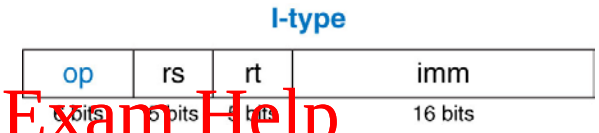
ICS51 - MIPS Single-Cycle Datapath Adding Instructions & Performance

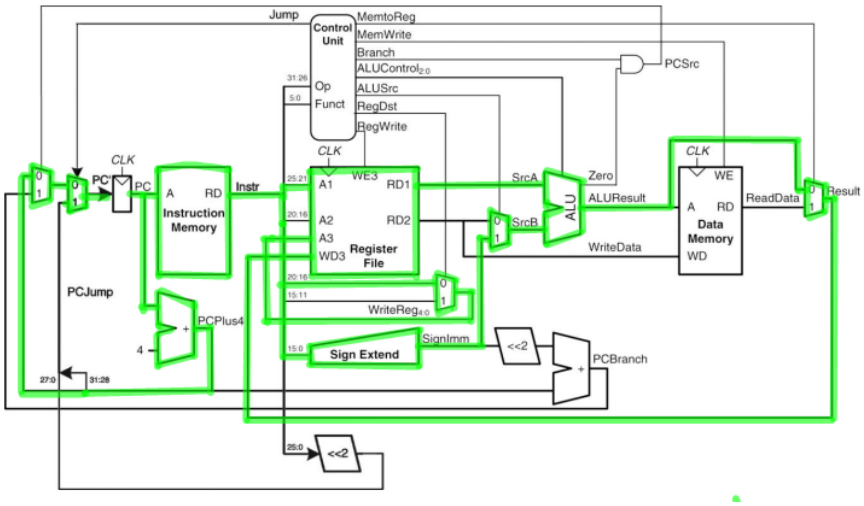
- The basic datapath we have built only implements the instructions lw, sw, beq, R-type and jump. In order to add the other instructions from the MIPS assembly instruction set additional hardware in the datapath is needed.
- To add a new instruction to the datapath, the key steps are:
  - Determine which group of existing instructions the new instruction is most like, if any.
  - Determine the flow of the data through the datapath, starting after the Fetch stage
    - Remember, the Fetch stage is the same for all instructions
  - Determine any necessary changes to the existing datapath and any additional hardware that may be required (always add hardware as a last resort)
  - Determine the control signals for the existing system and added hardware
    - You may need to change existing controls to add more bits.
    - Make sure you change the control values for the existing instructions and specify any new controls

Ex: `addi rt, rs, immediate` # Reg[rt] = Reg[rs] + sign-extended immediate



- The rs register (Reg[rs]) is read from the register file. Reg[rs] is added to the sign-extended immediate value. The result of the ALU is stored into Reg[rt]
- This instruction is most similar to the lw/sw instruction. It uses the immediate field and adds with the ALU. However instead of using the results of the ALU as the memory address it is stored into a register similar to an R-type instruction.
- No datapath changes are required. Why? Because the data can flow through the wires used by the lw/sw and R-type instruction already implemented
- The control signals for the entire datapath must be defined to operate this new instruction. By considering what task each component is performing for the instruction, we can determine which control signals need to be asserted/de-asserted (1/0). Only the added row for the control table is shown.

Instr	RegDst	RegWrite	ALUSrc	Mem Read	Mem Write	Memto Reg	Branch	ALUOp	Jump
addi	0	1	1	0	0	0	0	00	0



