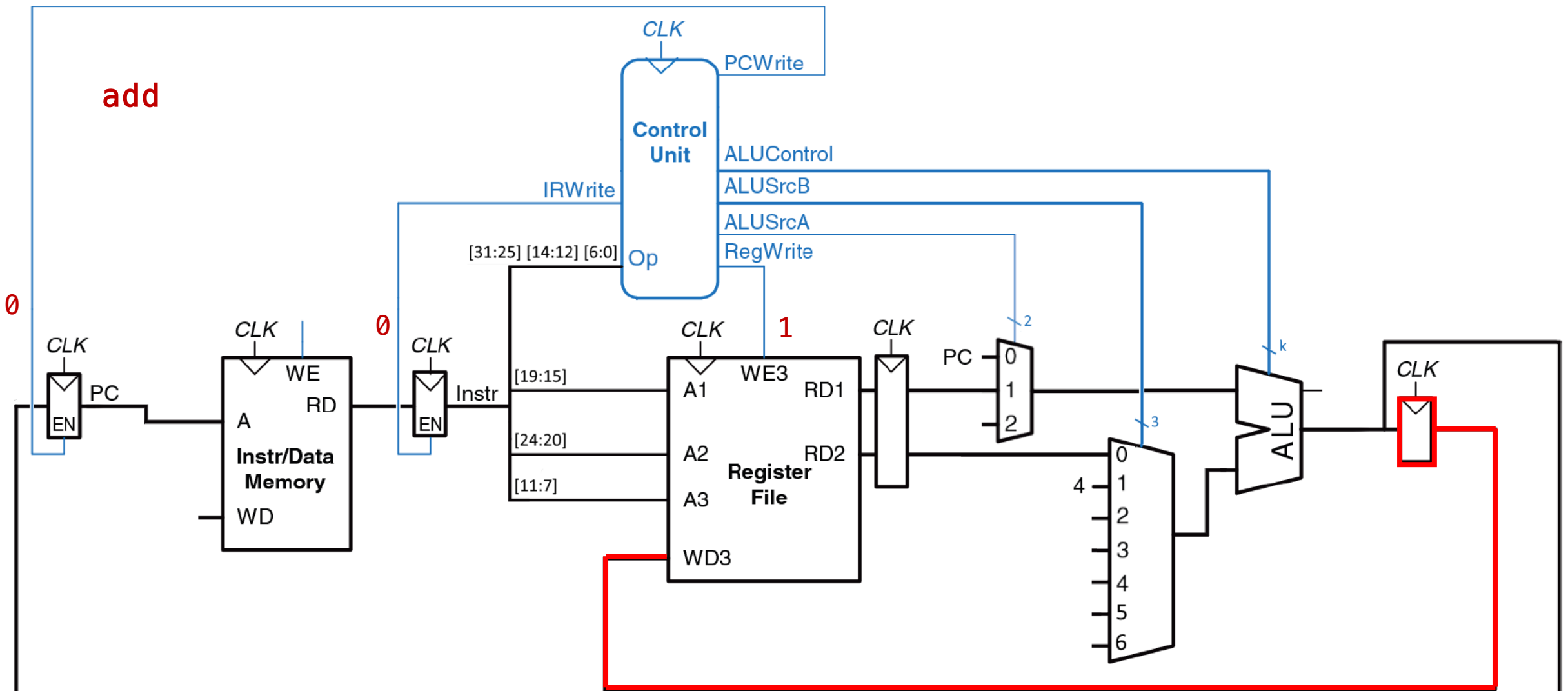
[1] $ReadInstr / PC = PC + 4$



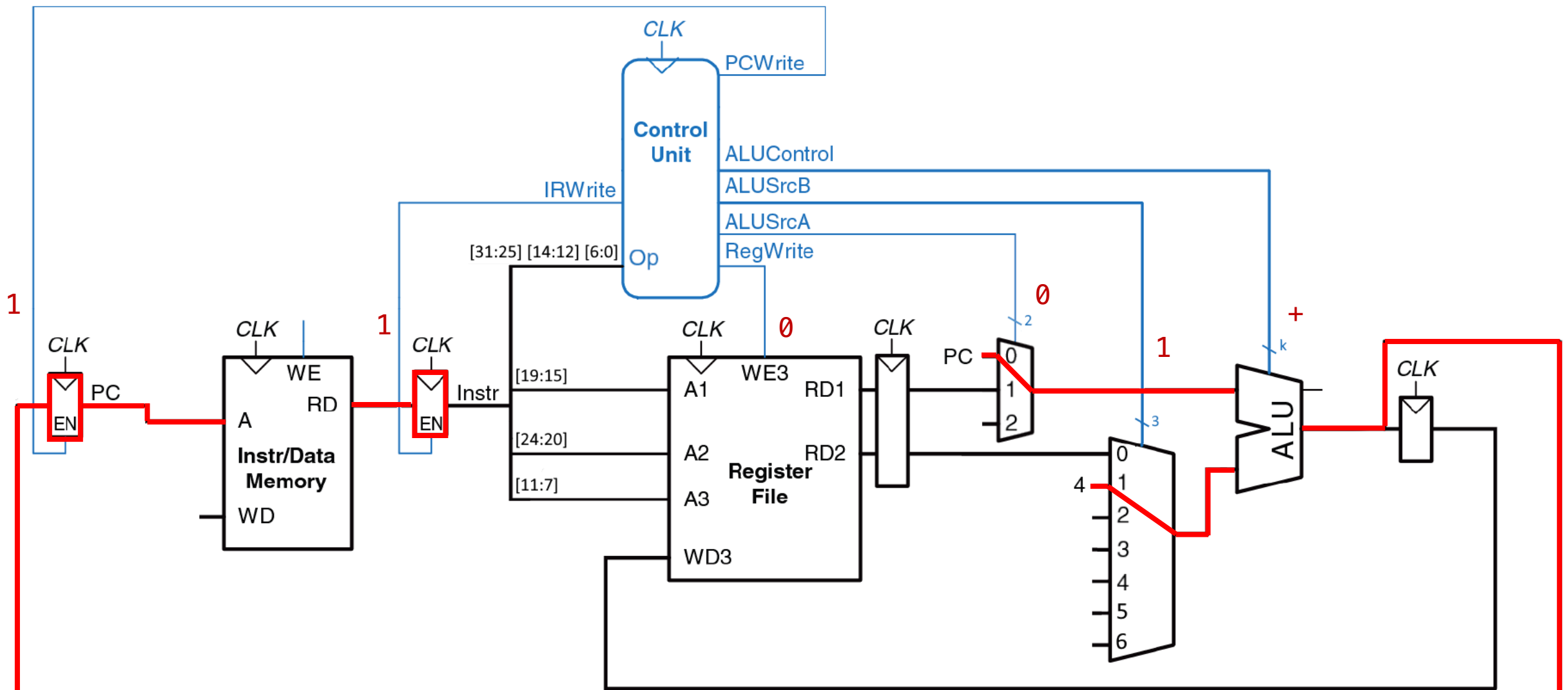
- [1] ReadInstr / $PC = PC + 4$
- [2] DecodeInstr



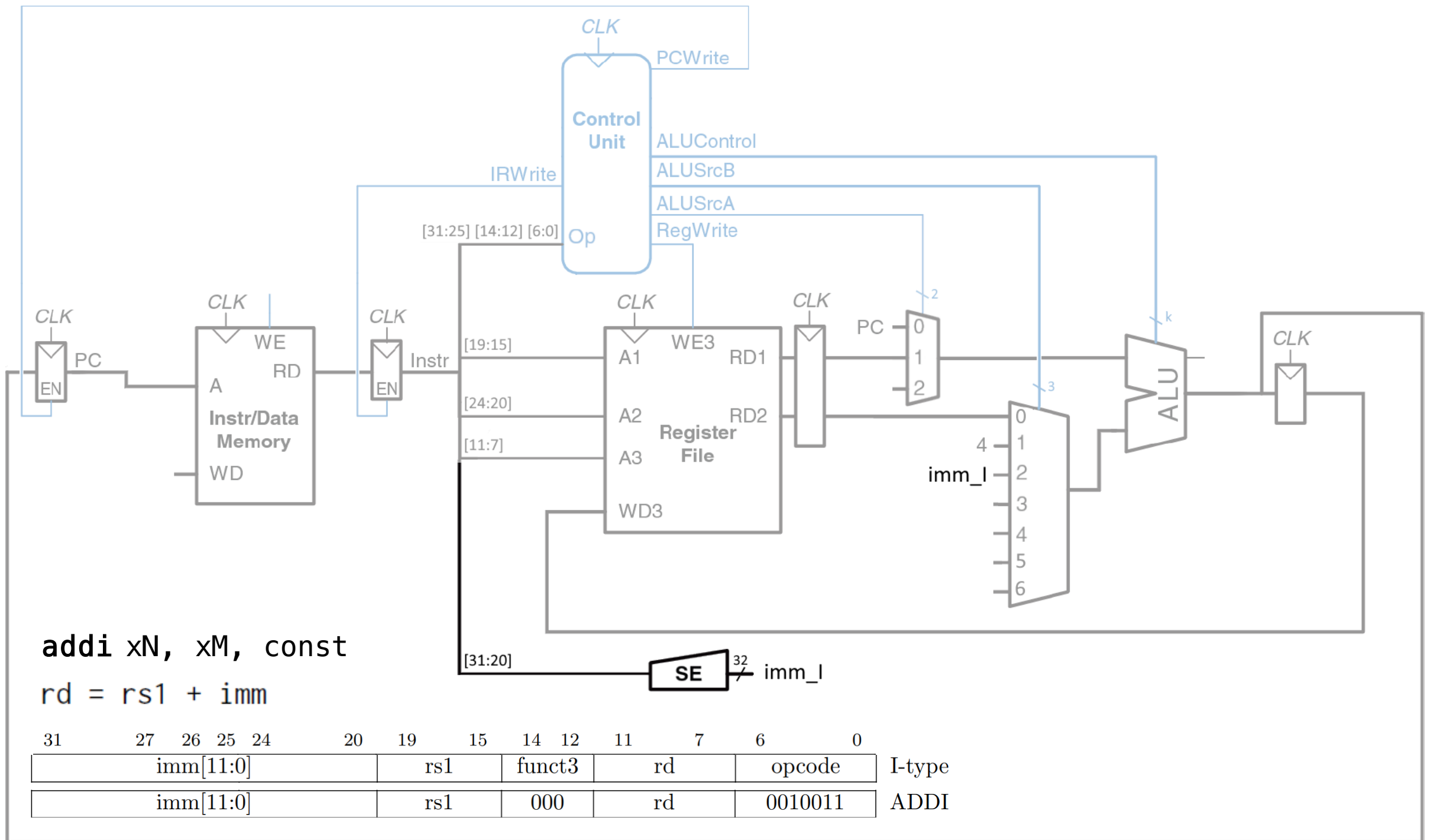
- [1] ReadInstr / PC = PC + 4
- [2] DecodeInstr
- [3] Execute

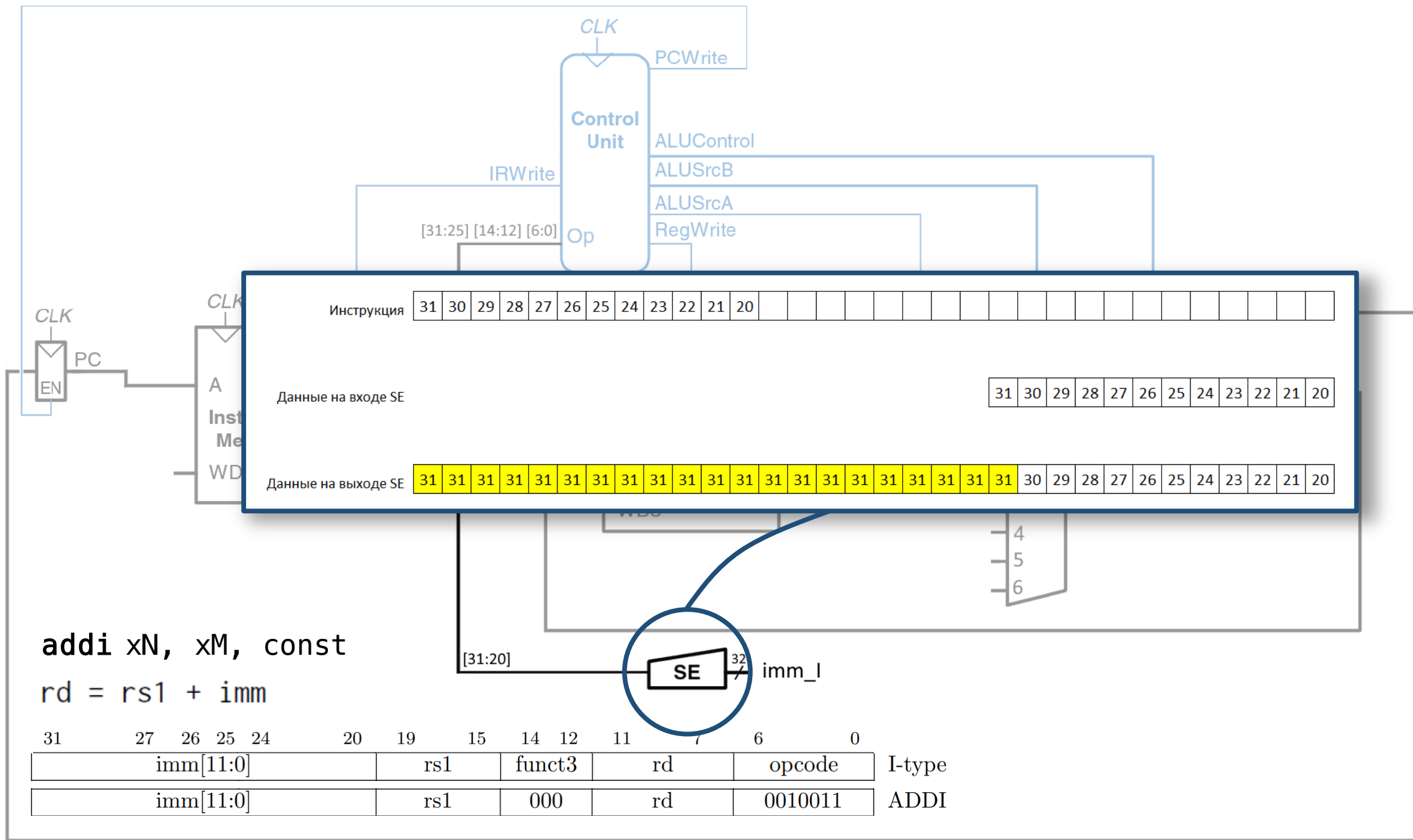


- [1] ReadInstr / $PC = PC + 4$
- [2] DecodeInstr
- [3] Execute
- [4] Write to RF

