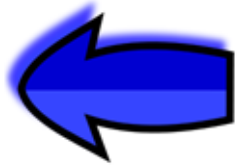
Processor Design

Simple RISC ISA

Basic Computer Architecture

Outline

* Overview of a Processor
* Detailed Design of each Stage
* The Control Unit
* Microprogrammed Processor
* Microassembly Language
* The Microcontrol Unit



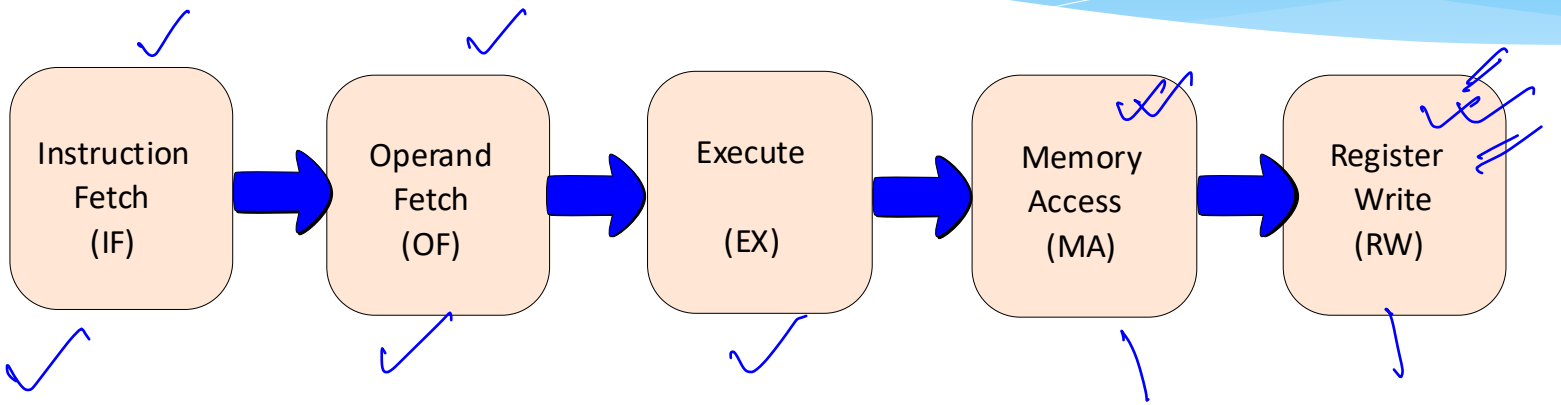
Processor Design

* The aim of processor design
  * Implement the entire SimpleRisc ISA
  * Process the binary format of instructions
  * Provide as much of performance as possible
* Basic Approach
  * Divide the processing into stages
  * Design each stage separately

A Car Assembly Line

* Similar to a car assembly line
  * Cast raw metal into the chassis of a car
  * Build the Engine
  * Assemble the engine and the chassis
  * Place the dashboard, and upholstery

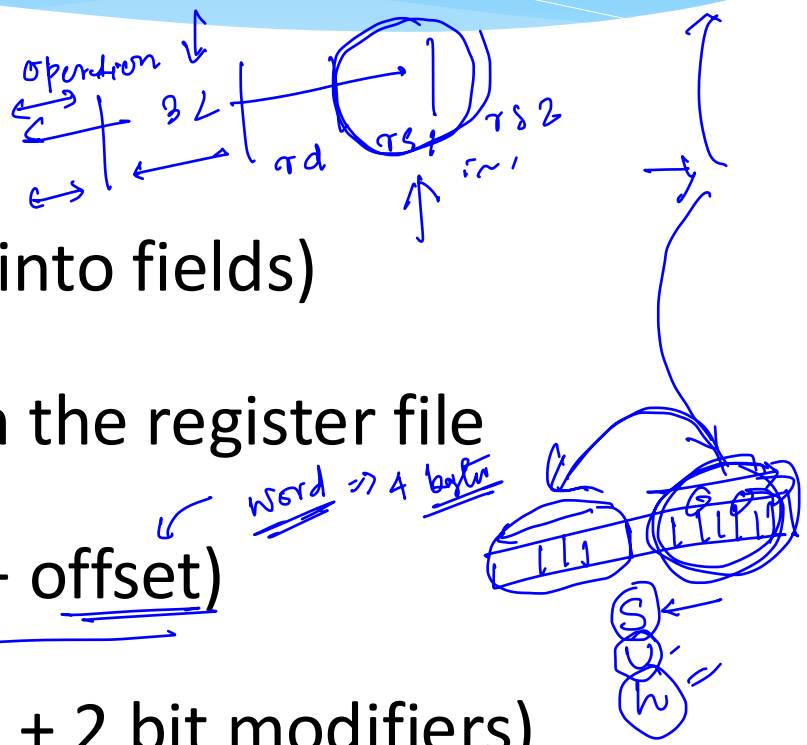
A Processor Divided Into Stages



* Instruction Fetch (IF)
  * Fetch an instruction from the instruction memory
  * Compute the address of the next instruction

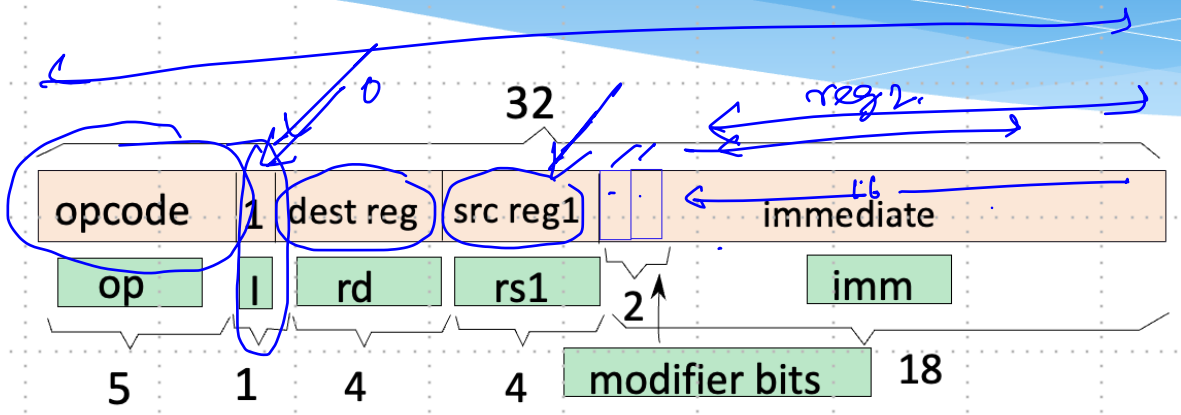
Operand Fetch (OF) Stage

* Operand Fetch (OF)
  * Decode the instruction (break it into fields)
  * Fetch the register operands from the register file
  * Compute the branch target (PC + offset)
  * Compute the immediate (16 bits + 2 bit modifiers)
  * Generate control signals (we will see later)



Recap from SimpleRISC ISA

Immediate Format



* opcode → type of the instruction
* I bit → 1 (second operand is an immediate)
* dest reg → rd
* source register 1 → rs1
* Immediate → imm
* modifier bits → 00 (default), 01 (u), 10 (h)

Execute (EX) Stage

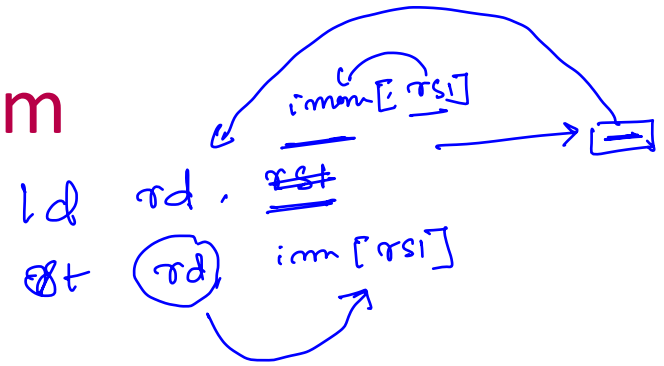
- The EX Stage
  - Contains an Arithmetic-Logical Unit (ALU)
    - This unit can perform all arithmetic operations (add, sub, mul, div, cmp, mod), and logical operations (and, or, not)
  - Contains the branch unit for computing the branch condition (beq, bgt)
  - Contains the flags register (updated by the cmp instruction)

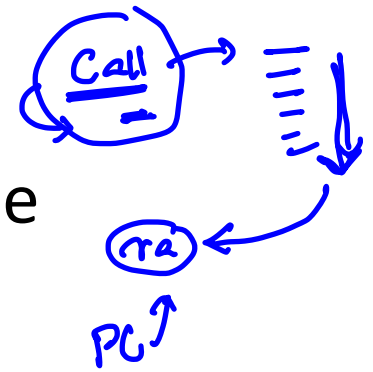
Special flags register → contains the result of the last comparison

- flags.E = 1 (equality), flags.GT = 1 (greater than)

