Basic CPU Implementation

Wei Wang

Optional Readings from Textbooks

- “Computer Organization and Design,” Chapter 5 “The Processor:Datapath and Control.”
- “Computer Architecture: A Quantitative Approach,” Appendix A “Instruction Set Principles.”

Road Map

- Execution Stages Recap
- Data Path for ALU Instructions
- Data Path for Memory Instructions
- Data Path for Branch Instructions
- Control Signals and Multicycle Implementation
- Exceptions
- Micro-programming

Execution Stages Recap

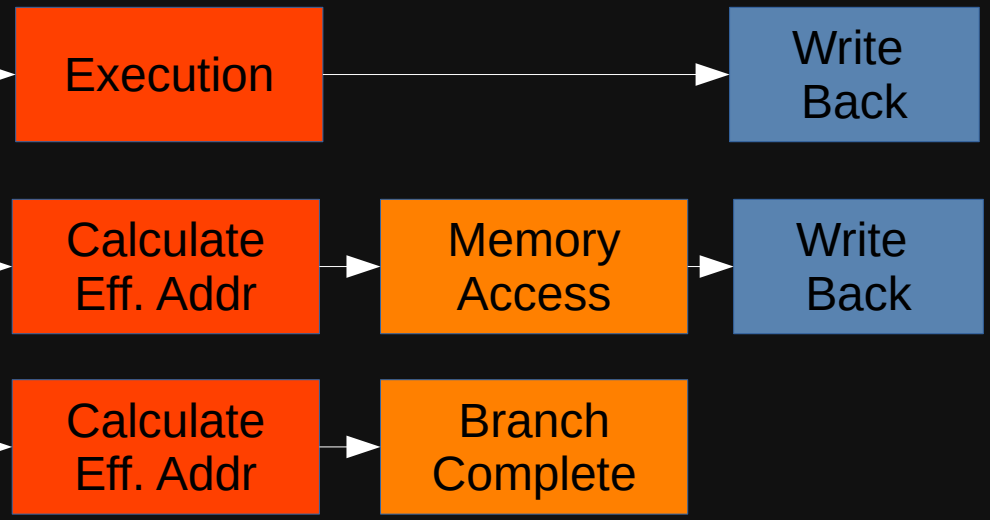
Instruction Execution Stages

- When implementing ISA, we typically partition the execution of an instruction into stages and implement each stage with transistors separately.
- Why
  - Many instructions share common steps in executions. Therefore, they can shared the common functional units.
  - Break down into stages with well-defined execution times (in CPU cycles) makes instruction scheduling and management easier.
  - Another divide-and-conquer or abstraction, simplifies the designing process.
  - To support CPU pipelining (more on this in the next lecture).

Stages of Instruction Execution

- Common stages for all instructions:

- Unique stages for different types of instructions:
  - ALU Ops:
  - Memory Ops:
  - Control Ops:



Five Common Stages of Instruction Executions

- In a typical yet simple RISC CPU implementation, the processor is partitioned into 5 connected stages.
- Every instruction goes through all stages, although this instruction does not necessarily trigger the functional units of every stage.

Data Path for ALU Instructions

ALU Instruction Execution Summary

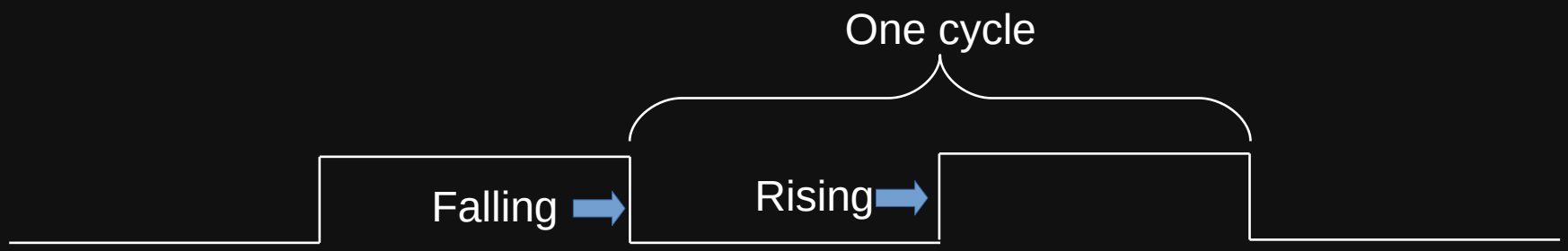
- An ALU instruction does arithmetic or logic operations on two source operands.
- Since we will learn a simple RISC implementations, the source operands and the destination operands are all in registers.
  - One source operand may also be a immediate value
- The instruction goes through four stages: IF, ID, EXEC and WB.
  - No memory access for ALU instructions.

Clock Cycles

- The start of each stage of execution is triggered by the clock signal.
  - The clock signal indicates the input data into the stage are stable and ready to be read; and the output data are stable and ready to be written out.
- The clocking methodology defines the approach used to determine when data is valid and stable relative to the clock.
- Here, we assume an edge-triggered clocking methodology.
  - The edge-triggered clocking methodology is a clocking scheme in which all state changes occur on a clock edge.

Edge-Triggered Clocking

- A typical clock cycle with rising (up) and falling (down) edges:



- The rising and falling edges trigger data read and write. For example,
  - All data reads of a stage happen at the falling edge and must be done before the clock rises again.
  - All data writes happen at the rising edge and must be done before the clock falls again.
- Of course, each clock cycle must be long enough for each stage to finish reading and writing data.
- No feedback in one cycle: a stage cannot read its own outputs in one cycle.

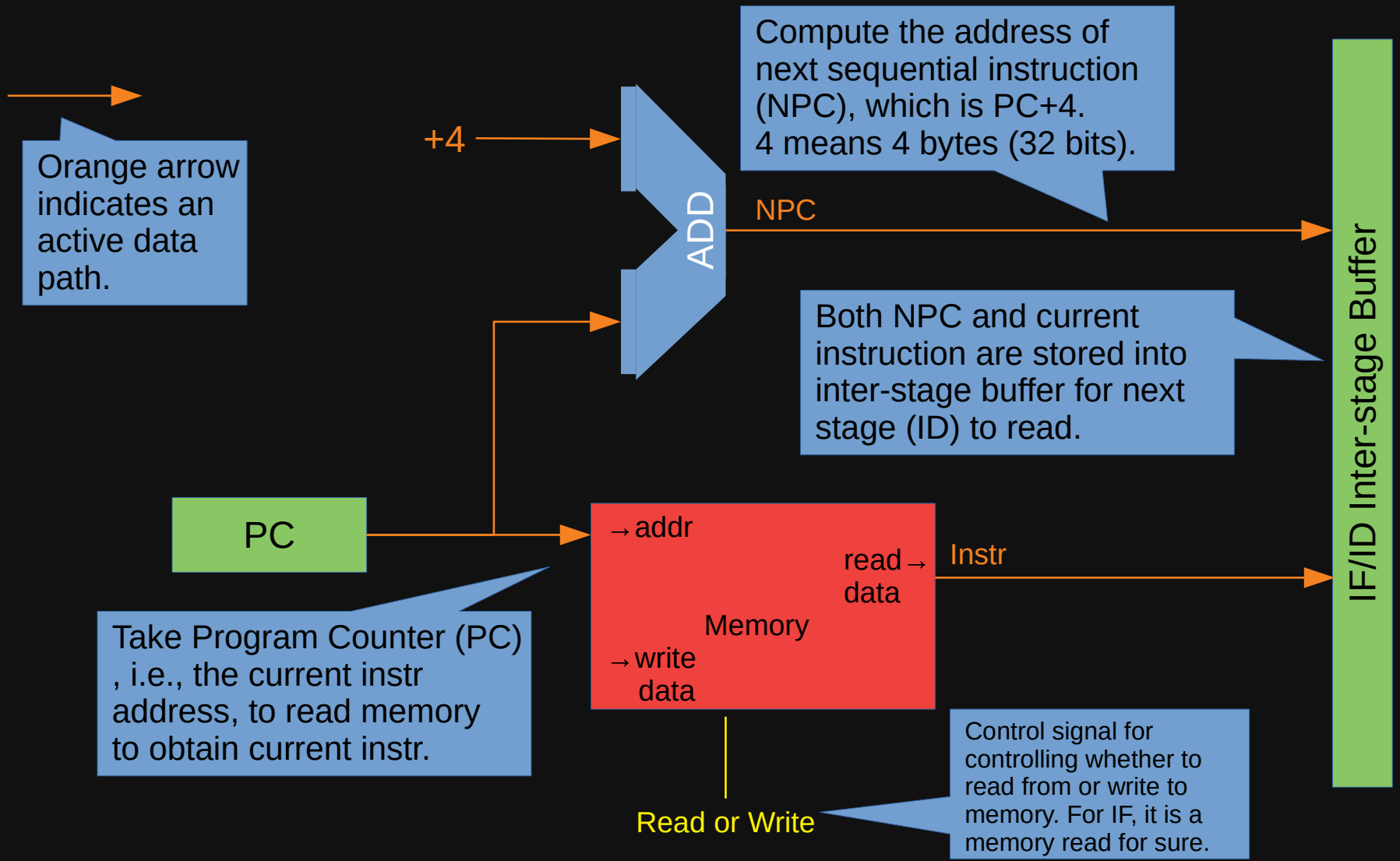
Control Signal

- Control signals are used for multiplexer (MUX) selection or for directing the operation of a functional unit; contrasts with a data signal, which contains information that is operated on by a functional unit.
- Control signals are usually generated by instruction decoder and the control unit to regulate the data flow in the data path.

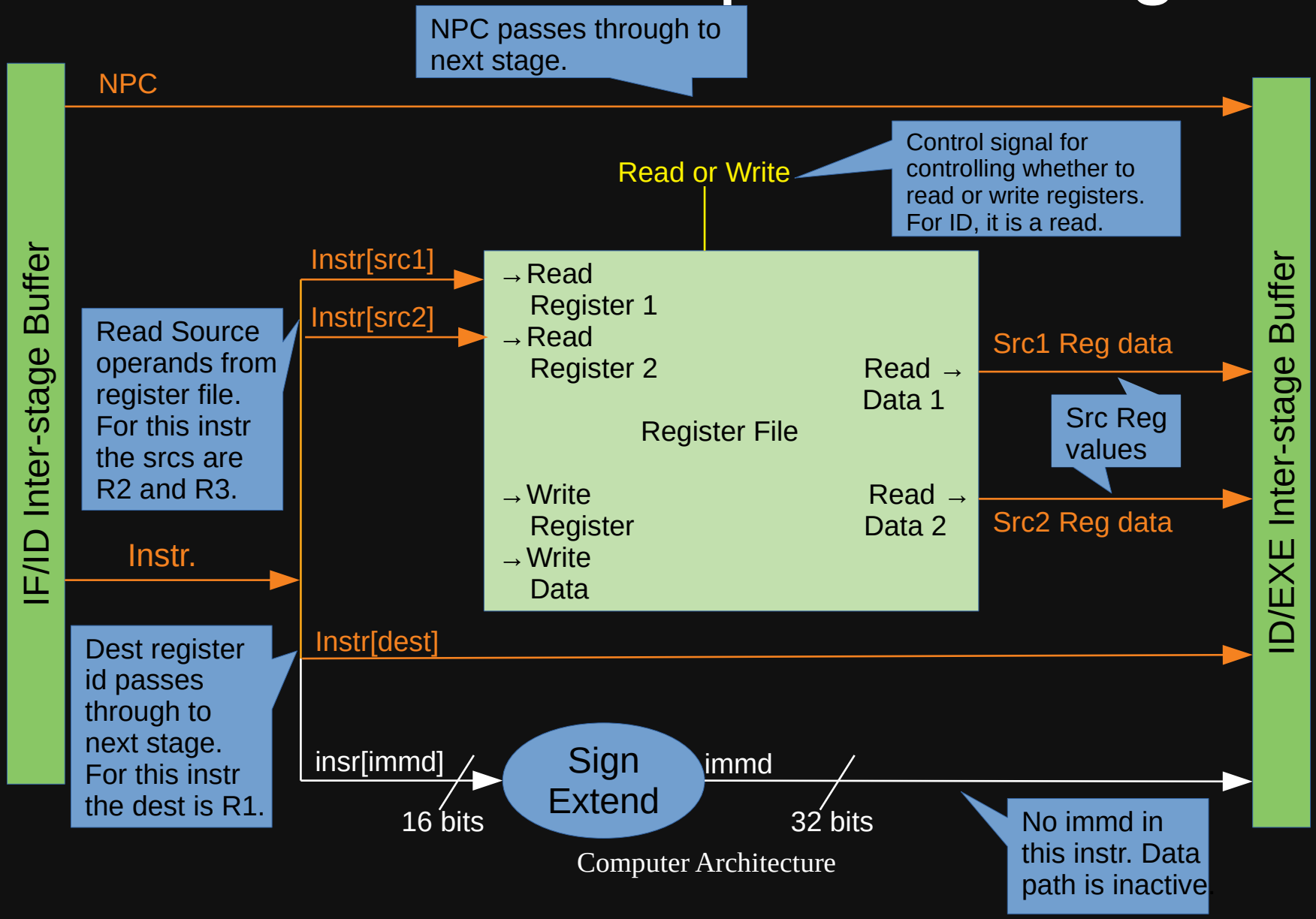
ALU Instruction Data Path with An Example

