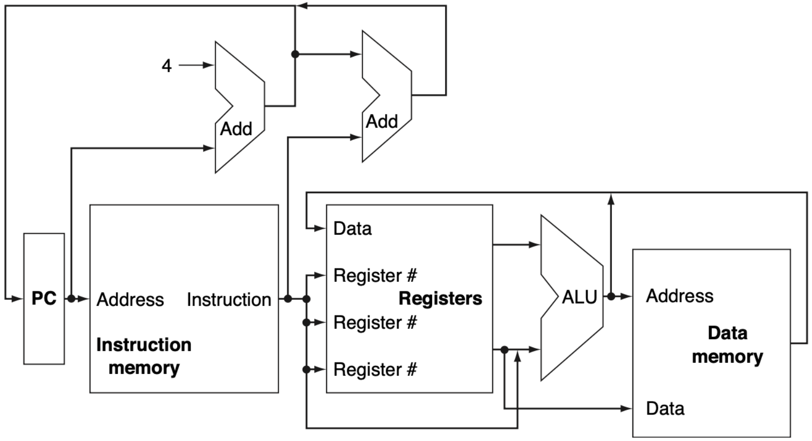
# Computer Architecture CSL3020

Deepak Mishra

http://home.iitj.ac.in/∼dmishra/
Department of Computer Science and Engineering
Indian Institute of Technology Jodhpur

# Processor

An abstract view

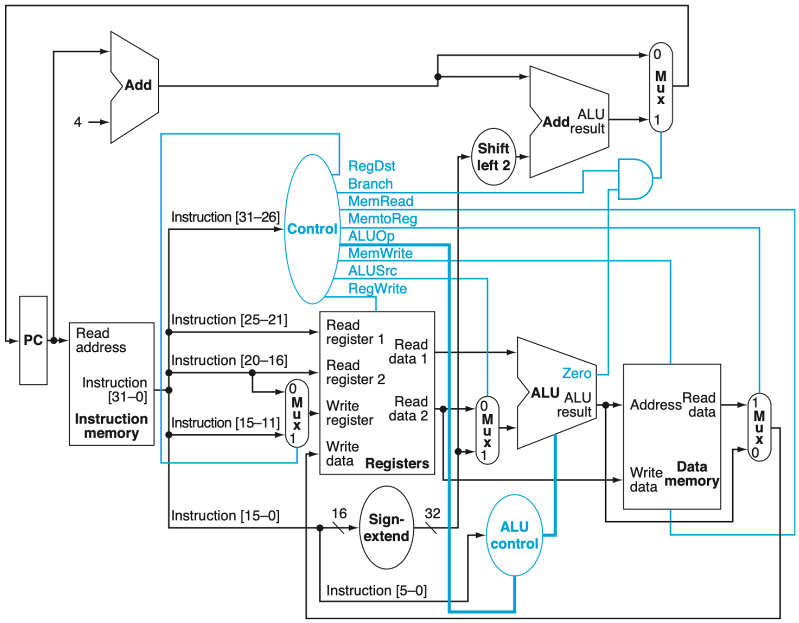


## Processor

An abstract view

- All instructions start by using the PC to supply instruction address.
- After the instruction is fetched, the register operands are specified.
- Operands then can be operated on to compute a memory address, an arithmetic result, or a comparison.
- The result from ALU or memory is written back into the register file.
- Branches use ALU to determine the next instruction address.
- The thick lines interconnecting the functional units represent buses, which consist of multiple signals.

# A Simple MIPS Processor



## Functional Units

Two types of functional units:

- *Combinational:* Elements that operate on data values; output is function of current input only
- *Sequential:* Elements that contain state (memory); output is function of current and previous inputs

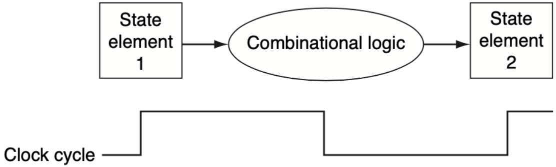
## Functional Units

Two types of functional units:

- *Combinational:* Elements that operate on data values; output is function of current input only
- *Sequential:* Elements that contain state (memory); output is function of current and previous inputs

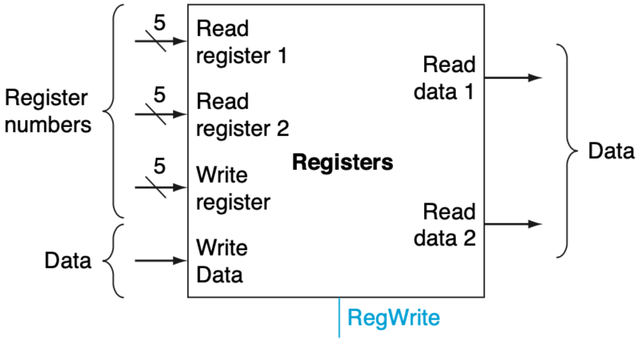
Clocked and timing

- Clocking methodology defines when signals can be read and written
- We will assume an edge-triggered clocking methodology.



