Lecture 25

- Logistics
  - HW8 due today
  - Ant extra credit due Friday
  - Final exam, Wednesday March 18, 2:30-4:20 pm here
    - Review session Monday, March 16, 4:30 pm, here

- Last lecture
  - Encoding & Partitioning examples

- Today
  - Pipelining & Retiming
  - Control vs Datapath in a simple computer design

---

Other sequential logic optimization techniques

- Pipelining --- allows faster clock speed
- Retiming --- can reduce registers or change delays

---

Pipelining related definitions

- Latency: Time to perform a computation
  - Data input to data output
- Throughput: Input or output data rate
  - Typically the clock rate
- Combinational delays drive performance
  - Define \( d \) = delay through slowest combinational stage
  - \( n \) = number of stages from input to output
  - Latency = \( n \times d \) (in sec)
  - Throughput = \( 1/d \) (in Hz)

---

Pipelining

- What?
  - Subdivide combinational logic
  - Add registers between logic
- Why?
  - Trade latency for throughput
  - Increased throughput
    - Reduce logic delays
    - Increase clock speed
  - Increased latency
    - Takes cycles to fill the pipe
    - Increase circuit utilization
    - Simultaneous computations

---

Pipelining example
Pipelining and clock skew

- Which is faster?
- Which is safer?

Retiming

- Pipelining adds registers
  - To increase the clock speed
- Retiming moves registers around
  - Reschedules computations to optimize performance
  - Minimize critical path
  - Optimize logic across register boundaries
  - Reduce register count
  - Without altering functionality

Retiming in a nutshell

- Change position of FFs
  - For speed
  - To suit implementation target
- Retiming modifies state assignment
  - Preserves FSM functionality

Retiming ground rules

- Rules:
  - Remove one register from each input and add one to each output
  - Remove one register from each output and add one to each input

Retiming examples

- Reduce register count
- Change output delays

Optimal pipelining

- Add registers
- Use retiming to optimize location

Input ——— Output
Added registers for pipelining
Example: Digital correlator

\[ y_t = \delta(x_t, a_0) + \delta(x_t-1, a_1) + \delta(x_t-2, a_2) + \delta(x_t-3, a_3) \]

\[ \delta \] is a comparator: \( \delta(x, a) = 1 \) if \( x = a \); 0 otherwise

\( y_t \) is the number of matches between input and pattern \( a_0, a_1, a_2, a_3 \)

Example: Digital correlator (cont'd)

Delays: Comparator = 3; adder = 7

Data-path and control

Digital hardware systems = data-path + control

- Datapath: registers, counters, combinational functional units (e.g., ALU), communication (e.g., busses)
- Control: FSM generating sequences of control signals that instructs datapath what to do next

Tri-state gates

- The third value
  - Logic values: "0", "1"
  - Don't care: "X" (must be 0 or 1 in real circuit)
  - Third value or state: "Z" — high impedance, infinite \( R \), no connection

Tri-state and multiplexing

- When using tri-state logic
  - (1) make sure never more than one "driver" for a wire at any one time
  - (pulling high and low at the same time can severely damage circuits)
  - (2) make sure to only use value on wire when its being driven (using a floating value may cause failures)
- Using tri-state gates to implement an economical multiplexer

Open-collector gates and wired-AND

- Open collector: another way to connect gate outputs to the same wire
  - Gate only has the ability to pull its output low
  - It cannot actively drive the wire high (default – pulled high through resistor)
- Wired-AND can be implemented with open collector logic
  - If \( A \) and \( B \) are "1", output is actively pulled low
  - If \( A \) and \( B \) are "0", output is low
  - If one gate output is low and the other high, then low wins
  - Low to high transition usually slower than it would have been with a gate pulling high
  - Hence, the two NAND functions are ANDed together
Structure of a computer

- **Block diagram view**

![Block diagram of computer components](image)

- **Central processing unit (CPU)**
  - Instruction unit: instruction fetch and interpretation FSM
  - Execution unit: functional units and registers

Processor Memory System

- **Control signals**
- **Data conditions**

- **Registers**
  - Selectively loaded – EN or LD input
  - Output enable – OE input
  - Multiple registers – group 4 or 8 in parallel

![Register file diagram](image)

- **Collections of registers in one package**
  - Two-dimensional array of FFs
  - Address used as index to a particular word
  - Can have separate read and write addresses so can do both at the same time

