Designing a datapath

Digital Systems – Lecture 21

Department of
COMPUTER
SCIENCE

In this part

Stage-by-stage development of a *single-cycle* datapath able to execute many Thumb instructions.

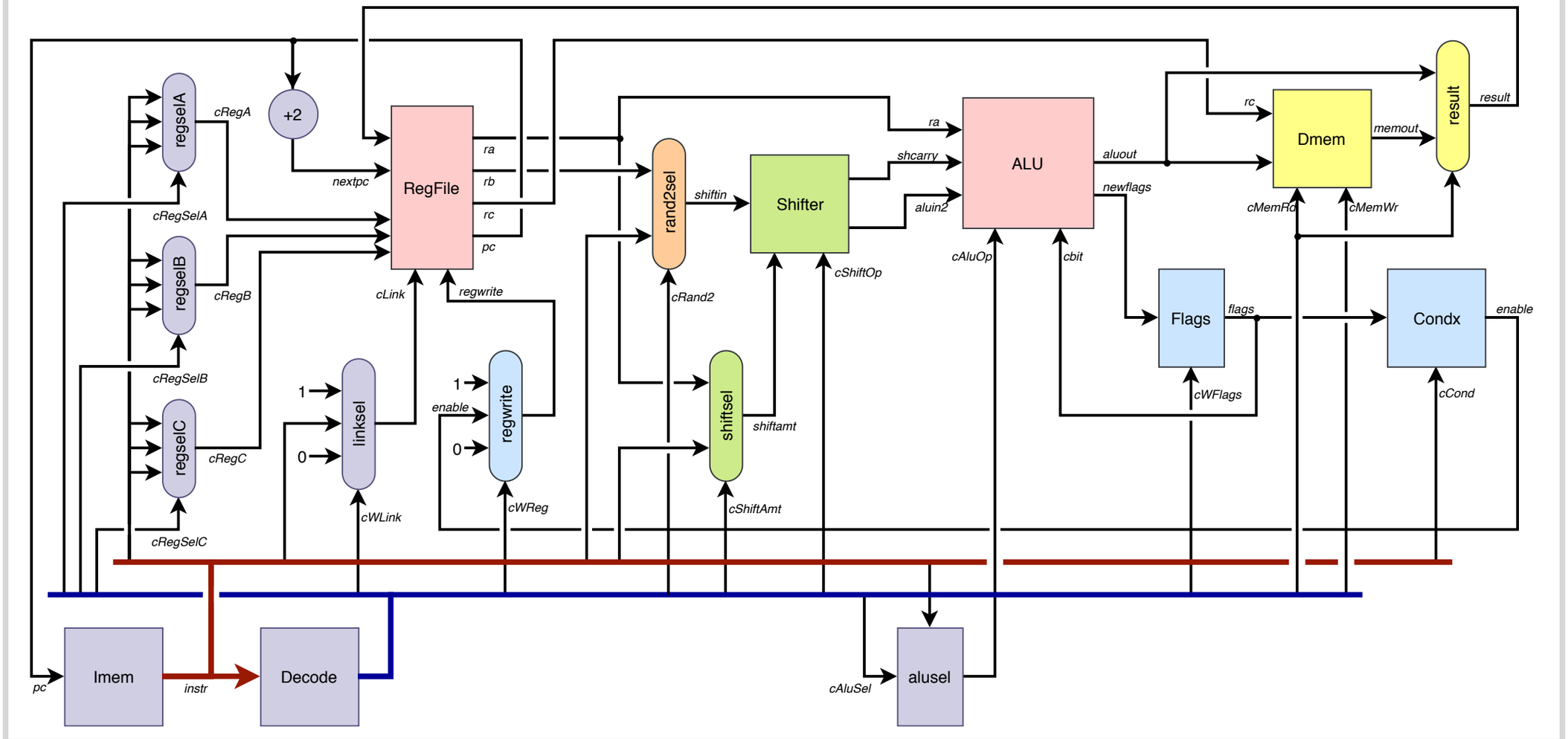
- We won't have pipelined fetch and decode – so the clock speed will be limited.
- We won't implement the control needed for multi-cycle instructions like `push` and `pop`.
- And no interrupts, either.

In this lecture

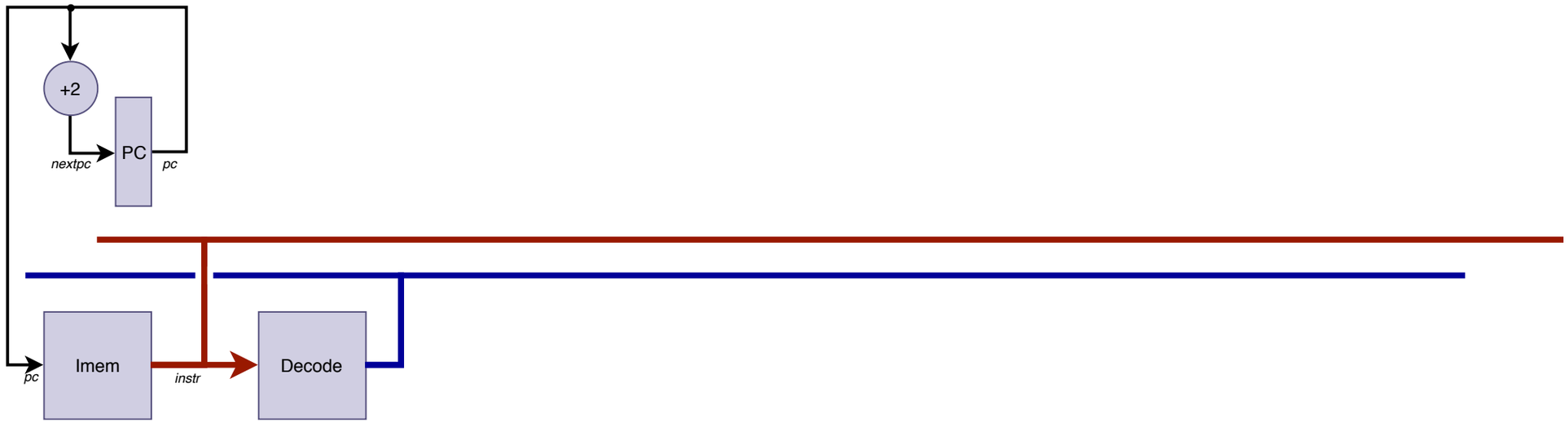
Stage-by-stage development of a datapath design:

- Fetching and decoding
- Arithmetic between registers
- Immediate operands
- Load and store instructions
- Shifts

