

Архитектуры процессорных систем

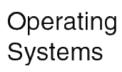
Лекция 7. Однотактный процессор RISC-V

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!"

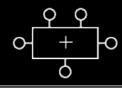
Architecture



Microarchitecture



Logic



Digital Circuits

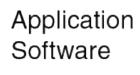


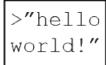
Devices



Physics







Operating Systems



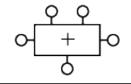
Architecture



Microarchitecture



Logic



Digital Circuits



Devices

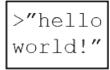


Physics



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





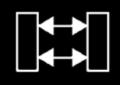
Operating Systems



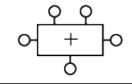
Architecture



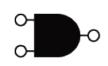




Logic



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

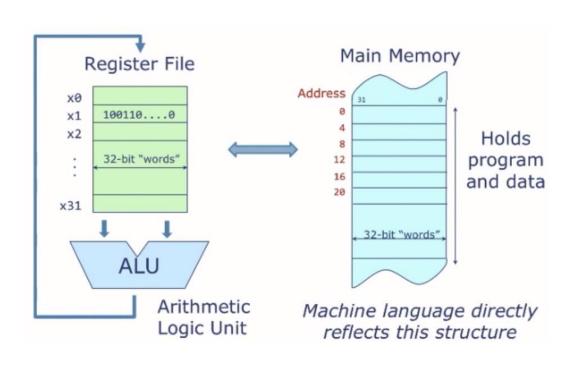
• Многотактная

• выполняет одну инструкцию за несколько более коротких тактов

• Конвейерная

• результат применения принципа конвейерной обработки к однотактной микроархитектуре

Особенности архитектуры RISC-V



- Регистровый файл
 - 32 регистра общего назначения
 - Каждый регистр 32 бита
 - XO = O
- 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 2³² байт или 2³⁰ 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
1b	Load Byte	I	0000011	0x0		rd = M[rs1+imm][0:7]	
1h	Load Half	I	0000011	0x1		rd = M[rs1+imm][0:15]	
lw	Load Word	I	0000011	0x2		rd = M[rs1+imm][0:31]	
1bu	Load Byte (U)	I	0000011	0x4		rd = M[rs1+imm][0:7]	zero-extends
1hu	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]	
beg	Branch ==	В	1100011	0x0		if(rs1 == rs2) PC += imm	
bne	Branch !=	В	1100011	0x1		if(rs1 != rs2) PC += imm	
blt	Branch <	В	1100011	0x4		if(rs1 < rs2) PC += imm	
bge	Branch <	В	1100011	0x5		if(rs1 >= rs2) PC += imm	
bltu	Branch < (U)	В	1100011	0x6		if(rs1 < rs2) PC += imm	zero-extends
bgeu	Branch \geq (U)	В	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
		_				1 1 2 2 4 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6	

Кодирование инструкций RISC-V

31	27	26	25	24	20	19	15	14	12	11	7	6		0
	func	t7		rs	2	rs1		fun	ct3		rd	op	code	
	in	mm[11:0]			rs1		fun	ct3		rd	op	code		
in	nm[1	1:5]		rs	2	rs1		fun	ct3	imn	n[4:0]	op	code	
imi	nm[12 10:5] rs2			2	rs1		fun	ct3	imm[4:1 11]	op	code		
	imm[3				m[31	:12]					rd	op	code	
			imn	n[20	10:1	11 19:12	2]				rd	op	code	

R-type I-type S-type B-type U-type 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	

funct7		rs2			r	s1		fu	nc	t3			rd				0	рсо	de		R -type
31 30 29 28 27 26	5 24	23 22 21	20	19	18 1	17 16	5 15	14	13	12	11	10	9	8	7	6	5 4	3	2 1	0	
		2				2			_				_					_		•	
0		3			•	2							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		

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

Machine Code

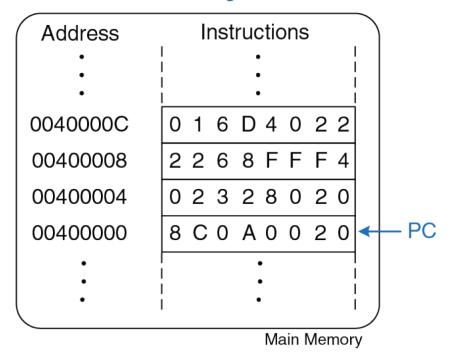
0x016D4022

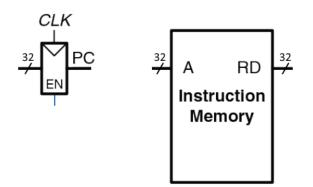
Assembly Code

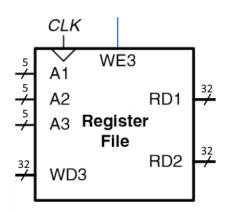
sub

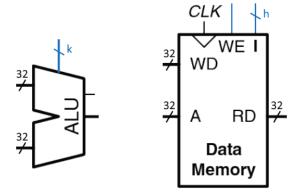
\$t0, \$t3, \$t5

Stored Program



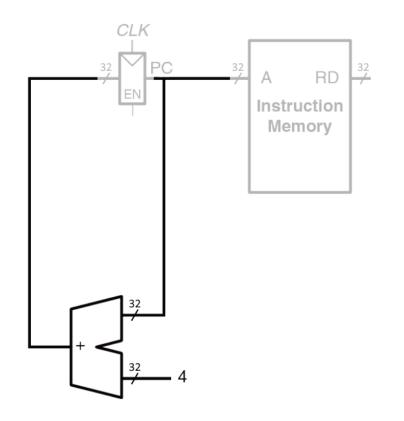


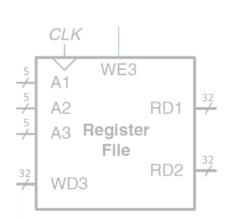


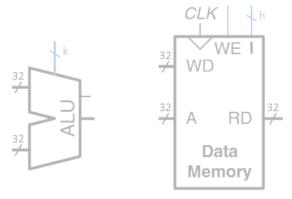


31	27	26	25	24	2	0	19	15	14	12	11	7	6	0	
	funct7				rs2		rs	1	fun	ct3		rd	opo	ode	R-type
	0000000)			rs2		rs	1	00	00		rd	0110	0011	ADD

add xN, xM, xK
rd = rs1 + rs2

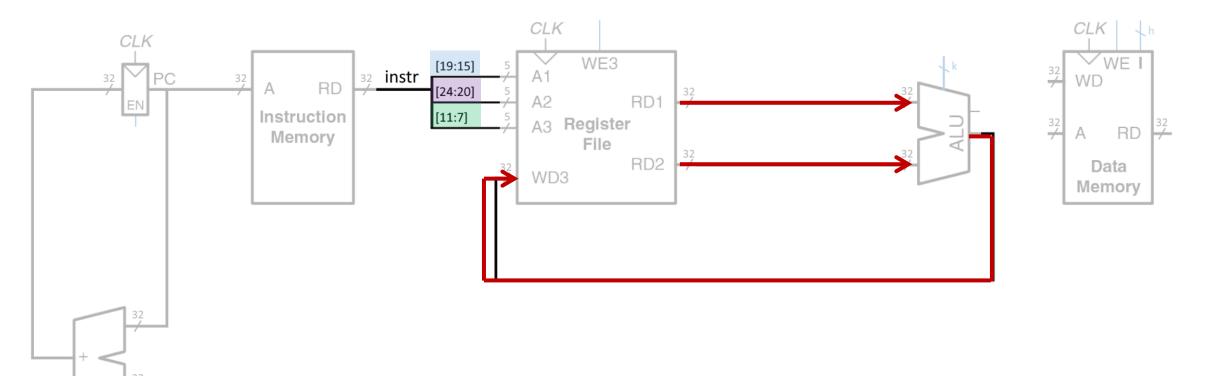






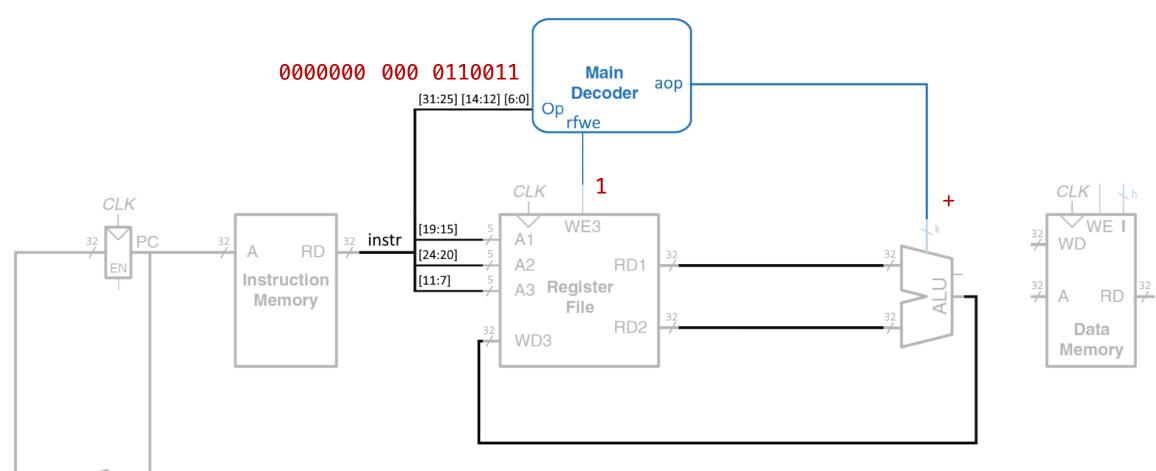
31	27	26	25	24		20	19	15	14	12	11	7	6	0	
	funct7				rs2		rs	1	fun	ct3		rd	opc	ode	R-type
	0000000)			rs2		rs	1	00	0		rd	0110	0011	ADD

add xN, xM, xK rd = rs1 + rs2



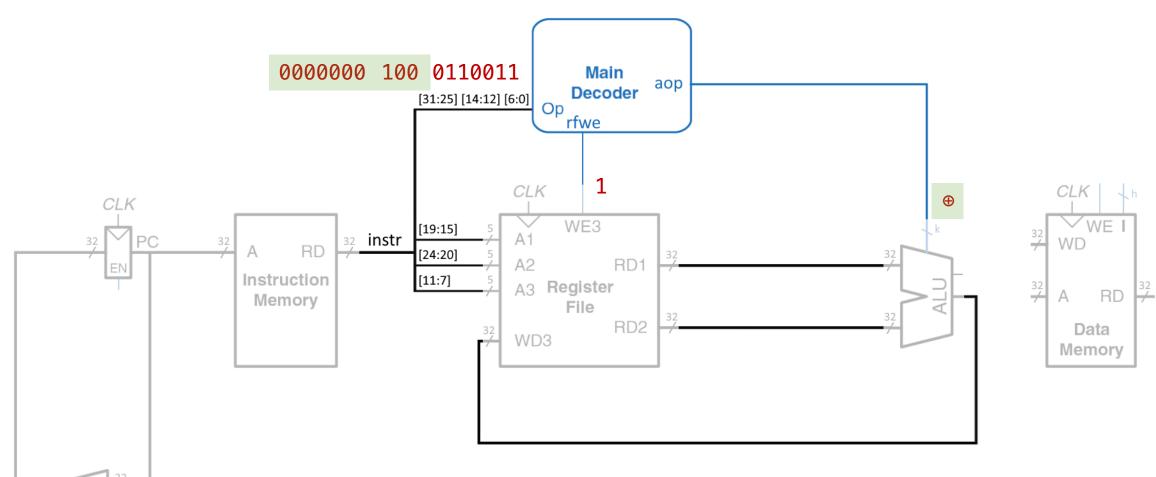
31	27	26	25	24	2	20	19	15	14	12	11	7	6	0	
	funct7				rs2		rs1		fun	ct3		rd	opo	code	R-type
	0000000				rs2		rsl	L	00	0	1	$\overline{\mathrm{rd}}$	011	0011	ADD

add xN, xM, xK rd = rs1 + rs2



31	27	26	25	24	2	0	19	15	14	12	11	7	6	0	
	funct7				rs2		rs	1	fun	ct3	1	rd	opo	code	R-type
	0000000)			rs2		rs	1	10	0	r	d	0110	0011	XOR