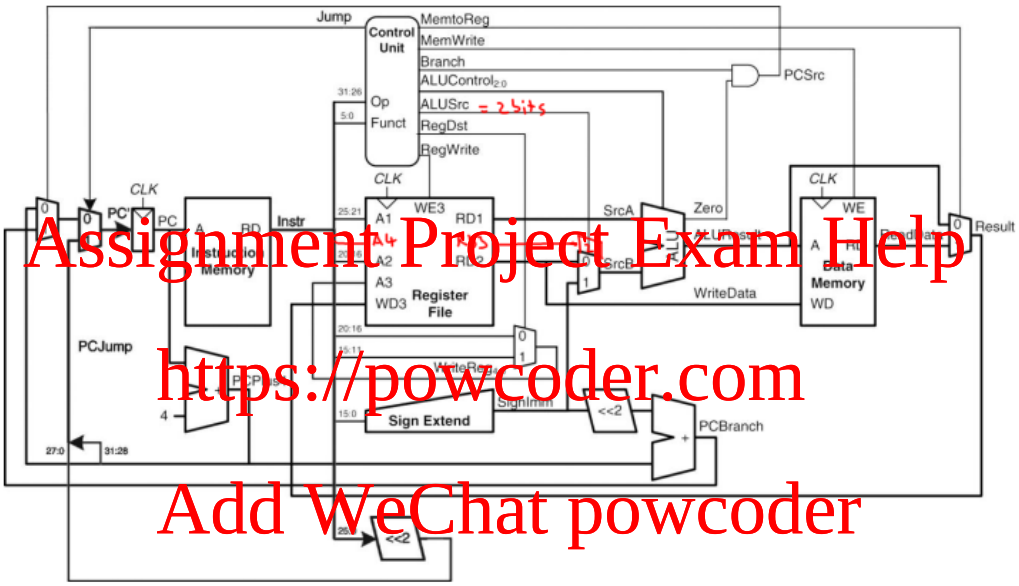
Ex: `swr rt, rs(rd)  # Mem[ Reg[rd] + Reg[rs] ] = Reg[rt]`

[swr Instruction Walk-through]

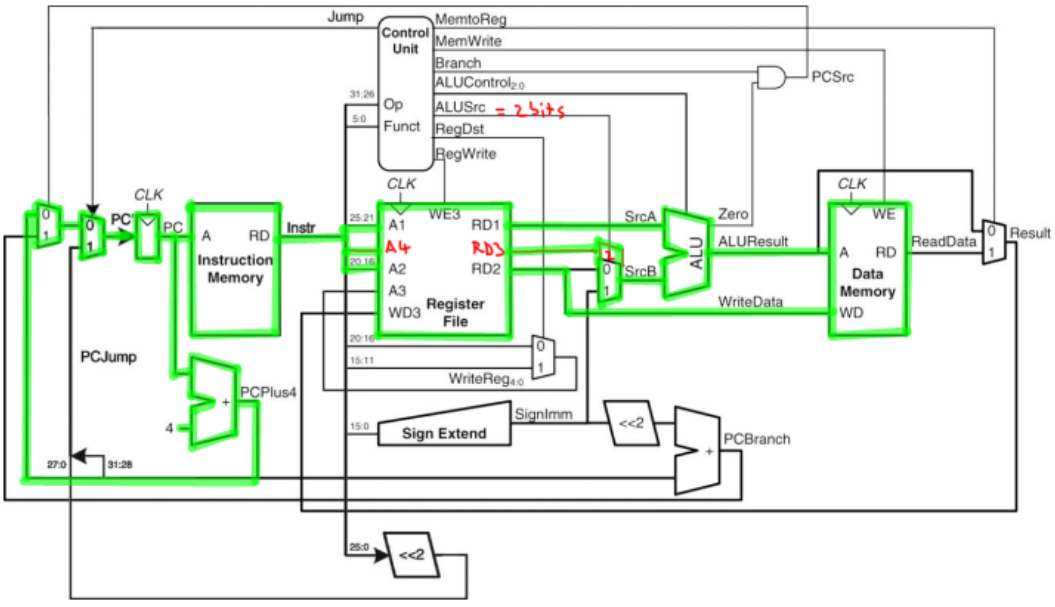
R-type

op	rs	rt	rd	shamt	funct
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits

- The instruction stores data in memory like sw, but to the address specified by the addition of 2 registers
- This instruction must be an R-format instruction because it specifies 3 registers
- We need to read the rs and rd registers from the register file and send the stored values to the ALU to be added.
- In addition, the rt register must be read from the register file to obtain the value to store at the ALU calculated memory address.
- Since each component of the datapath can only be used once in the single cycle datapath, we must modify the register file to read 3 registers: rs, rt, and rd.
- In order to pass the value of Reg[rd] to the ALU for the addition, we must expand the ALUSrc MUX to include this new input. The ALUSrc control must be expanded to 2 bits for all existing instructions
- The required changes to the datapath are shown in red



- The highlighted areas represent the instruction operation

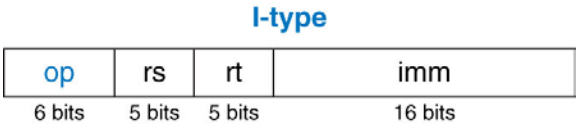


- The ALUSrc control must be expanded for all instructions and the control signals defined for the new instruction

