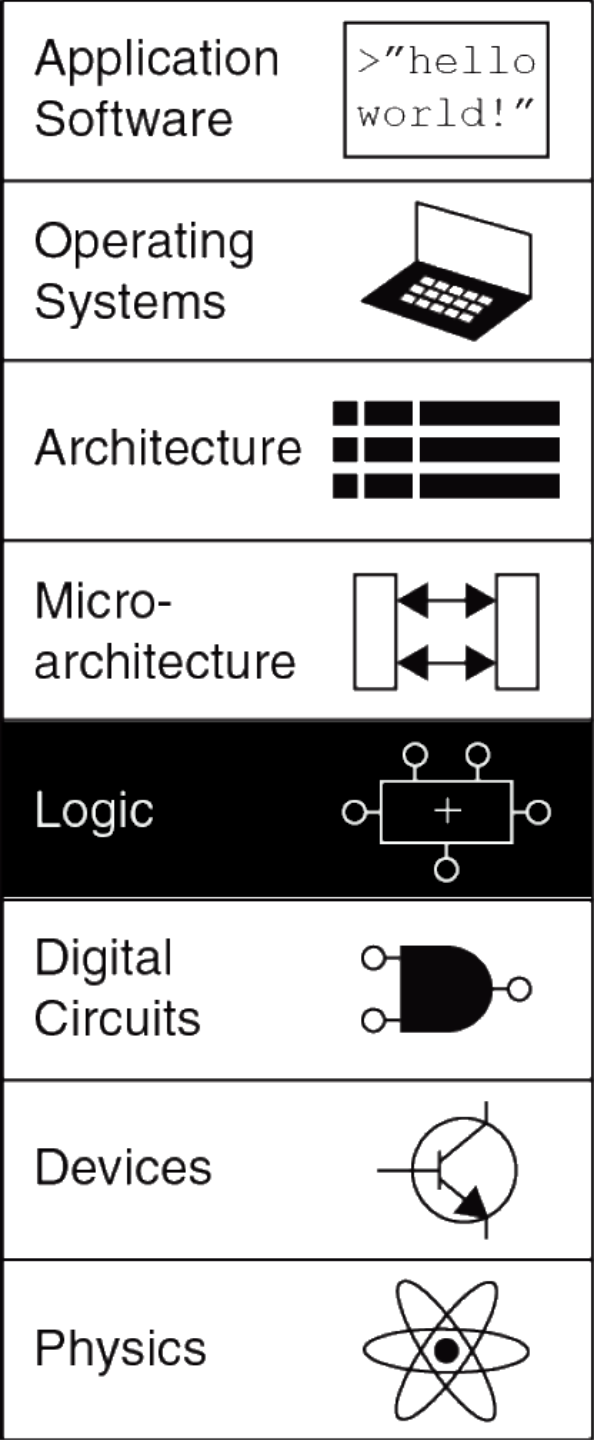
Архитектуры процессорных систем

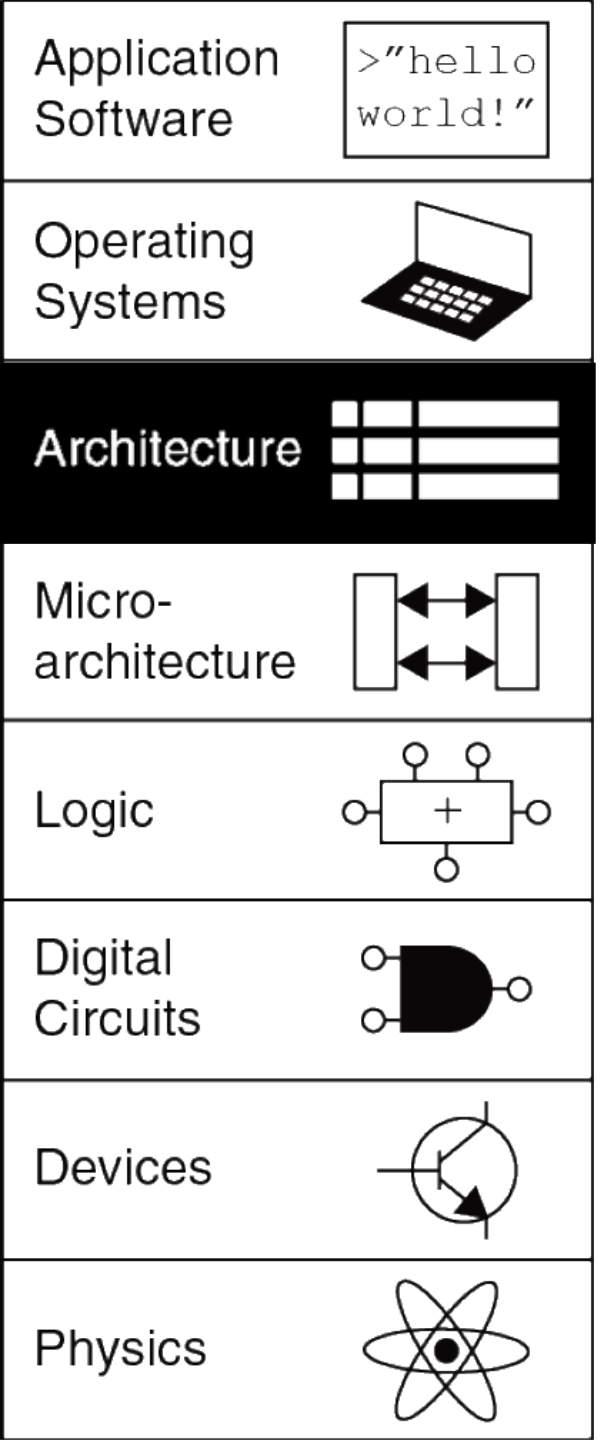
Лекция 7. Однотактный процессор RISC-V

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

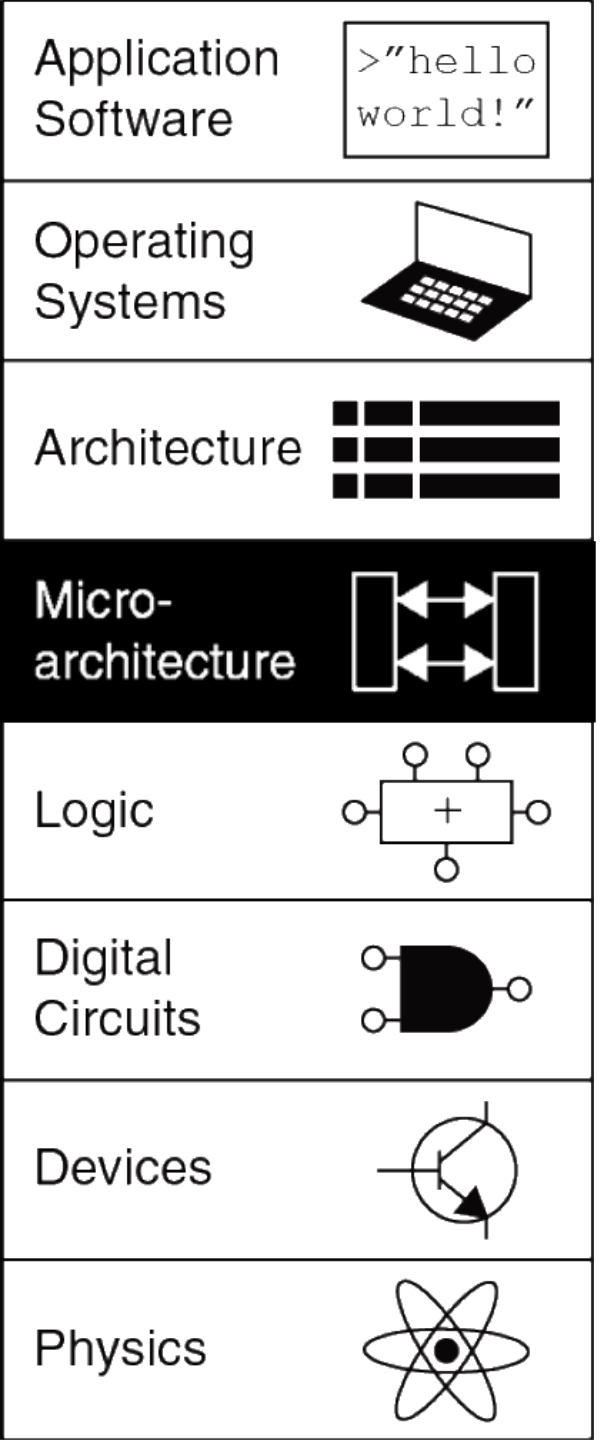
План лекции

- Классификация микроархитектур
- Кодирование инструкций RISC-V
- Синтез процессора с однотактной микроархитектурой
- Оценка производительности полученного процессора


Application Software
>"hello world!"
Operating Systems
Architecture
Micro-architecture
Logic
+
Digital Circuits
Devices
Physics



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

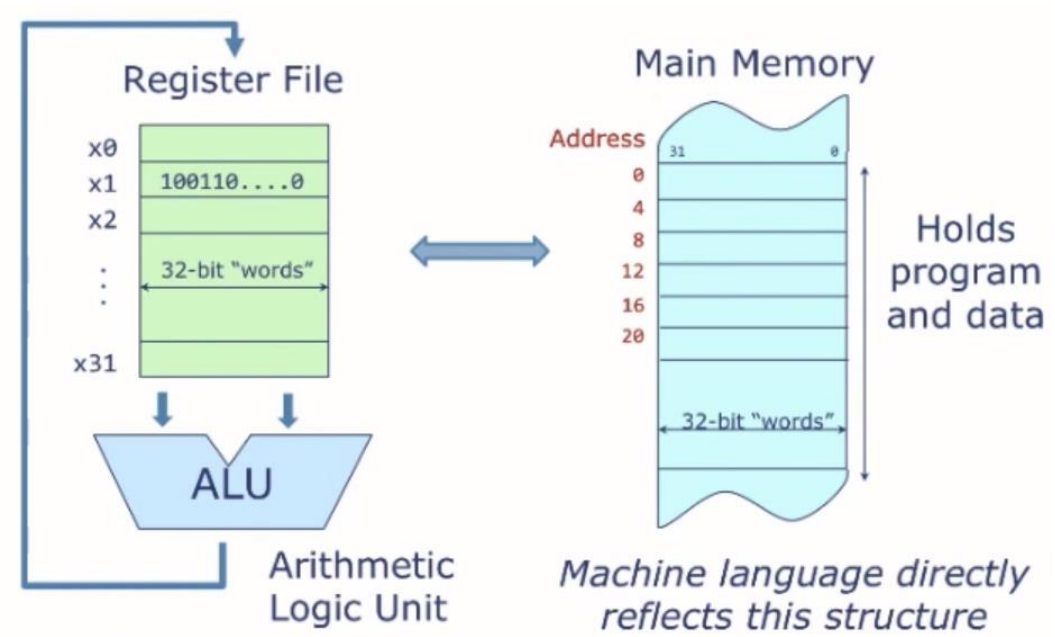


– физическая модель, которая устанавливает состав, порядок и принципы взаимодействия основных функциональных частей процессора (структурная организация)

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

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 `lalr register`
  - Условный переход `comp src1, src2, label`

Набор инструкций RISC-V

Inst	Name	FMT	Opcode	F3	F7	Description (C)	Note
add	ADD	R	0000011	0x0	0x00	rd = rs1 + rs2	
sub	SUB	R	0000011	0x0	0x20	rd = rs1 - rs2	
xor	XOR	R	0000011	0x4	0x00	rd = rs1 ^ rs2	
or	OR	R	0000011	0x6	0x00	rd = rs1 | rs2	
and	AND	R	0000011	0x7	0x00	rd = rs1 & rs2	
sll	Shift Left Logical	R	0000011	0x1	0x00	rd = rs1 << rs2	
srl	Shift Right Logical	R	0000011	0x2	0x00	rd = rs1 >> rs2	
sra	Shift Right Arith*	R	0000011	0x3	0x20	rd = rs1 >> rs2	msb-extends
slt	Set Less Than	R	0110011	0x2		rd = (rs1 < rs2)?1:0	
sltu	Set Less Than (U)	R	0110011	0x3		rd = (rs1 < rs2)?1:0	zero-extends
addi	ADD Immediate	I	0010011	0x0	0x00	rd = rs1 + imm	
xori	XOR Immediate	I	0010011	0x0	0x00	rd = rs1 ^ imm	
ori	OR Immediate	I	0010011	0x0	0x00	rd = rs1 | imm	
andi	AND Immediate	I	0010011	0x0	0x00	rd = rs1 & imm	
slli	Shift Left Logical Imm	I	0010011	0x1	0x00	rd = rs1 << imm	
srli	Shift Right Logical Imm	I	0010011	0x1	0x00	rd = rs1 >> imm	
srai	Shift Right Arith Imm	I	0010011	0x3	0x20	rd = rs1 >> imm	msb-extends
slt	Set Less Than Imm	I	0010011	0x2		rd = (rs1 < imm)?1:0	
sltu	Set Less Than Imm (U)	I	0010011	0x3		rd = (rs1 < imm)?1:0	zero-extends
lb	Load Byte	I	0000011	0x0		rd = M[rs1+imm][0:7]	
lh	Load Half	I	0000011	0x1		rd = M[rs1+imm][0:15]	
lw	Load Word	I	0000011	0x2		rd = M[rs1+imm][0:31]	
lbu	Load Byte (U)	I	0000011	0x4		rd = M[rs1+imm][0:7]	zero-extends
lhu	Load Half (U)	I	0000011	0x5		rd = M[rs1+imm][0:15]	zero-extends
sb	Store Byte	S	0100011	0x0		M[rs1+imm][0:7] = rs2[0:7]	
sh	Store Half	S	0100011	0x1		M[rs1+imm][0:15] = rs2[0:15]	
sw	Store Word	S	0100011	0x2		M[rs1+imm][0:31] = rs2[0:31]	
beq	Branch ==	B	1100011	0x0		if(rs1 == rs2) PC += imm	
bne	Branch !=	B	1100011	0x1		if(rs1 != rs2) PC += imm	
blt	Branch <	B	1100011	0x4		if(rs1 < rs2) PC += imm	
bge	Branch ≤	B	1100011	0x5		if(rs1 >= rs2) PC += imm	
bltu	Branch < (U)	B	1100011	0x6		if(rs1 < rs2) PC += imm	zero-extends
bgeu	Branch ≥ (U)	B	1100011	0x7		if(rs1 >= rs2) PC += imm	zero-extends
jal	Jump And Link	J	1101111			rd = PC+4; PC += imm	
jalr	Jump And Link Reg	I	1100111	0x0		rd = PC+4; PC = rs1	
lui	Load Upper Imm	U	0110111			rd = imm << 12	
auipc	Add Upper Imm to PC	U	0010111			rd = PC + (imm << 12)	
ecall	Environment Call	I	1110011	0x0	0x00	Transfer control to OS	imm: 0x000
ebreak	Environment Break	I	1110011	0x0	0x00	Transfer control to debugger	imm: 0x001

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

```
and x1, x2, x3
```

Inst	Name	FMT	Opcode	F3	F7	Description (C)	Note
and	AND	R	0000011	0x7	0x00	rd = rs1 & rs2	

R -type

funct7							rs2					rs1					funct3			rd					opcode						
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

0	3	2	7	1	3

0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	1	0	1	1	1	0	0	0	0	1	0	0	0	0	0	1	1

0	**0**	**3**	**1**	**7**	**0**	**8**	**3**

0x00317083

Пример программы RISC-V

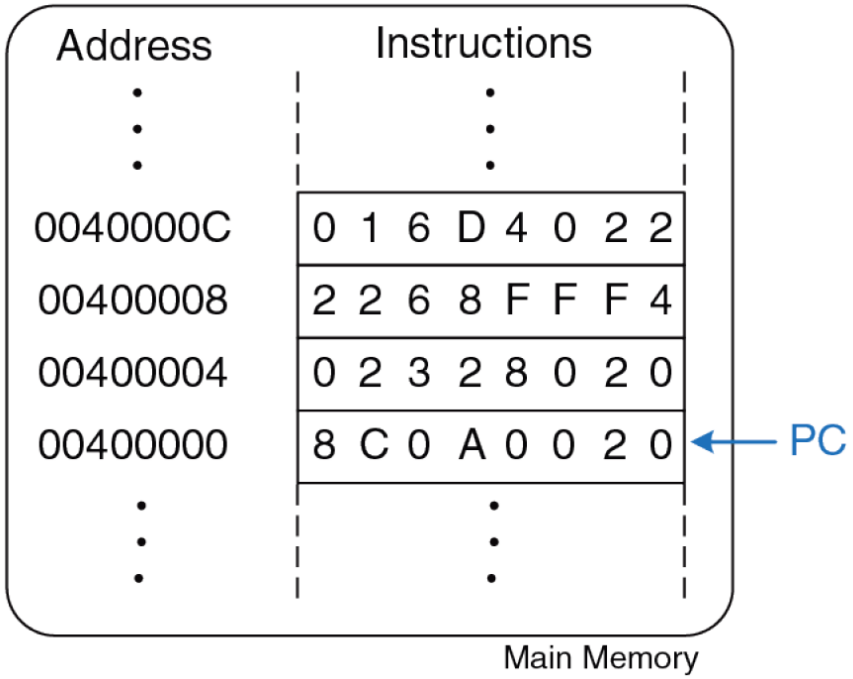
```
pc → li a0, 1
pc → li a1, 2
pc → addi sp, sp, -8
pc → sw ra, 0(sp)
pc → sw a1, 4(sp) // save a1
pc → jal ra, sum
pc → lw a1, 4(sp) // restore a1
pc → jal ra, sum
pc → lw ra, 0(sp)
pc → addi sp, sp, 8
```

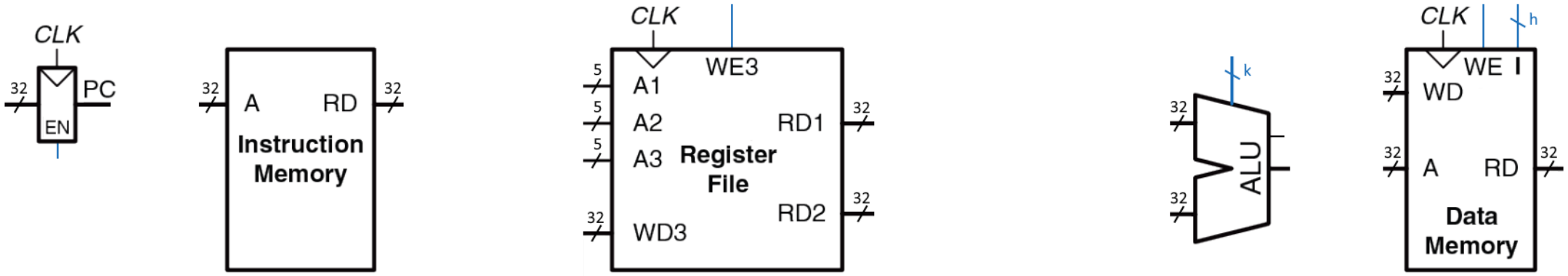
```
  sum:
pc →  add a0, a0, a1
pc →  ret
```

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`

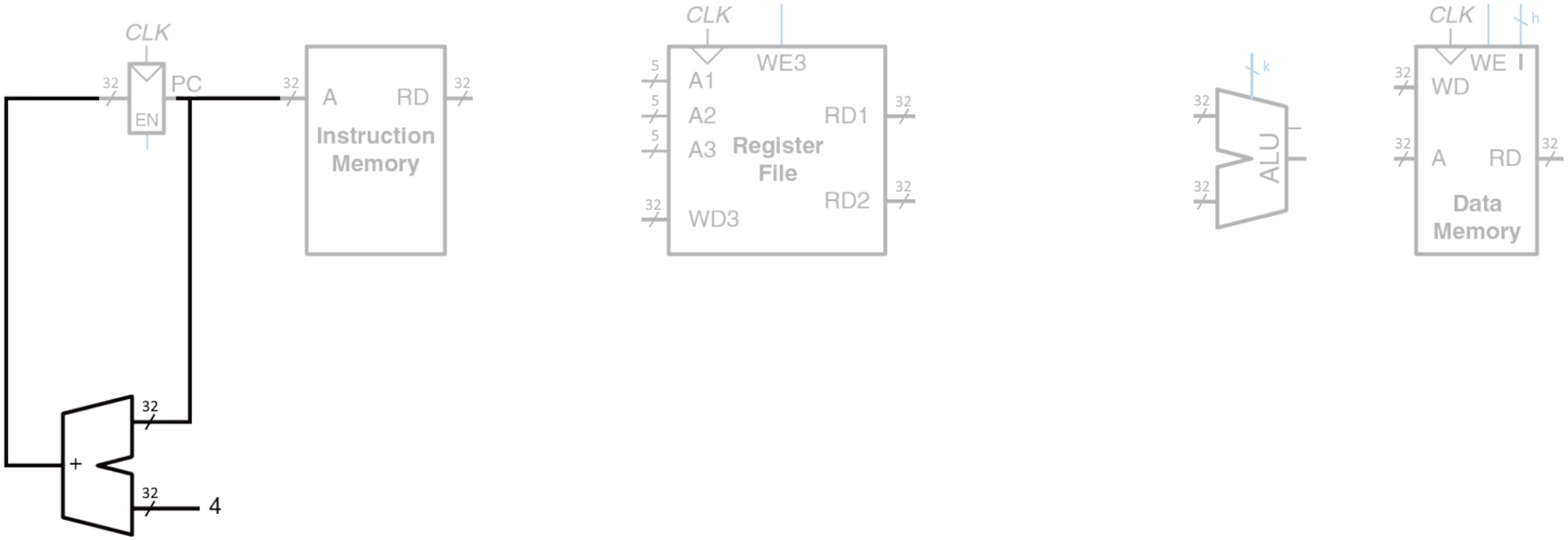
Stored Program




CLK
32
PC
EN
32
A
RD
32
Instruction Memory
CLK
WE3
5
A1
5
A2
5
A3
Register File
RD1
32
RD2
32
32
WD3
k
32
32
ALU
CLK
h
WE
32
WD
32
A
RD
32
Data Memory

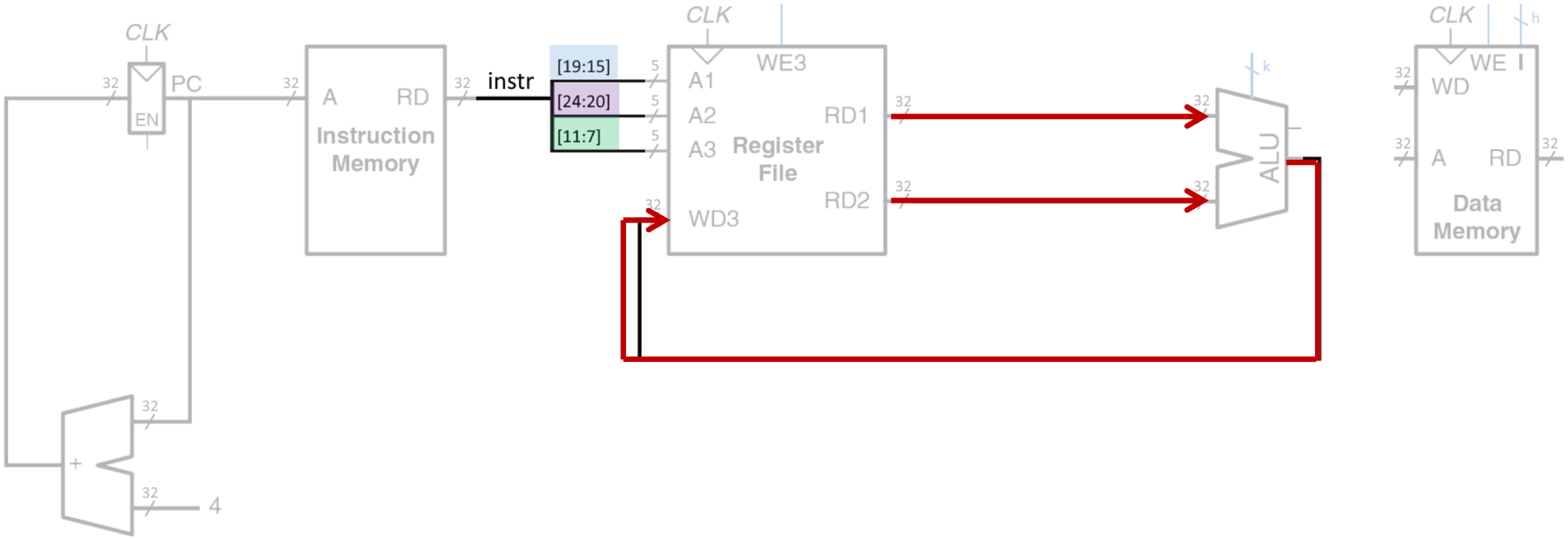
31 27	26 25	24 20	19 15	14 12	11 7	6 0	
funct7		rs2	rs1	funct3	rd	opcode	R-type
0000000		rs2	rs1	000	rd	0110011	ADD

```
add xN, xM, xK
rd = rs1 + rs2
```