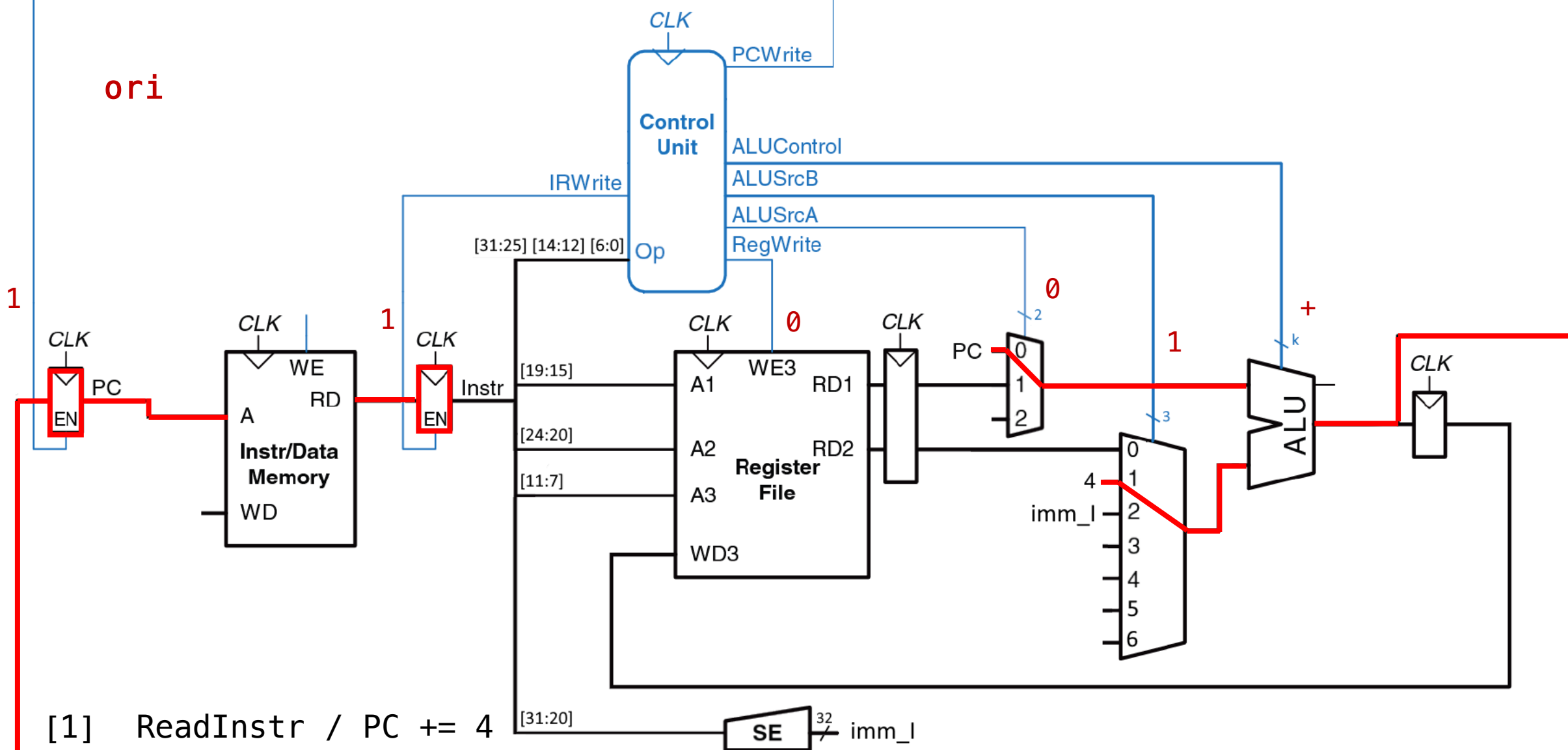


[1] ReadInstr / $PC = PC + 4$

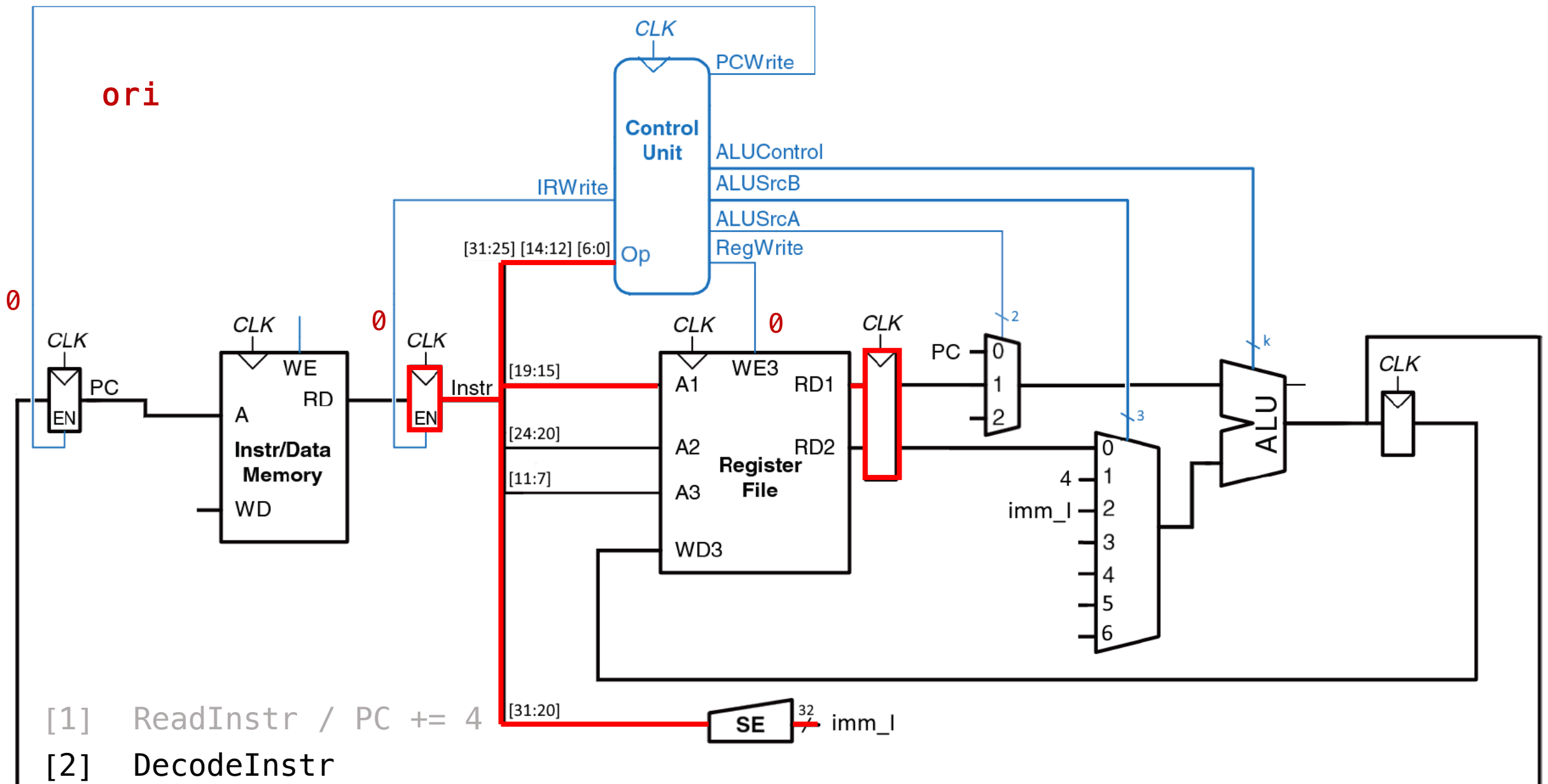




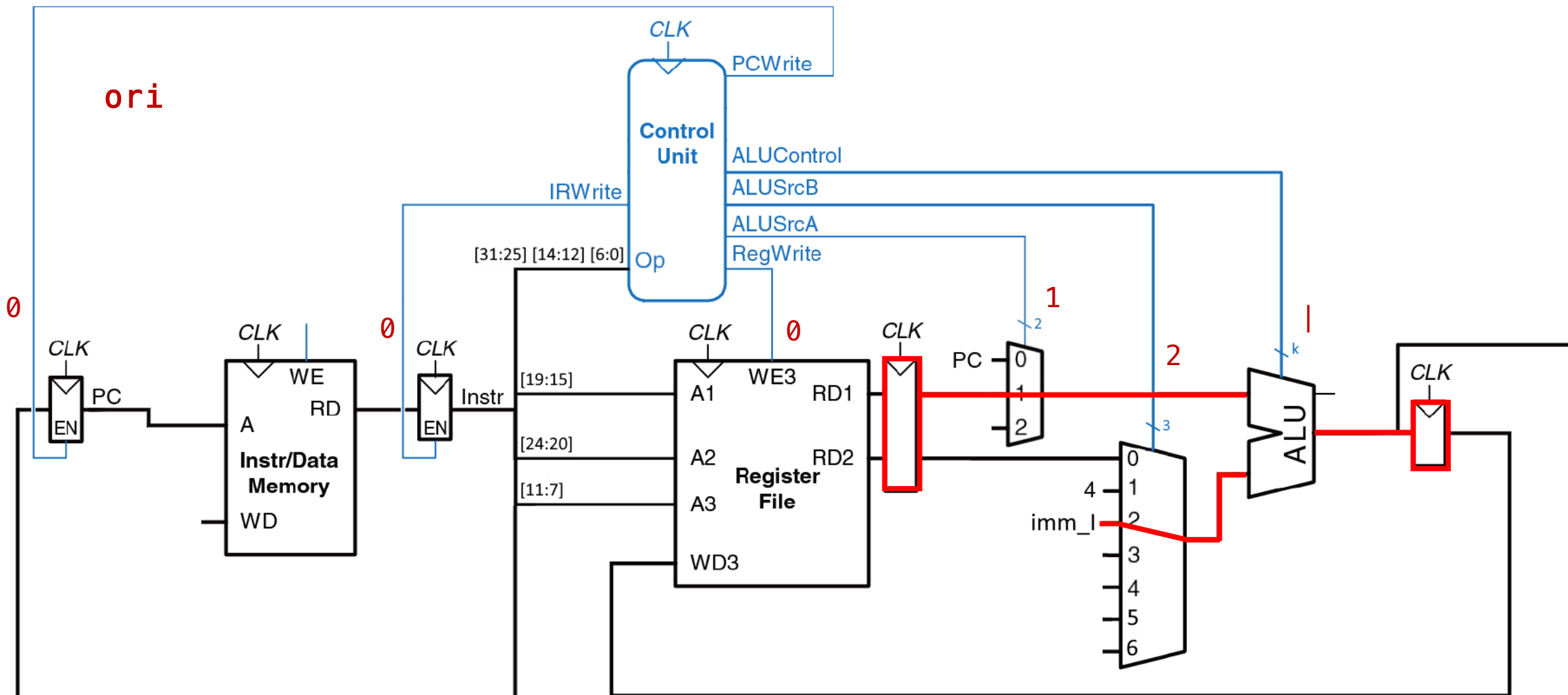
ori



ori

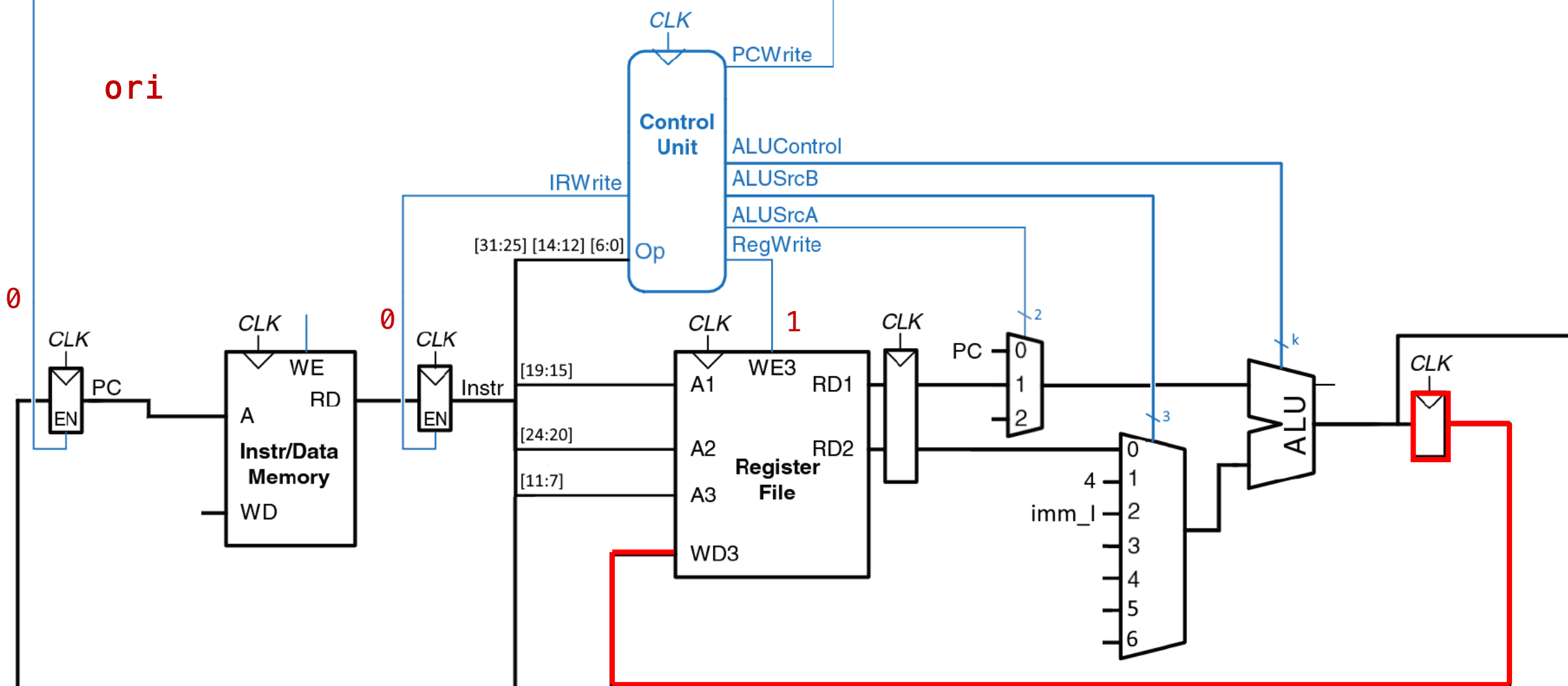


ori

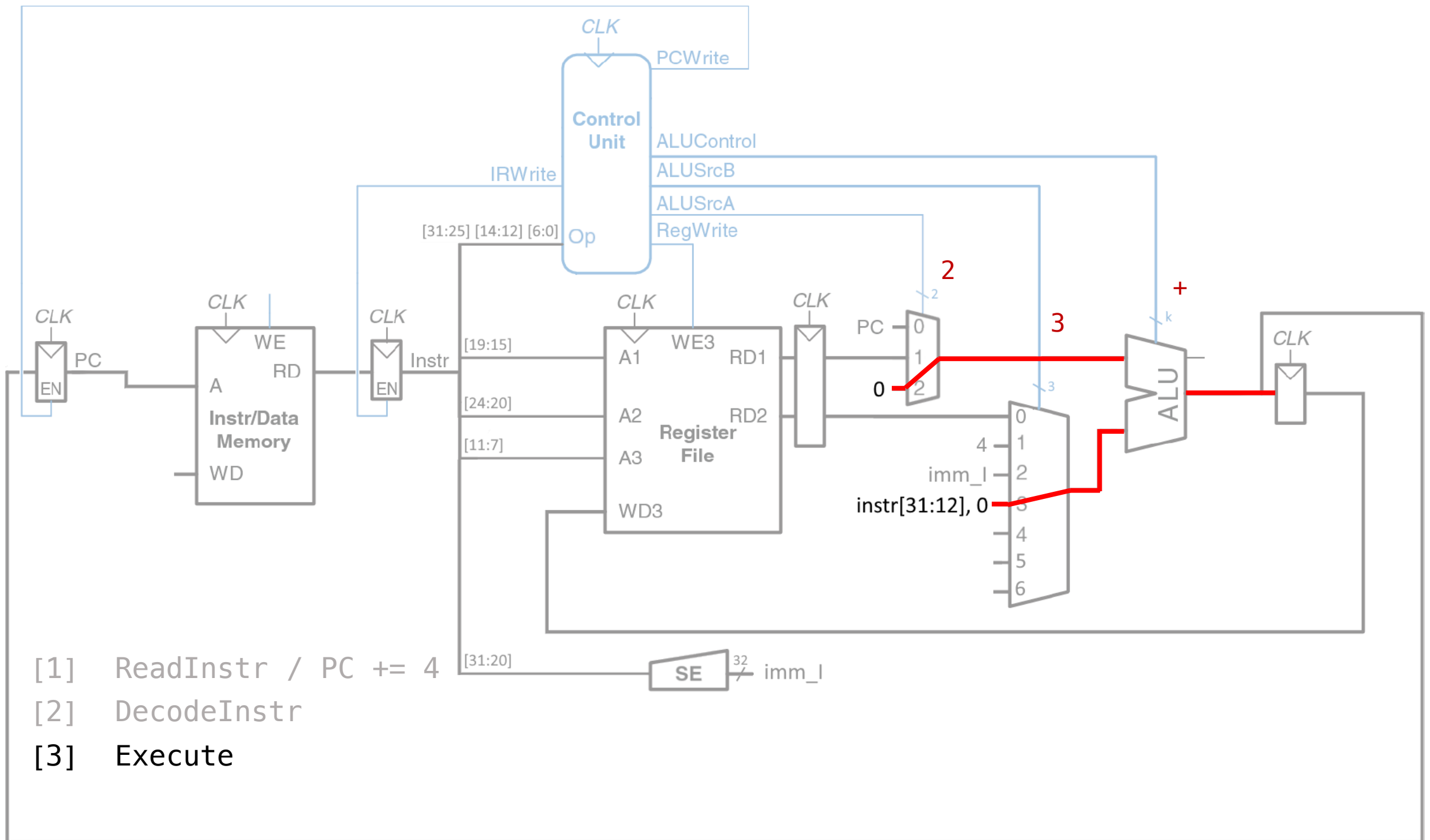


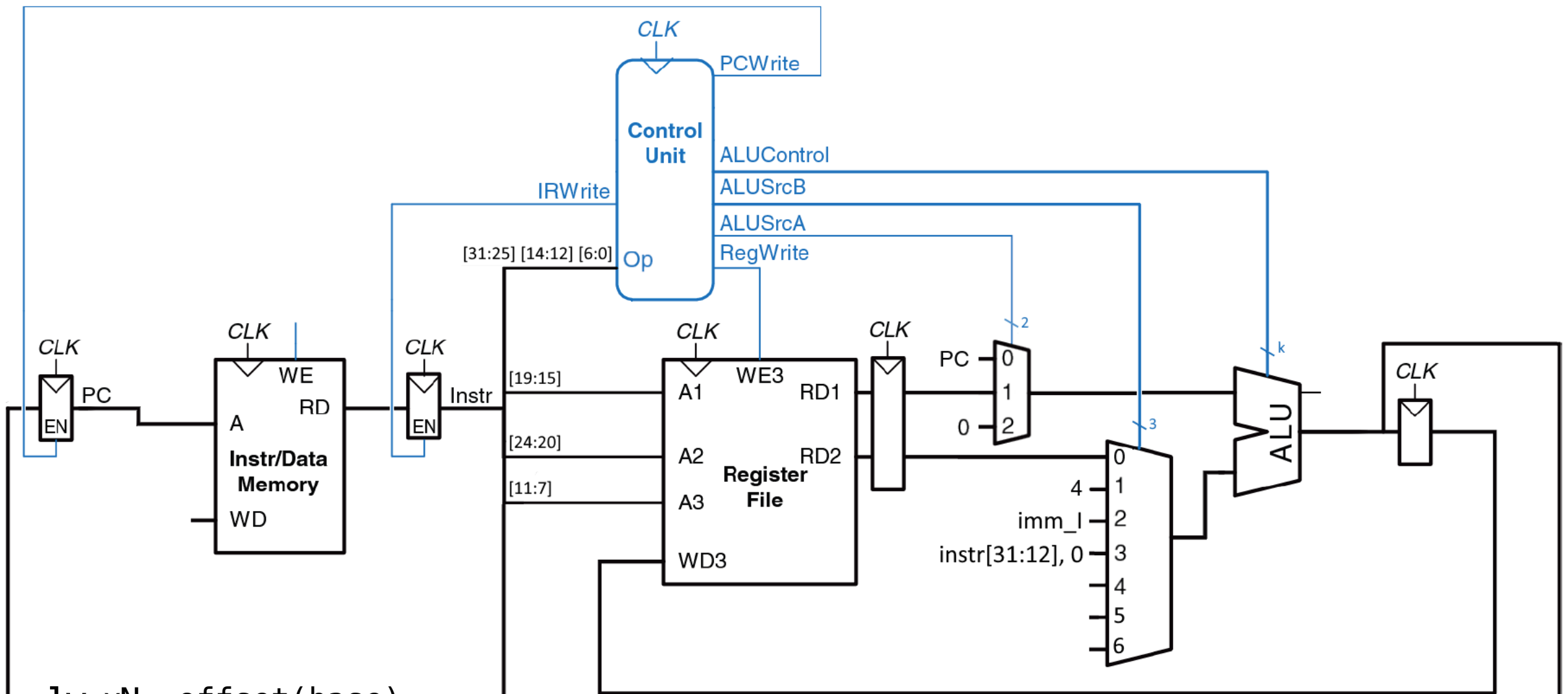
- [1] ReadInstr / PC += 4
- [2] DecodeInstr
- [3] Execute