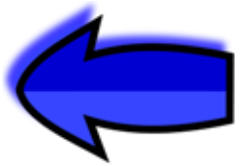
MA and RW Stages

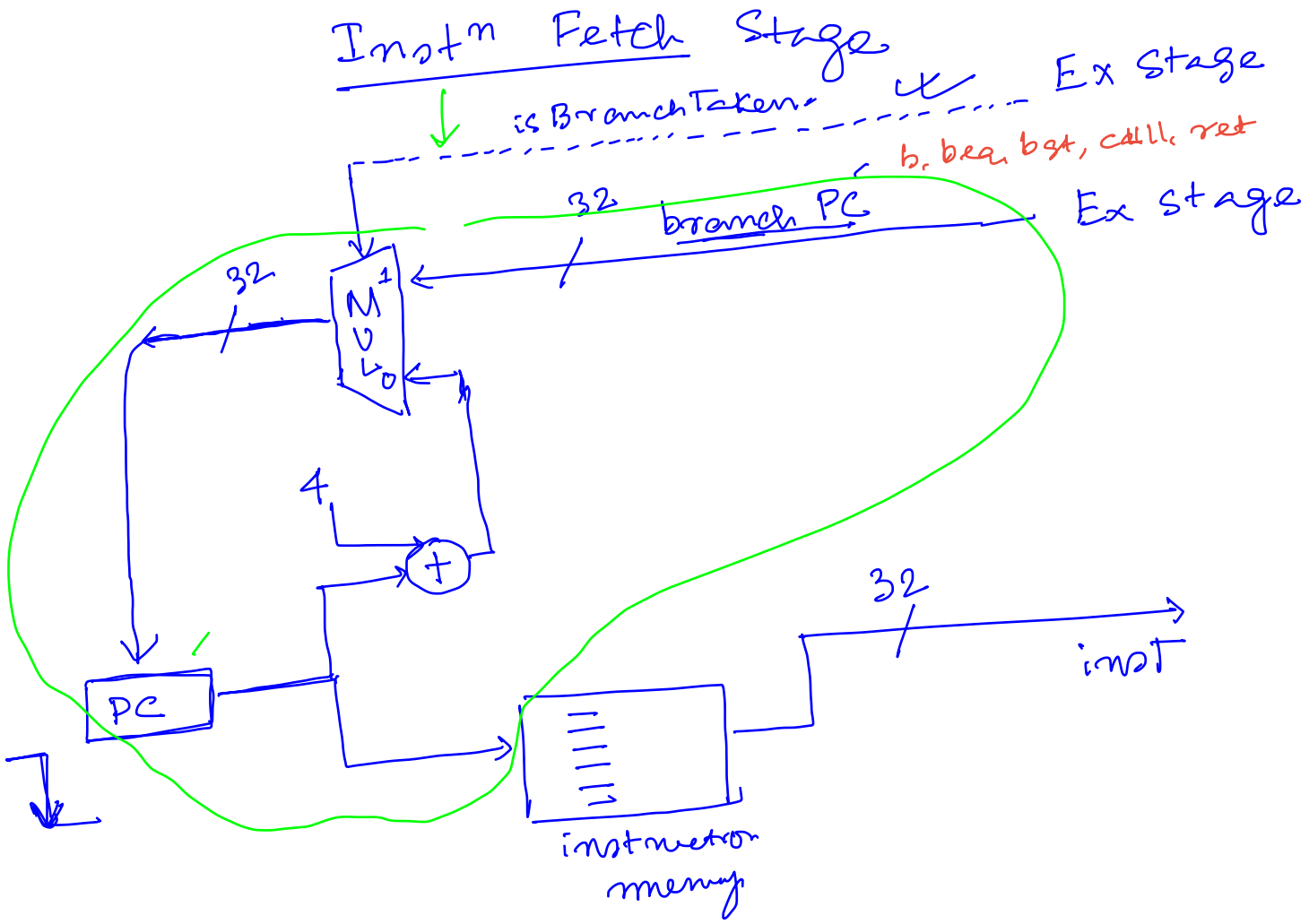
* MA (Memory Access) Stage
  * Interfaces with the memory system
  * Executes a load or a store
* RW (Register Write) Stage
  * Writes to the register file
  * In the case of a call instruction, it writes the return address to register, ra



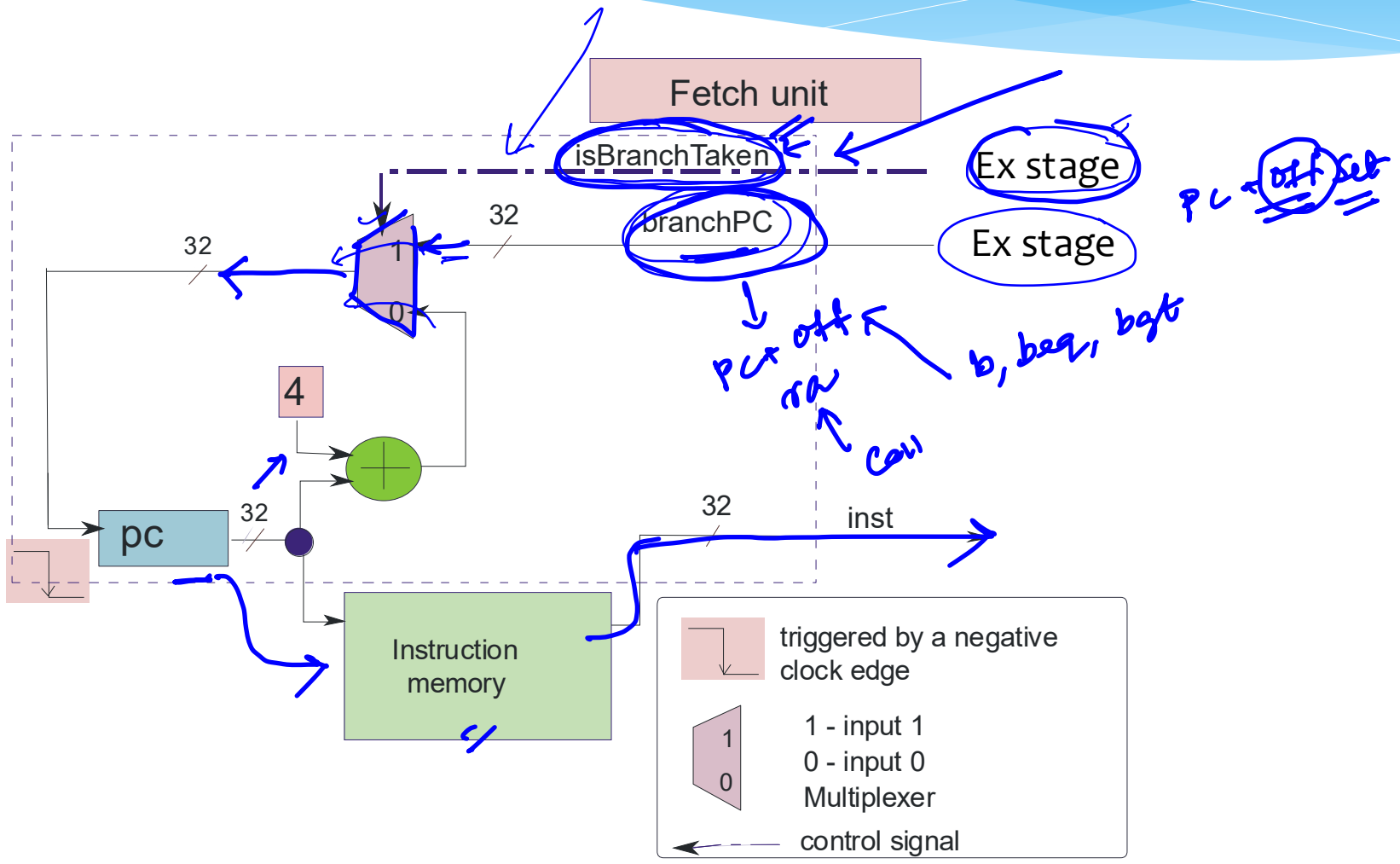


Outline

* Outline of a Processor
* Detailed Design of each Stage 
* The Control Unit
* Microprogrammed Processor
* Microassembly Language
* The Microcontrol Unit


Instn Fetch Stage
isBranchTaken
Ex stage
b, bea, bgt, call, ret
32
branch PC
Ex stage
32
M
U
X
1
0
4
+
PC
32
inst
instruction memory

Instruction Fetch (IF) Stage



The Fetch unit

- The pc register contains the program counter (negative edge triggered)
- We use the pc to access the instruction memory
- The multiplexer chooses between
  - pc + 4
  - branchTarget
- It uses a control signal → isBranchTaken

isBranchTaken

- isBranchTaken is a control signal
  - It is generated by the EX unit
- Conditions on isBranchTaken

