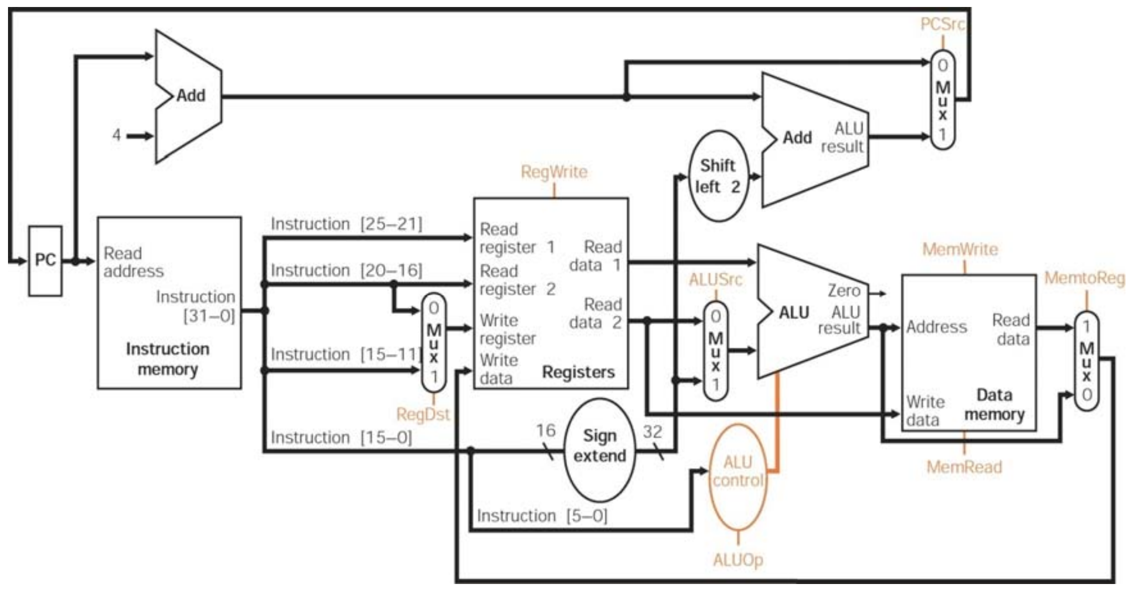
Chapter Four – I (2/4)

Step 4: Given Datapath: RTL → Control

- **Necessary Control Lines**



Designing the Main Control Unit

	31-26	25-21	20-16	15-11	10-6	5-0
R	op	rs	rt	rd	shamt	funct
I	op	rs	rt	16 bit address		
J	op	26 bit address				

- **Observations**
  - **Op field is always contained in bits 31-26**
  - **R-type, Branch, Store → two registers (rs and rt) needs to be read**
  - **Base register for Load, Store → rs (bits 25-21)**
  - **16-bit offset for Load, Store, Branch → bits 15-0**
  - **Destination register**
    - For Load, bits 20-16 (rt)
    - For R-type, bits 15-11 (rd)

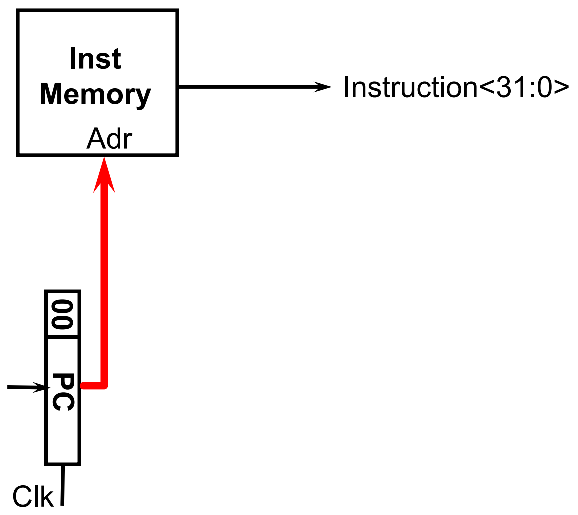
R-Type: The Add Instruction

31 26	21	16	11	6	0
op	**rs**	**rt**	**rd**	**shamt**	**funct**
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits

- **add rd, rs, rt**
  - **imem[PC]** **Read an instruction**
    - Fetch the instruction from the instruction memory
  - **R[rd] ← R[rs] + R[rt]** **Execute the actual operation**
    - Read two operands from the register file
    - Add two operands
    - Store the result to the register file
  - **PC ← PC + 4** **Set the next instruction's address**
    - Add the program counter and 4
    - Store the result to the program counter

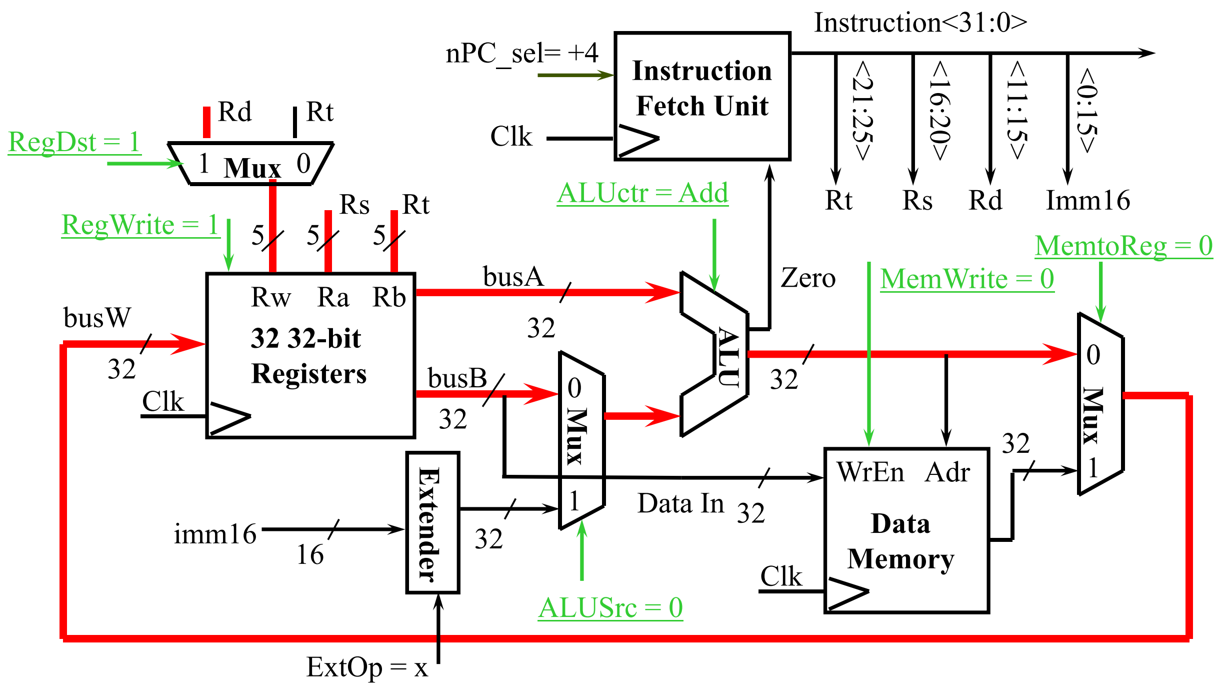
Instruction Fetch Unit at the Beginning of Add