- Consider the following instruction as an example:
  - Instruction: `add R1, R2, R3`
  - Operation: `R1 = R2 + R3`
  - Source registers: `R2` (src1) and `R3` (src2)
  - Destination register: `R1`

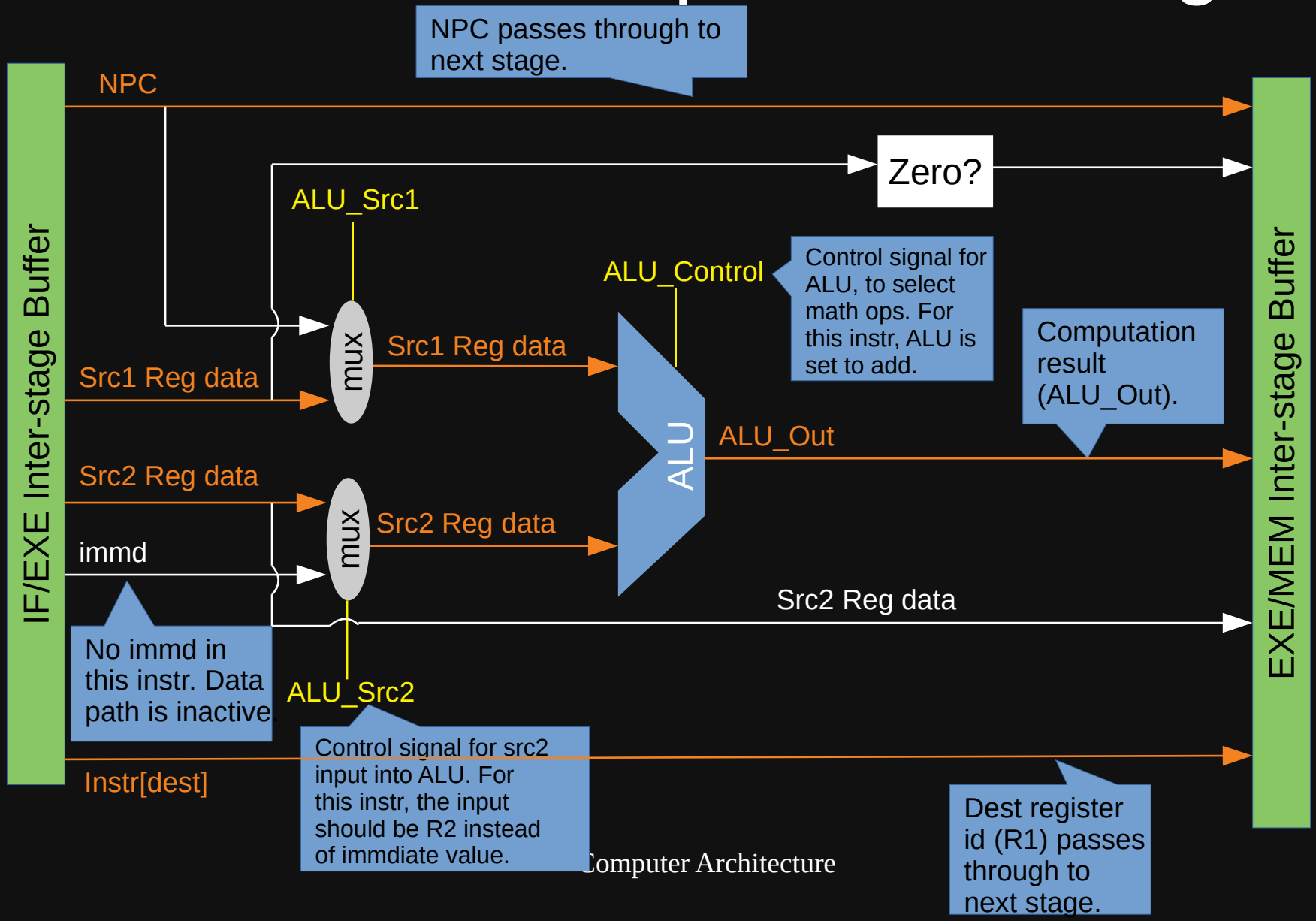
ALU Instruction Data Path with An Example: IF Stage



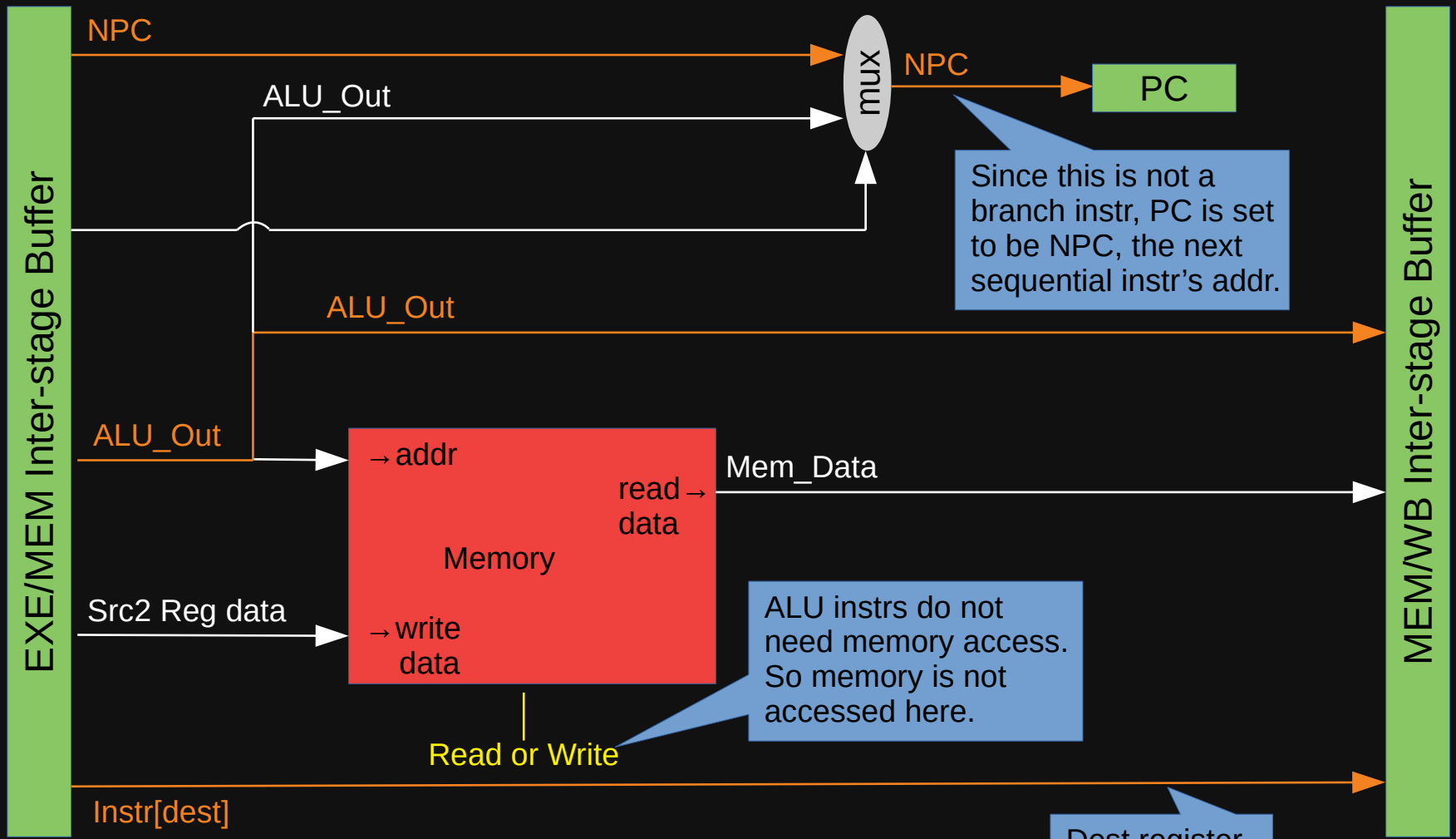
ALU Instruction Data Path with An Example: ID Stage



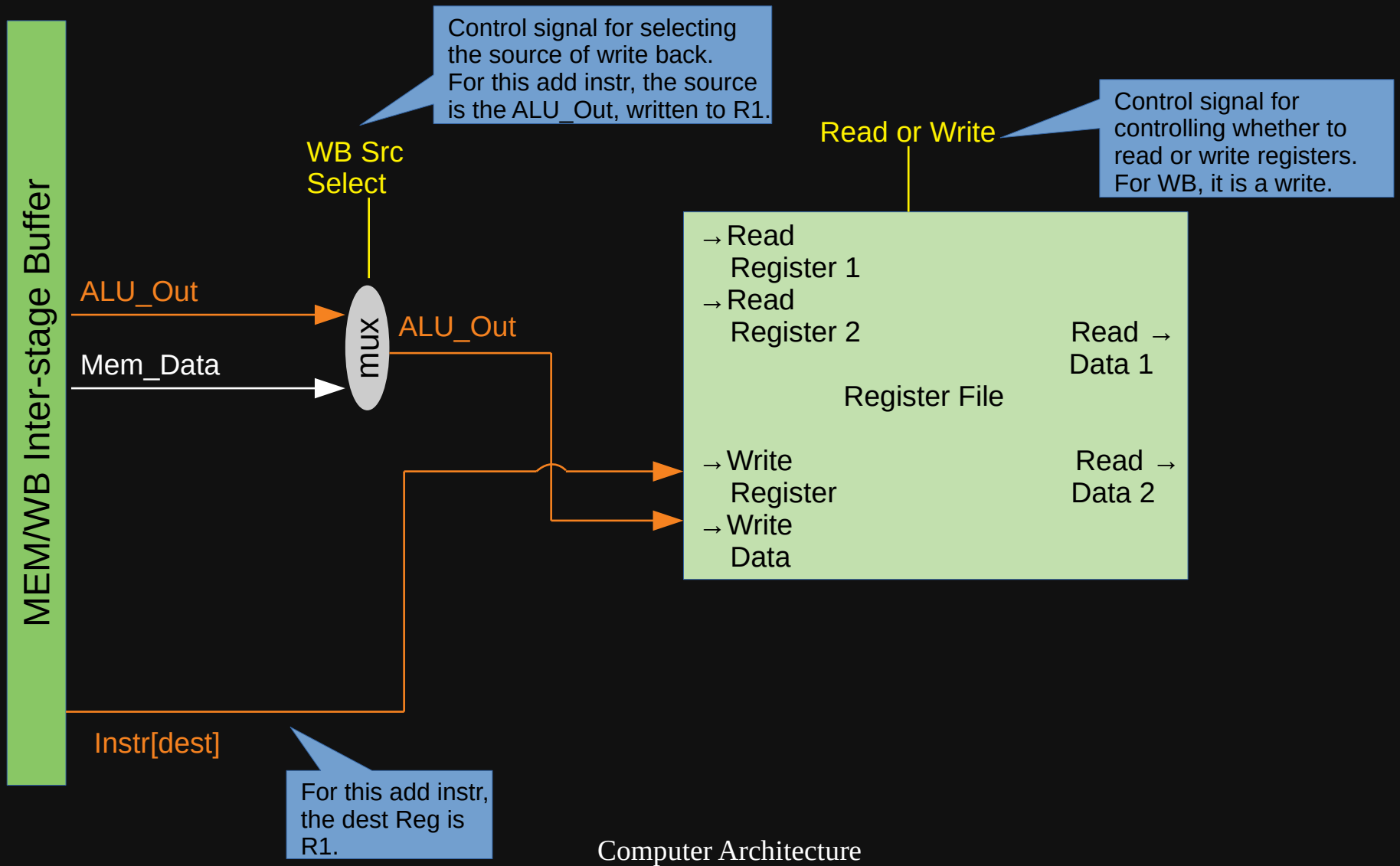
ALU Instruction Data Path with An Example: EXE Stage



