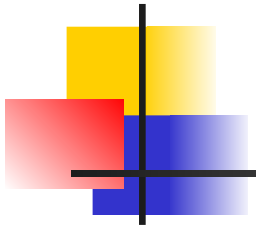
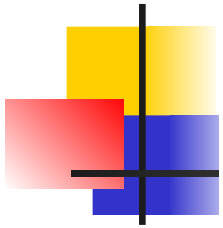


Multi cycle Processor Implementation

Single-Cycle Design: disadvantage

- Assuming fixed-period clock every instruction data-path uses one clock cycle implies:
  - CPI = 1
  - **cycle time determined by length of the longest instruction path (load)**
    - Critical path: load instruction
    - Instruction memory → register file → ALU → data memory → register file
    - **but several instructions could run in a shorter clock cycle: *waste of time***
  - **resources used more than once in the same cycle need to be duplicated**
    - ***waste of hardware*** *and chip area*

Performance of Single-Cycle Machines



Assume that the operation times for the major functional units in this implementation are the following:

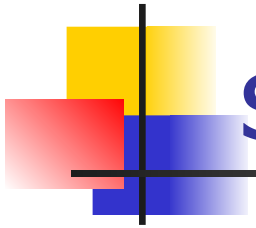
- Memory units: 200 picoseconds (ps)
- ALU and adders: 100 ps
- Register file (read or write): 50 ps

Assuming that the multiplexors, control unit, PC accesses, sign extension unit, and wires have no delay, which of the following implementations would be faster and by how much?

1. An implementation in which every instruction operates in 1 clock cycle of a fixed length.
2. An implementation where every instruction executes in 1 clock cycle using a variable-length clock, which for each instruction is only as long as it needs to be. (Such an approach is not terribly practical, but it will allow us to see what is being sacrificed when all the instructions must execute in a single clock of the same length.)

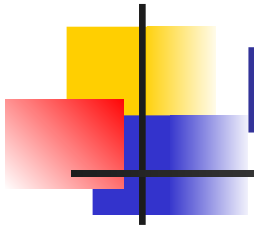
To compare the performance, assume the following instruction mix: 25% loads, 10% stores, 45% ALU instructions, 15% branches, and 5% jumps.

Fixing the problem with single-cycle designs



- One solution: a variable-period clock with different cycle times for each instruction class
  - ***unfeasible***, as implementing a variable-speed clock is technically difficult
- Another solution:
  - Use a clock with a smaller cycle time…
  - …have different instructions take different numbers of cycles

