Spiral 3-3

Single Cycle CPU

Learning Outcomes

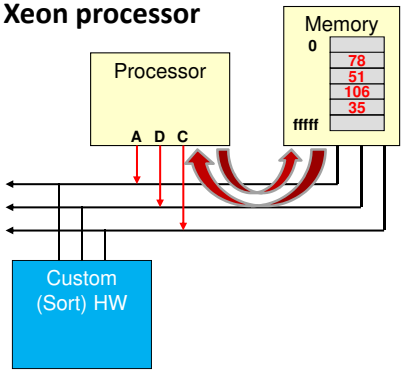
- I understand how the single-cycle CPU datapath supports each type of instruction
- I understand why each mux is needed to select appropriate inputs to the datapath components
- I know how to design the control signals as a function of the type of instruction

Hardware vs. Software

REVIEW

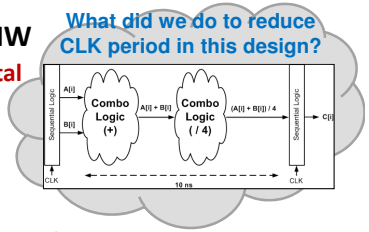
Sorting: Software Implementation

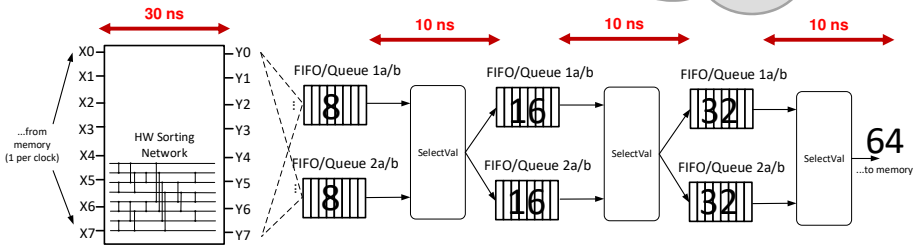
- To perform the algorithm in software means the processor fetches instructions, executes them, which causes the processor to then read and write the data in memory into it's sorted positions
- **Sorting 64 element on a 2.8 GHz Xeon processor**
  - **16 microseconds**
- **Can we do better w/ more HW?**



Sorting: Hardware Implementation

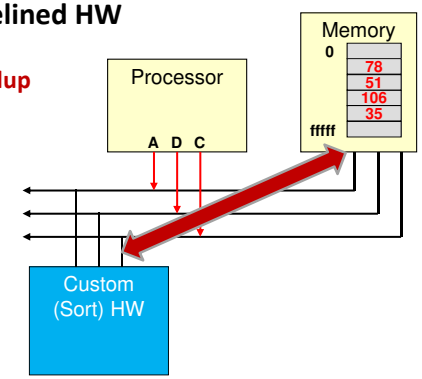
- **Sorting 64 element on a 2.8 GHz Xeon processor [SW only]**
  - **16 microseconds**
- **Sorting 64 numbers in [old] custom HW**
  - **CLK period = 30 ns => 6 microseconds total**
  - **30 ns is due to the 8 number HW sorter**
  - **Merging (Select-Val) stages are < 10 ns**
  - **Can we improve?**





Sorting: Final Comparison

- **Sorting 64 element on a 2.8 GHz Xeon processor [SW only]**
  - **16 microseconds total time**
- **Sorting 64 numbers in [old] custom HW**
  - **CLK period = 30 ns => 6 microseconds total = ~2.5x speedup**
- **Sorting 64 numbers in [old] pipelined HW**
  - **CLK period = 10 ns =>**
    2 microseconds total = ~8x speedup
  - **Processor is freed to do other work**