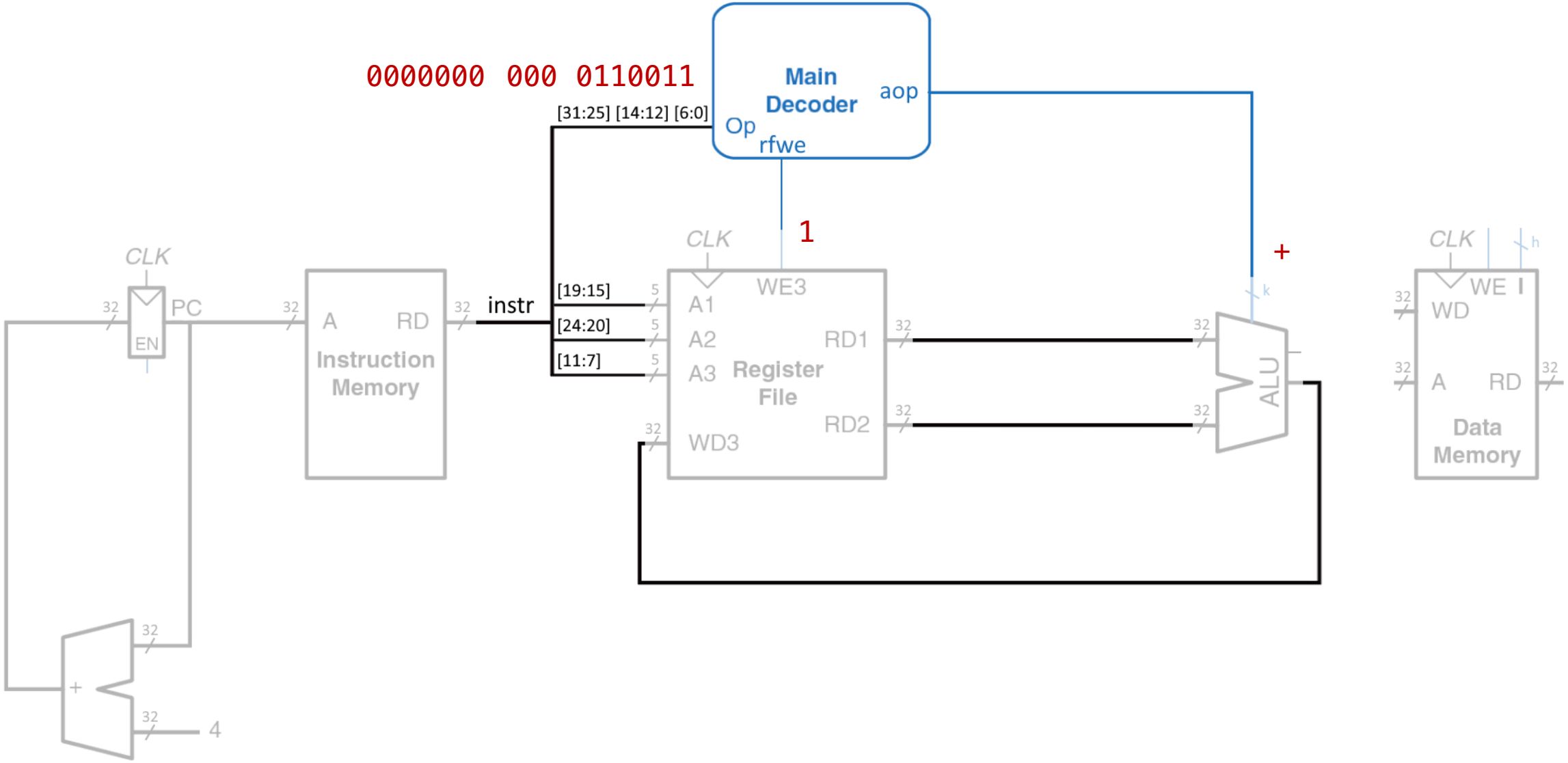
31 27	26 25	24 20	19 15	14 12	11 7	6 0	
funct7		rs2	rs1	funct3	rd	opcode	R-type
0000000		rs2	rs1	000	rd	0110011	ADD

```
add xN, xM, xK
rd = rs1 + rs2
```



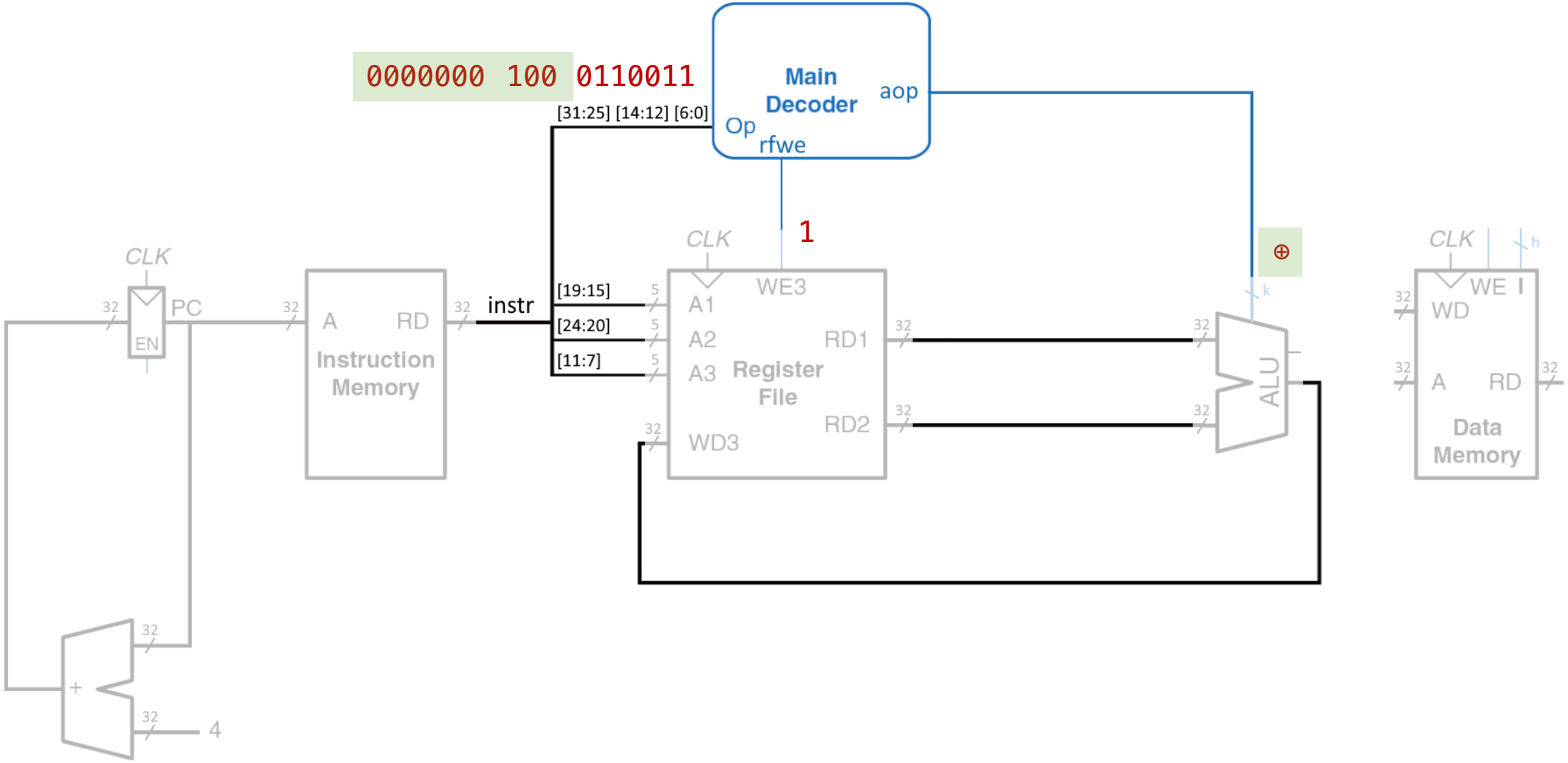
31 27	26 25	24 20	19 15	14 12	11 7	6 0	
funct7		rs2	rs1	funct3	rd	opcode	R-type
0000000		rs2	rs1	000	rd	0110011	ADD

```
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	100	rd	0110011	XOR

```
xor xN, xM, xK
rd = rs1 ^ rs2
```

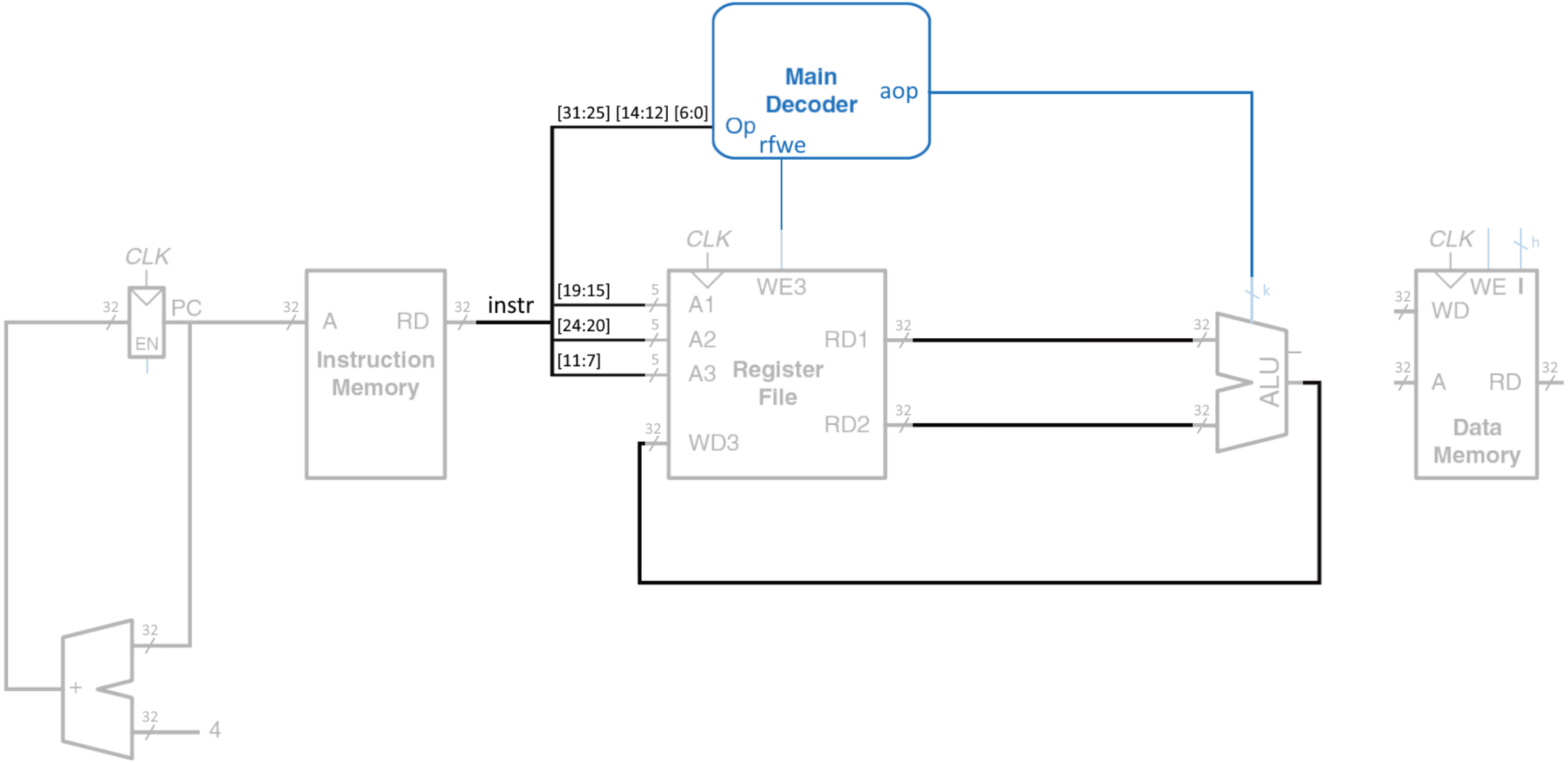


Набор инструкций RISC-V

Inst	Name	FMT	Opcode	F3	F7	Description (C)	Note
add	ADD	R	0000011	0x0	0x00	rd = rs1 + rs2	
sub	SUB	R	0000011	0x0	0x20	rd = rs1 - rs2	
xor	XOR	R	0000011	0x4	0x00	rd = rs1 ^ rs2	
or	OR	R	0000011	0x6	0x00	rd = rs1 | rs2	
and	AND	R	0000011	0x7	0x00	rd = rs1 & rs2	
sll	Shift Left Logical	R	0000011	0x1	0x00	rd = rs1 << rs2	
srl	Shift Right Logical	R	0000011	0x2	0x00	rd = rs1 >> rs2	
sra	Shift Right Arith*	R	0000011	0x3	0x20	rd = rs1 >> rs2	msb-extends
slt	Set Less Than	R	0110011	0x2		rd = (rs1 < rs2)?1:0	
sltu	Set Less Than (U)	R	0110011	0x3		rd = (rs1 < rs2)?1:0	zero-extends
addi	ADD Immediate	I	0010011	0x0	0x00	rd = rs1 + imm	
xori	XOR Immediate	I	0010011	0x0	0x00	rd = rs1 ^ imm	
ori	OR Immediate	I	0010011	0x0	0x00	rd = rs1 | imm	
andi	AND Immediate	I	0010011	0x0	0x00	rd = rs1 & imm	
slli	Shift Left Logical Imm	I	0010011	0x1	0x00	rd = rs1 << imm	
srli	Shift Right Logical Imm	I	0010011	0x1	0x00	rd = rs1 >> imm	
srai	Shift Right Arith Imm	I	0010011	0x3	0x20	rd = rs1 >> imm	msb-extends
slt	Set Less Than Imm	I	0010011	0x2		rd = (rs1 < imm)?1:0	
sltu	Set Less Than Imm (U)	I	0010011	0x3		rd = (rs1 < imm)?1:0	zero-extends
lb	Load Byte	I	0000011	0x0		rd = M[rs1+imm][0:7]	
lh	Load Half	I	0000011	0x1		rd = M[rs1+imm][0:15]	
lw	Load Word	I	0000011	0x2		rd = M[rs1+imm][0:31]	
lbu	Load Byte (U)	I	0000011	0x4		rd = M[rs1+imm][0:7]	zero-extends
lhu	Load Half (U)	I	0000011	0x5		rd = M[rs1+imm][0:15]	zero-extends
sb	Store Byte	S	0100011	0x0		M[rs1+imm][0:7] = rs2[0:7]	
sh	Store Half	S	0100011	0x1		M[rs1+imm][0:15] = rs2[0:15]	
sw	Store Word	S	0100011	0x2		M[rs1+imm][0:31] = rs2[0:31]	
beq	Branch ==	B	1100011	0x0		if(rs1 == rs2) PC += imm	
bne	Branch !=	B	1100011	0x1		if(rs1 != rs2) PC += imm	
blt	Branch <	B	1100011	0x4		if(rs1 < rs2) PC += imm	
bge	Branch ≤	B	1100011	0x5		if(rs1 >= rs2) PC += imm	
bltu	Branch < (U)	B	1100011	0x6		if(rs1 < rs2) PC += imm	zero-extends
bgeu	Branch ≥ (U)	B	1100011	0x7		if(rs1 >= rs2) PC += imm	zero-extends
jal	Jump And Link	J	1101111			rd = PC+4; PC += imm	
jalr	Jump And Link Reg	I	1100111	0x0		rd = PC+4; PC = rs1	
lui	Load Upper Imm	U	0110111			rd = imm << 12	
auipc	Add Upper Imm to PC	U	0010111			rd = PC + (imm << 12)	
ecall	Environment Call	I	1110011	0x0	0x00	Transfer control to OS	imm: 0x000
ebreak	Environment Break	I	1110011	0x0	0x00	Transfer control to debugger	imm: 0x001

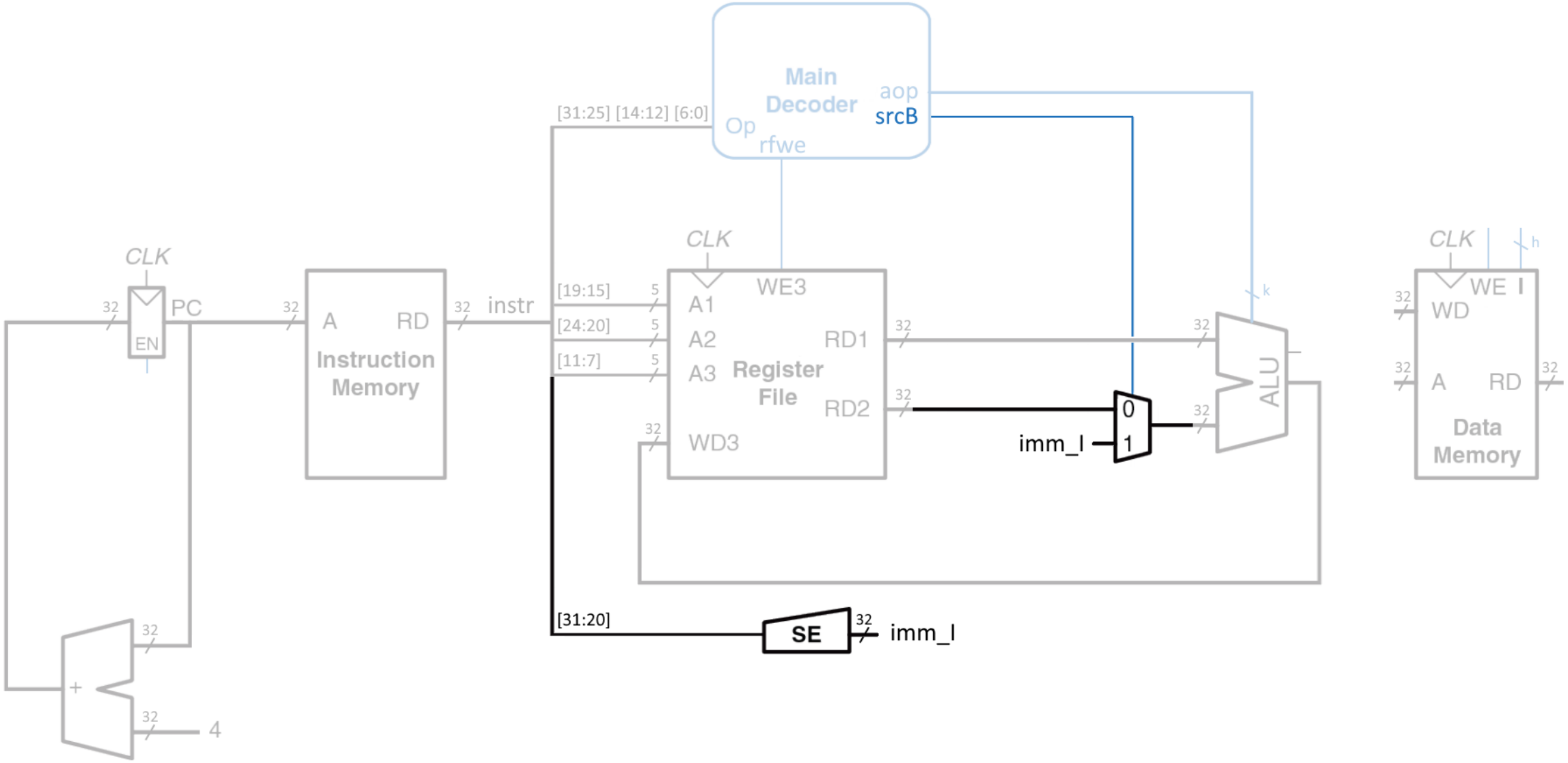
31 27 26 25 24 20	19 15	14 12	11 7	6 0	
imm[11:0]	rs1	funct3	rd	opcode	I-type
imm[11:0]	rs1	000	rd	0010011	ADDI

```
addi xN, xM, const
rd = rs1 + imm
```



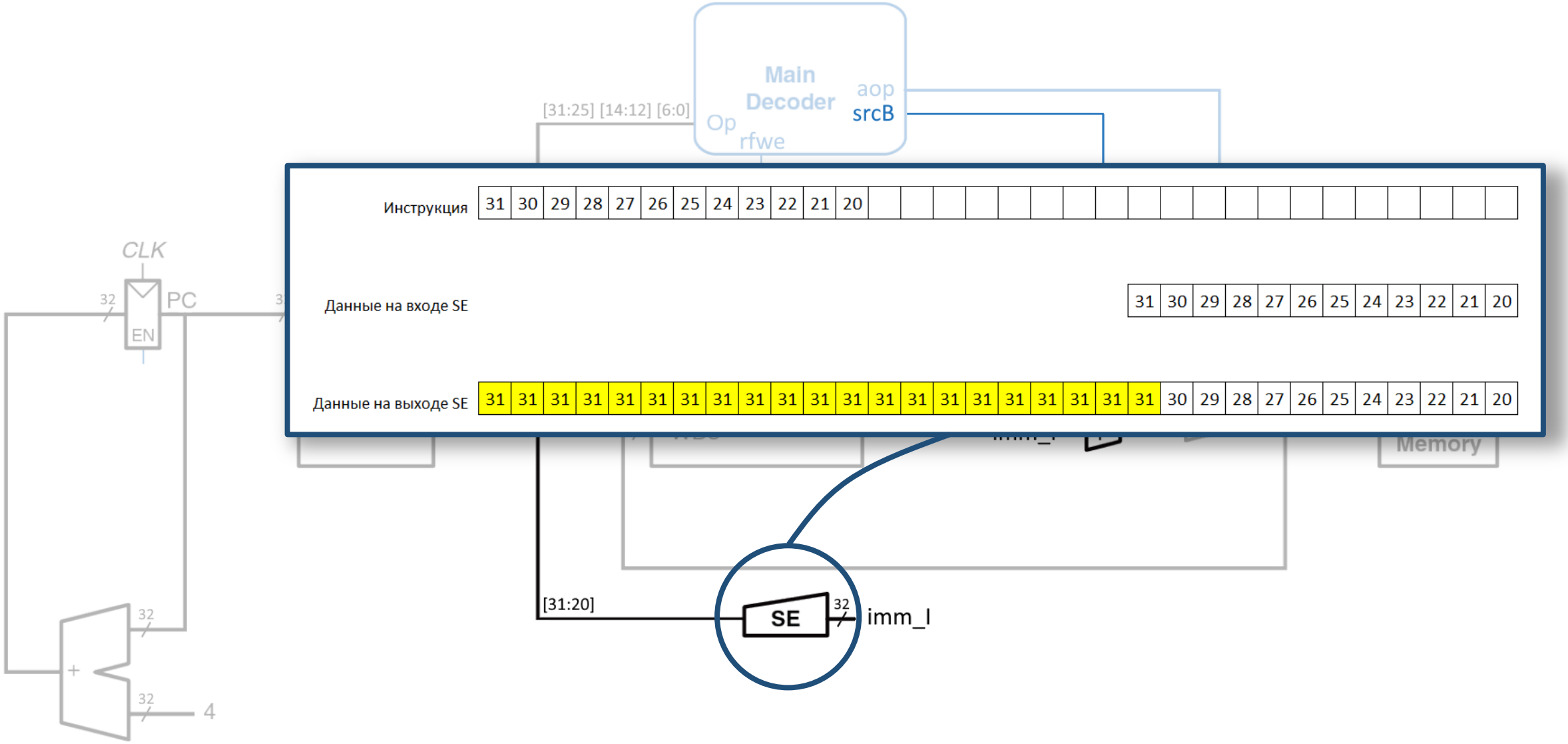
31 – 27, 26 – 25, 24 – 20	19 – 15	14 – 12	11 – 7	6 – 0	
imm[11:0]	rs1	funct3	rd	opcode	I-type
imm[11:0]	rs1	000	rd	0010011	ADDI