ori



- [1] ReadInstr / PC += 4
- [2] DecodeInstr
- [3] Execute
- [4] Write to RF

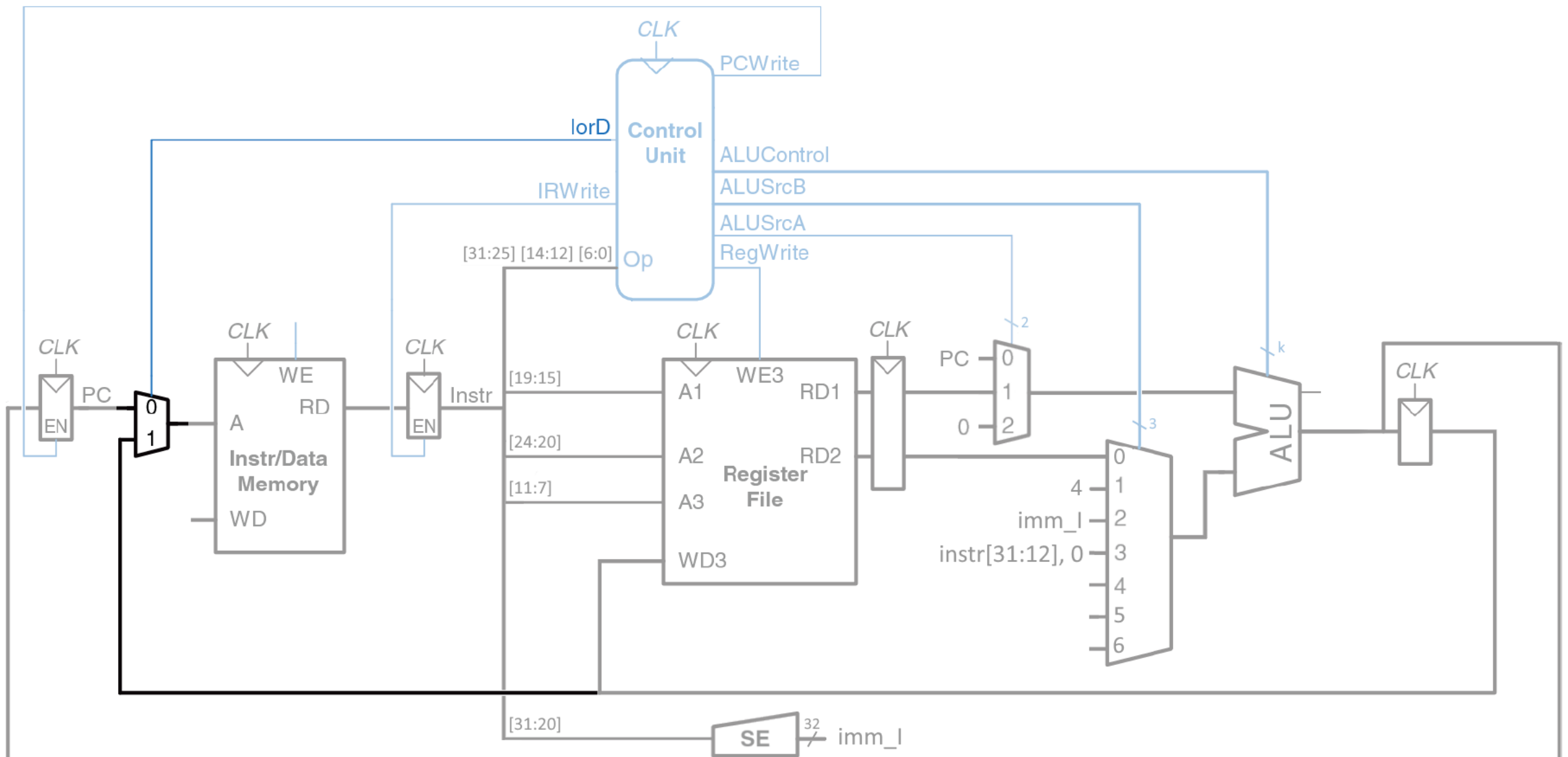




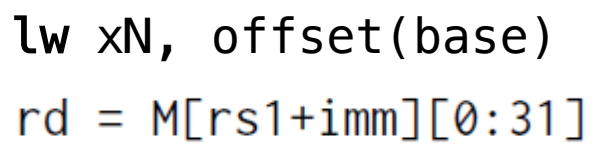
`lw xN, offset(base)`

`rd = M[rs1+imm][0:31]`

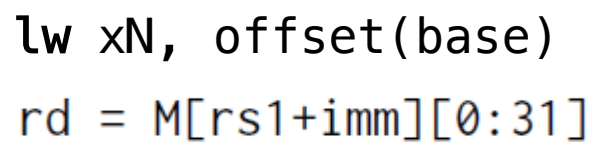
31	27	26	25	24	20	19	15	14	12	11	7	6	0	
imm[11:0]					rs1		funct3		rd		opcode			I-type
imm[11:0]					rs1		010		rd		0000011			LW

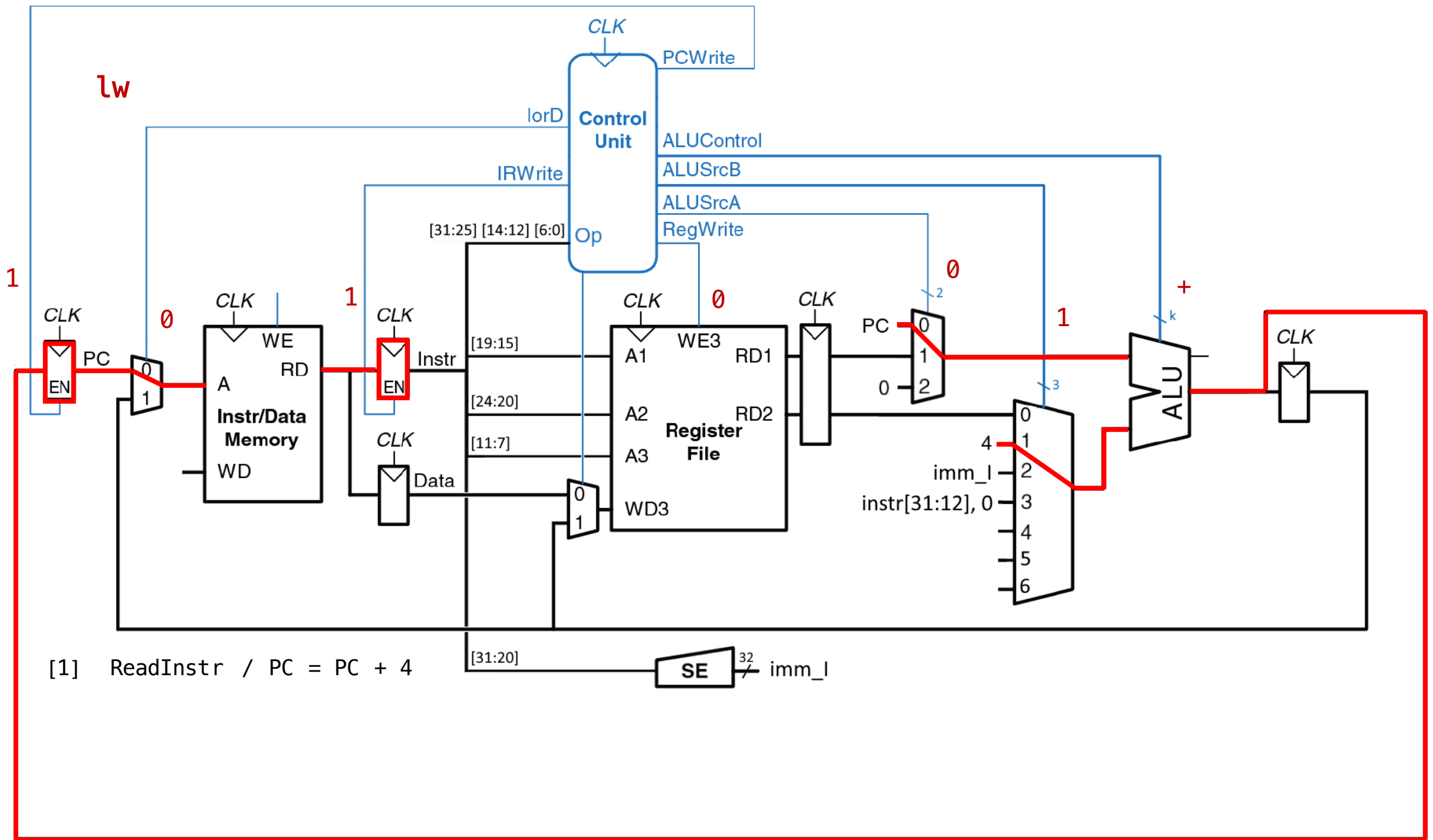


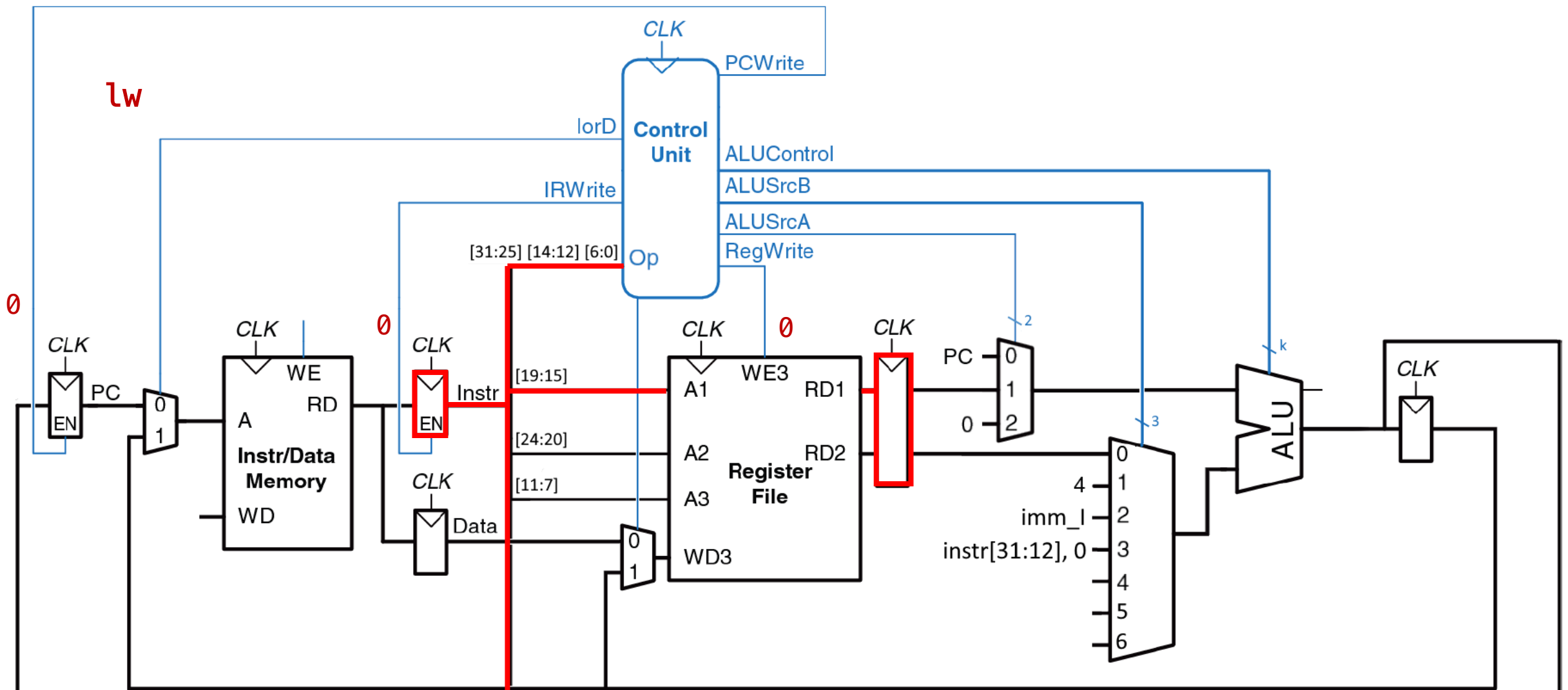
```
lw xN, offset(base)
rd = M[rs1+imm][0:31]
```



```
rd = M[rs1+imm][0:31]
```