ALU Instruction Data Path with An Example: MEM Stage



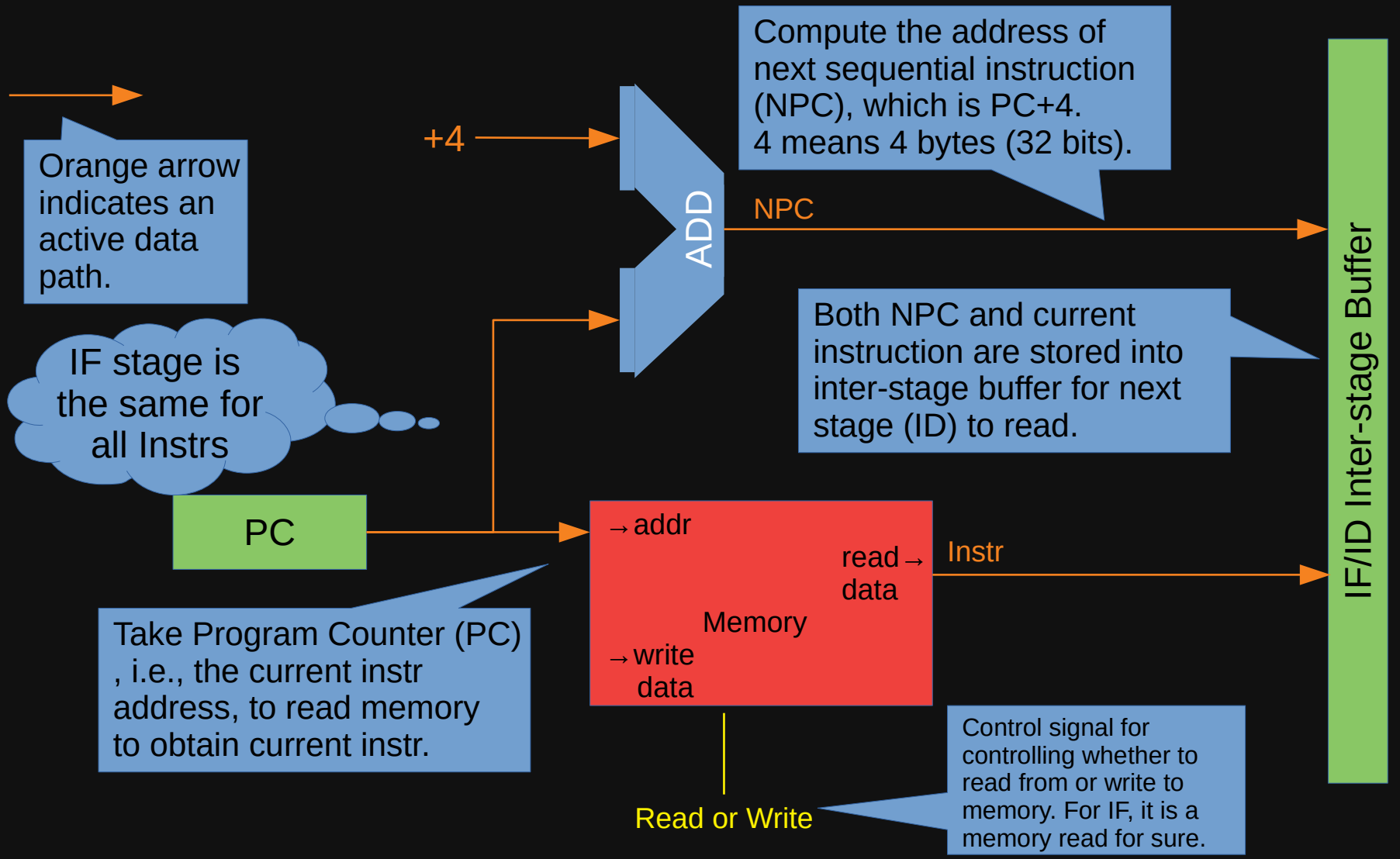
ALU Instruction Data Path with An Example: WB Stage



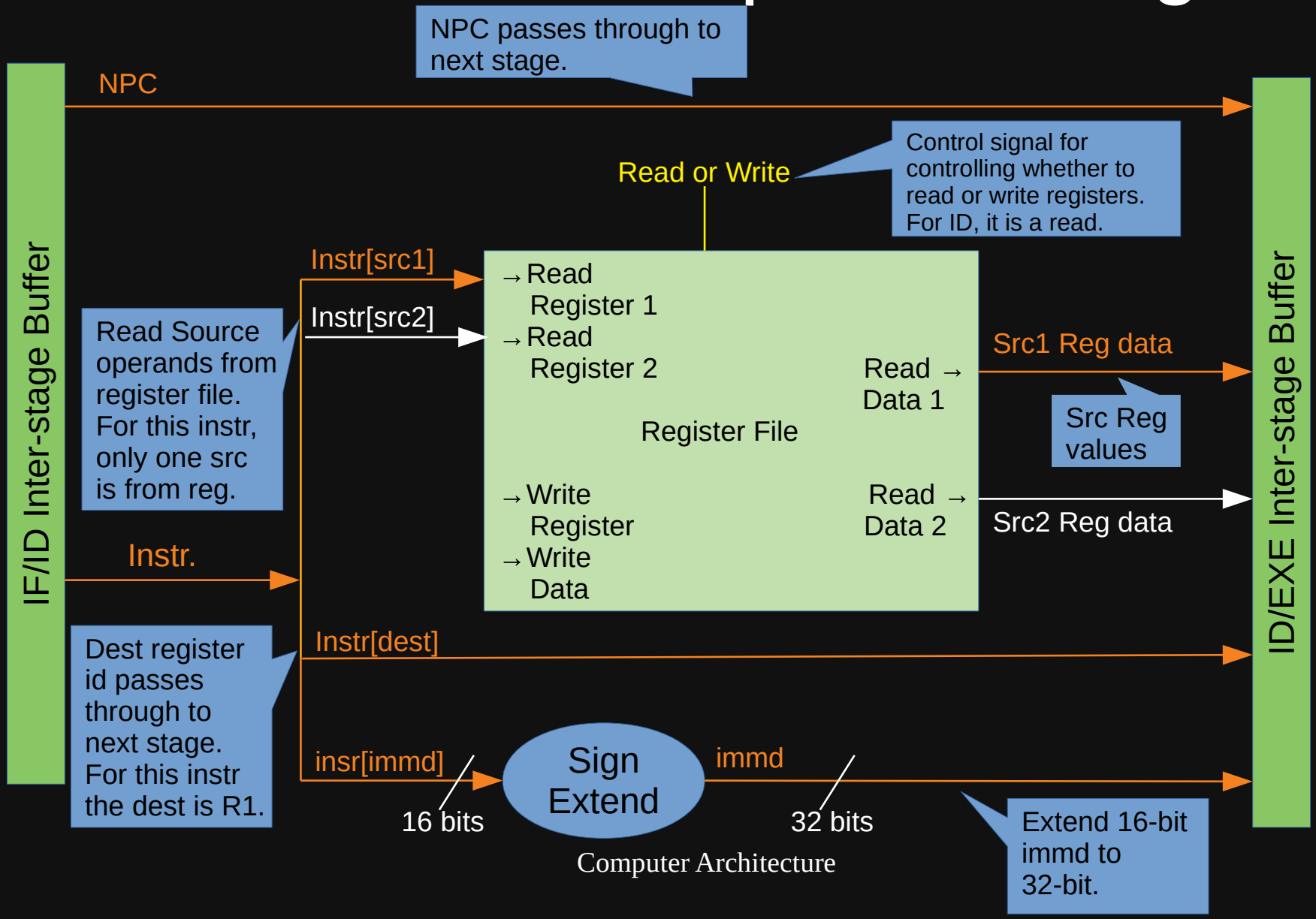
ALU Instruction Data Path with 2nd Example

- Consider the following instruction as an example:
  - Instruction: `sub R1, R2, 101`
  - Operation: `R1 = R2 - 101`
  - Source registers: `R2` (src1) and `101` (src2)
  - Destination register: `R1`
  - Immediate value: `101`

