

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**

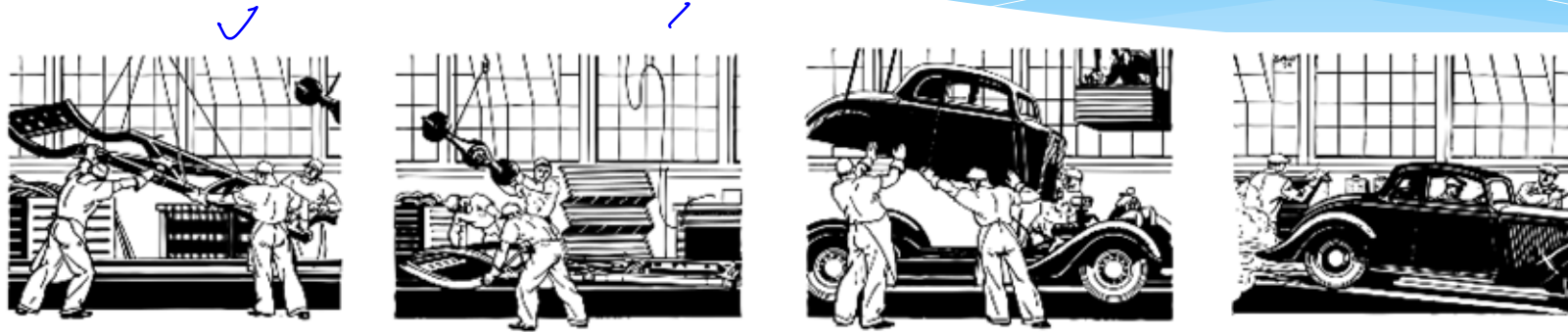
32 bits

- * **Implement** the entire SimpleRisc ISA
- * **Process** the binary format of instructions
- * Provide as much of **performance** as possible

- * **Basic Approach**

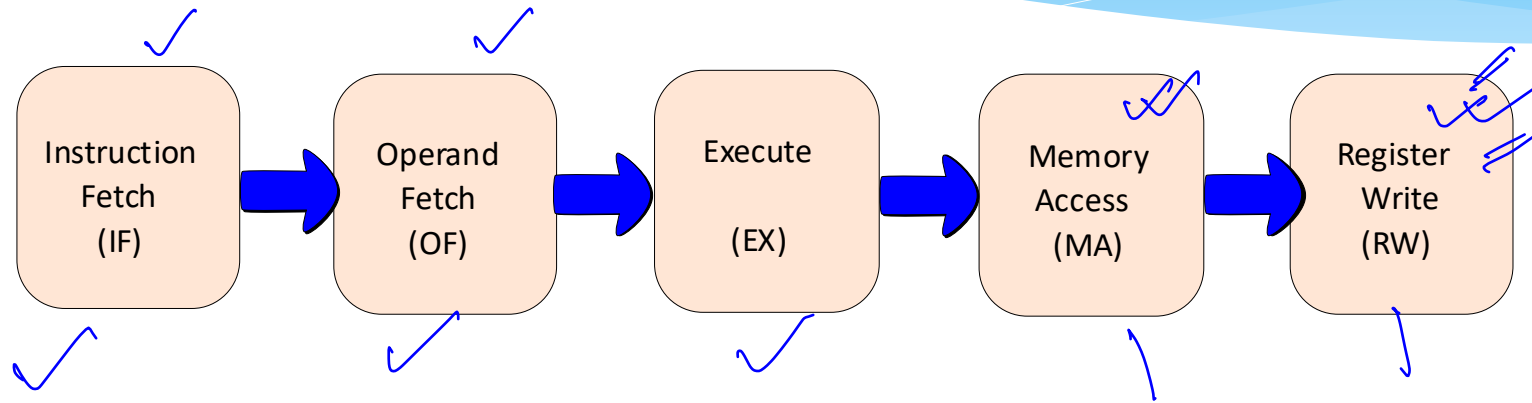
- * Divide the processing into stages
- * Design each stage separately

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)

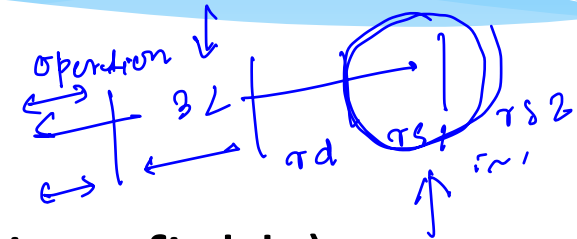
* **Decode** the instruction (break it into fields)

* **Fetch** the register operands from the register file

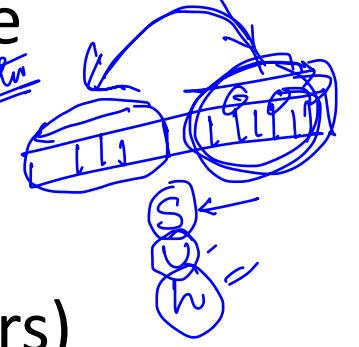
* Compute the **branch target** (PC + offset)

* Compute the **immediate** (16 bits + 2 bit modifiers)