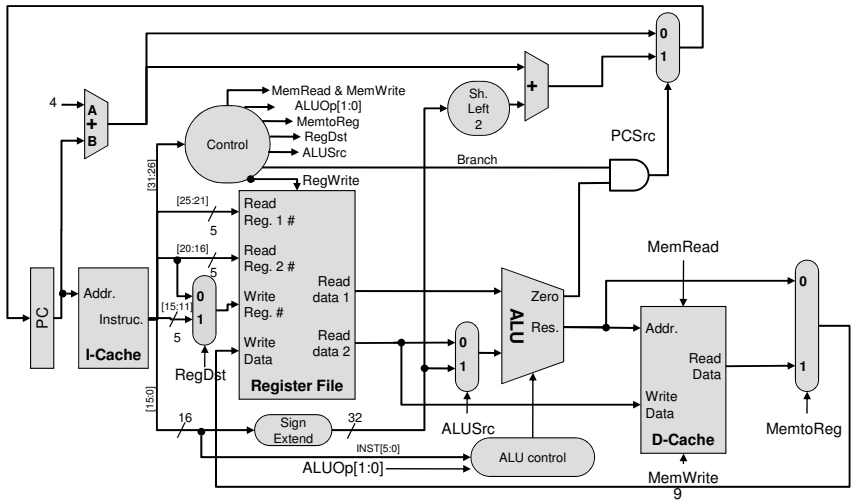
Building hardware to execute software

GENERAL PURPOSE HARDWARE

CPU Organization Scope

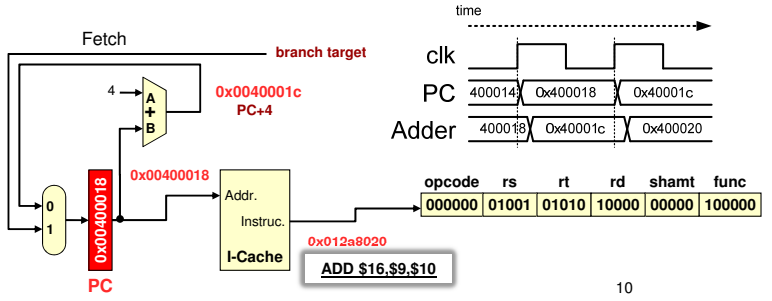
- We will build a CPU to implement our subset of the MIPS ISA
  - Memory Reference Instructions:
    - Load Word (LW)
    - Store Word (SW)
  - Arithmetic and Logic Instructions:
    - ADD, SUB, AND, OR, SLT
  - Branch and Jump Instructions:
    - Branch if equal (BEQ)
    - Jump unconditional (J)
- These basic instructions exercise a majority of the necessary datapath and control logic for a more complete implementation

Single-Cycle CPU Datapath



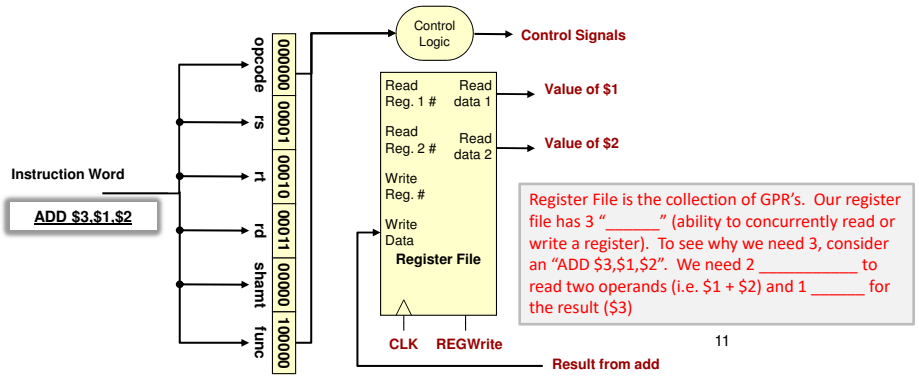
Fetch

- Address in PC is used to fetch instruction while it is also incremented by 4 to point to the next instruction
- Remember, the PC doesn't update until the end of the clock cycle / beginning of next cycle
- Mux provides a path for branch target addresses



Decode

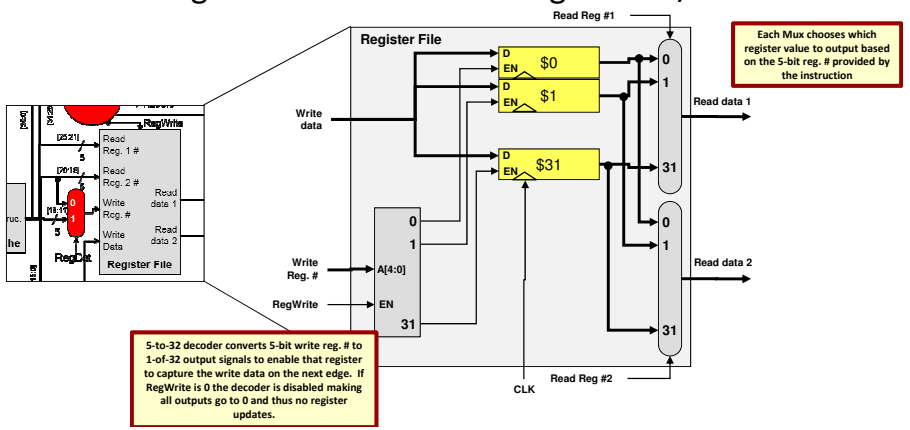
- Opcode and func. field are decoded to produce other control signals
- Execution of an ALU instruction (ADD $3,$1,$2) requires reading 2 register values and writing the result to a third
- REGWrite is an enable signal indicating the write data should be written to the specified register