```
addi xN, xM, const
rd = rs1 + imm
```



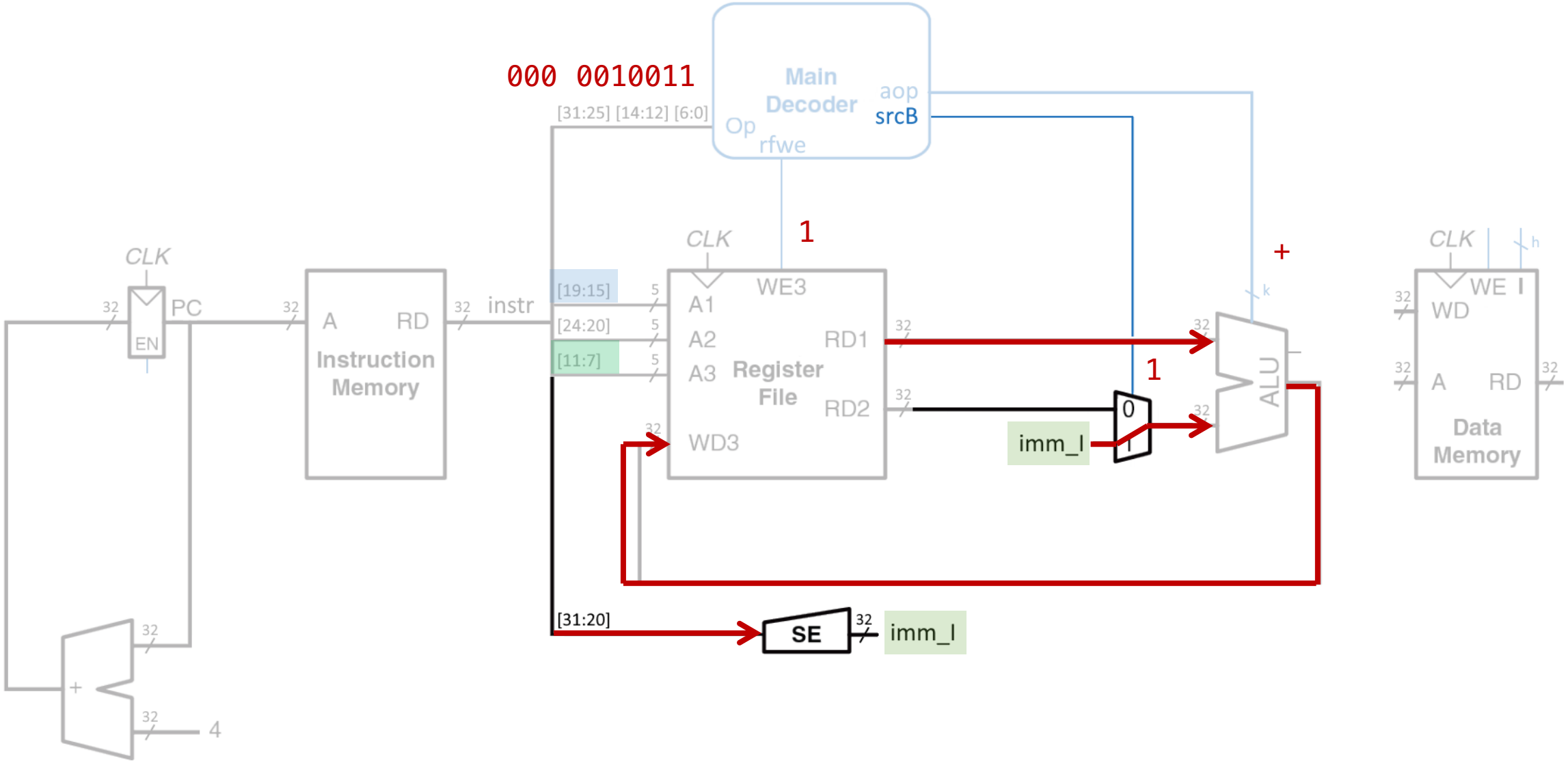
31 27	26 25 24 20	19 15	14 12	11 7	6 0	
imm[11:0]		rs1	funct3	rd	opcode	I-type
imm[11:0]		rs1	000	rd	0010011	ADDI

```
addi xN, xM, const
rd = rs1 + imm
```



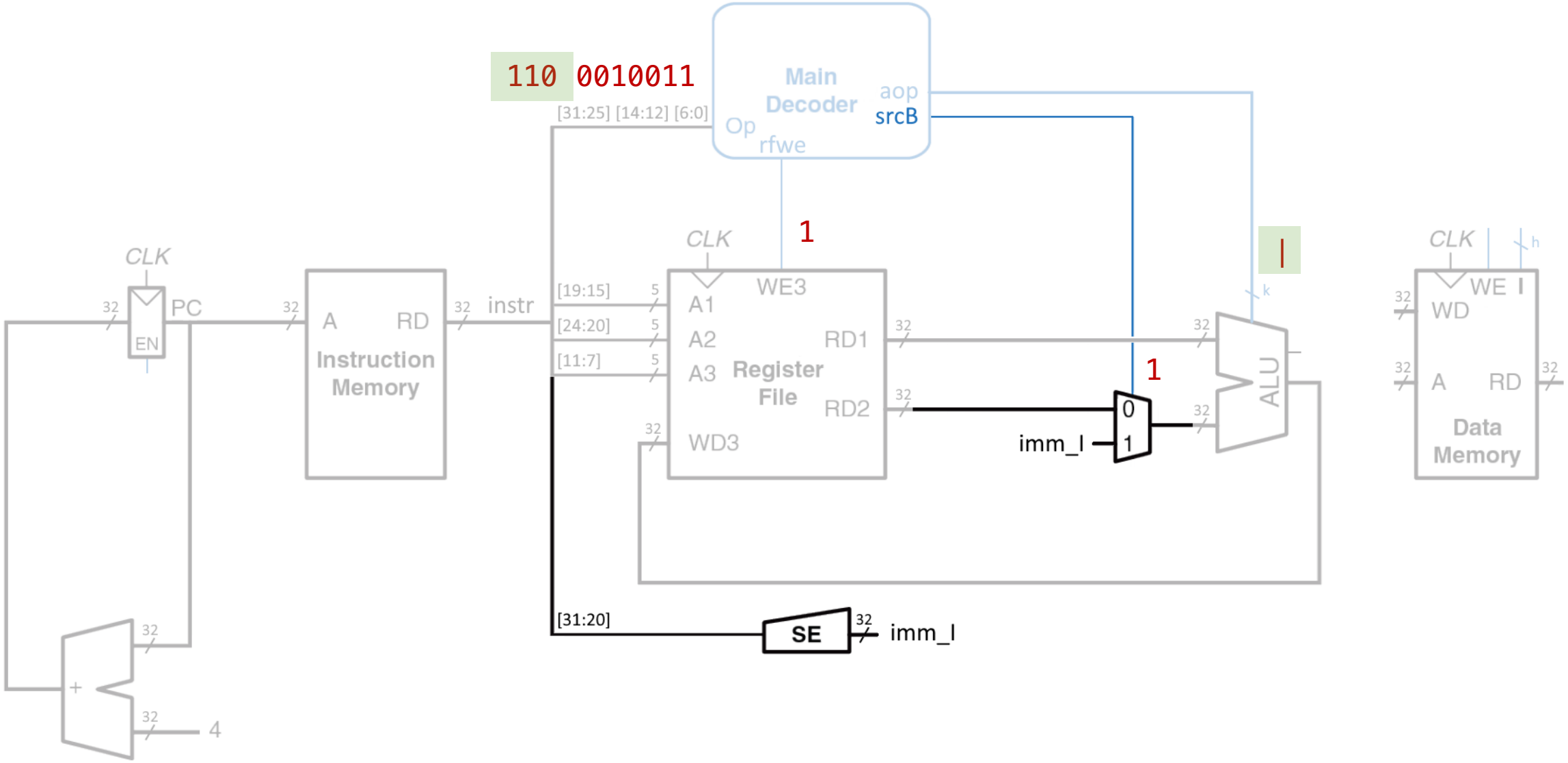
31 – 27, 26 – 25, 24 – 20	19 – 15	14 – 12	11 – 7	6 – 0	
imm[11:0]	rs1	funct3	rd	opcode	I-type
imm[11:0]	rs1	000	rd	0010011	ADDI

```
addi xN, xM, const
rd = rs1 + imm
```



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	110	rd	0010011	ORI

```
ori xN, xM, const
rd = rs1 | imm
```

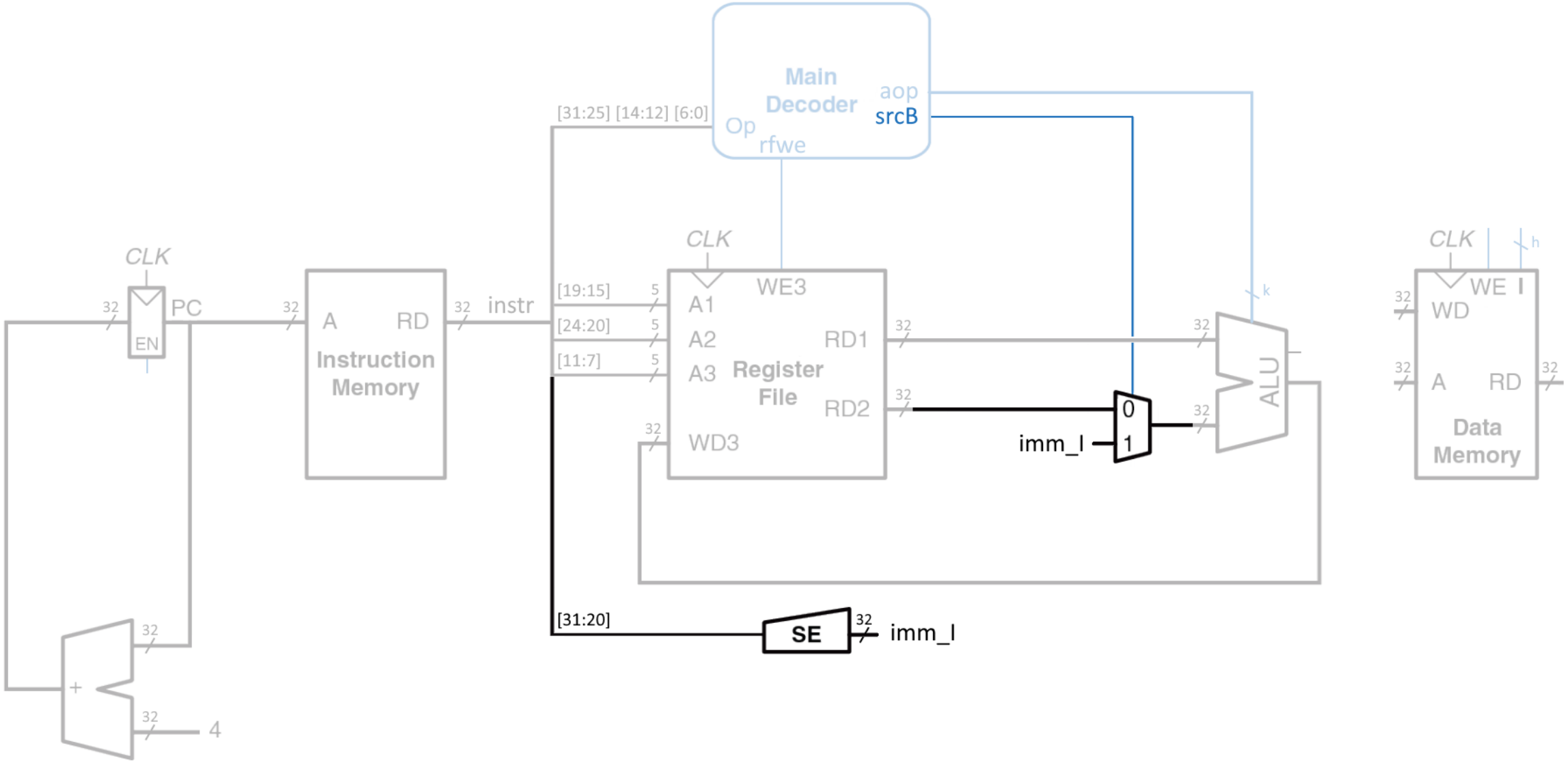


Набор инструкций RISC-V

Inst	Name	FMT	Opcode	F3	F7	Description (C)	Note
add	ADD	R	0000011	0x0	0x00	rd = rs1 + rs2	
sub	SUB	R	0000011	0x0	0x20	rd = rs1 - rs2	
xor	XOR	R	0000011	0x4	0x00	rd = rs1 ^ rs2	
or	OR	R	0000011	0x6	0x00	rd = rs1 | rs2	
and	AND	R	0000011	0x7	0x00	rd = rs1 & rs2	
sll	Shift Left Logical	R	0000011	0x1	0x00	rd = rs1 << rs2	
srl	Shift Right Logical	R	0000011	0x2	0x00	rd = rs1 >> rs2	
sra	Shift Right Arith*	R	0000011	0x3	0x20	rd = rs1 >> rs2	msb-extends
slt	Set Less Than	R	0110011	0x2		rd = (rs1 < rs2)?1:0	
sltu	Set Less Than (U)	R	0110011	0x3		rd = (rs1 < rs2)?1:0	zero-extends
addi	ADD Immediate	I	0010011	0x0	0x00	rd = rs1 + imm	
xori	XOR Immediate	I	0010011	0x0	0x00	rd = rs1 ^ imm	
ori	OR Immediate	I	0010011	0x0	0x00	rd = rs1 | imm	
andi	AND Immediate	I	0010011	0x0	0x00	rd = rs1 & imm	
slli	Shift Left Logical Imm	I	0010011	0x1	0x00	rd = rs1 << imm	
srli	Shift Right Logical Imm	I	0010011	0x1	0x00	rd = rs1 >> imm	
srai	Shift Right Arith Imm	I	0010011	0x3	0x20	rd = rs1 >> imm	msb-extends
slt	Set Less Than Imm	I	0010011	0x2		rd = (rs1 < imm)?1:0	
sltu	Set Less Than Imm (U)	I	0010011	0x3		rd = (rs1 < imm)?1:0	zero-extends
lb	Load Byte	I	0000011	0x0		rd = M[rs1+imm][0:7]	
lh	Load Half	I	0000011	0x1		rd = M[rs1+imm][0:15]	
lw	Load Word	I	0000011	0x2		rd = M[rs1+imm][0:31]	
lbu	Load Byte (U)	I	0000011	0x4		rd = M[rs1+imm][0:7]	zero-extends
lhu	Load Half (U)	I	0000011	0x5		rd = M[rs1+imm][0:15]	zero-extends
sb	Store Byte	S	0100011	0x0		M[rs1+imm][0:7] = rs2[0:7]	
sh	Store Half	S	0100011	0x1		M[rs1+imm][0:15] = rs2[0:15]	
sw	Store Word	S	0100011	0x2		M[rs1+imm][0:31] = rs2[0:31]	
beq	Branch ==	B	1100011	0x0		if(rs1 == rs2) PC += imm	
bne	Branch !=	B	1100011	0x1		if(rs1 != rs2) PC += imm	
blt	Branch <	B	1100011	0x4		if(rs1 < rs2) PC += imm	
bge	Branch ≤	B	1100011	0x5		if(rs1 >= rs2) PC += imm	
bltu	Branch < (U)	B	1100011	0x6		if(rs1 < rs2) PC += imm	zero-extends
bgeu	Branch ≥ (U)	B	1100011	0x7		if(rs1 >= rs2) PC += imm	zero-extends
jal	Jump And Link	J	1101111			rd = PC+4; PC += imm	
jalr	Jump And Link Reg	I	1100111	0x0		rd = PC+4; PC = rs1	
lui	Load Upper Imm	U	0110111			rd = imm << 12	
auipc	Add Upper Imm to PC	U	0010111			rd = PC + (imm << 12)	
ecall	Environment Call	I	1110011	0x0	0x00	Transfer control to OS	imm: 0x000
ebreak	Environment Break	I	1110011	0x0	0x00	Transfer control to debugger	imm: 0x001

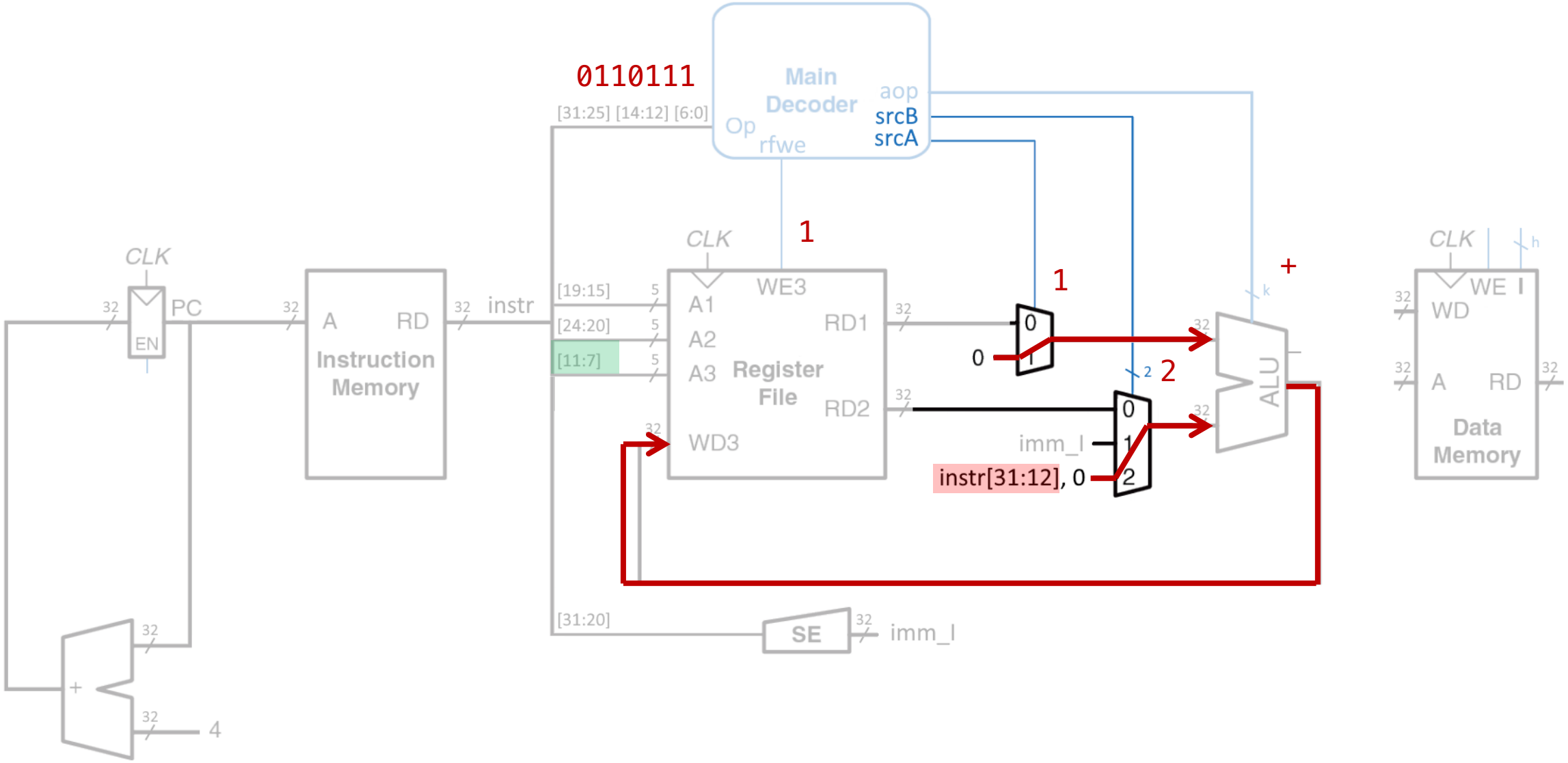
31 – 27	26 – 25	24 – 20	19 – 15	14 – 12	11 – 7	6 – 0	
imm[31:12]					rd	opcode	U-type
imm[31:12]					rd	0110111	LUI

```
lui xN, const
rd = imm << 12
```



31 – 27	26 – 25	24 – 20	19 – 15	14 – 12	11 – 7	6 – 0	
imm[31:12]					rd	opcode	U-type
imm[31:12]					rd	0110111	LUI