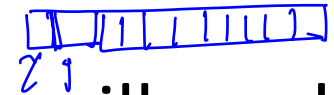
* ~~Generate **control signals** (we will see later)~~



word \Rightarrow 4 bytes

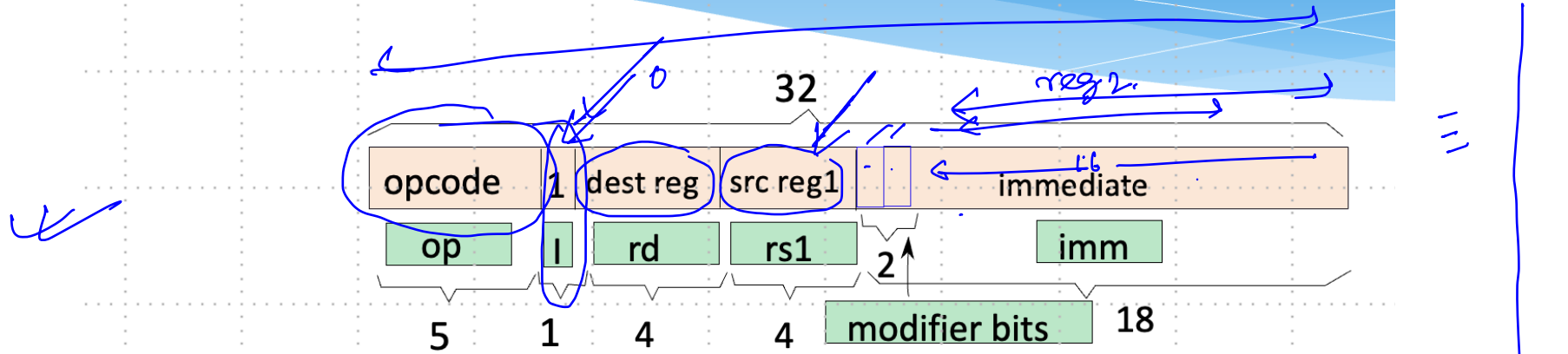


18 bits



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)

