Laboratory 5
Processor Datapath

Description of Mini-MIPS Architecture

16 bit data bus

8 bit address bus

Starting address of every program = 0 (PC initialized to 0 by a reset to begin execution)

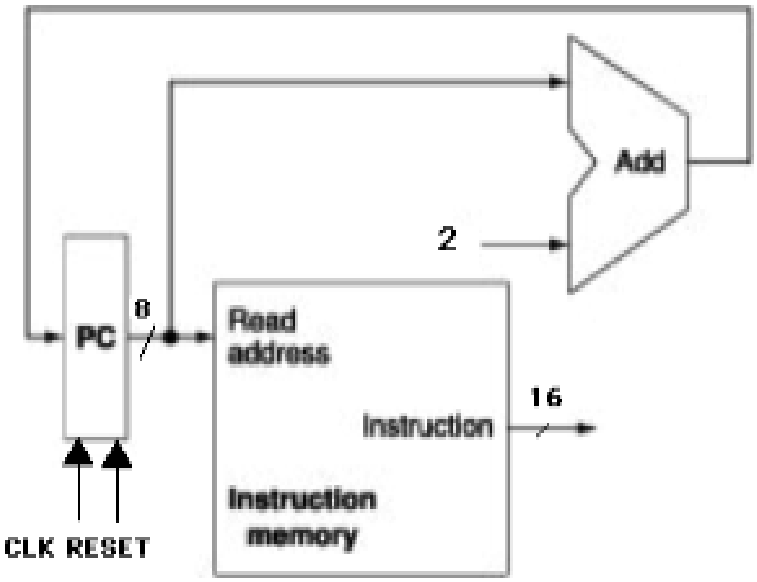
PC incremented by 2 to move to the next instruction.

16 registers
R0 = 0 (constant)
R1 =1 (constant)
R2-R15 general purpose

Instruction Set

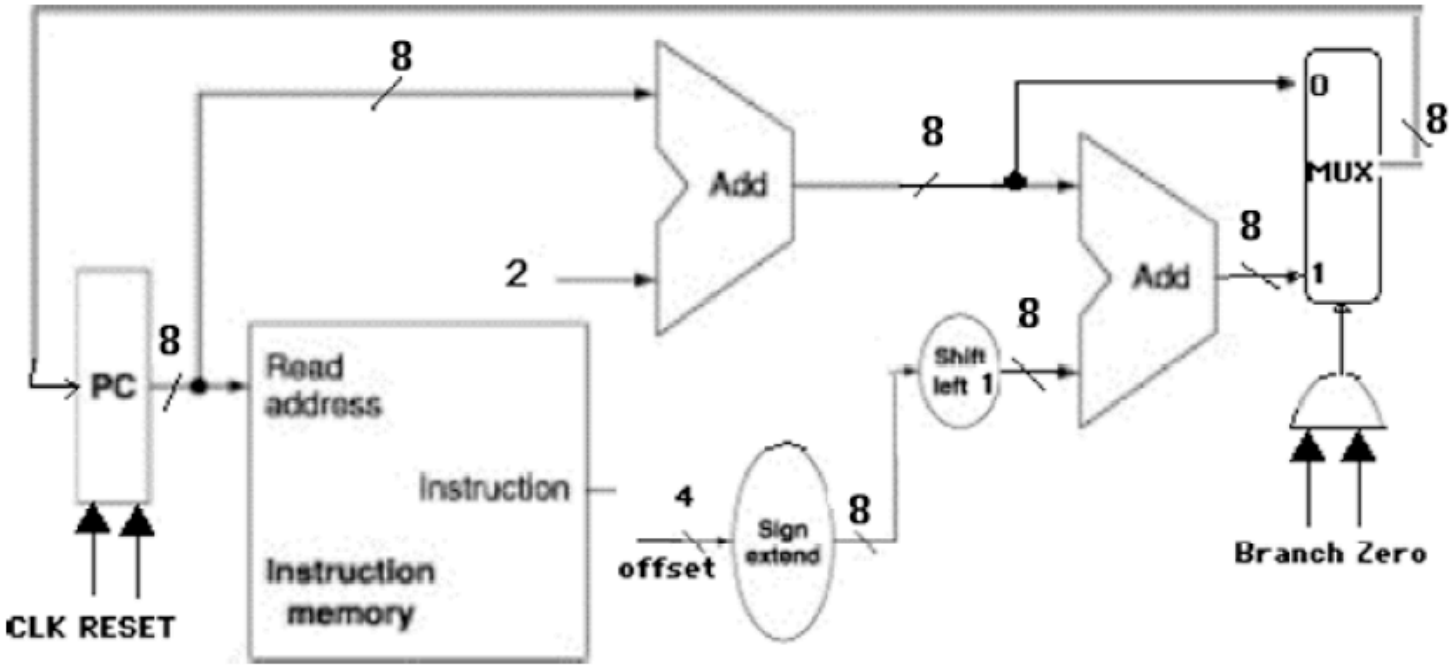
Instruction	**Meaning**	**Op** 4-bit	**Rs** 4-bit	**Rt** 4-bit	**Rd** 4-bit
LW Rs,Rt,offset	Rt loaded with word from Data Memory at address(Rs + offset)	0000	0-15	0-15	offset
SW Rs,Rt,offset	Data Memory address(Rs + offset) stored with word from Rt	0001	0-15	0-15	offset
ADD Rs,Rt,Rd	Rd := Rs + Rt	0010	0-15	0-15	0-15
SUB Rs,Rt,Rd	Rd := Rs - Rt	0011	0-15	0-15	0-15
AND Rs,Rt,Rd	Rd := Rs AND Rt	0100	0-15	0-15	0-15
OR Rs,Rt,Rd	Rd := Rs OR Rt	0101	0-15	0-15	0-15
SLT Rs,Rt,Rd	If Rs<Rt then Rd:=1 else Rd := 0	0110	0-15	0-15	0-15
BEQ Rs,Rt,offset	If Rs=Rt then pc:=pc+2+(offset*2) else pc:=pc+2	0111	0-15	0-15	offset
JMP offset	Jump to abs. addr = offset*2	1000	---12 bit offset-----		