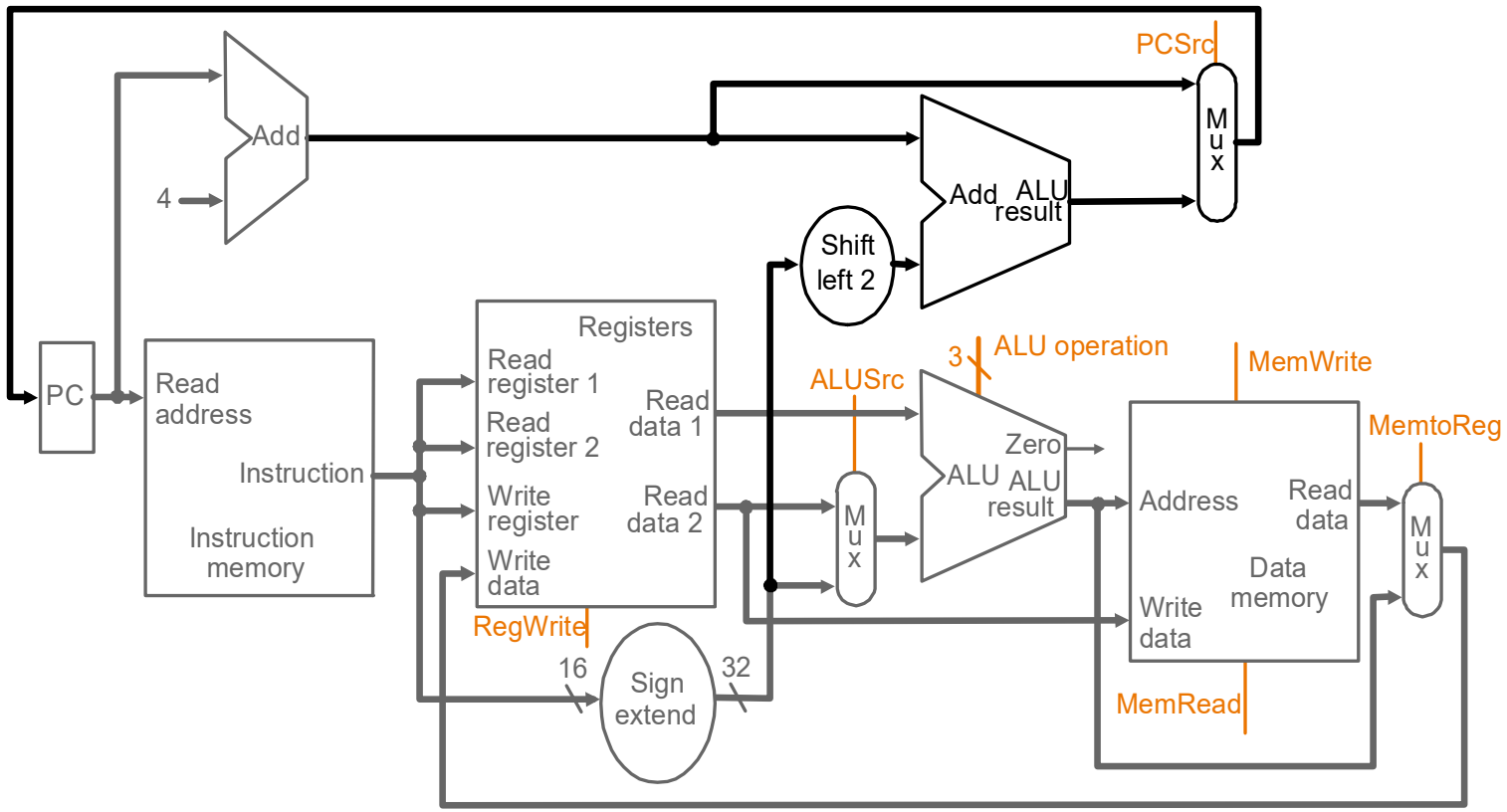
    by breaking instructions into steps and fitting each step into one cycle
  - *feasible:* ***multicyle approach***!