Register File is the collection of GPR's. Our register file has 3 "_____" (ability to concurrently read or write a register). To see why we need 3, consider an "ADD $3,$1,$2". We need 2 _____ to read two operands (i.e. $1 + $2) and 1 _____ for the result ($3)

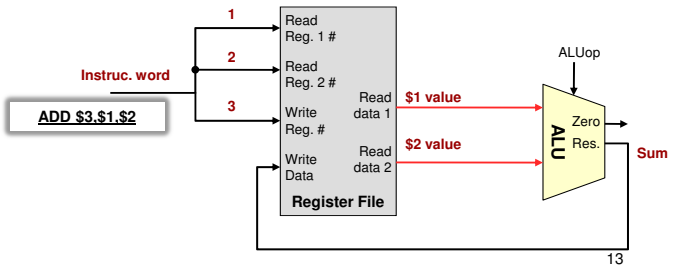
Register File

- 32 registers each storing 32-bits
- Read registers => Muxes to choose desired value
- Write register => Decoder and registers w/ enable



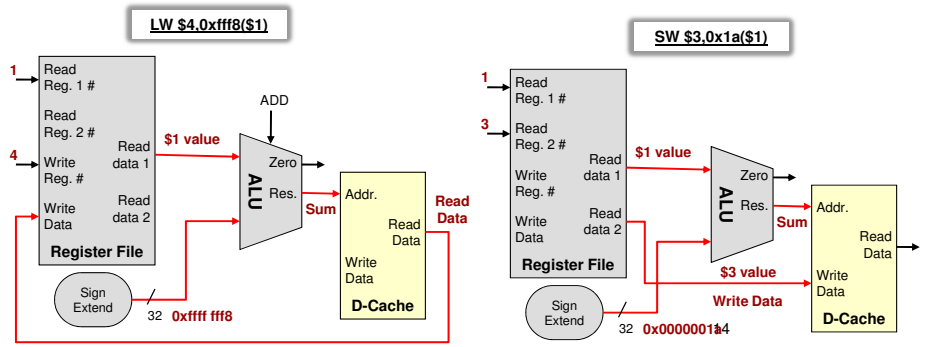
Datapath for ALU instruction

- ALU takes inputs from register file and performs the add, sub, and, or, slt, operations
- Result is written back to dest. register



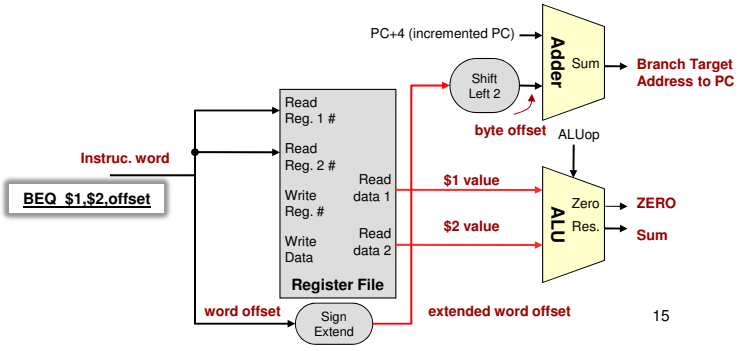
Memory Access Datapath

- Operands are read from register file while offset is sign extended
- ALU calculates _____
- Memory access is performed
- If LW, _____



