Processor Datapath

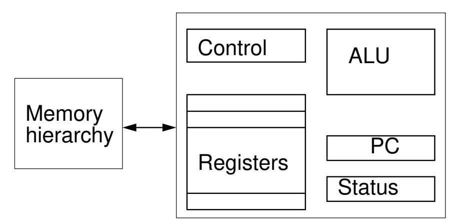
Levels in Processor Design

- We can talk about design at a variety of levels (from low to high):
  - Circuit design: transistors, resistors, capacitors, etc. Building gates, flip-flops, etc.
  - Logic design: putting gates (AND, OR, XOR, etc) and flip-flops together to build blocks such as registers, adders, memory. See CSE370.
  - Register transfer level: describes the execution of instructions by showing how information is transferred and manipulated between adders, registers, memory, etc.
  - Processor description: the ISA.
  - System description: includes memory hierarchy, IO, number of processors, etc.

Register Transfer Perspective

- We'll use either block diagrams or pseudocode to describe the operation/design of a simple processor which implements a subset of the MIPS ISA.
- We'll implement just a subset of the ISA:
  - Memory reference: lw and sw
  - Arithmetic: add, sub, and, or, stli
  - Control: beq, jump
- Key components:
  - Combinational: the output is a function of the inputs (e.g. an adder)
  - Sequential: state is remembered (e.g. a register)

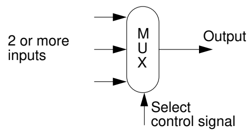
Data Path and Control Unit



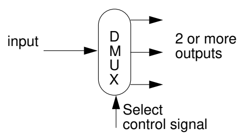
- Data path:
  - Combinational (ALU) + Sequential (Registers, PC, Status)
  - How data moves between components, what operations are performed on data.
- Control unit:
  - Sends signals to data path elements
  - Tells what data to move, where to move it, what ops to perform

Combinational Elements: (De)Multiplexor

- Multiplexor (mux) selects the value of one of its inputs to be routed to the output:

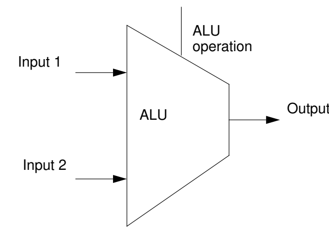


- Demultiplexor routes its input to one of its outputs:



Combinational Elements: ALU

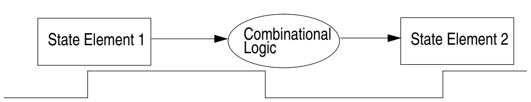
- ALU computes (combinational) output from its two inputs.
- Performs functions needed to execute arithmetic and logical instructions.
- Combinational logic has a “critical path” which determines the length of time needed for the output to stabilize given stable inputs. This is typically around 10ns.



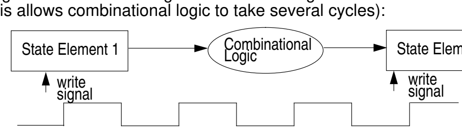
Synchronous Design

- Use a periodic clock, which controls when signals can be read and when they can be written. Values in storage elements can only be updated on clock edges.
- The clock determines when events occur, ie, when signals sent by control unit are obeyed in the datapath.

Changes occur on every clock edge:

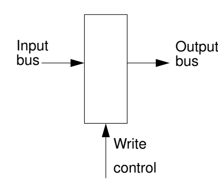


Changes occur on clock edges when a write signal is asserted (this allows combinational logic to take several cycles):



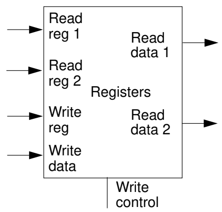
Building Blocks: Storage elements

- The basic building block is the register.
- Our registers store 32 bits.
- A register will only be written on the clock edge AND when the write control line is asserted.
- It can be read and written on the same clock, but the value read will be the OLD value.



Building Blocks: Register File

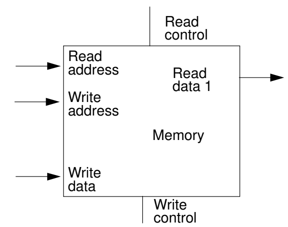
- Register file is an array of registers (32 in MIPS)
- ISA tells us that we should be able to read 2 registers and write 1 register in a given instruction.
- We need to know which registers to read/write, and what data to write.



