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.

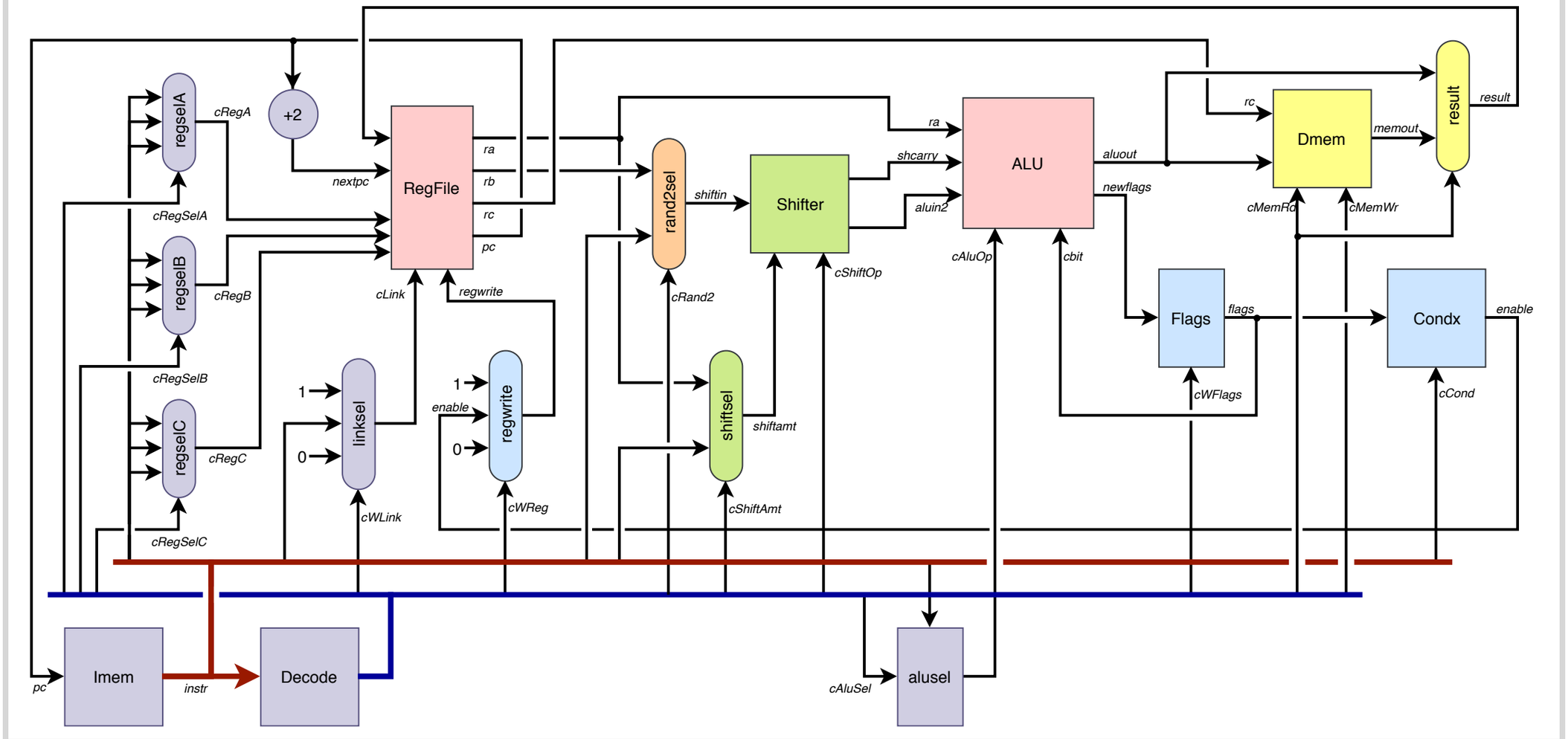
- We won't have pipelined fetch and decode – so the clock speed will be limited.
- We won't implement the control needed for multi-cycle instructions like `push` and `pop`.
- And no interrupts, either.

In this lecture

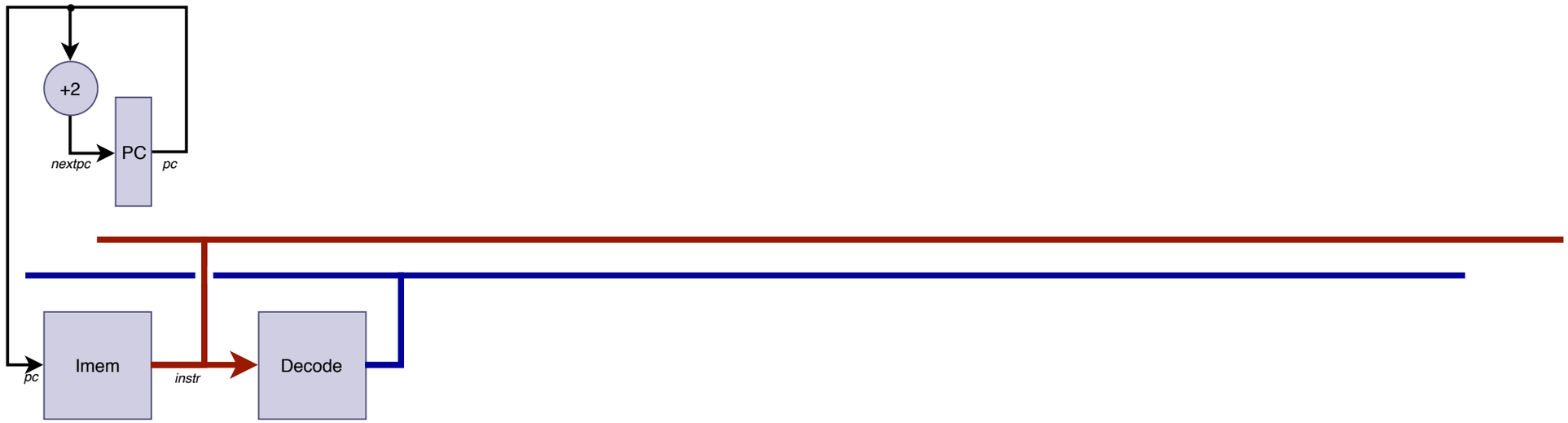
Stage-by-stage development of a datapath design:

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

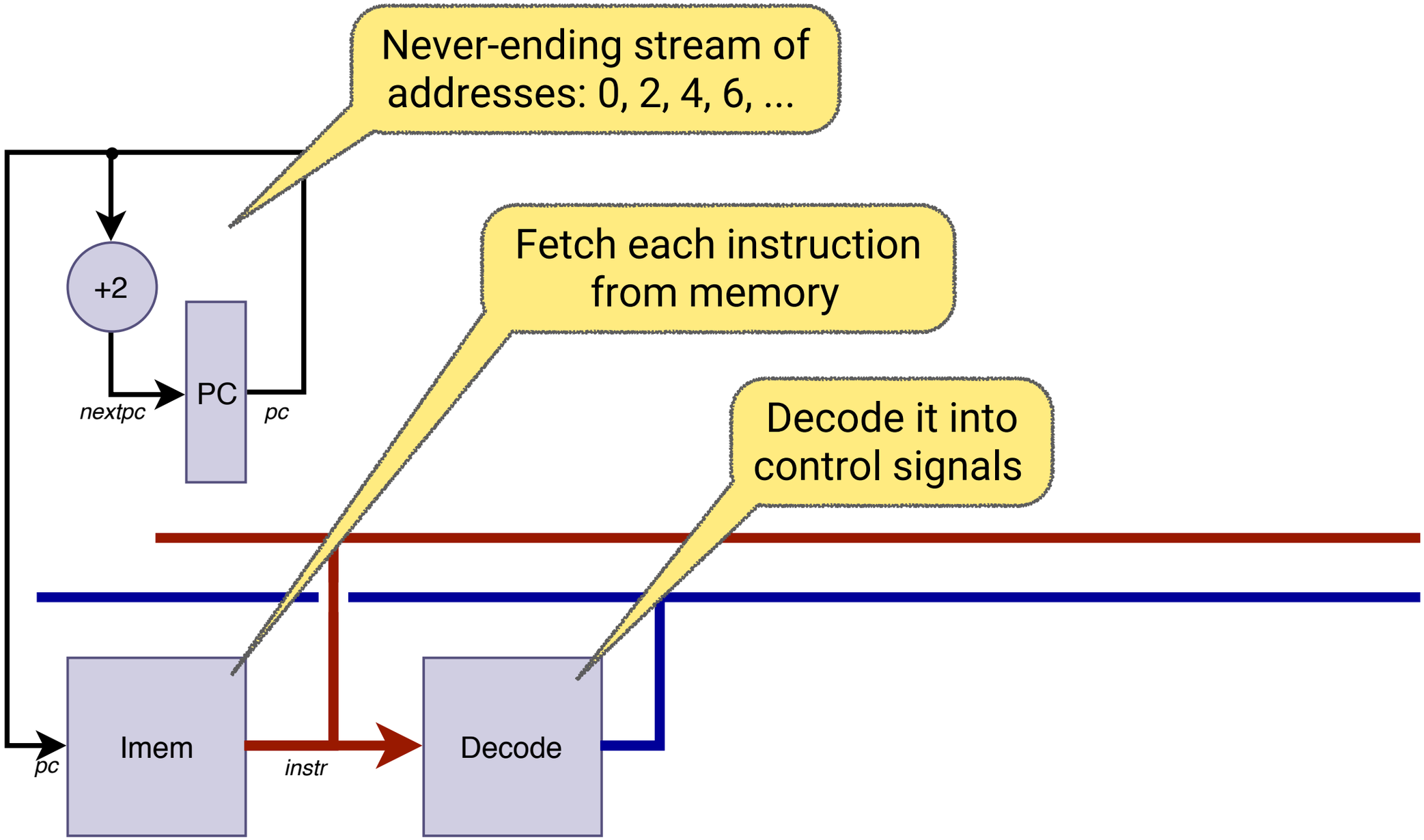
The complete datapath



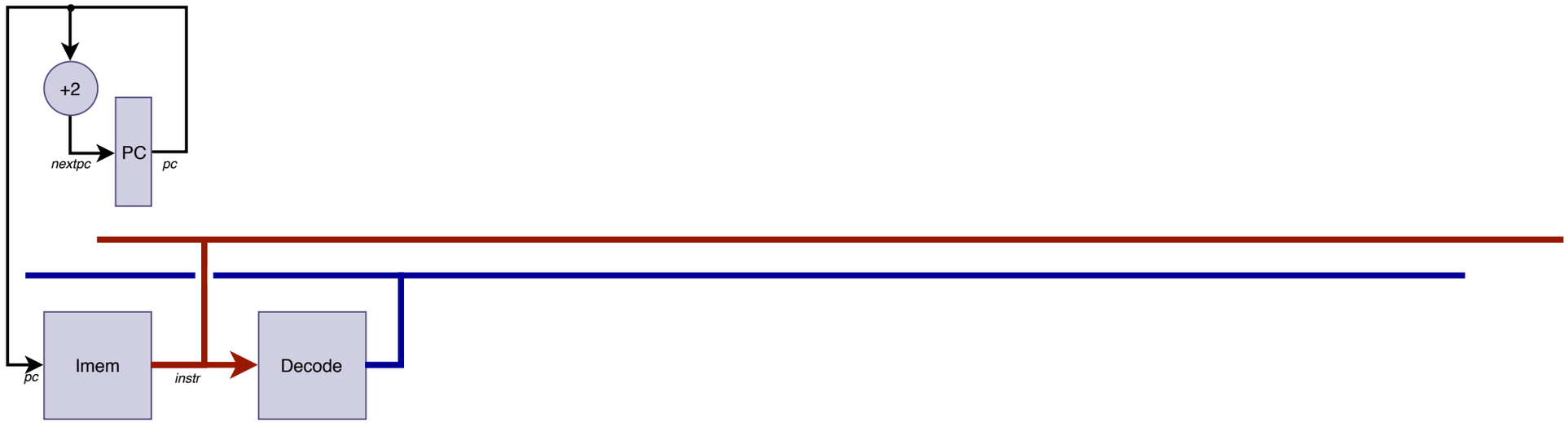
Stage 1: Instruction fetch



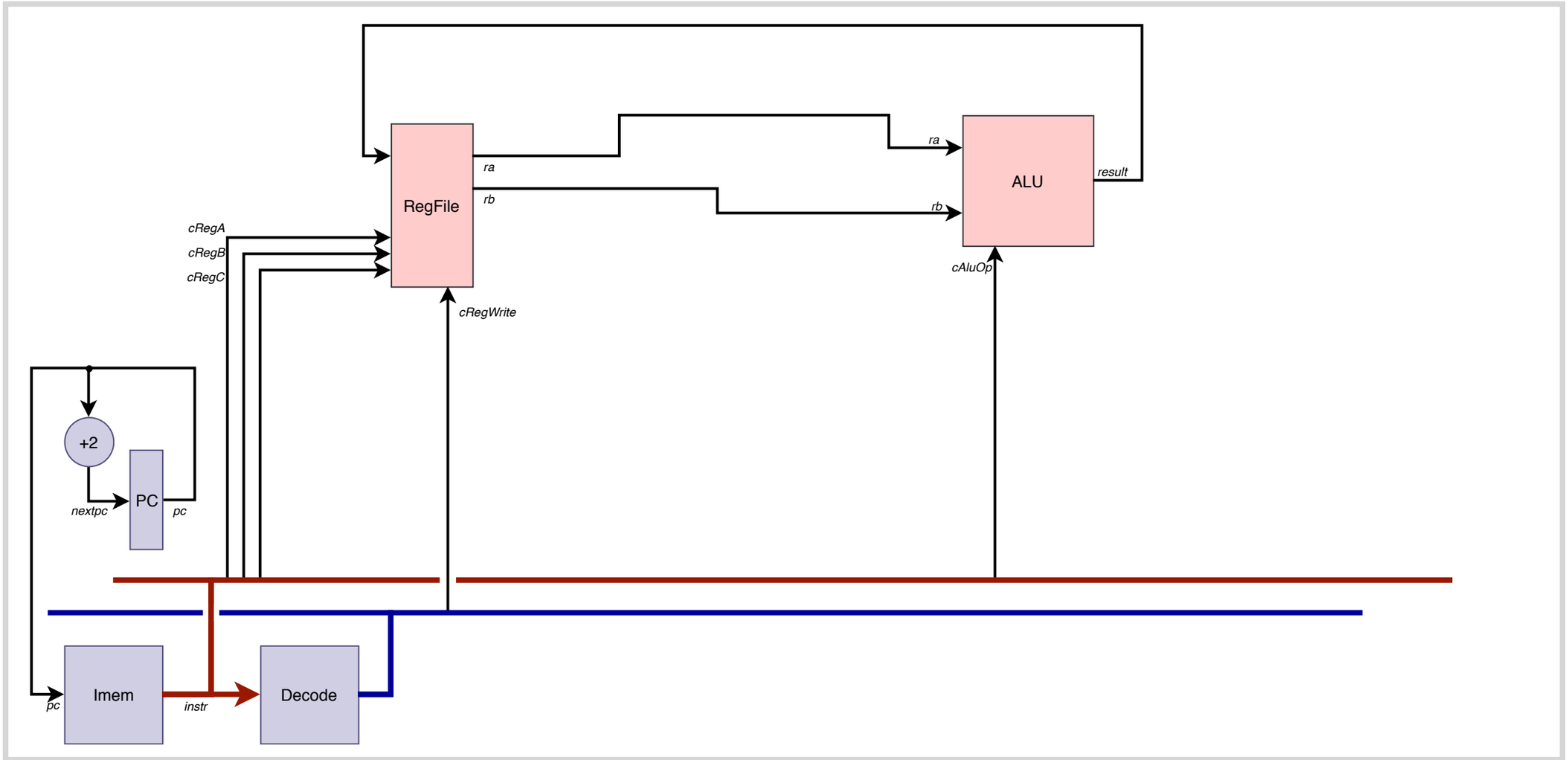
Stage 1: Instruction fetch



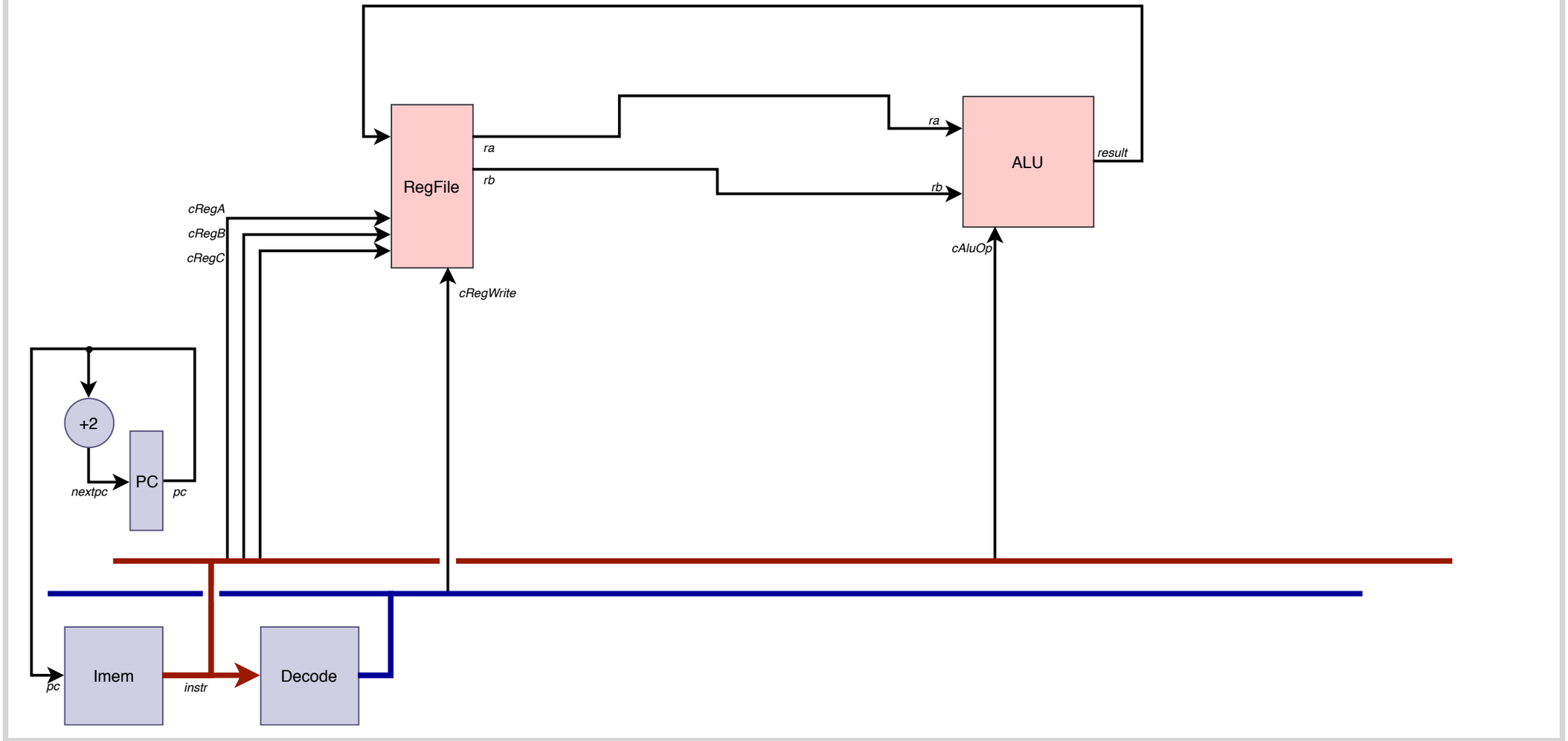
Stage 1: Instruction fetch



Stage 2: ALU operations



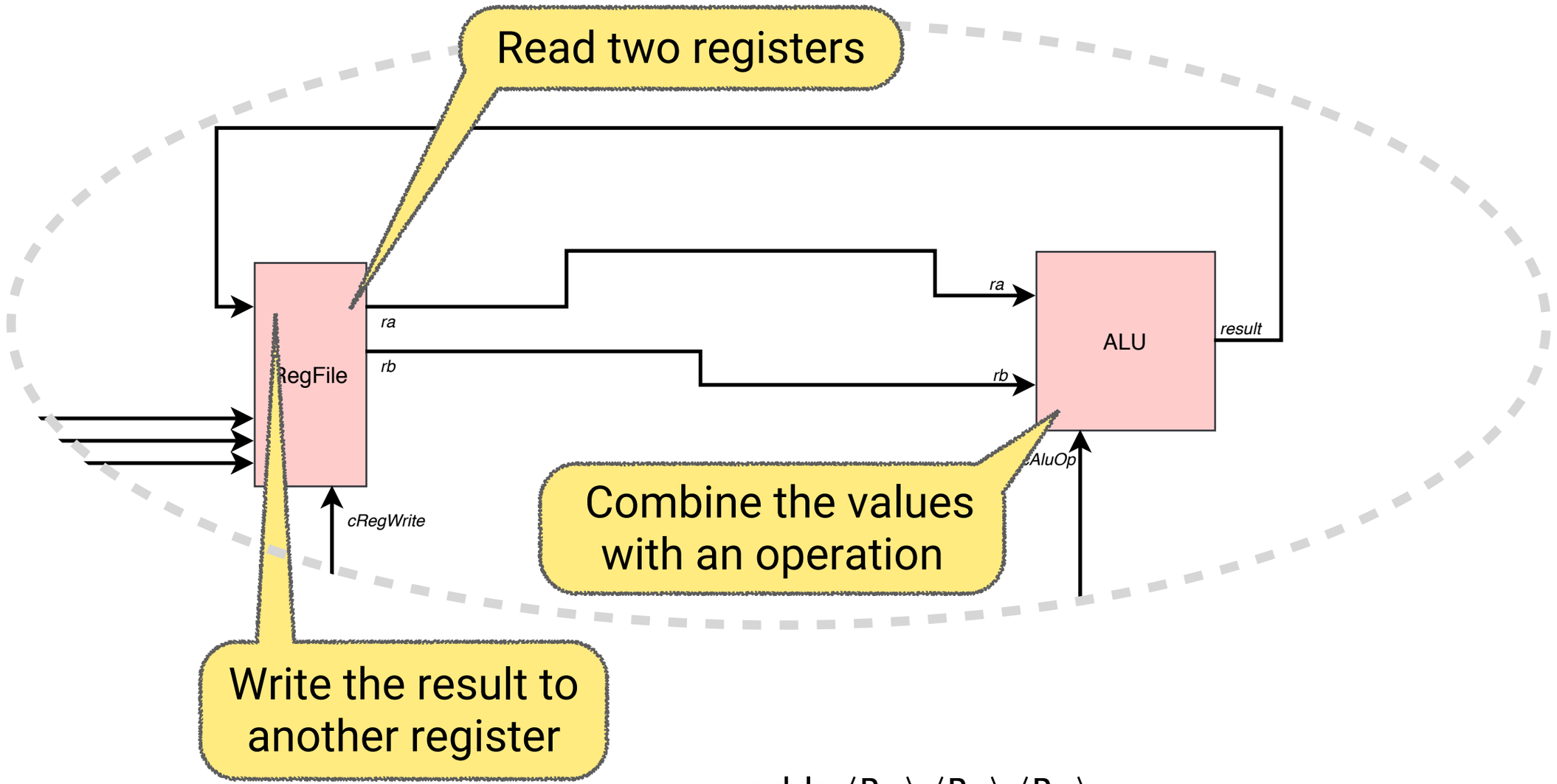
Stage 2: ALU operations



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		

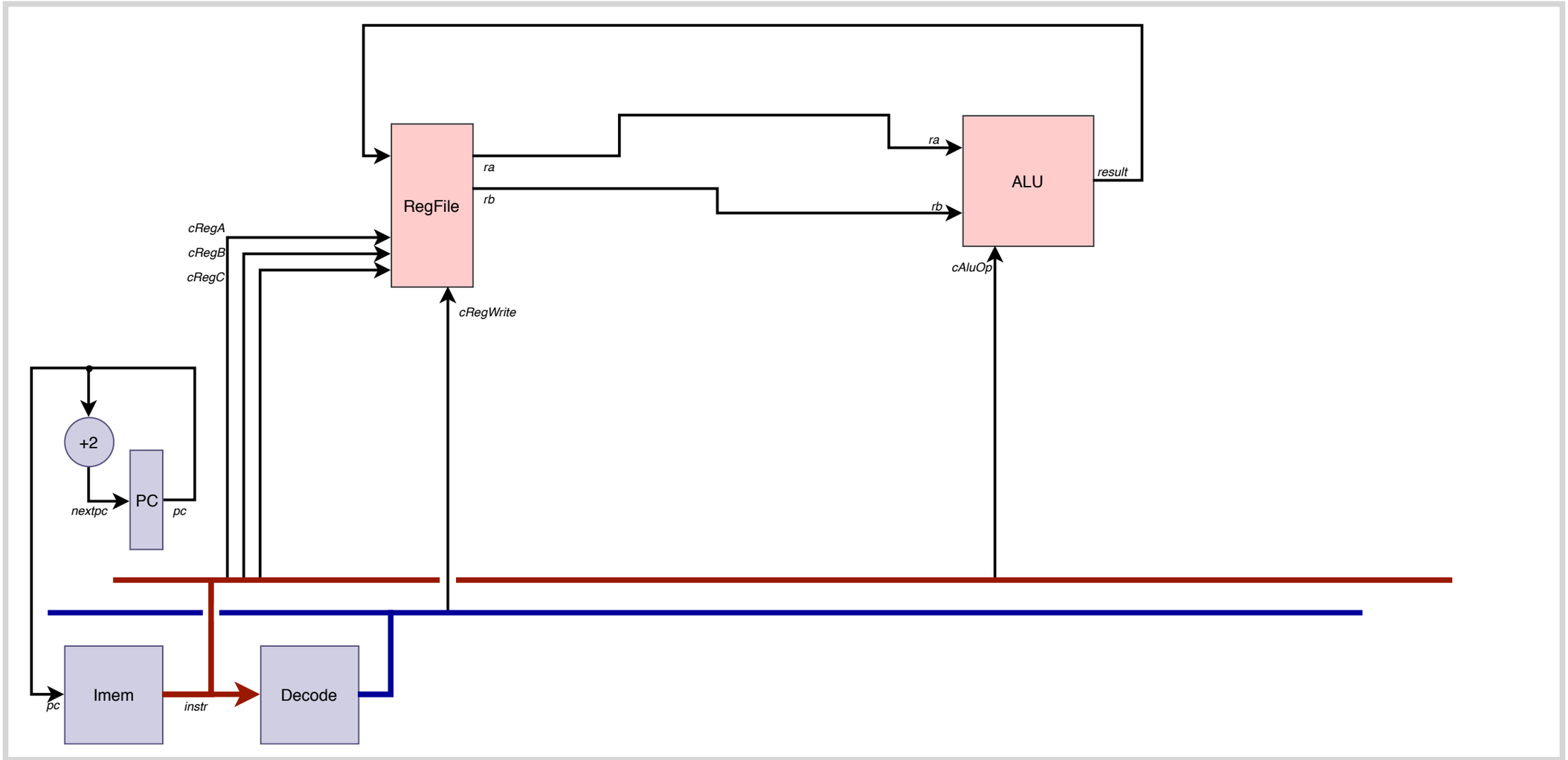
subs ⟨Rx⟩,⟨Ry⟩,⟨Rz⟩

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		

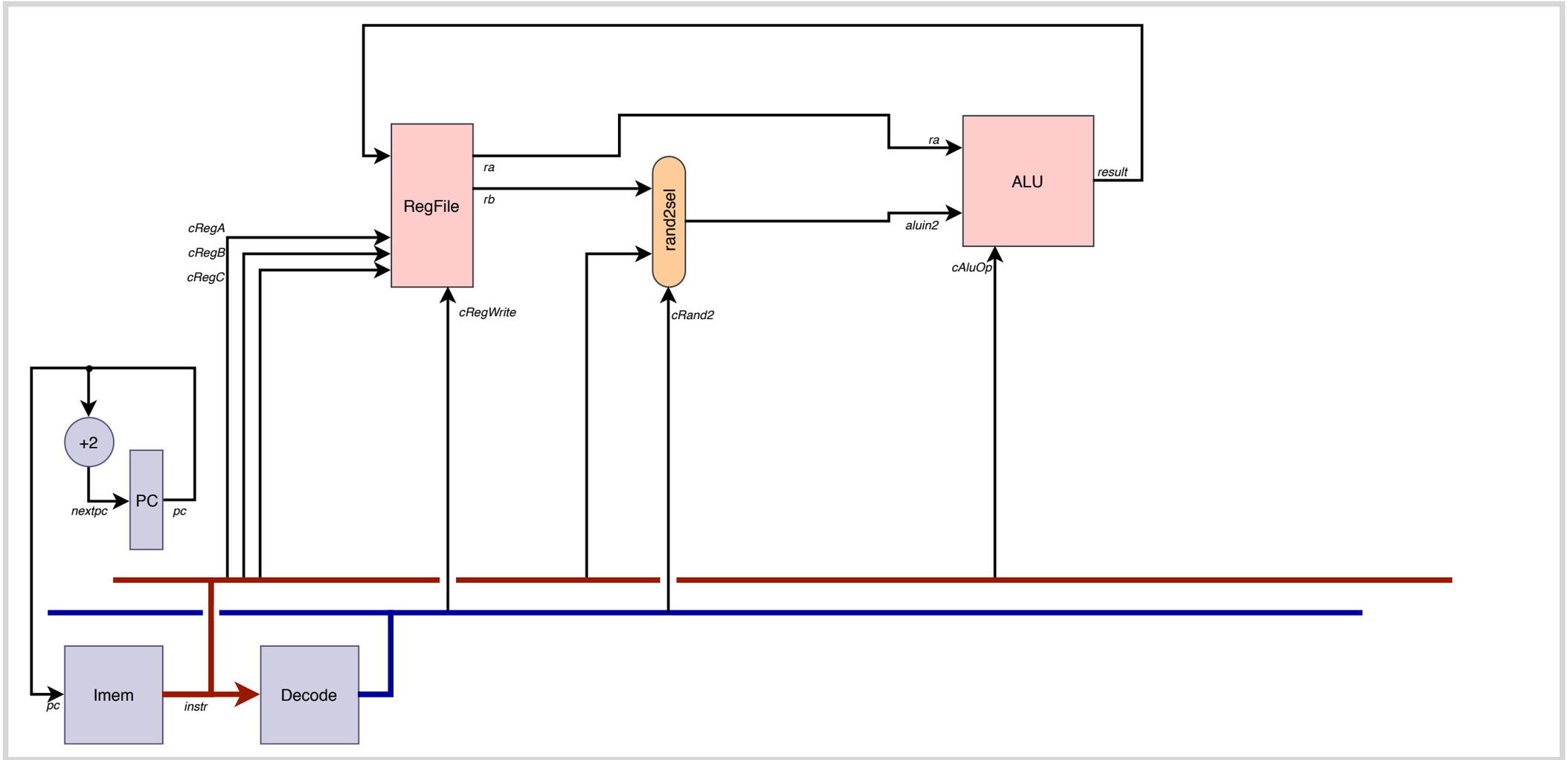
ands ⟨Rx⟩,⟨Ry⟩

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		

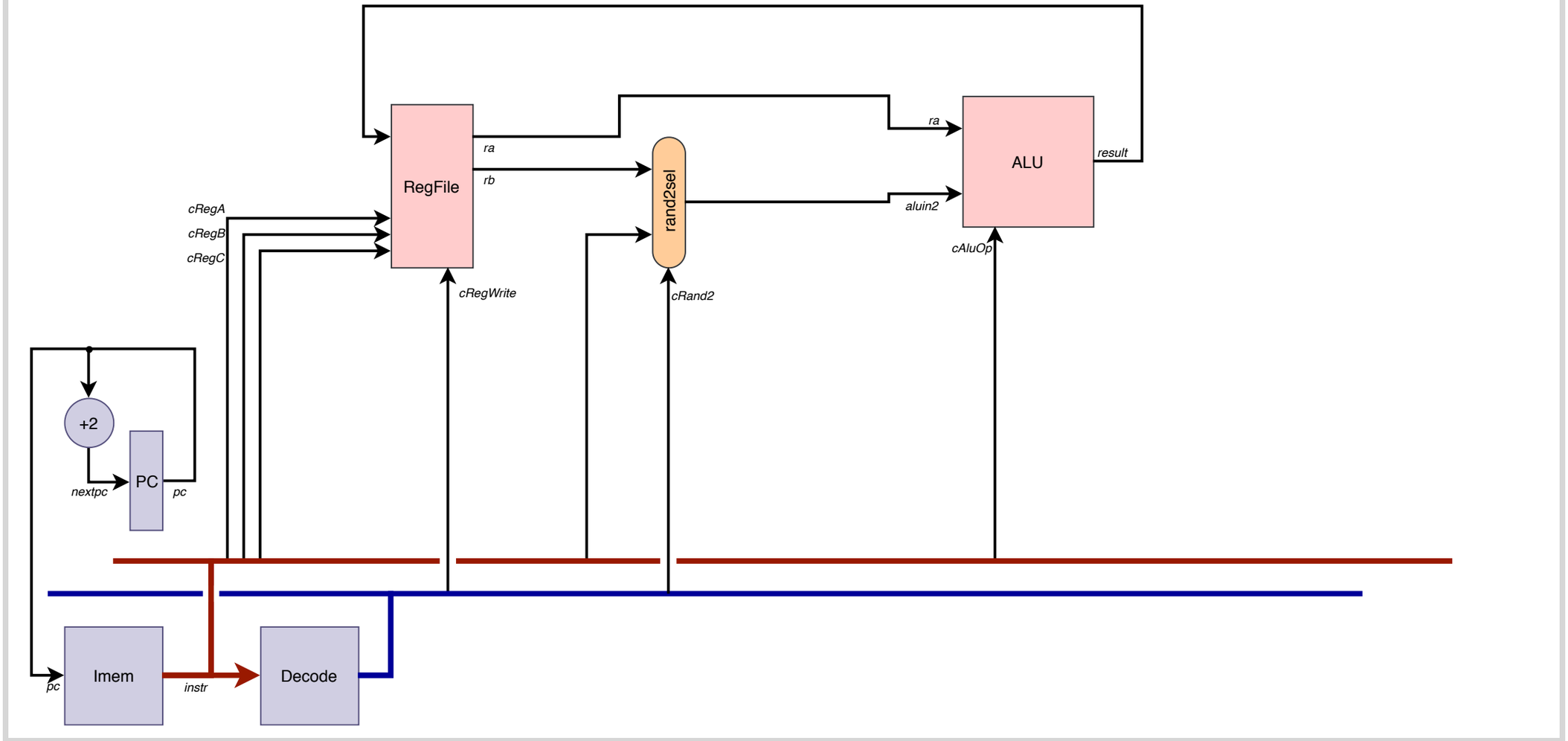
Stage 2: ALU operations



Stage 3: Immediate operands



