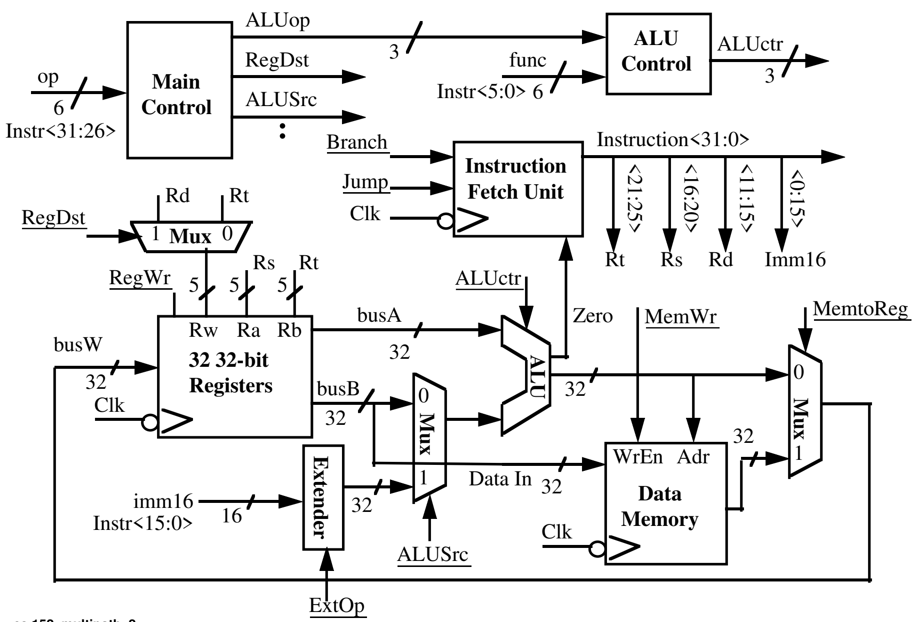
CS152
Computer Architecture and Engineering
Lecture 11: Designing a Multiple Cycle Processor

February 24, 1995

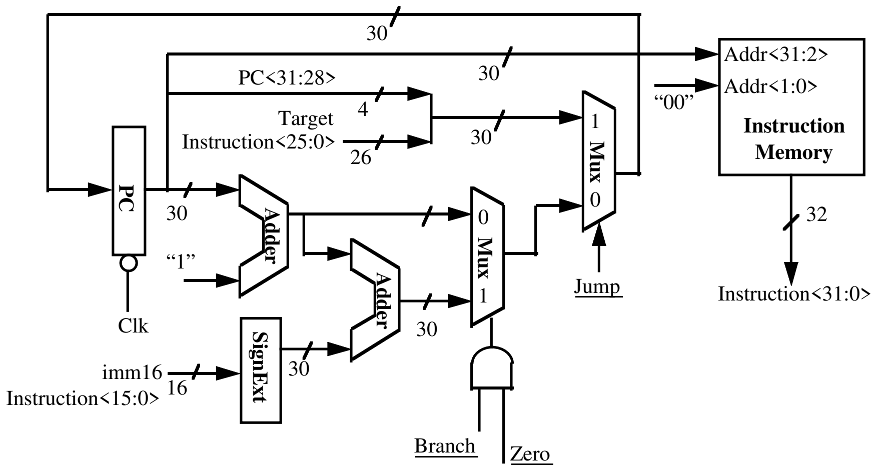
Dave Patterson (patterson@cs) and
Shing Kong (shing.kong@eng.sun.com)