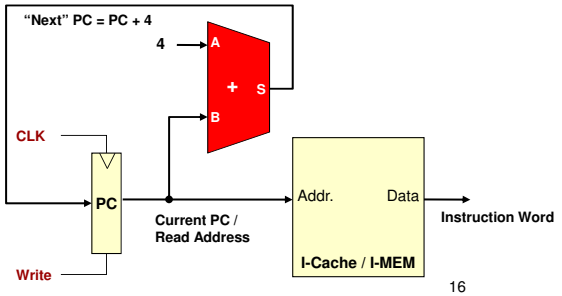
Branch Datapath

- BEQ requires…
  - ALU for comparison (examine 'zero' output)
  - Sign extension unit for branch offset
  - Adder to add PC and offset
    - Need a separate adder since ALU is used to perform comparison



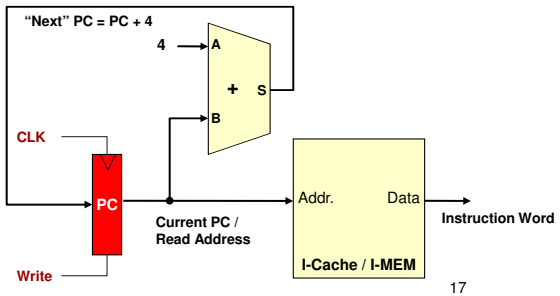
Fetch Datapath Question 1

- Can the adder used to increment the PC be an ALU and be used/shared for ALU instructions like ADD/SUB/etc.
  - In a single-cycle CPU, _____
    _____



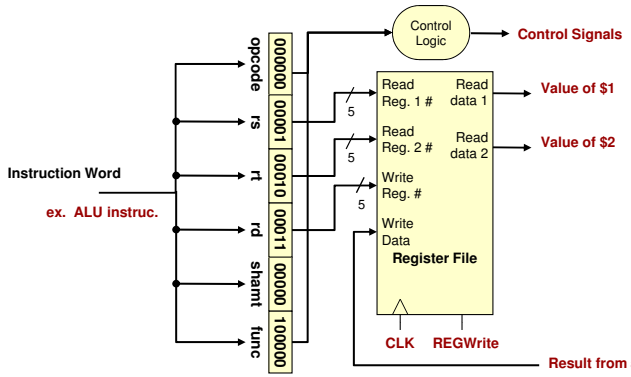
Fetch Datapath Question 2

- Do we need the "Write" enable signal on the PC register for our single-cycle CPU?
  - In the single-cycle CPU, _____



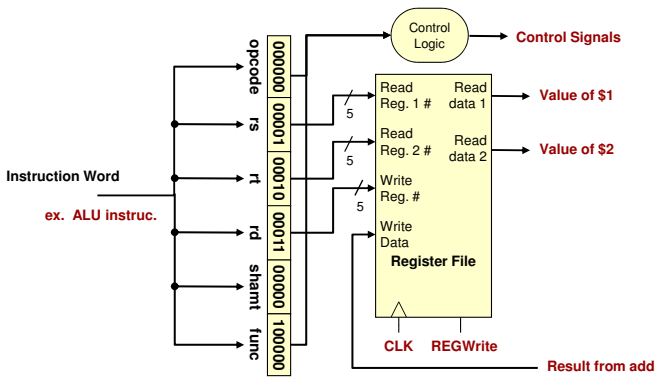
RegFile Question 1

- Why do we need the write enable signal, REGWrite?



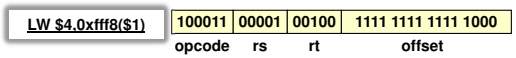
RegFile Question 2

- Can write to registers be level sensitive or does it have to be edge-sensitive?



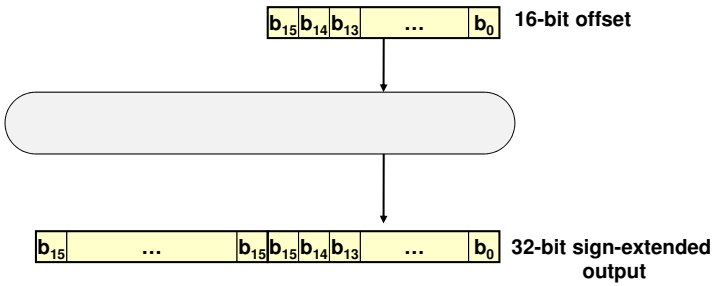
Sign Extension Unit

- In a 'LW' or 'SW' instructions with their base register + offset format, the instruction only contains the offset as a 16-bit value
  - Example: LW $4,-8($1)
  - Machine Code: 0x8c24fff8
    - -8 = 0xfff8
- The 16-bit offset must be extended to 32-bits before being added to base register