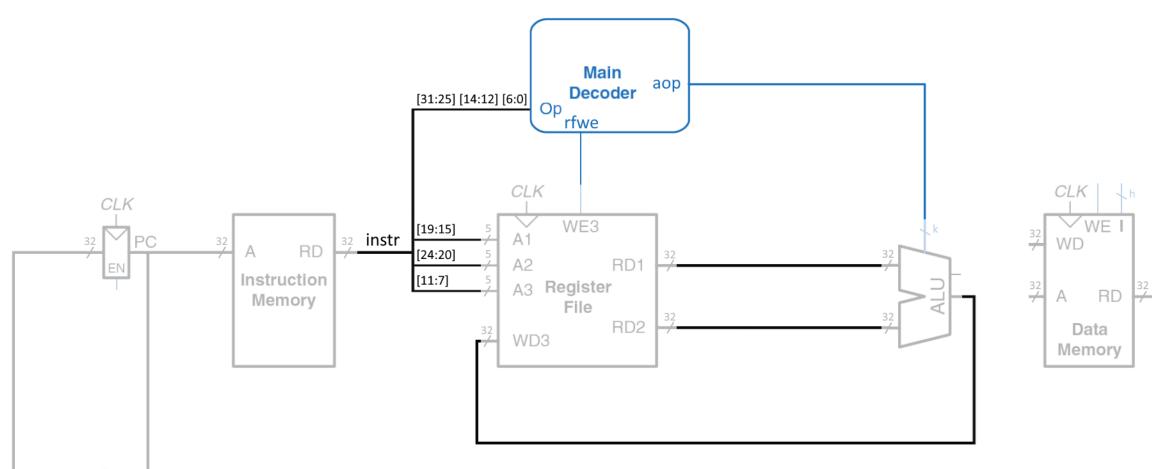
xor xN, xM, xK
rd = rs1 ^ rs2



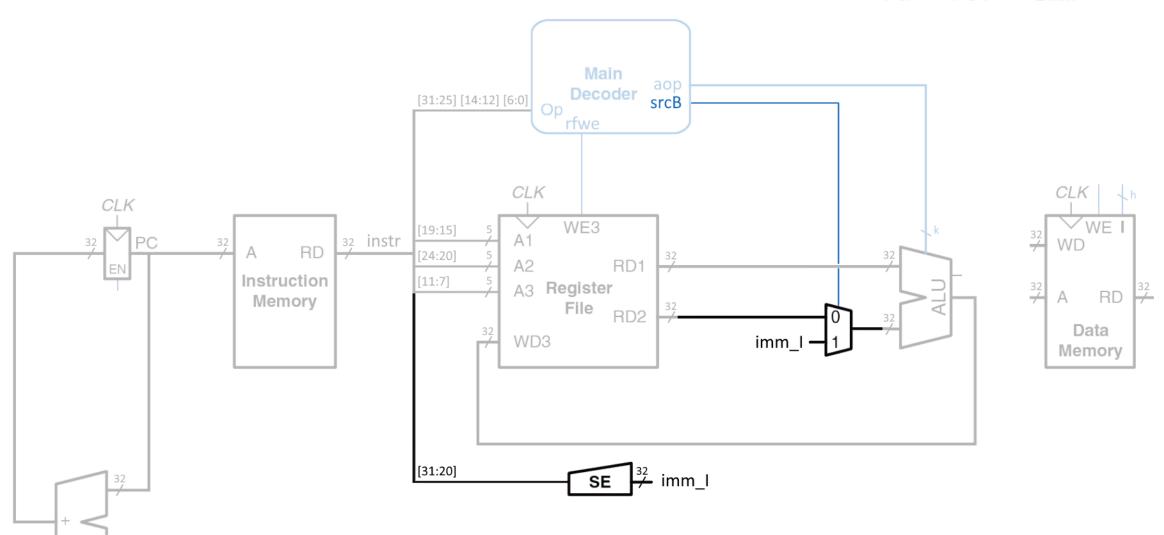
Набор инструкций RISC-V

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]	
1h	Load Half	I	0000011	0x1		rd = M[rs1+imm][0:15]	
lw	Load Word	I	0000011	0x2		rd = M[rs1+imm][0:31]	
1bu	Load Byte (U)	I	0000011	0x4		rd = M[rs1+imm][0:7]	zero-extends
1hu	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 ==	В	1100011	0x0		if(rs1 == rs2) PC += imm	
bne	Branch !=	В	1100011	0x1		if(rs1 != rs2) PC += imm	
blt	Branch <	В	1100011	0x4		if(rs1 < rs2) PC += imm	
bge	Branch ≤	В	1100011	0x5		if(rs1 >= rs2) PC += imm	
bltu	Branch < (U)	В	1100011	0x6		if(rs1 < rs2) PC += imm	zero-extends
bgeu	Branch \geq (U)	В	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
	1						·

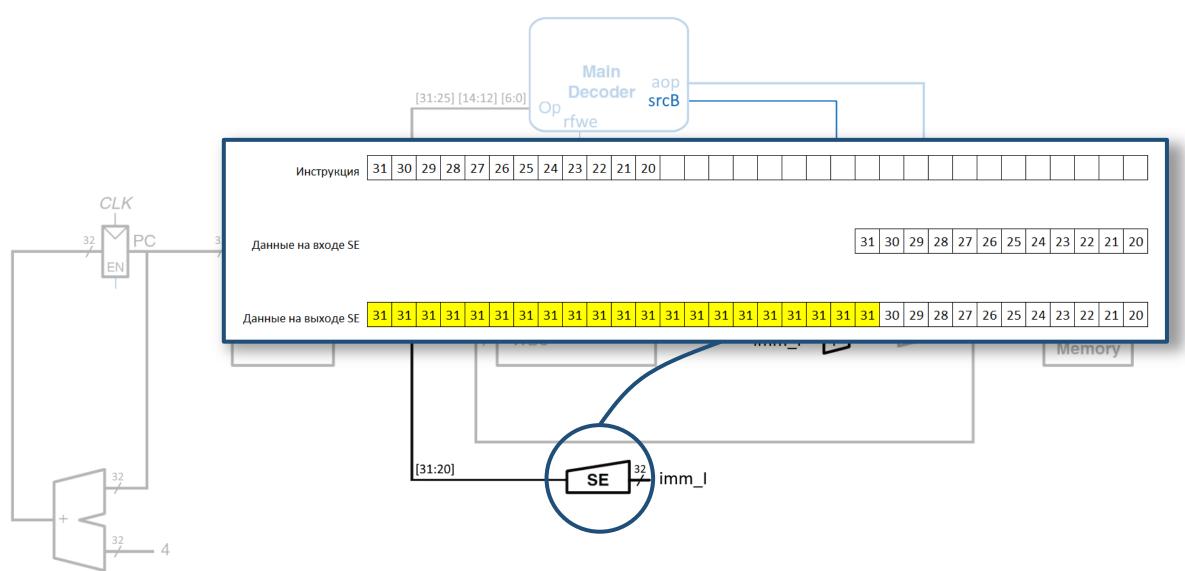
31	27 26 25 24	20	19 15	14 12	11 7	6 0	
	imm[11:0]		rs1	funct3	rd	opcode] I-type
	imm[11:0]		rs1	000	rd	0010011	ADDI



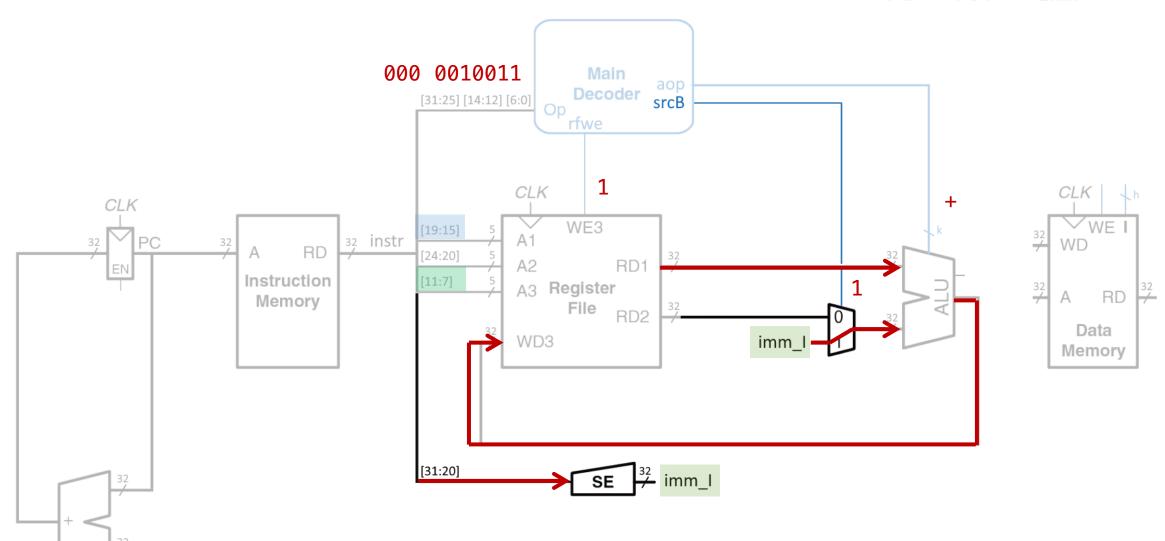
31	27 26 25 24	20	19 1	15	14 12	11	7	6	0	
	imm[11:0]		rs1		funct3	r	d	opcod	de] I-type
	imm[11:0]		rs1		000	r	d	00100	11	ADDI



imm[11:0] rs1 funct3 rd opco	
	e I-type
imm[11:0] rs1 000 rd 00100	.1 ADDI

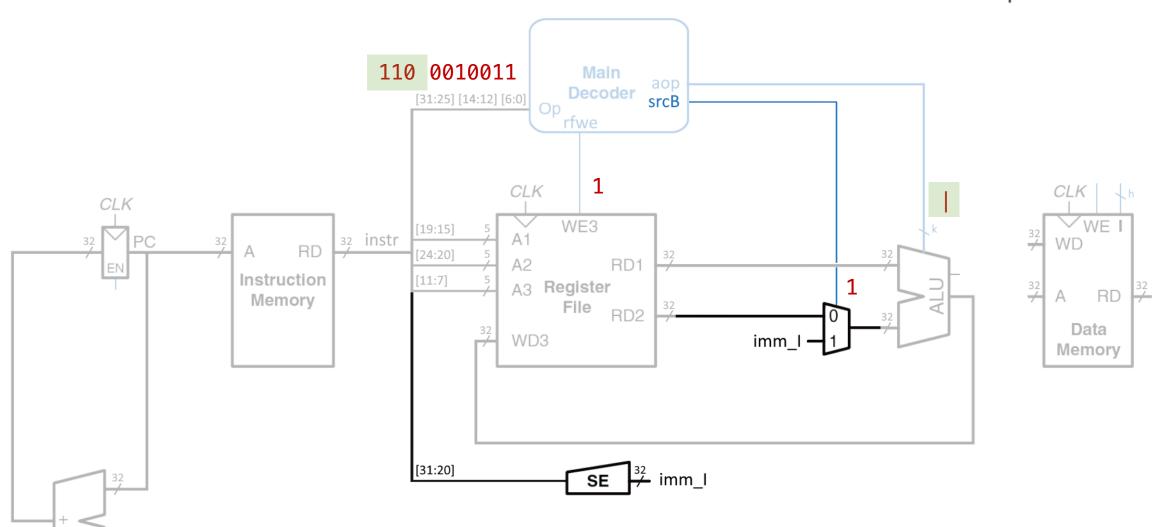


31	27 26 25 24	20	19	15	14	12	11	7	6	0	
	imm[11:0]		rs1		func	ct3	r	d	opc	ode	I-type
	imm[11:0]		rs1		00	0	r	d	0010	0011	ADDI



31	27 26 25 24	20	19	15	14	12	11	7	6	0	
	imm[11:0]		rs1		func	ct3	r	rd	opc	ode	I-type
	imm[11:0]		rs1		11	0	r	rd	0010	0011	ORI

ori xN, xM, const
rd = rs1 | imm

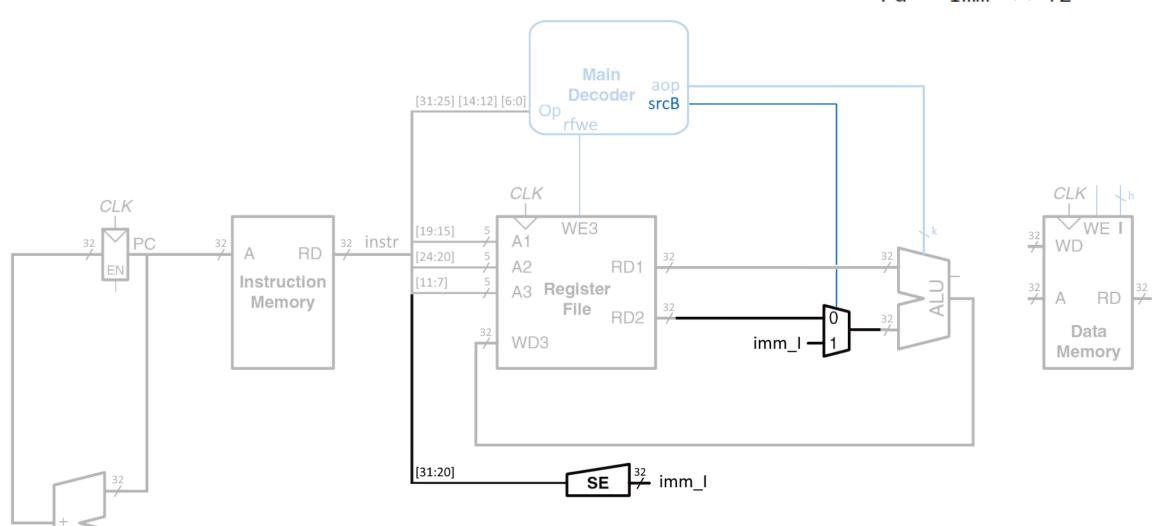


Набор инструкций RISC-V

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
1b	Load Byte	I	0000011	0x0		rd = M[rs1+imm][0:7]	
1h	Load Half	I	0000011	0x1		rd = M[rs1+imm][0:15]	
lw	Load Word	I	0000011	0x2		rd = M[rs1+imm][0:31]	
1bu	Load Byte (U)	I	0000011	0x4		rd = M[rs1+imm][0:7]	zero-extends
1hu	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]	
beg	Branch ==	В	1100011	0x0		if(rs1 == rs2) PC += imm	
bne	Branch !=	В	1100011	0x1		if(rs1 != rs2) PC += imm	
blt	Branch <	В	1100011	0x4		if(rs1 < rs2) PC += imm	
bge	Branch <	В	1100011	0x5		if(rs1 >= rs2) PC += imm	
bltu	Branch < (U)	В	1100011	0x6		if(rs1 < rs2) PC += imm	zero-extends
bgeu	Branch \geq (U)	В	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
32. 541					00	control to dobugger	

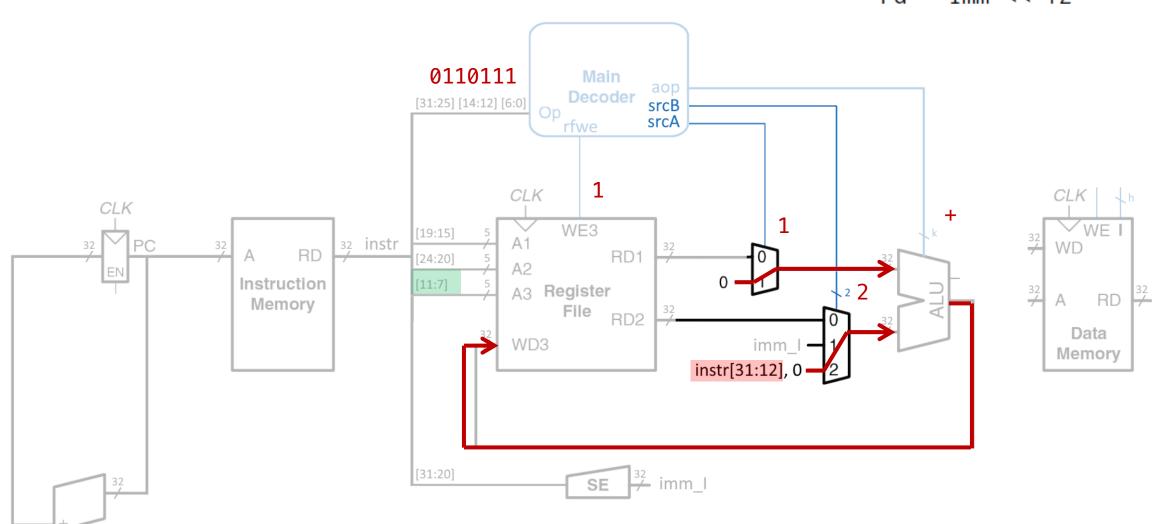
31	27	26	25	24	20	19	15	14	12	11	7	6	0	
				imn	n[31:12]					r	d	opc	ode	$\int U-typ\epsilon$
				imn	n[31:12]					r	rd	0110)111	LUI

