Datapath

- Abstract / Simplified View:

Two types of functional units:
- Combinational logic
- State elements: D-latches and D flip-flops
- Clocking methodology: edge triggered

![Datapath Diagram]

- Falling edge
- Rising edge
- Cycle time
Building the Datapath

Add ALU result

Shift left 2

MemRead

RegWrite

MemWrite

PCSrc

ALUSrc

ALU operation

Zero

Address

Read data

Data memory

MemtoReg

Add data

Write data
Control

- Selecting the operations to perform (ALU, read/write, etc.)
- Controlling the flow of data (multiplexor inputs)
- Information comes from the 32 bits of the instruction
- Example:

  add $8, $17, $18  Instruction Format:

  \[
  \begin{array}{cccccccc}
  000000 & 10001 & 10010 & 01000 & 00000 & 10000 & 00000 & 10000 \\
  \text{op} & \text{rs} & \text{rt} & \text{rd} & \text{shamt} & \text{funct} \\
  \end{array}
  \]

- ALU's operation based on instruction type and function code
Control
Control

- Simple combinational logic (truth tables)
Our Simple Control Structure

• All of the logic is combinational
• We wait for everything to settle down, and the right thing to be done
  – ALU might not produce “right answer” right away
  – we use write signals along with clock to determine when to write
• Cycle time determined by length of the longest path

We are ignoring some details like setup and hold times
Multicycle Approach

- Break up the instructions into steps, each step takes a cycle
  - balance the amount of work to be done
  - restrict each cycle to use only one major functional unit
- At the end of a cycle
  - store values for use in later cycles (easiest thing to do)
  - introduce additional “internal” registers
Multicycle Datapath
Implementing multicycle control

- Five Execution Steps
  - Instruction Fetch
  - Instruction Decode and Register Fetch
  - Execution, Memory Address Computation, or Branch Completion
  - Memory Access or R-type instruction completion
  - Write-back step

- Value of control signals is dependent upon:
  - what instruction is being executed
  - which step is being performed

- Use the information we’ve accumulated to specify a finite state machine
  - specify the finite state machine graphically, or
  - use microprogramming

- Implementation can be derived from specification
Finite state machine control
Microprogramming

Control unit

Microcode memory

Outputs

Datapath

Input

1

Adder

Microprogram counter

Address select logic

Instruction register
opcode field

1

0[b5–0]
Pipilining

- Improve performance by increasing instruction throughput

Ideal speedup is number of stages in the pipeline. Do we achieve this?
Implementation of the pipelined datapath

IF: Instruction fetch
ID: Instruction decode
EX: Execute
MEM: Memory access
WB: Write back
Datapath
Datapath with Control

Pass control signals along just like the data
Dependencies and forwarding

**Program execution order (in instructions):**
- `sub $2, $1, $3`
- `and $12, $2, $5`
- `or $13, $6, $2`
- `add $14, $2, $2`
- `sw $15, 100($2)`

**Value of register $2:**
- CC1: 10
- CC2: 10
- CC3: 10
- CC4: 10
- CC5: 10/20
- CC6: -20
- CC7: -20
- CC8: -20
- CC9: -20

**Value of MEM/EX:**
- CC1: X
- CC2: X
- CC3: X
- CC4: -20
- CC5: X
- CC6: X
- CC7: X
- CC8: X
- CC9: X

**Value of MEM/WB:**
- CC1: X
- CC2: X
- CC3: X
- CC4: -20
- CC5: X
- CC6: X
- CC7: X
- CC8: X
- CC9: X
When we decide to branch, other instructions are in the pipeline!
Pipeline with hazard detection and forwarding