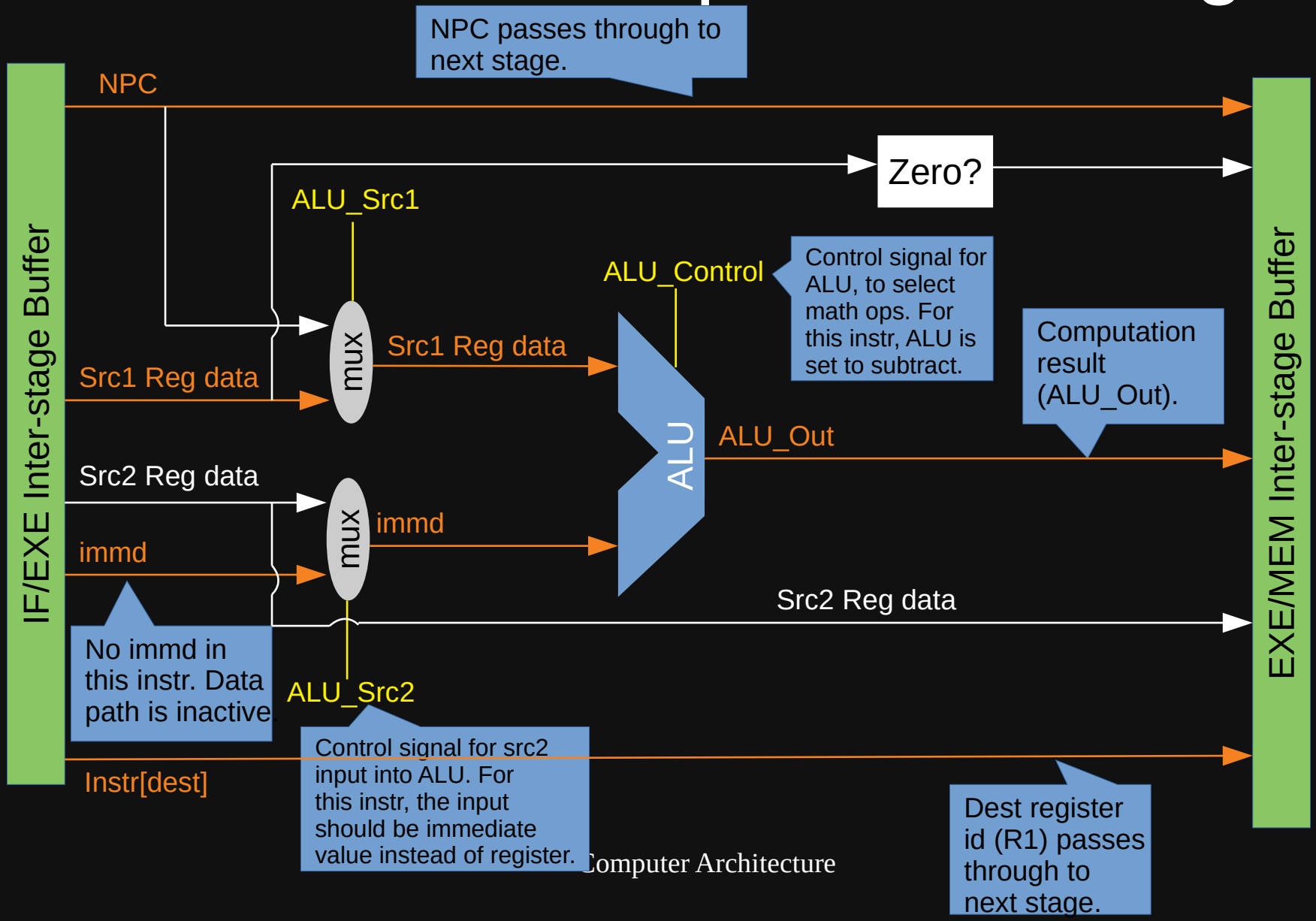
ALU Instruction Data Path with 2nd Example: IF Stage



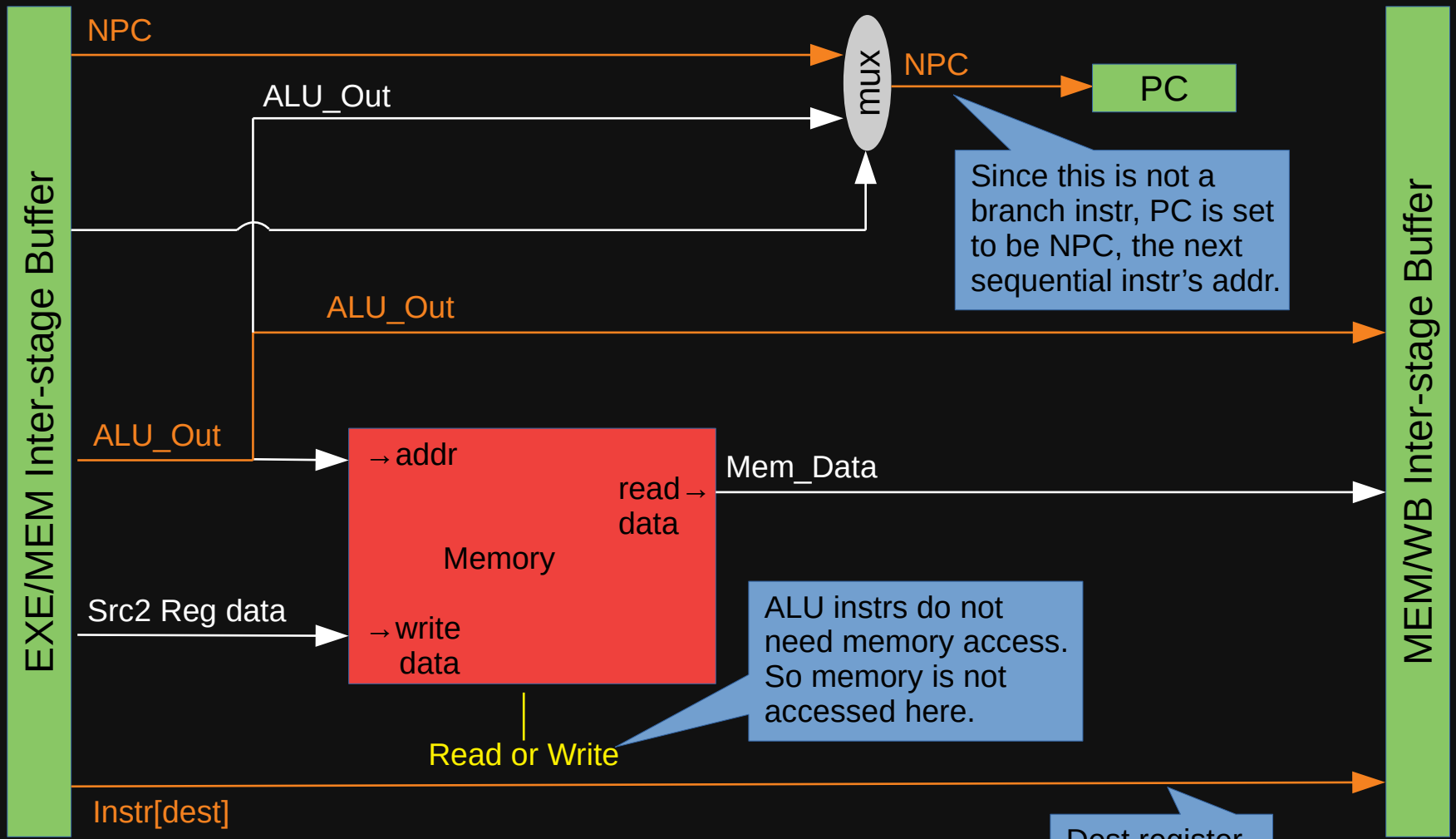
ALU Instruction Data Path with 2nd Example: ID Stage



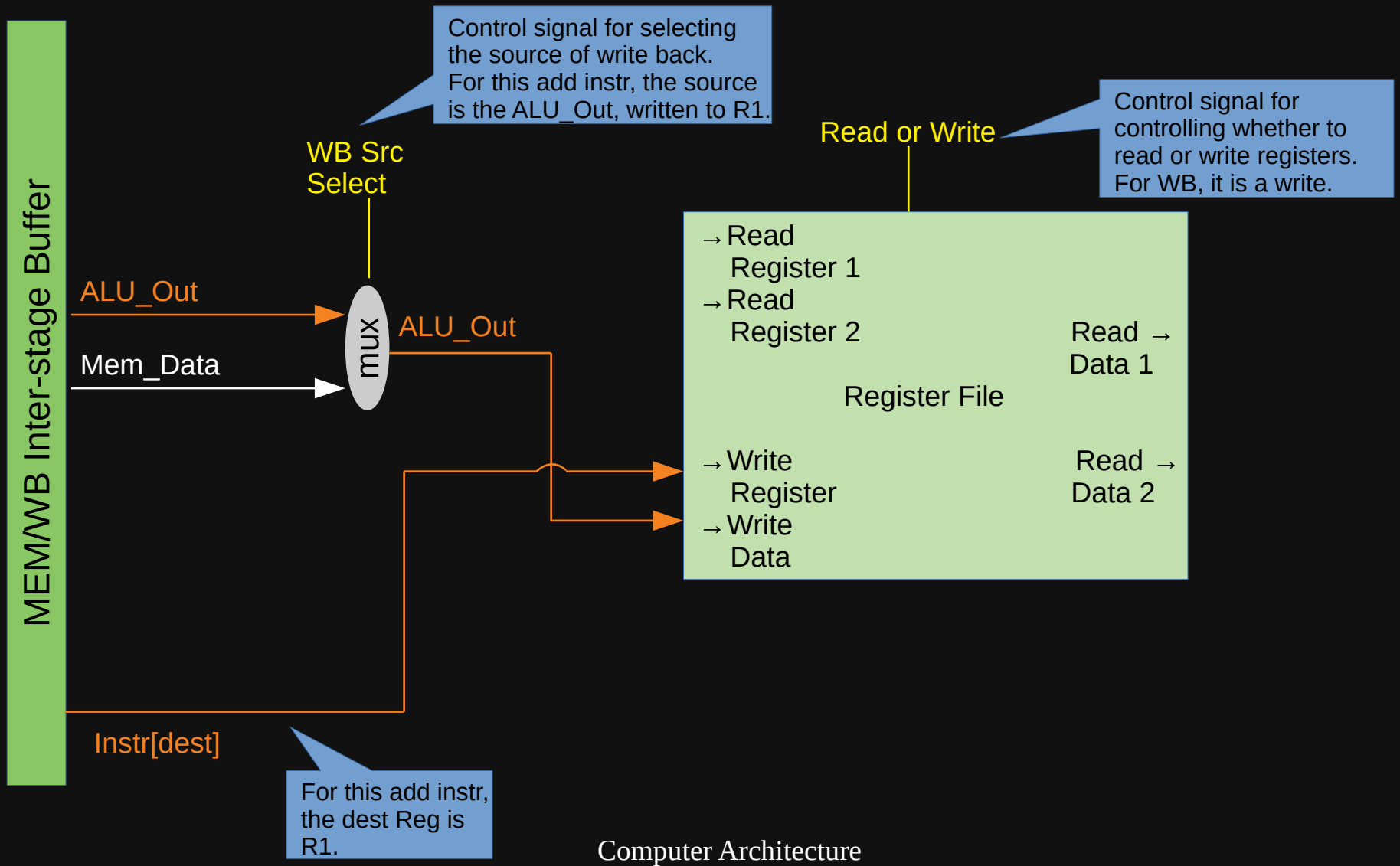
ALU Instruction Data Path with 2nd Example: EXE Stage



