RISC Design:
DLX – HFC Method

Virendra Singh

Associate Professor

Computer **A**rchitecture and **D**ependable **S**ystems **L**ab

Department of Electrical Engineering

Indian Institute of Technology Bombay

http://www.ee.iitb.ac.in/~viren/

E-mail: viren@ee.iitb.ac.in

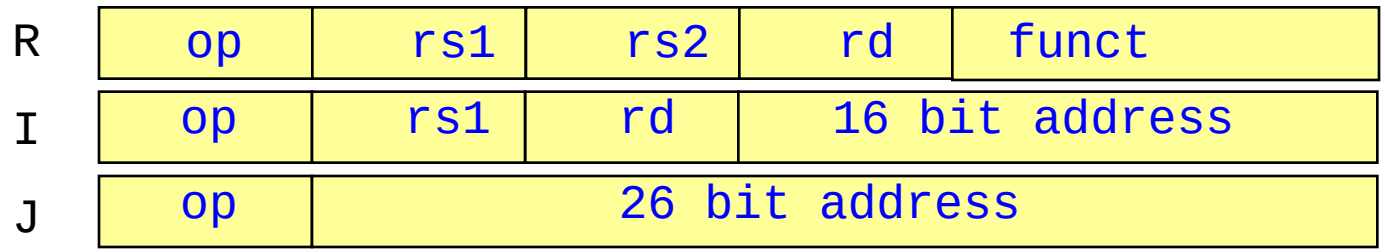
EE-309: Microprocessors

Lecture

CADSL

Overview of DLX

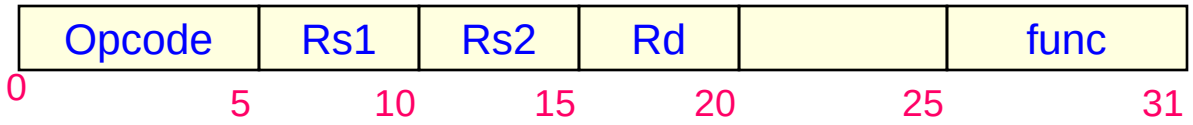
- simple instructions, all 32 bits wide
- very structured, no unnecessary baggage
- only three instruction formats



- rely on compiler to achieve performance

Instruction Set

Register-Register Instructions



Arithmetic and Logical Instruction

- ADD Rd, Rs1, Rs2 Regs[Rd] <= Reg[Rs1] + Reg[Rs2]
- SUB Rd, Rs1, Rs2 Regs[Rd] <= Reg[Rs1] - Reg[Rs2]
- AND Rd, Rs1, Rs2 Regs[Rd] <= Reg[Rs1] and Reg[Rs2]
- OR Rd, Rs1, Rs2 Regs[Rd] <= Reg[Rs1] or Reg[Rs2]
- XOR Rd, Rs1, Rs2 Regs[Rd] <= Reg[Rs1] xor Reg[Rs2]
- SUB Rd, Rs1, Rs2 Regs[Rd] <= Reg[Rs1]-Reg[Rs2]

DLX Instruction Set

ADD Rd, Rs1, Rs2	Rd ← Rs1 + Rs2 (overflow – exception)	R	000_000 000_100
SUB Rd, Rs1, Rs2	Rd ← Rs1 - Rs2 (overflow – exception)	R	000_000 000_110
AND Rd, Rs1, Rs2	Rd ← Rs1 and Rs2	R	000_000/ 001_000
OR Rd, Rs1, Rs2	Rd ← Rs1 or Rs2	R	000_000/ 001_001
XOR Rd, Rs1, Rs2	Rd ← Rs1 xor Rs2	R	000_000/ 001_010
SLL Rd, Rs1, Rs2	Rd ← Rs1 << Rs2 (logical) (5 lsb of Rs2 are significant)	R	000_000 001_100
SRL Rd, Rs1, Rs2	Rd ← Rs1 >> Rs2 (logical) (5 lsb of Rs2 are significant)	R	000_000 001_110
SRA Rd, Rs1, Rs2	Rd ← Rs1 >> Rs2 (arithmetic) (5 lsb of Rs2 are significant)	R	000_000 001_111

DLX Instruction Set

ADDI Rd, Rs1, Imm	Rd ← Rs1 + Imm (sign extended) (overflow – exception)	I	010_100
SUBI Rd, Rs1, Imm	Rd ← Rs1 – Imm (sign extended) (overflow – exception)	I	010_110
ANDI Rd, Rs1, Imm	Rd ← Rs1 and Imm (zero extended)	I	011_000
ORI Rd, Rs1, Imm	Rd ← Rs1 or Imm(zero extended)	I	011_001
XORI Rd, Rs1, Imm	Rd ← Rs1 xor Imm(zero extended)	I	011_010
SLLI Rd, Rs1, Imm	Rd ← Rs1 << Imm (logical) (5 lsb of Imm are significant)	I	011_100
SRLI Rd, Rs1, Imm	Rd ← Rs1 >> Imm (logical) (5 lsb of Imm are significant)	I	011_110
SRAI Rd, Rs1, Imm	Rd ← Rs1 >> Imm (arithmetic) (5 lsb of Imm are significant)	I	011_111

DLX Instruction Set

LHI Rd, Imm	Rd(0:15) ← Imm Rd(16:32) ← hex0000 (Imm: 16 bit immediate)	I	011_011
NOP	Do nothing	R	000_000 000_000

DLX Instruction Set

SEQ Rd, Rs1, Rs2	Rs1 = Rs2: Rd ← hex0000_0001 else: Rd ← hex0000_0000	R	000_000 010_000
SNE Rd, Rs1, Rs2	Rs1 /= Rs2: Rd ← hex0000_0001 else: Rd ← hex0000_0000	R	000_000 010_010
SLT Rd, Rs1, Rs2	Rs1 < Rs2: Rd ← hex0000_0001 else: Rd ← hex0000_0000	R	000_000 010_100
SLE Rd, Rs1, Rs2	Rs1 <= Rs2: Rd ← hex0000_0001 else: Rd ← hex0000_0000	R	000_000 010_110
SGT Rd, Rs1, Rs2	Rs1 > Rs2: Rd ← hex0000_0001 else: Rd ← hex0000_0000	R	000_000 011_000
SGE Rd, Rs1, Rs2	Rs1 >= Rs2: Rd ← hex0000_0001 else: Rd ← hex0000_0000	R	000_000 011_010

DLX Instruction Set

SEQI Rd, Rs1, Imm	Rs1 = Imm : Rd ← hex0000_0001 else: Rd ← hex0000_0000 (Imm: Sign extended 16 bit immediate)	I	100_000
SNEI Rd, Rs1, Imm	Rs1 /= Imm : Rd ← hex0000_0001 else: Rd ← hex0000_0000	I	100_010
SLTI Rd, Rs1, Imm	Rs1 < Imm : Rd ← hex0000_0001 else: Rd ← hex0000_0000	I	100_100
SLEI Rd, Rs1, Imm	Rs1 <= Imm : Rd ← hex0000_0001 else: Rd ← hex0000_0000	I	100_110
SGTI Rd, Rs1, Imm	Rs1 > Imm : Rd ← hex0000_0001 else: Rd ← hex0000_0000	I	101_000
SGEI Rd, Rs1, Imm	Rs1 >= Imm : Rd ← hex0000_0001 else: Rd ← hex0000_0000	I	101_010

DLX Instruction Set

BEQZ Rs, Label	Rs = 0: PC ← PC+4+Label Rs /= 0: PC ← PC+4 (Label: Sign extended16 bit immediate)	I	010_000
BNEZ Rs, Label	Rs /= 0: PC ← PC+4+Label Rs = 0: PC ← PC+4	I	010_001
J Label	PC ← PC + 4 + sign_extd(imm26)	J	001_100
JAL Label	R31 ← PC + 4 PC ← PC+ 4 + sign_extd(imm26)	J	001_100
JAL Label	R31 ← PC + 4 PC ← PC+ 4 + sign_extd(imm26)	J	001_101
JR Rs	PC ← Rs	I	001_110
JALR Rs	R31 ← PC + 4 PC ← Rs	I	001_111

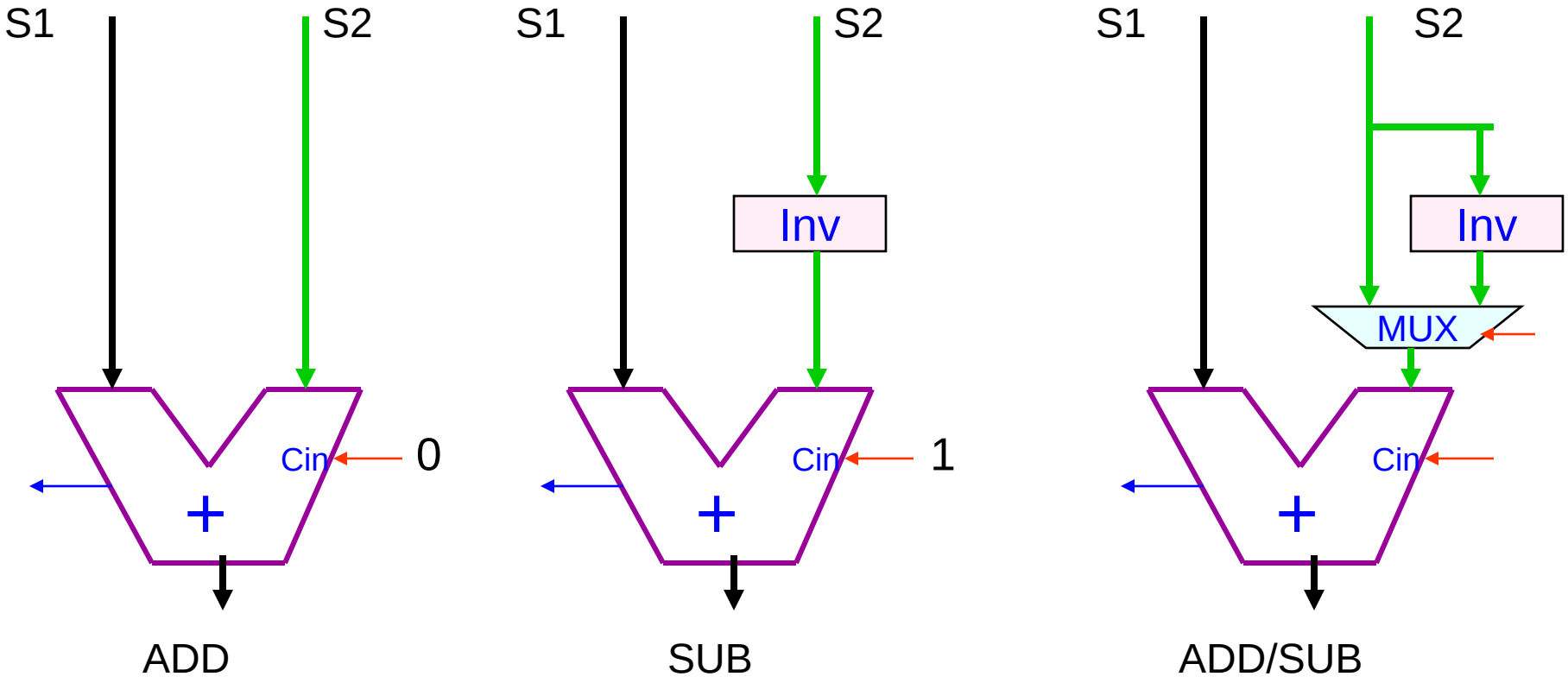
DLX Instruction Set

LW Rd, Rs2 (Rs1)	Rd ← M(Rs1 + Rs2) (word aligned address)	R	000_000 100_000
SW Rs2(Rs1), Rd	M(Rs1 + Rs2) ← Rd	R	000_000 101_000
LH Rd, Rs2 (Rs1)	Rd (16:31)← M(Rs1 + Rs2) (Rd sign extended to 32 bit)	R	000_000 100_001
SH Rs2(Rs1), Rd	M(Rs1 + Rs2) ← Rd(16:31)	R	000_000 101_001
LB Rd, Rs2 (Rs1)	Rd (24:31)← M(Rs1 + Rs2) (Rd sign extended to 32 bit)	R	000_000 101_010
SB Rs2(Rs1), Rd	M(Rs1 + Rs2) ← Rd(24:31)	R	000_000 101_010

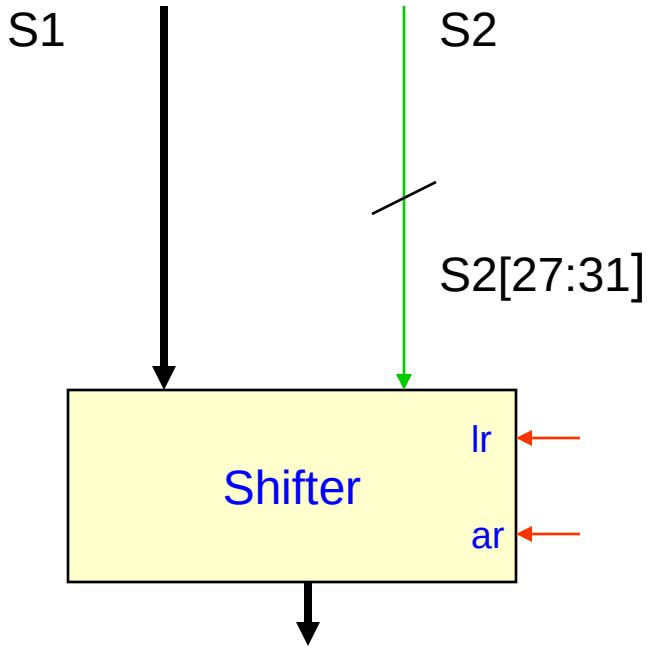
DLX Instruction Set

LWI Rd, Imm (Rs)	Rd ← M(Rs + Imm) (Imm: sign extended 16 bit) (word aligned address)	I	000_100
SWI Imm(Rs), Rd	M(Rs + Imm) ← Rd	I	001_000
LHI Rd, Imm (Rs)	Rd (16:31)← M(Rs + Imm) (Rd sign extended to 32 bit)	I	000_101
SHI Imm(Rs), Rd	M(Rs1 + Rs2) ← Rd(16:31)	I	001_001
LBI Rd, Imm (Rs)	Rd (24:31)← M(Rs + Imm) (Rd sign extended to 32 bit)	I	000_110
SBI Imm(Rs), Rd	M(Rs + Imm) ← Rd(24:31)	I	001_010

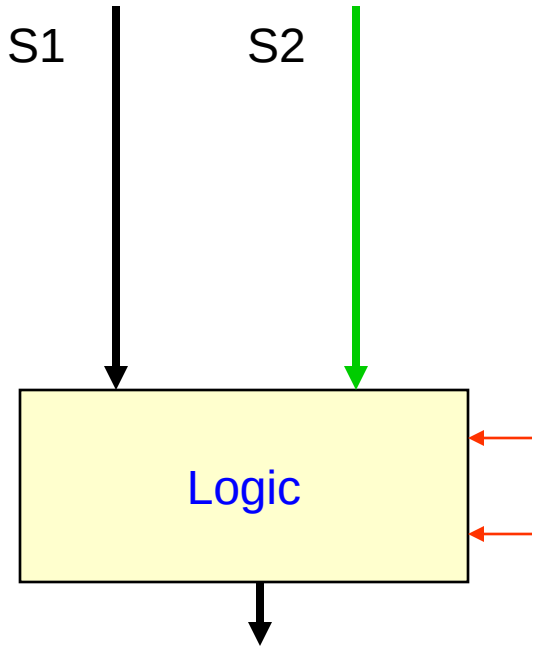
Arithmetic Logic Unit

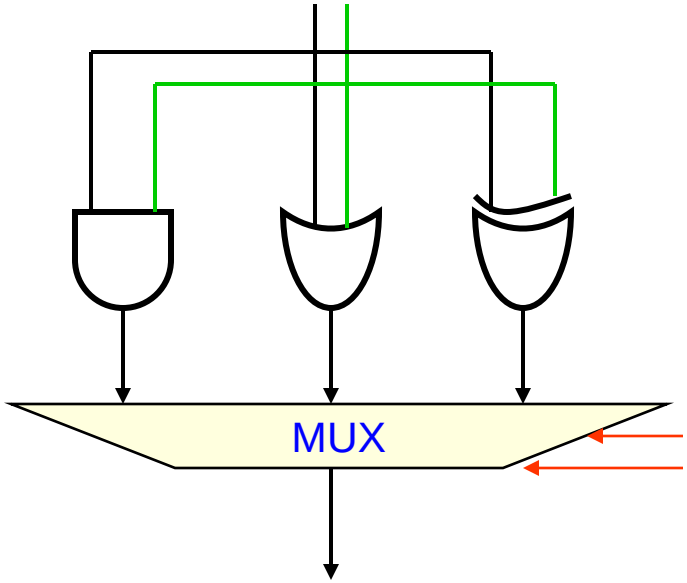


Arithmetic Logic Unit

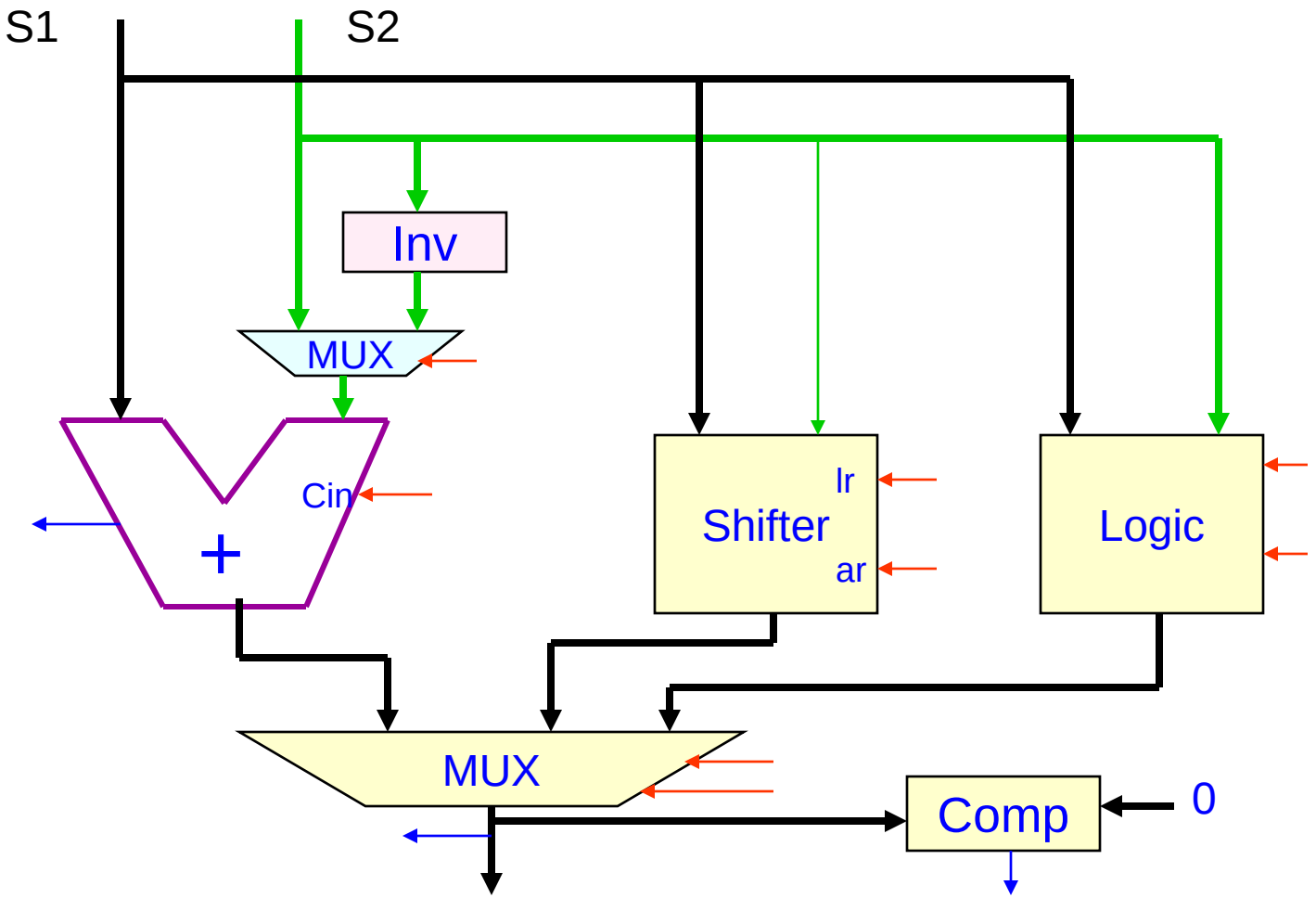


Arithmetic Logic Unit

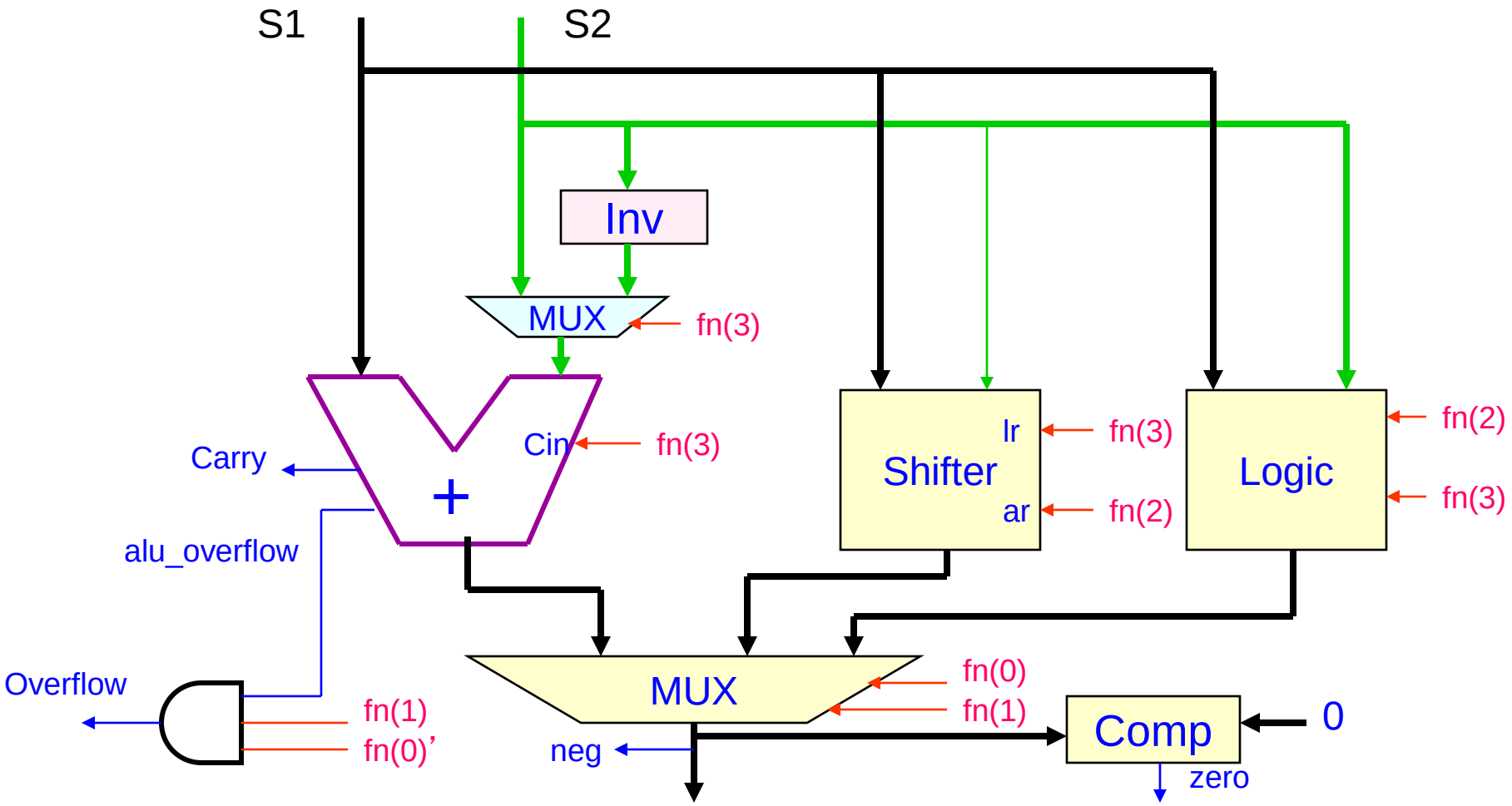




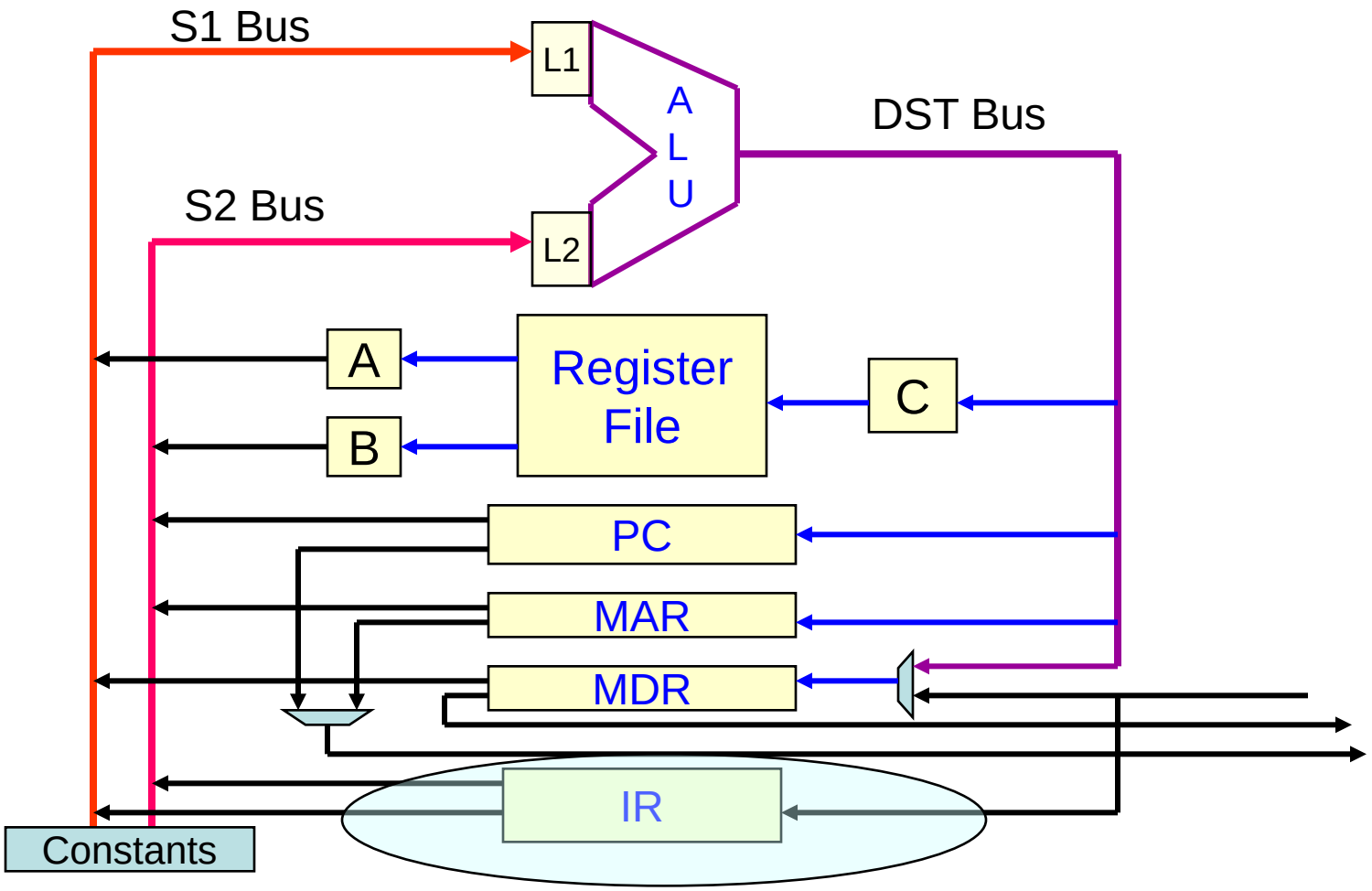
Arithmetic Logic Unit



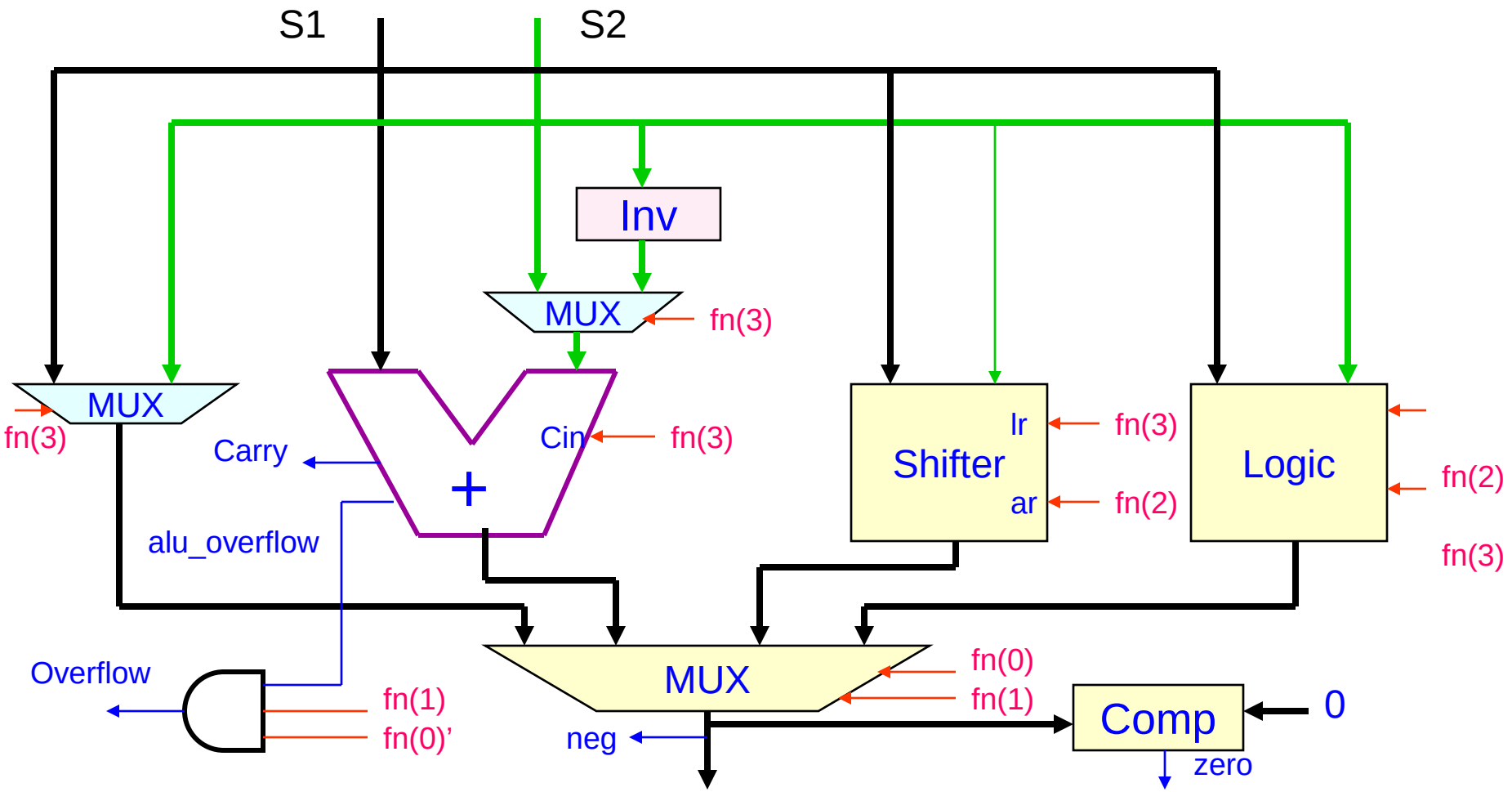
Arithmetic Logic Unit



Datapath of DLX



Arithmetic Logic Unit



Hardware Flowchart

ADD Rd, Rs1, Rs2

rf → a rf → b	pc → eab edb → ir
a → s1 → alu b → s2 → alu alu → dst → c	pc → s2 → alu +4 → s1 → alu alu → dst → pc
c → rf	

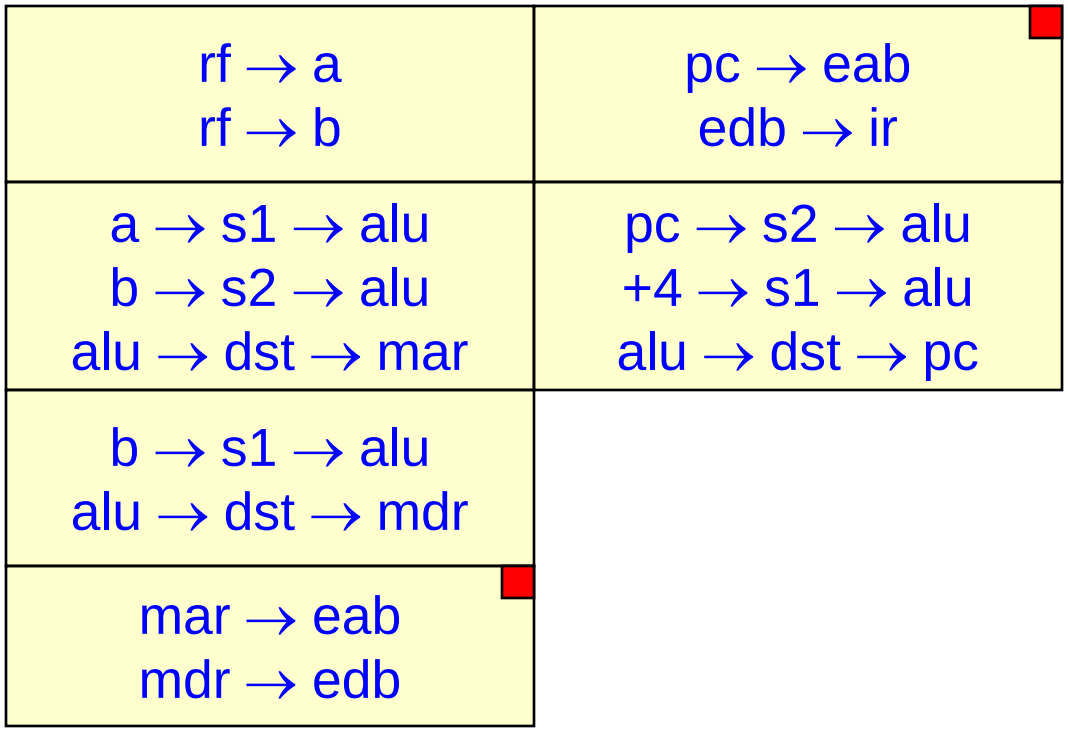
Hardware Flowchart

LW Rd, Rs2(Rs1)

rf → a rf → b	pc → eab edb → ir
a → s1 → alu b → s2 → alu alu → dst → mar	pc → s2 → alu +4 → s1 → alu alu → dst → pc
mar → eab edb → mdr	
mdr → s1 → alu alu → c	
c → rf	

Hardware Flowchart

SW Rd, Rs2(Rs1)



Hardware Flowchart

JAL Label

	pc → eab edb → ir
pc → s2 → alu alu → dst → c	pc → s2 → alu +4 → s1 → alu alu → dst → pc
ir → s1 → alu pc → s2 → alu alu → dst → mar pc → rf	
mar → s2 → alu alu → dst → pc	

Level 1 Hardware Flowchart

ADD Rd, Rs1, Rs2

rf → a rf → b	pc → eab edb → ir
a → s1 → alu b → s2 → alu alu → dst → c	pc → s2 → alu +4 → s1 → alu alu → dst → pc
c → rf	

ADD.I Rd, Rs1, Imm

rf → a	pc → eab edb → ir
a → s1 → alu ir → s2 → alu alu → dst → c	pc → s2 → alu +4 → s1 → alu alu → dst → pc
c → rf	

Level 1 Hardware Flowchart

SUB Rd, Rs1, Rs2

rf → a rf → b	pc → eab edb → ir
a → s1 → alu b → s2 → alu alu → dst → c	pc → s2 → alu +4 → s1 → alu alu → dst → pc
c → rf	

SUB.I Rd, Rs1, Imm

rf → a	pc → eab edb → ir
a → s1 → alu ir → s2 → alu alu → dst → c	pc → s2 → alu +4 → s1 → alu alu → dst → pc
c → rf	

Level 1 Hardware Flowchart

OR Rd, Rs1, Rs2

rf → a rf → b	pc → eab edb → ir
a → s1 → alu b → s2 → alu alu → dst → c	pc → s2 → alu +4 → s1 → alu alu → dst → pc
c → rf	

OR.I Rd, Rs1, Imm

rf → a	pc → eab edb → ir
a → s1 → alu ir → s2 → alu alu → dst → c	pc → s2 → alu +4 → s1 → alu alu → dst → pc
c → rf	

Level 1 Hardware Flowchart

AND Rd, Rs1, Rs2

rf → a rf → b	pc → eab edb → ir
a → s1 → alu b → s2 → alu alu → dst → c	pc → s2 → alu +4 → s1 → alu alu → dst → pc
c → rf	

AND.I Rd, Rs1, Imm

rf → a	pc → eab edb → ir
a → s1 → alu ir → s2 → alu alu → dst → c	pc → s2 → alu +4 → s1 → alu alu → dst → pc
c → rf	

Level 1 Hardware Flowchart

XOR Rd, Rs1, Rs2

rf → a rf → b	pc → eab edb → ir
a → s1 → alu b → s2 → alu alu → dst → c	pc → s2 → alu +4 → s1 → alu alu → dst → pc
c → rf	

XOR.I Rd, Rs1, Imm

rf → a	pc → eab edb → ir
a → s1 → alu ir → s2 → alu alu → dst → c	pc → s2 → alu +4 → s1 → alu alu → dst → pc
c → rf	

Level 1 Hardware Flowchart

SLL Rd, Rs1, Rs2

rf → a rf → b	pc → eab edb → ir
a → s1 → alu b → s2 → alu alu → dst → c	pc → s2 → alu +4 → s1 → alu alu → dst → pc
c → rf	

SLL.I Rd, Rs1, Imm

rf → a	pc → eab edb → ir
a → s1 → alu ir → s2 → alu alu → dst → c	pc → s2 → alu +4 → s1 → alu alu → dst → pc
c → rf	

Level 1 Hardware Flowchart

SRL Rd, Rs1, Rs2

rf → a rf → b	pc → eab edb → ir
a → s1 → alu b → s2 → alu alu → dst → c	pc → s2 → alu +4 → s1 → alu alu → dst → pc
c → rf	

SRL.I Rd, Rs1, Imm

rf → a	pc → eab edb → ir
a → s1 → alu ir → s2 → alu alu → dst → c	pc → s2 → alu +4 → s1 → alu alu → dst → pc
c → rf	

Level 1 Hardware Flowchart

SRA Rd, Rs1, Rs2

rf → a rf → b	pc → eab edb → ir
a → s1 → alu b → s2 → alu alu → dst → c	pc → s2 → alu +4 → s1 → alu alu → dst → pc
c → rf	

SRA.I Rd, Rs1, Imm

rf → a	pc → eab edb → ir
a → s1 → alu ir → s2 → alu alu → dst → c	pc → s2 → alu +4 → s1 → alu alu → dst → pc
c → rf	

Level 1 Hardware Flowchart

LHI Rd, Imm

Rd(0:15) <- Imm; Rd(16:31) <- hex0000

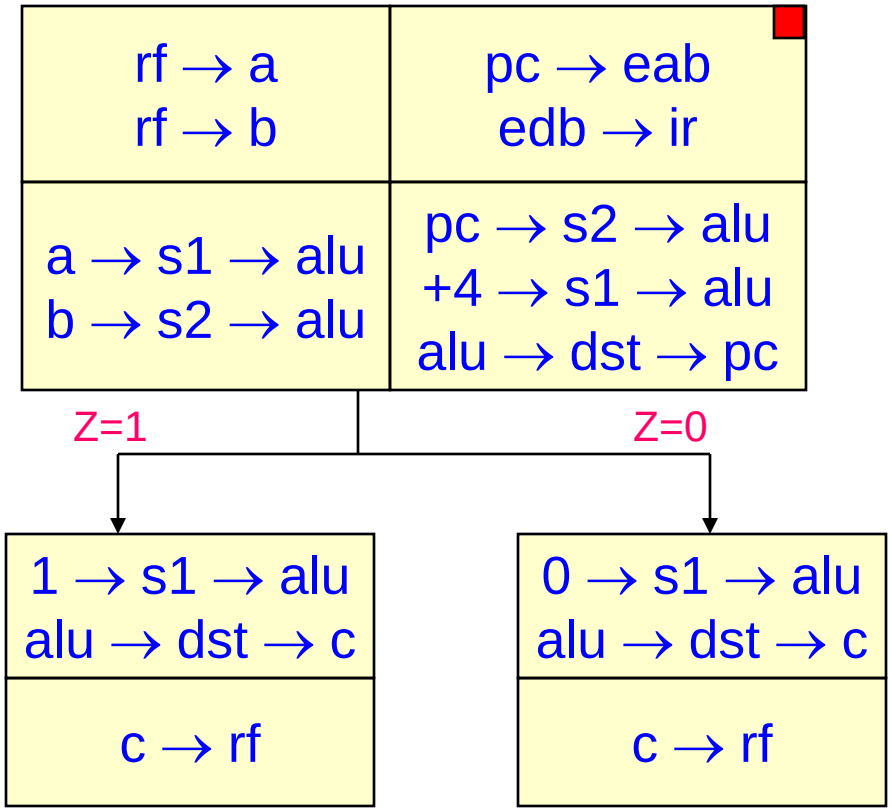
	pc → eab edb → ir
ir → s1 → alu 16 → s2 → alu alu → dst → c	pc → s2 → alu +4 → s1 → alu alu → dst → pc
c → rf	

NOP

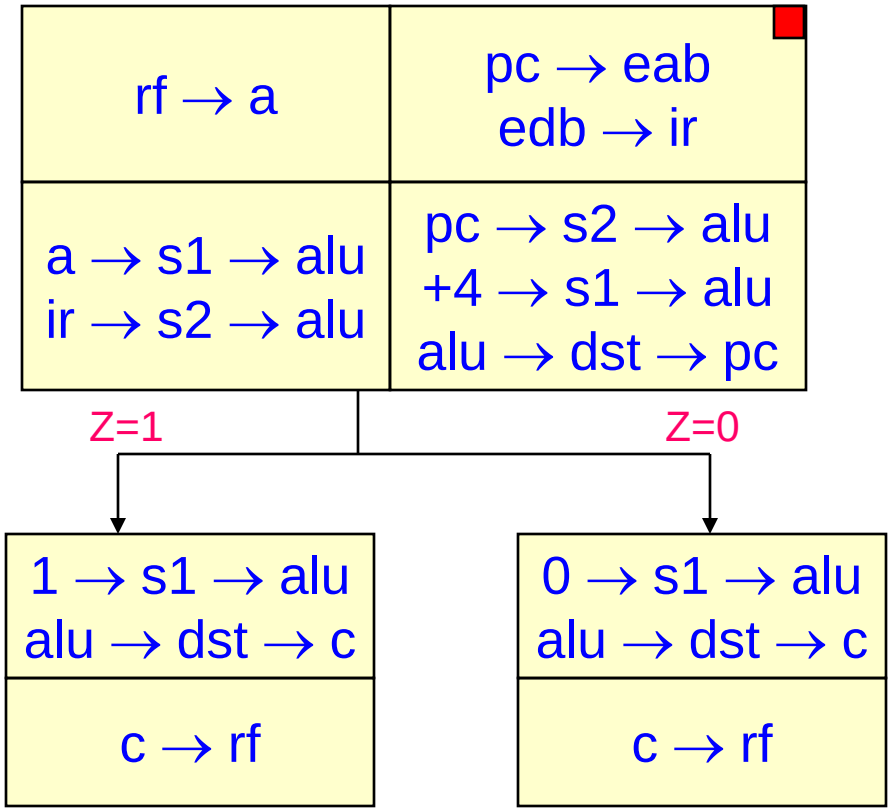
	pc → eab edb → ir
	pc → s2 → alu +4 → s1 → alu alu → dst → pc

Level 1 Hardware Flowchart

SEQ Rd, Rs1, Rs2

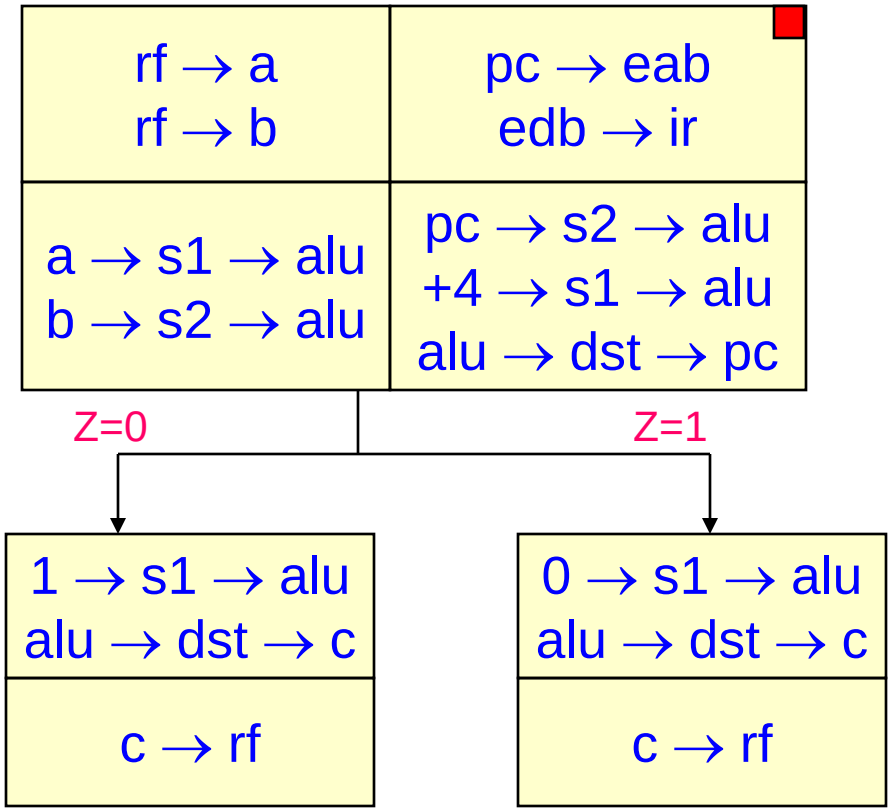


SEQ.I Rd, Rs1, Imm

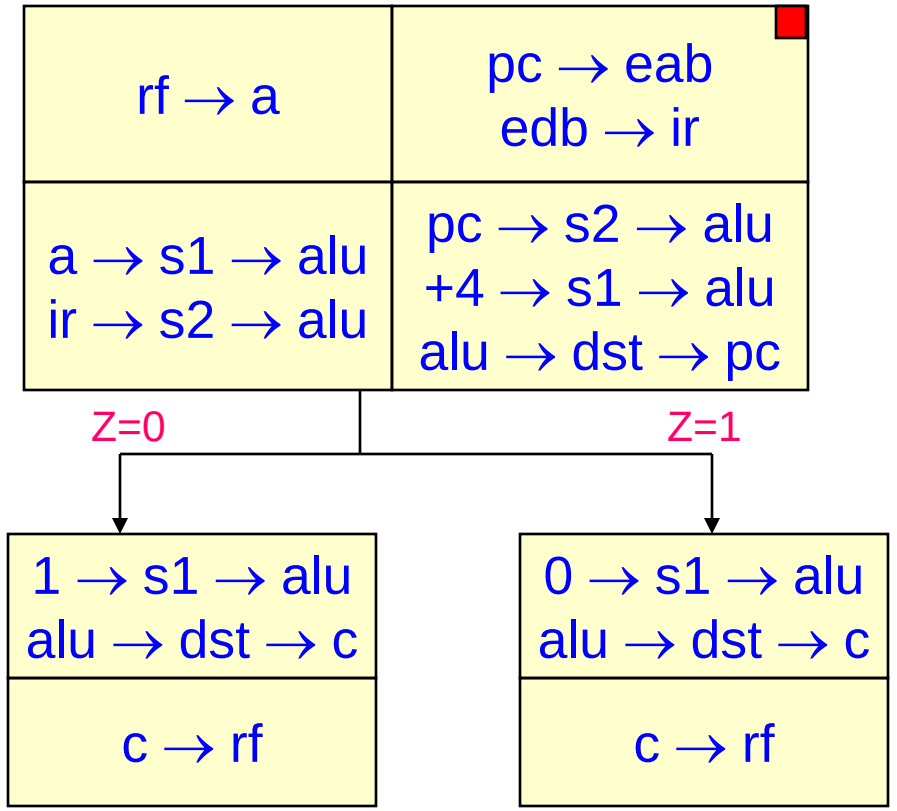


Level 1 Hardware Flowchart

SNE Rd, Rs1, Rs2

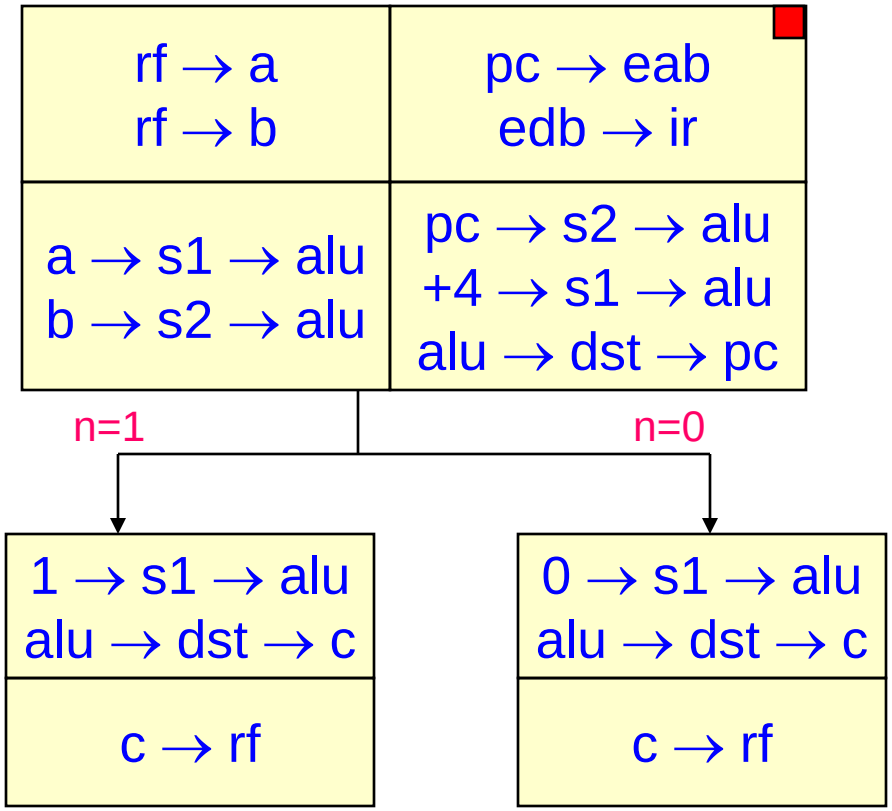


SNE.I Rd, Rs1, Imm

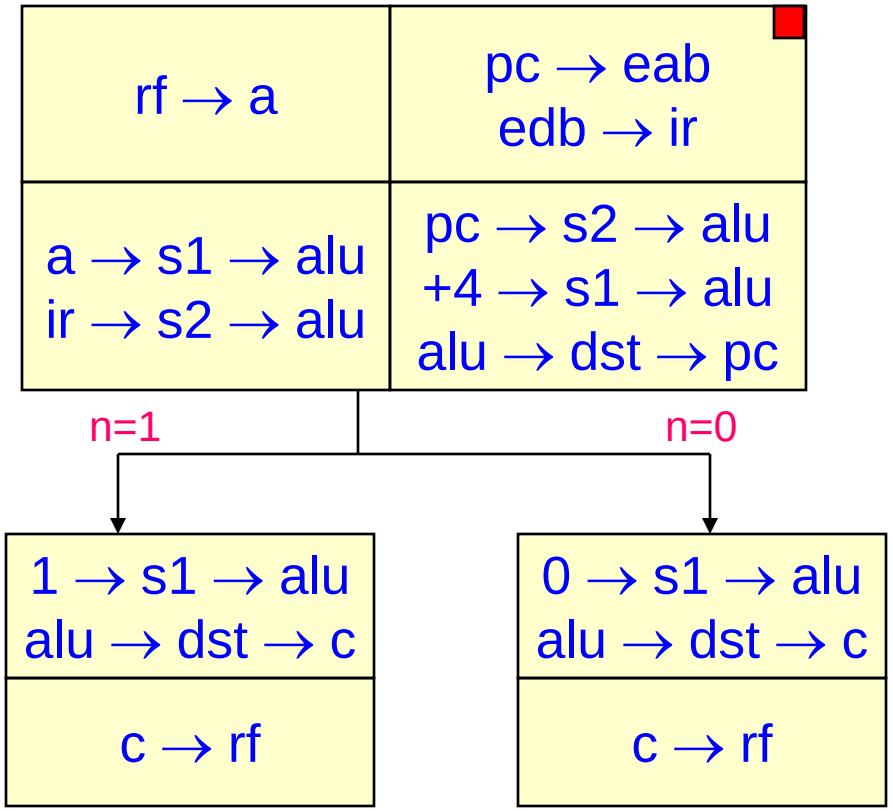


Level 1 Hardware Flowchart

SLT Rd, Rs1, Rs2

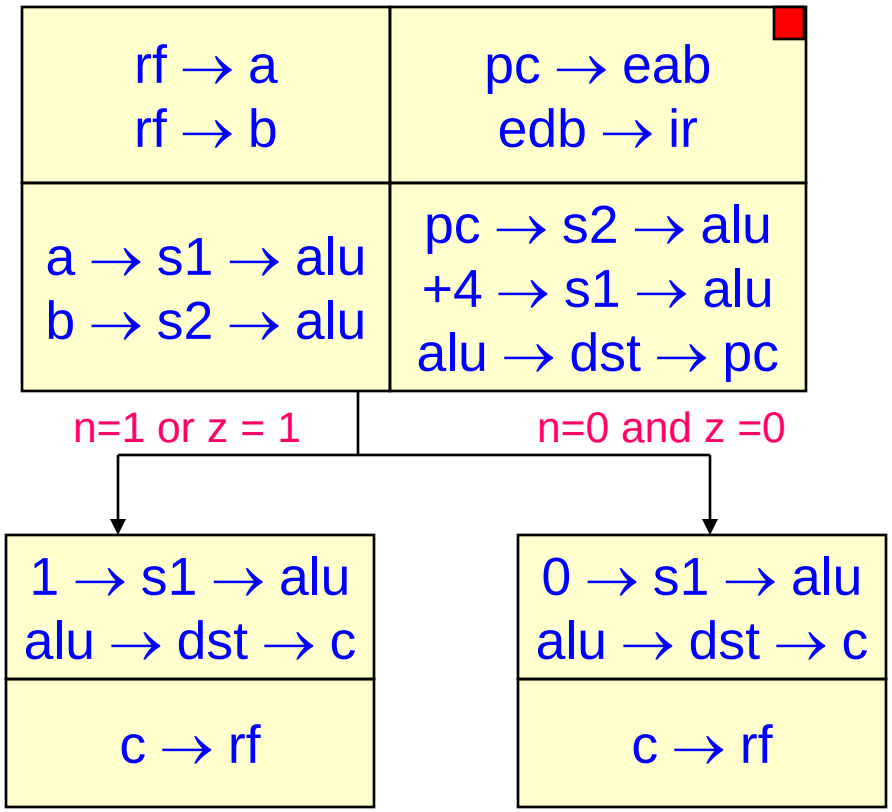


SLT.I Rd, Rs1, Imm

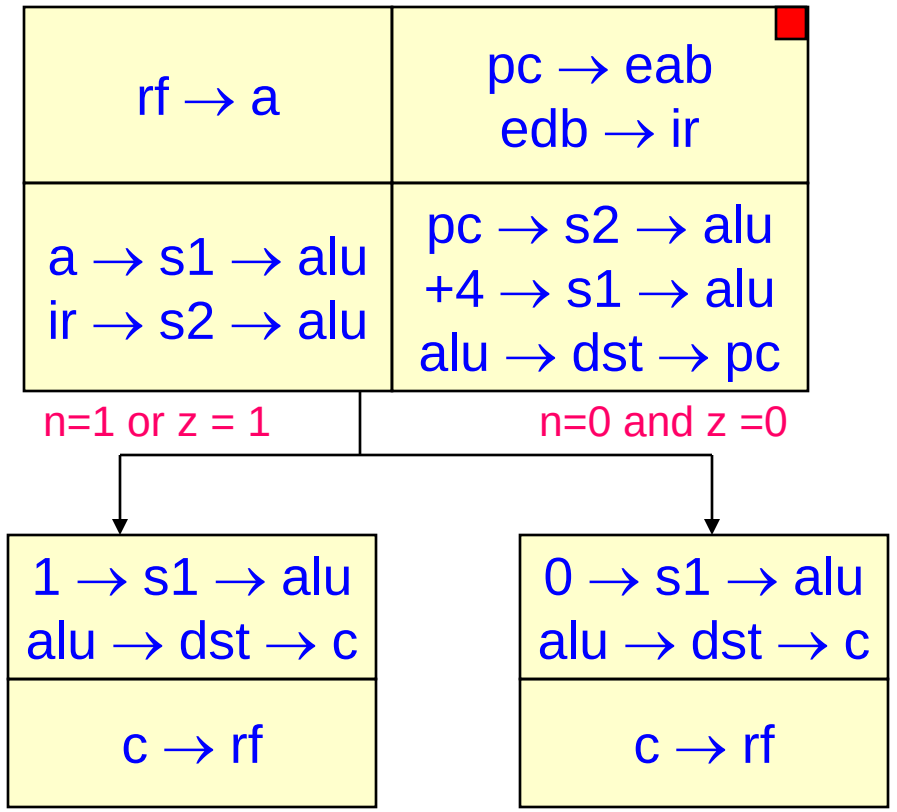


Level 1 Hardware Flowchart

SLE Rd, Rs1, Rs2

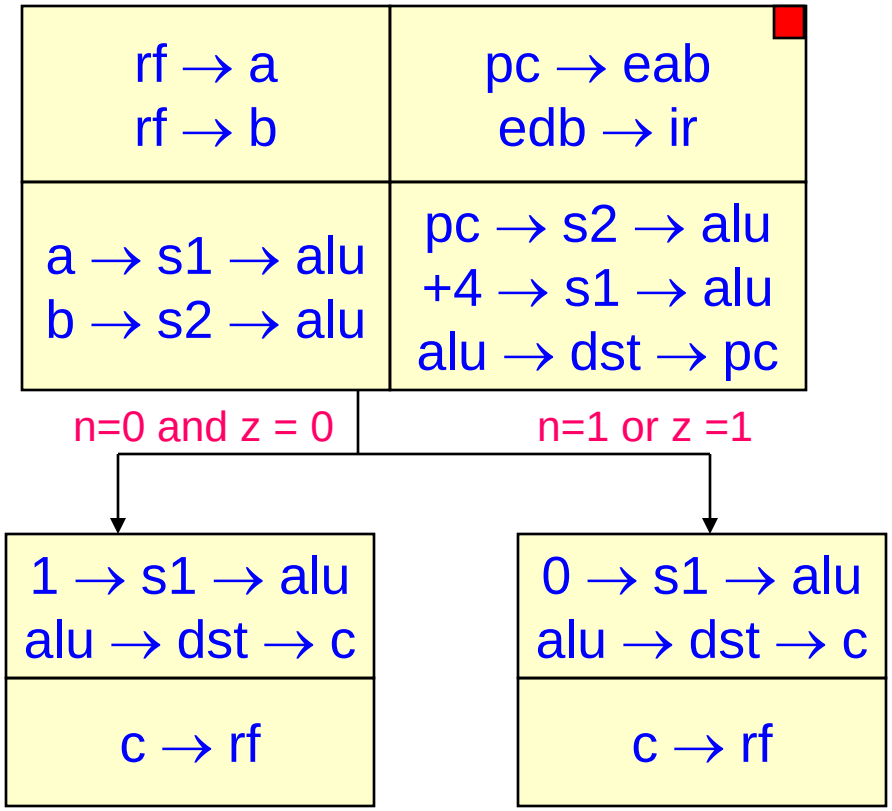


SLE.I Rd, Rs1, Imm

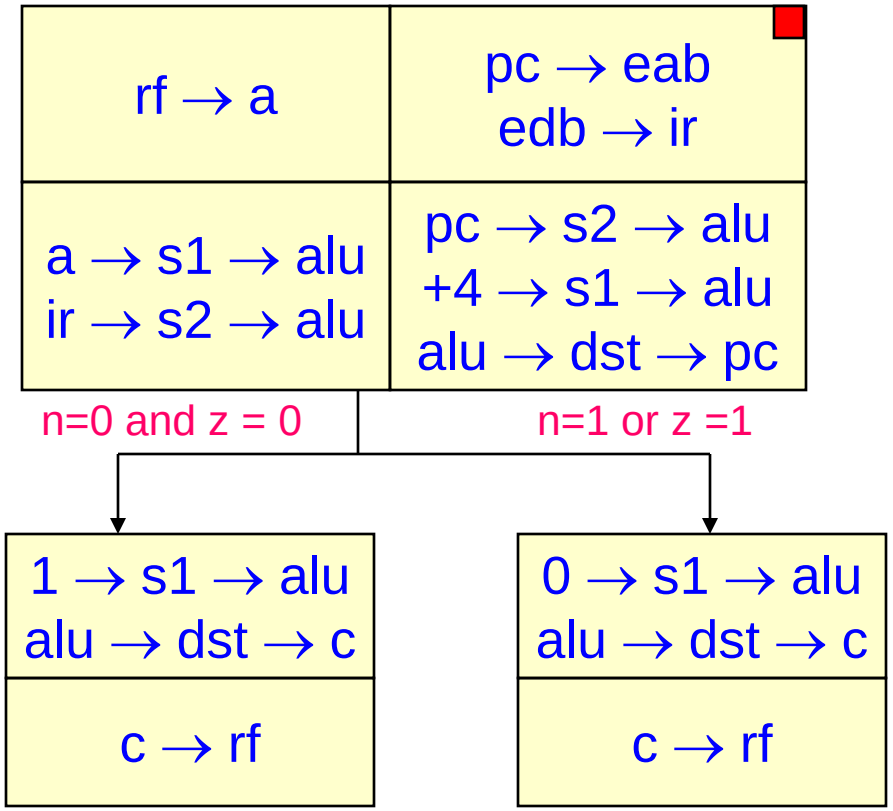


Level 1 Hardware Flowchart

SGT Rd, Rs1, Rs2

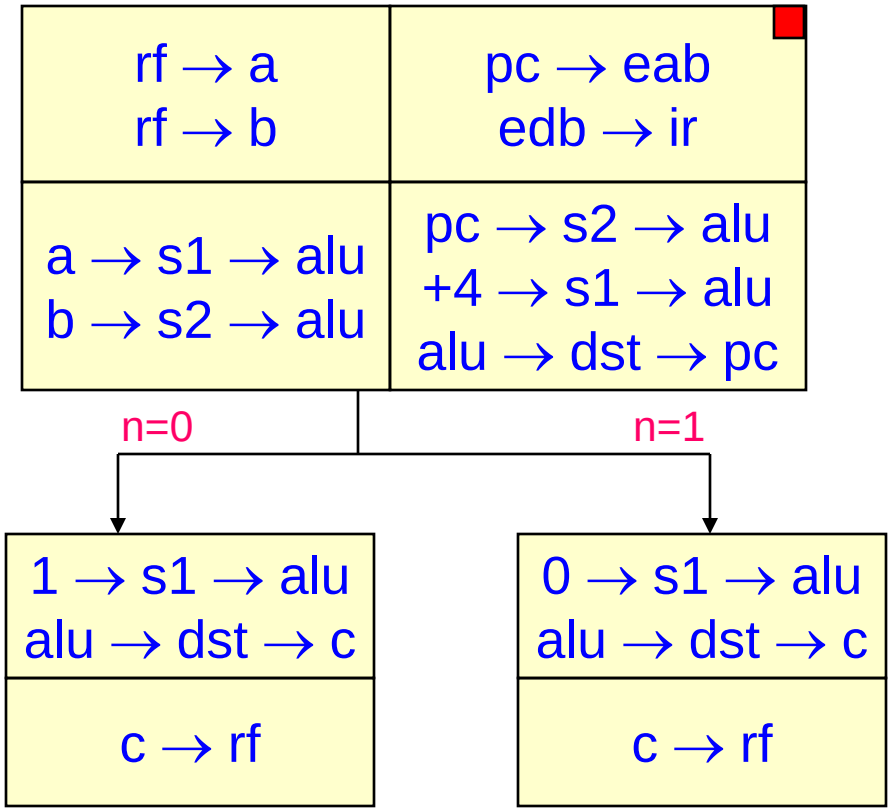


SGT.I Rd, Rs1, Imm

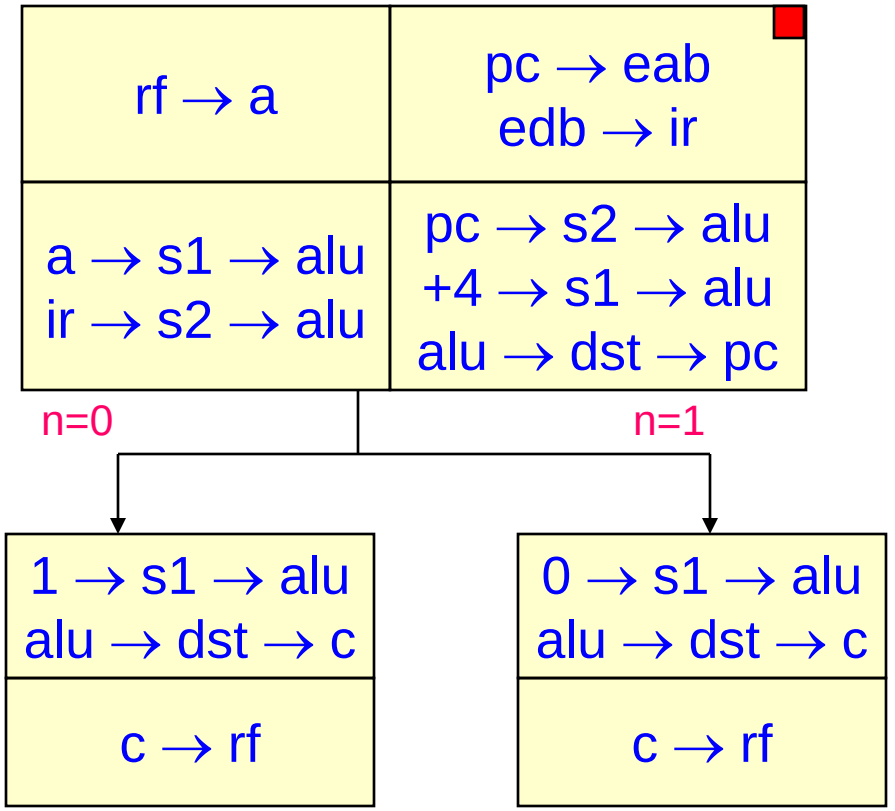


Level 1 Hardware Flowchart

SGE Rd, Rs1, Rs2

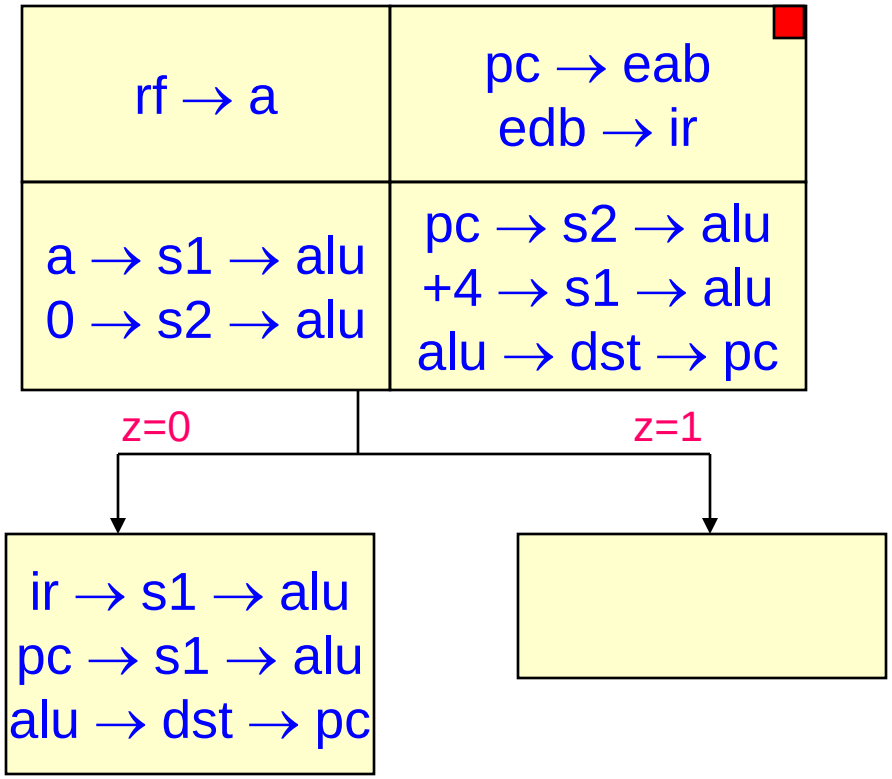


SGE.I Rd, Rs1, Imm

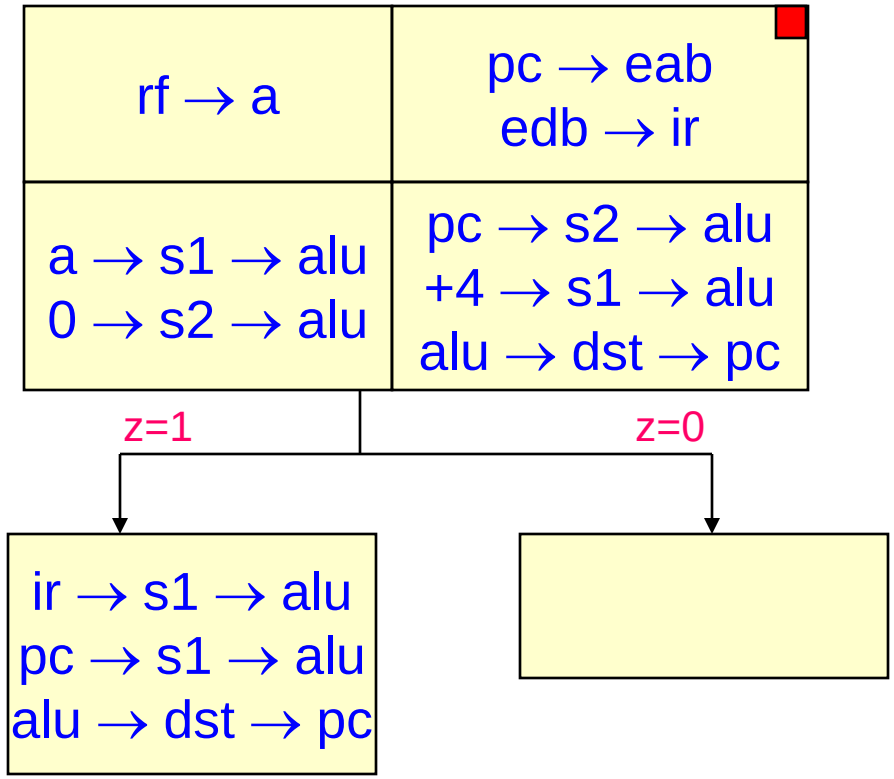


Level 1 Hardware Flowchart

BEQZ Rs, Label



BNEZ Rs, Label



Level 1 Hardware Flowchart

J Label

	pc → eab edb → ir
ir → s1 → alu pc → s2 → alu alu → dst → mar	pc → s2 → alu +4 → s1 → alu alu → dst → pc
mar → s2 → alu alu → dst → pc	