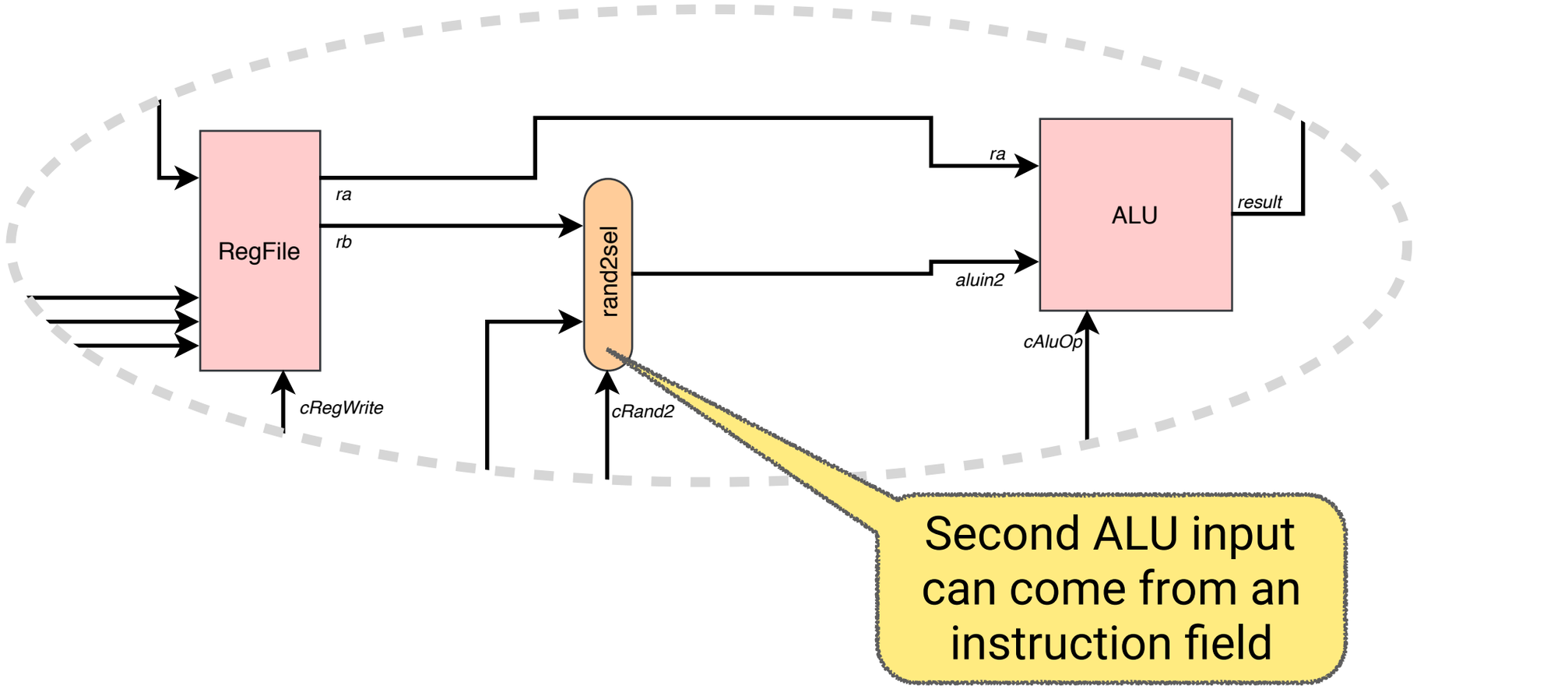
Stage 3: Immediate operands



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							

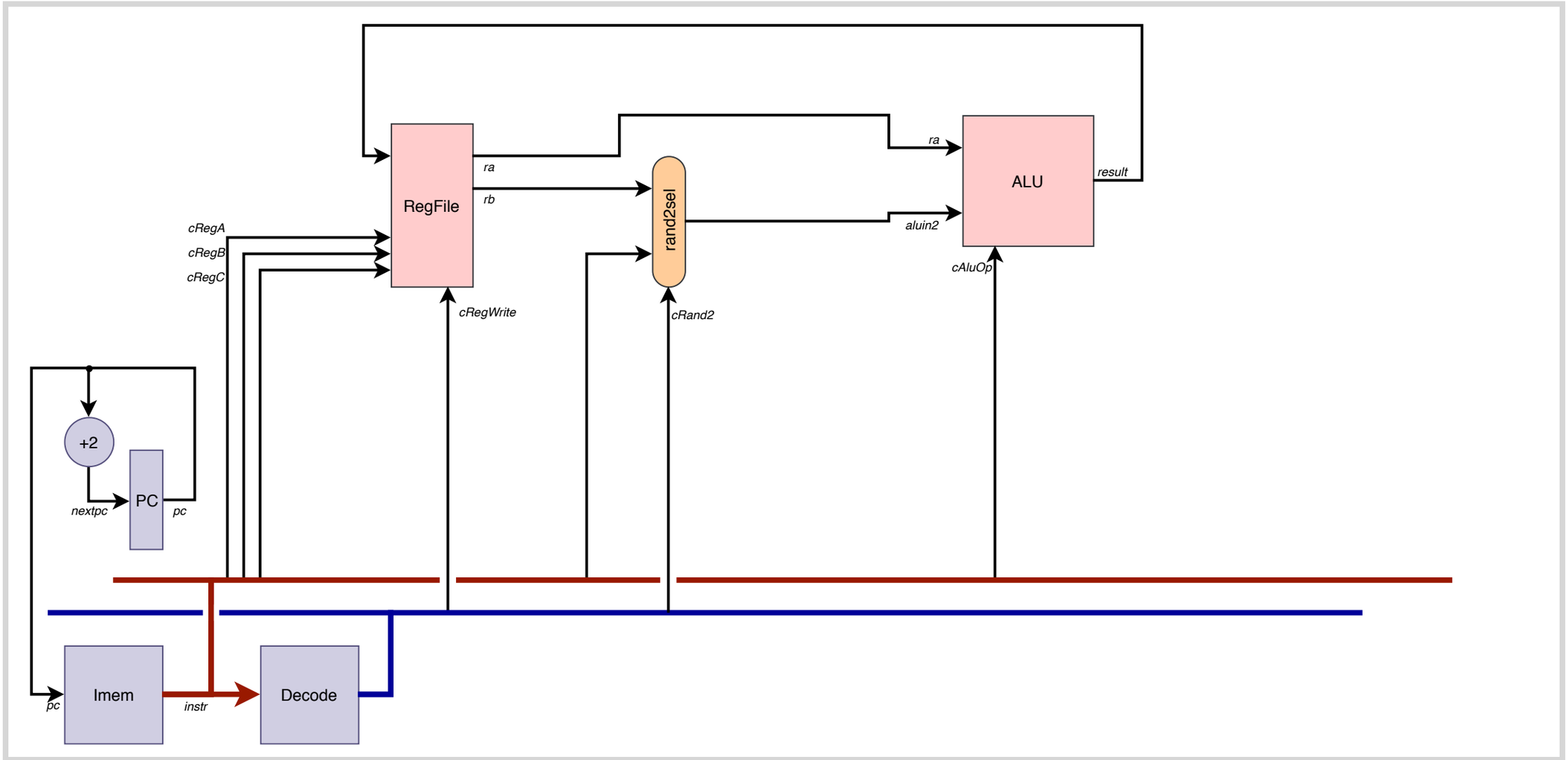
movs ⟨Rw⟩,#⟨imm8⟩

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

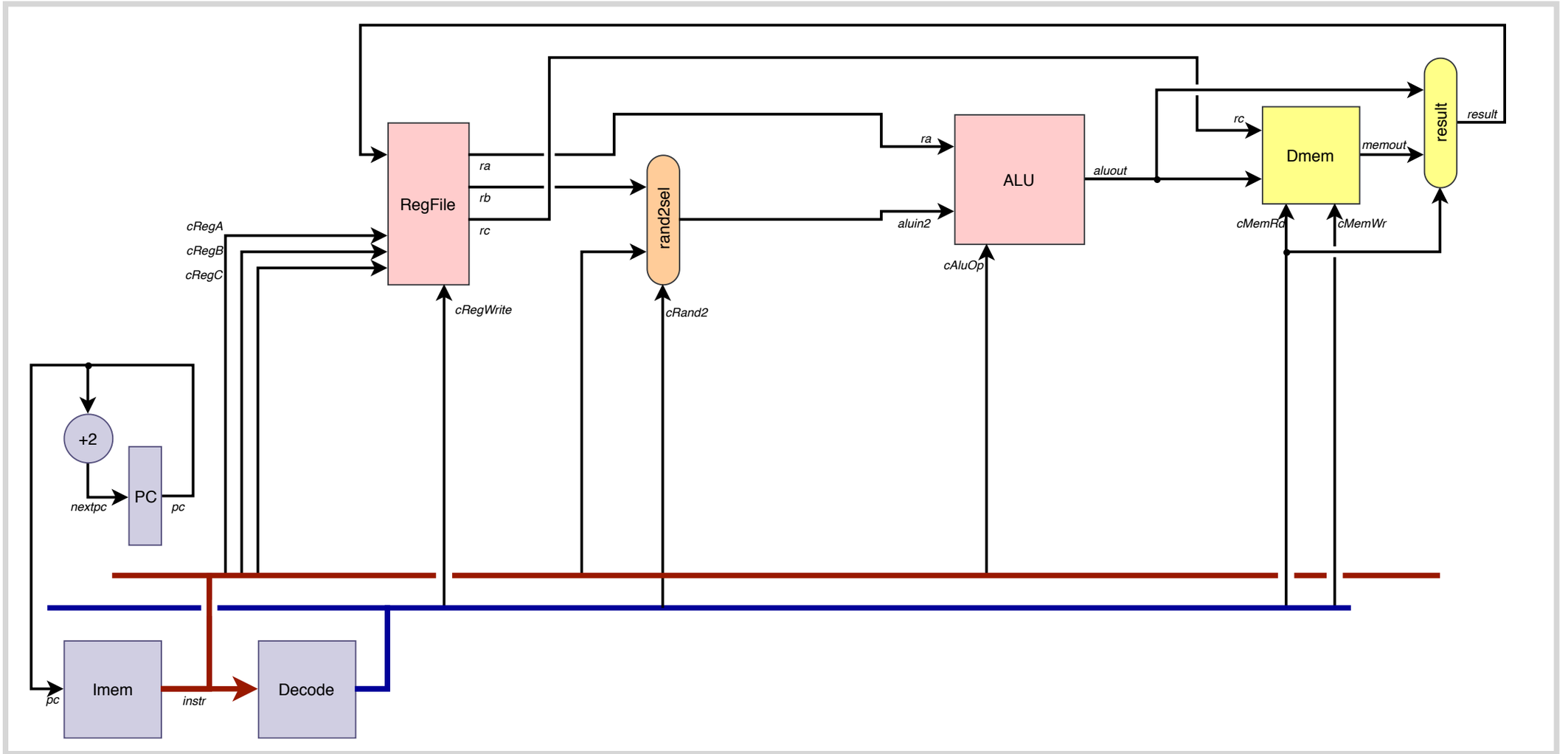
Control signals

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

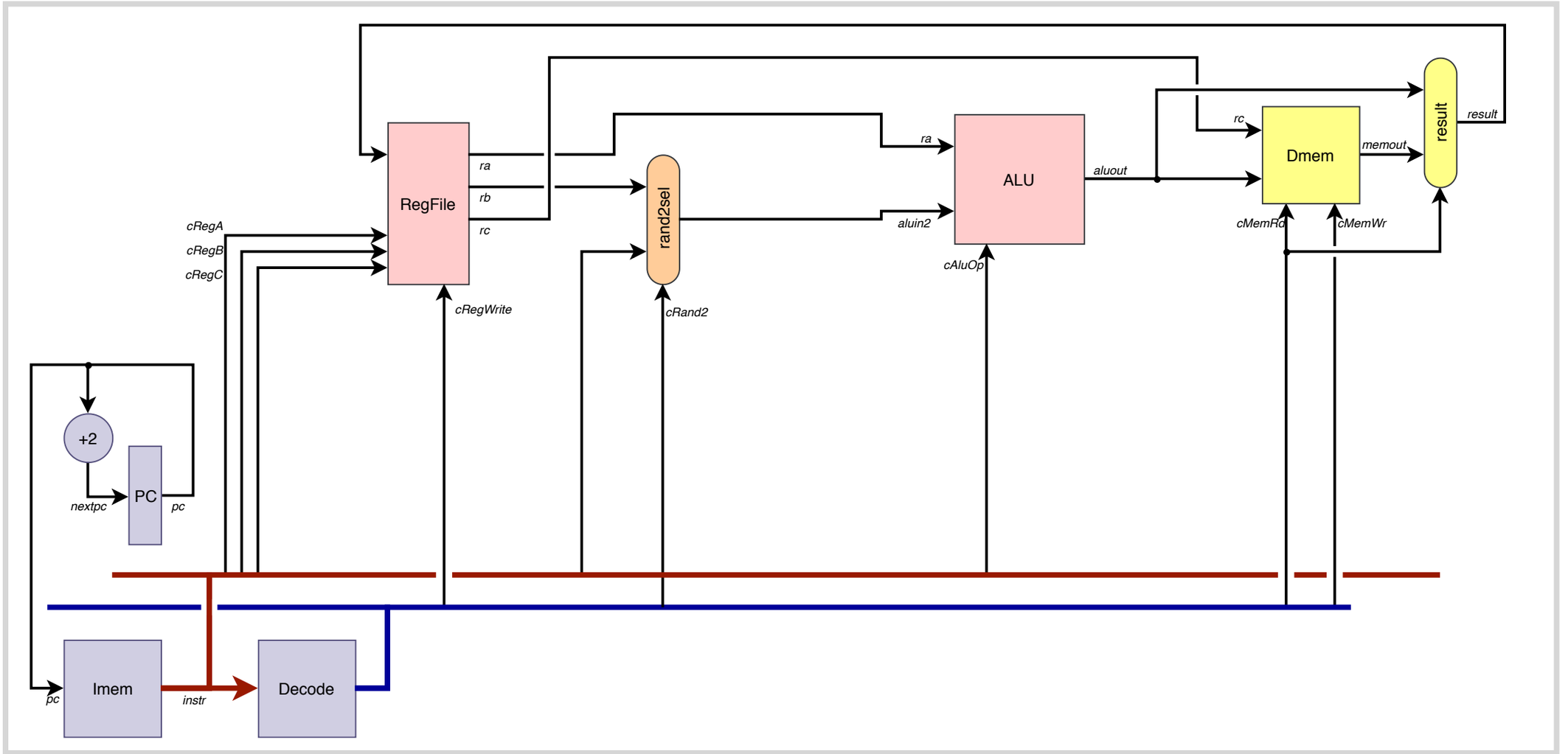
Stage 3: Immediate operands



Stage 4: Data memory



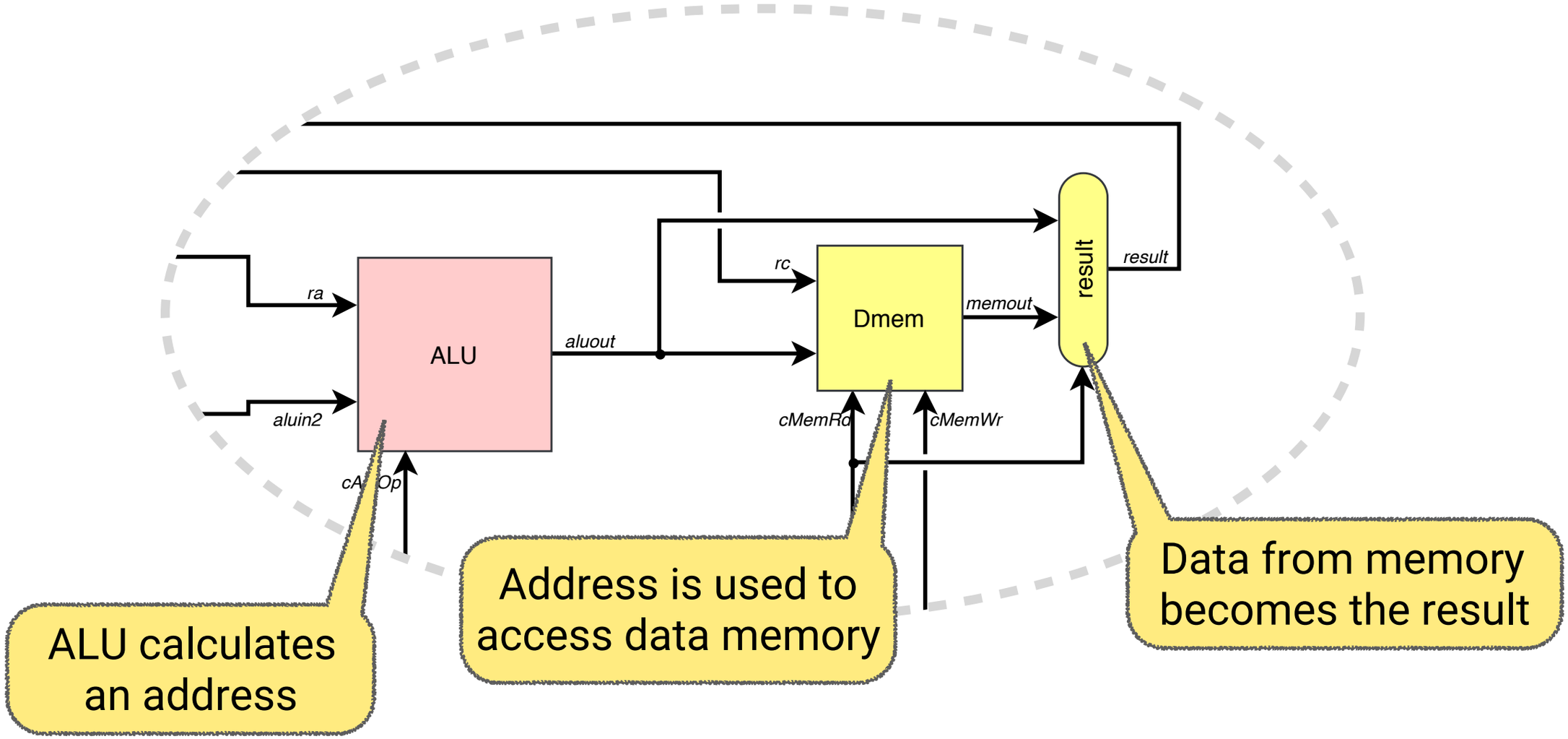
Stage 4: Data memory



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⟩]

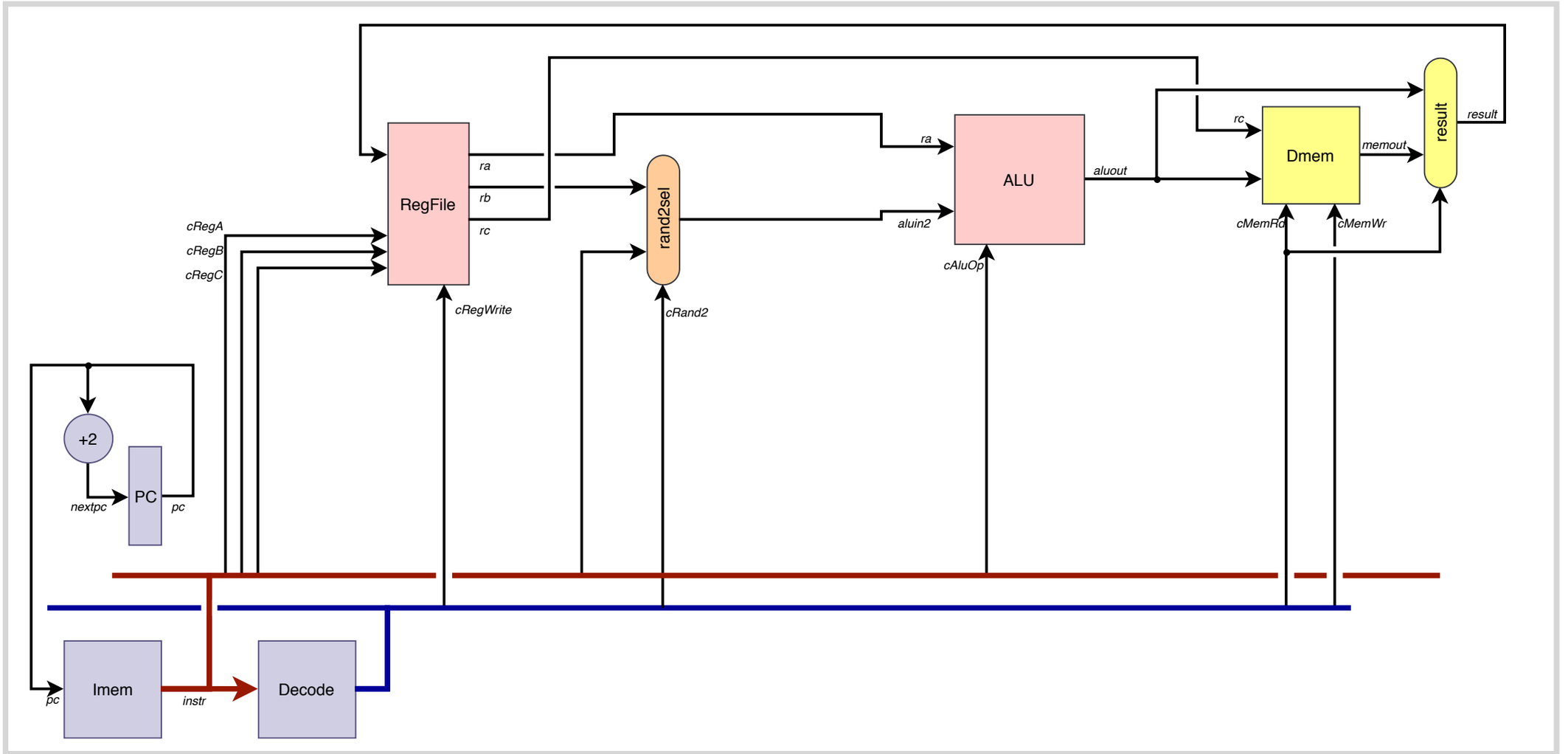
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	1	1	0	0	Rz			Ry			Rx		

str ⟨Rx⟩,[⟨Ry⟩,⟨Rz⟩]

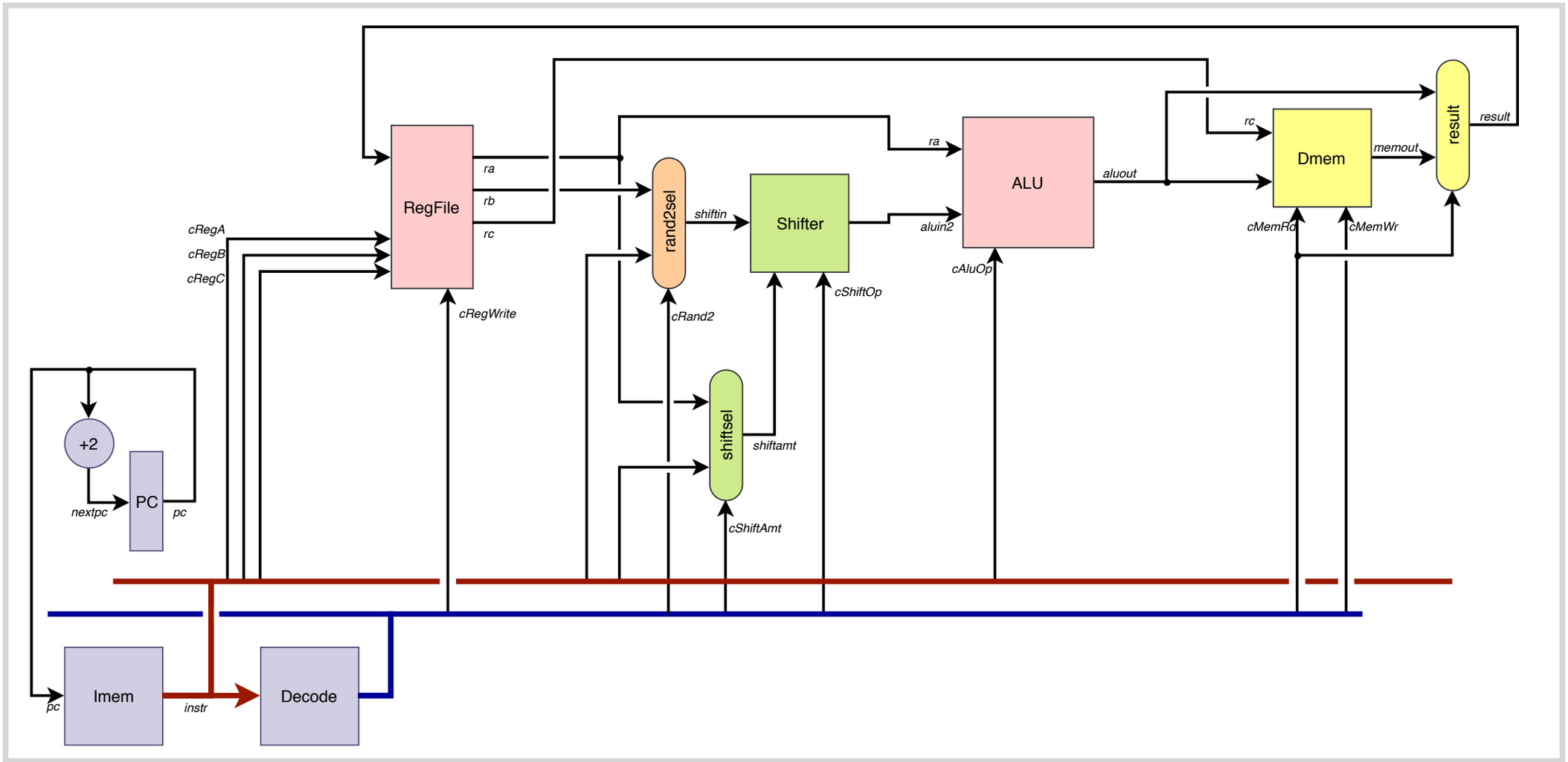
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	1	0	0	0	Rz			Ry			Rx		