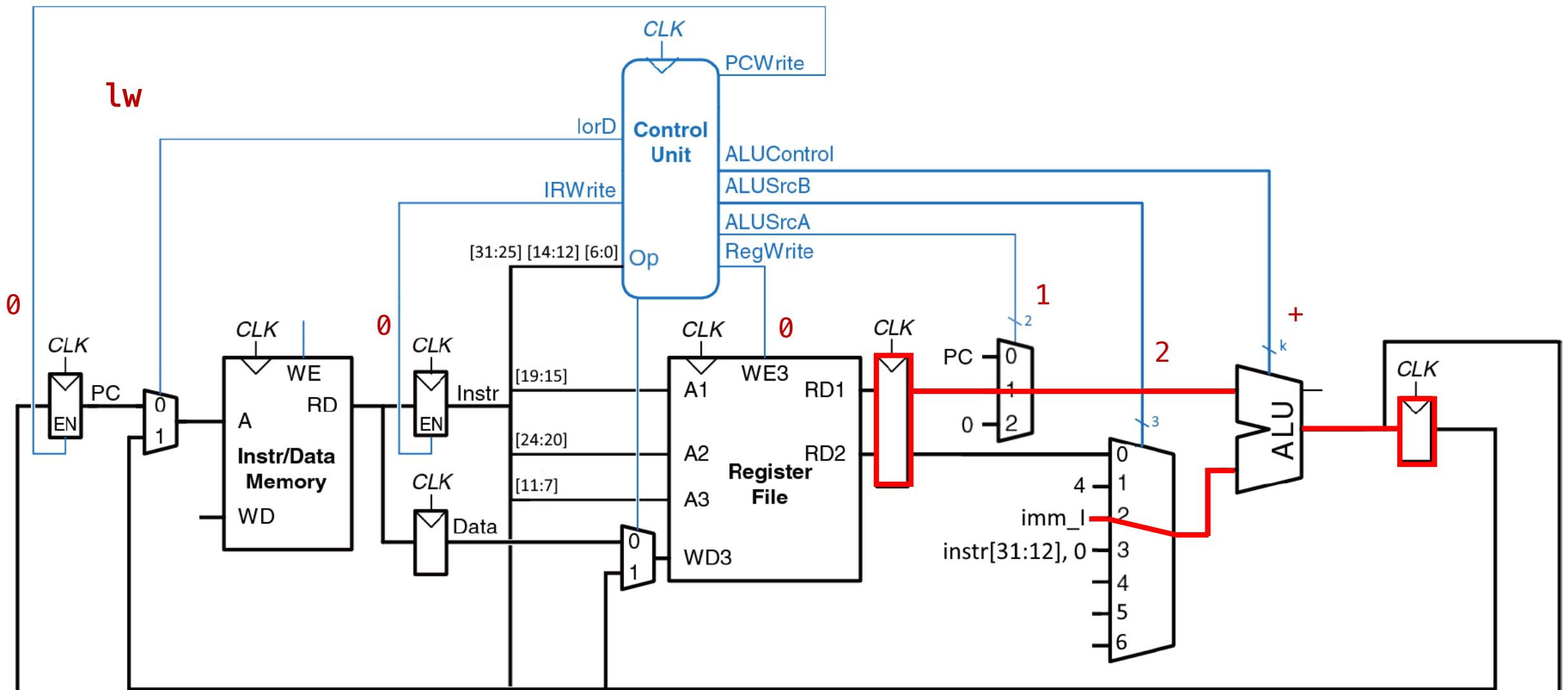






[1] ReadInstr / PC = PC + 4

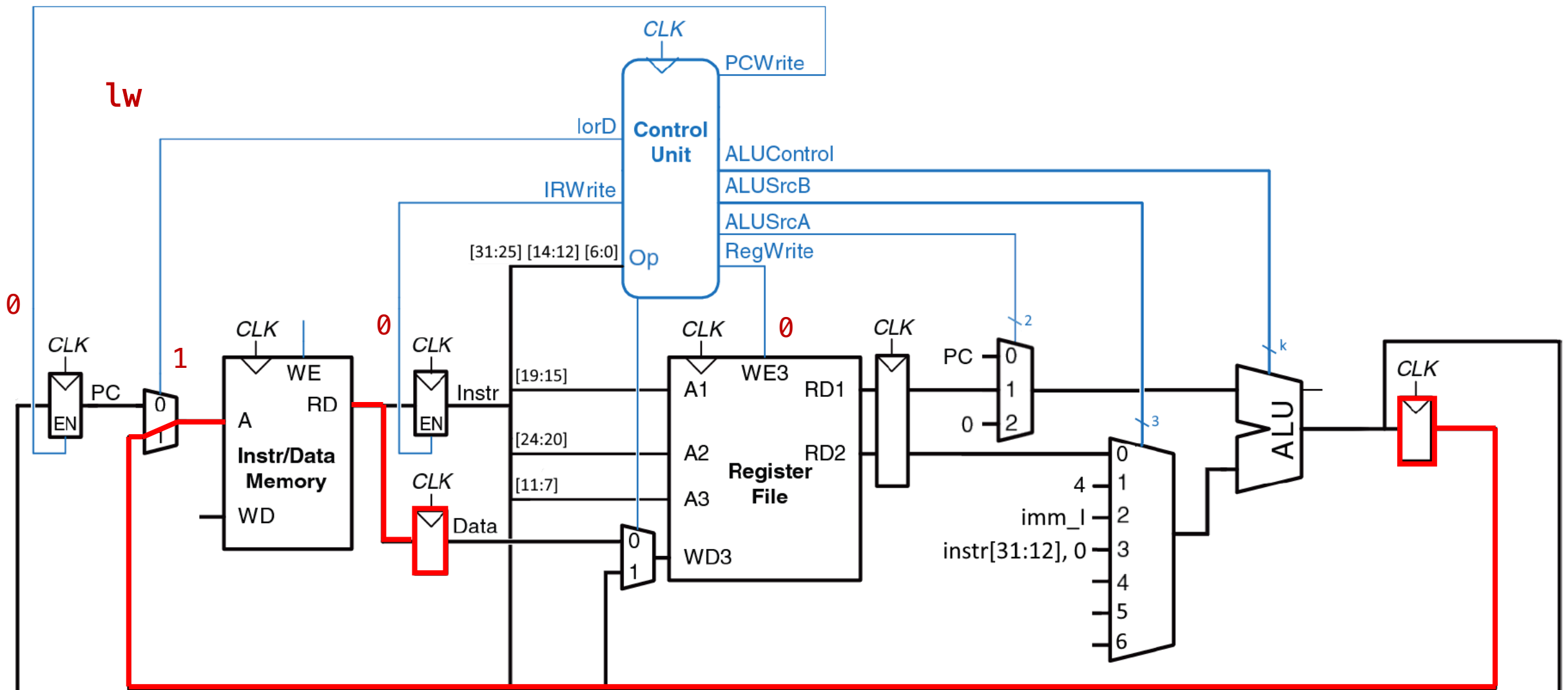
[2] DecodeInstr



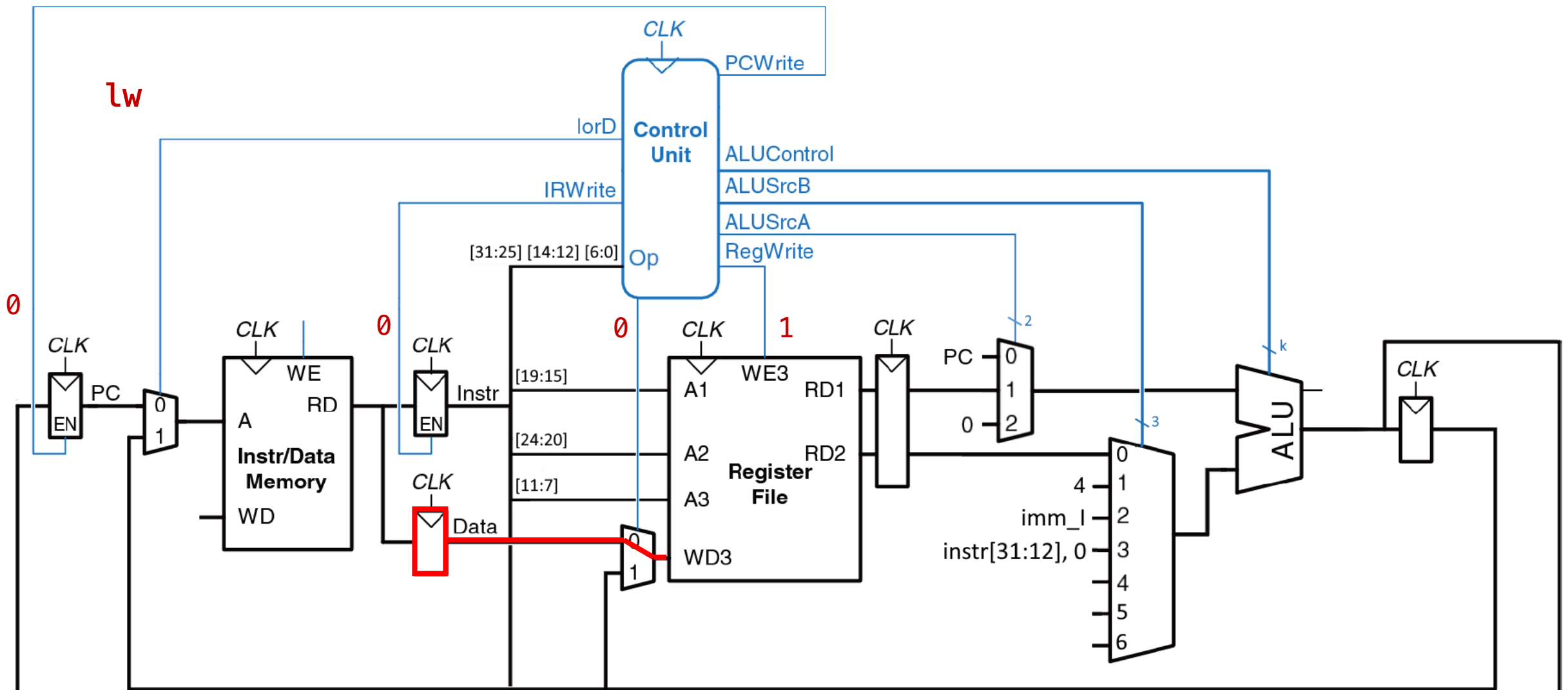
[1] ReadInstr / $PC = PC + 4$

[2] DecodeInstr

[3] Execute



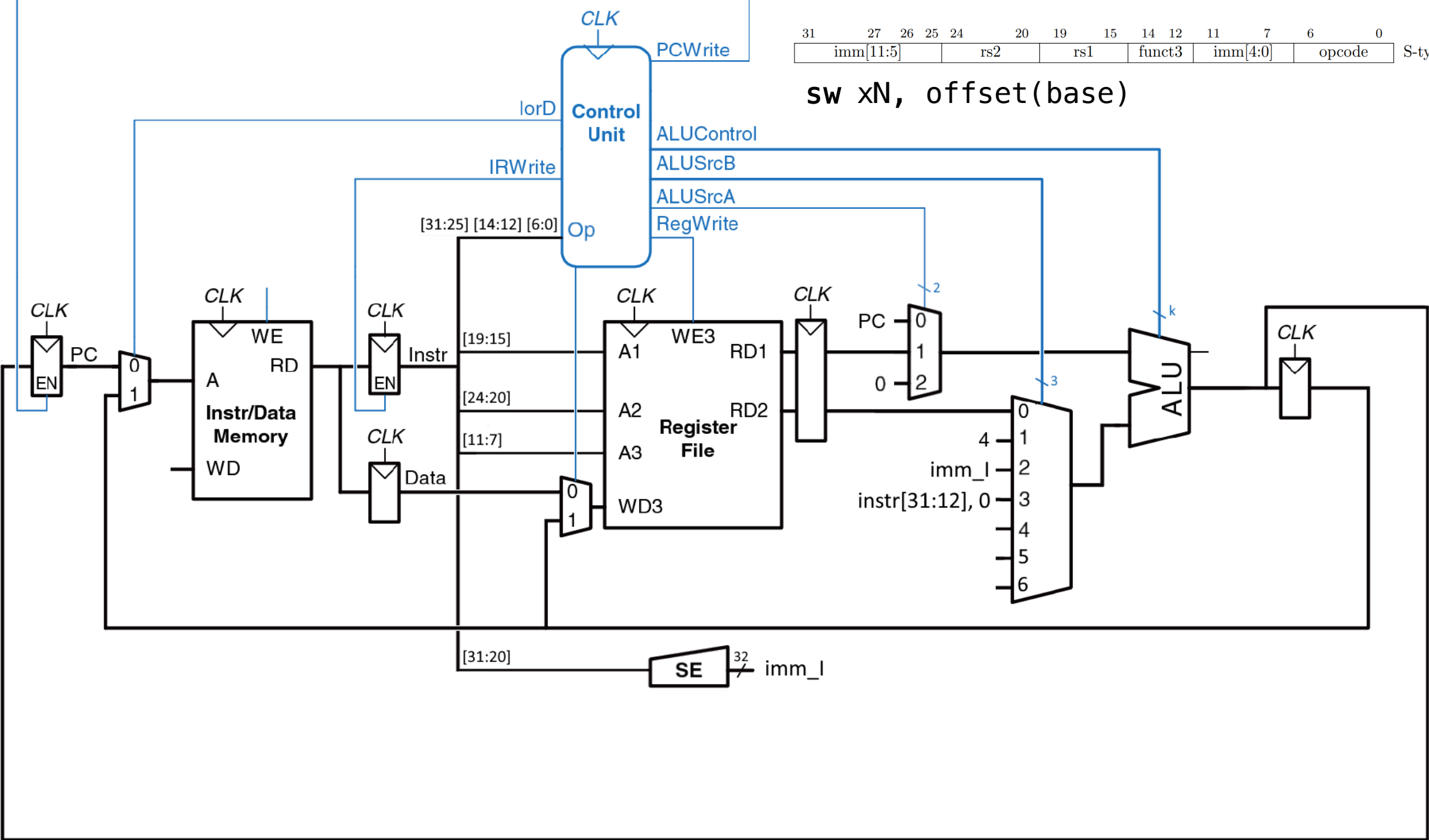
- [1] ReadInstr / $PC = PC + 4$
- [2] DecodeInstr
- [3] Execute
- [4] ReadMem

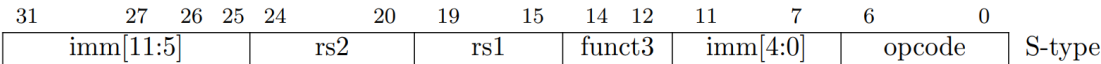


- [1] ReadInstr / $PC = PC + 4$
- [2] DecodeInstr
- [3] Execute
- [4] ReadMem
- [5] Write to RF

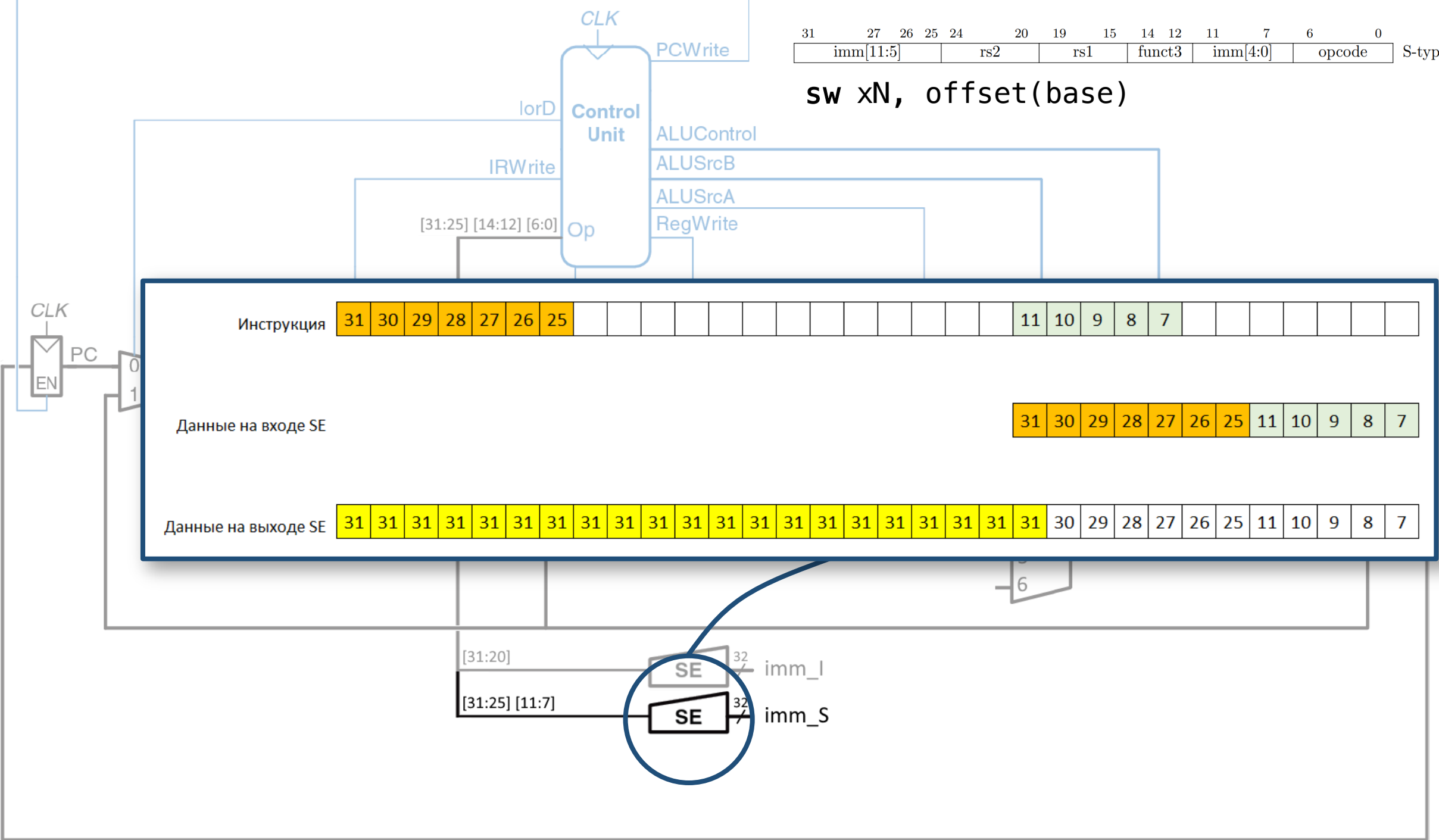
31	27	26	25	24	20	19	15	14	12	11	7	6	0	
imm[11:5]				rs2		rs1		funct3		imm[4:0]		opcode		S-type

sw xN, offset(base)



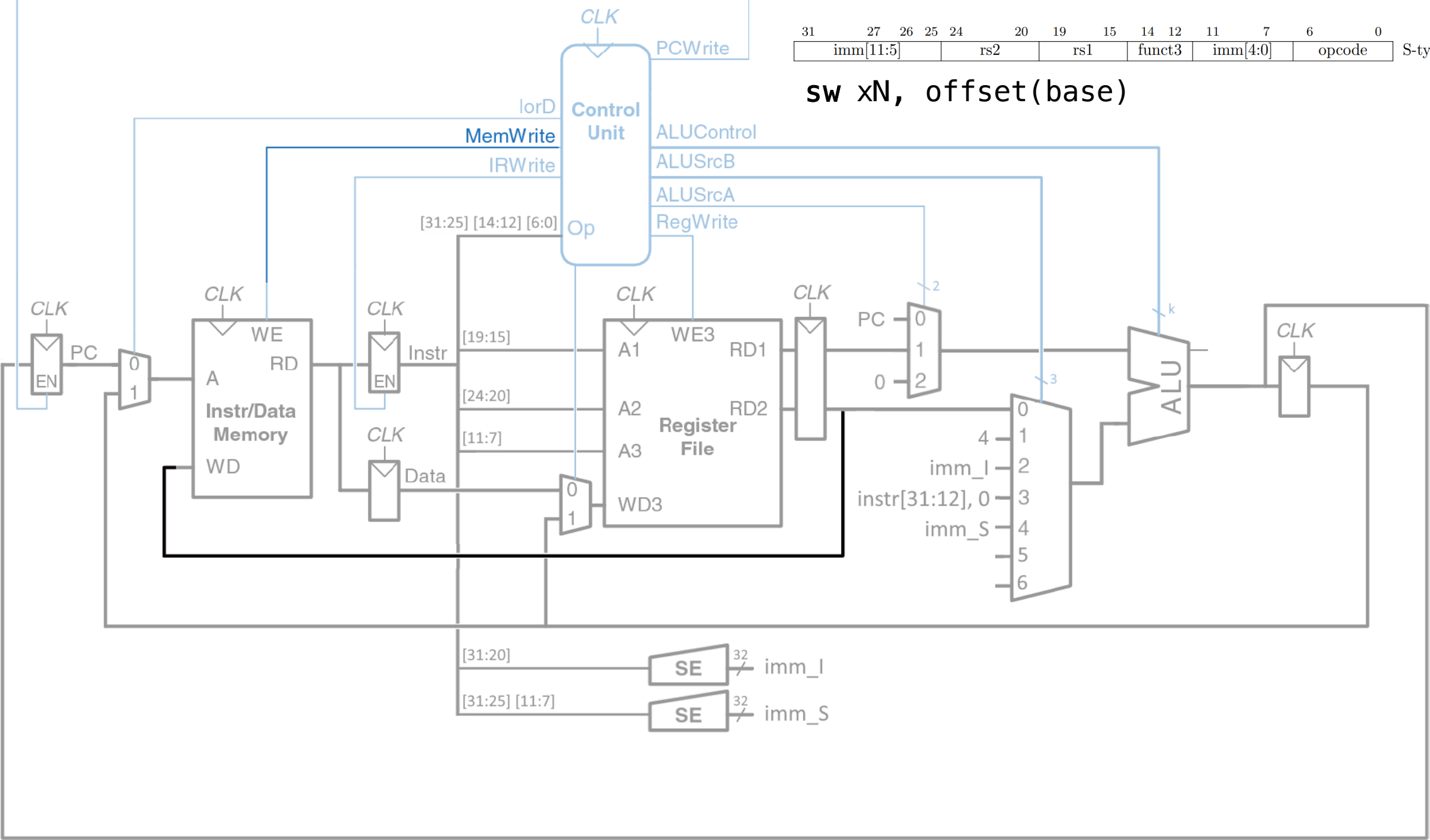


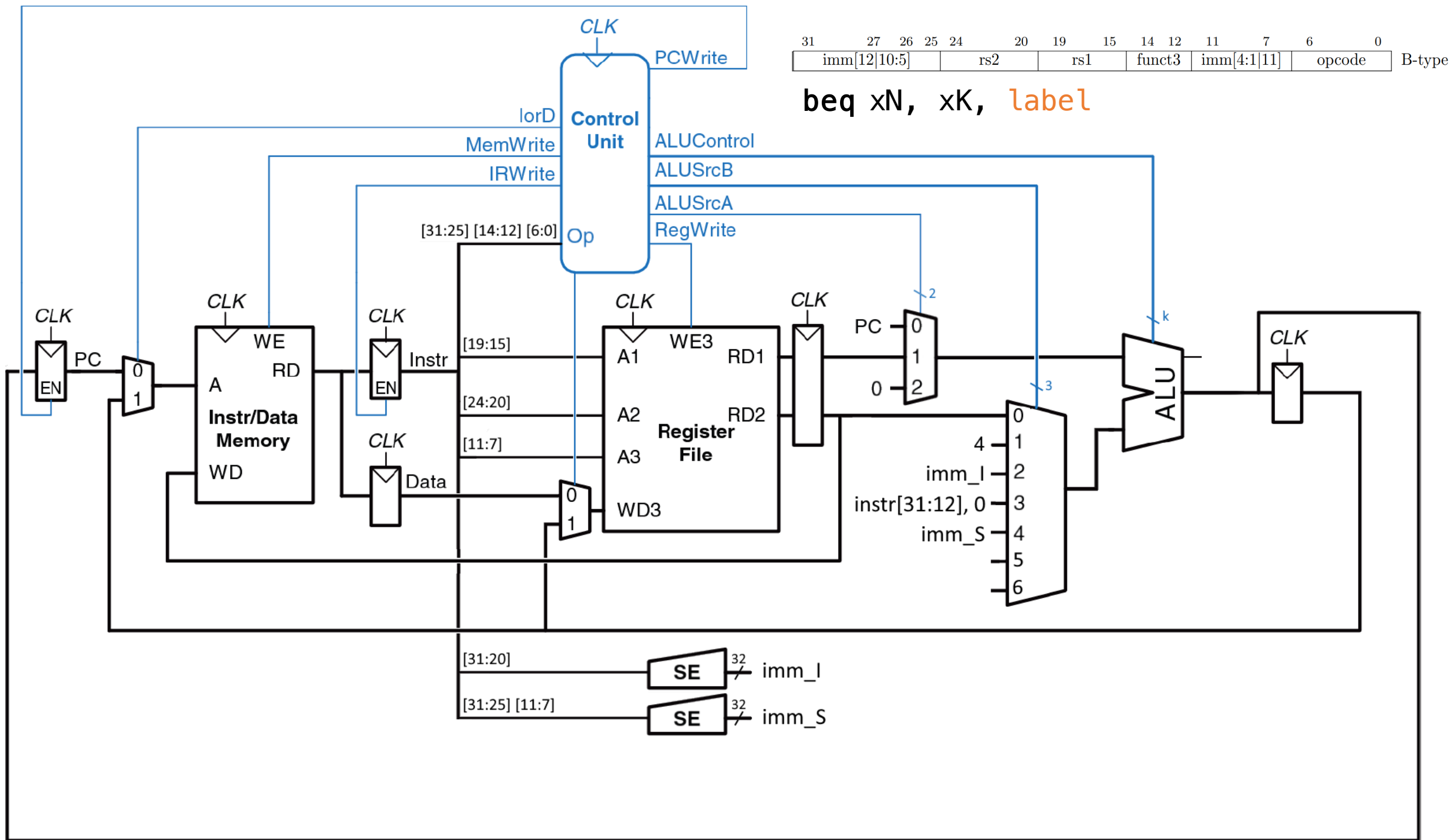
sw xN, offset(base)