ALU Instruction Data Path with 2nd Example: MEM Stage



ALU Instruction Data Path with 2nd Example: WB Stage

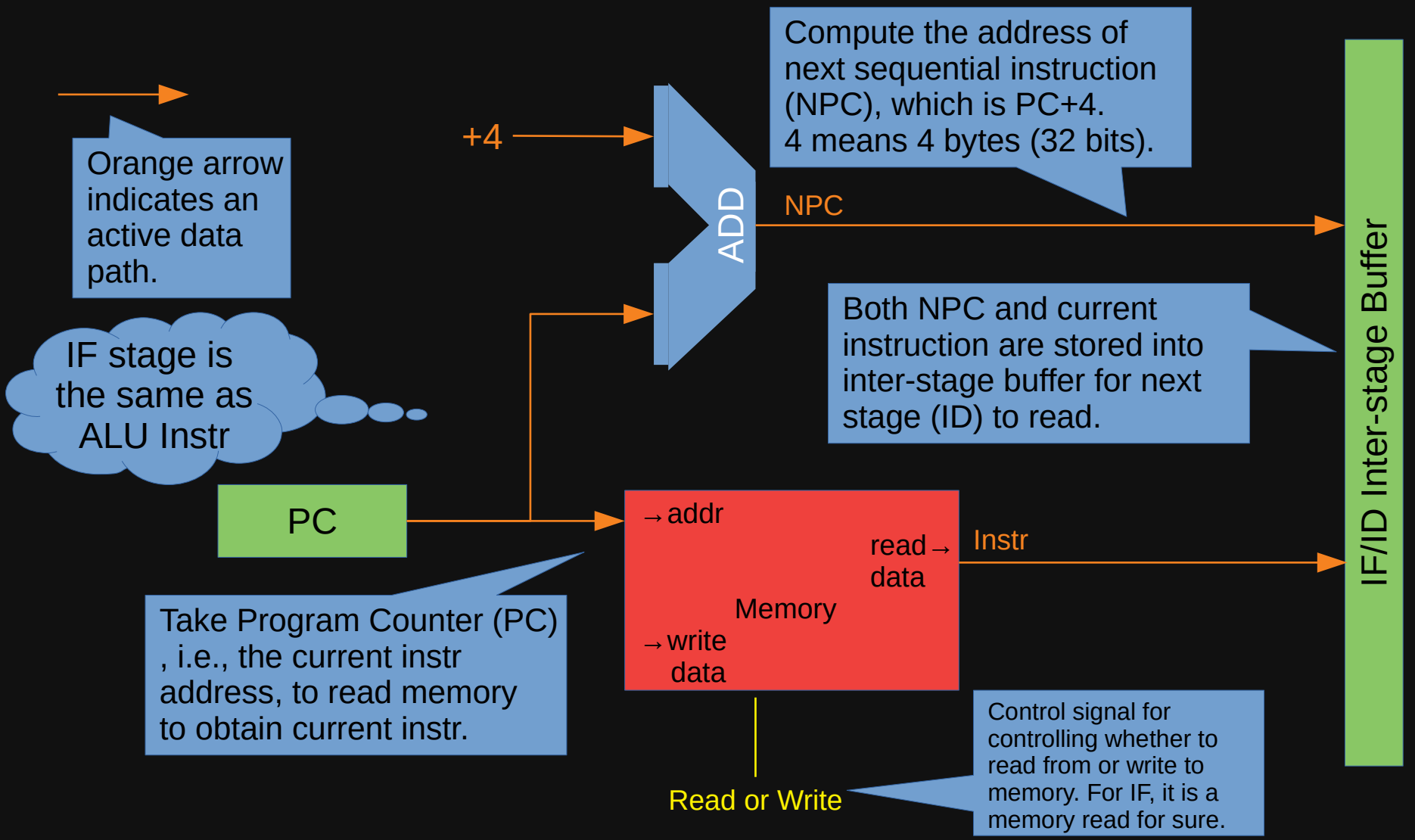


Data Path for Memory Instructions

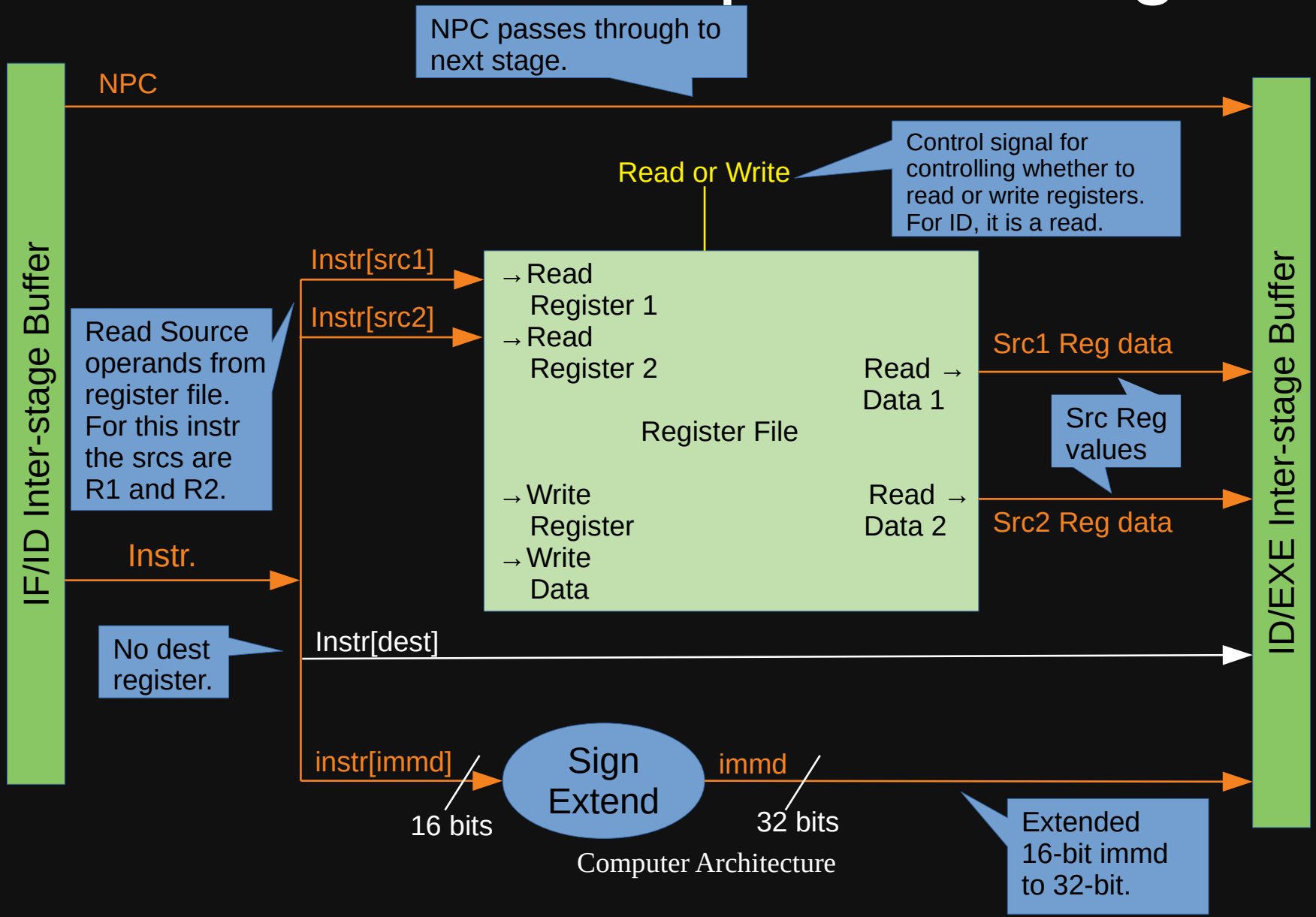
Memory Instruction Data Path with An Example

- Consider the following memory store instruction as an example:
  - Instruction: `mov [R1+100], R2`
  - Operation (memory write): `*(R1+100) = R2`
  - Source registers: `R1` (src1) and `R2` (src2)
  - Destination register: none
  - Immediate value: `100`

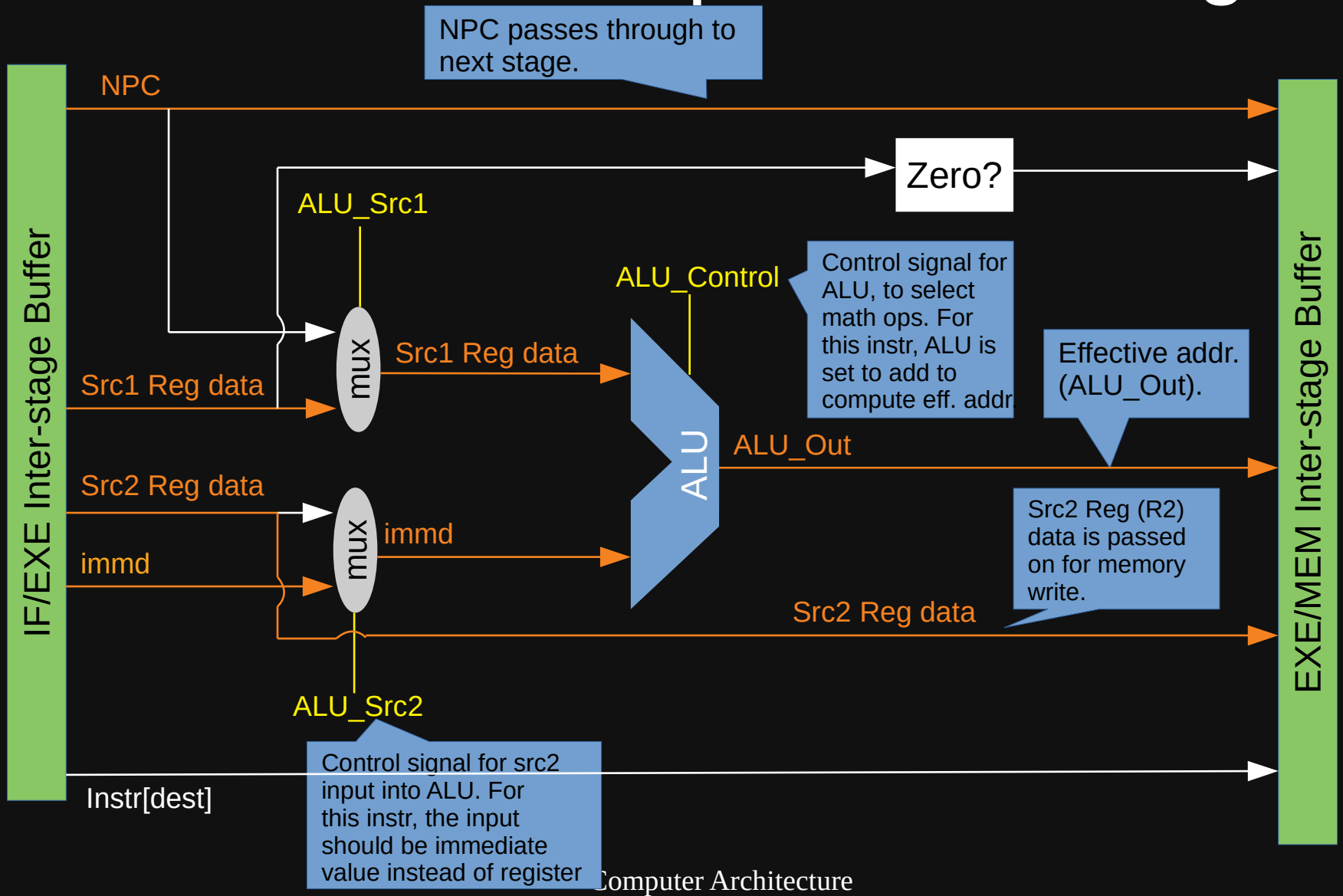
Memory Instruction Data Path with An Example: IF Stage