JR Rs

rf → a	pc → eab edb → ir
a → s1 → alu alu → dst → mar	pc → s2 → alu +4 → s1 → alu alu → dst → pc
mar → s2 → alu alu → dst → pc	

Level 1 Hardware Flowchart

JAL Label

	pc → eab edb → ir
pc → s2 → alu alu → dst → c	pc → s2 → alu +4 → s1 → alu alu → dst → pc
ir → s1 → alu pc → s2 → alu alu → dst → mar c → rf	
mar → s2 → alu alu → dst → pc	

JALR Rs

rf → a	pc → eab edb → ir
pc → s2 → alu alu → dst → c	pc → s2 → alu +4 → s1 → alu alu → dst → pc
a → s1 → alu alu → dst → mar c → rf	
mar → s2 → alu alu → dst → pc	

Level 1 Hardware Flowchart

LW Rd, Rs2(Rs1)

rf → a rf → b	pc → eab edb → ir
a → s1 → alu b → s2 → alu alu → dst → mar	pc → s2 → alu +4 → s1 → alu alu → dst → pc
mar → eab edb → mdr	
mdr → s1 → alu alu → dst → c	
c → rf	

LW.I Rd, Imm(Rs1)

rf → a	pc → eab edb → ir
a → s1 → alu ir → s2 → alu alu → dst → mar	pc → s2 → alu +4 → s1 → alu alu → dst → pc
mar → eab edb → mdr	
mdr → s1 → alu alu → dst → c	
c → rf	

Level 1 Hardware Flowchart

SW Rd, Rs2(Rs1)

rf → a rf → b	pc → eab edb → ir
a → s1 → alu b → s2 → alu alu → dst → mar rf → b	pc → s2 → alu +4 → s1 → alu alu → dst → pc
b → s2 → alu alu → dst → mdr	
mar → eab mdr → edb	

SW.I Rd, Imm(Rs1)

rf → a	pc → eab edb → ir
a → s1 → alu ir → s2 → alu alu → dst → mar rf → b	pc → s2 → alu +4 → s1 → alu alu → dst → pc
b → s2 → alu alu → dst → mdr	
mar → eab mdr → edb	

Level 1 Hardware Flowchart

LH Rd, Rs2(Rs1)

rf → a rf → b	pc → eab edb → ir
a → s1 → alu b → s2 → alu alu → dst → mar	pc → s2 → alu +4 → s1 → alu alu → dst → pc
mar → eab edb → mdr	
mdr → s1 → alu 16 → s2 → alu alu → dst → c	
c → rf	

LH.I Rd, Imm(Rs1)

rf → a	pc → eab edb → ir
a → s1 → alu ir → s2 → alu alu → dst → mar	pc → s2 → alu +4 → s1 → alu alu → dst → pc
mar → eab edb → mdr	
mdr → s1 → alu 16 → s1 → alu alu → dst → c	
c → rf	

Level 1 Hardware Flowchart

SH Rd, Rs2(Rs1)

rf → a rf → b	pc → eab edb → ir
a → s1 → alu b → s2 → alu alu → dst → mar rf → b	pc → s2 → alu +4 → s1 → alu alu → dst → pc
b → s2 → alu alu → dst → mdr	
mar → eab mdr → edb	

SH.I Rd, Imm(Rs1)

rf → a	pc → eab edb → ir
a → s1 → alu ir → s2 → alu alu → dst → mar rf → b	pc → s2 → alu +4 → s1 → alu alu → dst → pc
b → s2 → alu alu → dst → mdr	
mar → eab mdr → edb	

Level 1 Hardware Flowchart

LB Rd, Rs2(Rs1)

rf → a rf → b	pc → eab edb → ir
a → s1 → alu b → s2 → alu alu → dst → mar	pc → s2 → alu +4 → s1 → alu alu → dst → pc
mar → eab edb → mdr	
mdr → s1 → alu 24 → s2 → alu alu → dst → c	
c → rf	

LB.I Rd, Imm(Rs1)

rf → a	pc → eab edb → ir
a → s1 → alu ir → s2 → alu alu → dst → mar	pc → s2 → alu +4 → s1 → alu alu → dst → pc
mar → eab edb → mdr	
mdr → s1 → alu 24 → s1 → alu alu → dst → c	
c → rf	

Level 1 Hardware Flowchart

SB Rd, Rs2(Rs1)

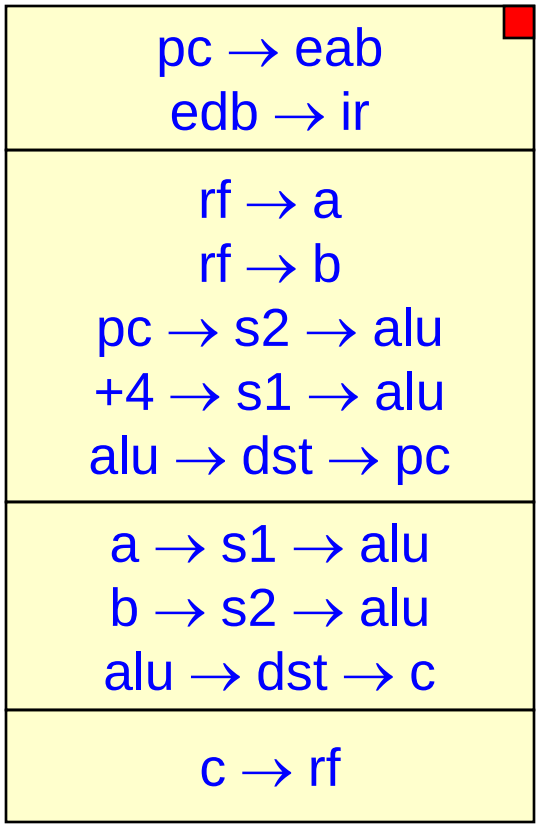
rf → a rf → b	pc → eab edb → ir
a → s1 → alu b → s2 → alu alu → dst → mar rf → b	pc → s2 → alu +4 → s1 → alu alu → dst → pc
b → s2 → alu alu → dst → mdr	
mar → eab mdr → edb	

SB.I Rd, Imm(Rs1)

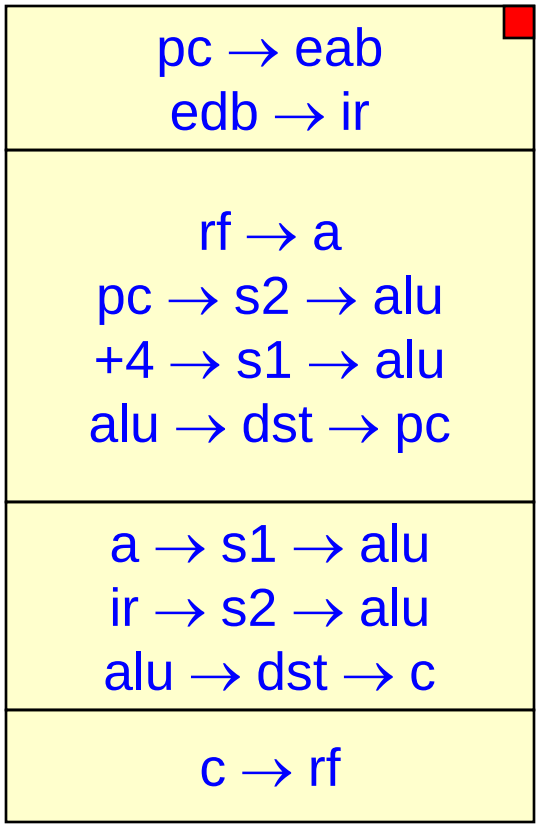
rf → a	pc → eab edb → ir
a → s1 → alu ir → s2 → alu alu → dst → mar rf → b	pc → s2 → alu +4 → s1 → alu alu → dst → pc
b → s2 → alu alu → dst → mdr	
mar → eab mdr → edb	

Merged Level 1 Hardware Flowchart

ADD Rd, Rs1, Rs2

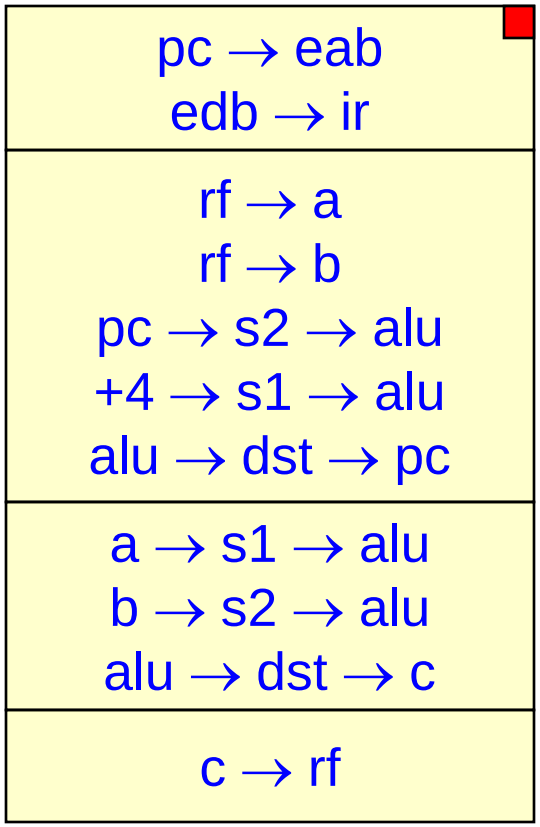


ADD.l Rd, Rs1, Imm

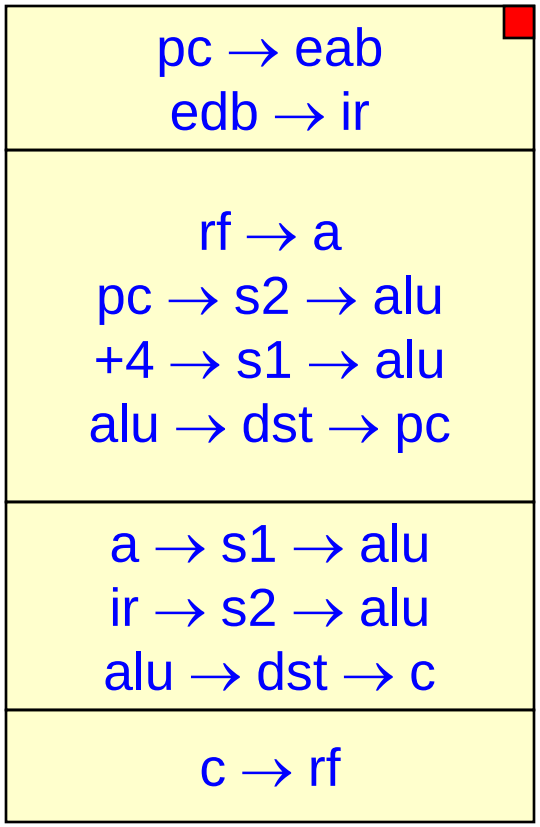


Merged Level 1 Hardware Flowchart

SUB Rd, Rs1, Rs2

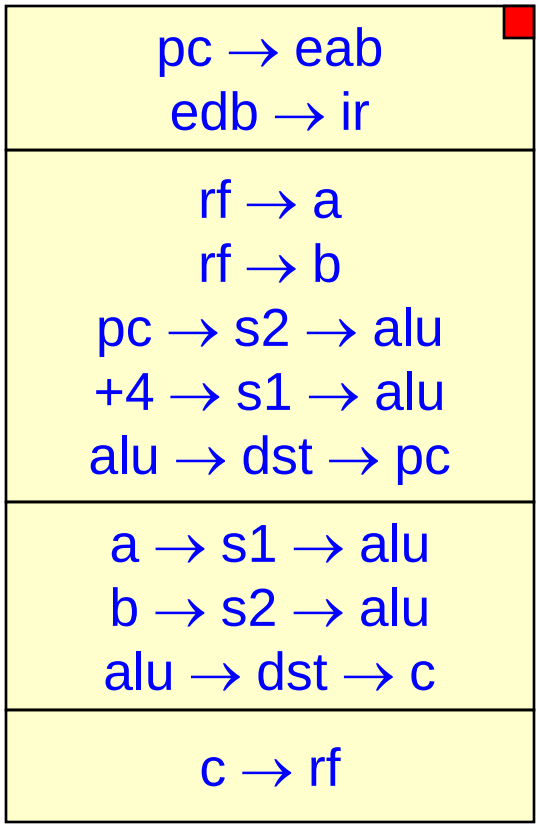


SUB.I Rd, Rs1, Imm

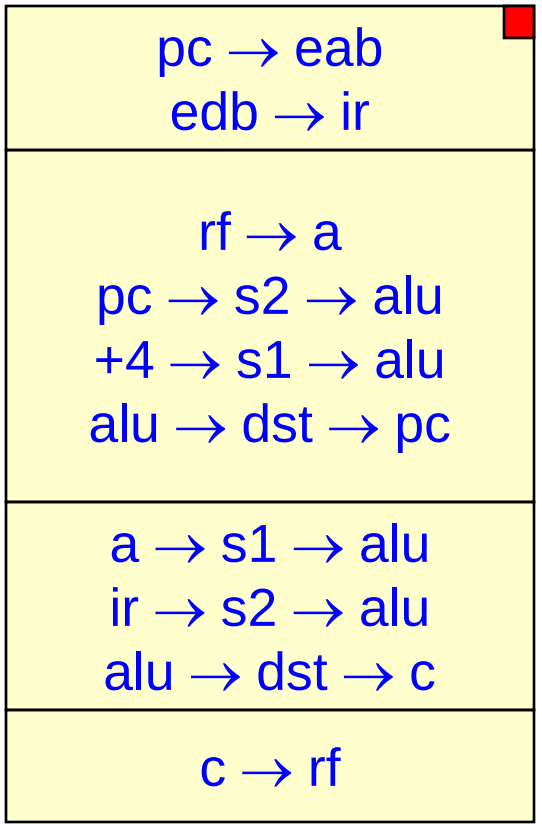


Merged Level 1 Hardware Flowchart

AND Rd, Rs1, Rs2

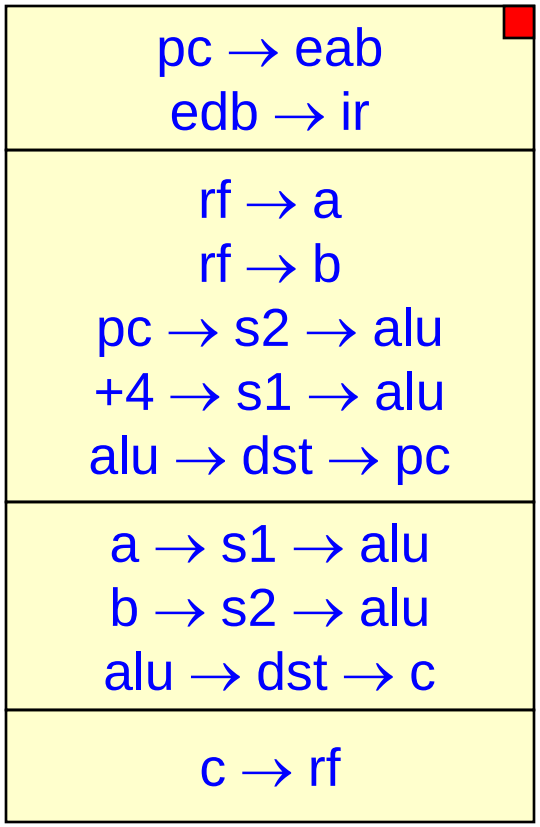


AND.I Rd, Rs1, Imm

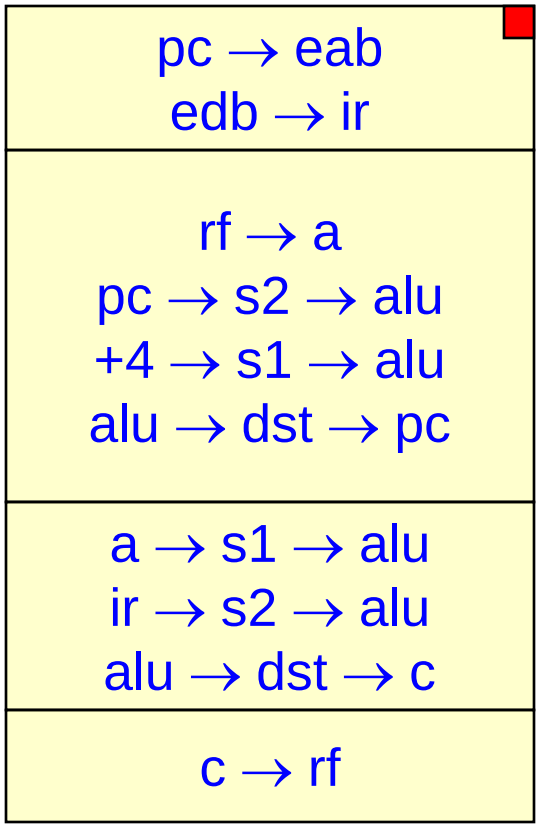


Merged Level 1 Hardware Flowchart

OR Rd, Rs1, Rs2

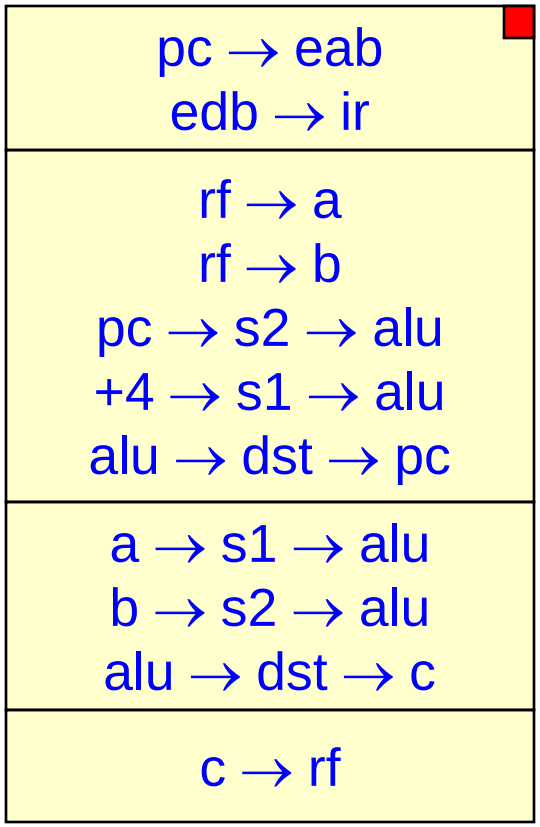


OR.I Rd, Rs1, Imm

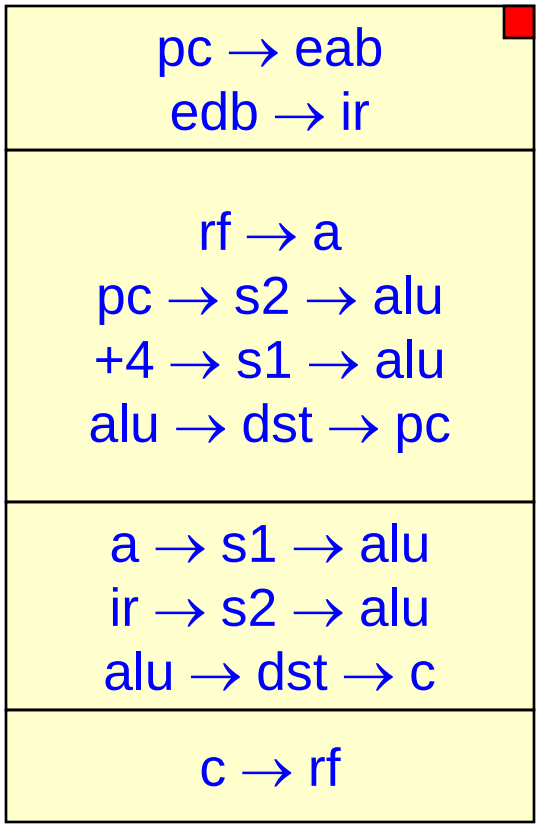


Merged Level 1 Hardware Flowchart

XOR Rd, Rs1, Rs2

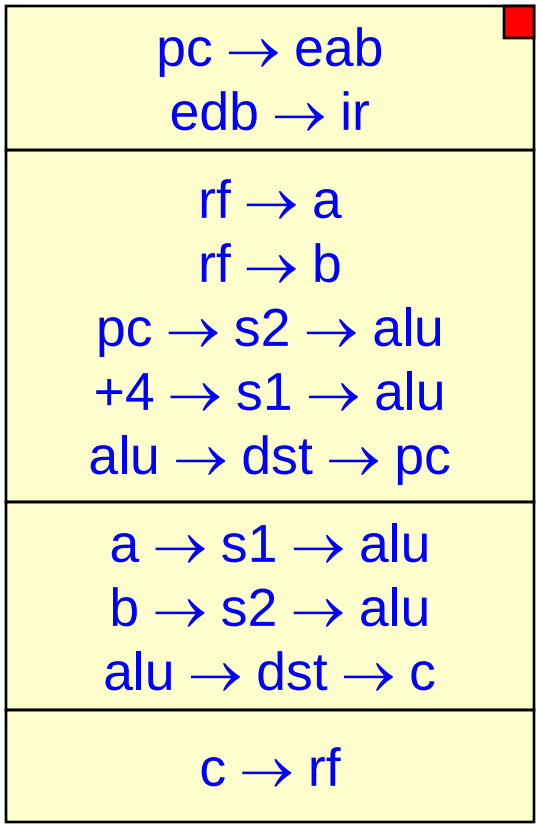


XOR.I Rd, Rs1, Imm

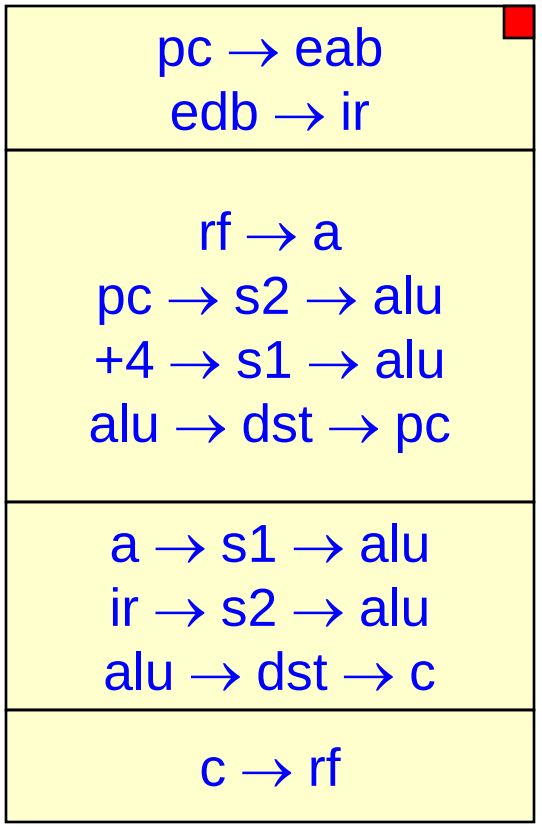


Merged Level 1 Hardware Flowchart

SLL Rd, Rs1, Rs2

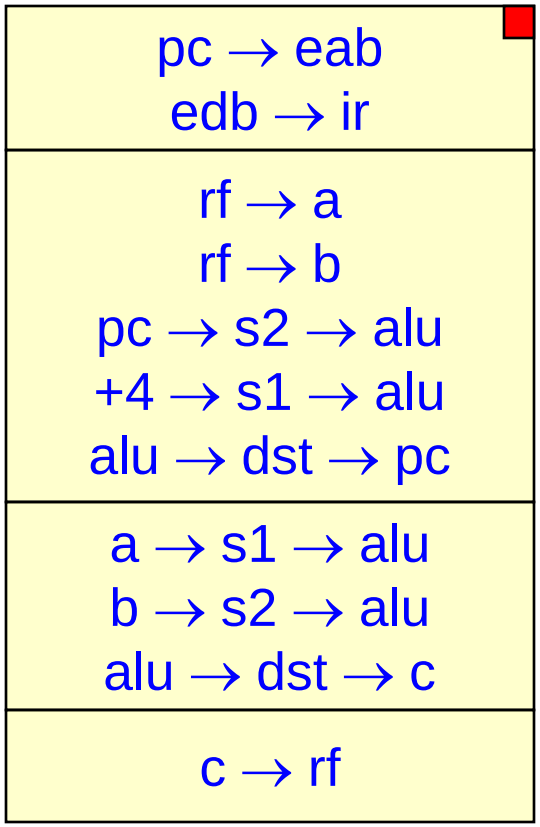


SLL.I Rd, Rs1, Imm

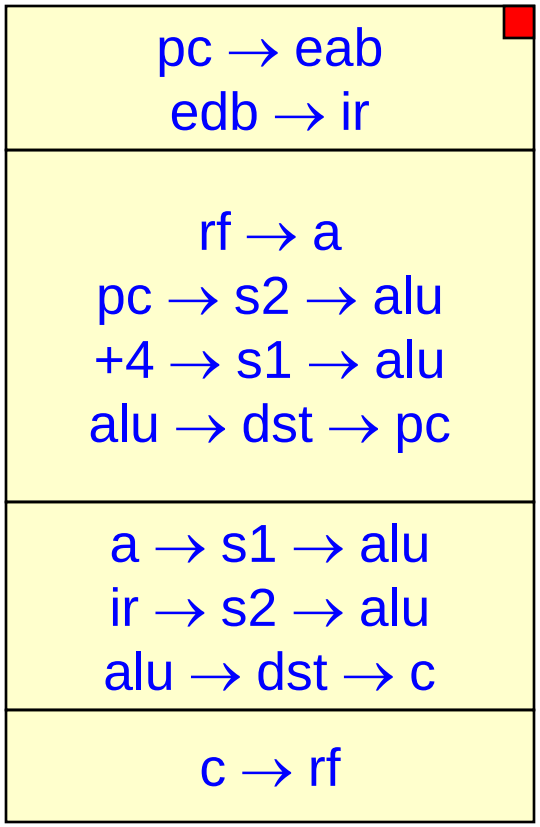


Merged Level 1 Hardware Flowchart

SRL Rd, Rs1, Rs2

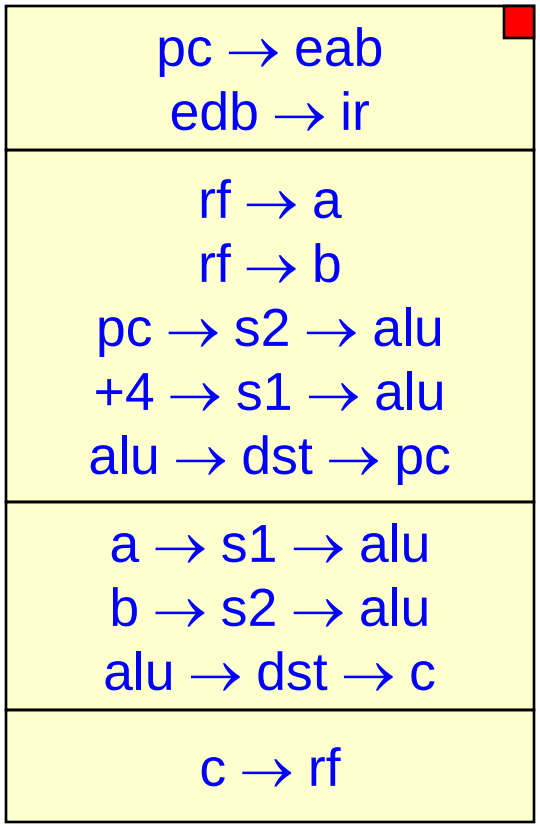


SRL.I Rd, Rs1, Imm

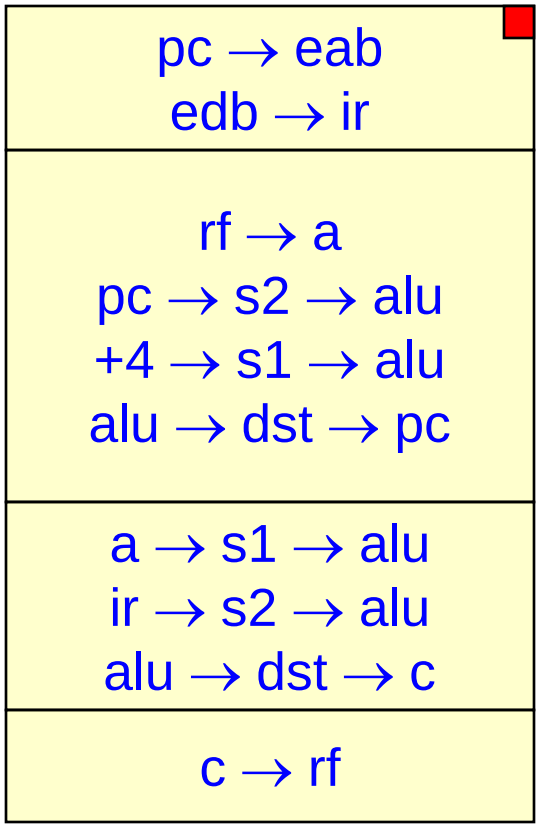


Merged Level 1 Hardware Flowchart

SRA Rd, Rs1, Rs2



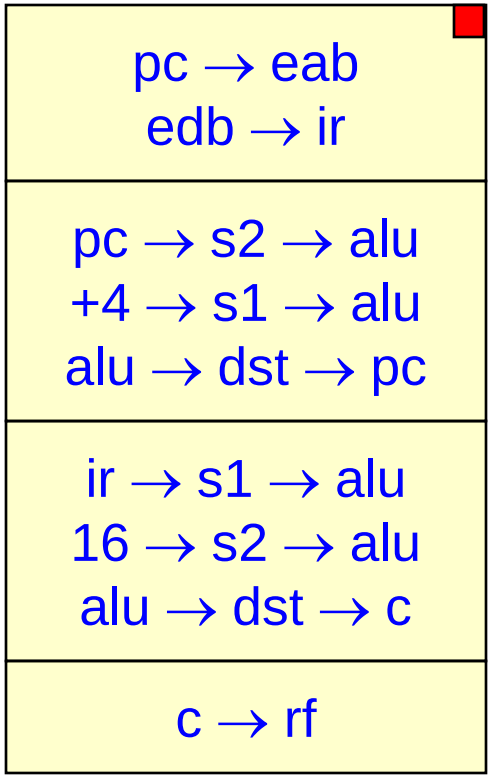
SRA.I Rd, Rs1, Imm



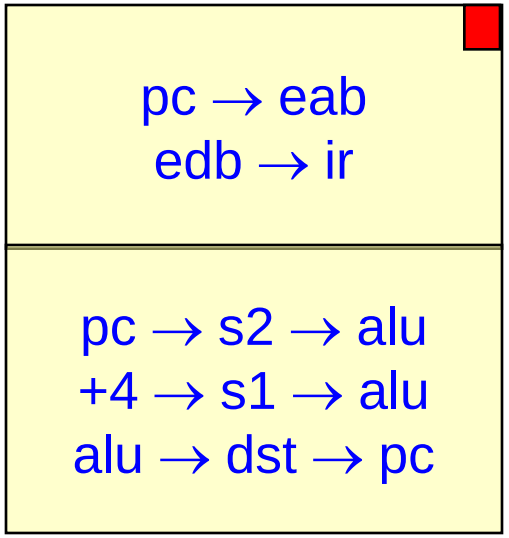
Merged Level 1 Hardware Flowchart

LHI Rd, Imm

Rd(0:15) <- Imm; Rd(16:31) <- hex0000

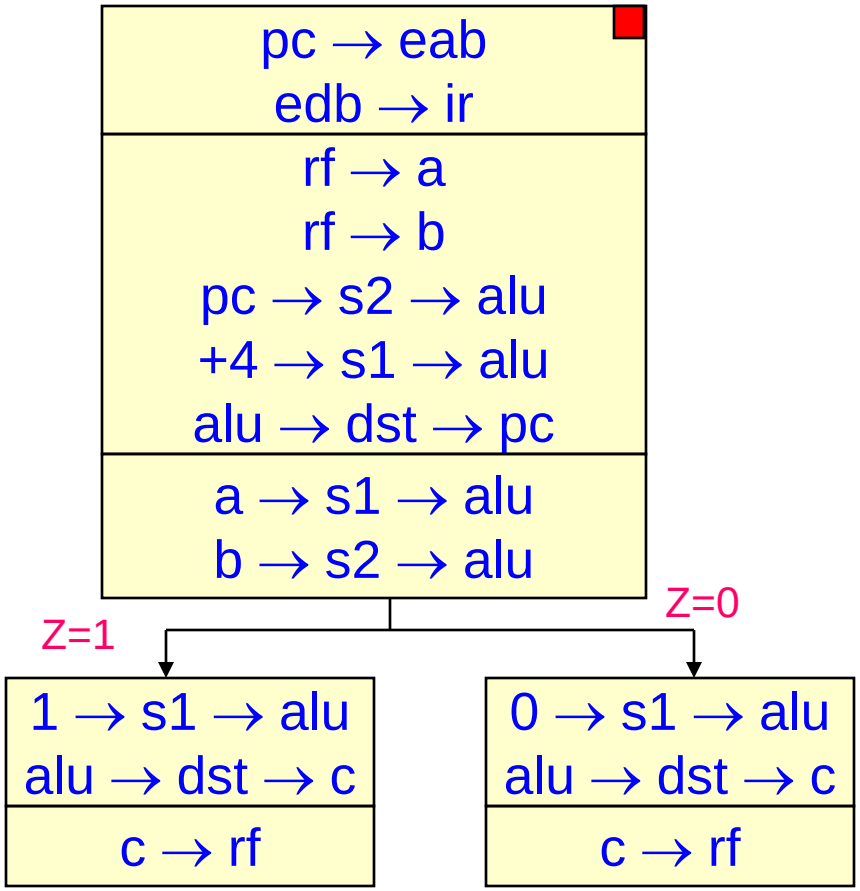


NOP

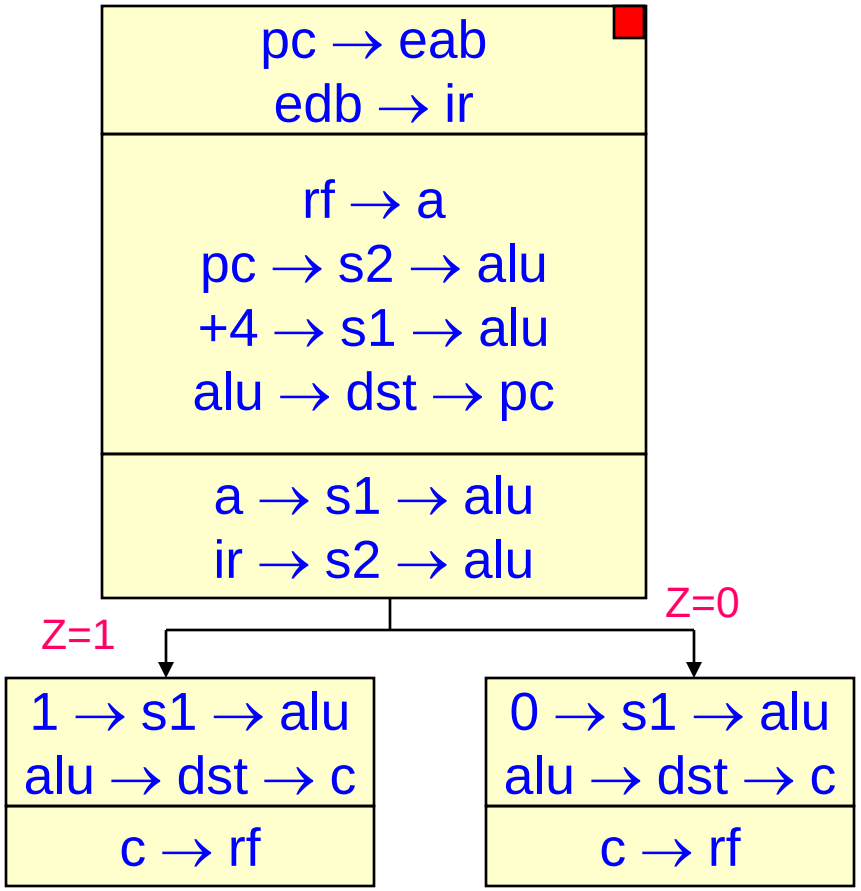


Merged Level 1 Hardware Flowchart

SEQ Rd, Rs1, Rs2

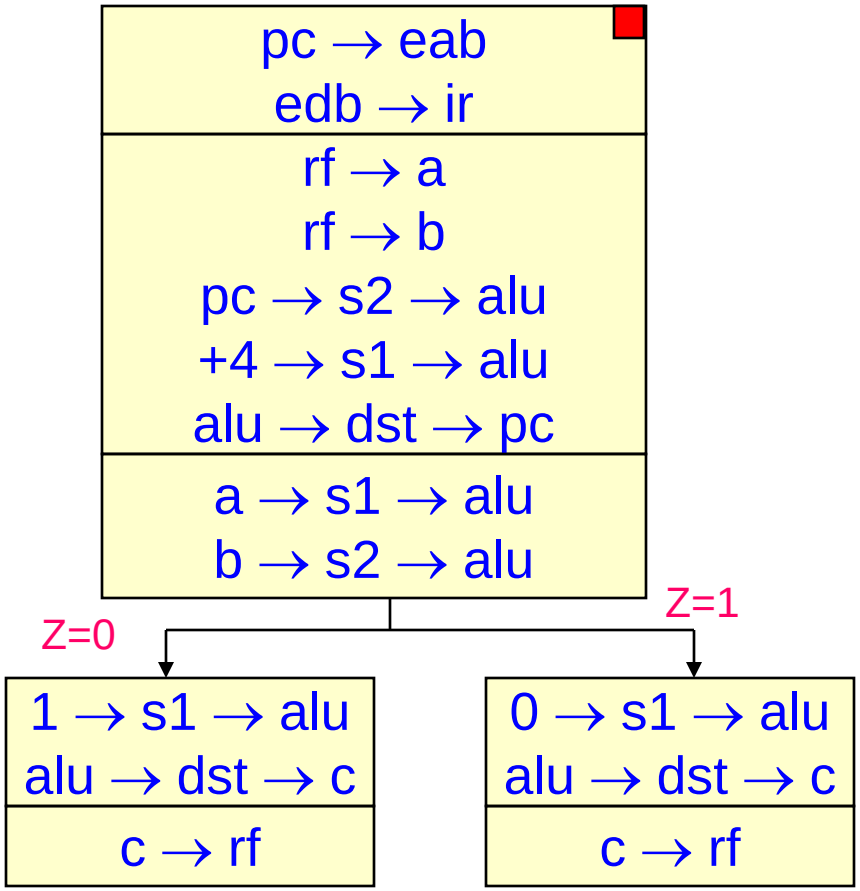


SEQ.I Rd, Rs1, Imm

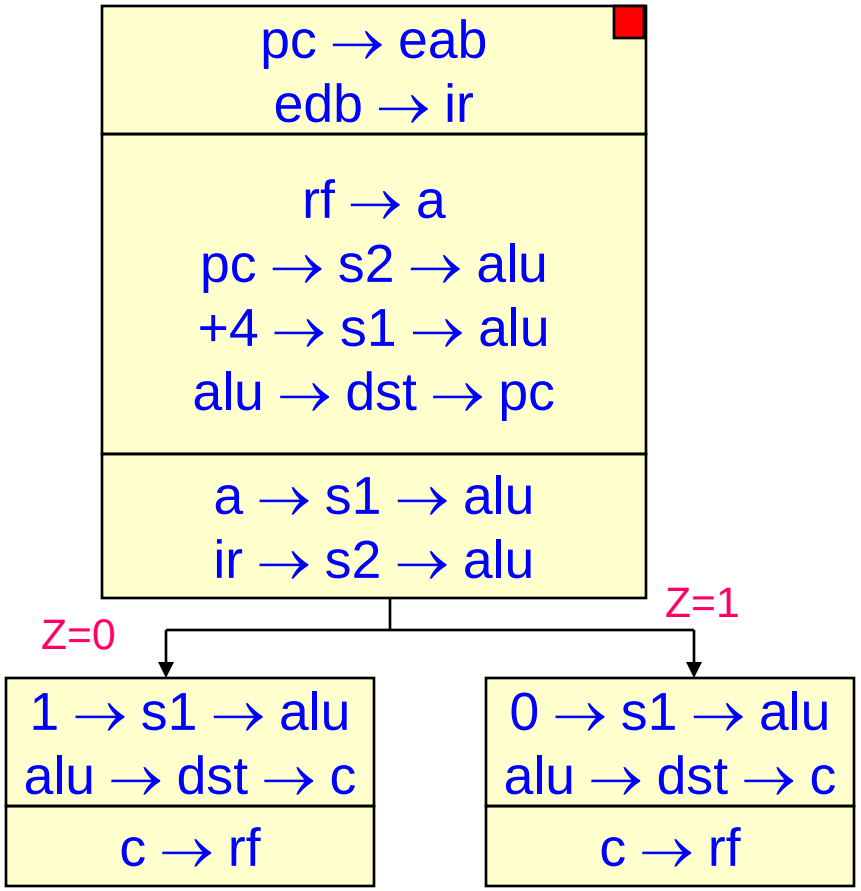


Merged Level 1 Hardware Flowchart

SNE Rd, Rs1, Rs2

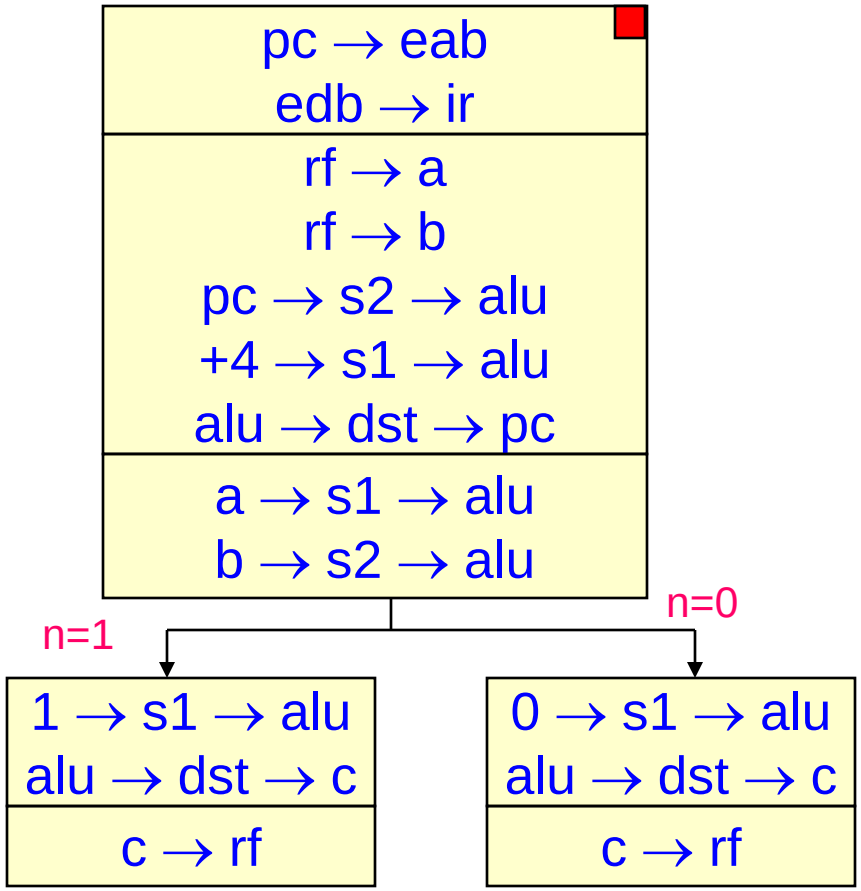


SNE.I Rd, Rs1, Imm

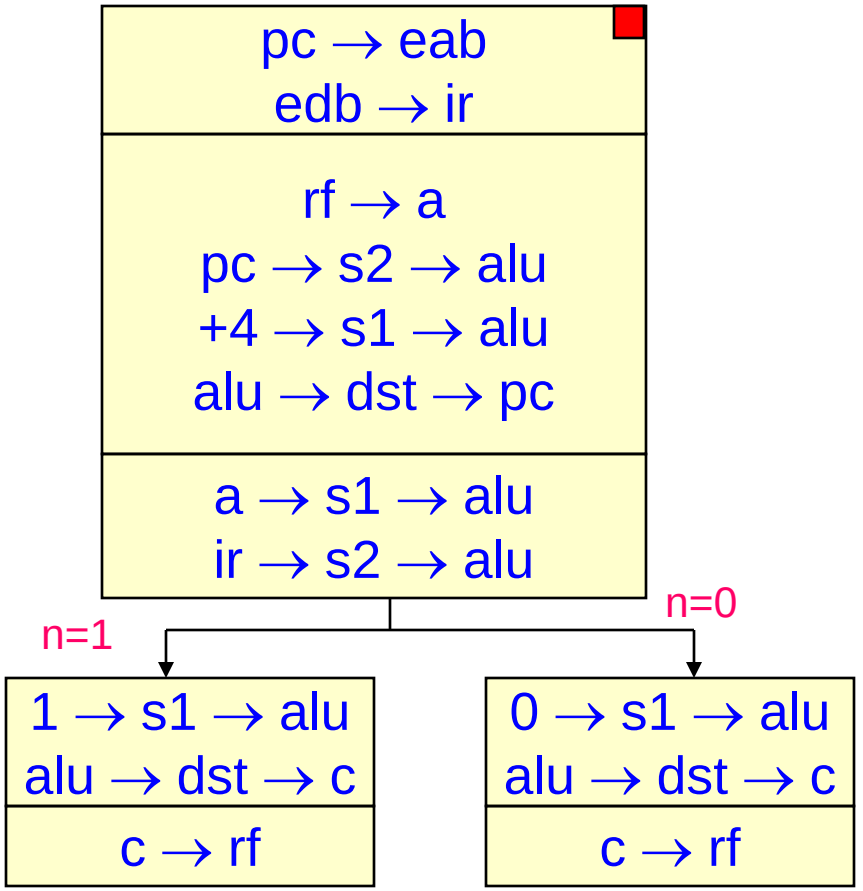


Merged Level 1 Hardware Flowchart

SLT Rd, Rs1, Rs2

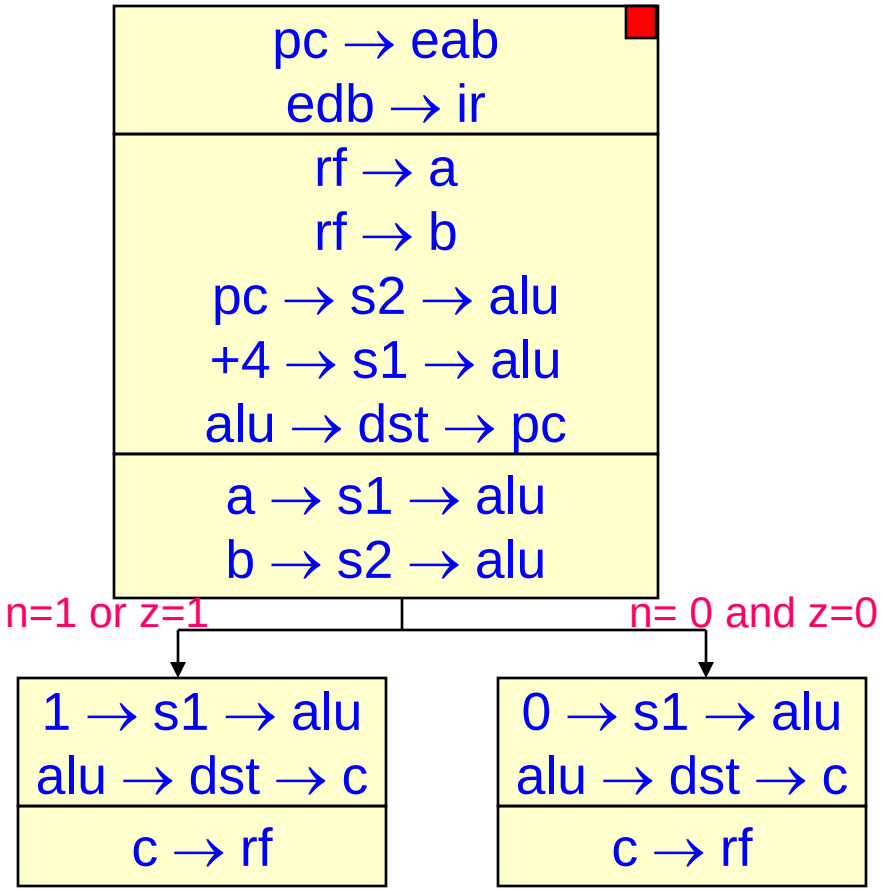


SLT.I Rd, Rs1, Imm

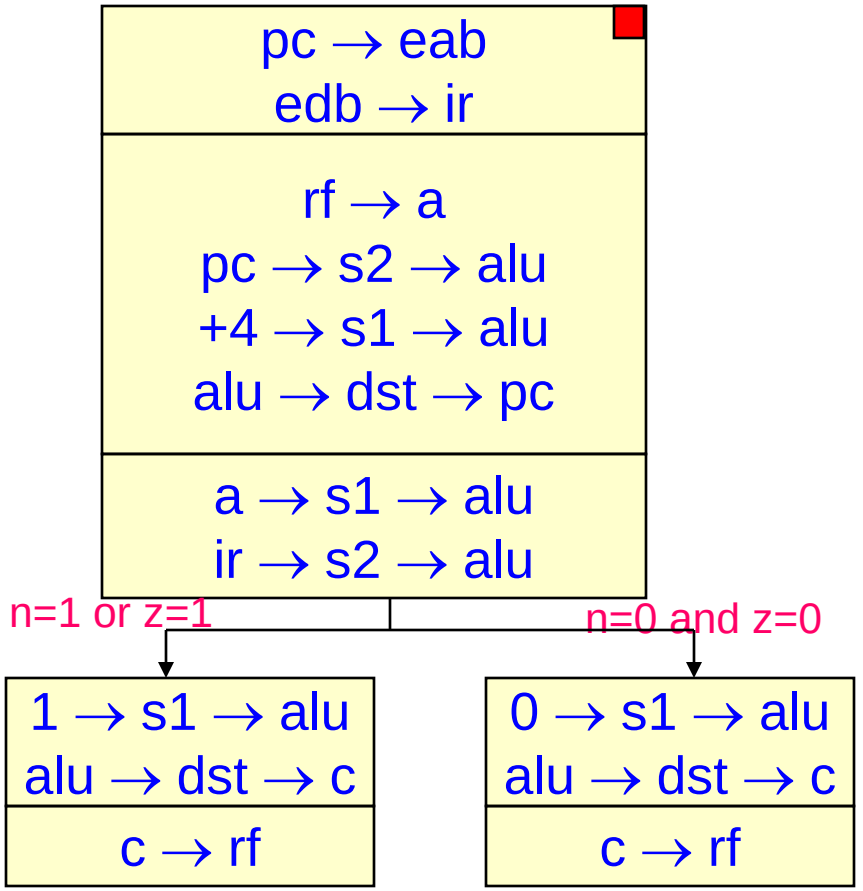


Merged Level 1 Hardware Flowchart

SLE Rd, Rs1, Rs2

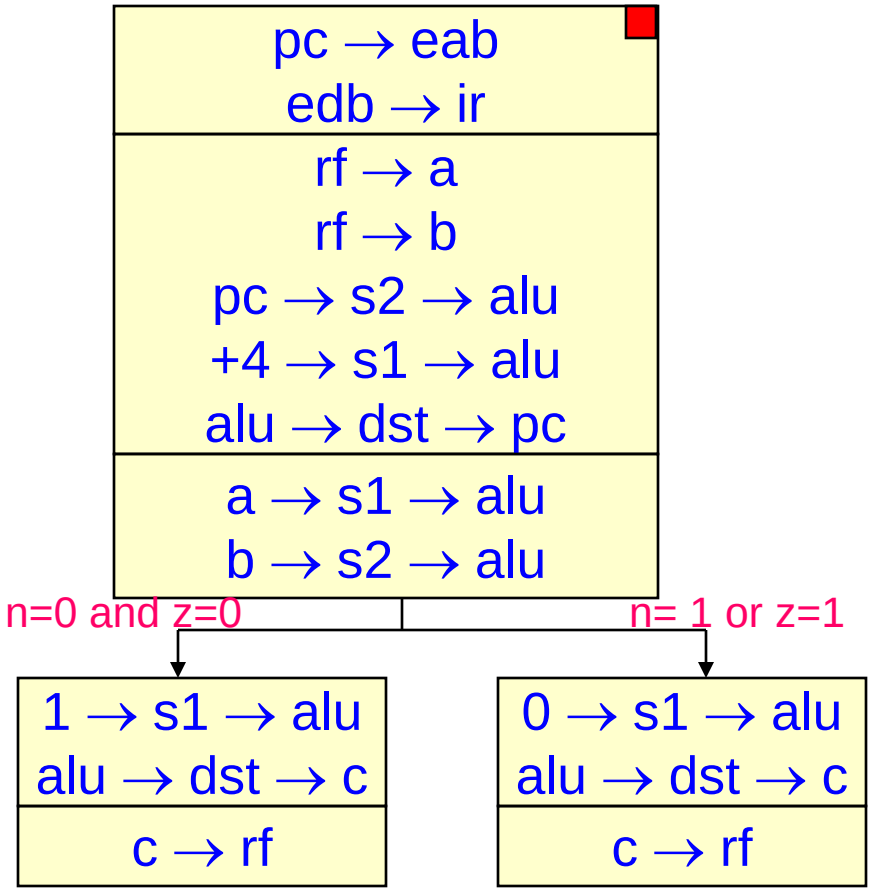


SLE.I Rd, Rs1, Imm

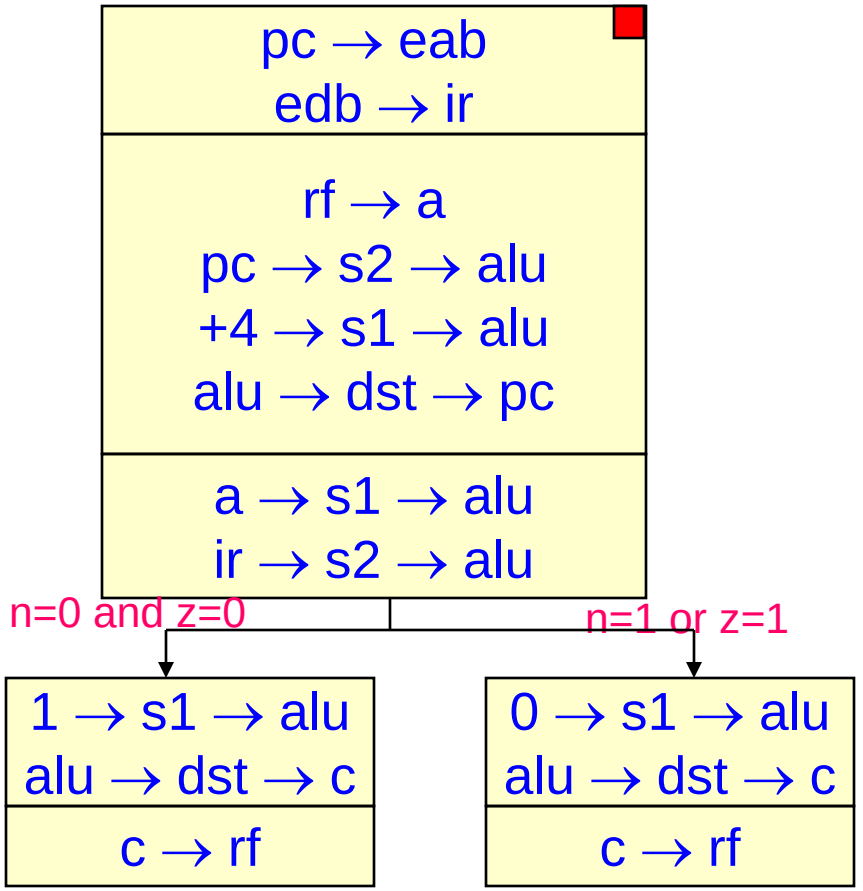


Merged Level 1 Hardware Flowchart

SGT Rd, Rs1, Rs2

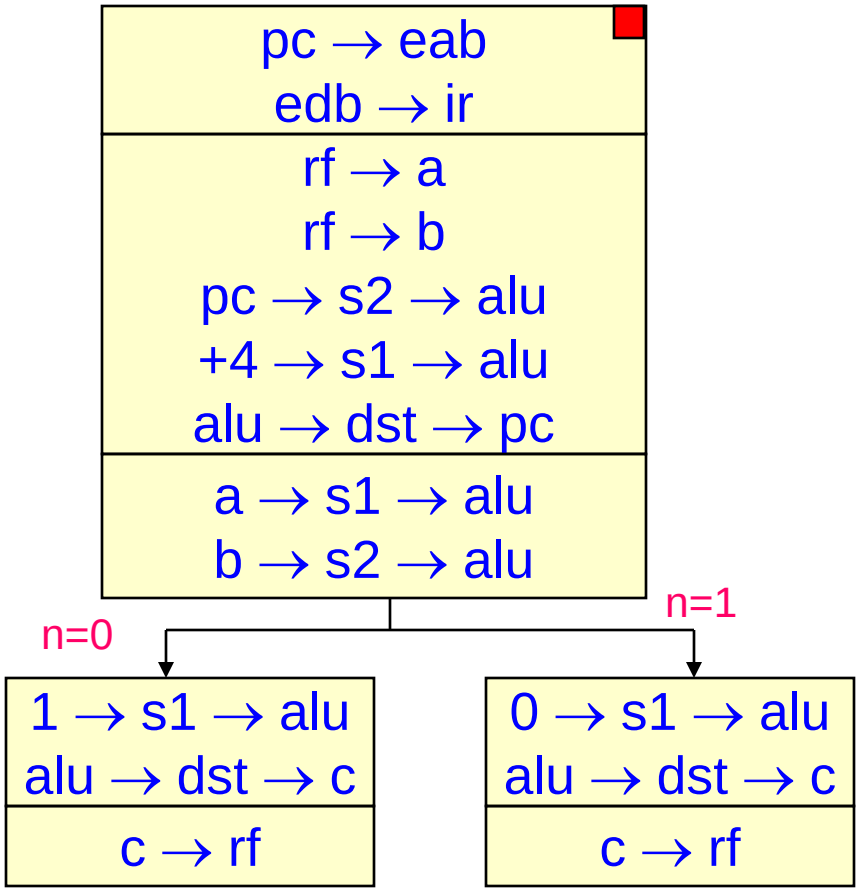


SGT.I Rd, Rs1, Imm

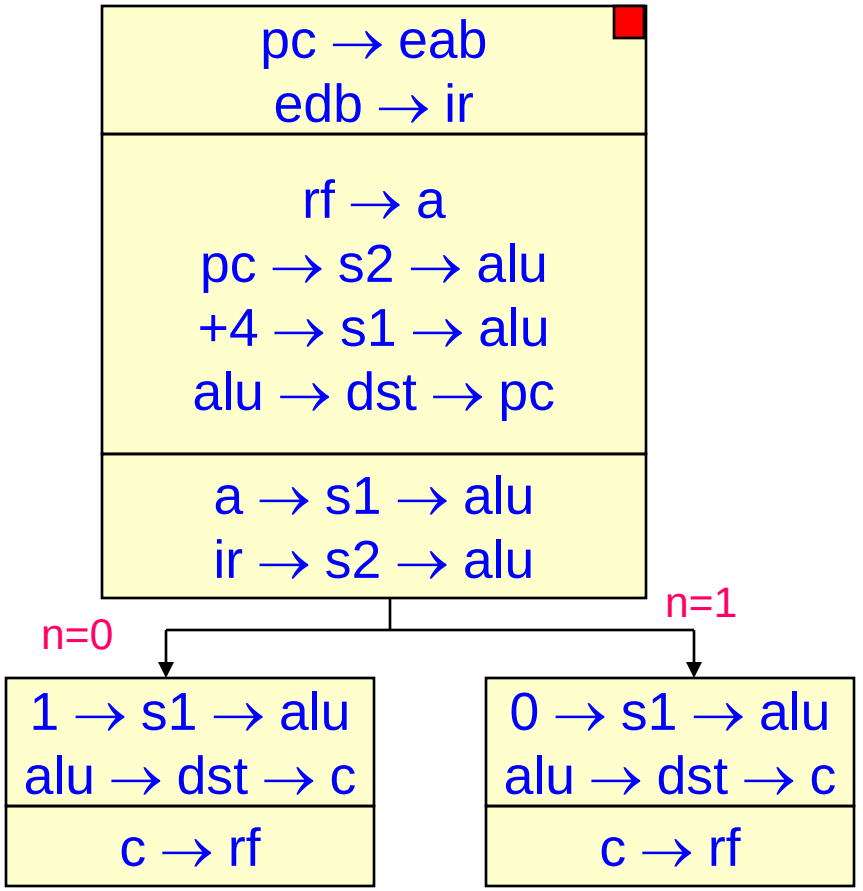


Merged Level 1 Hardware Flowchart

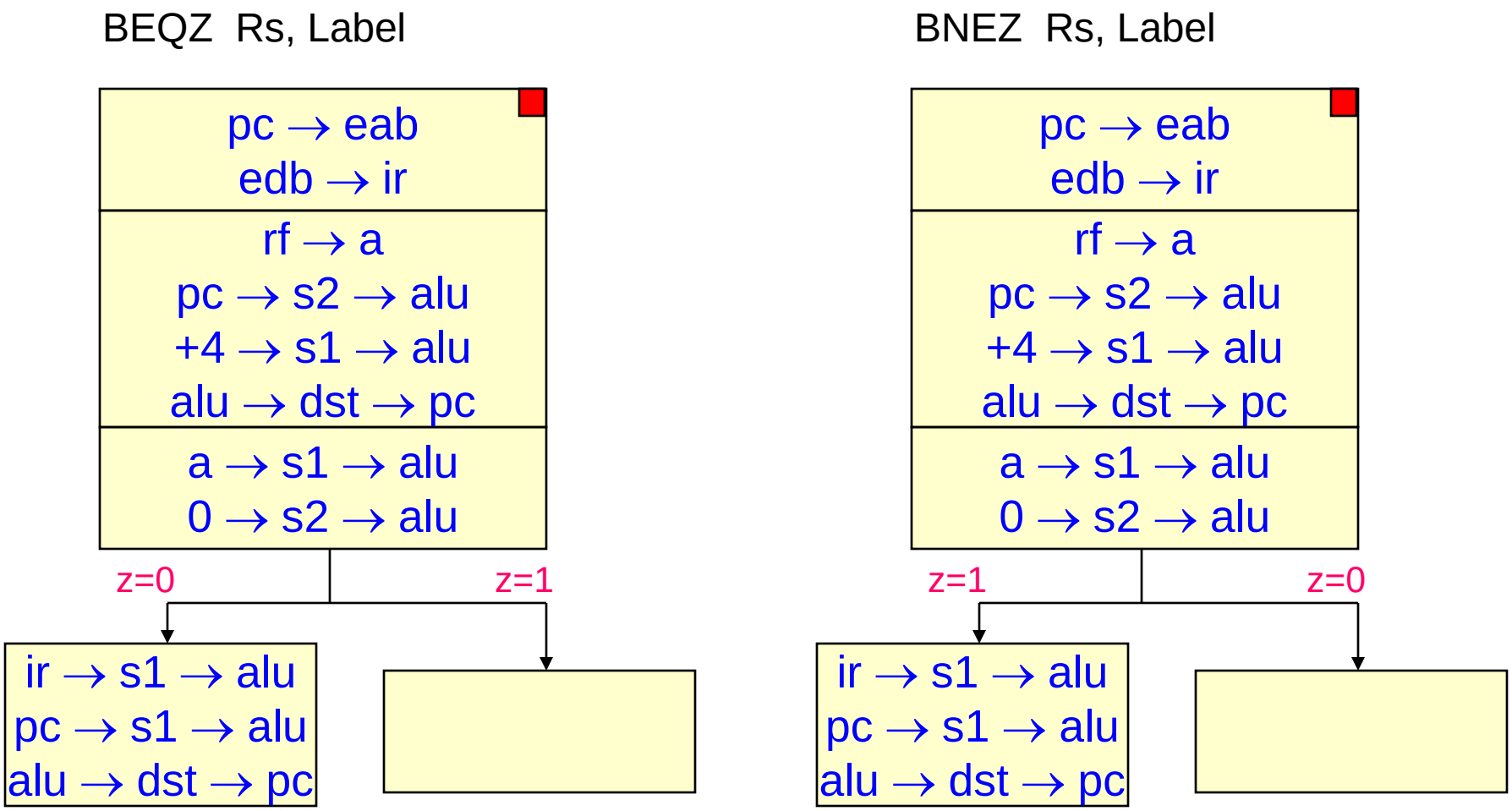
SGE Rd, Rs1, Rs2



SGE.I Rd, Rs1, Imm



Merged Level 1 Hardware Flowchart



Merged Level 1 Hardware Flowchart

J Label

pc → eab edb → ir
pc → s2 → alu +4 → s1 → alu alu → dst → pc
ir → s1 → alu pc → s2 → alu alu → dst → mar
mar → s2 → alu alu → dst → pc

JR Rs

pc → eab edb → ir
rf → a pc → s2 → alu +4 → s1 → alu alu → dst → pc
a → s1 → alu alu → dst → mar
mar → s2 → alu alu → dst → pc