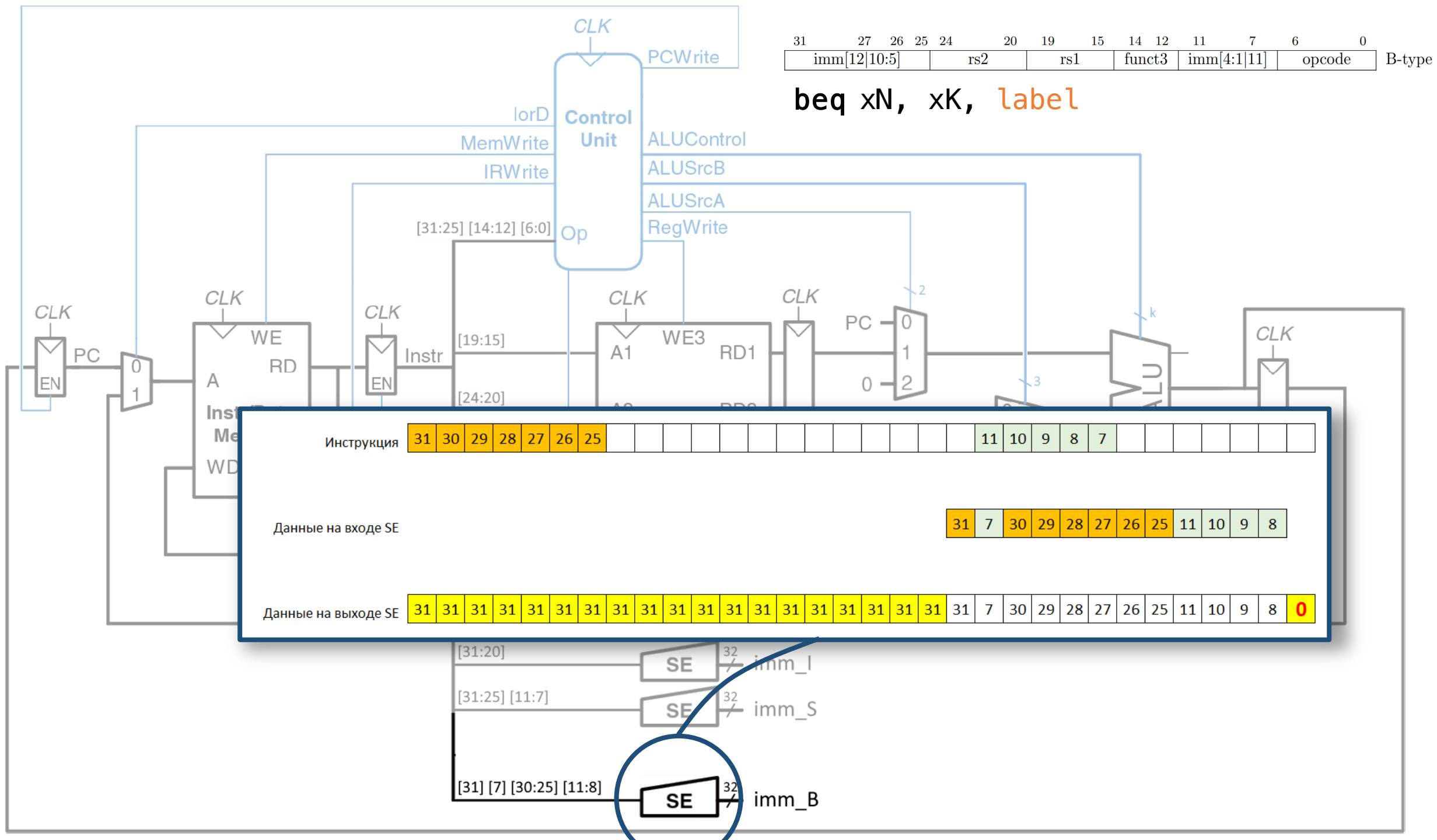


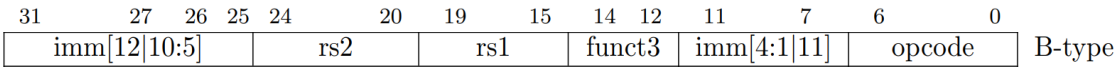
31	27	26	25	24	20	19	15	14	12	11	7	6	0		
imm[11:5]					rs2		rs1		funct3		imm[4:0]		opcode		S-type

sw xN, offset(base)