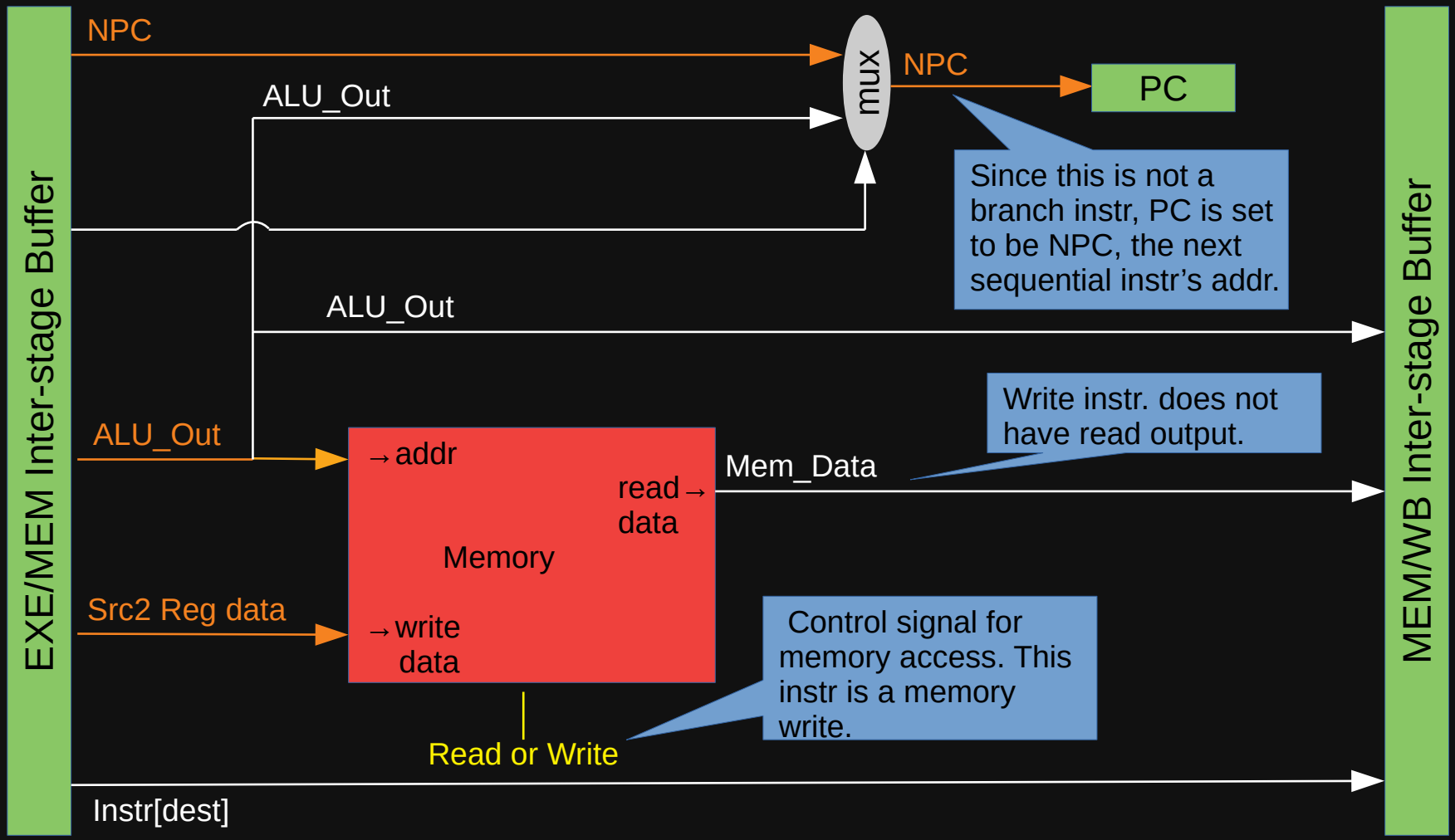
Memory Instruction Data Path with An Example: ID Stage



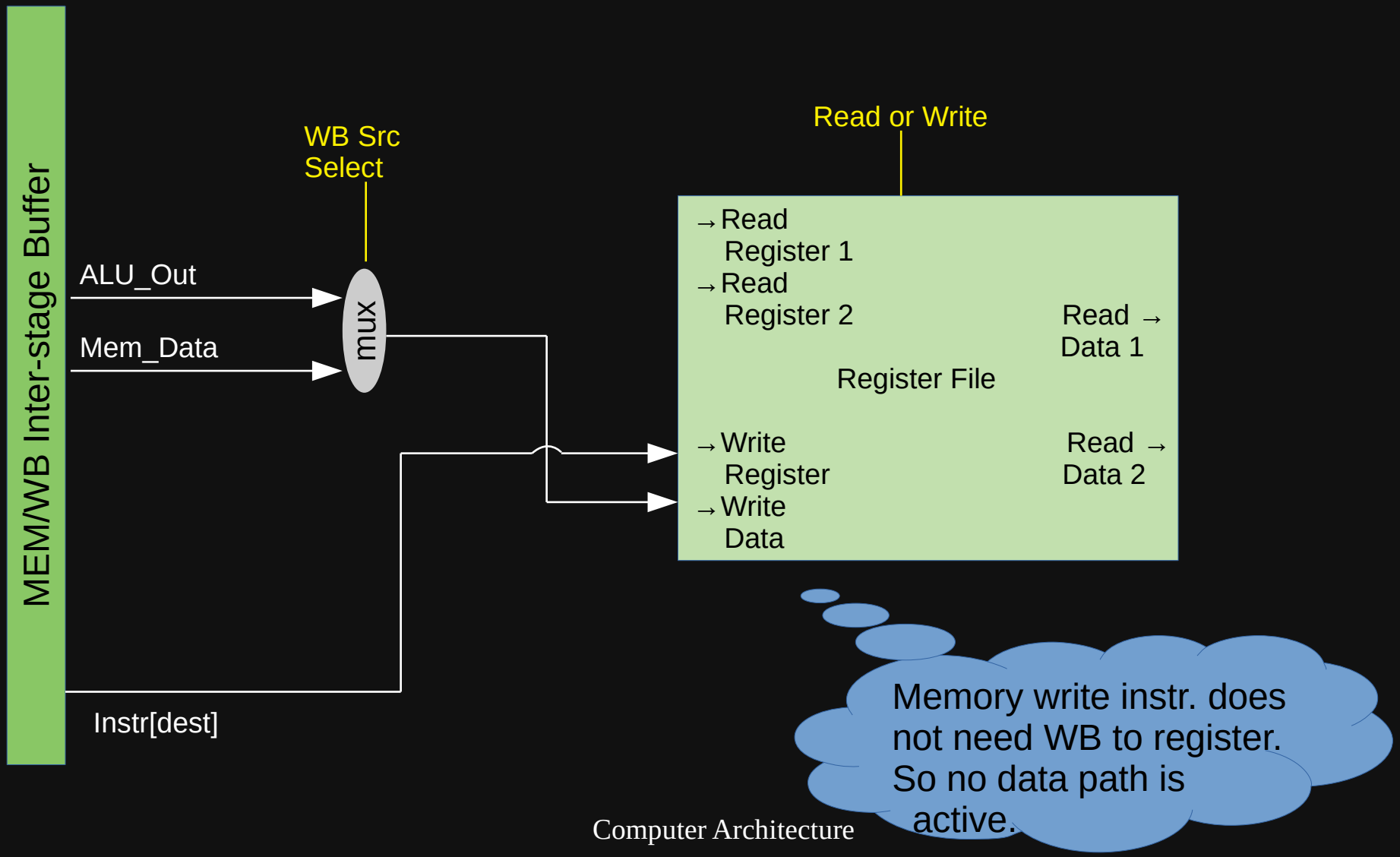
Memory Instruction Data Path with An Example: EXE Stage



Memory Instruction Data Path with An Example: MEM Stage



Memory Instruction Data Path with An Example: WB Stage



Data Path for Branch Instructions

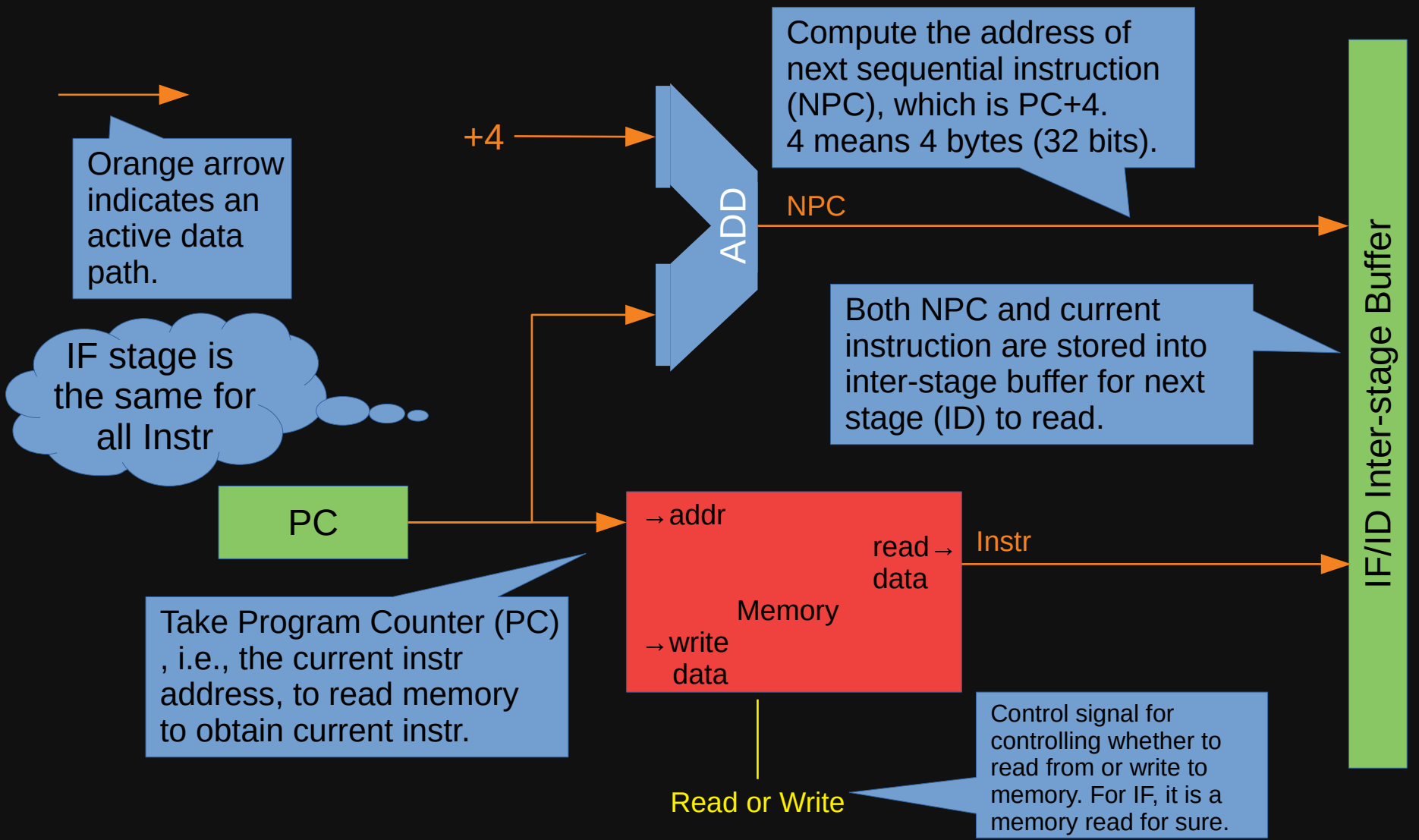
Branch Instruction Data Path with An Example

- Consider the following instruction as an example:
  - Instruction: `jnz 96`
  - Operation (if the condition is not zero):

