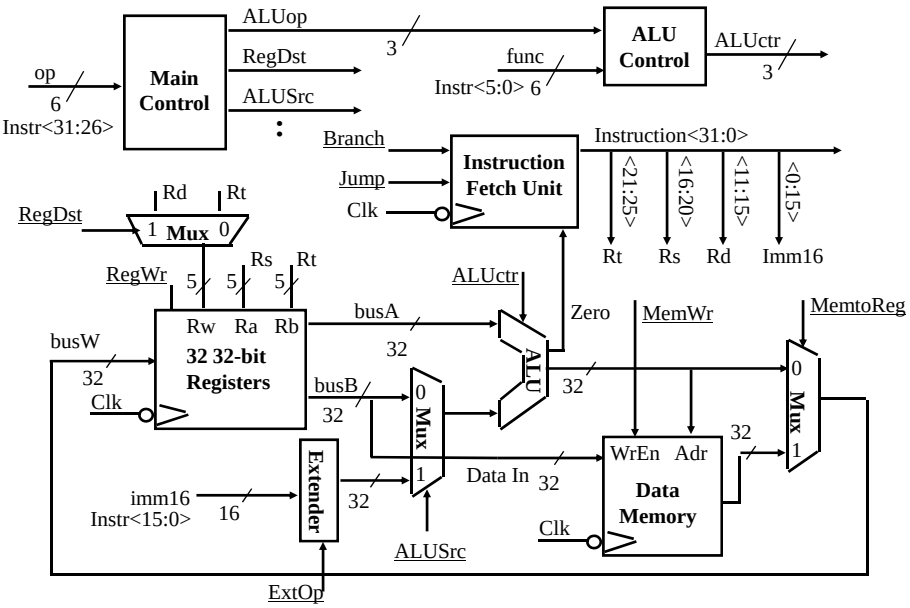
CS420/520
Computer Architecture I

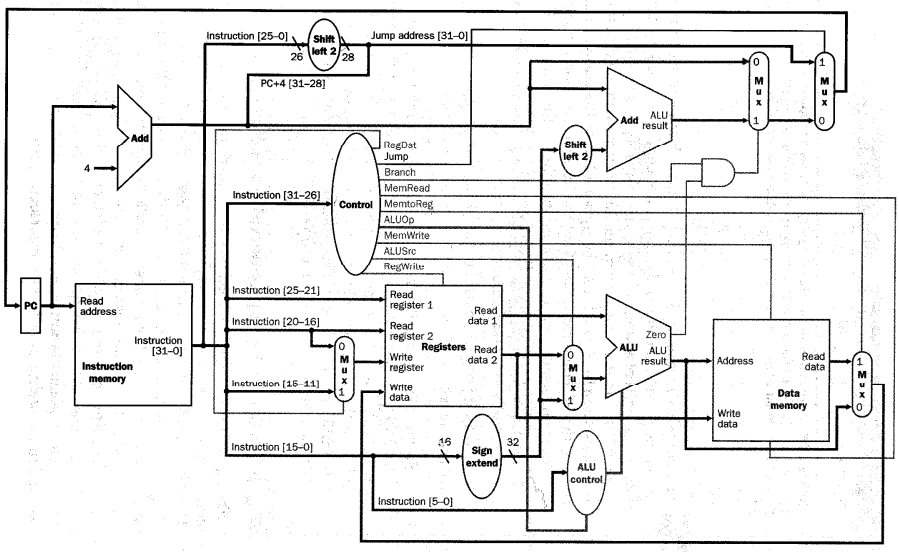
Designing a Multiple Cycle Processor

Dr. Xiaobo Zhou
Department of Computer Science

Review: A Single Cycle Processor



Review: Putting it All Together: An Extended View



Drawbacks of this Single Cycle Processor

- Long cycle time:
  - Cycle time must be long enough for the load instruction:
    - Instruction Memory Access Time +
    - Register File Access (Reading) Time +
    - ALU Delay (address calculation) +
    - Data Memory Access Time +
    - Register File Setup & Writing Time
- Cycle time is much longer than needed for all other instructions. Examples:
  - R-type instructions do not require data memory access
  - Jump does not require ALU operation nor data memory access

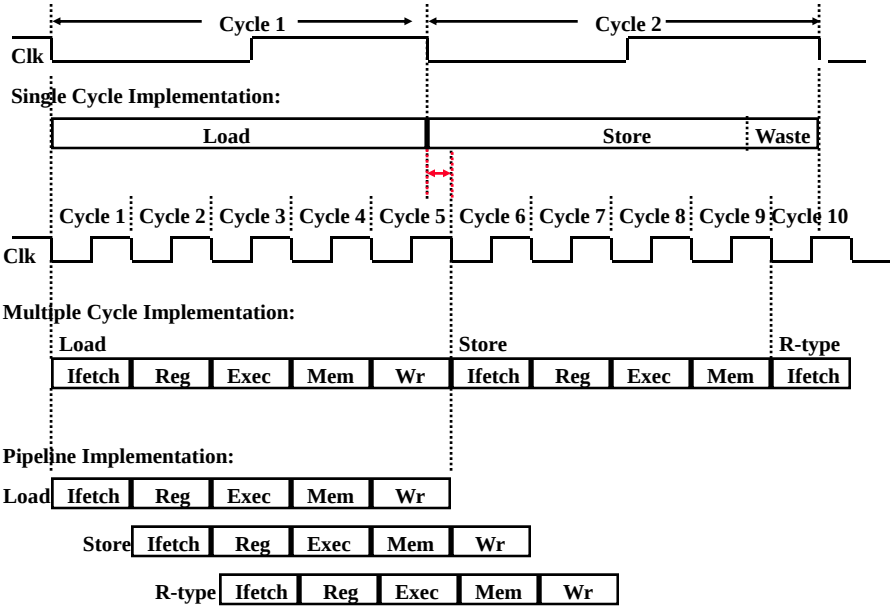
Performance Comparisons

- Assume that the operation times for the major functional units in a single cycle implementation are the following
  - Memory units: 2 ns
  - ALU and Adders: 2ns
  - Register file (read or write): 1 ns
  - The multiplexors, control unit, PC accesses, sign extension unit, and wires have no delay.
- Questions: which of the following implementations would be faster, and by how much?
  - A fixed-single-cycle implementation
  - A *flexible*-single-cycle implementation: using a variable-length clock, which for each instruction is only as long as it needs to be
- To compare, assume the following instruction mix:
  - 24% loads, 12% stores, 44% ALUs, 20% branches