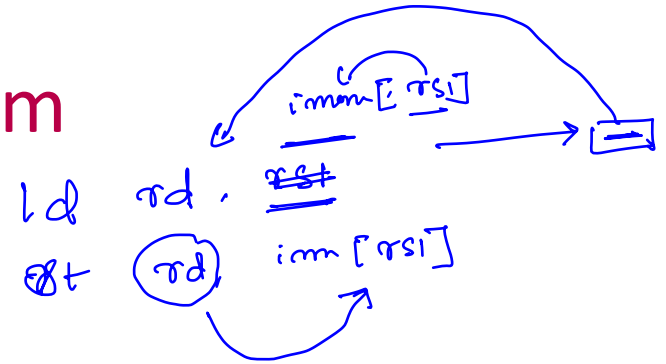
b →
beq →
bgt →

A - B

cmp A B

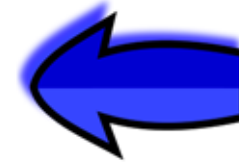
MA and RW Stages

- * 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**

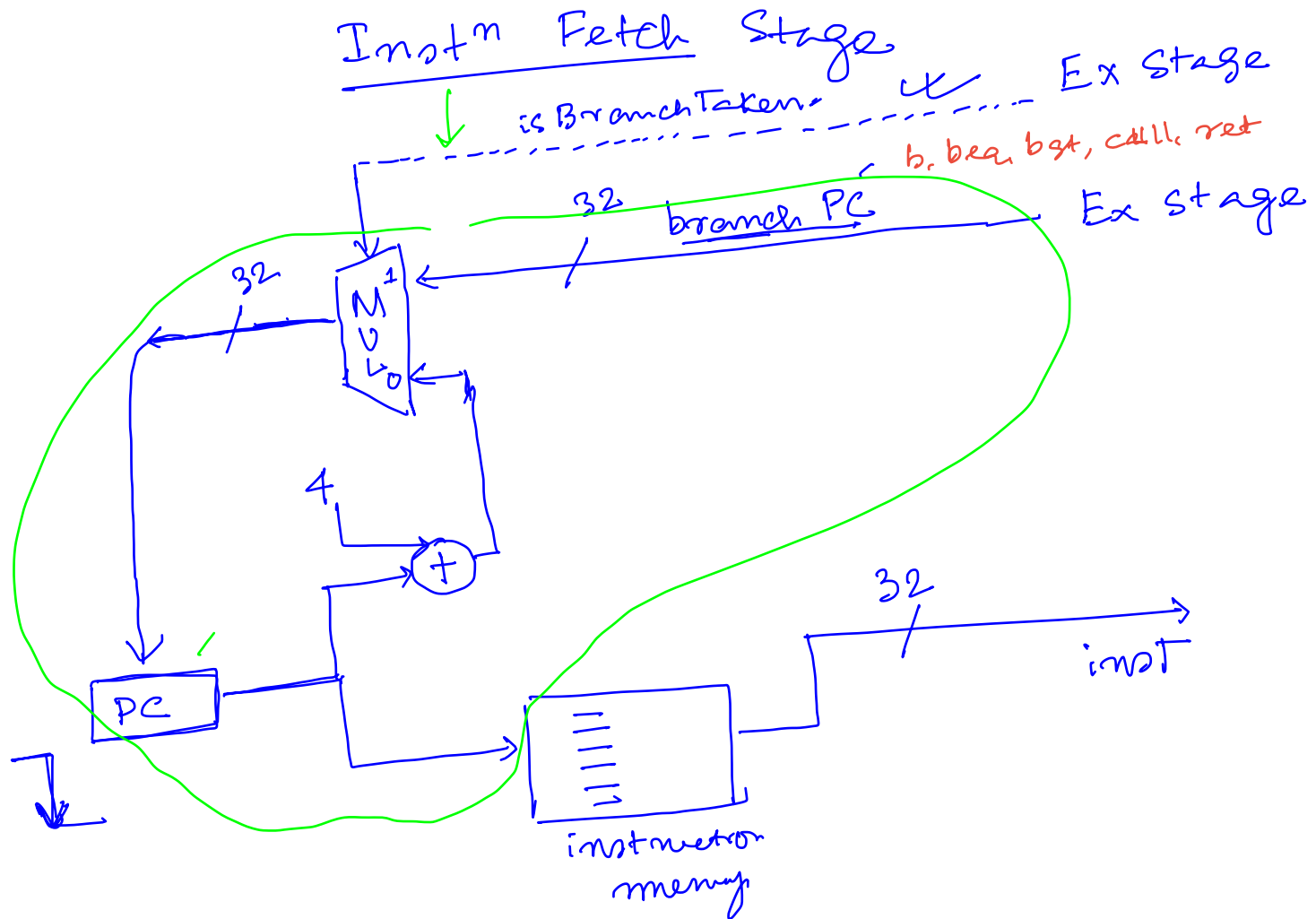


Outline

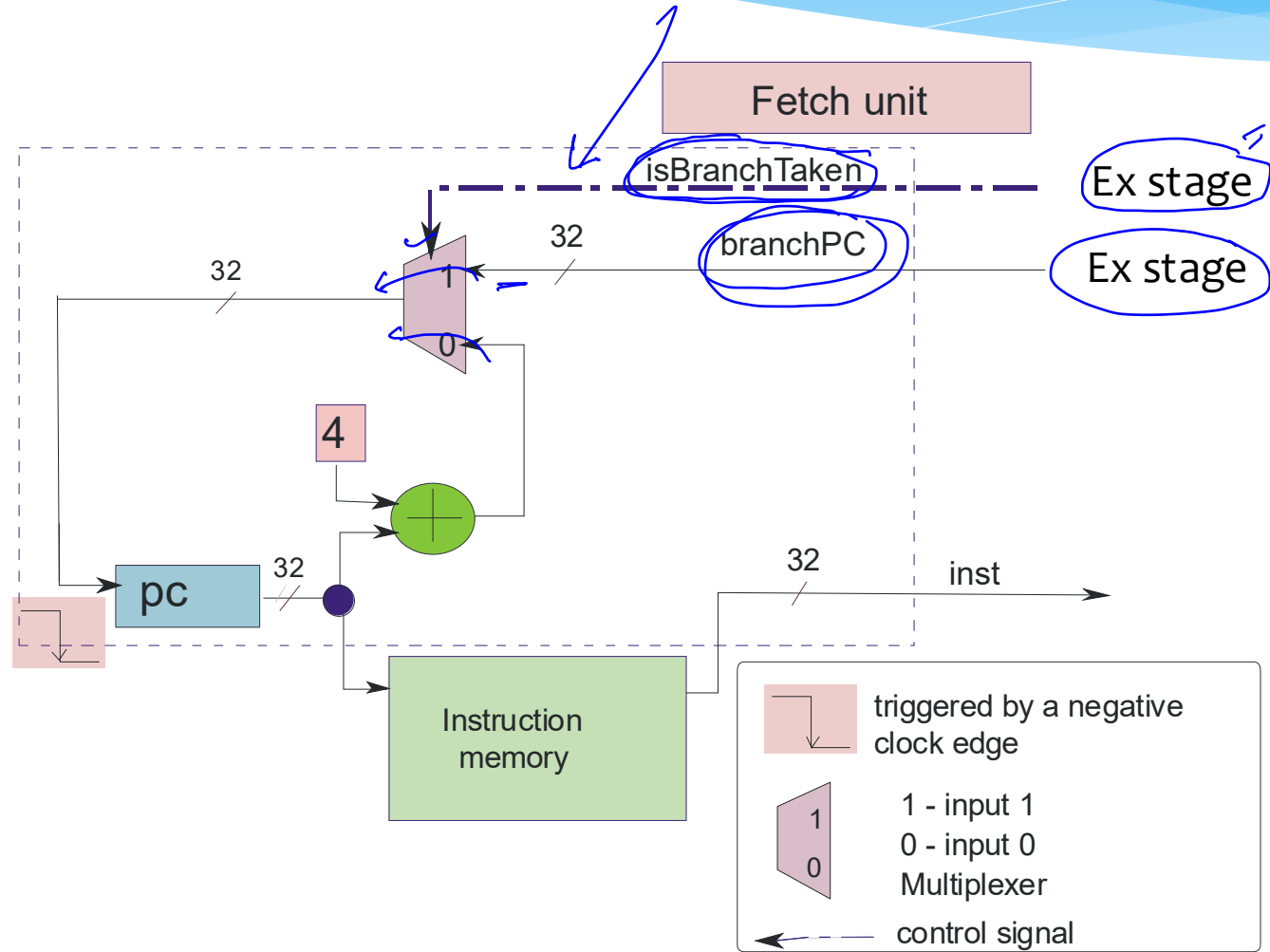
- * Outline of a Processor
- * Detailed Design of each Stage
- * The Control Unit
- * Microprogrammed Processor
- * Microassembly Language
- * The Microcontrol Unit