Instruction Fetch



Branch Address

Either **PC + 2** or **PC + 2 + (2*offset)** is the next value of the PC.



On a BEQ instruction, **BEQ Rs,Rt,offset**

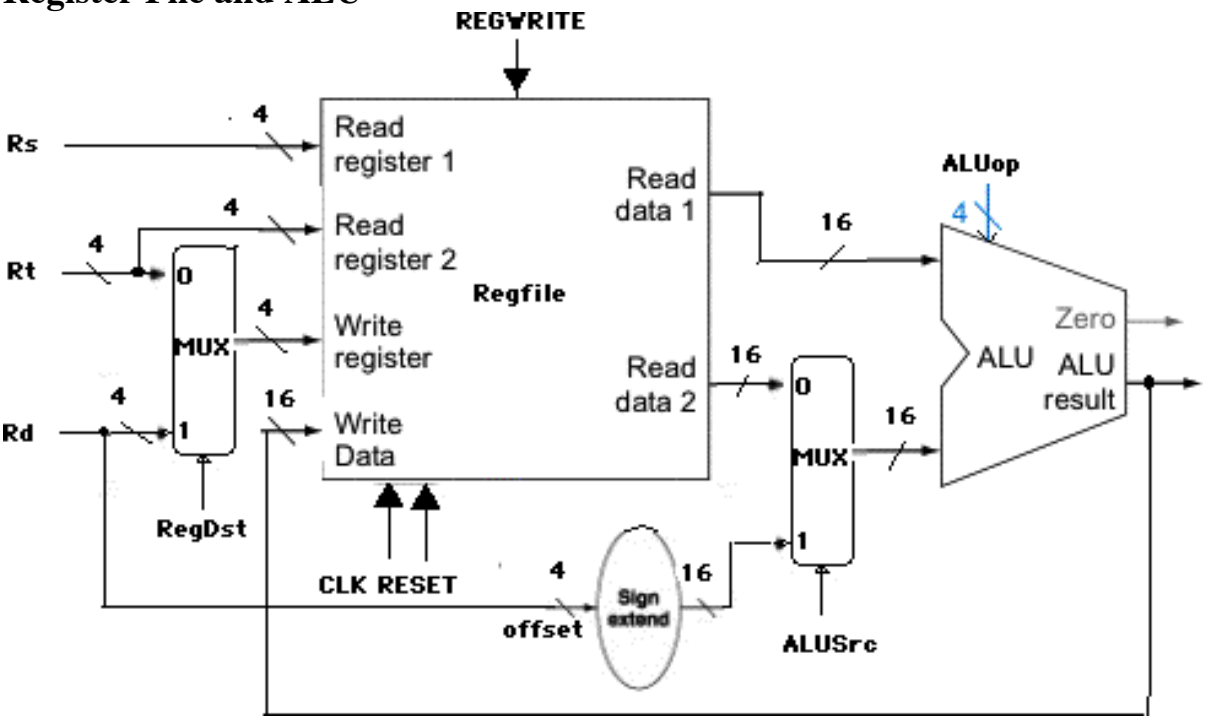
- The **offset** = number of instructions away from the next value of the PC to branch to, so must be multiplied by 2.
- Since **offset** is 4 bits, it must be sign-extended to 8 bits to be added to the PC.