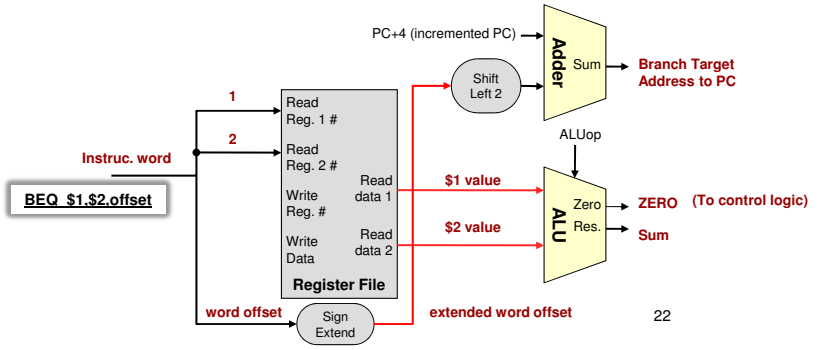
Sign Extension Questions

- What logic is inside a sign-extension unit?
  - How do we sign extend a number?
  - Do you need a shift register?



Branch Datapath Question

- Is it okay to start adding branch offset even before determining whether the branch is taken or not?

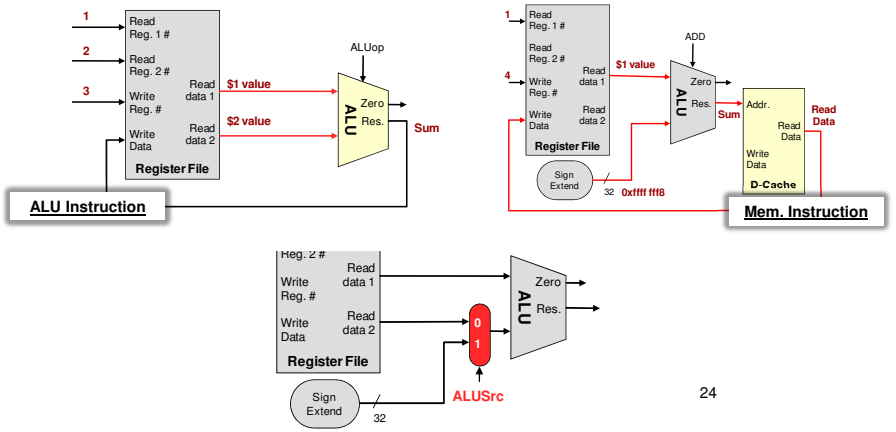


Combining Datapaths

- Now we will take the datapaths for each instruction type and try to combine them into one
- Anywhere we have multiple options for a certain input we can use a mux to select the appropriate value for the given instruction
- Select bits must be generated to control the mux

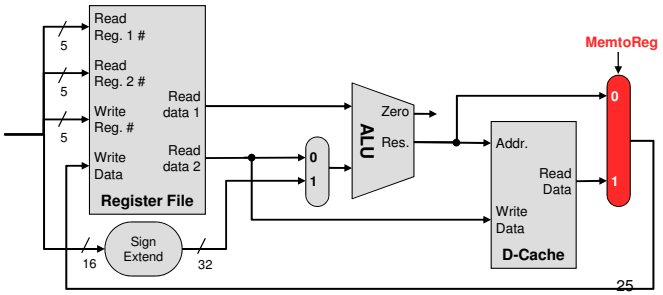
ALUSrc Mux

- Mux controlling second input to ALU
  - ALU instruction provides Read Register 2 data to the 2nd input of ALU
  - LW/SW uses 2nd input of ALU as an offset to form effective address