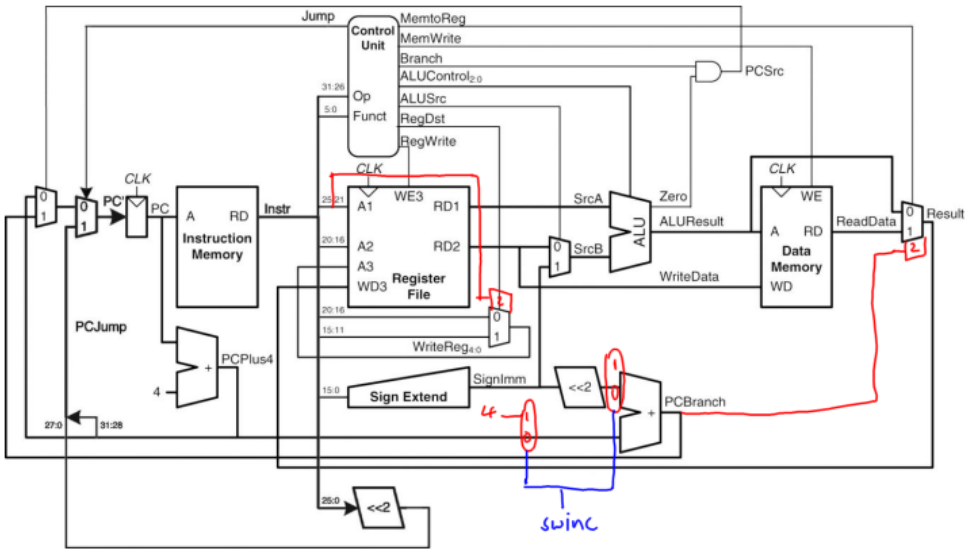
Instr	RegDst	RegWrite	ALUSrc	Mem Read	Mem Write	Memto Reg	Branch	ALUOp	Jump
lw	0	1	01	1	0	1	0	00	0
sw	X	0	01	0	1	X	0	00	0
R-type	1	1	00	0	0	0	0	10	0
beq	X	0	00	0	0	X	1	01	0
Jump	X	0	XX	0	0	X	X	XX	1
swr	X	0	10	0	1	X	0	00	0

Ex: `swinc rt, immediate(rs)` `# Mem[Reg[rs]+immediate] = Reg[rt]`
`# Reg[rs] = Reg[rs] +4`