A **2x8 multiplexer** circuit to selects the next value of the PC. The value of the **Branch** and **Zero** bits are used to determine which is used:

- The **Branch** control line = 1 if a BEQ instruction is being executed.

- The **Zero** bit from the ALU is used to check whether Rs = Rt: it is 1 if Rs – Rt = 0 (meaning they're equal). If **Branch** = 1 and **Zero** = 1, then the next value of the PC will be the branch address ; otherwise, it will simply be PC + 2:

Register File and ALU



R-type instructions ADD,SUB,AND,OR,SLT (opcode Rs Rt Rd)

- read Rs and Rt from register file
- perform an ALU operation on the contents of the registers
- write the result to register Rd in register file

Memory Access instructions LW,SW (opcode Rs Rt offset)

- memory address = Rs + sign-extended 4-bit offset
- if **SW**, the value to be stored to memory is from Rt.
- if **LW**, Rt is loaded with the value read from memory

Register written to (**Write Register**) is Rd **or** Rt if a **LW** instruction (chosen by a 2x4 MUX which is controlled by **RegDst**)

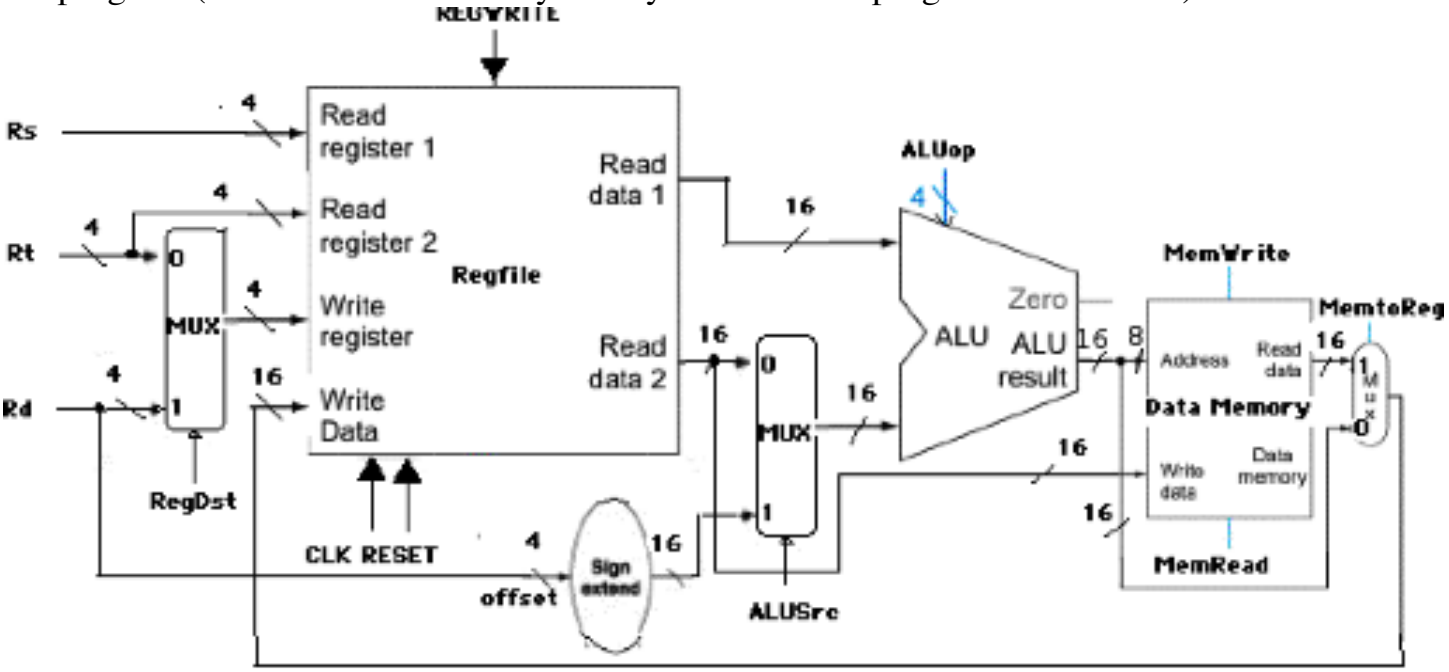
ALU calculates Rs + Rt, **or** Rs + sign-extended offset.