Slides available on http://http.cs.berkeley.edu/~patterson

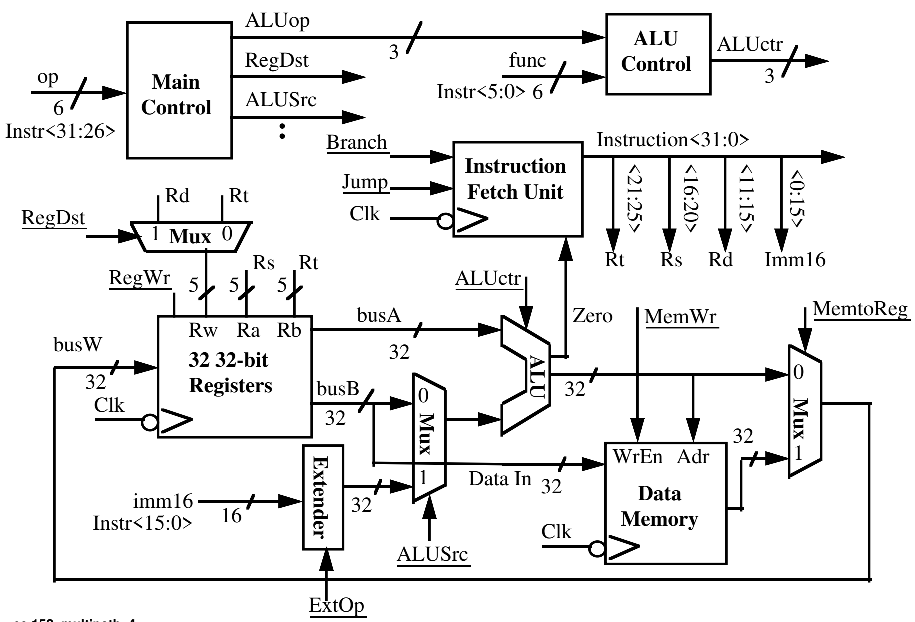
A Single Cycle Processor



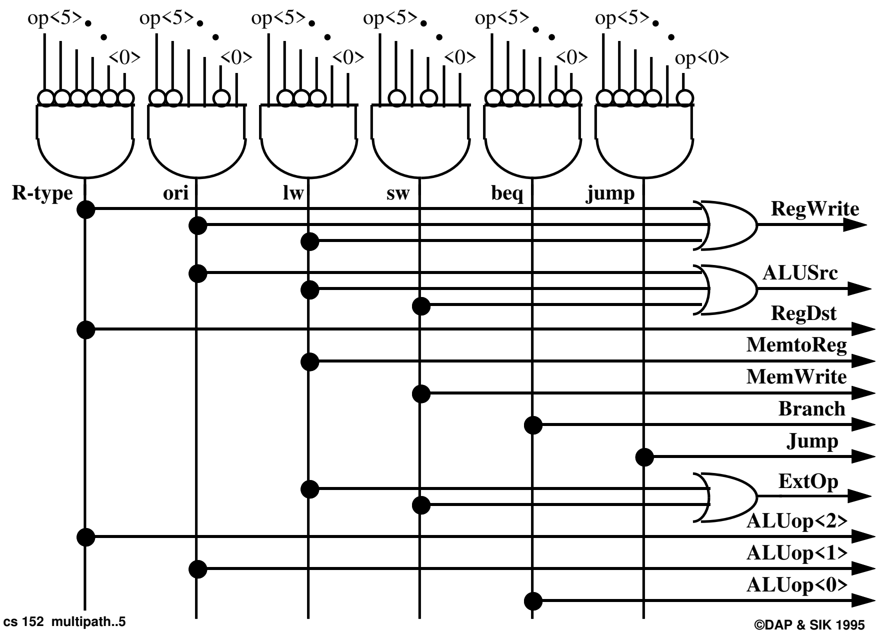
Push: Instruction Fetch Unit



Pop: A Single Cycle Processor



Push: The Main Control



Outline of Today's Lecture

- Recap and Introduction (5 minutes)
- Introduction to the Concept of Multiple Cycle Processor (15 minutes)
- Questions and Administrative Matters (5 minutes)
- Multiple Cycle Implementation of R-type Instructions (15 minutes)
- What is a Multiple Cycle Delay Path and Why is it Bad? (10 minutes)
- Break (5 minutes)
- Multiple Cycle Implementation of Or Immediate (5 minutes)
- Multiple Cycle Implementation of Load and Store (15 minutes)
- Putting it all Together (5 minutes)

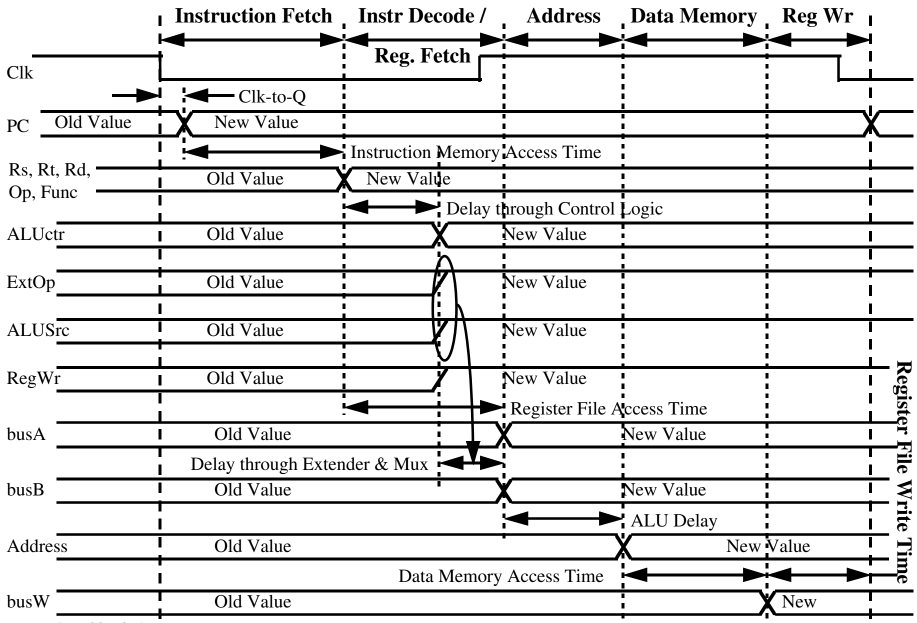
Drawbacks of this Single Cycle Processor

- Long cycle time:
  - Cycle time must be long enough for the load instruction:
    - PC's Clock -to-Q +
    - Instruction Memory Access Time +
    - Register File Access Time +
    - ALU Delay (address calculation) +
    - Data Memory Access Time +
    - Register File Setup Time +
    - Clock Skew
- 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

Overview of a Multiple Cycle Implementation

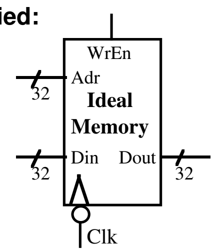
- The root of the single cycle processor's problems:
  - The cycle time has to be long enough for the slowest instruction
- Solution:
  - Break the instruction into smaller steps
  - Execute each step (instead of the entire instruction) in one cycle
    - Cycle time: time it takes to execute the longest 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
    - Jump only takes three cycles
  - Allows a functional unit to be used more than once per instruction

The Five Steps of a Load Instruction

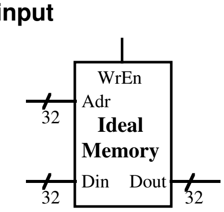


Register File & Memory Write Timing: Ideal vs. Reality

° **In previous lectures, register file and memory are simplified:**
- **Write happens at the clock tick**
- **Address, data, and write enable must be stable one "set-up" time before the clock tick**

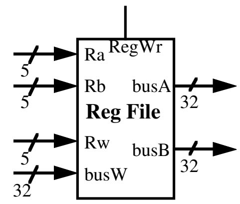


° **In real life:**
- **Neither register file nor ideal memory has the clock input**
- **The write path is a combinational logic delay path:**
  - **Write enable goes to 1 and Din settles down**
  - **Memory write access delay**
  - **Din is written into mem[address]**
- **Important: Address and Data must be stable BEFORE Write Enable goes to 1**

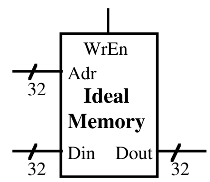


Race Condition Between Address and Write Enable

- This "real" (no clock input) register file may not work reliably in the single cycle processor because:
  - We cannot guarantee Rw will be stable BEFORE RegWr = 1
  - There is a "race" between Rw (address) and RegWr (write enable)



- The "real" (no clock input) memory may not work reliably in the single cycle processor because:
  - We cannot guarantee Address will be stable BEFORE WrEn = 1
  - There is a race between Adr and WrEn

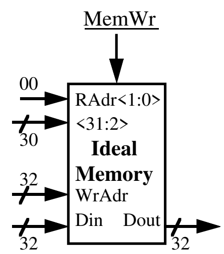


How to Avoid this Race Condition?