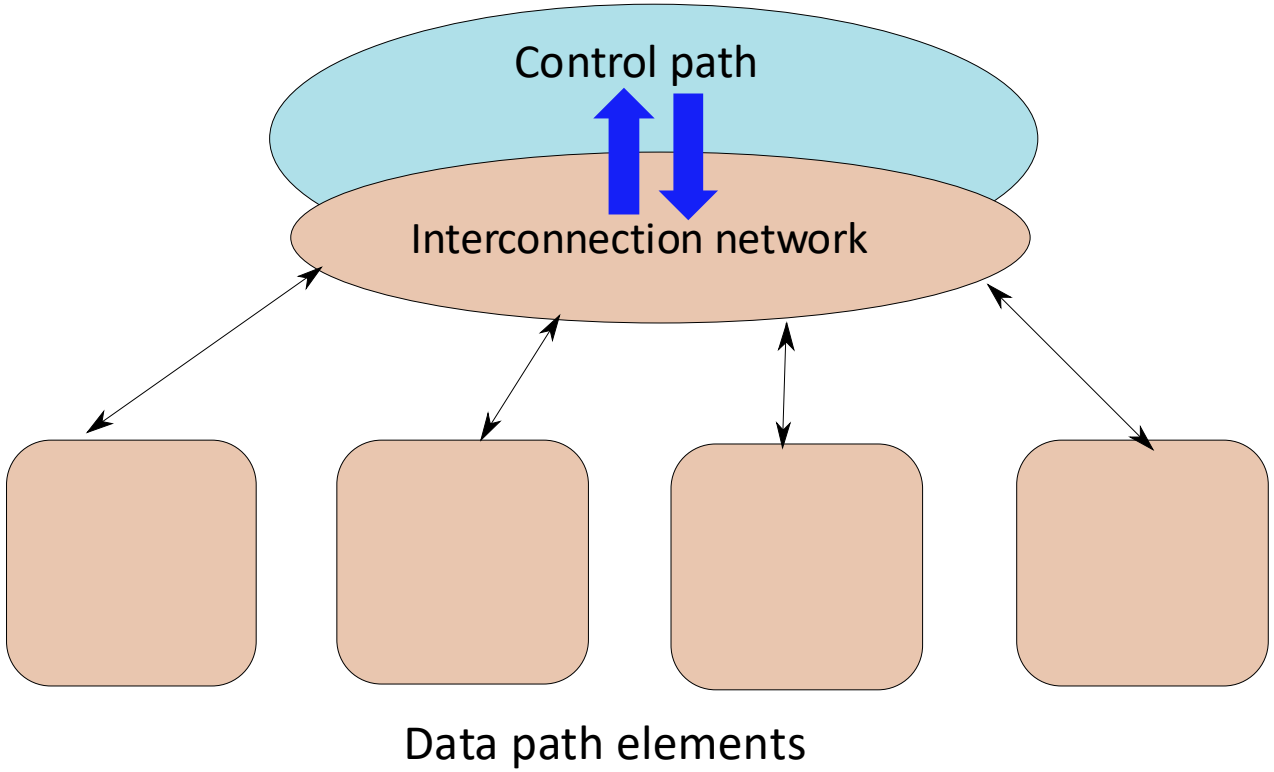
Instruction	Value of *isBranchTaken*
non-branch instruction	0
call	1
ret	1
b	1
beq	branch taken – 1 branch not taken – 0
bgt	branch taken – 1 branch not taken – 0

Data Path and Control Path

* The data path consists of all the elements in a processor that are dedicated to storing, retrieving, and processing data such as register files, memory, and the ALU.

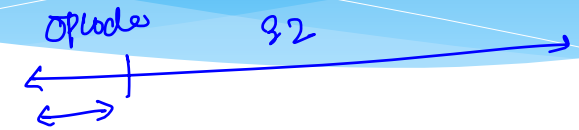
* The control path primarily contains the control unit, whose role is to generate the appropriate signals to control the movement of instructions, and data in the data path.

Control Path

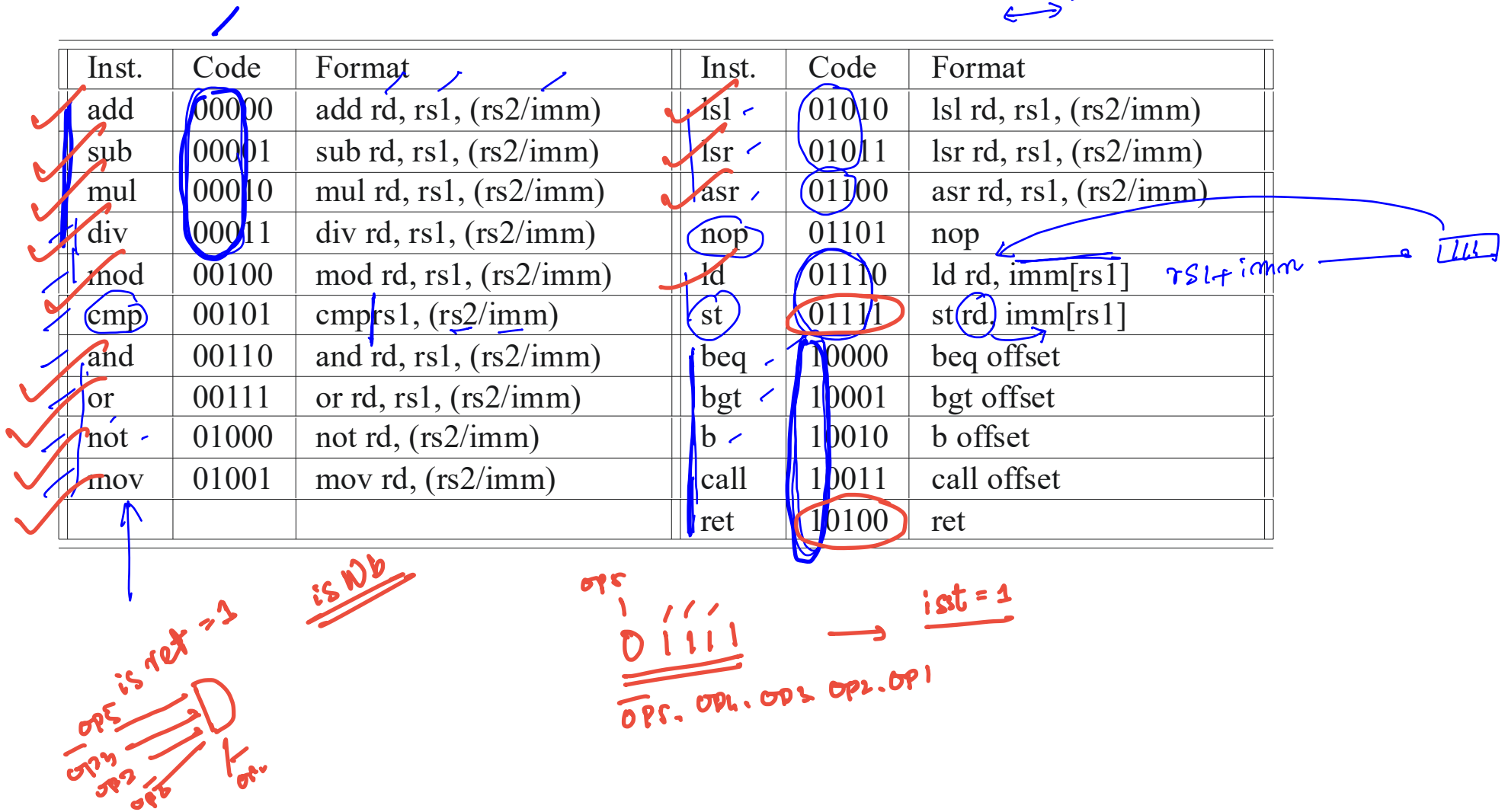


We will currently look at the hardwired control path.

Operand Fetch Unit



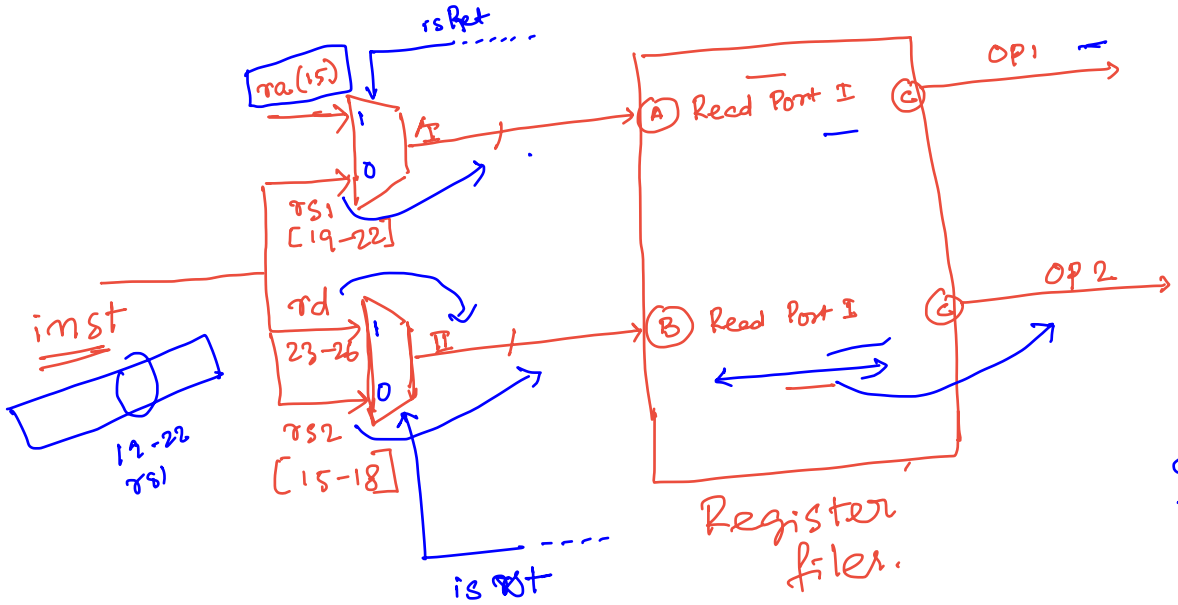
Inst.	Code	Format	Inst.	Code	Format
add	00000	add rd, rs1, (rs2/imm)	lsl	01010	lsl rd, rs1, (rs2/imm)
sub	00001	sub rd, rs1, (rs2/imm)	lsr	01011	lsr rd, rs1, (rs2/imm)
mul	00010	mul rd, rs1, (rs2/imm)	asr	01100	asr rd, rs1, (rs2/imm)
div	00011	div rd, rs1, (rs2/imm)	nop	01101	nop
mod	00100	mod rd, rs1, (rs2/imm)	ld	01110	ld rd, imm[rs1]
cmp	00101	cmprs1, (rs2/imm)	st	01111	st rd, imm[rs1]
and	00110	and rd, rs1, (rs2/imm)	beq	10000	beq offset
or	00111	or rd, rs1, (rs2/imm)	bgt	10001	bgt offset
not	01000	not rd, (rs2/imm)	b	10010	b offset
mov	01001	mov rd, (rs2/imm)	call	10011	call offset
			ret	10100	ret