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

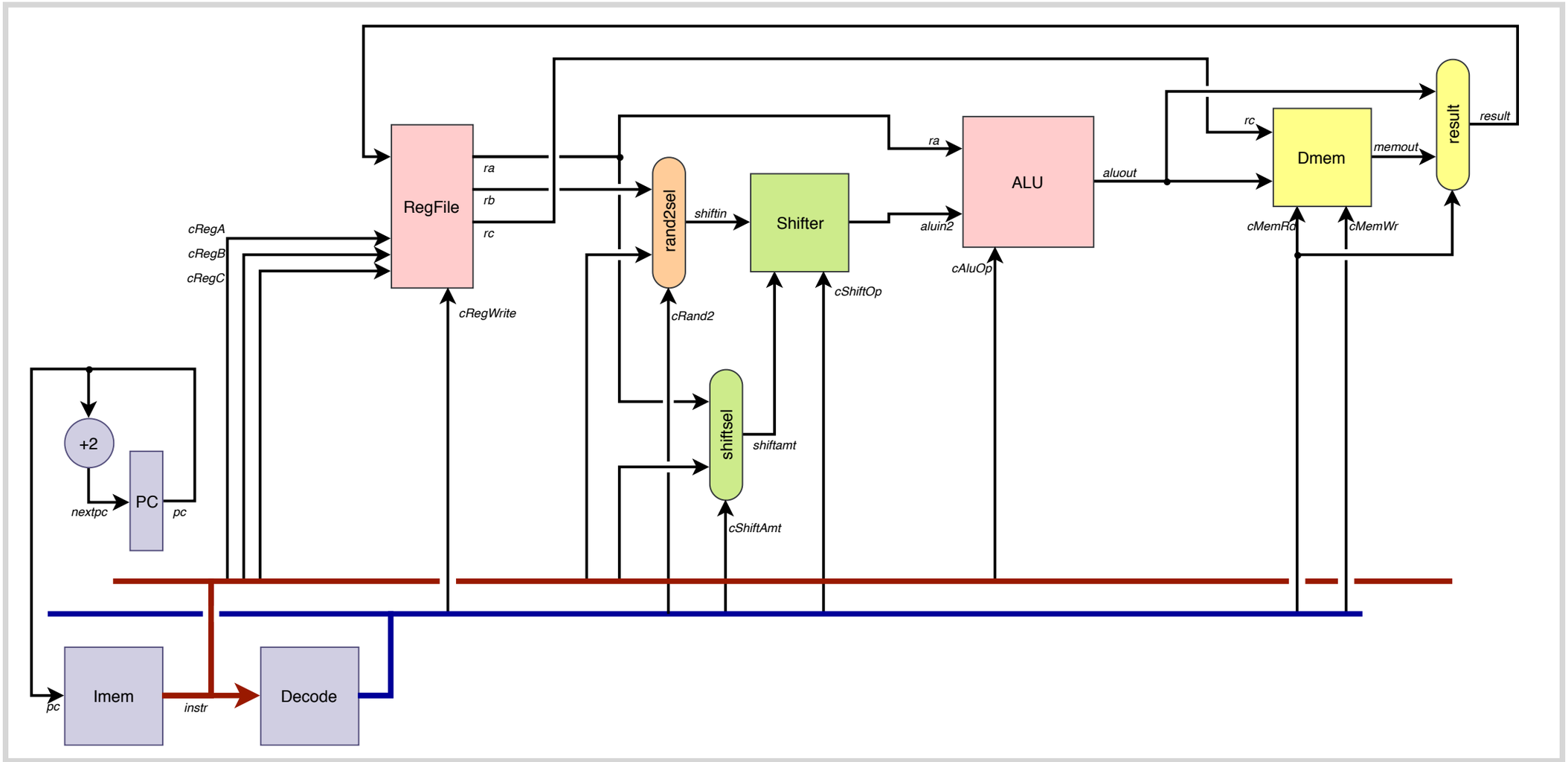
Stage 4: Data memory



Stage 5: Barrel shifter



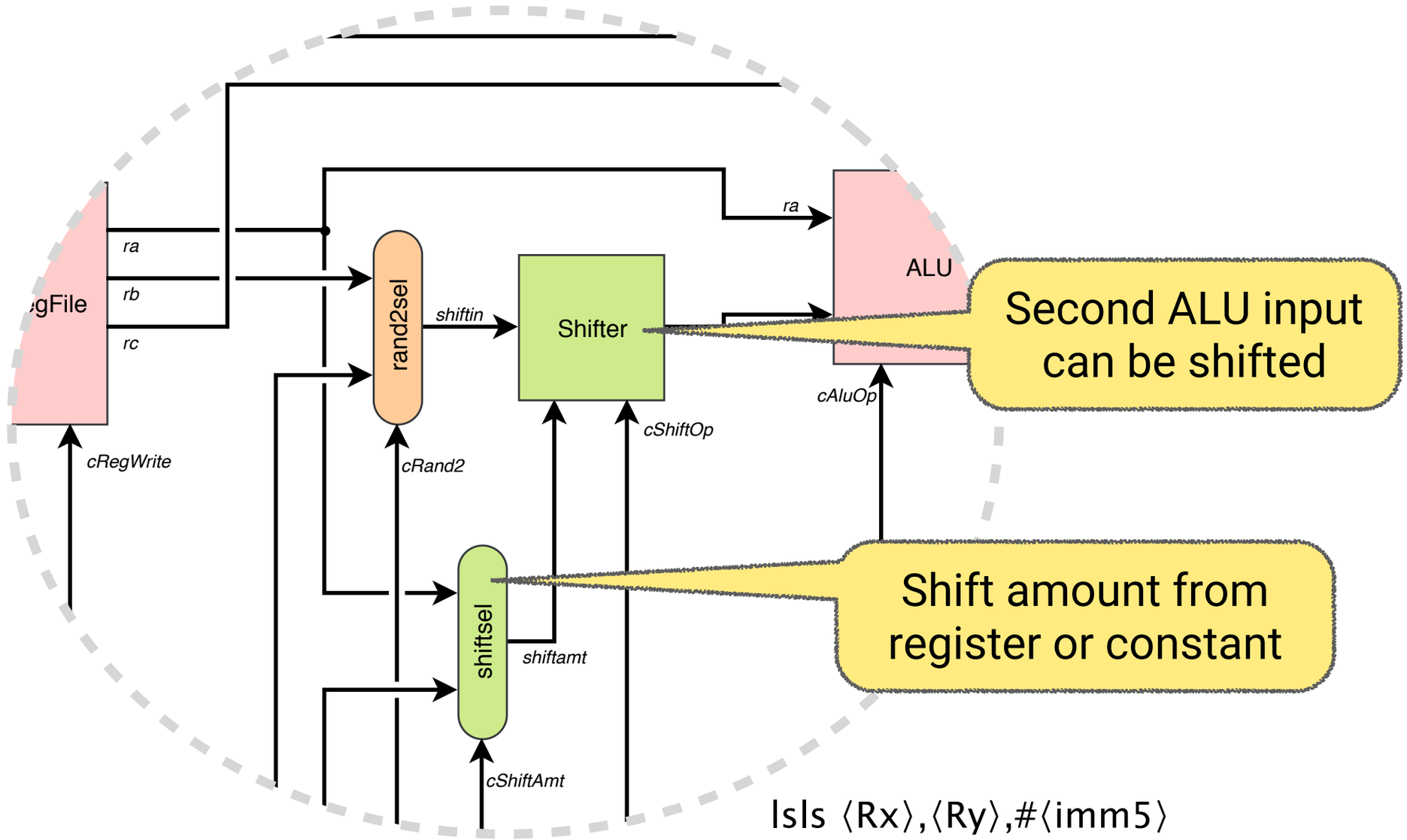
Stage 5: Barrel shifter



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		

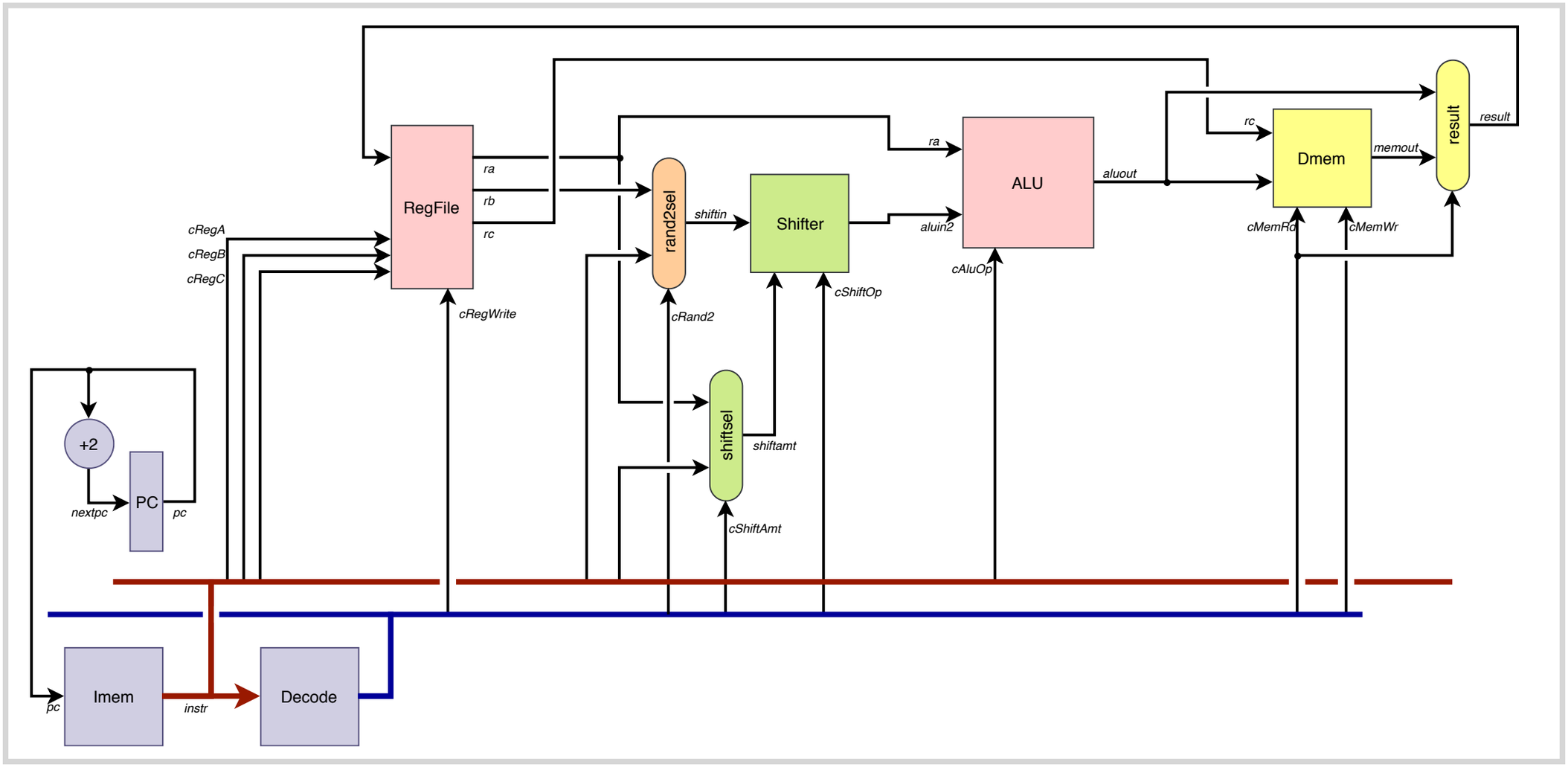
str ⟨Rx⟩,[⟨Ry⟩,#⟨imm5⟩]