- Fetch the instruction from Instruction memory:
  - Instruction = mem[PC]



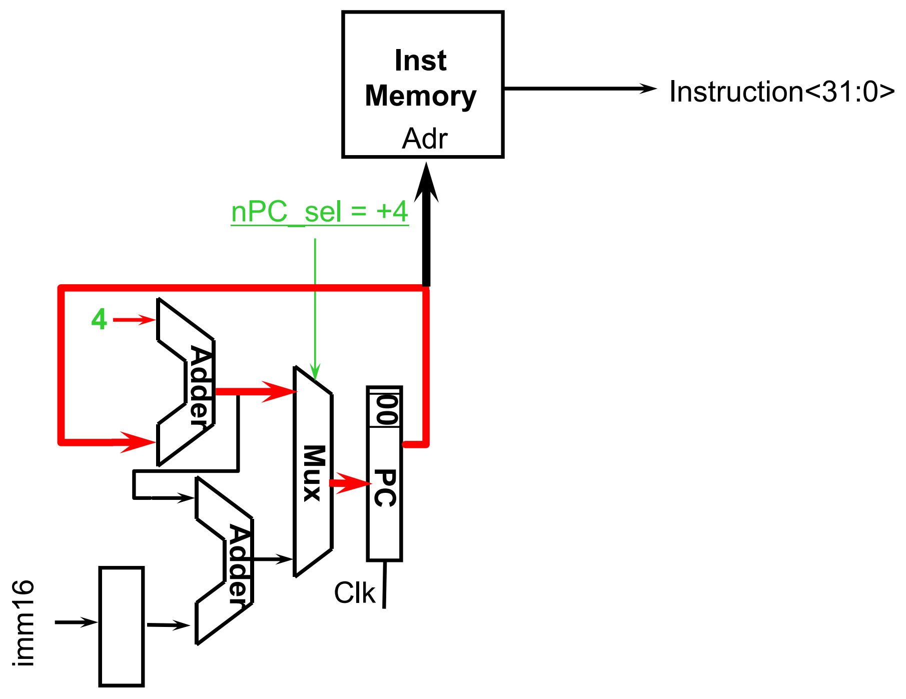
The Single Cycle Datapath during Add

- R[rd] ← R[rs] op R[rt]



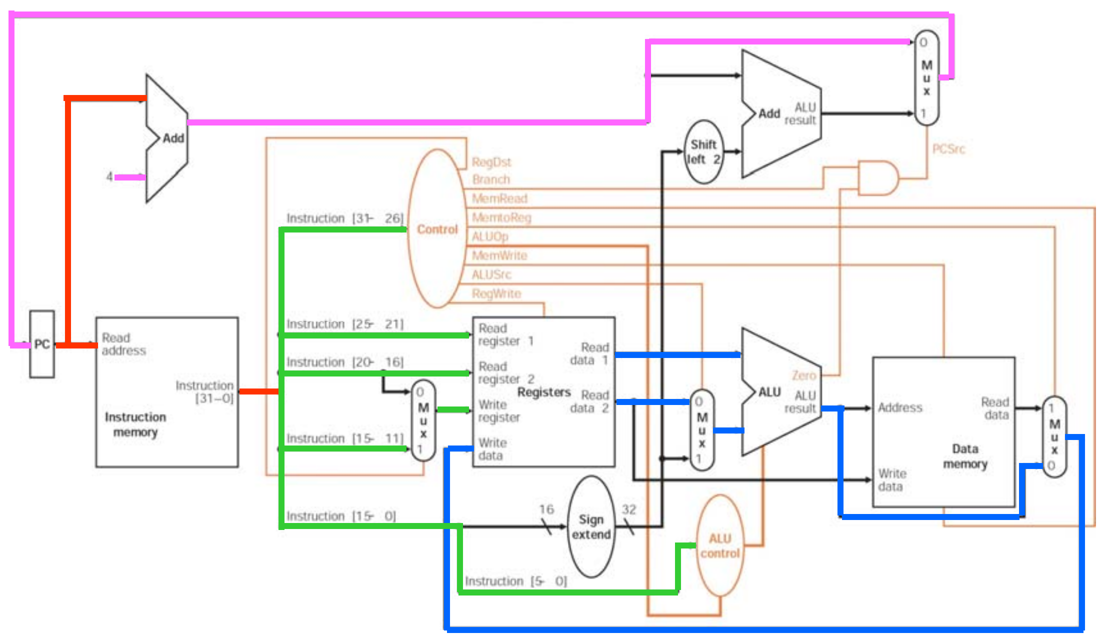
Instruction Fetch Unit at the End of Add