Overview of a Multiple Cycle Implementation

- The root of the single cycle processor's problems:
  - The cycle time has to be long enough for the slowest instruction
- Solution:
  - Break the instruction into smaller steps
  - Execute each step (instead of the entire instruction) in one cycle
    - Cycle time: time it takes to execute the longest step
    - Keep all the steps to have similar length
  - This is the essence of the multiple cycle processor
- The advantages of the multiple cycle processor:
  - Cycle time is much shorter
  - Different instructions take different number of cycles to complete
    - Load takes five cycles
    - Reg-type only takes four cycles; Jump 3; Branch 3 or 4
  - Allows a functional unit to be used more than once per instruction
    - Adder + ALU
    - Instruction mem + Data mem

Single Cycle, Multiple Cycle, vs. Pipeline



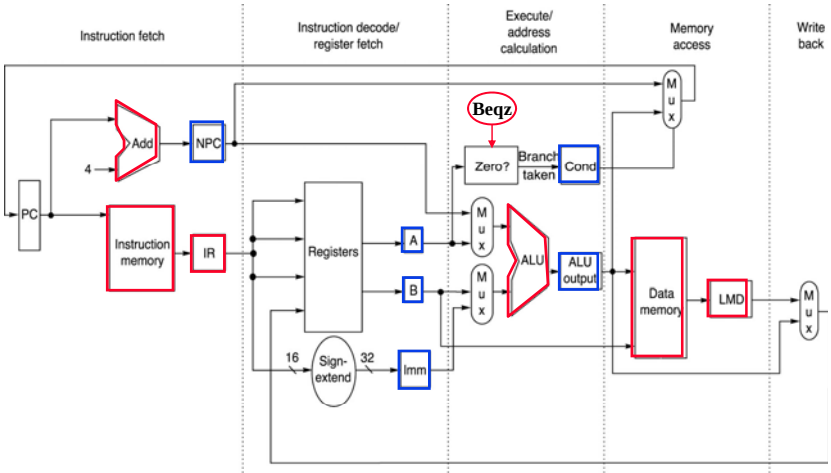
Performance Comparisons (Cont.)

- Assume that the operation times for the major functional units in a single cycle implementation are the following
  - Memory units: 2 ns
  - ALU and Adders: 2ns
  - Register file (read or write): 1 ns
  - The multiplexors, control unit, PC accesses, sign extension unit, and wires have no delay.
- Questions: which of the following implementations would be faster, and by how much?
  - A fixed-single-cycle implementation
  - A *flexible*-single-cycle implementation: using a variable-length clock, which for each instruction is only as long as it needs to be
  - A multiple-cycle implementation
- To compare, assume the following instruction mix:
  - 24% loads, 12% stores, 44% ALUs, 20% branches

Instruction Executions on a Multi-Cycle Datapath (BEQZ)

- In theory, you need a register to hold a signal value if:
  - (1) The signal is computed in one clock cycle and used in another.
- Instruction Fetch Cycle (IF)
  - IR <- Mem[PC]; NPC <- PC + 4
- Instruction Decode / Register Fetch (ID/RF)
  - A <- Reg[rs]; B<- Reg[rt]; Imm <- Sign-Ext(Imme 16)
- Execution (EXEC)
  - Mem reference (Load/Store): ALUOutput <- A + Imm
  - Reg-Reg ALU: ALUOutput <- A op B; or Reg-Imme ALU: A op Imm
  - BEQZ: ALUOutput <- NPC + (Imm << 2); Cond <- (A == 0)
- Memory access / BEQZ completion (MEM)
  - Memory reference (Load): LMD <- Mem[ALUputput]
  - Memory reference (Store): Mem[ALUputput] <- B
  - BEQZ: if (Cond) PC<- ALUOutput
- Write-back cycle(WB)
  - Reg-Reg ALU: Reg[rd] <- ALUOutput; or Reg-Imm ALU: Reg[rt] <- ALUOutput
  - Load: Reg[rt] <- LMD

An Idealized Multiple Cycle Datapath



- Allows a functional unit to be used more than once per instruction
  - Adder + ALU
  - Instruction mem + Data mem

Revisit Instruction Executions (in case of BEQ)

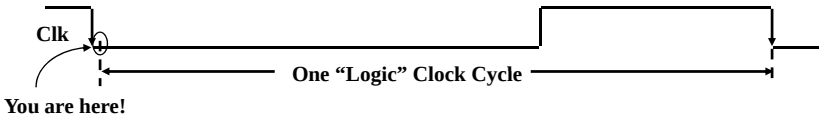
Step Name	Action for R-type instructions	Action for memory-reference instructions	Action for Branches	Action for Reg-Imme
Instruction Fetch	IR = Mem [PC], PC = PC+4			
Instruction decode/ Register fetch	A = Reg [IR[25:21]] B= Reg [IR[20:16]] Target ← ALUout = PC + (sign-extend (IR[15:0]) << 2)			
Execution, address computation, branch /jump completion	ALUout = A op B	ALUout = A + sign-extend (IR [15:0])	If (A == B) then PC = Target	ALUout = A op Imme
Memory access		Load : Data-out = Mem [ALUout] or Store : Mem [ALUout] = B		
Write-back	Reg [IR[15:11]] = ALUout	Load : Reg [IR[20:16]] = Data-out		Reg [IR[20:16]] = ALU-out

Control signals have to be set *per-Step*, instead of per instruction in single cycle datapath!