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

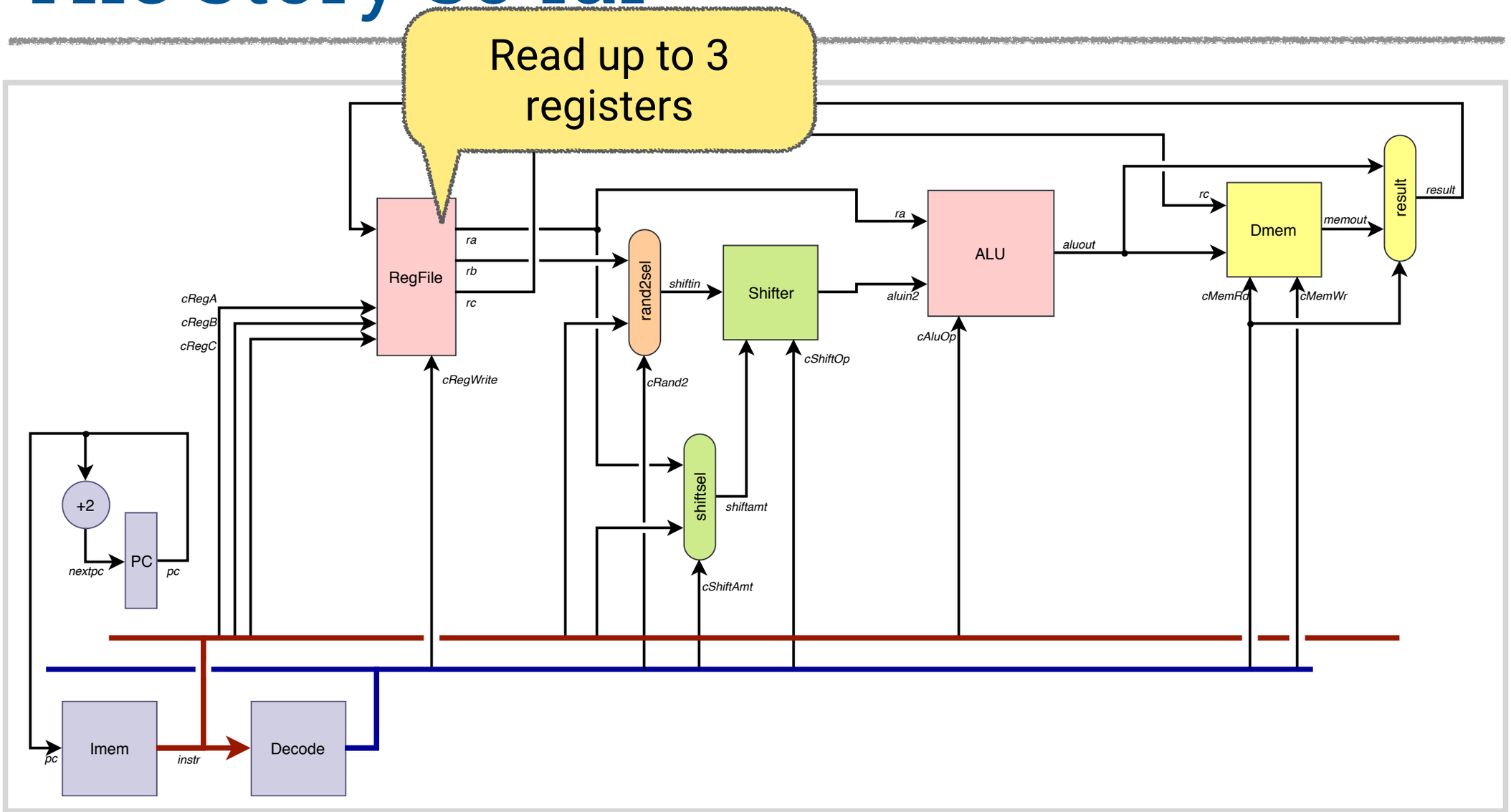
Updated control signals

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

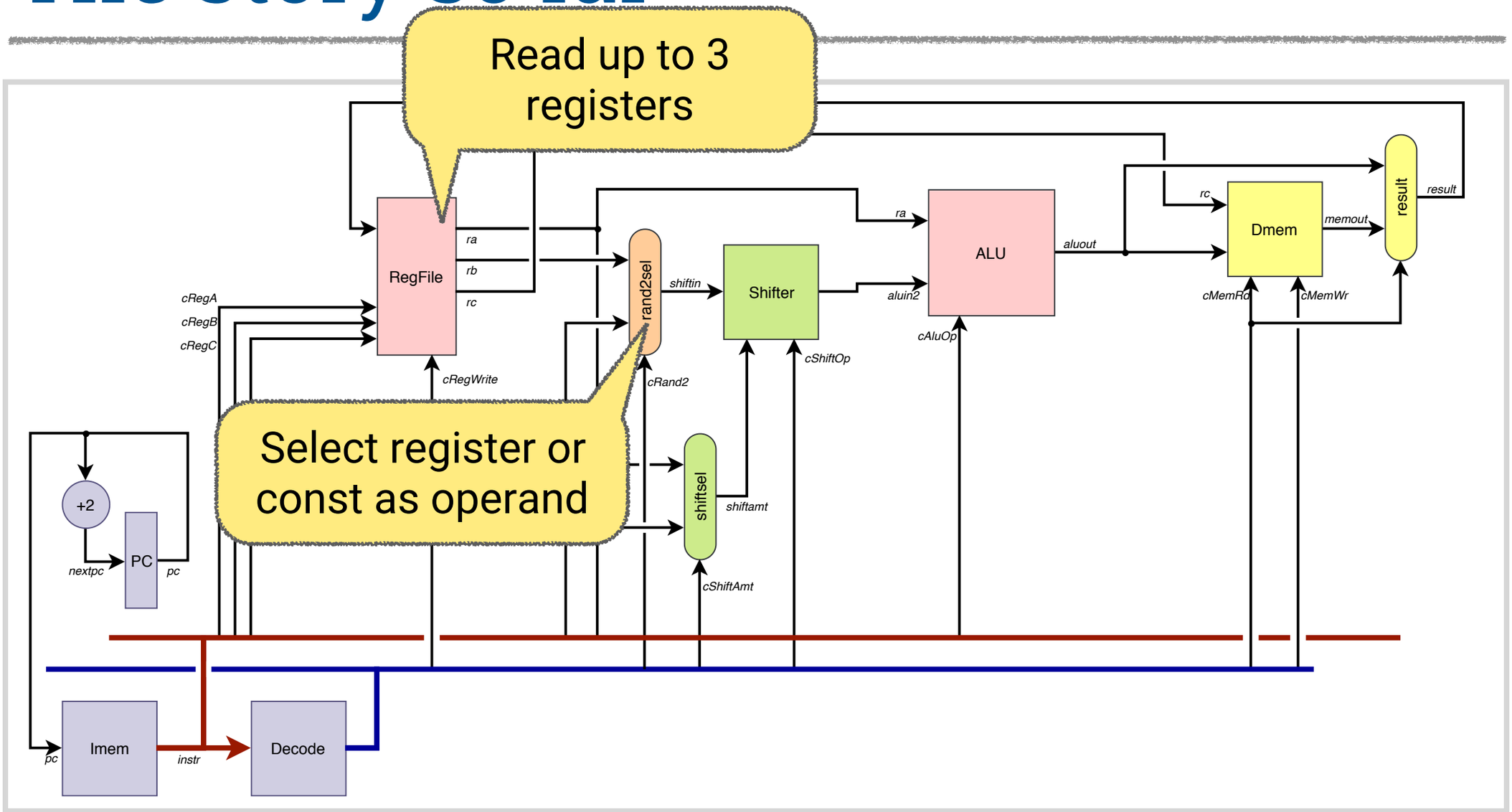
The story so far



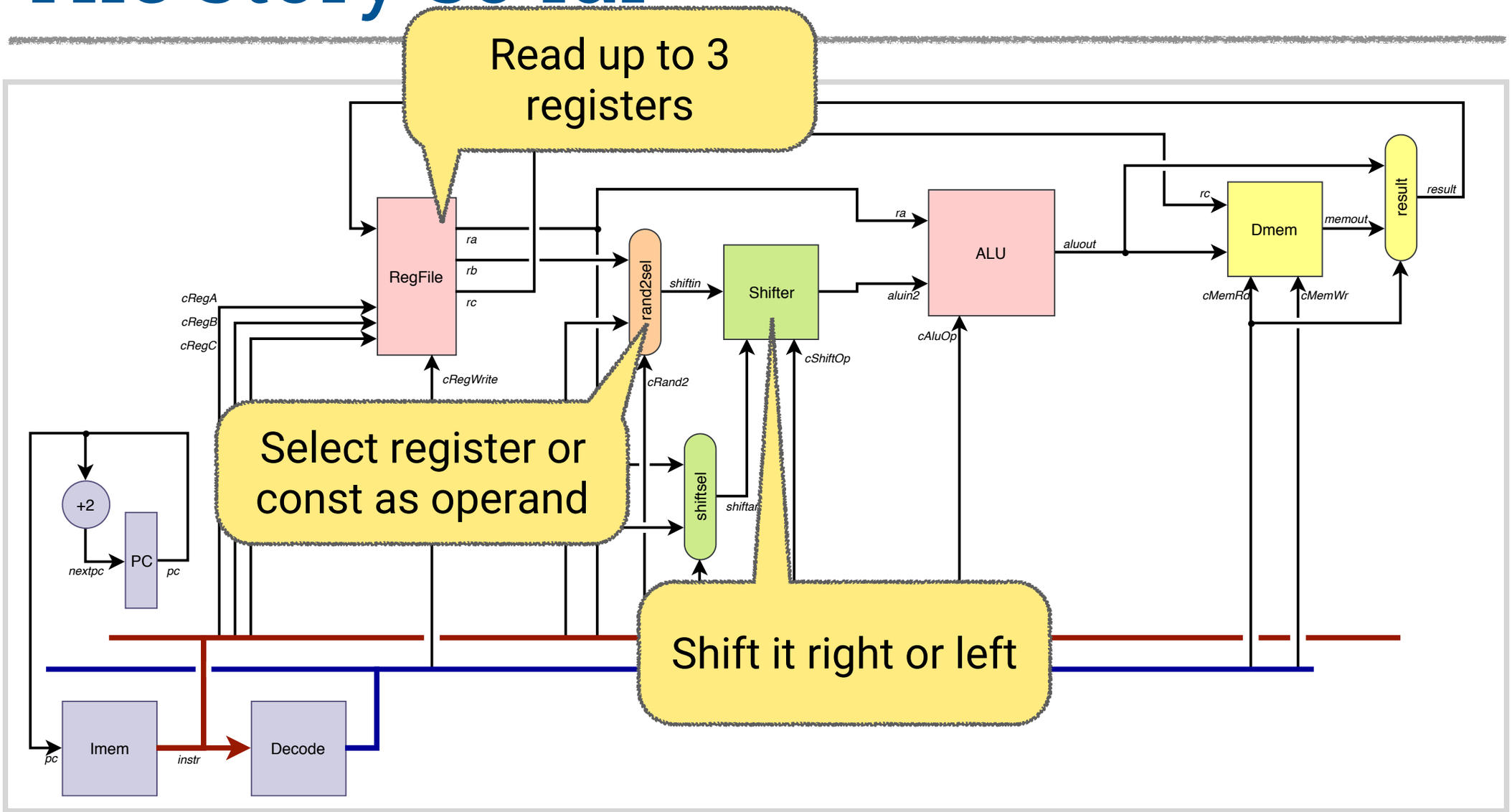
The story so far



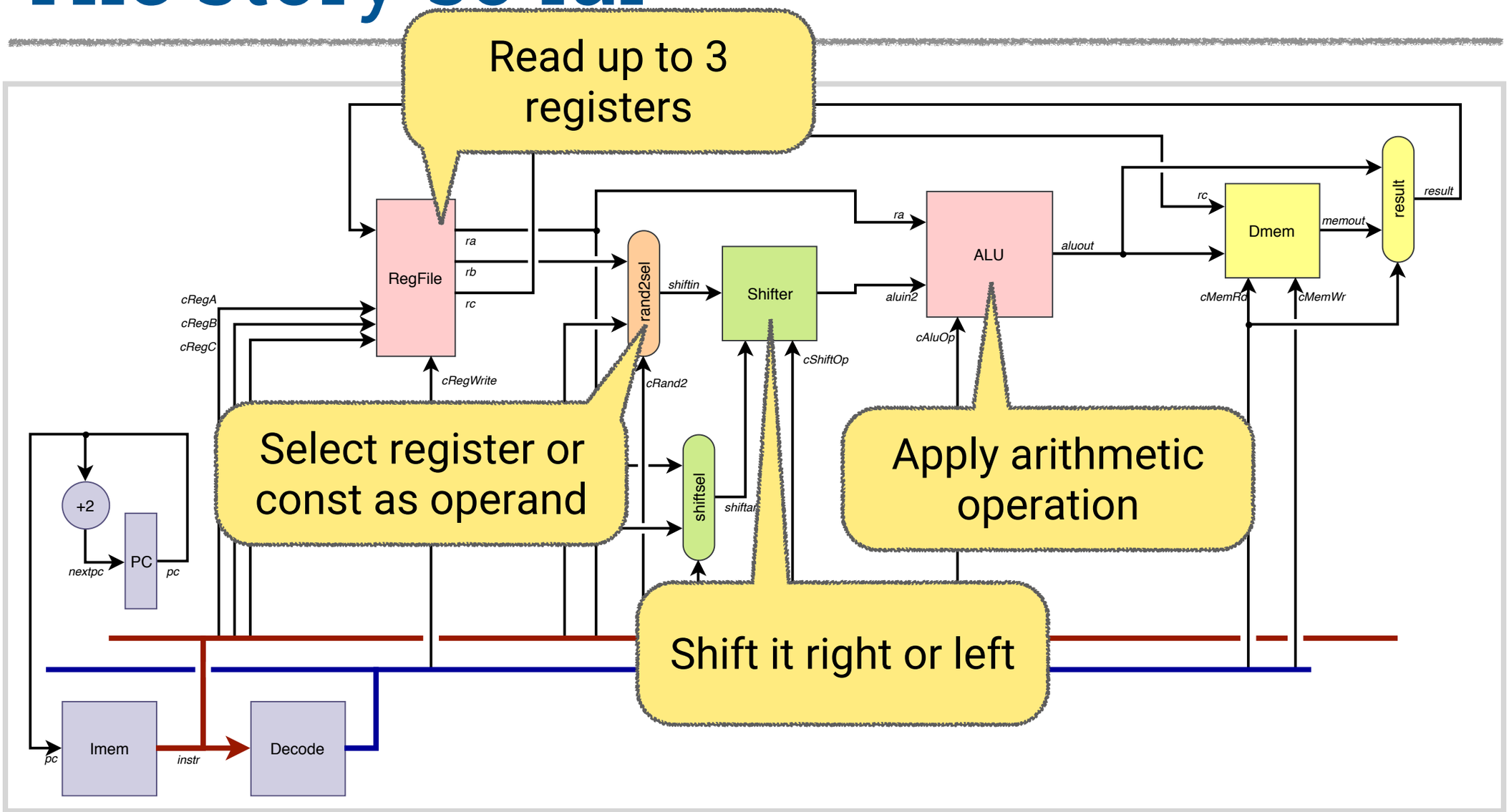
The story so far



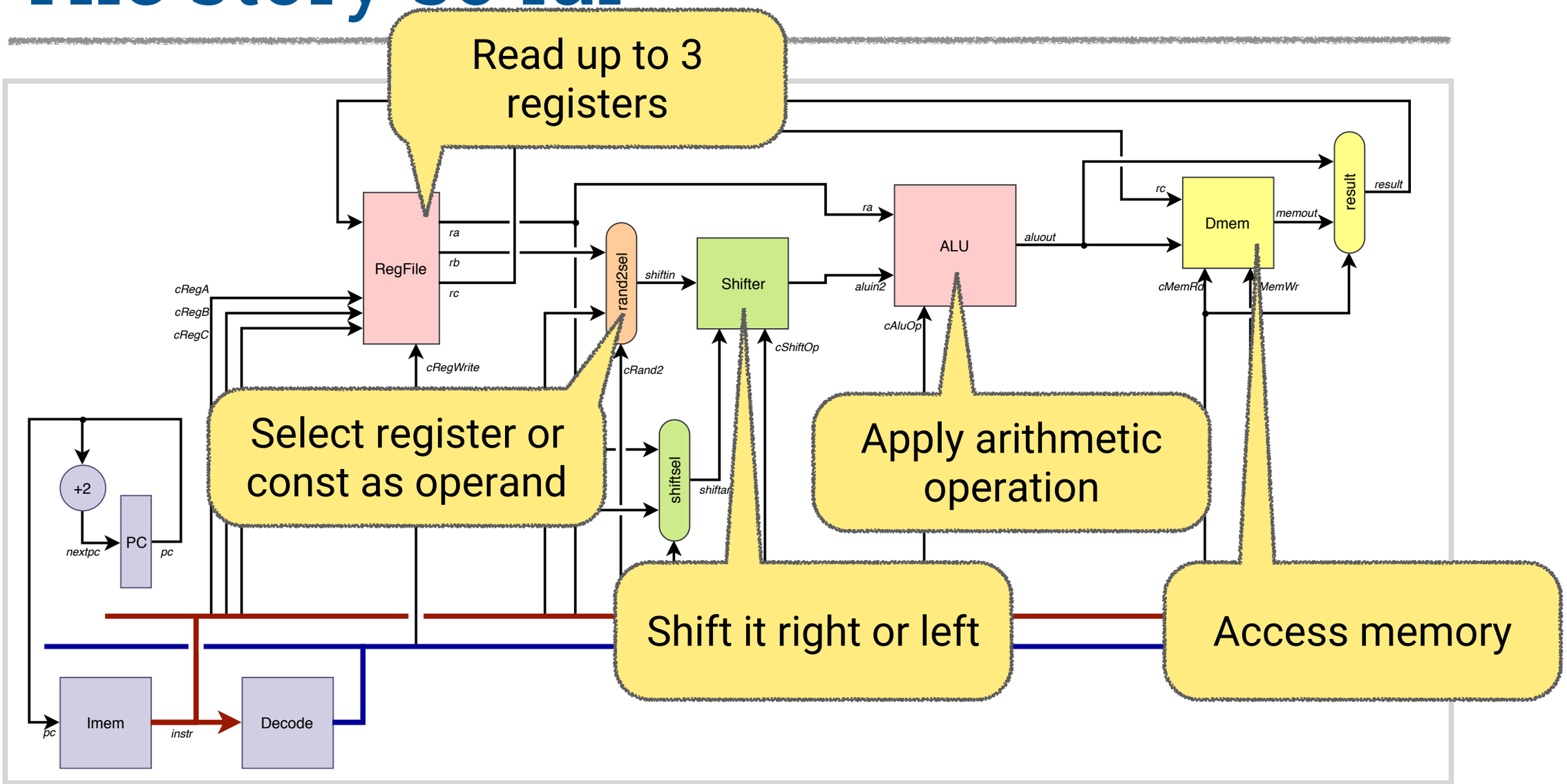
The story so far



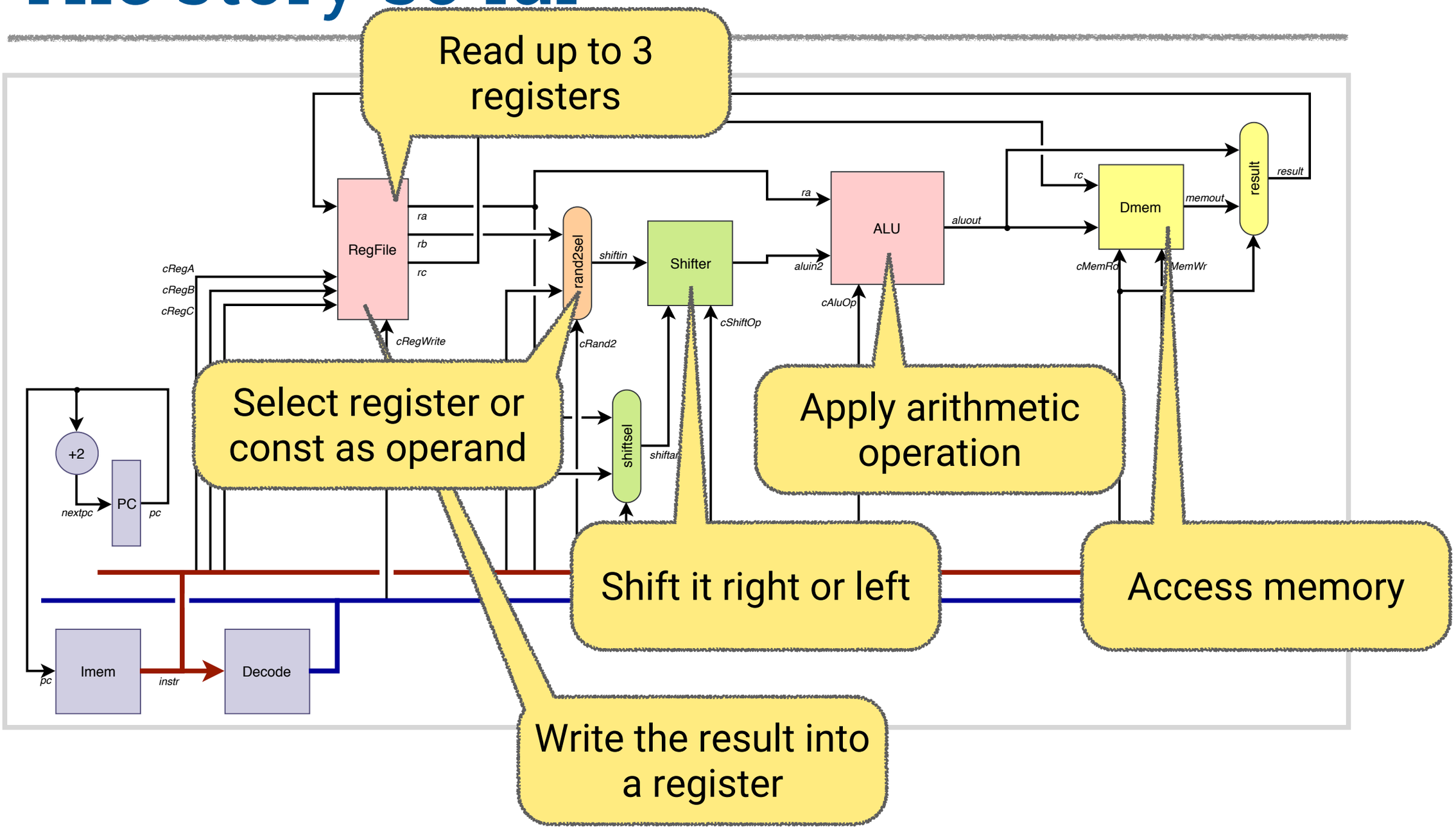
The story so far



The story so far



The story so far



Data and control

Digital Systems – Lecture 22

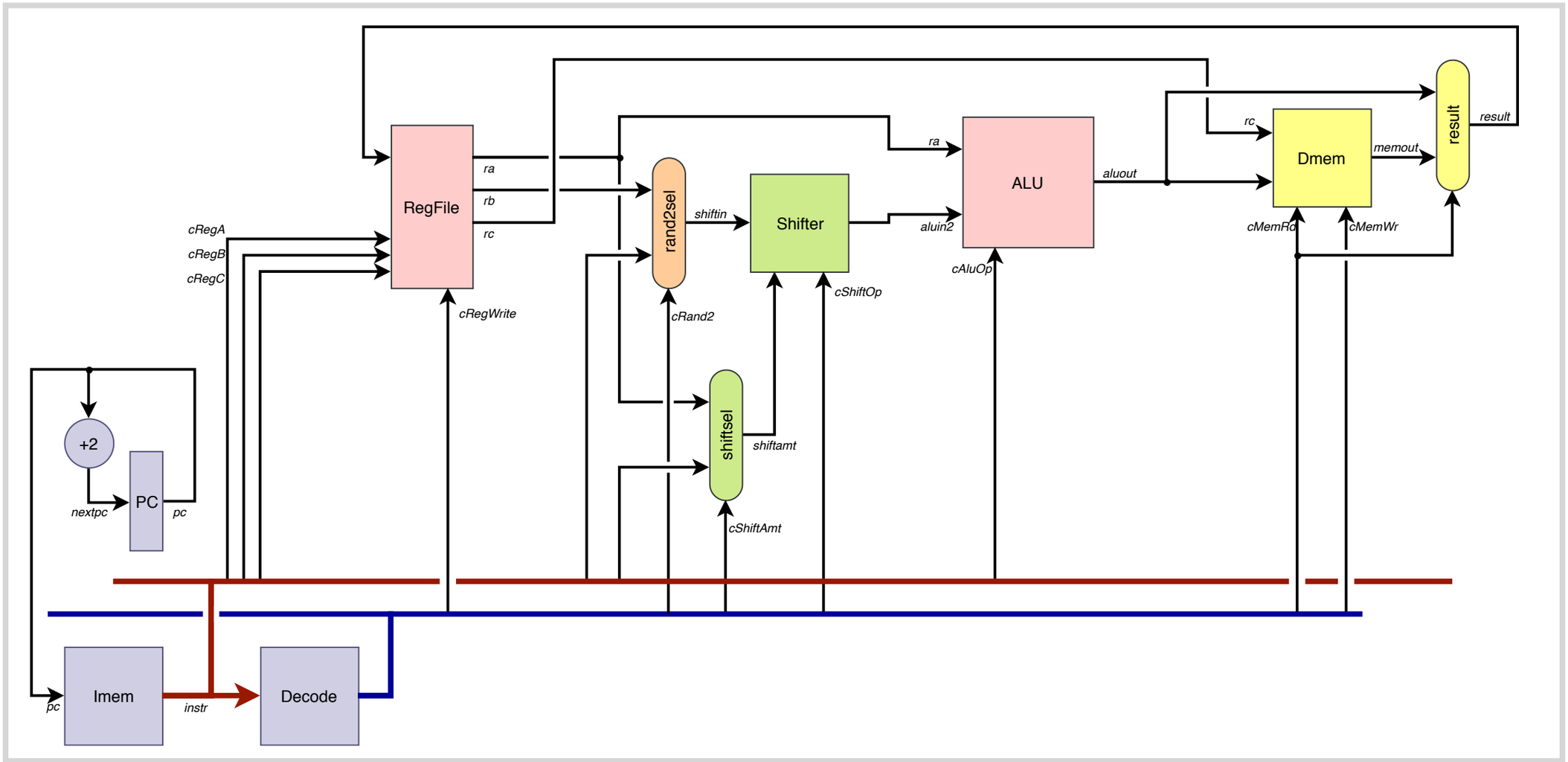
Department of
COMPUTER
SCIENCE

In this lecture

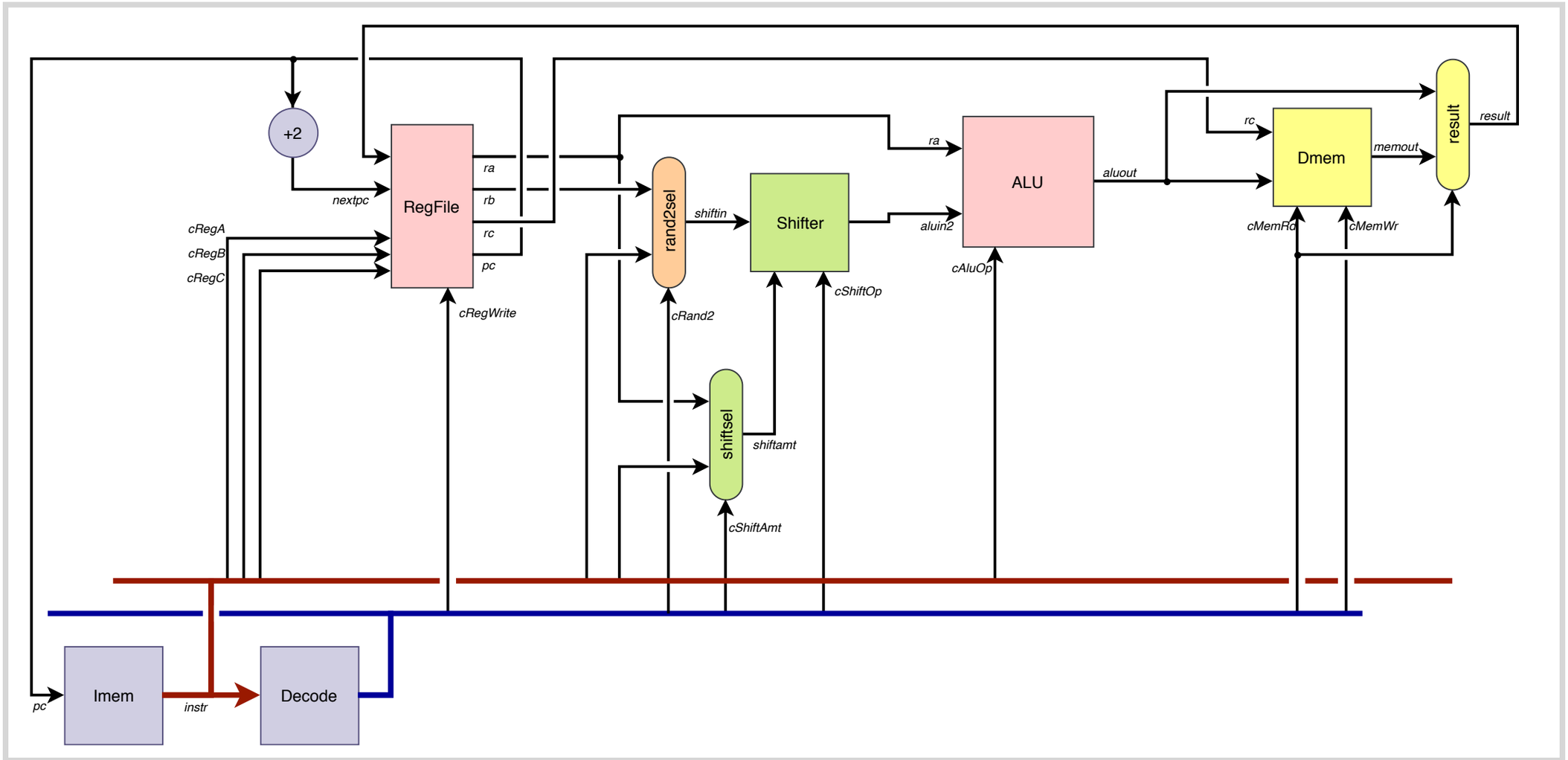
Further stages that enhance control elements of the datapath

- PC as a register
- Instruction decoding
- Subroutine calls
- Conditional execution