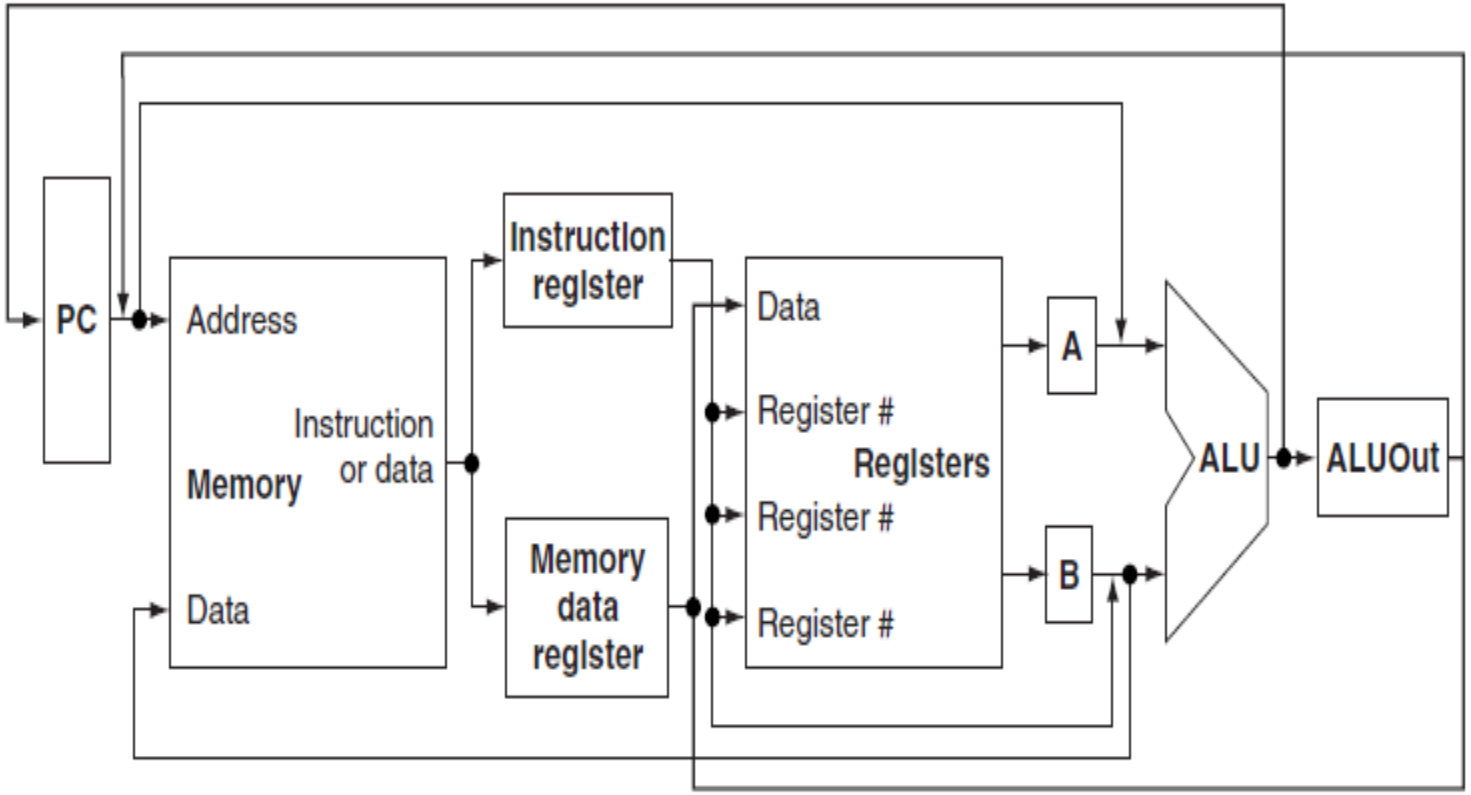
Multi cycle implementation

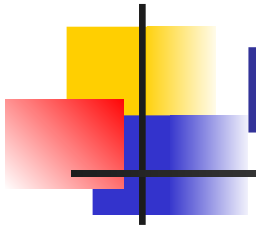
- **An implementation in which an instruction is executed in multiple clock cycle**
- Break up the instructions into *steps*
  - each step takes one clock cycle
  - It allows a functional unit to be used more than once per instruction, since it is used on different clock
  - This Sharing can reduce amount of hardware required