- Solution for the multiple cycle implementation:
  - Make sure Address is stable by the end of Cycle N
  - Assert Write Enable signal ONE cycle later at Cycle (N + 1)
  - Address cannot change until Write Enable is disasserted

Dual-Port Ideal Memory

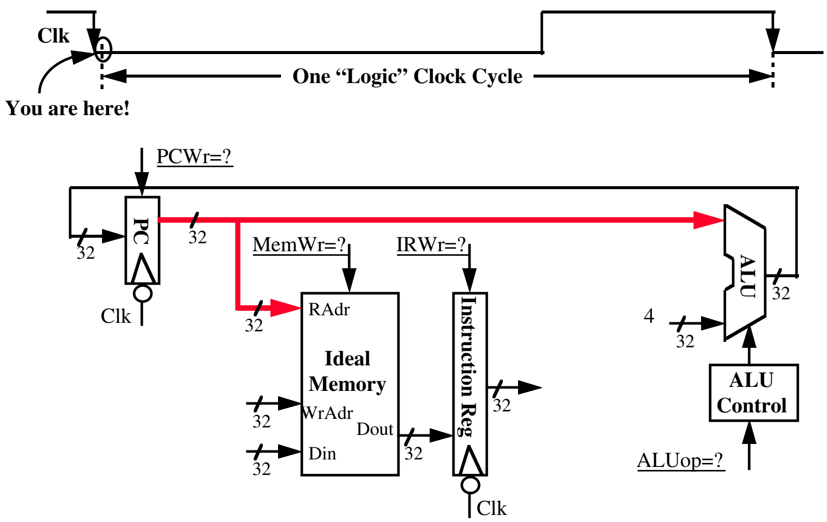
- Dual Port Ideal Memory
  - Independent Read (RAdr, Dout) and Write (WAdr, Din) ports
  - Read and write (to different location) can occur at the same cycle
- Read Port is a combinational path:
  - Read Address Valid -->
  - Memory Read Access Delay -->
  - Data Out Valid
- Write Port is also a combinational path:
  - MemWrite = 1 -->
  - Memory Write Access Delay -->
  - Data In is written into location[WrAdr]



Questions and Administrative Matters

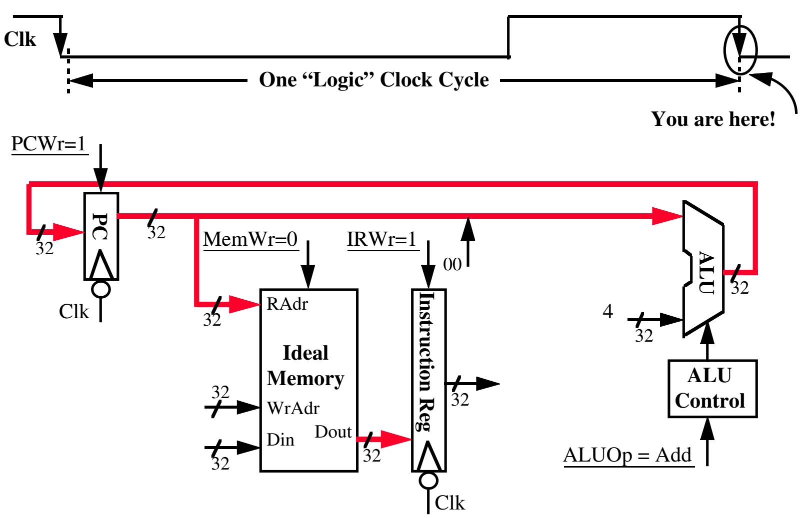
Instruction Fetch Cycle: In the Beginning

° **Every cycle begins right AFTER the clock tick:**
  - **mem[PC]   PC<31:0> + 4**