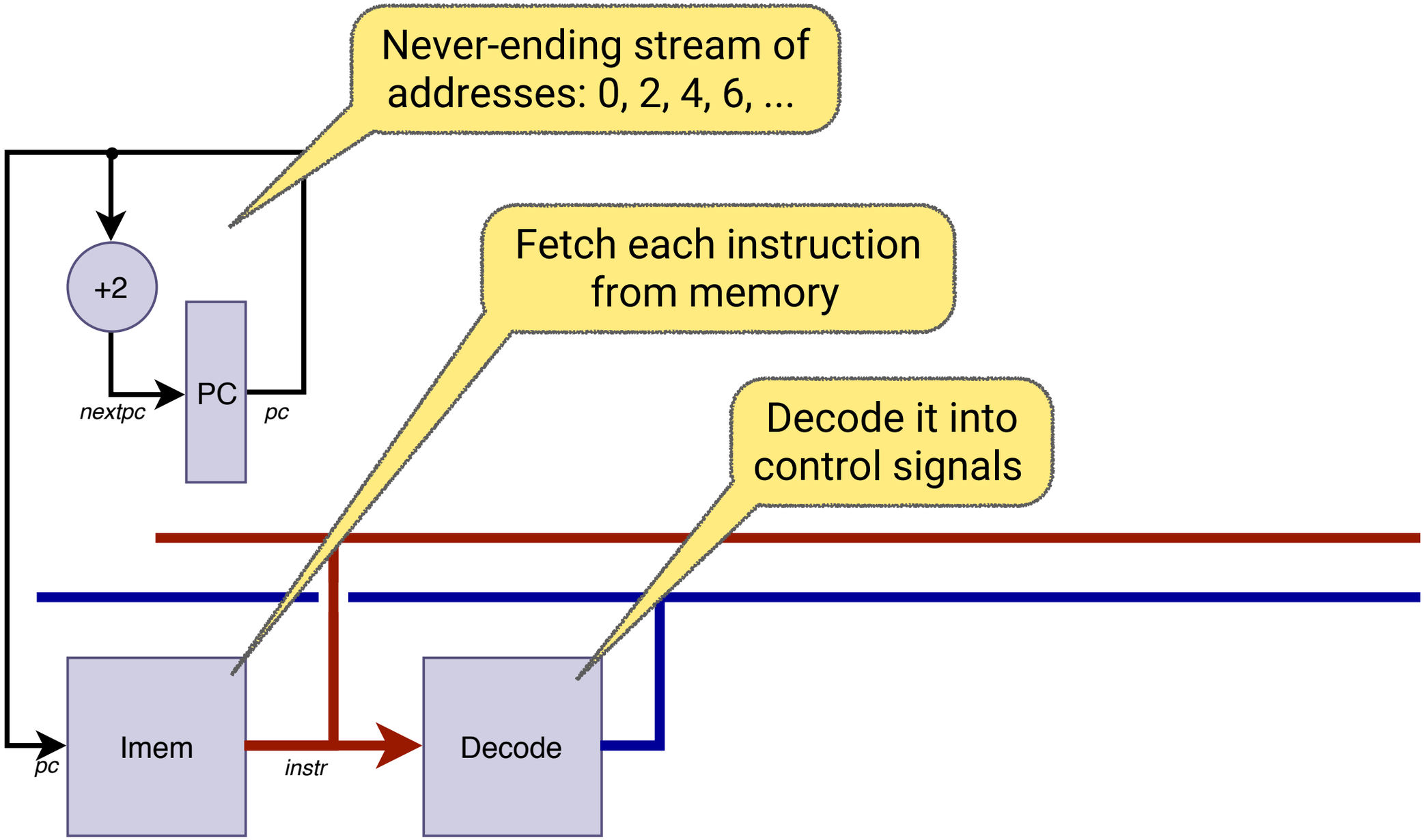
The complete datapath



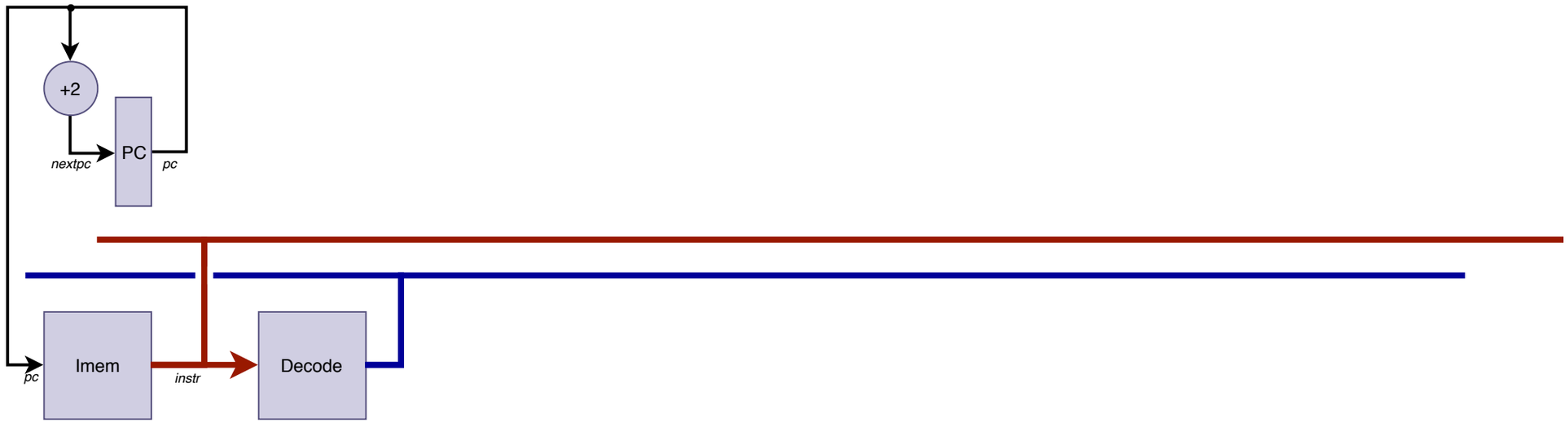
Stage 1: Instruction fetch



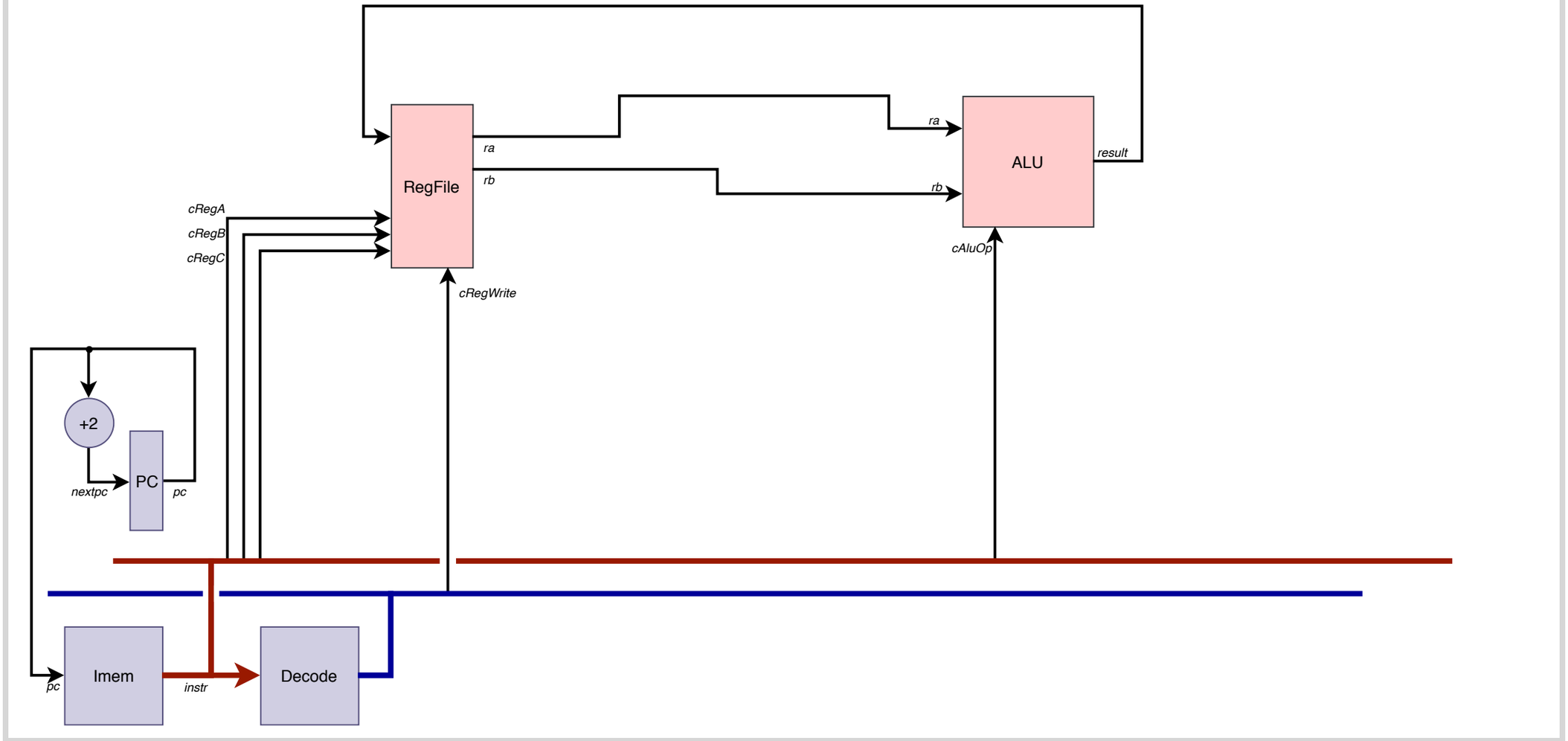
Stage 1: Instruction fetch



Stage 1: Instruction fetch



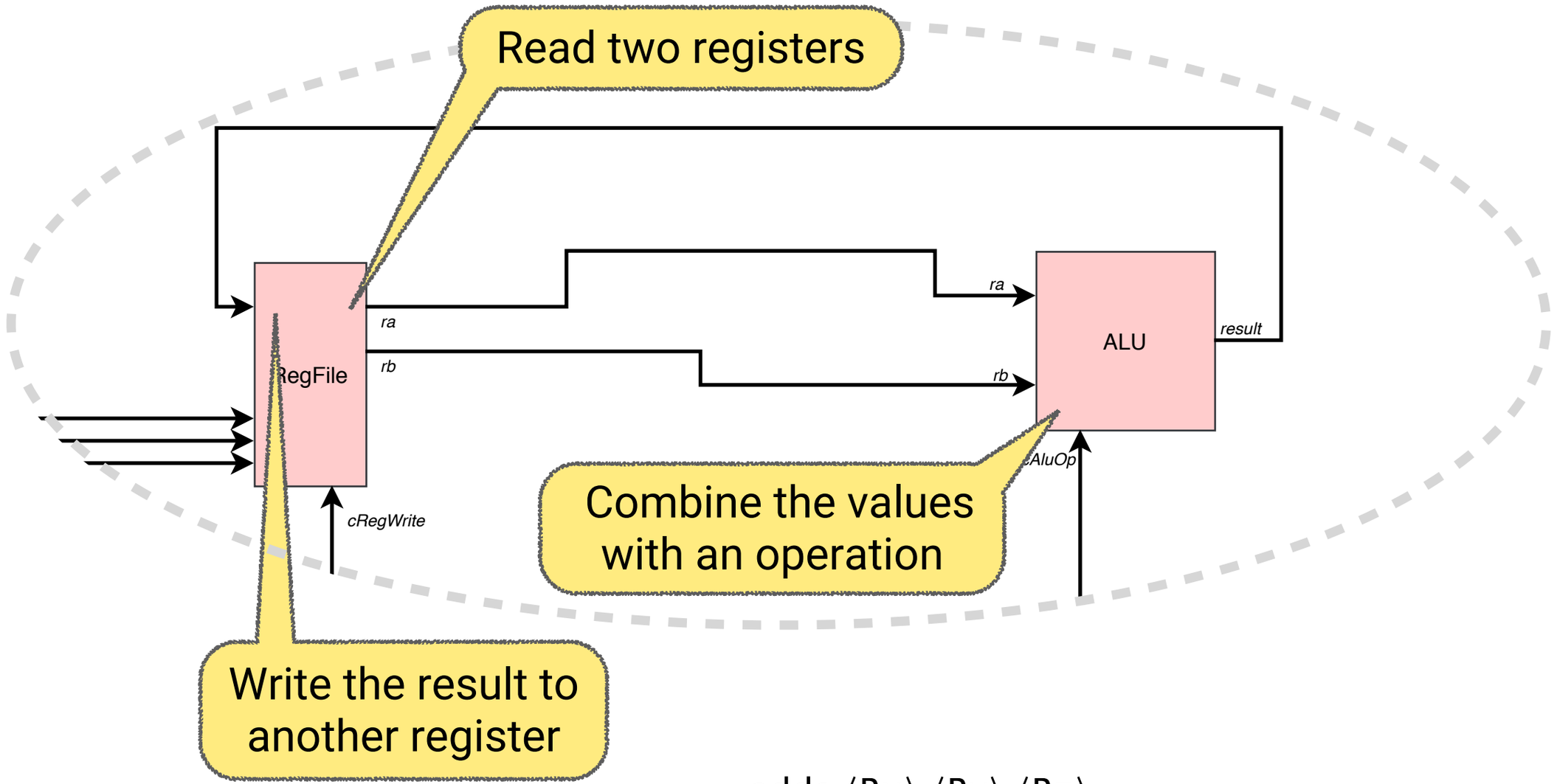
Stage 2: ALU operations



adds ⟨Rx⟩,⟨Ry⟩,⟨Rz⟩

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	1	1	0	0	Rz			Ry			Rx		