lui xN, const
rd = imm << 12</pre>



31	27	26	25	24	20	19	15	14	12	11	7	6	0	
				imn	n[31:12]					1	rd	ope	code	U-type
	imm[31:12]								1	rd	011	0111	LUI	

lui xN, const
rd = imm << 12</pre>

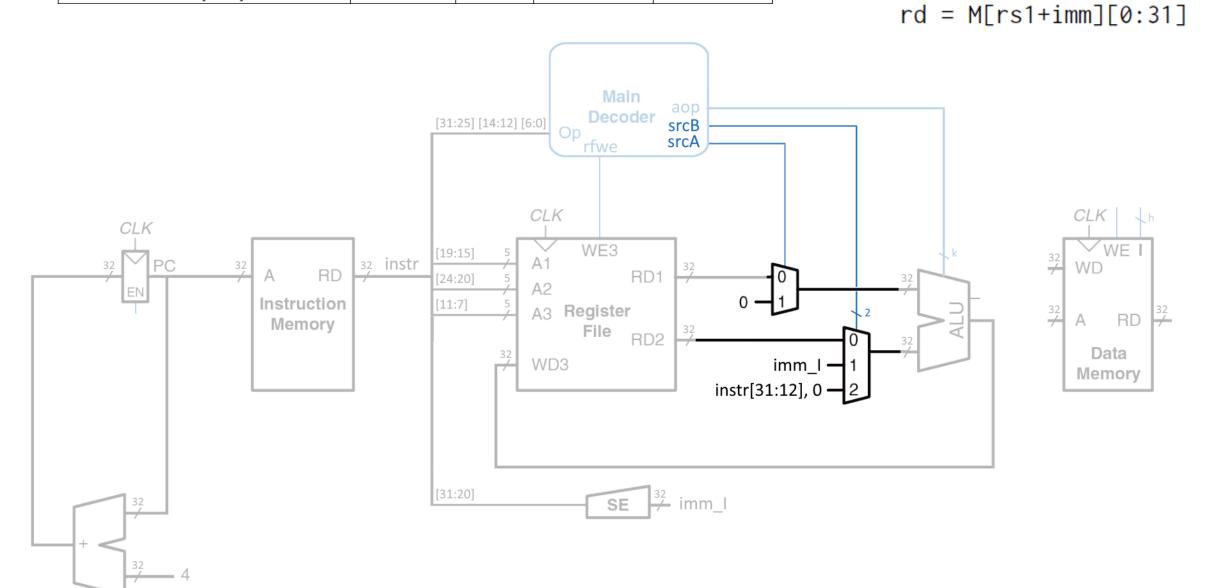


Набор инструкций RISC-V

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
1b	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]	
1bu	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 ==	В	1100011	0x0		if(rs1 == rs2) PC += imm	
bne	Branch !=	В	1100011	0x1		if(rs1 != rs2) PC += imm	
blt	Branch <	В	1100011	0x4		if(rs1 < rs2) PC += imm	
bge	Branch ≤	В	1100011	0x5		if(rs1 >= rs2) PC += imm	
bltu	Branch < (U)	В	1100011	0x6		if(rs1 < rs2) PC += imm	zero-extends
bgeu	Branch \geq (U)	В	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

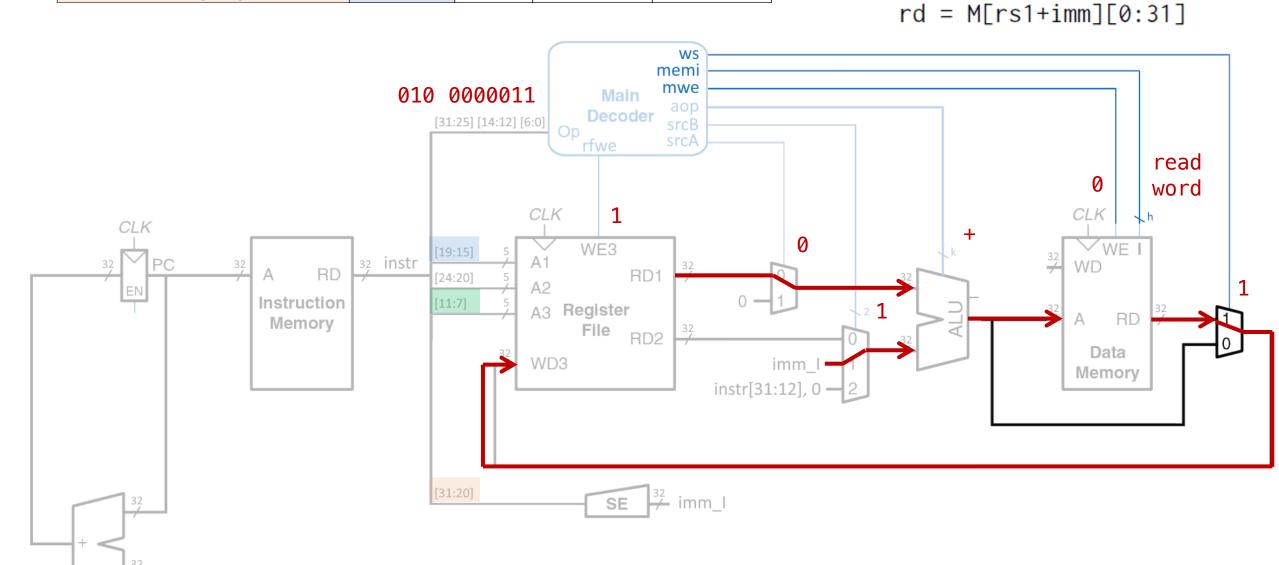
31	27 26 25 24	20	19	15	14	12	11	7	6	0	
	imm[11:0]		rs1	-	fun	ct3		rd	opce	ode	I-type
	imm[11:0]		rs1		01	.0		rd	0000	0011	LW

lw xN, offset(base)



31	27 26 25 24	20	19	15	14	12	11	7	6	0	
	imm[11:0]		rs	1	fun	ct3	1	·d	ope	ode] I-type
	imm[11:0]		rs	1	01	.0	1	rd .	0000	0011	LW

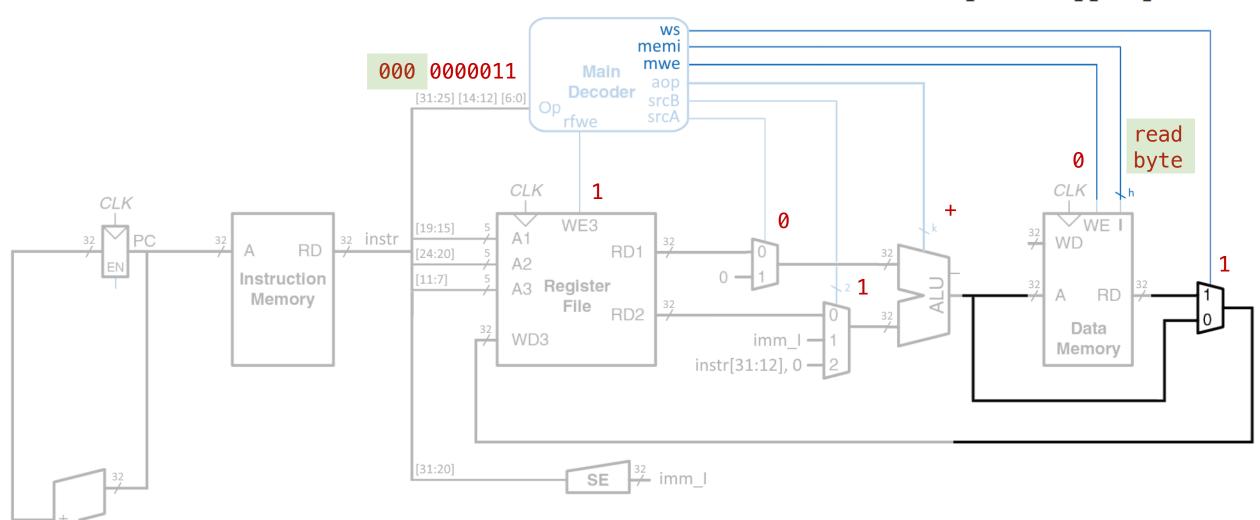
lw xN, offset(base)



31	27 26 25 24	20	19	15	14	12	11	7	6	0	
	imm[11:0]		rs	1	fun	ct3		rd	opc	ode	I-type
	imm[11:0]		rsi	1	00	00		rd	0000	0011	LB

lb xN, offset(base)

rd = M[rs1+imm][0:7]



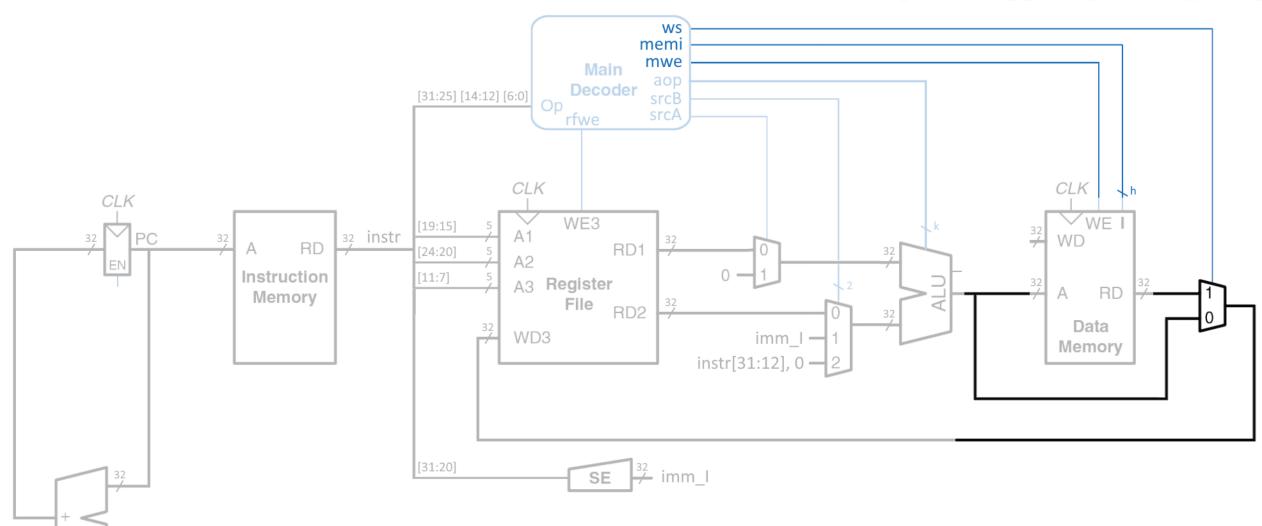
Набор инструкций RISC-V

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]	
1h	Load Half	I	0000011	0x1		rd = M[rs1+imm][0:15]	
lw	Load Word	I	0000011	0x2		rd = M[rs1+imm][0:31]	
1bu	Load Byte (U)	I	0000011	0x4		rd = M[rs1+imm][0:7]	zero-extends
1hu	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 ==	В	1100011	0x0		if(rs1 == rs2) PC += imm	
bne	Branch !=	В	1100011	0x1		if(rs1 != rs2) PC += imm	
blt	Branch <	В	1100011	0x4		if(rs1 < rs2) PC += imm	
bge	Branch ≤	В	1100011	0x5		if(rs1 >= rs2) PC += imm	
bltu	Branch < (U)	В	1100011	0x6		if(rs1 < rs2) PC += imm	zero-extends
bgeu	Branch \geq (U)	В	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	2	0	19	15	14	12	11	7	6	0	
	imm[11:	5]			rs2		rs	1	fun	ct3	imn	n[4:0]	opo	code	S-type
	imm[11:5	5]			rs2		rs	1	01	.0	imn	n[4:0]	010	0011	SW

sw xN, offset(base)

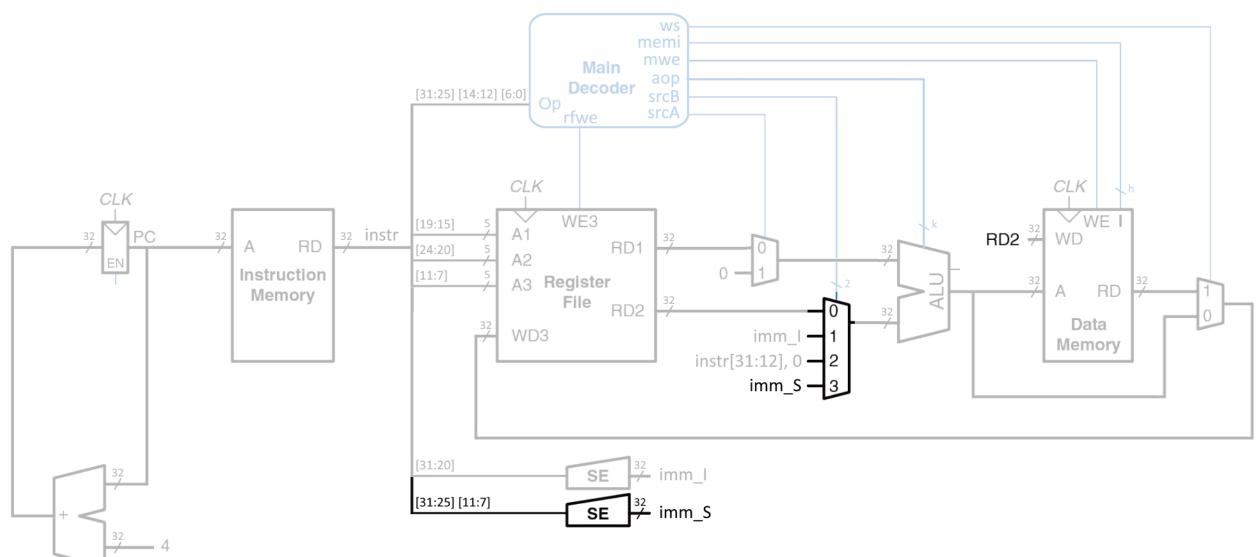
M[rs1+imm][0:31] = rs2[0:31]



31	27 2	6 25	24	20	19	15	14	12	11	7	6	0	
	imm[11:5]		rs2		rs	1	fun	ct3	imm	1[4:0]	opo	code	S-type
	imm[11:5]		rs2		rs	1	01	0	imm	[4:0]	010	0011	SW

sw xN, offset(base)

M[rs1+imm][0:31] = rs2[0:31]