Instⁿ Fetch Stage



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 \rightarrow **isBranchTaken** ^{1/0}

isBranchTaken

- * isBranchTaken is a **control** signal
 - * It is generated by the EX unit
- * Conditions on isBranchTaken = 1

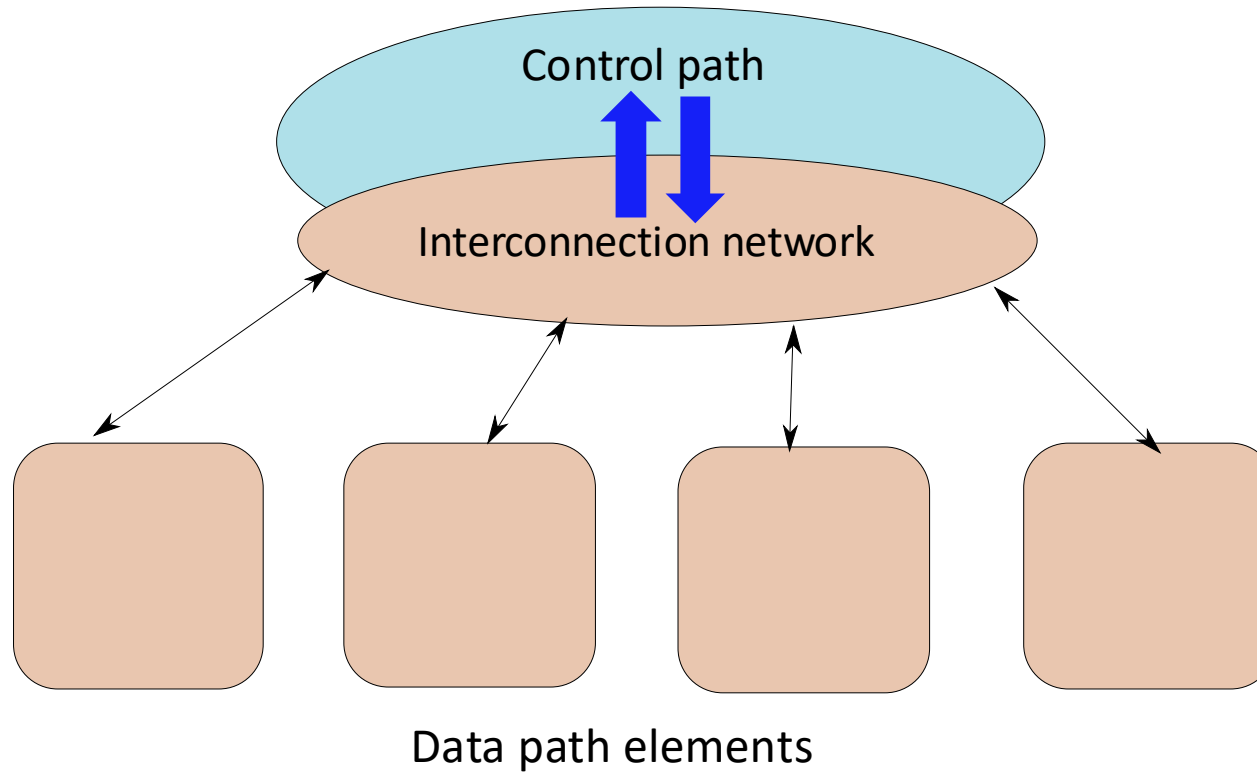
Instruction	Value of <i>isBranchTaken</i>
non-branch instruction	0
✓ <i>call</i>	1
✓ <i>ret</i>	1
✓ <i>b</i>	1
✓ <i>beq</i>	branch taken – 1 branch not taken – 0
✓ <i>bgt</i>	branch taken – 1 branch not taken – 0

$\frac{\text{flags.E} = 1}{\text{flags.gt} = 1}$

Data Path and Control Path

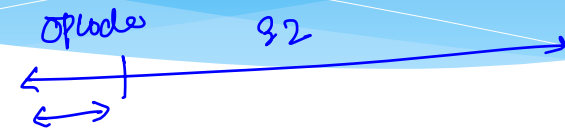
- * The **data path** consists of all the elements in a processor that are dedicated to storing, retrieving, and processing data such as **register files, memory, and the ALU**.
- * The **control path** primarily contains the **control unit**, whose role is to generate the appropriate signals to control the **movement of instructions**, and **data** in the data path.