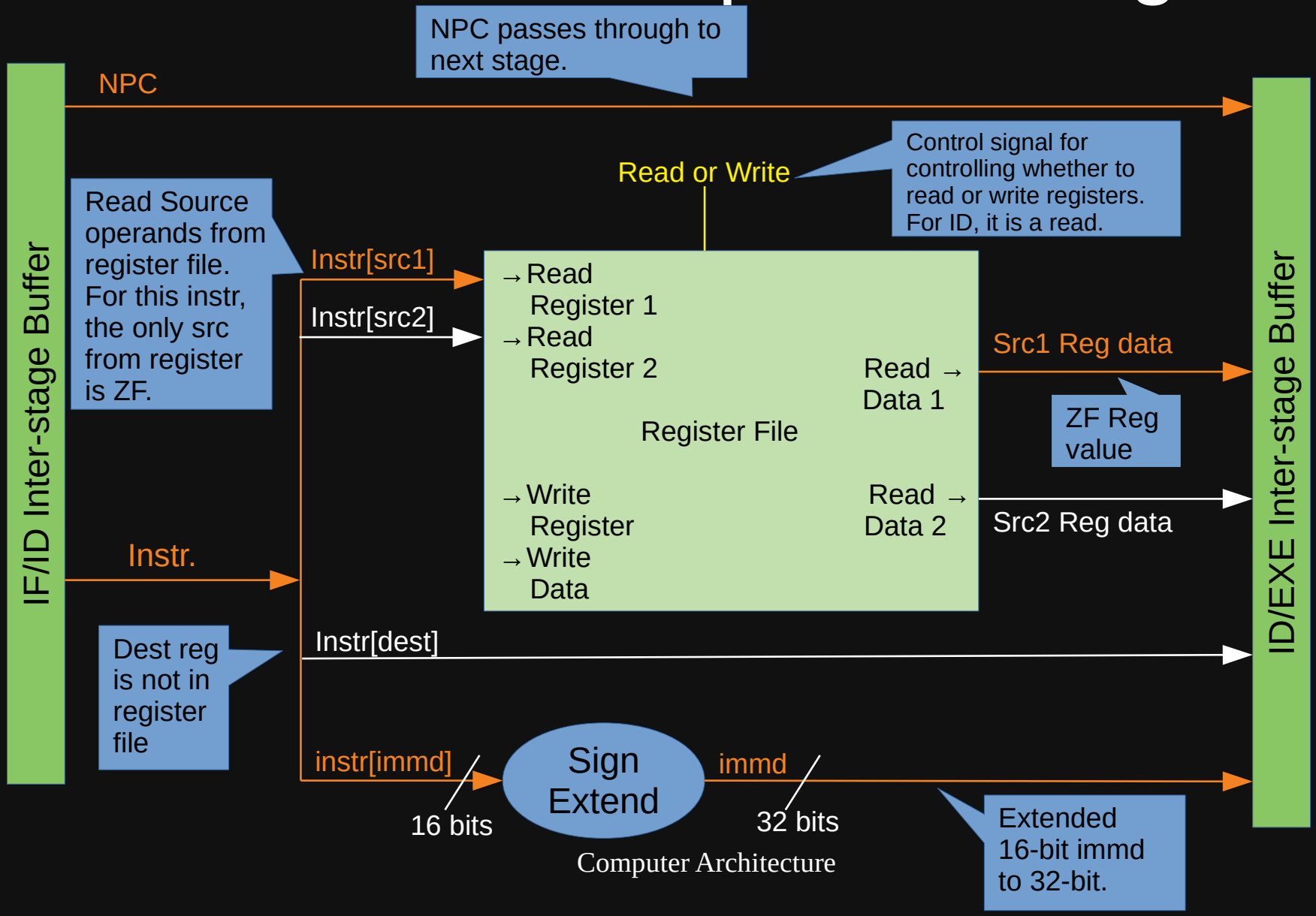
```
if(not zero)
   PC = PC + 4 + 96 //go to instr 100B away
else
   PC = PC + 4 // go to next sequential instr
```

  - Source registers: `ZF` (src1, zero flag register) and `PC` (src2, not from register file)
  - Destination register: `PC`
  - Immediate value: `96`
  - Note this branch instr is a relative jump, with the target address is +96 bytes relative to the next sequential instruction.

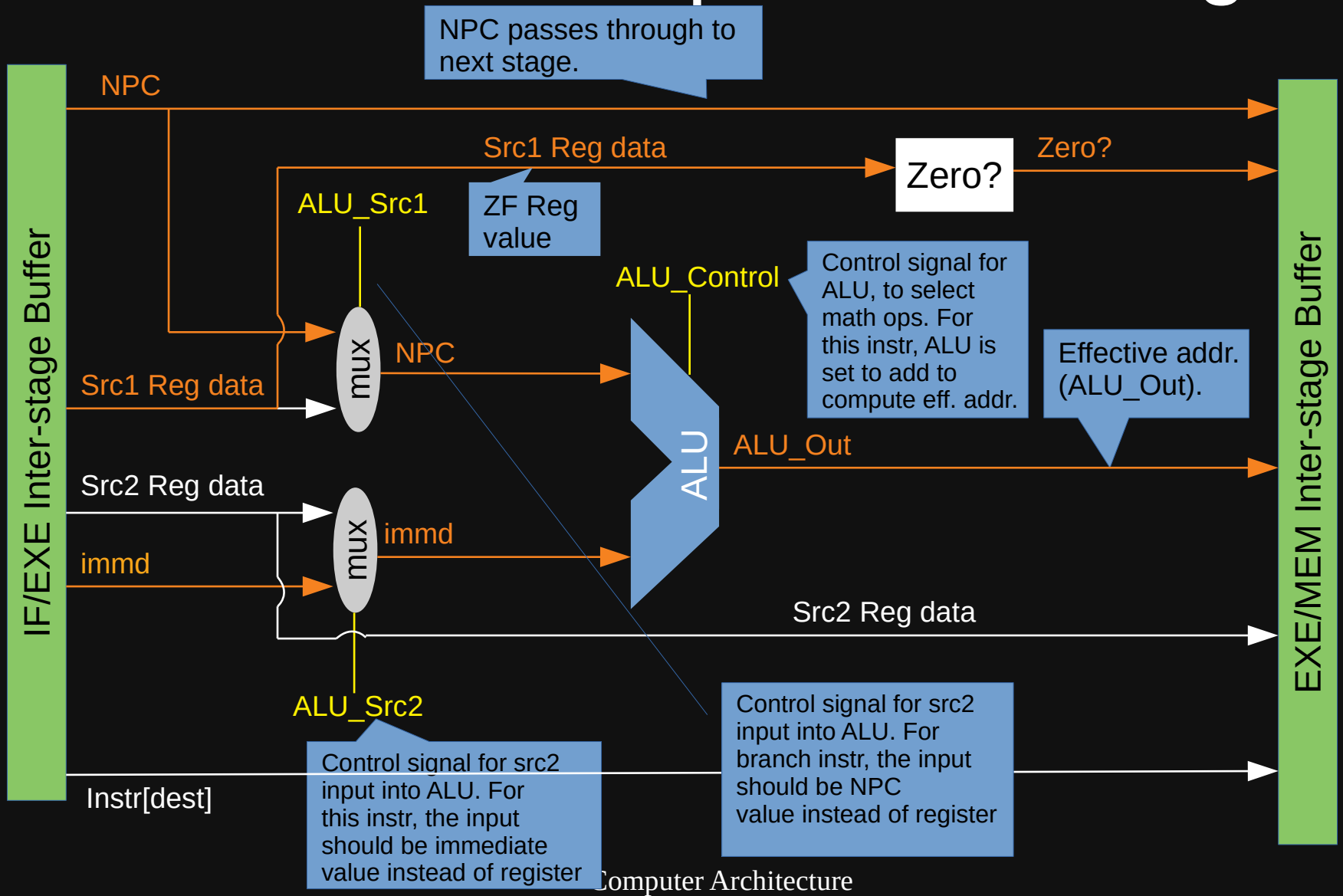
Branch Instruction Data Path with An Example: IF Stage



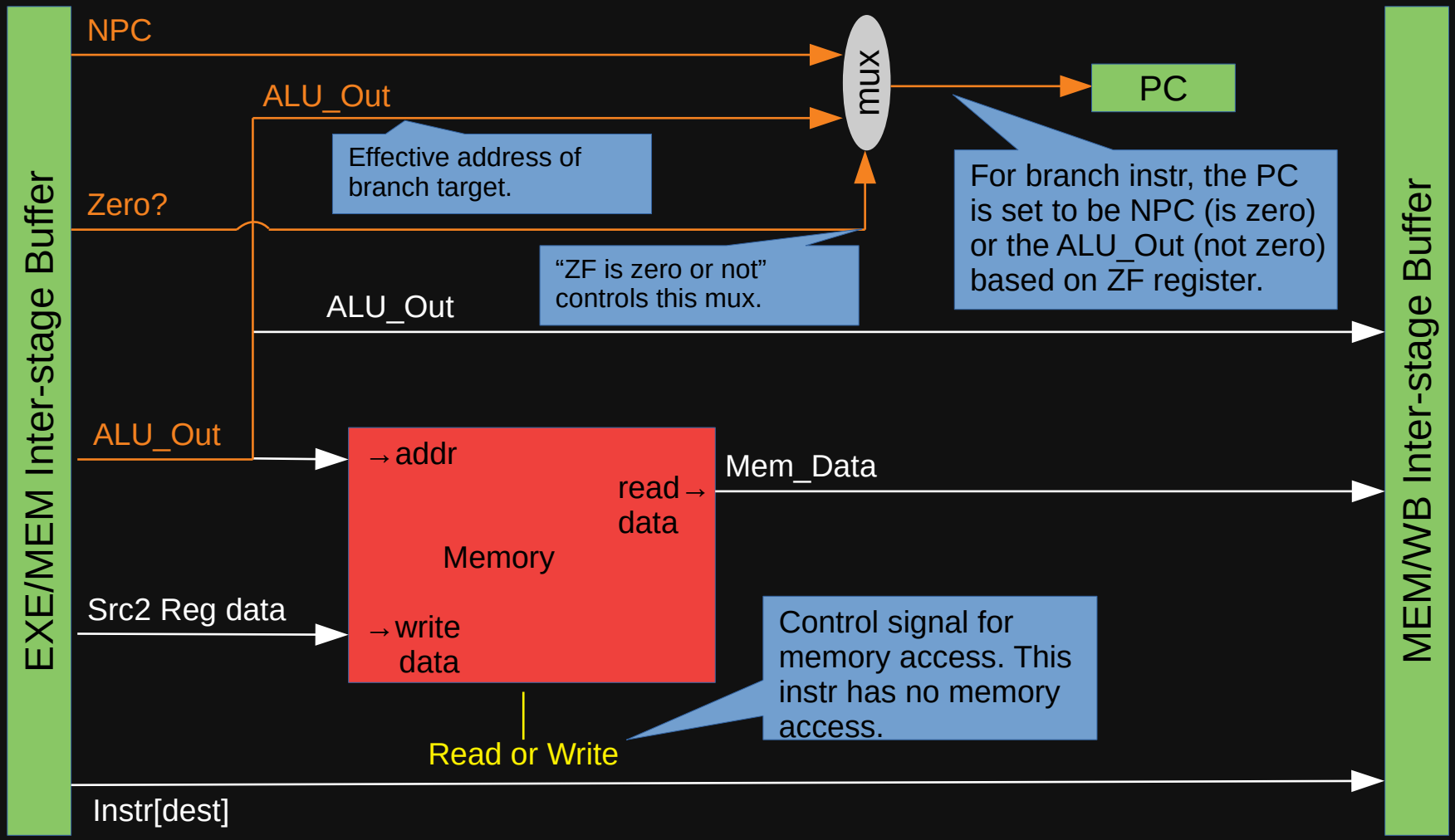
Branch Instruction Data Path with An Example: ID Stage