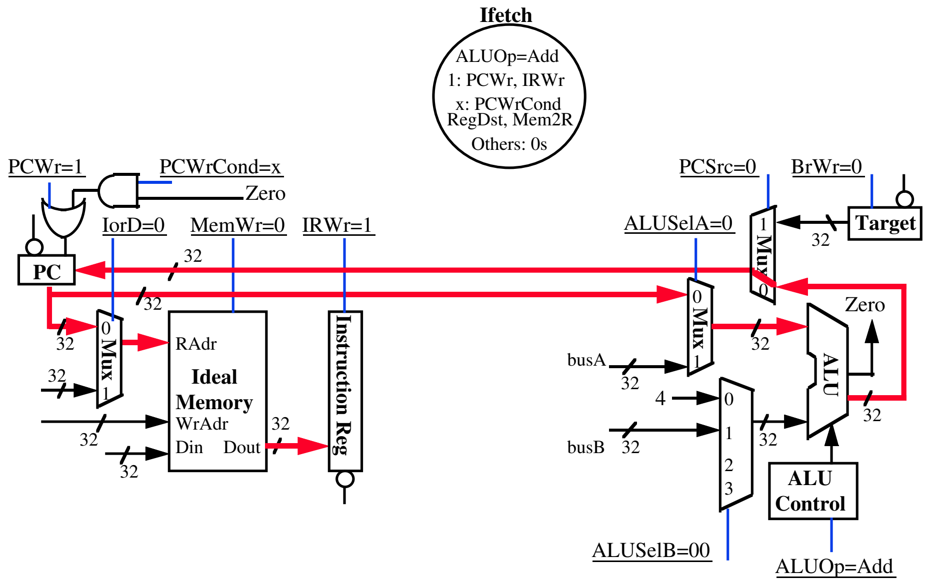


Instruction Fetch Cycle: The End

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

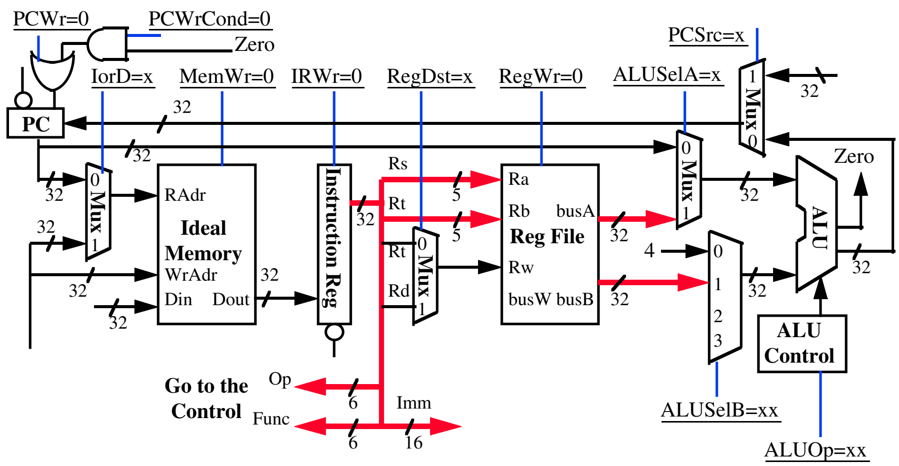


Instruction Fetch Cycle: Overall Picture



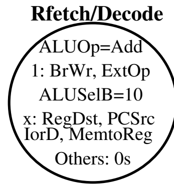
Register Fetch / Instruction Decode

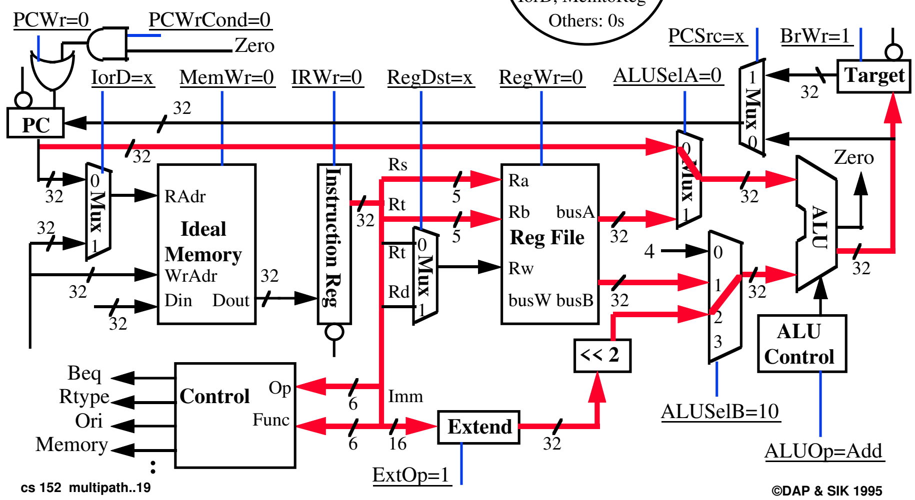
- busA <- RegFile[rs] ; busB <- RegFile[rt] ;
- ALU is not being used: ALUctr = xx



Register Fetch / Instruction Decode (Continue)

- busA <- Reg[rs] ; busB <- Reg[rt] ;
- Target <- PC + SignExt(Imm16)*4

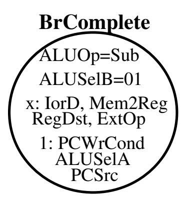


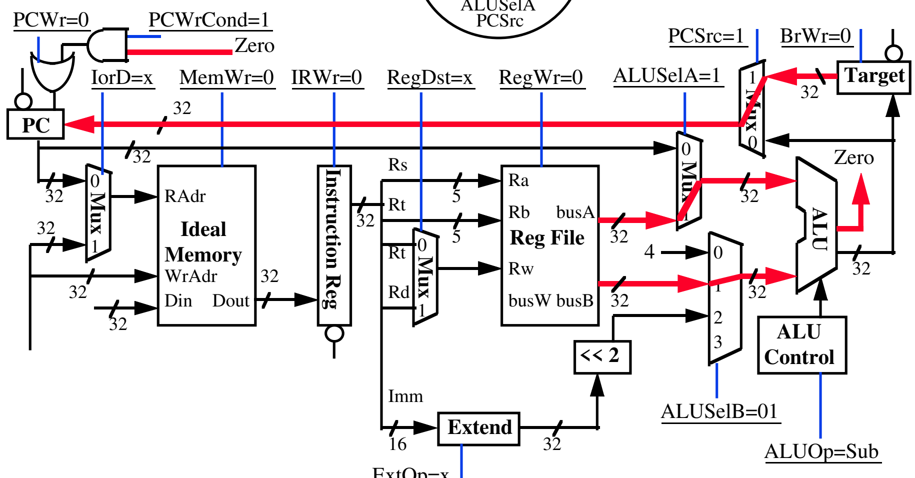


cs 152 multipath..19

Branch Completion

- if (busA == busB)
  - PC <- Target

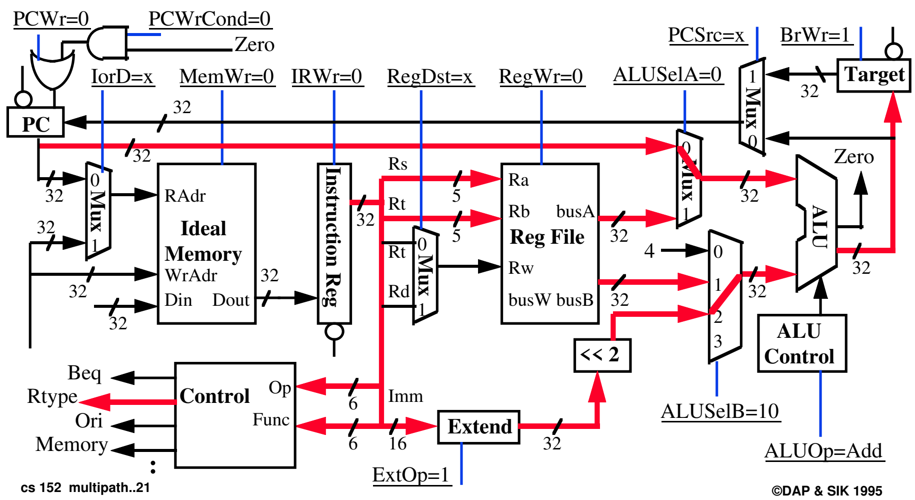




cs 152 multipath..20

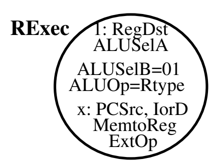
Instruction Decode: We have a R-type!