Merged Level 1 Hardware Flowchart

JAL Label

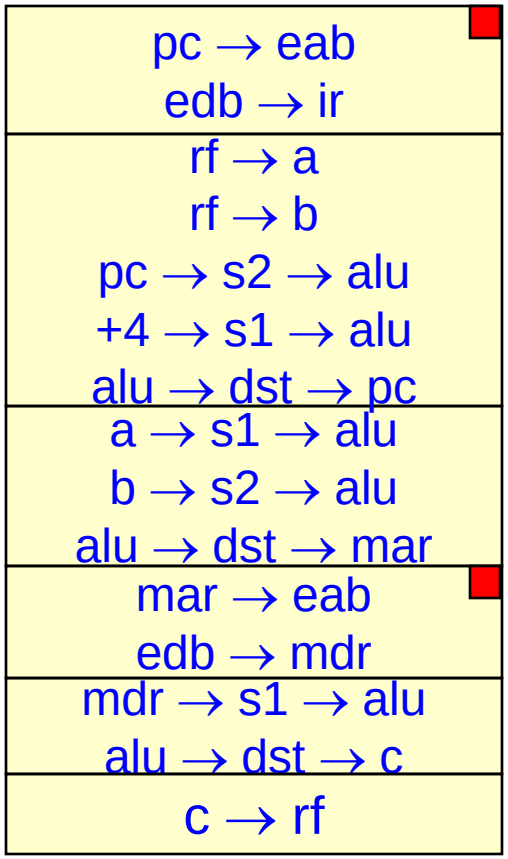
pc → eab edb → ir
pc → s2 → alu +4 → s1 → alu alu → dst → pc
pc → s2 → alu alu → dst → c
ir → s1 → alu pc → s2 → alu alu → dst → mar c → rf
mar → s2 → alu alu → dst → pc

JALR Rs

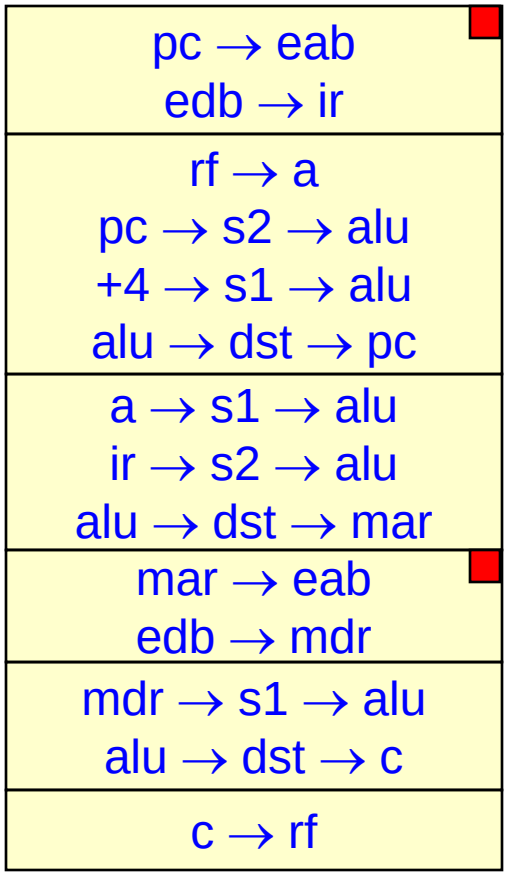
pc → eab edb → ir
rf → a pc → s2 → alu +4 → s1 → alu alu → dst → pc
pc → s2 → alu alu → dst → c
a → s1 → alu alu → dst → mar c → rf
mar → s2 → alu alu → dst → pc

Merged Level 1 Hardware Flowchart

LW Rd, Rs2(Rs1)



LW.I Rd, Imm(Rs1)



Merged Level 1 Hardware Flowchart

SW Rd, Rs2(Rs1)

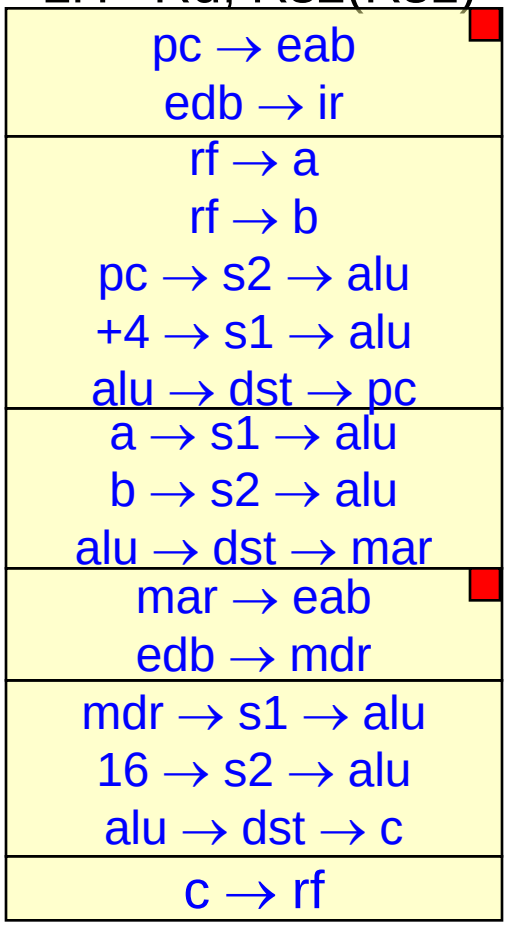
pc → eab edb → ir
rf → a rf → b pc → s2 → alu +4 → s1 → alu alu → dst → pc
a → s1 → alu b → s2 → alu alu → dst → mar rf → b
b → s2 → alu alu → dst → mdr
mar → eab mdr → edb

SW.I Rd, Imm(Rs1)

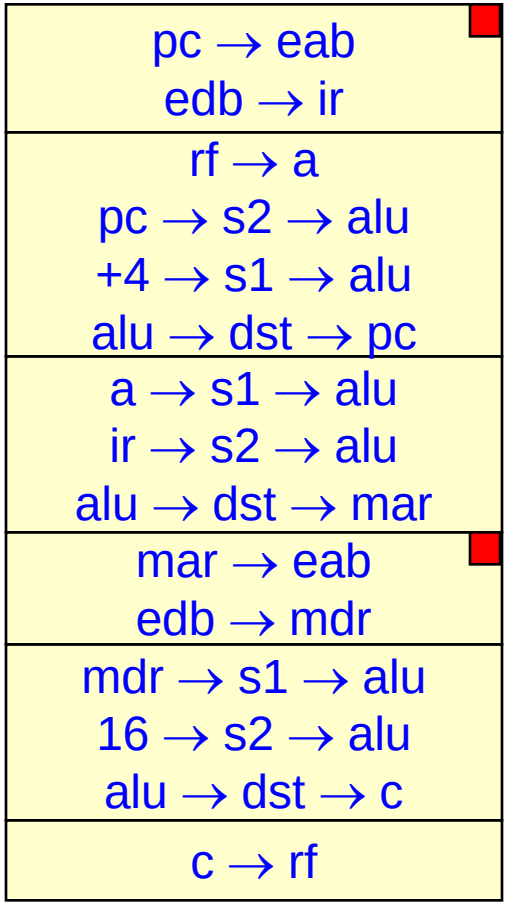
pc → eab edb → ir
rf → a pc → s2 → alu +4 → s1 → alu alu → dst → pc
a → s1 → alu ir → s2 → alu alu → dst → mar rf → b
b → s2 → alu alu → dst → mdr
mar → eab mdr → edb

Merged Level 1 Hardware Flowchart

LH Rd, Rs2(Rs1)



LW.I Rd, Imm(Rs1)



Merged Level 1 Hardware Flowchart

SH Rd, Rs2(Rs1)

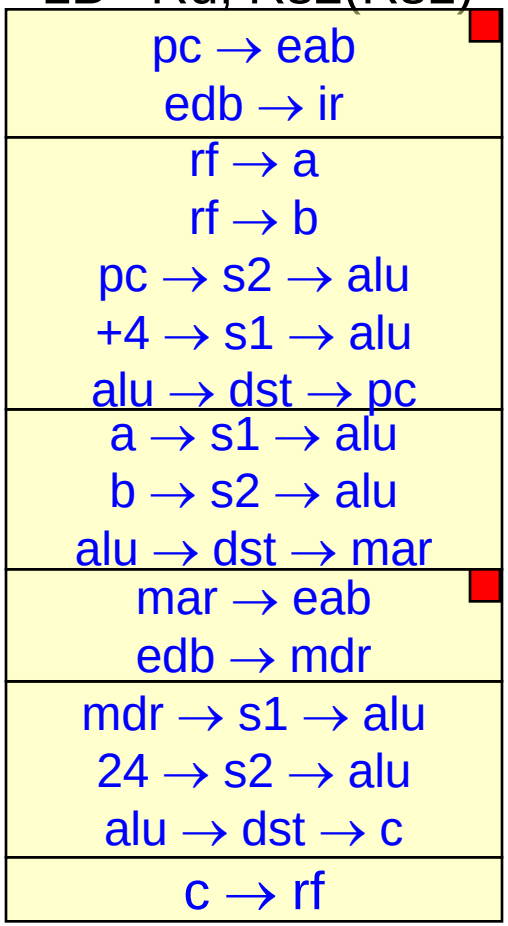
pc → eab edb → ir
rf → a rf → b pc → s2 → alu +4 → s1 → alu alu → dst → pc
a → s1 → alu b → s2 → alu alu → dst → mar rf → b
b → s2 → alu alu → dst → mdr
mar → eab mdr → edb

SH.I Rd, Imm(Rs1)

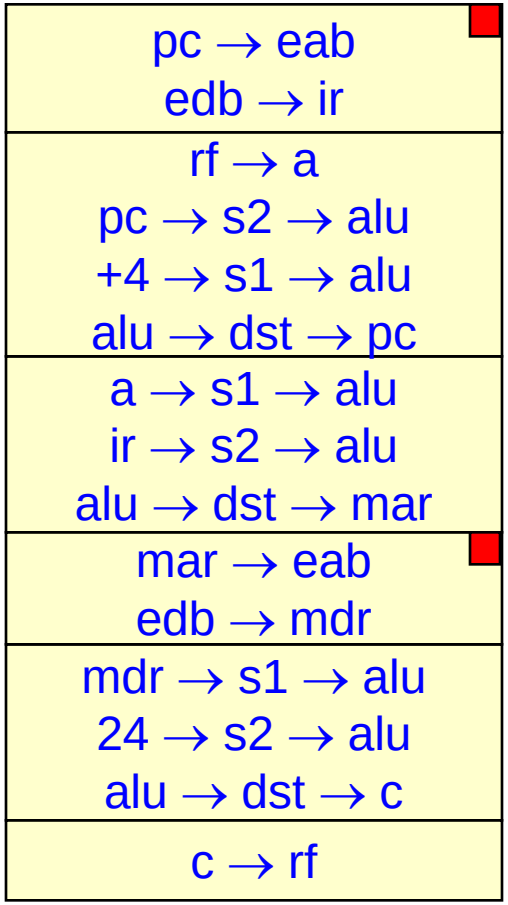
pc → eab edb → ir
rf → a pc → s2 → alu +4 → s1 → alu alu → dst → pc
a → s1 → alu ir → s2 → alu alu → dst → mar rf → b
b → s2 → alu alu → dst → mdr
mar → eab mdr → edb

Merged Level 1 Hardware Flowchart

LB Rd, Rs2(Rs1)

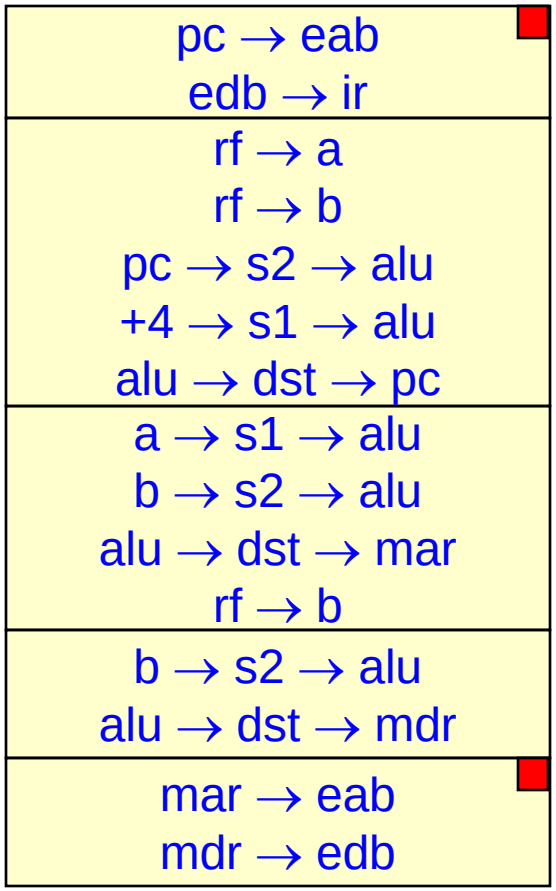


LB.I Rd, Imm(Rs1)

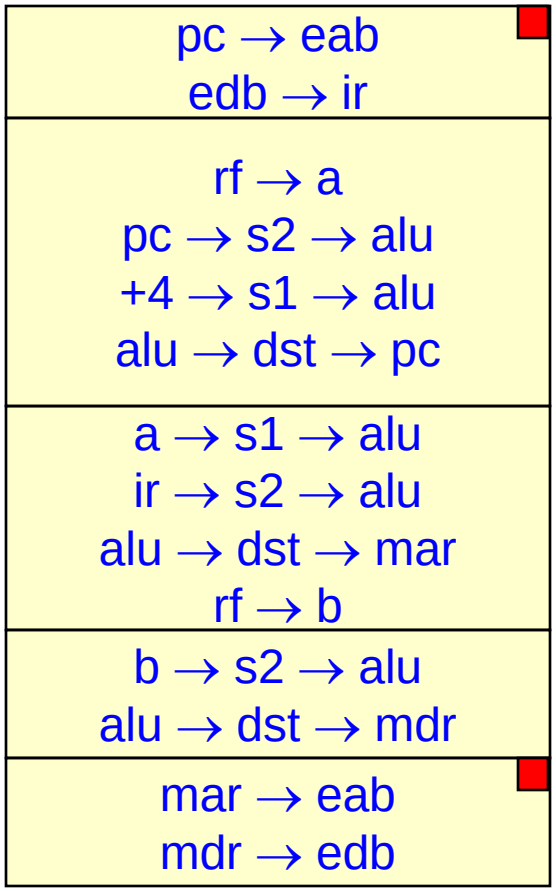


Merged Level 1 Hardware Flowchart

SB Rd, Rs2(Rs1)

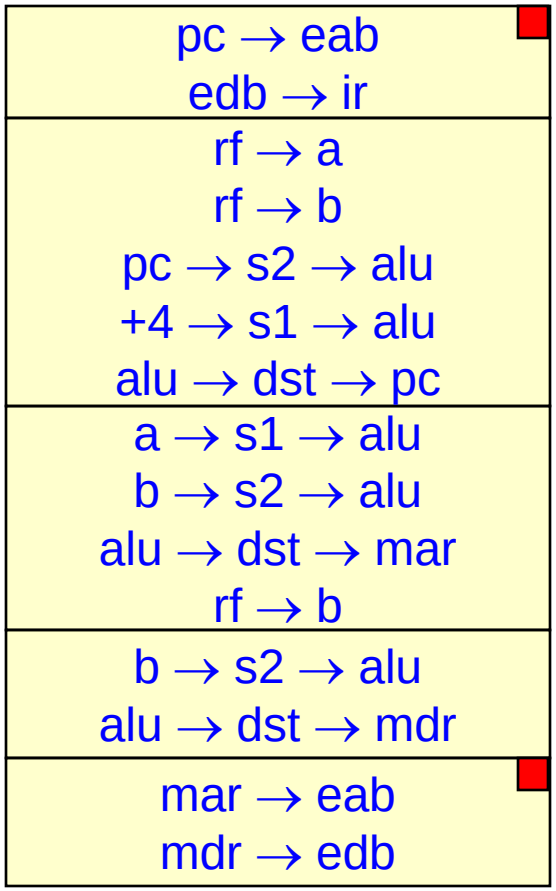


SB.I Rd, Imm(Rs1)

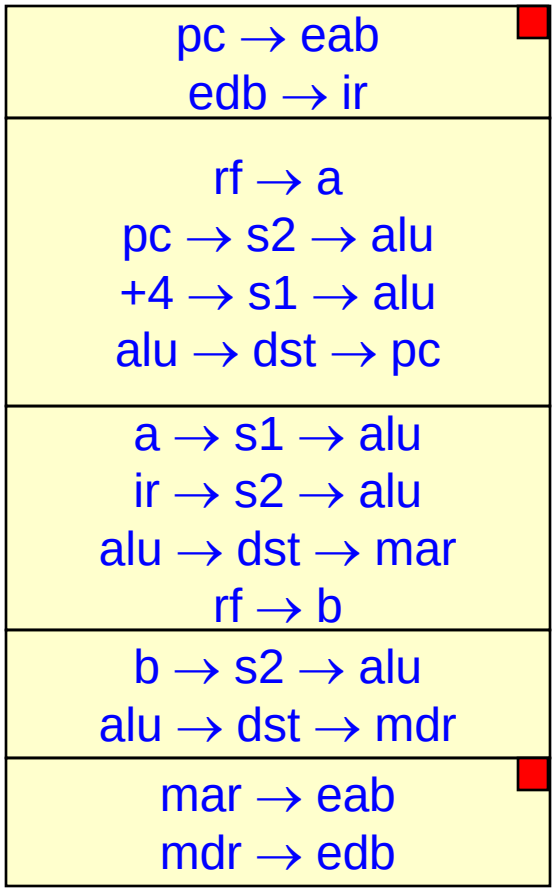


Merged Level 1 Hardware Flowchart

SB Rd, Rs2(Rs1)

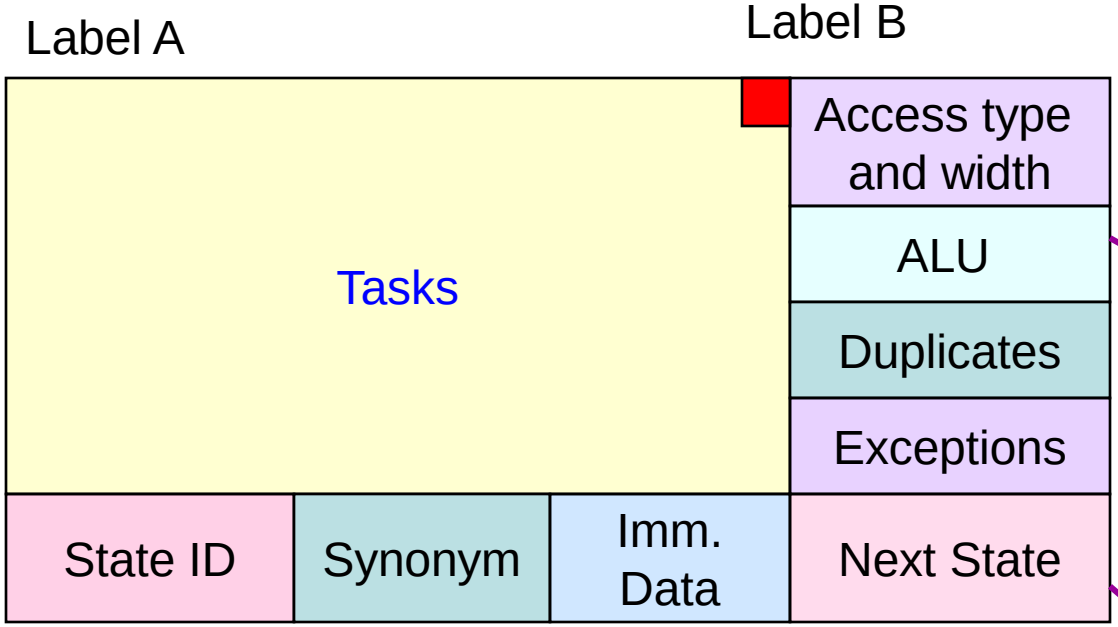


SB.I Rd, Imm(Rs1)



Level 2 Flowcharts

Format for Level 2 flowchart state



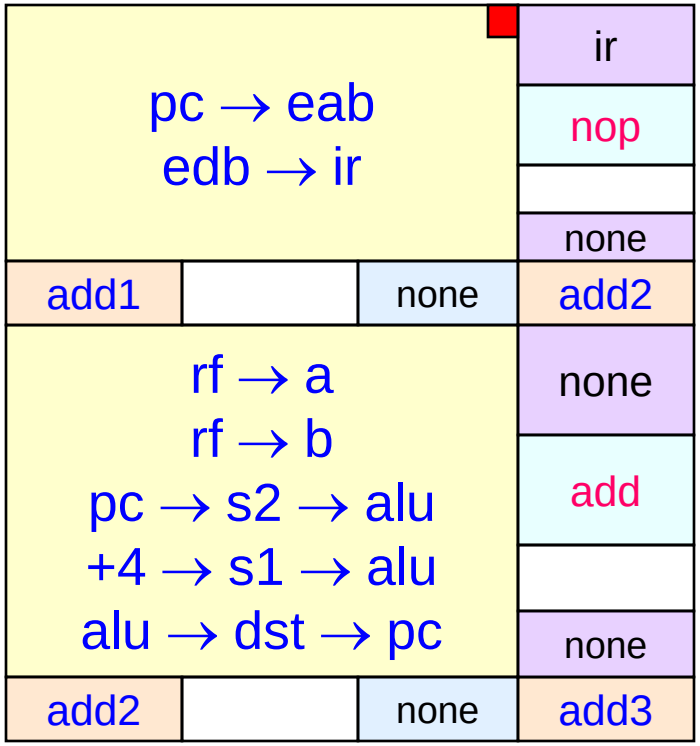
- DR – RW, RH, RB
- DW – WW, WH, WB
- IR - Instruction Read
- NA – No aceess

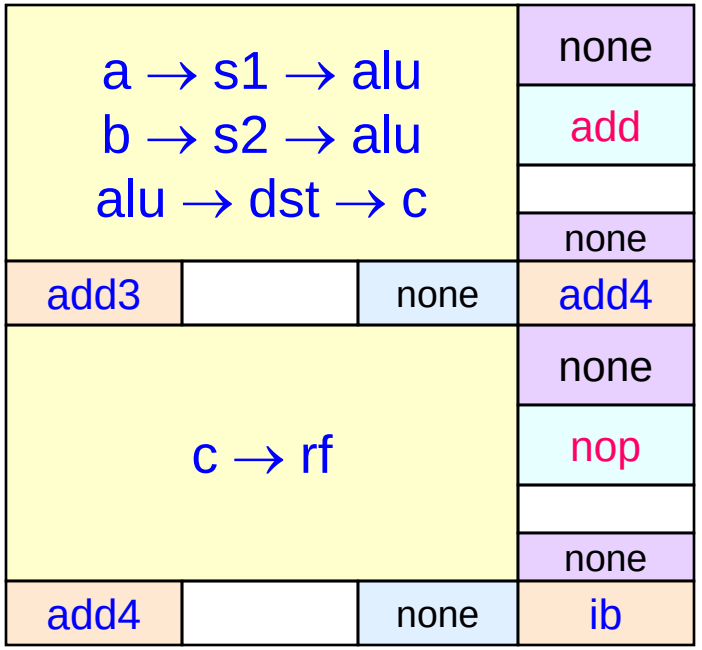
- add, sub
- and, or, xor
- sll, srl, sra
- s1_pass, s2_pass
- nop