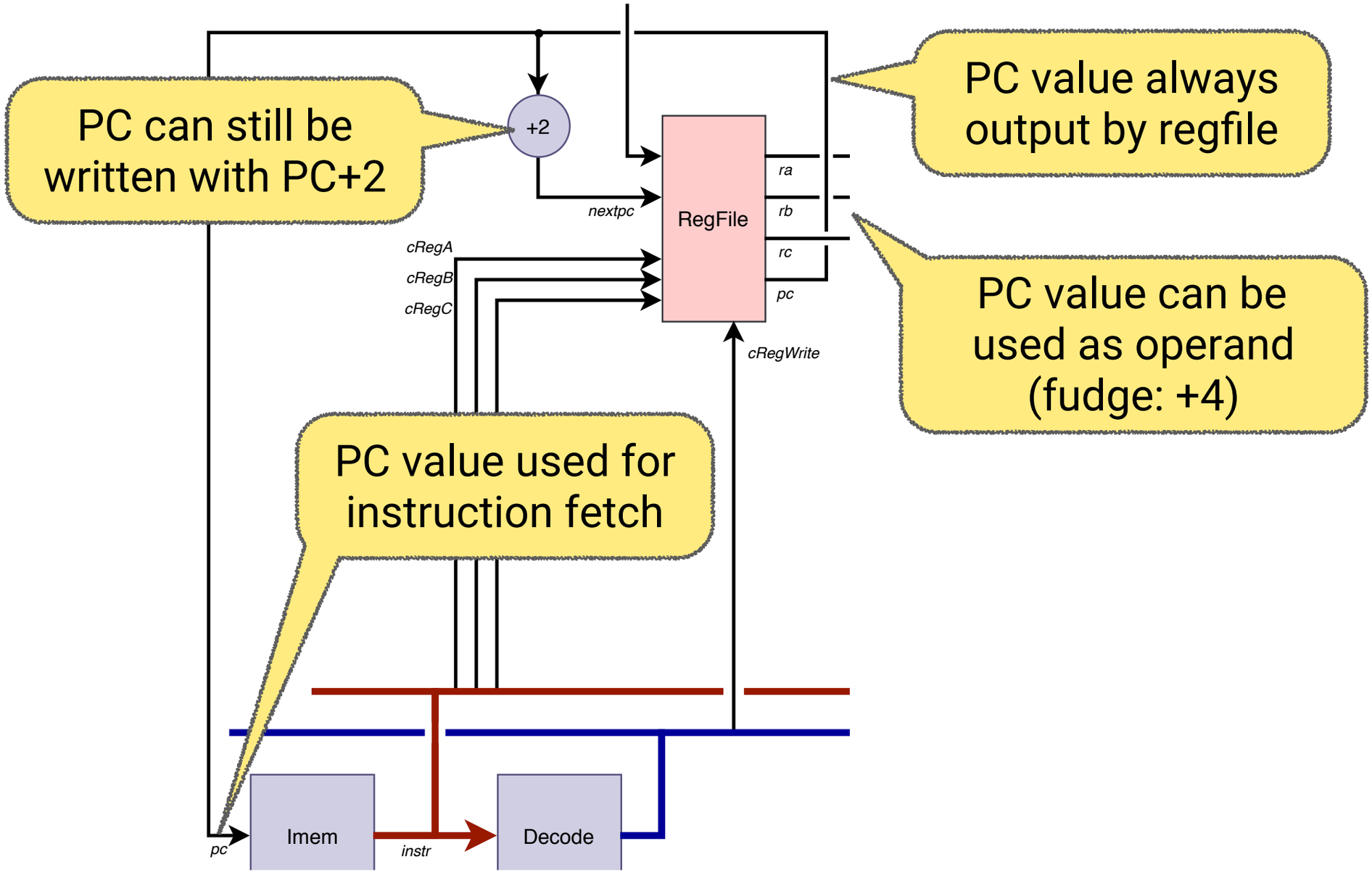
The story from last time



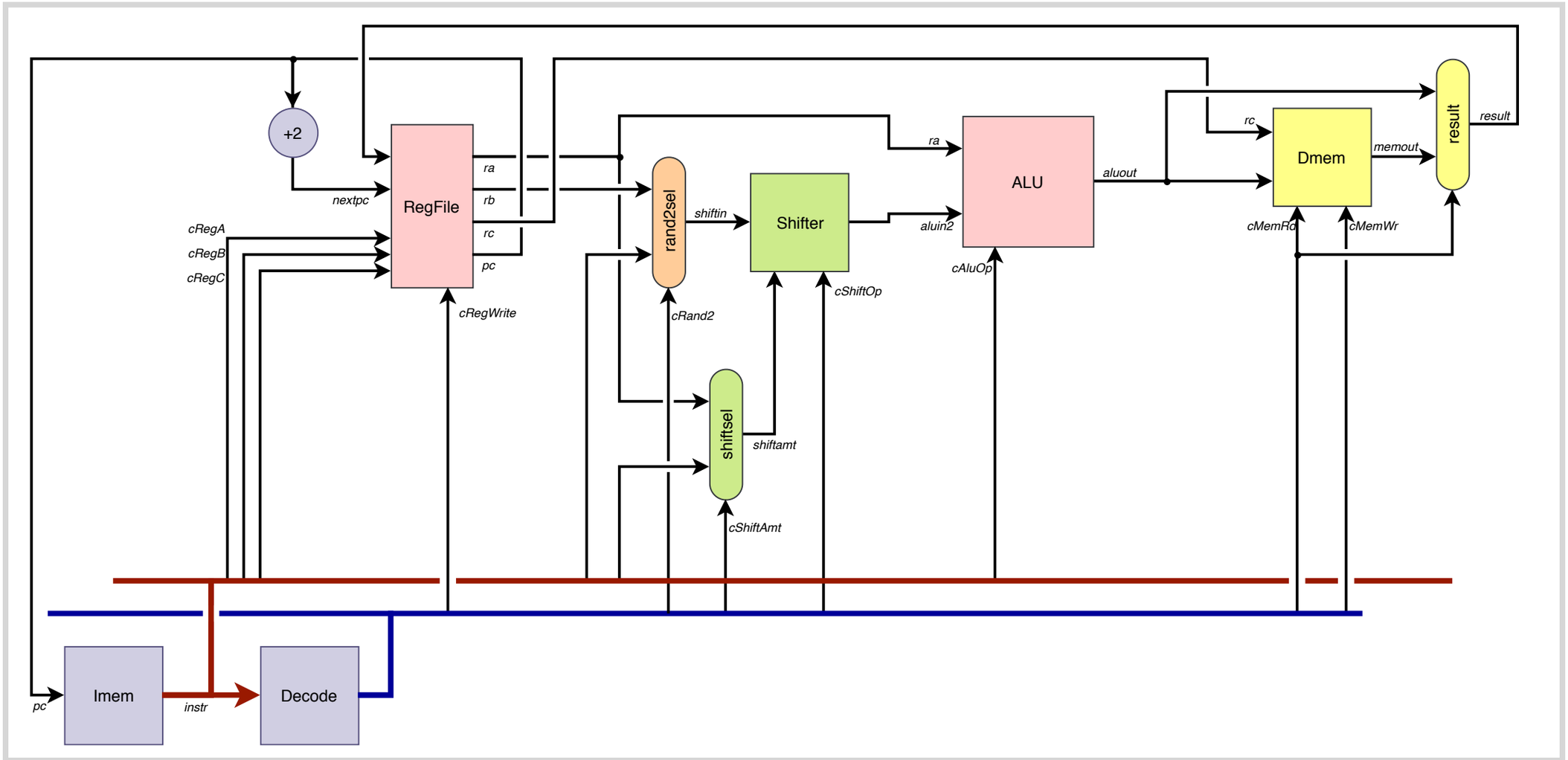
Stage 6: PC as a register



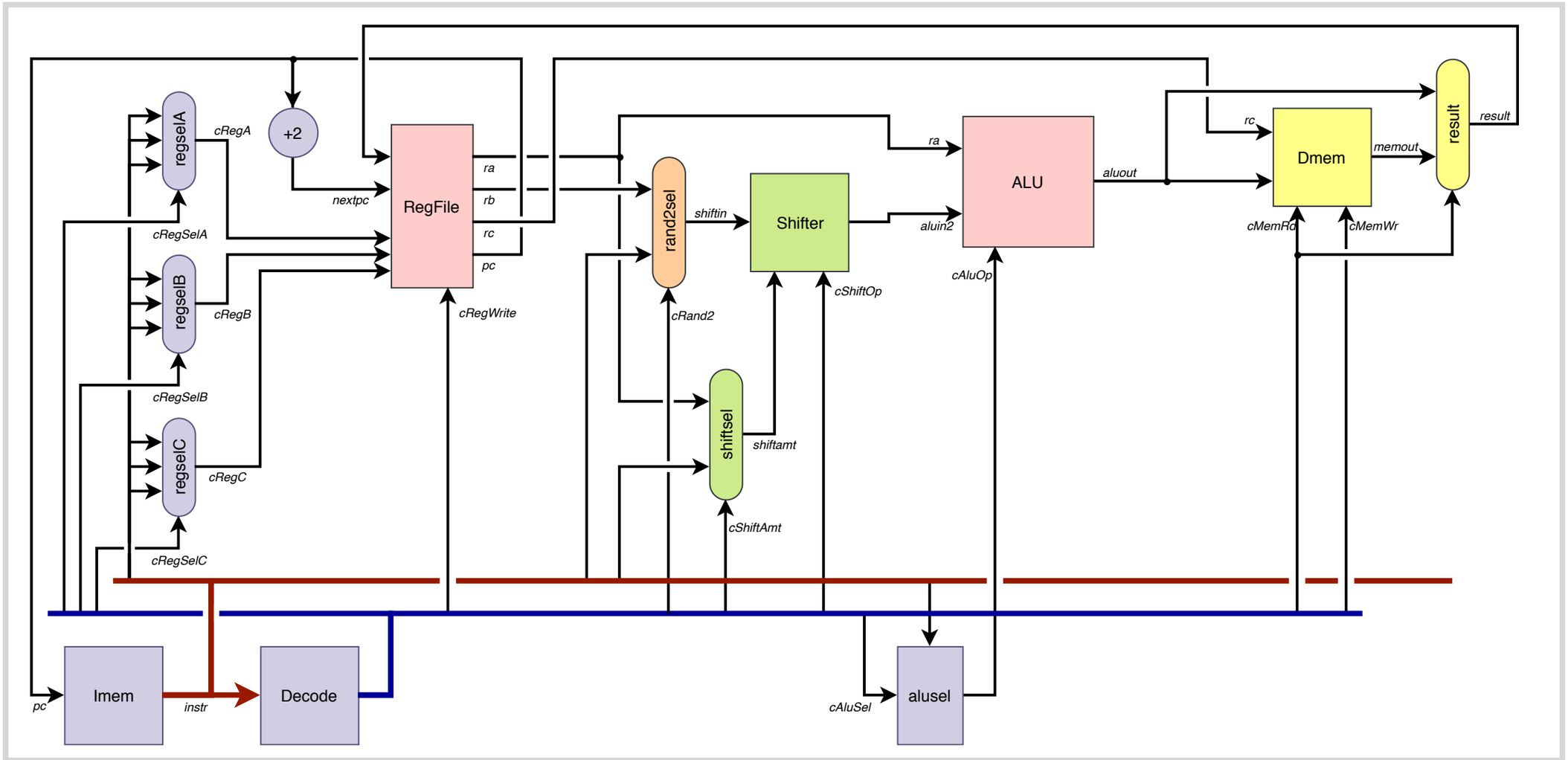
Stage 6: PC as a register



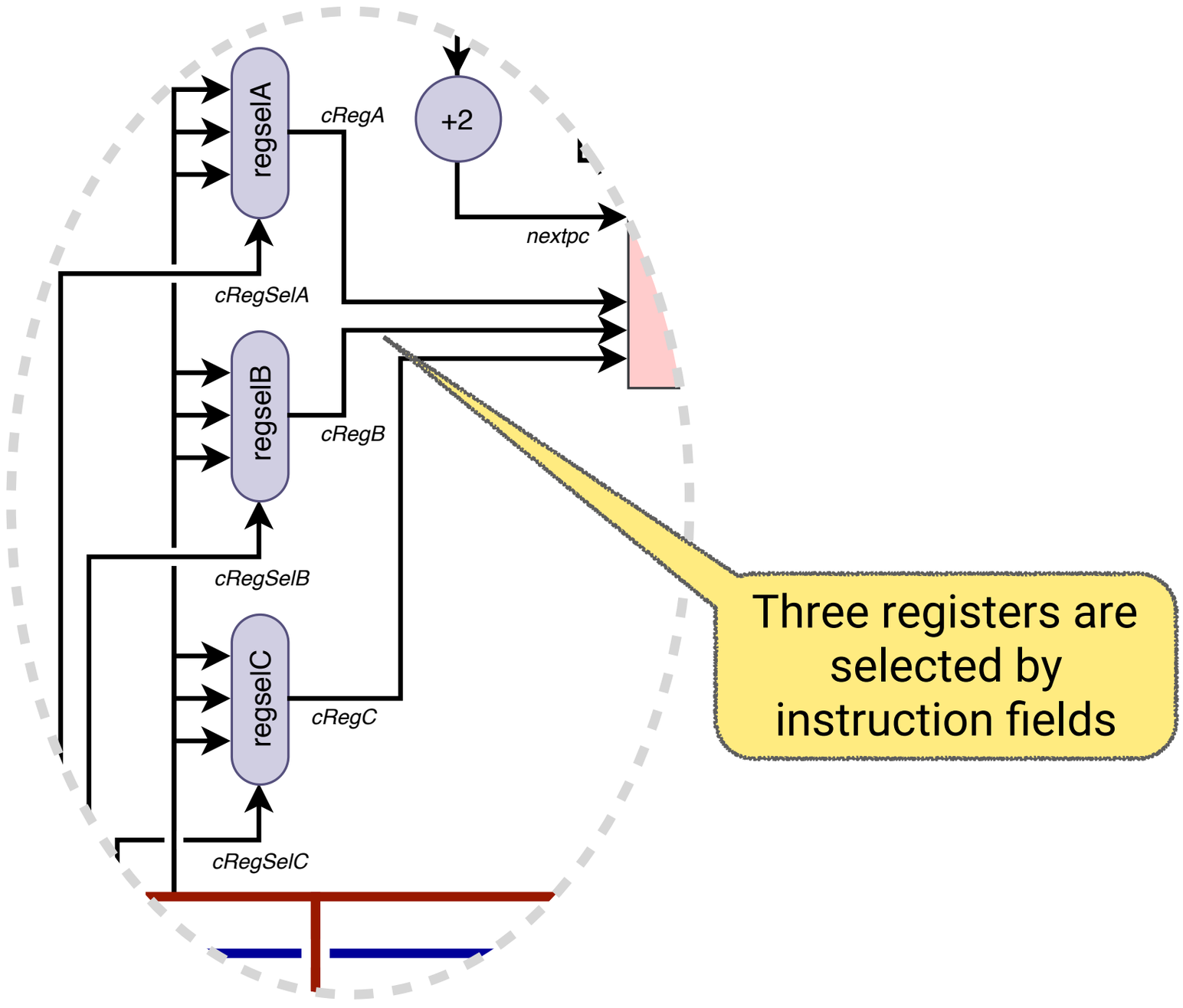
Stage 6: PC as a register



Stage 7: Instruction decoding



Stage 7: Instruction decoding



Controls for register selection

Instruction	cReg SelA	cReg SelB	cReg SelC	cRand2	cShiftOp/ Amt	cAlu Sel	cMem Rd/Wr	cReg Write
adds/subs r/i3	Ry	Rz	Rx	RegB	Lsl/Sh0	Sg9	F/F	T
movs i8	–	–	Rw	Imm8	Lsl/Sh0	Mov	F/F	T
ldr r	Ry	Rz	Rx	RegB	Lsl/Sh0	Add	T/F	T
str r	Ry	Rz	Rx	RegB	Lsl/Sh0	Add	F/T	F
lsls i5	–	Ry	Rx	RegB	Lsl/ShImm	Mov	F/F	T
ands r	Rx	Ry	Rx	RegB	Lsl/Sh0	And	F/F	T
rors r	Ry	Rx	Rx	RegB	Ror/ShReg	Mov	F/F	T
ldr i5	Ry	–	Rx	Imm5	Lsl/Sh2	Add	T/F	T
str i5	Ry	–	Rx	Imm5	Lsl/Sh2	Add	F/T	F

Instructions with fixed registers

add sp,sp,#⟨imm7⟩

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	1	0	0	0	0	0	imm7						

sub sp,sp,#⟨imm7⟩

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	1	0	0	0	0	1	imm7						

add ⟨Rw⟩,sp,#⟨imm8⟩

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

ldr ⟨Rw⟩,[sp,#⟨imm8⟩]

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

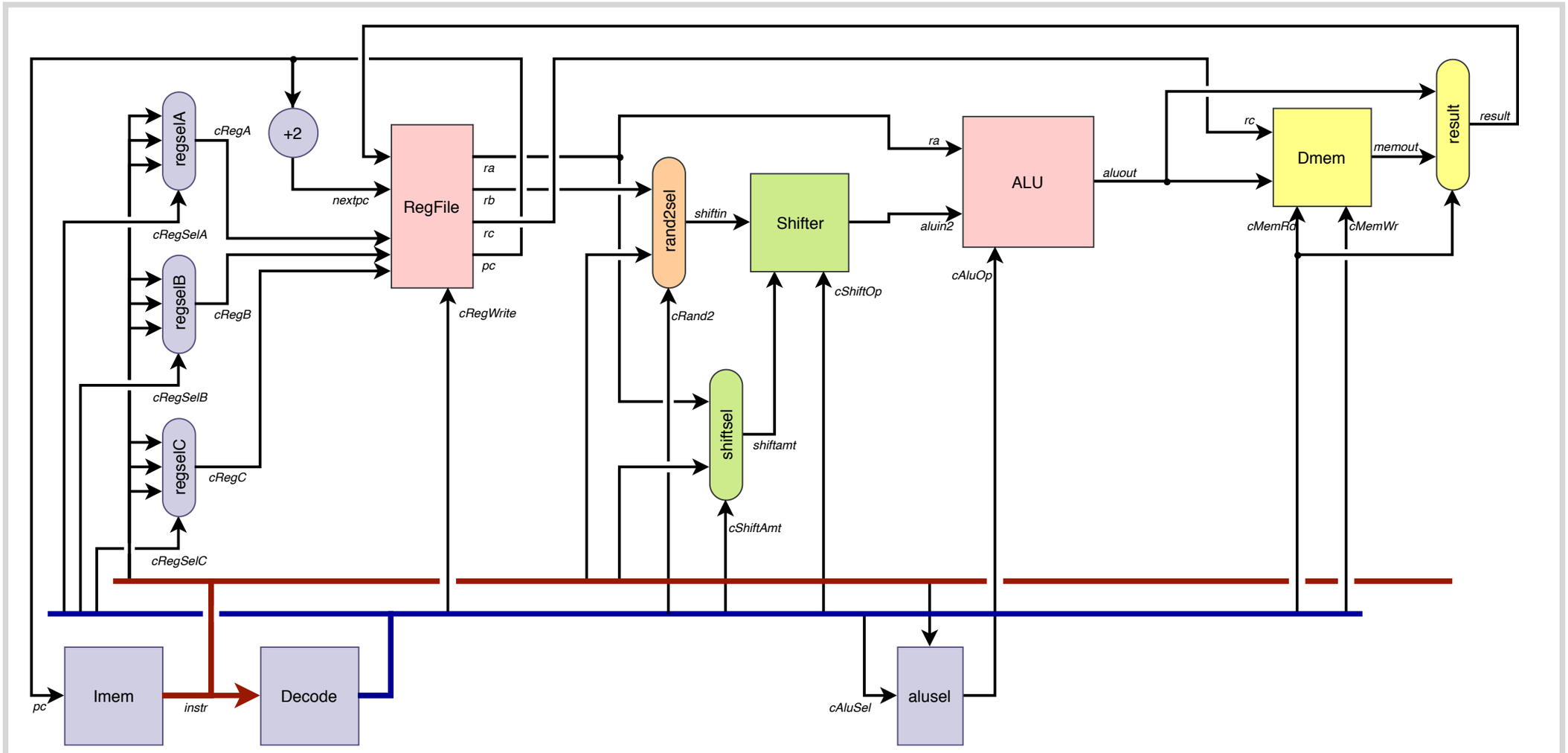
b ⟨disp11⟩

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	0	0	disp11										

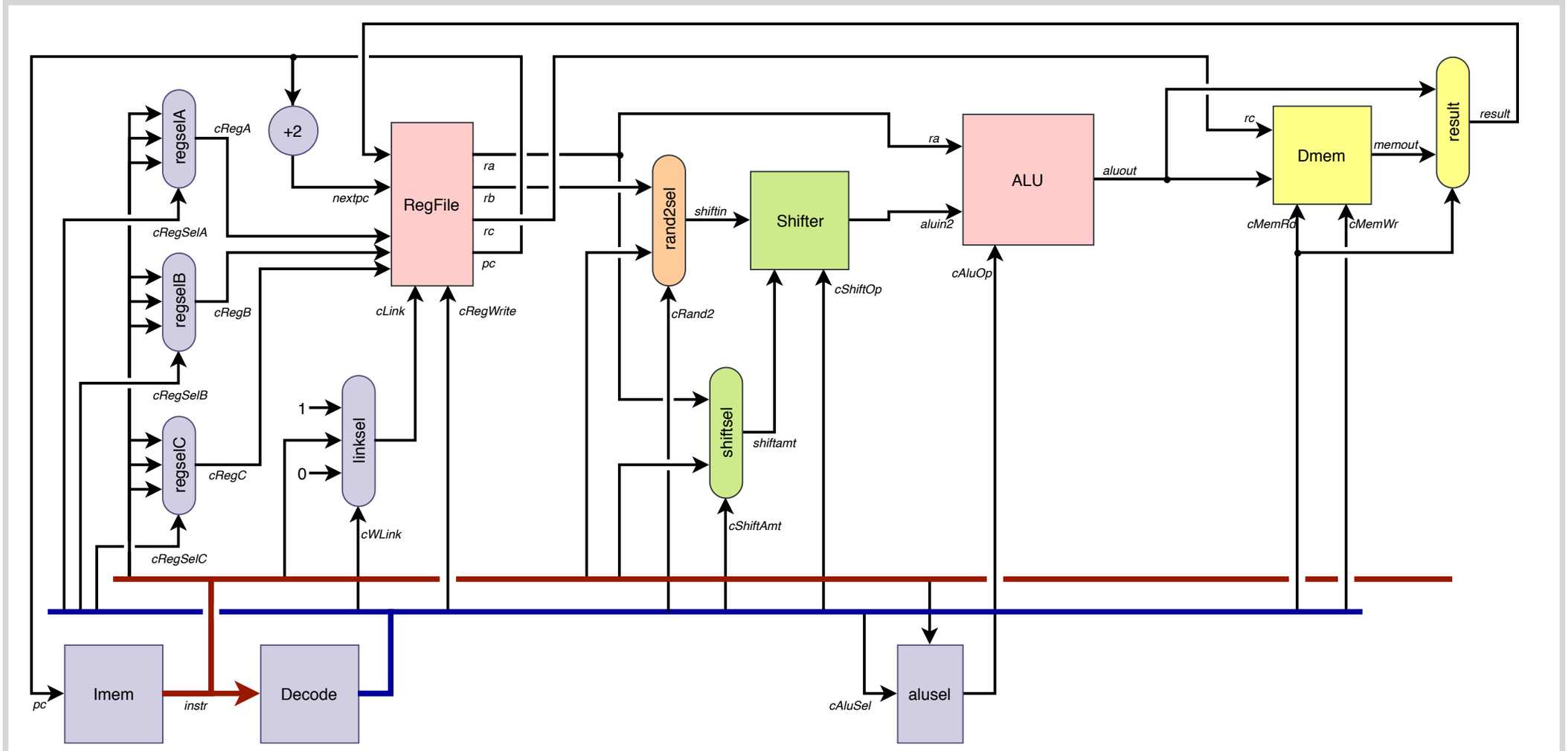
More control rules

Instruction	cReg SelA	cReg SelB	cReg SelC	cRand2	cShiftOp/ Amt	cAlu Sel	cMem Rd/Wr	cReg Write
add/sub sp	Rsp	–	Rsp	Imm7	Lsl/Sh2	Sg7	F/F	T
add rsp	Rsp	–	Rw	Imm8	Lsl/Sh2	Add	F/F	T
ldr sp	Rsp	–	Rw	Imm8	Lsl/Sh2	Add	T/F	T
str sp	Rsp	–	Rw	Imm8	Lsl/Sh2	Add	F/T	F
b	Rpc	–	Rpc	SImm11	Lsl/Sh1	Add	F/F	T

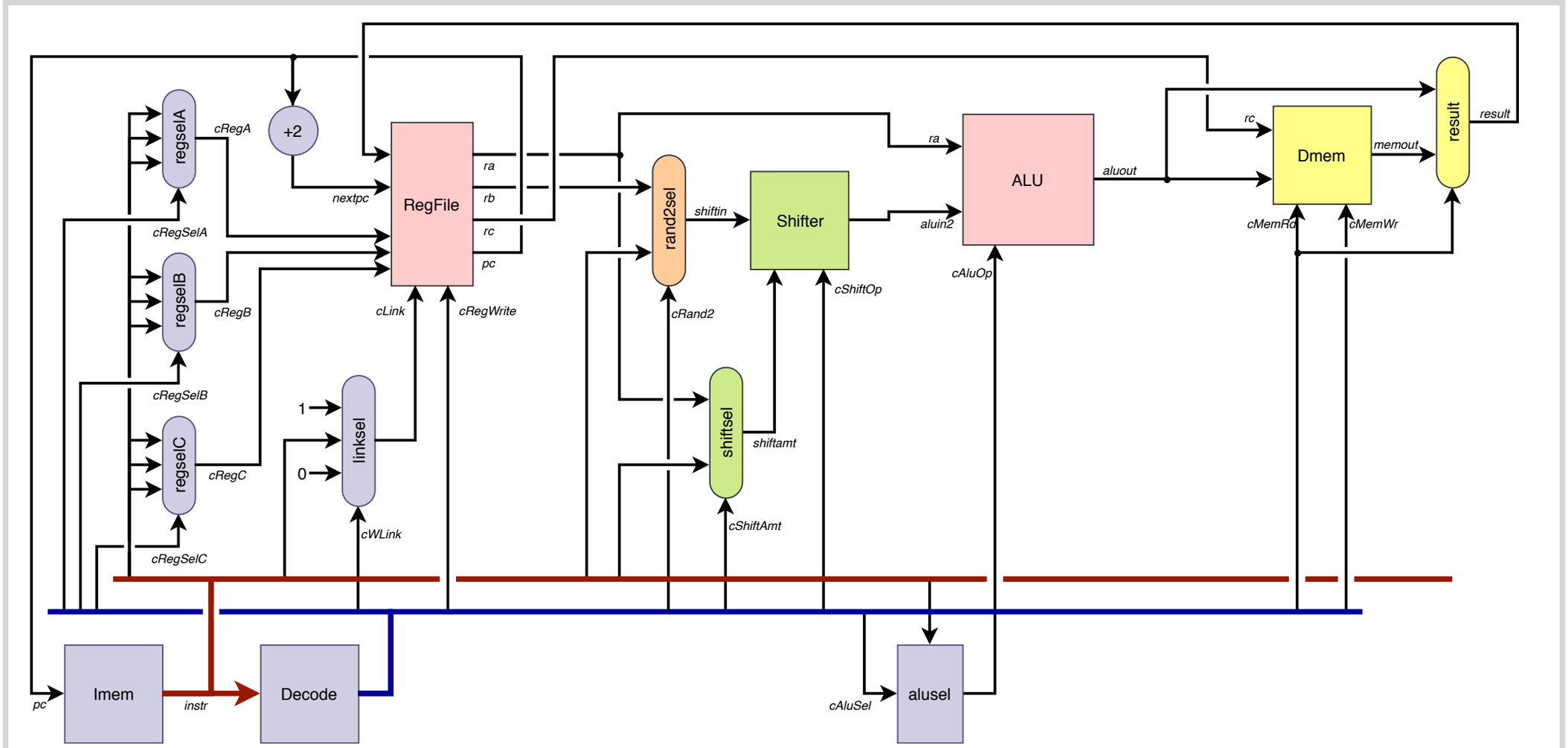
Stage 7: Instruction decoding



Stage 8: Subroutine calls



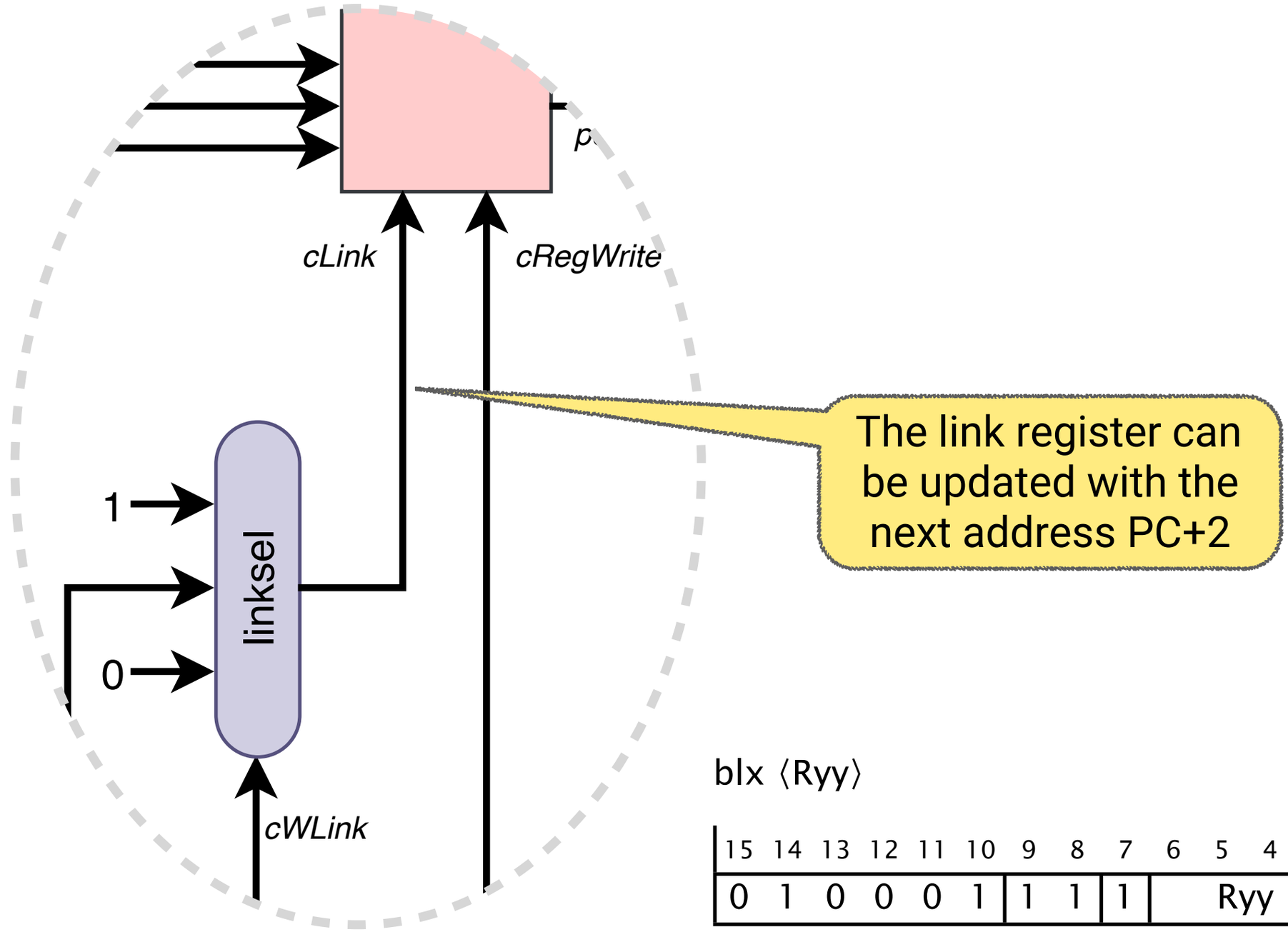
Stage 8: Subroutine calls



blx ⟨Ryy⟩

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

Stage 8: Subroutine calls

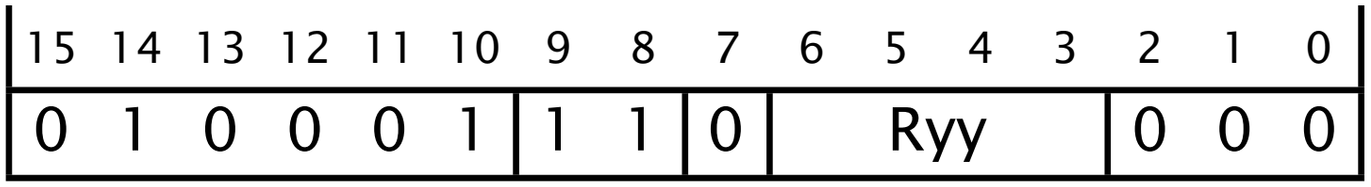


blx ⟨Ryy⟩

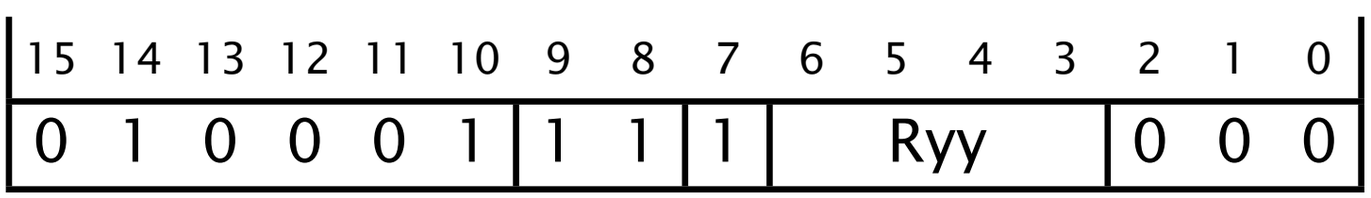
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	0	1	1	1	1	Ryy				0	0	0