Control Path

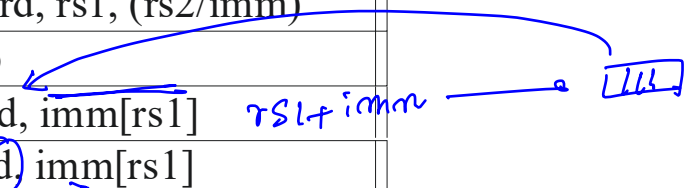
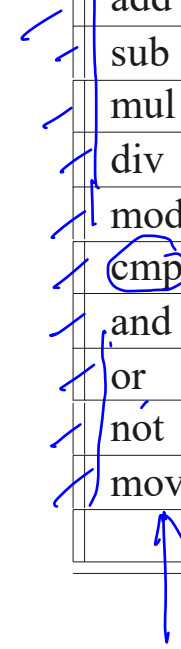


We will currently look at the hardwired control path.

Operand Fetch Unit



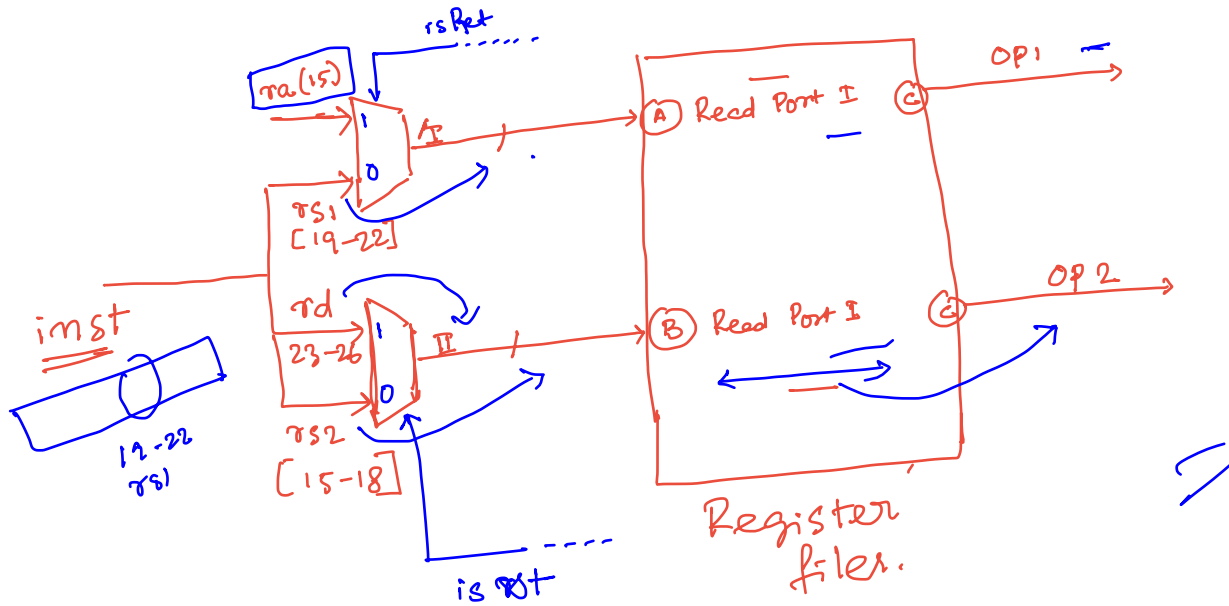
Inst.	Code	Format	Inst.	Code	Format
add	00000	add rd, rs1, (rs2/imm)	lsl	01010	lsl rd, rs1, (rs2/imm)
sub	00001	sub rd, rs1, (rs2/imm)	lsr	01011	lsr rd, rs1, (rs2/imm)
mul	00010	mul rd, rs1, (rs2/imm)	asr	01100	asr rd, rs1, (rs2/imm)
div	00011	div rd, rs1, (rs2/imm)	nop	01101	nop
mod	00100	mod rd, rs1, (rs2/imm)	ld	01110	ld rd, imm[rs1]
cmp	00101	cmp rs1, (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