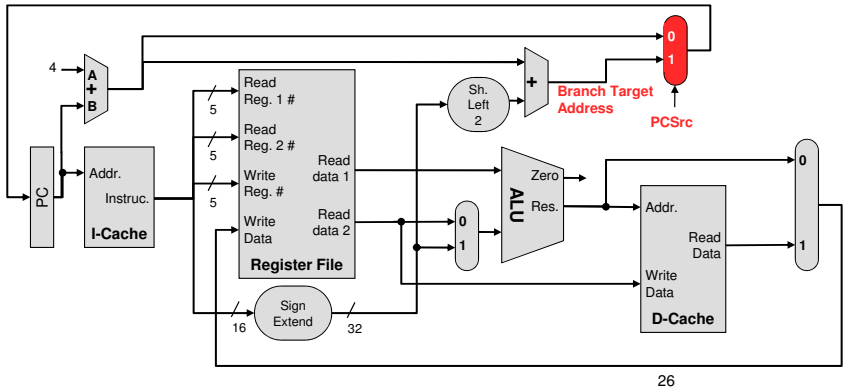
MemtoReg Mux

- Mux controlling writeback value to register file
  - ALU instructions use the result of the ALU
  - LW uses the read data from data memory



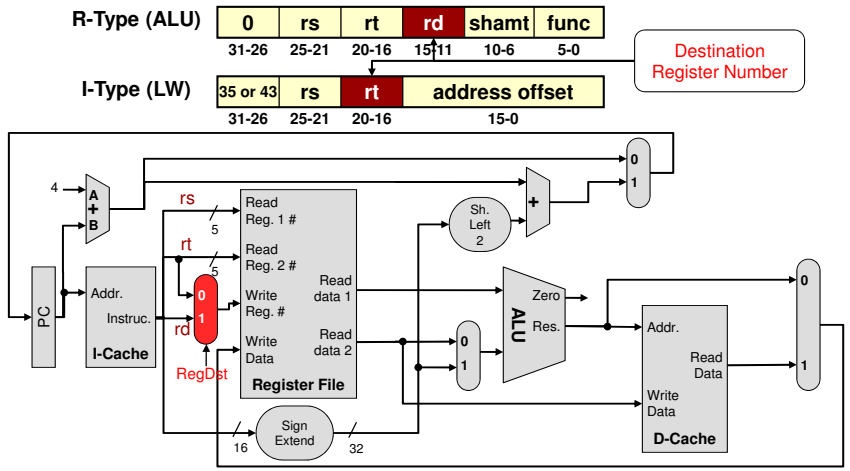
PCSrc Mux

- Next instruction can either be at the next sequential address (PC+4) or the branch target address (PC+offset)