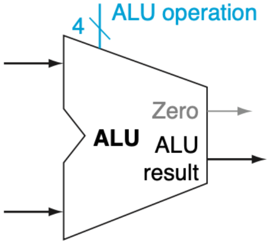
## A Simple MIPS Processor - RF

- The RF (register file) contains all the registers and has two read ports and one write port.
- A register write must be explicitly indicated by asserting the write control signal.



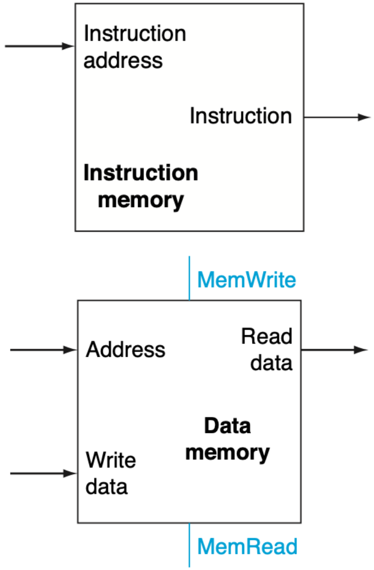
## A Simple MIPS Processor - ALU

- Operation to be performed by the ALU is controlled with the ALU operation signal.
- Zero detection output is used to implement branches.



## A Simple MIPS Processor - Memories

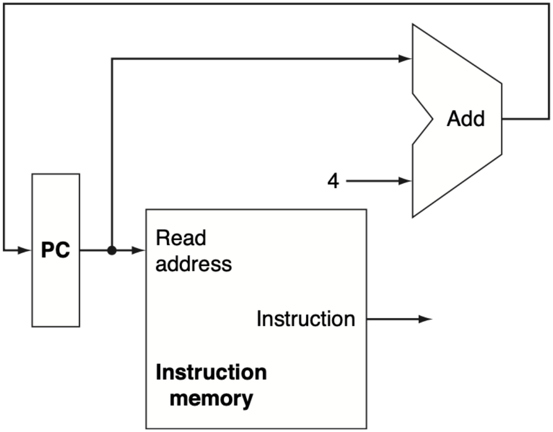
- We assume that the instruction memory need to provide only read access because the datapath does not write instructions.
- In contrast, the data memory needs to provide both read and write access with control signals.



## A Simple MIPS Processor - MIPS ISA Subset

- Arithmetic - logic instructions
    - add, sub, AND, OR, slt
- Memory reference instructions
    - lw, sw
- Branching instructions
    - beq, j

## A Simple MIPS Processor - Instruction Fetch

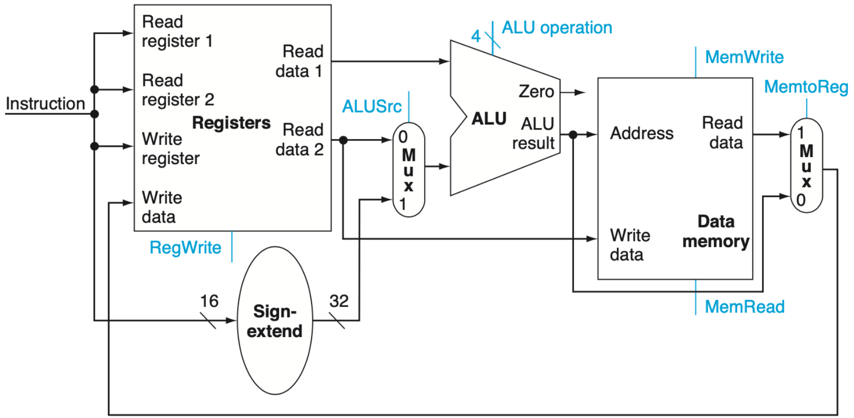


## A Simple MIPS Processor

Implementing R-type and memory instructions

- Arithmetic and logical instructions use ALU with the inputs coming from the two registers.
- Memory instructions also use ALU for address calculation using sign-extended 16-bit offset.
- The value stored into a destination register comes from ALU (for an R-type instruction) or memory (for a load).