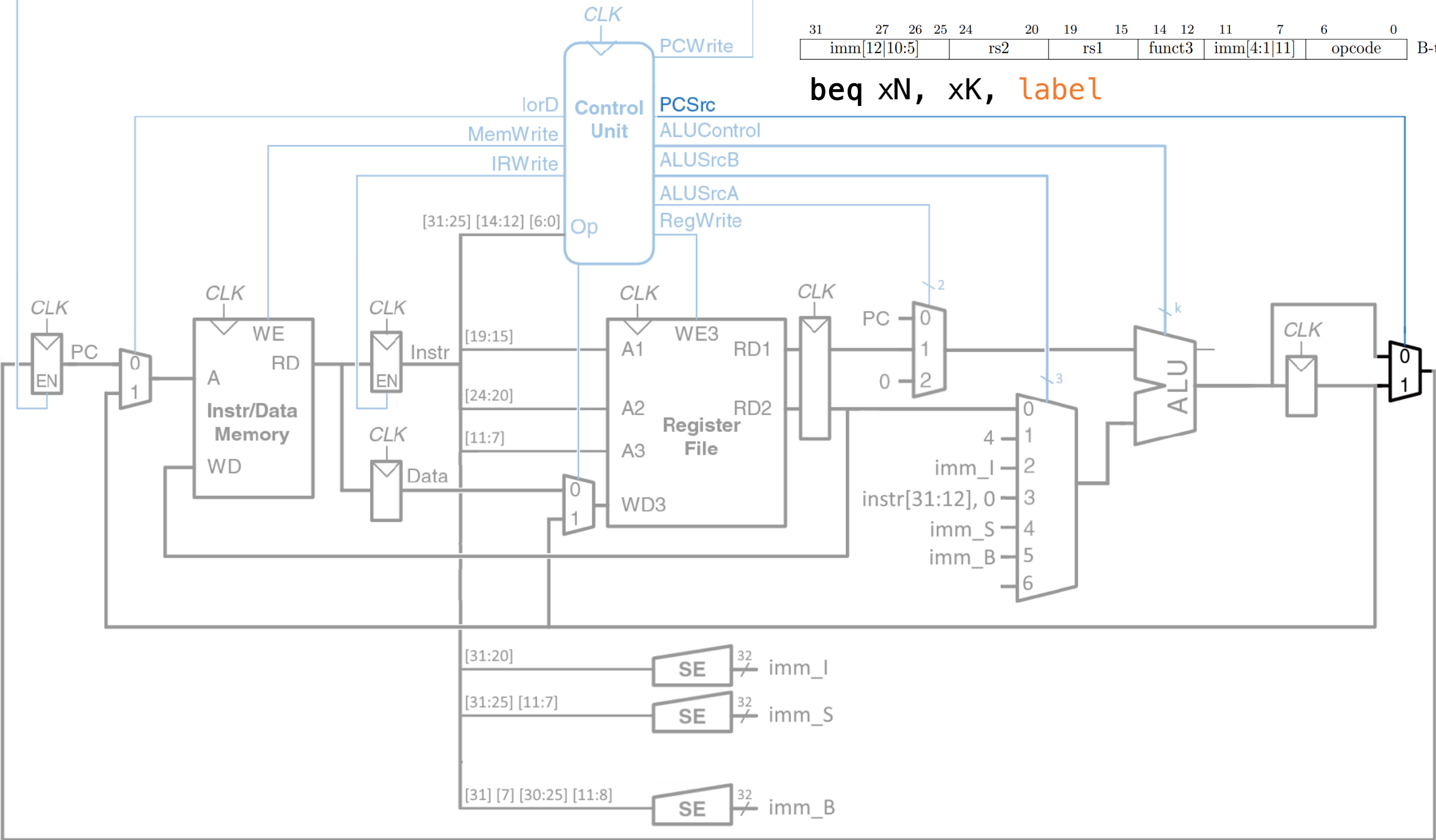


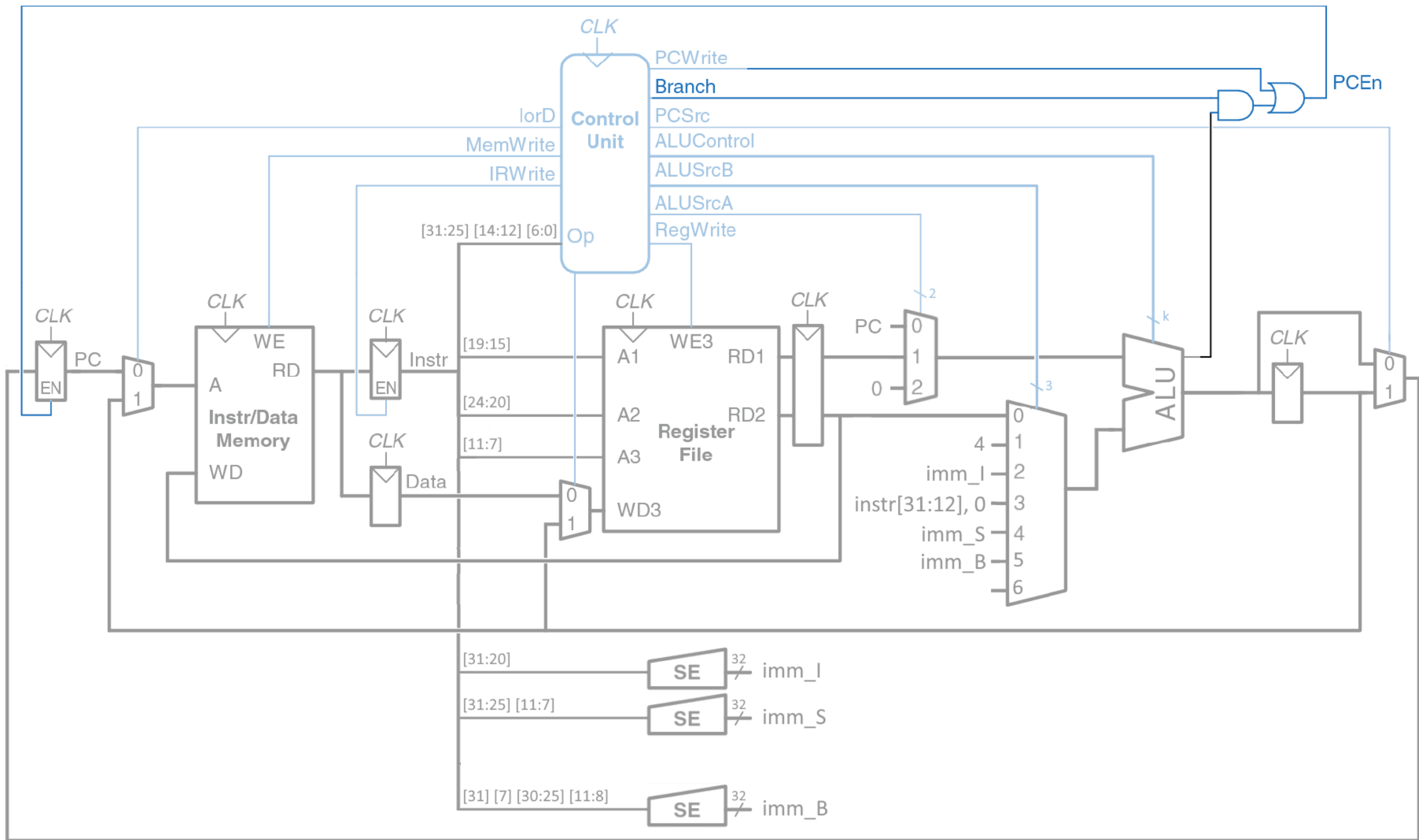


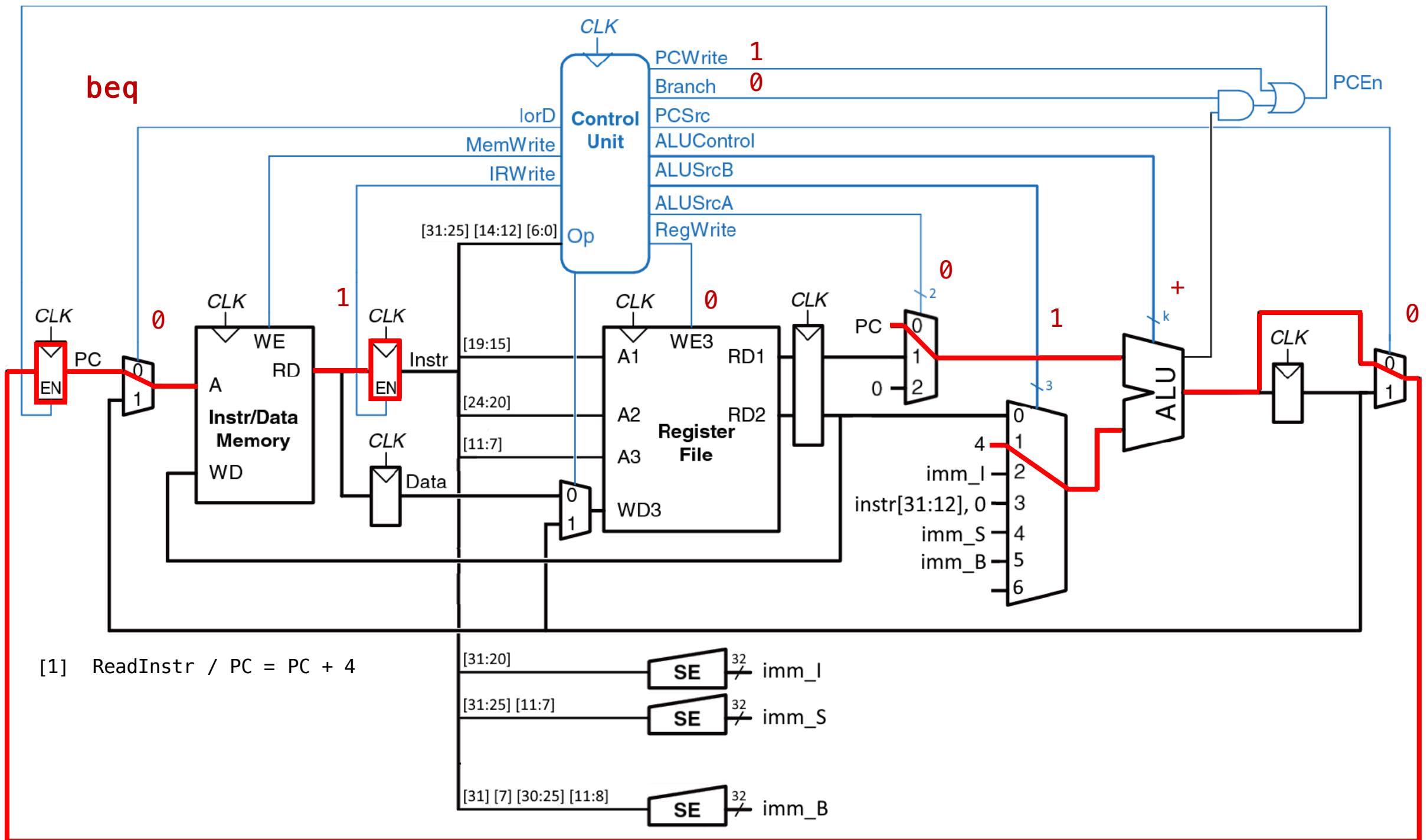


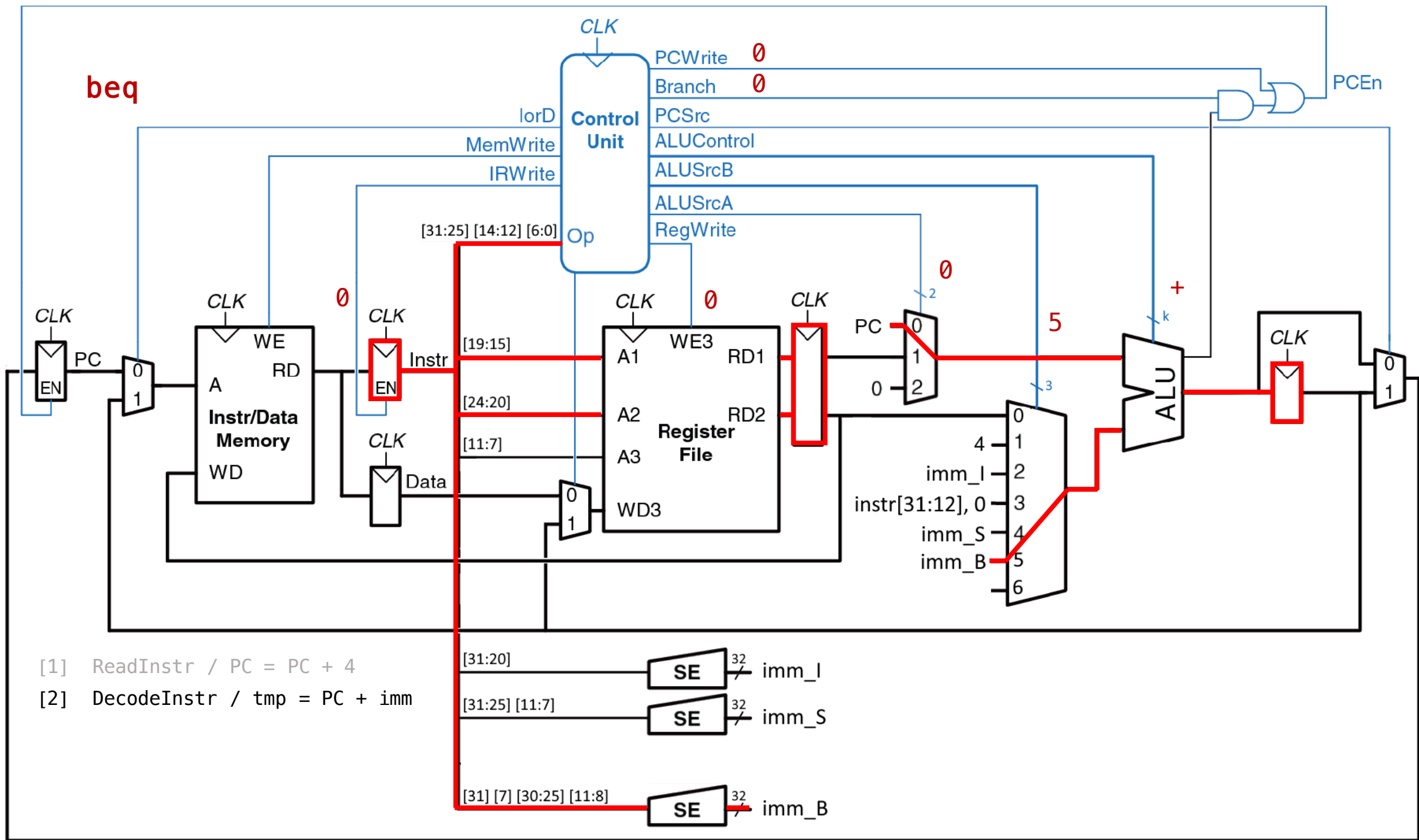


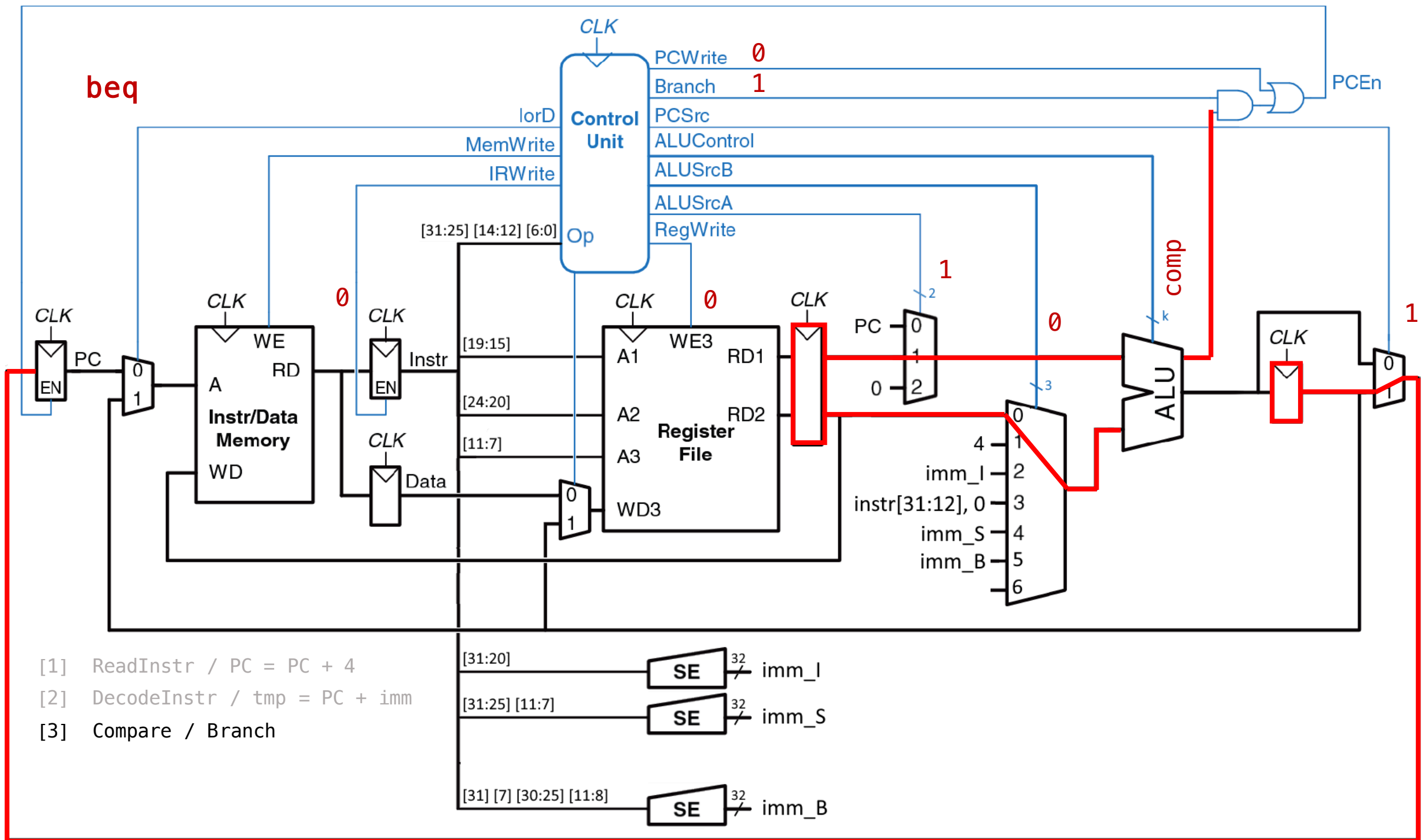
beq xN, xK, label



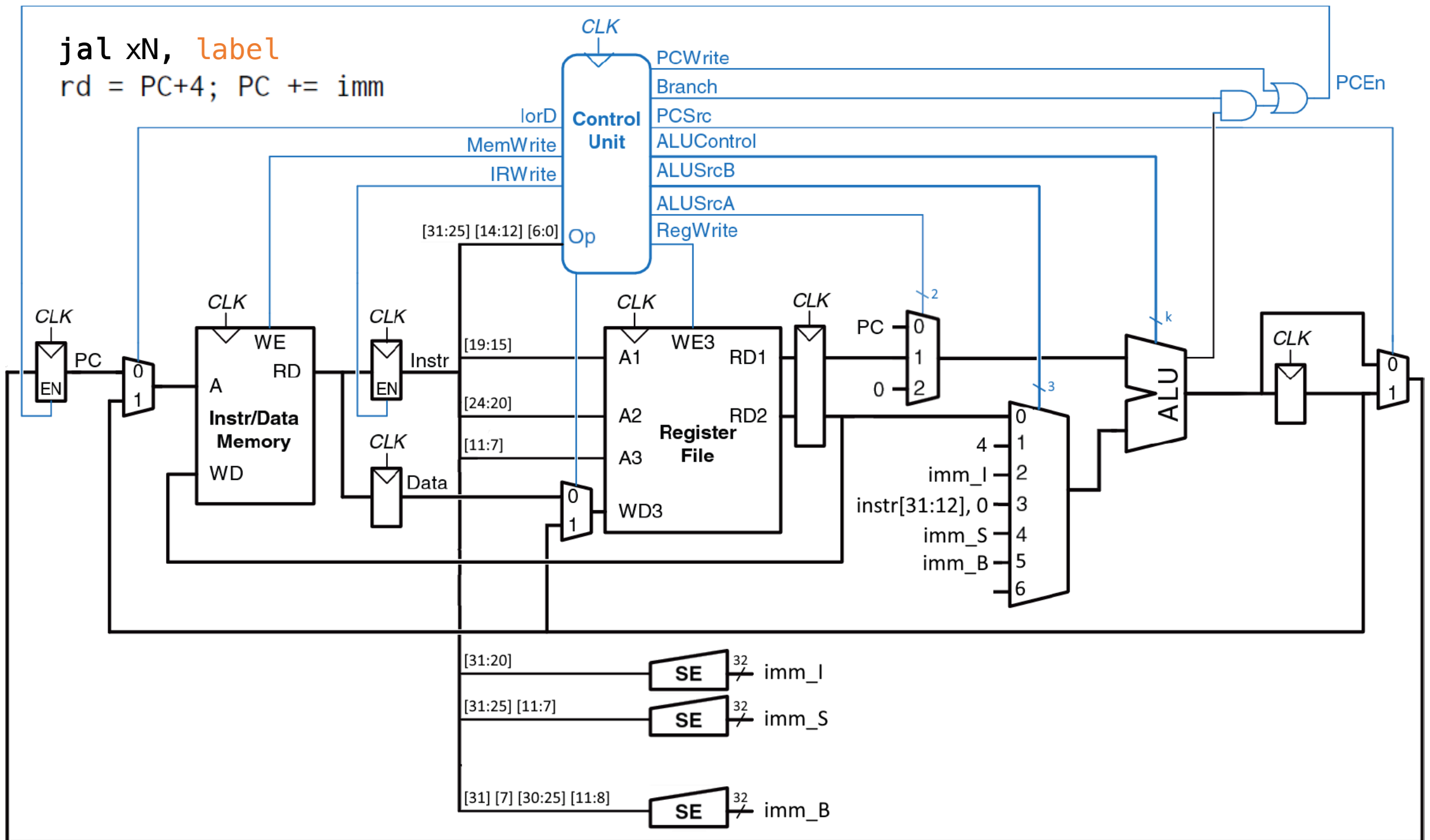




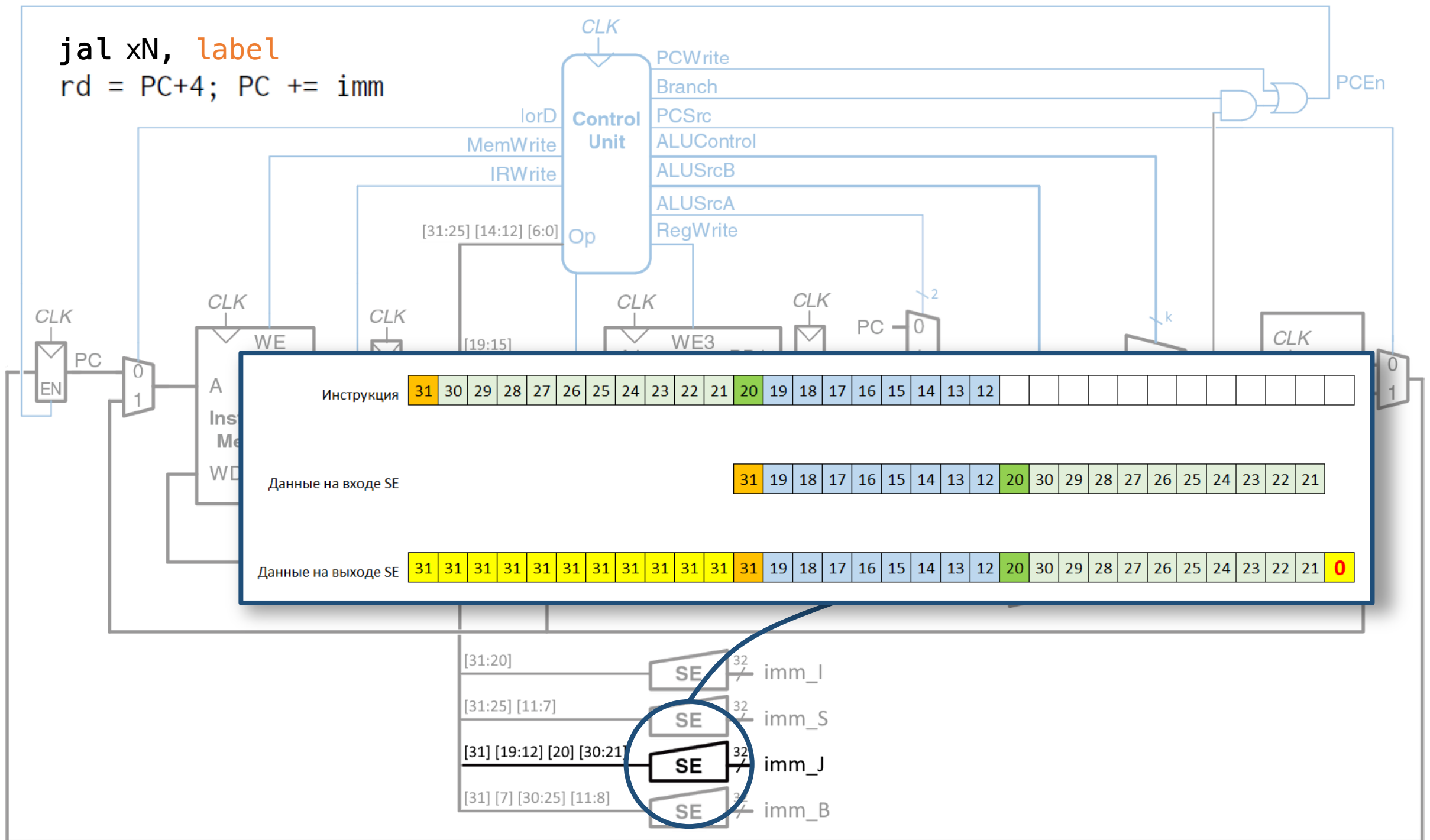


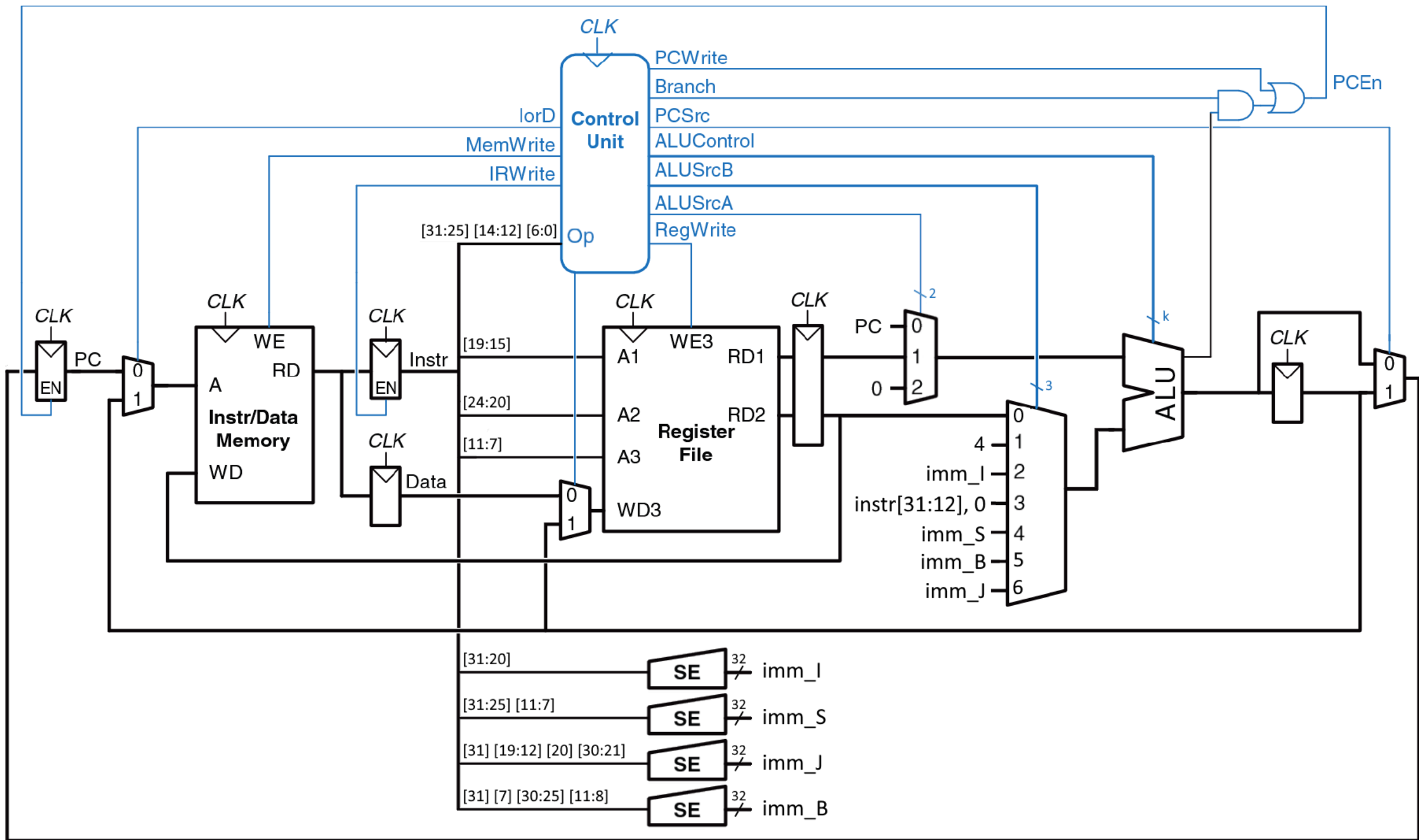


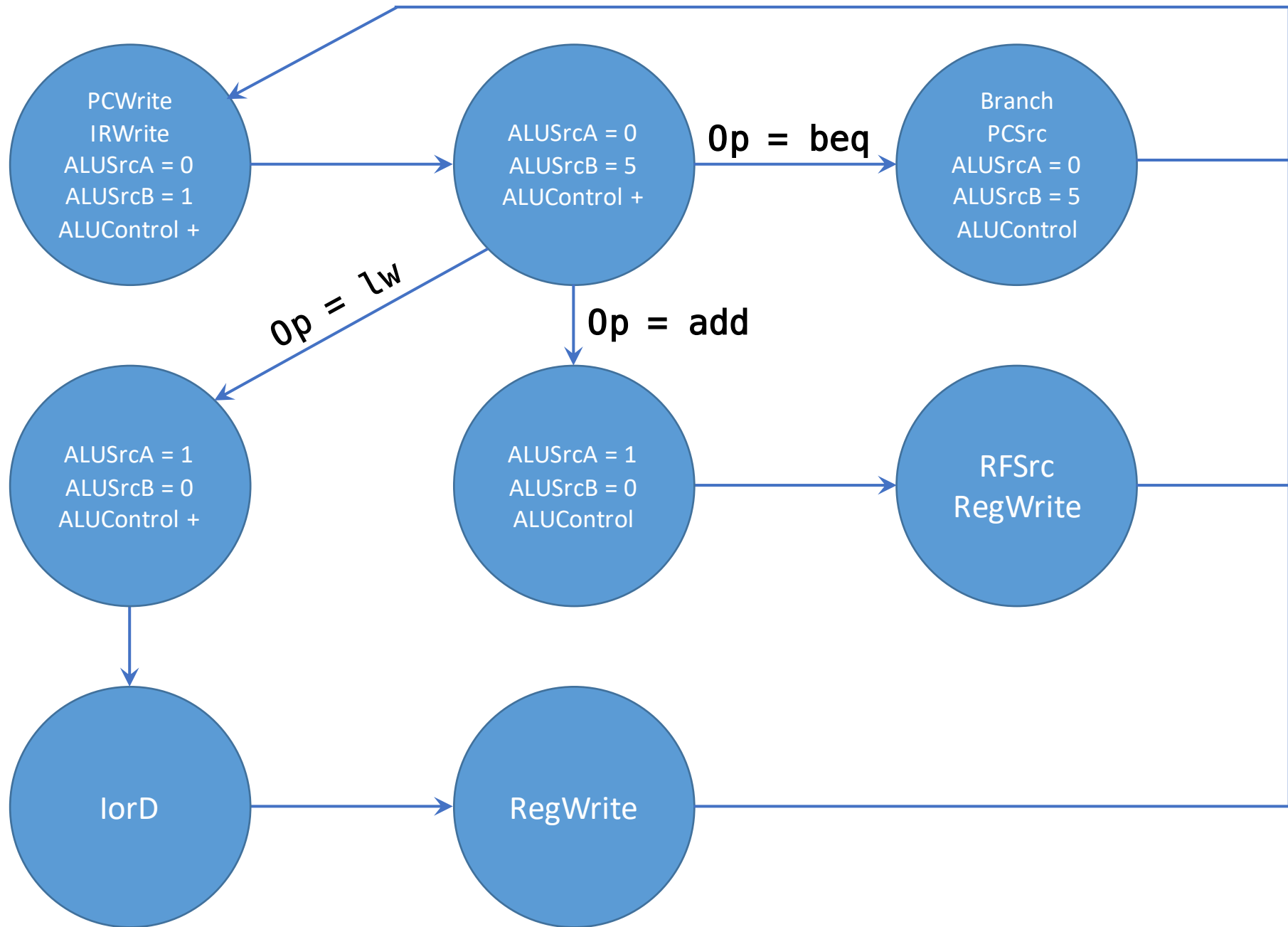
```
jal xN, label
rd = PC+4; PC += imm
```