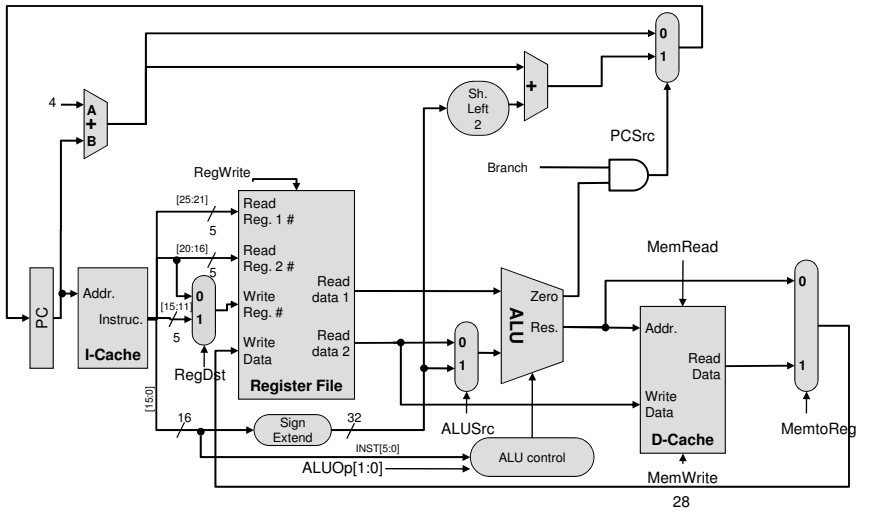


RegDst Mux

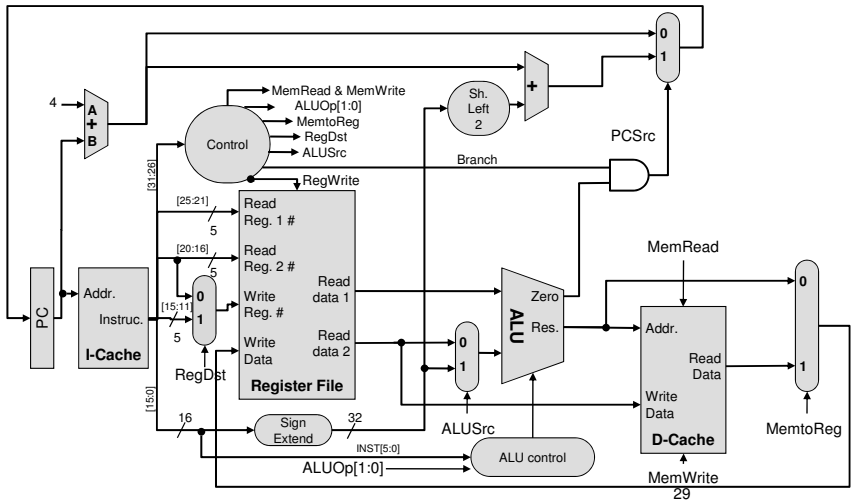
- Different destination register ID fields for ALU and LW instructions



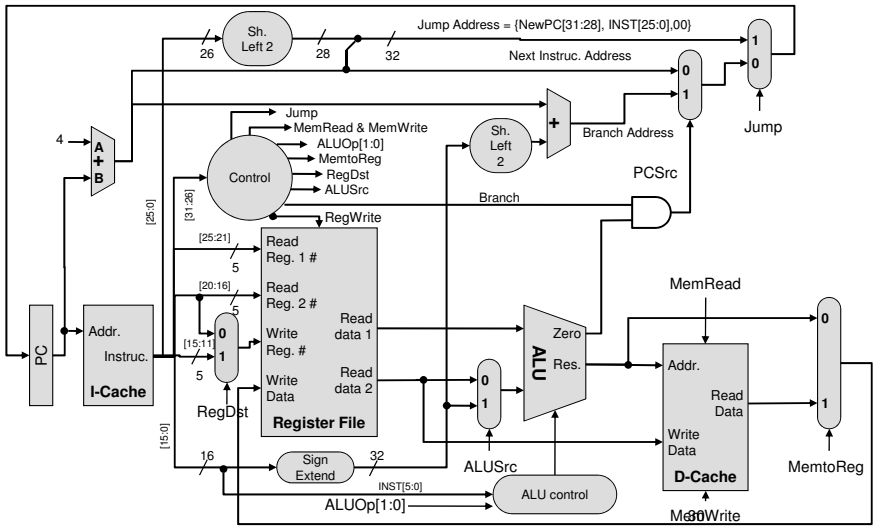
Single-Cycle CPU Datapath



Single-Cycle CPU Datapath

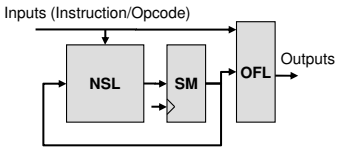


Jump Instruc. Implementation



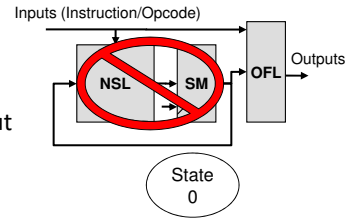
Control Unit Design for Single-Cycle CPU

- Control Unit: Maps instruction to control signals
- Traditional Control Unit
  - FSM: Produces control signals asserted at different times
  - Design NSL, SM, OFL
- Single-Cycle Control Unit
  - Every cycle we perform the same steps: Fetch, Decode, Execute
  - Signals are not necessarily time based but instruction based => only combinational logic



Traditional Control Unit

# of FF's in tightly-encoded state assignment:
5-8 states: _____, 9-16 states: _____

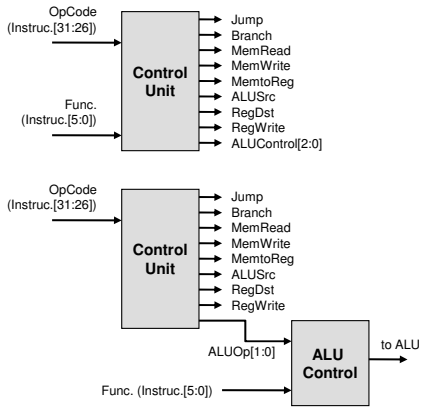


Single-Cycle Control Unit

Only 1 state => _____ FF's

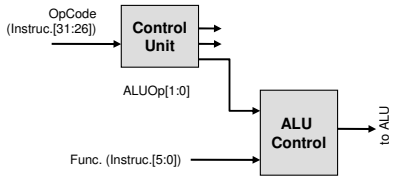
Control Unit

- Most control signals are a function of the opcode (i.e. LW/SW, R-Type, Branch, Jump)
- ALU Control is a function of opcode AND function bits.