```
lui xN, const
rd = imm << 12
```

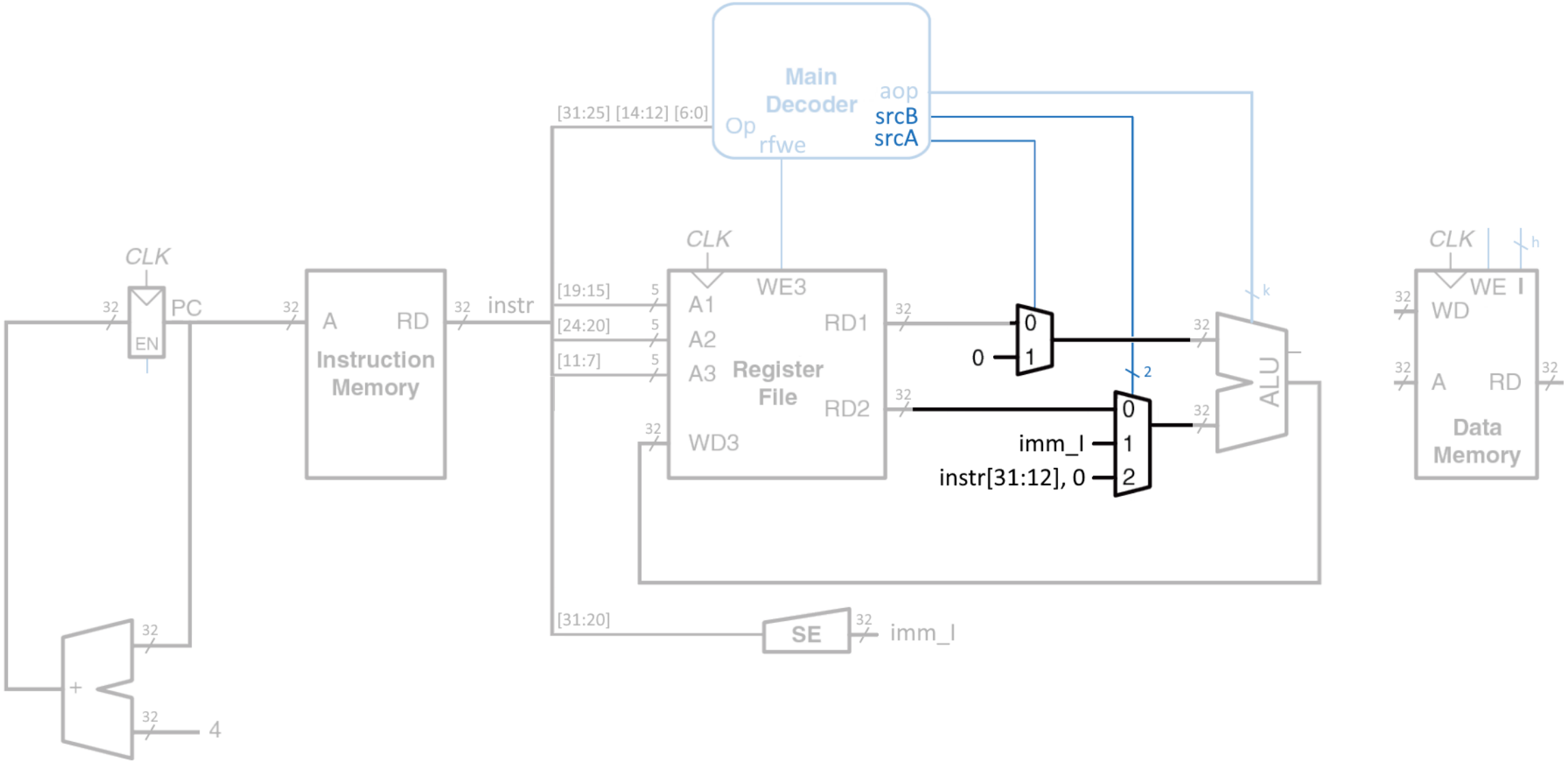


Набор инструкций RISC-V

Inst	Name	FMT	Opcode	F3	F7	Description (C)	Note
add	ADD	R	0000011	0x0	0x00	rd = rs1 + rs2	
sub	SUB	R	0000011	0x0	0x20	rd = rs1 - rs2	
xor	XOR	R	0000011	0x4	0x00	rd = rs1 ^ rs2	
or	OR	R	0000011	0x6	0x00	rd = rs1 | rs2	
and	AND	R	0000011	0x7	0x00	rd = rs1 & rs2	
sll	Shift Left Logical	R	0000011	0x1	0x00	rd = rs1 << rs2	
srl	Shift Right Logical	R	0000011	0x2	0x00	rd = rs1 >> rs2	
sra	Shift Right Arith*	R	0000011	0x3	0x20	rd = rs1 >> rs2	msb-extends
slt	Set Less Than	R	0110011	0x2		rd = (rs1 < rs2)?1:0	
sltu	Set Less Than (U)	R	0110011	0x3		rd = (rs1 < rs2)?1:0	zero-extends
addi	ADD Immediate	I	0010011	0x0	0x00	rd = rs1 + imm	
xori	XOR Immediate	I	0010011	0x0	0x00	rd = rs1 ^ imm	
ori	OR Immediate	I	0010011	0x0	0x00	rd = rs1 | imm	
andi	AND Immediate	I	0010011	0x0	0x00	rd = rs1 & imm	
slli	Shift Left Logical Imm	I	0010011	0x1	0x00	rd = rs1 << imm	
srli	Shift Right Logical Imm	I	0010011	0x1	0x00	rd = rs1 >> imm	
srai	Shift Right Arith Imm	I	0010011	0x3	0x20	rd = rs1 >> imm	msb-extends
slt	Set Less Than Imm	I	0010011	0x2		rd = (rs1 < imm)?1:0	
sltu	Set Less Than Imm (U)	I	0010011	0x3		rd = (rs1 < imm)?1:0	zero-extends
lb	Load Byte	I	0000011	0x0		rd = M[rs1+imm][0:7]	
lh	Load Half	I	0000011	0x1		rd = M[rs1+imm][0:15]	
lw	Load Word	I	0000011	0x2		rd = M[rs1+imm][0:31]	
lbu	Load Byte (U)	I	0000011	0x4		rd = M[rs1+imm][0:7]	zero-extends
lhu	Load Half (U)	I	0000011	0x5		rd = M[rs1+imm][0:15]	zero-extends
sb	Store Byte	S	0100011	0x0		M[rs1+imm][0:7] = rs2[0:7]	
sh	Store Half	S	0100011	0x1		M[rs1+imm][0:15] = rs2[0:15]	
sw	Store Word	S	0100011	0x2		M[rs1+imm][0:31] = rs2[0:31]	
beq	Branch ==	B	1100011	0x0		if(rs1 == rs2) PC += imm	
bne	Branch !=	B	1100011	0x1		if(rs1 != rs2) PC += imm	
blt	Branch <	B	1100011	0x4		if(rs1 < rs2) PC += imm	
bge	Branch ≤	B	1100011	0x5		if(rs1 >= rs2) PC += imm	
bltu	Branch < (U)	B	1100011	0x6		if(rs1 < rs2) PC += imm	zero-extends
bgeu	Branch ≥ (U)	B	1100011	0x7		if(rs1 >= rs2) PC += imm	zero-extends
jal	Jump And Link	J	1101111			rd = PC+4; PC += imm	
jalr	Jump And Link Reg	I	1100111	0x0		rd = PC+4; PC = rs1	
lui	Load Upper Imm	U	0110111			rd = imm << 12	
auipc	Add Upper Imm to PC	U	0010111			rd = PC + (imm << 12)	
ecall	Environment Call	I	1110011	0x0	0x00	Transfer control to OS	imm: 0x000
ebreak	Environment Break	I	1110011	0x0	0x00	Transfer control to debugger	imm: 0x001

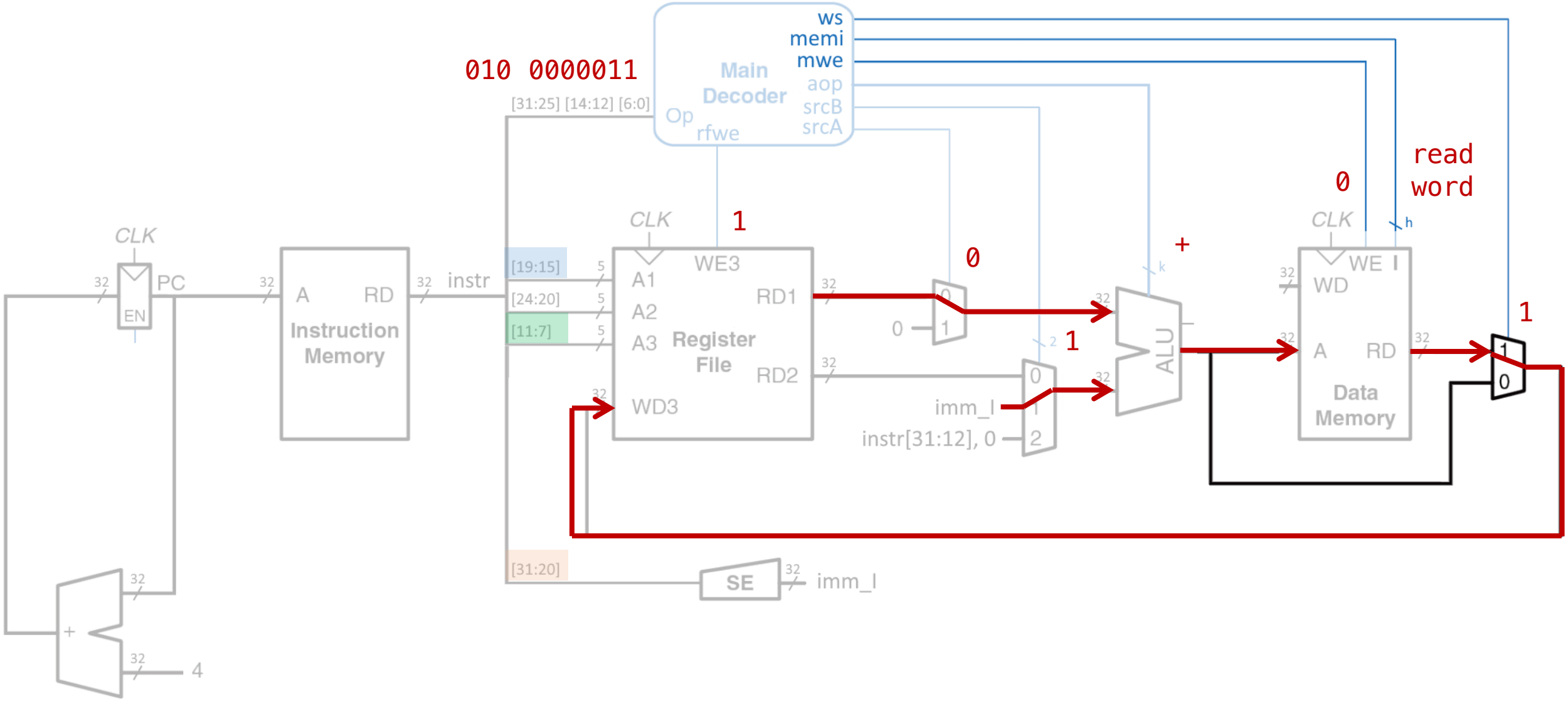
31 – 27, 26 – 25, 24 – 20	19 – 15	14 – 12	11 – 7	6 – 0	
imm[11:0]	rs1	funct3	rd	opcode	I-type
imm[11:0]	rs1	010	rd	0000011	LW

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



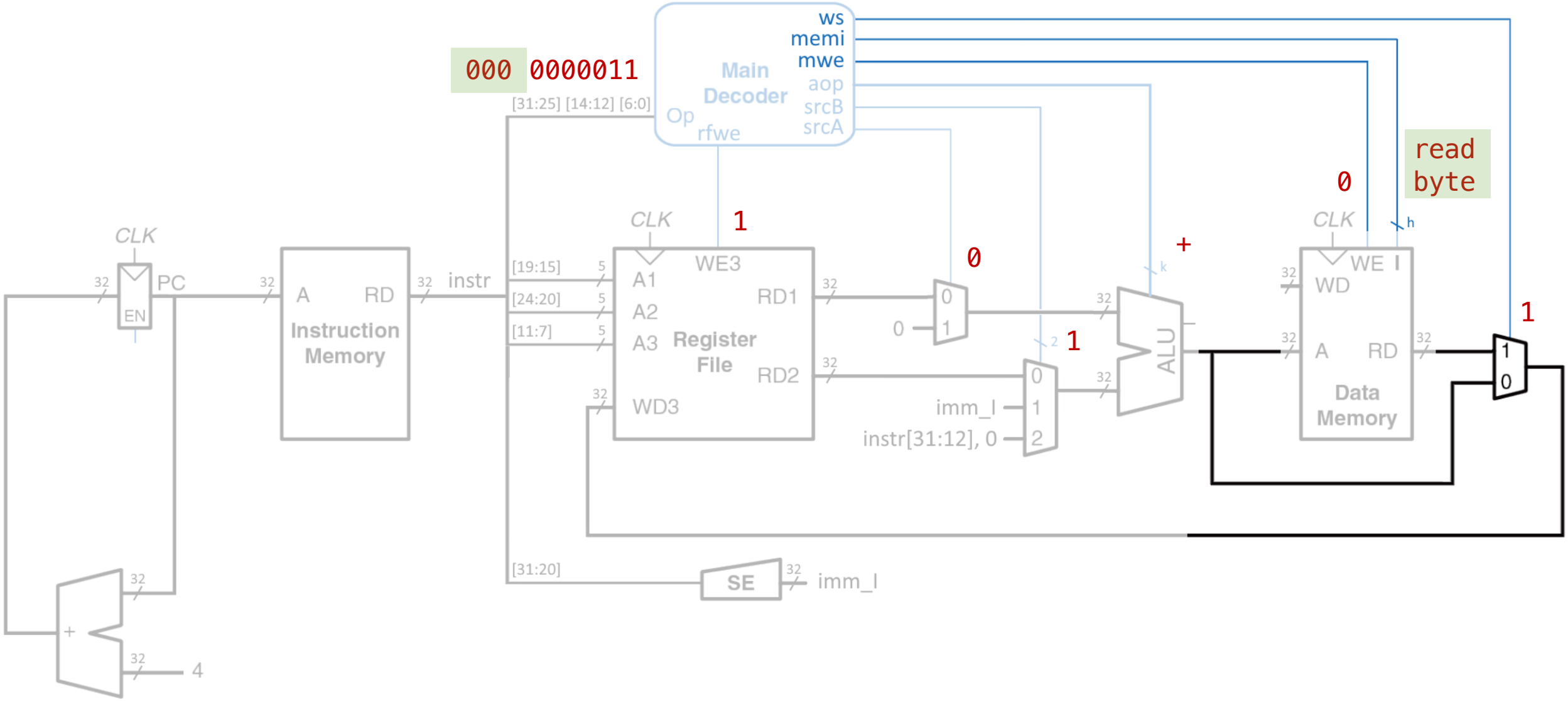
31 – 27	26 – 25	24 – 20	19 – 15	14 – 12	11 – 7	6 – 0	
imm[11:0]			rs1	funct3	rd	opcode	I-type
imm[11:0]			rs1	010	rd	0000011	LW

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



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	000	rd	0000011	LB

```
lb xN, offset(base)
rd = M[rs1+imm][0:7]
```

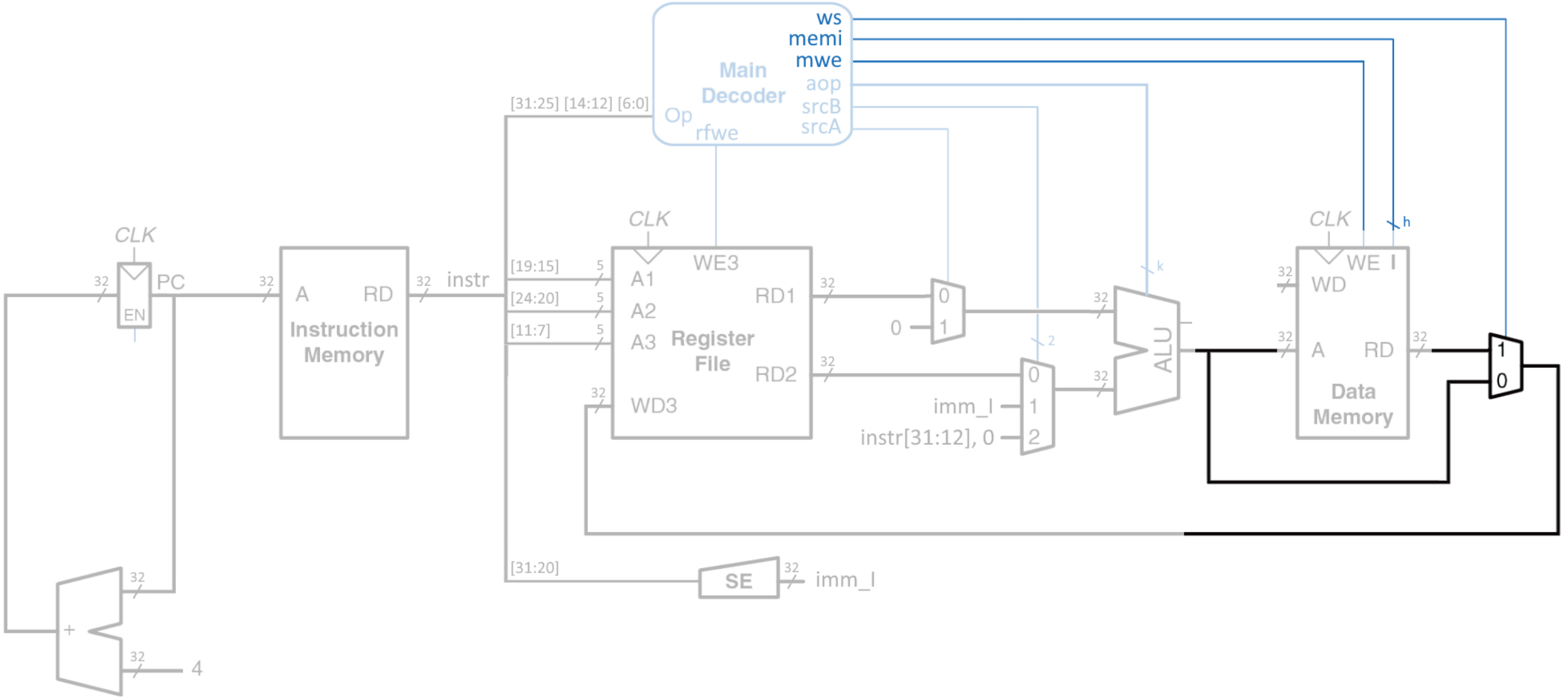


Набор инструкций RISC-V

Inst	Name	FMT	Opcode	F3	F7	Description (C)	Note
add	ADD	R	0000011	0x0	0x00	rd = rs1 + rs2	
sub	SUB	R	0000011	0x0	0x20	rd = rs1 - rs2	
xor	XOR	R	0000011	0x4	0x00	rd = rs1 ^ rs2	
or	OR	R	0000011	0x6	0x00	rd = rs1 | rs2	
and	AND	R	0000011	0x7	0x00	rd = rs1 & rs2	
sll	Shift Left Logical	R	0000011	0x1	0x00	rd = rs1 << rs2	
srl	Shift Right Logical	R	0000011	0x2	0x00	rd = rs1 >> rs2	
sra	Shift Right Arith*	R	0000011	0x3	0x20	rd = rs1 >> rs2	msb-extends
slt	Set Less Than	R	0110011	0x2		rd = (rs1 < rs2)?1:0	
sltu	Set Less Than (U)	R	0110011	0x3		rd = (rs1 < rs2)?1:0	zero-extends
addi	ADD Immediate	I	0010011	0x0	0x00	rd = rs1 + imm	
xori	XOR Immediate	I	0010011	0x0	0x00	rd = rs1 ^ imm	
ori	OR Immediate	I	0010011	0x0	0x00	rd = rs1 | imm	
andi	AND Immediate	I	0010011	0x0	0x00	rd = rs1 & imm	
slli	Shift Left Logical Imm	I	0010011	0x1	0x00	rd = rs1 << imm	
srli	Shift Right Logical Imm	I	0010011	0x1	0x00	rd = rs1 >> imm	
srai	Shift Right Arith Imm	I	0010011	0x3	0x20	rd = rs1 >> imm	msb-extends
slt	Set Less Than Imm	I	0010011	0x2		rd = (rs1 < imm)?1:0	
sltu	Set Less Than Imm (U)	I	0010011	0x3		rd = (rs1 < imm)?1:0	zero-extends
lb	Load Byte	I	0000011	0x0		rd = M[rs1+imm][0:7]	
lh	Load Half	I	0000011	0x1		rd = M[rs1+imm][0:15]	
lw	Load Word	I	0000011	0x2		rd = M[rs1+imm][0:31]	
lbu	Load Byte (U)	I	0000011	0x4		rd = M[rs1+imm][0:7]	zero-extends
lhu	Load Half (U)	I	0000011	0x5		rd = M[rs1+imm][0:15]	zero-extends
sb	Store Byte	S	0100011	0x0		M[rs1+imm][0:7] = rs2[0:7]	
sh	Store Half	S	0100011	0x1		M[rs1+imm][0:15] = rs2[0:15]	
sw	Store Word	S	0100011	0x2		M[rs1+imm][0:31] = rs2[0:31]	
beq	Branch ==	B	1100011	0x0		if(rs1 == rs2) PC += imm	
bne	Branch !=	B	1100011	0x1		if(rs1 != rs2) PC += imm	
blt	Branch <	B	1100011	0x4		if(rs1 < rs2) PC += imm	
bge	Branch ≤	B	1100011	0x5		if(rs1 >= rs2) PC += imm	
bltu	Branch < (U)	B	1100011	0x6		if(rs1 < rs2) PC += imm	zero-extends
bgeu	Branch ≥ (U)	B	1100011	0x7		if(rs1 >= rs2) PC += imm	zero-extends
jal	Jump And Link	J	1101111			rd = PC+4; PC += imm	
jalr	Jump And Link Reg	I	1100111	0x0		rd = PC+4; PC = rs1	
lui	Load Upper Imm	U	0110111			rd = imm << 12	
auipc	Add Upper Imm to PC	U	0010111			rd = PC + (imm << 12)	
ecall	Environment Call	I	1110011	0x0	0x00	Transfer control to OS	imm: 0x000
ebreak	Environment Break	I	1110011	0x0	0x00	Transfer control to debugger	imm: 0x001

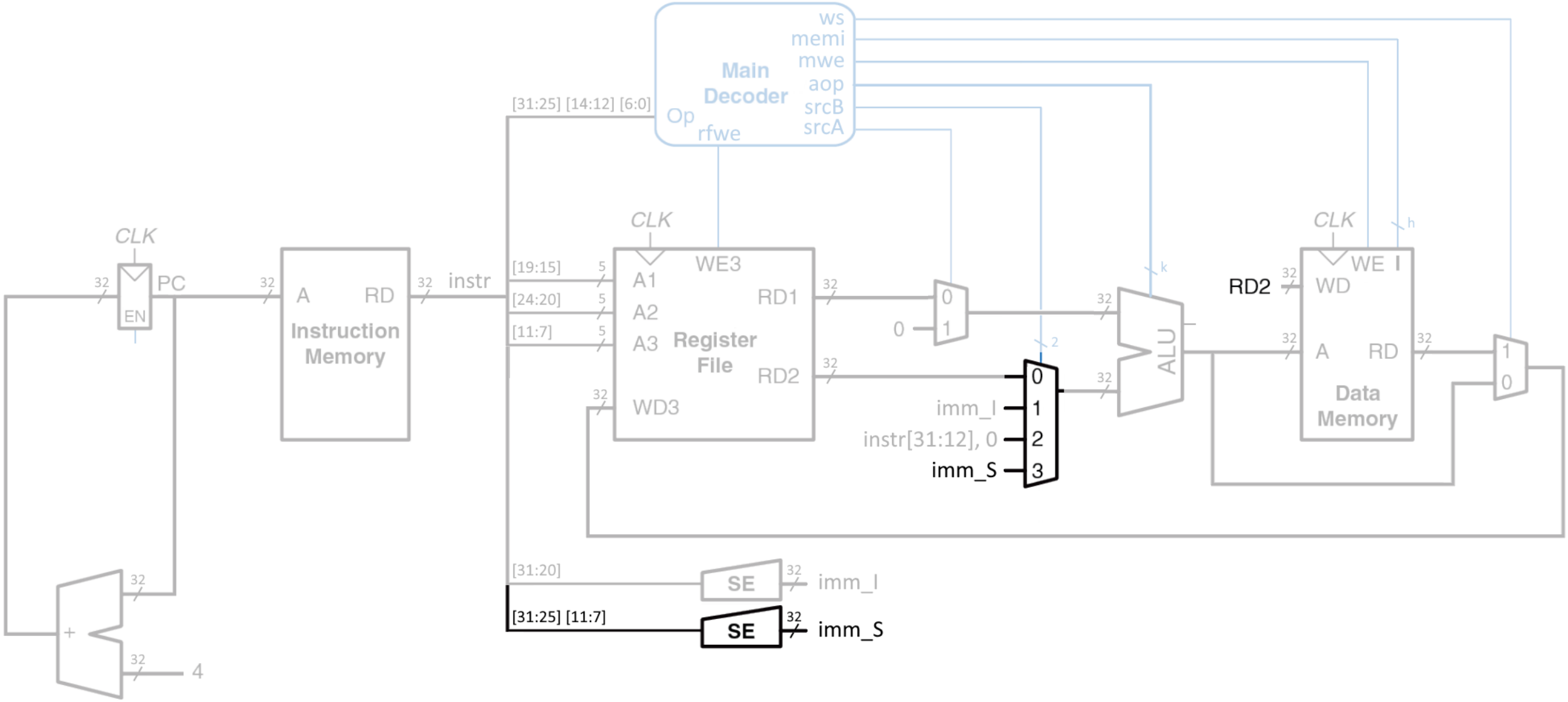
31 – 27	26 – 25	24 – 20	19 – 15	14 – 12	11 – 7	6 – 0	
imm[11:5]		rs2	rs1	funct3	imm[4:0]	opcode	S-type
imm[11:5]		rs2	rs1	010	imm[4:0]	0100011	SW