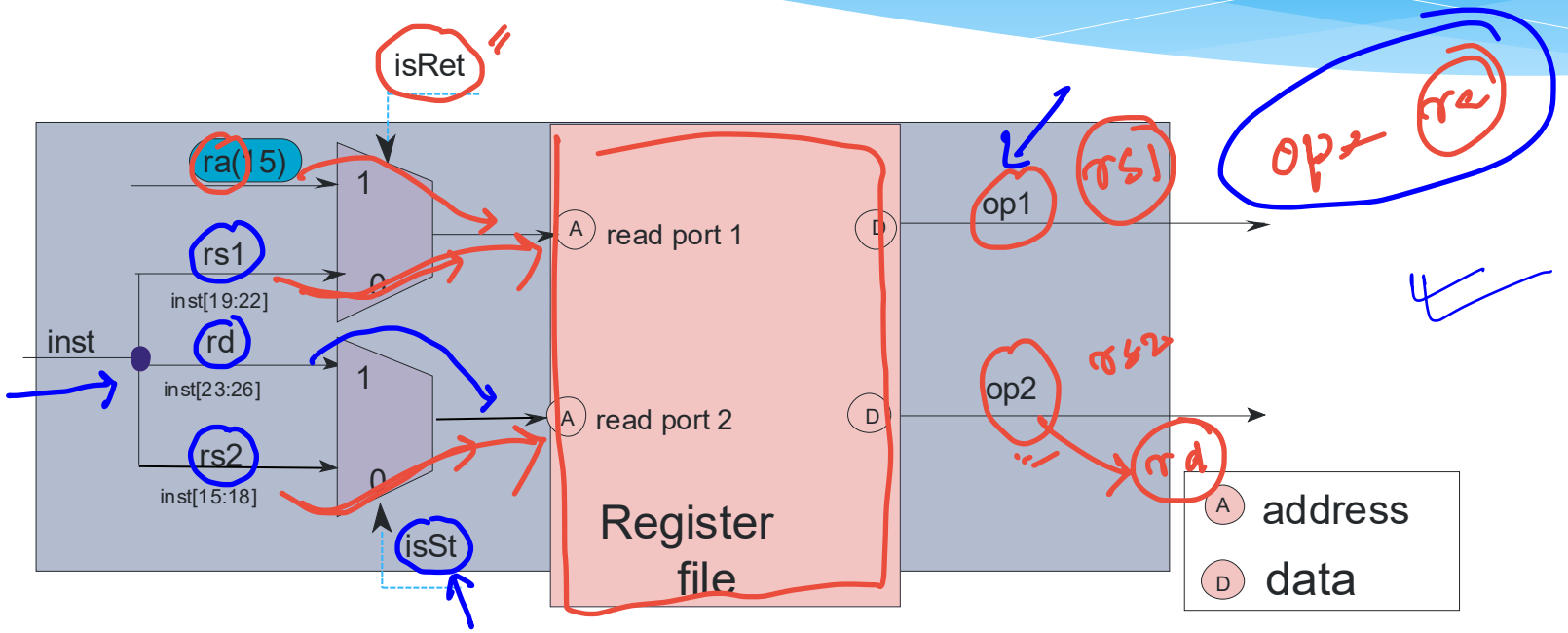
Instruction Formats

Format	Definition				
branch	*op* (28-32)	*offset* (1-27)			
register	*op* (28-32)	*I* (27)	*rd* (23-26)	*rs1* (19-22)	*rs2* (15-18)
immediate	*op* (28-32)	*I* (27)	*rd* (23-26)	*rs1* (19-22)	*imm* (1-18)
op → opcode, *offset* → branch offset, *I* → immediate bit, *rd* → destination register					
*rs*1 → source register 1, *rs*2 → source register 2, *imm* → immediate operand					

Word offset



Register File Read

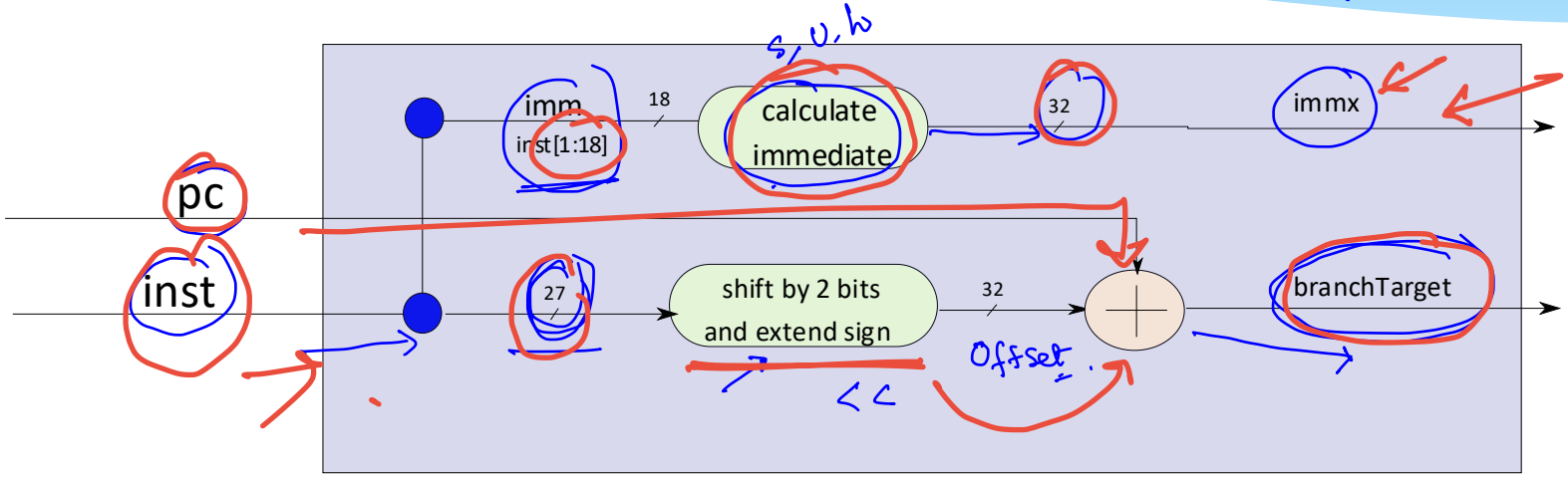


- First input → rs1 or ra(15) (ret instruction)
- Second input → rs2 or rd (store instruction)

Register File Access

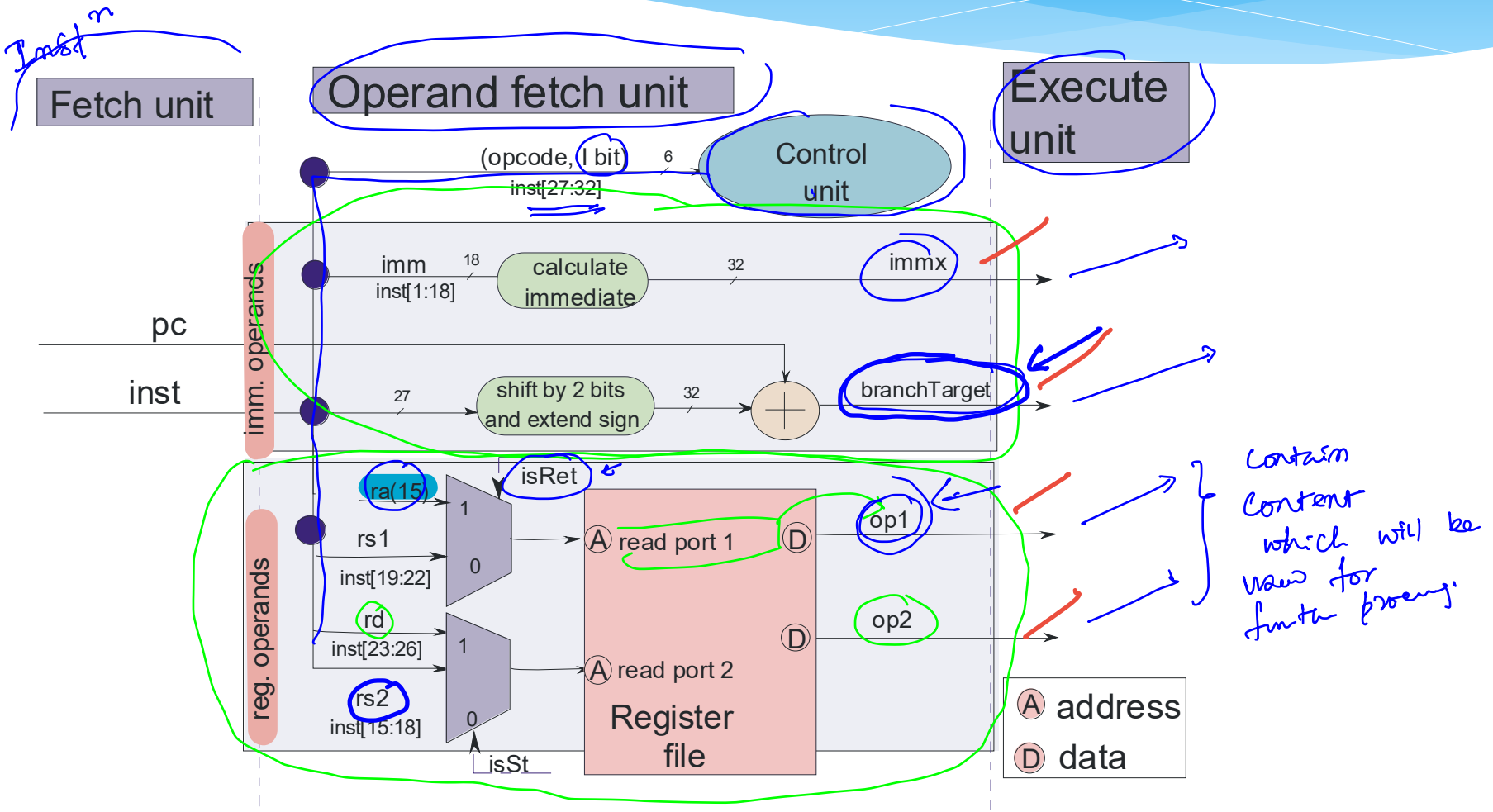
- The register file has two read ports
  - 1st Input
  - 2nd Input
- The two outputs are op1, and op2
  - op1 is the branch target (return address) in the case of a ret instruction, or rs1
  - op2 is the value that needs to be stored in the case of a store instruction, or rs2