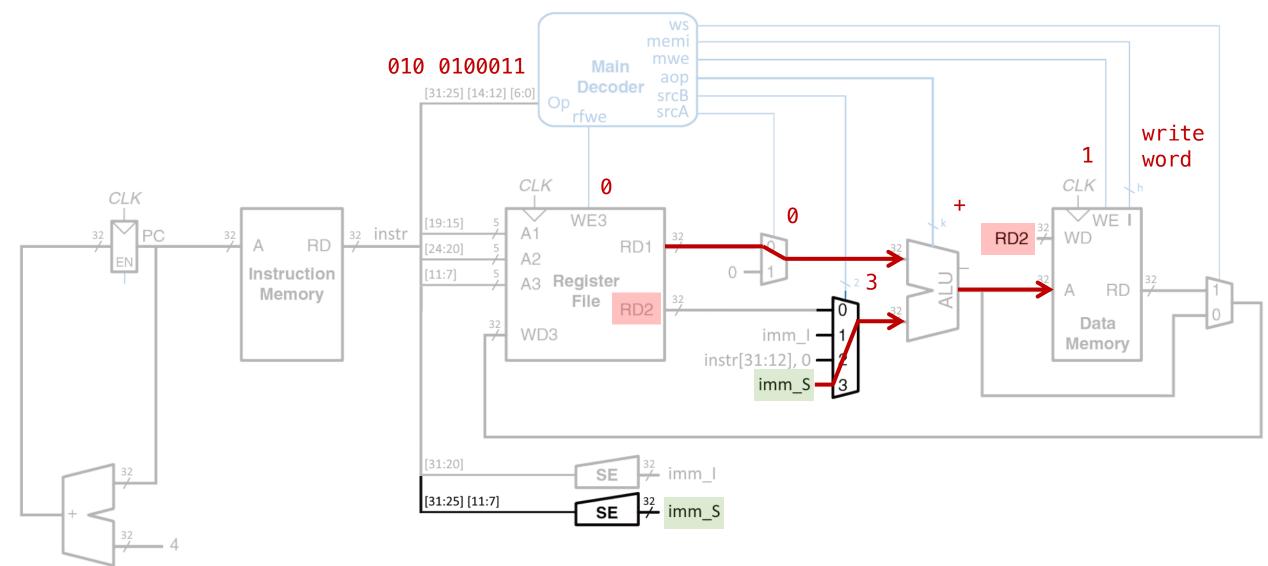


31		26 25			15 14 12	11 7	6 0	7 a .					
	imm[11	_	rs2	rs1	funct3	imm[4:0]	opcode	S-type	<pre>sw xN, offset(base)</pre>				
imm[11		5]	rs2	rs1	010	[mm[4:0]]	0100011	brack SW	M[rs1+imm][0:31] = rs2[0:31]				
									M[[S]+IMM][0.3] - [S2[0.3]				
							ws memi						
						Main	mwo						
					[31:25] [14:12] [6	_ Op	er srcB ———						
						' rfwe	srcA						
	ſ			4 20 20 20	27 25 25								
	CLK		Инструкция <mark>З</mark>	31 30 29 28	3 27 26 25				11 10 9 8 7				
	32 F EN	Лациь	ые на входе SE						31 30 29 28 27 26 25 11 10 9 8 7				
		даппо	ле на входе эс										
		Данные	е на выходе SE	31 31 31 31	31 31 31 3	31 31 31 31	31 31 31 31 31	1 31 31 3	1 31 31 30 29 28 27 26 25 11 10 9 8 7				
	L							imm_S -	3				
								s					
_	32				[31:20]	SE	imm_I						
	ĺ				[31:25] [11:7]	SE	3						
	32	1					J /"""_3						
_		- +											

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	rs	s1	01	10	imn	1[4:0]	0100	0011	brack SW

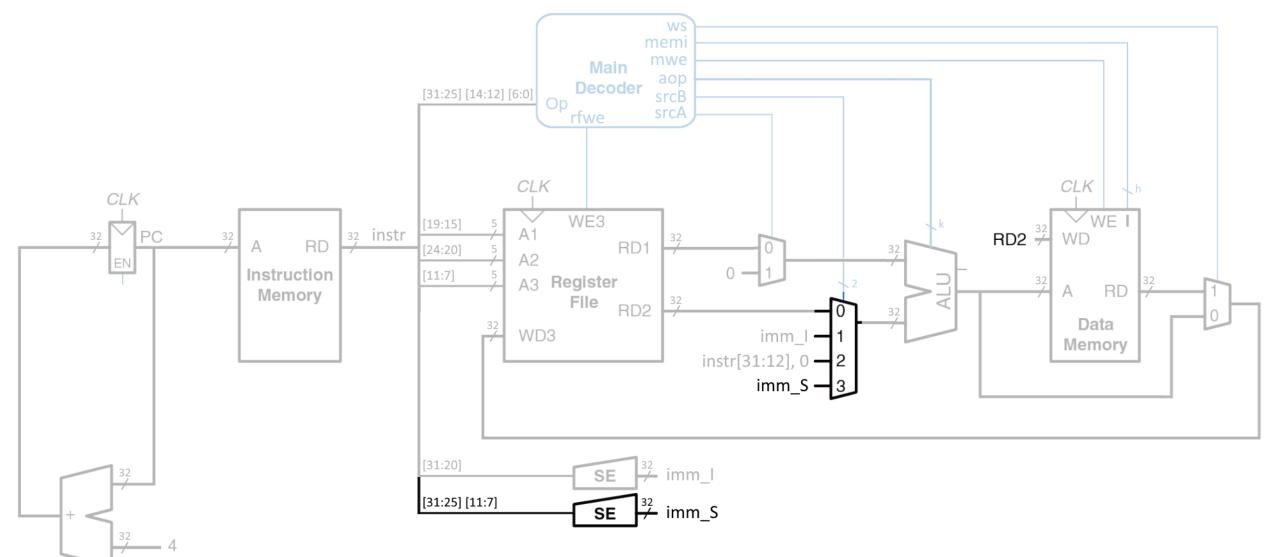
sw xN, offset(base)

M[rs1+imm][0:31] = rs2[0:31]