Stage 2: ALU operations



adds ⟨Rx⟩,⟨Ry⟩,⟨Rz⟩

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	1	1	0	0	Rz			Ry			Rx		

Reg-to-reg operations

adds ⟨Rx⟩,⟨Ry⟩,⟨Rz⟩

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	1	1	0	0	Rz			Ry			Rx		

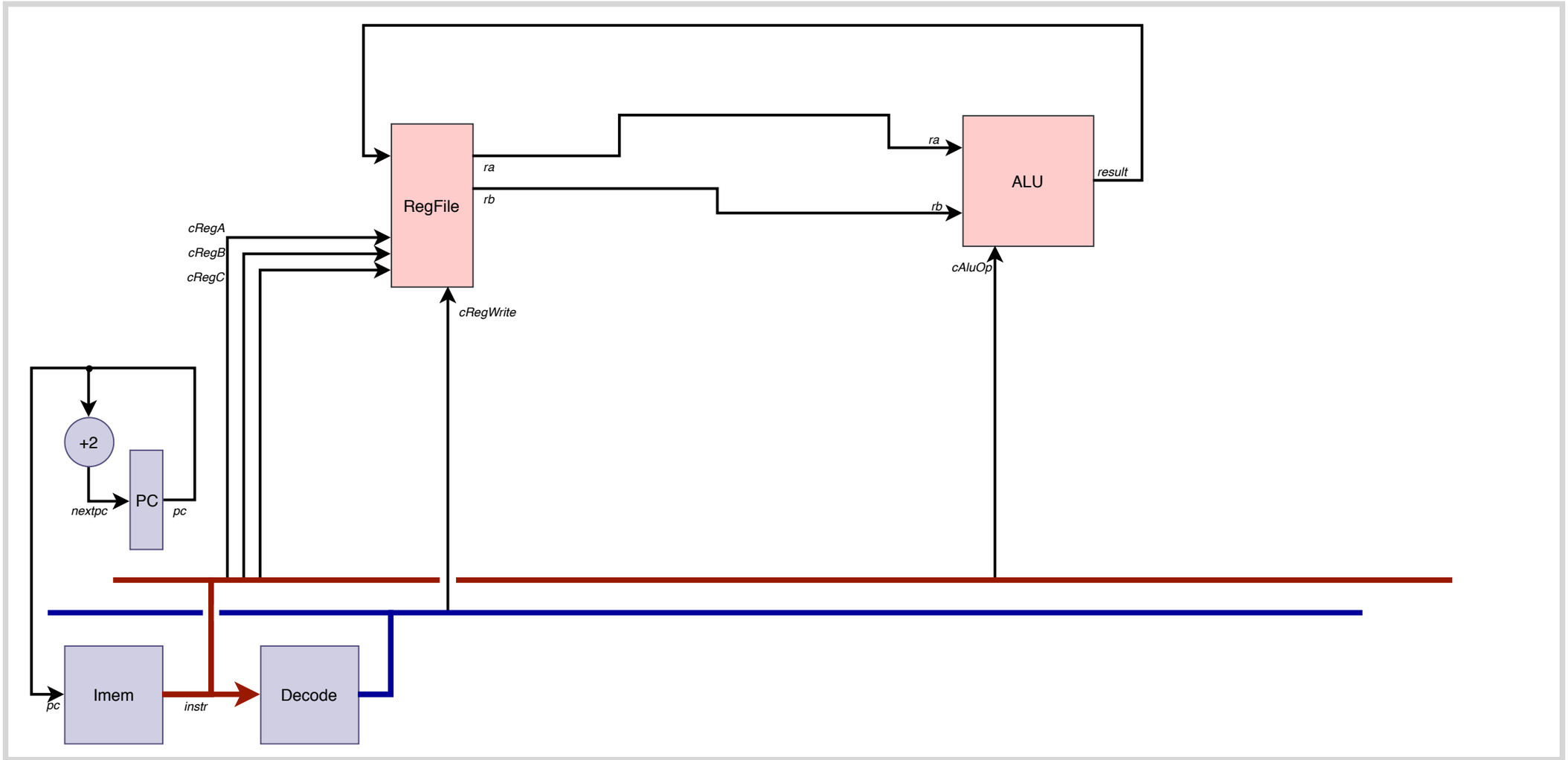
subs ⟨Rx⟩,⟨Ry⟩,⟨Rz⟩

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	1	1	0	1	Rz			Ry			Rx		

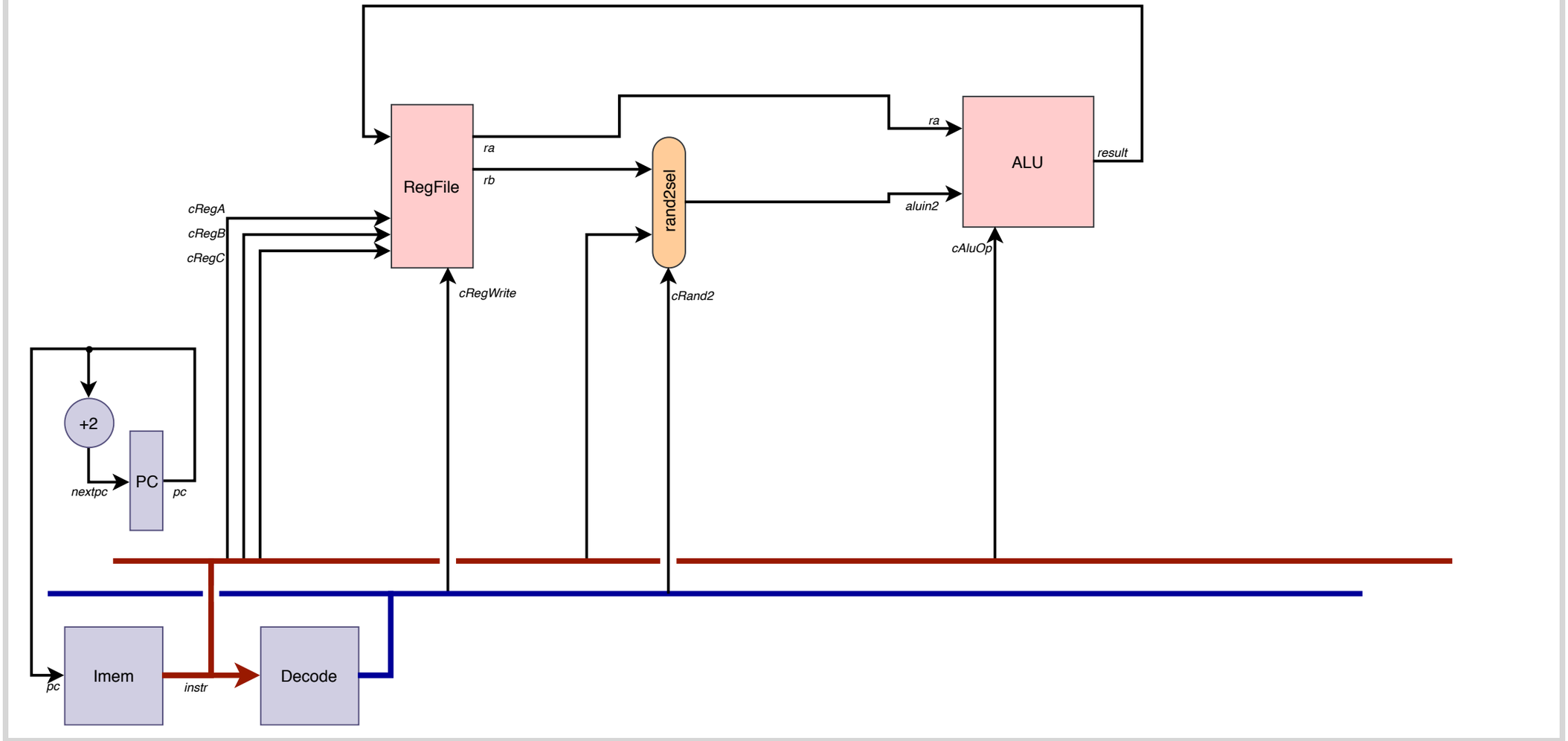