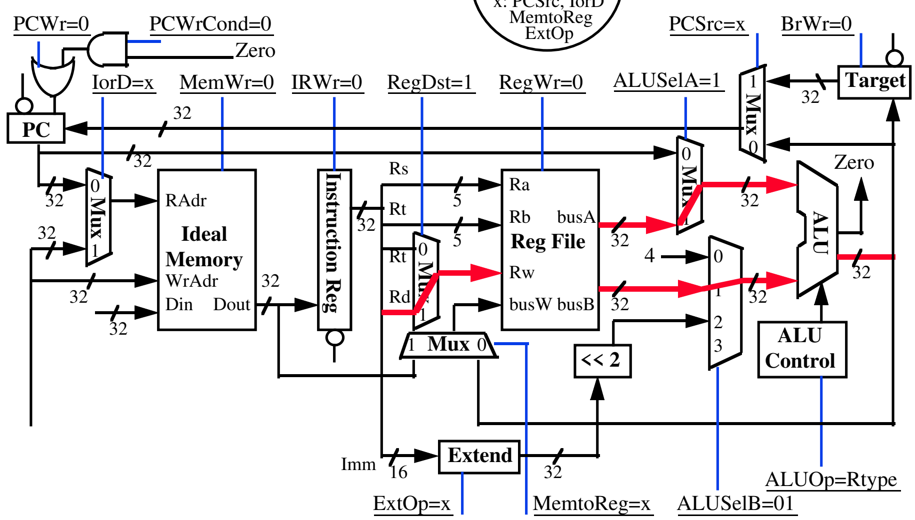
° Next Cycle: R-type Execution



R-type Execution

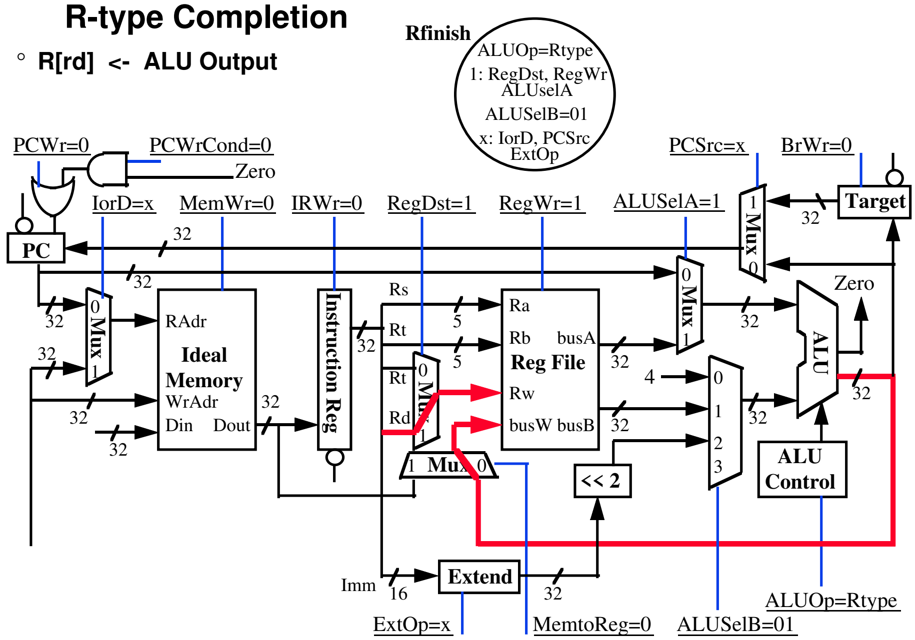
° ALU Output <- busA op busB





R-type Completion

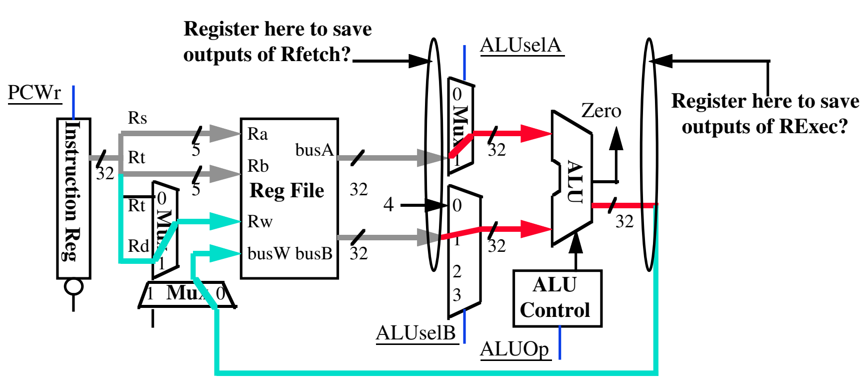
° R[rd] <- ALU Output



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 (P. 316, P&H), 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 regiser 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

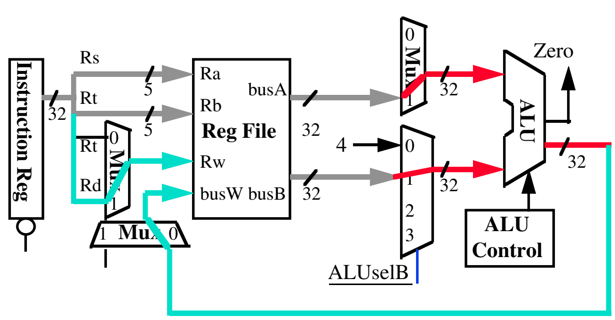
Pros and Cons of a Multiple Cycle Delay Path

° A 3-cycle path example:

- IR (storage) -> Reg File Read -> ALU -> Regr File Write (storage)

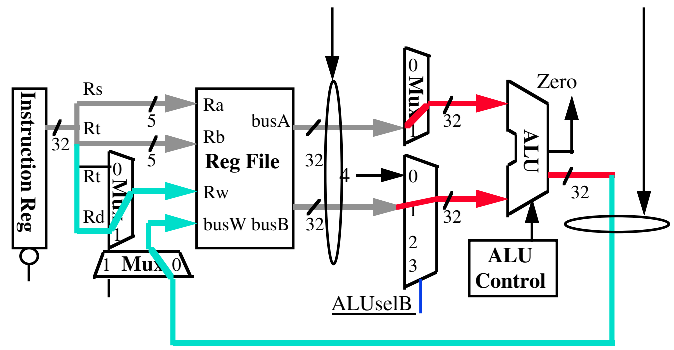
° Advantages:

- Register savings
- We can share time among cycles:
  - If ALU takes longer than one cycle, still "a OK" as long as the entire path takes less than 3 cycles to finish



Pros and Cons of a Multiple Cycle Delay Path (Continue)

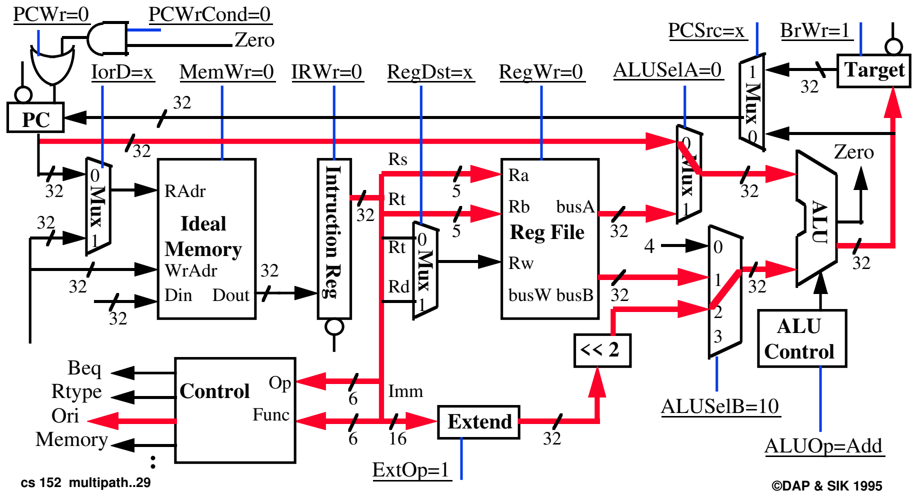
° **Disadvantge:**
  - **Static timing analyzer, which ONLY looks at delay between two storage elements, will report this as a timing violation**
  - **You have to ignore the static timing analyzer's warnings**
  - **But you may end up ignoring real timing violations**
  - **I always TRY to put in registers between cycles to avoid MCP**



Break (5 Minutes)

Instruction Decode: We have an Ori!

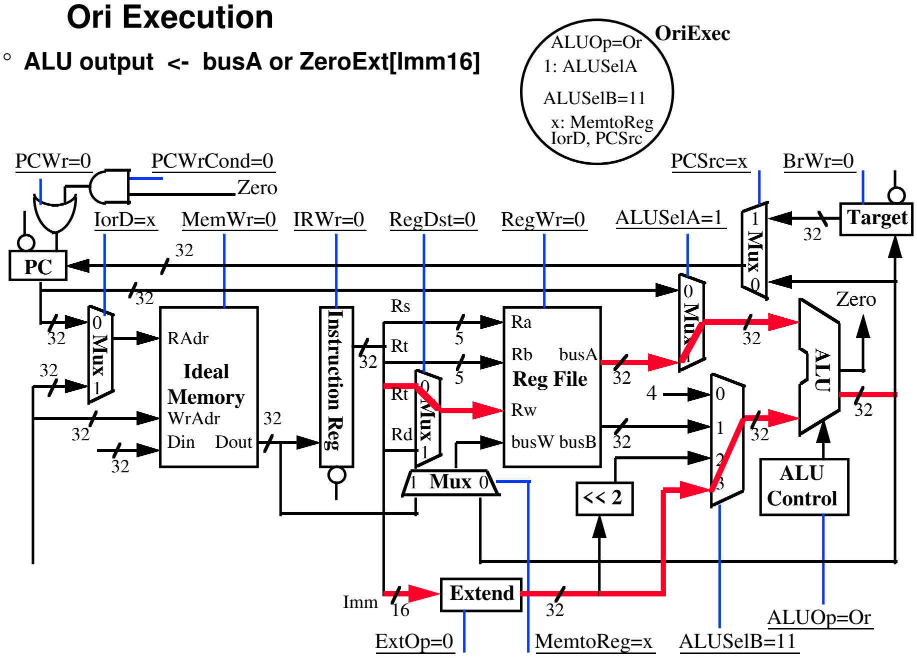
° Next Cycle: Ori Execution



cs 152 multipath..29

Ori Execution

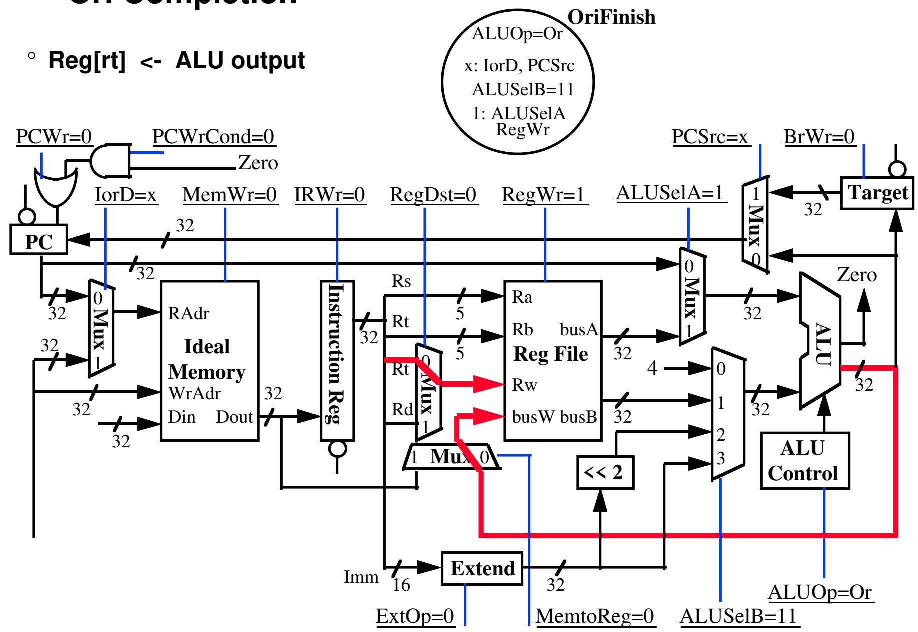
° ALU output <- busA or ZeroExt[Imm16]