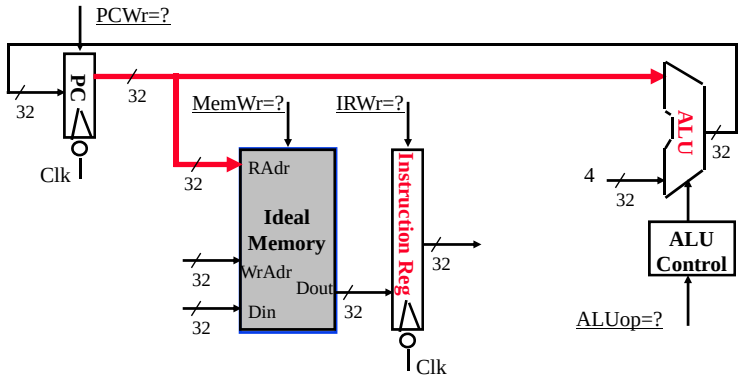
Instruction Fetch Cycle: In the Beginning

° **Every cycle begins right AFTER the clock tick:**

- **mem[PC] PC<31:0> + 4**

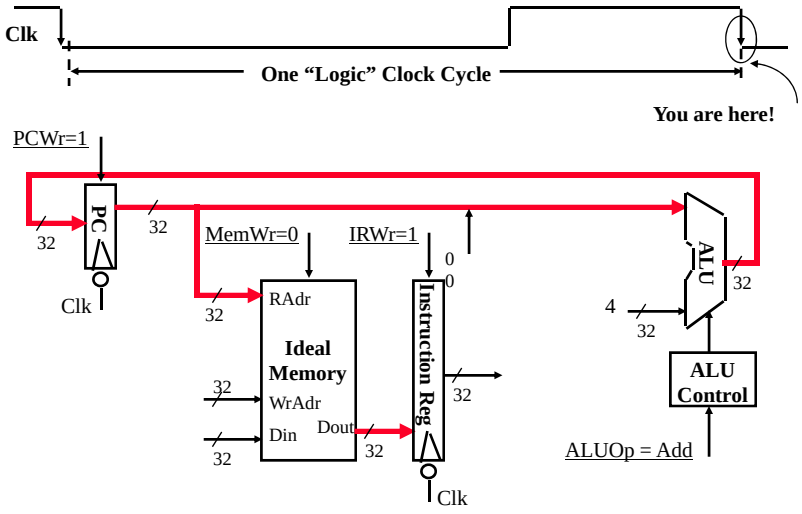


You are here!

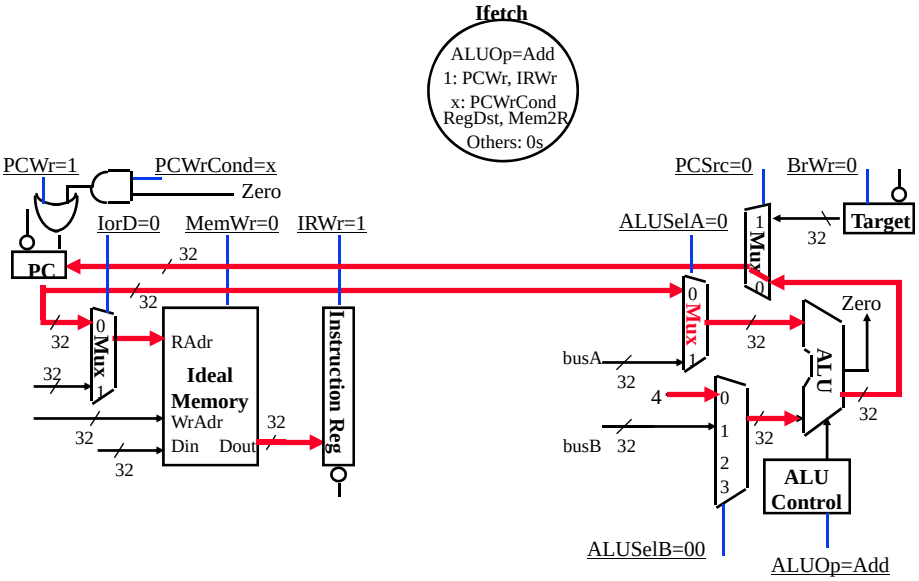


Instruction Fetch Cycle: The End

° Every cycle ends AT the next clock tick (storage element updates):
  • IR <-- mem[PC] PC<31:0> <-- PC<31:0> + 4