- **4 by 4 register file**
  - 16 D-FFs
  - Organized as four words of four bits each
  - Write-enable (load)
  - Read-enable (output enable)

- **Memories**
  - Larger collections of storage elements
    - Implemented not as FFs but as much more efficient latches
    - High-density memories use 1 to 5 switches (transistors) per memory bit
  - Static RAM – 1024 words each 4 bits wide
    - Once written, memory holds forever (not true for denser dynamic RAM)
    - Address lines to select word
    - Read enable
      - Same as output enable
      - Also called chip select
    - Write enable (same as load enable)
  - Bi-directional data lines
    - Output when reading, input when writing

- **Instruction sequencing**
  - Example – an instruction to add the contents of two registers (Rx and Ry) and place result in a third register (Rz)
  - Step 1: Get the ADD instruction from memory into an instruction register (IR)
  - Step 2: Decode instruction
    - Instruction in IR has the code of an ADD instruction
    - Register indices used to generate output enables for registers Rx and Ry
    - Register index used to generate load signal for register Rz
  - Step 3: Execute instruction
    - Enable Rx and Ry output and direct to ALU
    - Setup ALU to perform ADD operation
    - Direct result to Rz so that it can be loaded into register

- **Instruction types**
  - Data manipulation
    - Add, subtract
    - Increment, decrement
    - Multiply
    - Shift, rotate
    - Immediate operands
  - Data staging
    - Load/store data to/from memory
    - Register-to-register move
  - Control
    - Conditional/unconditional branches in program flow
    - Subroutine call and return
Elements of the control unit (aka instruction unit)

- Standard FSM elements
  - state register
  - next-state logic
  - output logic (datapath/control signalling)
  - Moore or synchronous Mealy machine to avoid loops unbroken by FF
- Plus additional “control” registers
  - instruction register (IR)
  - program counter (PC)
- Inputs/outputs
  - outputs control elements of data path
  - inputs from data path used to alter flow of program (test if zero)

Instruction execution

- Control state diagram (for each diagram)
  - reset
  - fetch instruction
  - decode
  - execute
- Instructions partitioned into three classes
  - branch
  - load/store
  - register-to-register
- Different sequence through diagram for each instruction type

Data path (hierarchy)

- Arithmetic circuits constructed in hierarchical and iterative fashion
  - each bit in datapath is functionally identical
  - 4-bit, 8-bit, 16-bit, 32-bit, 32-bit datapaths

Data path (ALU)

- ALU block diagram
  - input: data and operation to perform
  - output: result of operation and status information

Data path (ALU + registers)

- Accumulator
  - special register
  - one of the inputs to ALU
  - output of ALU stored back in accumulator
- One-address instructions
  - operation and address of one operand
  - other operand and destination is accumulator register
  - AC ← AC op Mem[addr]
  - “single address instructions” (AC implicit operand)
- Multiple registers
  - part of instruction used to choose register operands

Data path (bit-slice)

- Bit-slice concept – iterate to build n-bit wide datapaths
Instruction path

- Program counter
  - keeps track of program execution
  - address of next instruction to read from memory
  - may have auto-increment feature or use ALU

- Instruction register
  - current instruction
  - includes ALU operation and address of operand
  - also holds target of jump instruction
  - immediate operands

- Relationship to data path
  - PC may be incremented through ALU
  - contents of IR may also be required as input to ALU

Data path (memory interface)

- Memory
  - separate data and instruction memory (Harvard architecture)
  - two address busses, two data busses
  - single combined memory (Princeton architecture)
  - single address bus, single data bus

- Separate memory
  - ALU output goes to data memory input
  - register input from data memory output
  - data memory address from instruction register
  - instruction register from instruction memory output
  - instruction memory address from program counter

- Single memory
  - address from PC or IR
  - memory output to instruction and data registers
  - memory input from ALU output

Block diagram of processor

- Register transfer view of Princeton architecture
  - which register outputs are connected to which register inputs
  - arrows represent data-flow, other are control signals from control FSM
  - MAR may be a simple multiplexer rather than separate register
  - MBR is split in two (REG and IR)
  - load control for each register

Block diagram of processor

- Register transfer view of Harvard architecture
  - which register outputs are connected to which register inputs
  - arrows represent data-flow, other are control signals from control FSM
  - two MARs (PC and IR)
  - two MBRs (REG and IR)
  - load control for each register