ALU Control

- ALU Control needs to know what instruction type it is:
  - R-Type (op. depends on func. code)
  - LW/SW (op. = ADD)
  - BEQ (op. = SUB)
- Let main control unit produce ALUOp[1:0] to indicate _____, then use function bits if necessary to tell the ALU what to do



Instruction	ALUOp[1:0]
LW/SW	00
Branch	01
R-Type	10

Control unit maps instruction opcode to ALUOp[1:0] encoding

ALU Control Truth Table

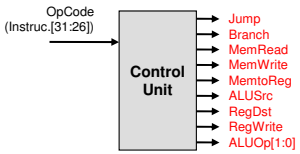
- ALUControl[2:0] is a function of: ALUOp[1:0] and Func.[5:0]

Instruc.	ALUOp[1:0]	Instruction Operation	Func.[5:0]	Desired ALU Action
LW	00	Load word	X	Add
SW	00	Store word	X	Add
Branch	01	BEQ	X	Subtract
R-Type	10	AND	100100	And
R-Type	10	OR	100101	Or
R-Type	10	Add	100000	Add
R-Type	10	Sub	100010	Subtract
R-Type	10	SLT	101010	Set on less than

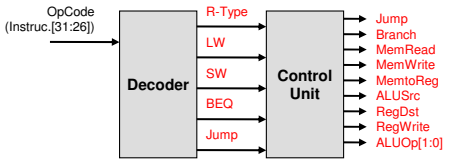
Produce each ALUControl[2:0] bit from the ALUOp and Func. inputs

Control Signal Generation

- Other control signals are a function of the opcode
- We could write a full truth table or (because we are only implementing a small subset of instructions) simply decode the opcodes of the specific instructions we are implementing and use those intermediate signals to generate the actual control signals



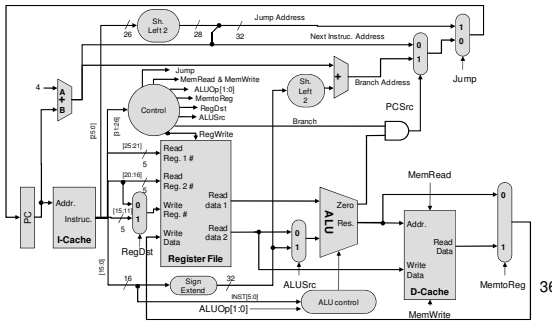
Could generate each control signal by writing a full truth table of the 6-bit opcode



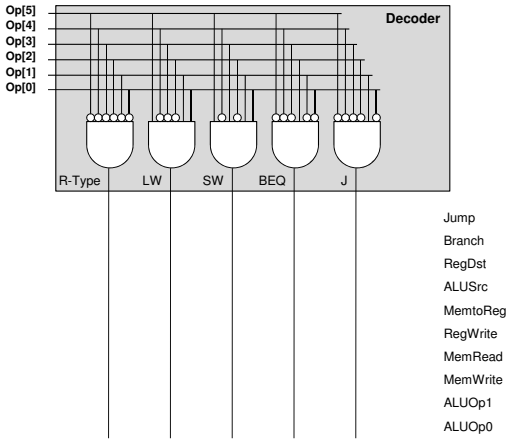
Simpler for human to design if we decode the opcode and then use individual "instruction" signals to generate desired control signals

Control Signal Truth Table

R-Type	LW	SW	BEQ	J	Jump	Branch	Reg Dst	ALU Src	Memto-Reg	Reg Write	Mem Read	Mem Write	ALU Op[1]	ALU Op[0]
1	0	0	0	0	0	0					0	0	1	0
0	1	0	0	0	0	0					1	0	0	0
0	0	1	0	0	0	0					0	1	0	0
0	0	0	1	0	0	1					0	0	0	1
0	0	0	0	1	1	X					0	0	X	X



Control Signal Logic



Credits

- These slides were derived from Gandhi Puvvada's EE 457 Class Notes