- BC – Branch conditionally
- StateID – Direct

Level 2 Hardware Flowchart

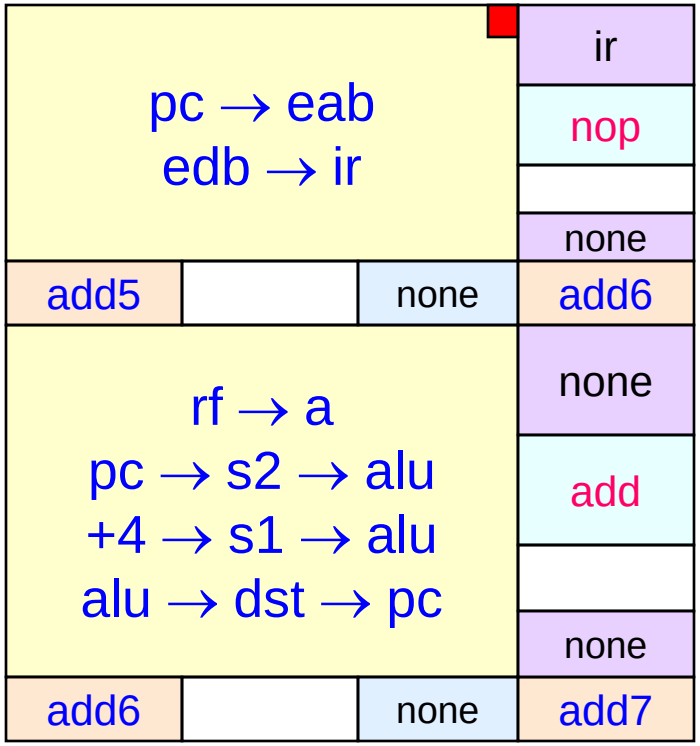
ADD Rd, Rs1, Rs2

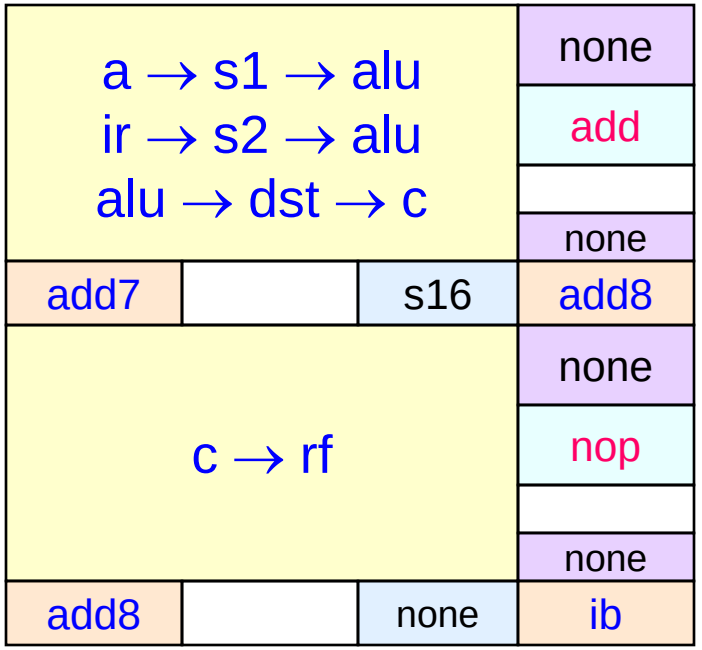




Level 2 Hardware Flowchart

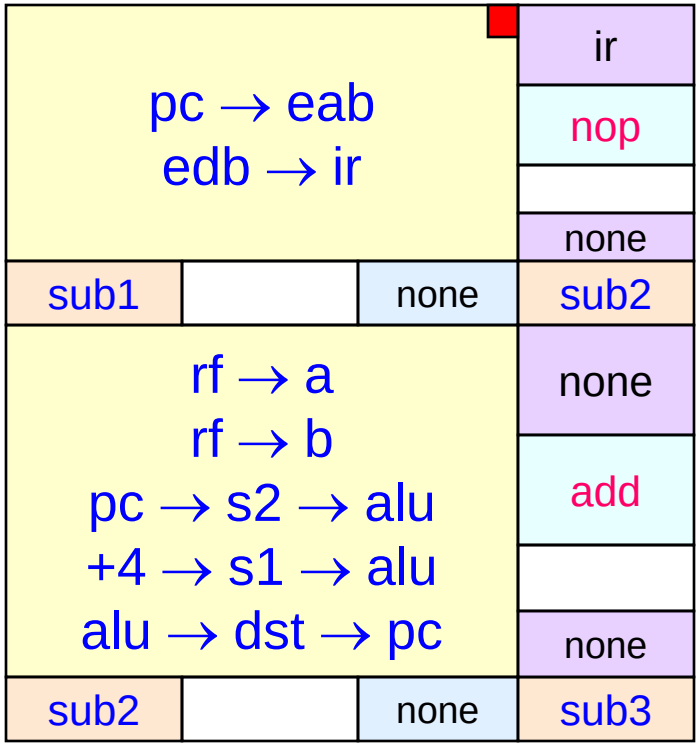
ADD.I Rd, Rs1, Imm

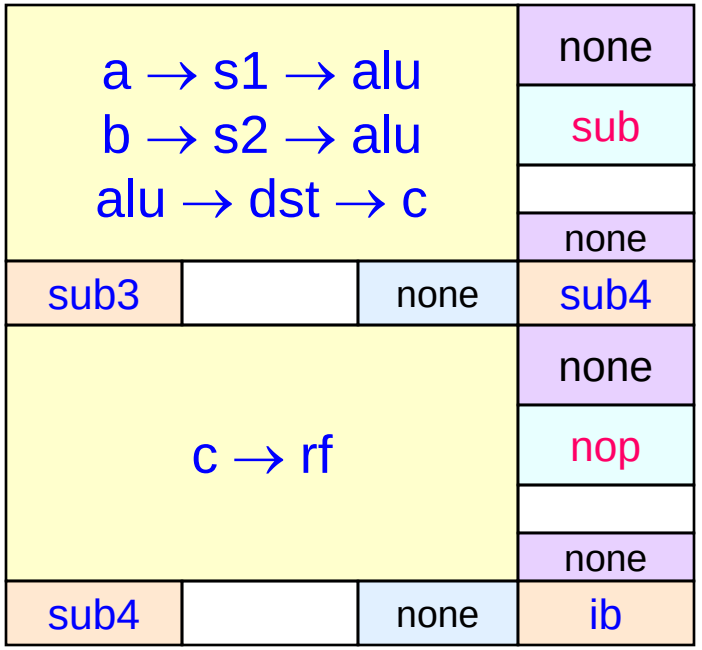




Level 2 Hardware Flowchart

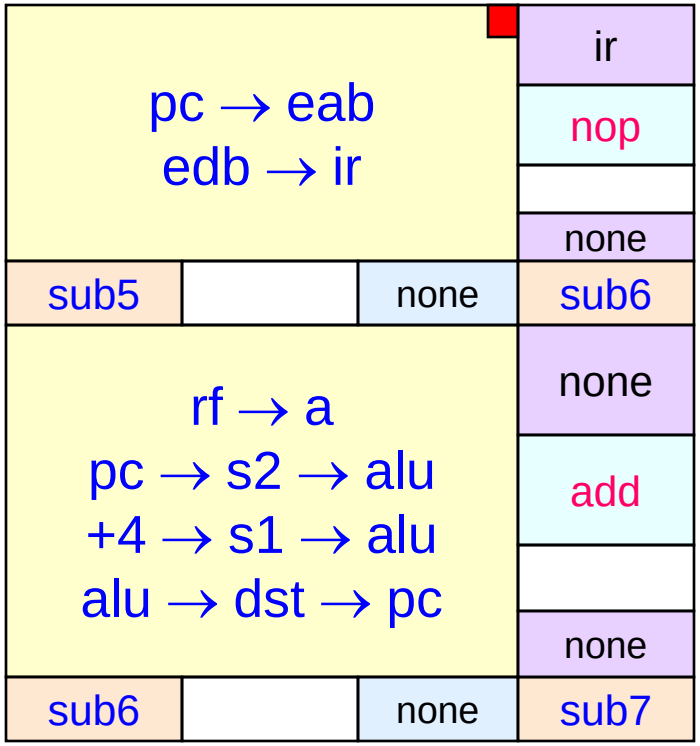
SUB Rd, Rs1, Rs2

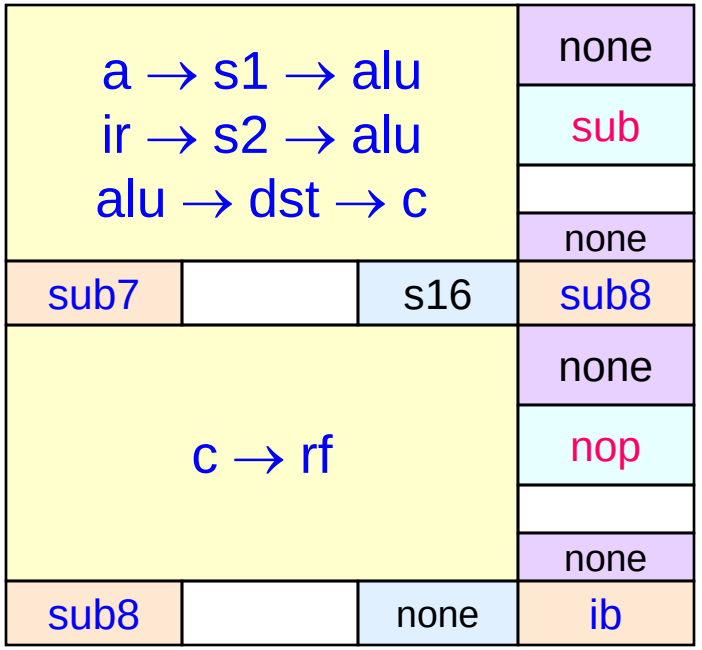




Level 2 Hardware Flowchart

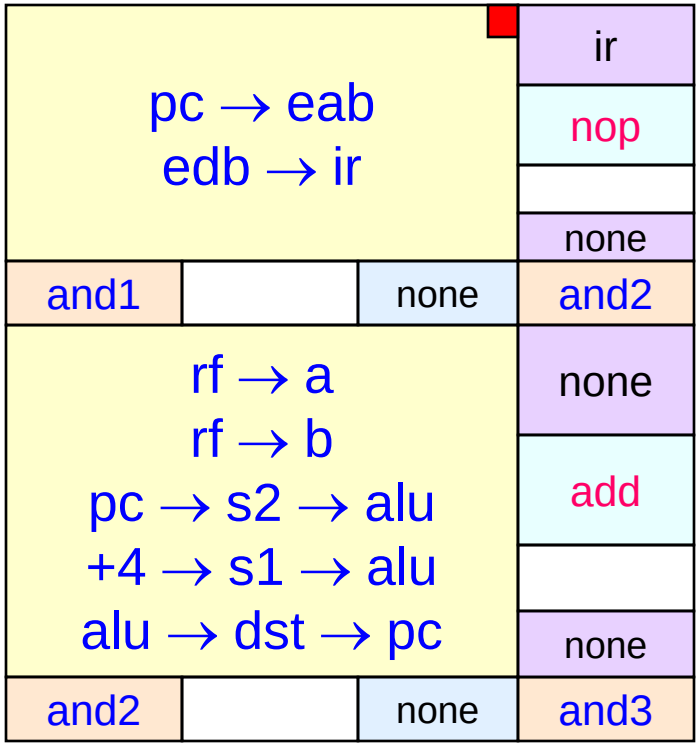
SUB.I Rd, Rs1, Imm

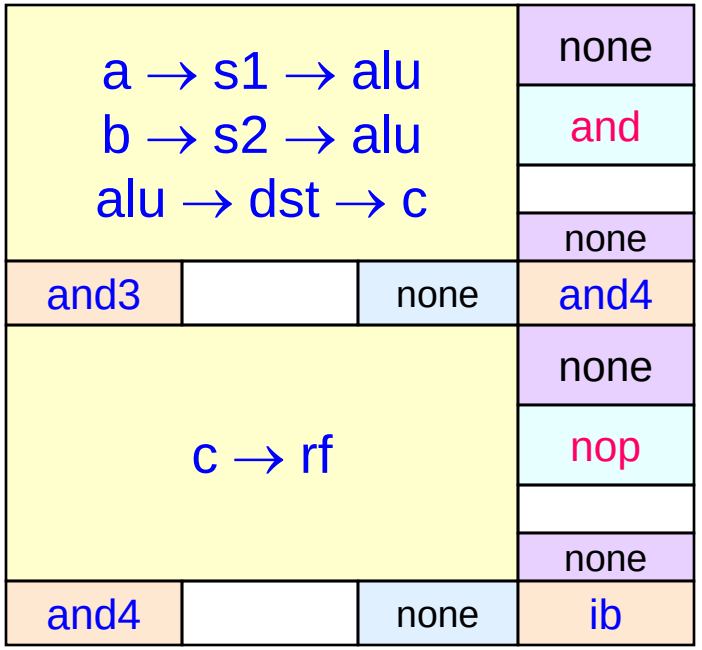




Level 2 Hardware Flowchart

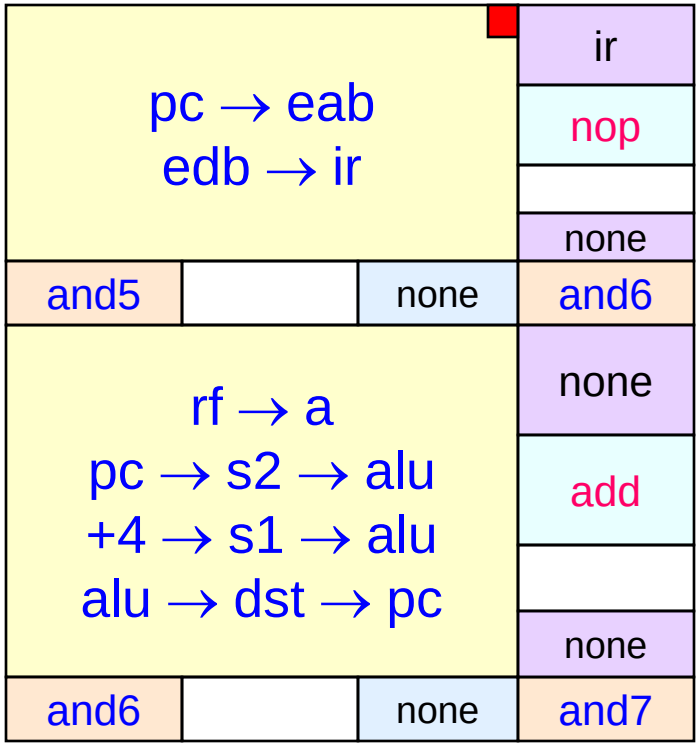
AND Rd, Rs1, Rs2

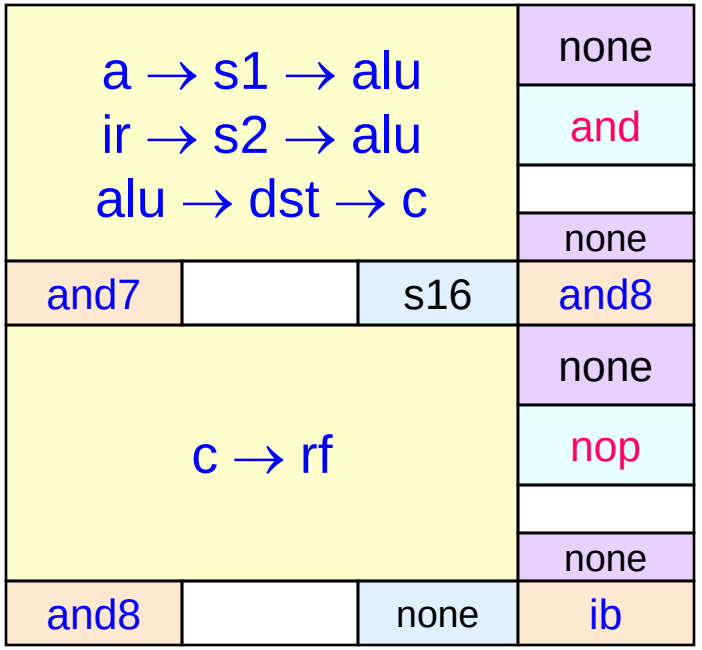




Level 2 Hardware Flowchart

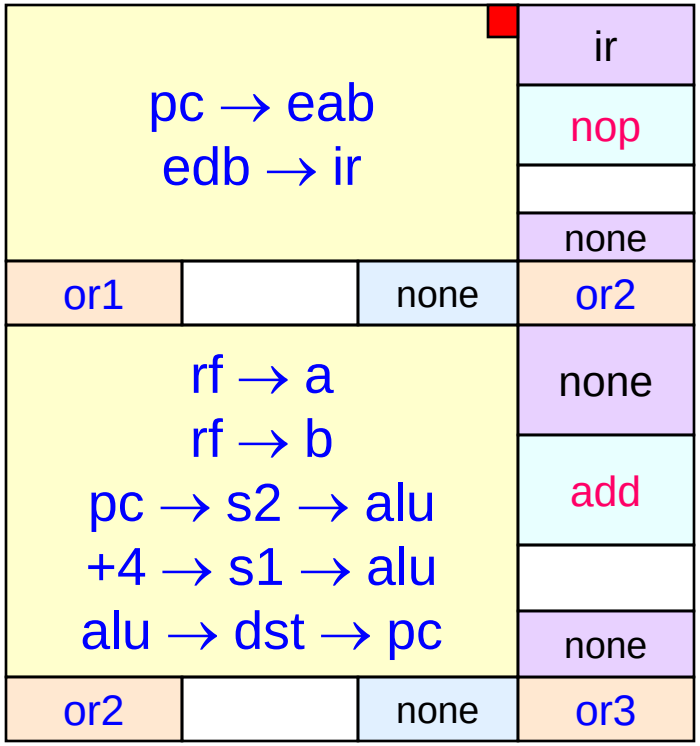
AND.I Rd, Rs1, Imm

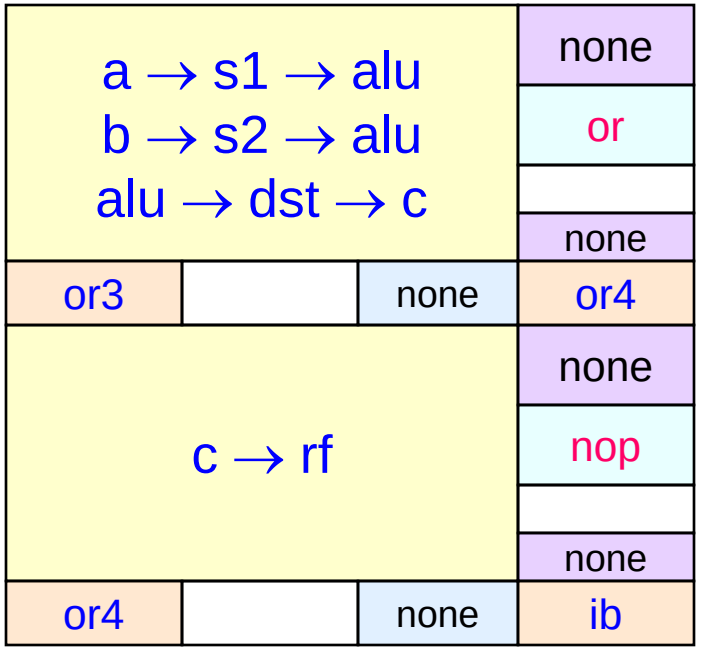




Level 2 Hardware Flowchart

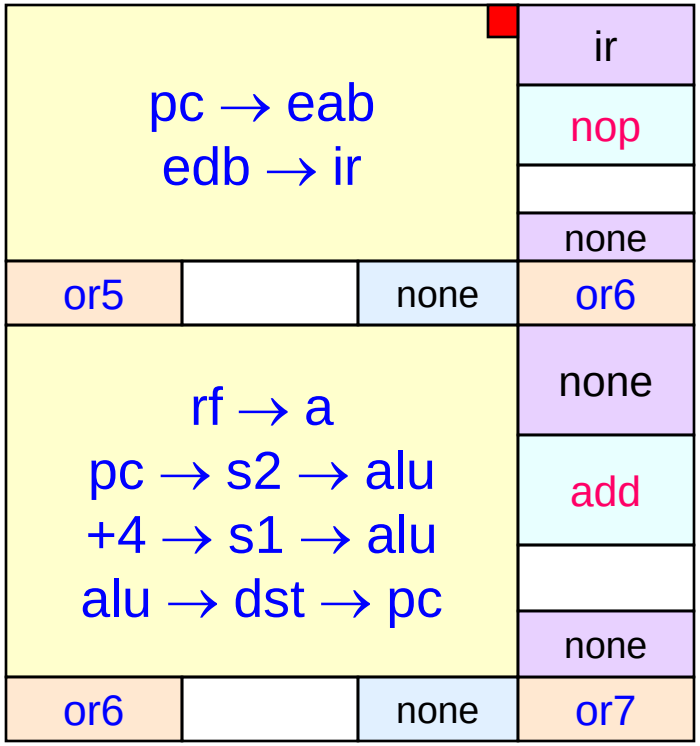
OR Rd, Rs1, Rs2

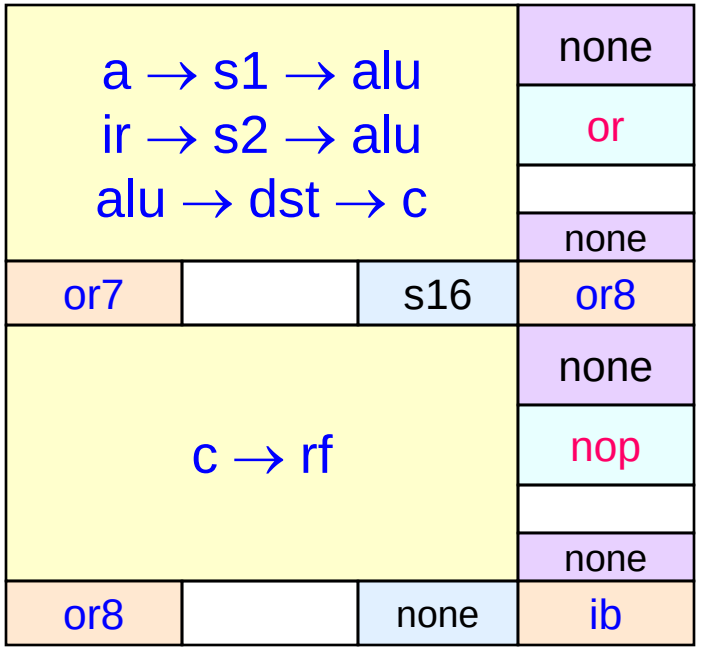




Level 2 Hardware Flowchart

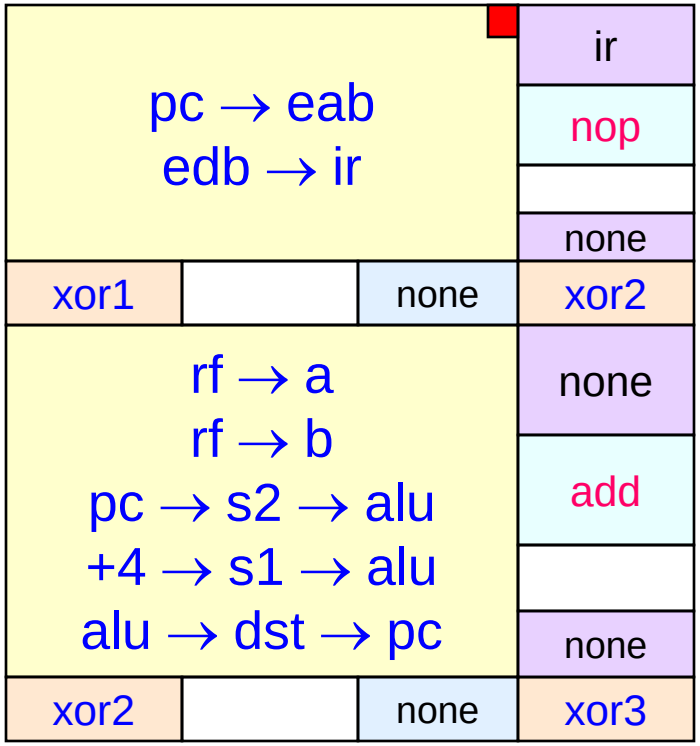
OR.I Rd, Rs1, Imm

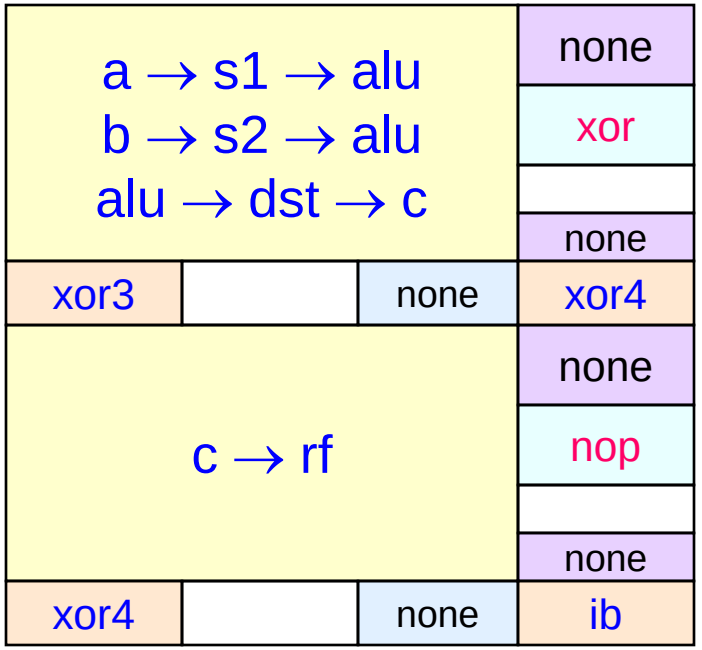




Level 2 Hardware Flowchart

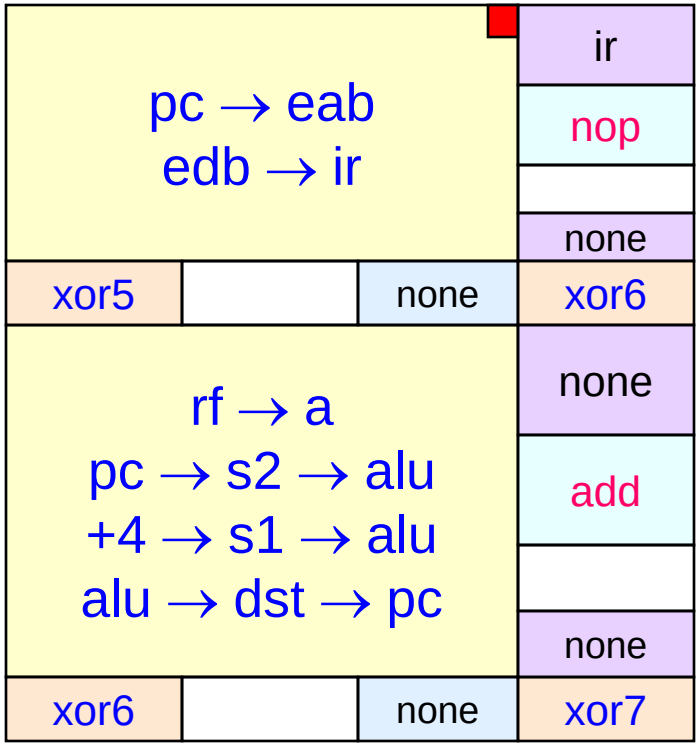
XOR Rd, Rs1, Rs2

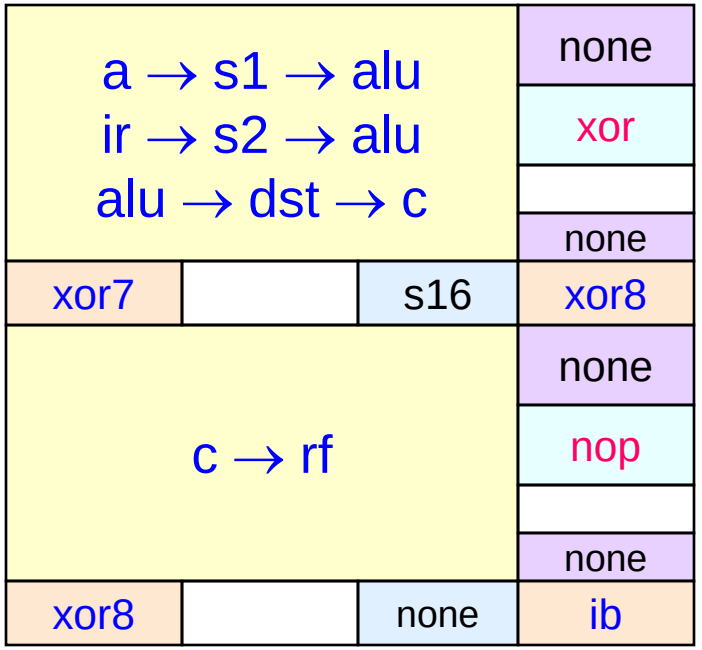




Level 2 Hardware Flowchart

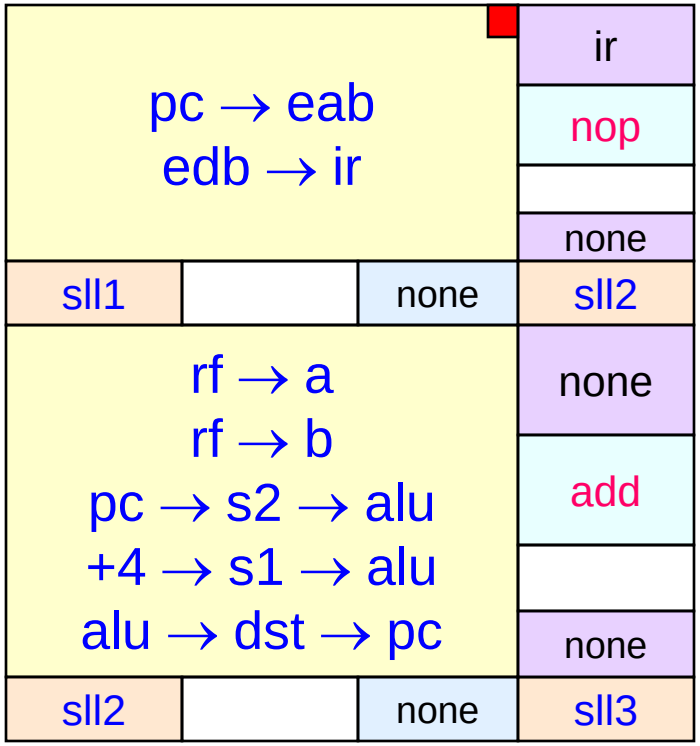
SUB.I Rd, Rs1, Imm

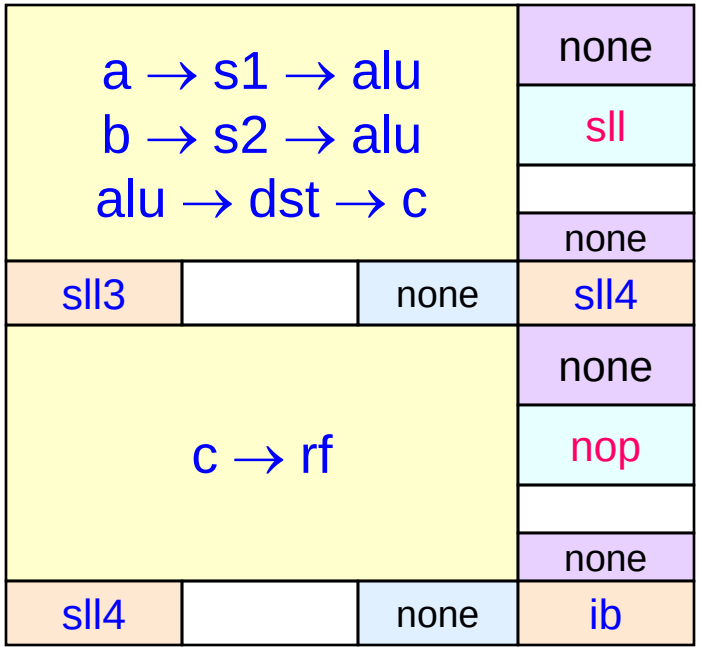




Level 2 Hardware Flowchart

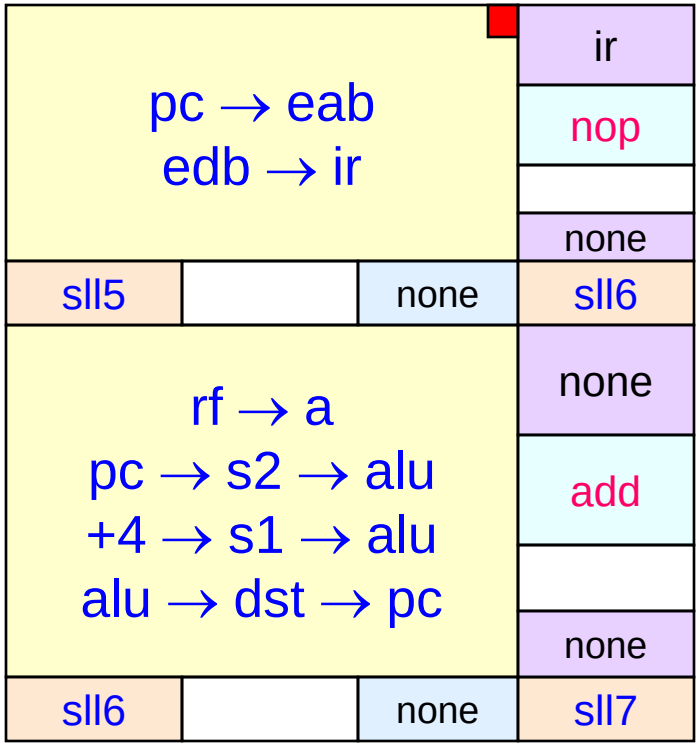
SLL Rd, Rs1, Rs2

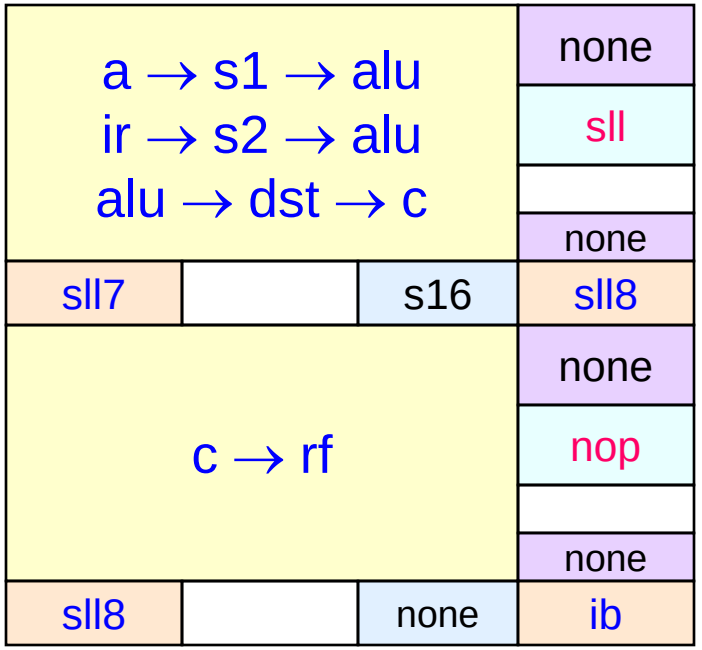




Level 2 Hardware Flowchart

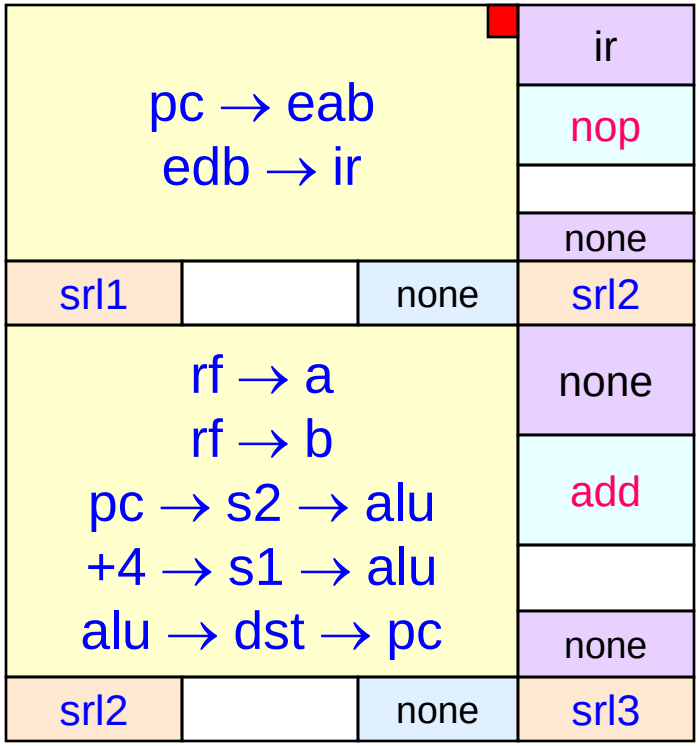
SLL.I Rd, Rs1, Imm

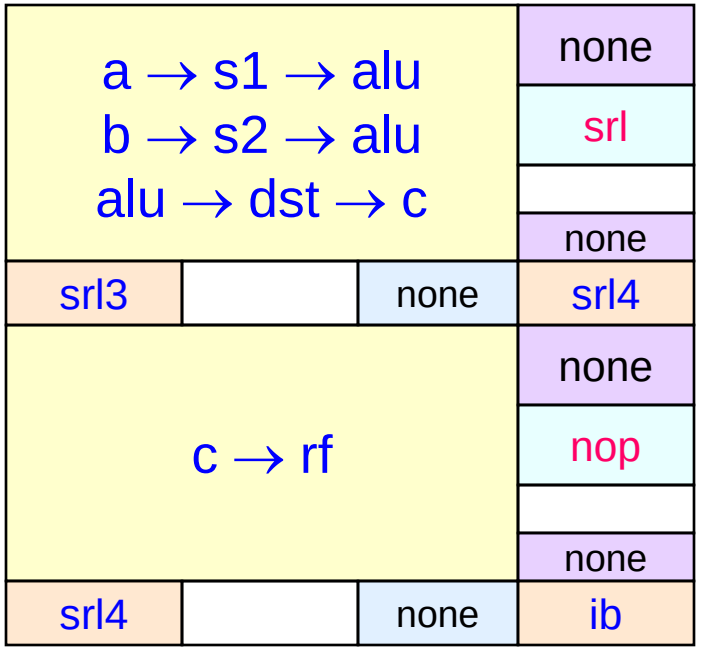




Level 2 Hardware Flowchart

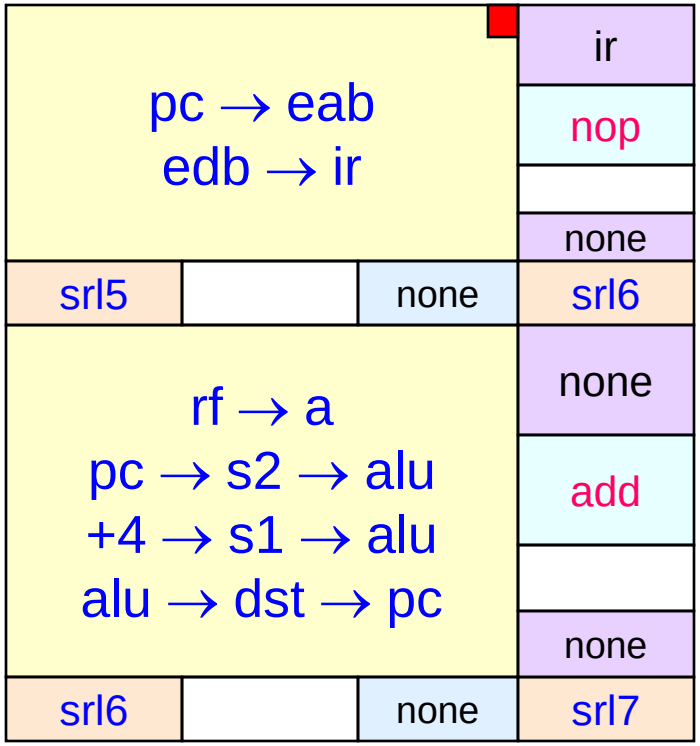
SRL Rd, Rs1, Rs2

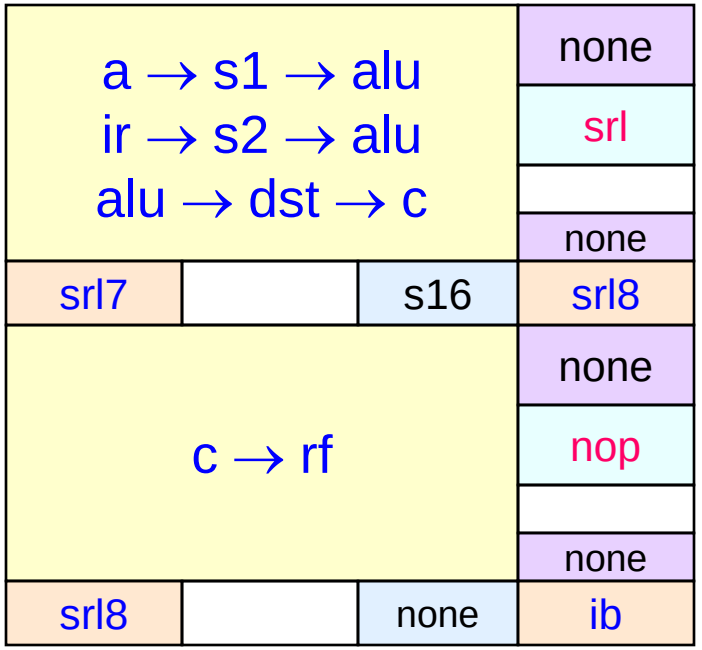




Level 2 Hardware Flowchart

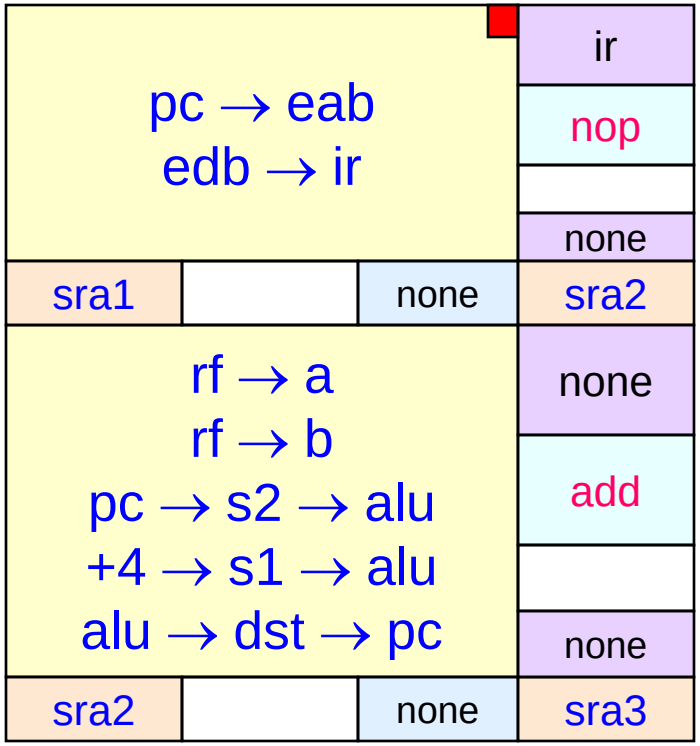
SRL.I Rd, Rs1, Imm

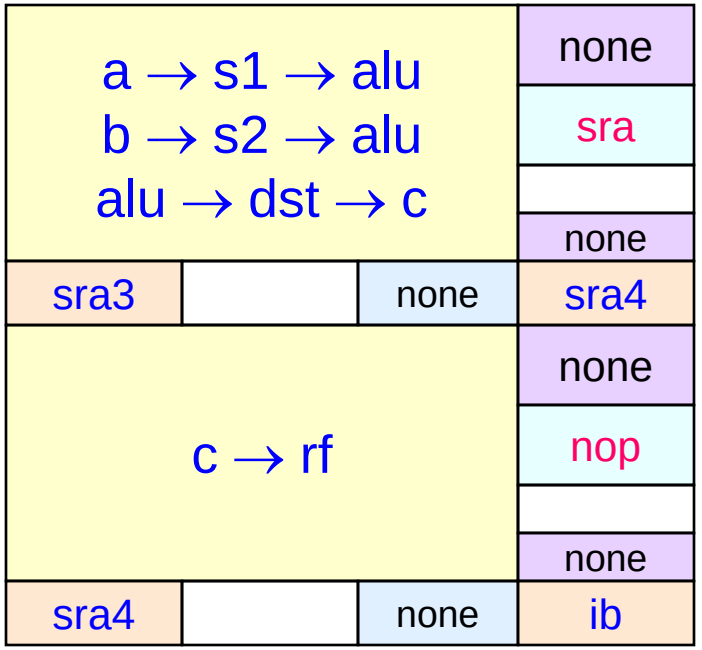




Level 2 Hardware Flowchart

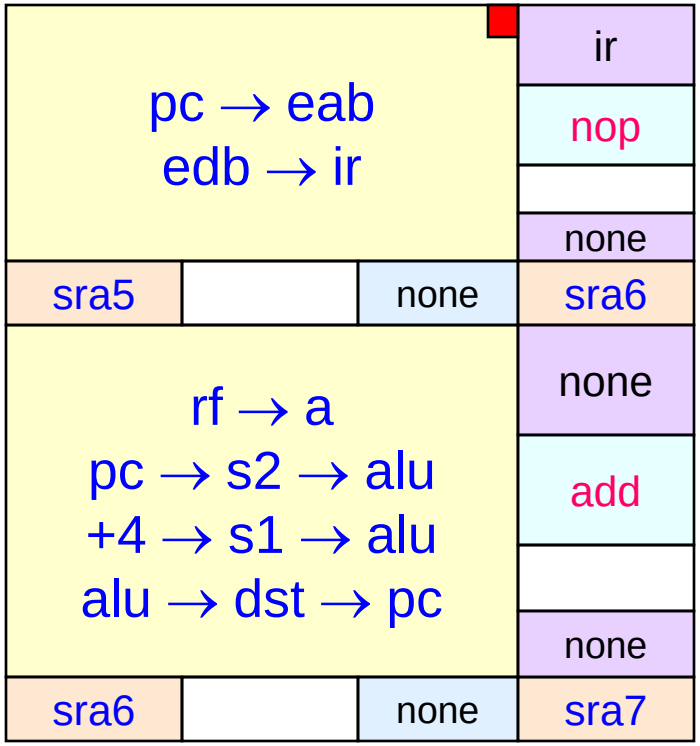
SRA Rd, Rs1, Rs2

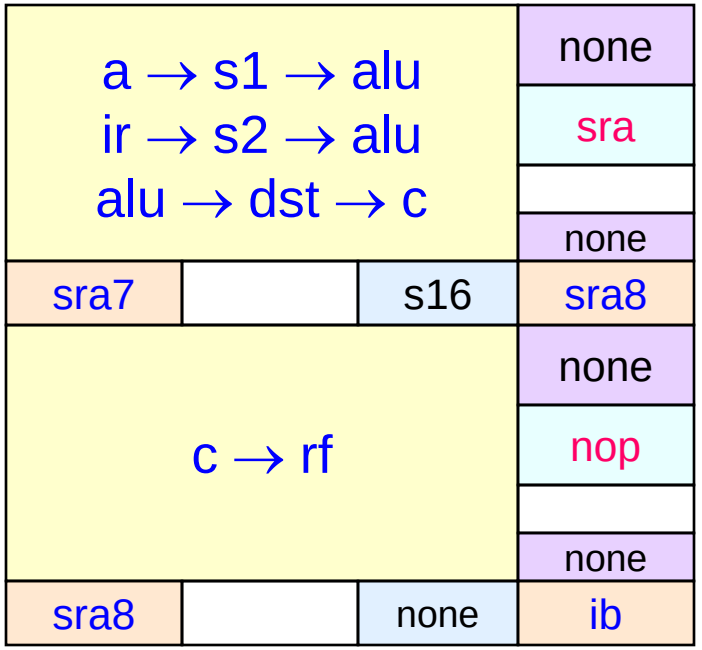




Level 2 Hardware Flowchart

SRA.I Rd, Rs1, Imm

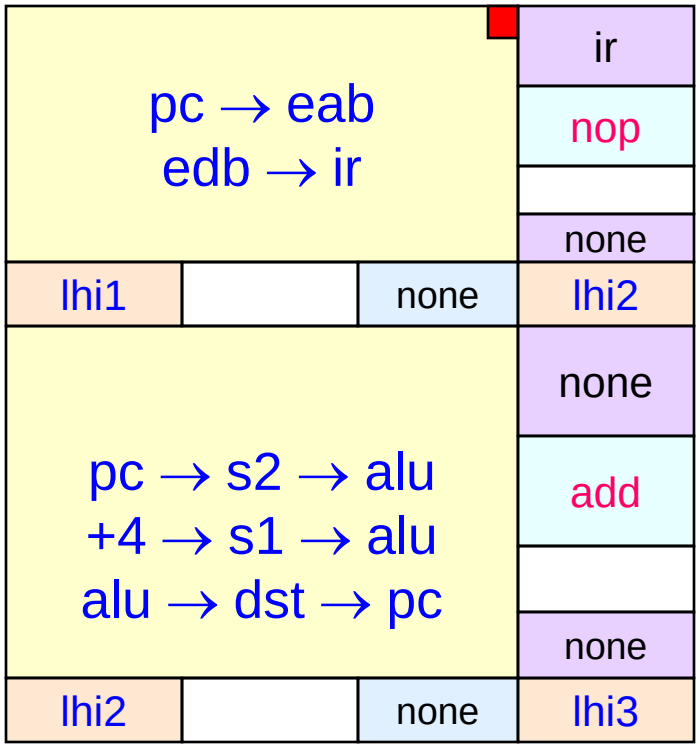


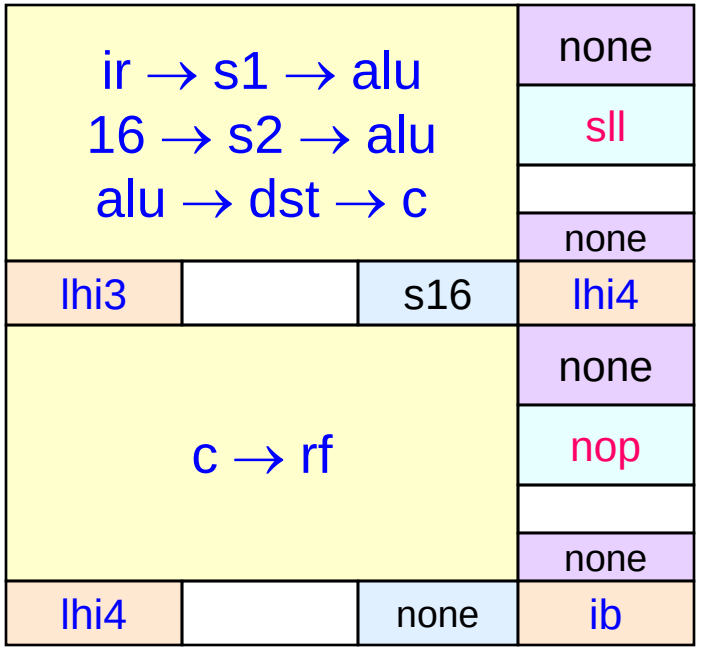


Level 2 Hardware Flowchart

LHI Rd, Imm

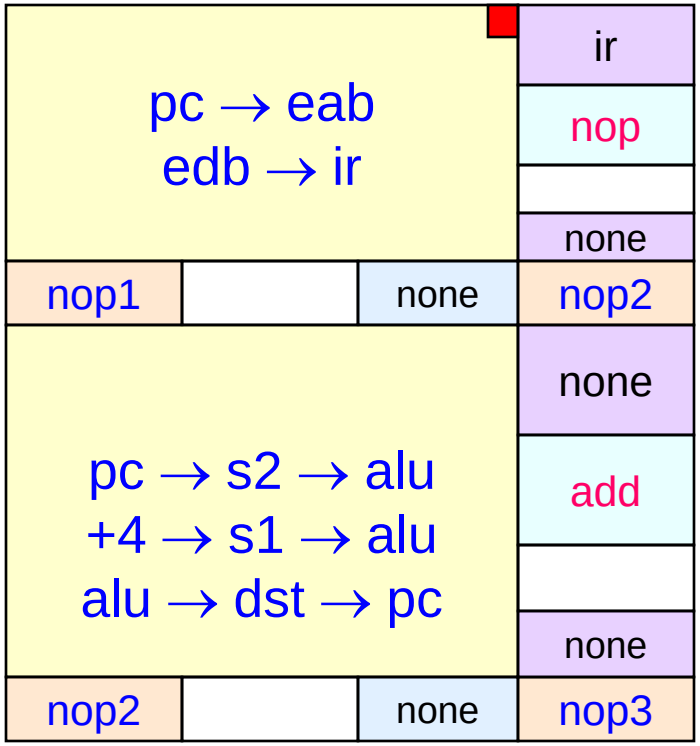
Rd(0:15) <- Imm; Rd(16:31) <- hex0000





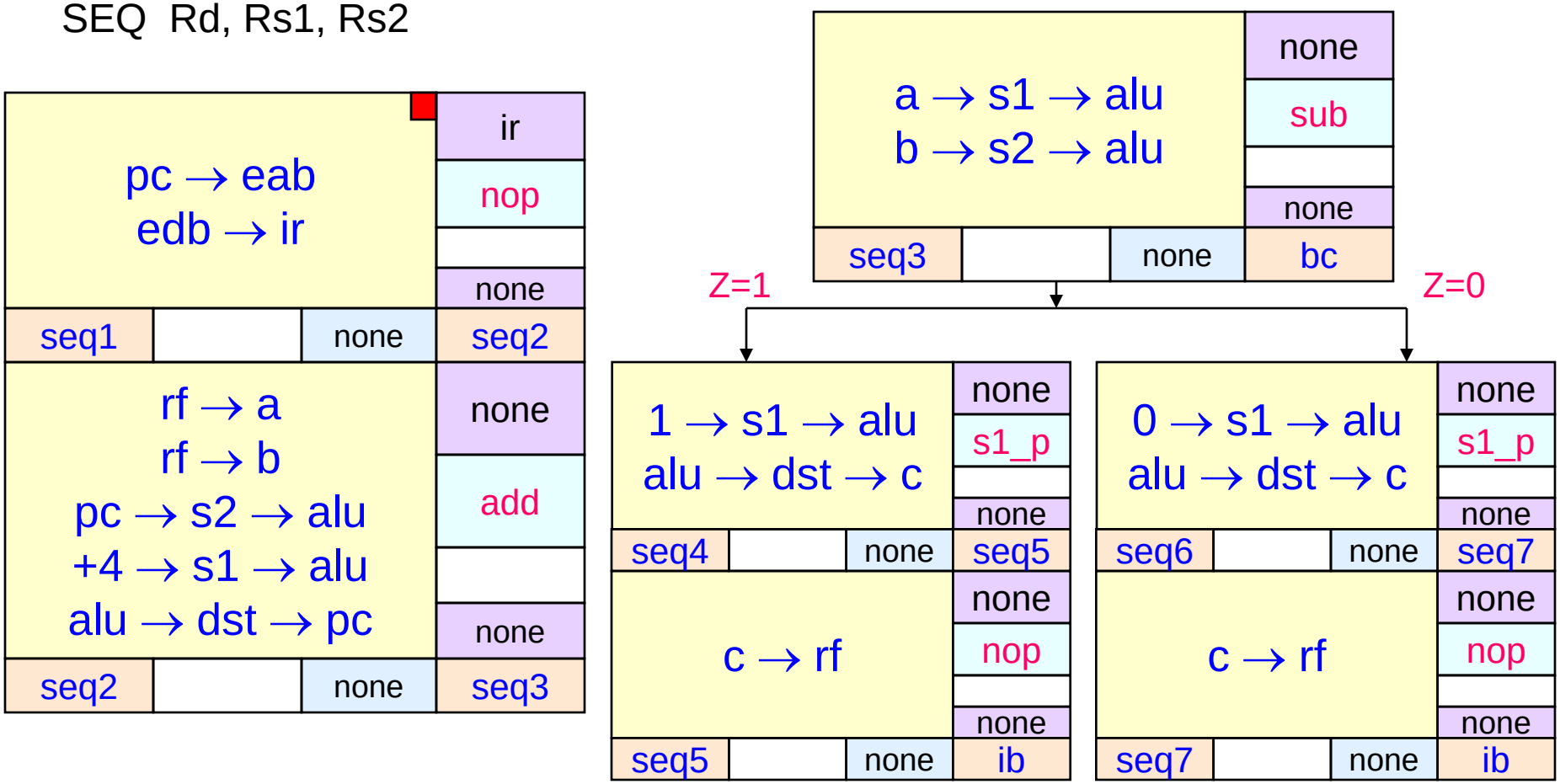
Level 2 Hardware Flowchart

NOP



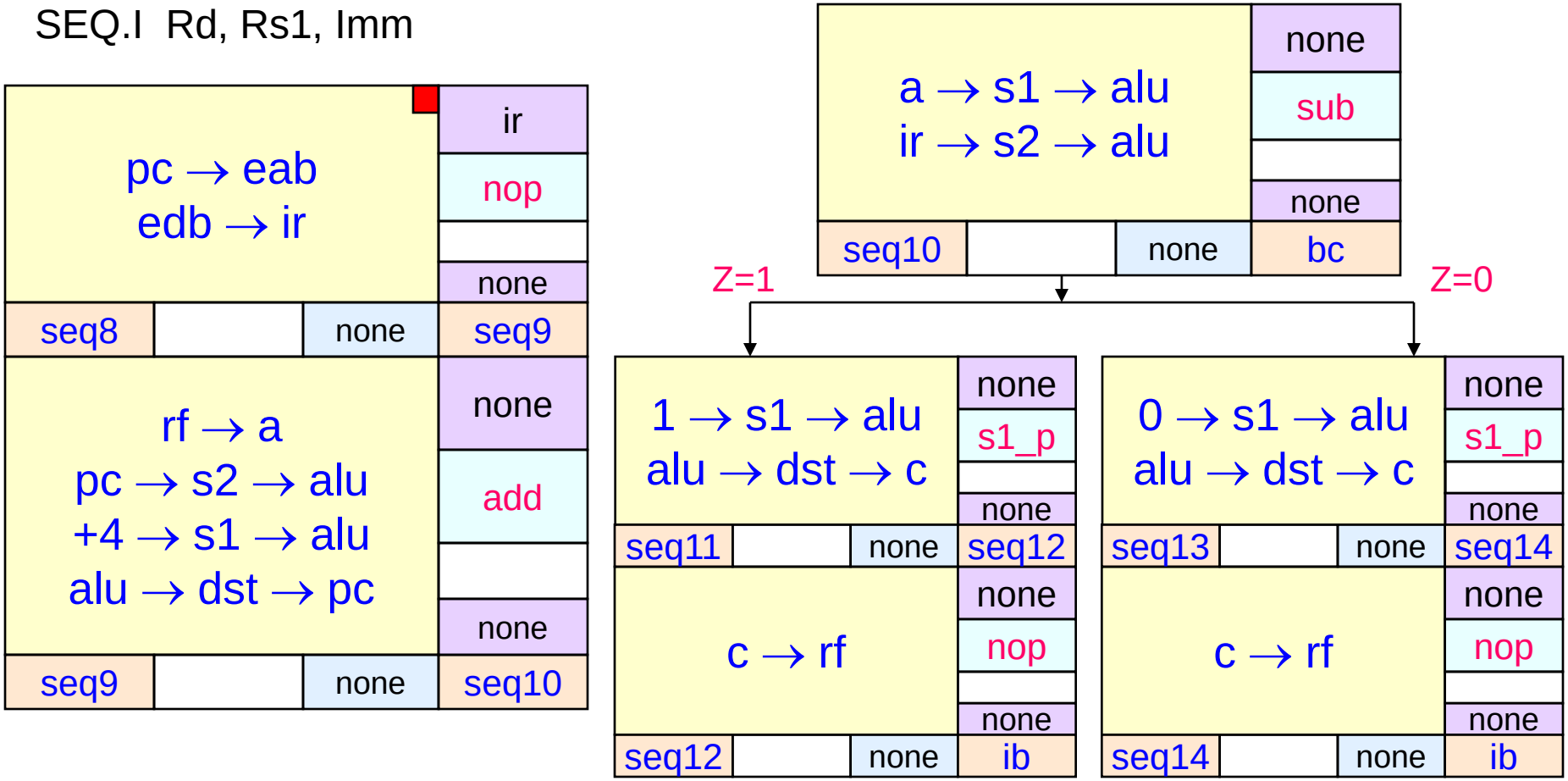
Level 2 Hardware Flowchart

SEQ Rd, Rs1, Rs2



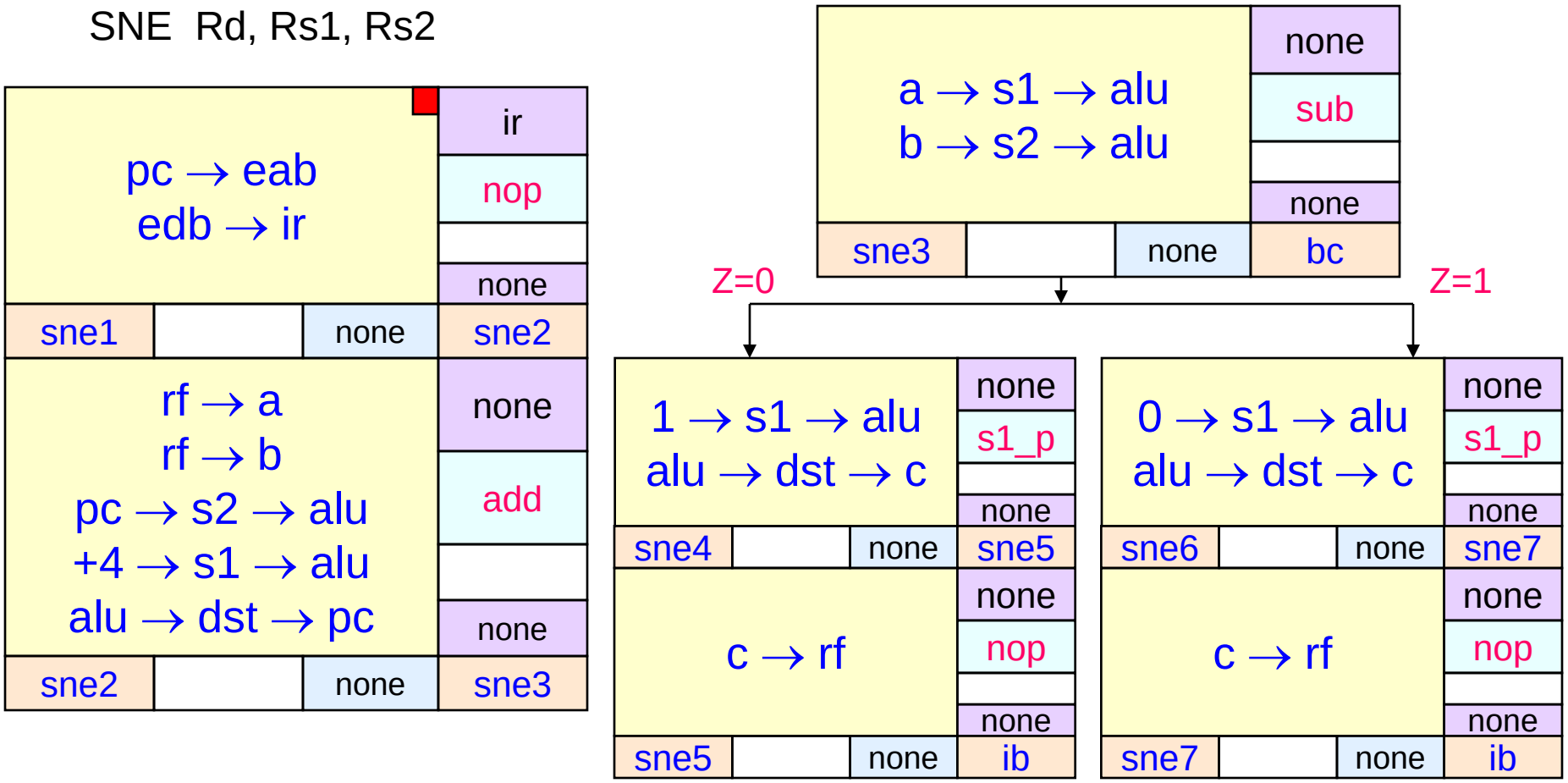
Level 2 Hardware Flowchart

SEQ.I Rd, Rs1, Imm



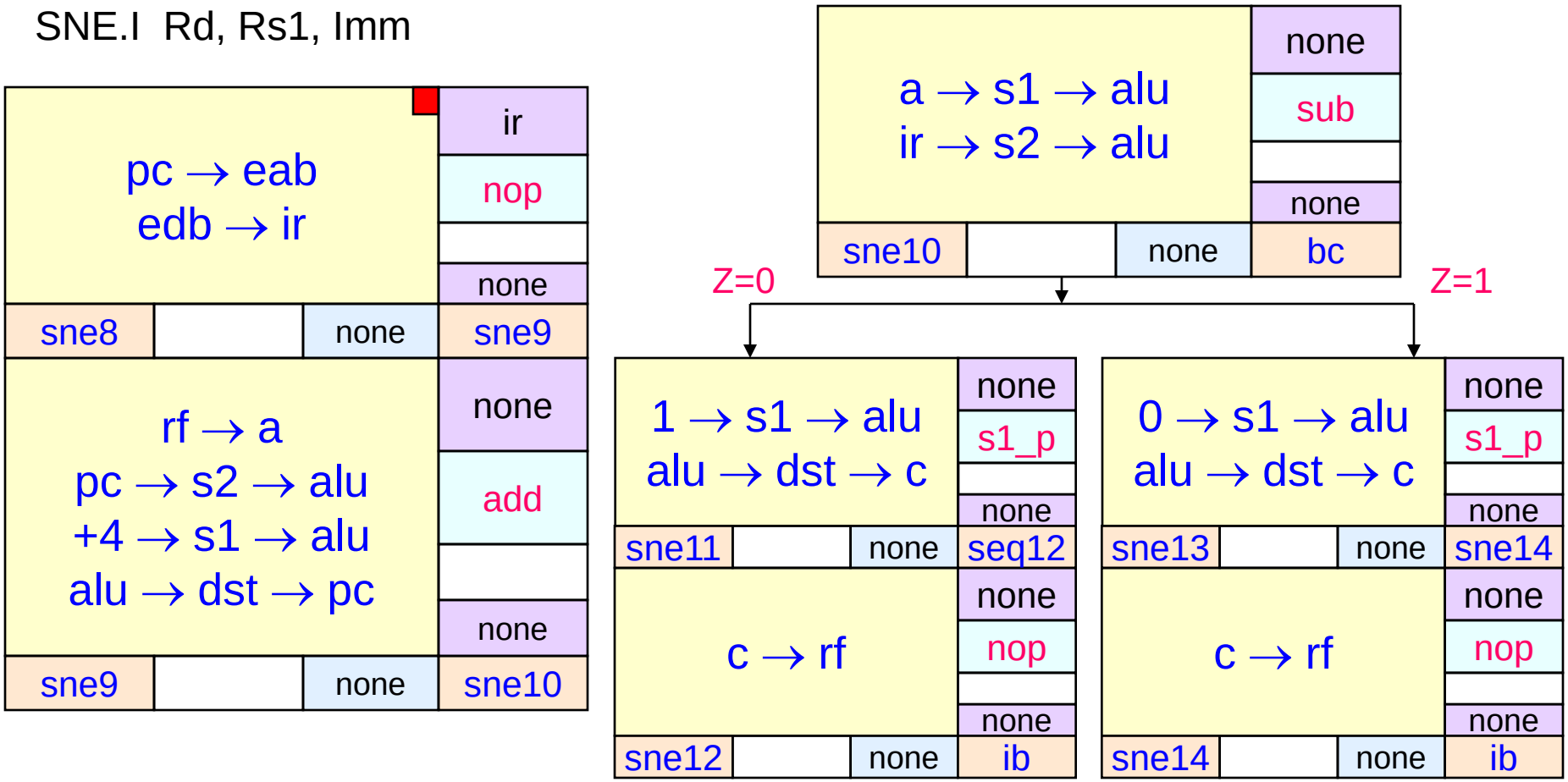
Level 2 Hardware Flowchart

SNE Rd, Rs1, Rs2



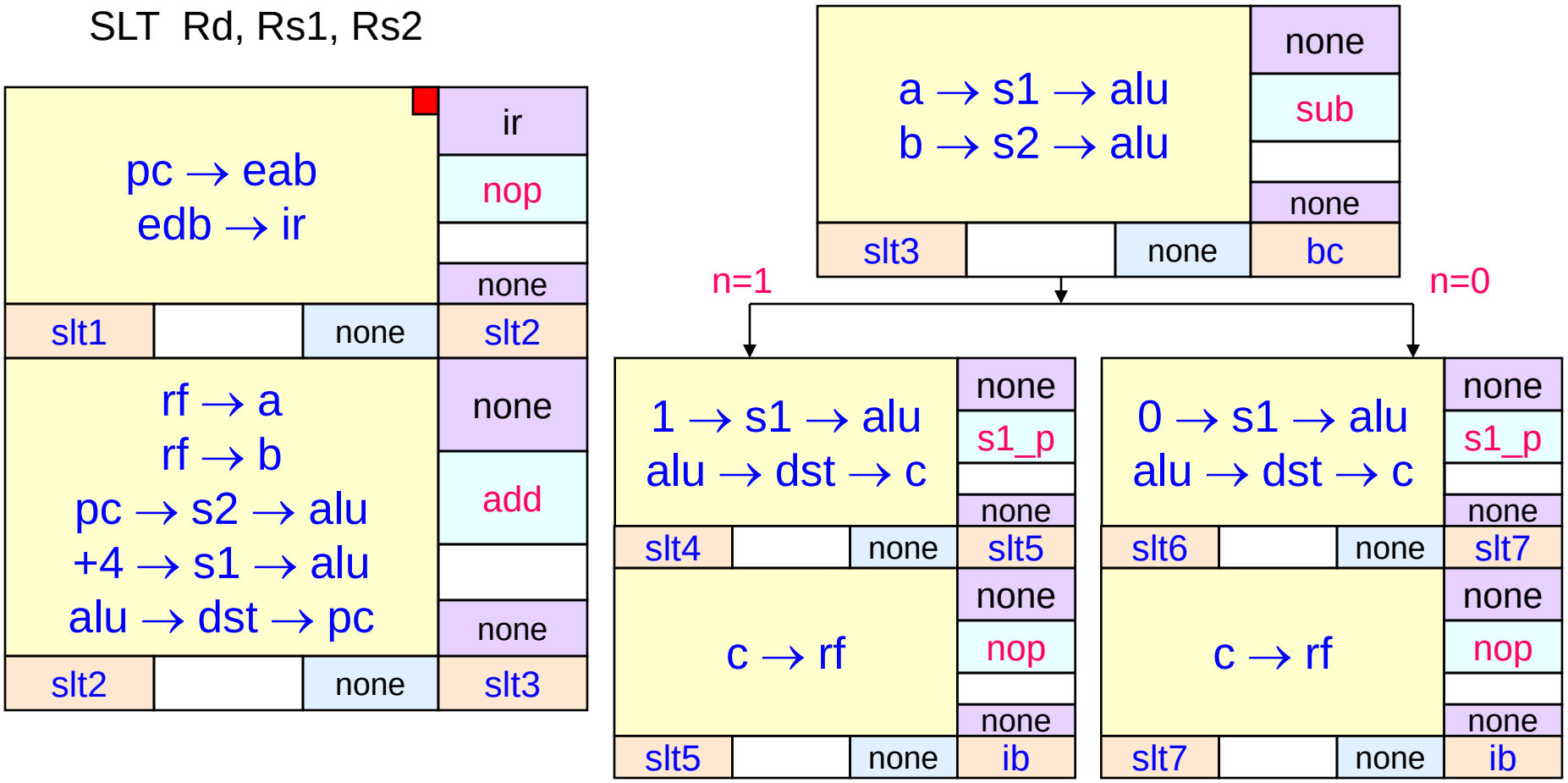
Level 2 Hardware Flowchart

SNE.I Rd, Rs1, Imm



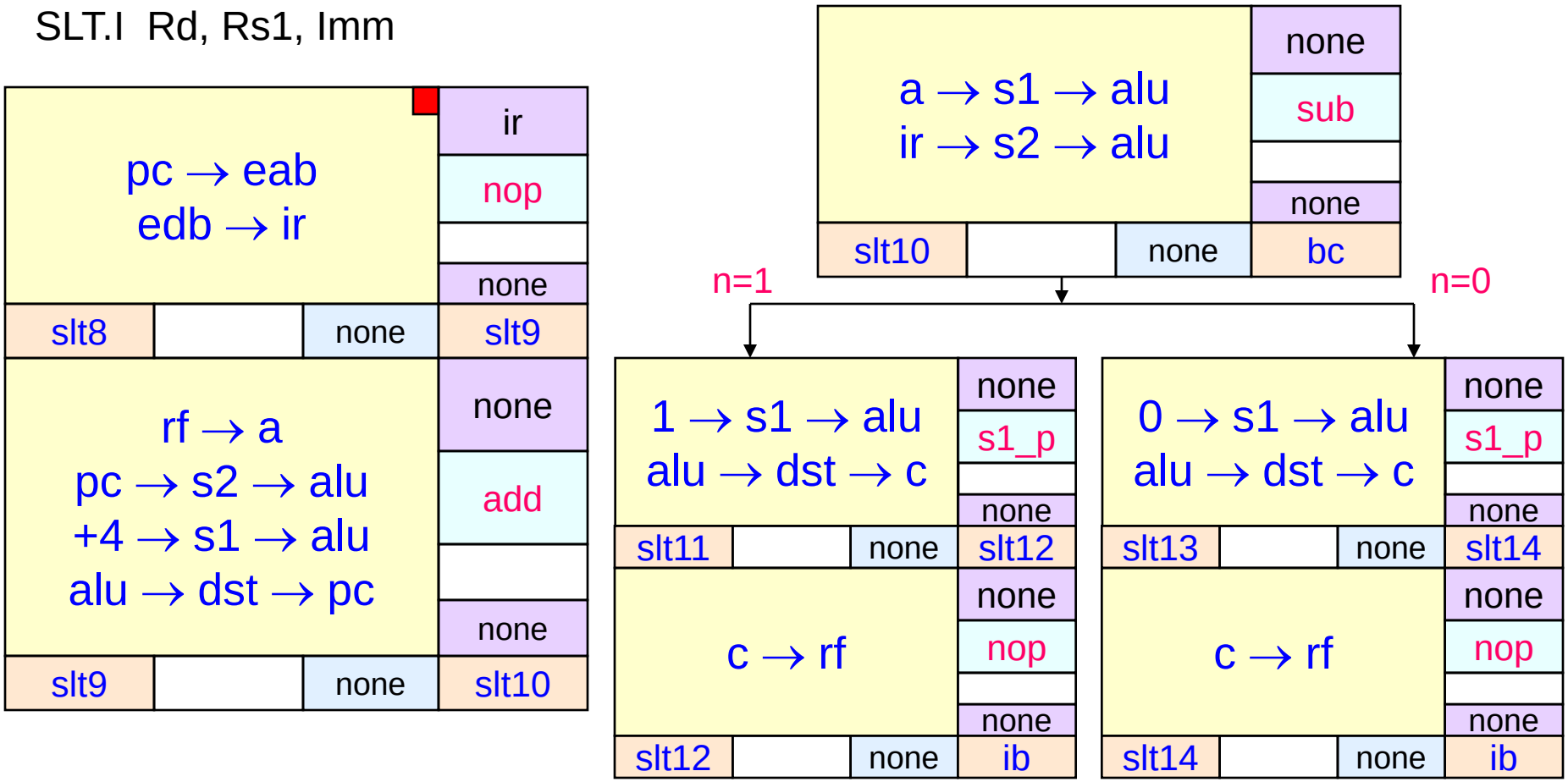
Level 2 Hardware Flowchart

SLT Rd, Rs1, Rs2



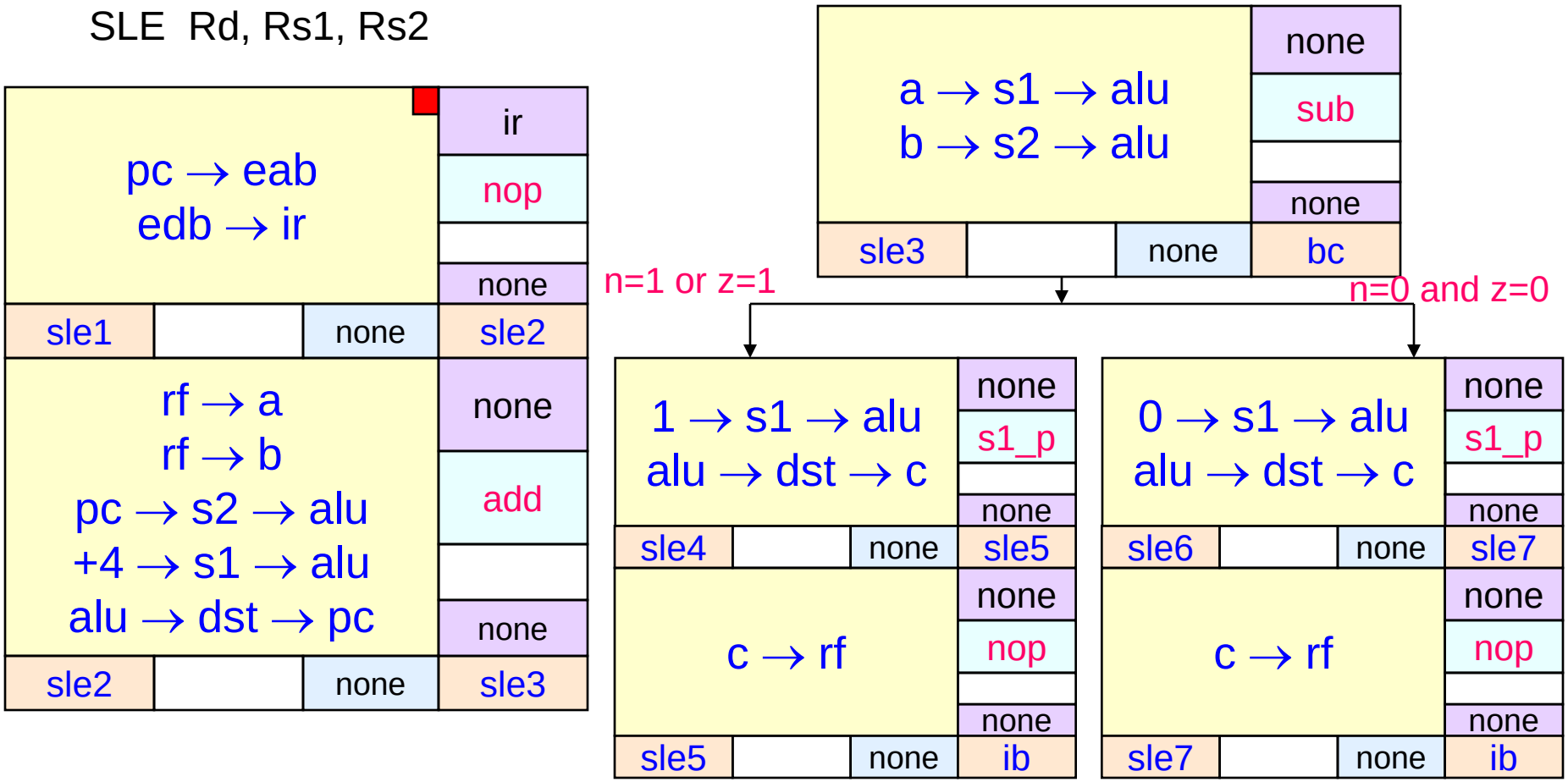
Level 2 Hardware Flowchart

SLT.I Rd, Rs1, Imm



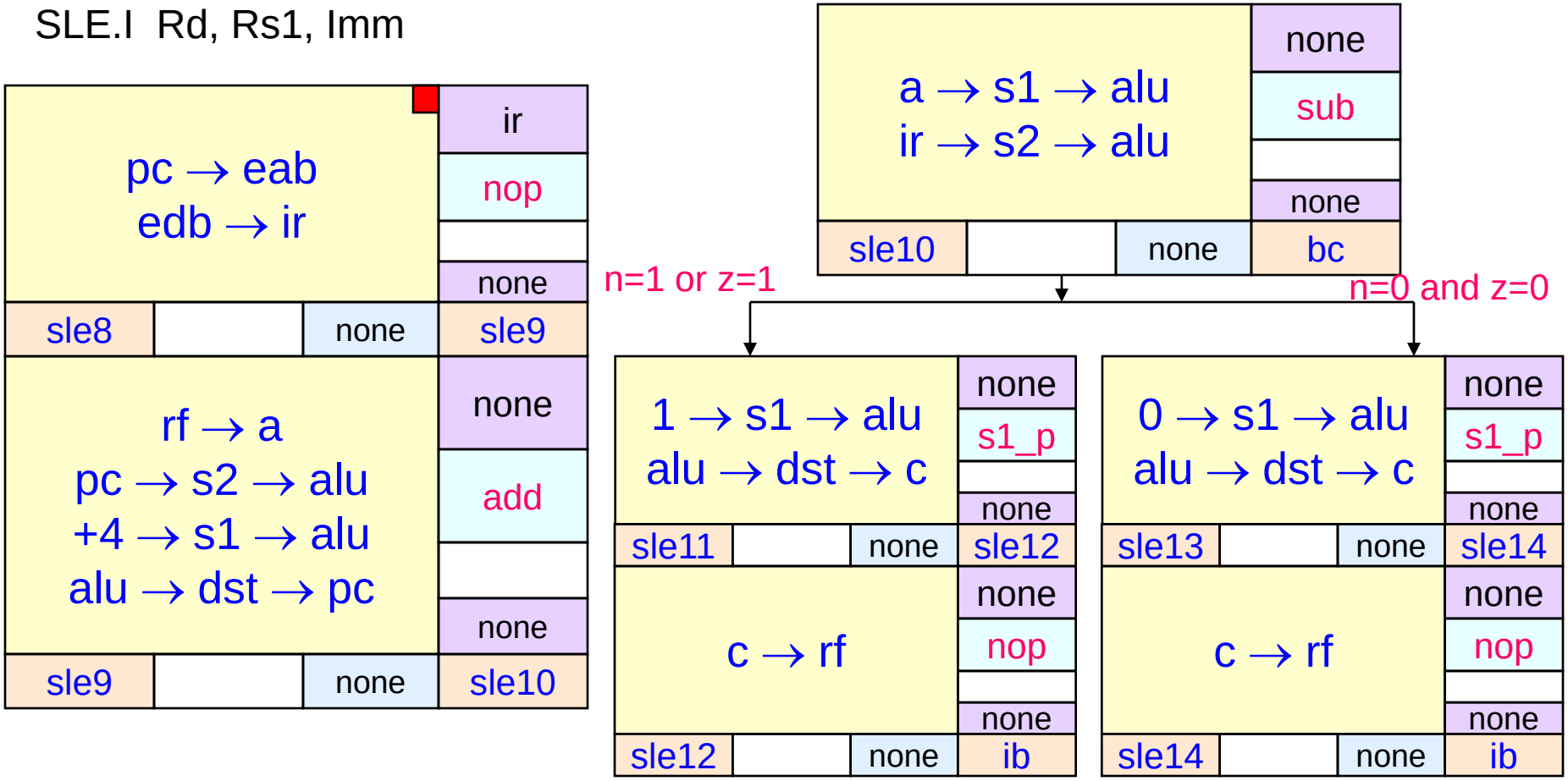
Level 2 Hardware Flowchart

SLE Rd, Rs1, Rs2



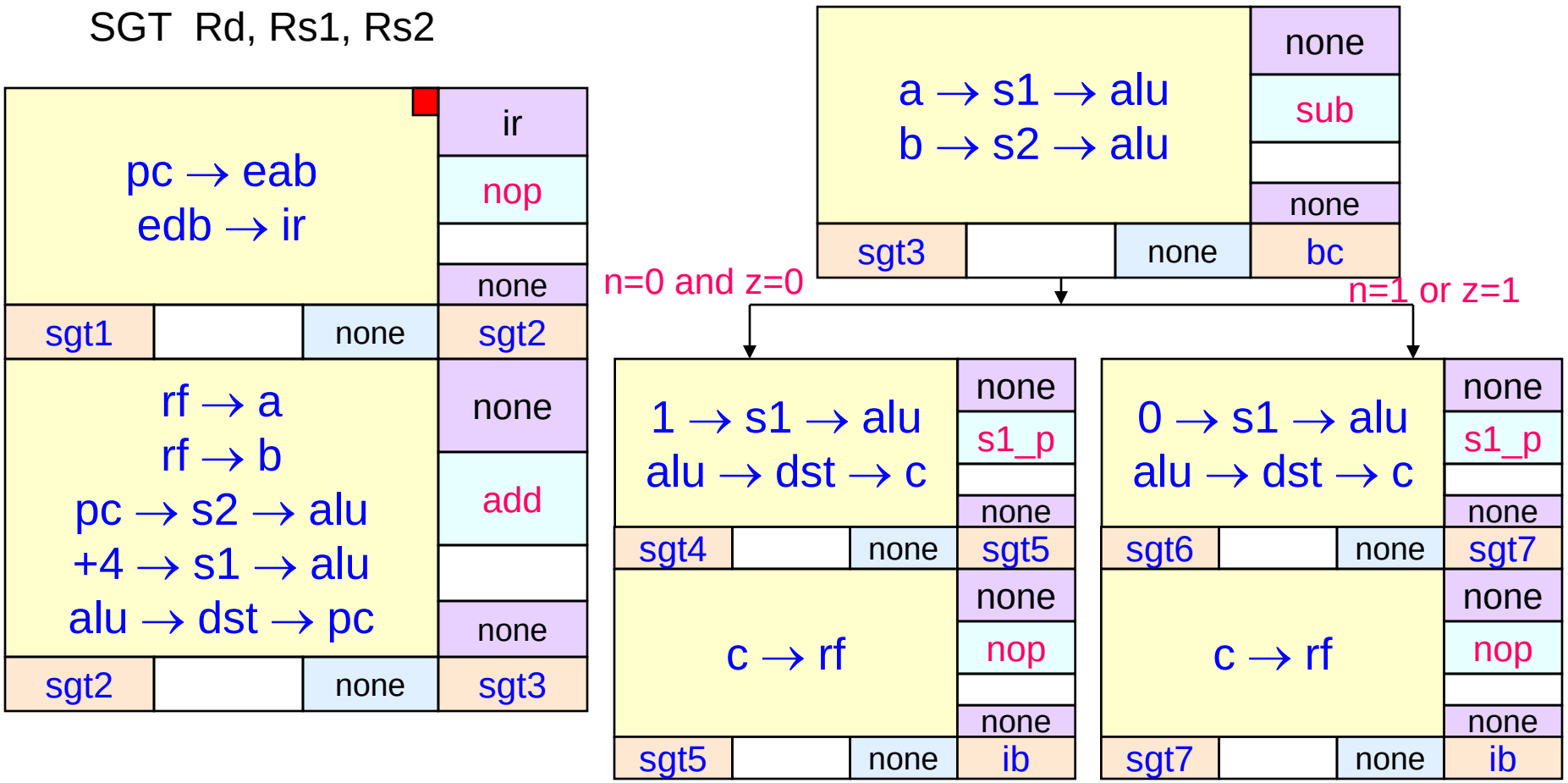
Level 2 Hardware Flowchart

SLE.I Rd, Rs1, Imm



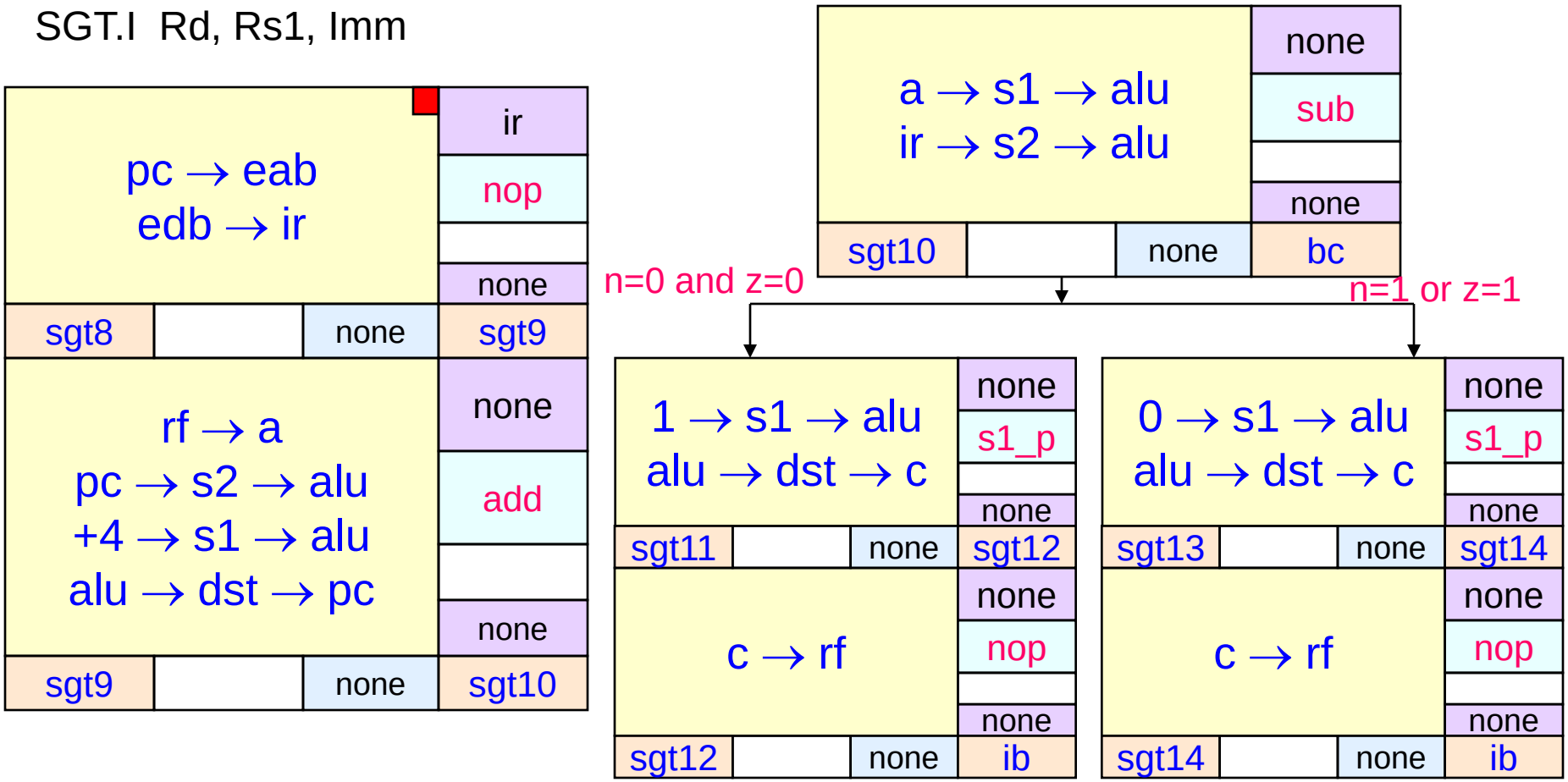
Level 2 Hardware Flowchart

SGT Rd, Rs1, Rs2



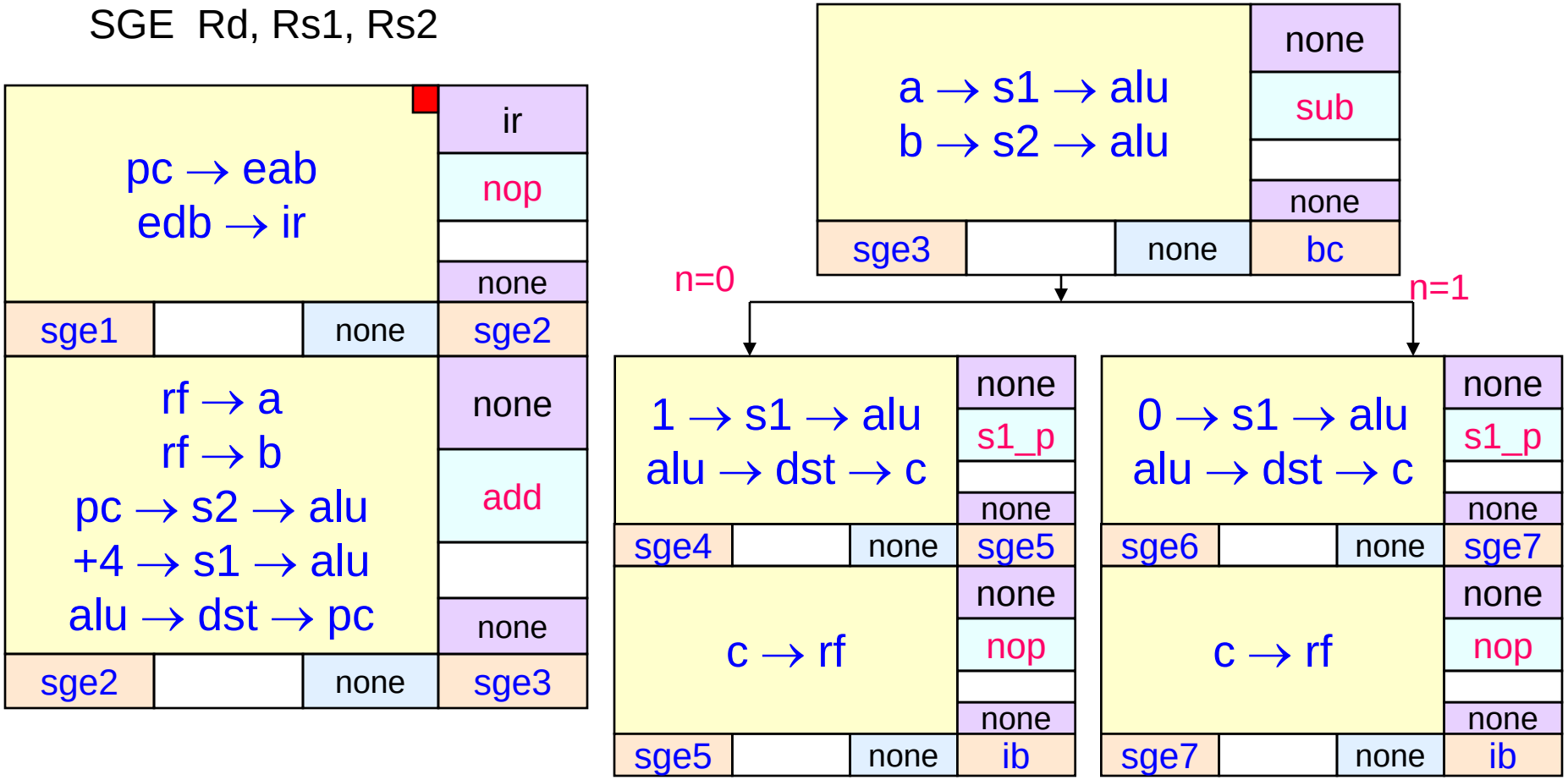