- **PC ← PC + 4**



R-type Instruction

- **Control for R-type instructions**



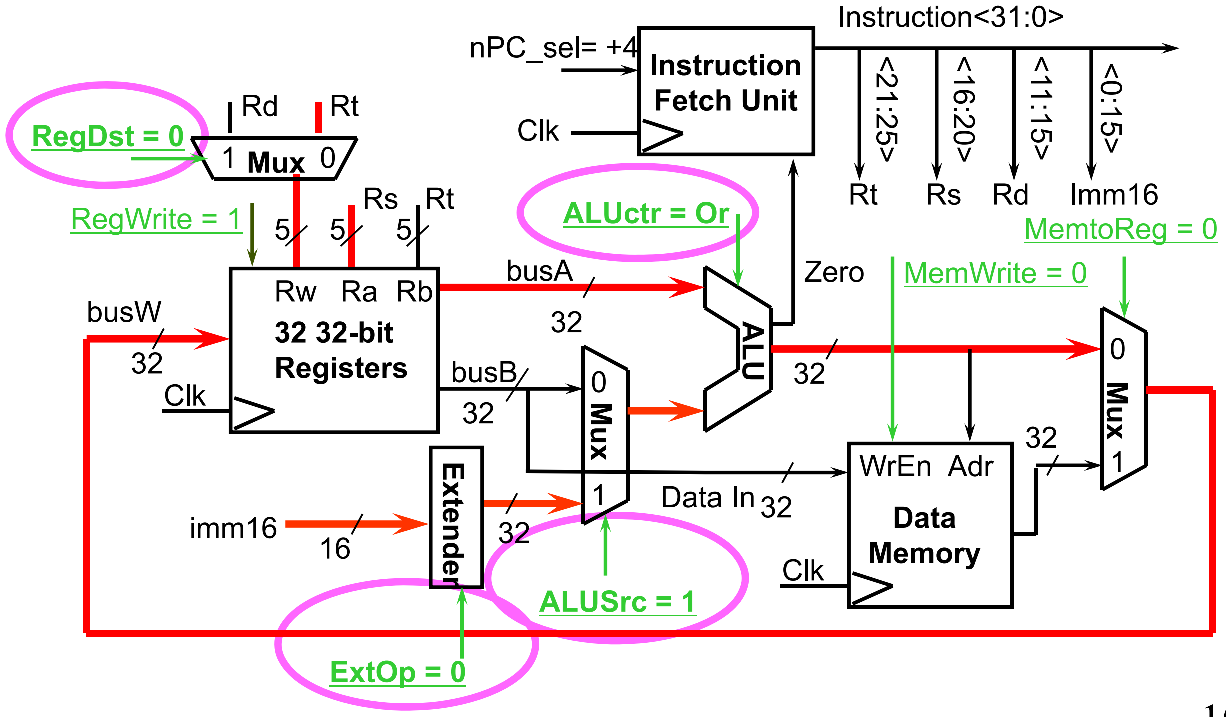
I-Type: The `Or immediate` Instruction

31 26	21	16	0
op	**rs**	**rt**	**immediate**

- **ori rt, rs, imm16**
  - **imem[PC]**
  - **R[rt] ← R[rs] | ZeroExt(imm16)**
    - Read an operand from the register file
    - Or the operand and the zero extended immediate value
    - Store the result to the register file
  - **PC ← PC + 4**

The Single Cycle Datapath for `Or Immediate`

- **R[rt] ← R[rs] or ZeroExt(Imm16)**



I-Type: The Load, Store Instruction

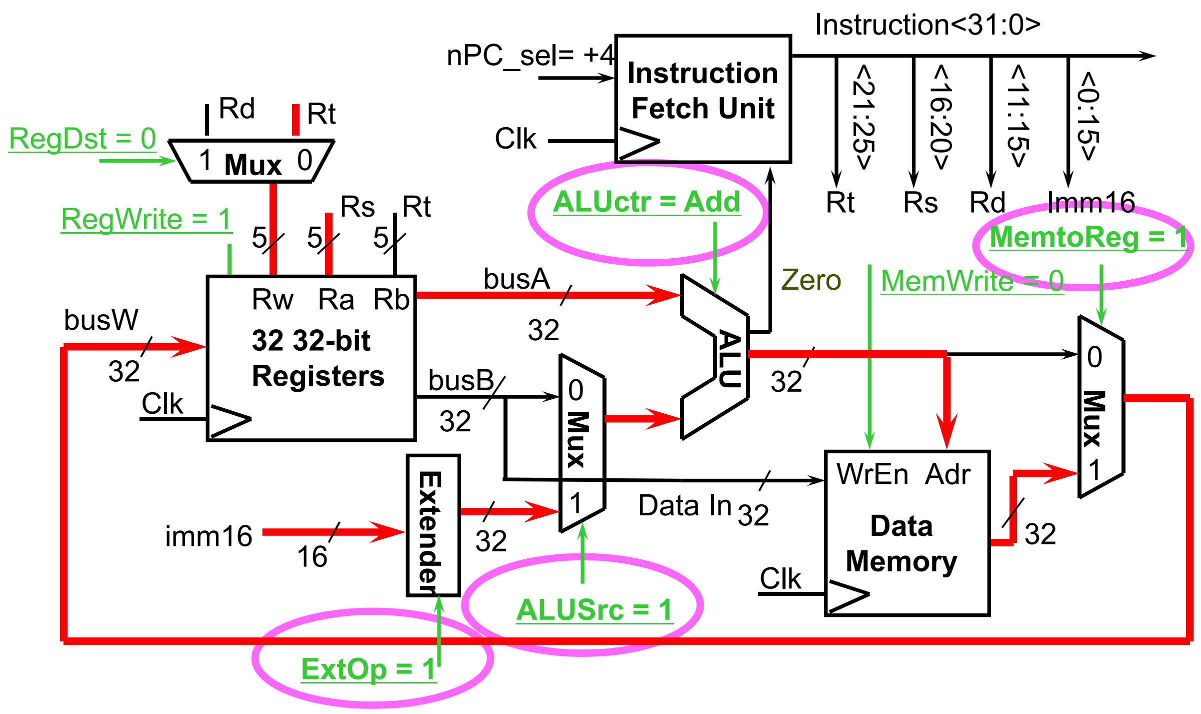
31	26	21	16	0
op	rs	rt	immediate	

- **lw rt, imm16(rs)**
  - **imem[PC]**
  - **R[rt] ← dmem[R[rs] + SignExt(imm16)]**
    - Read an operand from the register file
    - Add the operand and the immediate value
    - Read a datum from the data memory with the result address
    - Store the datum to the register file
  - **PC ← PC + 4**