Immediate and Branch Unit

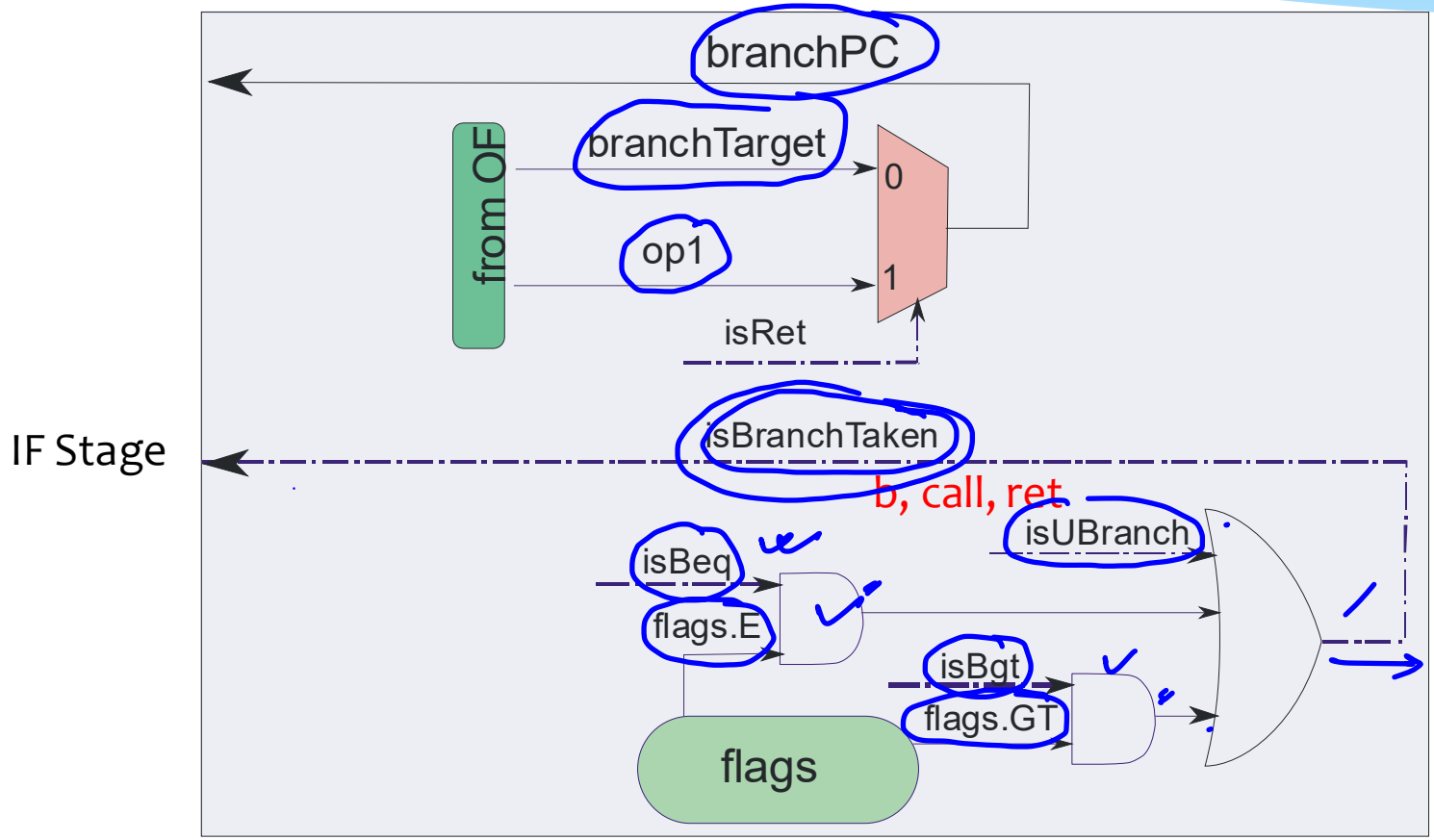


- Compute immx (extended immediate), branchTarget, irrespective of the instruction format.
- For the branchTarget we need to choose between the embedded target and op1 (ret)

OF Unit

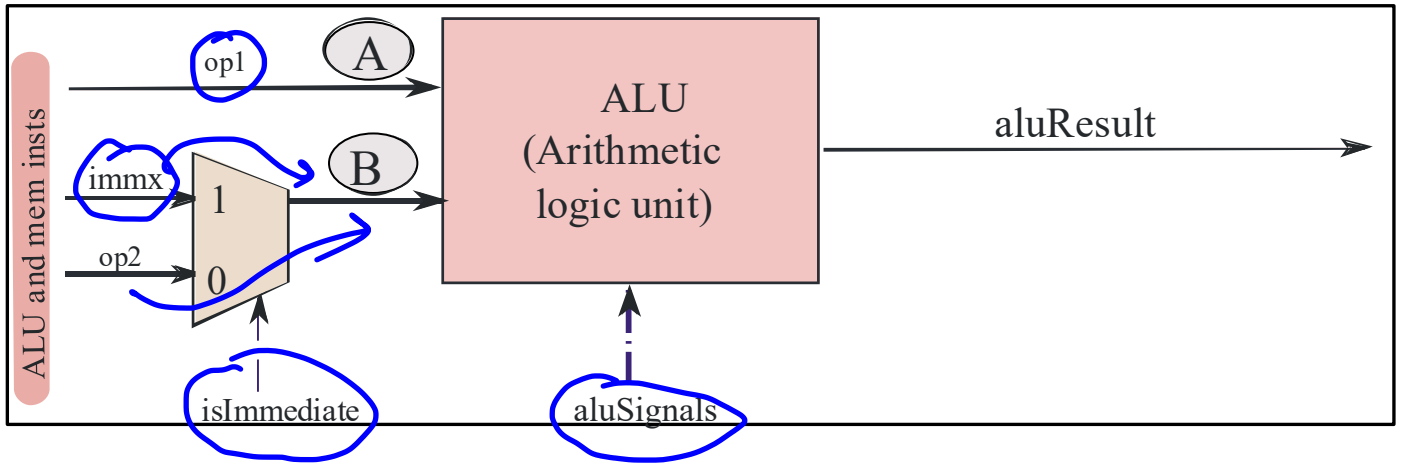

Fetch unit
Operand fetch unit
Execute unit
(opcode, I bit)
inst[27:32]
6
Control unit
imm
inst[1:18]
18
calculate immediate
32
immx
imm. operands
pc
inst
27
shift by 2 bits and extend sign
32
branchTarget
isRet
ra(15)
rs1
inst[19:22]
rd
inst[23:26]
rs2
inst[15:18]
1
0
reg. operands
A read port 1
A read port 2
Register file
D
op1
op2
isSt
A address
D data
Contain Content which will be used for further processing