Level 2 Hardware Flowchart

SGT.I Rd, Rs1, Imm



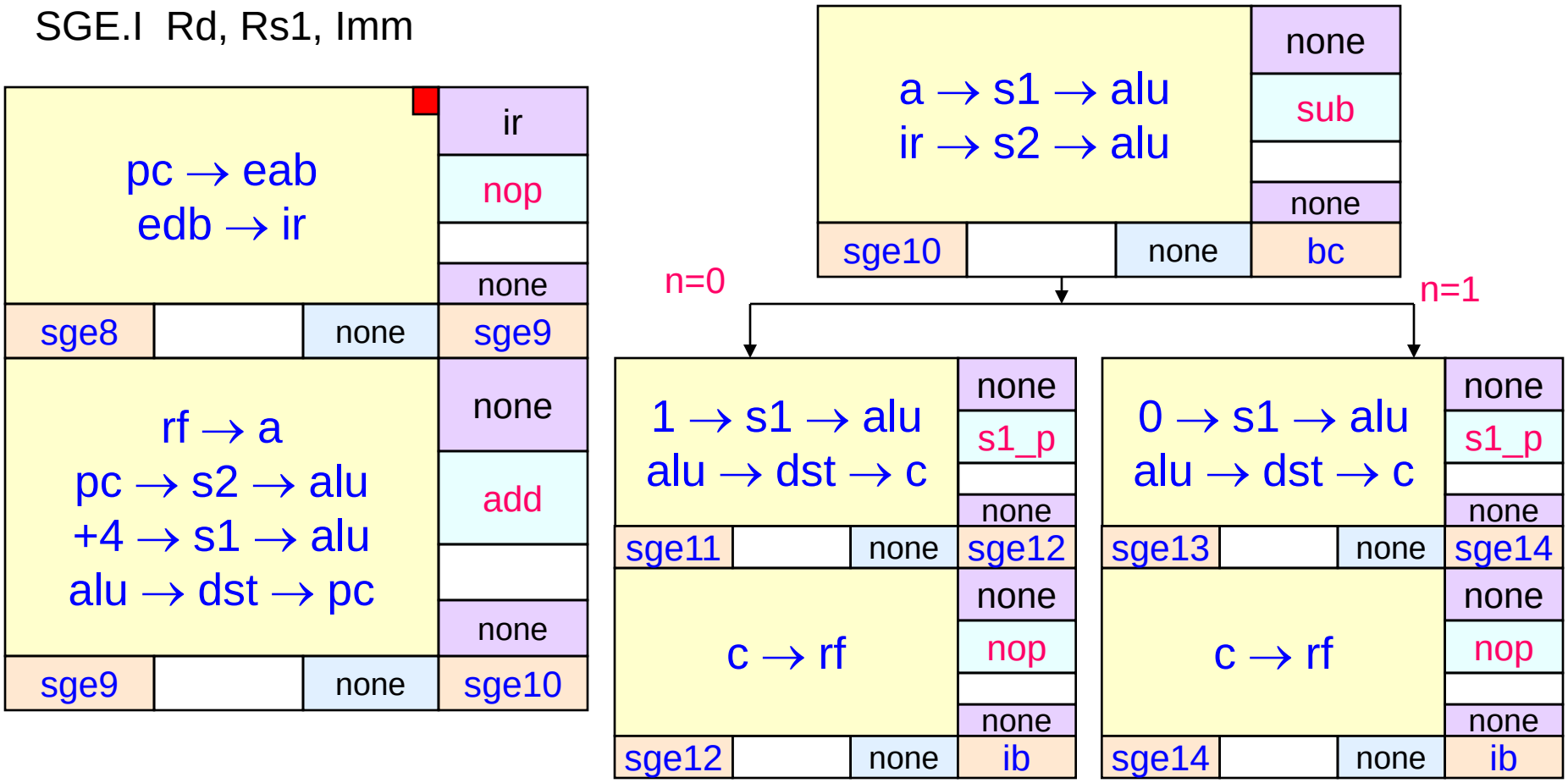
Level 2 Hardware Flowchart

SGE Rd, Rs1, Rs2



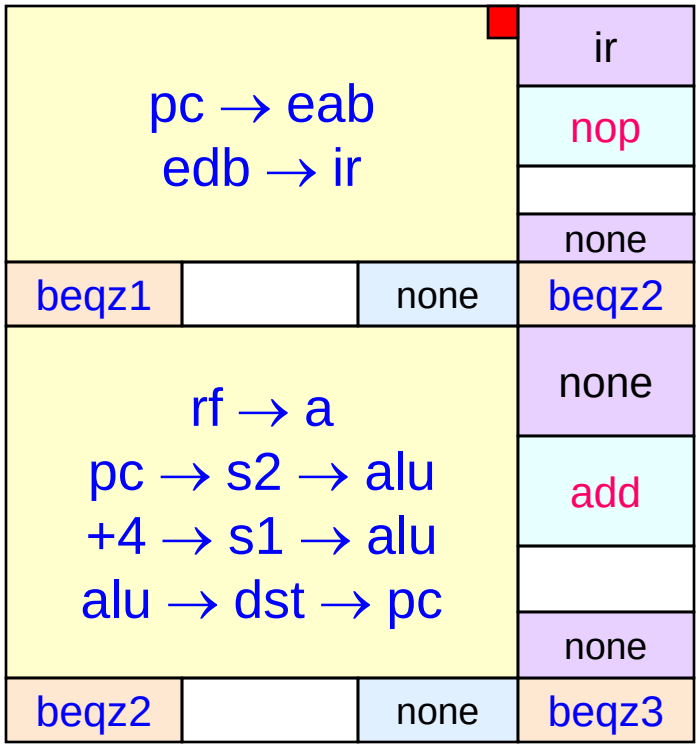
Level 2 Hardware Flowchart

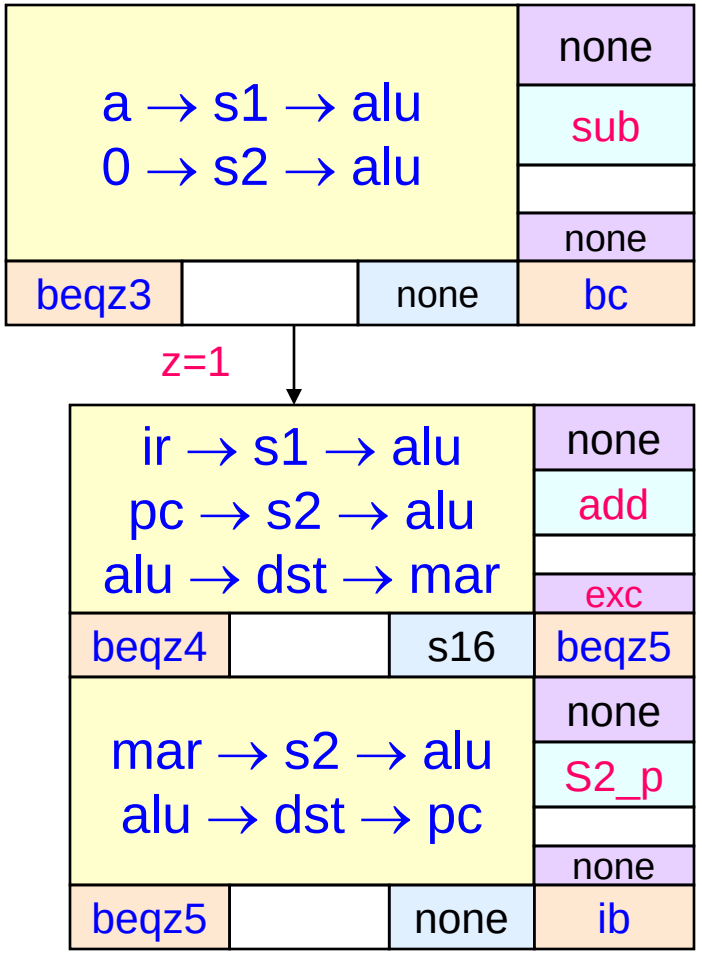
SGE.I Rd, Rs1, Imm



Level 2 Hardware Flowchart

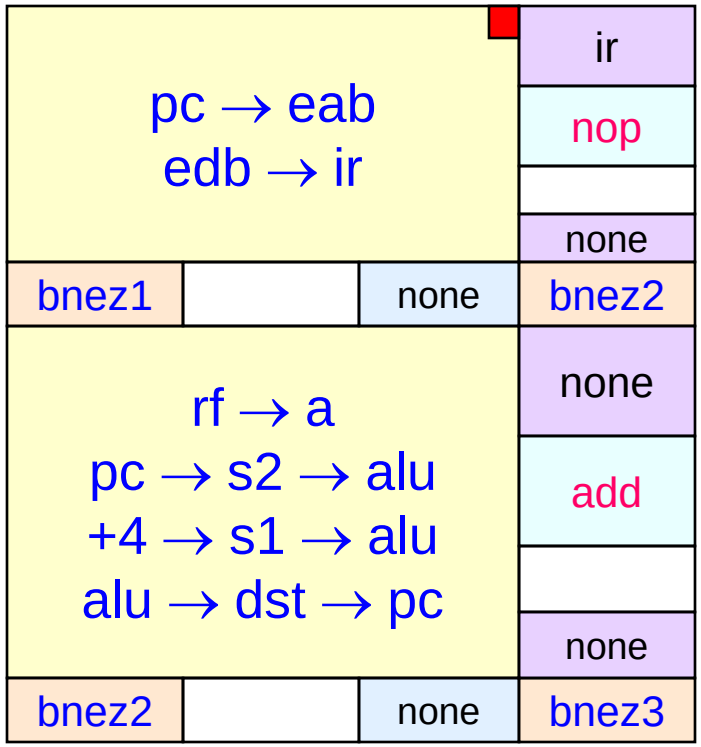
BEQZ Rs, Label

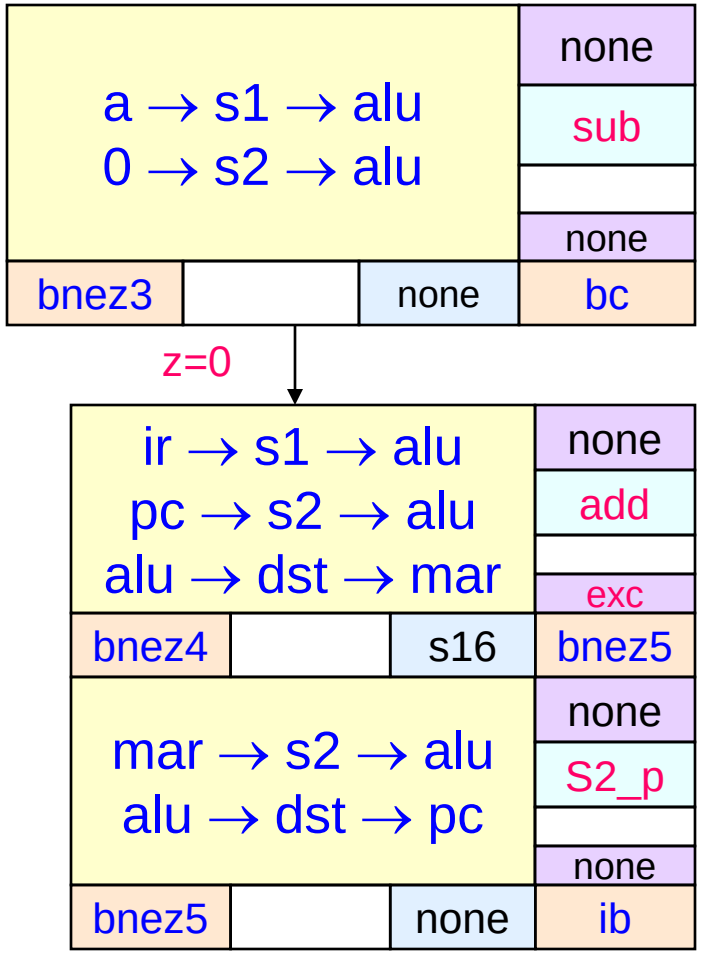




Level 2 Hardware Flowchart

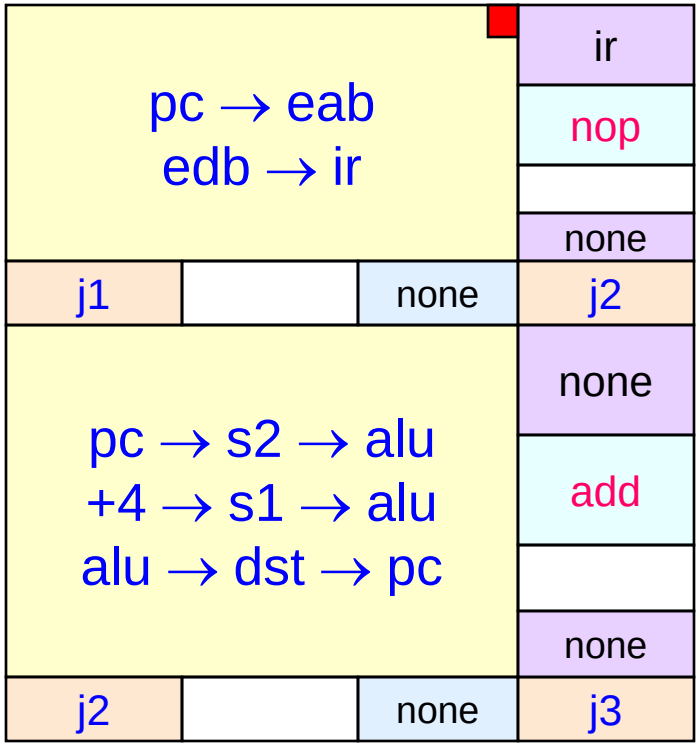
BNEZ Rs, Label

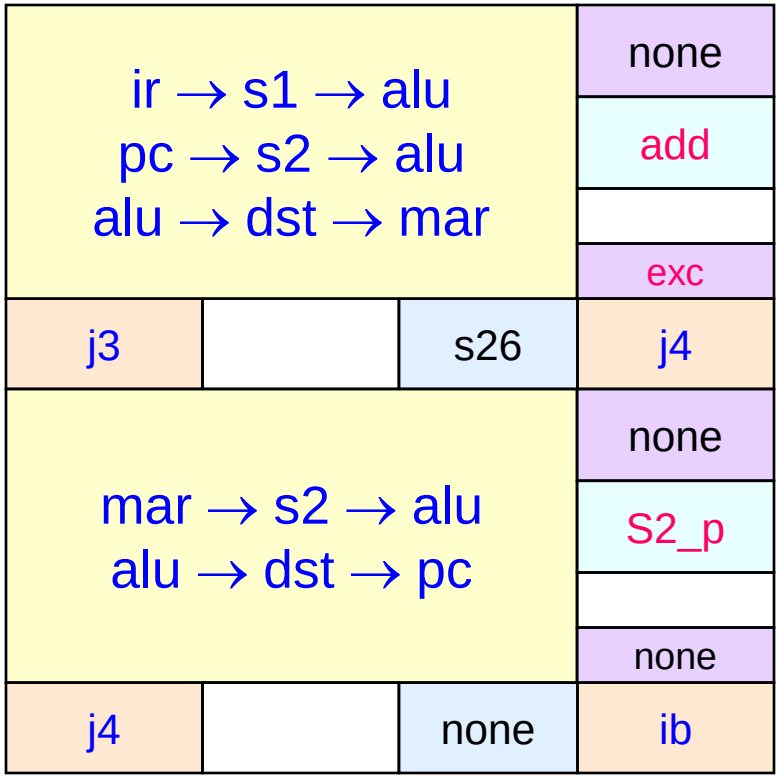




Level 2 Hardware Flowchart

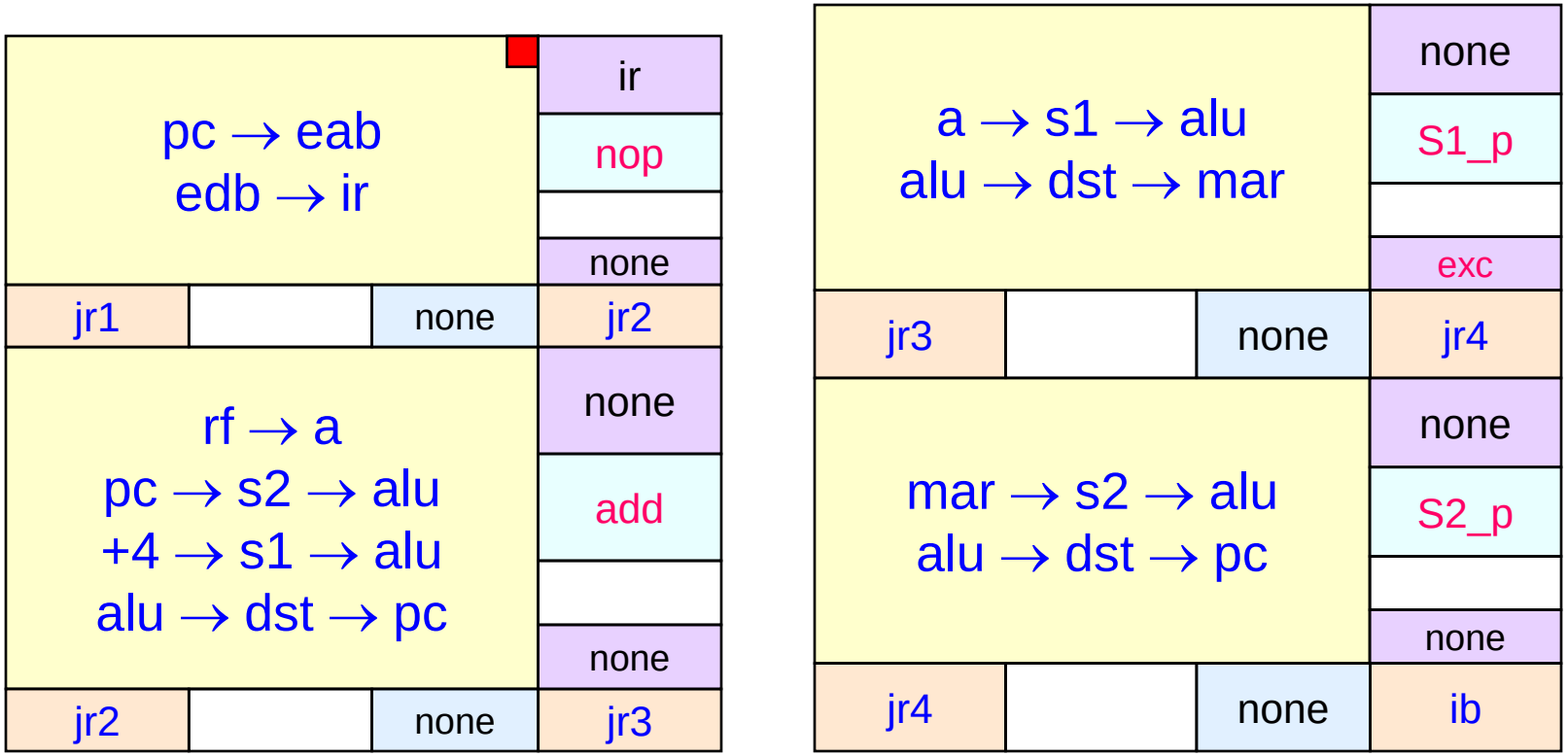
J Label



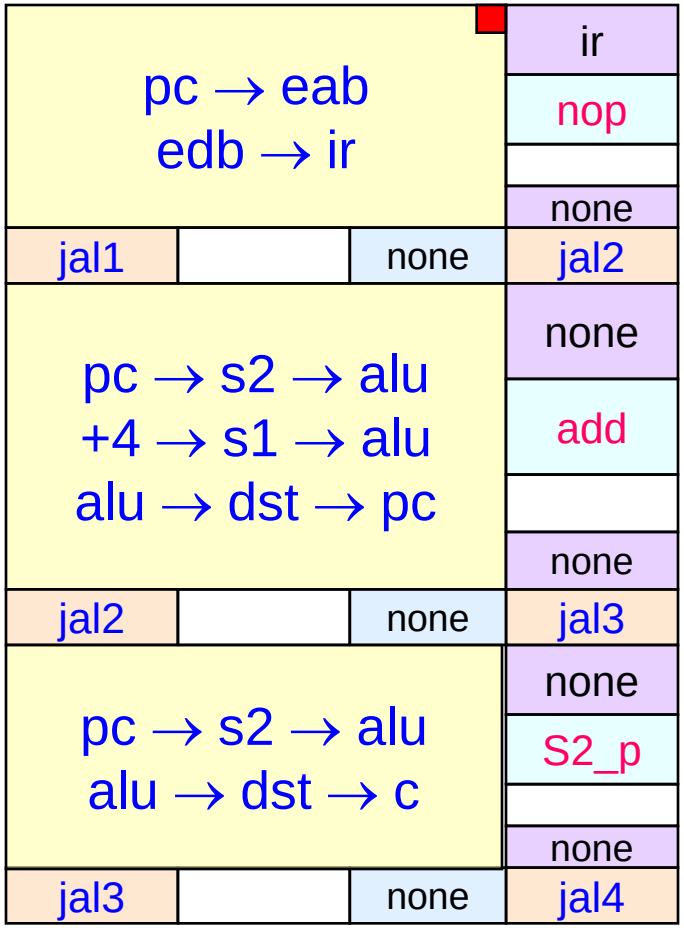


Level 2 Hardware Flowchart

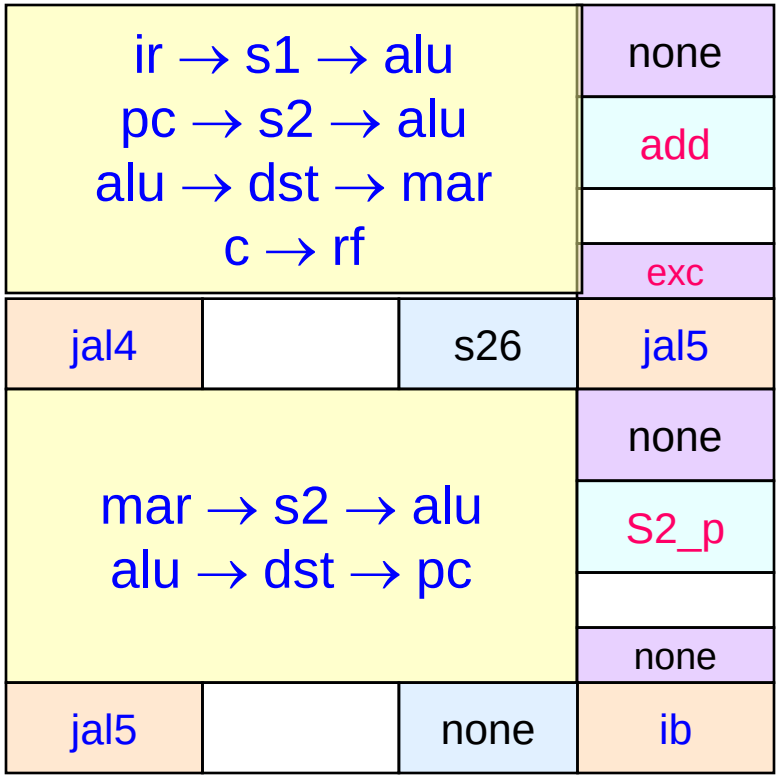
JR Rs



Level 2 Hardware Flowchart

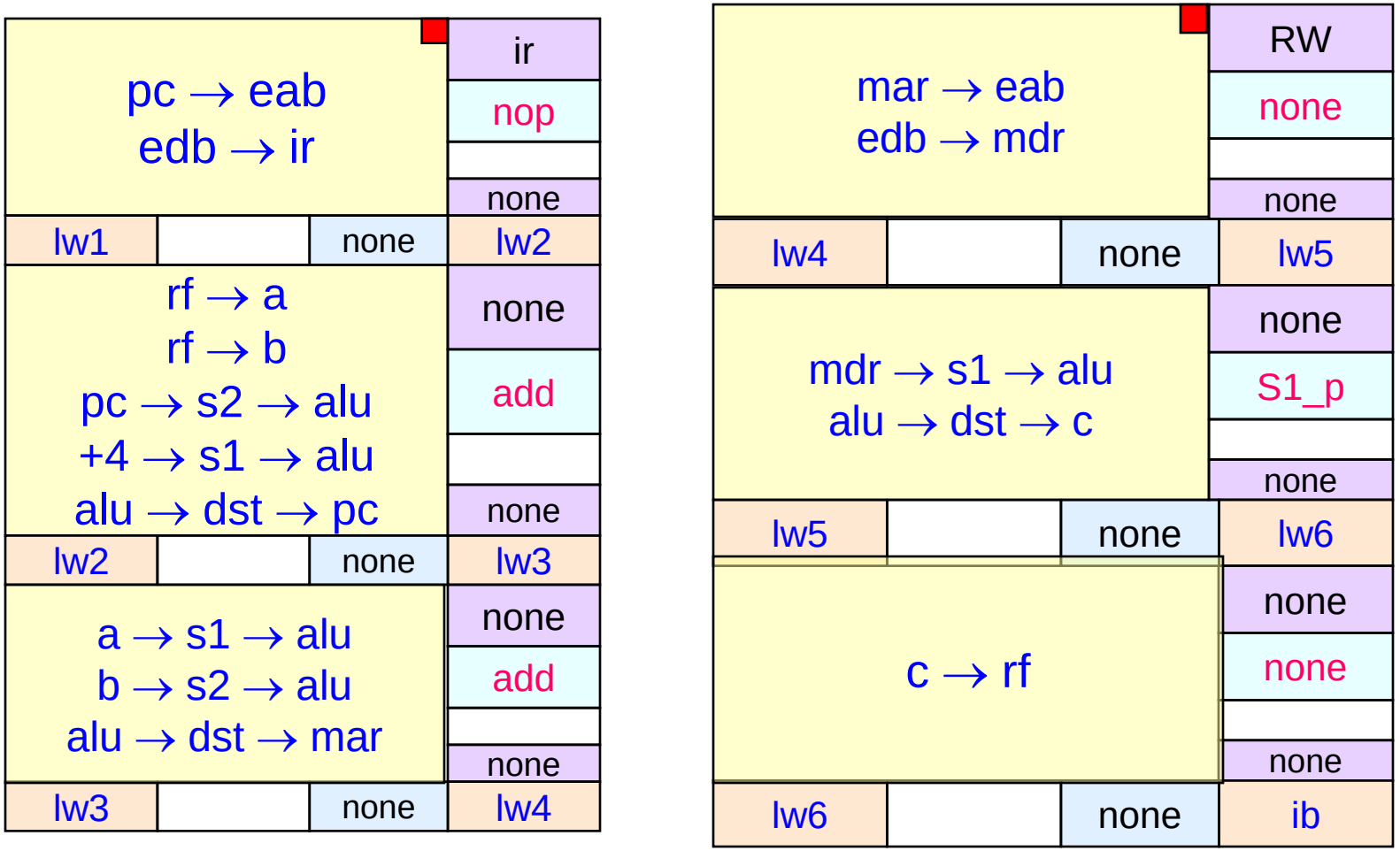


JAL Label



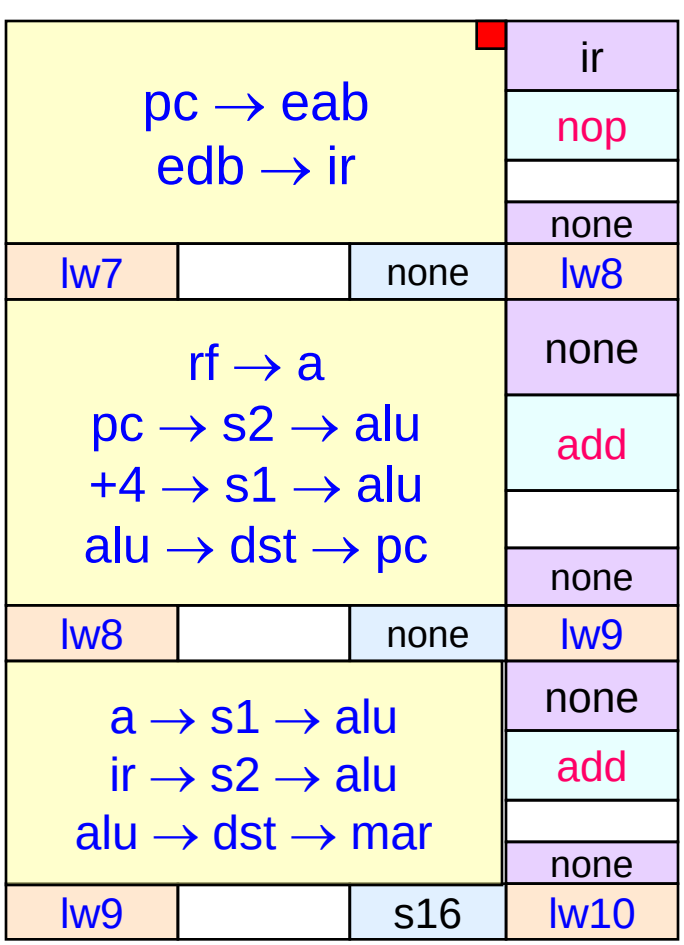
Level 2 Hardware Flowchart

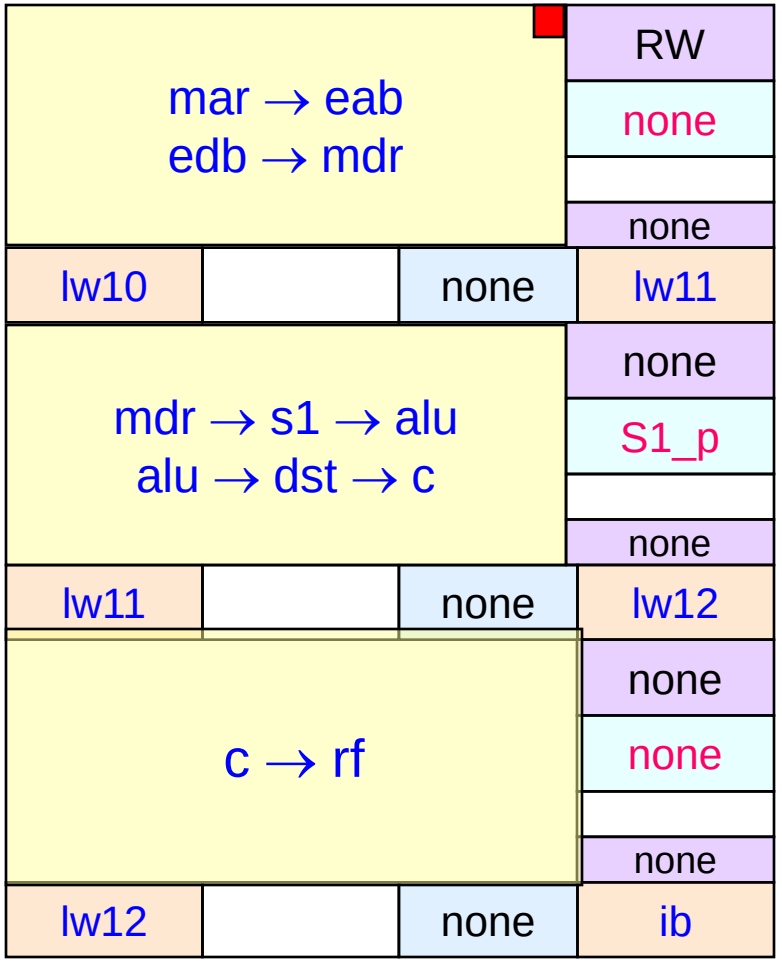
LW Rd, Rs2(Rs1)



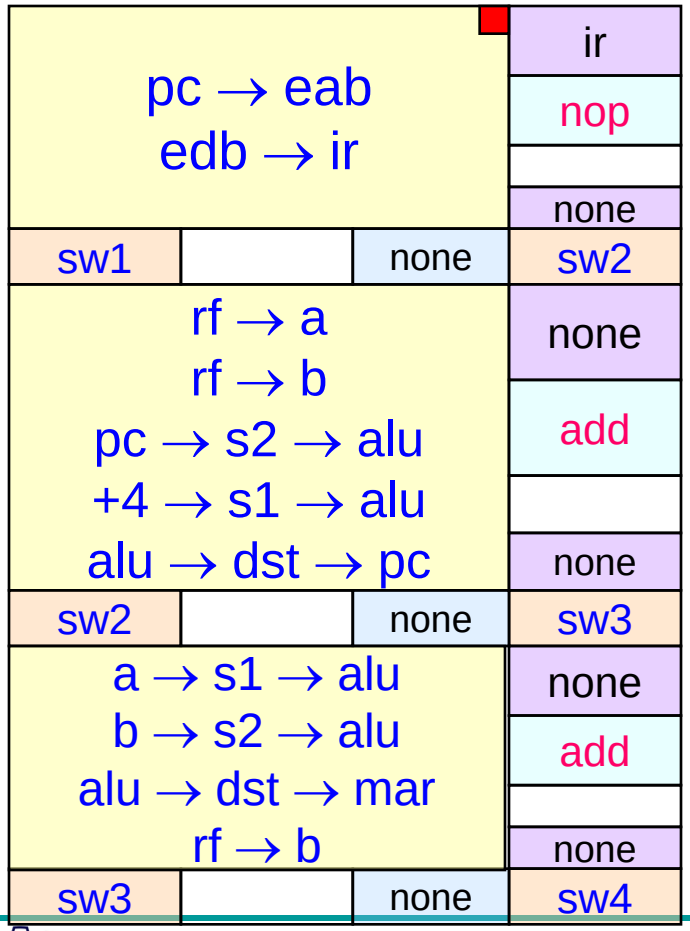
Level 2 Hardware Flowchart

LW.I Rd, Imm(Rs1)

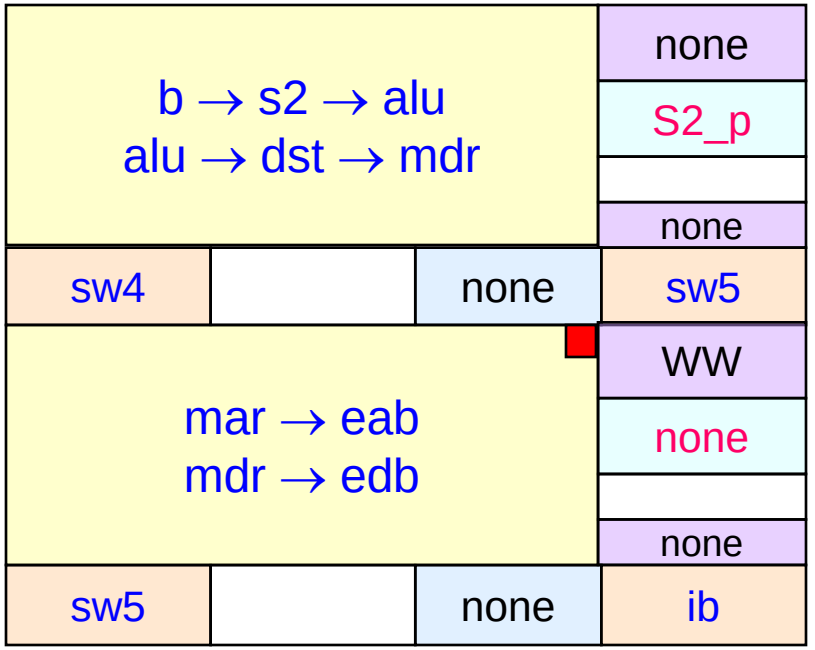




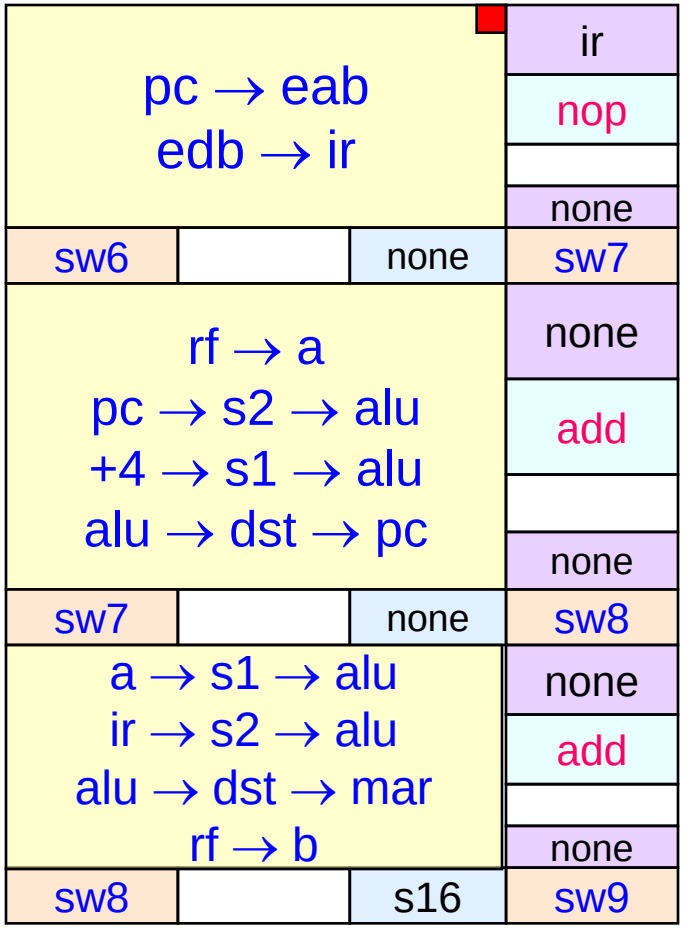
Level 2 Hardware Flowchart



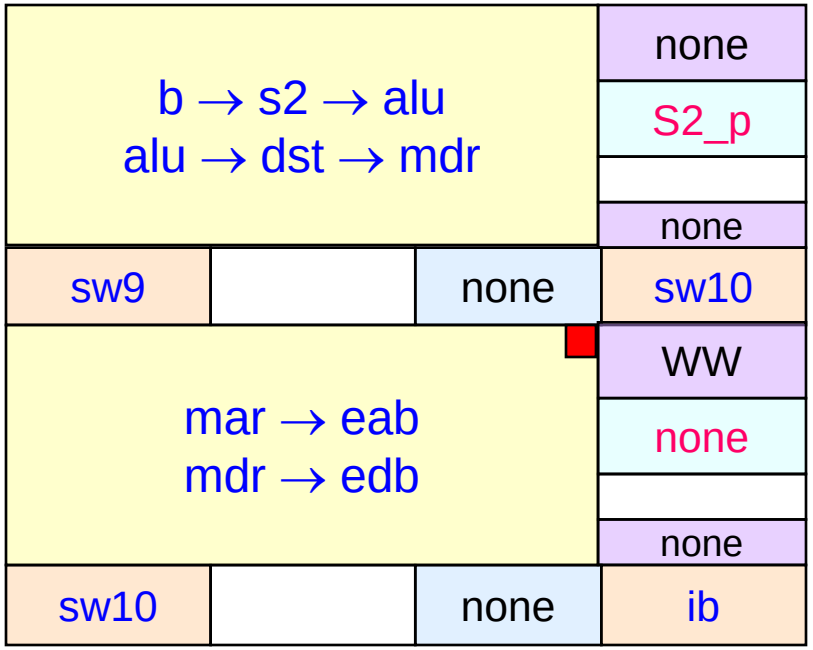
SW Rd, Rs2(Rs1)



Level 2 Hardware Flowchart

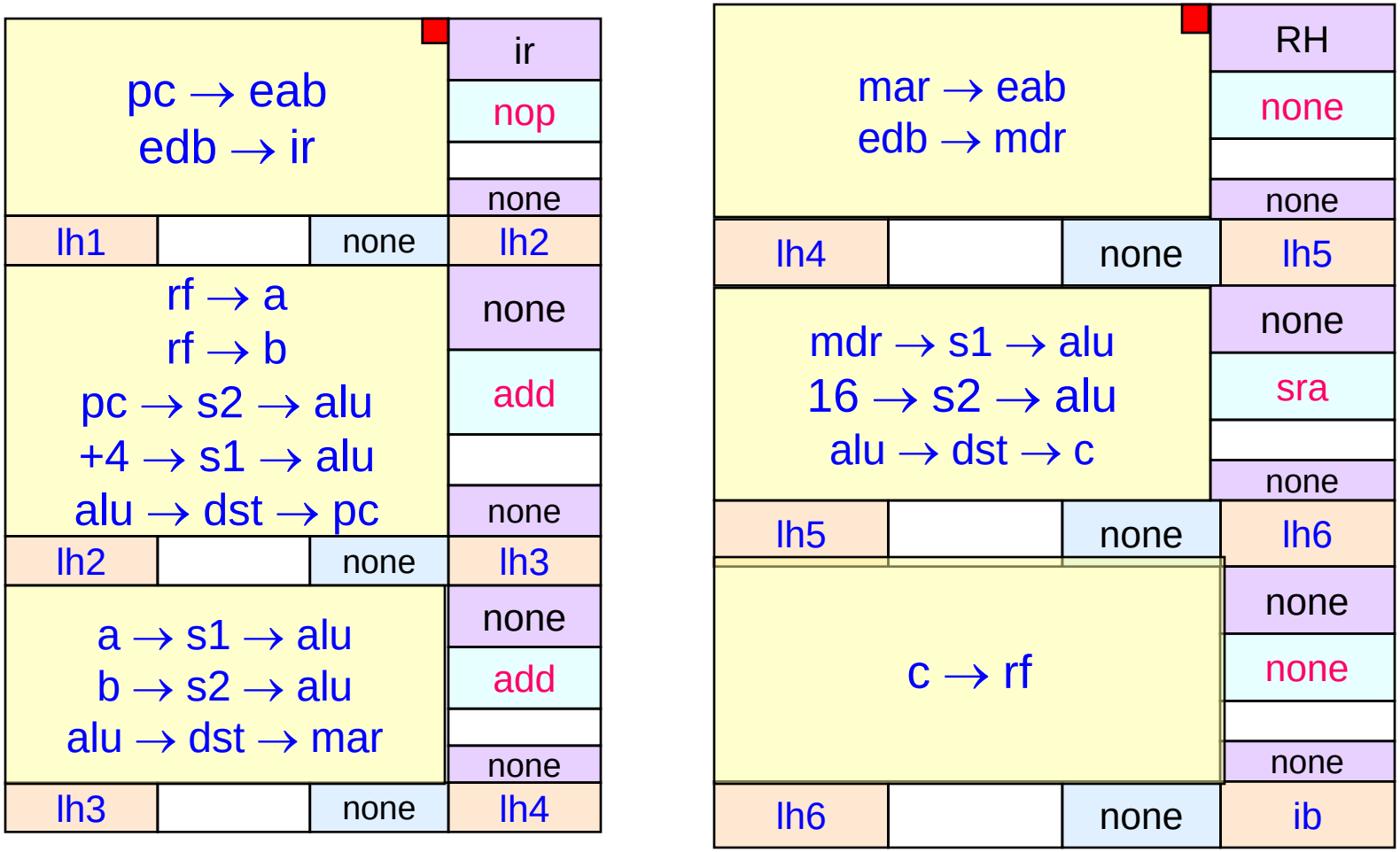


SW.I Rd, Imm(Rs1)



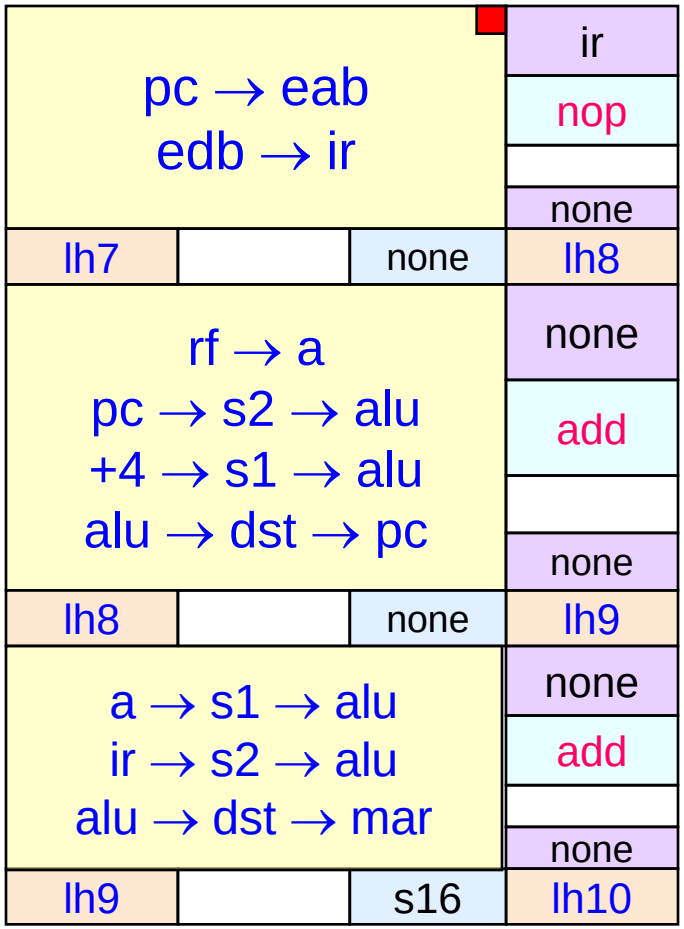
Level 2 Hardware Flowchart

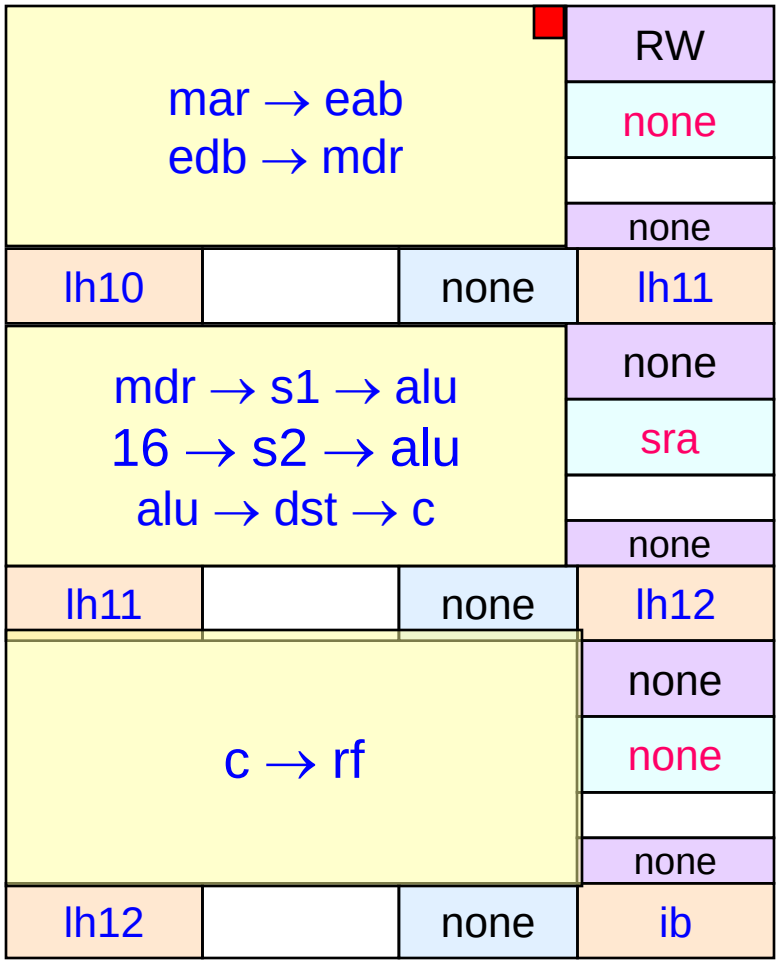
LH Rd, Rs2(Rs1)



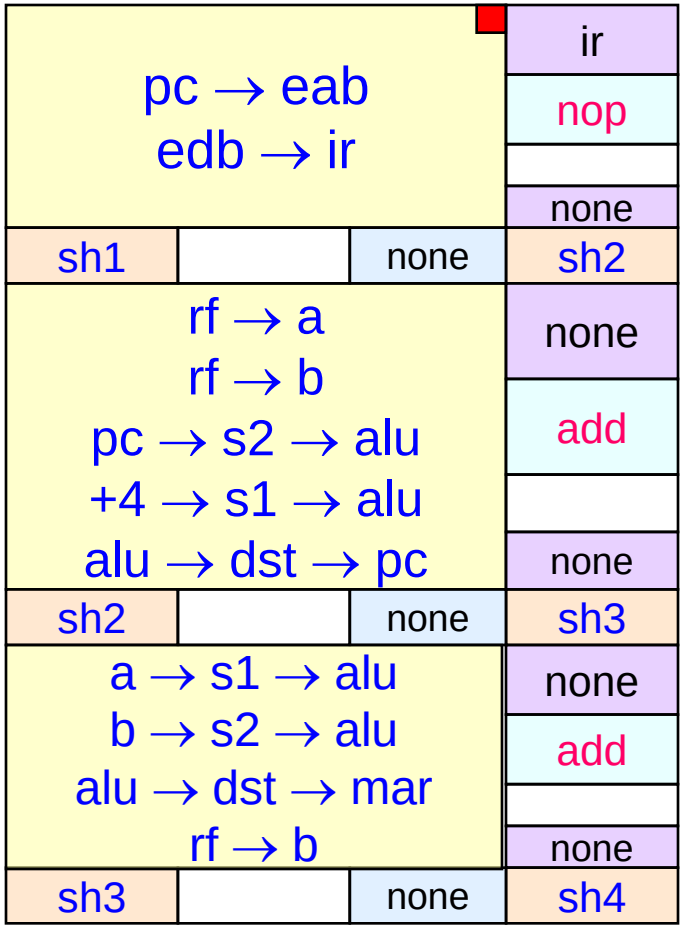
Level 2 Hardware Flowchart

LH.I Rd, Imm(Rs1)

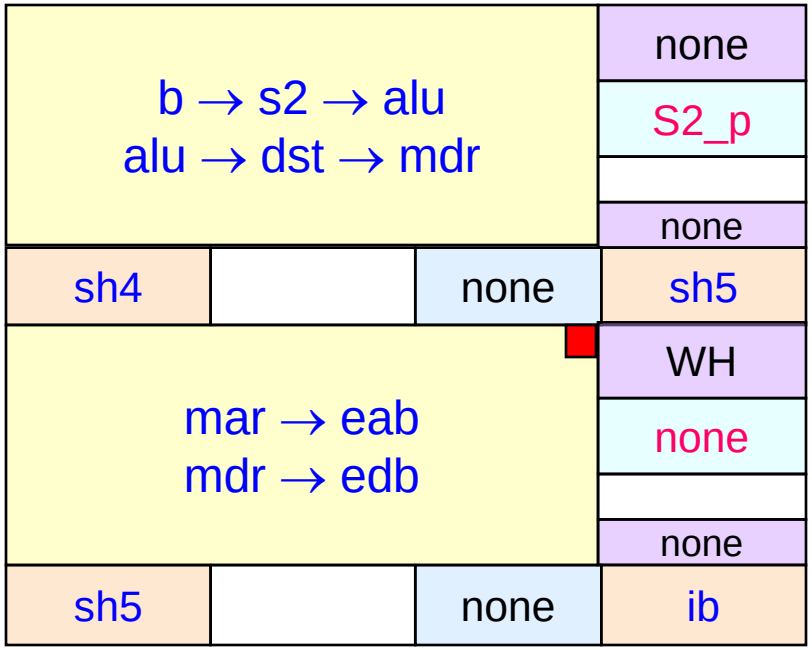




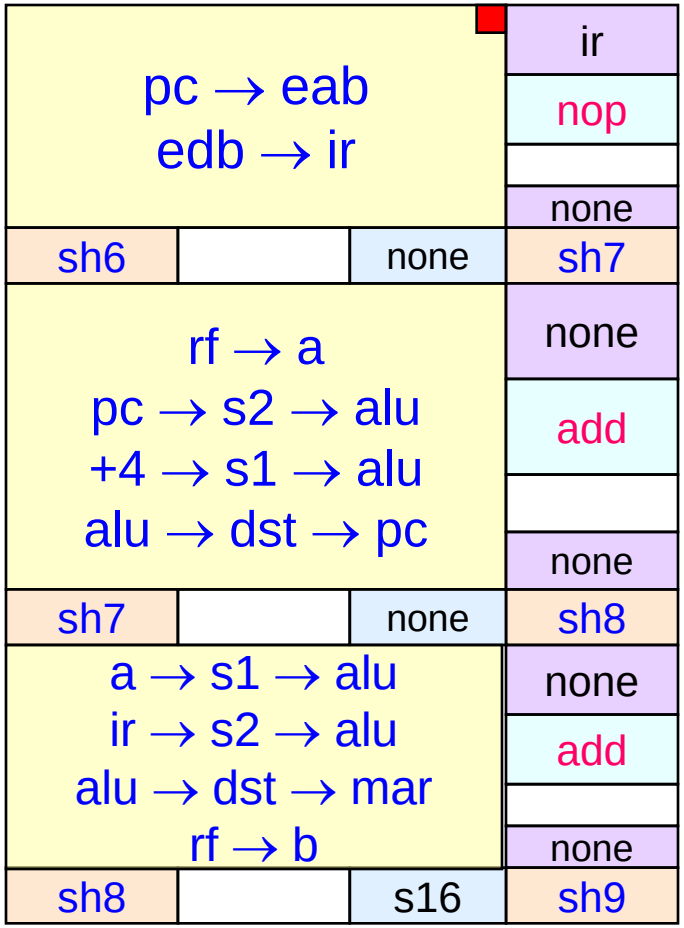
Level 2 Hardware Flowchart



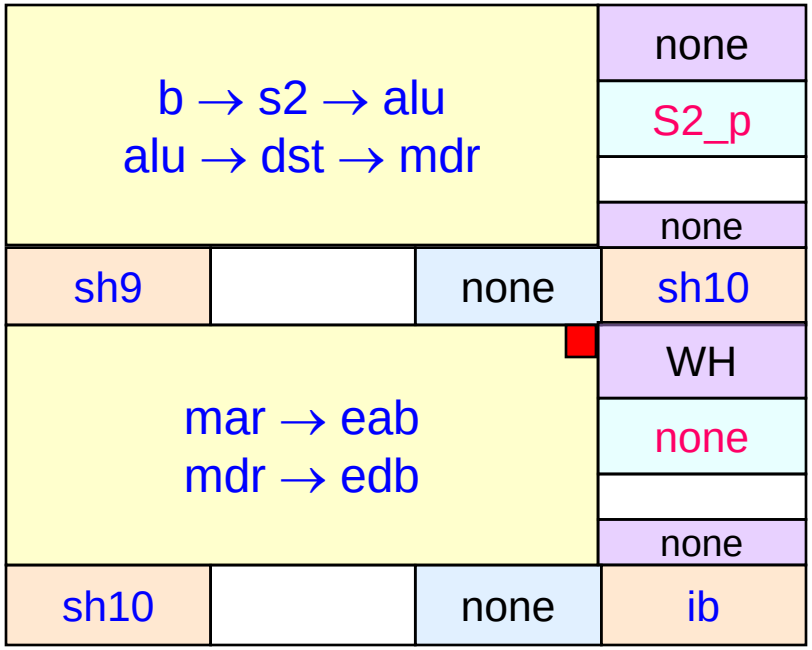
SH Rd, Rs2(Rs1)



Level 2 Hardware Flowchart

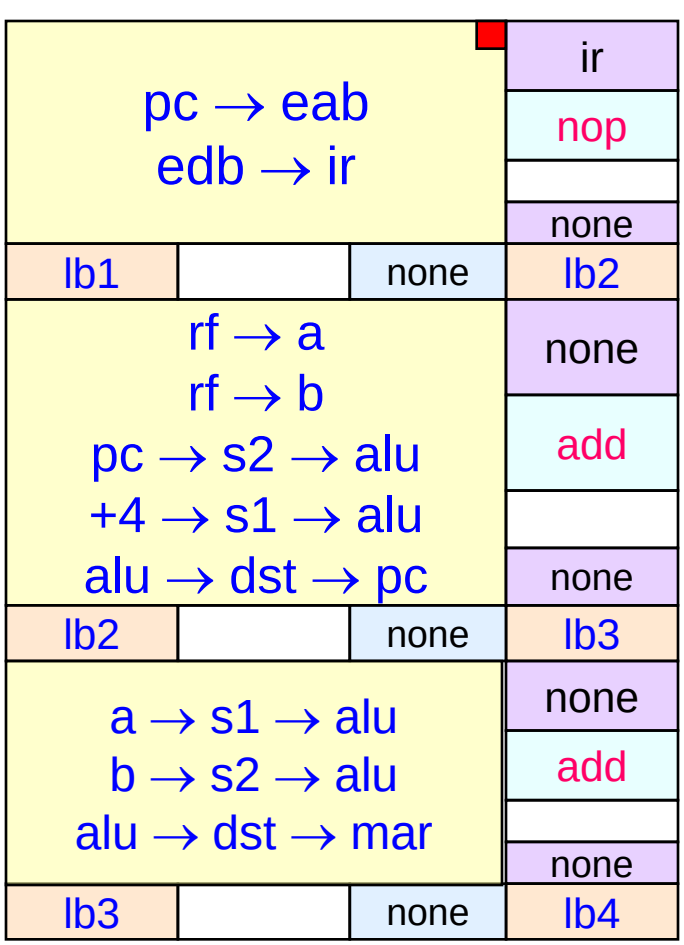


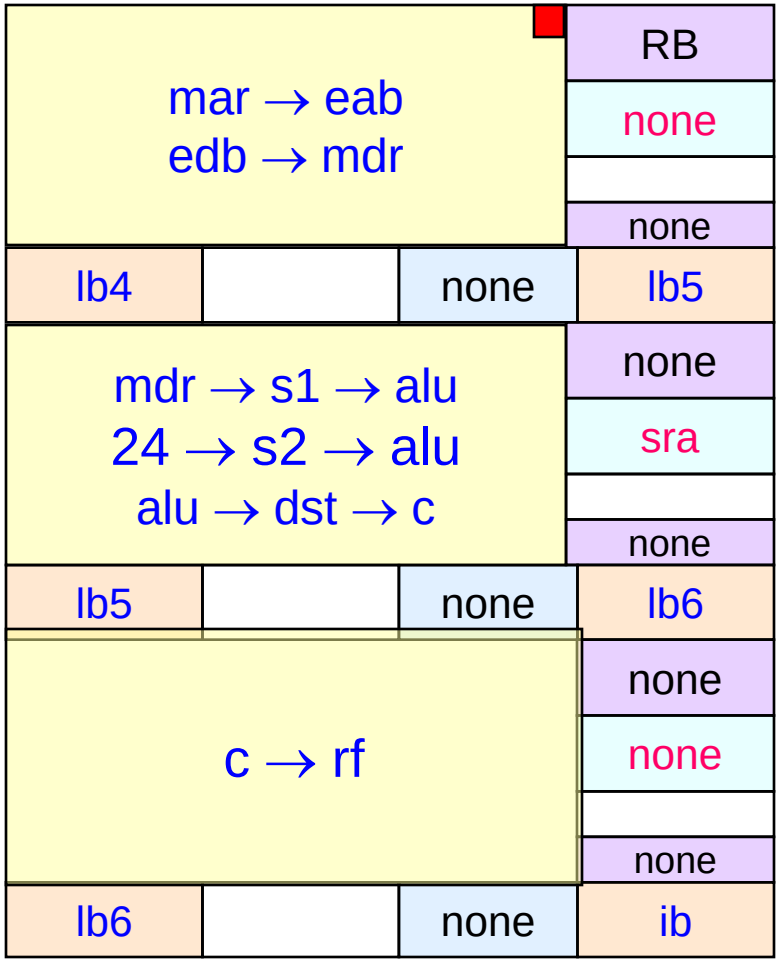
SH.I Rd, Imm(Rs1)



Level 2 Hardware Flowchart

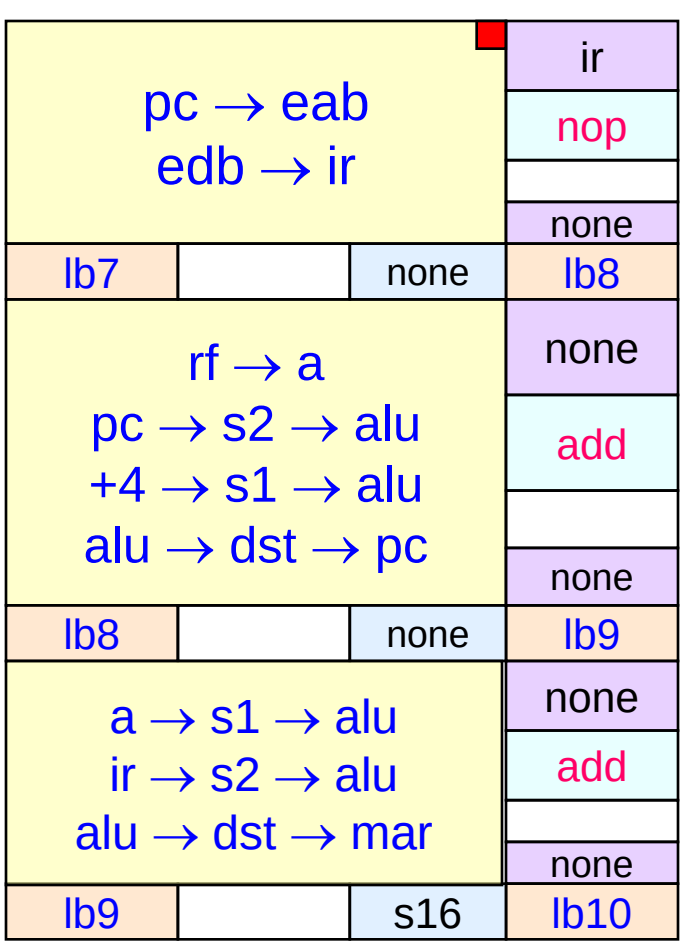
LB Rd, Rs2(Rs1)

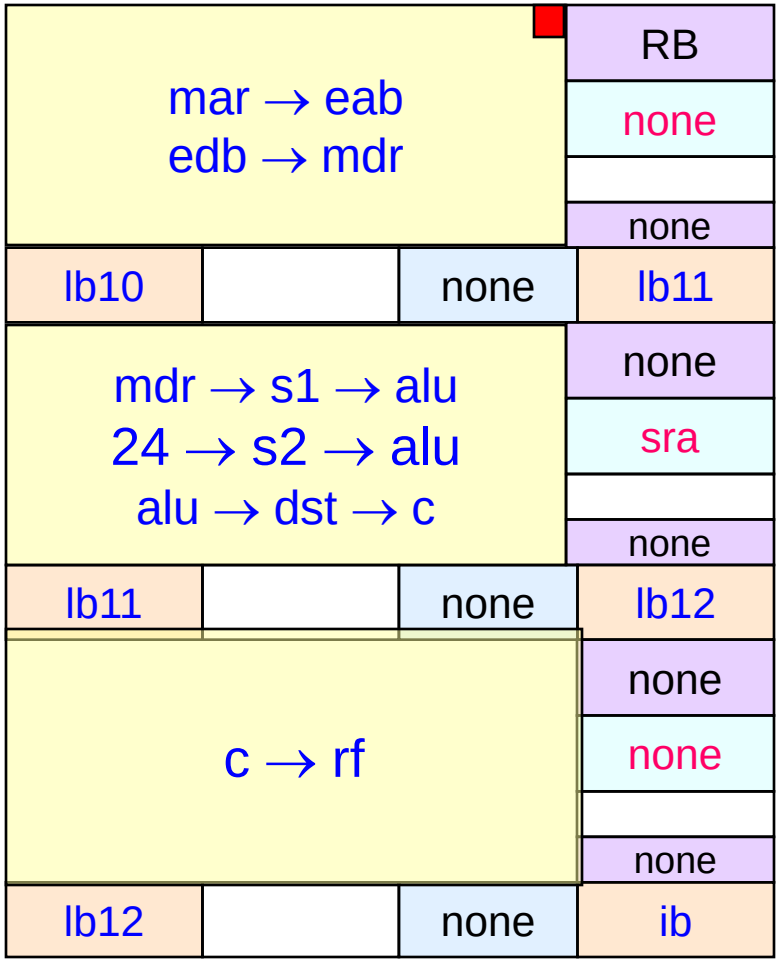




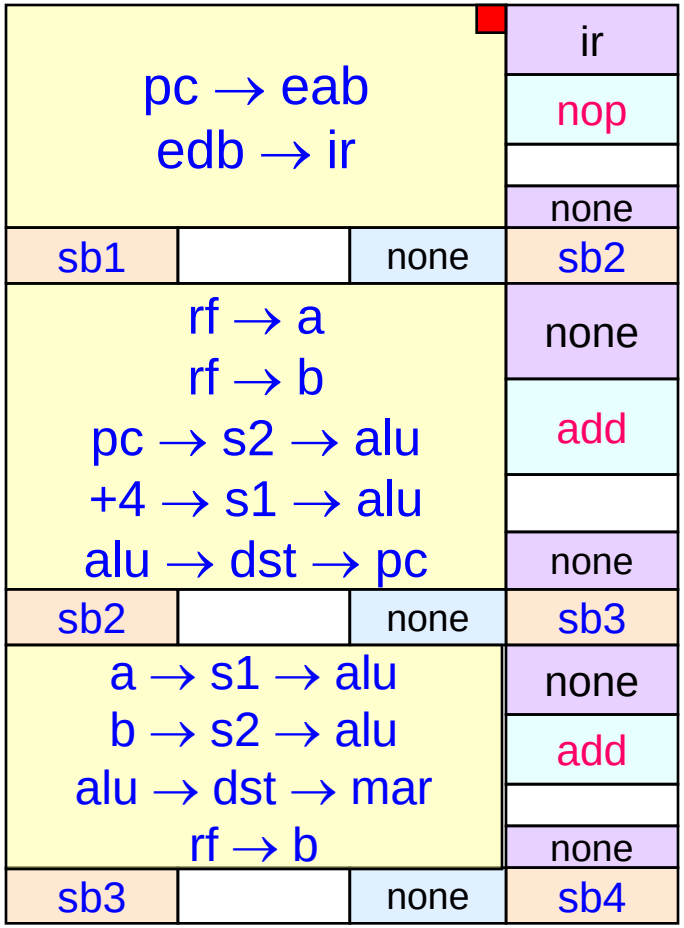
Level 2 Hardware Flowchart

LB.I Rd, Imm(Rs1)

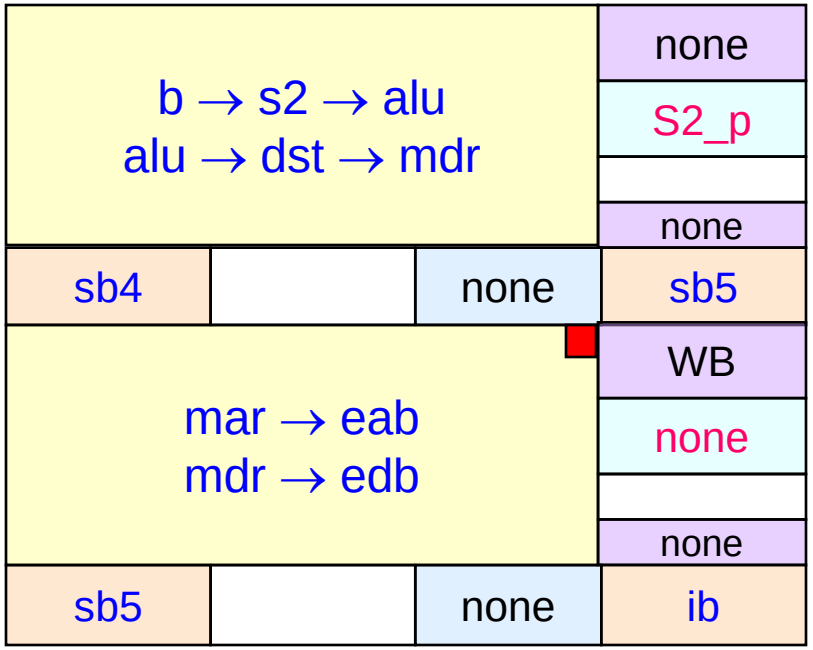




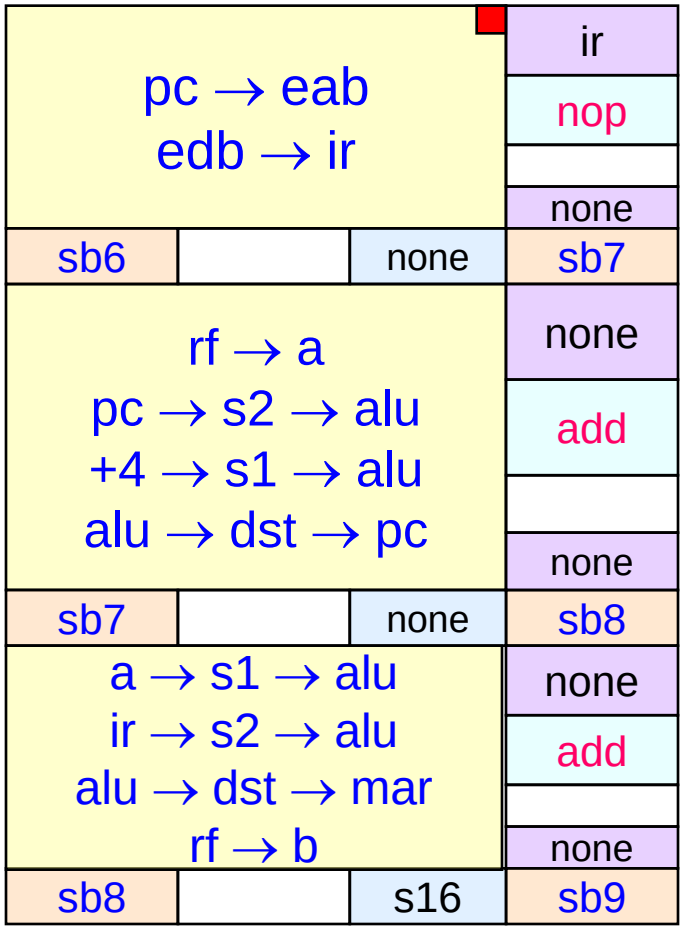
Level 2 Hardware Flowchart



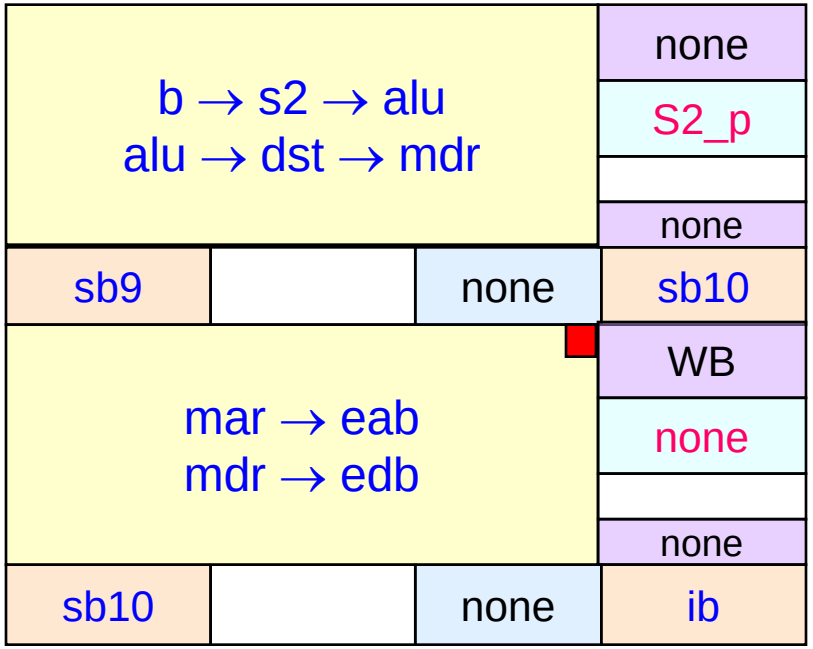
SB Rd, Rs2(Rs1)



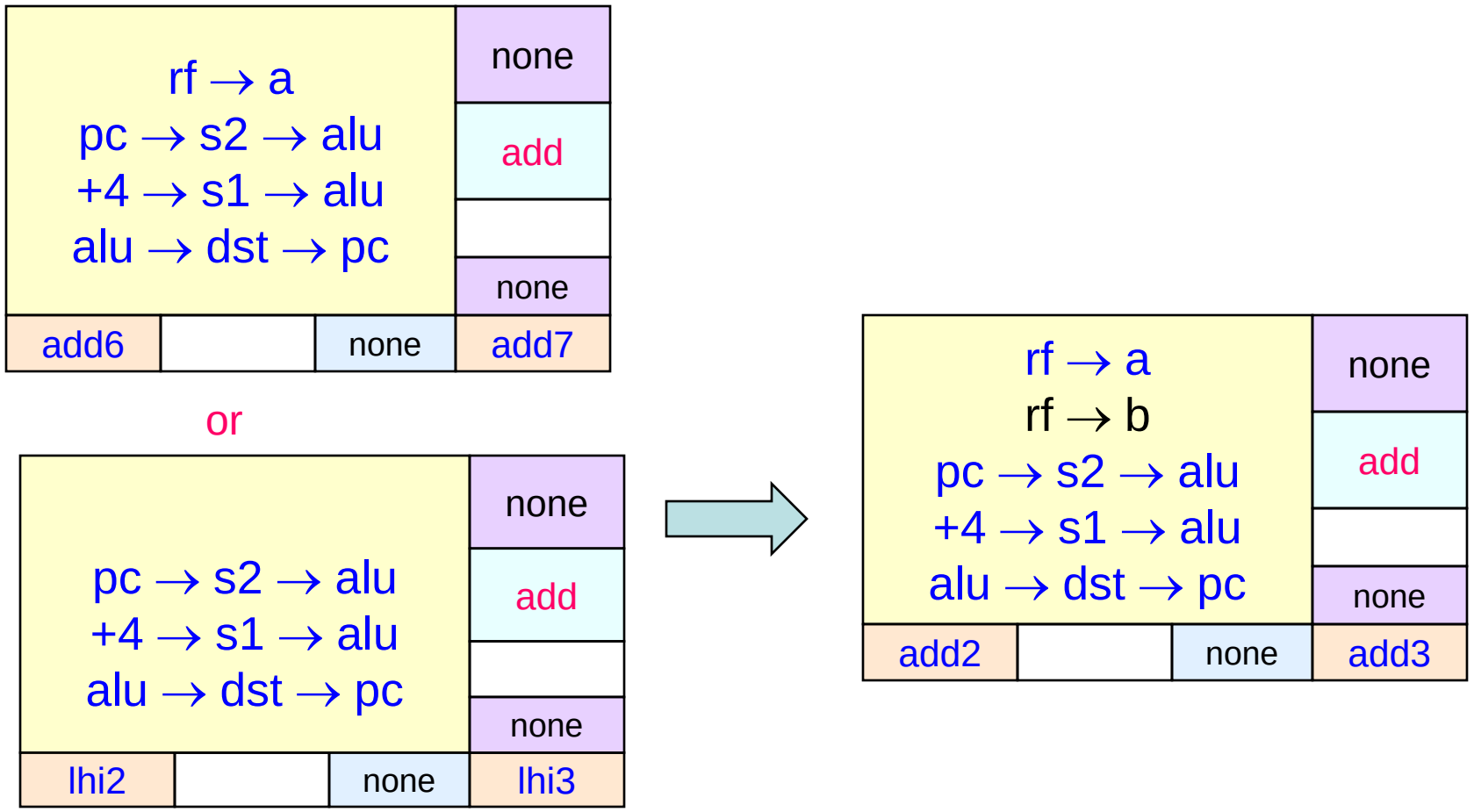
Level 2 Hardware Flowchart



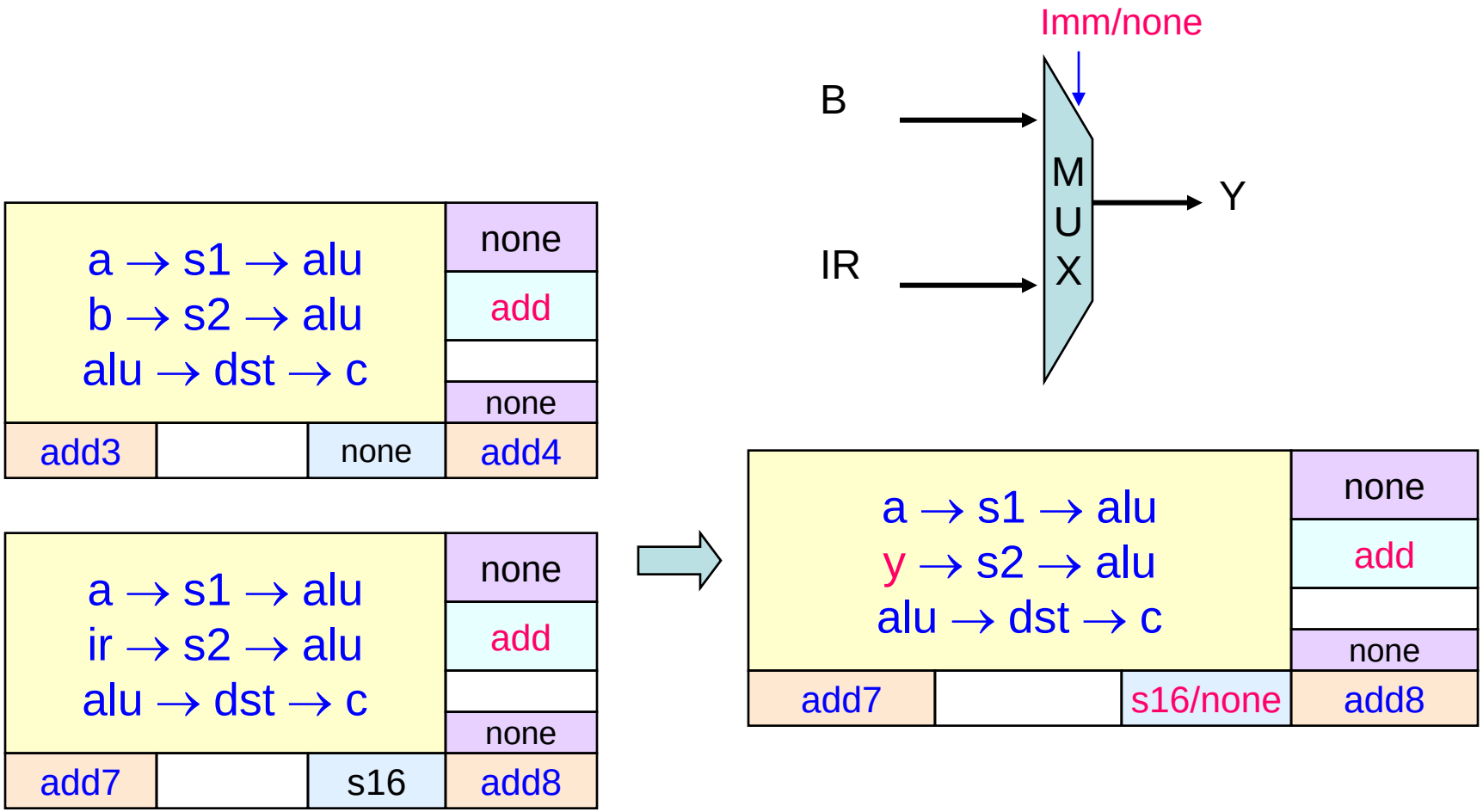
SB.I Rd, Imm(Rs1)



Modified Level 2 Hardware Flowchart

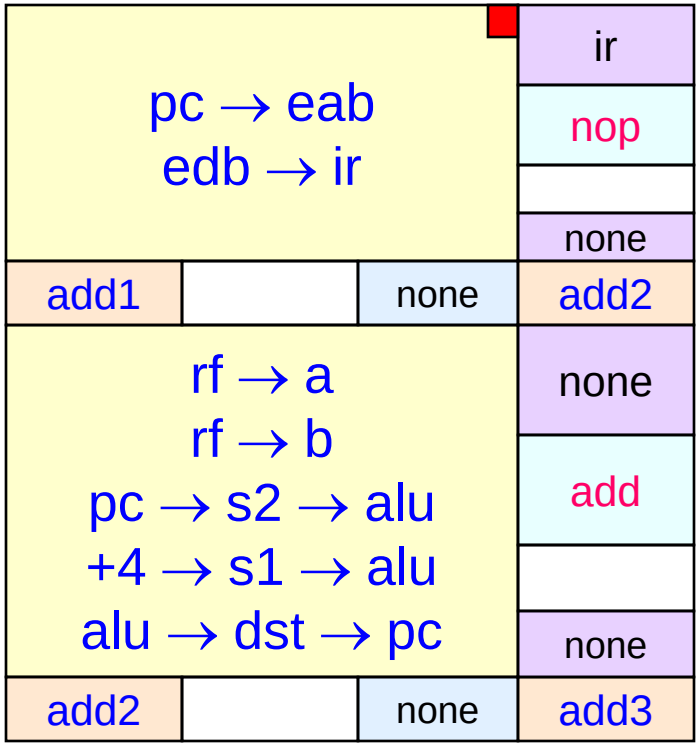


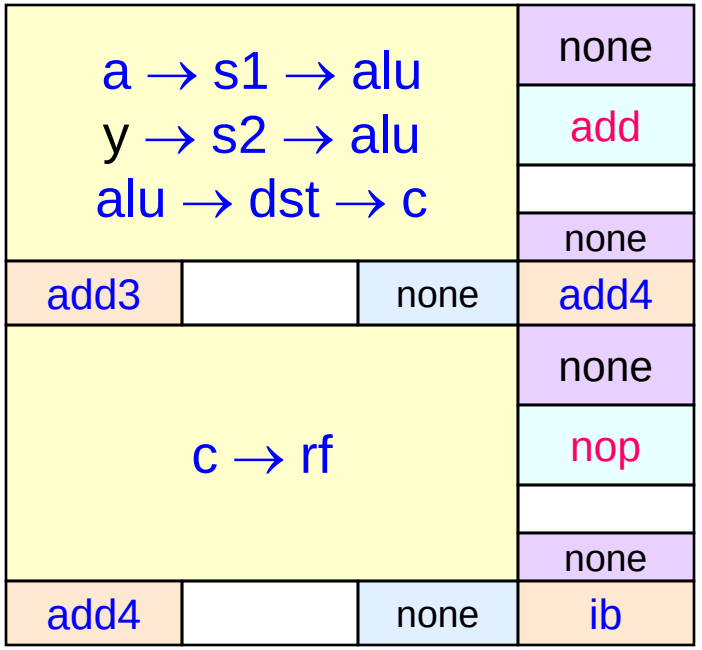
Modified Level 2 Hardware Flowchart



Modified Level 2 Hardware Flowchart

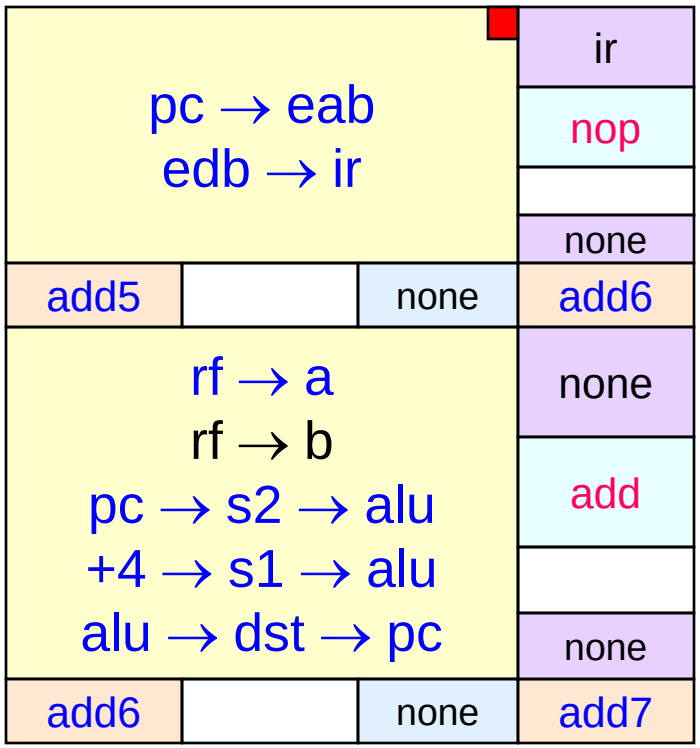
ADD Rd, Rs1, Rs2

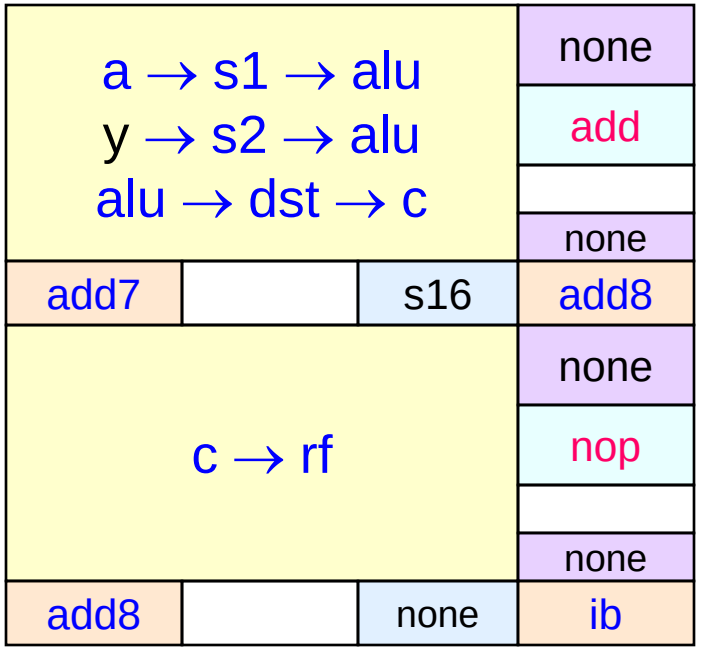




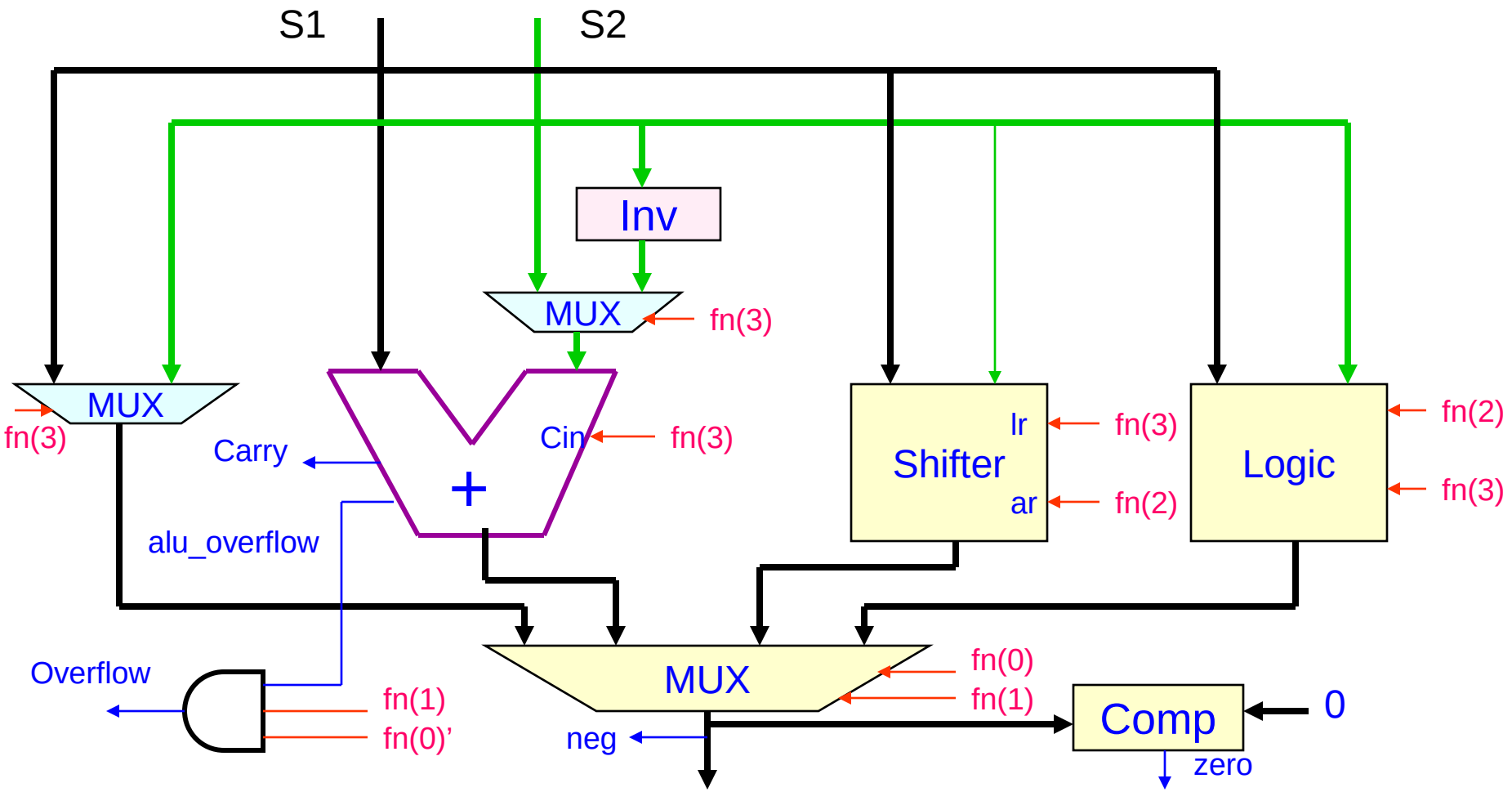
Modified Level 2 Hardware Flowchart

ADD.I Rd, Rs1, Imm





Arithmetic Logic Unit



Level 2 Hardware Flowchart: Merging

- Merge identical states

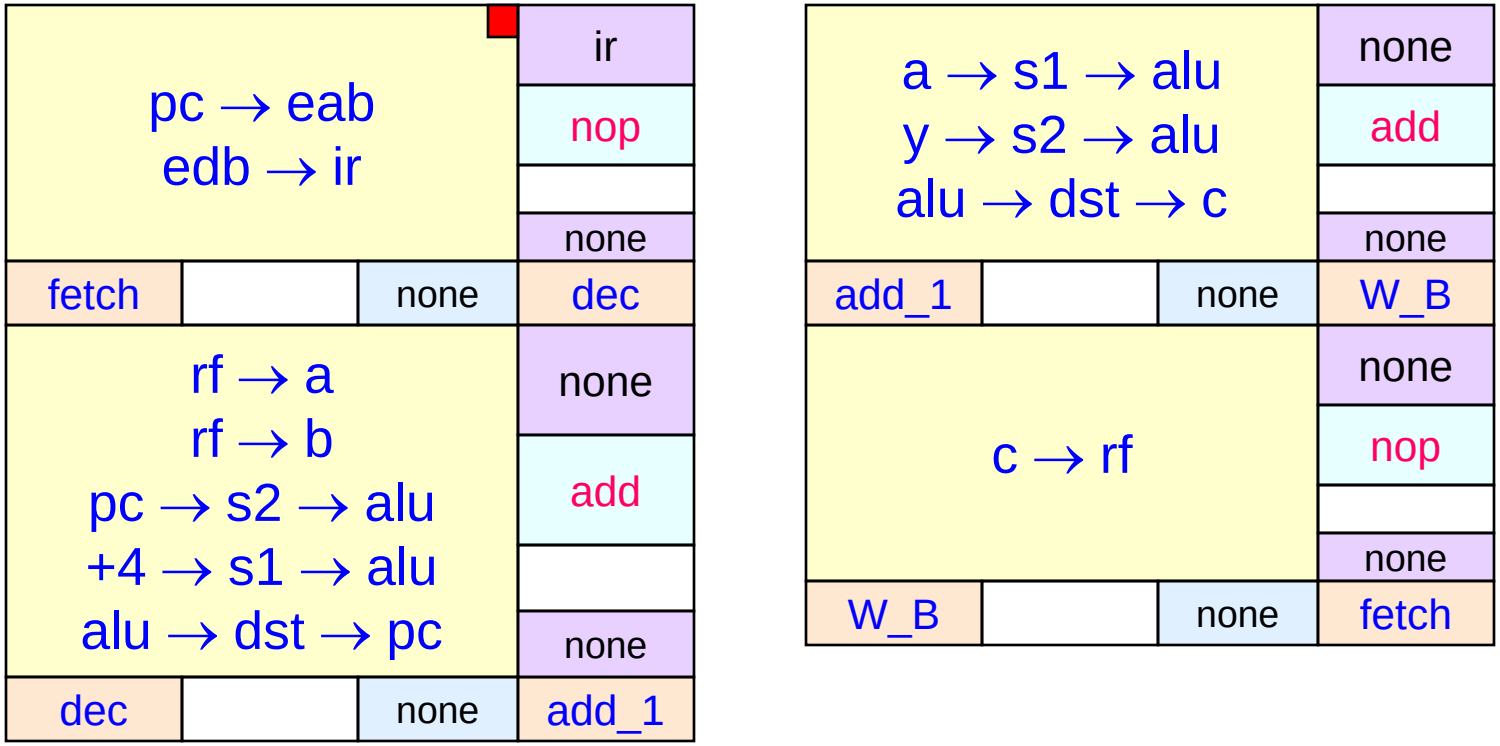
add1 = add5 = sub1 = sub5 = and1 = and 5 = or1 = 0r5 = xor1 = xor5 = sll1 = sll5 = srl 1 = srl5 = sra1 = sra5 = lhi1 = nop1 = seq1 = seq8 = sne1 = sne8 = sgt1 = sge8 = slt1 = slt 8 = sle1 = sle8 = beqz1 = bnez1 = j1 = jr1 = jal1= jalr1 = lw1 = lw7 = sw1 = sw6 = lh1 = lh7 = sh1 = sh6 = lb1 = lb7 = sb1 = sb6 Ξ fetch