- **sw rt, imm16(rs)**
  - **dmem[R[rs] + SignExt(imm16)] ← R[rt]**
    - Read two operands from the register file
    - Add one operand and the immediate value
    - Store the other operand to the data memory with the result address

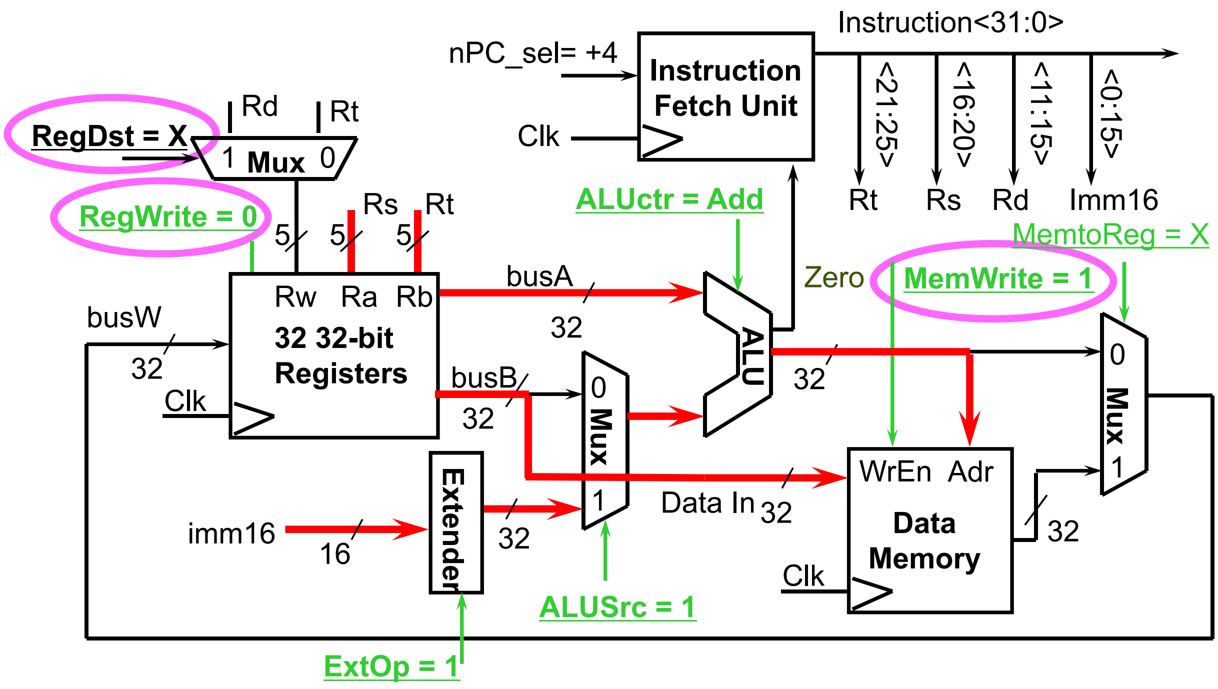
The Single Cycle Datapath for Load

- **R[rt] ← DMem[R[rs] + SignExt[imm16]]**



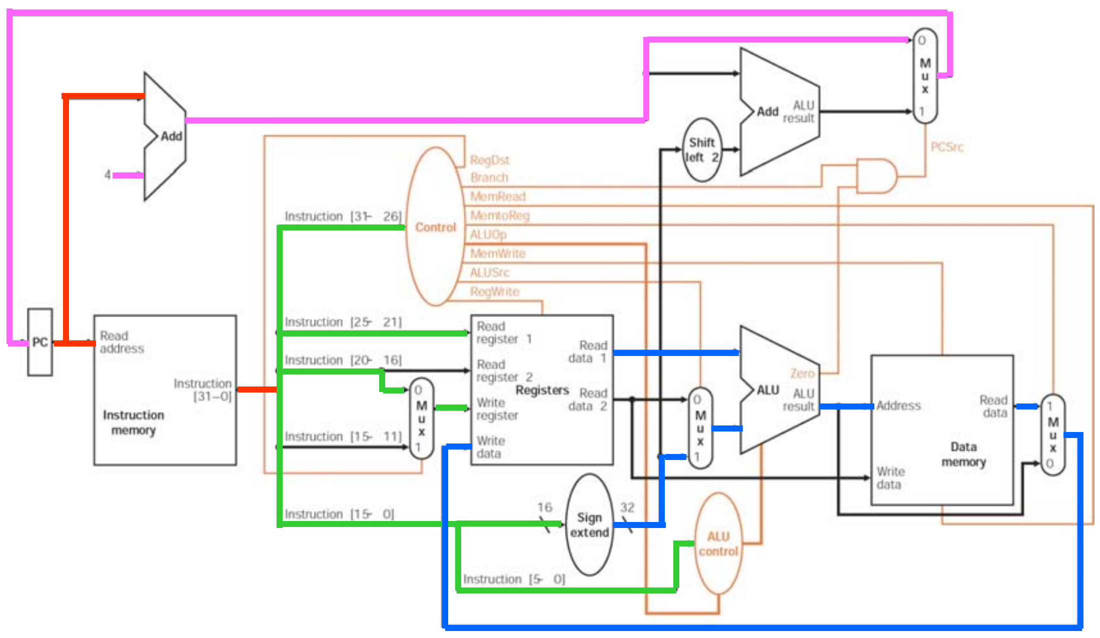
The Single Cycle Datapath for `Store`

- **dmem[R[rs] + SignExt(imm16)] ← R[rt]**



Load Instruction