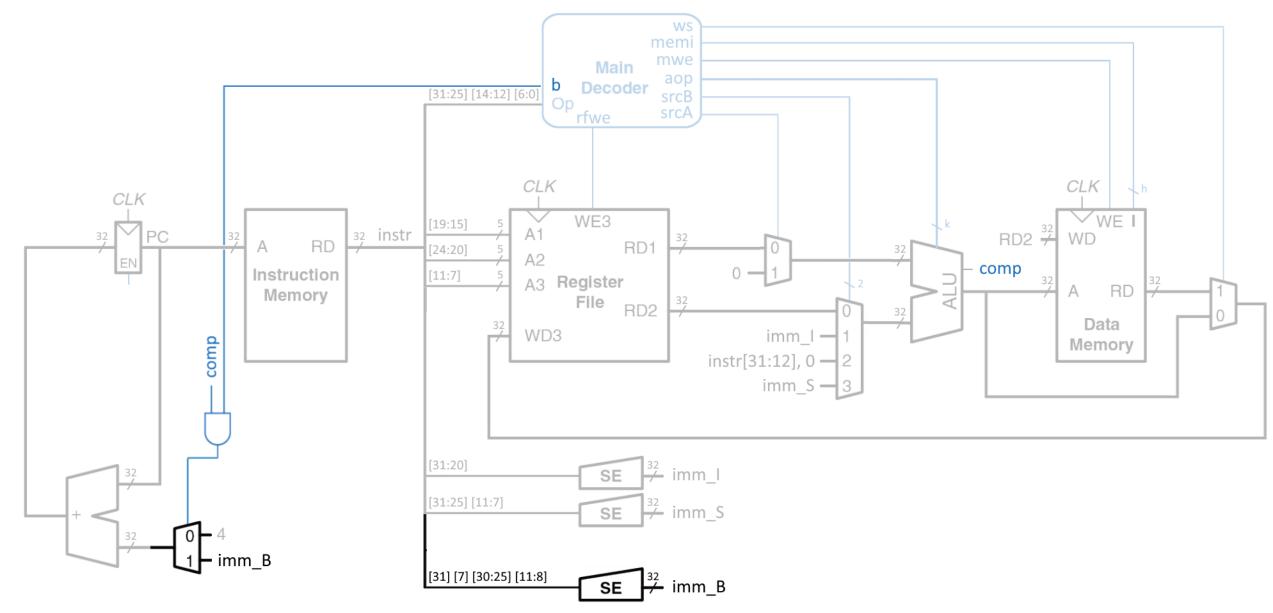


Sub Sub R 0000011 0x0 0x00 rd = rs1 + rs2	Inst	Name	FMT	Opcode	F3	F7	Description (C)	Note
xor XOR R 0000011 0x4 0x00 rd = rs1 ^ rs2 rd = rs1 rs2 and AND R 0000011 0x6 0x00 rd = rs1 rs2 rd = rs1 rs2 s11 Shift Right Logical R 0000011 0x1 0x00 rd = rs1 > rs2 rs1 shift Right Logical R 0000011 0x2 0x00 rd = rs1 >> rs2 rs2 ssa Shift Right Logical R 0000011 0x2 0x00 rd = rs1 >> rs2 rs5 sst sss ssit sss sss ssit sss rd = rs1 rs2 rd = rs1 > rs2 msb-exter sit Set Less Than R 0110011 0x2 0x00 rd = rs1 + rs2 rd = (rs1 < rs2)?1:0	add	ADD	R	0000011	0x0	0x00	rd = rs1 + rs2	
or OR R 0000011 0x6 0x00 rd = rs1 rs2 and AND R 0000011 0x7 0x00 rd = rs1 rs2 st st <th< td=""><td>sub</td><td>SUB</td><td>R</td><td>0000011</td><td>0x0</td><td>0x20</td><td>rd = rs1 - rs2</td><td></td></th<>	sub	SUB	R	0000011	0x0	0x20	rd = rs1 - rs2	
and AND R 0000011 0x7 0x00 rd = rs1 & rs2 s11 Shift Left Logical R 0000011 0x1 0x00 rd = rs1 < rs2 sr1 Shift Right Logical R 0000011 0x2 0x00 rd = rs1 < rs2 sr2 Shift Right Arith* R 0000011 0x3 0x20 rd = rs1 >> rs2 sr3 Shift Right Arith* R 0000011 0x3 0x20 rd = rs1 >> rs2 st1 Set Less Than R 0110011 0x3 rd = (rs1 < rs2)?1:0 zero-exter s1t Set Less Than (U) R 0110011 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 sr1i Shift Left Logical Imm I 0010011 0x0 0x00 rd = rs1 imm sr1i Shift Right Logical Imm I 0010011 0x1 0x00 rd = rs1 < imm sr1i Shift Right Logical Imm I 0010011 0x1 0x00 rd = rs1 < imm sr1i Shift Right Logical Imm I 0010011 0x1 0x00 rd = rs1 >> imm sr1i Shift Right Logical Imm I 0010011 0x2 rd = (rs1 < imm)?1:0 st1 Set Less Than Imm (U) I 0010011 0x2 rd = (rs1 < imm)?1:0 st1 Set Less Than Imm I 0010011 0x2 rd = (rs1 < imm)?1:0 st1 Set Less Than Imm (U) I 0010011 0x2 rd = (rs1 < imm)?1:0 st1 Load Byte I 0000011 0x2 rd = (rs1 < imm)?1:0 tb Load Byte U I 0000011 0x2 rd = (rs1 < imm)?1:0 tb Load Byte (U) I 0000011 0x2 rd = (rs1 < imm)?1:0 sb Store Byte S 0100011 0x4 rd = M[rs1+imm][0:7] zero-exter sb Store Byte S 0100011 0x0 M[rs1+imm][0:15] rs2[0:7] sh Store Half S 0100011 0x1 M[rs1+imm][0:15] rs2[0:7] sh Store Word S 0100011 0x1 M[rs1+imm][0:15] rs2[0:7] sh Store Word S 0100011 0x1 if(rs1 = rs2) PC += imm be Branch B 1100011 0x4 if(rs1 < rs2) PC += imm if(rs1 < rs2) PC += imm be Branch CU Branch < B 1100011 0x7 if(rs1 >= rs2) PC += imm zero-exter bgeu Branch ≥ (U) B 1100011 0x7 if(rs1 >= rs2) PC += imm zero-exter lut Load Upper Imm U 0 0110111 rd = rd = PC+4; PC += imm zero-exter lut Load Upper Imm U 0 0110111 rd = rd = PC+4; PC += imm zero-exter lut Load Upper Imm U 0 0110111 rd = rd = PC+4; PC += imm zero-exter	xor	XOR	R	0000011	0x4	0x00	rd = rs1 ^ rs2	
Shift Left Logical R 0000011 0x1 0x00 rd = rs1 < rs2 sr1 Shift Right Logical R 0000011 0x2 0x00 rd = rs1 >> rs2 sr3 Shift Right Arith* R 0000011 0x3 0x20 rd = rs1 >> rs2 msb-exter slt Set Less Than R 0110011 0x2 rd = (rs1 < rs2)?1:0 zero-exter solution xor xo	or	OR	R	0000011	0x6	0x00	rd = rs1 rs2	
sr1 Shift Right Logical sra R 0000011 000011 0x2 0x20 0x00 rd = rs1 >> rs2 rd = rs1 >> rs2 msb-exter slt Set Less Than R 0110011 0x2 0x20 rd = rs1 >> rs2 msb-exter slt Set Less Than (U) R 0110011 0x0 0x2 rd = (rs1 < rs2)?1:0	and	AND	R	0000011	0x7	0x00	rd = rs1 & rs2	
sra Shift Right Arith* R 0000011 0x3 0x20 rd = rs1 >> rs2 msb-exter slt Set Less Than R 0110011 0x2 rd = (rs1 < rs2)?1:0	sll	Shift Left Logical	R	0000011	0x1	0x00	rd = rs1 << rs2	
slt Set Less Than R 0110011 0x2 rd = (rs1 < rs2)?1:0 zero-exter addi ADD Immediate I 0010011 0x0 0x00 rd = (rs1 < rs2)?1:0 zero-exter addi ADD Immediate I 0010011 0x0 0x00 rd = rs1 + imm zero-exter ori OR Immediate I 0010011 0x0 0x00 rd = rs1 * imm and solit AND Immediate I 0010011 0x0 0x00 rd = rs1 * imm and slt Shift Right Logical Imm I 0010011 0x1 0x00 rd = rs1 * imm ms shift Right Arith Imm I 0010011 0x1 0x00 rd = rs1 * imm msb-exter slt Set Less Than Imm I 0010011 0x3 0x20 rd = rs1 * imm msb-exter slt Set Less Than Imm I 0010011 0x3 ox20 rd = rs1 * imm msb-exter slt Set Less Than Imm I 0010011	srl	Shift Right Logical	R	0000011	0x2	0x00	rd = rs1 >> rs2	
sltu Set Less Than (U) R 0110011 0x3 rd = (rs1 < rs2)?1:0 zero-extent addi 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 slti Shift Right Logical Imm I 0010011 0x1 0x00 rd = rs1 imm sr1 Shift Right Arith Imm I 0010011 0x1 0x00 rd = rs1 imm mm st Set Less Than Imm I 0010011 0x1 0x00 rd = rs1 imm mm sltu Set Less Than Imm I 0010011 0x2 rd = rs1 imm mm msb-extent sltu Set Less Than Imm I 0010011 0x2 rd = rs1 imm mm msb-extent	sra	Shift Right Arith*	R	0000011	0x3	0x20	rd = rs1 >> rs2	msb-extends
Addi	slt	Set Less Than	R	0110011	0x2		rd = (rs1 < rs2)?1:0	
xor1 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 sl1 Shift Left Logical Imm I 0010011 0x00 rd = rs1 imm sr1 Shift Right Arith Imm I 0010011 0x1 0x00 rd = rs1 >> imm msb-exter slt Set Less Than Imm I 0010011 0x3 0x20 rd = (rs1 < imm)?1:0	sltu	Set Less Than (U)	R	0110011	0x3		rd = (rs1 < rs2)?1:0	zero-extends
ori OR Immediate I 0010011 0x0 0x00 rd = rs1 imm andi AND Immediate I 0010011 0x0 0x00 rd = rs1 & imm s1li 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 0x2 rd = rs1 >> imm msb-exter slt Set Less Than Imm I 0010011 0x2 rd = (rs1 < imm)?1:0	addi	ADD Immediate	I	0010011	0x0	0x00	rd = rs1 + imm	
andi AND Immediate	xori	XOR Immediate	I	0010011	0x0	0x00	rd = rs1 ^ imm	
slli Shift Left Logical Imm I 0010011 0x1 0x00 rd = rs1 < x imm srli Shift Right Logical Imm I 0010011 0x1 0x00 rd = rs1 >> imm msb-exter slt Set Less Than Imm I 0010011 0x2 rd = (rs1 < imm)?1:0 zero-exter sltu Set Less Than Imm (U) I 0010011 0x3 rd = (rs1 < imm)?1:0 zero-exter lb Load Byte I 0000011 0x0 rd = M[rs1+imm][0:7] zero-exter lw Load Half I 0000011 0x1 rd = M[rs1+imm][0:15] zero-exter lw Load Byte (U) I 0000011 0x2 rd = M[rs1+imm][0:7] zero-exter sb Store Byte S 0100011 0x0 M[rs1+imm][0:7] zero-exter sb Store Byte S 0100011 0x0 M[rs1+imm][0:7] rs2[0:7] sh Store Byte S 0100011 0x0 M[rs1+imm][0:7] rs2[0:7] <tr< td=""><td>ori</td><td>OR Immediate</td><td>I</td><td>0010011</td><td>0x0</td><td>0x00</td><td>rd = rs1 imm</td><td></td></tr<>	ori	OR Immediate	I	0010011	0x0	0x00	rd = rs1 imm	
srli Shift Right Logical Imm I 0010011 0x1 0x00 rd = rs1 >> imm msb-extend srai Shift Right Arith Imm I 0010011 0x3 0x20 rd = rs1 >> imm msb-extend slt Set Less Than Imm I 0010011 0x2 rd = (rs1 < imm)?1:0	andi	AND Immediate	I	0010011	0x0	0x00	rd = rs1 & imm	
srai Shift Right Arith Imm I 0010011 0x3 0x20 rd = rs1 >> imm msb-extend slt Set Less Than Imm I 0010011 0x2 rd = (rs1 < imm)?1:0 zero-extend lb Load Byte I 0000011 0x0 rd = M[rs1+imm][0:7] zero-extend lb Load Byte I 0000011 0x1 rd = M[rs1+imm][0:7] zero-extend lw Load Half I 0000011 0x2 rd = M[rs1+imm][0:31] zero-extend lbu Load Byte (U) I 0000011 0x2 rd = M[rs1+imm][0:7] zero-extend sb Store Byte S 0100011 0x5 rd = M[rs1+imm][0:7] zero-extend sb Store Byte S 0100011 0x0 M[rs1+imm][0:7] = rs2[0:7] sh Store Byte S 0100011 0x1 M[rs1+imm][0:15] = rs2[0:7] sw Store Word S 0100011 0x2 M[rs1+imm][0:31] = rs2[0:31] beq Branch != B	slli	Shift Left Logical Imm	I	0010011	0x1	0x00	rd = rs1 << imm	
slt Set Less Than Imm I 0010011 0x2 rd = (rs1 < imm)?1:0 zero-extent lb Load Byte I 0010011 0x3 rd = (rs1 < imm)?1:0	srli	Shift Right Logical Imm	I	0010011	0x1	0x00	rd = rs1 >> imm	
situ Set Less Than Imm (U) I 0010011 0x3 rd = (rs1 < imm)?1:0 zero-extent 1b Load Byte I 0000011 0x0 rd = M[rs1+imm][0:7] rd = M[rs1+imm][0:7] rd = M[rs1+imm][0:15] rd = M[rs1+imm][0:15] rd = M[rs1+imm][0:31] rd = M[rs1+imm][0:31] rd = M[rs1+imm][0:7] zero-extent rd = M[rs1+imm][0:7] rd = M[rs1+imm	srai		I	0010011	0x3	0x20	rd = rs1 >> imm	msb-extends
Description	slt	Set Less Than Imm	I	0010011	0x2		rd = (rs1 < imm)?1:0	
1h Load Half I 0000011 0x1 rd = M[rs1+imm][0:15] rd = M[rs1+imm][0:31] 1w Load Word I 0000011 0x2 rd = M[rs1+imm][0:31] zero-exter 1bu Load Byte (U) I 0000011 0x4 rd = M[rs1+imm][0:7] zero-exter 1hu Load Half (U) I 0000011 0x5 rd = M[rs1+imm][0:15] zero-exter 1hu Load Half (U) I 0000011 0x5 rd = M[rs1+imm][0:7] zero-exter 1hu Load Half (U) I 0000011 0x0 M[rs1+imm][0:15] zero-exter 1hu Load Half (U) I 0000011 0x0 M[rs1+imm][0:15] zero-exter 1hu Branch = B B 0100011 0x1 M[rs1+imm][0:15] rs2[0:7] M[rs1+imm][0:15] rs2[0:7] M[rs1+imm][0:15] rs2[0:7] mm M[rs1+imm][0:15] rs2[0:7] mm mm if(rs1 = rs2[0:7] mm mm if(rs1 = rs2[0:7] mm mm if(rs1 = rs2[0:7] mm	sltu	Set Less Than Imm (U)	I	0010011	0x3		rd = (rs1 < imm)?1:0	zero-extends
lw Load Word I 0000011 0x2 rd = M[rs1+imm][0:31] zero-exter lbu Load Byte (U) I 0000011 0x4 rd = M[rs1+imm][0:7] zero-exter lhu Load Half (U) I 0000011 0x5 rd = M[rs1+imm][0:15] zero-exter sb Store Byte S 0100011 0x0 M[rs1+imm][0:15] = 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[0:15] sw Store Word S 0100011 0x2 M[rs1+imm][0:31] = rs2[0:31] beq Branch == B 1100011 0x0 if(rs1 == rs2) PC += imm blt Branch <	lb	Load Byte	I	0000011	0x0		rd = M[rs1+imm][0:7]	
lbu Load Byte (U) I 0000011 0x4 rd = M[rs1+imm][0:7] zero-exter 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 <	1h	Load Half	I	0000011	0x1		rd = M[rs1+imm][0:15]	
Ihu 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] M[rs1+imm][0:15] = rs2[0:7] M[rs1+imm][0:15] = rs2[0:7] M[rs1+imm][0:15] = rs2[0:7] M[rs1+imm][0:15] = rs2[0:15] M[rs1+imm][0:15] = rs2[0:15] M[rs1+imm][0:15] = rs2[0:7] M[rs1+imm][0:15] = rs2[0:15] M[rs1+imm][0:15] = rs2[lw	Load Word	I	0000011	0x2		rd = M[rs1+imm][0:31]	
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 zero-extended bltu Branch < (U) B 1100011 0x6 if(rs1 >= rs2) PC += imm zero-extended bgeu Branch ≥ (U) B 1100011 0x7 if(rs1 >= rs2) PC += imm zero-extended jal Jump And Link J 1101111 rd = PC+4; PC += imm rd = PC+4; PC = rs1 lui Load Upper Imm U 0110111 rd = imm << 12	1bu	Load Byte (U)	I	0000011	0x4		rd = M[rs1+imm][0:7]	zero-extends
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 < (U) B 1100011 0x5 if(rs1 <= rs2) PC += imm zero-extended bgeu Branch ≥ (U) B 1100011 0x7 if(rs1 >= rs2) PC += imm zero-extended jal Jump And Link J 1101111 rd = PC+4; PC += imm zero-extended lui Load Upper Imm U 0110111 volume 12	1hu	Load Half (U)	I	0000011	0x5		rd = M[rs1+imm][0:15]	zero-extends
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 <	sb	Store Byte	S	0100011	0x0		M[rs1+imm][0:7] = rs2[0:7]	
beq Branch == B 1100011 0x0 if(rs1 == rs2) PC += imm bne Branch!= B 1100011 0x1 if(rs1 != rs2) PC += imm blt Branch <	sh	Store Half	S	0100011	0x1		M[rs1+imm][0:15] = rs2[0:15]	
bne Branch!= B 1100011 0x1 if(rs1 != rs2) PC += imm blt Branch <	SW	Store Word	S	0100011	0x2		M[rs1+imm][0:31] = rs2[0:31]	
blt Branch < B 1100011 0x4 if(rs1 < rs2) PC += imm pc bge Branch ≤ B 1100011 0x5 if(rs1 >= rs2) PC += imm pc bltu Branch < (U)	beq	Branch ==	В	1100011	0x0		if(rs1 == rs2) PC += imm	
bge Branch ≤ B 1100011 0x5 if(rs1 >= rs2) PC += imm zero-extend bltu Branch < (U)	bne	Branch !=	В	1100011	0x1		if(rs1 != rs2) PC += imm	
bltu Branch < (U) B 1100011 0x6 if(rs1 < rs2) PC += imm zero-exter bgeu Branch ≥ (U) B 1100011 0x7 if(rs1 >= rs2) PC += imm zero-exter 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	blt	Branch <	В	1100011	0x4		if(rs1 < rs2) PC += imm	
bgeu Branch ≥ (U) B 1100011 0x7 if(rs1 >= rs2) PC += imm zero-extend jal Jump And Link J 1101111 rd = PC+4; PC += imm rd = PC+4; PC = rs1 lui Load Upper Imm U 0110111 rd = imm << 12	bge	Branch ≤	В	1100011	0x5		if(rs1 >= rs2) PC += imm	
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	bltu	Branch < (U)	В	1100011	0x6		if(rs1 < rs2) PC += imm	zero-extends
jalr Jump And Link Reg I 1100111 0x0 rd = PC+4; PC = rs1 lui Load Upper Imm U 0110111 rd = imm << 12	bgeu	Branch \geq (U)	В	1100011	0x7		if(rs1 >= rs2) PC += imm	zero-extends
lui Load Upper Imm U 0110111 rd = imm << 12	jal	Jump And Link	J	1101111			rd = PC+4; PC += imm	
	jalr		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)	auipc	Add Upper Imm to PC	U	0010111			rd = PC + (imm << 12)	
ecall Environment Call I 1110011 0x0 0x00 Transfer control to OS imm: 0x0	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: 0x0	ebreak	Environment Break	I	1110011	0x0	0x00	Transfer control to debugger	imm: 0x001

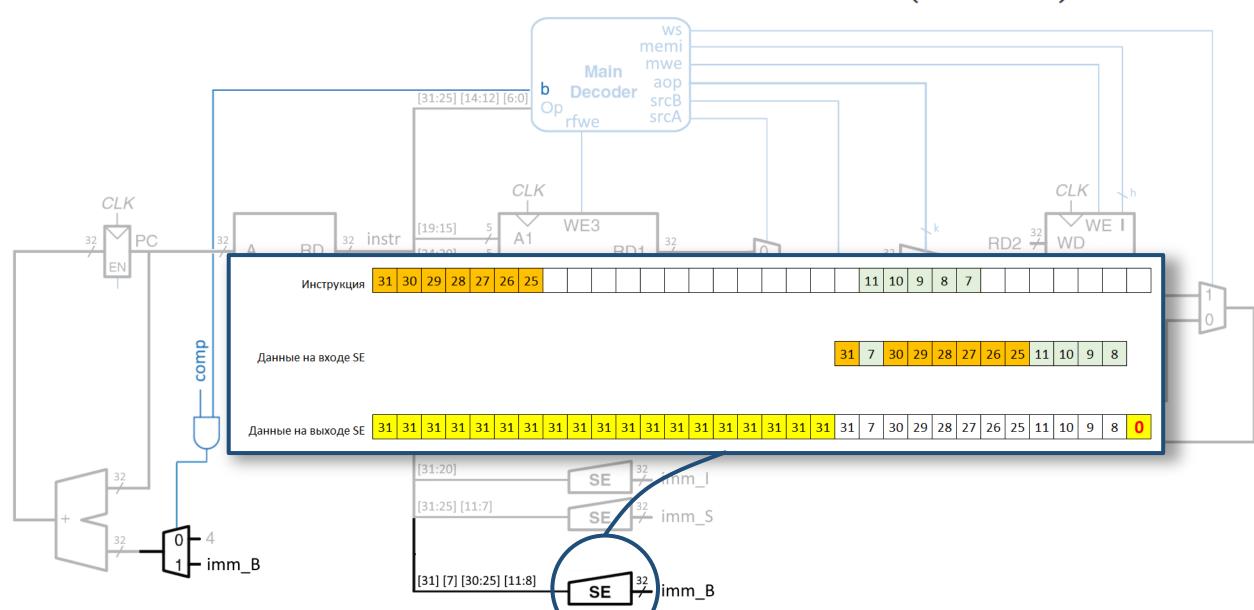
31	27	26	25	24		20	19	15	14	12	11	7	6	0	
imn	n[12 10]):5]			rs2		rs	1	fun	ct3	imm	4:1 11]	opo	code	B-type
imn	n[12 10]	0:5]			rs2		rs	1	00	00	imm	[4:1 11]	110	0011	BEQ