Level 2 Hardware Flowchart: Merging

add2 = add6 = sub2 = sub6 = and2 = and6 = or2 = 0r6 =
xor2 = xor6 = sll2 = sll6 = srl2 = srl6 = sra2 = sra6 =
lhi2 = nop2 = seq2 = seq9 = sne2 = sne9 = sgt2 =
sge9 = slt2 = slt 9 = sle2 = sle9 = beqz2 = bnez9 = j2
= jr2 = jal2= jalr2 = lw2 = lw8 = sw2 = sw7 = lh2 =
lh8 = sh2 = sh7 = lb2 = lb8 = sb2 = sb7 Ξ dec

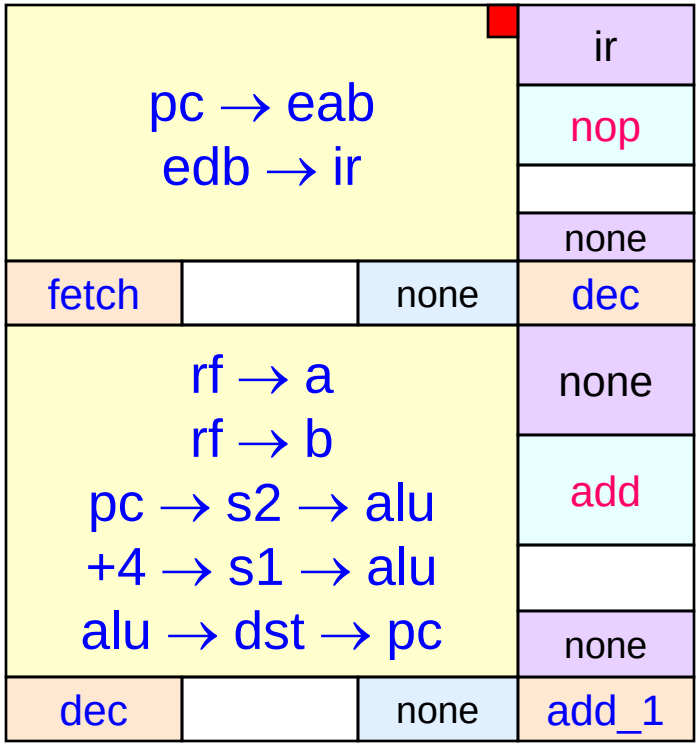
Merged Level 2 Hardware Flowchart

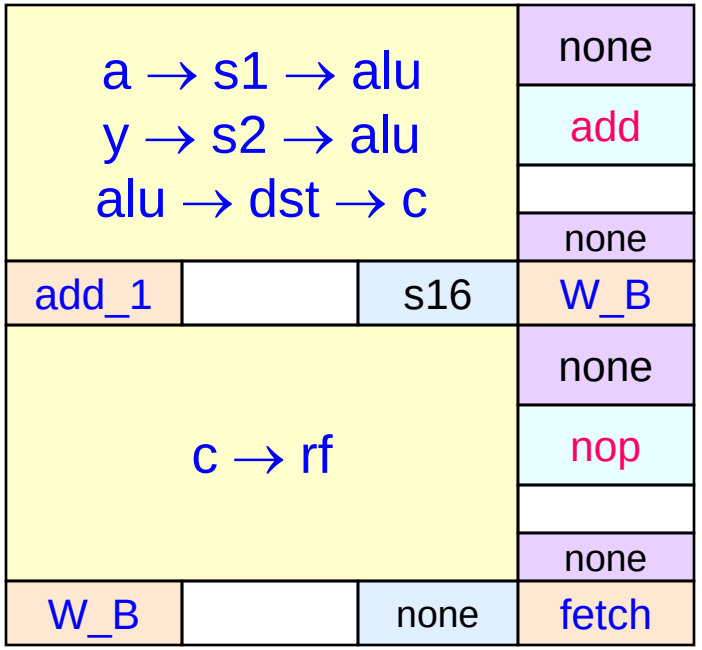
ADD Rd, Rs1, Rs2



Merged Level 2 Hardware Flowchart

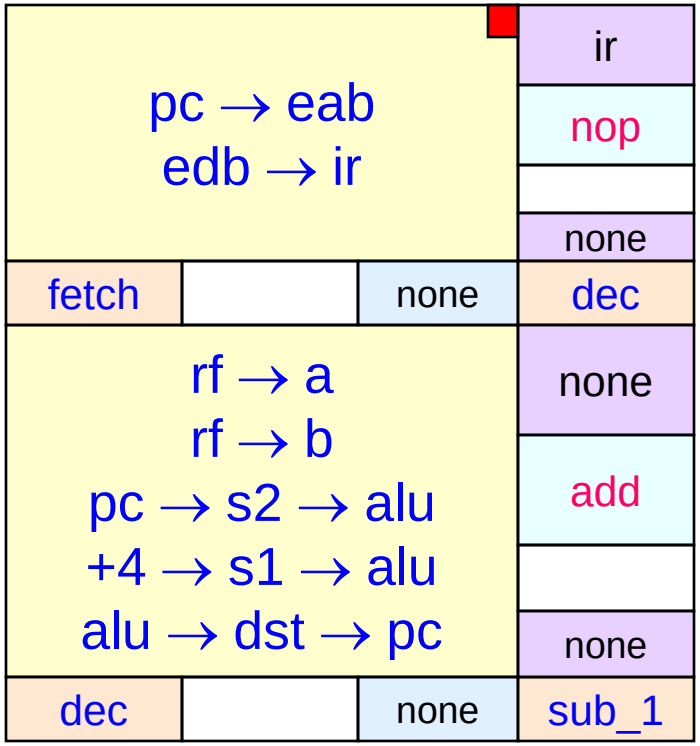
ADD.I Rd, Rs1, Imm

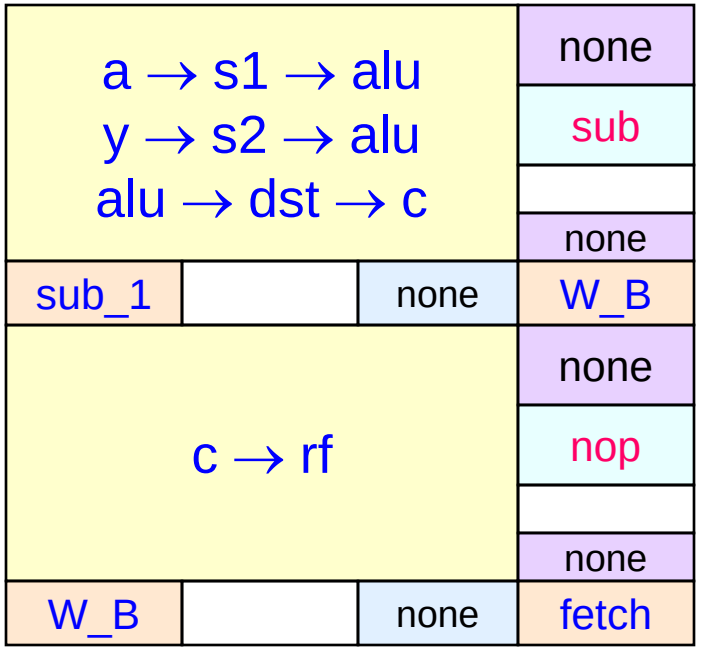




Merged Level 2 Hardware Flowchart

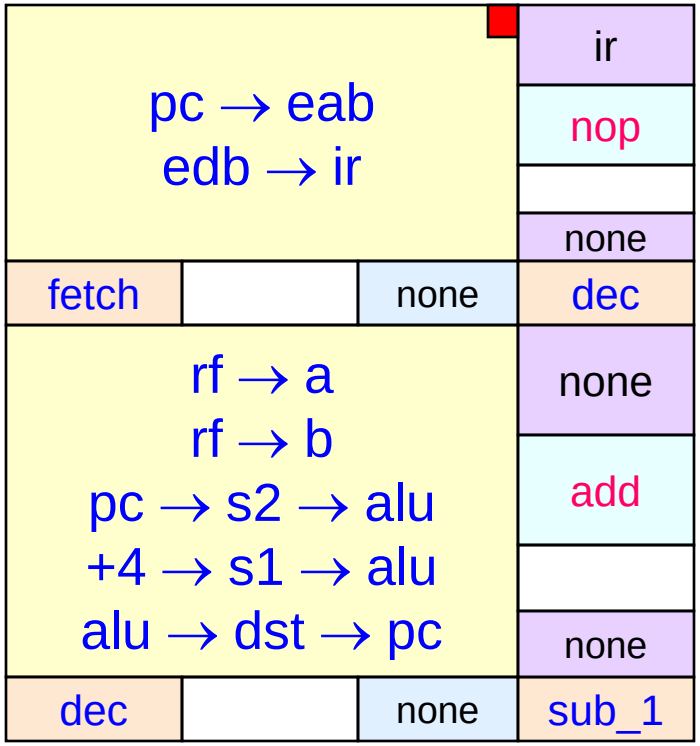
SUB Rd, Rs1, Rs2

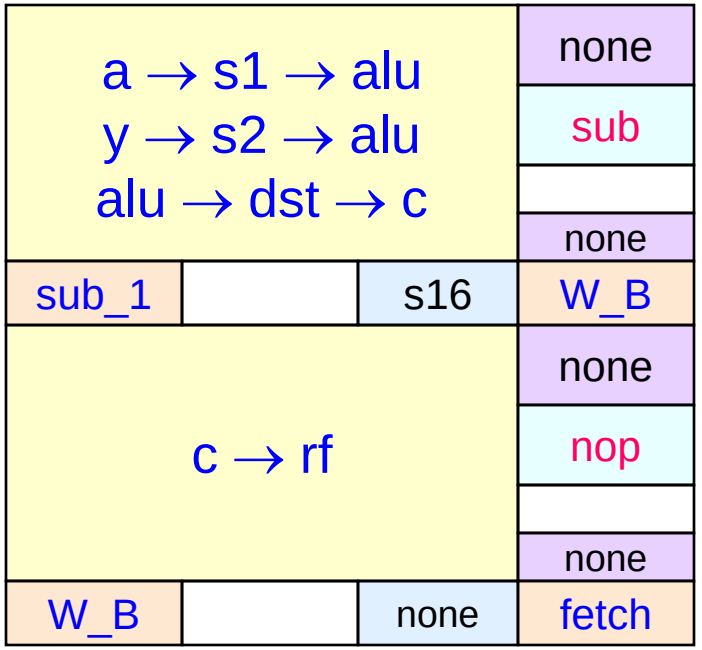




Merged Level 2 Hardware Flowchart

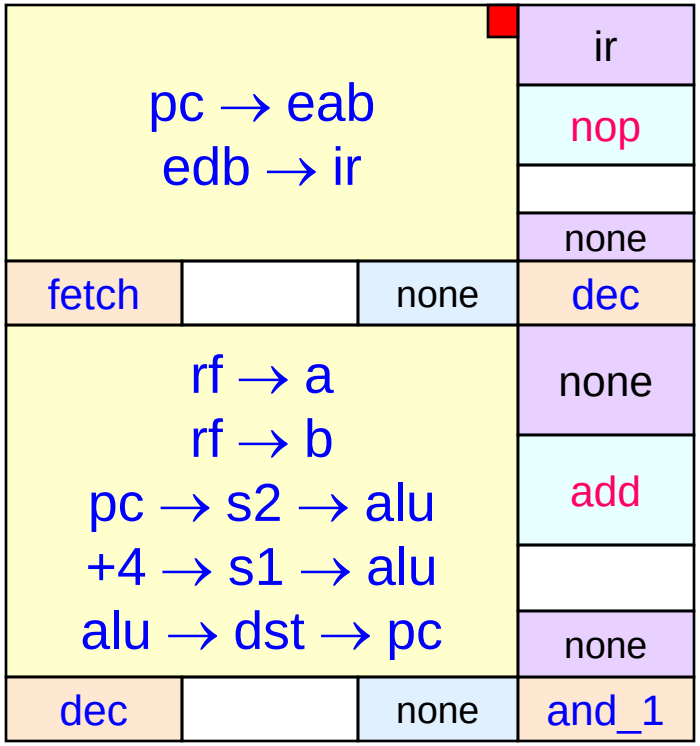
SUB.I Rd, Rs1, Imm

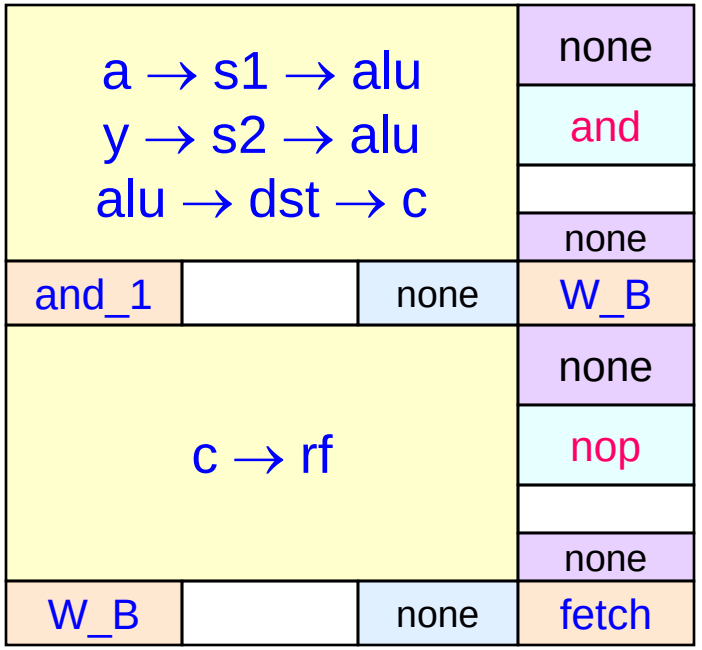




Merged Level 2 Hardware Flowchart

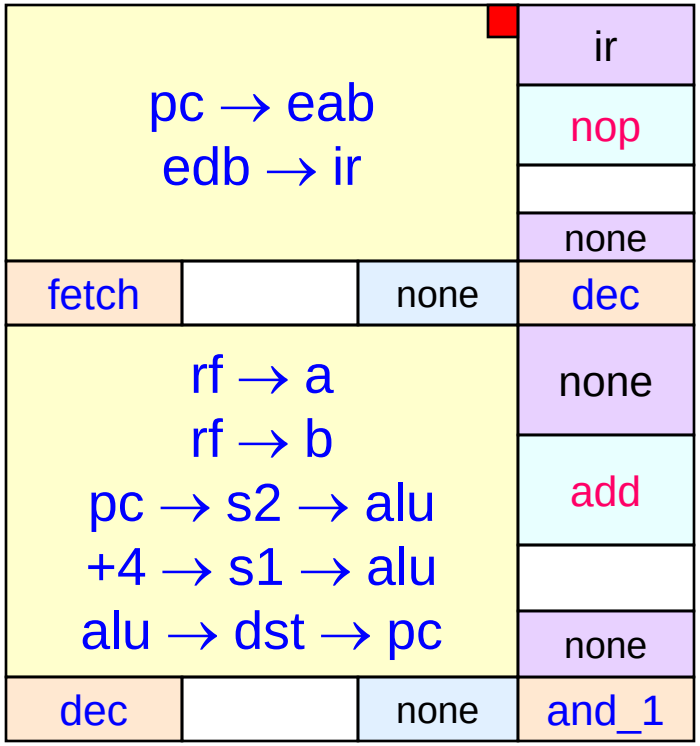
AND Rd, Rs1, Rs2

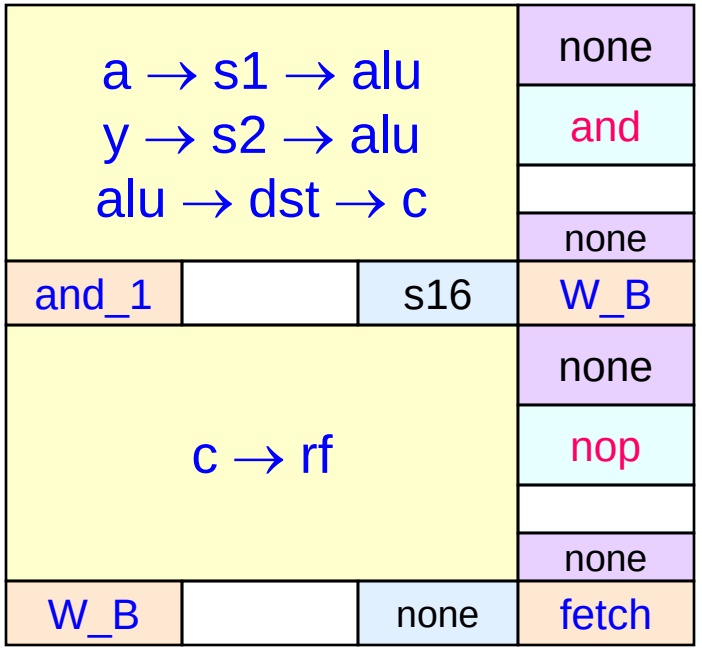




Merged Level 2 Hardware Flowchart

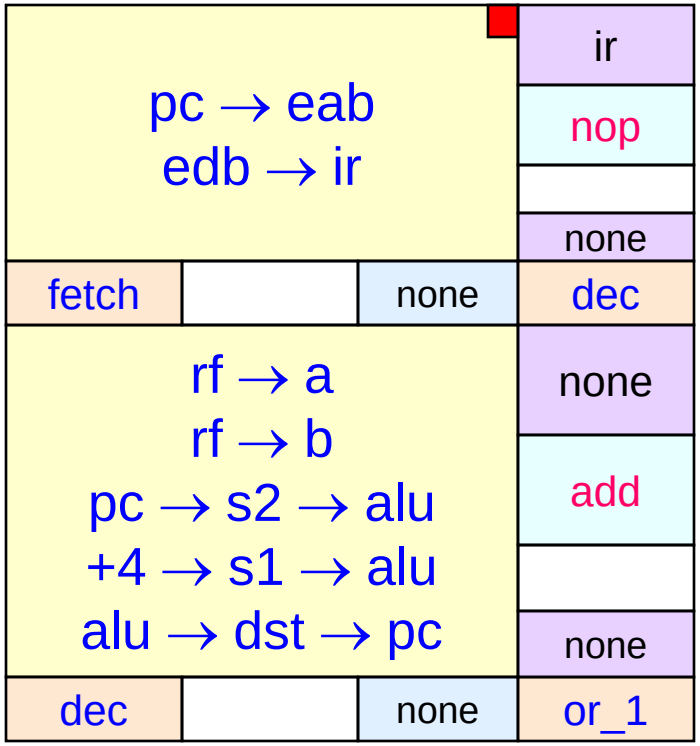
AND.I Rd, Rs1, Imm

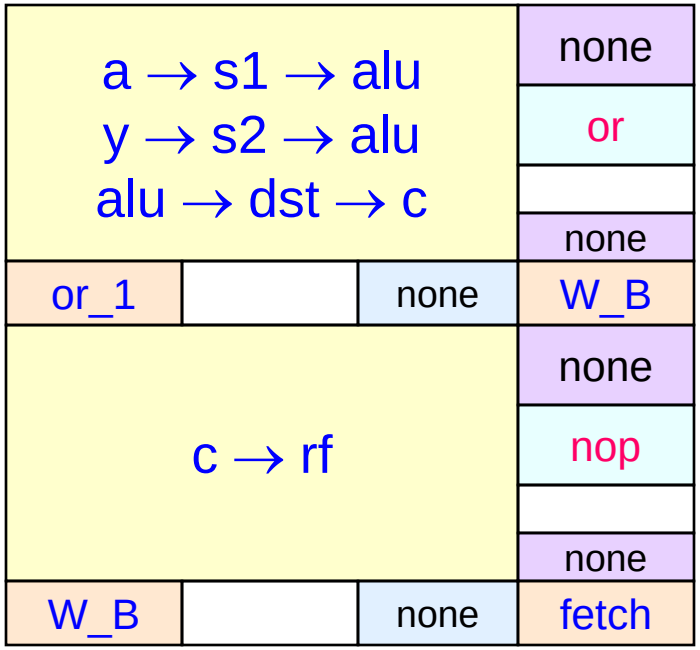




Merged Level 2 Hardware Flowchart

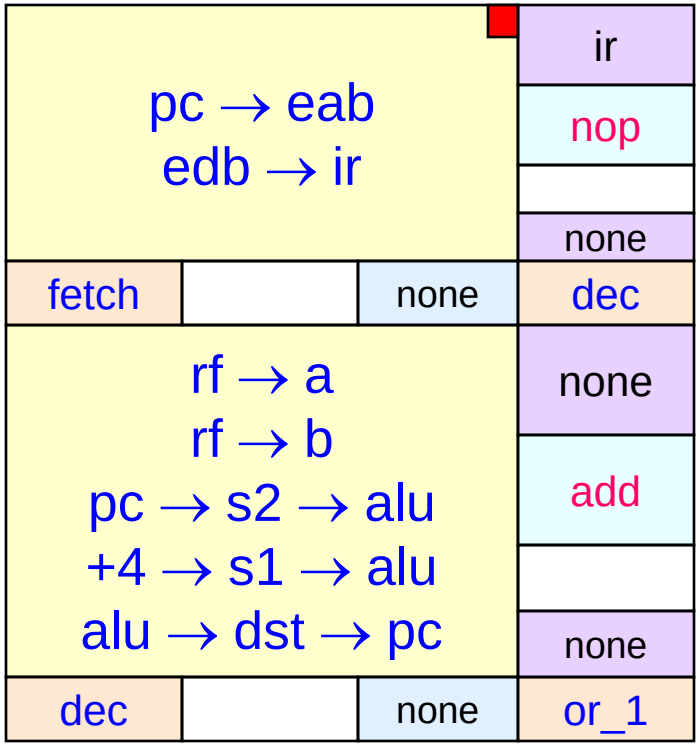
OR Rd, Rs1, Rs2

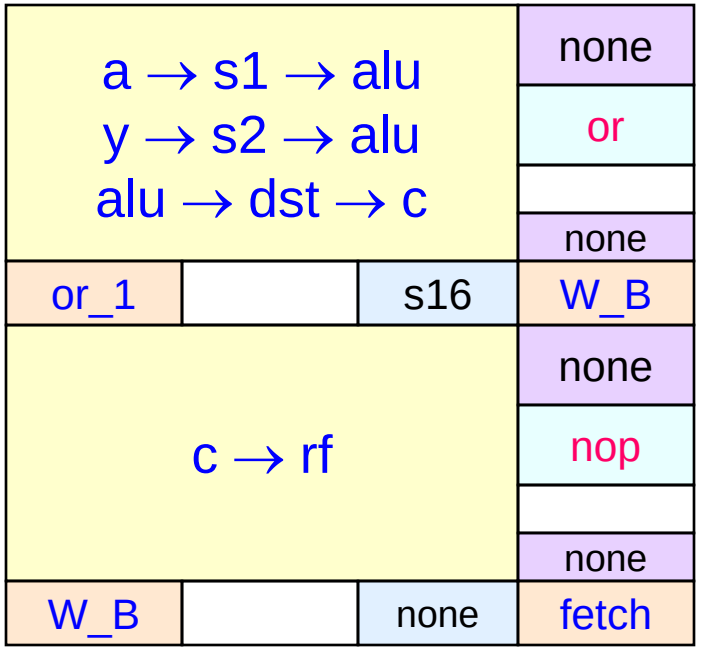




Merged Level 2 Hardware Flowchart

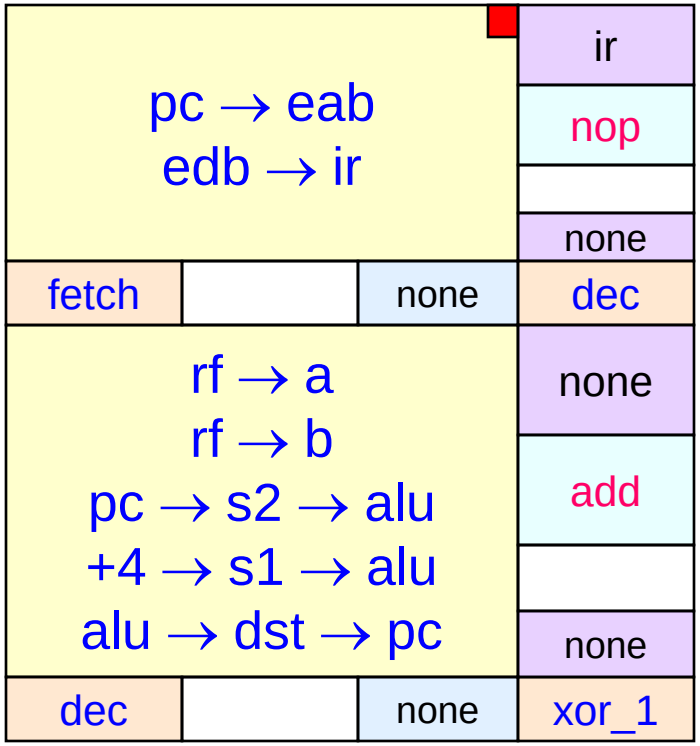
OR.I Rd, Rs1, Imm

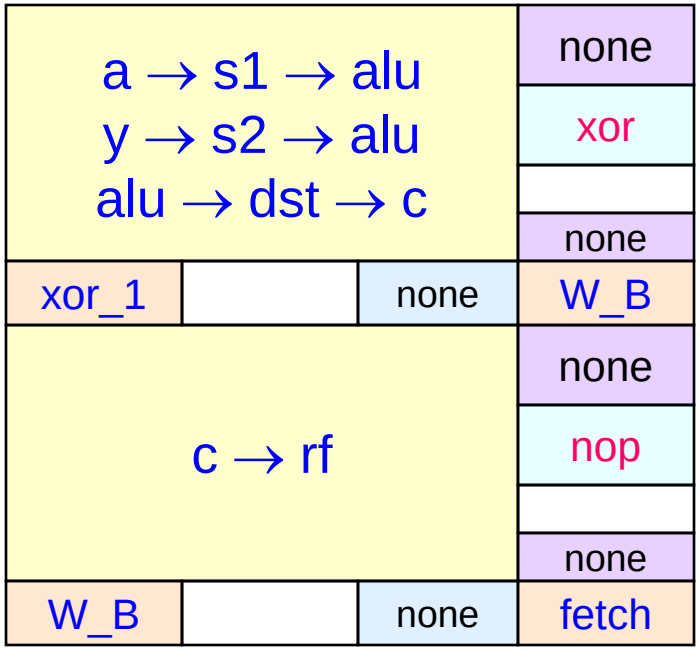




Merged Level 2 Hardware Flowchart

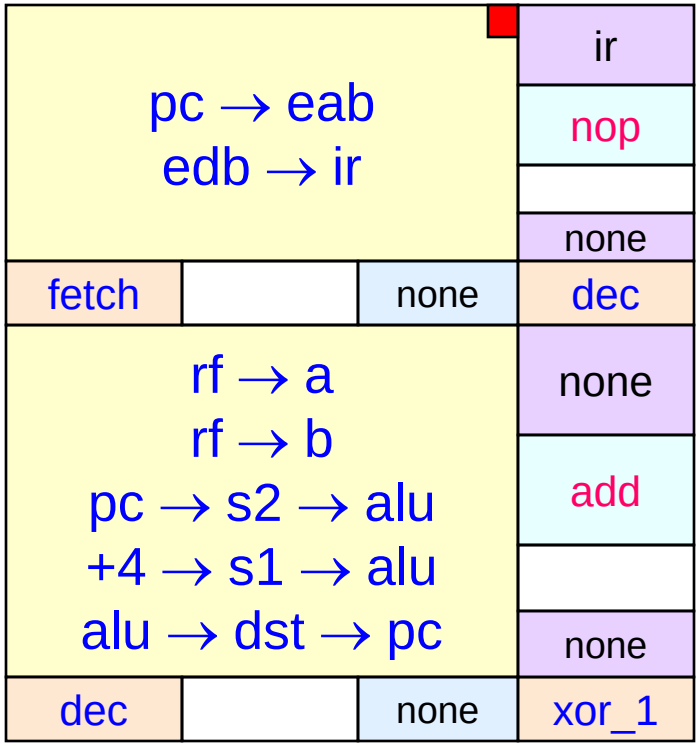
XOR Rd, Rs1, Rs2

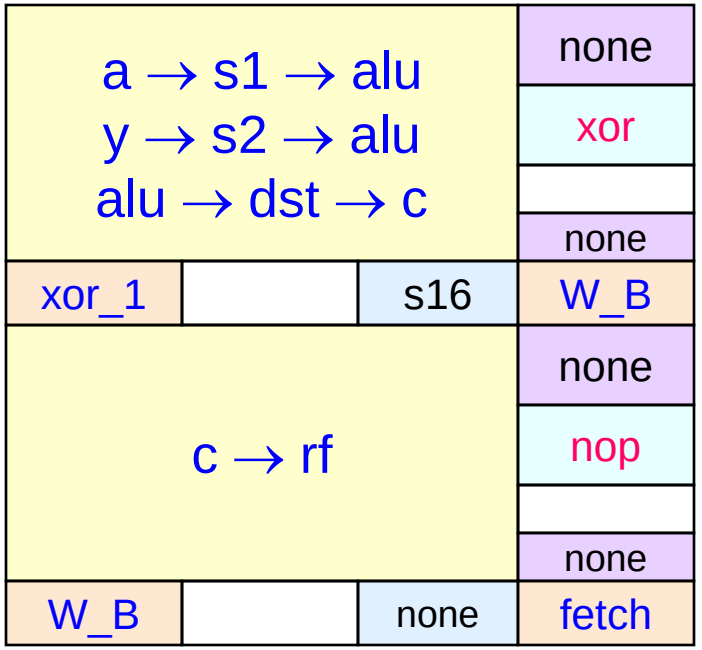




Merged Level 2 Hardware Flowchart

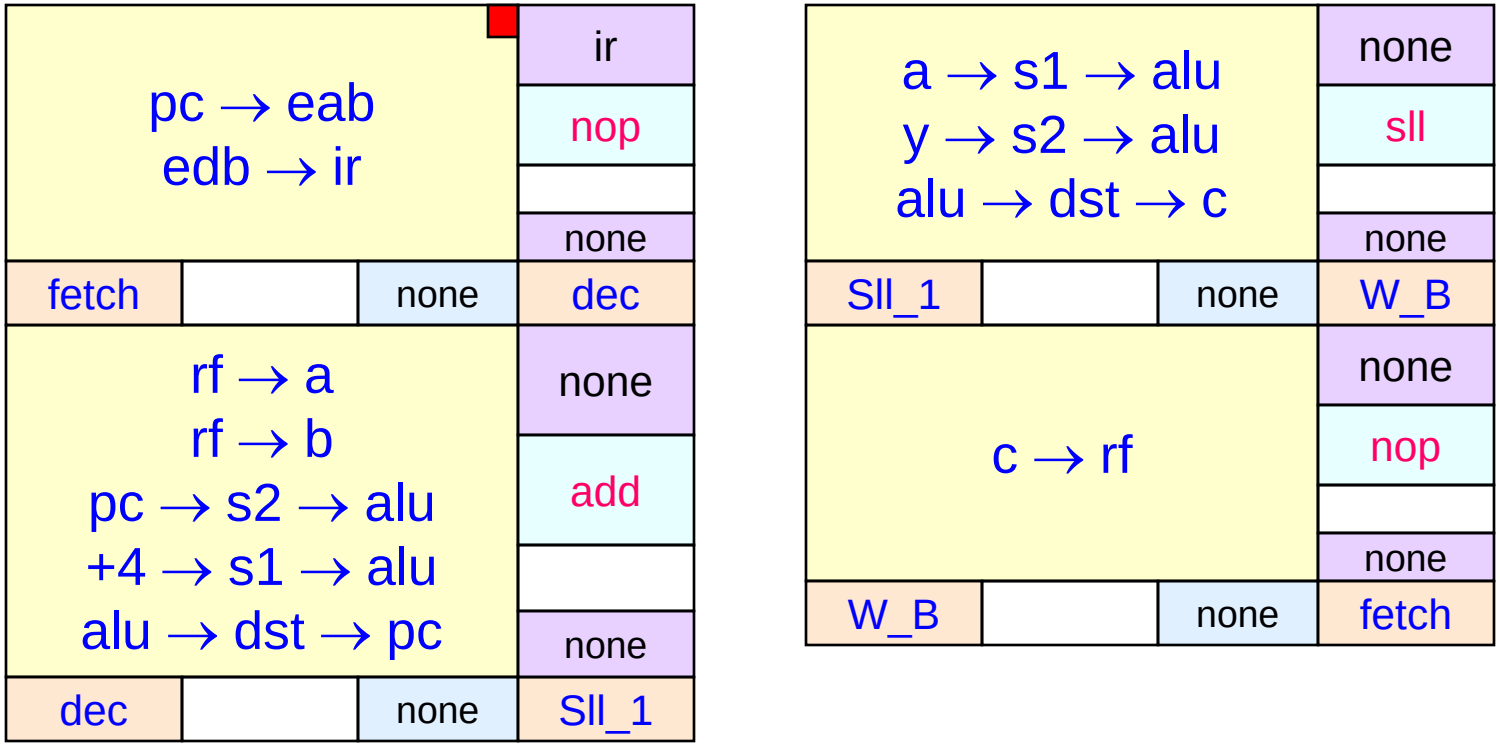
XOR.I Rd, Rs1, Imm





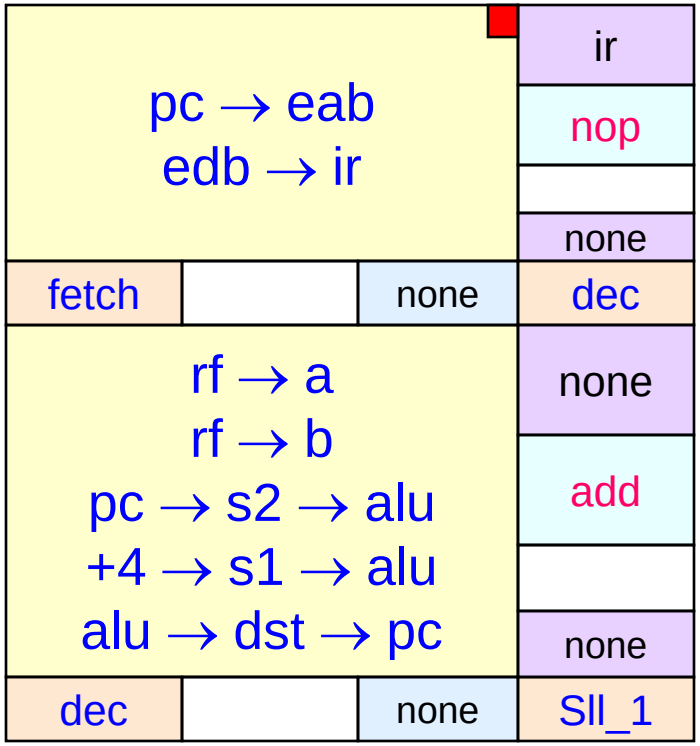
Merged Level 2 Hardware Flowchart

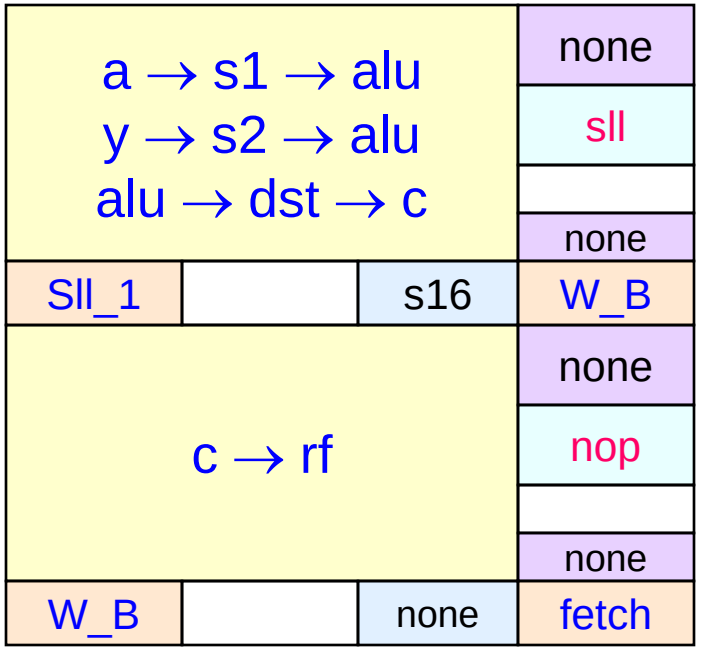
SLL Rd, Rs1, Rs2



Merged Level 2 Hardware Flowchart

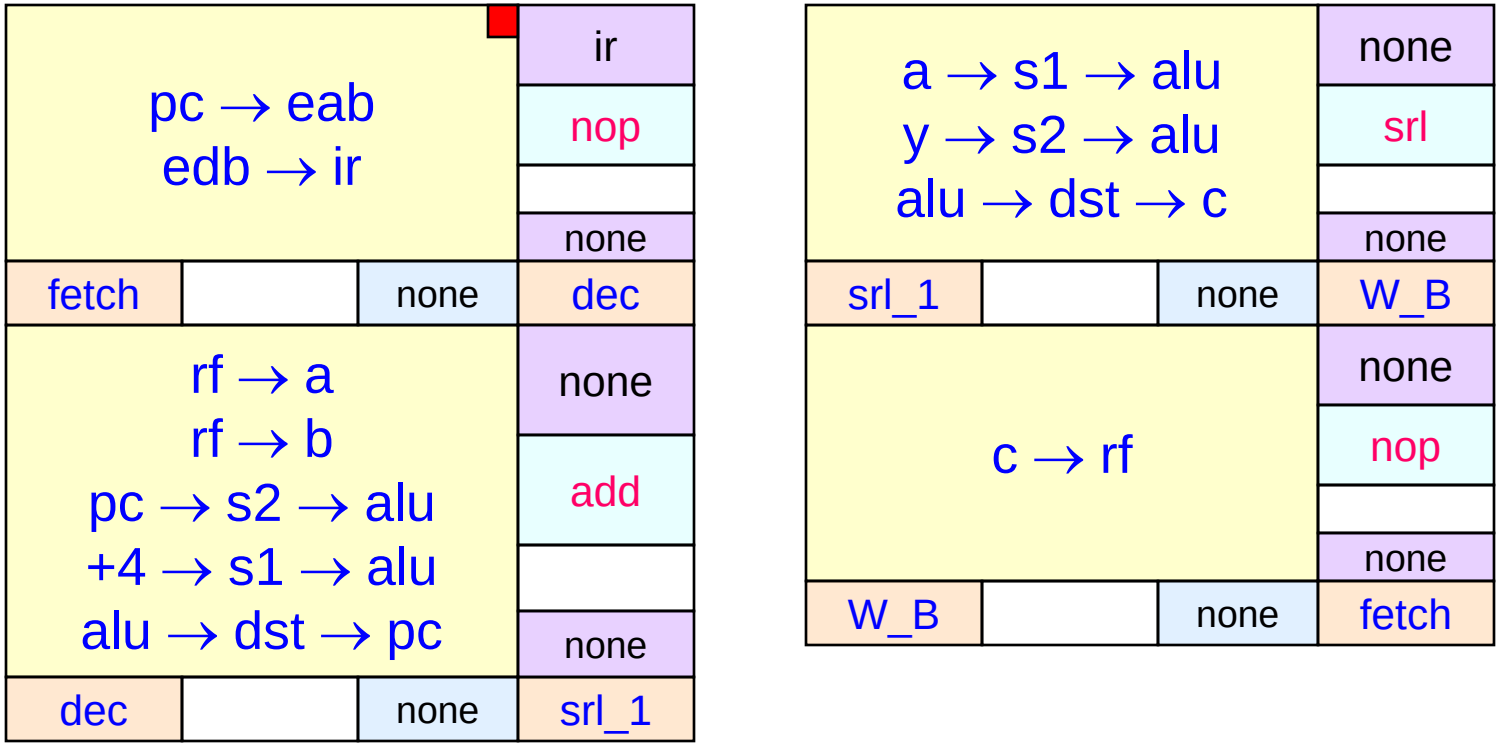
SLL.I Rd, Rs1, Imm





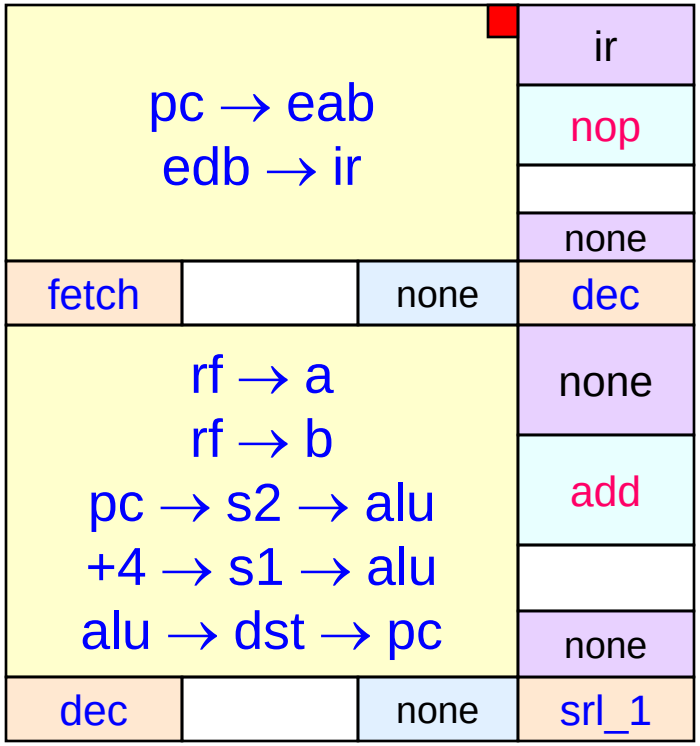
Merged Level 2 Hardware Flowchart

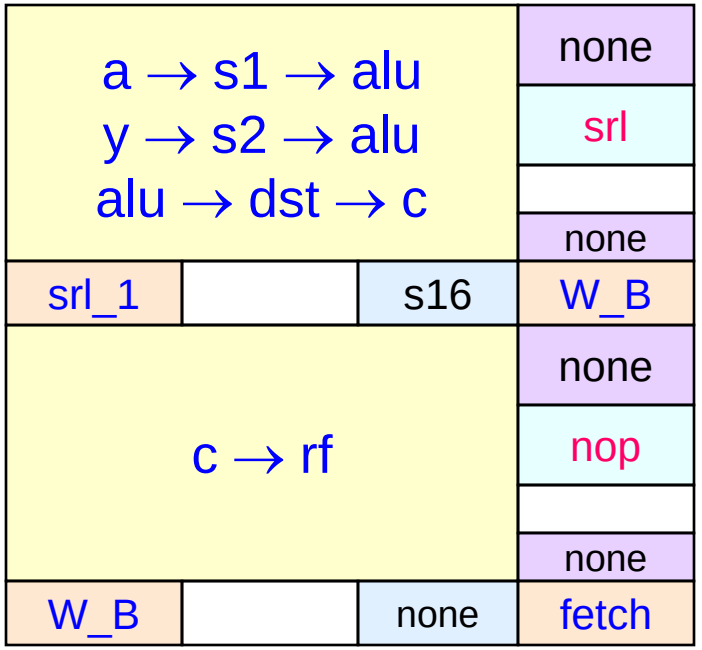
SRL Rd, Rs1, Rs2



Merged Level 2 Hardware Flowchart

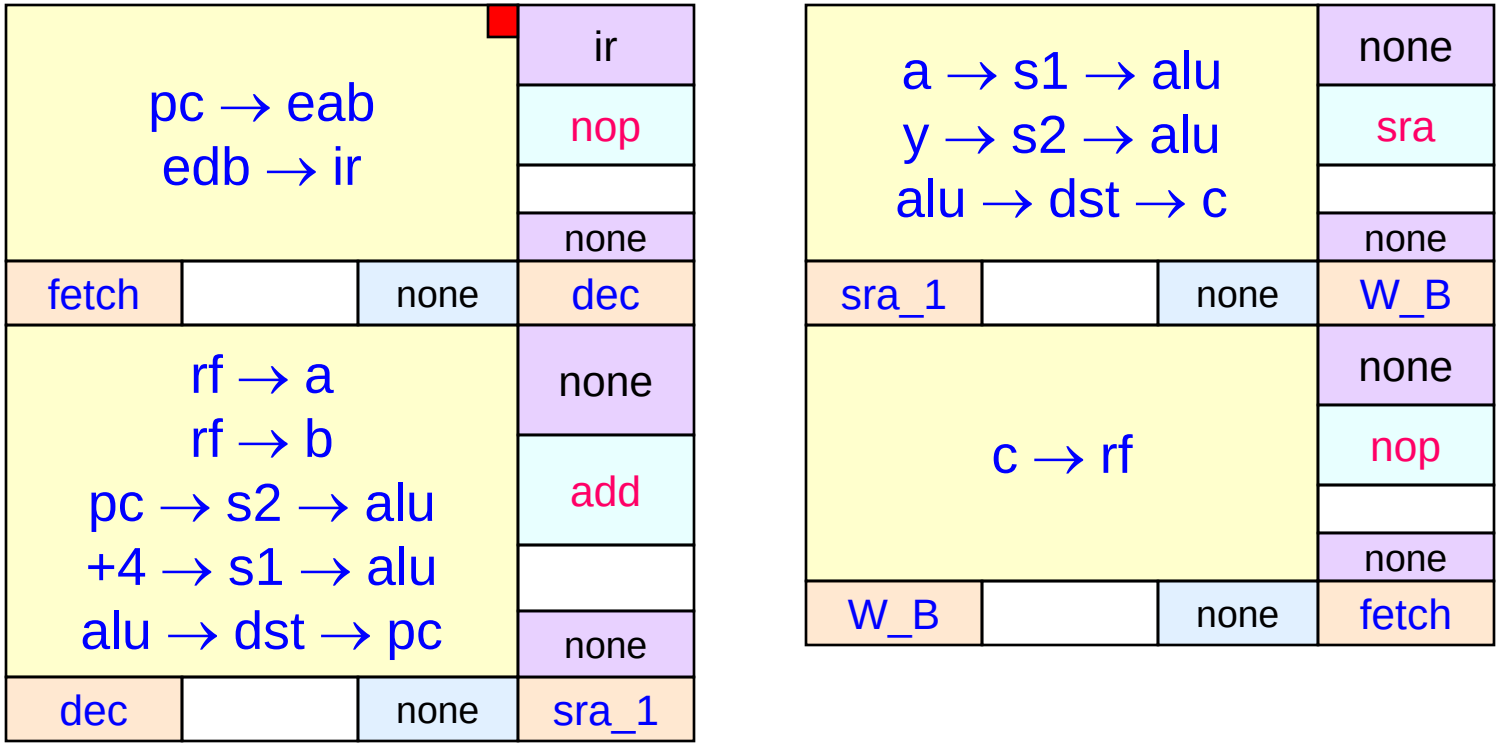
SRL.I Rd, Rs1, Imm





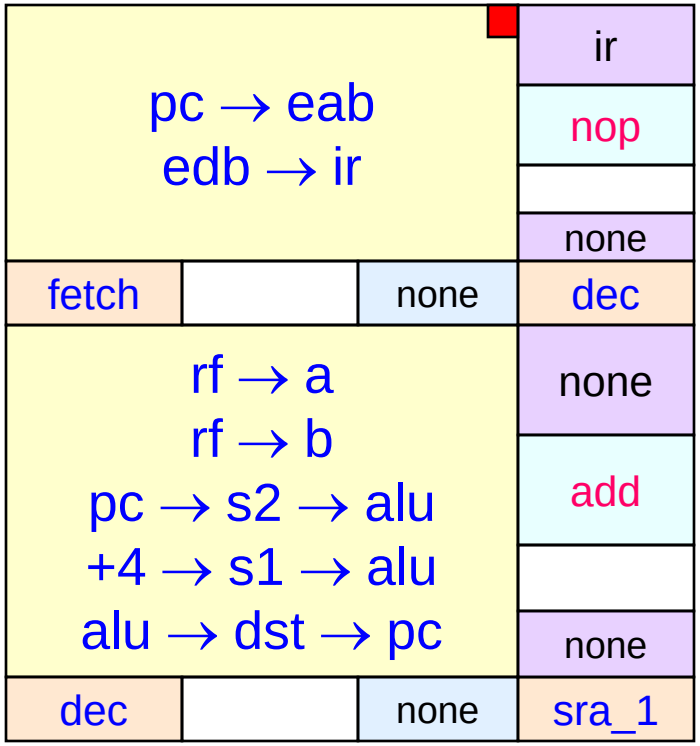
Merged Level 2 Hardware Flowchart

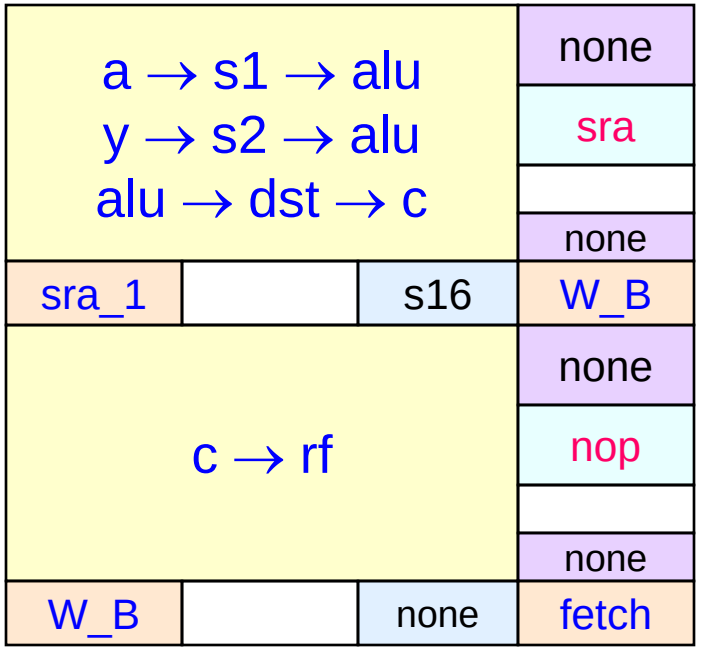
SRA Rd, Rs1, Rs2