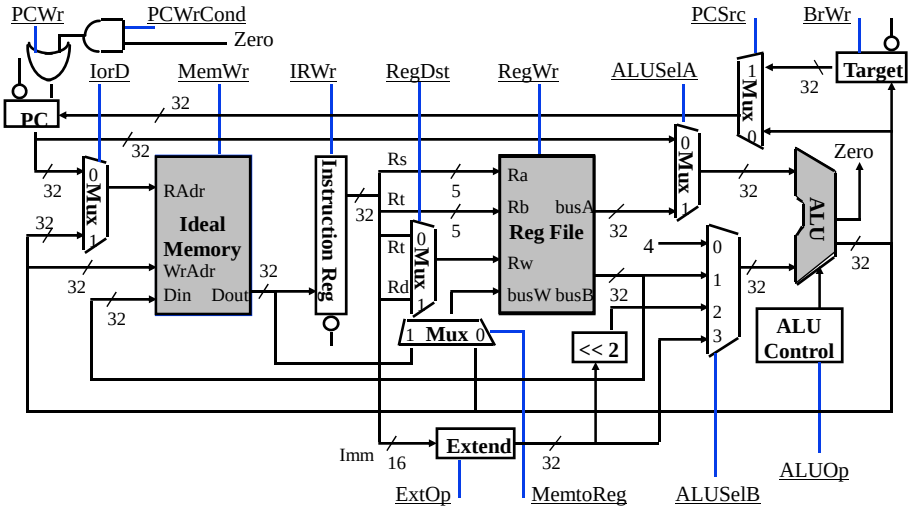


Instruction Fetch Cycle: Overall Picture



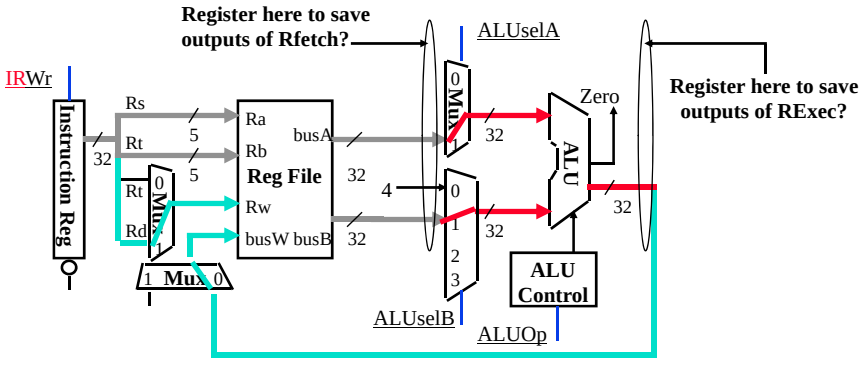
Putting it all together: Multiple Cycle Datapath

° MCP: A functional unit to be used more than once per instruction
  • Not good for pipelining



A Multiple Cycle Delay Path

° There is no register to save the results between:
  • Register Fetch: busA <- Reg[rs] ; busB <- Reg[rt]
  • R-type Execution: ALU output <- busA op busB
  • R-type Completion: Reg[rd] <- ALU output

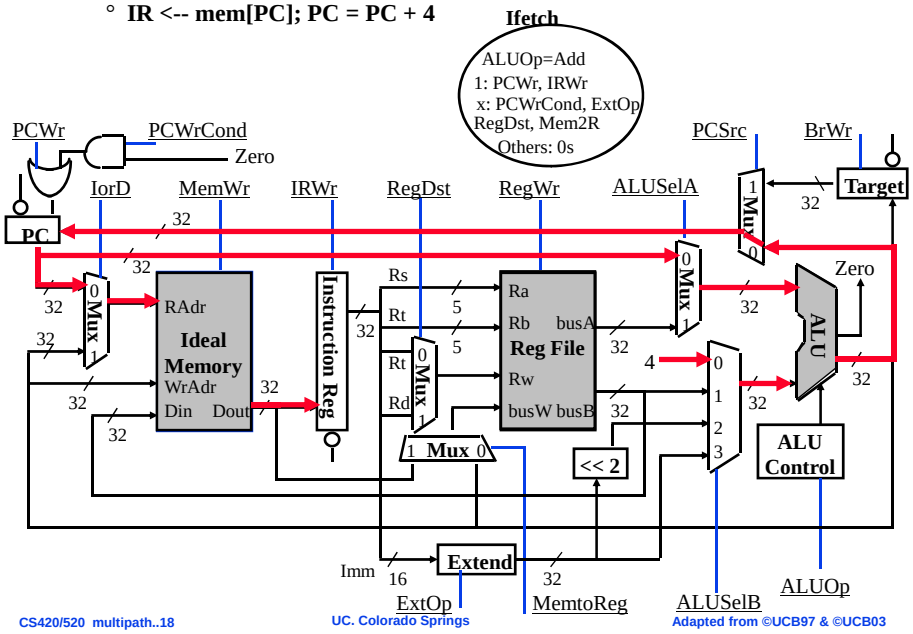


A Multiple Cycle Delay Path (Continue)

- Register is NOT needed to save the outputs of Register Fetch:
  - IRWr = 0: busA and busB will not change after Register Fetch
- Register is NOT needed to save the outputs of R-type Execution:
  - busA and busB will not change after Register Fetch
  - Control signals ALUSelA, ALUSelB, and ALUOp will not change after R-type Execution
  - Consequently ALU output will not change after R-type Execution
- In theory, you need a register to hold a signal value if:
  - (1) The signal is computed in one clock cycle and used in another.
  - (2) AND the inputs to the functional block that computes this signal can change before the signal is written into a state element.
- You can save a register if Cond 1 is true BUT Cond 2 is false:
  - But in practice, this will introduce a multiple cycle delay path:
    - A logic delay path that takes multiple cycles to propagate from one storage element to the next storage element