ands ⟨Rx⟩,⟨Ry⟩

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	0	0	0	0	0	0	Ry			Rx		

Stage 2: ALU operations



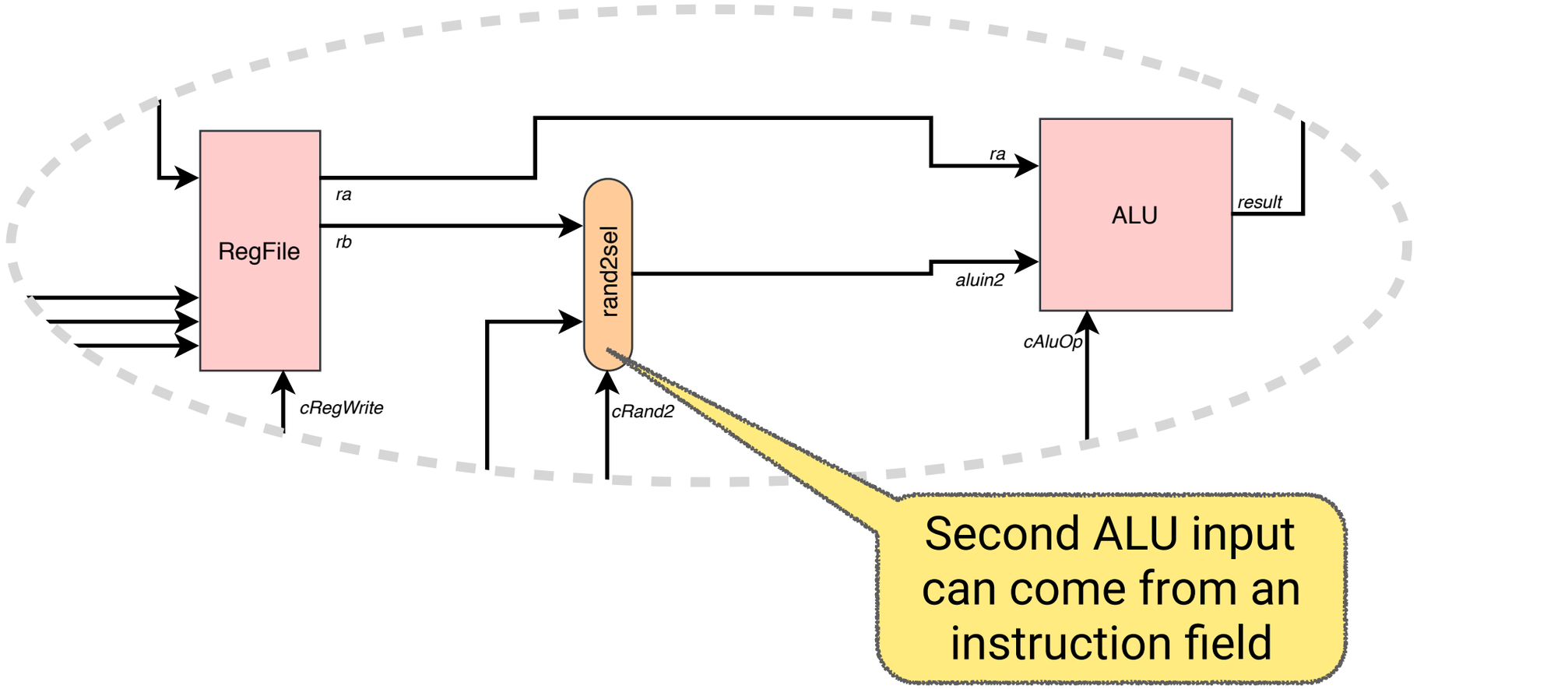
Stage 3: Immediate operands



adds ⟨Rx⟩,⟨Ry⟩,#⟨imm3⟩

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	1	1	1	0	imm3			Ry			Rx		

Stage 3: Immediate operands



adds ⟨Rx⟩,⟨Ry⟩,#⟨imm3⟩