```
sw xN, offset(base)
M[rs1+imm][0:31] = rs2[0:31]
```



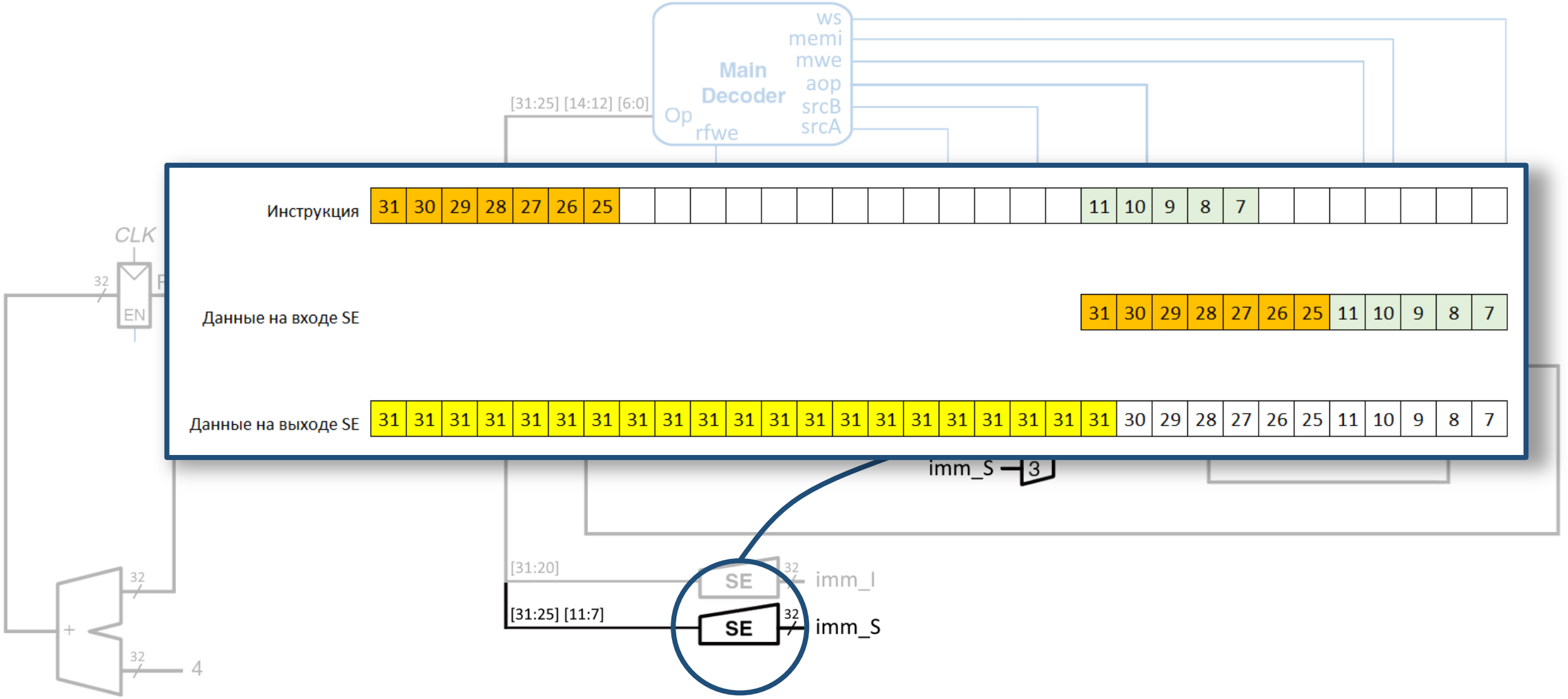
31 27	26 25	24 20	19 15	14 12	11 7	6 0	
imm[11:5]		rs2	rs1	funct3	imm[4:0]	opcode	S-type
imm[11:5]		rs2	rs1	010	imm[4:0]	0100011	SW

```
sw xN, offset(base)
M[rs1+imm][0:31] = rs2[0:31]
```



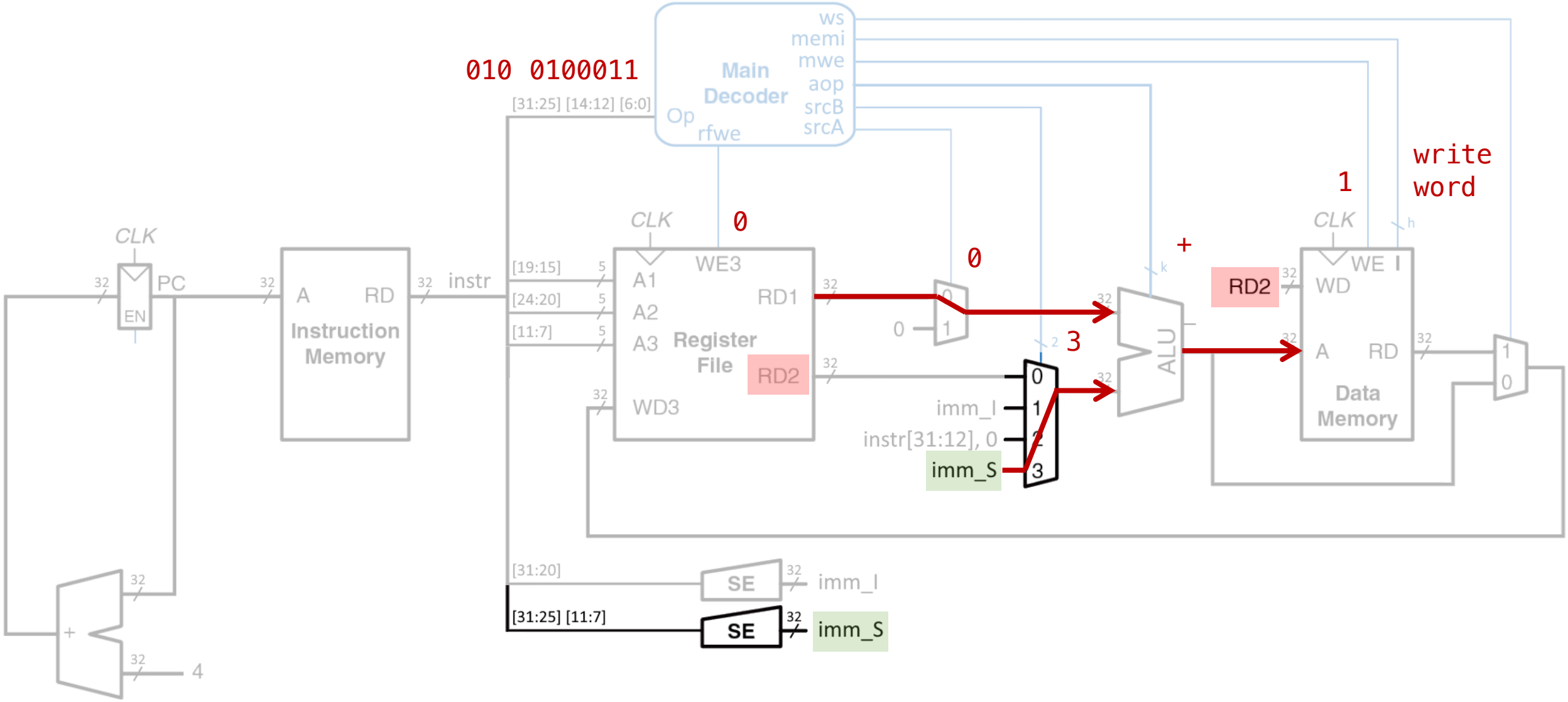
31 27 26 25	24 20	19 15	14 12	11 7	6 0	
imm[11:5]	rs2	rs1	funct3	imm[4:0]	opcode	S-type
imm[11:5]	rs2	rs1	010	imm[4:0]	0100011	SW

```
sw xN, offset(base)
M[rs1+imm][0:31] = rs2[0:31]
```



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
imm[11:5]		rs2	rs1	010	imm[4:0]	0100011	SW

```
sw xN, offset(base)
M[rs1+imm][0:31] = rs2[0:31]
```

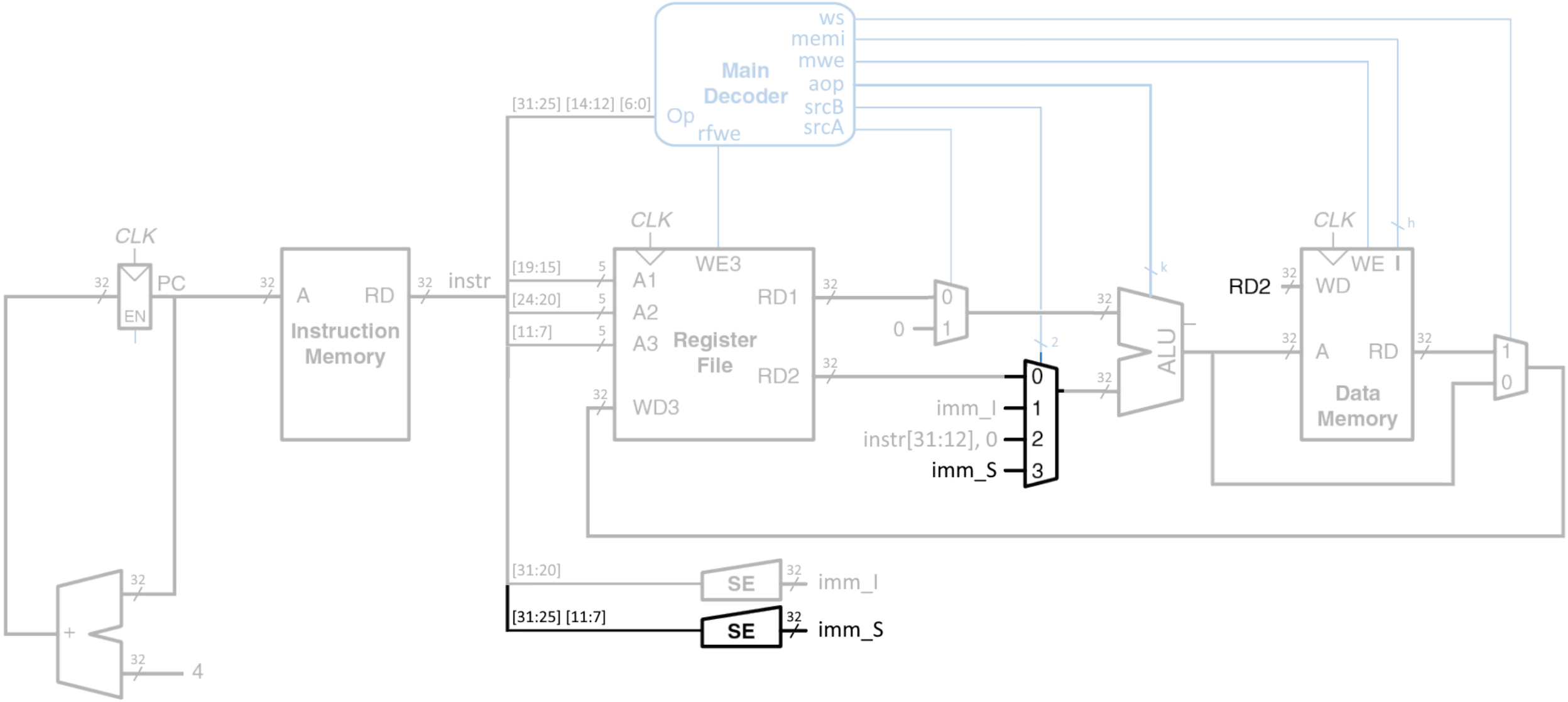


Набор инструкций RISC-V

Inst	Name	FMT	Opcode	F3	F7	Description (C)	Note
add	ADD	R	0000011	0x0	0x00	rd = rs1 + rs2	
sub	SUB	R	0000011	0x0	0x20	rd = rs1 - rs2	
xor	XOR	R	0000011	0x4	0x00	rd = rs1 ^ rs2	
or	OR	R	0000011	0x6	0x00	rd = rs1 | rs2	
and	AND	R	0000011	0x7	0x00	rd = rs1 & rs2	
sll	Shift Left Logical	R	0000011	0x1	0x00	rd = rs1 << rs2	
srl	Shift Right Logical	R	0000011	0x2	0x00	rd = rs1 >> rs2	
sra	Shift Right Arith*	R	0000011	0x3	0x20	rd = rs1 >> rs2	msb-extends
slt	Set Less Than	R	0110011	0x2		rd = (rs1 < rs2)?1:0	
sltu	Set Less Than (U)	R	0110011	0x3		rd = (rs1 < rs2)?1:0	zero-extends
addi	ADD Immediate	I	0010011	0x0	0x00	rd = rs1 + imm	
xori	XOR Immediate	I	0010011	0x0	0x00	rd = rs1 ^ imm	
ori	OR Immediate	I	0010011	0x0	0x00	rd = rs1 | imm	
andi	AND Immediate	I	0010011	0x0	0x00	rd = rs1 & imm	
slli	Shift Left Logical Imm	I	0010011	0x1	0x00	rd = rs1 << imm	
srli	Shift Right Logical Imm	I	0010011	0x1	0x00	rd = rs1 >> imm	
srai	Shift Right Arith Imm	I	0010011	0x3	0x20	rd = rs1 >> imm	msb-extends
slt	Set Less Than Imm	I	0010011	0x2		rd = (rs1 < imm)?1:0	
sltu	Set Less Than Imm (U)	I	0010011	0x3		rd = (rs1 < imm)?1:0	zero-extends
lb	Load Byte	I	0000011	0x0		rd = M[rs1+imm][0:7]	
lh	Load Half	I	0000011	0x1		rd = M[rs1+imm][0:15]	
lw	Load Word	I	0000011	0x2		rd = M[rs1+imm][0:31]	
lbu	Load Byte (U)	I	0000011	0x4		rd = M[rs1+imm][0:7]	zero-extends
lhu	Load Half (U)	I	0000011	0x5		rd = M[rs1+imm][0:15]	zero-extends
sb	Store Byte	S	0100011	0x0		M[rs1+imm][0:7] = rs2[0:7]	
sh	Store Half	S	0100011	0x1		M[rs1+imm][0:15] = rs2[0:15]	
sw	Store Word	S	0100011	0x2		M[rs1+imm][0:31] = rs2[0:31]	
beq	Branch ==	B	1100011	0x0		if(rs1 == rs2) PC += imm	
bne	Branch !=	B	1100011	0x1		if(rs1 != rs2) PC += imm	
blt	Branch <	B	1100011	0x4		if(rs1 < rs2) PC += imm	
bge	Branch ≤	B	1100011	0x5		if(rs1 >= rs2) PC += imm	
bltu	Branch < (U)	B	1100011	0x6		if(rs1 < rs2) PC += imm	zero-extends
bgeu	Branch ≥ (U)	B	1100011	0x7		if(rs1 >= rs2) PC += imm	zero-extends
jal	Jump And Link	J	1101111			rd = PC+4; PC += imm	
jalr	Jump And Link Reg	I	1100111	0x0		rd = PC+4; PC = rs1	
lui	Load Upper Imm	U	0110111			rd = imm << 12	
auipc	Add Upper Imm to PC	U	0010111			rd = PC + (imm << 12)	
ecall	Environment Call	I	1110011	0x0	0x00	Transfer control to OS	imm: 0x000
ebreak	Environment Break	I	1110011	0x0	0x00	Transfer control to debugger	imm: 0x001

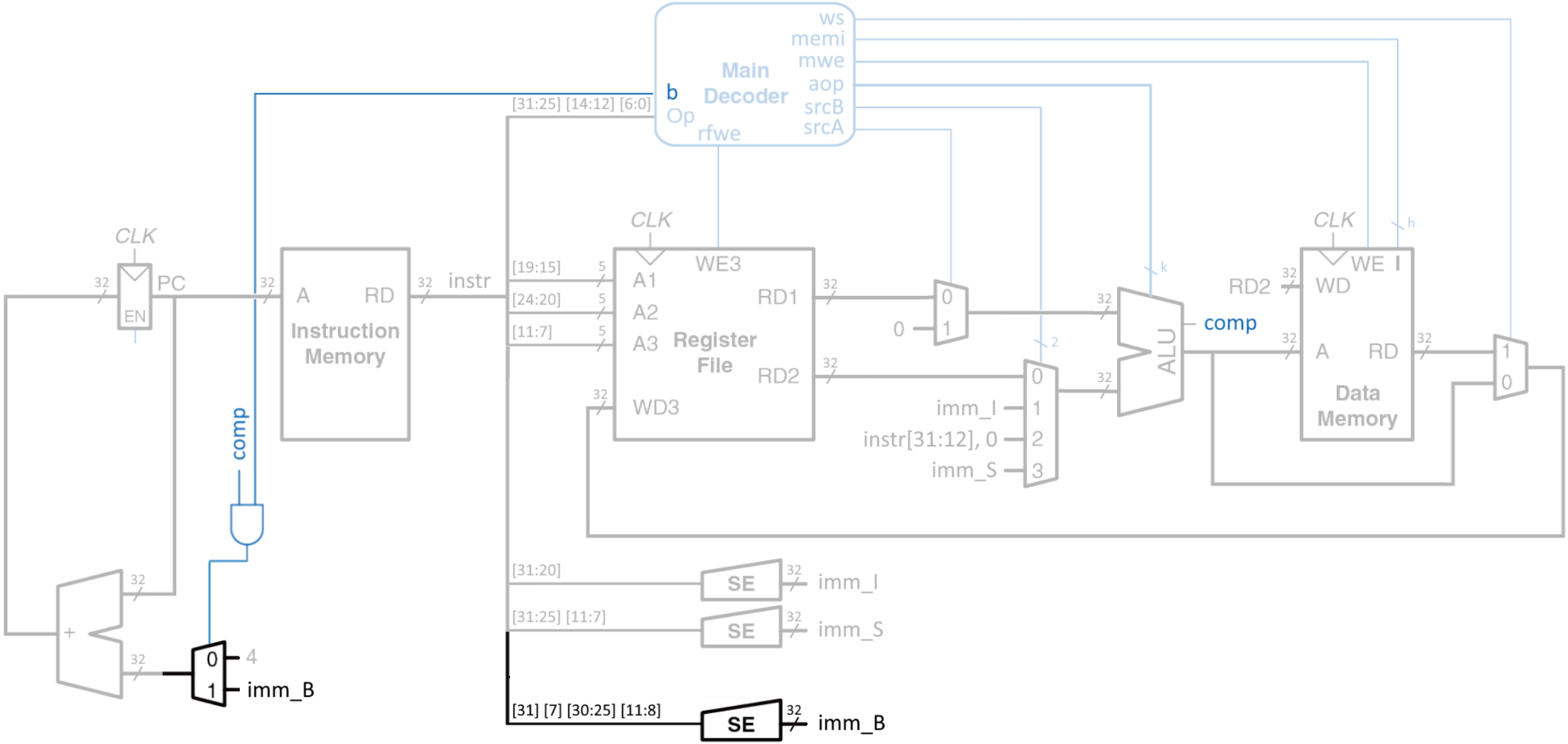
31 27 26 25	24 20	19 15	14 12	11 7	6 0	
imm[12|10:5]	rs2	rs1	funct3	imm[4:1|11]	opcode	B-type
imm[12|10:5]	rs2	rs1	000	imm[4:1|11]	1100011	BEQ

```
beq xN, xK, label
if(rs1 == rs2) PC += imm
```



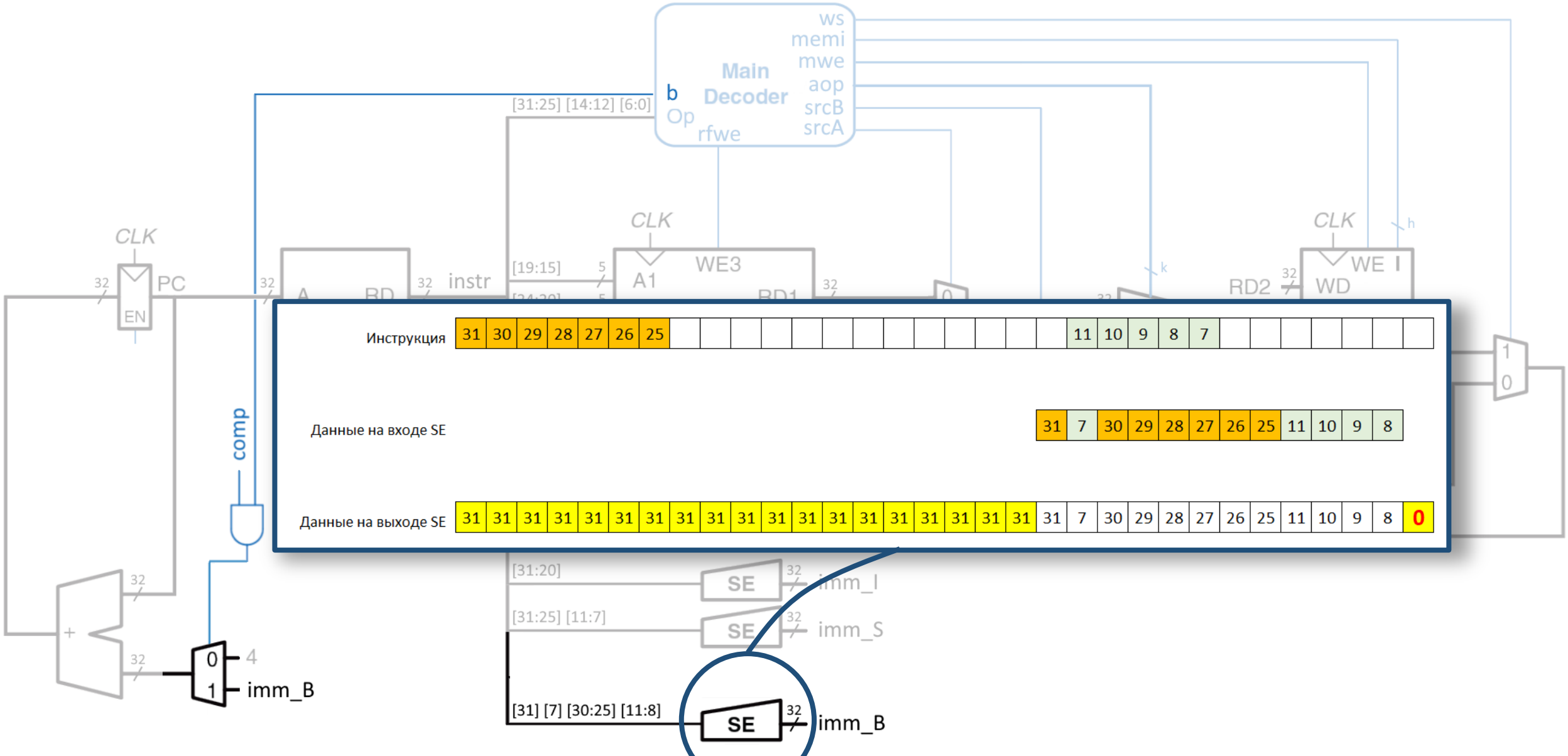
31 27	26 25	24 20	19 15	14 12	11 7	6 0	
imm[12|10:5]		rs2	rs1	funct3	imm[4:1|11]	opcode	B-type
imm[12|10:5]		rs2	rs1	000	imm[4:1|11]	1100011	BEQ

```
beq xN, xK, label
if(rs1 == rs2) PC += imm
```



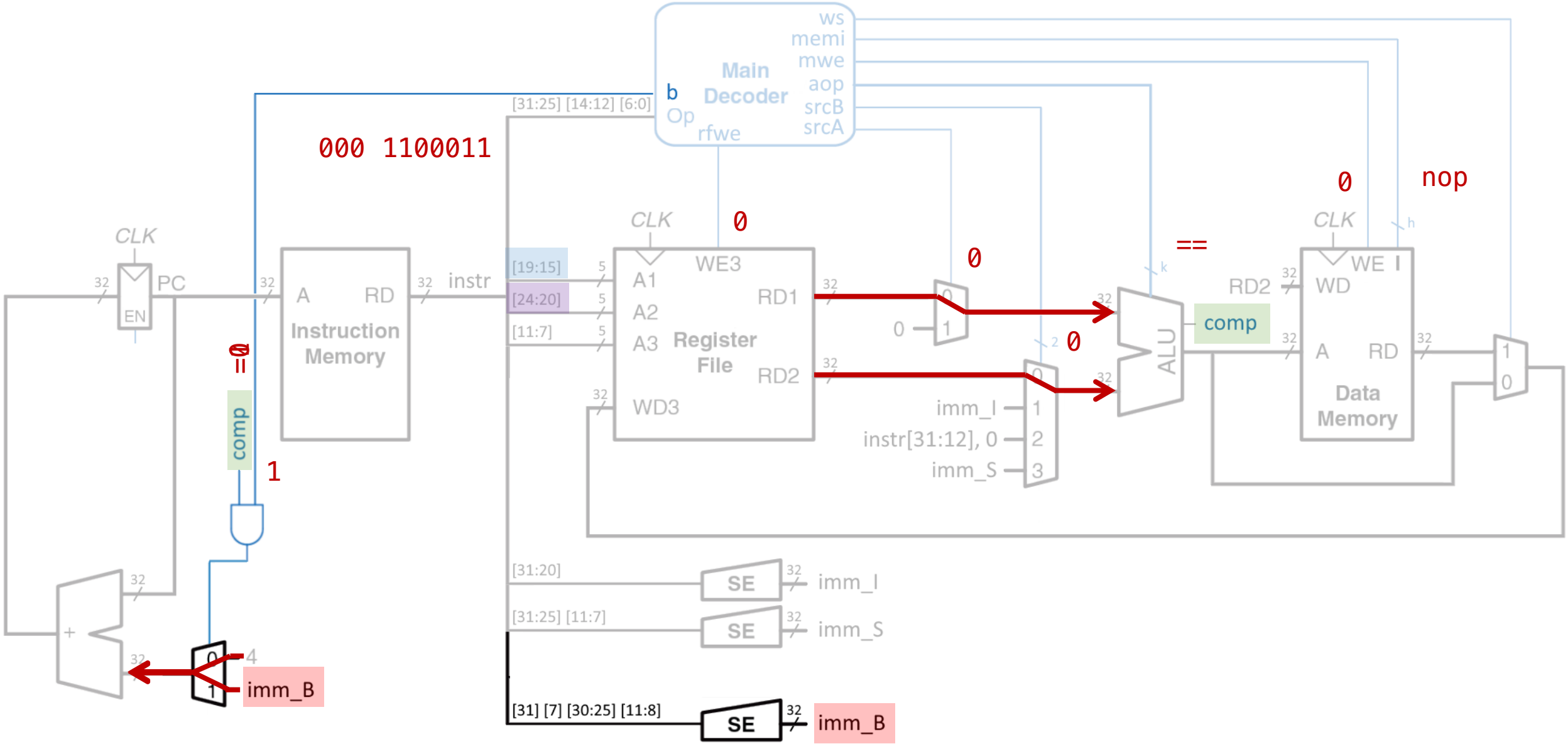
31 27	26 25	24 20	19 15	14 12	11 7	6 0	
imm[12|10:5]		rs2	rs1	funct3	imm[4:1|11]	opcode	B-type
imm[12|10:5]		rs2	rs1	000	imm[4:1|11]	1100011	BEQ