cs 152 multipath..30

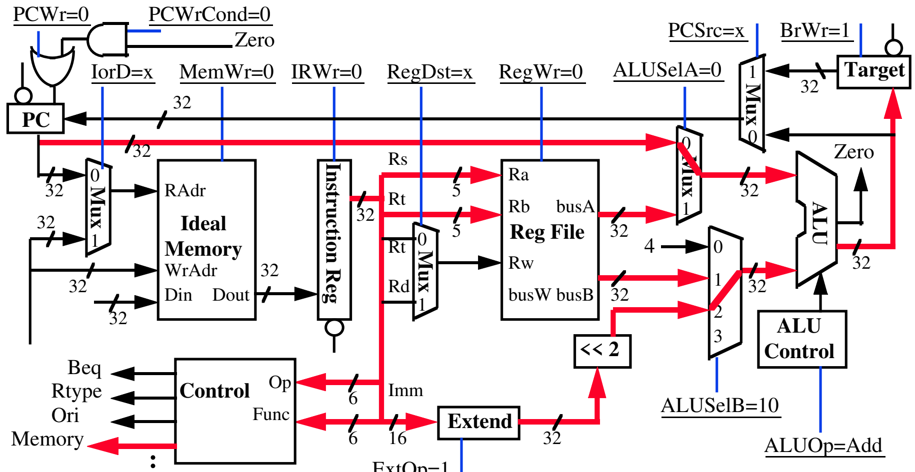
Ori Completion

° Reg[rt] <- ALU output



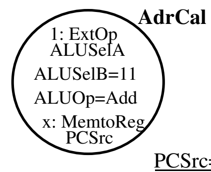
Instruction Decode: We have a Memory Access!

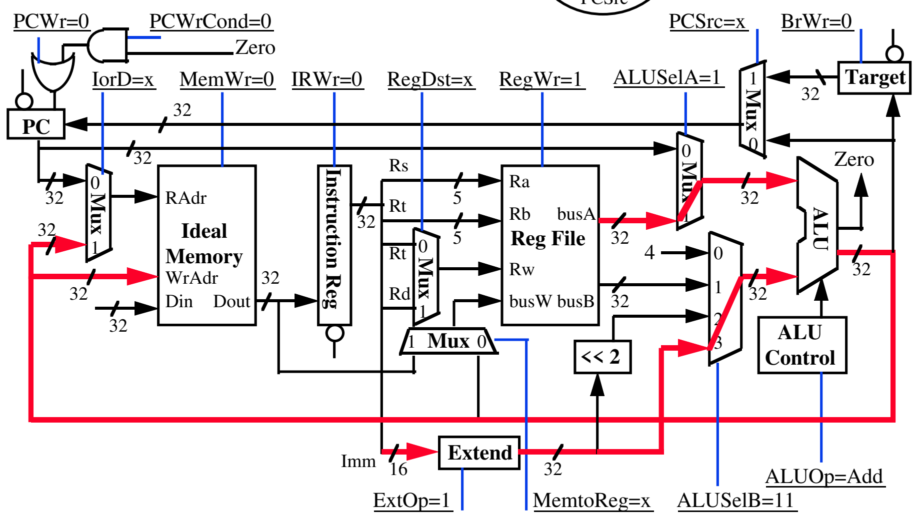
° Next Cycle: Memory Address Calculation



Memory Address Calculation

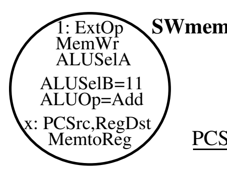
° ALU output <- busA + SignExt[Imm16]

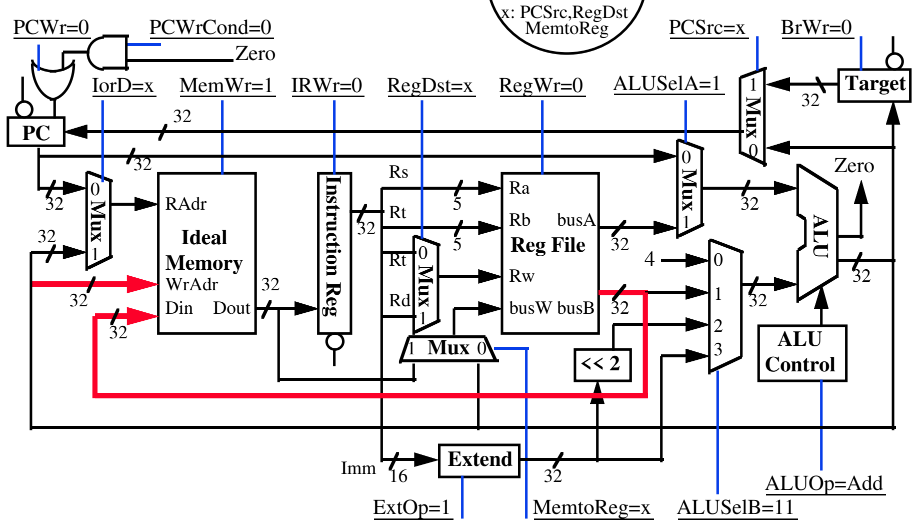




Memory Access for Store

° mem[ALU output] <- busB



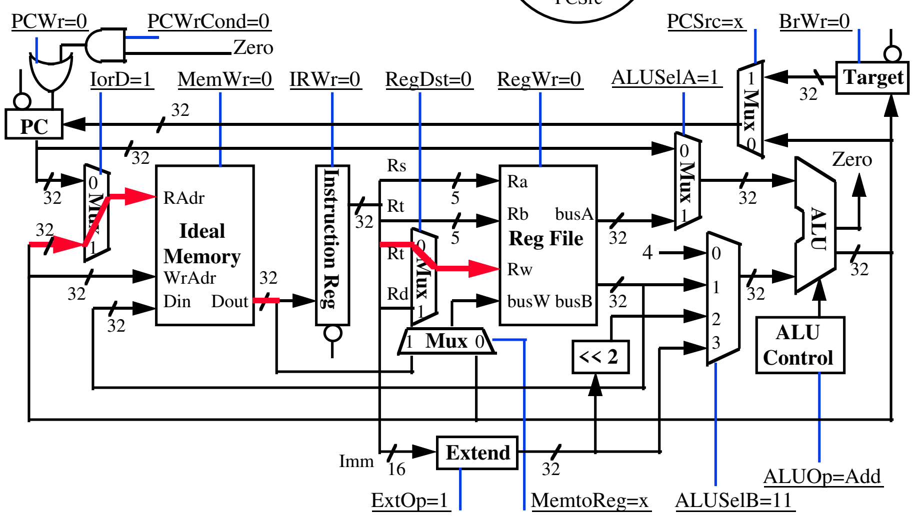


Memory Access for Load

- Mem Dout <- mem[ALU output]