Instruction Fetch Cycle

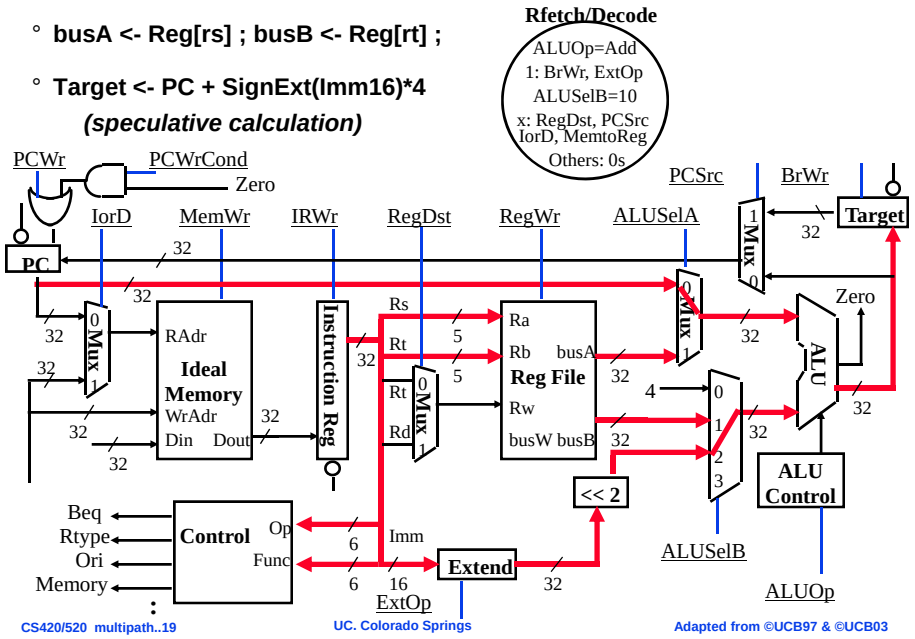
- IR <-- mem[PC]; PC = PC + 4



Register Fetch / Instruction Decode (Continue)

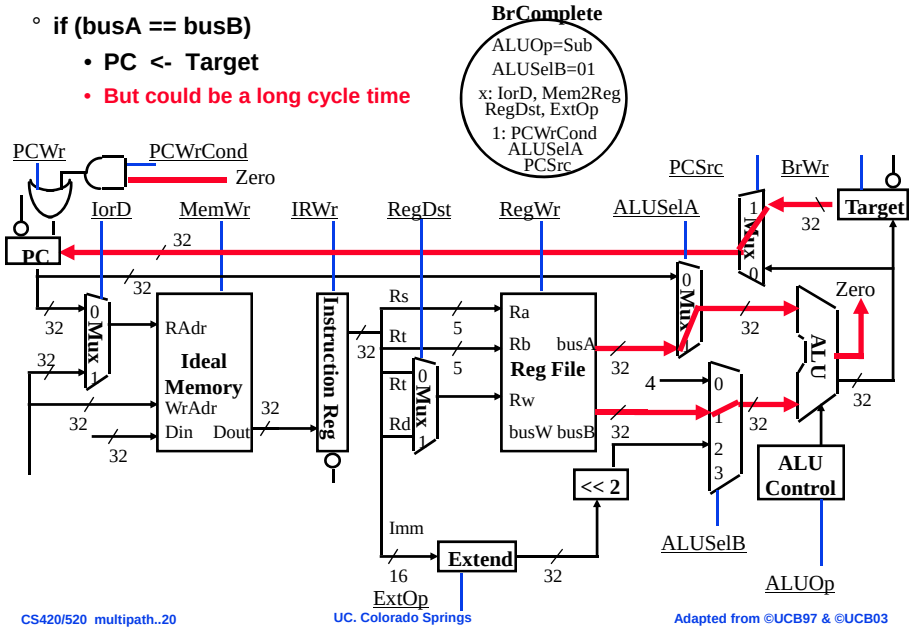
° busA <- Reg[rs] ; busB <- Reg[rt] ;

° Target <- PC + SignExt(Imm16)*4
(speculative calculation)



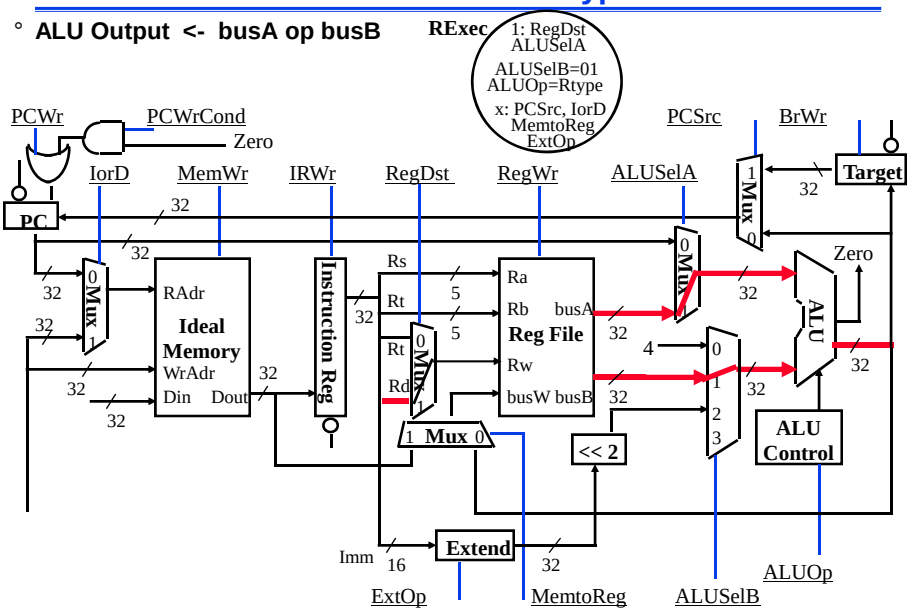
Instruction Decode: We have a Branch!