```
beq xN, xK, label
if(rs1 == rs2) PC += imm
```



31 27	26 25	24 20	19 15	14 12	11 7	6 0	
imm[12|10:5]		rs2	rs1	funct3	imm[4:1|11]	opcode	B-type
imm[12|10:5]		rs2	rs1	000	imm[4:1|11]	1100011	BEQ

```
beq xN, xK, label
if(rs1 == rs2) PC += imm
```

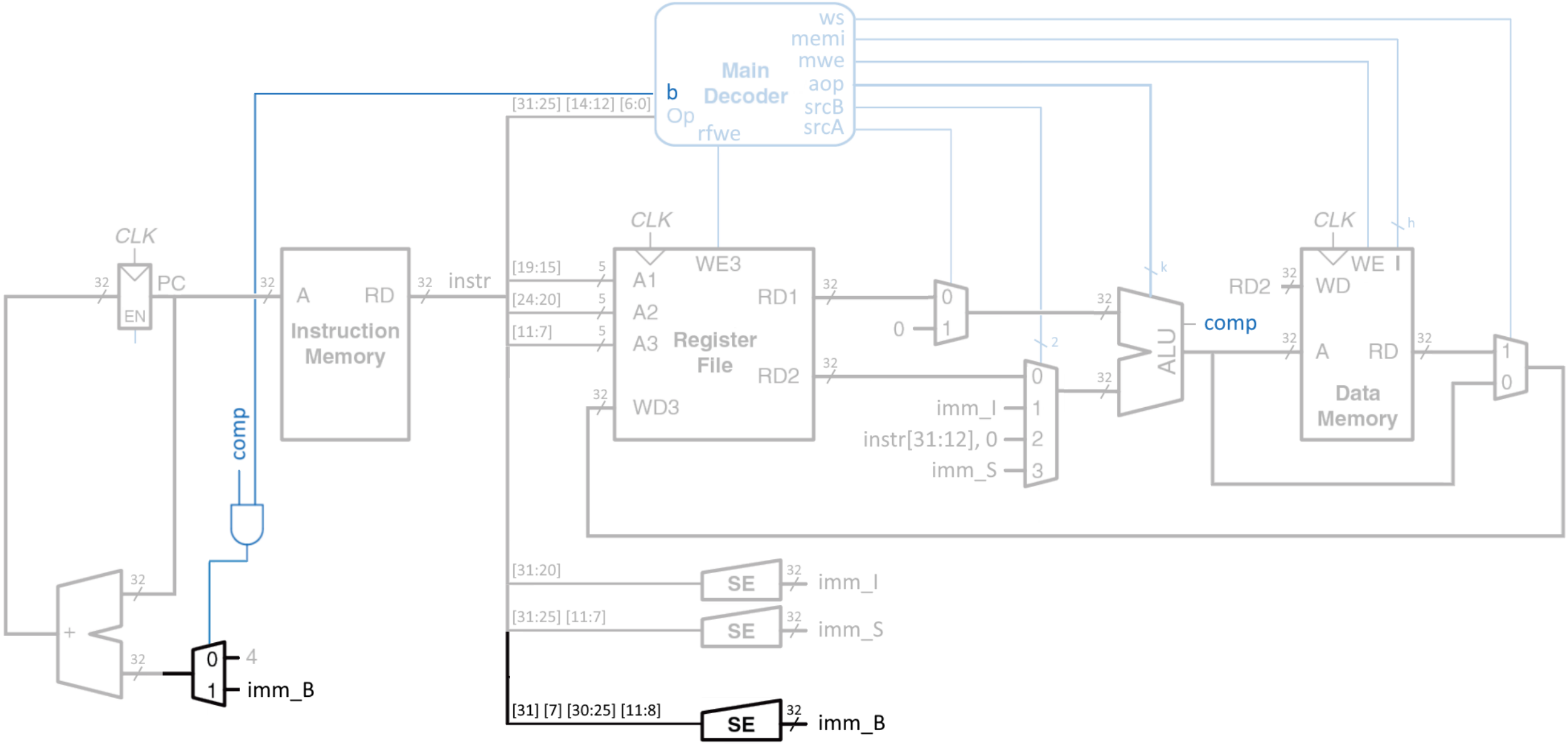


Набор инструкций RISC-V

Inst	Name	FMT	Opcode	F3	F7	Description (C)	Note
add	ADD	R	0000011	0x0	0x00	rd = rs1 + rs2	
sub	SUB	R	0000011	0x0	0x20	rd = rs1 - rs2	
xor	XOR	R	0000011	0x4	0x00	rd = rs1 ^ rs2	
or	OR	R	0000011	0x6	0x00	rd = rs1 | rs2	
and	AND	R	0000011	0x7	0x00	rd = rs1 & rs2	
sll	Shift Left Logical	R	0000011	0x1	0x00	rd = rs1 << rs2	
srl	Shift Right Logical	R	0000011	0x2	0x00	rd = rs1 >> rs2	
sra	Shift Right Arith*	R	0000011	0x3	0x20	rd = rs1 >> rs2	msb-extends
slt	Set Less Than	R	0110011	0x2		rd = (rs1 < rs2)?1:0	
sltu	Set Less Than (U)	R	0110011	0x3		rd = (rs1 < rs2)?1:0	zero-extends
addi	ADD Immediate	I	0010011	0x0	0x00	rd = rs1 + imm	
xori	XOR Immediate	I	0010011	0x0	0x00	rd = rs1 ^ imm	
ori	OR Immediate	I	0010011	0x0	0x00	rd = rs1 | imm	
andi	AND Immediate	I	0010011	0x0	0x00	rd = rs1 & imm	
slli	Shift Left Logical Imm	I	0010011	0x1	0x00	rd = rs1 << imm	
srli	Shift Right Logical Imm	I	0010011	0x1	0x00	rd = rs1 >> imm	
srai	Shift Right Arith Imm	I	0010011	0x3	0x20	rd = rs1 >> imm	msb-extends
slt	Set Less Than Imm	I	0010011	0x2		rd = (rs1 < imm)?1:0	
sltu	Set Less Than Imm (U)	I	0010011	0x3		rd = (rs1 < imm)?1:0	zero-extends
lb	Load Byte	I	0000011	0x0		rd = M[rs1+imm][0:7]	
lh	Load Half	I	0000011	0x1		rd = M[rs1+imm][0:15]	
lw	Load Word	I	0000011	0x2		rd = M[rs1+imm][0:31]	
lbu	Load Byte (U)	I	0000011	0x4		rd = M[rs1+imm][0:7]	zero-extends
lhu	Load Half (U)	I	0000011	0x5		rd = M[rs1+imm][0:15]	zero-extends
sb	Store Byte	S	0100011	0x0		M[rs1+imm][0:7] = rs2[0:7]	
sh	Store Half	S	0100011	0x1		M[rs1+imm][0:15] = rs2[0:15]	
sw	Store Word	S	0100011	0x2		M[rs1+imm][0:31] = rs2[0:31]	
beq	Branch ==	B	1100011	0x0		if(rs1 == rs2) PC += imm	
bne	Branch !=	B	1100011	0x1		if(rs1 != rs2) PC += imm	
blt	Branch <	B	1100011	0x4		if(rs1 < rs2) PC += imm	
bge	Branch ≤	B	1100011	0x5		if(rs1 >= rs2) PC += imm	
bltu	Branch < (U)	B	1100011	0x6		if(rs1 < rs2) PC += imm	zero-extends
bgeu	Branch ≥ (U)	B	1100011	0x7		if(rs1 >= rs2) PC += imm	zero-extends
jal	Jump And Link	J	1101111			rd = PC+4; PC += imm	
jalr	Jump And Link Reg	I	1100111	0x0		rd = PC+4; PC = rs1	
lui	Load Upper Imm	U	0110111			rd = imm << 12	
auipc	Add Upper Imm to PC	U	0010111			rd = PC + (imm << 12)	
ecall	Environment Call	I	1110011	0x0	0x00	Transfer control to OS	imm: 0x000
ebreak	Environment Break	I	1110011	0x0	0x00	Transfer control to debugger	imm: 0x001

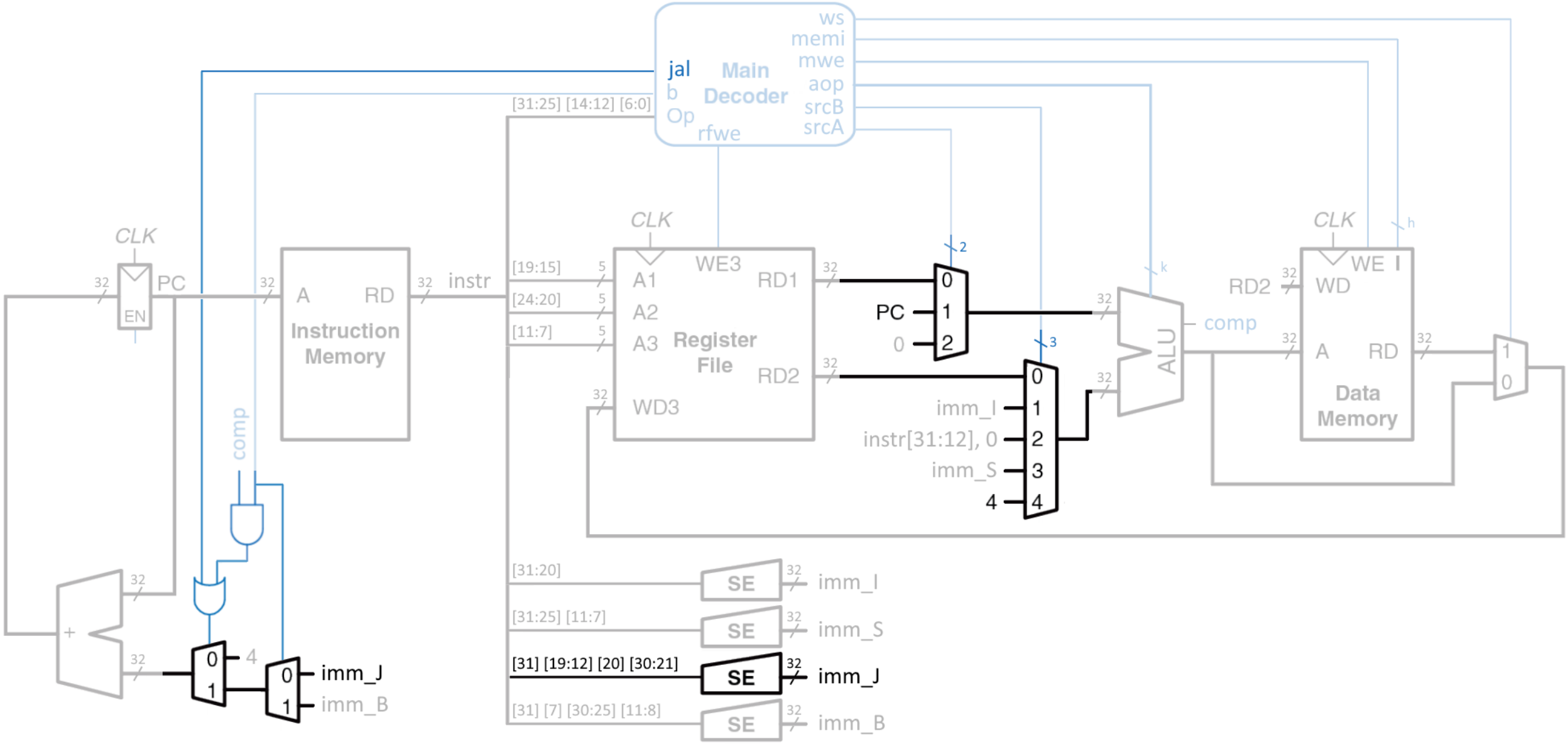
31 27	26 25 24 20	19 15	14 12	11 7	6 0	
imm[20|10:1|11|19:12]				rd	opcode	J-type
imm[20|10:1|11|19:12]				rd	1101111	JAL

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



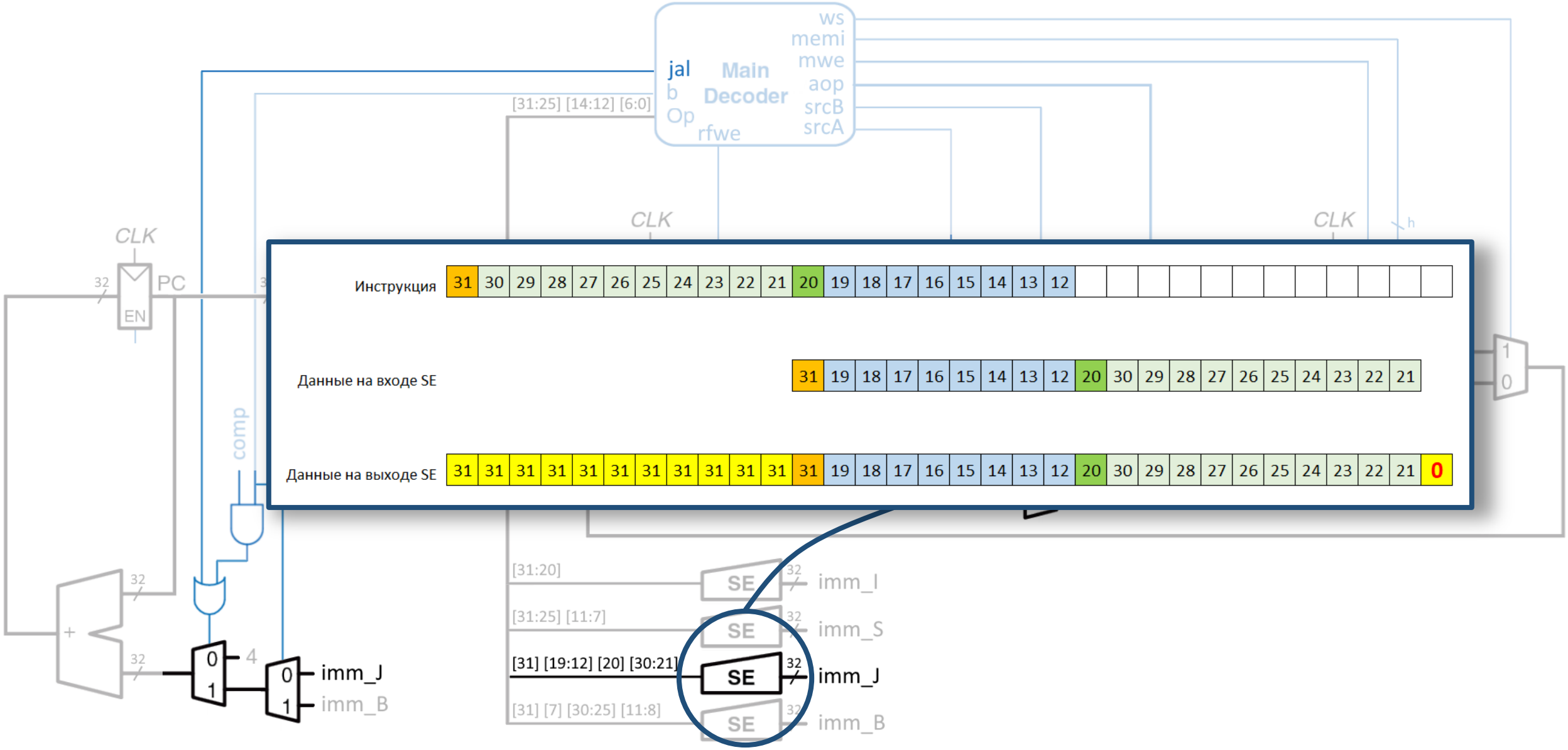
31 – 27	26 – 25	24 – 20	19 – 15	14 – 12	11 – 7	6 – 0	
imm[20|10:1|11|19:12]					rd	opcode	J-type
imm[20|10:1|11|19:12]					rd	1101111	JAL

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



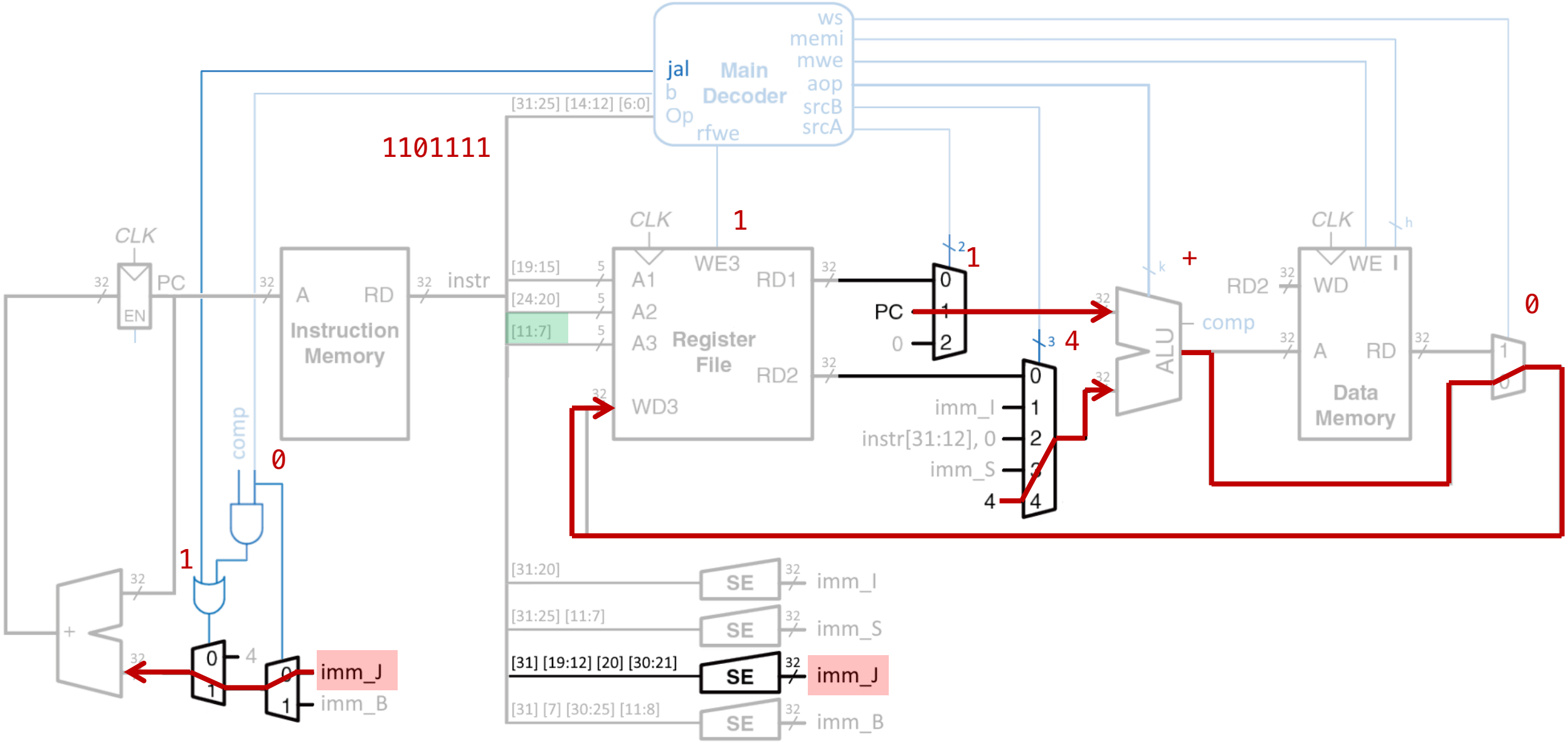
31 27	26 25 24 20	19 15	14 12	11 7	6 0	
imm[20|10:1|11|19:12]				rd	opcode	J-type
imm[20|10:1|11|19:12]				rd	1101111	JAL

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



31 – 27	26 – 25	24 – 20	19 – 15	14 – 12	11 – 7	6 – 0	
imm[20|10:1|11|19:12]					rd	opcode	J-type
imm[20|10:1|11|19:12]					rd	1101111	JAL

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

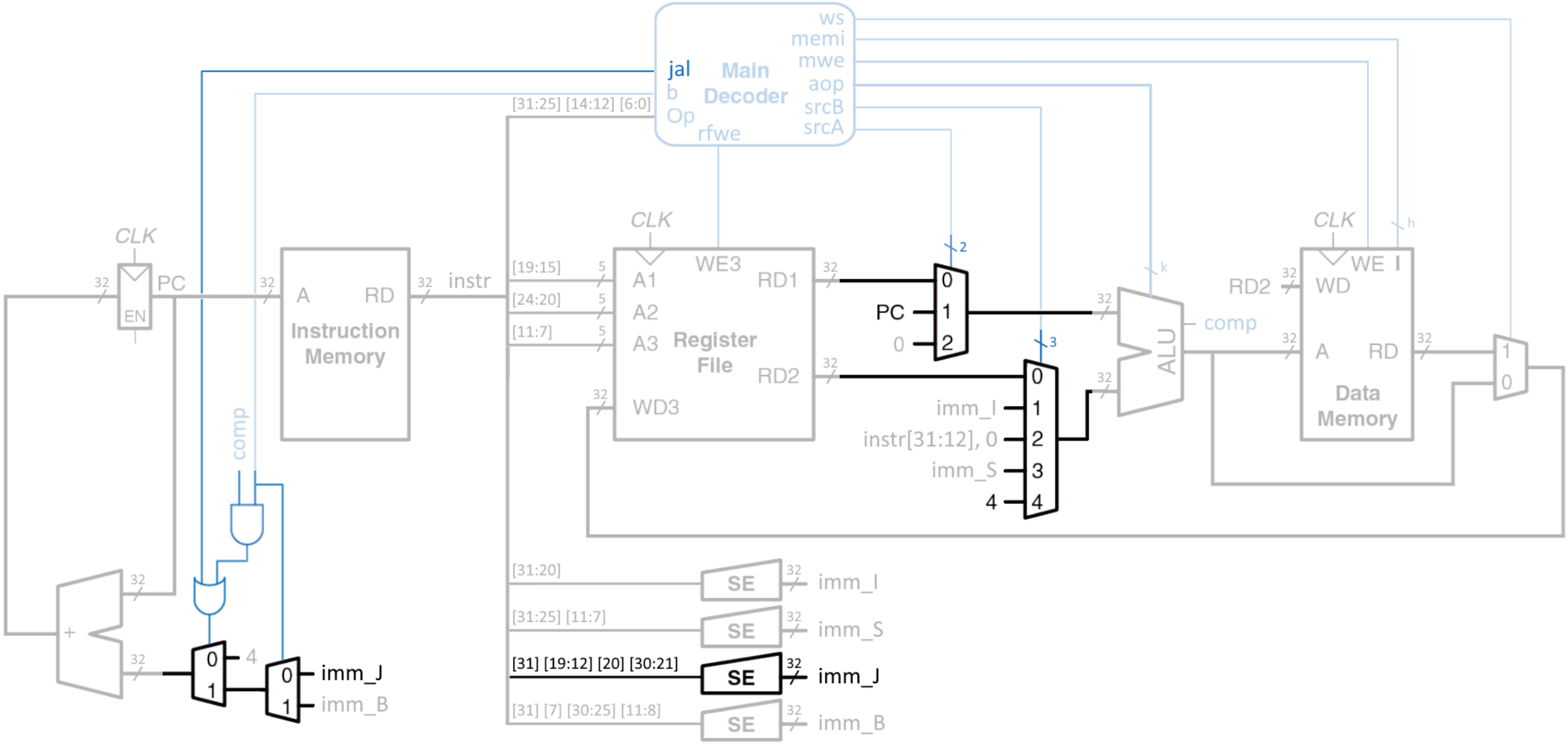


Набор инструкций RISC-V

Inst	Name	FMT	Opcode	F3	F7	Description (C)	Note
add	ADD	R	0000011	0x0	0x00	rd = rs1 + rs2	
sub	SUB	R	0000011	0x0	0x20	rd = rs1 - rs2	
xor	XOR	R	0000011	0x4	0x00	rd = rs1 ^ rs2	
or	OR	R	0000011	0x6	0x00	rd = rs1 | rs2	
and	AND	R	0000011	0x7	0x00	rd = rs1 & rs2	
sll	Shift Left Logical	R	0000011	0x1	0x00	rd = rs1 << rs2	
srl	Shift Right Logical	R	0000011	0x2	0x00	rd = rs1 >> rs2	
sra	Shift Right Arith*	R	0000011	0x3	0x20	rd = rs1 >> rs2	msb-extends
slt	Set Less Than	R	0110011	0x2		rd = (rs1 < rs2)?1:0	
sltu	Set Less Than (U)	R	0110011	0x3		rd = (rs1 < rs2)?1:0	zero-extends
addi	ADD Immediate	I	0010011	0x0	0x00	rd = rs1 + imm	
xori	XOR Immediate	I	0010011	0x0	0x00	rd = rs1 ^ imm	
ori	OR Immediate	I	0010011	0x0	0x00	rd = rs1 | imm	
andi	AND Immediate	I	0010011	0x0	0x00	rd = rs1 & imm	
slli	Shift Left Logical Imm	I	0010011	0x1	0x00	rd = rs1 << imm	
srli	Shift Right Logical Imm	I	0010011	0x1	0x00	rd = rs1 >> imm	
srai	Shift Right Arith Imm	I	0010011	0x3	0x20	rd = rs1 >> imm	msb-extends
slt	Set Less Than Imm	I	0010011	0x2		rd = (rs1 < imm)?1:0	
sltu	Set Less Than Imm (U)	I	0010011	0x3		rd = (rs1 < imm)?1:0	zero-extends
lb	Load Byte	I	0000011	0x0		rd = M[rs1+imm][0:7]	
lh	Load Half	I	0000011	0x1		rd = M[rs1+imm][0:15]	
lw	Load Word	I	0000011	0x2		rd = M[rs1+imm][0:31]	
lbu	Load Byte (U)	I	0000011	0x4		rd = M[rs1+imm][0:7]	zero-extends
lhu	Load Half (U)	I	0000011	0x5		rd = M[rs1+imm][0:15]	zero-extends
sb	Store Byte	S	0100011	0x0		M[rs1+imm][0:7] = rs2[0:7]	
sh	Store Half	S	0100011	0x1		M[rs1+imm][0:15] = rs2[0:15]	
sw	Store Word	S	0100011	0x2		M[rs1+imm][0:31] = rs2[0:31]	
beq	Branch ==	B	1100011	0x0		if(rs1 == rs2) PC += imm	
bne	Branch !=	B	1100011	0x1		if(rs1 != rs2) PC += imm	
blt	Branch <	B	1100011	0x4		if(rs1 < rs2) PC += imm	
bge	Branch ≤	B	1100011	0x5		if(rs1 >= rs2) PC += imm	
bltu	Branch < (U)	B	1100011	0x6		if(rs1 < rs2) PC += imm	zero-extends
bgeu	Branch ≥ (U)	B	1100011	0x7		if(rs1 >= rs2) PC += imm	zero-extends
jal	Jump And Link	J	1101111			rd = PC+4; PC += imm	
jalr	Jump And Link Reg	I	1100111	0x0		rd = PC+4; PC = rs1	
lui	Load Upper Imm	U	0110111			rd = imm << 12	
auipc	Add Upper Imm to PC	U	0010111			rd = PC + (imm << 12)	
ecall	Environment Call	I	1110011	0x0	0x00	Transfer control to OS	imm: 0x000
ebreak	Environment Break	I	1110011	0x0	0x00	Transfer control to debugger	imm: 0x001

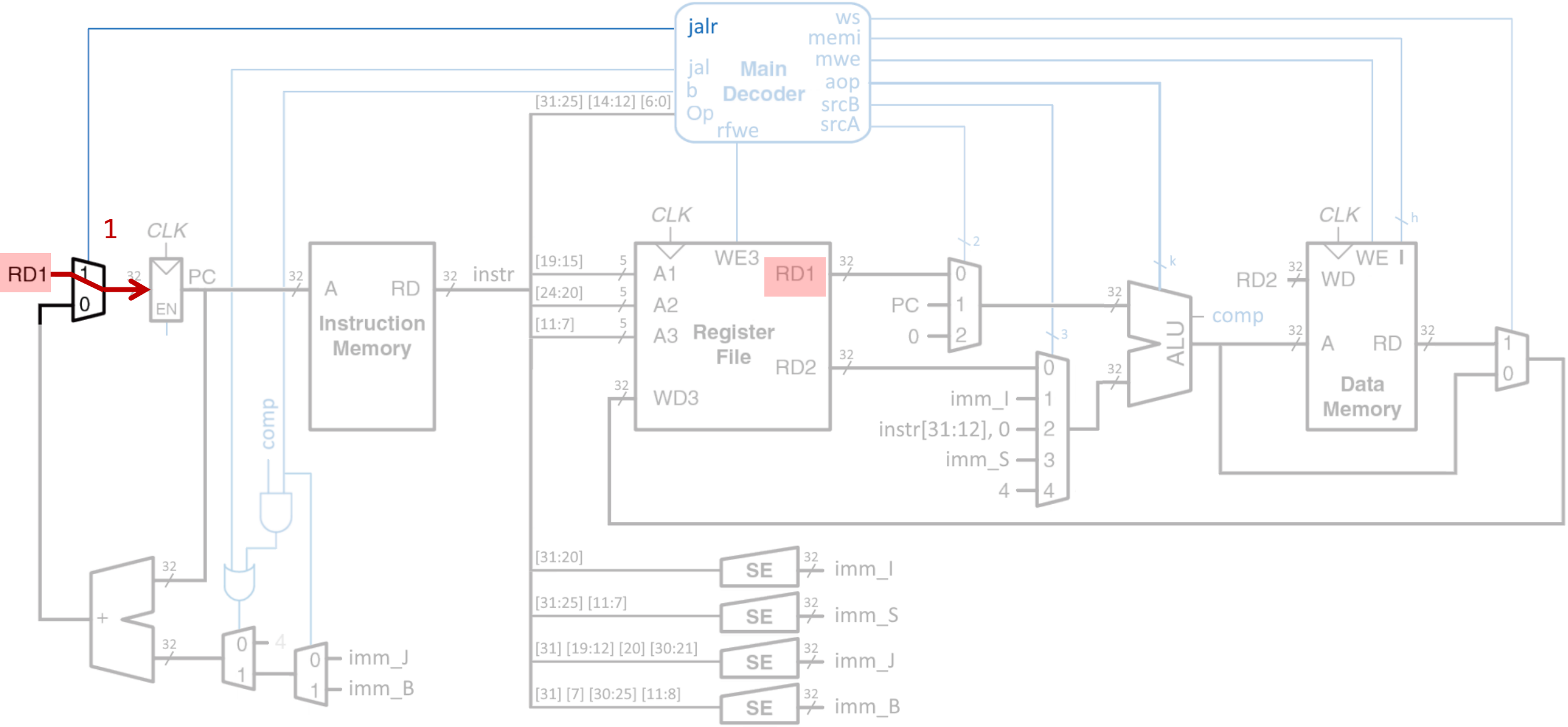
31 – 27	26 – 25	24 – 20	19 – 15	14 – 12	11 – 7	6 – 0	
imm[11:0]			rs1	funct3	rd	opcode	I-type
imm[11:0]			rs1	000	rd	1100111	JALR