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	1	1	1	0	imm3			Ry			Rx		

Operations with immediate field

adds ⟨Rx⟩,⟨Ry⟩,#⟨imm3⟩

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	1	1	1	0	imm3			Ry			Rx		

adds ⟨Rw⟩,#⟨imm8⟩

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	1	1	0	Rw			imm8							

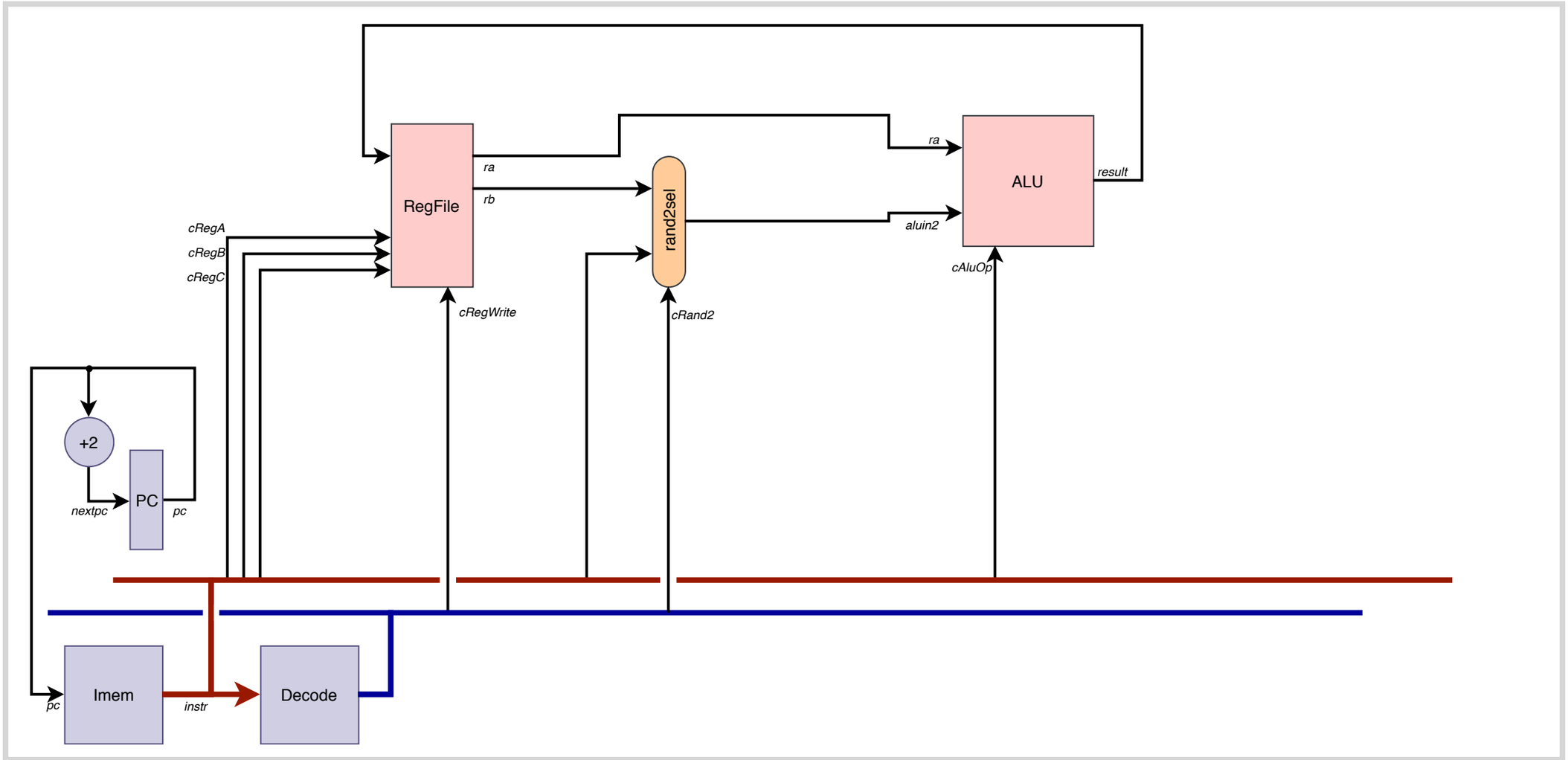
movs ⟨Rw⟩,#⟨imm8⟩

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	1	0	0	Rw			imm8							