EX Stage – Branch Unit



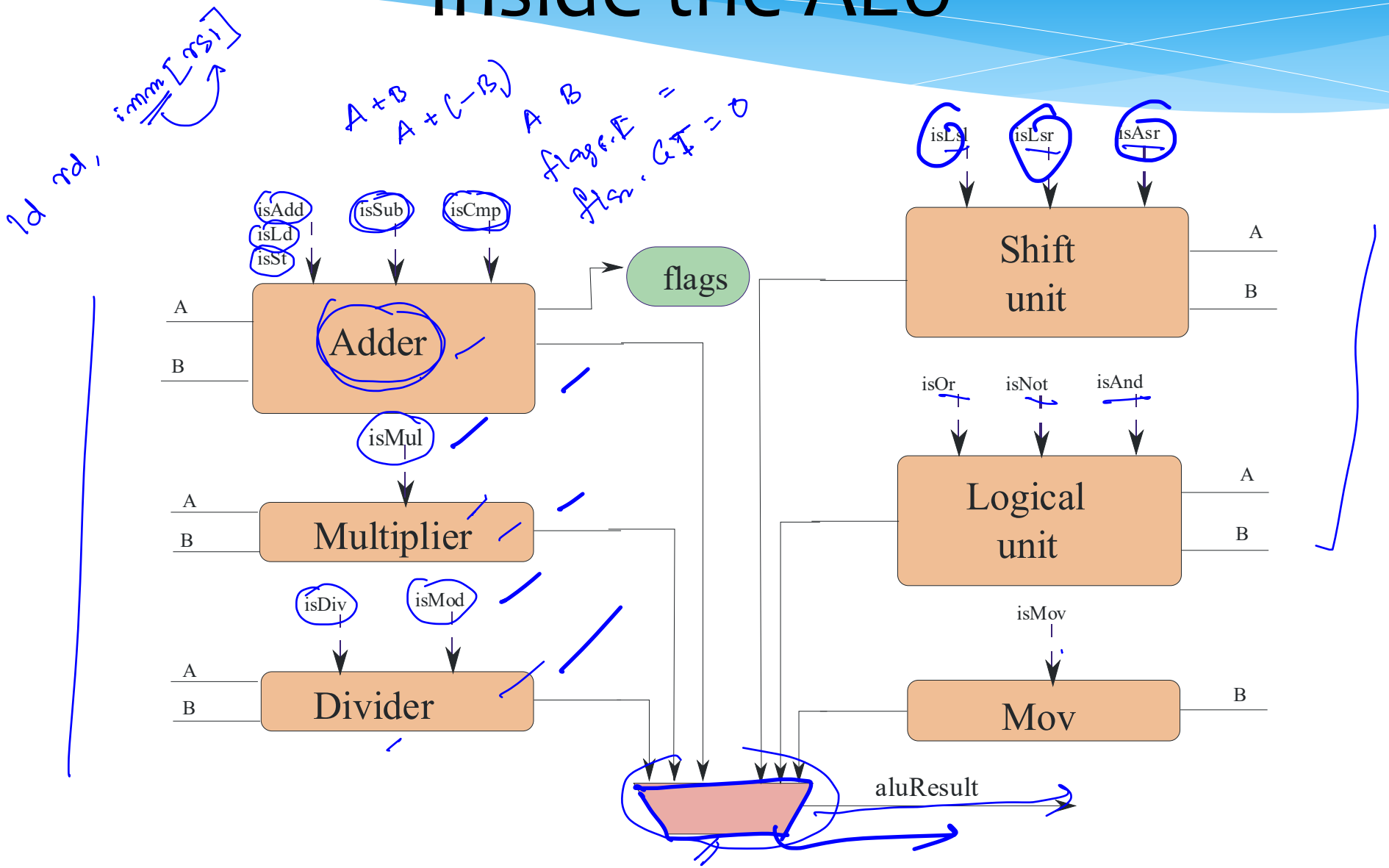
Generates the isBranchTaken Signal

ALU



Choose between immx and op2 based on the value of the I bit

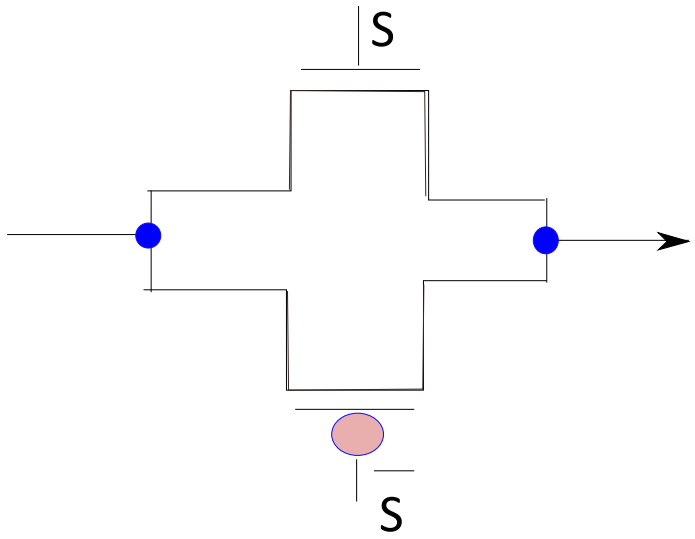
Inside the ALU

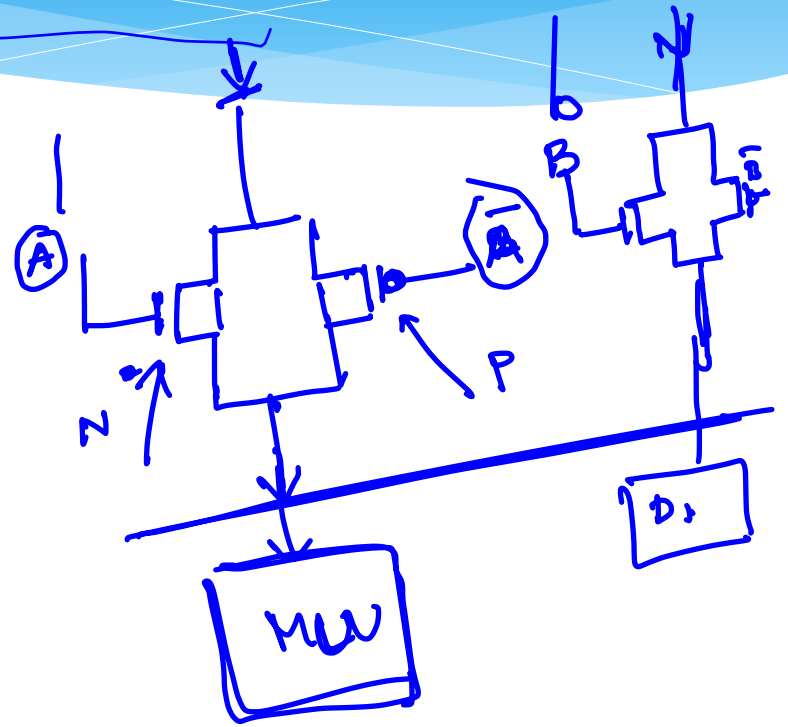


Disabling some Inputs

- We do not want all the units of the ALU to be active at the same time because of we want to save power
- The instruction will only use 1 unit
- Power is dissipated when the inputs or outputs make a transition (0 → 1, 1 → 0)
  - We shall avoid a transition by not letting the new inputs to propagate to units that do not require them
  - They will thus have the old inputs (no switching)