- Input A of the ALU is always Rs

- Input B of the ALU is Rt **or** the offset (chosen by a 2x16 multiplexer, which is controlled by **ALUSrc**):

Data Memory

We need an additional memory for values loaded or stored (**LW** or **SW**) during execution of the program (the instruction memory is only used to store program instructions).



RegDst (chooses whether Rd or Rt goes to the **Write data** input on the Regfile)

If 0, destination is Rd. If 1, destination is Rt.

RegWr (control line to RegFile)

If 1, writes the value on the **Write data** input to the register specified by **Write register**

ALUSrc (chooses the source of the second ALU operand)

If 0, the operand is the second register file output.

If 1, the operand is the sign-extended, lowest 4 bits of the instruction.

MemRd (control signal to data memory)

If 0, value stored at address in data memory is read from **Read data**.

MemWr (control signal to data memory)

If 0, data memory address written with value from **Rt** on the **Write data** input.

MemtoReg (chooses the value to be written back to the Regfile)

If 0, the value comes from the ALU (**R-type instruction**)

If 1, the value comes from data memory (**LW**)

ALUop(**4 bits**)	ALU function
0	a AND b
1	a OR b
2	a + b (add)
6	a - b
7	set on less than

Control Logic for the ALU

ALU can perform 5 possible operations:

ALUop	ALU function
0	a AND b
1	a OR b
2	a + b (add)
6	a - b
7	set on less than

Need an ALU Control Unit to select the proper operation for each instruction:

Instruction	Opcode	ALU operation	ALUop
LW	0	add	2
SW	1	add	2
ADD	2	add	2
SUB	3	sub	6
AND	4	and	0
OR	5	or	1
SLT	6	slt	7
BEQ	7	sub	6
JMP	8	don't care	don't care

Op3	Op2	Op1	Op0	ALUop3	ALUop2	ALUop1	ALUop0
0	0	0	0	0	0	1	0
0	0	0	1	0	0	1	0
0	0	1	0	0	0	1	0
0	0	1	1	0	1	1	0
0	1	0	0	0	0	0	0
0	1	0	1	0	0	0	1
0	1	1	0	0	1	1	1
0	1	1	1	0	1	1	0

Use a 3x8 decoder to produce the ALUop

Control Unit

Must provide control signals for all other devices in datapath (MUXs, Regfile, Data Memory)

Instruction	Opcode	RegDst	RegWr	ALUSrc	MemRd	MemWr	MemtoReg
LW	0000	1	1	1	0	1	1
SW	0001	1	0	1	1	0	0
ADD	0010	0	1	0	1	1	0
SUB	0011	0	1	0	1	1	0
AND	0100	0	1	0	1	1	0
OR	0101	0	1	0	1	1	0
SLT	0110	0	1	0	1	1	0
BEQ	0111	0	0	0	1	1	0
JMP	1000	0	0	0	1	1	0