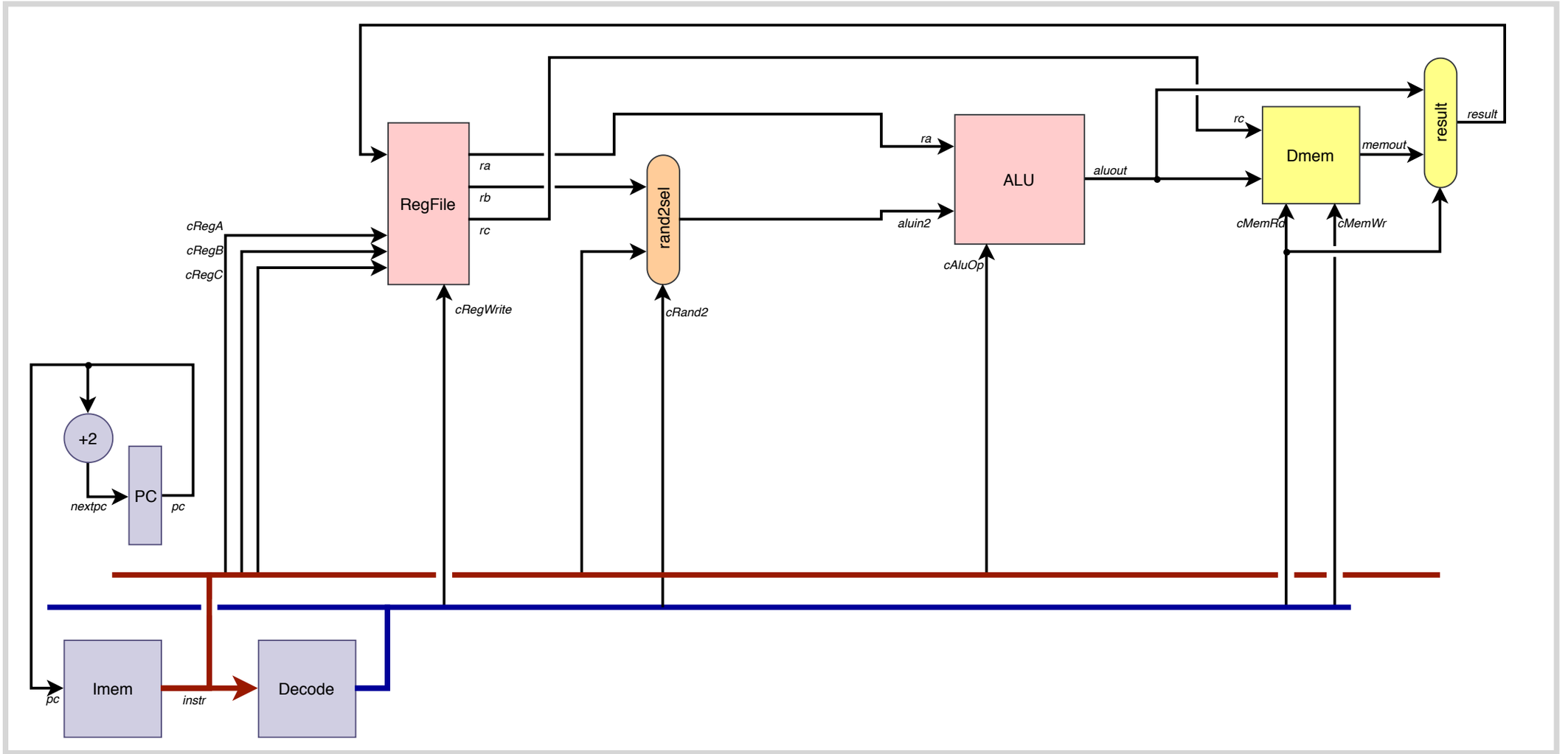
Control signals

Instruction	cRand2	cAluOp	cRegWrite
`adds r`	RegB	Add	T
`adds i3`	Imm3	Add	T
`adds i8`	Imm8	Add	T
`subs r`	RegB	Sub	T
`subs i3`	Imm3	Sub	T
`subs i8`	Imm8	Sub	T
`ands r`	RegB	And	T
`movs r`	RegB	Mov	T
`movs i8`	Imm8	Mov	T

Stage 3: Immediate operands



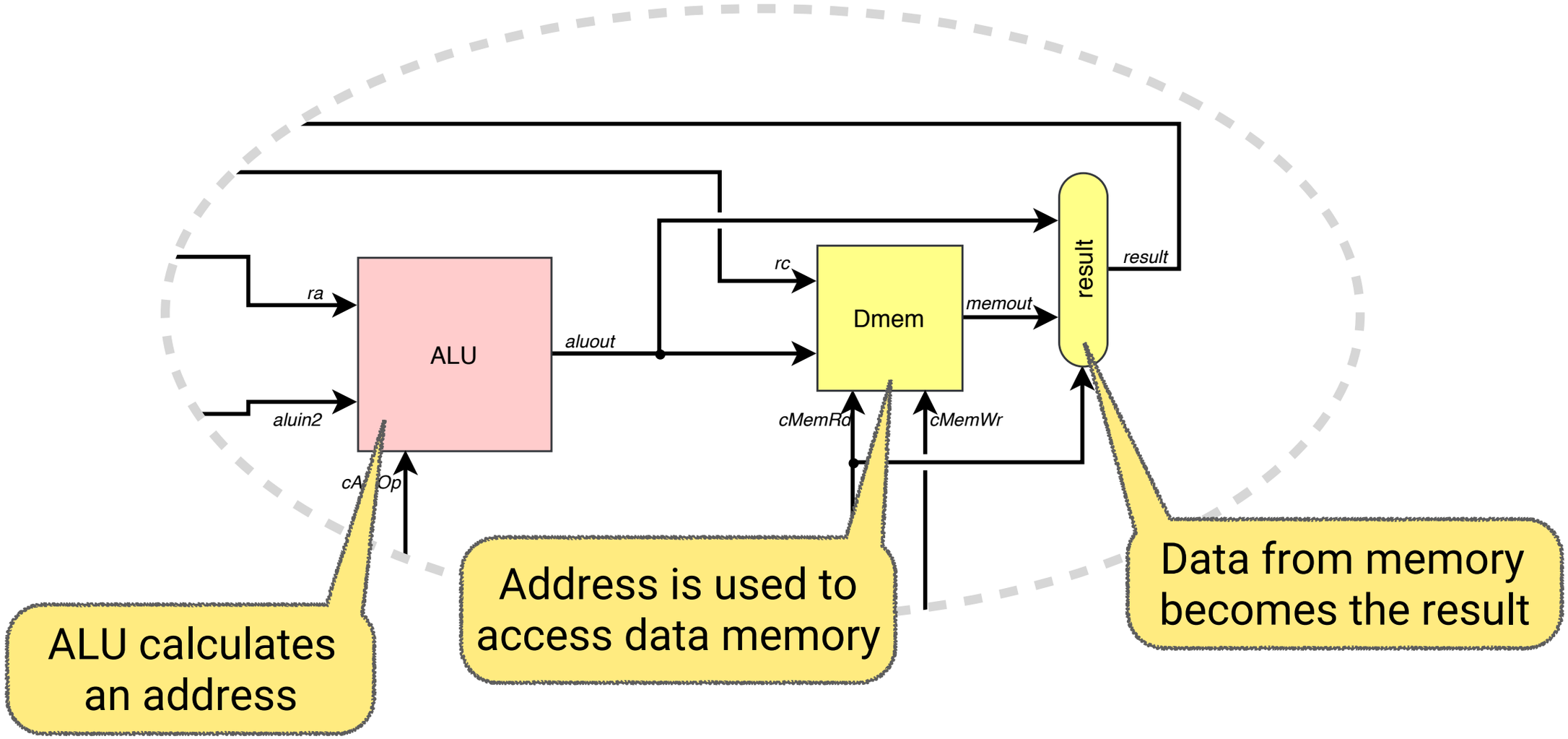
Stage 4: Data memory



ldr ⟨Rx⟩,[⟨Ry⟩,⟨Rz⟩]