Subroutine call and return

bx ⟨Ryy⟩



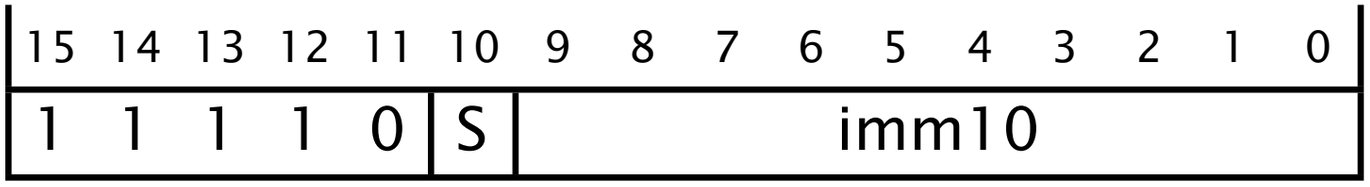
blx ⟨Ryy⟩



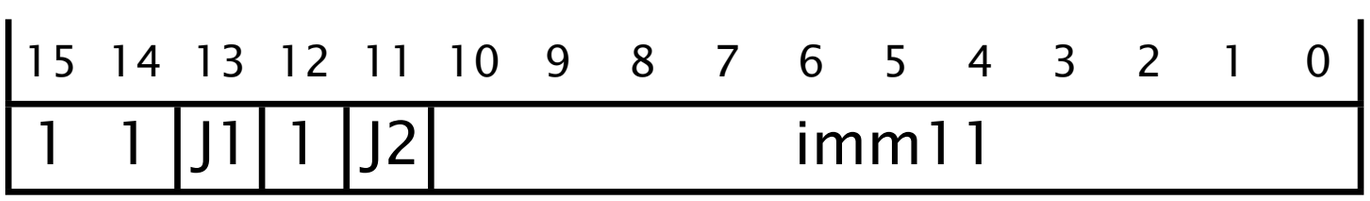
Instruction	cRegSel A/B/C	cRand2	cShiftOp/ Amt	cAlu Sel	cMem Rd/Wr	cReg Write	cWLink
`bx/blx r`	`-/Ryy/Rpc`	`RegB`	`Lsl/Sh0`	`Mov`	`F/F`	`T`	`C`

Bl done the old-fashioned way

bl1, ⟨simm10⟩

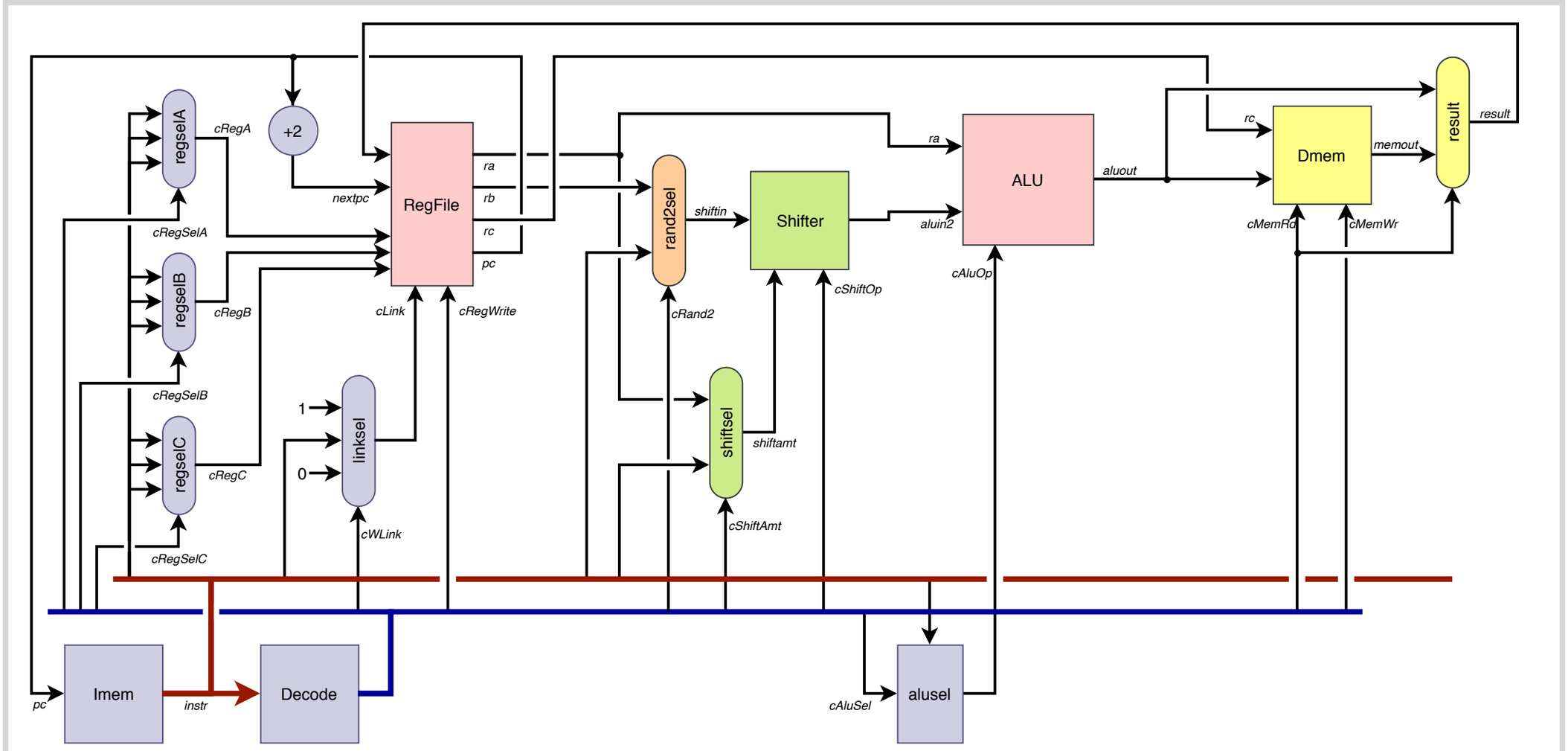


bl2, ⟨imm11⟩

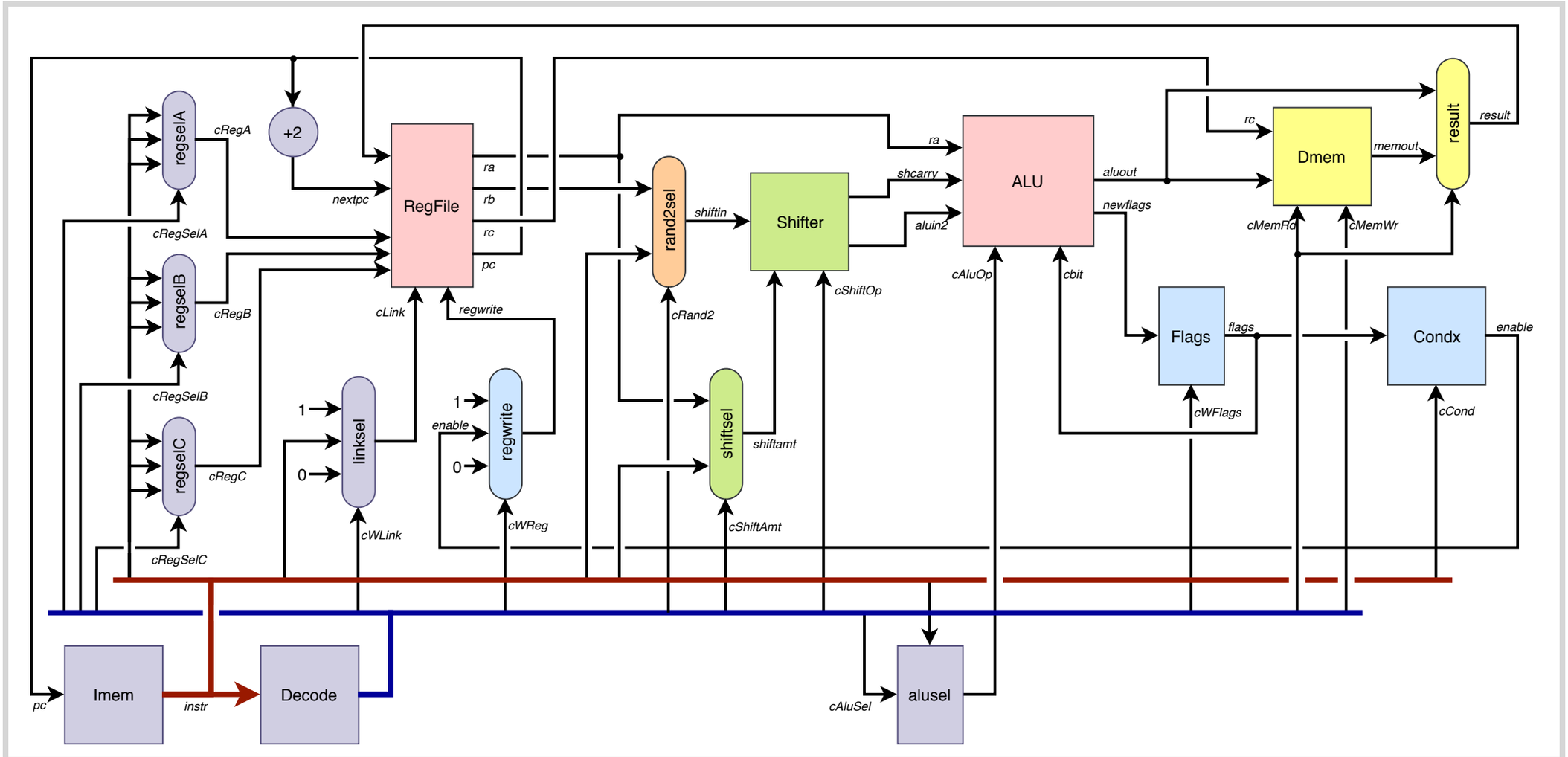


Instruction	cRegSel A/B/C	cRand2	cShiftOp/ Amt	cAlu Sel	cMem Rd/Wr	cReg Write	cWLink
bl1	Rpc/–/Rlr	SImm11	Lsl/Sh12	Add	F/F	T	N
bl2	Rlr/–/Rpc	Imm11	Lsl/Sh1	Add	F/F	T	Y

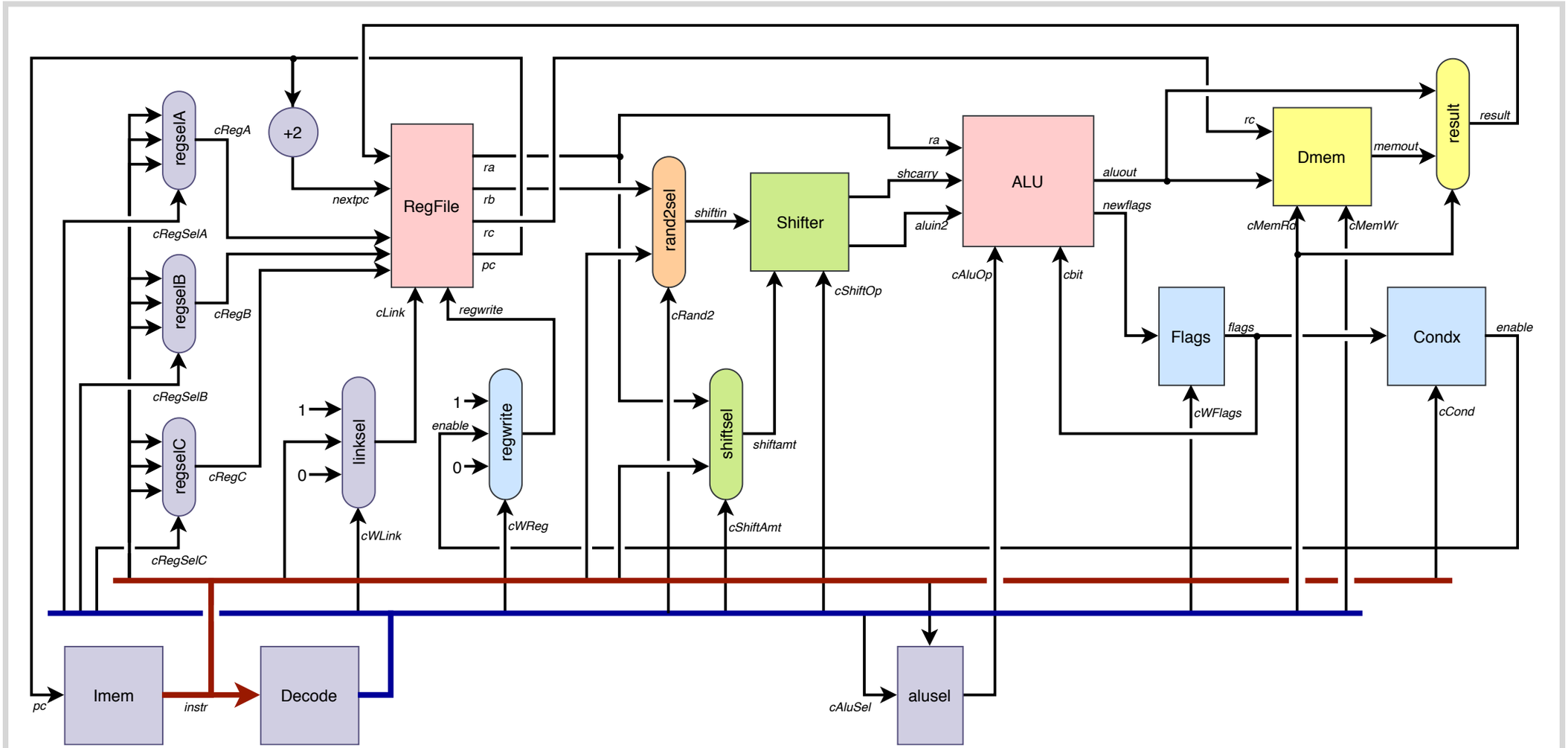
Stage 8: Subroutine calls



Stage 9: Conditional execution



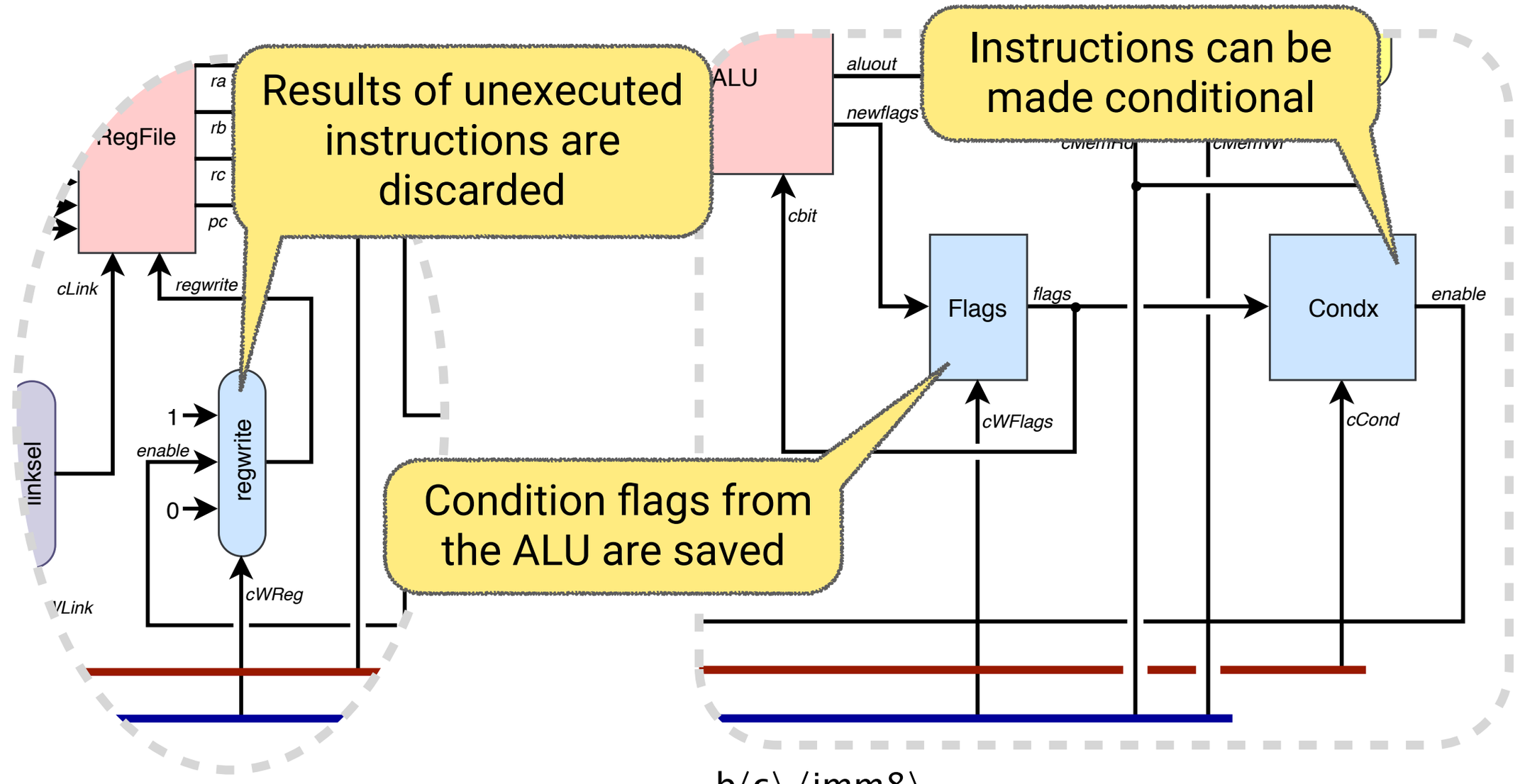
Stage 9: Conditional execution



b⟨c⟩ ⟨imm8⟩

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

Stage 9: Conditional execution

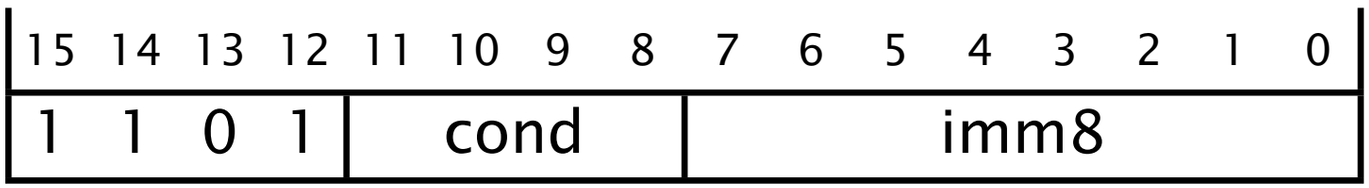


b⟨c⟩ ⟨imm8⟩

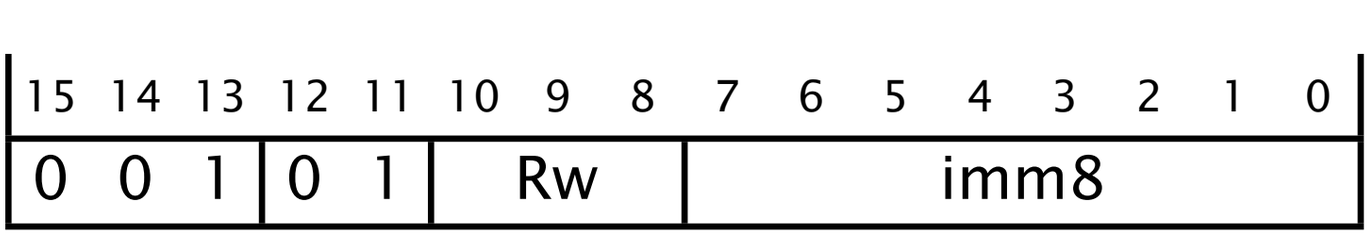
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	0	1	cond				imm8							