Merged Level 2 Hardware Flowchart

SRA.I Rd, Rs1, Imm

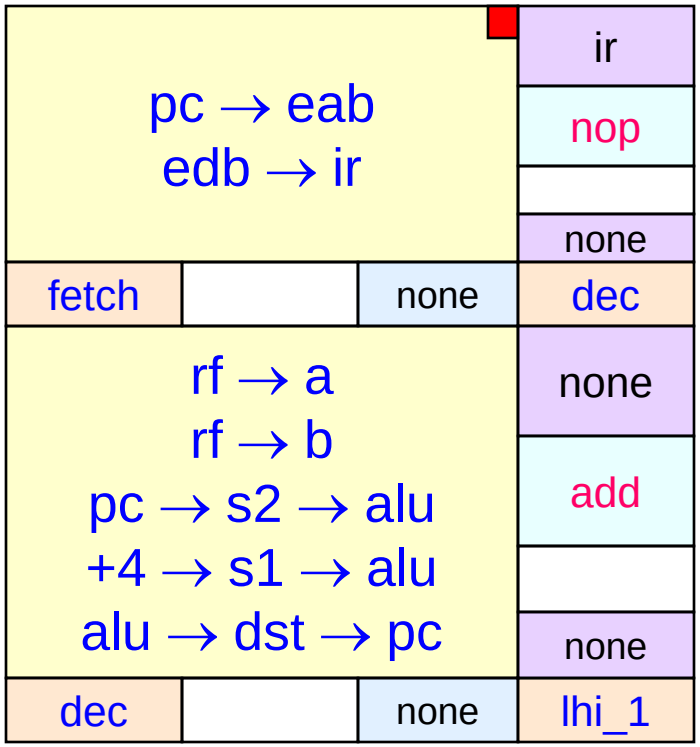


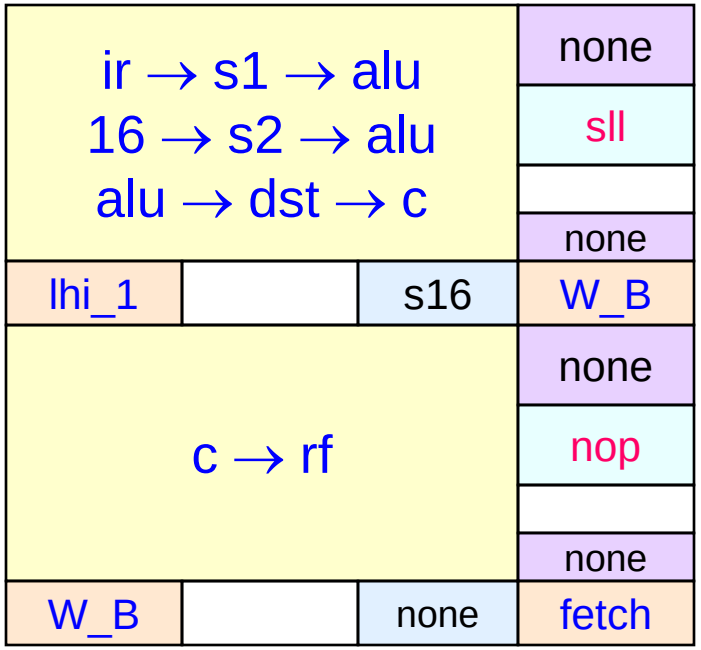


Merged Level 2 Hardware Flowchart

LHI Rd, Imm

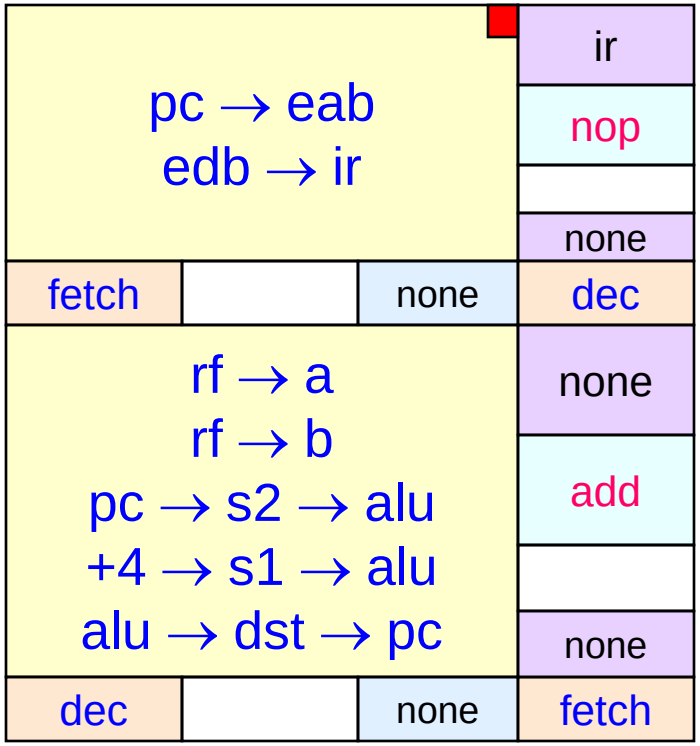
Rd(0:15) <- Imm; Rd(16:31) <- hex0000





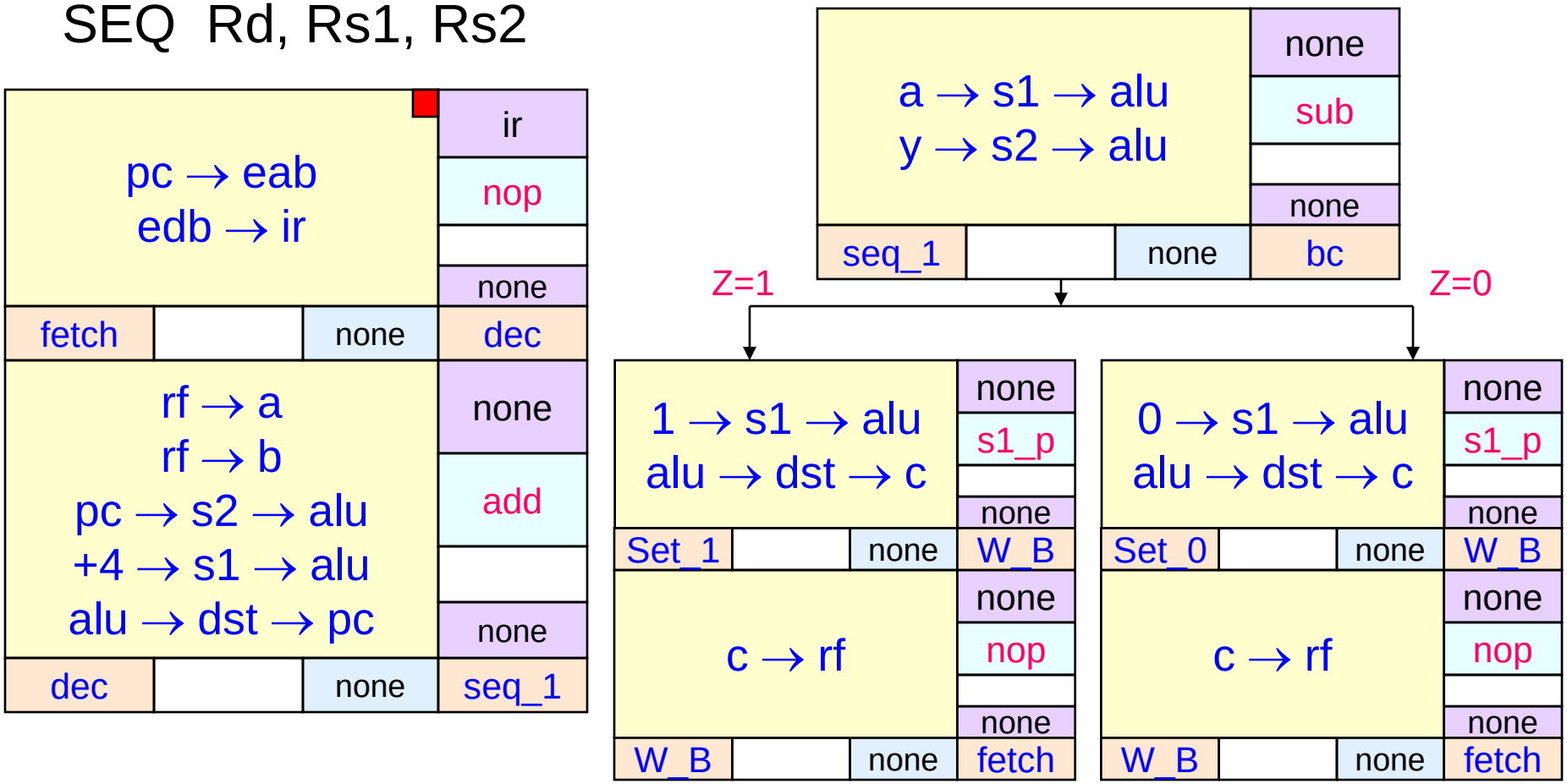
Merged Level 2 Hardware Flowchart

NOP



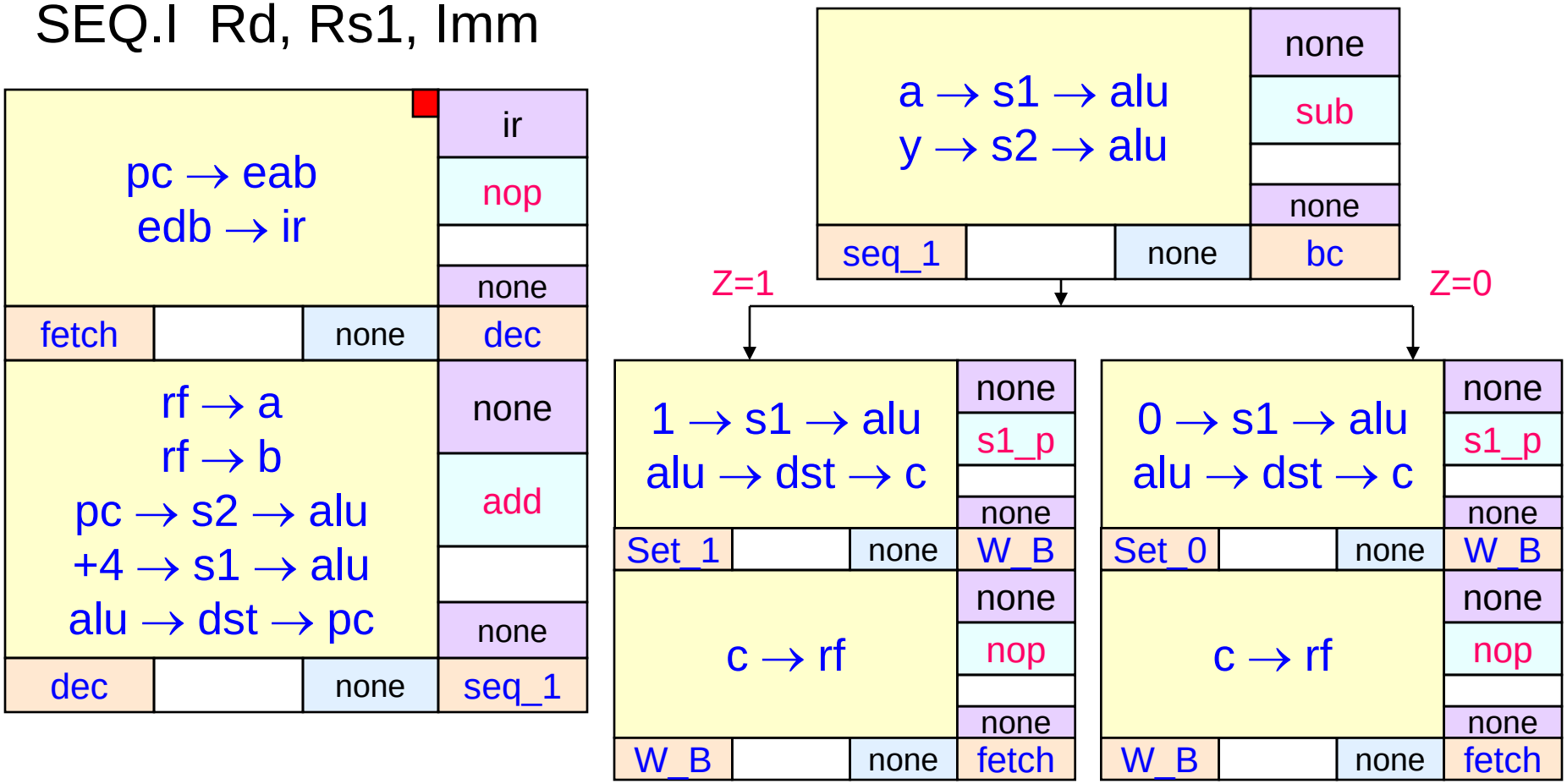
Merged Level 2 Hardware Flowchart

SEQ Rd, Rs1, Rs2



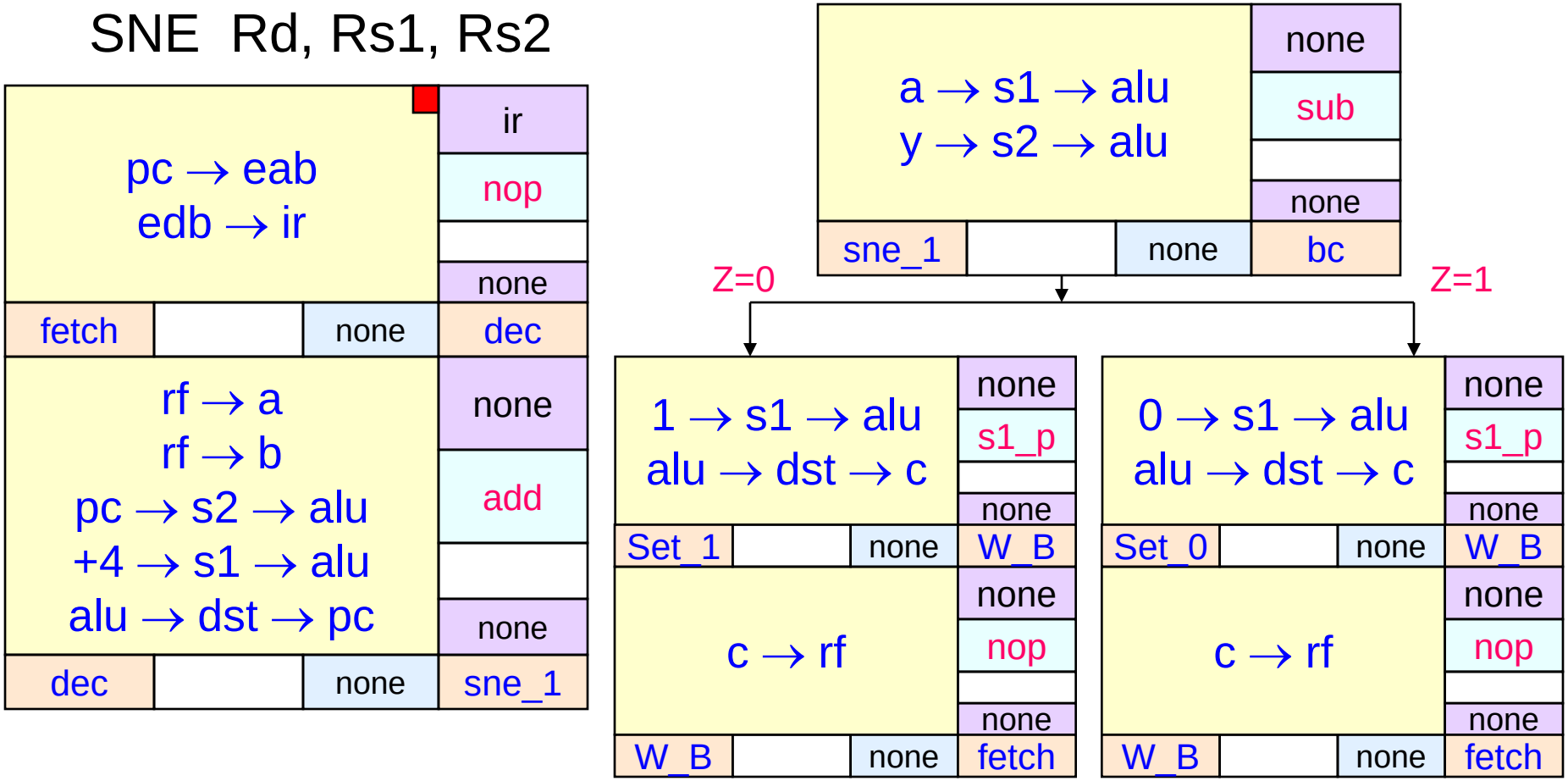
Merged Level 2 Hardware Flowchart

SEQ.I Rd, Rs1, Imm



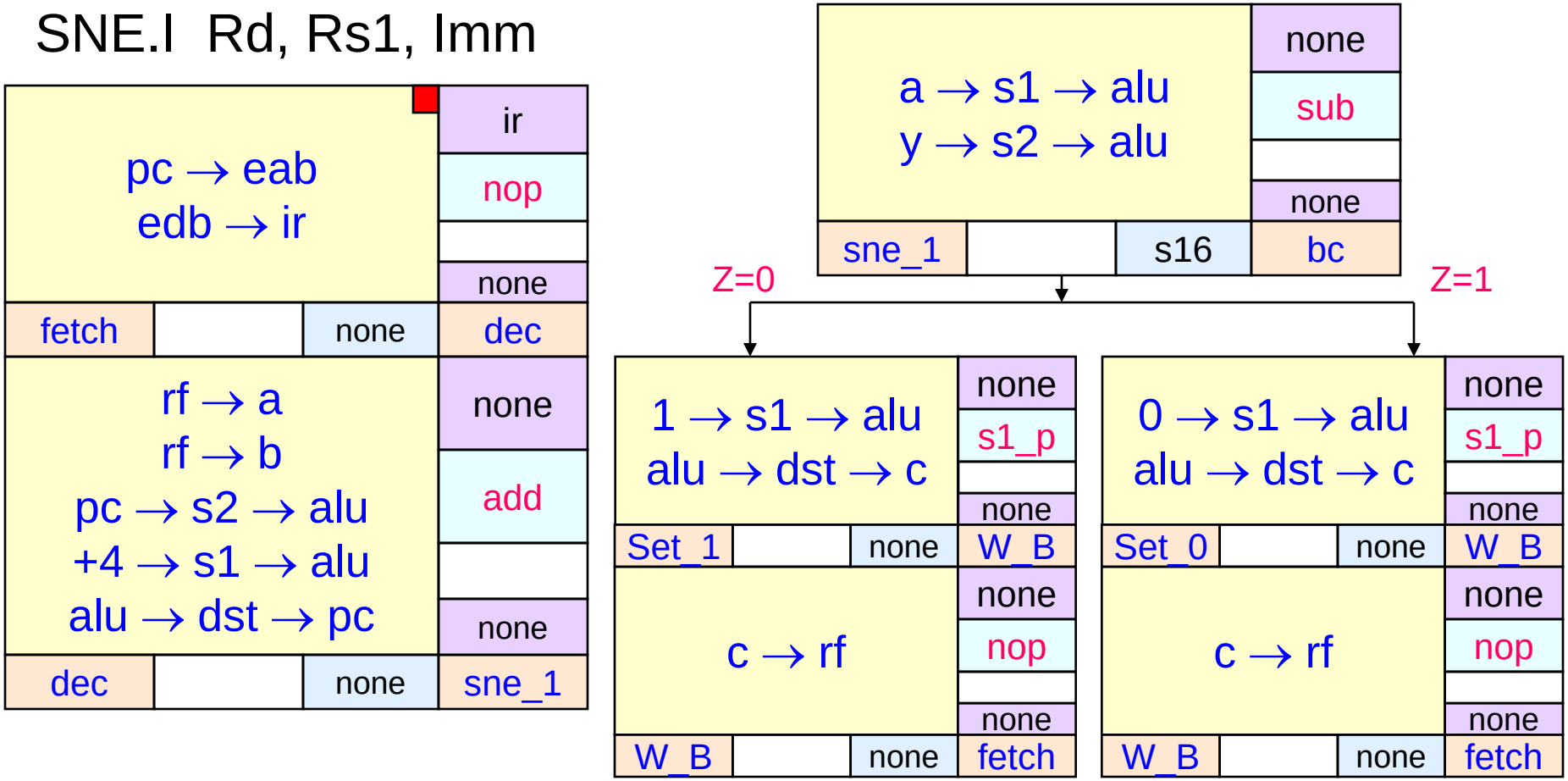
Merged Level 2 Hardware Flowchart

SNE Rd, Rs1, Rs2



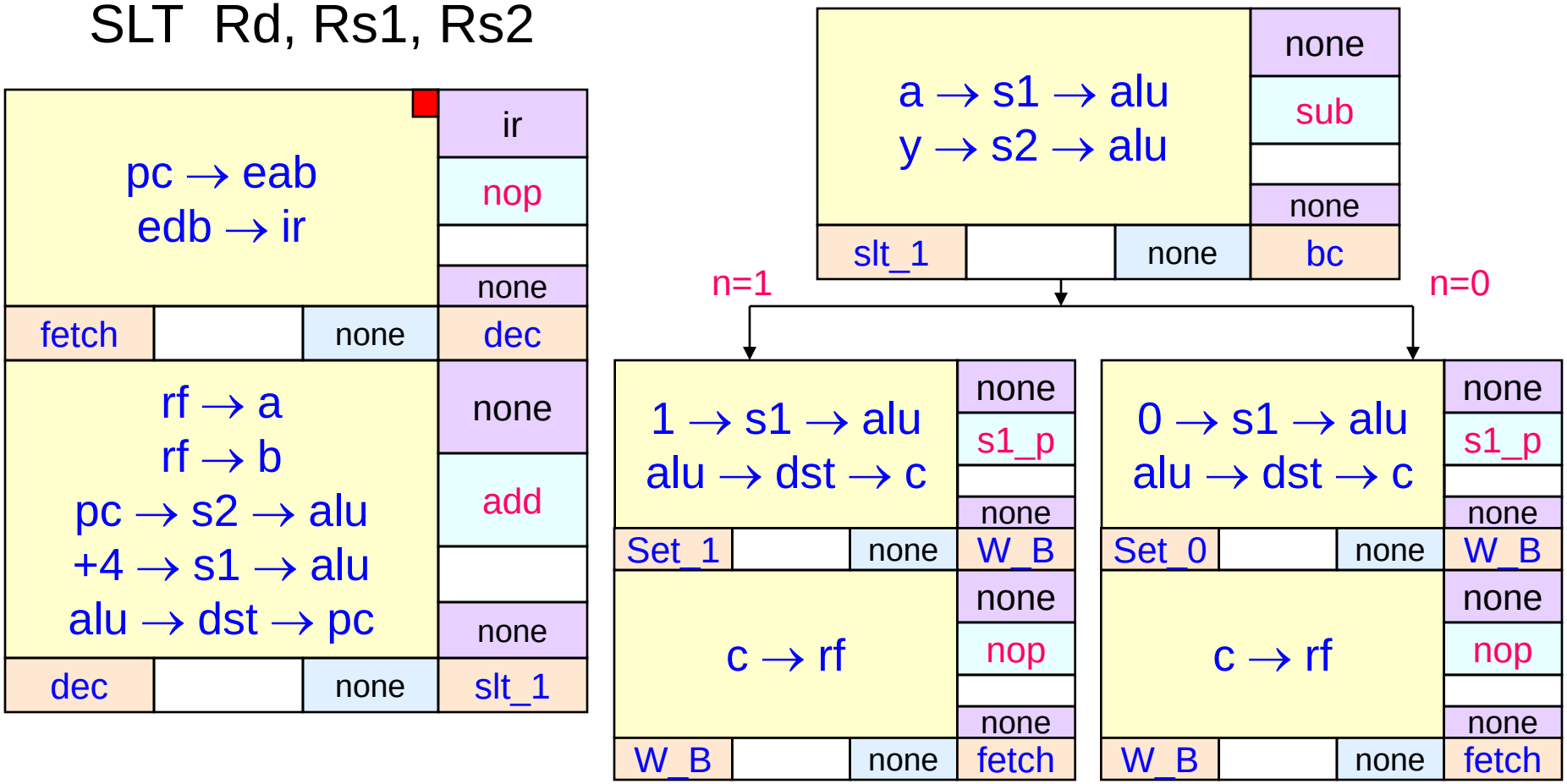
Merged Level 2 Hardware Flowchart

SNE.I Rd, Rs1, Imm



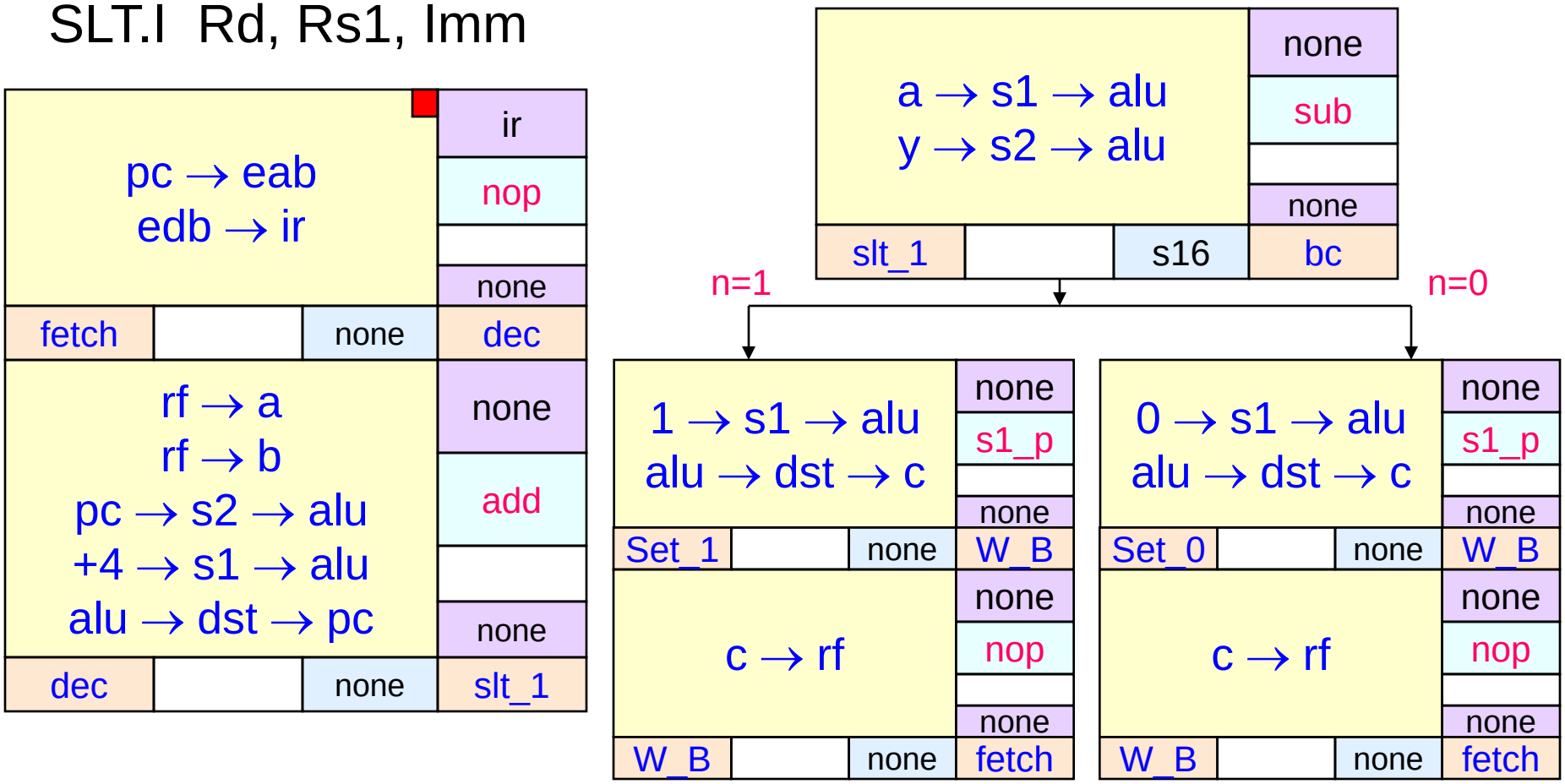
Merged Level 2 Hardware Flowchart

SLT Rd, Rs1, Rs2



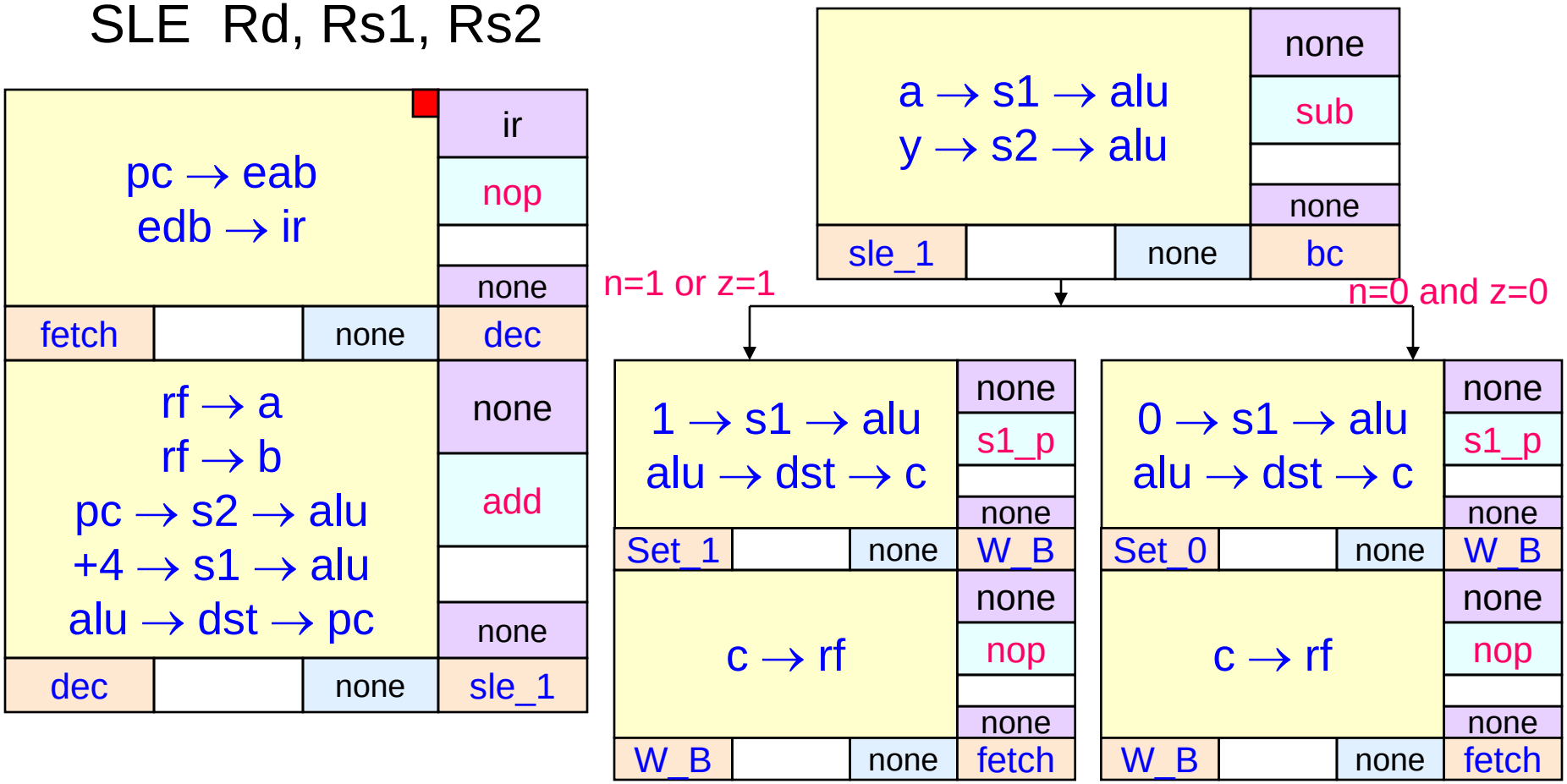
Merged Level 2 Hardware Flowchart

SLT.I Rd, Rs1, Imm



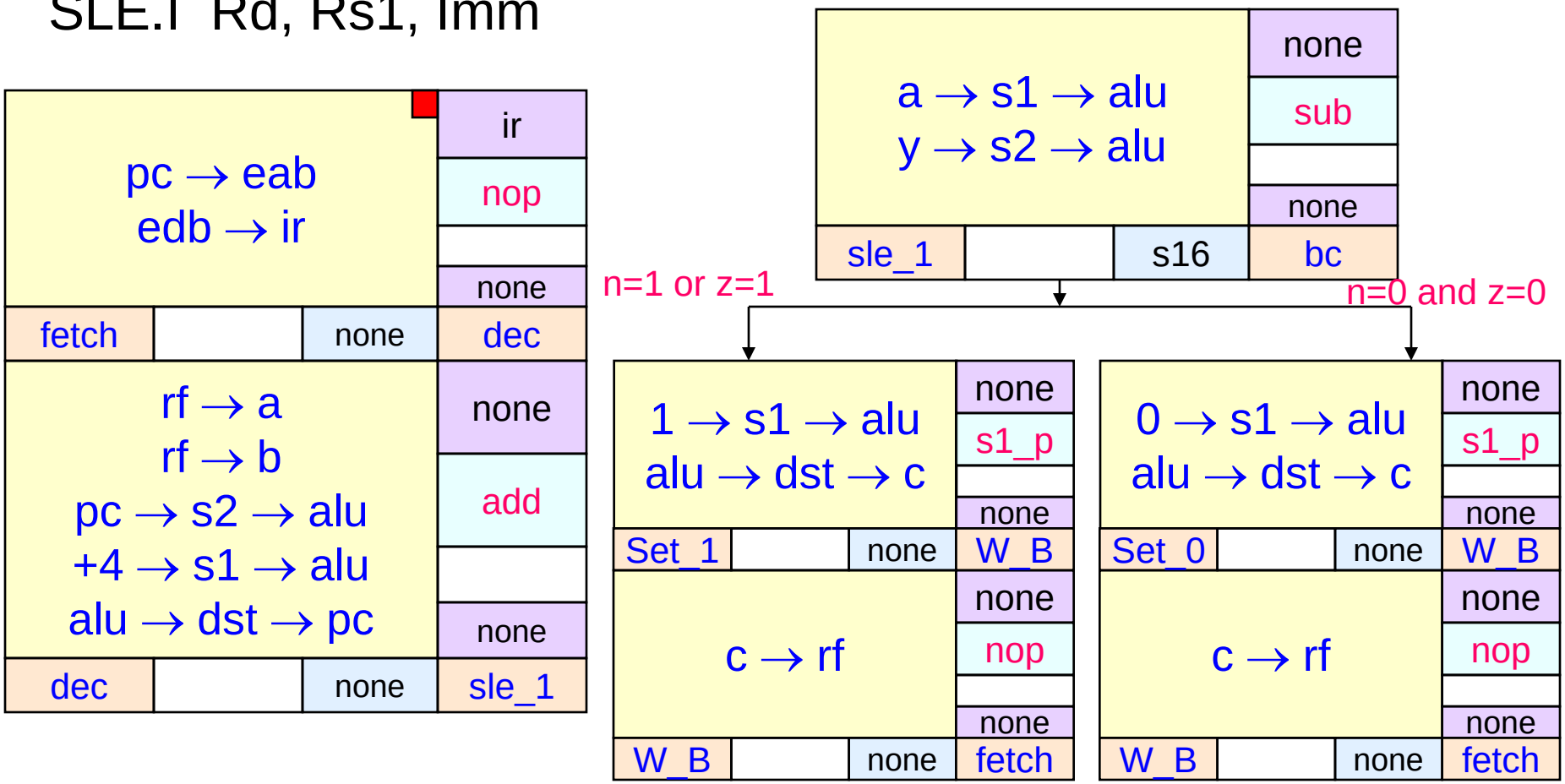
Merged Level 2 Hardware Flowchart

SLE Rd, Rs1, Rs2



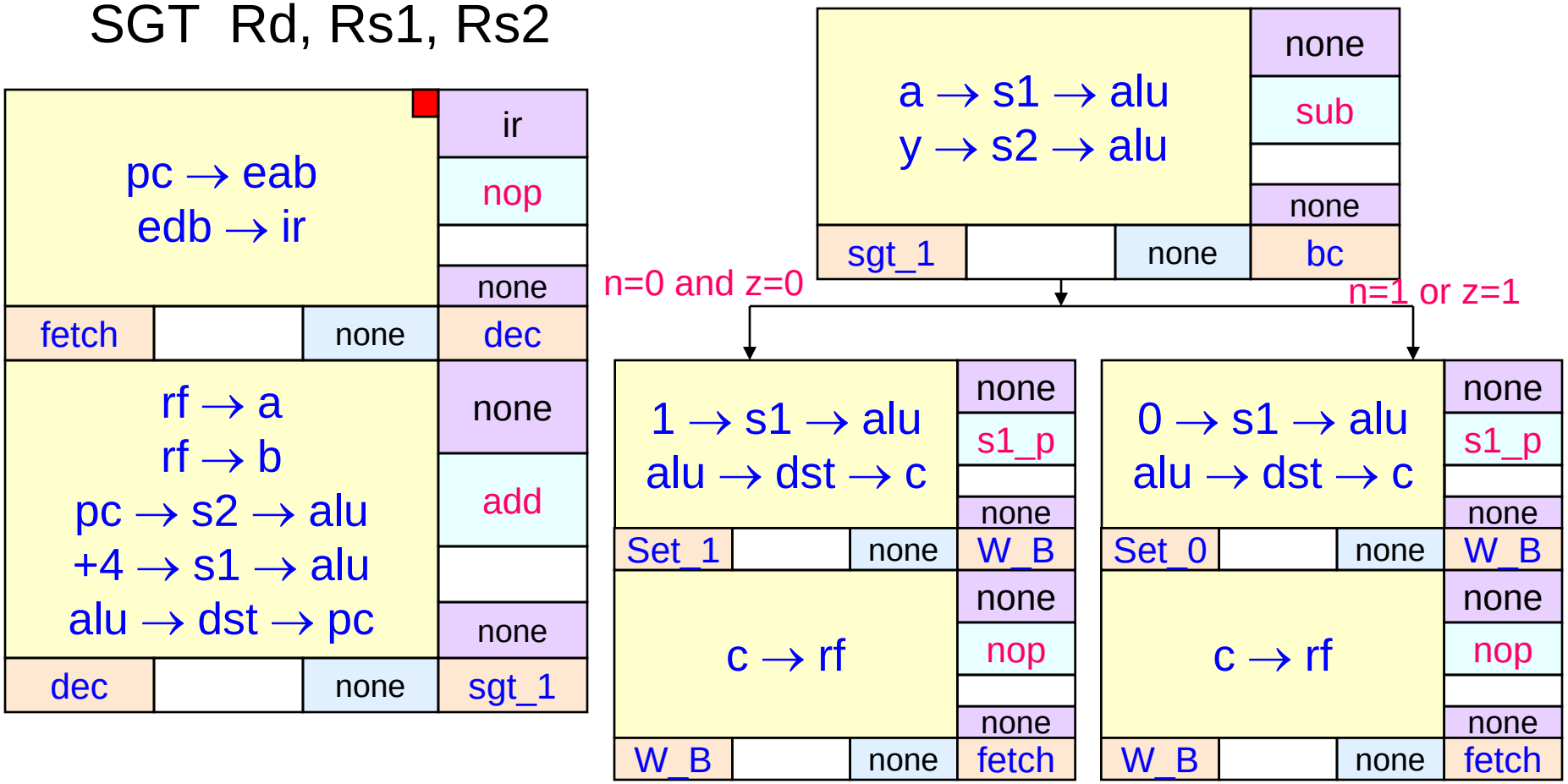
Merged Level 2 Hardware Flowchart

SLE.I Rd, Rs1, Imm



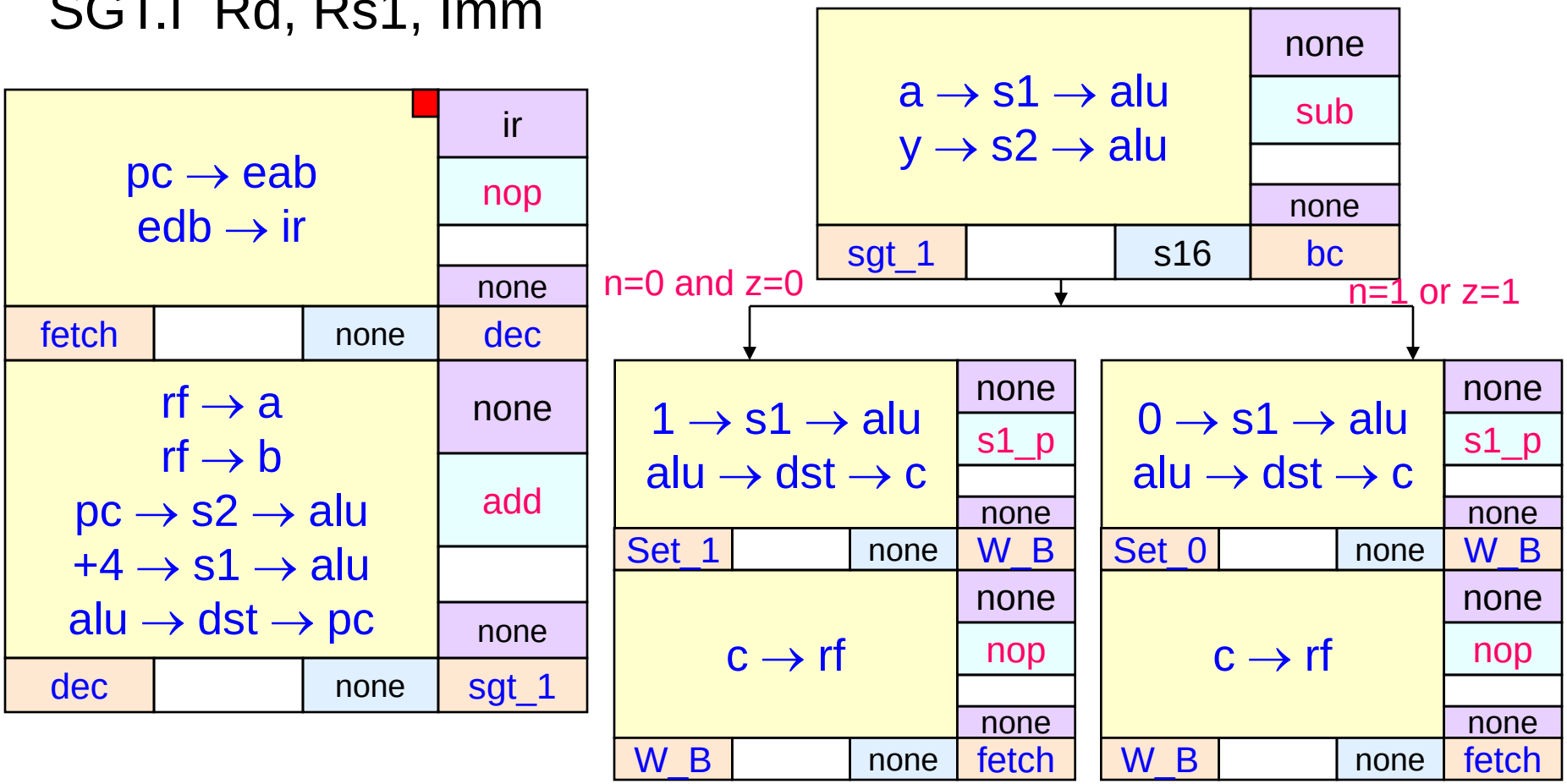
Merged Level 2 Hardware Flowchart

SGT Rd, Rs1, Rs2



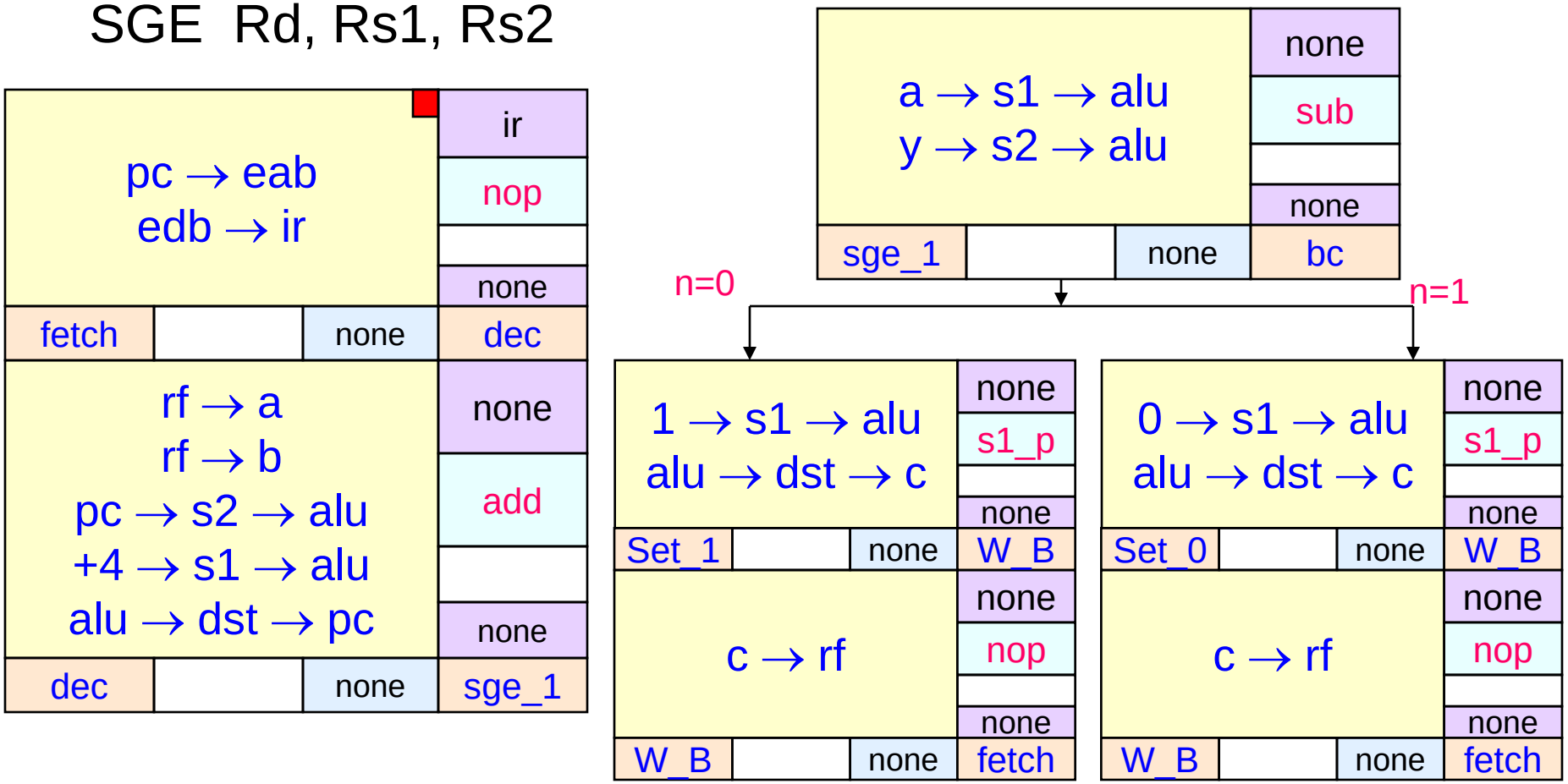
Merged Level 2 Hardware Flowchart

SGT.I Rd, Rs1, Imm



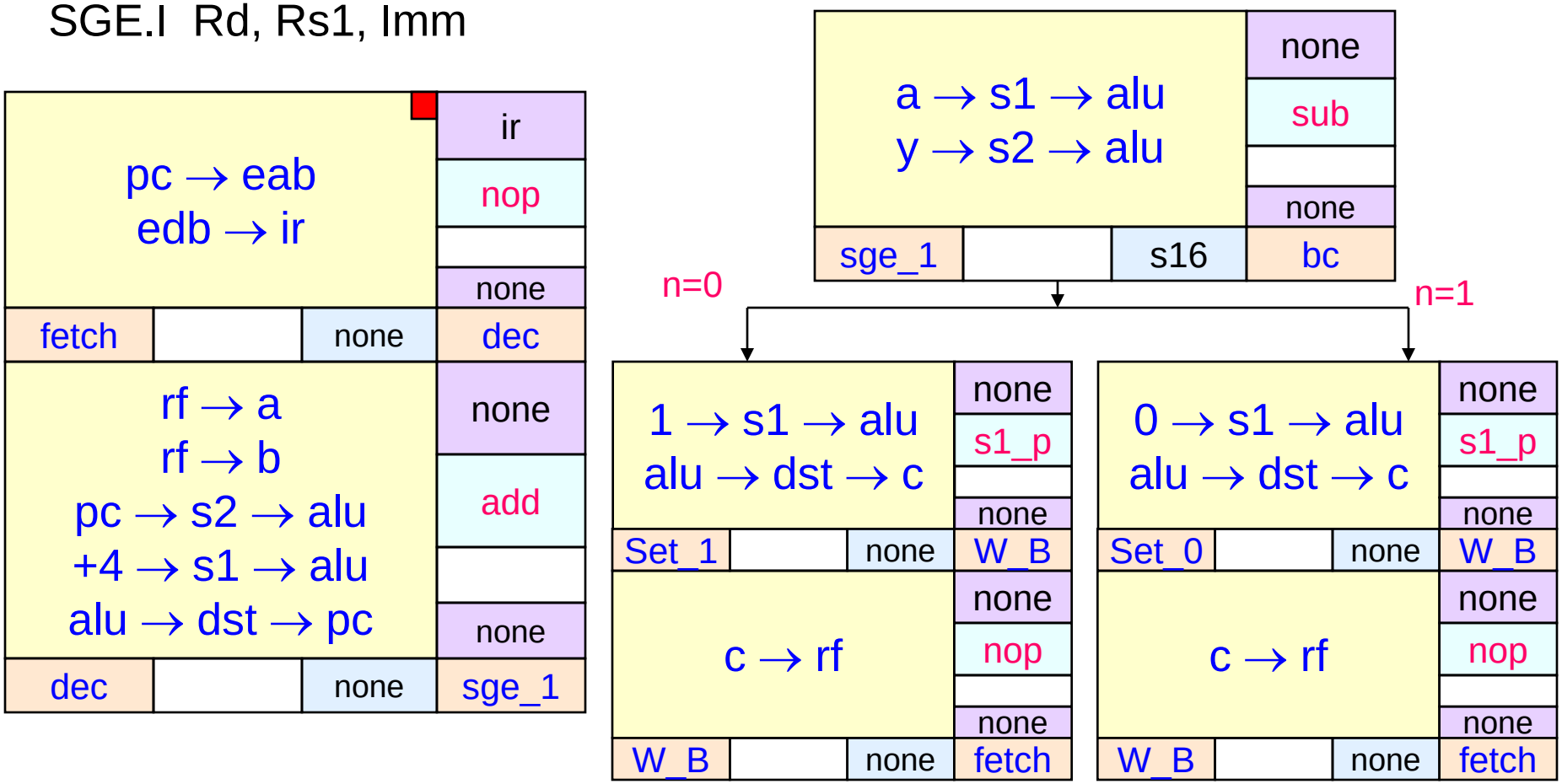
Merged Level 2 Hardware Flowchart

SGE Rd, Rs1, Rs2



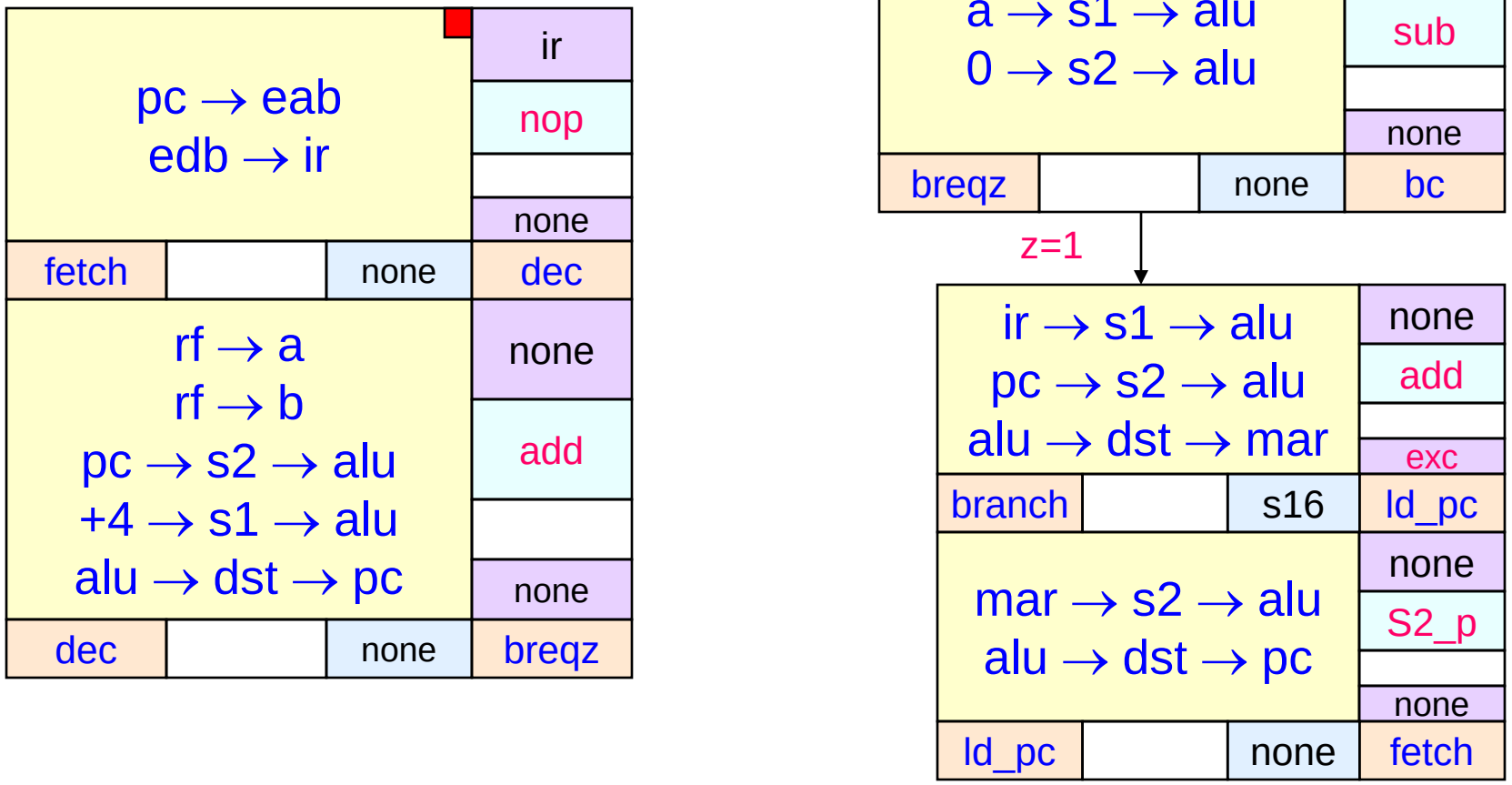
Merged Level 2 Hardware Flowchart

SGE.I Rd, Rs1, Imm



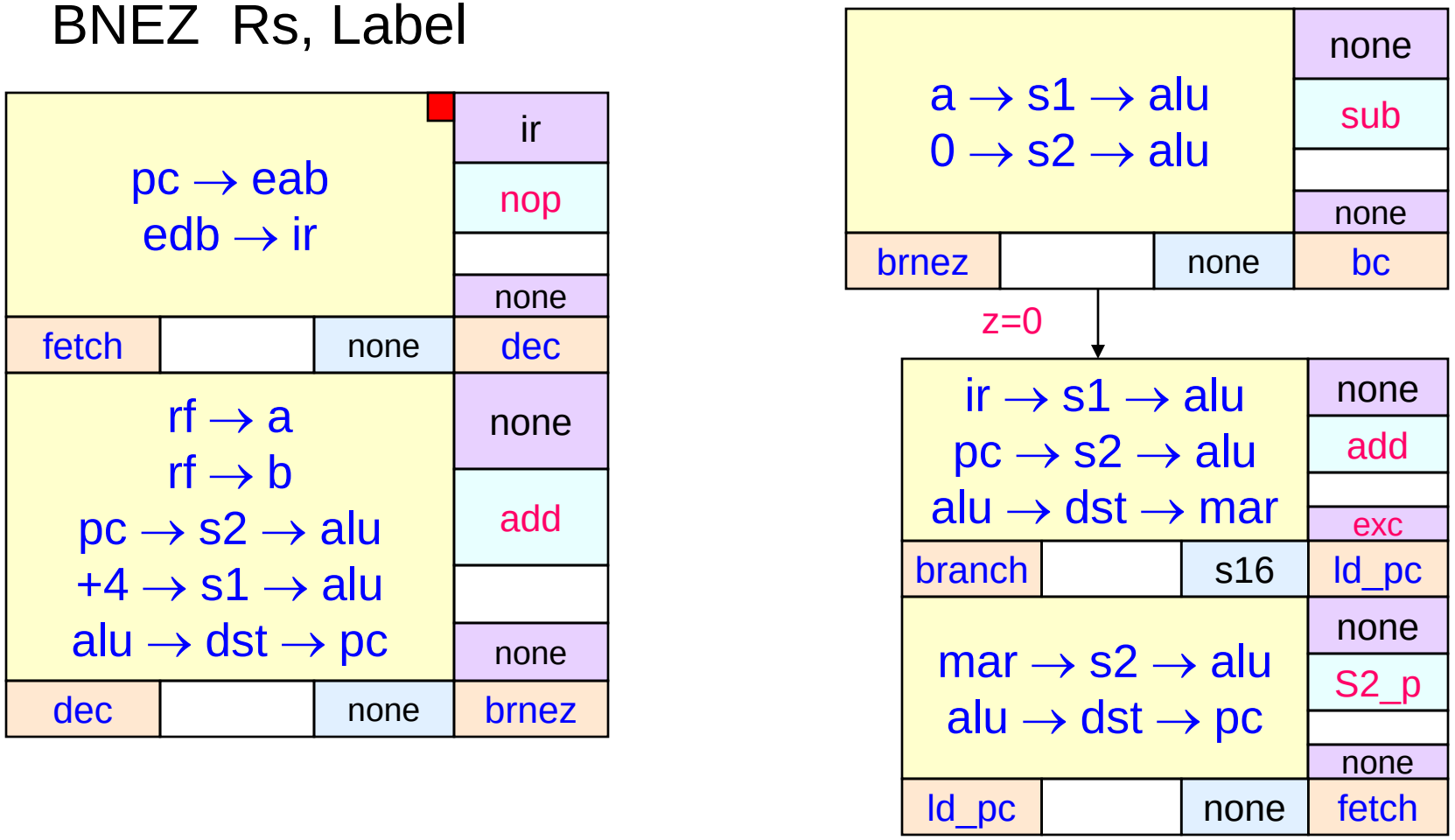
Merged Level 2 Hardware Flowchart

BEQZ Rs, Label



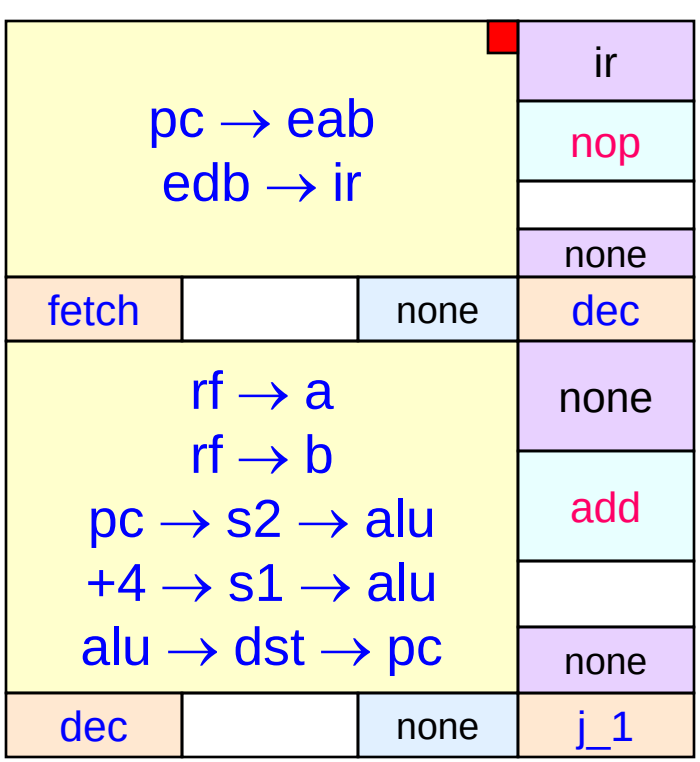
Merged Level 2 Hardware Flowchart

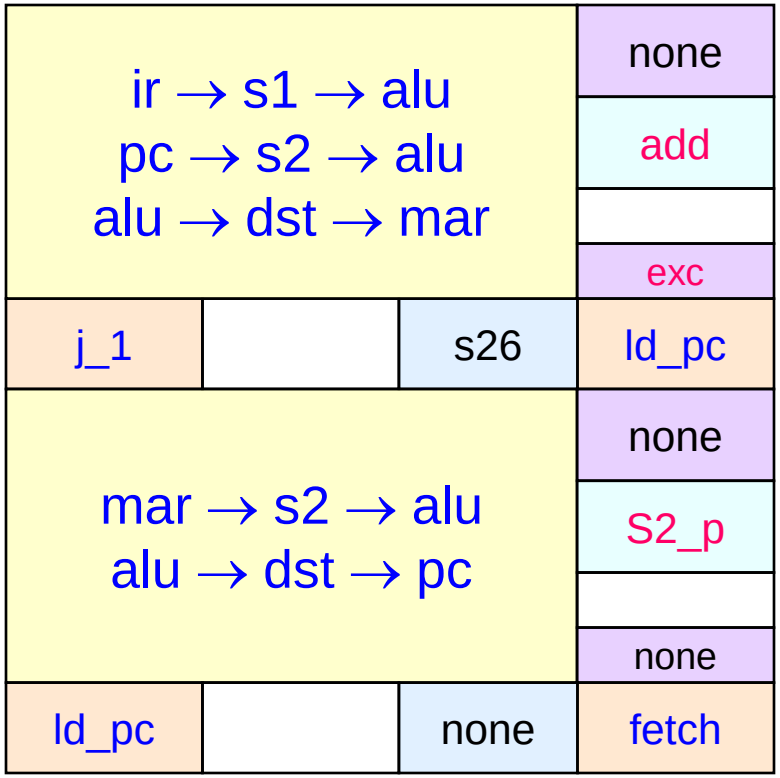
BNEZ Rs, Label



Merged Level 2 Hardware Flowchart

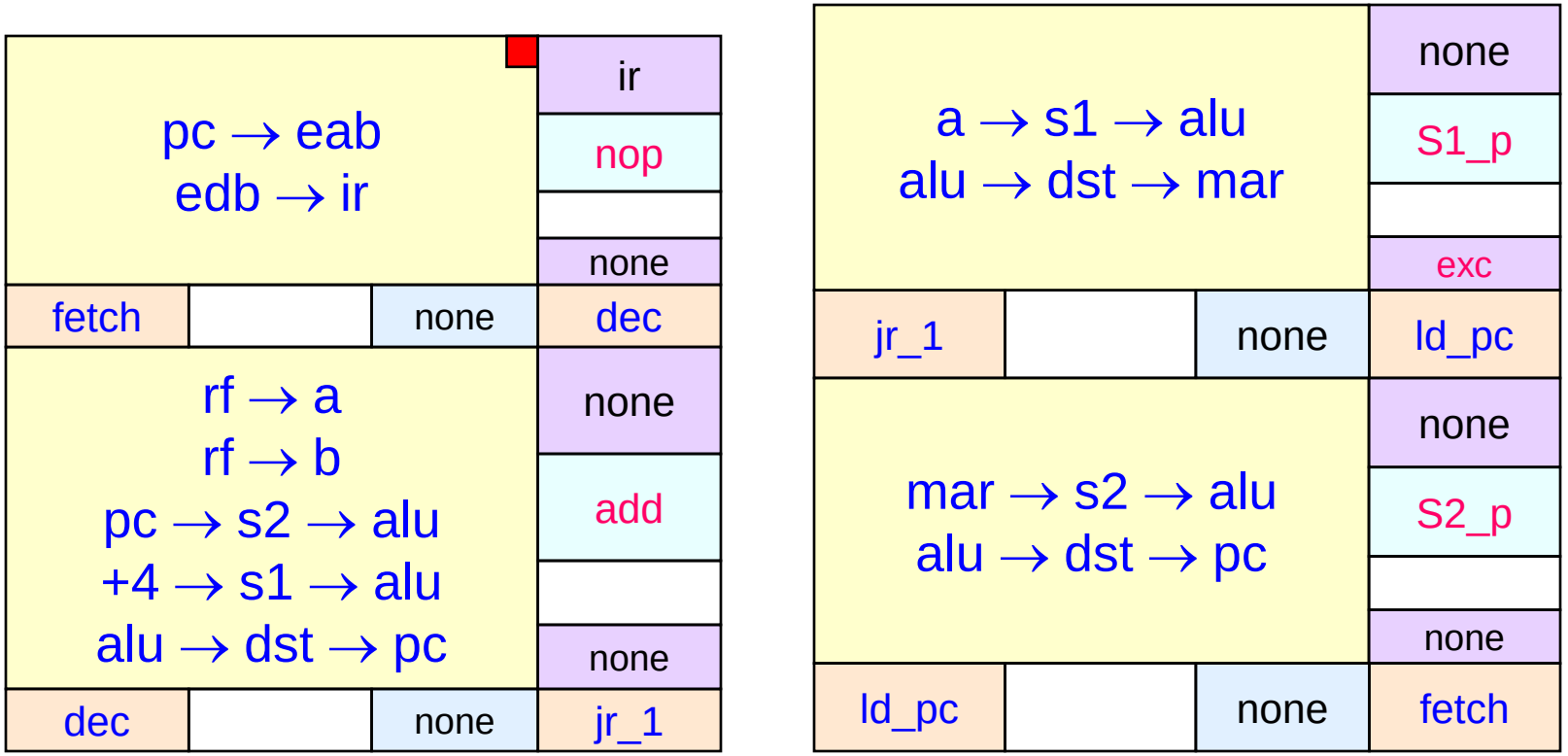
J Label



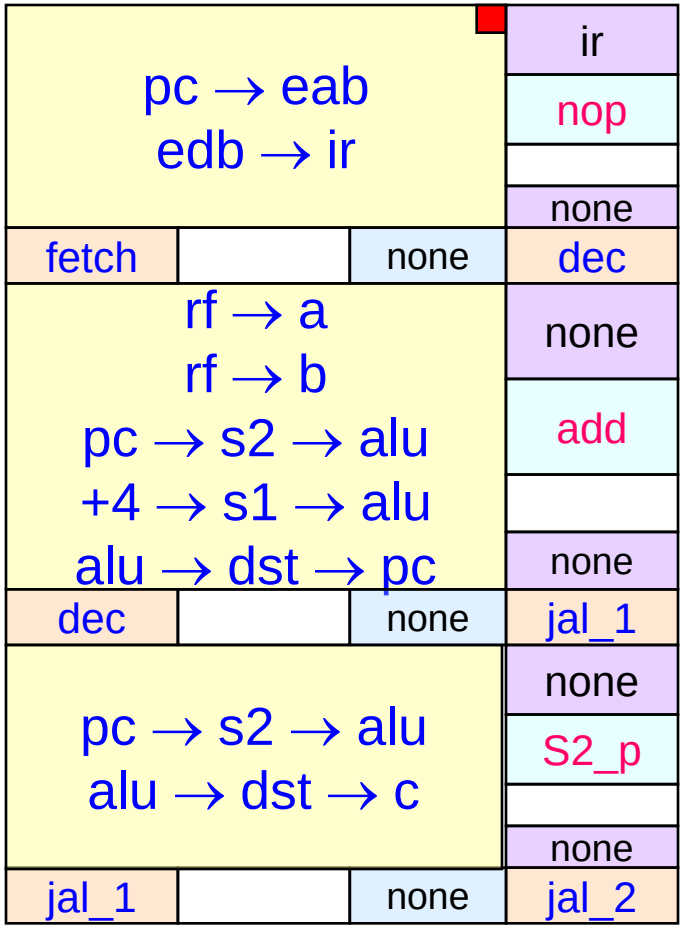


Merged Level 2 Hardware Flowchart

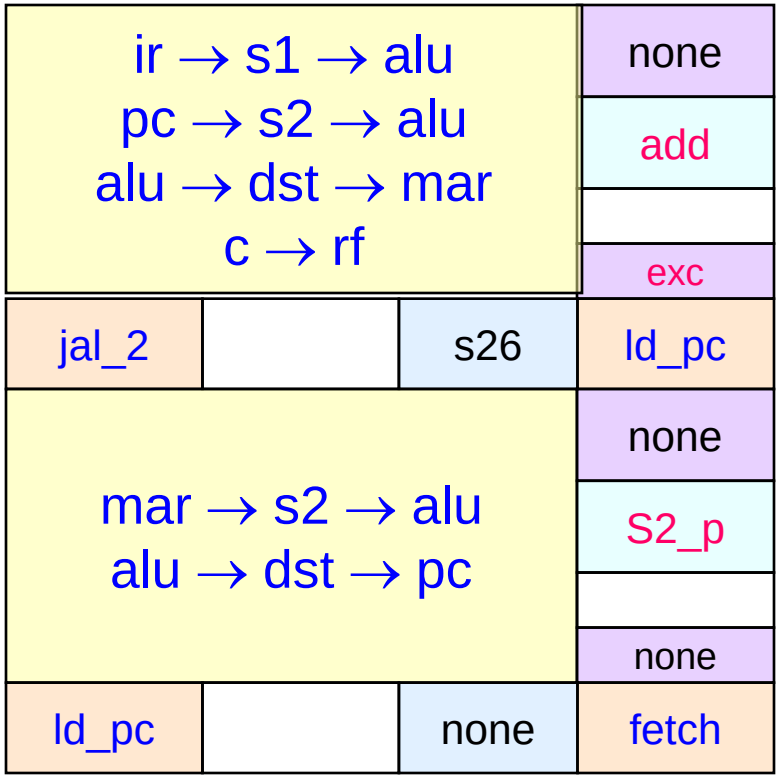
JR Rs



Merged Level 2 Hardware Flowchart

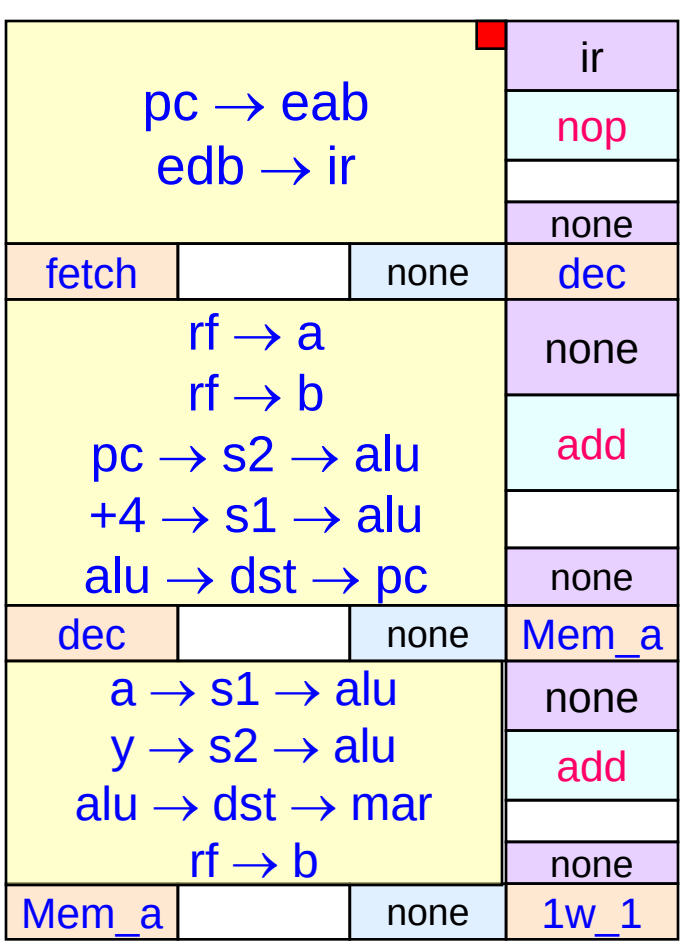


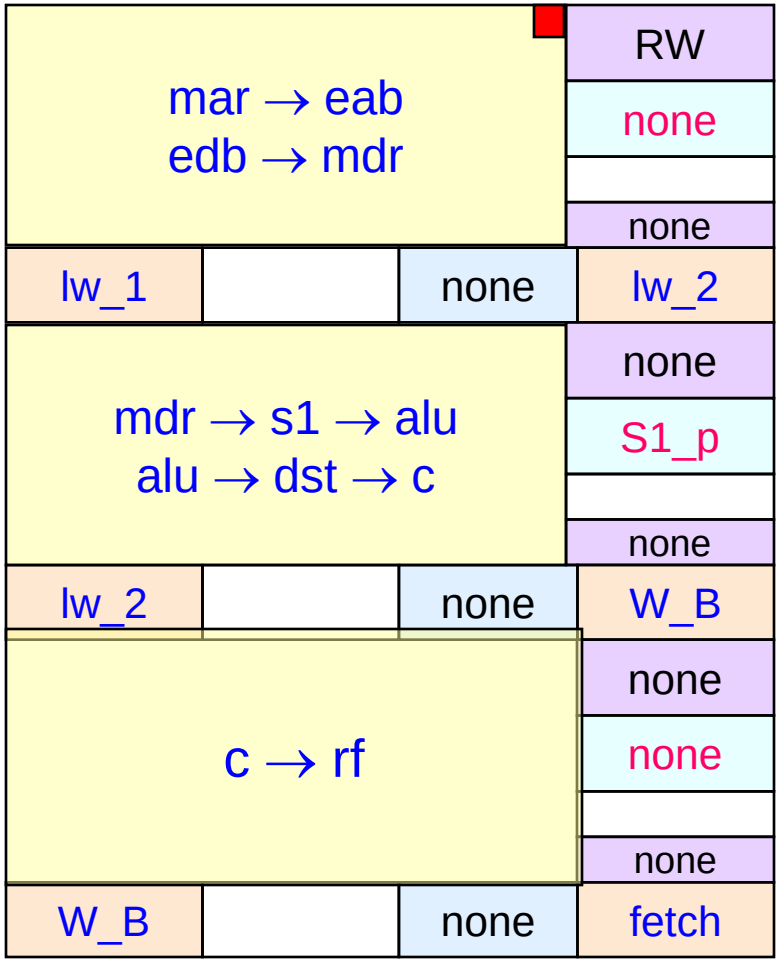
JAL Label



Merged Level 2 Hardware Flowchart

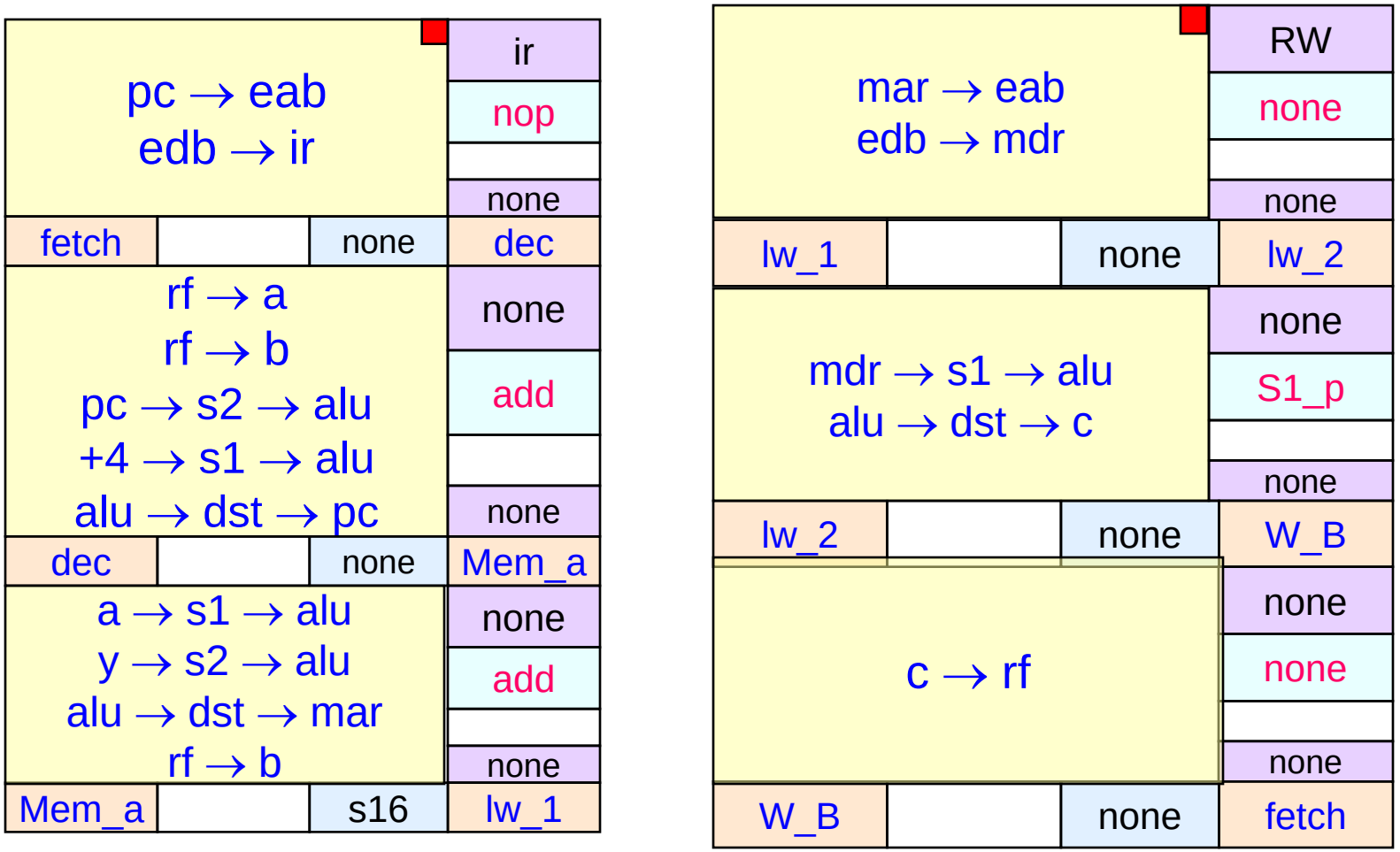
LW Rd, Rs2(Rs1)



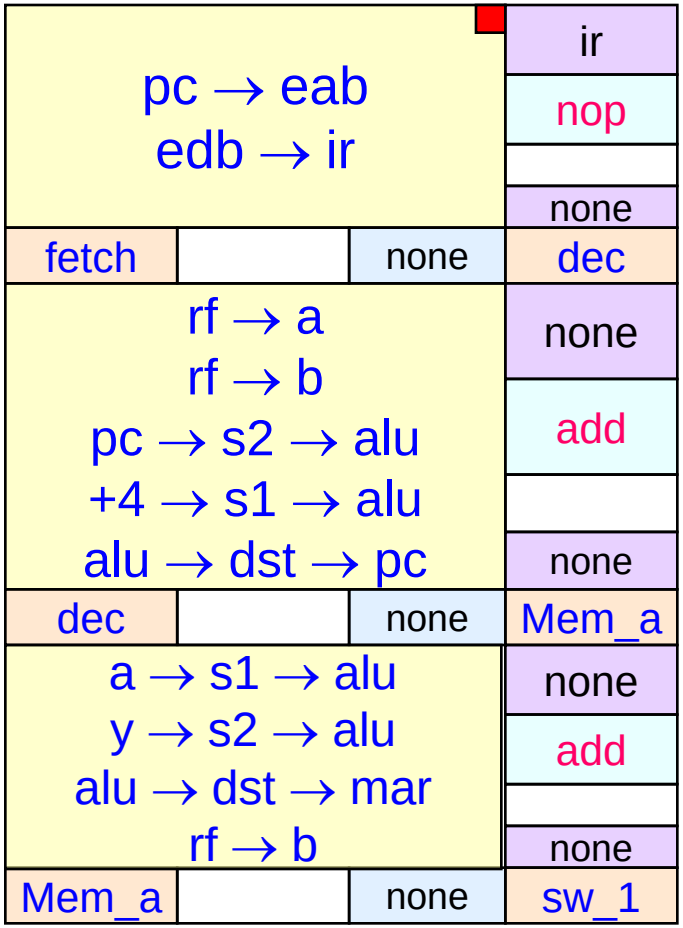


Merged Level 2 Hardware Flowchart

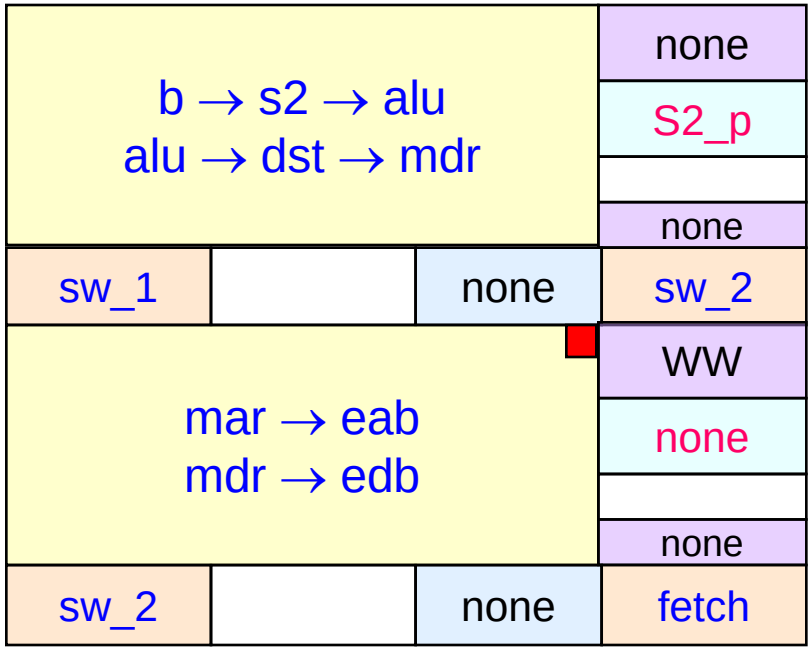
LW.I Rd, Imm(Rs1)