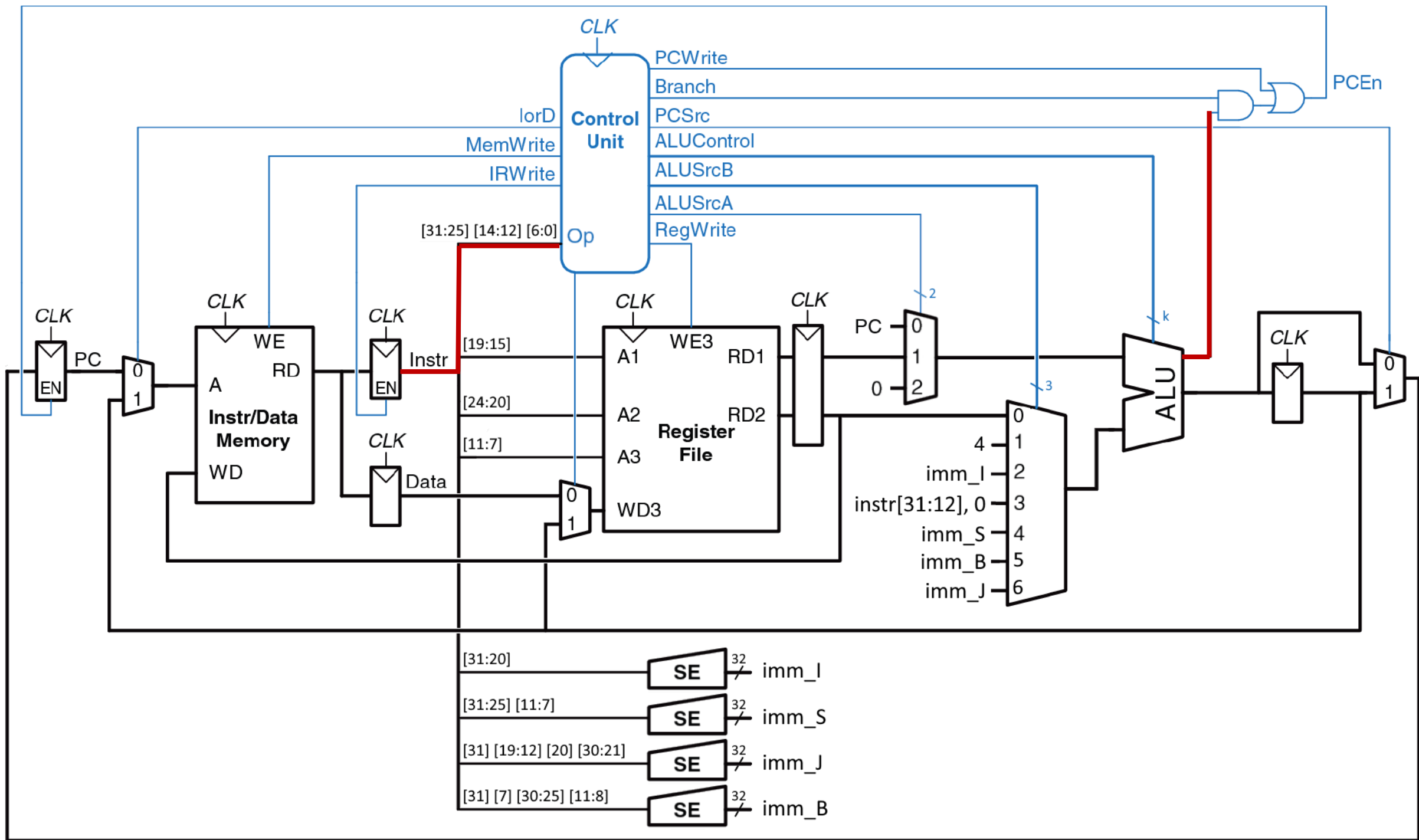


jal xN, label
 rd = PC+4; PC += imm

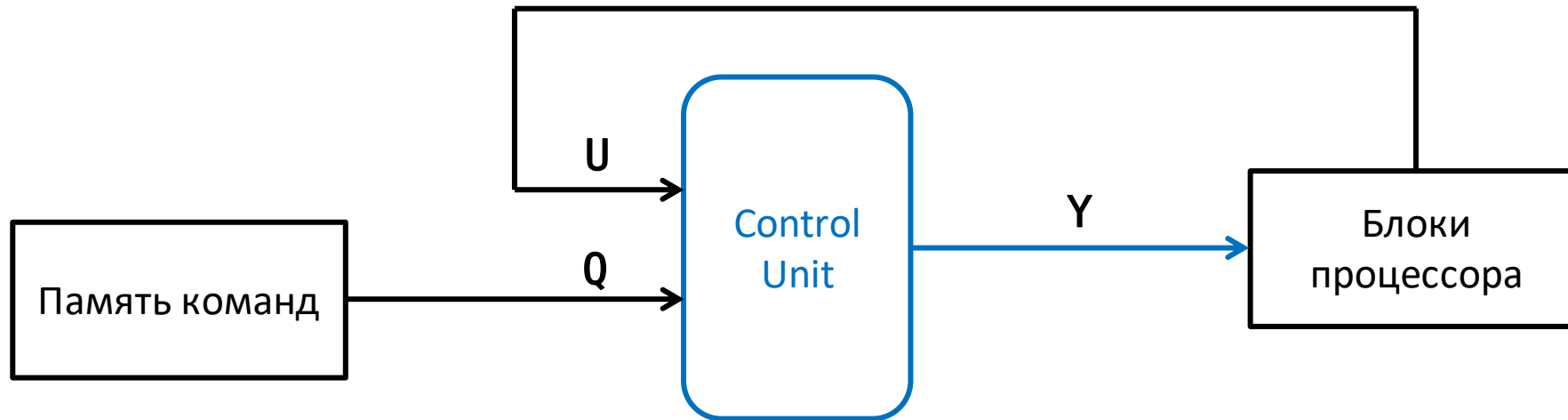




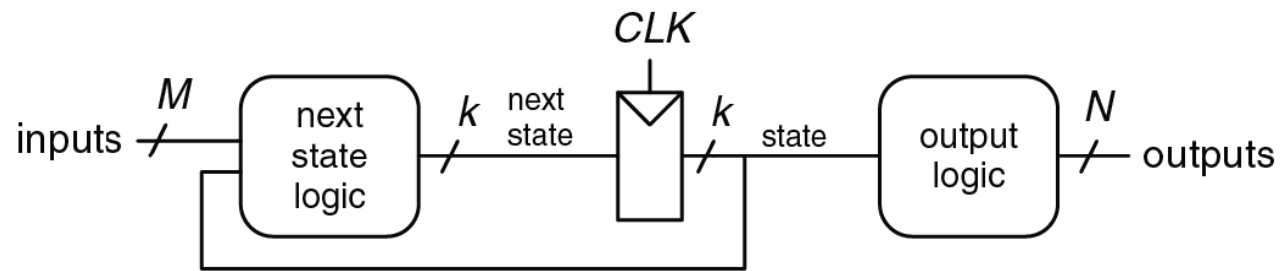




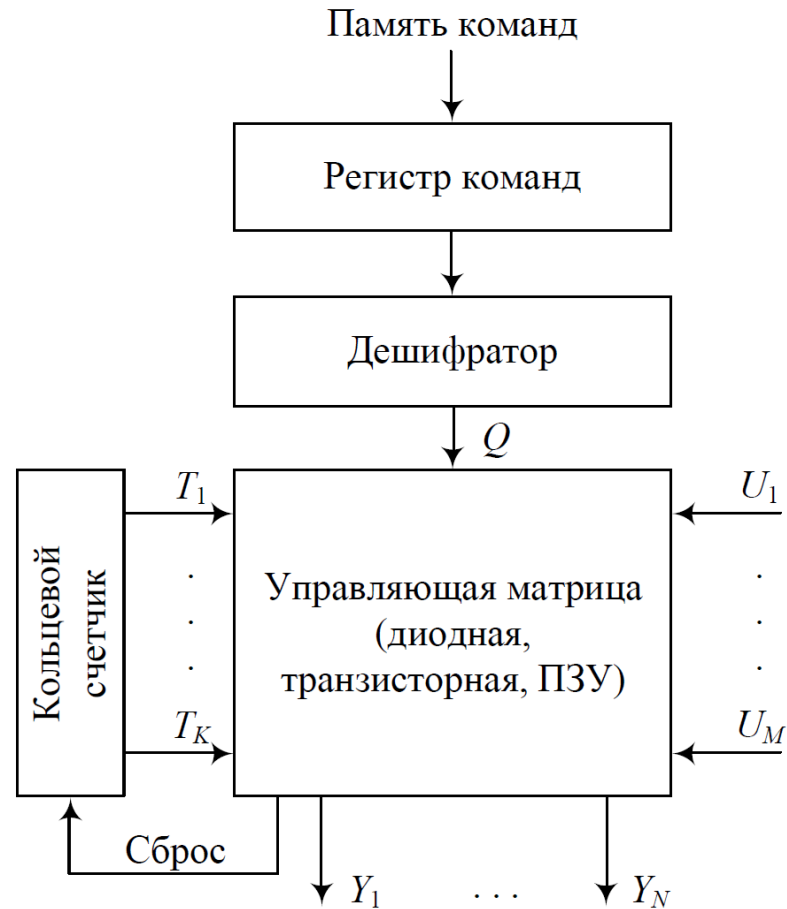
Обобщенная структура



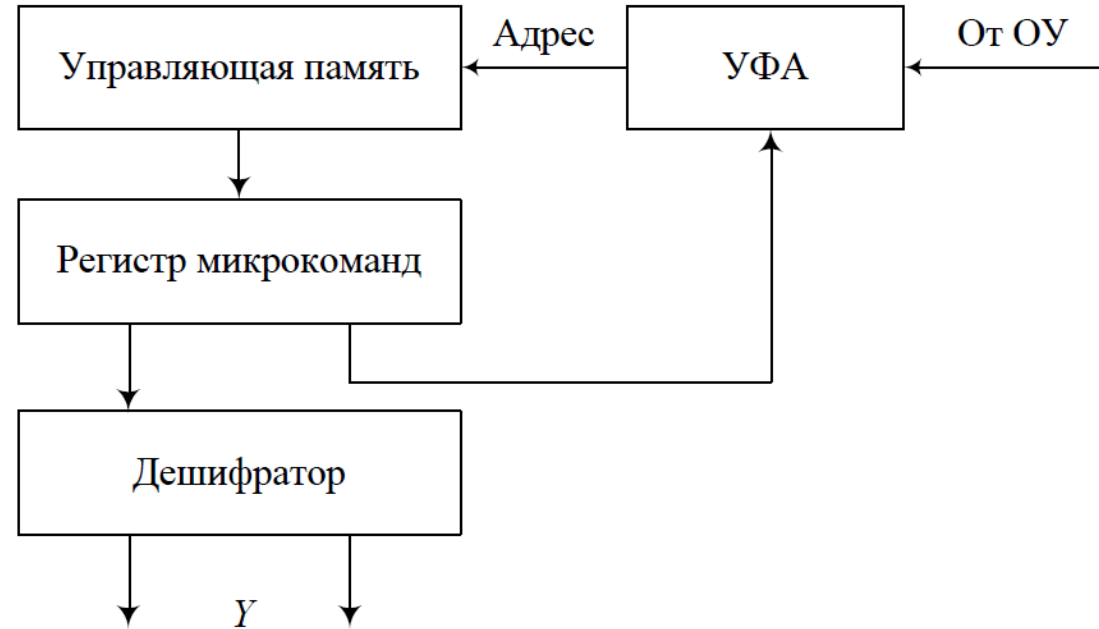
Конечный автомат



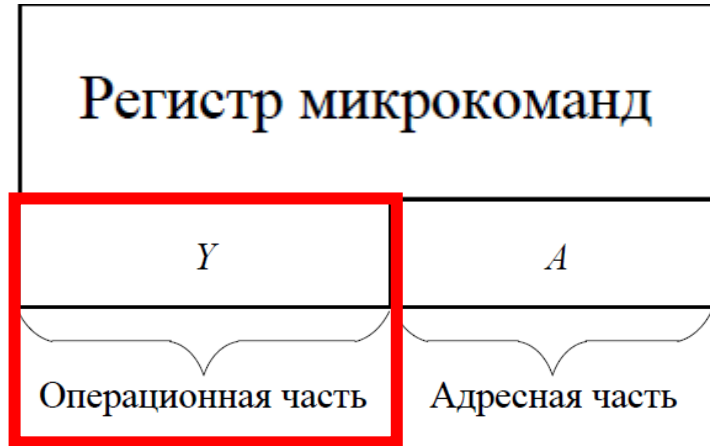
УУ с жесткой структурой



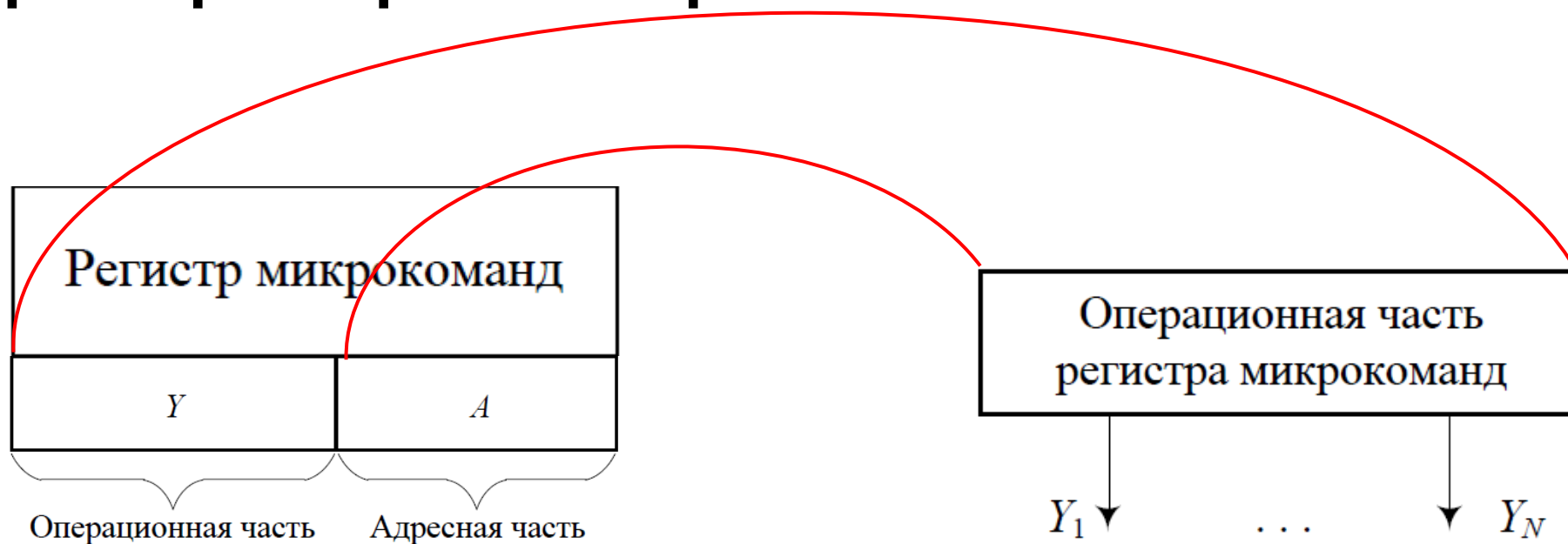
Устройство с микропрограммным управлением