Single-cycle data path



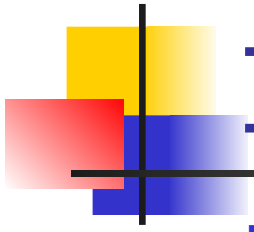
Multi-cycle : abstract implementation





Multicycle Vs single cycle

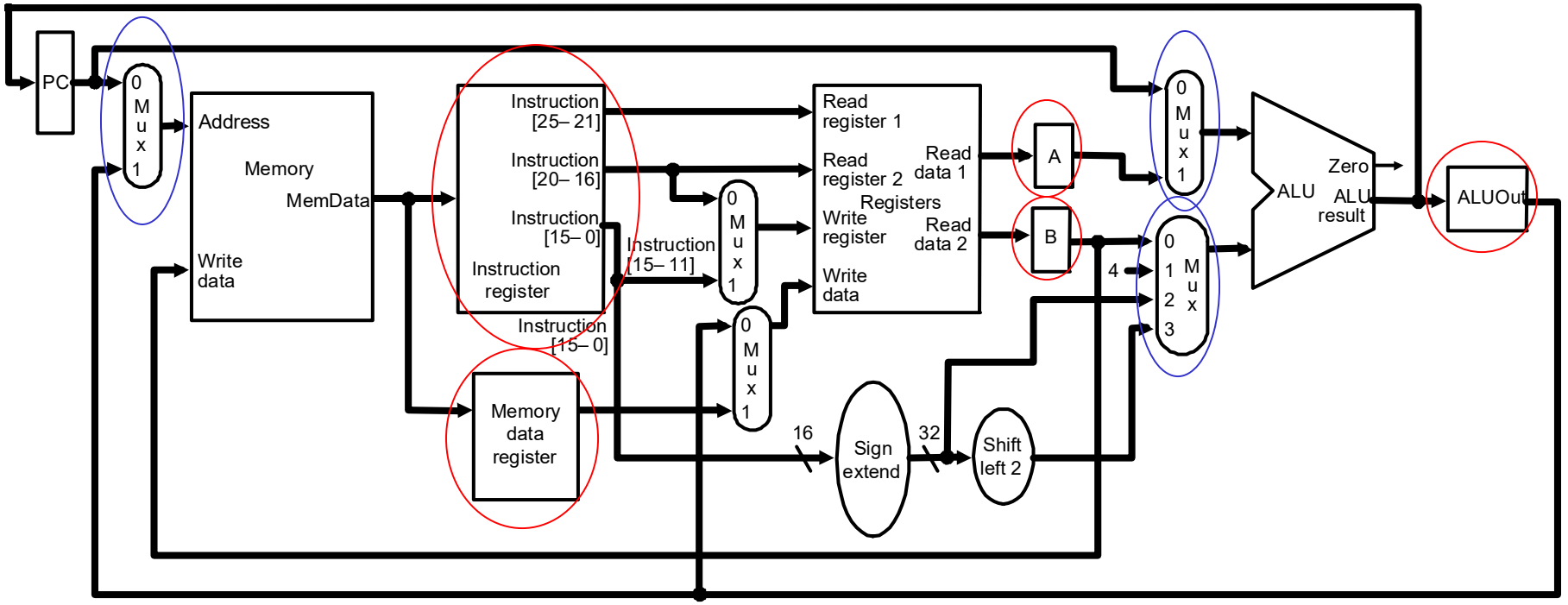
- Note the particularities of multicycle & single-cycle diagrams
  - single memory for data & instructions
  - single ALU, no extra adders
  - extra registers to hold data between clock cycles
- Data used by same instruction in a later cycle must be stored into additional registers