° if (busA == busB)
- PC <- Target
- But could be a long cycle time



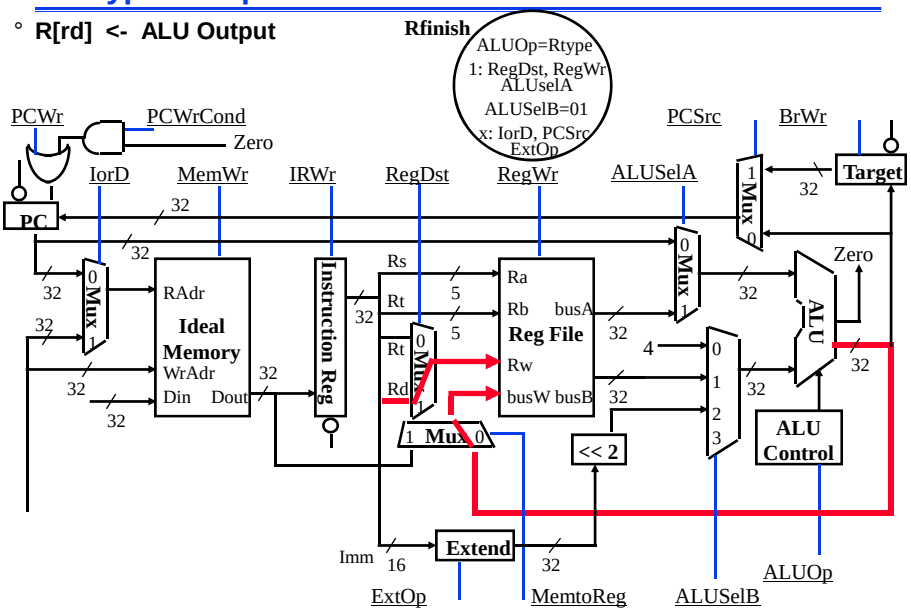
Instruction Decode: We have a R-type!

° ALU Output <- busA op busB



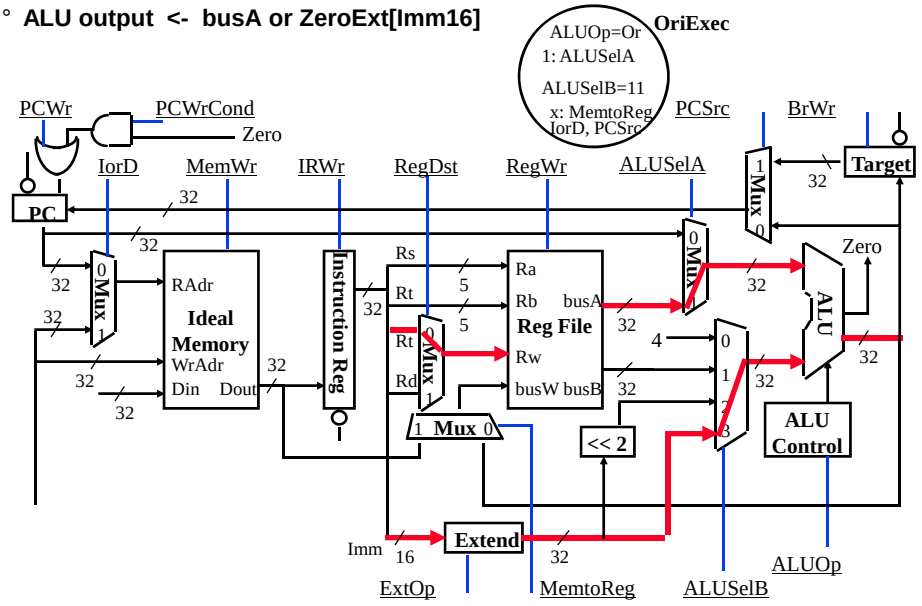
R-type Completion

° R[rd] <- ALU Output



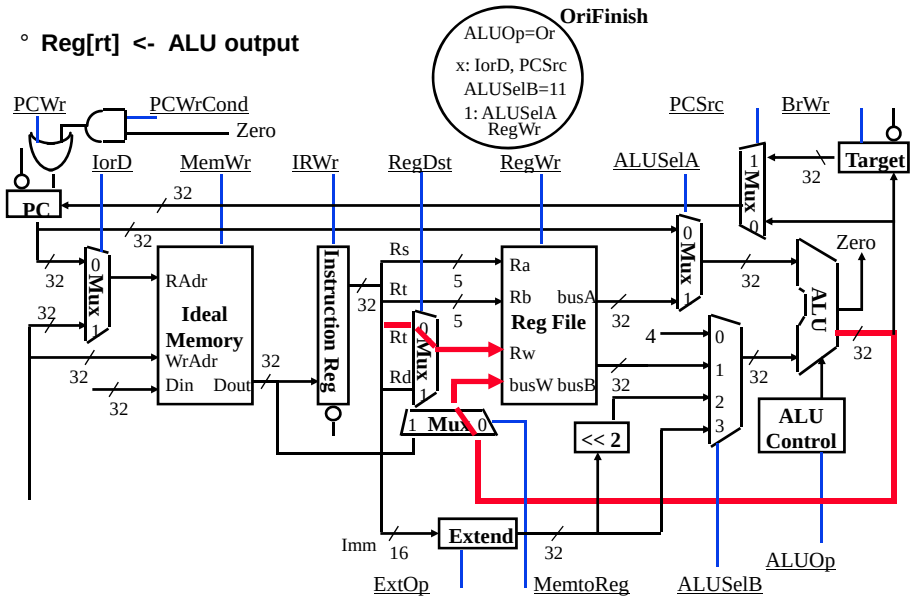
Instruction Decode: We have an Ori!

° ALU output <- busA or ZeroExt[Imm16]