```
jalr xN, xK
rd = PC+4; PC = rs1
```



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	000	rd	1100111	JALR

```
jalr xN, xK
rd = PC+4; PC = rs1
```

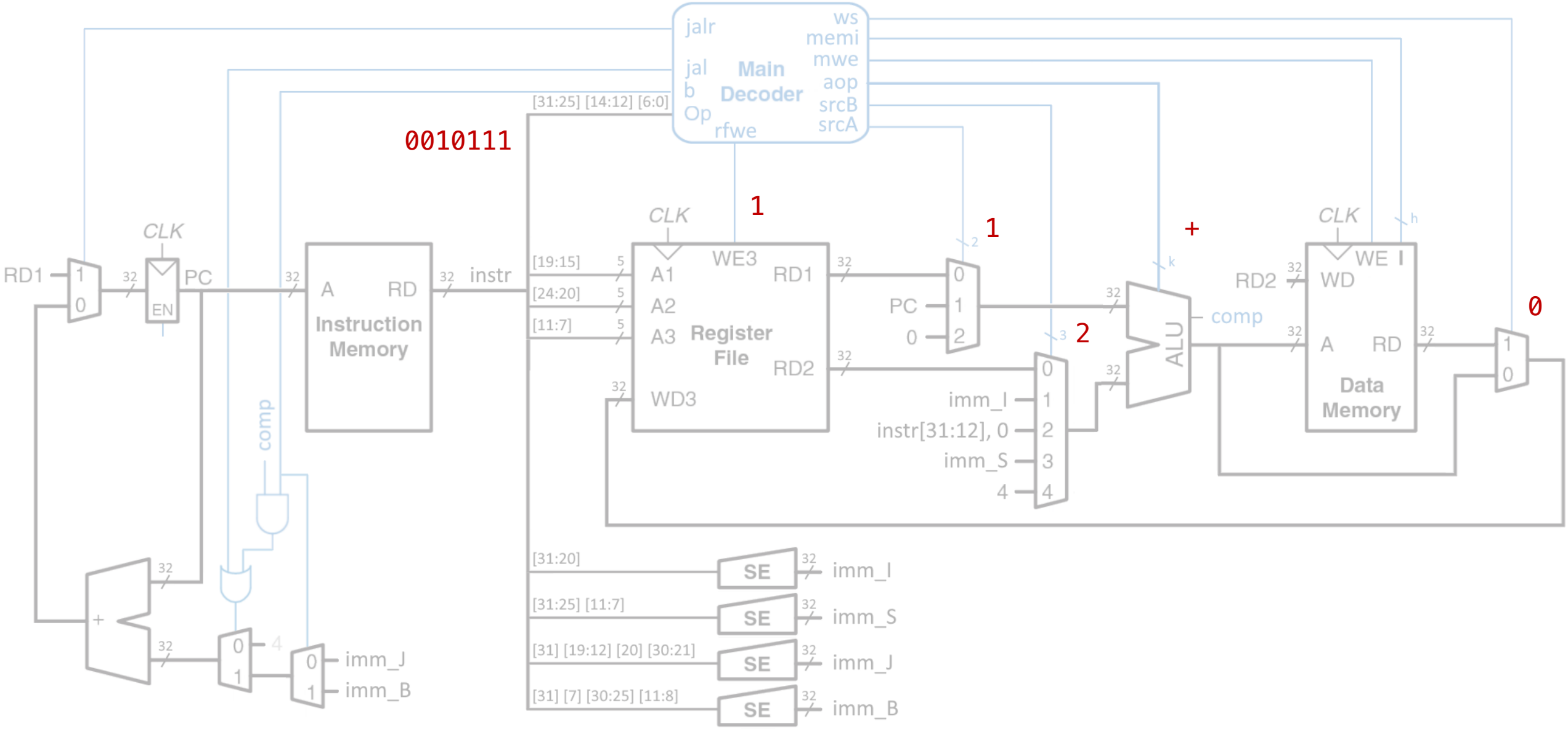


Набор инструкций RISC-V

Inst	Name	FMT	Opcode	F3	F7	Description (C)	Note
add	ADD	R	0000011	0x0	0x00	rd = rs1 + rs2	
sub	SUB	R	0000011	0x0	0x20	rd = rs1 - rs2	
xor	XOR	R	0000011	0x4	0x00	rd = rs1 ^ rs2	
or	OR	R	0000011	0x6	0x00	rd = rs1 | rs2	
and	AND	R	0000011	0x7	0x00	rd = rs1 & rs2	
sll	Shift Left Logical	R	0000011	0x1	0x00	rd = rs1 << rs2	
srl	Shift Right Logical	R	0000011	0x2	0x00	rd = rs1 >> rs2	
sra	Shift Right Arith*	R	0000011	0x3	0x20	rd = rs1 >> rs2	msb-extends
slt	Set Less Than	R	0110011	0x2		rd = (rs1 < rs2)?1:0	
sltu	Set Less Than (U)	R	0110011	0x3		rd = (rs1 < rs2)?1:0	zero-extends
addi	ADD Immediate	I	0010011	0x0	0x00	rd = rs1 + imm	
xori	XOR Immediate	I	0010011	0x0	0x00	rd = rs1 ^ imm	
ori	OR Immediate	I	0010011	0x0	0x00	rd = rs1 | imm	
andi	AND Immediate	I	0010011	0x0	0x00	rd = rs1 & imm	
slli	Shift Left Logical Imm	I	0010011	0x1	0x00	rd = rs1 << imm	
srli	Shift Right Logical Imm	I	0010011	0x1	0x00	rd = rs1 >> imm	
srai	Shift Right Arith Imm	I	0010011	0x3	0x20	rd = rs1 >> imm	msb-extends
slt	Set Less Than Imm	I	0010011	0x2		rd = (rs1 < imm)?1:0	
sltu	Set Less Than Imm (U)	I	0010011	0x3		rd = (rs1 < imm)?1:0	zero-extends
lb	Load Byte	I	0000011	0x0		rd = M[rs1+imm][0:7]	
lh	Load Half	I	0000011	0x1		rd = M[rs1+imm][0:15]	
lw	Load Word	I	0000011	0x2		rd = M[rs1+imm][0:31]	
lbu	Load Byte (U)	I	0000011	0x4		rd = M[rs1+imm][0:7]	zero-extends
lhu	Load Half (U)	I	0000011	0x5		rd = M[rs1+imm][0:15]	zero-extends
sb	Store Byte	S	0100011	0x0		M[rs1+imm][0:7] = rs2[0:7]	
sh	Store Half	S	0100011	0x1		M[rs1+imm][0:15] = rs2[0:15]	
sw	Store Word	S	0100011	0x2		M[rs1+imm][0:31] = rs2[0:31]	
beq	Branch ==	B	1100011	0x0		if(rs1 == rs2) PC += imm	
bne	Branch !=	B	1100011	0x1		if(rs1 != rs2) PC += imm	
blt	Branch <	B	1100011	0x4		if(rs1 < rs2) PC += imm	
bge	Branch ≤	B	1100011	0x5		if(rs1 >= rs2) PC += imm	
bltu	Branch < (U)	B	1100011	0x6		if(rs1 < rs2) PC += imm	zero-extends
bgeu	Branch ≥ (U)	B	1100011	0x7		if(rs1 >= rs2) PC += imm	zero-extends
jal	Jump And Link	J	1101111			rd = PC+4; PC += imm	
jalr	Jump And Link Reg	I	1100111	0x0		rd = PC+4; PC = rs1	
lui	Load Upper Imm	U	0110111			rd = imm << 12	
auipc	Add Upper Imm to PC	U	0010111			rd = PC + (imm << 12)	
ecall	Environment Call	I	1110011	0x0	0x00	Transfer control to OS	imm: 0x000
ebreak	Environment Break	I	1110011	0x0	0x00	Transfer control to debugger	imm: 0x001

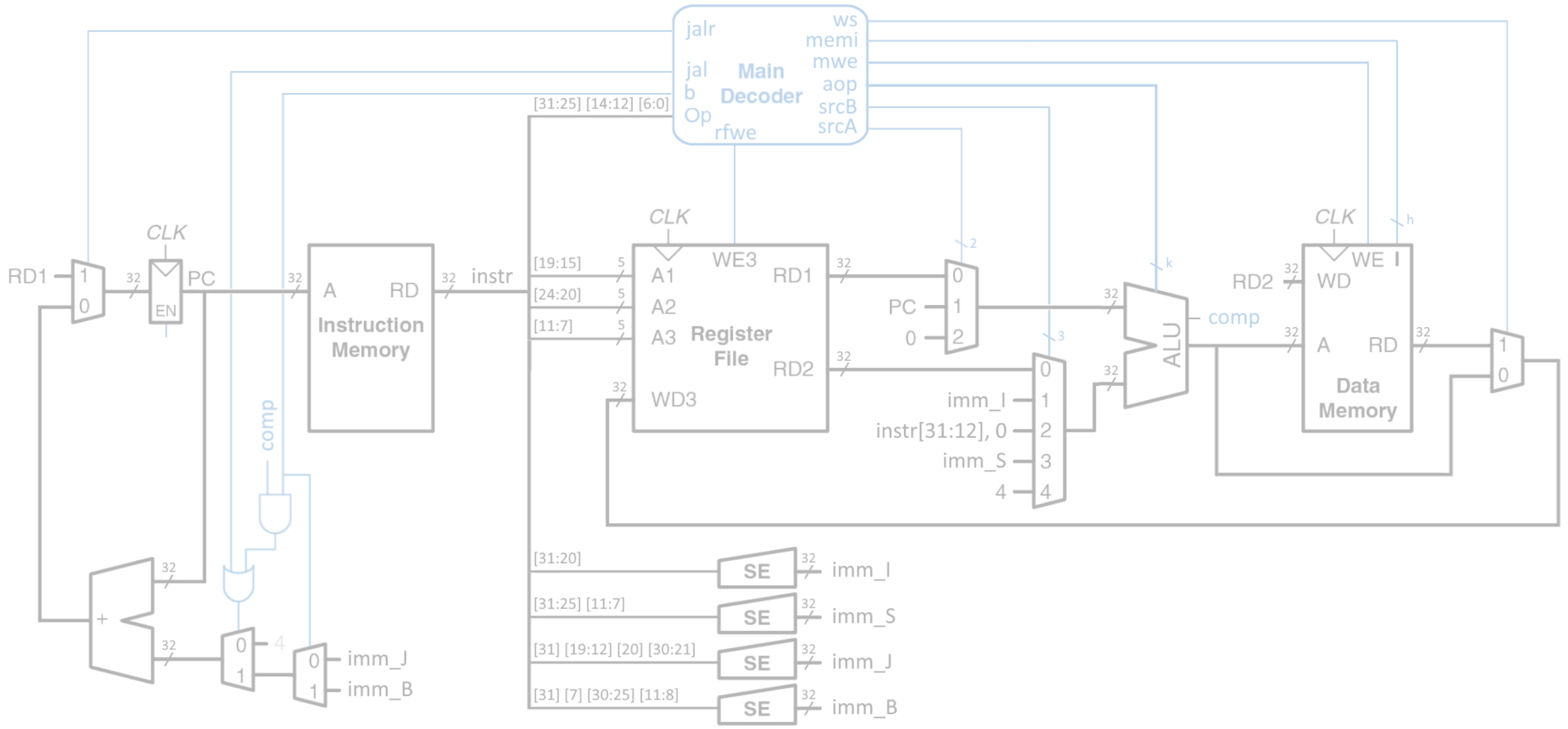
31 27	26 25	24 20	19 15	14 12	11 7	6 0	
imm[31:12]					rd	opcode	U-type
imm[31:12]					rd	0010111	AUIPC

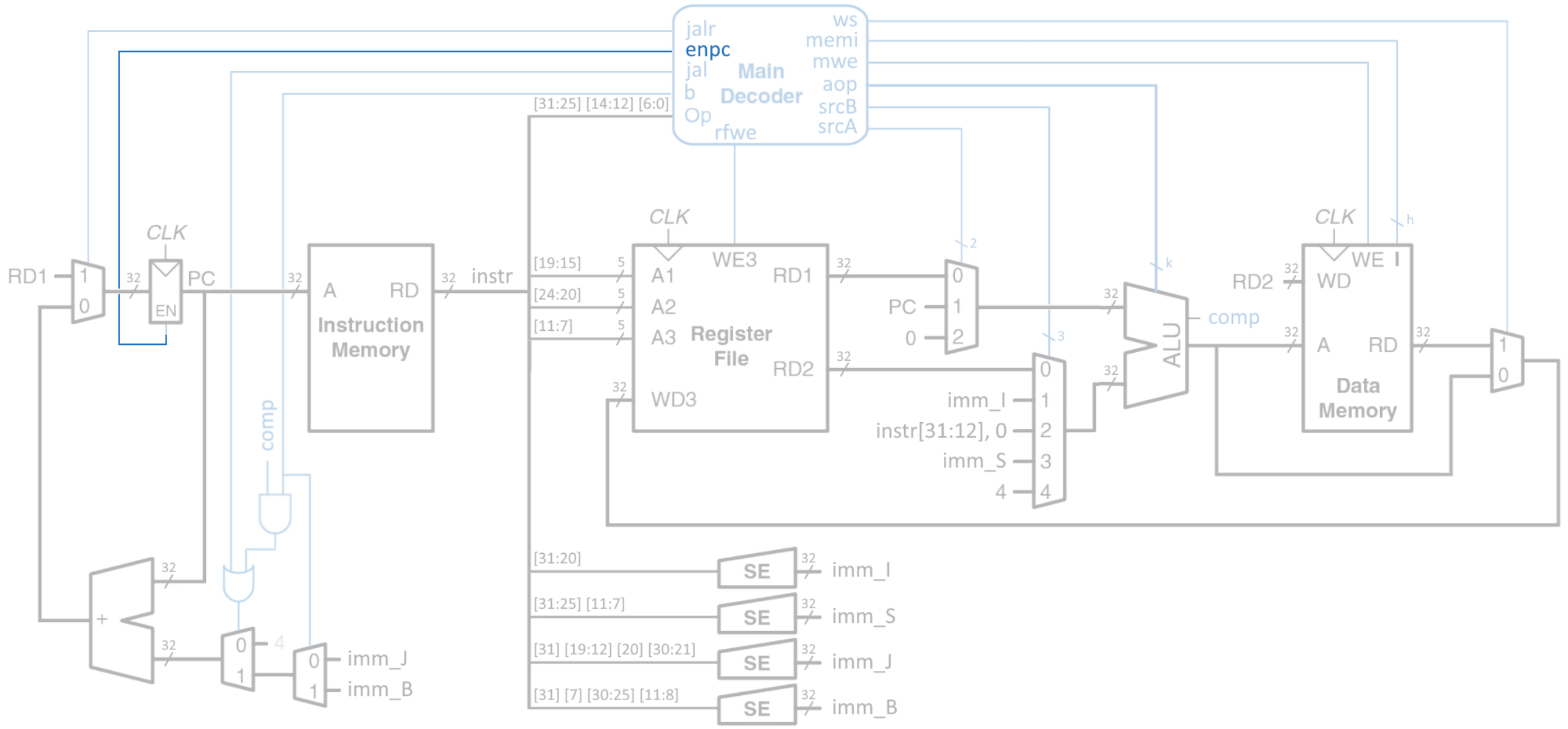
```
auipc xN, label
rd = PC + (imm << 12)
```

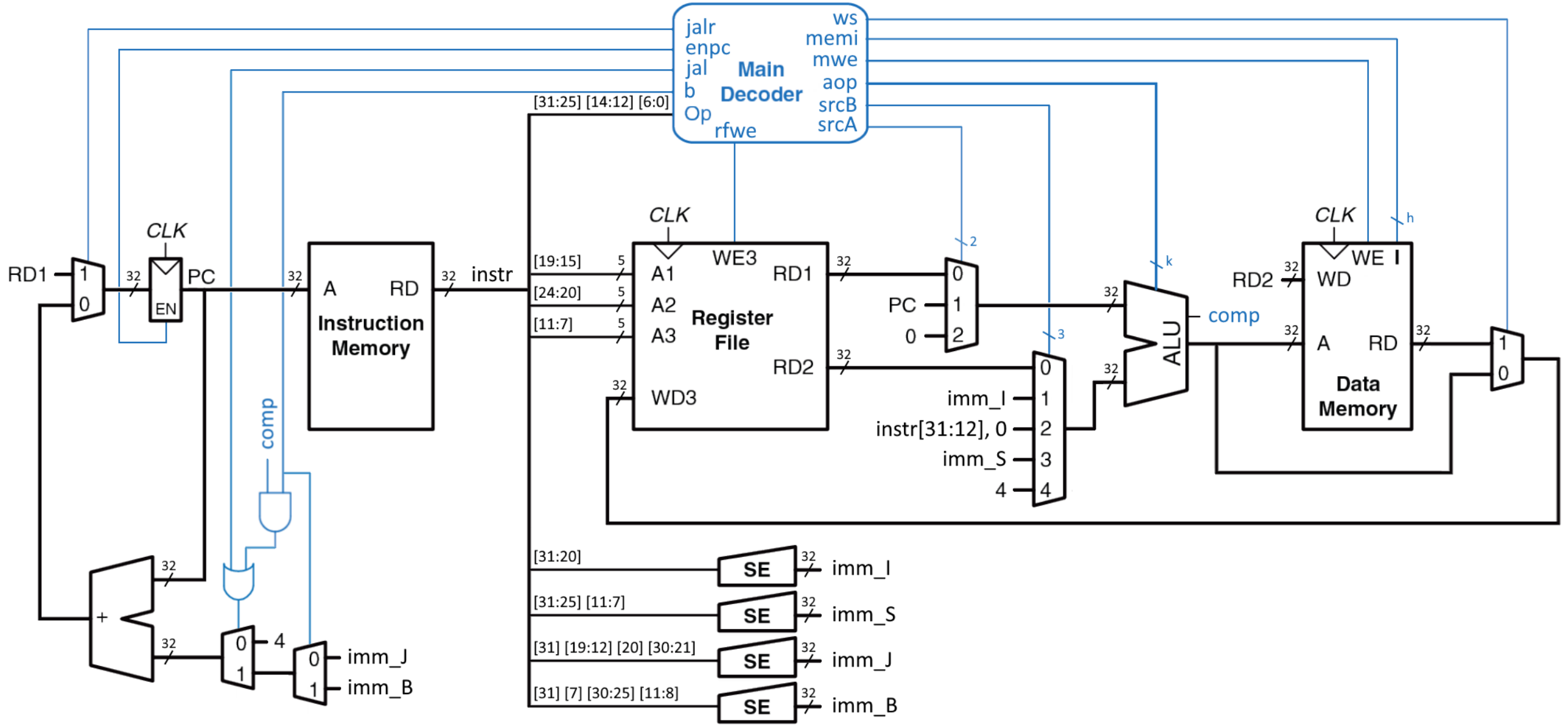


Набор инструкций RISC-V

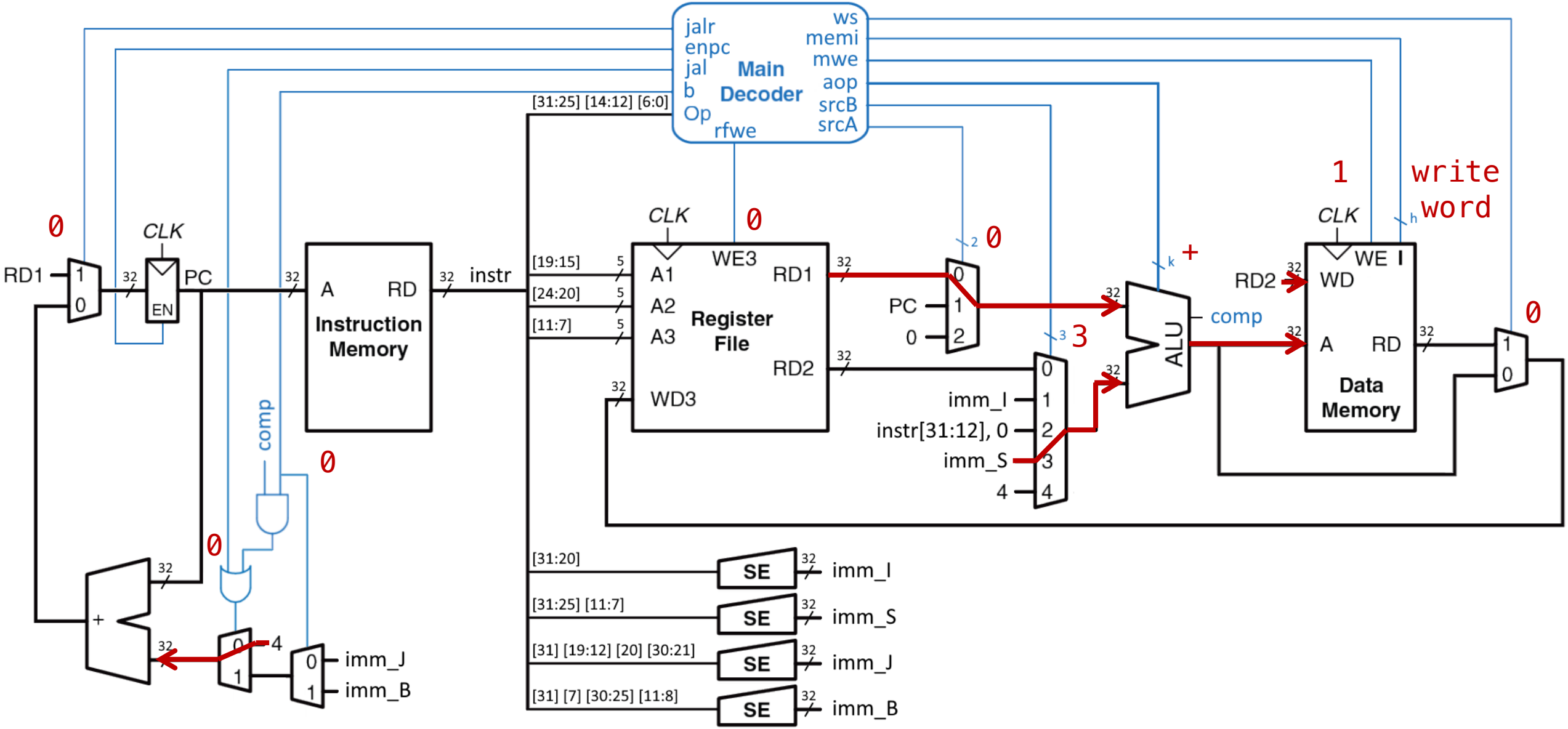
Inst	Name	FMT	Opcode	F3	F7	Description (C)	Note
add	ADD	R	0000011	0x0	0x00	rd = rs1 + rs2	
sub	SUB	R	0000011	0x0	0x20	rd = rs1 - rs2	
xor	XOR	R	0000011	0x4	0x00	rd = rs1 ^ rs2	
or	OR	R	0000011	0x6	0x00	rd = rs1 | rs2	
and	AND	R	0000011	0x7	0x00	rd = rs1 & rs2	
sll	Shift Left Logical	R	0000011	0x1	0x00	rd = rs1 << rs2	
srl	Shift Right Logical	R	0000011	0x2	0x00	rd = rs1 >> rs2	
sra	Shift Right Arith*	R	0000011	0x3	0x20	rd = rs1 >> rs2	msb-extends
slt	Set Less Than	R	0110011	0x2		rd = (rs1 < rs2)?1:0	
sltu	Set Less Than (U)	R	0110011	0x3		rd = (rs1 < rs2)?1:0	zero-extends
addi	ADD Immediate	I	0010011	0x0	0x00	rd = rs1 + imm	
xori	XOR Immediate	I	0010011	0x0	0x00	rd = rs1 ^ imm	
ori	OR Immediate	I	0010011	0x0	0x00	rd = rs1 | imm	
andi	AND Immediate	I	0010011	0x0	0x00	rd = rs1 & imm	
slli	Shift Left Logical Imm	I	0010011	0x1	0x00	rd = rs1 << imm	
srli	Shift Right Logical Imm	I	0010011	0x1	0x00	rd = rs1 >> imm	