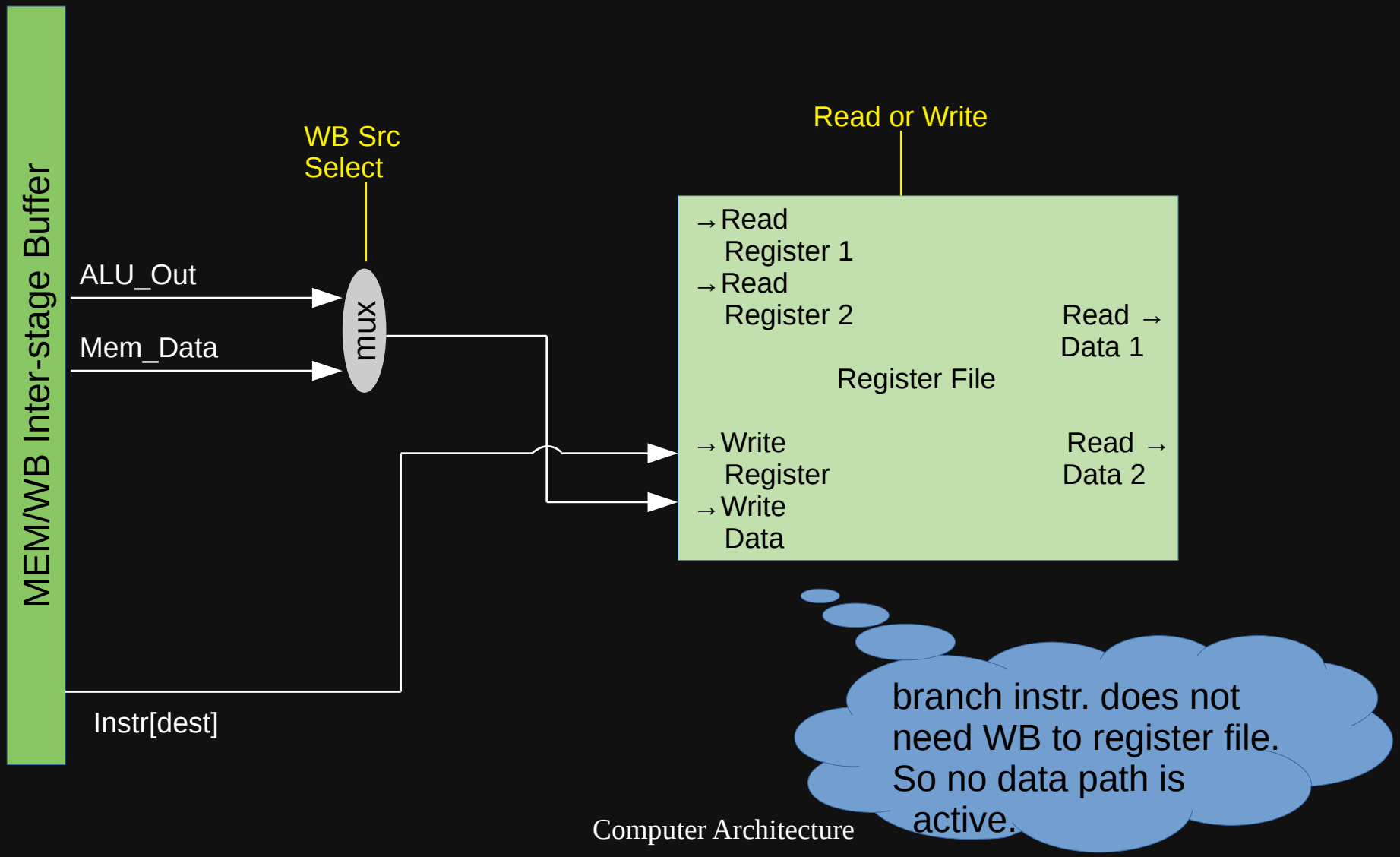
Branch Instruction Data Path with An Example: EXE Stage



Branch Instruction Data Path with An Example: MEM Stage

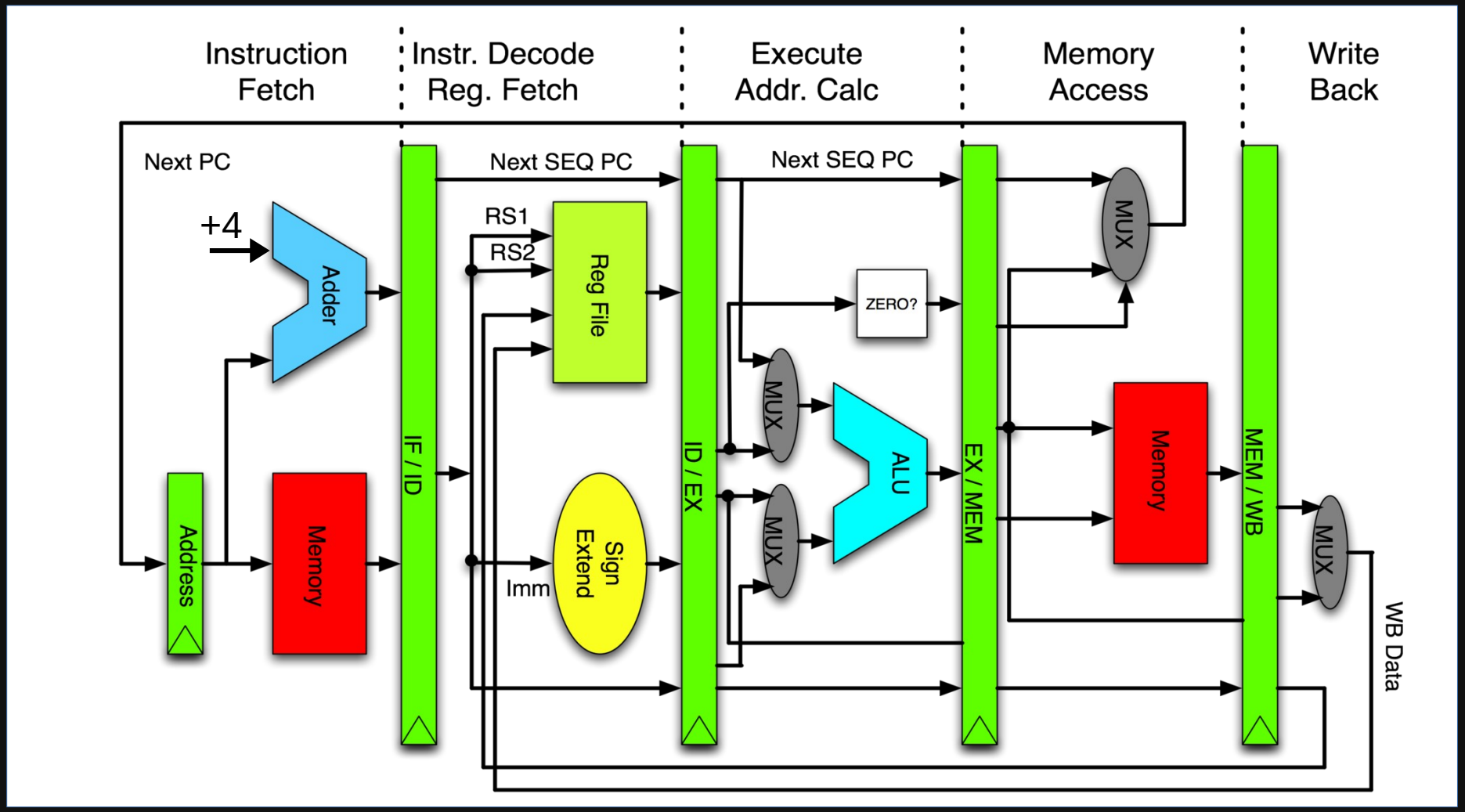


Branch Instruction Data Path with An Example: WB Stage



Putting All Stages in One Figure

A Simple Implementation of the 5-stage RISC CPU



* figure by Hellisp from Wikibooks.org

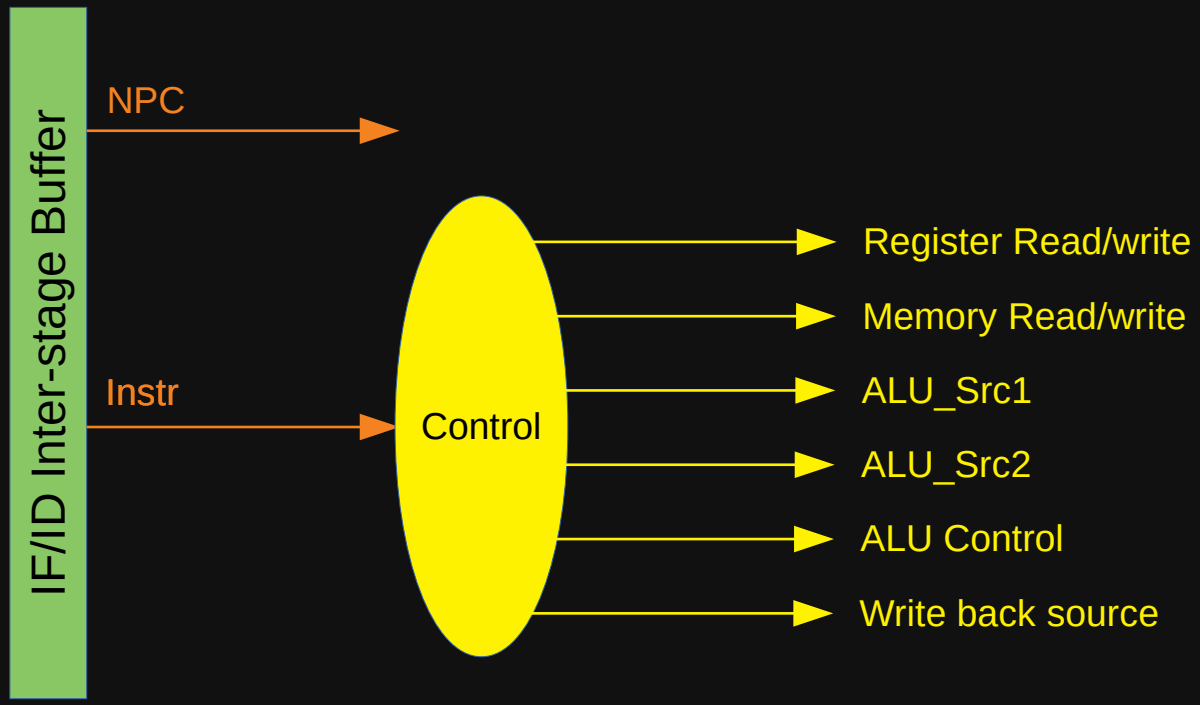
Control Signals and Multi-cycle Implementation

Control Signals

- There are many control signals in the data path that controls various multiplexors and the read/ write to register files and memory.
- The control signals are usually determined by the instruction.
- A central Control unit generates these control signals based on current instruction.

Control Signals

- At the ID phase, the instruction is also sent to the control unit to generate corresponding control signals.



Inter-stage Buffers

- The design we have seen takes one cycle to execute every stage.
- Therefore, an instruction typically take multiple cycles to execute.
- The design that execute an instruction in multiple cycles is called Multi-cycle Implementation.
- The inter-stage buffers/registers are typically required for multi-cycle implementation to store data between stages.
  - Typically, we need at least instruction register, ALU source registers, ALU output registers and memory load/store buffers.

Multi-cycle Implementation

- A main benefit of multi-cycle implementation is that different type of instructions takes different cycles to execute.
  - ALU instr: 4 stages => 4 cycles
  - Memory loads: 5 stages => 5 cycles
  - Memory stores: 4 stages => 4 cycles
  - Branches: 4 stages => 4 cycles
- Given a program with a mixture of all types of instructions, the average execution time for an instruction will be less than 5 cycles.
  - i.e., CPI < 5 cycles

Exceptions

Exceptions

- There are many unexpected events that can happen with in an processors. E.g.,
  - Undefined instruction (wrong opcode or instr encoding)
  - Divide by zero
  - Arithmetic overflow
  - External interrupts (mostly I/O requests and software system calls).
- Basically, an exception is an unscheduled event that disrupts program execution.
  - An interrupt is an exception that comes from outside of the processor.

Exceptions and Control Unit

- Control unit is also responsible for handling exceptions and interrupts.
- When there is an exception, control is required to store current processor states (i.e., save context), and transfer to the exception handling mechanism.
  - Sometimes, exception handling also involves waking up the OS and notify the OS of the exception.

Micro-programming

Micro-programming

- Modern processors are too complex to use the hardwired design we have seen in previous slides.
  - The last hardwired Intel processor was Pentium 4.
- Therefore, the main components in processor (especially the control) are actually implemented as a programmable micro-controllers.
  - Essentially all kinds of logic gates that can be connected in various ways based on user needs.
- Computer architects write high-level code to program these micro-controllers to dictate how processor operates.
  - Programmable controllers also allows fixing processor bugs after processors are released.