Internal registers

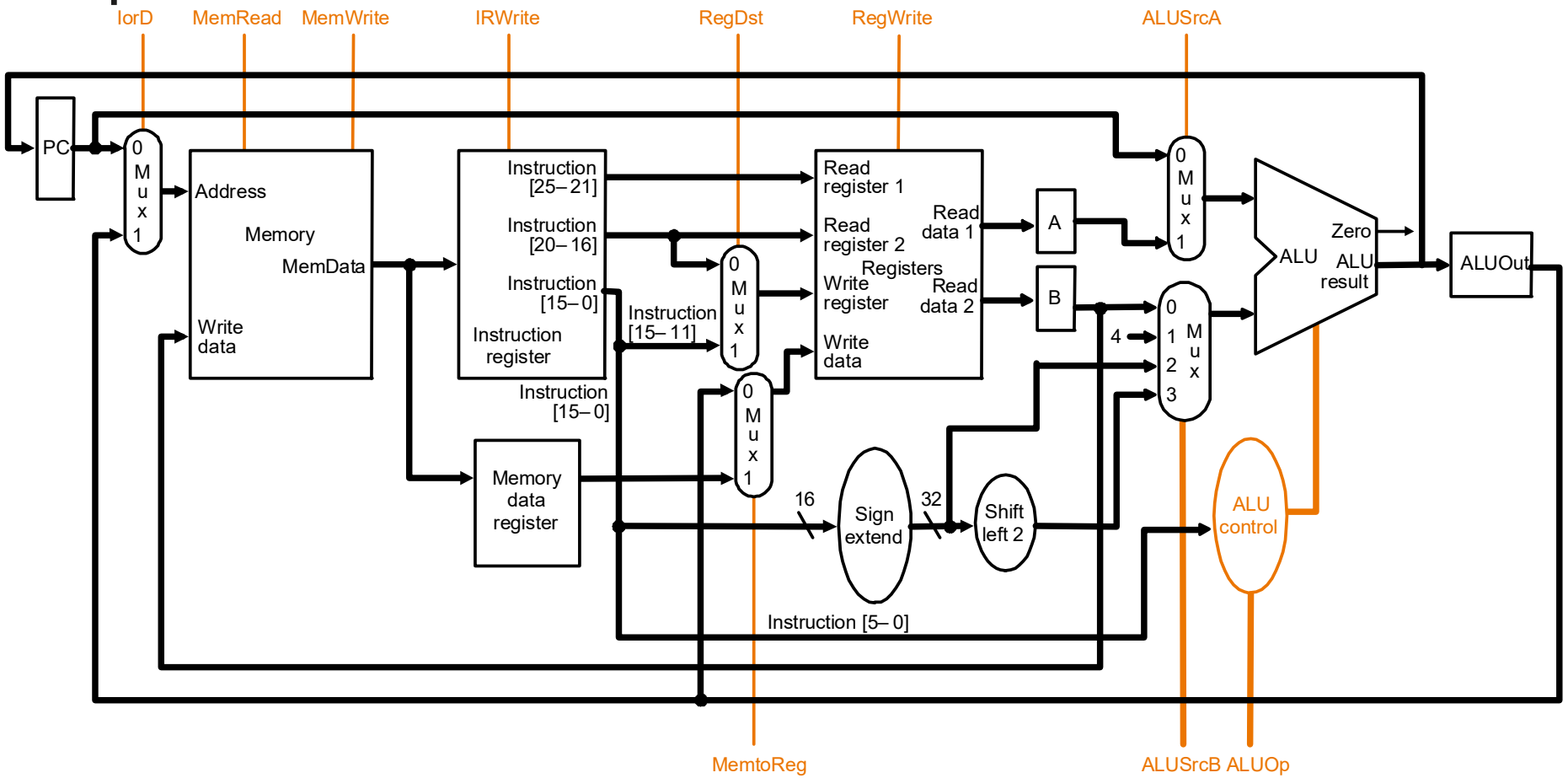
- Between steps/cycles
  - At the end of one cycle, store the data that are required to be used in later cycles of the same instruction
    - need to **introduce additional internal (programmer-invisible) registers** for this purpose
- Temporary registers are
  - Instruction Registers (IR), Memory Data Register (MDR) to save output of memory
  - Registers A, B to hold register operands
  - ALUout register holds the output of ALU