- Typical access time is under 5ns.

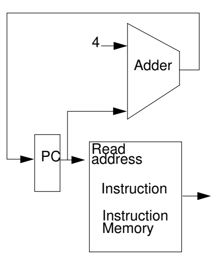
Memory

- Memory is like a register file, but much larger and slower.
- Can only read or write one location per cycle.



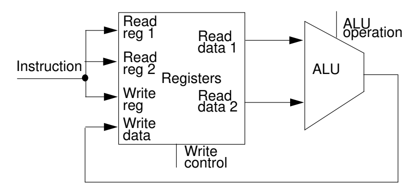
- Typical access time (for primary memory) is around 100ns. For cache memory it is closer to 10ns.

Instruction Fetch Datapath



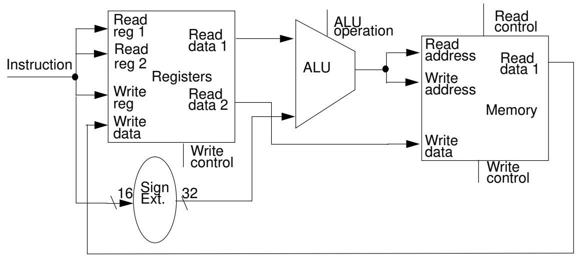
- Our implementation will fully execute one instruction per clock cycle: single cycle implementation.
- The PC tells us the read address.
- On each clock edge, a new value for PC will be latched into the PC register.

Datapath for R-type Instruction



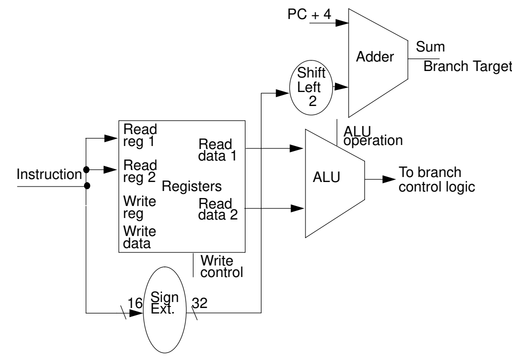
- The instruction bits name the read and write regs (rs, rt, rd).
- On the clock edge, the data is read, which moves through the ALU, hopefully in time to be latched into the write port at the next clock edge.

Datapath for Load/Store



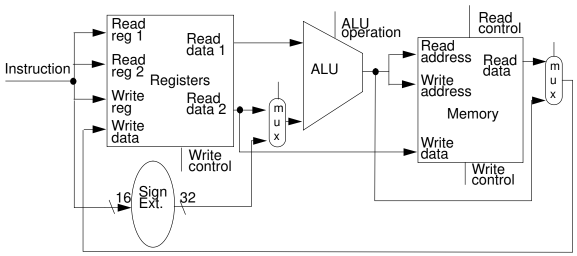
- The instruction bits tell us the registers to use (src/dest and base register) and the 16 bit signed offset.
- We use the ALU to compute the effective address, which is passed along to the data memory.

Datapath For Branch



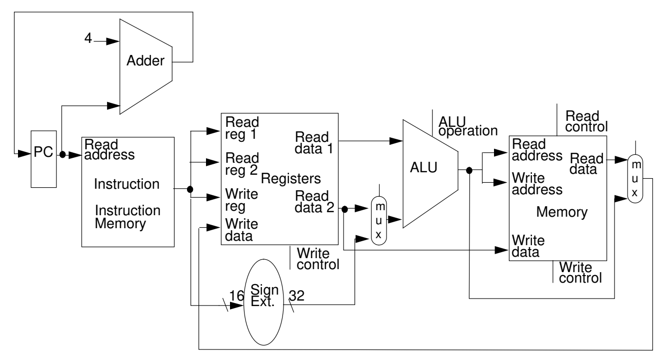
- Question: Why can't we just use the ALU to compute the branch target address?

Combining Memory and R-type



- Note that we add 2 muxes:
  - One to select the second ALU input
  - One to select the source for the register writeback (memory or ALU result)

Adding Instruction Fetch



Full Datapath: Adding Branches