Ori Completion

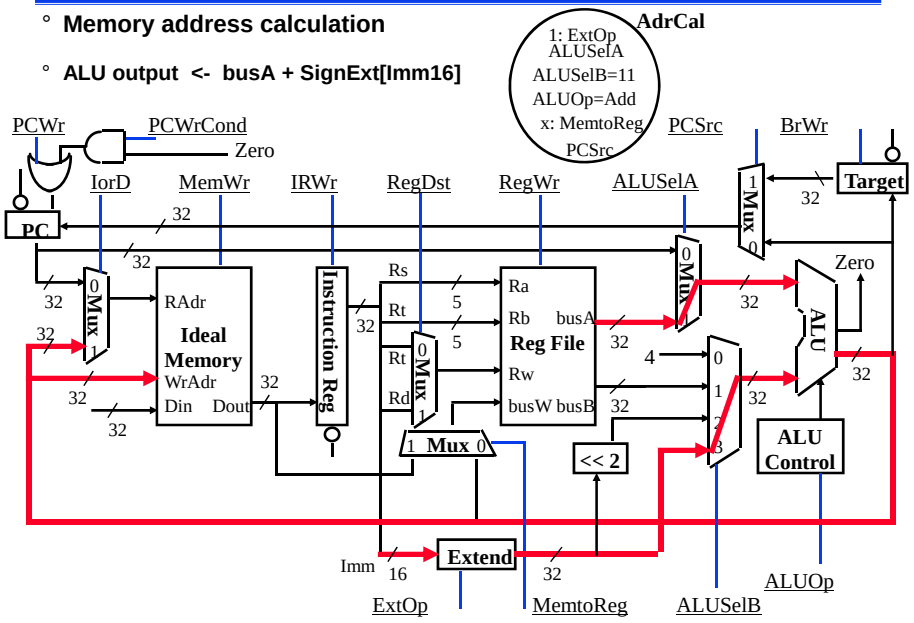
° Reg[rt] <- ALU output



Instruction Decode: We have a Memory Access!

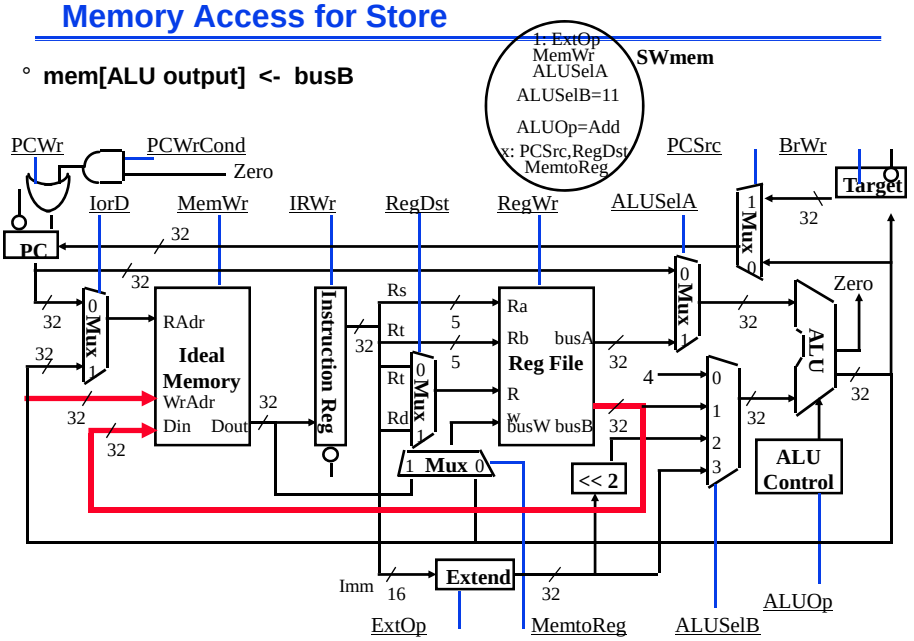
° Memory address calculation

° ALU output <- busA + SignExt[Imm16]