- **Control for I-type Load instructions**



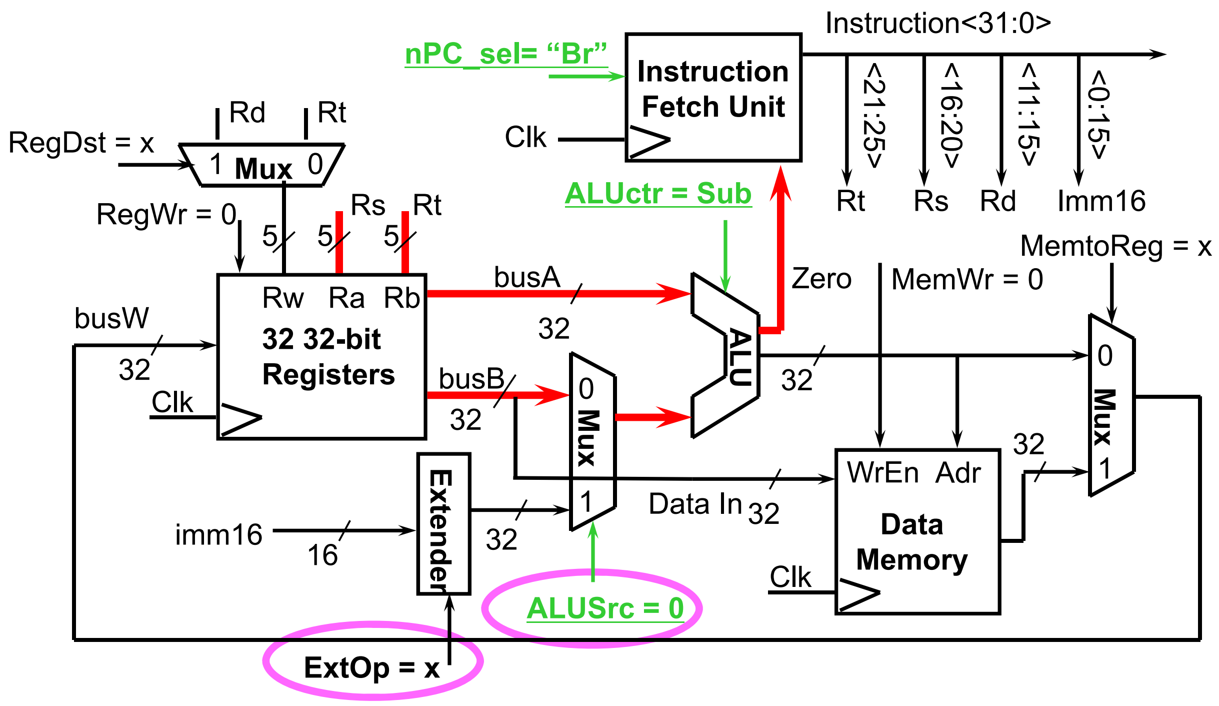
I-Type: The Branch Instruction

31	26	21	16	0
op	rs	rt	immediate	

- **beq rs, rt, imm16**
  - **imem[PC]** **Read an instruction**
  - **Zero = R[rs] == R[rt]** **Execute the actual operation**
    - Read an operand from the register file
    - Subtract one operand from the other
  - **Baddr = PC + 4 + SignExt(imm16)** **Calculate branch address**
    - Add PC and the immediate value
  - **PC ← PC + 4 or Baddr** **Set the next instruction's address**