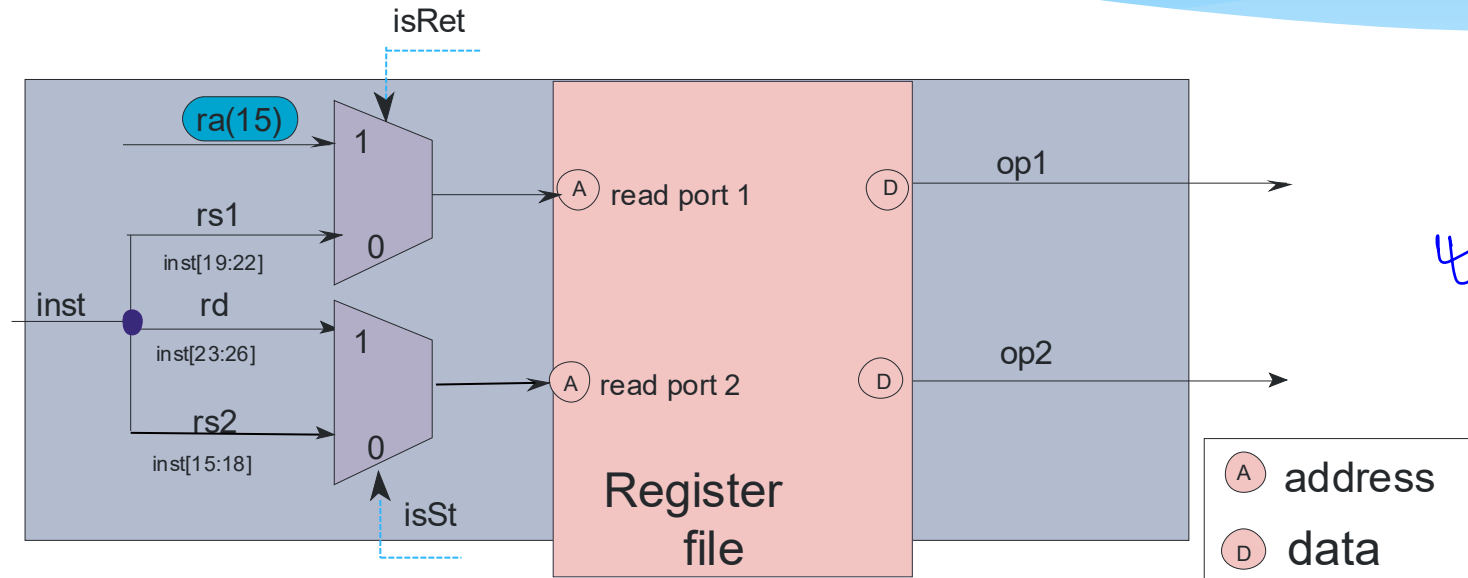
Format	Definition				
<i>branch</i>	<i>op</i> (28-32)	<i>offset</i> (1-27)	Word offset		
<i>register</i>	<i>op</i> (28-32)	<i>I</i> (27)	<i>rd</i> (23-26)	<i>rs1</i> (19-22)	<i>rs2</i> (15-18)
<i>immediate</i>	<i>op</i> (28-32)	<i>I</i> (27)	<i>rd</i> (23-26)	<i>rs1</i> (19-22)	<i>imm</i> (1-18)
<i>op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

Register File read
Immediate
branch Target



U.S.
 [11] ← 16

Register File Read

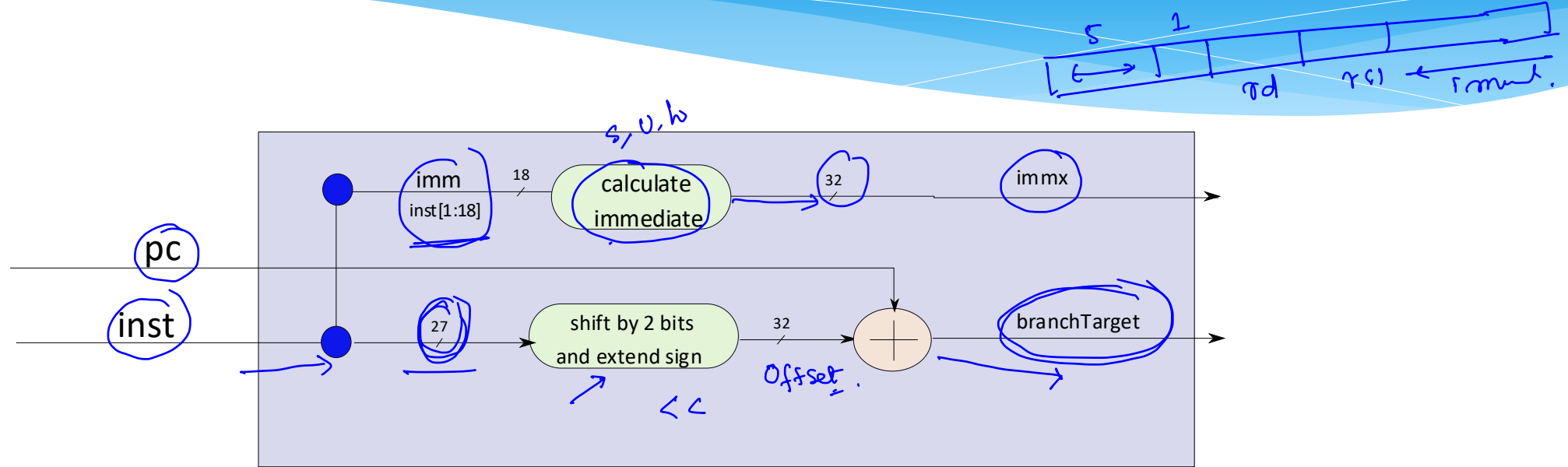


- * **First input** → *rs1* or *ra(15)* (ret instruction)
- * **Second input** → *rs2* or *rd* (store instruction)