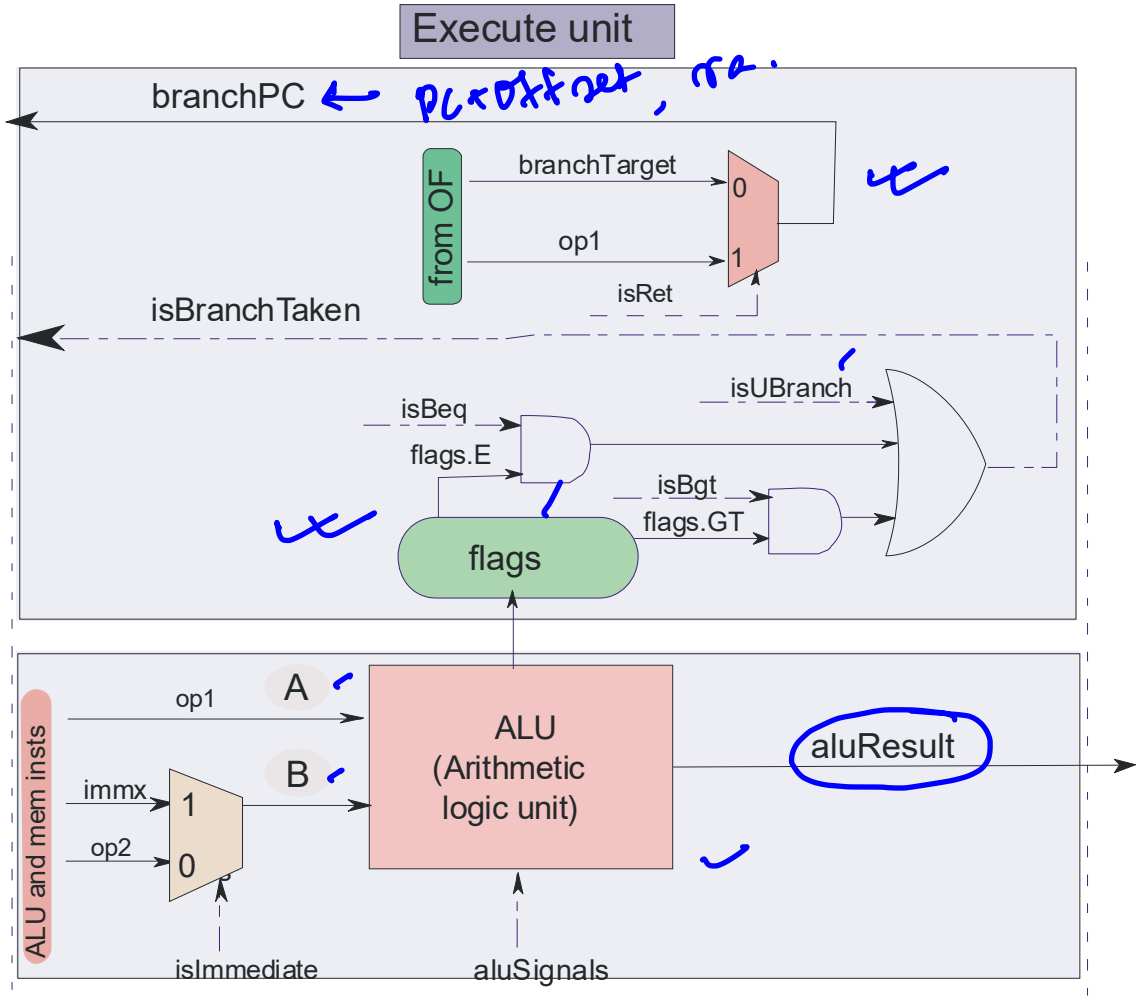
Use a Transmission Gate



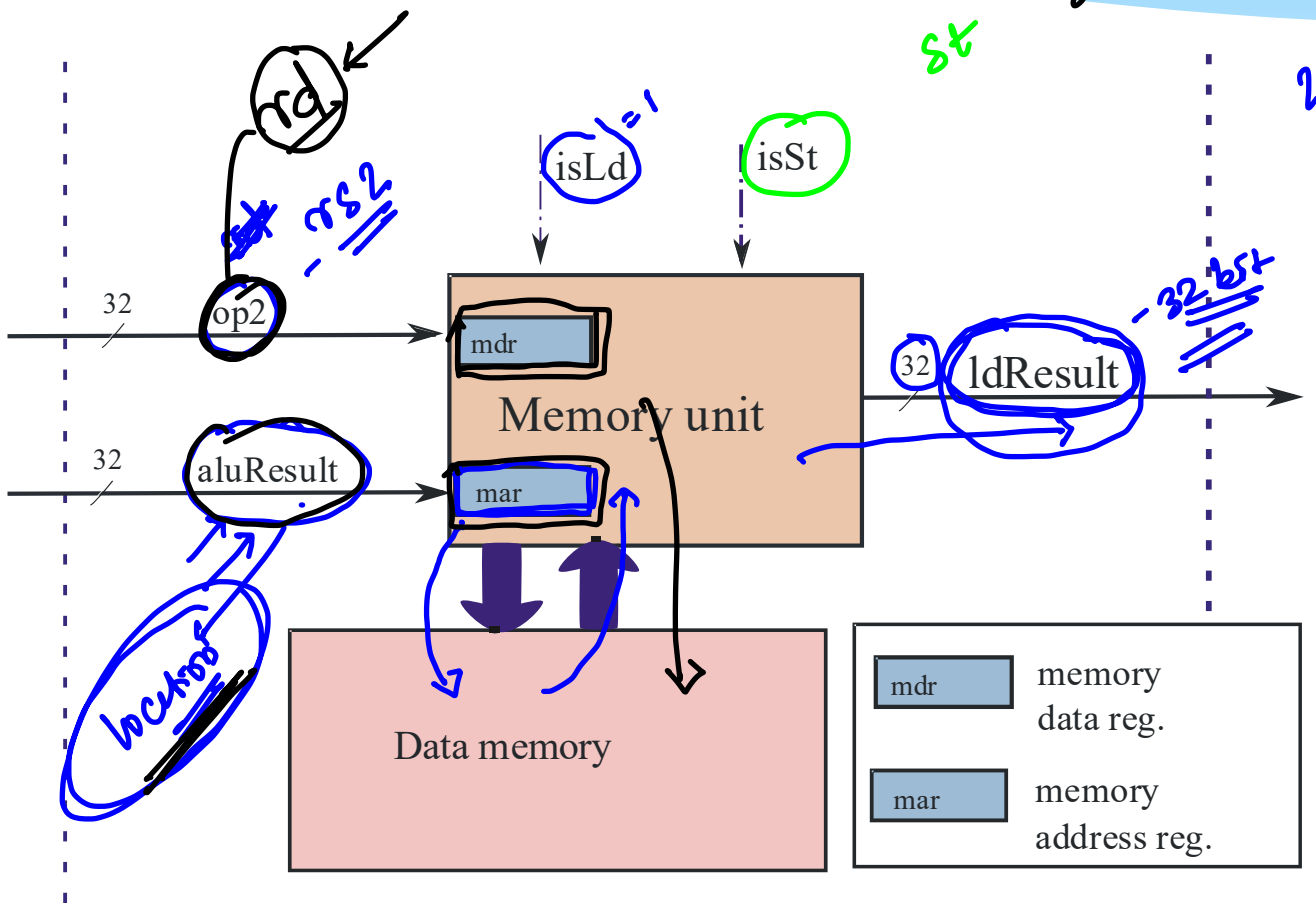


- output = input (if S = 1)
- Otherwise, the output is totally disconnected from the input