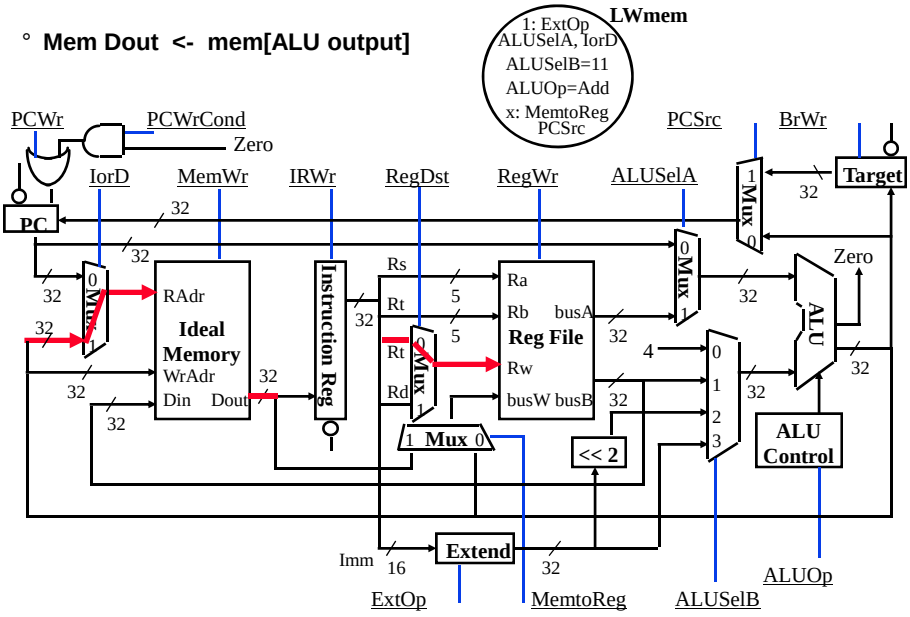
Memory Access for Store

° mem[ALU output] <- busB



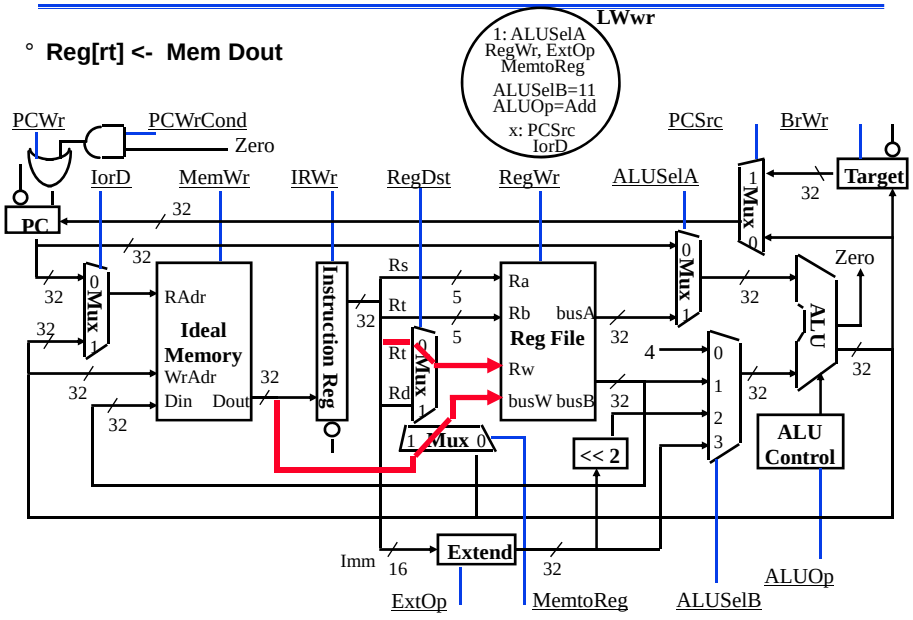
Memory Access for Load

° Mem Dout <- mem[ALU output]



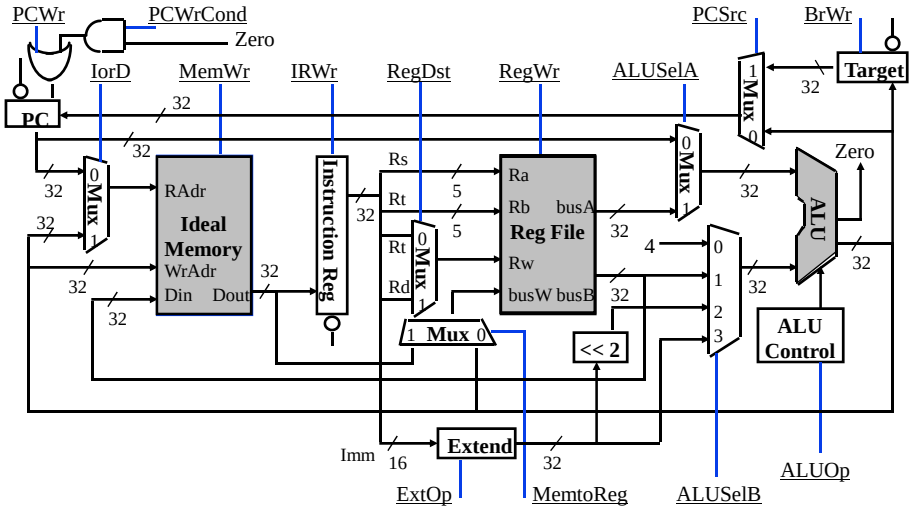
Write Back for Load

° Reg[rt] <- Mem Dout

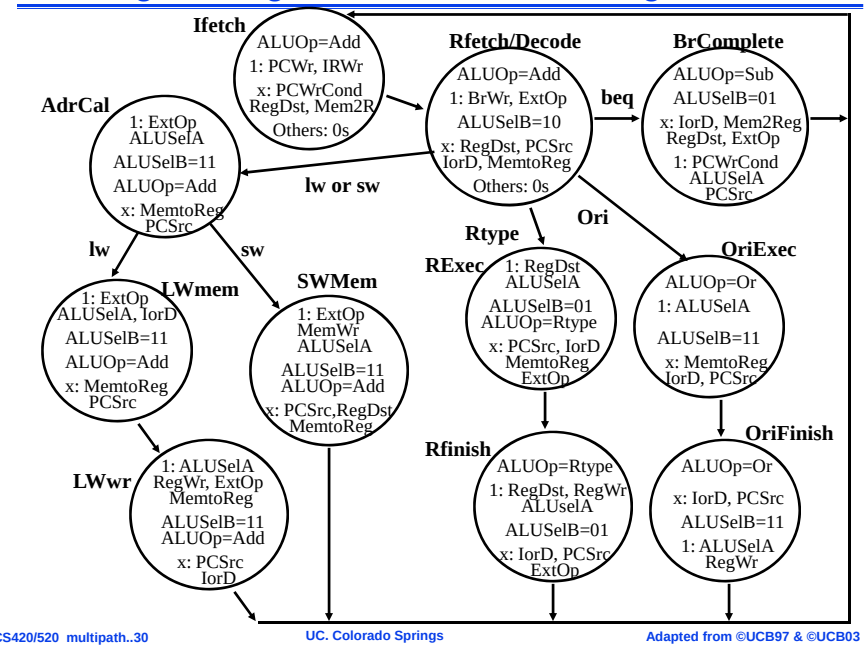


Putting it all together: Multiple Cycle Datapath

° MCP: A functional unit to be used more than once per instruction
  • Not good for pipelining



Putting it all together: Control State Diagram



Summary

- Disadvantages of the Single Cycle Processor
  - Long cycle time
  - Cycle time is too long for all instructions except the Load
- Multiple Cycle Processor:
  - Divide the instructions into smaller steps
  - Execute each step (instead of the entire instruction) in one cycle
- Do NOT confuse Multiple Cycle Processor w/ Multiple Cycle Delay Path
  - Multiple Cycle Processor executes each instruction in multiple clock cycles
  - Multiple Cycle Delay Path: a combinational logic path between two storage elements that takes more than one clock cycle to complete
- It is possible (desirable) to build a MC Processor without MCDP:
  - Use a register to save a signal's value whenever a signal is generated in one clock cycle and used in another cycle later
- More reading: CA3/4: A26-A30, CO2 (Ch 5.4); CO3 (Ch 5.5)