srai	Shift Right Arith Imm	I	0010011	0x3	0x20	rd = rs1 >> imm	msb-extends
slt	Set Less Than Imm	I	0010011	0x2		rd = (rs1 < imm)?1:0	
sltu	Set Less Than Imm (U)	I	0010011	0x3		rd = (rs1 < imm)?1:0	zero-extends
lb	Load Byte	I	0000011	0x0		rd = M[rs1+imm][0:7]	
lh	Load Half	I	0000011	0x1		rd = M[rs1+imm][0:15]	
lw	Load Word	I	0000011	0x2		rd = M[rs1+imm][0:31]	
lbu	Load Byte (U)	I	0000011	0x4		rd = M[rs1+imm][0:7]	zero-extends
lhu	Load Half (U)	I	0000011	0x5		rd = M[rs1+imm][0:15]	zero-extends
sb	Store Byte	S	0100011	0x0		M[rs1+imm][0:7] = rs2[0:7]	
sh	Store Half	S	0100011	0x1		M[rs1+imm][0:15] = rs2[0:15]	
sw	Store Word	S	0100011	0x2		M[rs1+imm][0:31] = rs2[0:31]	
beq	Branch ==	B	1100011	0x0		if(rs1 == rs2) PC += imm	
bne	Branch !=	B	1100011	0x1		if(rs1 != rs2) PC += imm	
blt	Branch <	B	1100011	0x4		if(rs1 < rs2) PC += imm	
bge	Branch ≤	B	1100011	0x5		if(rs1 >= rs2) PC += imm	
bltu	Branch < (U)	B	1100011	0x6		if(rs1 < rs2) PC += imm	zero-extends
bgeu	Branch ≥ (U)	B	1100011	0x7		if(rs1 >= rs2) PC += imm	zero-extends
jal	Jump And Link	J	1101111			rd = PC+4; PC += imm	
jalr	Jump And Link Reg	I	1100111	0x0		rd = PC+4; PC = rs1	
lui	Load Upper Imm	U	0110111			rd = imm << 12	
auipc	Add Upper Imm to PC	U	0010111			rd = PC + (imm << 12)	
ecall	Environment Call	I	1110011	0x0	0x00	Transfer control to OS	imm: 0x000
ebreak	Environment Break	I	1110011	0x0	0x00	Transfer control to debugger	imm: 0x001


jalr
jal
b
Op
rfwe
Main Decoder
ws
memi
mwe
aop
srcB
srcA
[31:25] [14:12] [6:0]
CLK
RD1
1
0
32
PC
EN
A
RD
Instruction Memory
instr
[19:15]
[24:20]
[11:7]
5
A1
A2
A3
WE3
RD1
RD2
WD3
Register File
PC
0
imm_I
instr[31:12], 0
imm_S
4
2
3
k
ALU
comp
RD2
WD
WE
h
Data Memory
comp
+
imm_J
imm_B
[31:20]
[31:25] [11:7]
[31] [19:12] [20] [30:21]
[31] [7] [30:25] [11:8]
SE

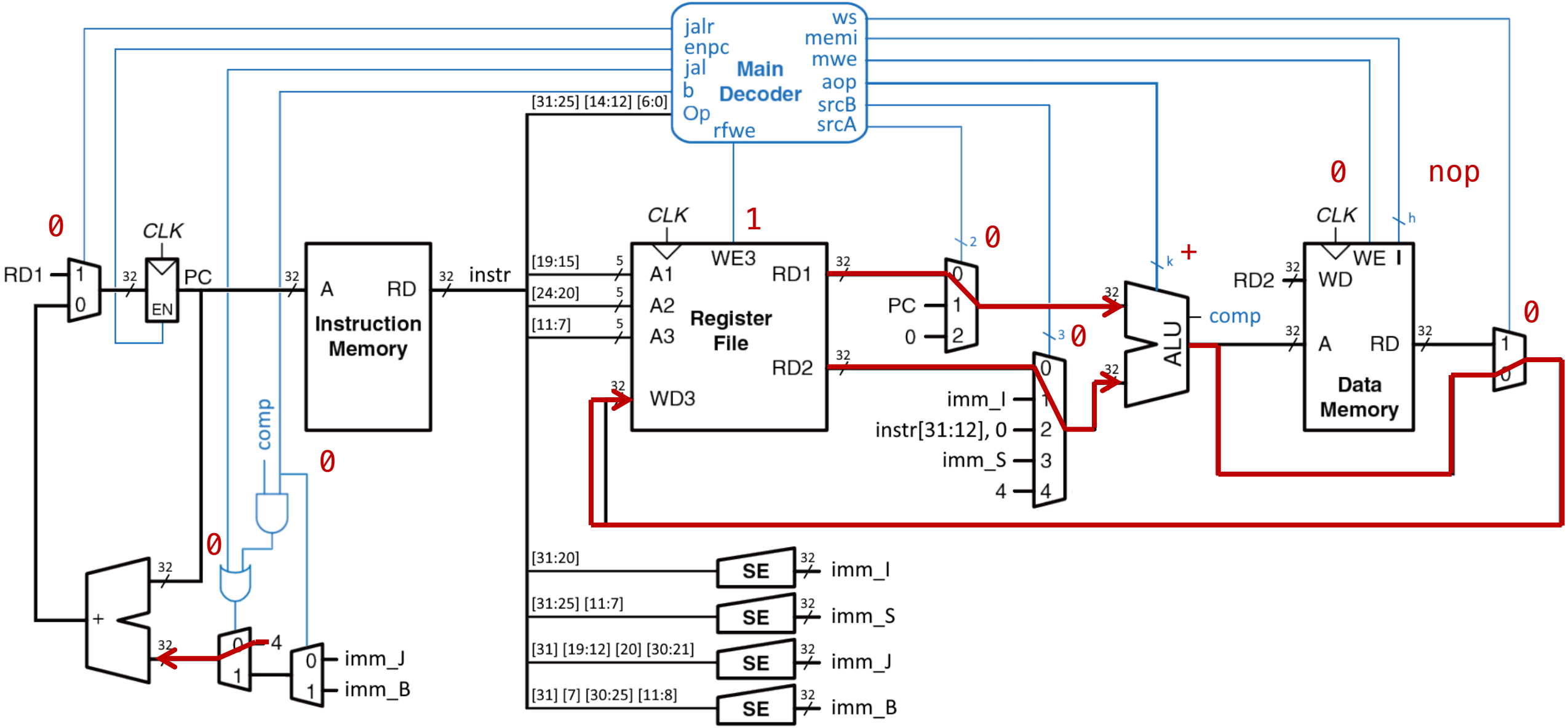

jalr
enpc
jal
b
Op
rfwe
Main Decoder
ws
memi
mwe
aop
srcB
srcA
[31:25] [14:12] [6:0]
CLK
RD1
1
0
32
EN
PC
A
RD
Instruction Memory
instr
[19:15]
[24:20]
[11:7]
5
A1
A2
A3
WD3
WE3
RD1
RD2
Register File
2
PC
0
imm_I
instr[31:12], 0
imm_S
4
3
k
ALU
comp
h
WE
WD
Data Memory
comp
+
imm_J
imm_B
[31:20]
[31:25] [11:7]
[31] [19:12] [20] [30:21]
[31] [7] [30:25] [11:8]
SE


jalr
enpc
jal
b
Op
Main Decoder
rfwe
ws
memi
mwe
aop
srcB
srcA
[31:25] [14:12] [6:0]
CLK
RD1
1
0
32
EN
PC
A
RD
Instruction Memory
instr
[19:15]
[24:20]
[11:7]
5
A1
A2
A3
WE3
RD1
RD2
WD3
Register File
2
PC
0
imm_I
instr[31:12], 0
imm_S
4
3
k
ALU
comp
h
WE
WD
Data Memory
comp
+
4
imm_J
imm_B
[31:20]
[31:25] [11:7]
[31] [19:12] [20] [30:21]
[31] [7] [30:25] [11:8]
SE
imm_I
imm_S
imm_J
imm_B

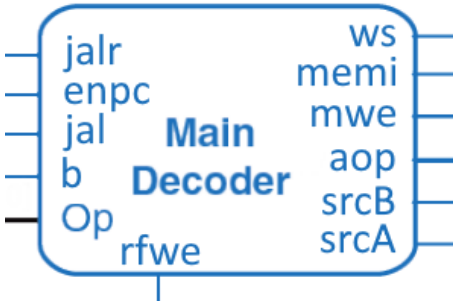
	func7, func3, opcode	jalr	jal	b	rfwe	srcA	srcB	aop	mwe	memi	ws
SW	010 0100011	0	0	0	0	00	011	+	1	write word	0



	func7, func3, opcode	jalr	jal	b	rfwe	srcA	srcB	aop	mwe	memi	ws
add	0000000 010 0100011	0	0	0	1	00	000	+	0	nop	0



Дешифратор инструкций



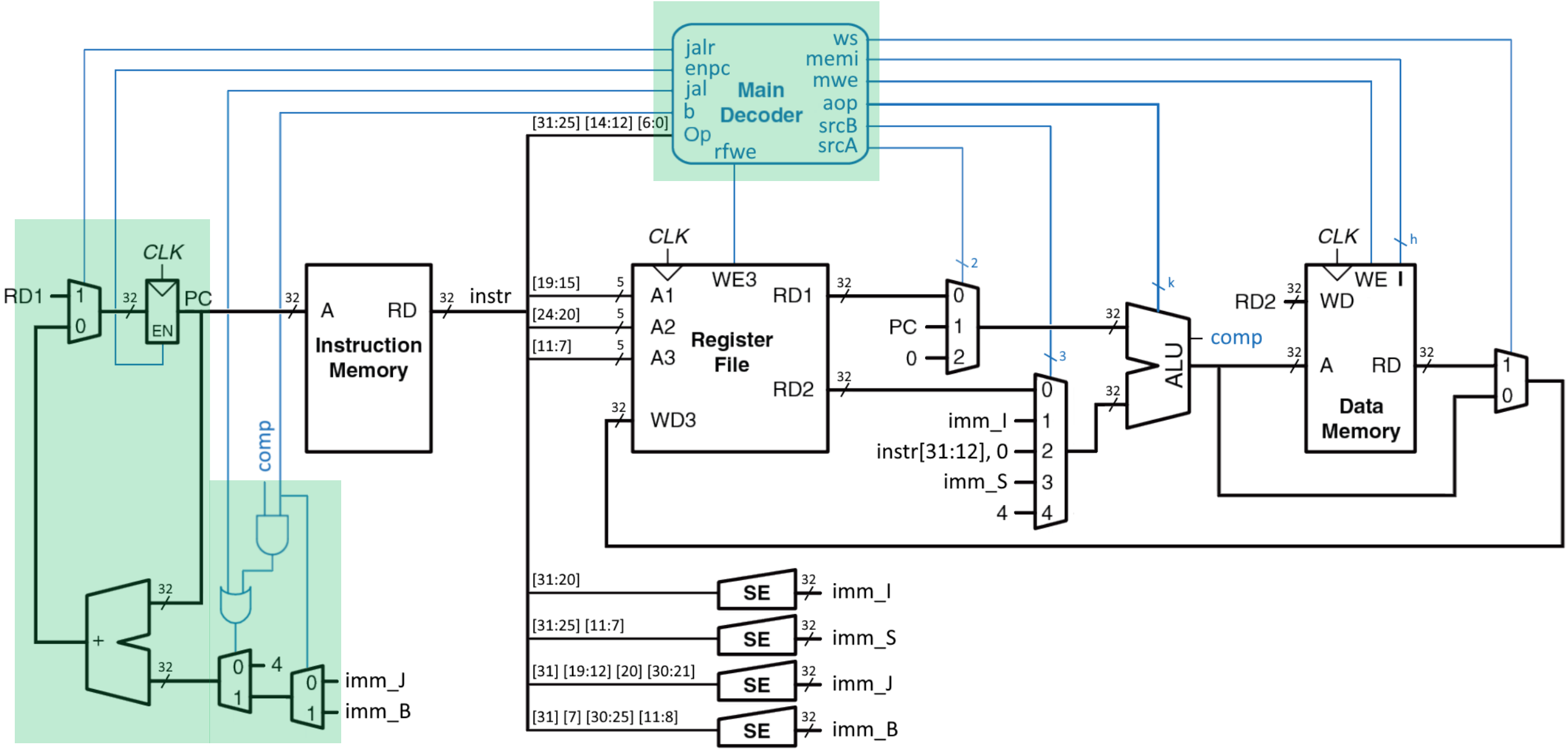
	func7, func3, opcode	jalr	jal	b	rfwe	srcA	srcB	aop	mwe	memi	ws
sw	010 0100011	0	0	0	0	00	011	+	1	write word	0
add	0000000 010 0100011	0	0	0	1	00	000	+	0	nop	0

…

…

…

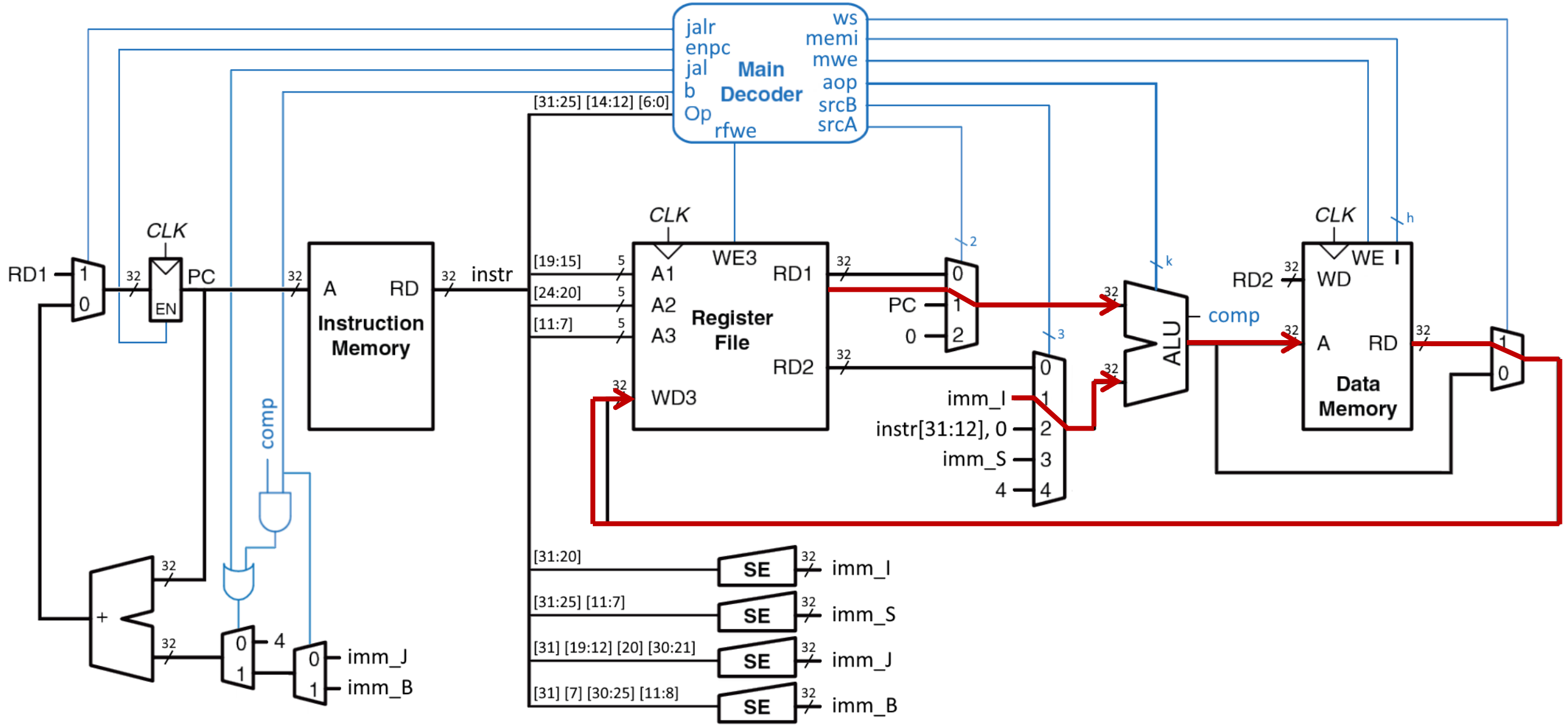
Где устройство управления?



Оценка производительности

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

Самая длинная инструкция? lw



Оценка производительности

Элемент	Параметр	Задержка (пс)
Задержка распространения clk-to-Q в регистре	t_{pcq}	30
Время предустановки регистра	t_{setup}	20
Мультиплексор	t_{mux}	25
АЛУ	t_{ALU}	200
Чтение из памяти	t_{mem}	250
Чтение из регистрового файла	t_{RFread}	150
Время предустановки регистрового файла (register file setup)	tRF_{setup}	20

$$T_c = t_{pcq\_PC} + 2t_{mem} + t_{RFread} + t_{ALU} + t_{mux} + t_{RFsetup}$$

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

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