Multicycle Datapath



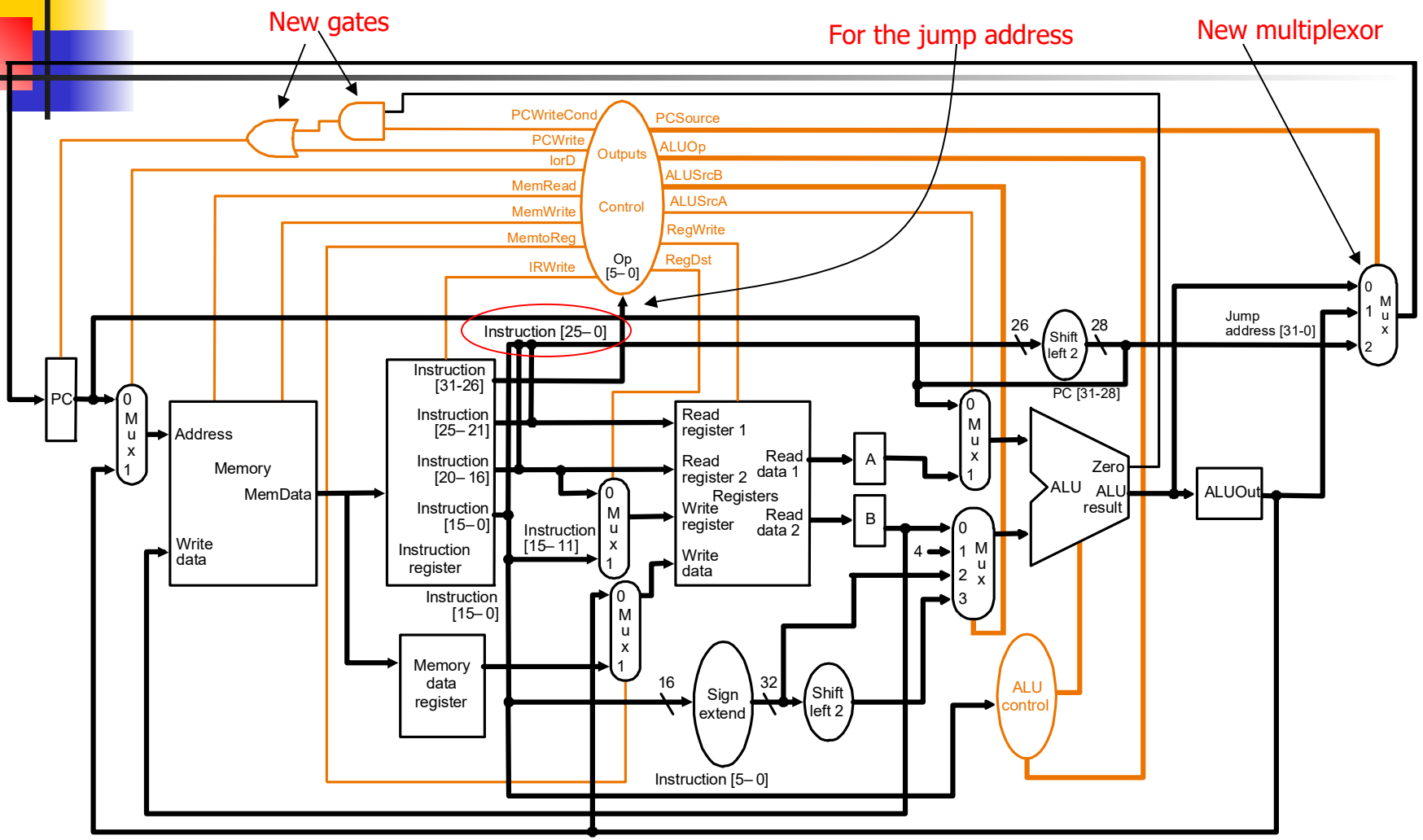
Basic multicycle MIPS datapath handles R-type instructions and load/stores: new internal register in red ovals, new multiplexors in blue ovals

Multicycle Datapath with Control

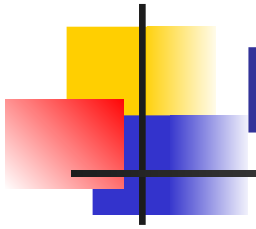


… with control lines and the ALU control block added – *not all* control lines are shown

Multicycle Datapath with Control



Complete multicycle MIPS datapath (with branch and jump capability) and showing the main control block and all control lines



Multi-cycle design:Advantages

- The ability to allow instructions to take different numbers of clock cycles
- The ability to share functional units with in the execution of a single instruction