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	1	1	0	0	Rz			Ry			Rx		

Stage 4: Data memory



ldr ⟨Rx⟩,[⟨Ry⟩,⟨Rz⟩]

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	1	1	0	0	Rz			Ry			Rx		

Load and store instructions

ldr ⟨Rx⟩,[⟨Ry⟩,⟨Rz⟩]

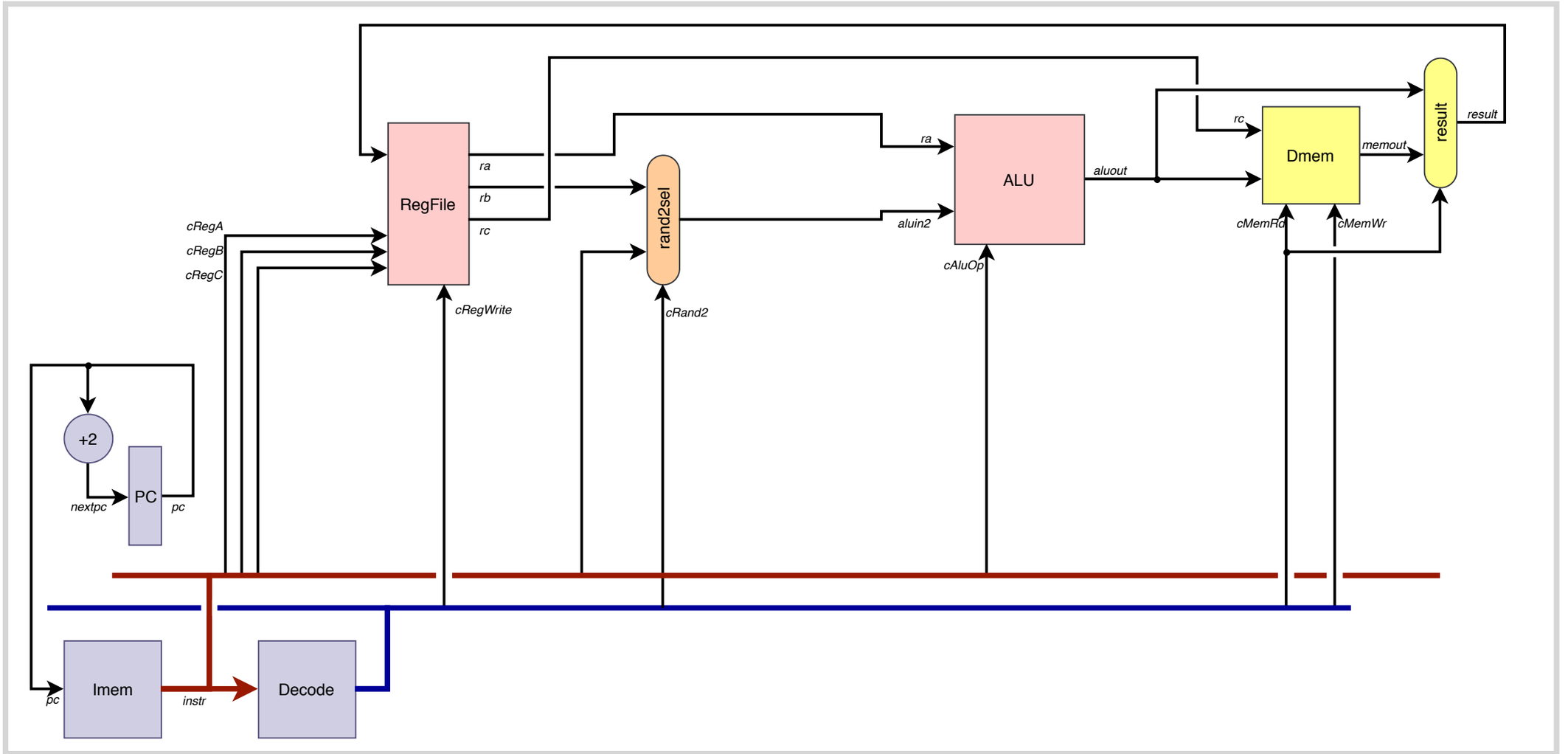
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	1	1	0	0	Rz			Ry			Rx		

str ⟨Rx⟩,[⟨Ry⟩,⟨Rz⟩]

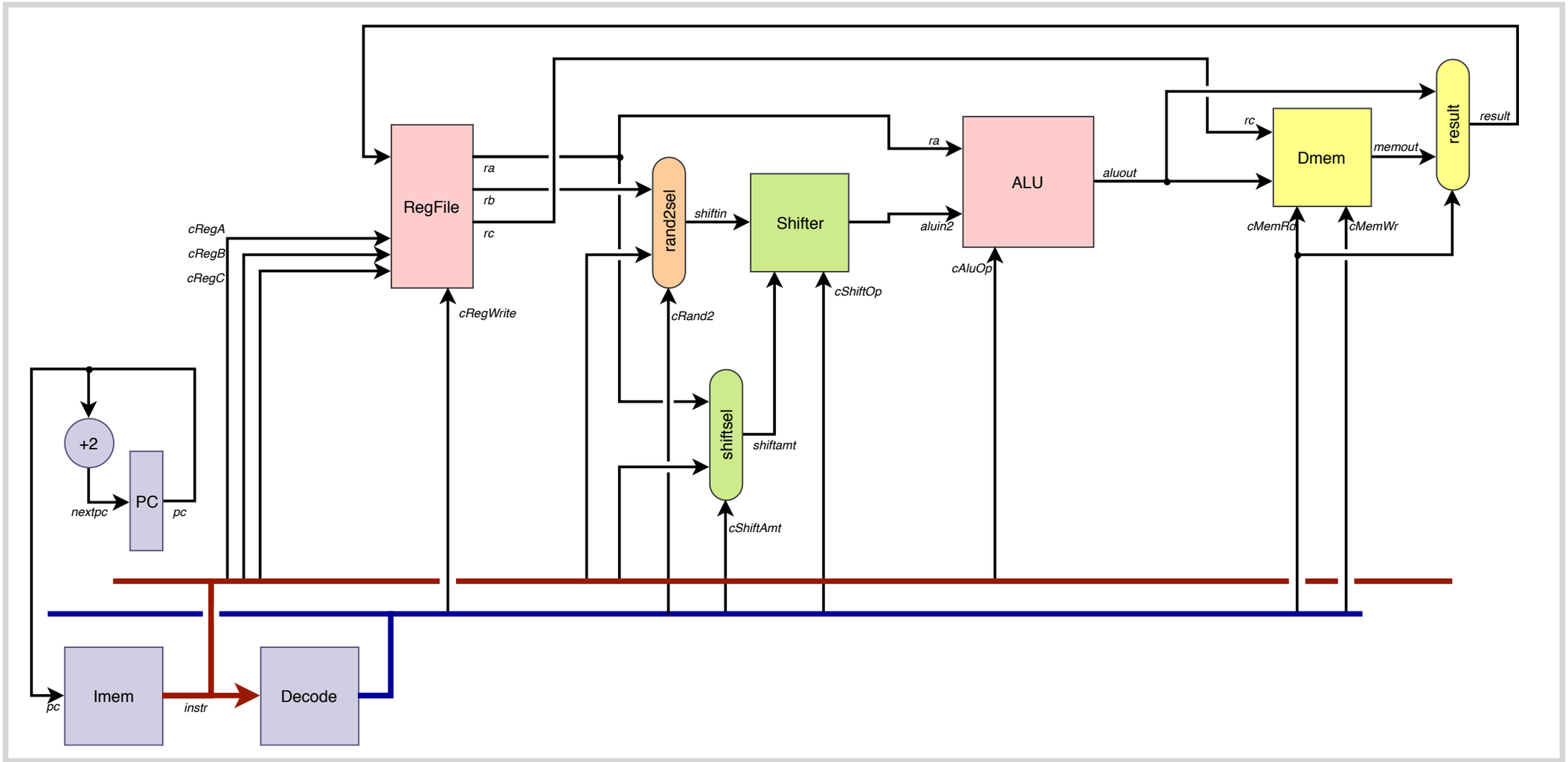
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	1	0	0	0	Rz			Ry			Rx		