# A Simple MIPS Processor - R-type



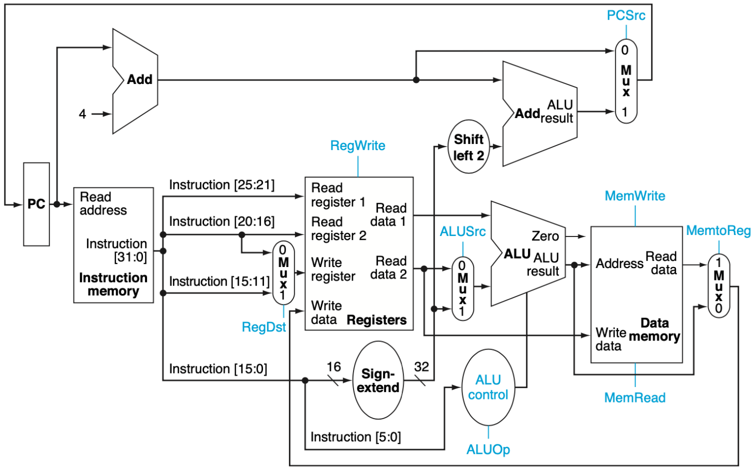
## A Simple MIPS Processor

Implementing branch instruction (beq)

- The branch datapath must do two operations – compute the branch target address and compare the register contents.
- It needs an ALU to evaluate the branch condition and a separate adder to compute the branch target as the sum of the incremented PC and the sign-extended offset.
- In sign extension, the lower 16 bits of the instruction are first extended to 32 bits and then shifted left 2 bits for word alignment.

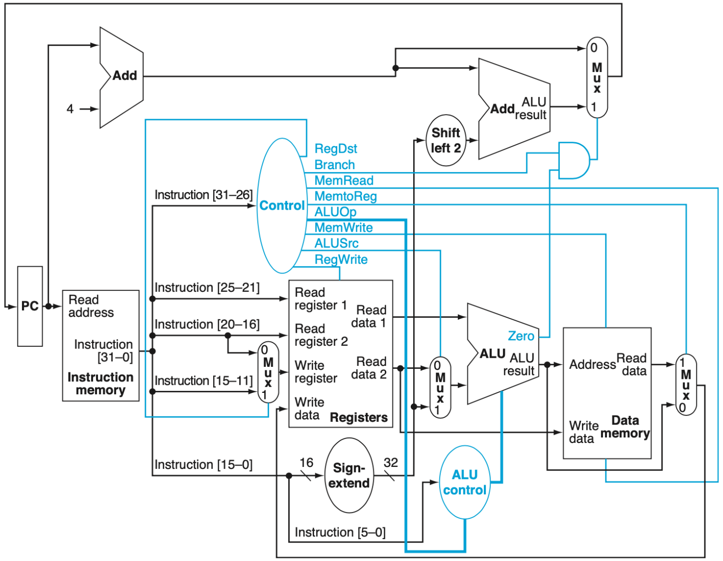
## A Simple MIPS Processor - Branching

Putting it all together



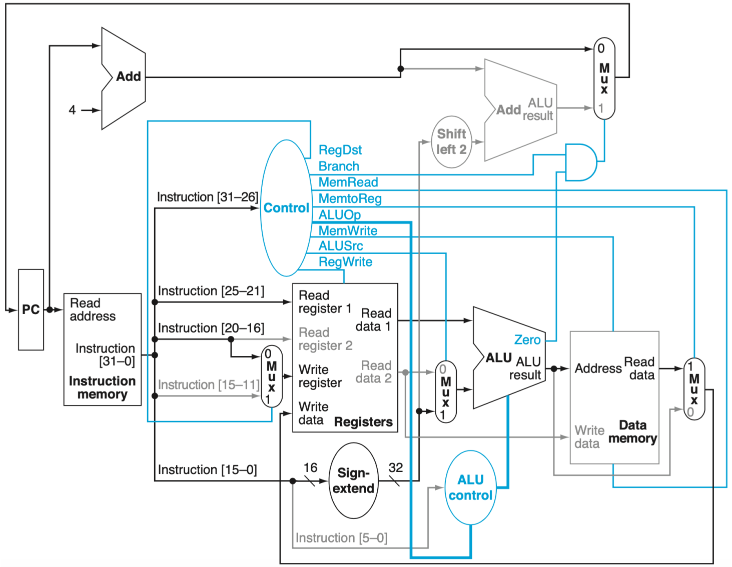
# A Simple MIPS Processor

Putting it all together with control



# A Simple MIPS Processor - I-type operation

Operation for an I-type instruction – lw $t1, 100($t2)