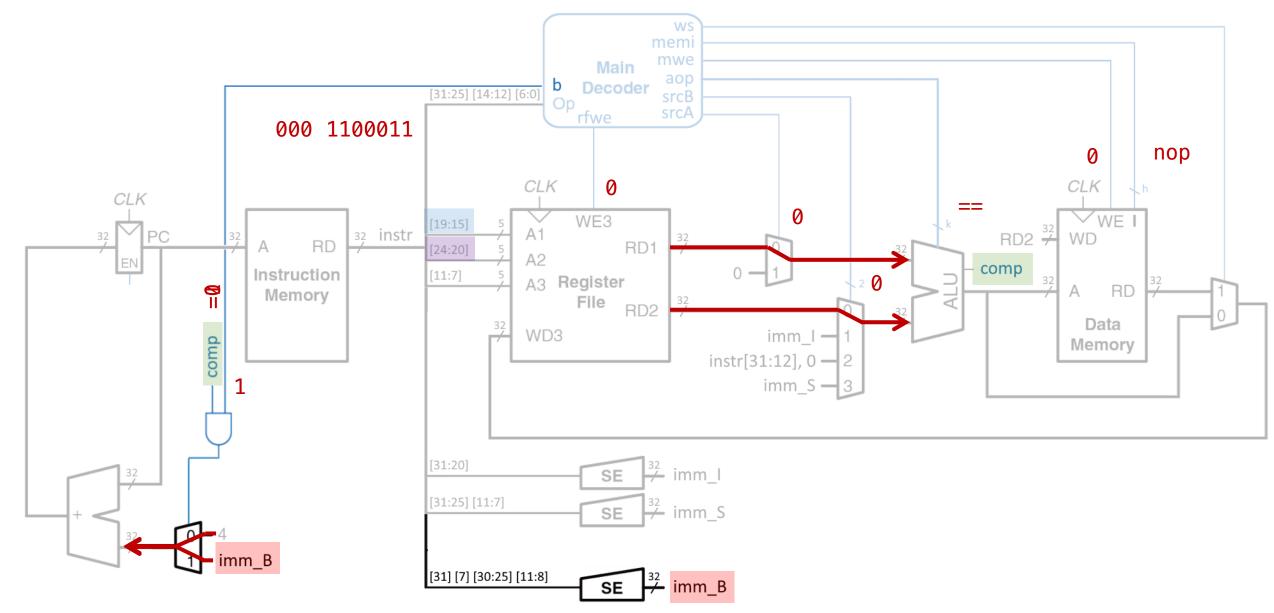
31	27	26	25	24		20	19	15	14	12	11	7	6	0	
imn	n[12 10]):5]			rs2		rs	1	fun	ct3	imm[4:1 11]	opo	code	B-type
imn	n[12 10]	0:5]			rs2		rs	1	00	00	imm	[4:1 11]	110	0011	BEQ



31	27	26	25	24	20	19	15	14	12	11	7	6	0	
imn	n[12 10]	0:5]			rs2	rs	1	fun	ct3	imm[4	4:1 11]	opce	ode	B-type
imn	n[12 10]	0:5]			rs2	rs	1	00	00	imm[4:1 11]	1100	0011	m BEQ



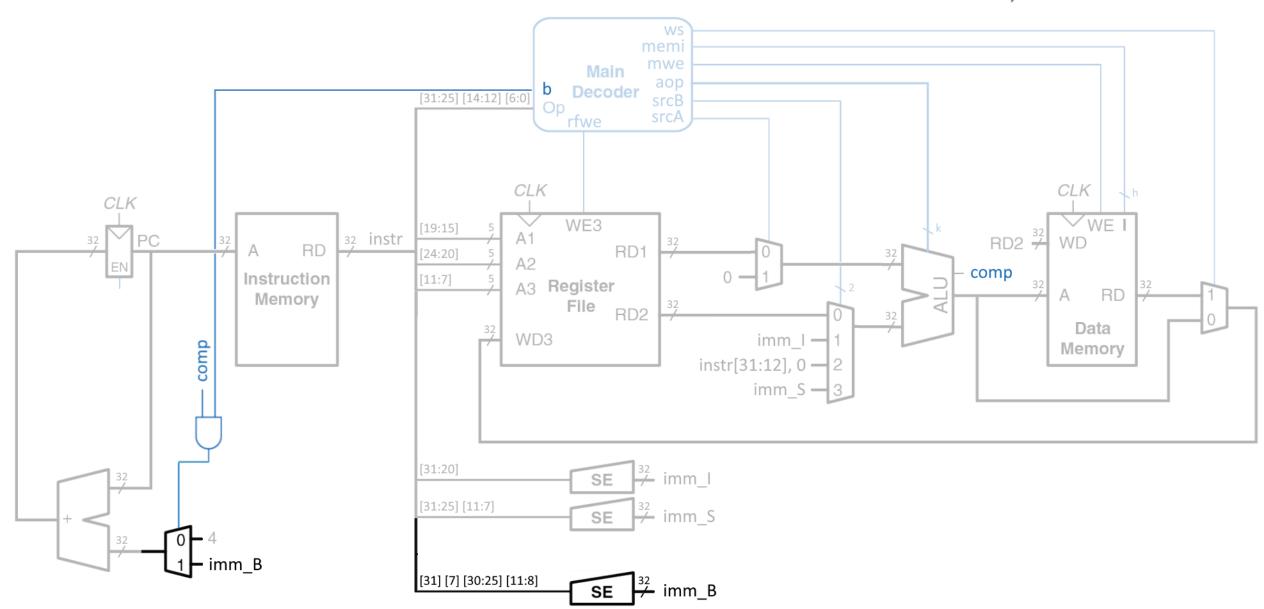
31	27	26	25	24		20	19	15	14	12	11	7	6	0	
imn	n[12 10]	0:5]			rs2		rs	1	fun	ct3	imm	[4:1 11]	op	code	B-type
imn	n[12 10]	0:5]			rs2		rs	1	00	00	imm	[4:1 11]	110	00011	BEQ



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]	
1h	Load Half	I	0000011	0x1		rd = M[rs1+imm][0:15]	
lw	Load Word	I	0000011	0x2		rd = M[rs1+imm][0:31]	
1bu	Load Byte (U)	I	0000011	0x4		rd = M[rs1+imm][0:7]	zero-extends
1hu	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 ==	В	1100011	0x0		if(rs1 == rs2) PC += imm	
bne	Branch !=	В	1100011	0x1		if(rs1 != rs2) PC += imm	
blt	Branch <	В	1100011	0x4		if(rs1 < rs2) PC += imm	
bge	Branch ≤	В	1100011	0x5		if(rs1 >= rs2) PC += imm	
bltu	Branch < (U)	В	1100011	0x6		if(rs1 < rs2) PC += imm	zero-extends
bgeu	Branch \geq (U)	В	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
- CDT CUIK	Zii.iioiiiiciit Dicar	1		ONO	UNUU	de. control to debugger	

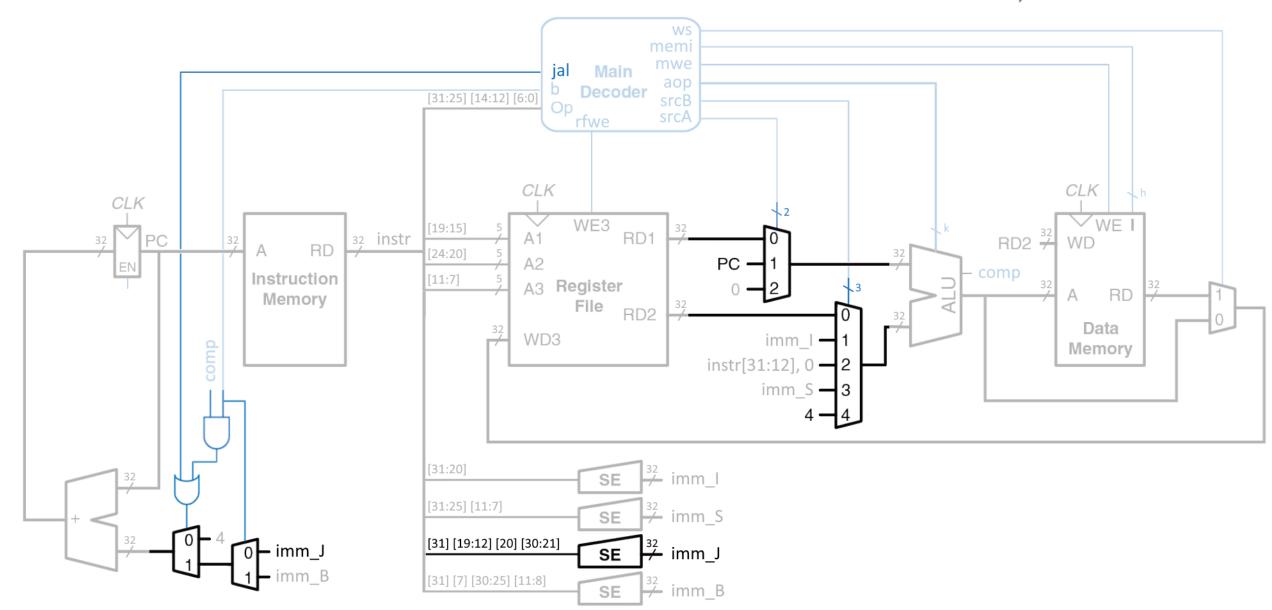
31	27	26	25	24	20 1	9 15	14	12	11	7	6	0	
			imn	n[20]	10:1 11 19:12	2]]	rd	opo	code	J-type
			imn	n[20]	10:1 11 19:12	2]]	rd	110	1111	JAL

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



31	27	26	25	24	20)	19	15	14	12	11	7	6	0	
			imn	n[20	10:1 11	19:	12]					rd	opo	code	J-type
			imn	n[20]	10:1 11	19:	12]					rd	110	1111	JAL

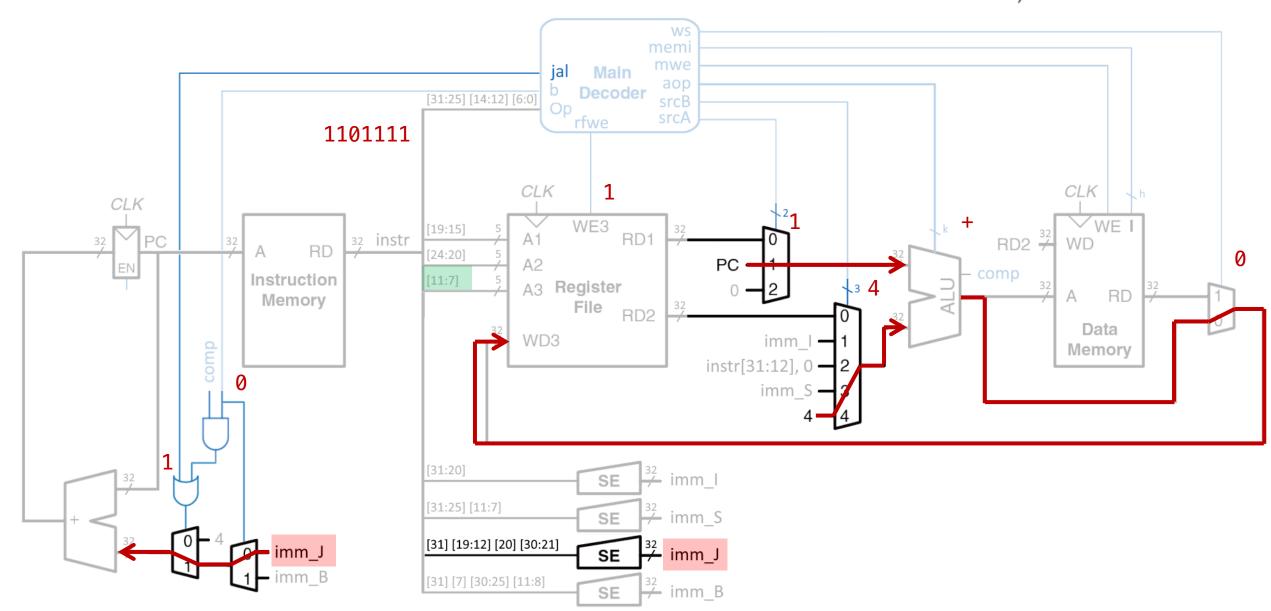
jal xN, label
rd = PC+4; PC += imm



31	27 26	$\frac{6 25 24}{\text{imm}[20]}$	$ \begin{array}{ccc} & 20 & 19 \\ 0 10:1 11 19:12 \end{array} $		14 12	11 rd	7	6 opcode	0 J-typ	oe iolyn labol
		imm[2	0 10:1 11 19:12			rd		1101111	JAL	jal xN, label
				•						rd = PC+4; PC += imm
				[31:	:25] [14:12] [6	lh n	Main ecode ve	ws memi mwe aop srcB srcA		
CL	_K					CLK				CLK h
32 E	PC	3	Инструкция	31 30 29	28 27 26	25 24 23	3 22 2:	1 20 19 18 17	16 15 14	1 13 12
		du	Данные на входе SE					31 19 18 17	7 16 15 14	1 13 12 20 30 29 28 27 26 25 24 23 22 21
		comp	ļанные на выходе SE	31 31 31	31 31 31	31 31 31	31 3:	1 31 19 18 17	7 16 15 14	1 13 12 20 30 29 28 27 26 25 24 23 22 21 0
+ 3	32		– imm_J – imm_B	[31]	:20] :25] [11:7]] [19:12] [20] [] [7] [30:25] [1	$\neg au$	SE SE SE	32/imm_I 32/imm_S 32/imm_J 32/imm_B		

31	27	26	25	24		20	19	15	1	4	12	11		7	6	0	
			imn	n[20]	10:1 1	11 19	9:12]						rd		op	code	J-type
			imn	n[20]	10:1	11 19	9:12]						rd		110	01111	JAL