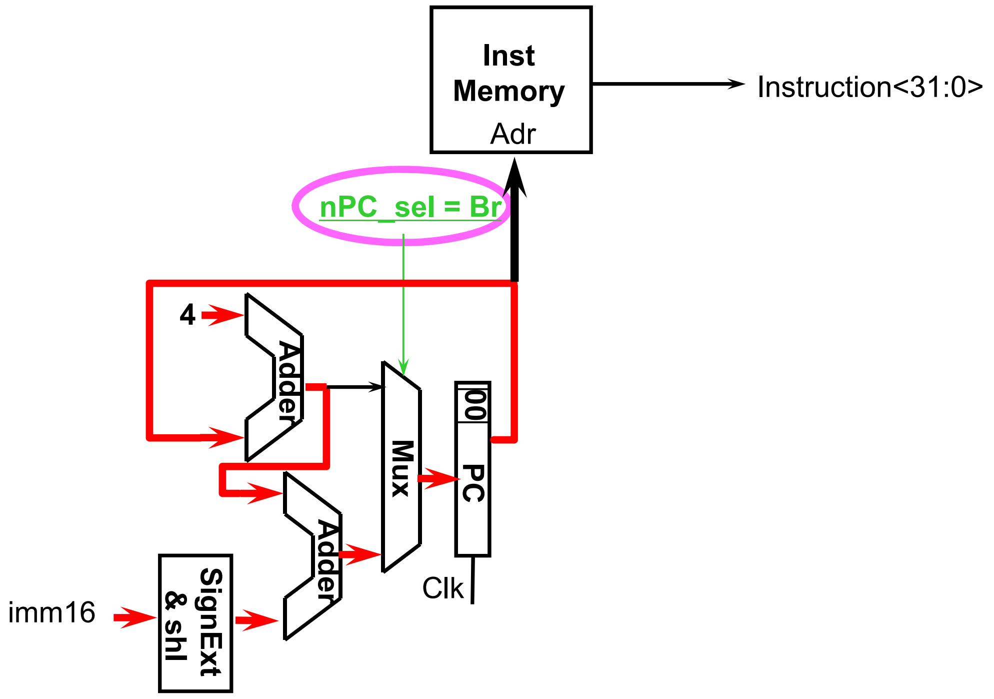
The Single Cycle Datapath for Branch

- **if (R[rs] - R[rt] == 0) then Zero = 1 ; else Zero = 0**



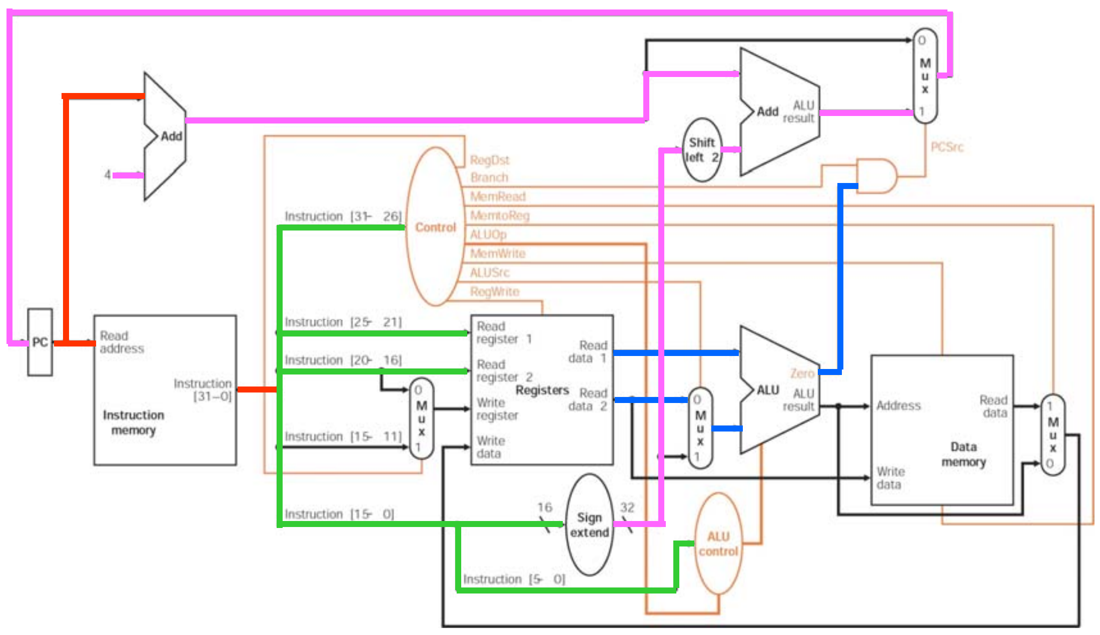
Instruction Fetch Unit at the End of Branch