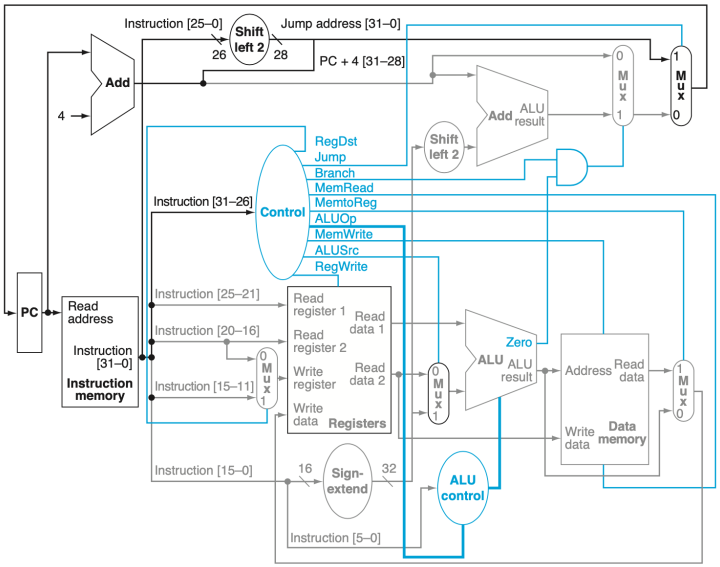
## A Simple MIPS Processor

Extending to include the jump instruction

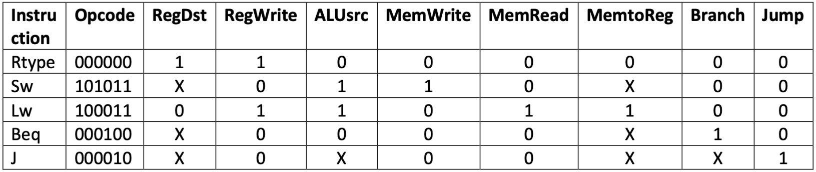
- We can implement a jump by storing into the PC the concatenation of
  - the upper 4 bits of the current PC + 4 (these are bits 31:28 of the sequentially following instruction address)
  - the 26-bit immediate field of the jump instruction
  - the bits 00
- An additional multiplexer is required to choose between the jump target and the branch target.

# A Simple MIPS Processor - Jump

Including the jump instruction



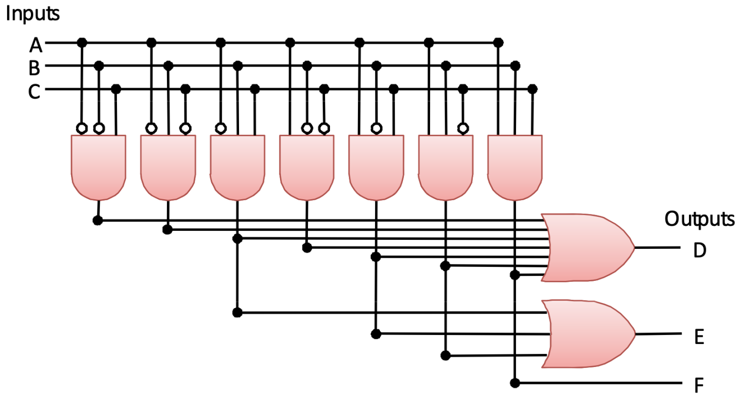
# A Simple MIPS Processor - Control



| Instruction | Opcode | RegDst | RegWrite | ALUsrc | MemWrite | MemRead | MemtoReg | Branch | Jump |
|---|---|---|---|---|---|---|---|---|---|
| Rtype | 000000 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| Sw | 101011 | X | 0 | 1 | 1 | 0 | X | 0 | 0 |
| Lw | 100011 | 0 | 1 | 1 | 0 | 1 | 1 | 0 | 0 |
| Beq | 000100 | X | 0 | 0 | 0 | 0 | X | 1 | 0 |
| J | 000010 | X | 0 | X | 0 | 0 | X | X | 1 |

# A Simple MIPS Processor - Control

| Instruction | Opcode | RegDst | RegWrite | ALUsrc | MemWrite | MemRead | MemtoReg | Branch | Jump |
|---|---|---|---|---|---|---|---|---|---|
| Rtype | 000000 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| Sw | 101011 | X | 0 | 1 | 1 | 0 | X | 0 | 0 |
| Lw | 100011 | 0 | 1 | 1 | 0 | 1 | 1 | 0 | 0 |
| Beq | 000100 | X | 0 | 0 | 0 | 0 | X | 1 | 0 |
| J | 000010 | X | 0 | X | 0 | 0 | X | X | 1 |



## Hardwired vs. Microprogrammed Control

| **Hardwired** | **Microprogrammed** |
|---|---|
| Generates control signals using logic circuits | Generates control signals using micro instructions stored in control memory |
| Faster and costlier | Slower but less costly |
| Difficult to modify | Easy to modify as the modification need to be done only at the instruction level |
| Used in RISC, cannot handle complex instructions | Used in CISC |

# A Simple MIPS Processor - Control

A few comments on control

- Controls are set using 6-bit opcode field, op ([31:26]), of instruction.
- It is easy to control the operation while using single-cycle implementation.
- However single-cycle design is inefficient as the clock cycle is determined by the longest possible path in the processor.
- Although the CPI is 1, the overall performance of a single-cycle implementation is poor, since the clock cycle is too long.
- An alternative is Pipelining.