Compare and branch

b⟨c⟩ ⟨imm8⟩

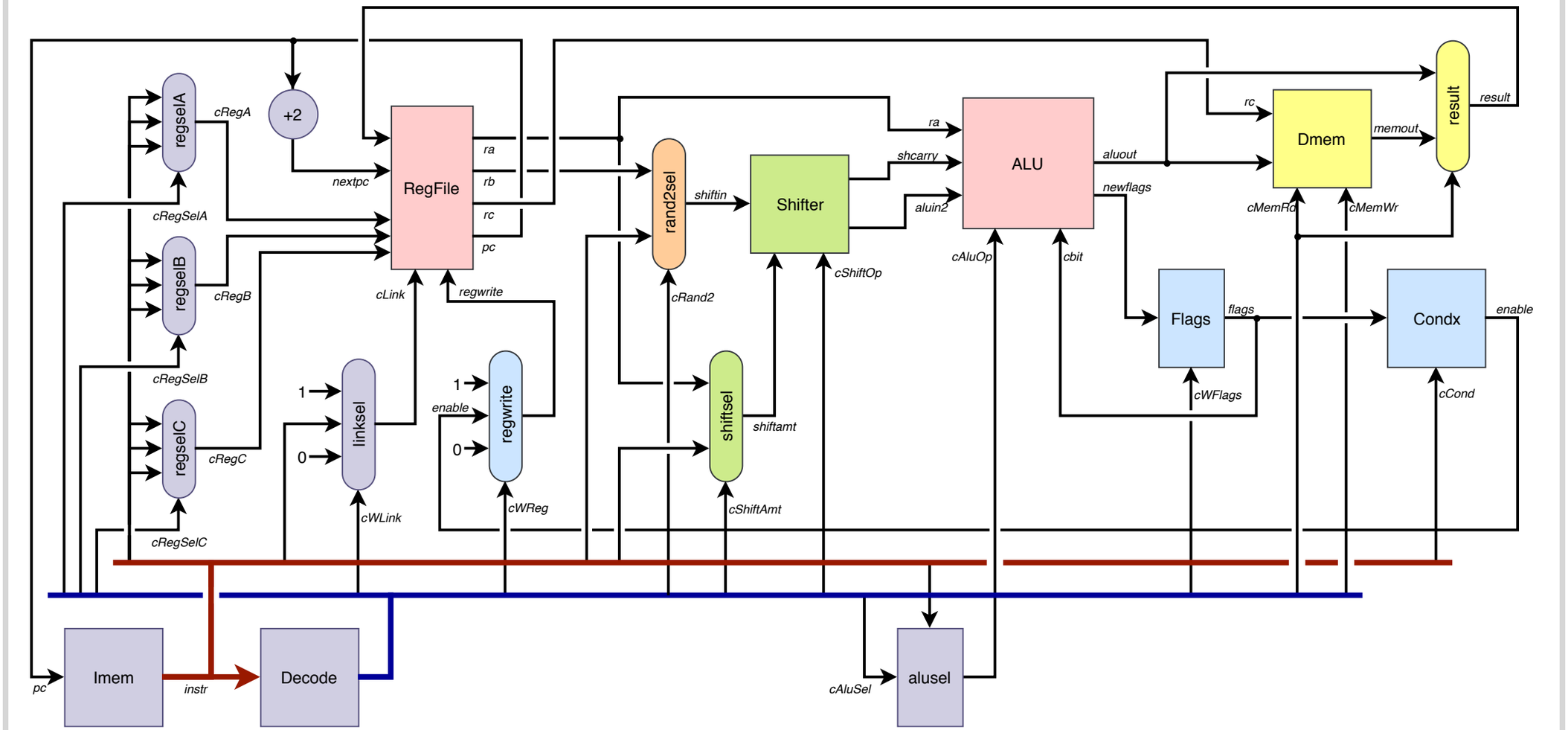


cmp ⟨Rw⟩, #⟨imm8⟩

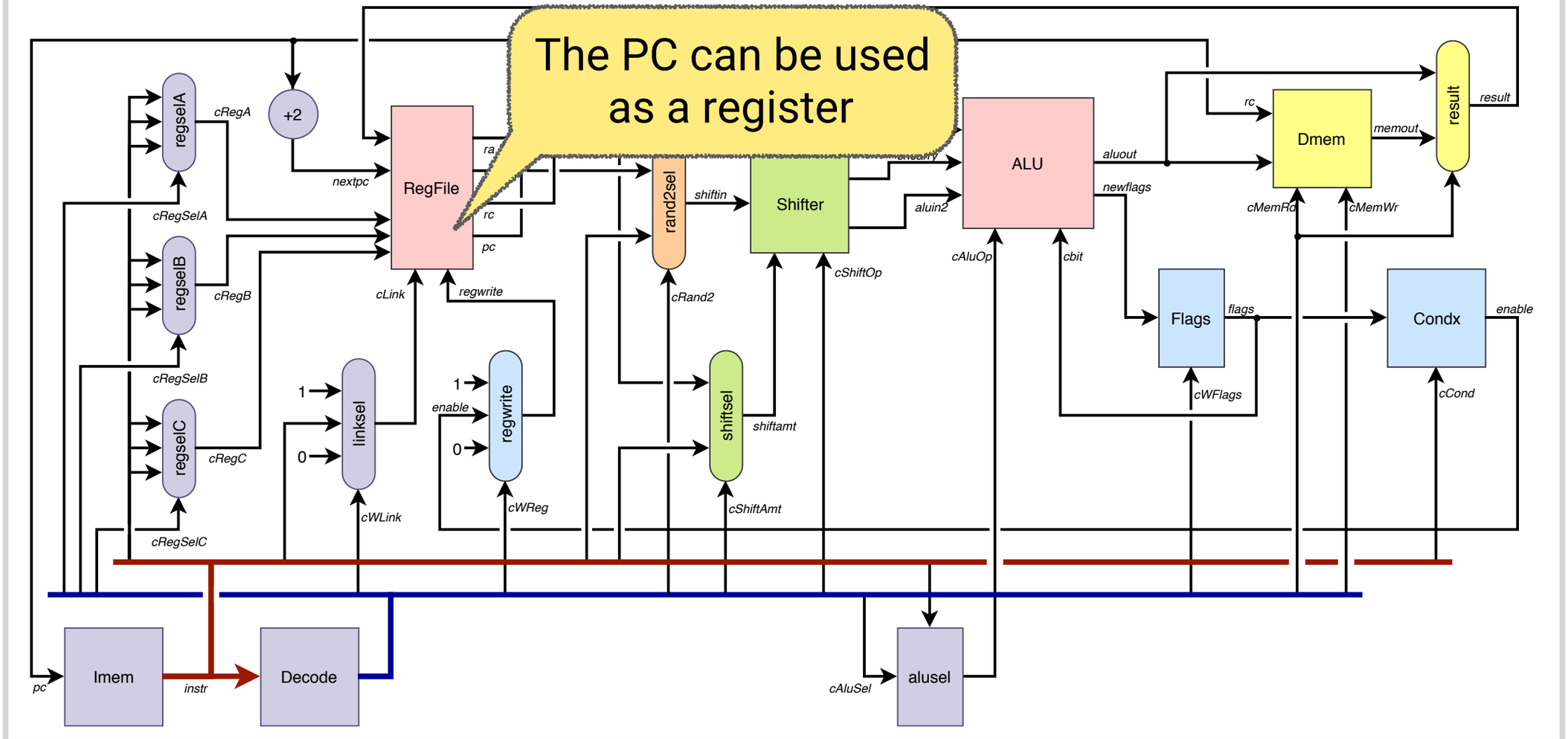


Instruction	cRegSel A/B/C	cRand2	cShiftOp/ Amt	cAlu Sel	cMem Rd/Wr	cWFlags/ Reg/Link
b<c>	Rpc/–/Rpc	SImm8	Lsl/Sh1	Add	F/F	F/C/N
cmp ri	Rw/–/–	Imm8	Lsl/Sh0	Sub	F/F	T/N/N
subs ri	Rw/–/Rw	Imm8	Lsl/Sh0	Sub	F/F	T/Y/N

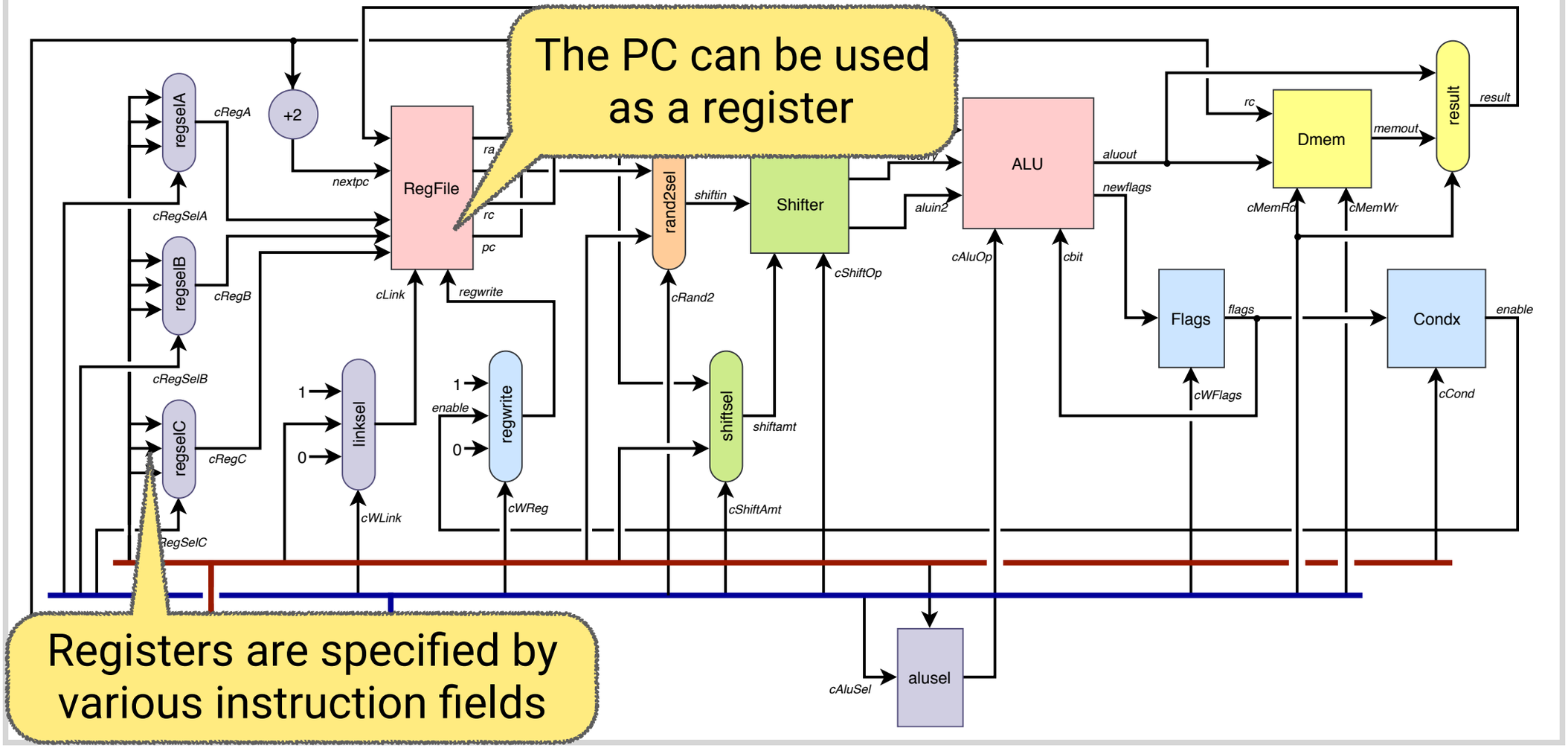
The complete datapath



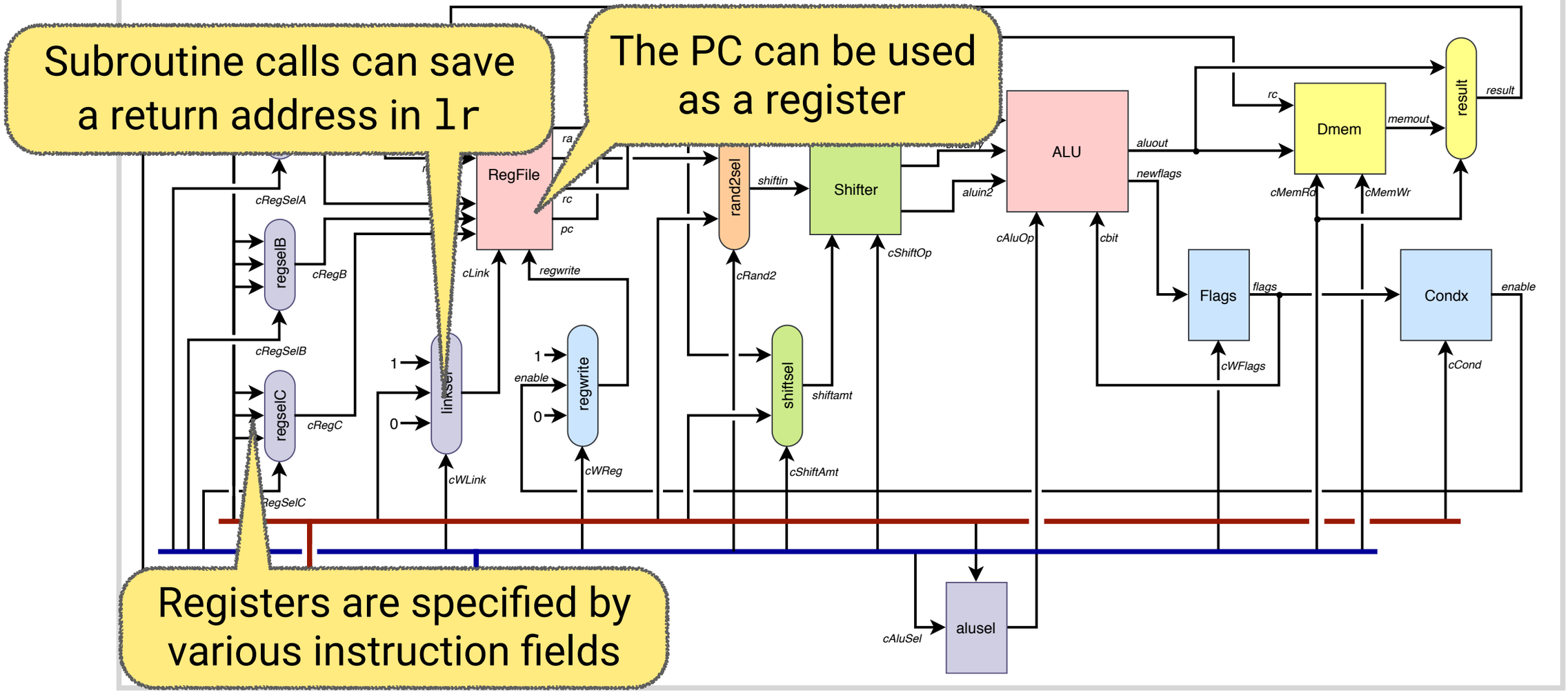
The complete datapath



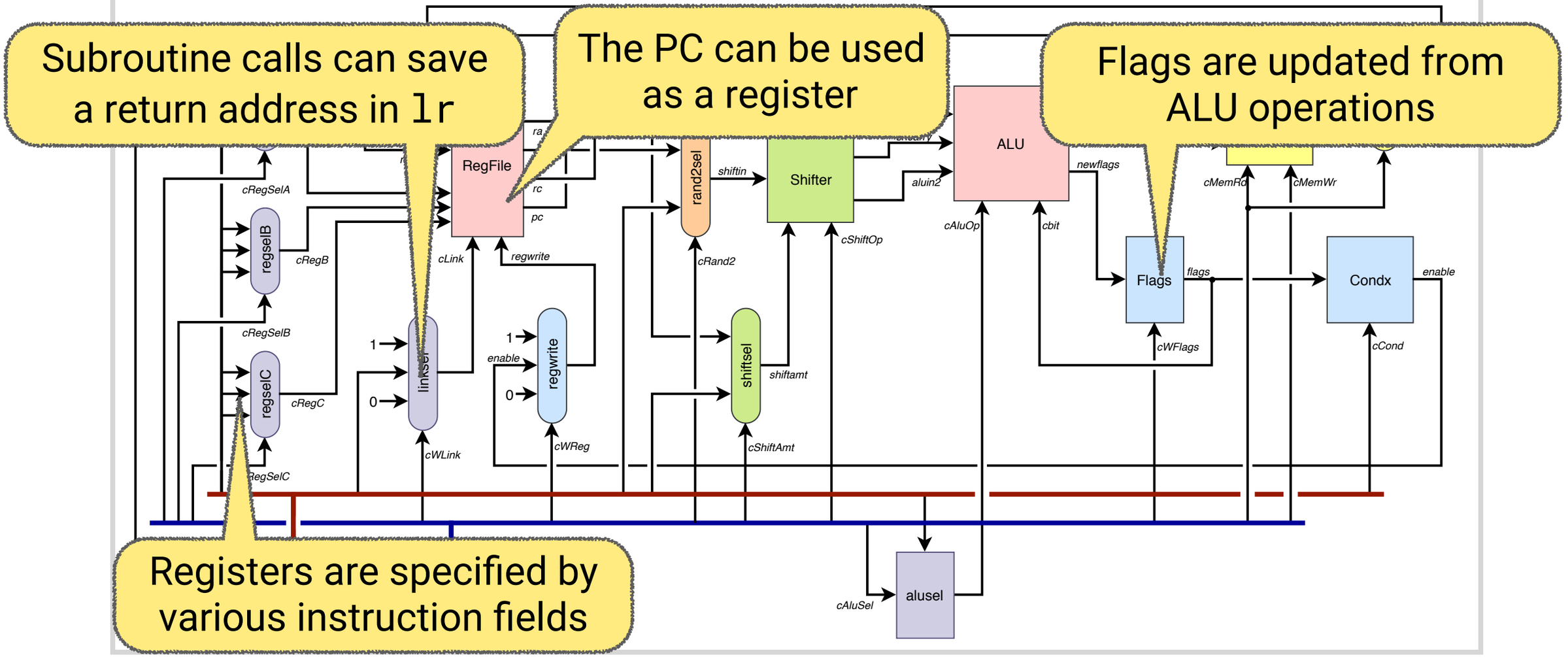
The complete datapath



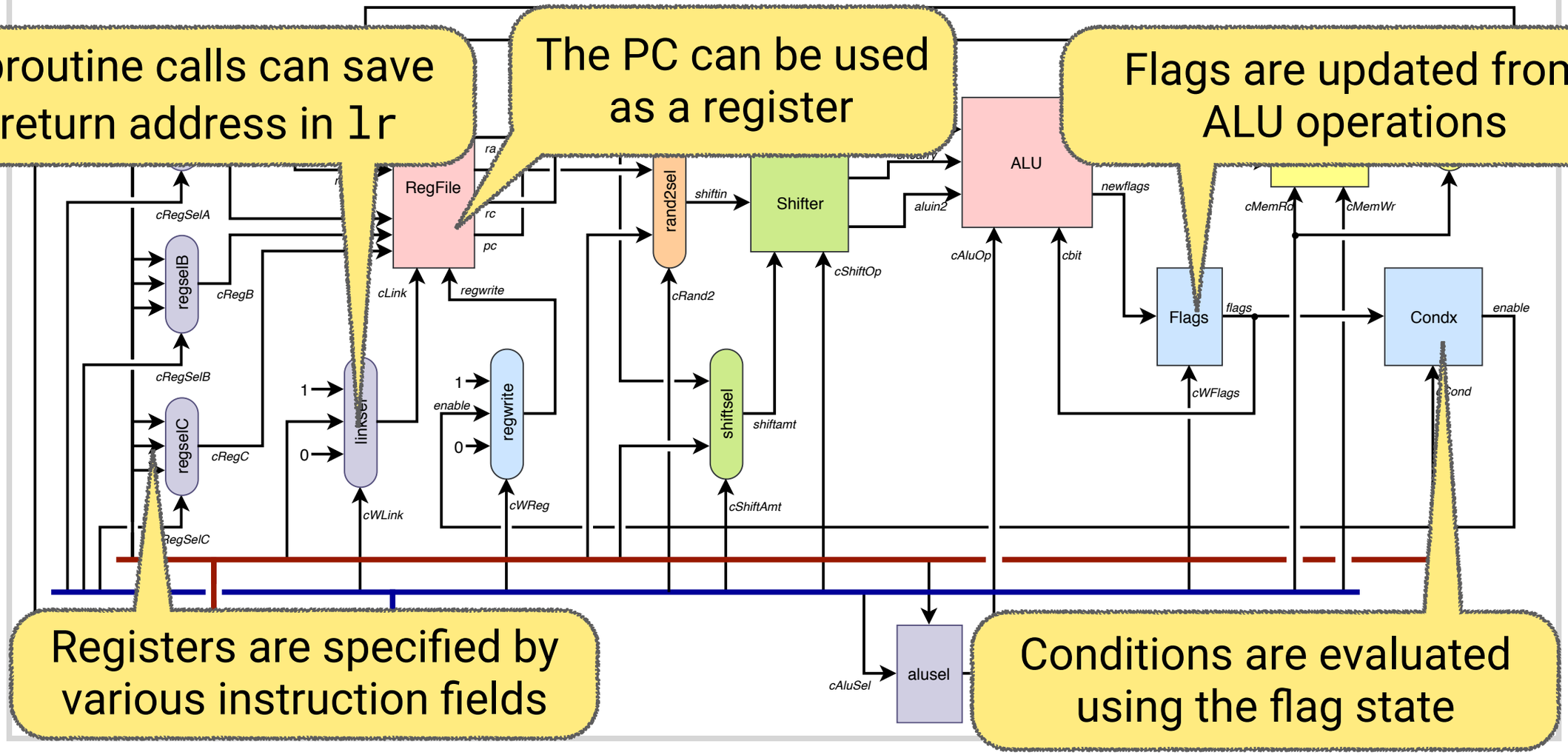
The complete datapath



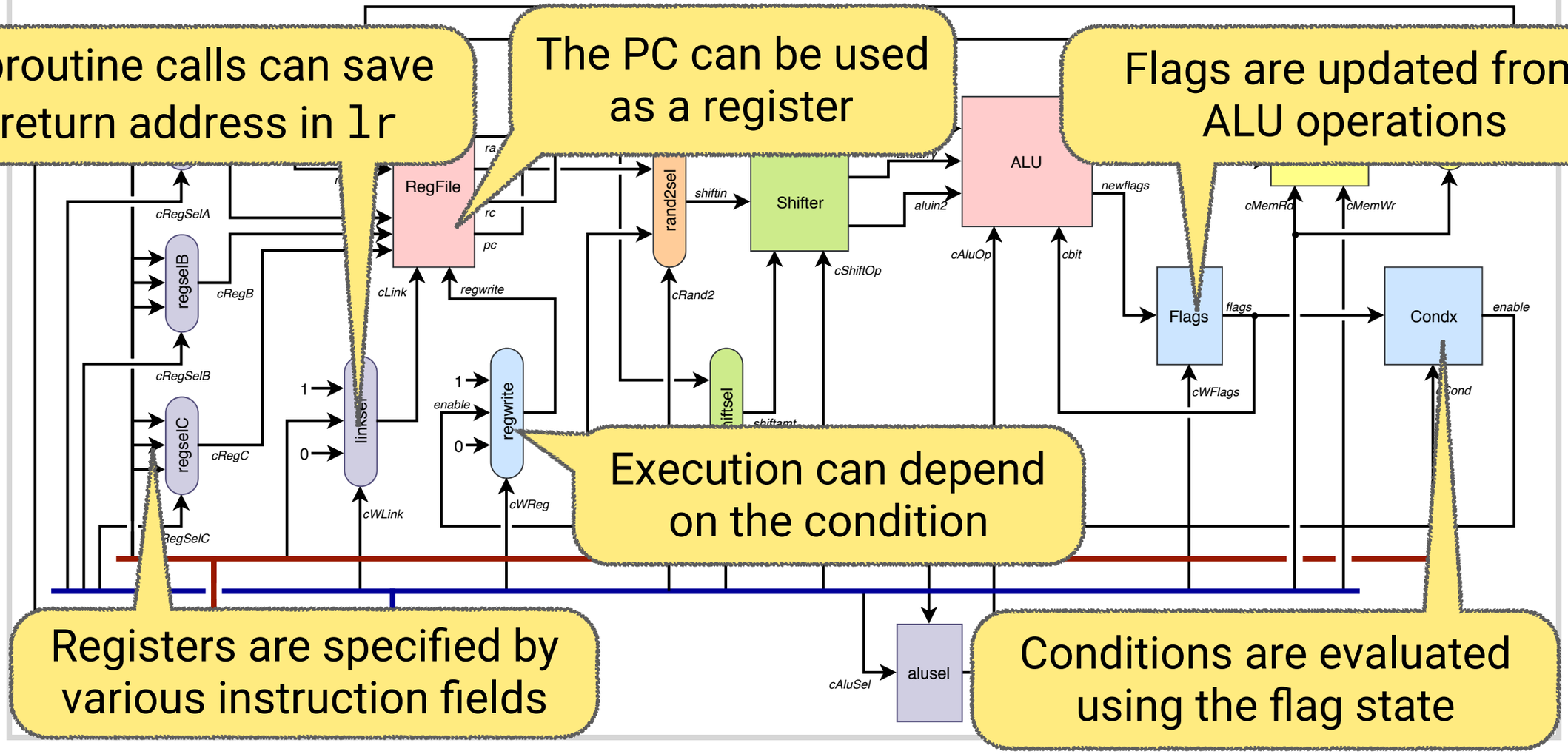
The complete datapath



The complete datapath



The complete datapath



A simulator

"Lab 5" consists of a simulator following the design given in these lectures.

It can load and execute programs prepared with the ARM assembler and linker, provided they use no instructions that we haven't implemented.

A document on the wiki presents the program and explanation in 'literate' form.