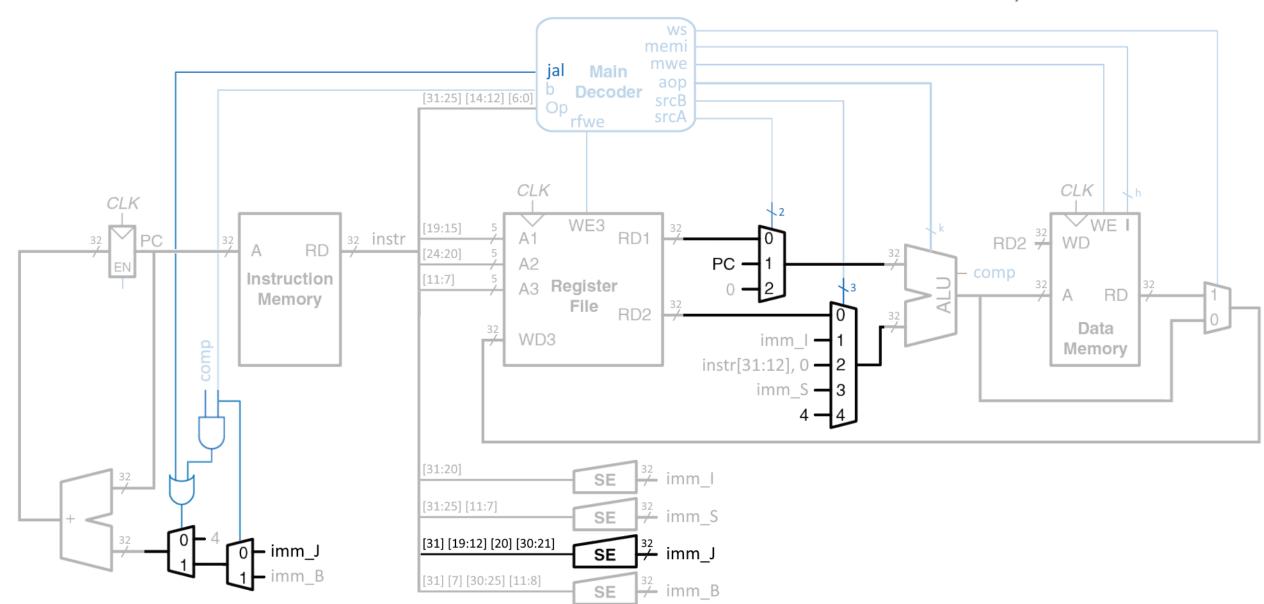
jal xN, label
rd = PC+4; PC += imm



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
1b	Load Byte	I	0000011	0x0		rd = M[rs1+imm][0:7]	
1h	Load Half	I	0000011	0x1		rd = M[rs1+imm][0:15]	
lw	Load Word	I	0000011	0x2		rd = M[rs1+imm][0:31]	
1bu	Load Byte (U)	I	0000011	0x4		rd = M[rs1+imm][0:7]	zero-extends
1hu	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]	
beg	Branch ==	В	1100011	0x0		if(rs1 == rs2) PC += imm	
bne	Branch !=	В	1100011	0x1		if(rs1 != rs2) PC += imm	
blt	Branch <	В	1100011	0x4		if(rs1 < rs2) PC += imm	
bge	Branch <	В	1100011	0x5		if(rs1 >= rs2) PC += imm	
bltu	Branch < (U)	В	1100011	0x6		if(rs1 < rs2) PC += imm	zero-extends
bgeu	Branch \geq (U)	В	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
32. 0010					00	control to dobugger	

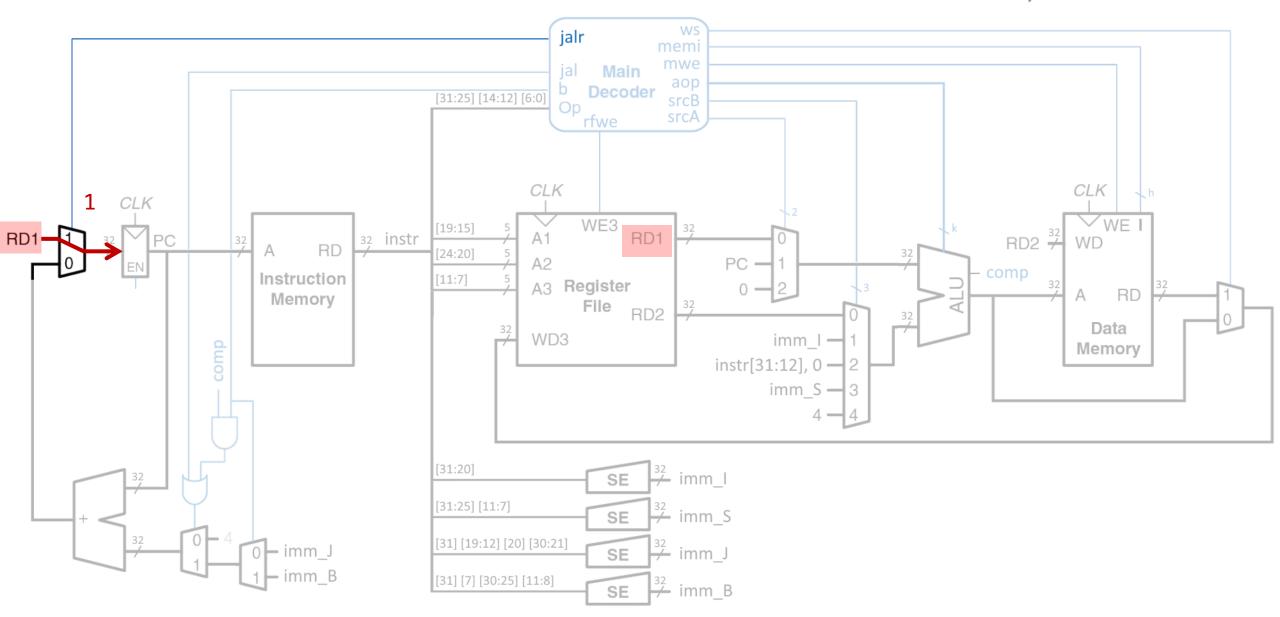
31 27 26 25 24 20	19 15	14 12	11 7	6 0	
imm[11:0]	rs1	funct3	rd	opcode] I-type
imm[11:0]	rs1	000	rd	1100111	JALR

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



31 27 26 2	5 24 20	19 15	5 14 12	11	7 6 0	
imm[11	.:0]	rs1	funct3	rd	opcode] I-type
imm[11	.:0]	rs1	000	rd	1100111	JALR

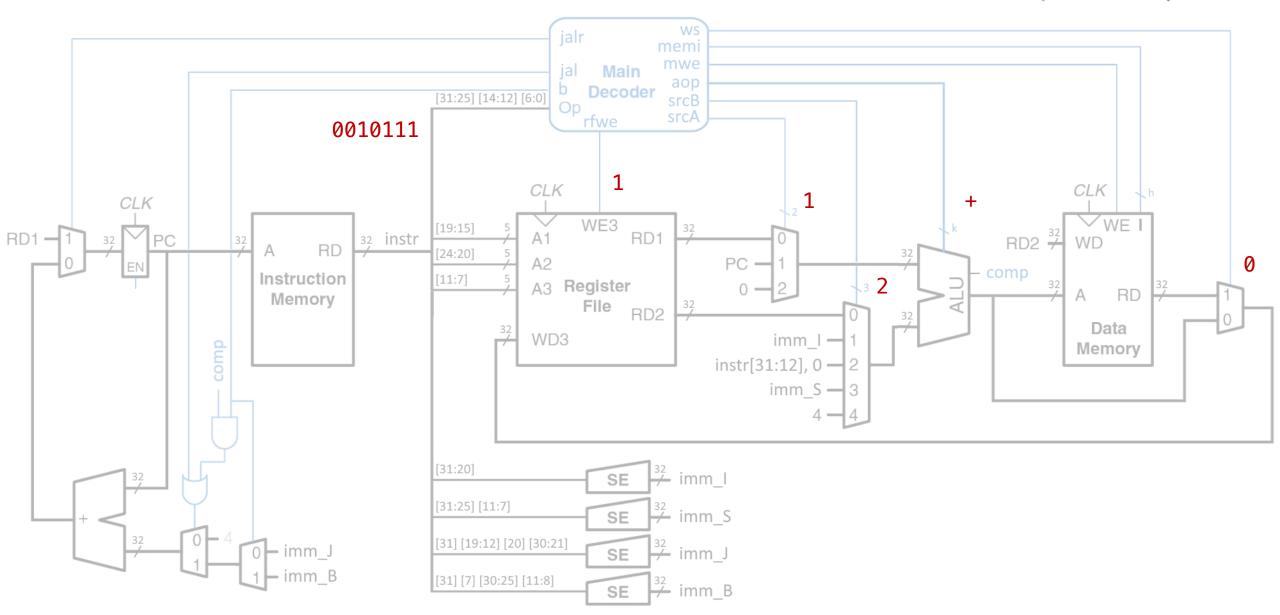
jalr xN, xK
rd = PC+4; PC = rs1



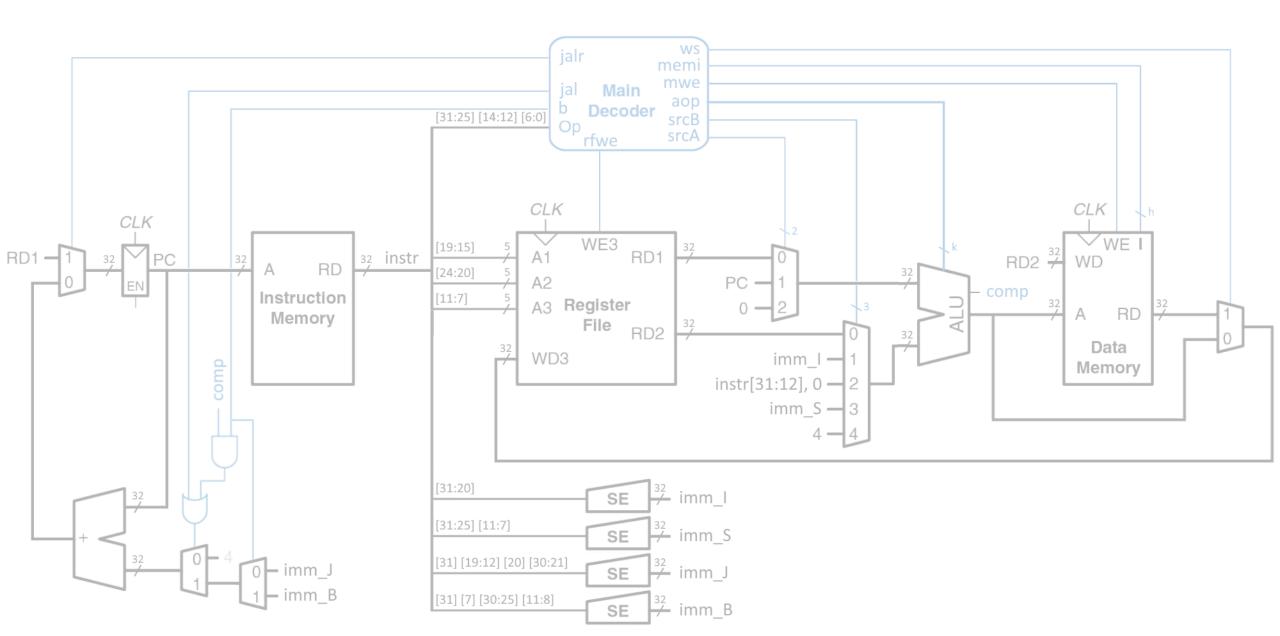
Inst	Name	FMT	Opcode	F3	F7	Description (C)	Note
add	ADD	R	0000011	0x0	0x00	rd = rs1 + rs2	
sub	SUB	R	0000011	0x0	0x20	rd = rs1 - rs2	
xor	XOR	R	0000011	0x4	0x00	rd = rs1 ^ rs2	
or	OR	R	0000011	0x6	0x00	rd = rs1 rs2	
and	AND	R	0000011	0x7	0x00	rd = rs1 & rs2	
sll	Shift Left Logical	R	0000011	0x1	0x00	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
1b	Load Byte	I	0000011	0x0		rd = M[rs1+imm][0:7]	
1h	Load Half	I	0000011	0x1		rd = M[rs1+imm][0:15]	
lw	Load Word	I	0000011	0x2		rd = M[rs1+imm][0:31]	
1bu	Load Byte (U)	I	0000011	0x4		rd = M[rs1+imm][0:7]	zero-extends
1hu	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]	
beg	Branch ==	В	1100011	0x0		if(rs1 == rs2) PC += imm	
bne	Branch !=	В	1100011	0x1		if(rs1 != rs2) PC += imm	
blt	Branch <	В	1100011	0x4		if(rs1 < rs2) PC += imm	
bge	Branch <	В	1100011	0x5		if(rs1 >= rs2) PC += imm	
bltu	Branch < (U)	В	1100011	0x6		if(rs1 < rs2) PC += imm	zero-extends
bgeu	Branch \geq (U)	В	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
					0,,00	comment to acougnit	

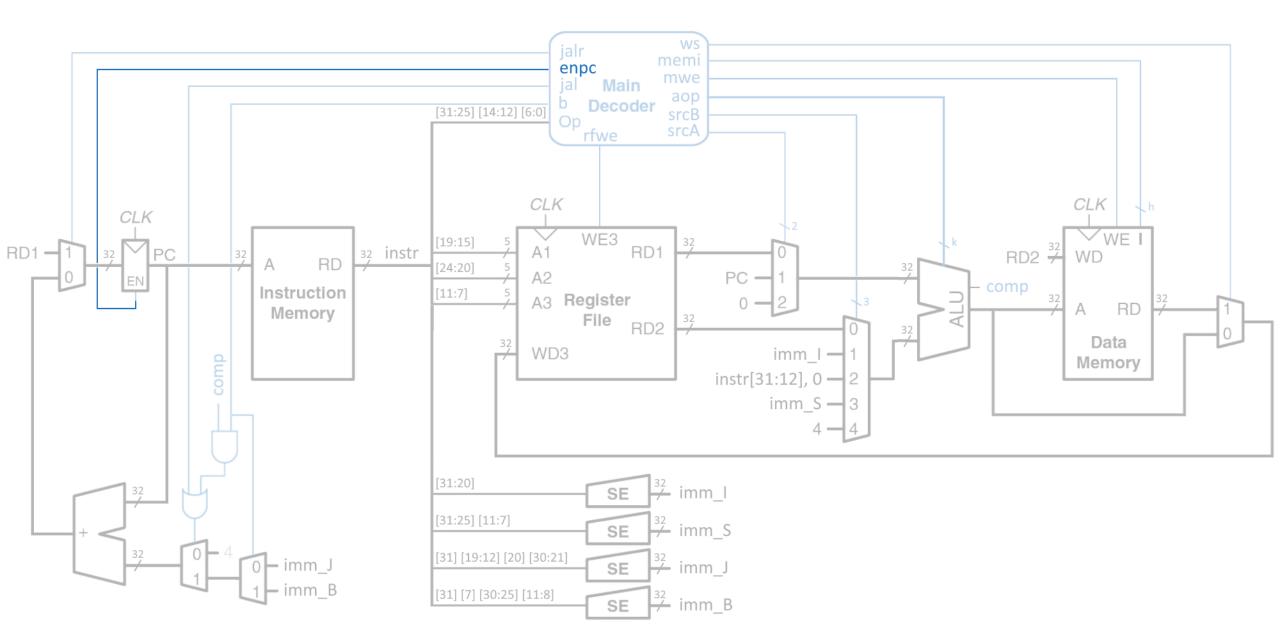
31	27	26	25	24	20	19	15	14	12	11	7	6	0	
				imı	m[31:12]					$^{\mathrm{r}}$	d	opo	code	U-type
				imr	m[31:12]					r	d	0010)111	AUIPC