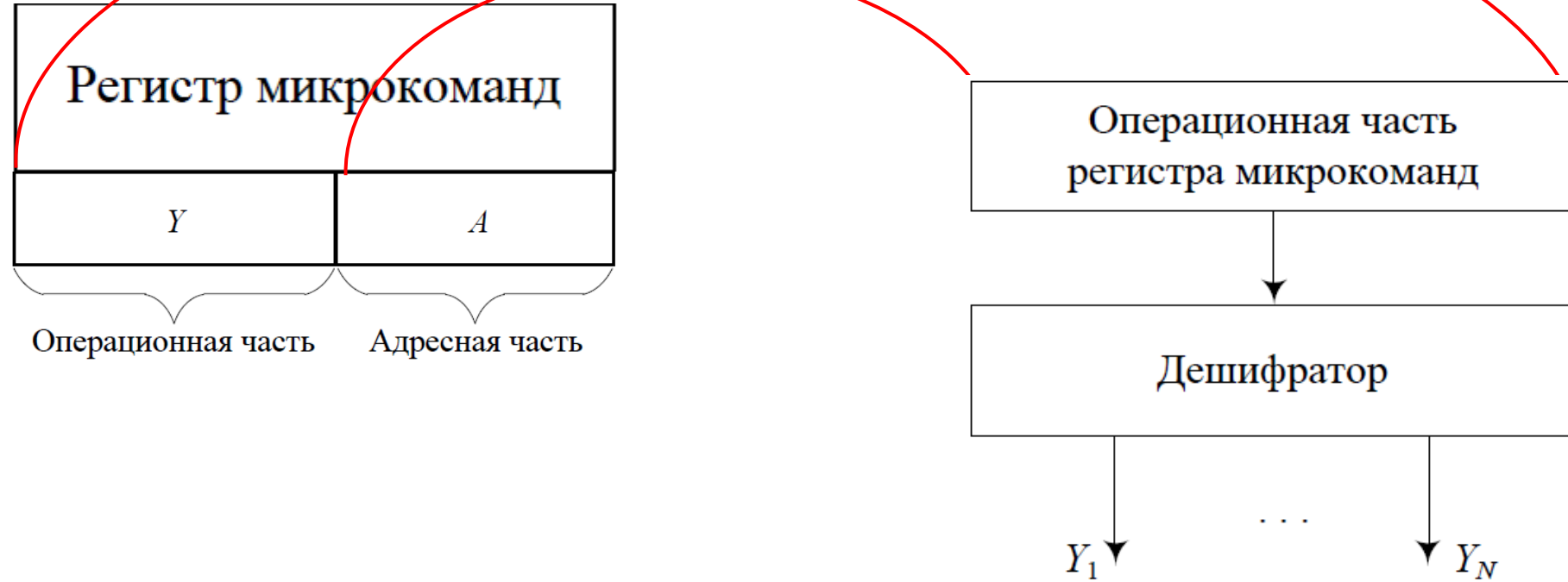
Устройство с микропрограммным управлением



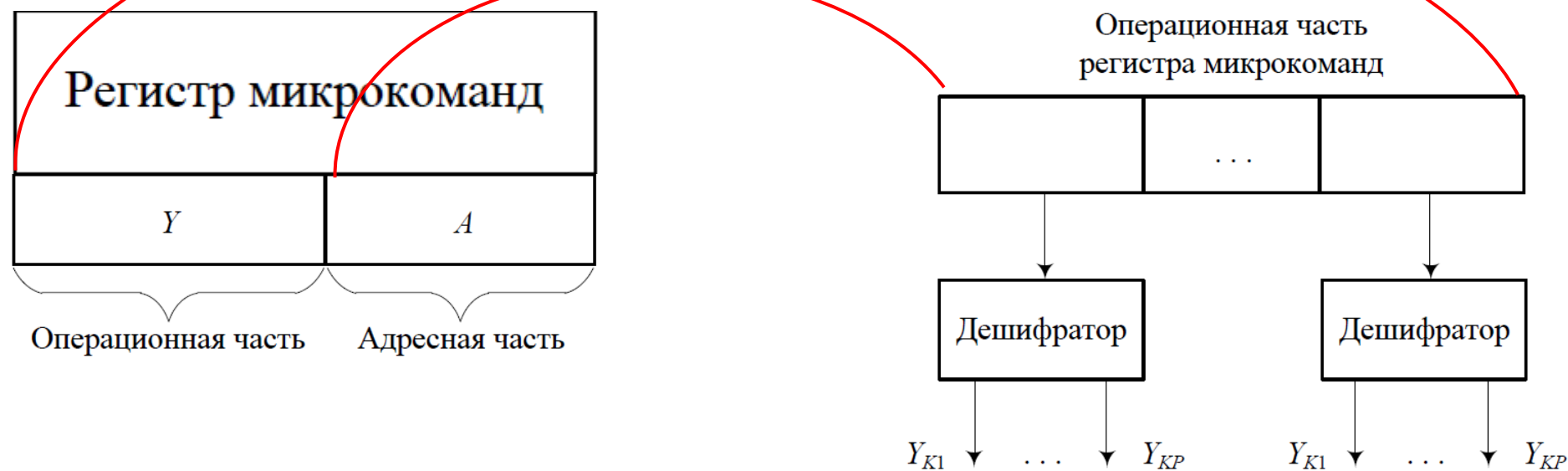
Горизонтальное микропрограммирование



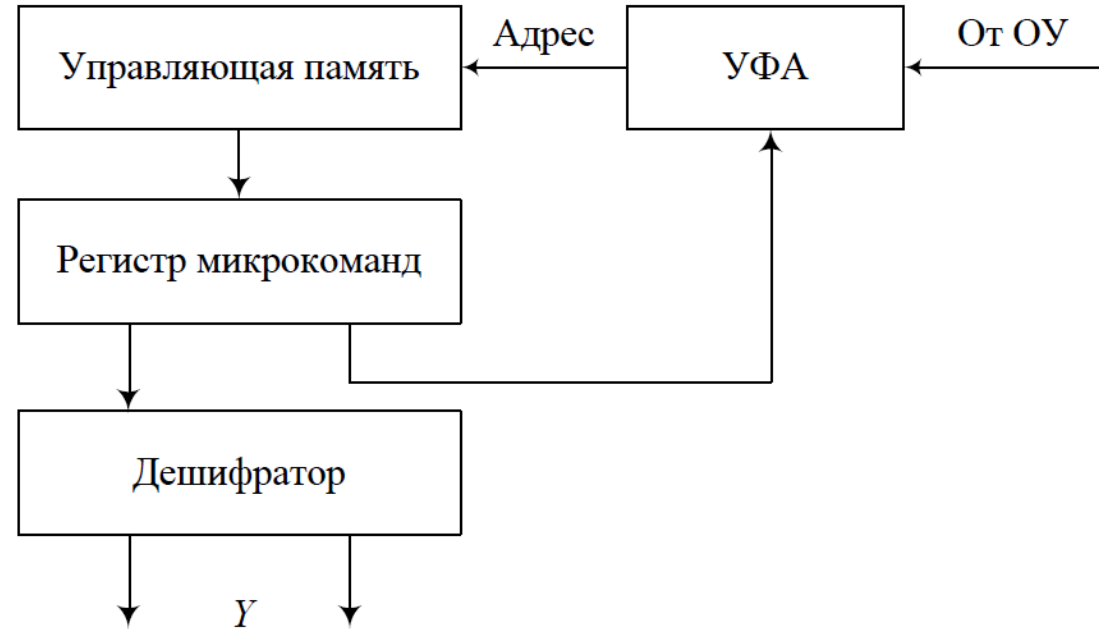
Вертикальное микропрограммирование



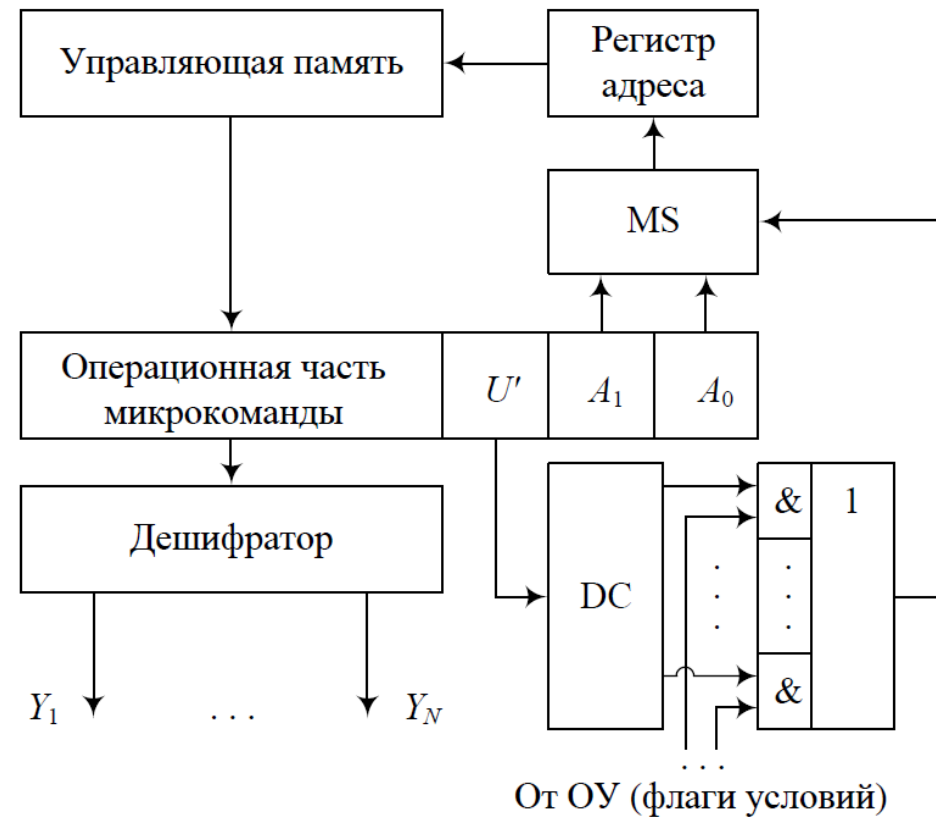
Квази-вертикальное микропрограммирование



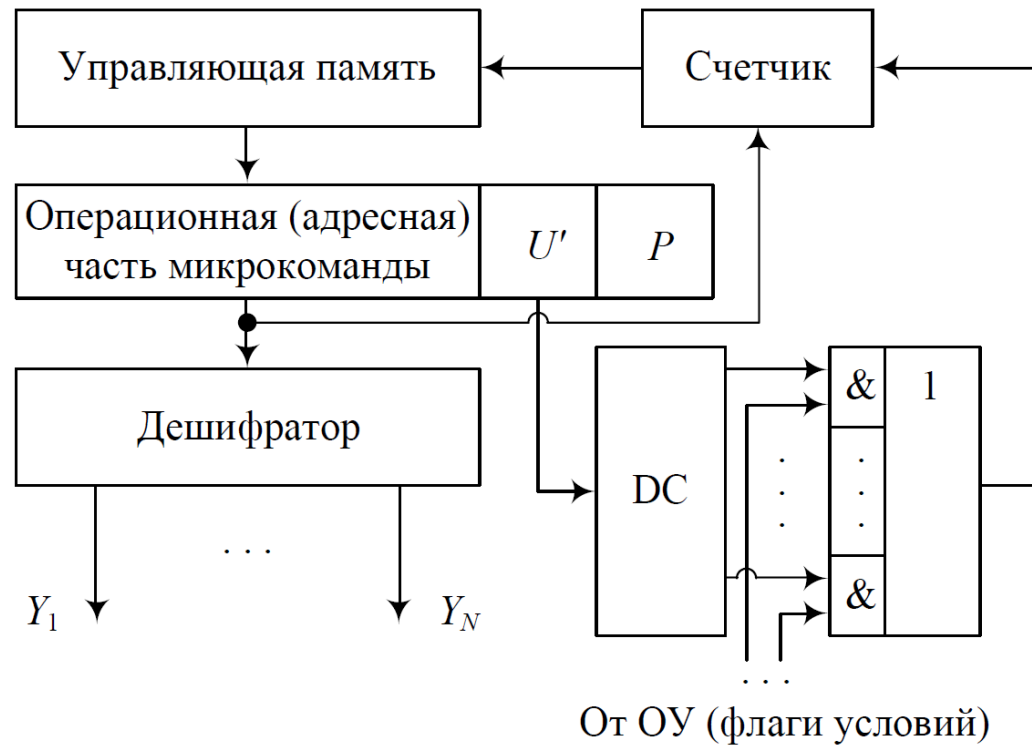
Устройство с микропрограммным управлением

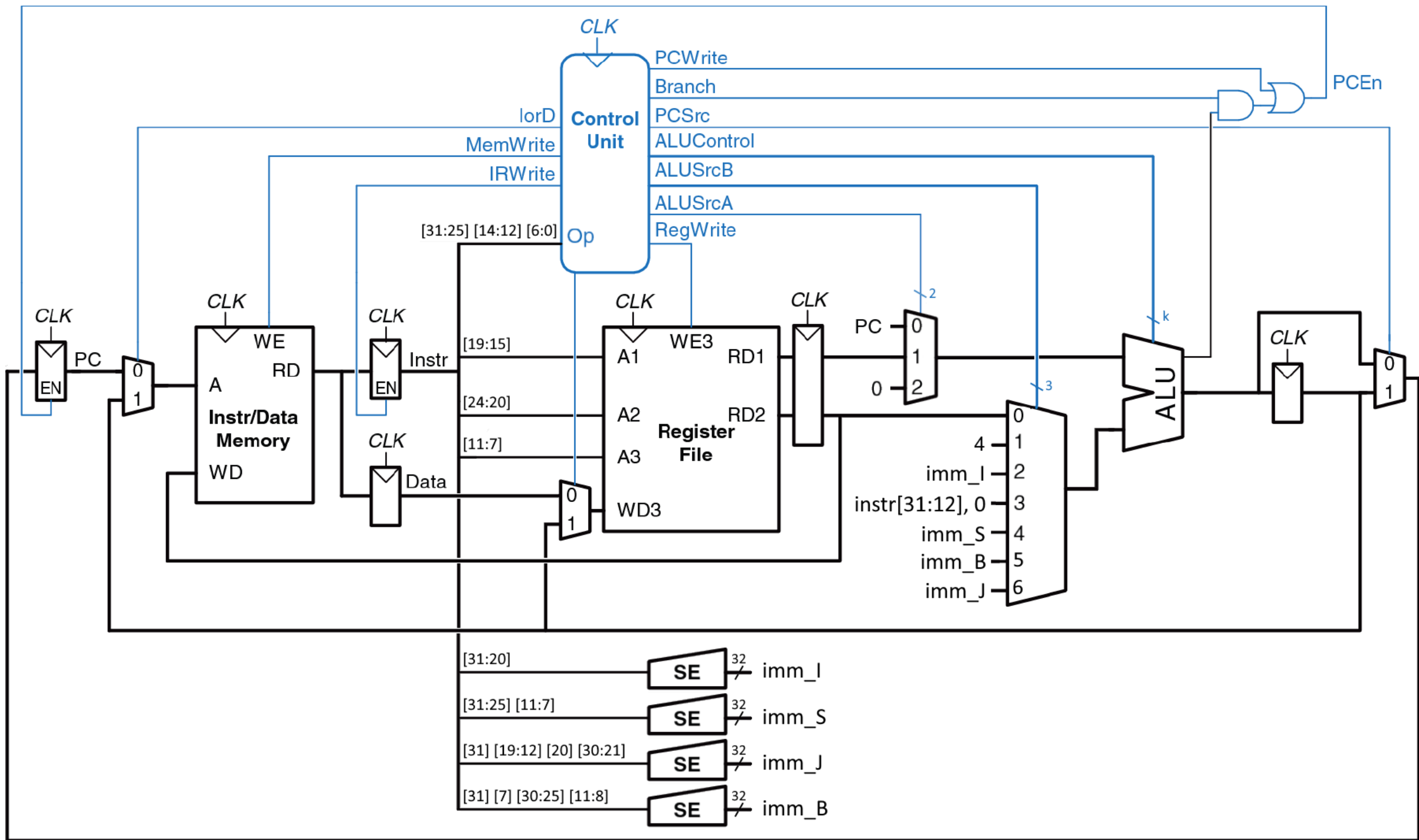


УМУ с принудительной адресацией



УМУ с естественной адресацией





Оценка производительности

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

4.12



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

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

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

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