LWmem

1: ExtOp
ALUSelA, IorD
ALUSelB=11
ALUOp=Add
x: MemtoReg
PCSrc

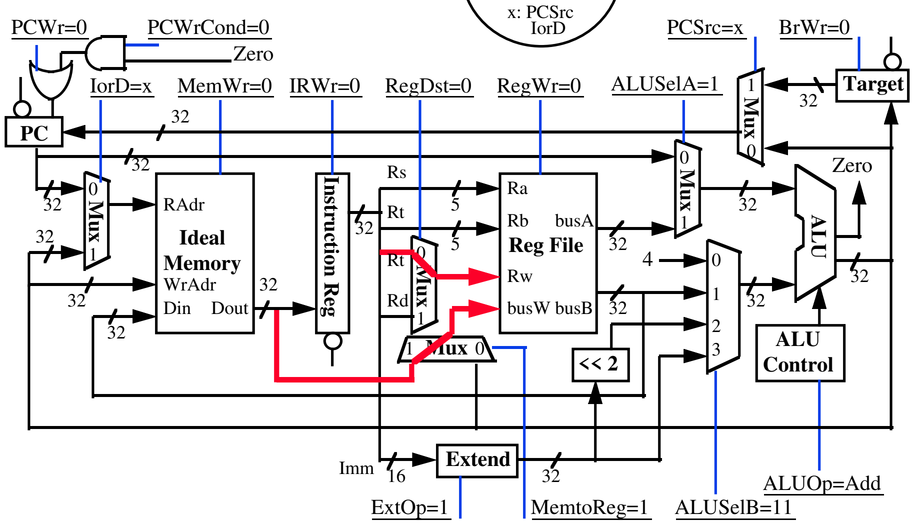


Write Back for Load

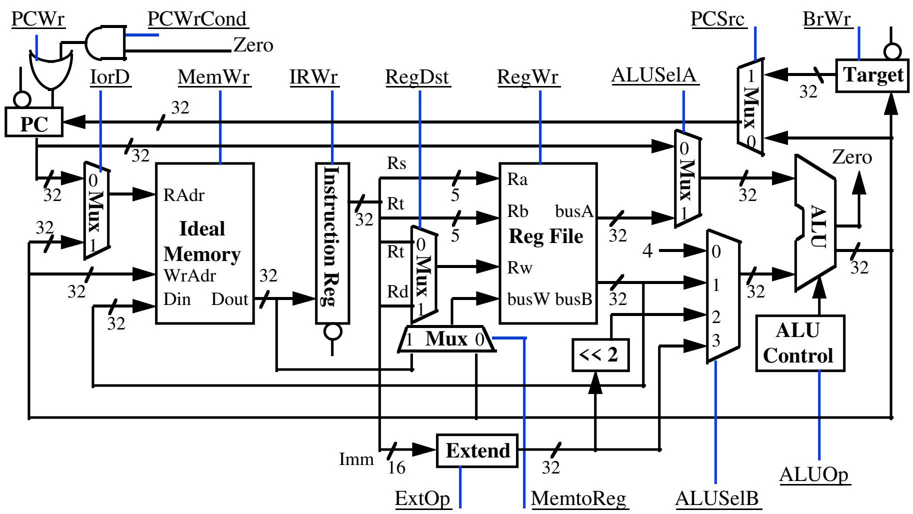
- Reg[rt] <- Mem Dout

LWwr

1: ALUSelA
RegWr, ExtOp
MemtoReg
ALUSelB=11
ALUOp=Add
x: PCSrc
IorD



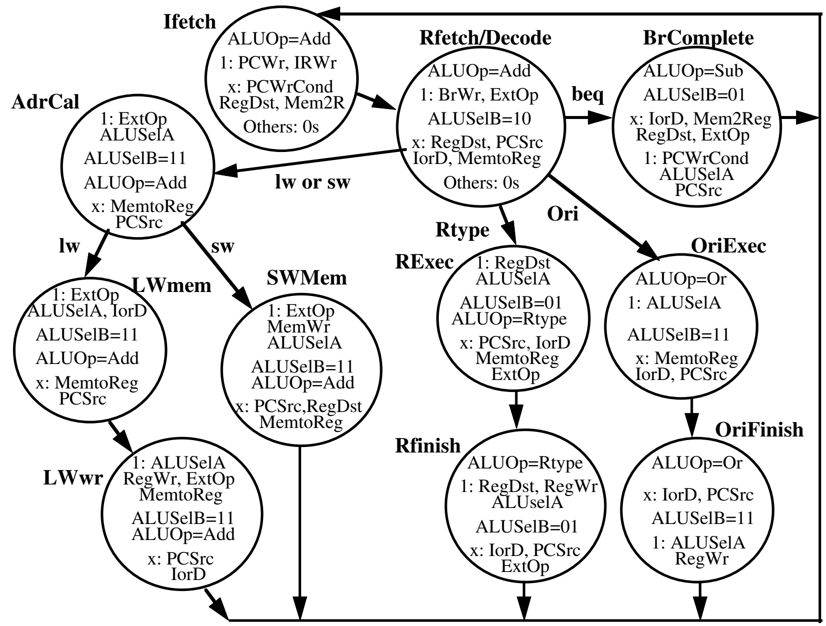
Putting it all together: Multiple Cycle Datapath



Summary

- Disadvantages of the Singple Cycle Proccessor
  - 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 cofuse Multiple Cycle Processor with Multipe Cycle Dealy 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

Putting it all together: Control State Diagram



Where to get more information?

- **Next two lectures:**
  - **Multiple Cycle Controller: Appendix C of your text book.**
  - **Microprogramming: Section 5.5 of your text book.**
- **D. Patterson, “Microprograming,” Scientific America, March 1983.**
- **D. Patterson and D. Ditzel, “The Case for the Reduced Instruction Set Computer,” Computer Architecture News 8, 6 (October 15, 1980)**