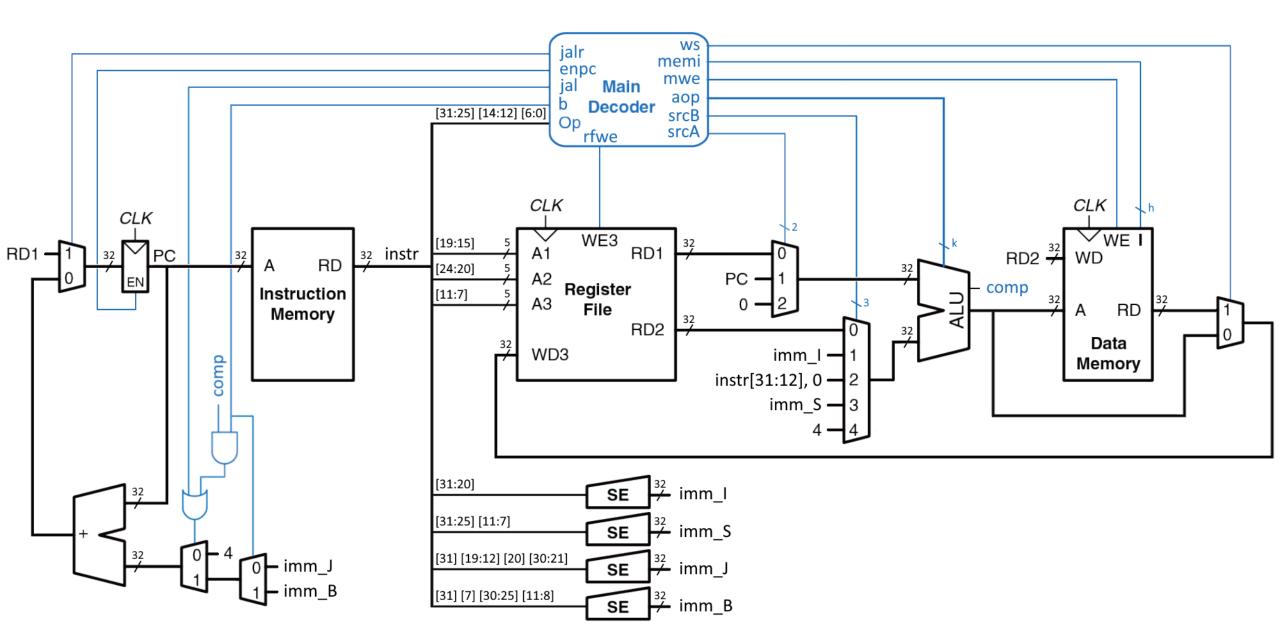
auipc xN, label
rd = PC + (imm << 12)</pre>



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
1b	Load Byte	I	0000011	0x0		rd = M[rs1+imm][0:7]	
1h	Load Half	I	0000011	0x1		rd = M[rs1+imm][0:15]	
1w	Load Word	I	0000011	0x2		rd = M[rs1+imm][0:31]	
1bu	Load Byte (U)	I	0000011	0x4		rd = M[rs1+imm][0:7]	zero-extends
1hu	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 ==	В	1100011	0x0		if(rs1 == rs2) PC += imm	
bne	Branch !=	В	1100011	0x1		if(rs1 != rs2) PC += imm	
blt	Branch <	В	1100011	0x4		if(rs1 < rs2) PC += imm	
bge	Branch ≤	В	1100011	0x5		if(rs1 >= rs2) PC += imm	
bltu	Branch < (U)	В	1100011	0x6		if(rs1 < rs2) PC += imm	zero-extends
bgeu	Branch \geq (U)	В	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

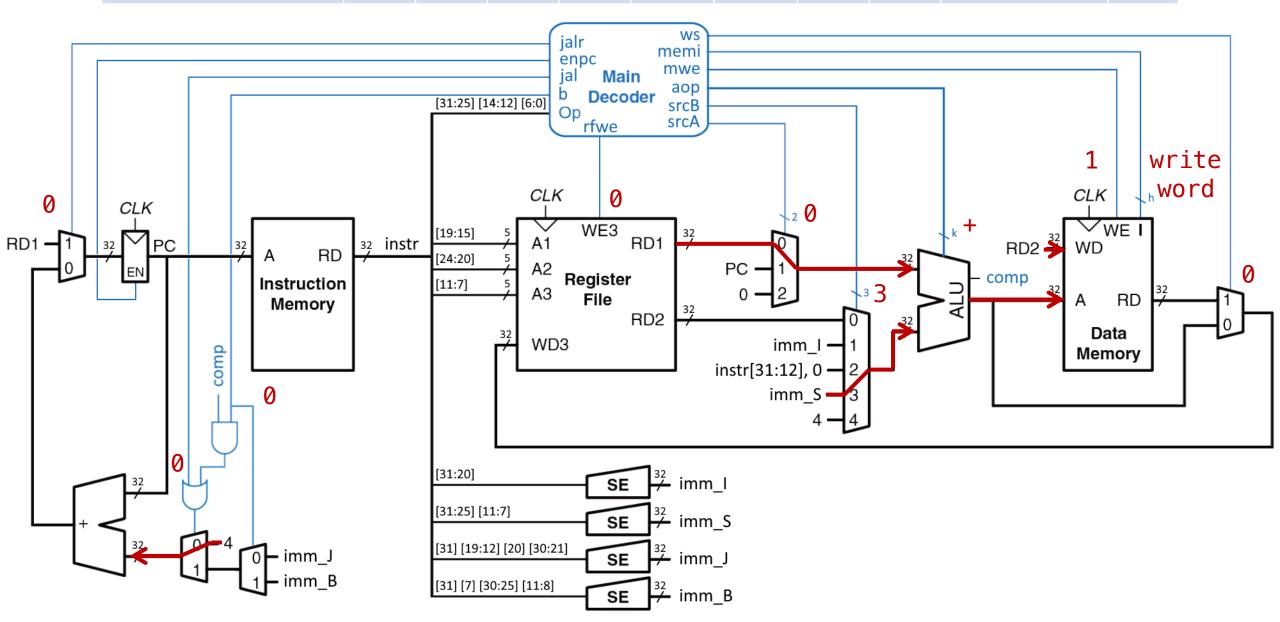






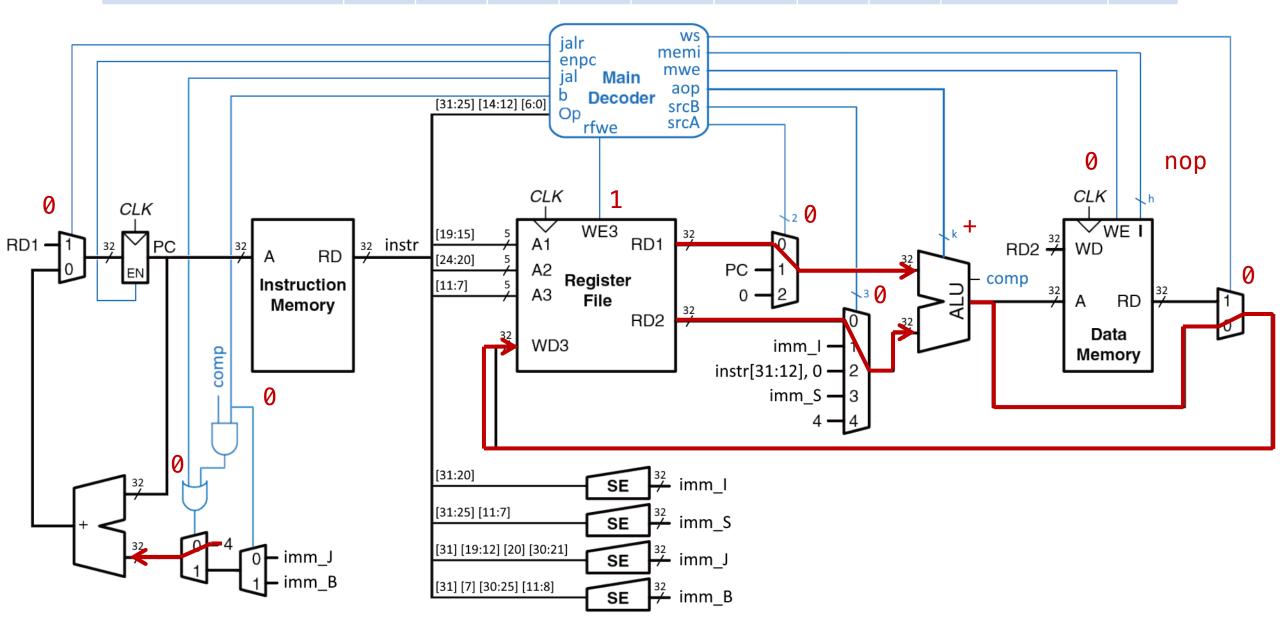
func7, func3, opcode jalr jal rfwe b srcA srcB memi aop mwe WS 010 0100011 0 0 011 write word 0 00 0 0 +

SW

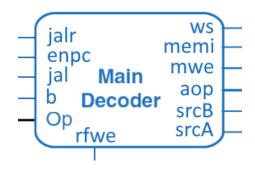


add

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



Дешифратор инструкций

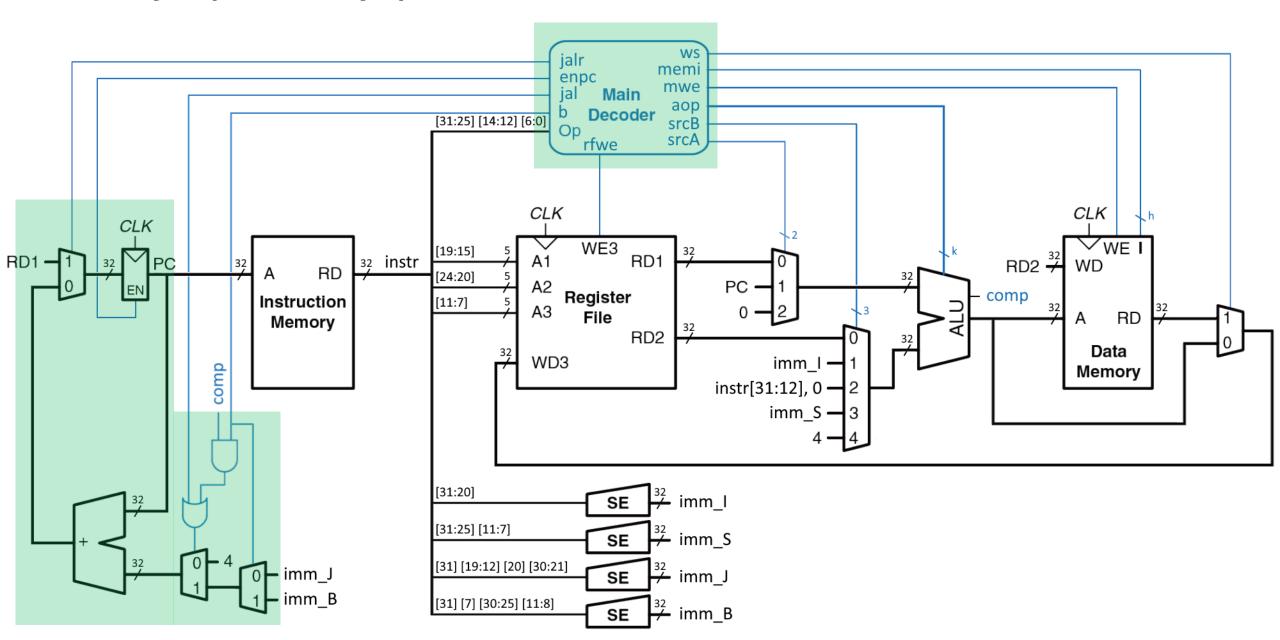


SW	
add	

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

...

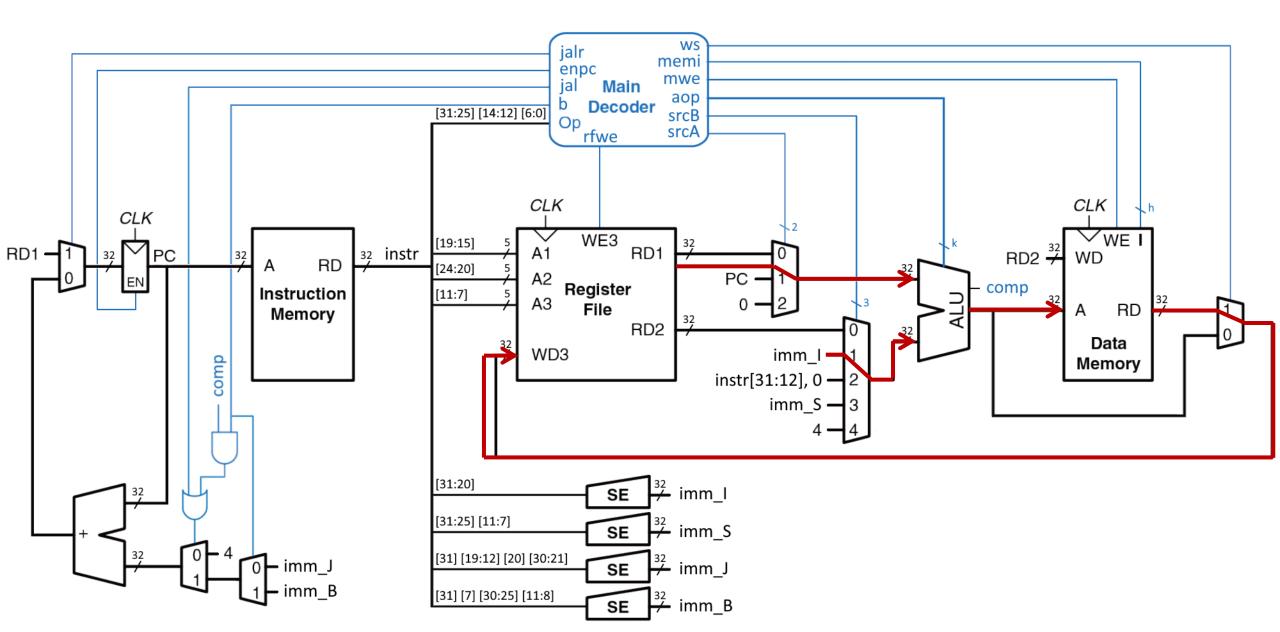
Где устройство управления?



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	$t_{\sf pcq}$	30
Время предустановки регистра	$t_{\sf setup}$	20
Мультиплексор	t_{mux}	25
АЛУ	t_{ALU}	200
Чтение из памяти	$t_{\sf mem}$	250
Чтение из регистрового файла	t_{RFread}	150
Время предустановки регистрового файла (register file setup)	t <i>RF</i> _{setup}	20

$$T_c = t_{pcq_PC} + 2t_{mem} + t_{RFread} + t_{ALU} + t_{mux} + t_{RFsetup}$$

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

$$T_1$$
 = (100 × 10⁹ команд) (1 такт/команду) (925 × 10⁻¹² с/такт) = 92,5 с.