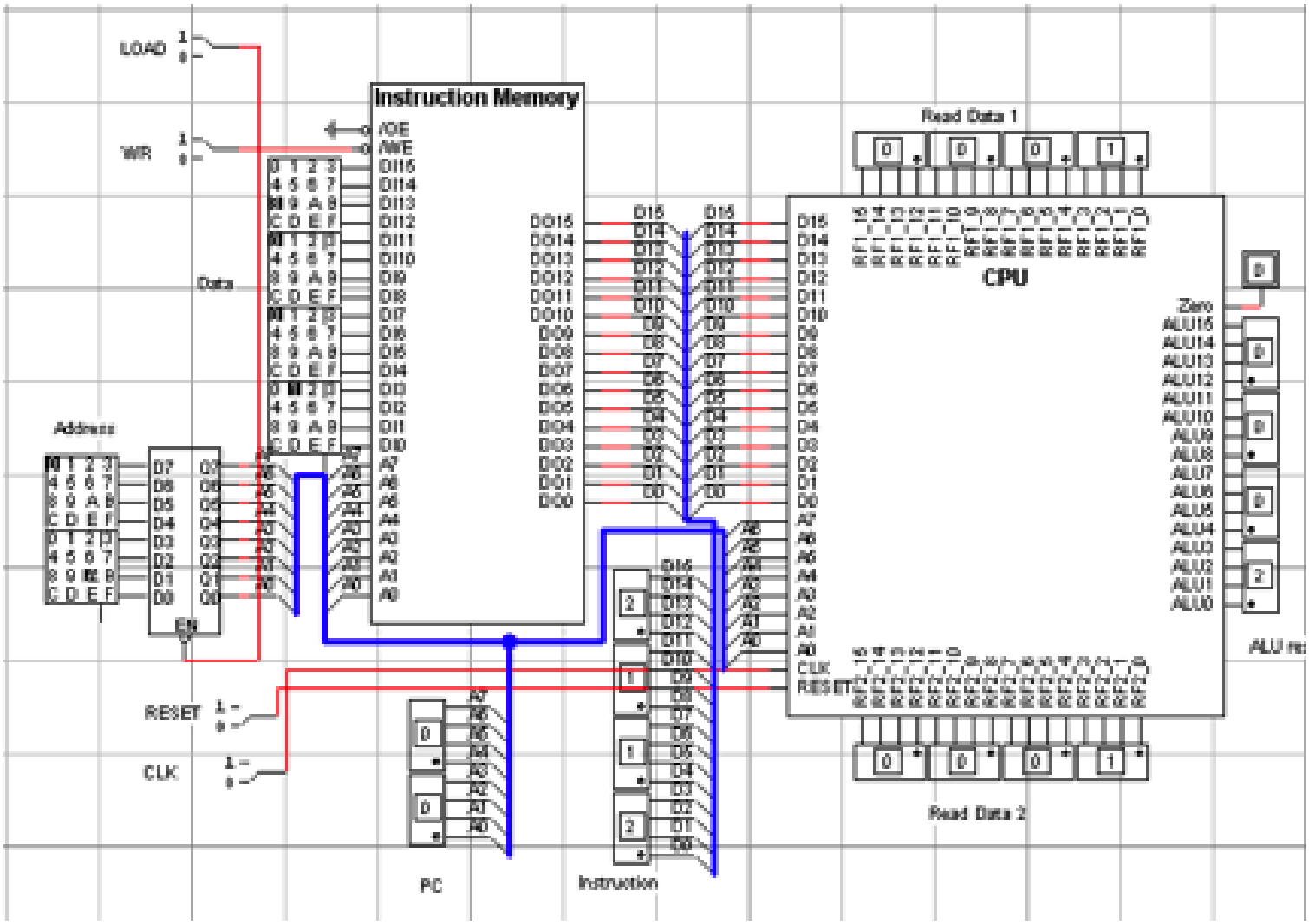
Can produce with logic gates or a 4x16 decoder (two 3x8 decoders)

Programming the Single-Cycle CPU

Mini-MIPS program which loops repeatedly to access memory:

Address	Instruction	Meaning	
0:	5002	OR R0 R0 R2	#R2 gets 0
2:	5003	OR R0 R0 R3	#R3 gets 0
4:	1220	SW R2 R2 0	#address n: gets n (start of loop!)
6:	0230	LW R2 R3 0	#R3 gets n
8:	2122	ADD R1 R2 R2	#R2 gets R2 + 1
A:	8002	JUMP 002	#jump to 2*2 (address 4) = beginning of loop

Full Implementation



Procedure to Load/Execute a New Program

1. Disconnect the address bus of the Instruction Memory from the CPU
2. Set **LOAD** = 0

3. Set **address** and **data** switches for instruction
4. Set **WR** = 0, then back to 1
5. Repeat steps 3 and 4 until all instructions are loaded to memory

6. Set **LOAD** = 1
7. Reconnect address bus to CPU

8. Set **Reset** = 1, then back to 0
9. Set **CLK** = 1, then back to 0, for each instruction.