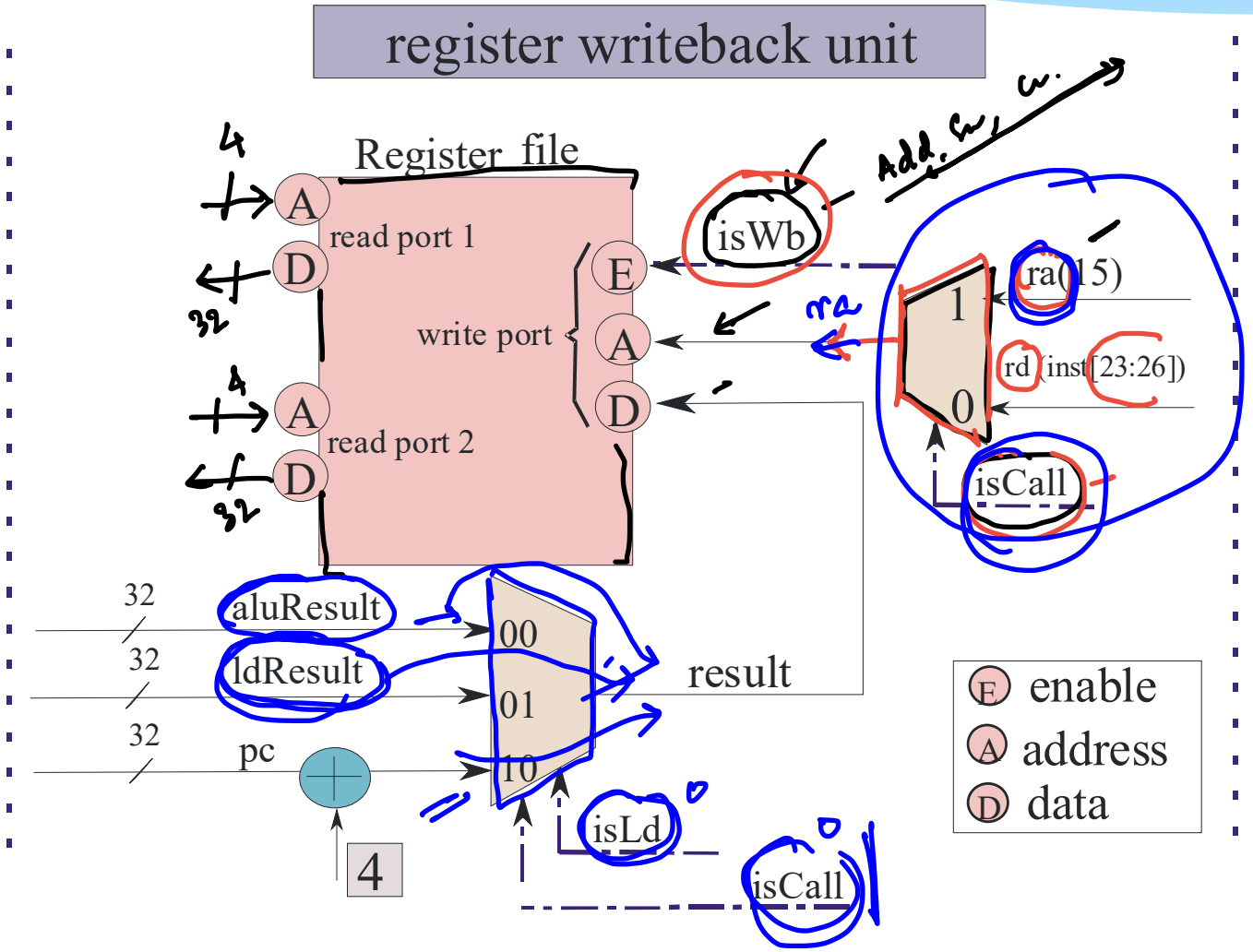
EX Unit

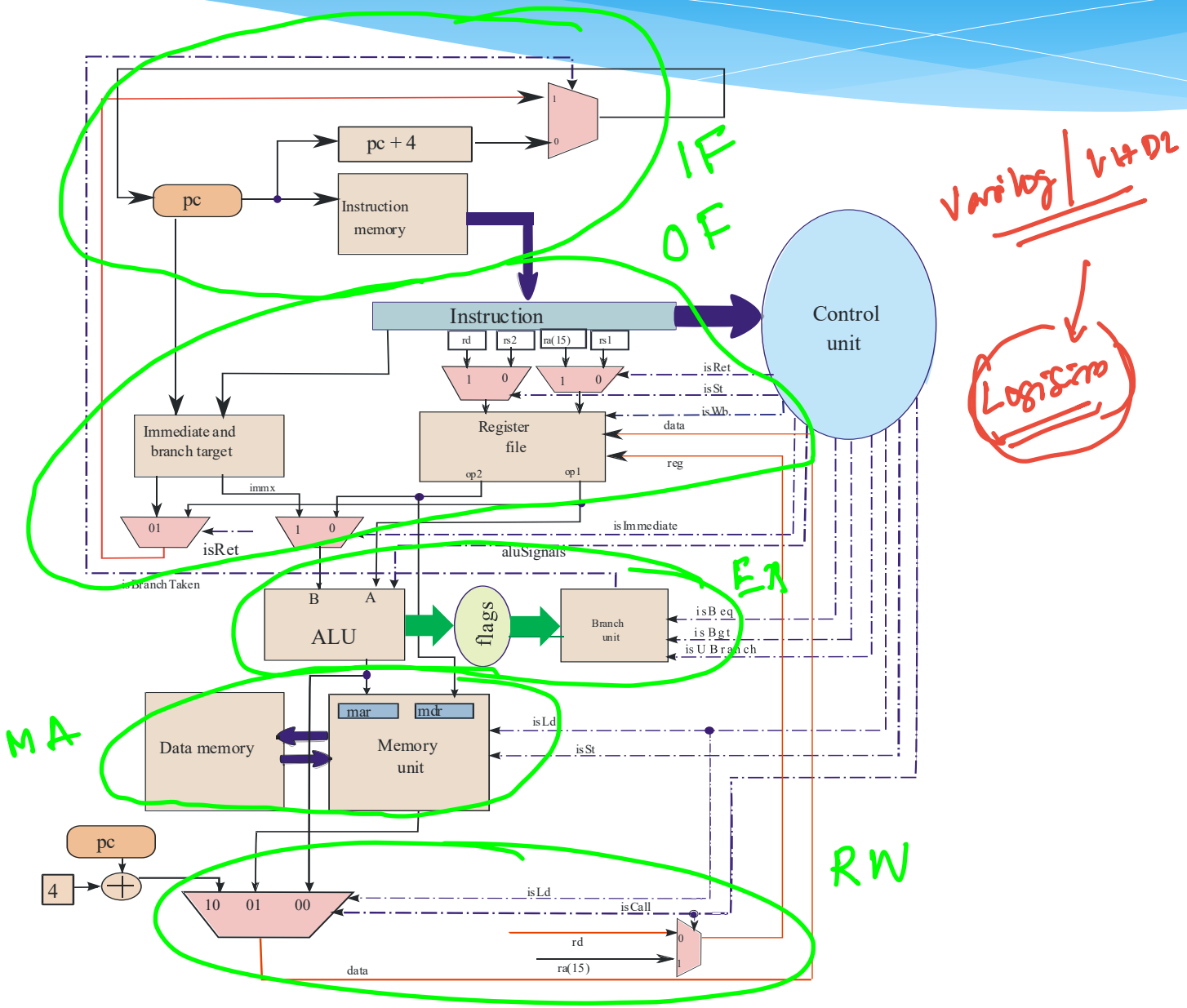

Execute unit
branchPC
from OF
branchTarget
op1
0
1
isRet
isBranchTaken
isUBranch
isBeq
flags.E
isBgt
flags.GT
flags
ALU and mem insts
op1
A
B
immx
op2
1
0
ALU
(Arithmetic
logic unit)
aluResult
isImmediate
aluSignals

MA Unit


rd
rs2
isLd
isSt
32
op2
mdr
Memory unit
32
ldResult
32
aluResult
mar
Data memory
mdr
memory
data reg.
mar
memory
address reg.

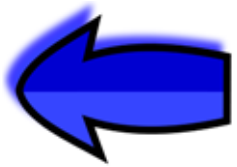
RW Unit


register writeback unit
Register file
A
D
read port 1
write port
E
A
D
A
D
read port 2
32
32
4
4
isWb
1
0
ra(15)
rd (inst[23:26])
isCall
32
aluResult
32
ldResult
32
pc
00
01
10
result
4
isLd
isCall
enable
address
data

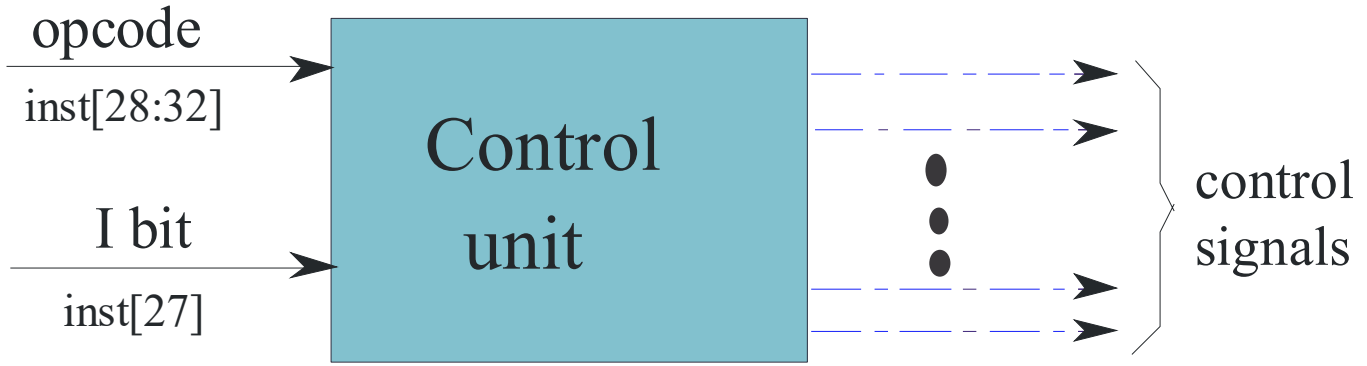

pc + 4
pc
Instruction memory
Instruction
Control unit
rd
rs2
ra(15)
rs1
isRet
isSt
isWb
data
reg
Immediate and branch target
Register file
op2
op1
immx
01
isRet
isImmediate
aluSignals
isBranchTaken
B
A
ALU
flags
Branch unit
isBeq
isBgt
isUBranch
mar
mdr
Data memory
Memory unit
isLd
isSt
pc
4
10
01
00
isLd
isCall
rd
ra(15)
data
IF
OF
EX
MA
RW
Verilog / VHDL
Logisim

Outline

* Outline of a Processor
* Detailed Design of each Stage
* The Control Unit
* Microprogrammed Processor
* Microassembly Language
* The Microcontrol Unit



The Ha rdwired Control Unit



* Given the opcode and the immediate bit
  * It generates all the control signals

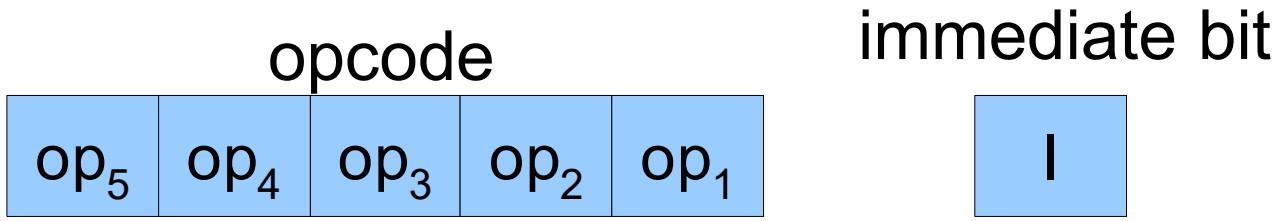
Control Signals

SerialNo.	Signal	Condition
1	*isSt*	Instruction: *st*
2	*isLd*	Instruction: *ld*
3	*isBeq*	Instruction: *beq*
4	*isBgt*	Instruction: *bgt*
5	*isRet*	Instruction: *ret*
6	*isImmediate*	*I* bit set to 1
7	*isWb*	Instructions: *add*, *sub*, *mul*, *div*, *mod*, *and*, *or*, *not*, *mov*, *ld*, *lsl*, *lsr*, *asr*, *call*
8	*isUBranch*	Instructions: *b*, *call*, *ret*
9	*isCall*	Instructions: *call*

Control Signals – II

	aluSignal	
10	*isAdd*	Instructions: *add*, *ld*, *st*
11	*isSub*	Instruction: *sub*
12	*isCmp*	Instruction: *cmp*
13	*isMul*	Instruction: *mul*
14	*isDiv*	Instruction: *div*
15	*isMod*	Instruction: *mod*
16	*isLsl*	Instruction: *lsl*
17	*isLsr*	Instruction: *lsr*
18	*isAsr*	Instruction: *asr*
19	*isOr*	Instruction: *or*
20	*isAnd*	Instruction: *and*
21	*isNot*	Instruction: *not*
22	*isMov*	Instruction: *mov*

Control signal Logic



Serial No.	Signal	Condition
1	*isSt*	$\overline{op_5}.op_4.op_3.op_2.op_1$
2	*isLd*	$\overline{op_5}.op_4.op_3.op_2.\overline{op_1}$
3	*isBeq*	$op_5.\overline{op_4}.\overline{op_3}.\overline{op_2}.\overline{op_1}$
4	*isBgt*	$op_5.\overline{op_4}.\overline{op_3}.\overline{op_2}.op_1$
5	*isRet*	$op_5.\overline{op_4}.op_3.\overline{op_2}.\overline{op_1}$
6	*isImmediate*	I
7	*isWb*	$\sim(op_5 + \overline{op_5}.op_3.op_1.(op_4 + \overline{op_2})) + op_5.\overline{op_4}.\overline{op_3}.op_2.op_1$
8	*isUbranch*	$op_5.\overline{op_4}.(\overline{op_3}.op_2 + op_3.\overline{op_2}.\overline{op_1})$
9	*isCall*	$op_5.\overline{op_4}.\overline{op_3}.op_2.op_1$

Control Signal Logic - II

aluSignals		
10	*isAdd*	$\overline{op5}.\overline{op4}.\overline{op3}.\overline{op2}.\overline{op1} + \overline{op5}.op4.op3.op2$
11	*isSub*	$\overline{op5}.\overline{op4}.\overline{op3}.\overline{op2}.op1$
12	*isCmp*	$\overline{op5}.\overline{op4}.op3.\overline{op2}.op1$
13	*isMul*	$\overline{op5}.\overline{op4}.\overline{op3}.op2.\overline{op1}$
14	*isDiv*	$\overline{op5}.\overline{op4}.\overline{op3}.op2.op1$
15	*isMod*	$\overline{op5}.\overline{op4}.op3.\overline{op2}.\overline{op1}$
16	*isLsl*	$\overline{op5}.op4.\overline{op3}.op2.\overline{op1}$
17	*isLsr*	$\overline{op5}.op4.\overline{op3}.op2.op1$
18	*isAsr*	$\overline{op5}.op4.op3.\overline{op2}.\overline{op1}$
19	*isOr*	$\overline{op5}.\overline{op4}.op3.op2.op1$
20	*isAnd*	$\overline{op5}.\overline{op4}.op3.op2.\overline{op1}$
21	*isNot*	$\overline{op5}.op4.\overline{op3}.\overline{op2}.\overline{op1}$
22	*isMov*	$\overline{op5}.op4.\overline{op3}.\overline{op2}.op1$