Register File Access

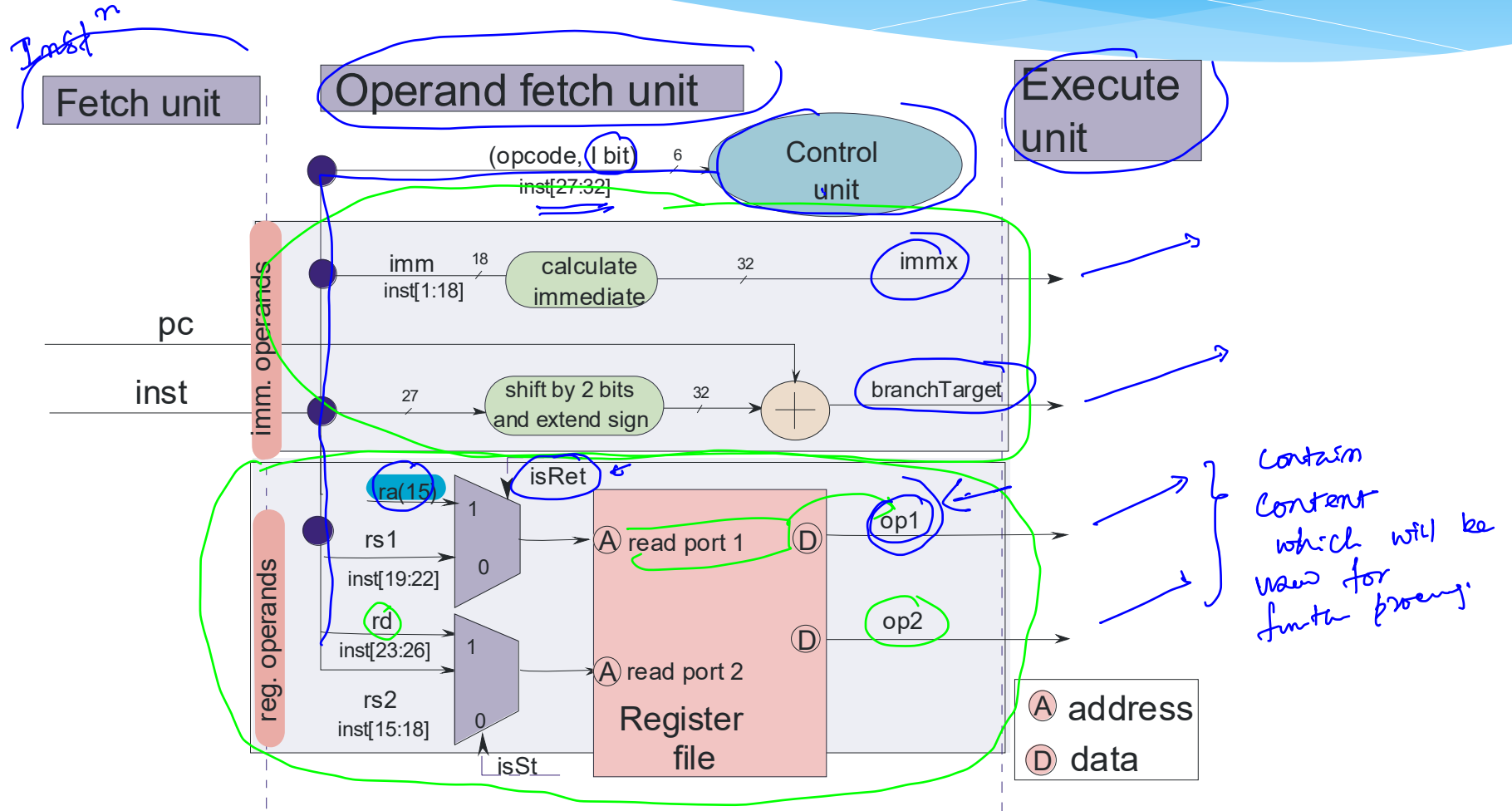
- * The **register file** has two **read ports**
 - * 1st Input
 - * 2nd Input
- * The two outputs are **op1**, and **op2**
 - * **op1** is the **branch target (return address)** in the case of a **ret** instruction, or **rs1**
 - * **op2** is the value that needs to be stored in the case of a **store**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