Instruction	cRand2	cAluOp	cMemRd	cMemWr	cRegWrite
`adds r`	RegB	Add	F	F	T
`movs i8`	Imm8	Mov	F	F	T
`ldr r`	RegB	Add	T	F	T
`str r`	RegB	Add	F	T	F

Stage 4: Data memory



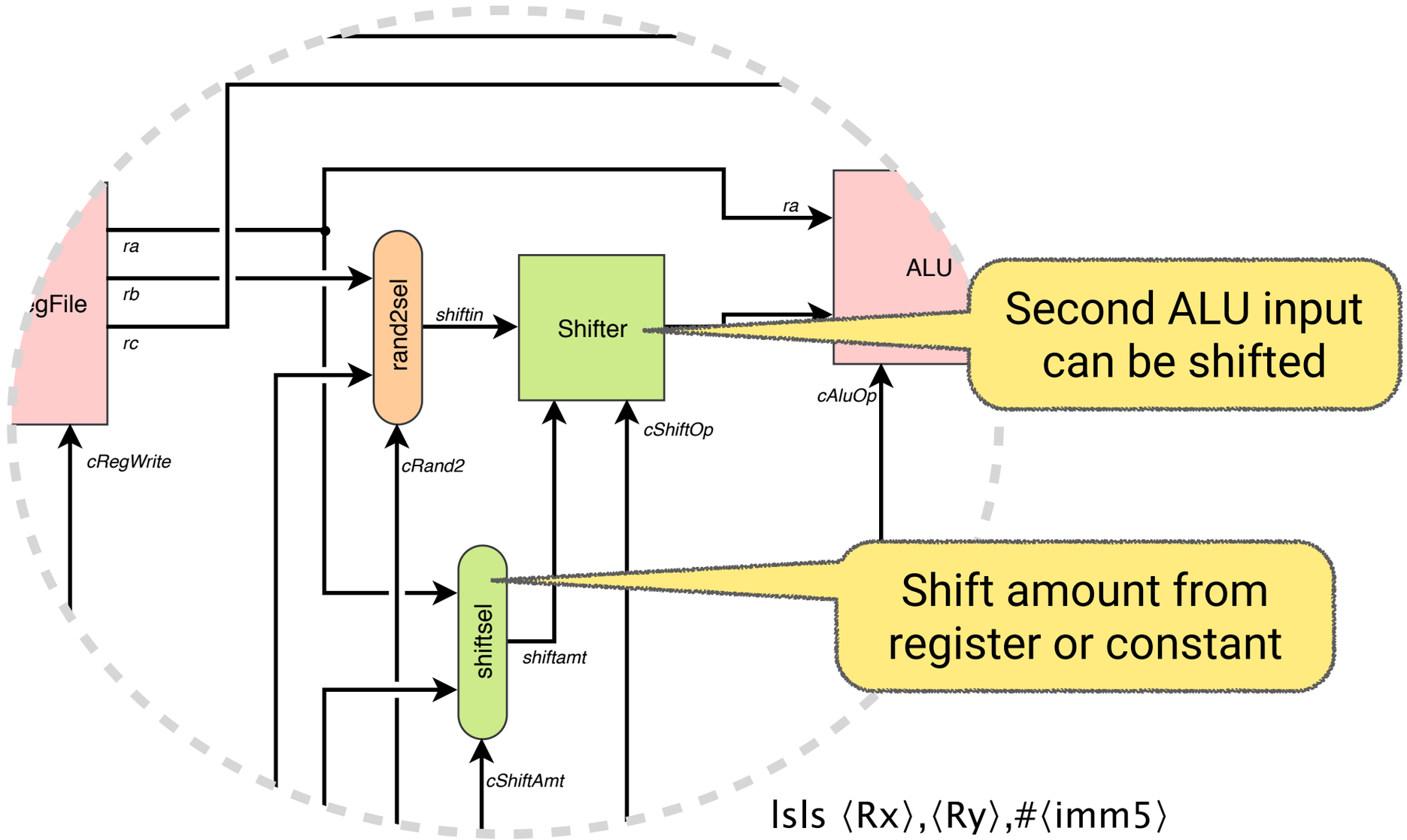
Stage 5: Barrel shifter



lsls ⟨Rx⟩,⟨Ry⟩,#⟨imm5⟩

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	imm5					Ry			Rx		

Stage 5: Barrel shifter



lsls ⟨Rx⟩,⟨Ry⟩,#⟨imm5⟩

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	imm5					Ry			Rx		

Shifts and immediate load/stores

lsls ⟨Rx⟩,⟨Ry⟩,#⟨imm5⟩

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	imm5					Ry			Rx		

rors ⟨Rx⟩,⟨Ry⟩

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	0	0	0	1	1	1	Ry			Rx		

ldr ⟨Rx⟩,[⟨Ry⟩,#⟨imm5⟩]

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	0	1	imm5					Ry			Rx		

str ⟨Rx⟩,[⟨Ry⟩,#⟨imm5⟩]

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	0	0	imm5					Ry			Rx		

Updated control signals

Instruction	cRand2	cShiftOp	cShiftAmt	cAluOp	cMemRd/Wr	cRegWrite
adds r	RegB	Lsl	Sh0	Add	F/F	T
movs i8	Imm8	Lsl	Sh0	Mov	F/F	T
ldr r	RegB	Lsl	Sh0	Add	T/F	T
str r	RegB	Lsl	Sh0	Add	F/T	F
lsls i5	RegB	Lsl	ShImm	Mov	F/F	T
rors r	RegB	Ror	ShReg	Mov	F/F	T
ldr i5	Imm5	Lsl	Sh2	Add	T/F	T
str i5	Imm5	Lsl	Sh2	Add	F/T	F

The story so far