- if (Zero==1) then PC ← PC + 4 + SignExt(imm16)<<2; else PC ← PC + 4



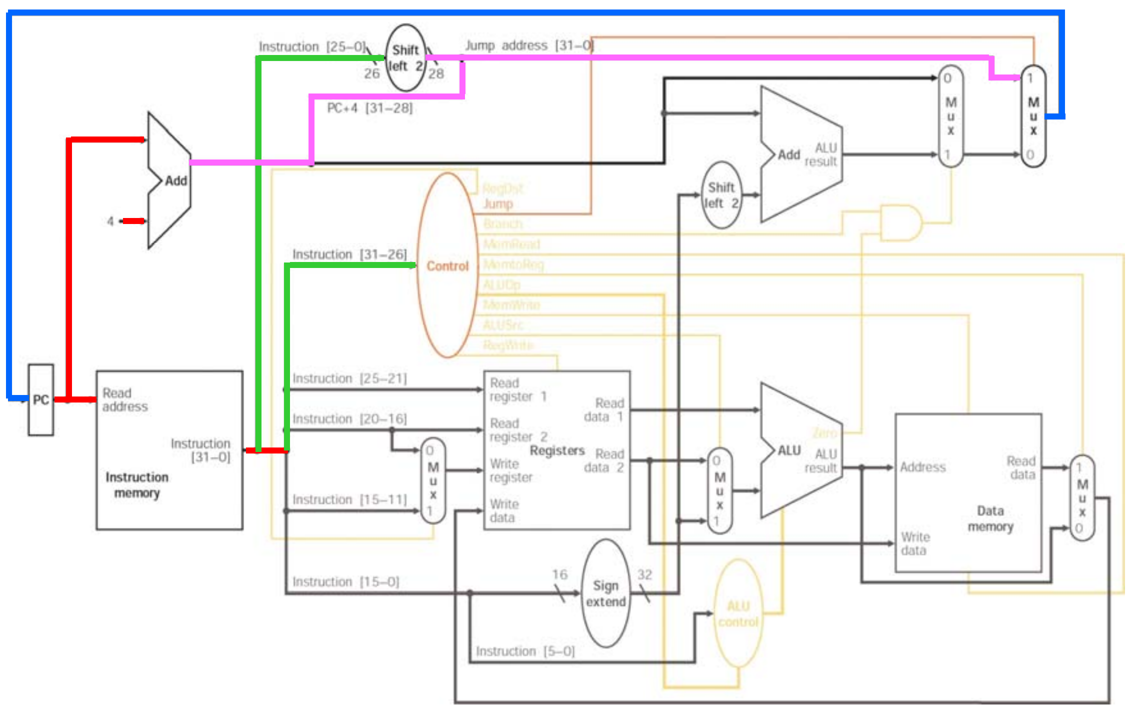
BEQ Instruction

- Control for I-type BEQ instructions

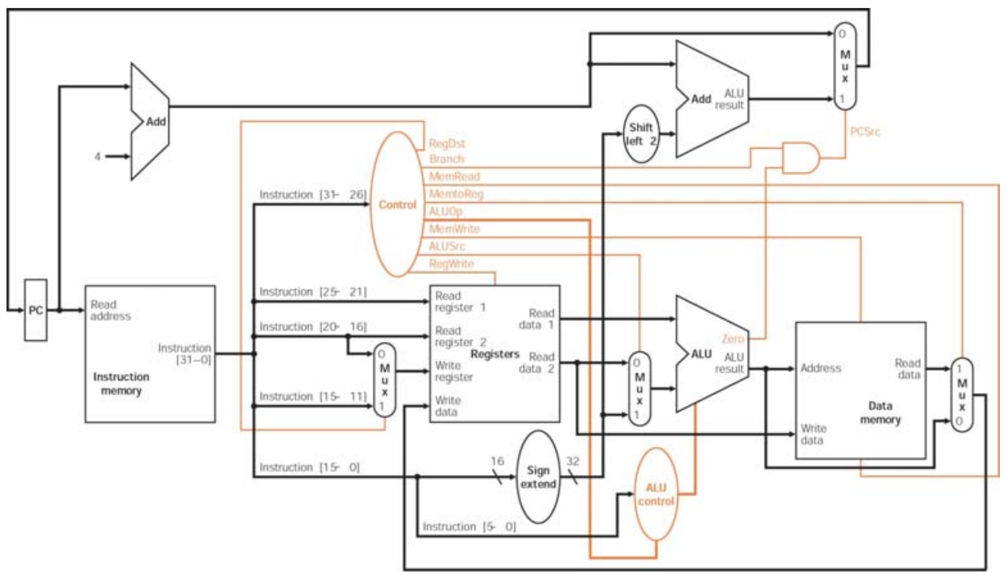


Jump Instruction

- PC ← "PC+4[31:28]";"imm26<<2"



Simple Datapath with the Control Unit

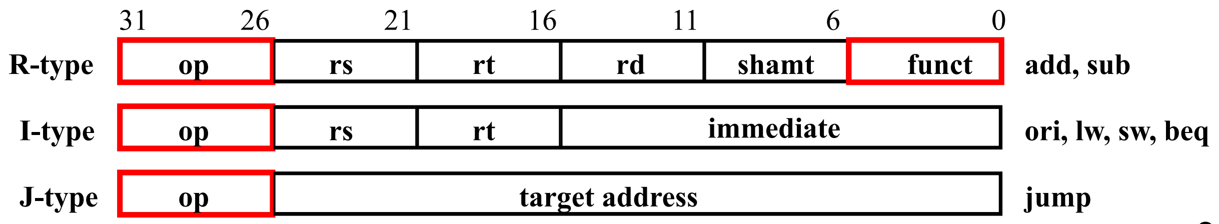


Instr	RegDst	ALUSrc	MemtoReg	RegWrite	MemRead	MemWrite	Branch	ALUOp1	ALUOp2
R	1	0	0	1	0	0	0	1	0
Load	0	1	1	1	1	0	0	0	0
Store	X	1	X	0	0	1	0	0	0
Beq	X	0	X	0	0	0	1	0	1

A Summary of the Control Signals

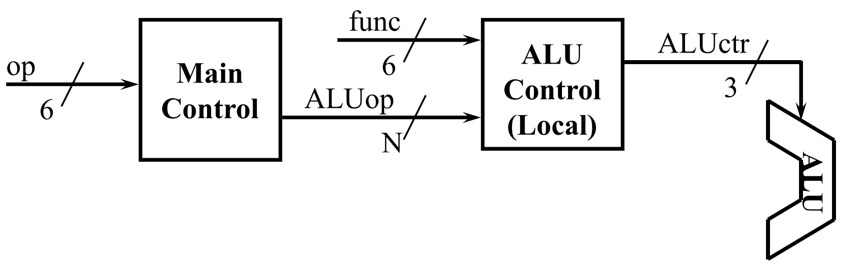
See Appendix A

	add	sub	ori	lw	sw	beq	jump
func	10 0000	10 0010	**We Don't Care :-)**				
op	00 0000	00 0000	00 1101	10 0011	10 1011	00 0100	00 0010
RegDst	1	1	0	0	x	x	x
ALUSrc	0	0	1	1	1	0	x
MemtoReg	0	0	0	1	x	x	x
RegWrite	1	1	1	1	0	0	0
MemWrite	0	0	0	0	1	0	0
nPCsel	0	0	0	0	0	1	0
Jump	0	0	0	0	0	0	1
ExtOp	x	x	0	1	1	x	x
ALUctr<2:0>	Add	Subtract	Or	Add	Add	Subtract	xxx



The Concept of Local Decoding

	R-type	ori	lw	sw	beq	jump
op	00 0000	00 1101	10 0011	10 1011	00 0100	00 0010
RegDst	1	0	0	x	x	x
ALUSrc	0	1	1	1	0	x
MemtoReg	0	0	1	x	x	x
RegWrite	1	1	1	0	0	0
MemWrite	0	0	0	1	0	0
Branch	0	0	0	0	1	0
Jump	0	0	0	0	0	1
ExtOp	x	0	1	1	x	x
ALUop<N:0>	"R-type"	Or	Add	Add	Subtract	xxx



Opcode & ALU Operation Mapping

Name	Opcode (Hex)	Opcode (Bin)					
		Op5	Op4	Op3	Op2	Op1	Op0
R-format	0	0	0	0	0	0	0
lw	35	1	0	0	0	1	1
sw	43	1	0	1	0	1	1
beq	4	0	0	0	1	0	0