Immediate and Branch Unit

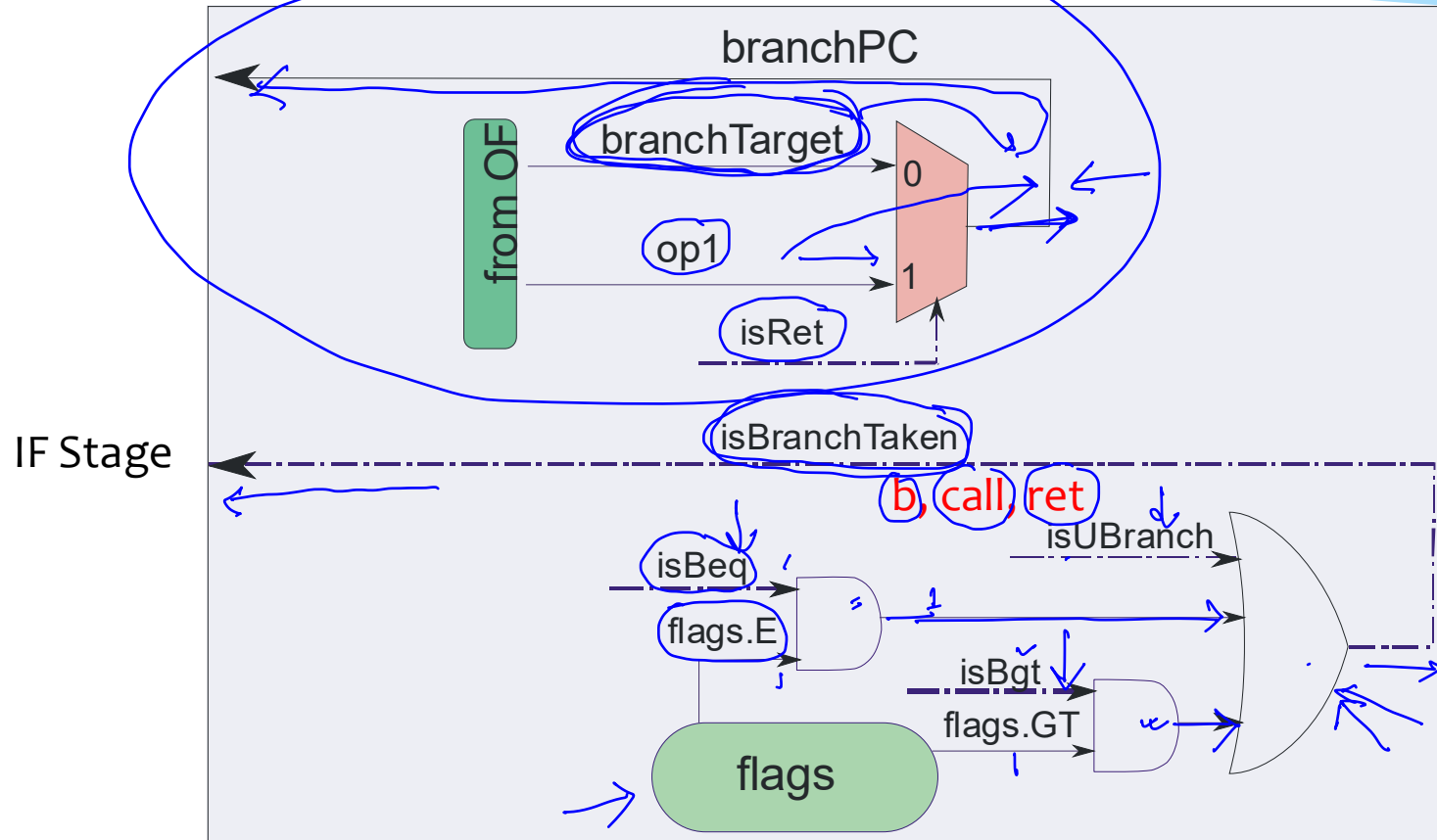


- * Compute `immx` (extended immediate), `branchTarget`, irrespective of the `instruction format`.
- * For the `branchTarget` we need to choose between the embedded target and op1 (ret)

OF Unit

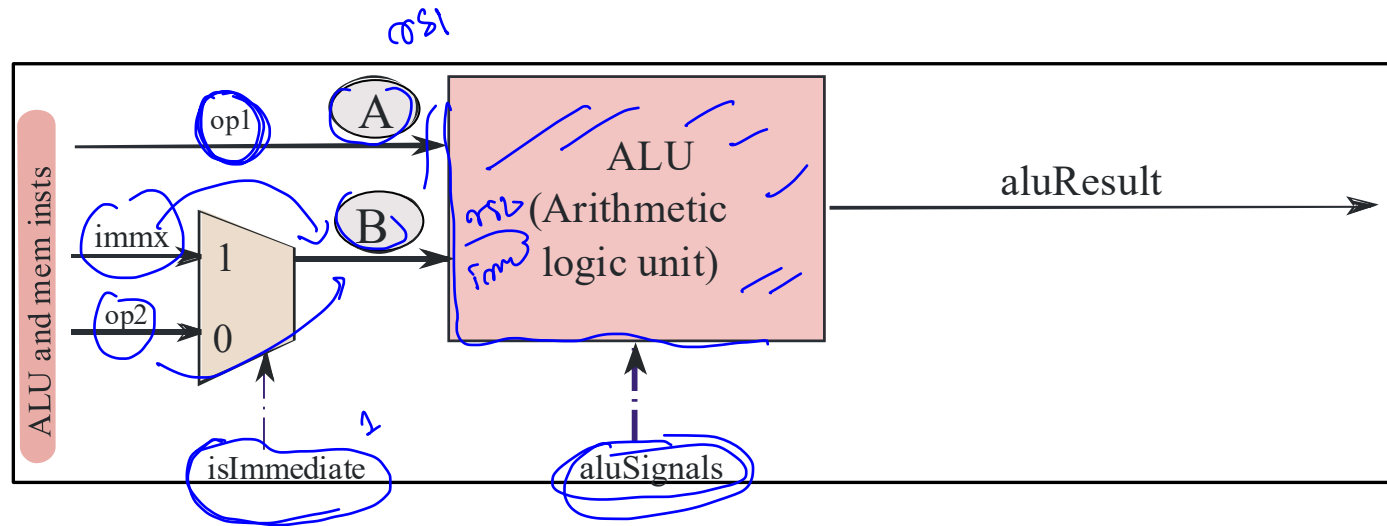


EX Stage – Branch Unit



Generates the `isBranchTaken` Signal

ALU



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

Inside the ALU