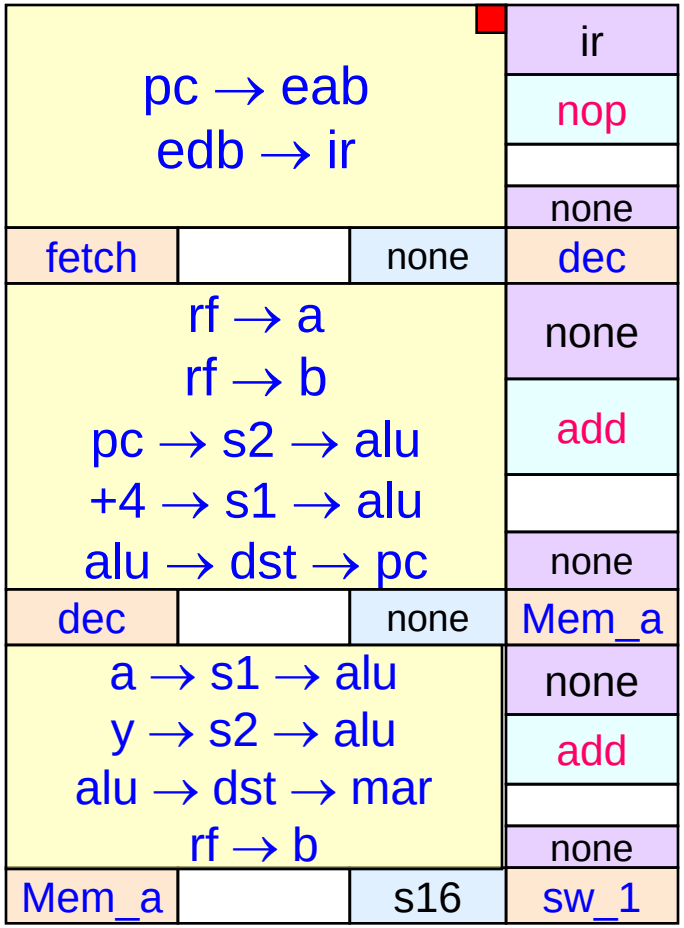
Merged Level 2 Hardware Flowchart



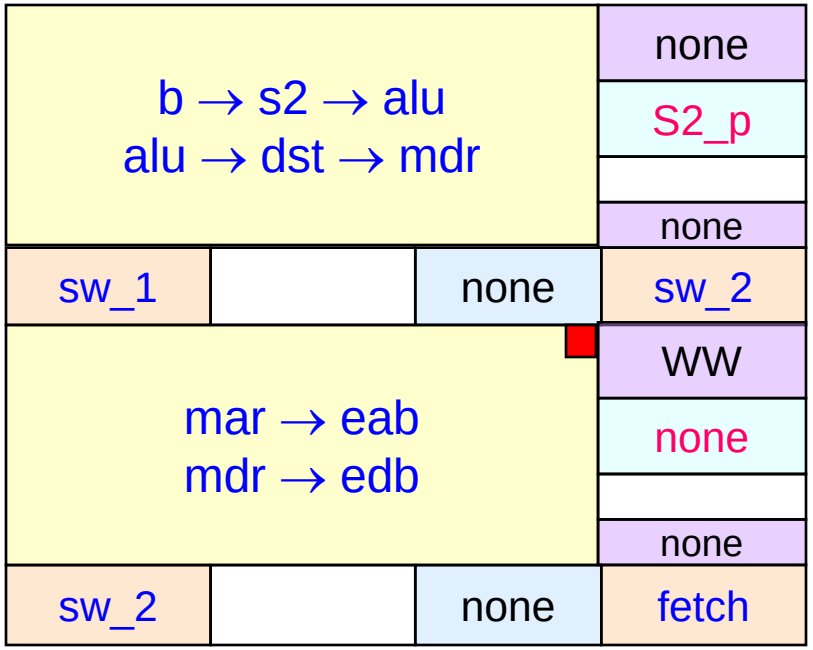
SW Rd, Rs2(Rs1)



Merged Level 2 Hardware Flowchart

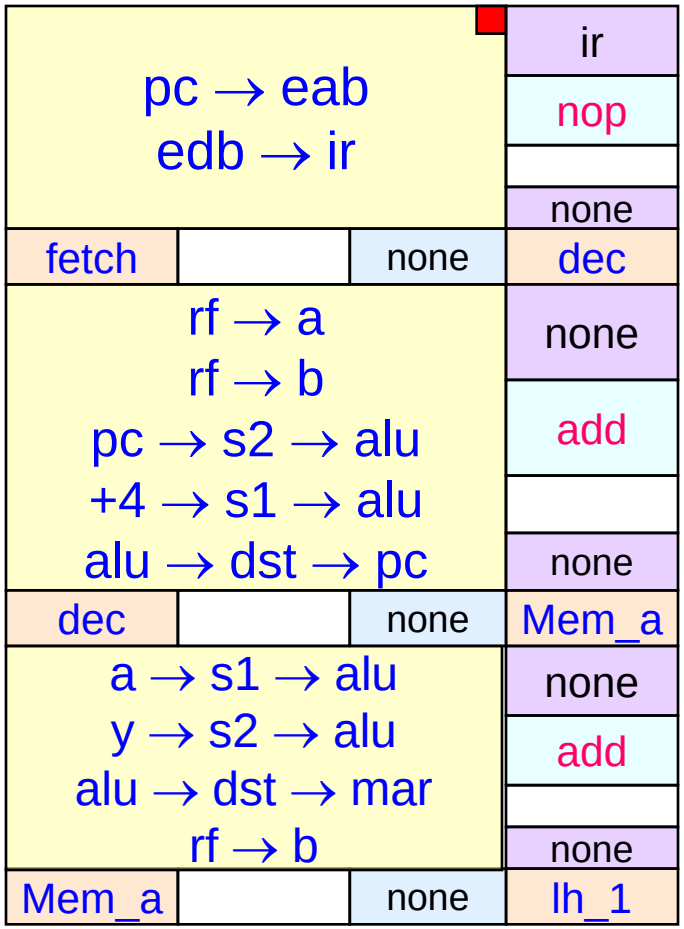


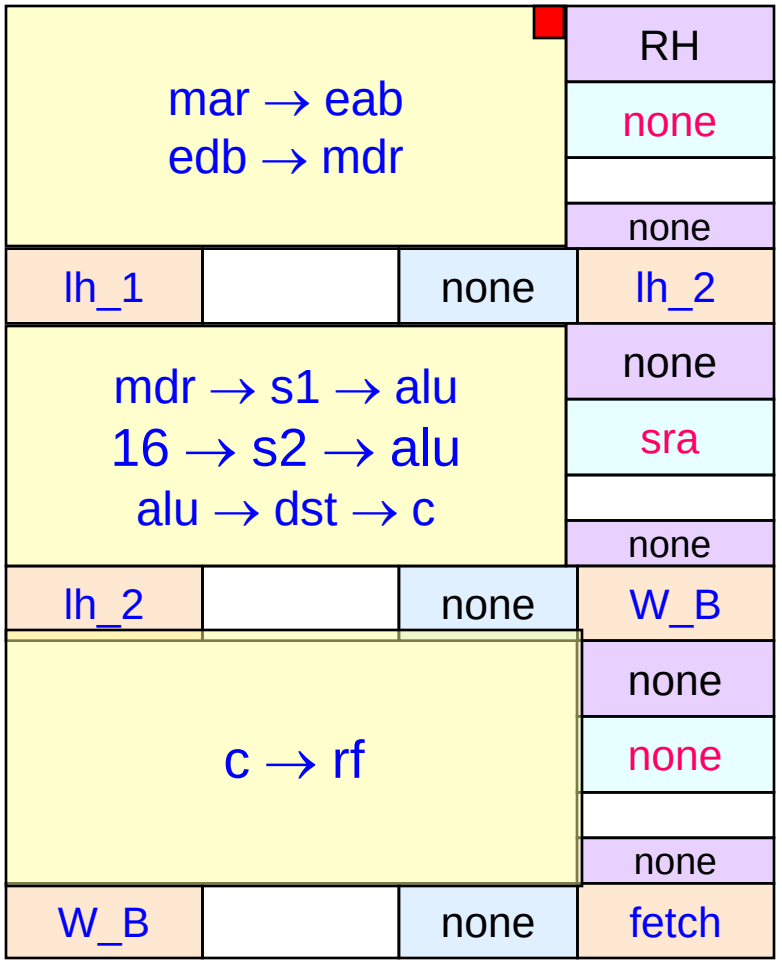
SW.I Rd, Imm(Rs1)



Merged Level 2 Hardware Flowchart

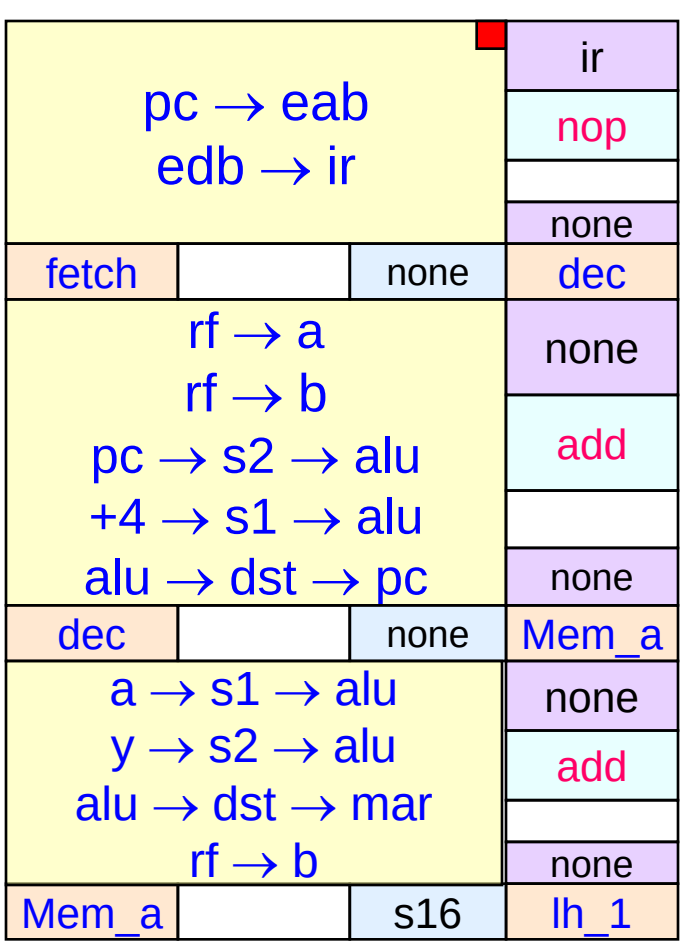
LH Rd, Rs2(Rs1)

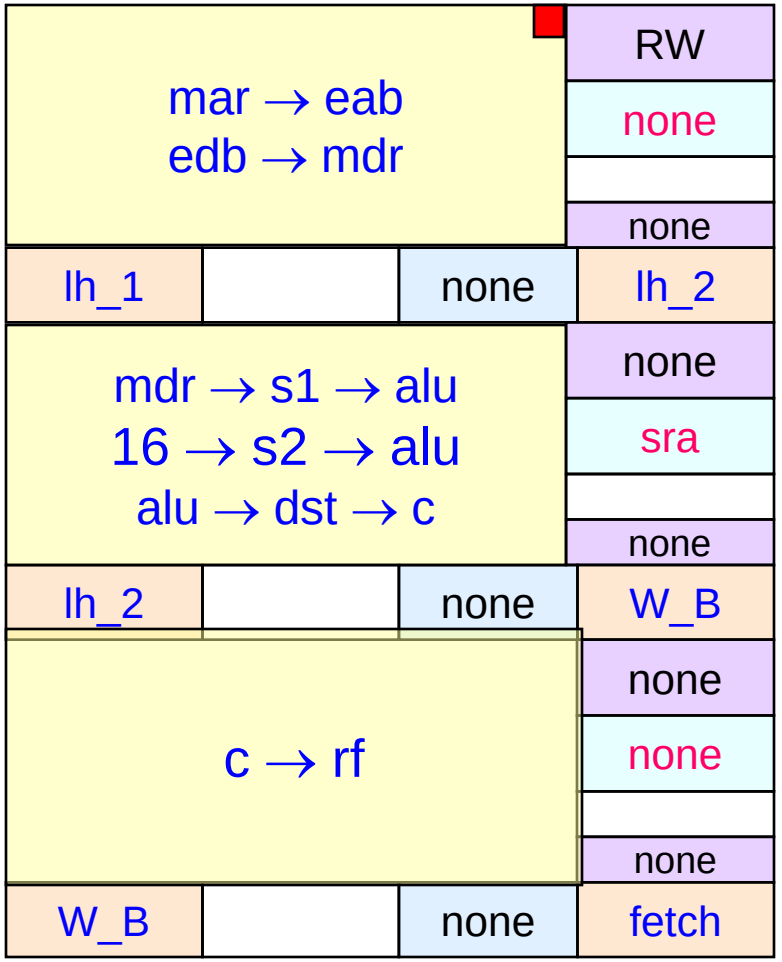




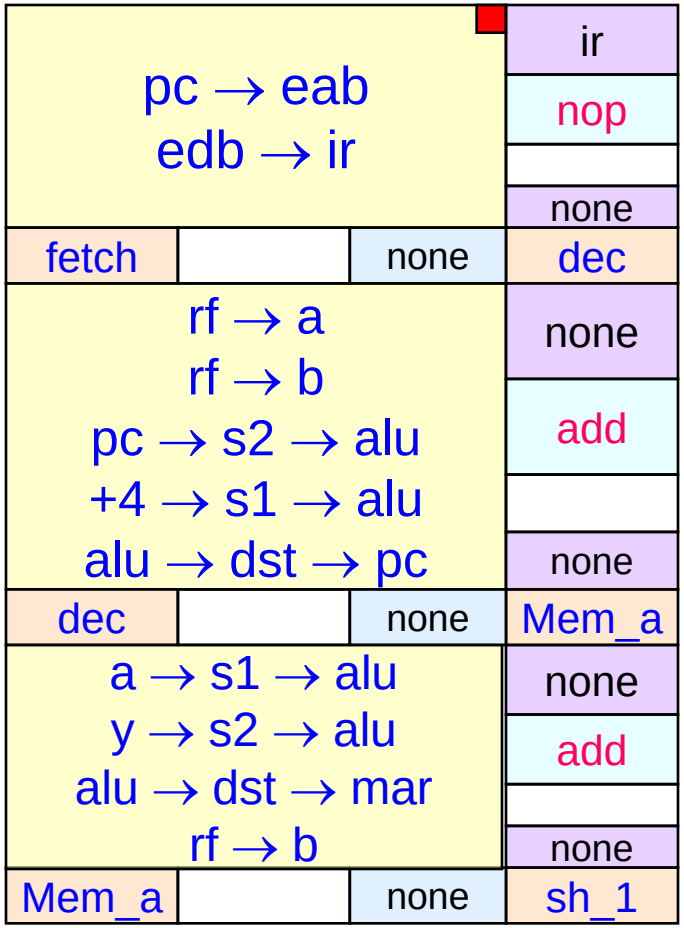
Merged Level 2 Hardware Flowchart

LH.I Rd, Imm(Rs1)

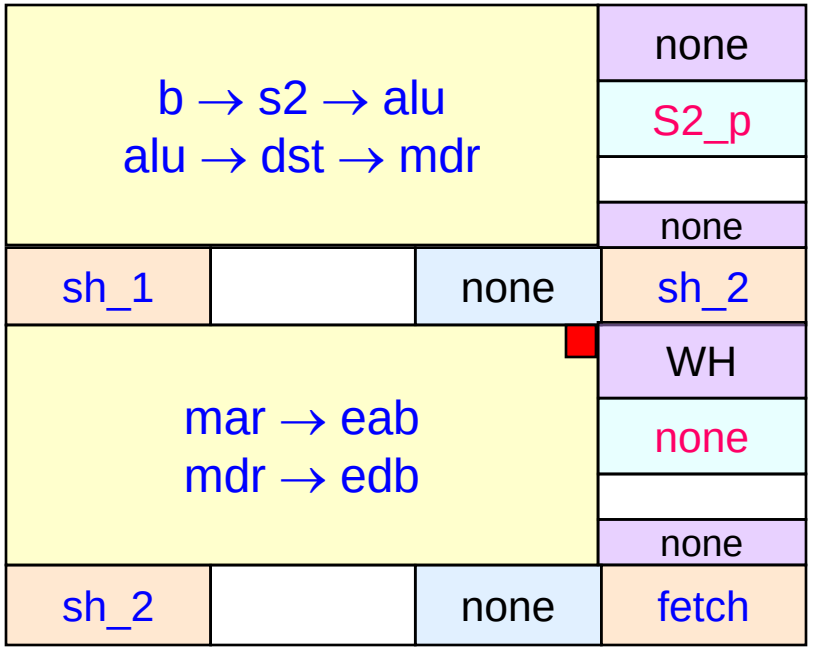




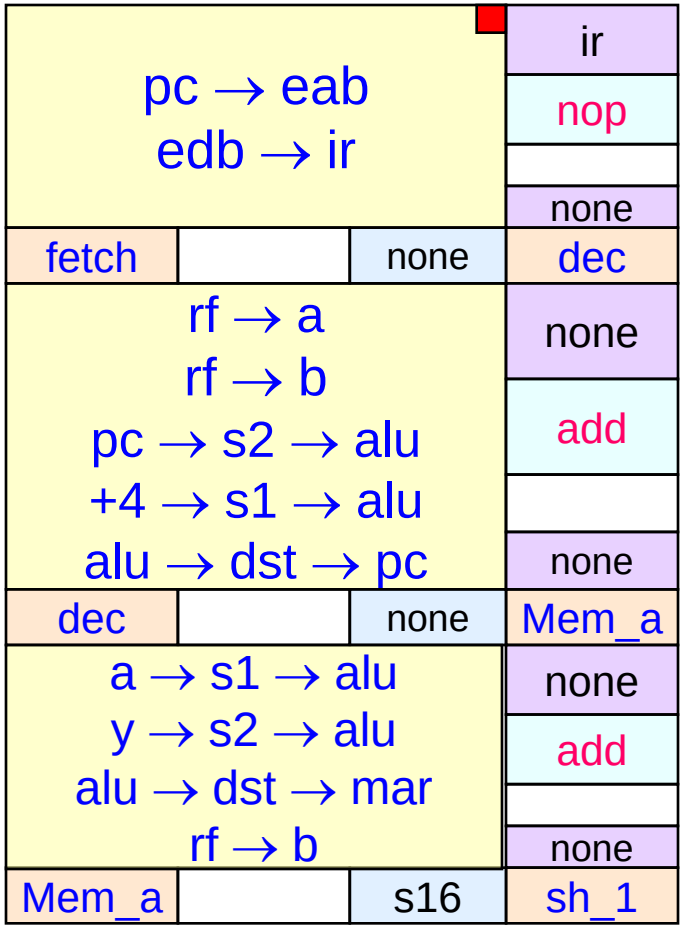
Merged Level 2 Hardware Flowchart



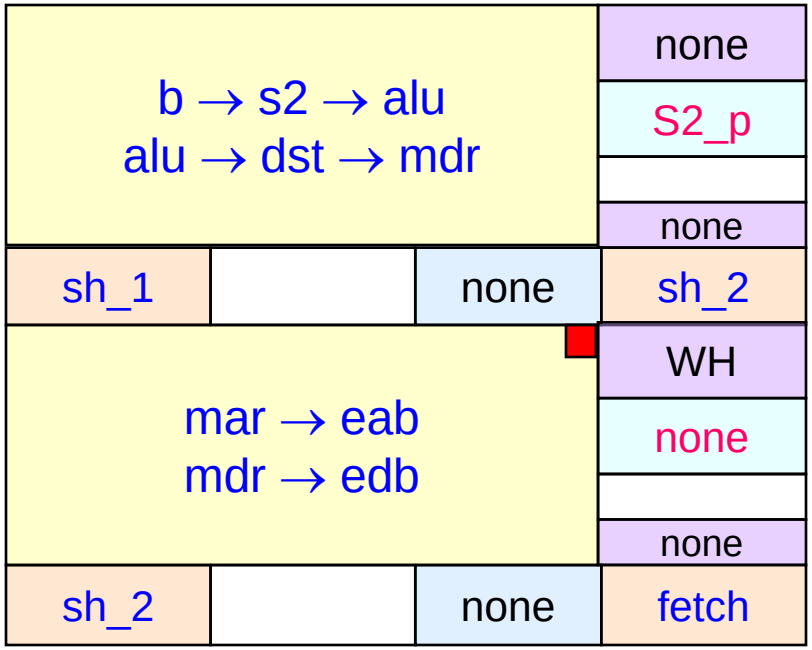
SH Rd, Rs2(Rs1)



Merged Level 2 Hardware Flowchart

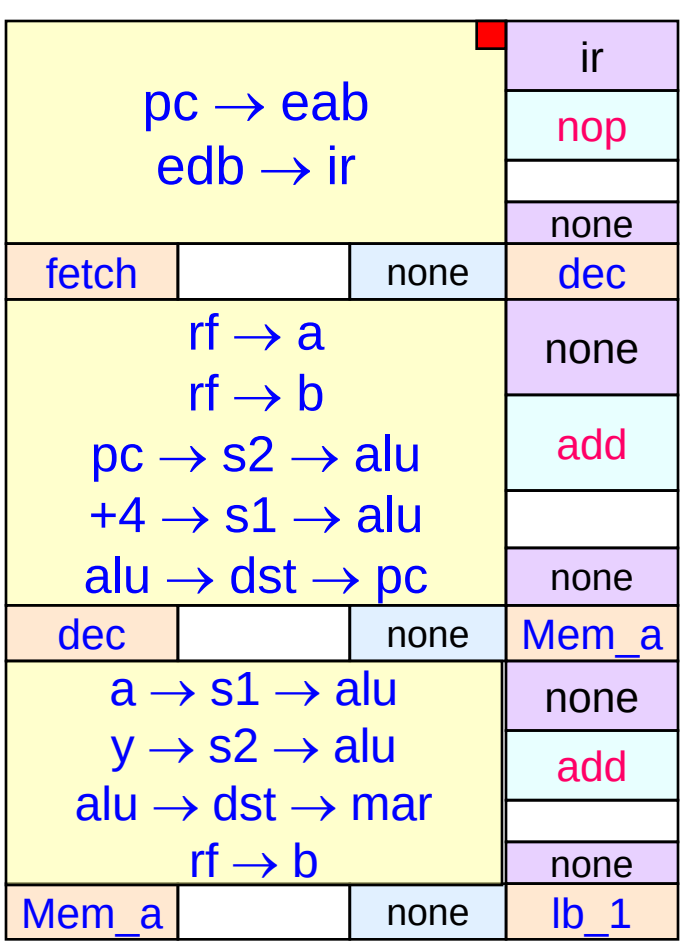


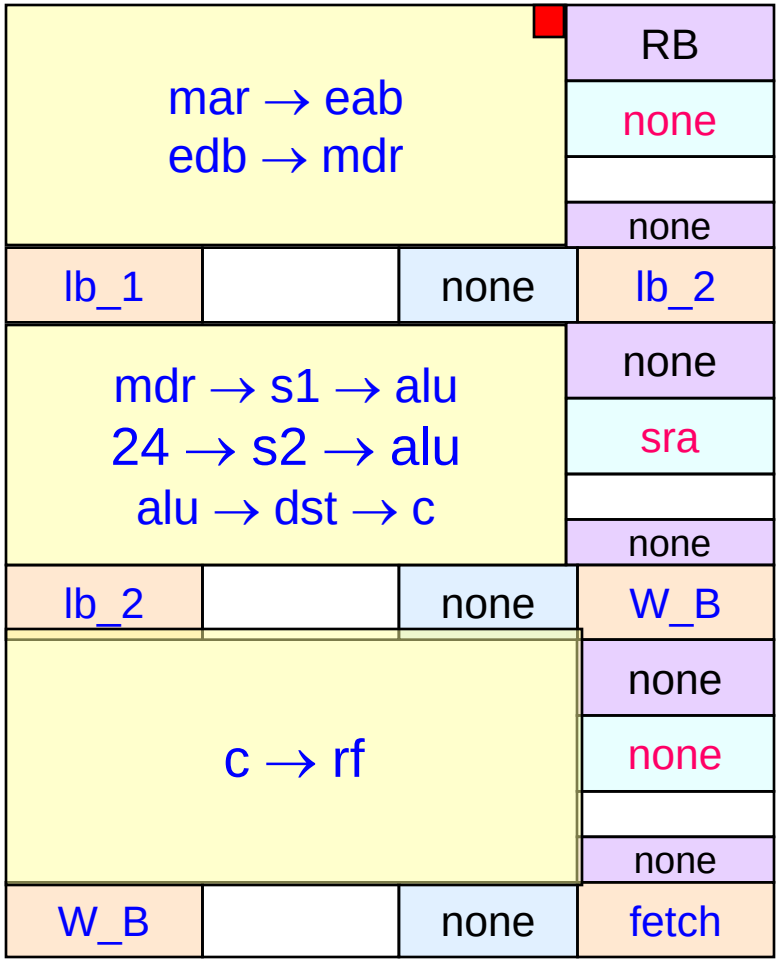
SH.I Rd, Imm(Rs1)



Merged Level 2 Hardware Flowchart

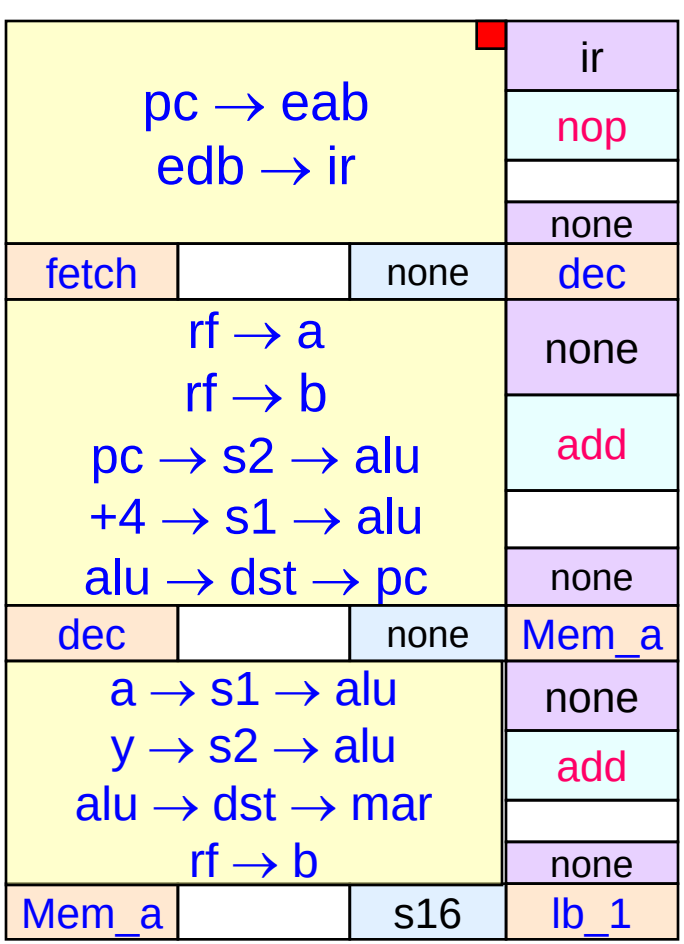
LB Rd, Rs2(Rs1)

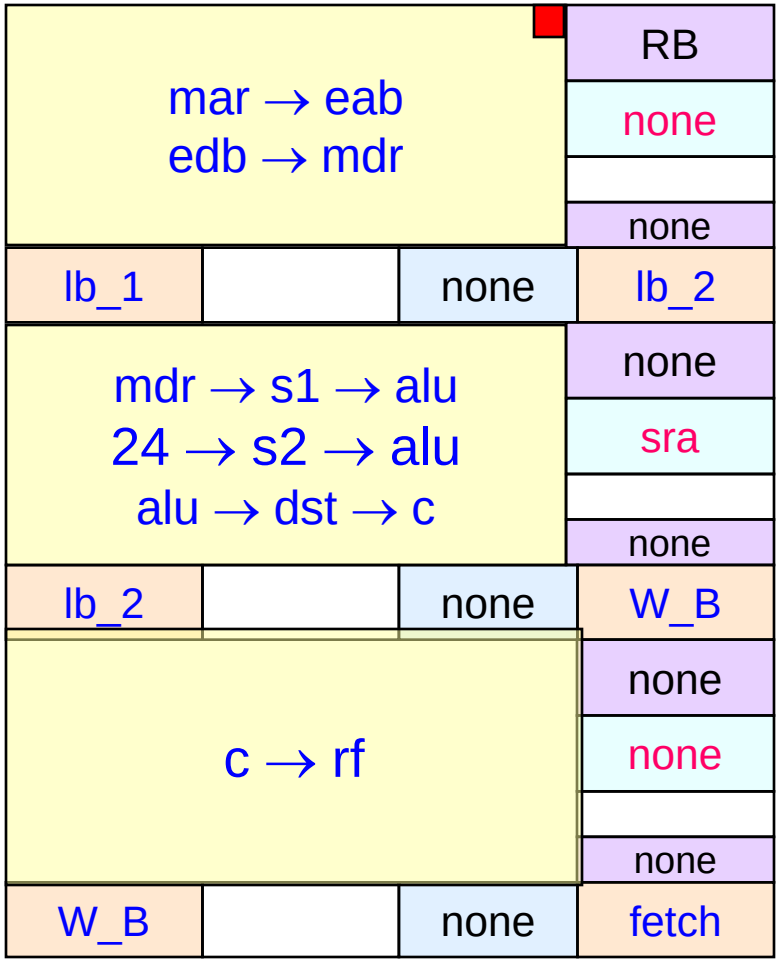




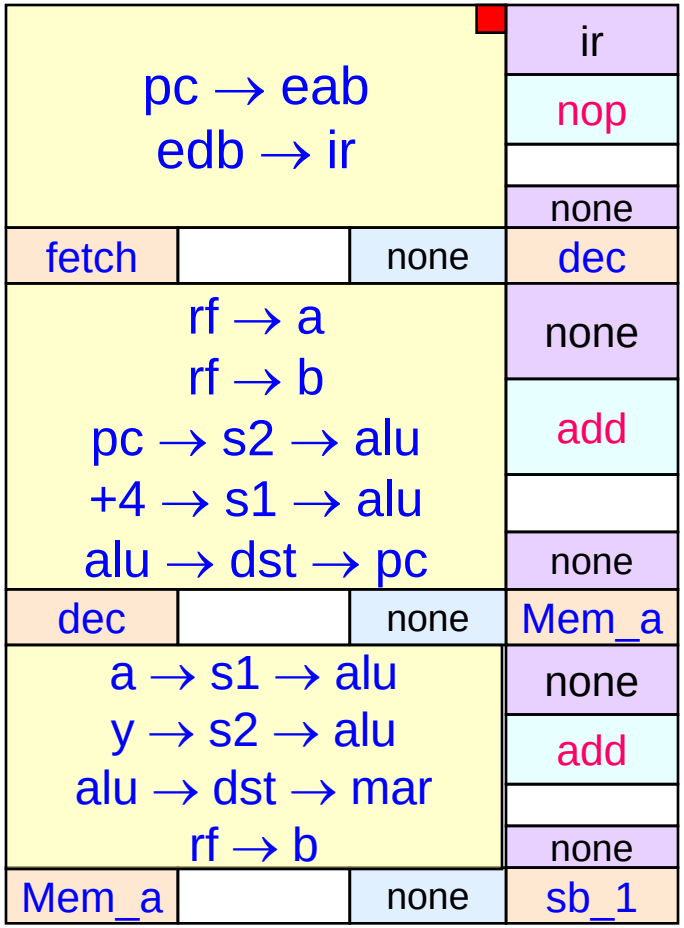
Merged Level 2 Hardware Flowchart

LB.I Rd, Imm(Rs1)

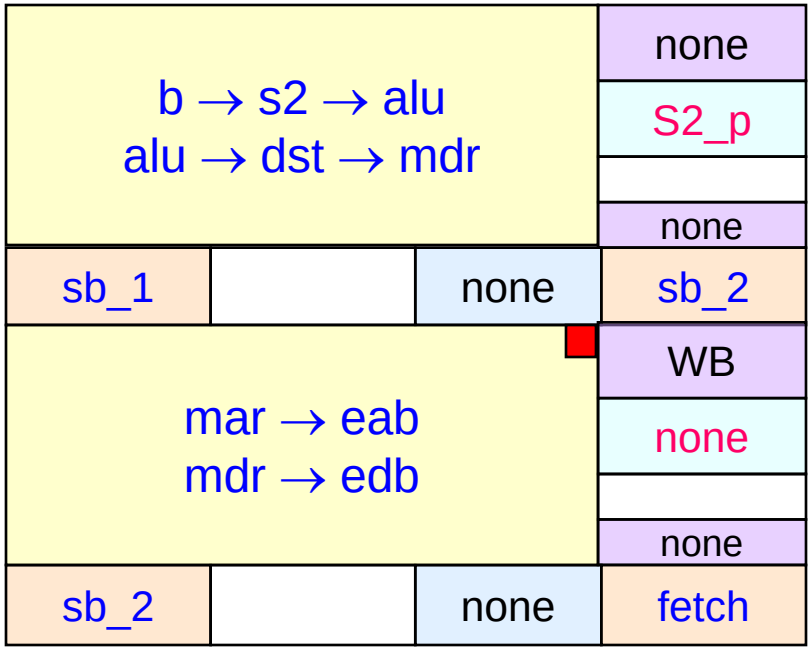




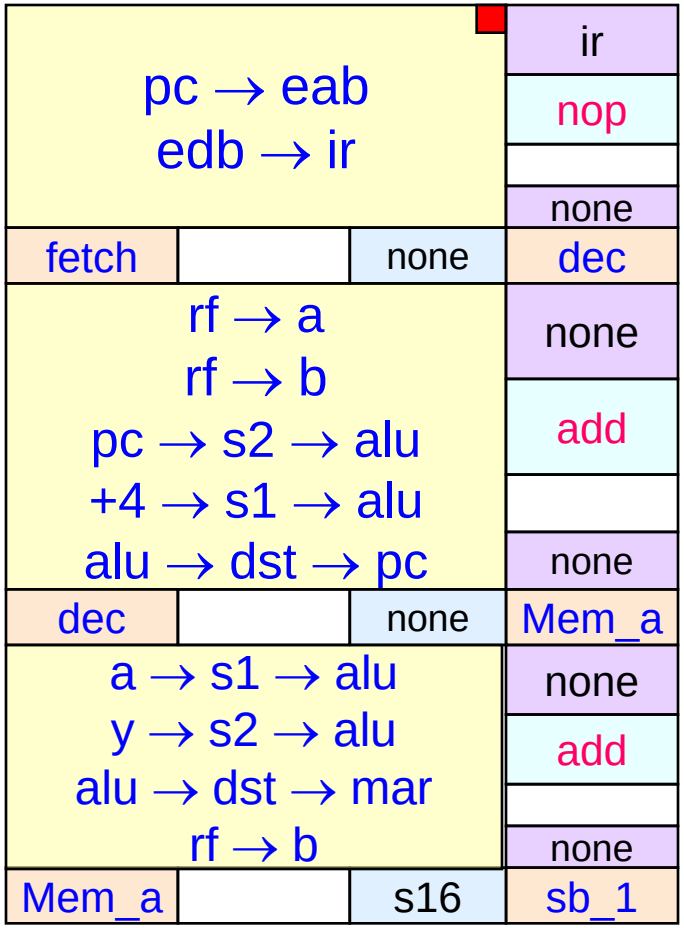
Merged Level 2 Hardware Flowchart



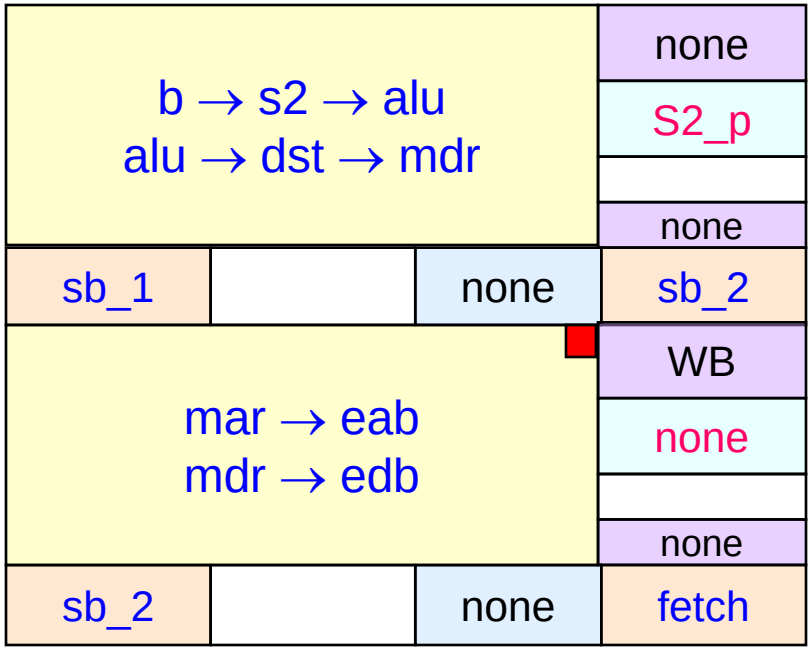
SB Rd, Rs2(Rs1)



Merged Level 2 Hardware Flowchart



SB.I Rd, Imm(Rs1)

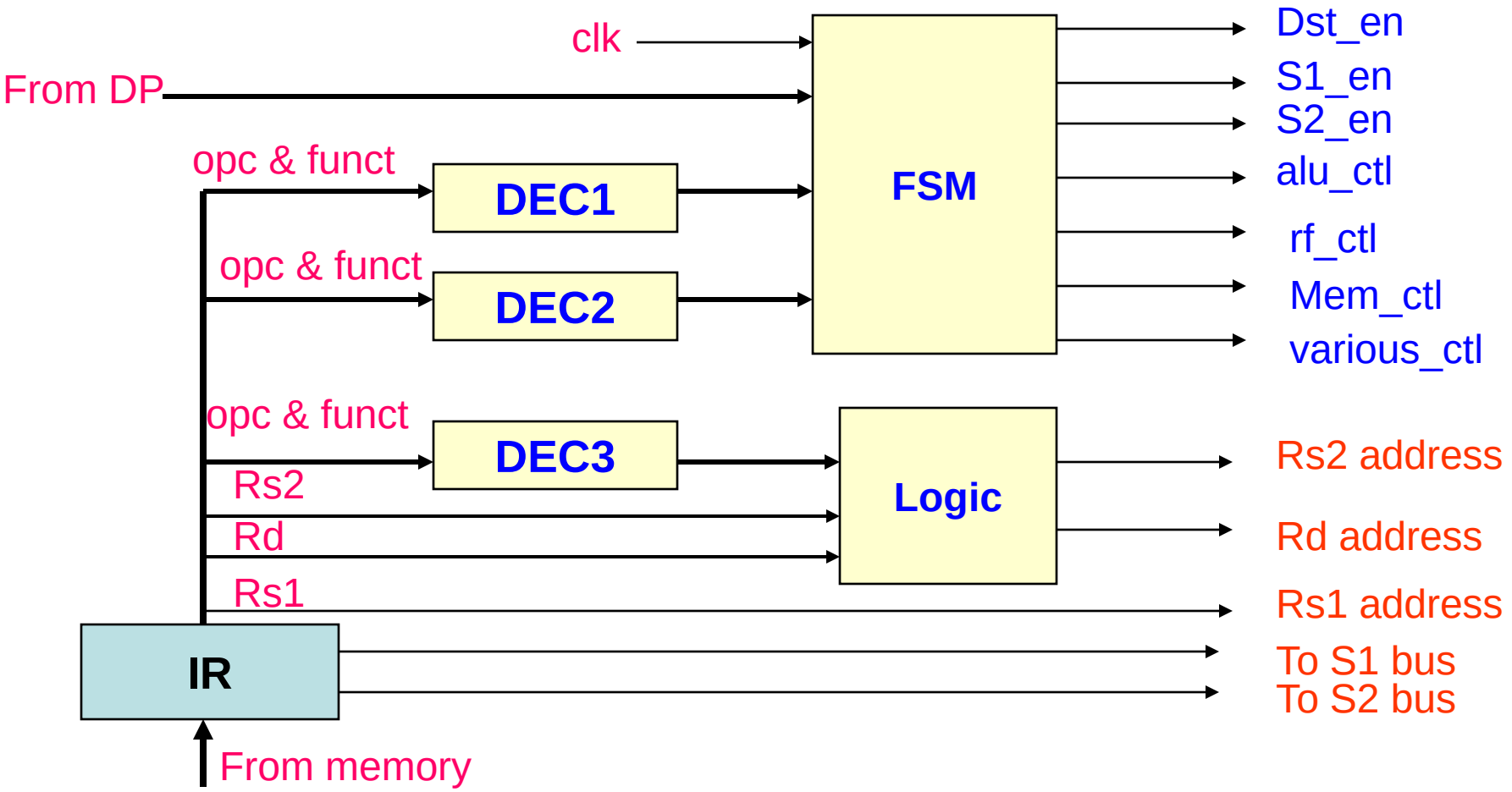


Implementation - States

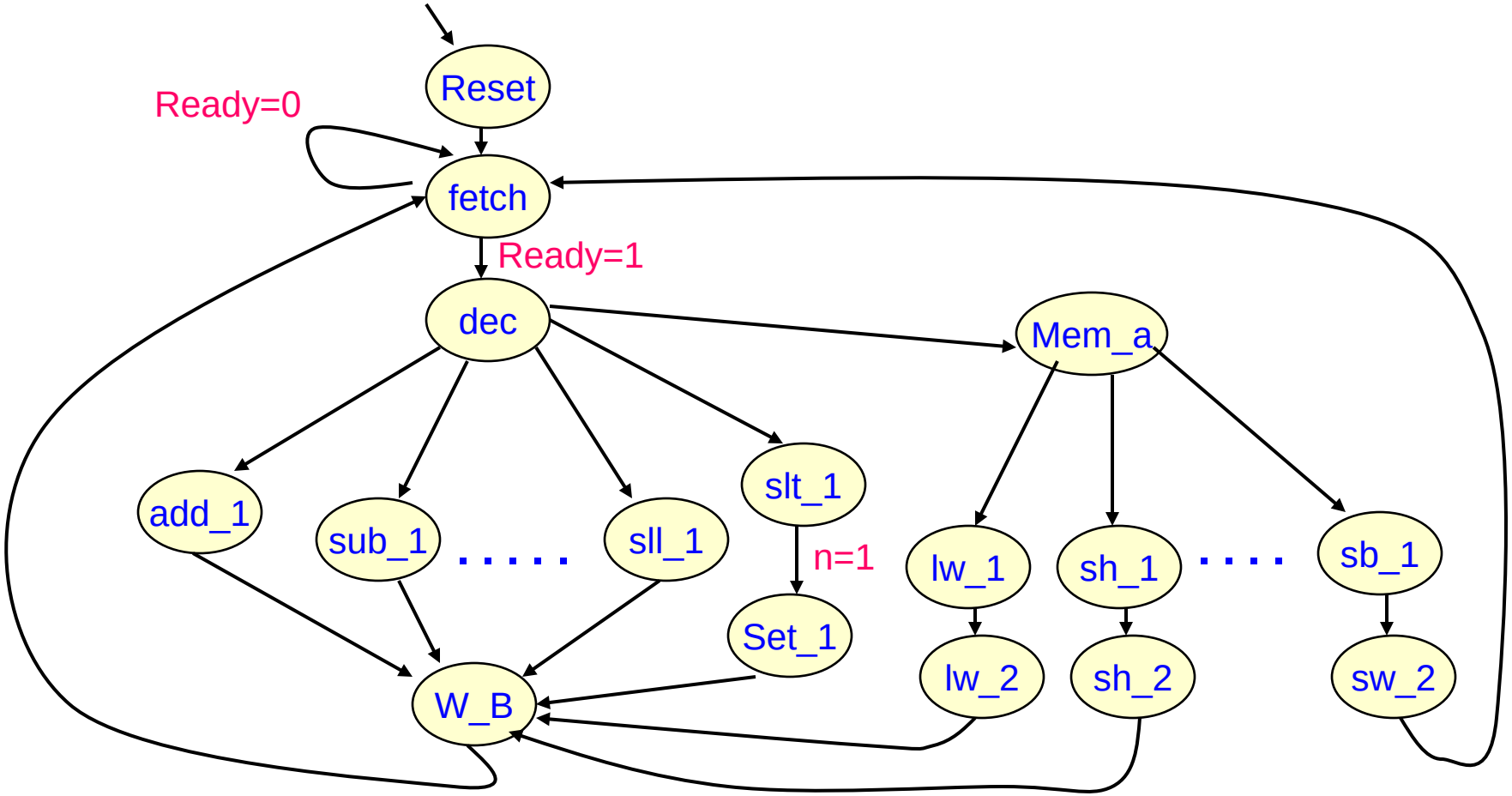
States:

- fetch, dec, mem_a
- add_1, sub_1, and_1, or_1, xor_1,sll_1, srl_1, sra_1
- seq_1, neq_1, slt_1, sle_1, sgt_1, sge_1
- Lw_1, lw_2, lh_1, lh_2, lb_1, lb_2, sw_11, sw_2, sh_1, sh_2, sb_1, sb_2
- breqz, brnez, branch, set_1, set_0
- reset

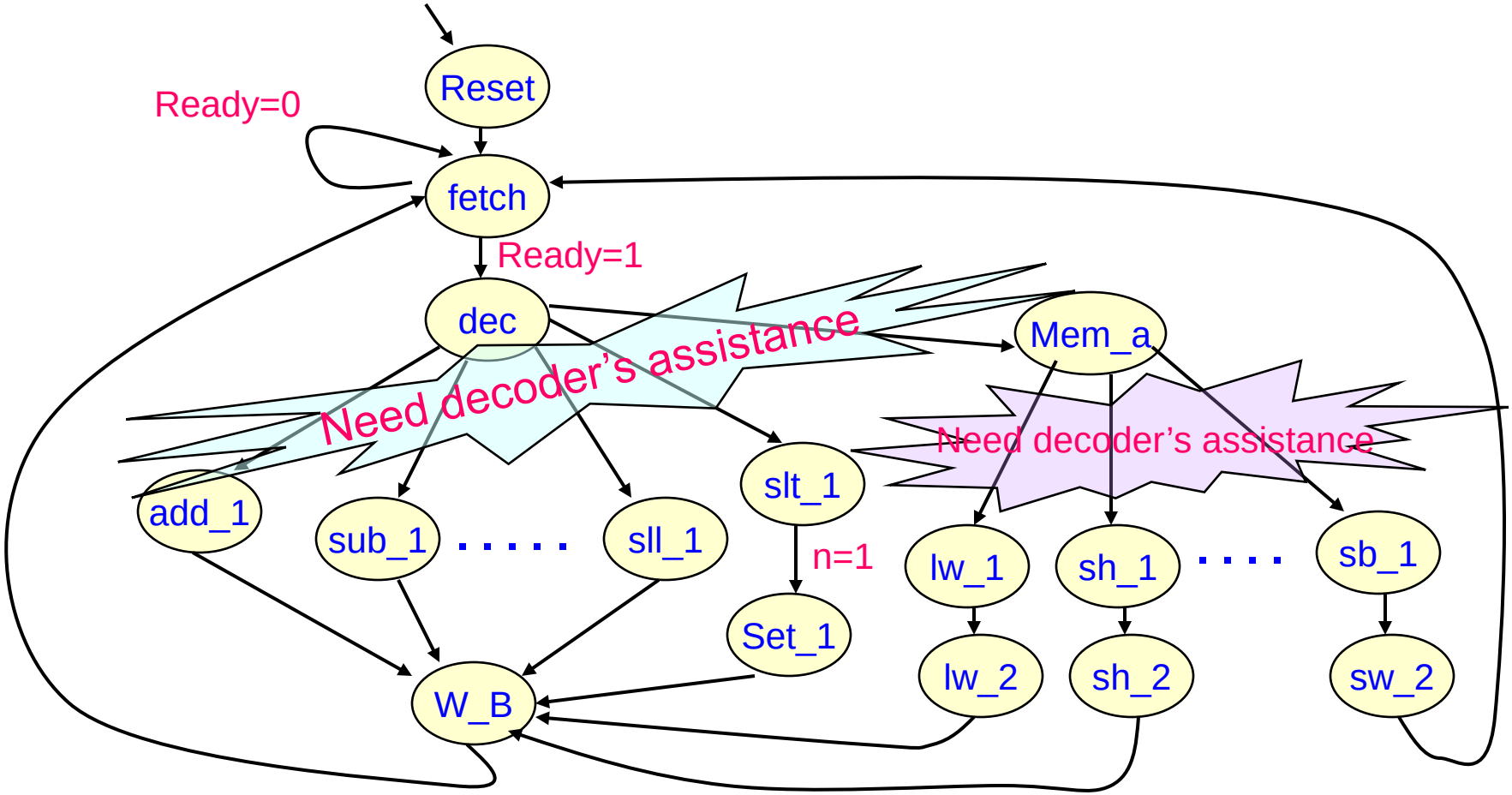
Implementation - Controller



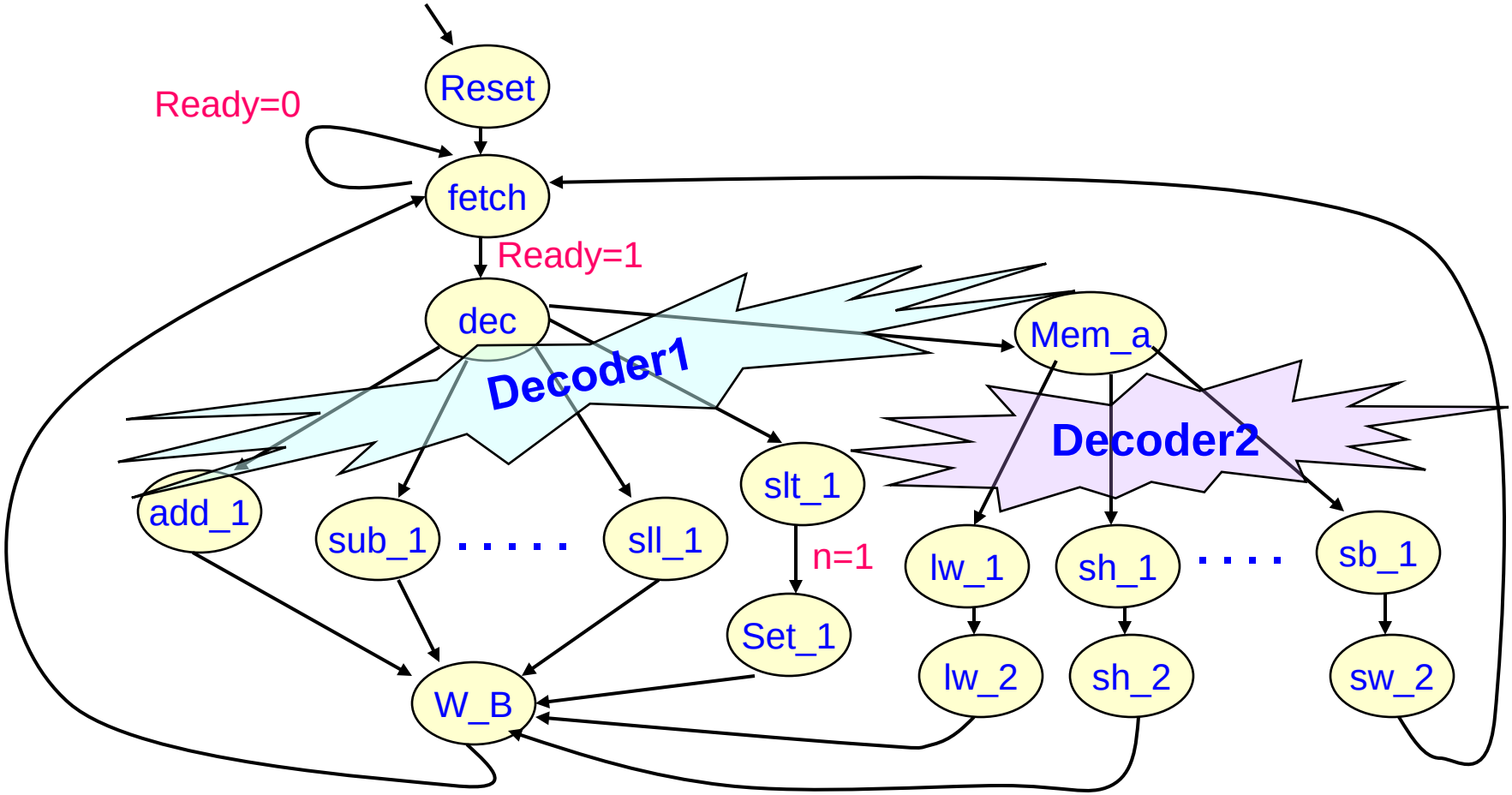
State Transition



State Transition



State Transition



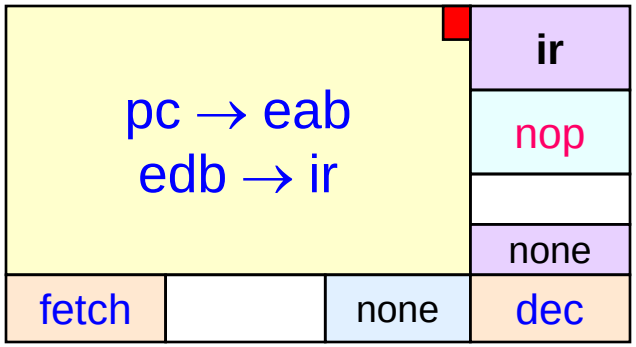
Implementation

Reset

```
when reset_1 =>
s1_enab <= s1_const;
s2_enab <= s2_none;
alu_op_sel <= alu_S1_p;
dest_enab <= dest_pc;
const_sel <= const_00;
rf_op_sel <= rfop_none;
immed_sel <= imm_dcare;
exc_enab <= exc_none;
mem_ctrl <= mem_fetch;
next_state <= fetch;
```

Implementation

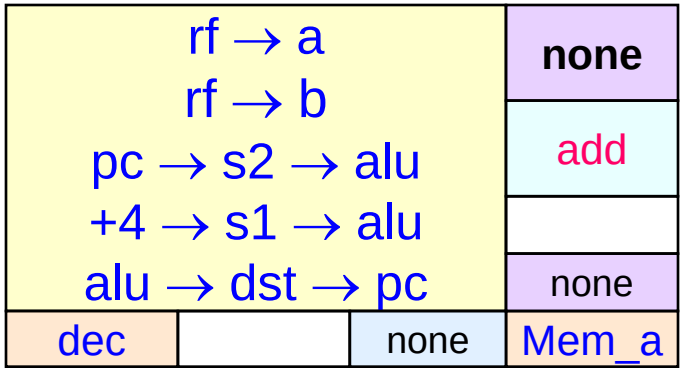
Fetch




```
when fetch =>
s1_enab <= s1_none;
s2_enab <= s2_none;
alu_op_sel <= alu_dcare;
dest_enab <= dest_none;
const_sel <= const_dcare;
rf_op_sel <= rfop_none;
immed_sel <= imm_dcare;
exc_enab <= exc_none;
mem_ctrl <= mem_fetch;
if ready = '1' then next_state <= dec;
else  next_state <= fetch;
end if;
```

Implementation

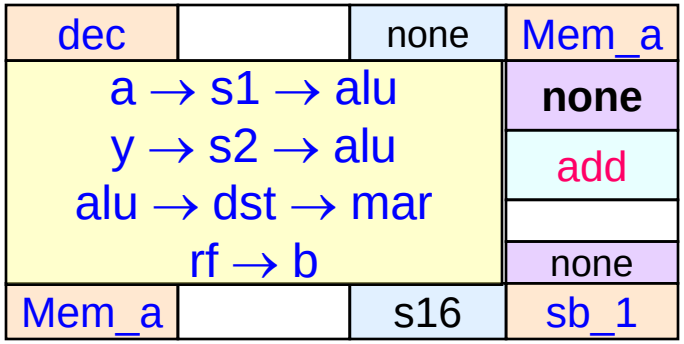
Dec




```
when dec =>
s1_enab <= s1_const;
s2_enab <= s2_pc;
alu_op_sel <= alu_add;
dest_enab <= dest_pc;
const_sel <= const_04;
rf_op_sel <= rfop_ab_rf;
immed_sel <= imm_dcare;
exc_enab <= exc_none;
mem_ctrl <= mem_none;
next_state <= dec_1_in;
```

Implementation

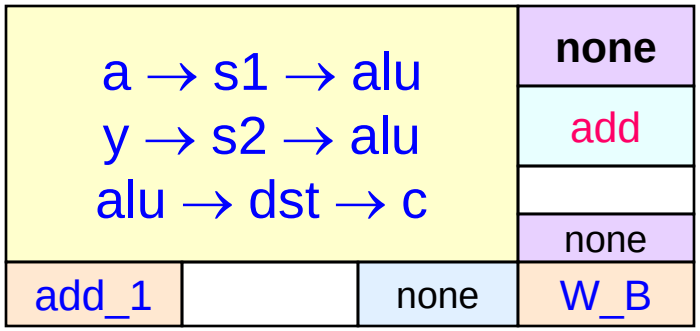
Mem_a




```
when Mem_a =>
s1_enab <= s1_a;
s2_enab <= s2_y;
alu_op_sel <= alu_add;
dest_enab <= dest_mar;
const_sel <= const_dcare;
rf_op_sel <= rfop_ab_rfx;
immed_sel <= imm_dcare;
exc_enab <= exc_none;
mem_ctrl <= mem_none;
next_state <= dec_2_in;
```

Implementation

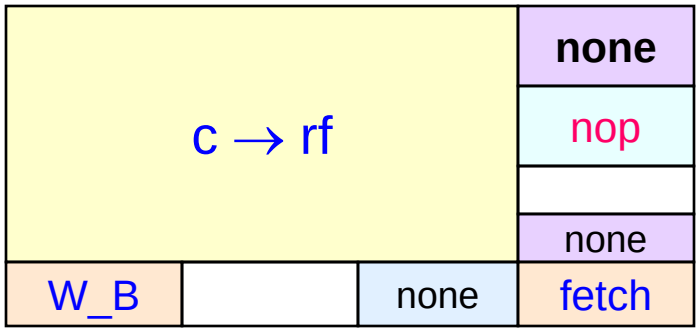
Add_1




```
when add_1 =>
s1_enab <= s1_a;
s2_enab <= s2_y;
alu_op_sel <= alu_add;
dest_enab <= dest_c;
const_sel <= const_dcare;
rf_op_sel <= rfop_none;
immed_sel <= imm_s16;
exc_enab <= exc_none;
mem_ctrl <= mem_none;
next_state <= W_B;
```

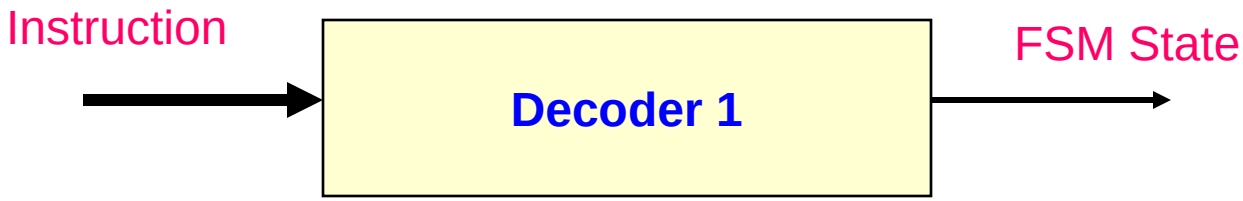
Implementation

W_B




```
when W_B =>
s1_enab <= s1_none;
s2_enab <= s2_none;
alu_op_sel <= alu_none;
dest_enab <= dest_c;
const_sel <= const_dcare;
rf_op_sel <= rfop_none;
immed_sel <= imm_none;
exc_enab <= exc_none;
mem_ctrl <= mem_none;
next_state <= fetch;
```

Implementation: Decoder1




```
decoder1 : process(instuct)
  begin
    case instruct(0 to 5) is
            when op_rr_alu =>
            case instr_in(26 to 31) is
              when funct_nop =>
                dec1_out <= fetch;
              when funct_add =>
                dec1_out <= add_1;
            ……..
            ……..
```

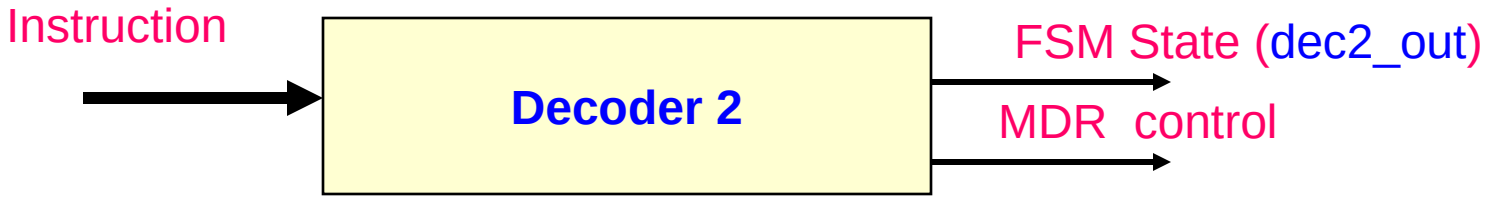
Implementation: Decoder1

```
when  funct_lw | funct_lh | funct_lb |
          funct_sw | funct_sh |funct_sb =>
             dec1_out <= Mem_a;
when others =>
             dec1_out <= fetch; // - it should be illegal state
end case;
 when op_lw_i | op_lh_i | op_lb_i | op_sw_i | op_sh_i |
          op_sb_i => dec1_out <= Mem_a;
when op_lhi =>
          dec1_out <= lhi;
     when op_add_i =>
          dec1_out <= add_1;
……….
……….
```

Implementation: Decoder1

```
when op_jal =>
        dec1_out <= jal_1;
when op_jr =>
        dec1_out <= jr_1;
……
……
when others =>
        dec1_out <= fetch; //  it should be illegal state
   end case;
end process decoder1;
```

Implementation: Decoder2

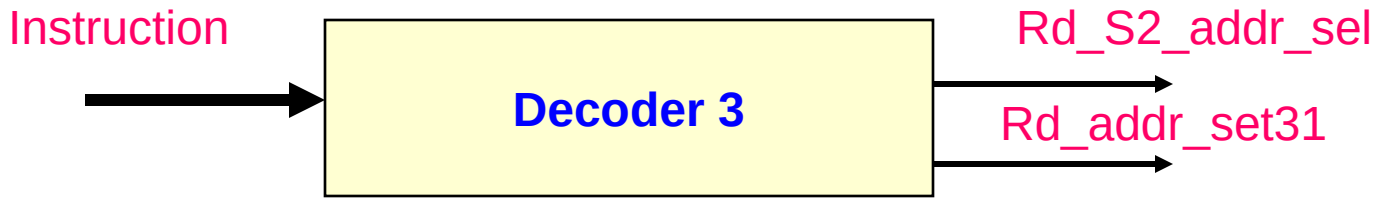



```
decoder2 : process(instuct)
 begin
case instruct(0 to 5) is
    when op_rr_alu =>
          case instr_in(26 to 31) is
        when funct_lw =>
            dec2_out <= lw_1;
            mdr_ctrl_out <= mdr_ctl_none;
          when rr_func_lh =>
            dec2_out <= lh_1;          ……..
          …….end case;
```

Implementation: Decoder2

```
when op_lw_i =>
        dec2_out <= lw_1;
        mdr_ctrl_out <= mdr_ctl_none;
when op_lh_i =>
        dec2_out <= lh_1;
        mdr_ctrl_out <= mdr_ctl_lh;
.............
.............
when others =>
        dec2_out <= sb_1;
        mdr_ctrl_out <= mdr_ctl_sb;
 end case;
  end process decoder2;
```

Implementation: Decoder3




```
decoder3 : process(instuct)
 begin
case instr_in(0 to 5) is
    when op_rr_alu =>
     case instr_in(26 to 31) is
      when        funct_add | funct_sub | tfunct_and | funct_or |
          funct_xor | funct_seq | funct_sne | funct_slt | funct_sle |
          funct_sgt | funct_sge | funct_lw | funct_lh | funct_lb
          |funct_sw | funct_sh |      funct_sb =>
          rd_s2_adr_sel <= '1' ;
          rd_adr_set31 <= '0' ; ……..
```

Implementation: Decoder3

```
            when others =>
                rd_s2_adr_sel <= '0' ;
                rd_adr_set31 <= '0' ;
            end case;
when op_jal | op_jalr =>
  rd_s2_adr_sel <= '0' ;
  rd_adr_set31 <= '1' ;
.............
.............
when others =>
  rd_s2_adr_sel <= '0' ;
  rd_adr_set31 <= '0' ;
end case;
end process decoder3;
```

Implementation: Controller (FSM)

```
architecture fsm_1 of fsm is
     signal cur_state: fsm_state;
     signal nxt_state: fsm_state;

State_change: process(clk, reset)
begin
  if (clk'event and clk=1)
     if reset = '1' then
            cur_state <= reset_1;
     else
            cur_state <= nxt_state;
     end if;
  end if;
end process state_change;
```

Implementation: Controller (FSM)

```
Next_state_logic: process(cur_state, reset, ready, dec_1_in,
dec_2_in, alu_neg, alu_z)
begin
  case cur_state is
          when reset_1 =>
              nxt_state <= fetch;
           when fetch =>
              if ready = '1' then
                 nxt_state <= dec;
              else
              nxt_state <= fetch;
          when dec =>
              nxt_state <= dec_1_in;
       ..........
```

Implementation: Controller (FSM)

```
when Mem_a =>
        nxt_state <= dec_2_in;
. . . . . .
when slt_1 =>
        if alu_neg = '1' then
          nxt_state <= Set_1;
        else
          nxt_state <= Set_0;
        end if;
. . . . . .
. . . . .
end case;
end process Next_state_logic;
```

Implementation: Controller (FSM)

```
Output_logic: process(cur_state)
begin
  case cur_state is
            when reset_1 =>
              s1_enab <= s1_const;
              s2_enab <= s2_none;
              alu_op_sel <= alu_S1_p;
              dest_enab <= dest_pc;
              const_sel <= const_00;
              rf_op_sel <= rfop_none;
              immed_sel <= imm_dcare;
              exc_enab <= exc_none;
              mem_ctrl <= mem_fetch;
```

Implementation: Controller (FSM)

```
when fetch =>
  s1_enab <= s1_none;
  s2_enab <= s2_none;
  alu_op_sel <= alu_dcare;
  dest_enab <= dest_none;
  const_sel <= const_dcare;
  rf_op_sel <= rfop_none;
  immed_sel <= imm_dcare;
  exc_enab <= exc_none;
  mem_ctrl <= mem_fetch;
. . . . . .
. . . . . .
. . . . . .
```

Implementation: Controller (FSM)

```
when W_B =>
s1_enab <= s1_none;
s2_enab <= s2_none;
alu_op_sel <= alu_none;
dest_enab <= dest_c;
const_sel <= const_dcare;
rf_op_sel <= rfop_none;
immed_sel <= imm_none;
exc_enab <= exc_none;
mem_ctrl <= mem_none;
. . . . . .
. . . . . .
end case;
end process output_logic;
```

Thank You