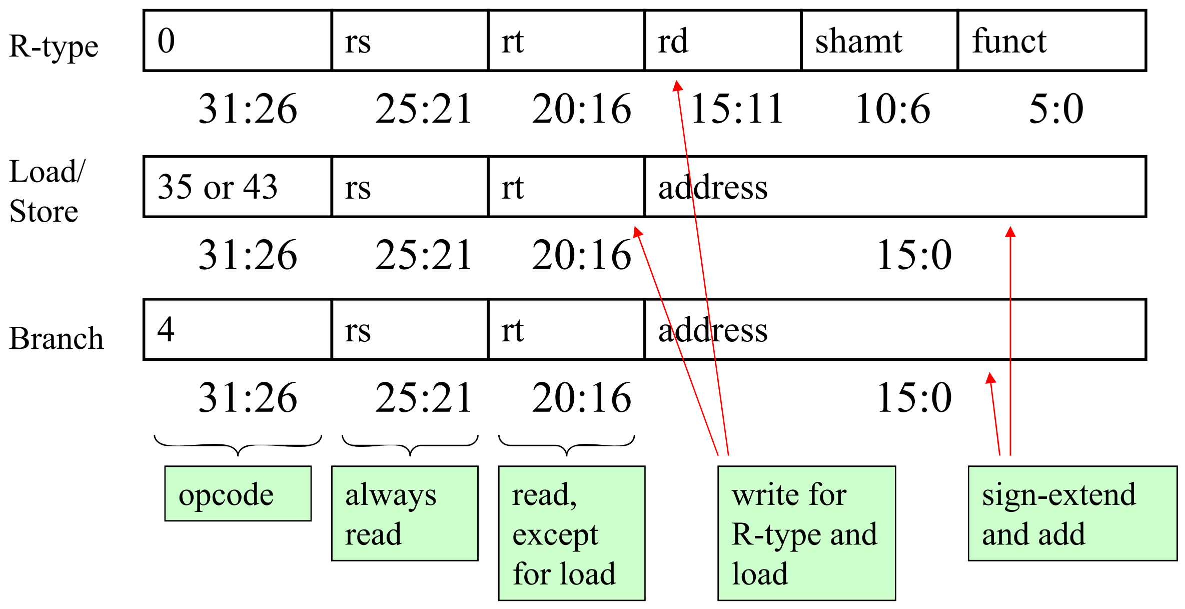
Op	Operation	ALUOp	Funct	ALU action	ALUctr
LW	Load word	000	XXXXXX	Add	010
SW	Store word	000	XXXXXX	Add	010
BEQ	Branch equal	001	XXXXXX	Sub	110
R-type	add	100	100000	Add	010
R-type	sub	100	100010	Sub	110
R-type	AND	100	100100	AND	000
R-type	OR	100	100101	OR	001
R-type	slt	100	101010	Slt	111

Control with Opcode

Input or Output	Signal Name	R-format	lw	sw	beq
Inputs	Op5	0	1	1	0
	Op4	0	0	0	0
	Op3	0	0	1	0
	Op2	0	0	0	1
	Op1	0	1	1	0
	Op0	0	1	1	0
Outputs	RegDst	1	0	X	X
	ALUSrc	0	1	1	0
	MemtoReg	0	1	X	X
	RegWrite	1	1	0	0
	MemRead	0	1	0	0
	MemWrite	0	0	1	0
	Branch	0	0	0	1
	ALUOp3	1	0	0	0
	ALUOp2	0	0	0	0
	ALUOp1	0	0	0	1

Control Signals from Instructions

- Control signals derived from instruction



Logic for each control signal

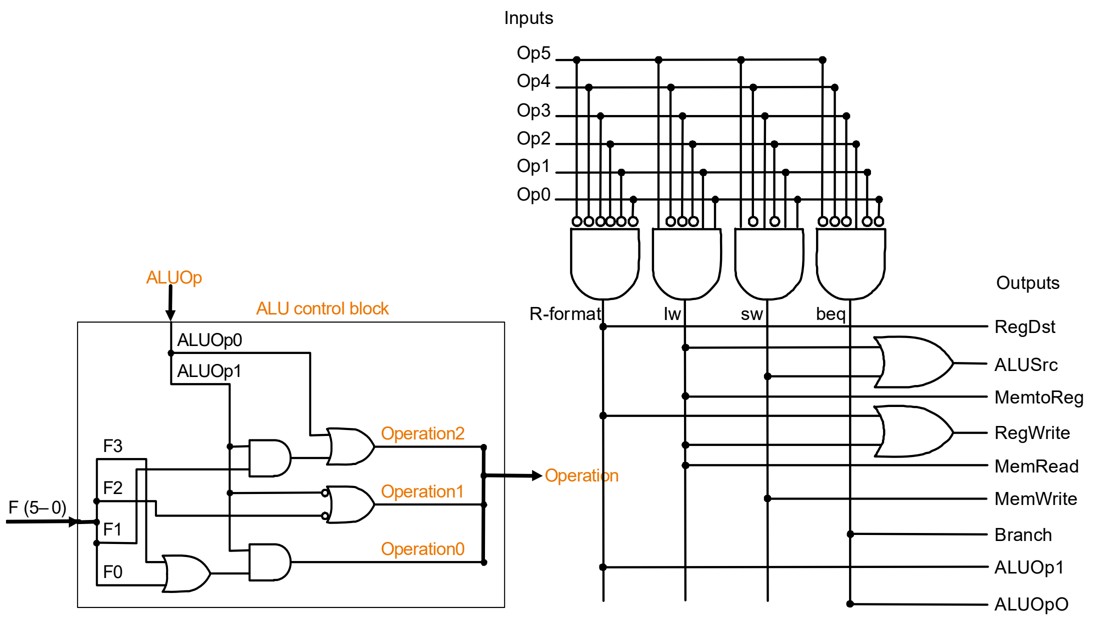
• nPC_sel	=	if (OP == BEQ) then EQUAL else 0
• ALUsrc	=	if (OP == “000000”) then “regB” else “immed”

• ALUctr	=	if (OP == “000000”) then funct
		elseif (OP == ORi) then “OR”
		elseif (OP == BEQ) then “sub”
		else “add”

• MemWr	=	(OP == Store)
• MemtoReg	=	(OP == Load)
• RegWr:	=	if ((OP == Store) || (OP == BEQ)) then 0 else 1
• RegDst:	=	if ((OP == Load) || (OP == ORi)) then 0 else 1

Control Logic

- Simple combinational logic (truth tables)



Summary

- 5 steps to design a processor
  - 1. Analyze instruction set => datapath requirements
  - 2. Select set of datapath components & establish clock methodology
  - 3. Design datapath meeting the requirements
  - 4. Analyze implementation of each instruction to determine setting of control points that effects the register transfer.
  - 5. Design the control logic
- MIPS makes it easier
  - Instructions same size
  - Source registers always in same place
  - Immediates same size, location
  - Operations always on registers/immediates
- Single cycle datapath ➔ CPI=1, CCT ➔ long