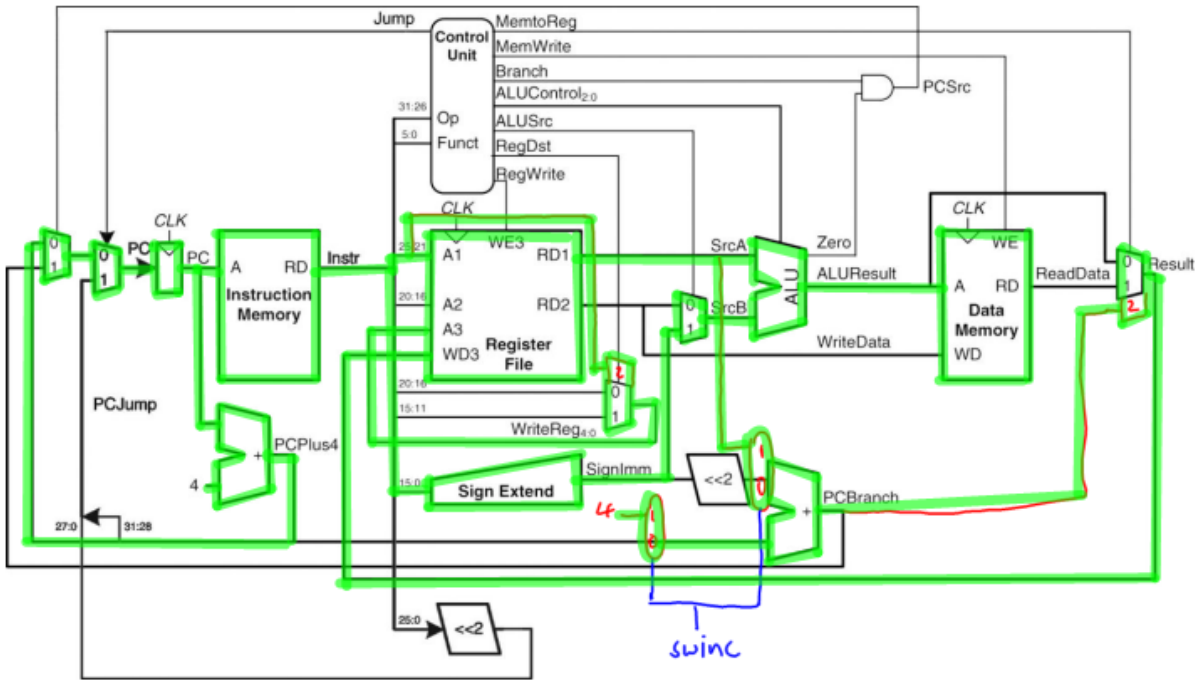
swinc Instruction Walk-through



- This instruction performs the standard sw instruction and additionally increments Reg[rs] by 4. All of this is performed in one instruction -> one clock cycle
  - Compound instructions such as this one is commonly found in CISC instruction sets
- The sw portion of the instruction is unchanged. No datapath modifications are required for this half of the instruction.
- For the second half of the instruction, the value in the rs register must be incremented by 4. The only existing adder in the datapath which is not used already is the PCBranch adder. By adding multiplexors to the input of this adder we can add RD1 and the value 4 together.
- The result, Reg[rs] +4, needs to be written back to the register file. To do so, the output of the PCBranch adder must go into the MemtoReg mux as a new input. The control of this mux is expanded from 1-bit to 2-bits. This provides a path for the data to the Write data input (WD3) of the register file.
- To store the result into the rs register, the 5-bits of rs must be connected to the Write register line (WriteReg) of the register file. As rs is not a current option, the RegDst mux must be expanded to take the 5-bit rs value. RegDst is expanded and the control becomes 2-bits.
  - Remember, when a control is expanded, it must be expanded for all existing instructions)
- The required changes to the datapath are shown in red



- The highlighted areas represent the instruction operation



- The RegDest & MemtoReg controls must be expanded for all instructions. A new control for the added multiplexors is needed (swinc) and the existing control signals must be defined for the new instruction

Instr	RegDst	RegWrite	ALUSrc	Mem Read	Mem Write	MemtoReg	Branch	ALUOp	Jump	Swinc
lw	00	1	1	1	0	01	0	00	0	X
sw	XX	0	1	0	1	XX	0	00	0	X
R-type	01	1	0	0	0	00	0	10	0	X
beq	XX	0	0	0	0	XX	1	01	0	0
Jump	XX	0	X	0	0	XX	X	XX	1	X
swinc	10	1	1	0	1	10	0	00	0	1

Critical Path of Single Cycle Datapath

Single Cycle Datapath Critical Path

- We must wait for everything (all signals) to propogate & the correct values to be calculated
  - Ex: ALU might not produce “right answer” right away
- We use write signals along with clock to determine when to write (falling edge of clock cycle)
- Clock cycle time determined by length of the longest path in the datapath
- Cycle time can be calculated based on the critical timing of each component.
- Ex: Consider the single cycle datapath and the following component delays:
  - Instruction and data memory (200ps)
  - ALU and adders (100ps)
  - register file access, read and write (50ps)
  - Assume all other units are negligible (muxes, control unit, shifters, etc)

Instr Class	(simplified) Functional Units used by instruction class					Total
R-type	Fetch 200ps	Regs (R) 50ps	ALU 100ps	Regs (W) 50ps		400ps
lw	Fetch 200ps	Regs (R) 50ps	ALU 100ps	Mem 200ps	Regs (W) 50ps	600ps
sw	Fetch 200ps	Regs (R) 50ps	ALU 100ps	Mem 200ps		550ps
branch	Fetch 200ps	Regs (R) 50ps	ALU 100ps			350ps
jump	Fetch 200ps					200ps

  - Critical path: load instruction (requires the longest time to execute, uses the most components in sequence)
  - Instruction memory → register file → ALU → data memory → register file
- Because each instruction will take a single clock cycle, all instructions must wait the length of time of the Load instruction.
  - It is not feasible to vary period for different instructions
  - This violates the MIPS design principle: Making the common case fast

