Introduction to Computer Engineering

CS/ECE 252, Spring 2017
Rahul Nayar
Computer Sciences Department
University of Wisconsin – Madison
Revision
Decoder

- A decoder is a circuit that changes a code into a set of signals
- The simplest is the 1-to-2 line decoder
Decoder

• 2-to-4 line decoder

<table>
<thead>
<tr>
<th>A_1</th>
<th>A_0</th>
<th>D_3</th>
<th>D_2</th>
<th>D_1</th>
<th>D_0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Decoder

• 4-to-16 line decoder
Multipluxer (MUX)

- **A multiplexer (or mux)** is a device that selects one of several digital input signals and forwards the selected input into a single line.
Multipluxer (MUX)
DeMultipluxer (DeMUX)

The demultiplexer takes one single input data line and then switches it to any one of a number of individual output lines one at a time.
Half Adder

The half adder adds two single binary digits $A$ and $B$. It has two outputs, sum ($S$) and carry ($C$). The carry signal represents an overflow into the next digit of a multi-digit addition.
**Half Adder**

Add two bits, produce one-bit sum and carry-out.

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>S</th>
<th>Cout</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0 0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0 1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1 0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1 1</td>
</tr>
</tbody>
</table>
Full Adder from Half Adder

Add two bits and carry-in, produce one-bit sum and carry-out.

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>Cin</th>
<th>S</th>
<th>C_out</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
D Latch

WE: Connected to CLK
Latch: Level triggered
Flip-Flop: Edge triggered
D Latch (Master Slave Flip Flop)

- A D flip flop takes only a single input, the D (data) input. The master-slave configuration has the advantage of being edge-triggered, making it easier to use in larger circuits, since the inputs to a flip-flop often depend on the state of its output.
- The circuit consists of two D flip-flops connected together. When the clock is high, the D input is stored in the first latch, but the second latch cannot change state. When the clock is low, the first latch's output is stored in the second latch, but the first latch cannot change state.
- The result is that output can only change state when the clock makes a transition from high to low.
D Latch (Master Slave Flip Flop)

- Positive Edge Triggered
- Slave is controlled by CLK and Master controlled by `CLK
- Negative Edge Master takes in values
Von Neumann Model

MEMORY
- MAR
- MDR

PROCESSING UNIT
- ALU
- TEMP

CONTROL UNIT
- PC
- IR

INPUT
- Keyboard
- Mouse
- Scanner
- Disk

OUTPUT
- Monitor
- Printer
- LED
- Disk
Instruction Processing

1. Fetch instruction from memory
2. Decode instruction
3. Evaluate address
4. Fetch operands from memory
5. Execute operation
6. Store result
LC-3 Overview: Memory and Registers

Memory
- address space: $2^{16}$ locations (16-bit addresses)
- addressability: 16 bits
- 64 KB

Registers
- eight general-purpose registers: R0 - R7
  - each 16 bits wide
- other registers
  - PC (program counter), condition codes
LC-3 Overview: Instruction Set

Opcodes

• 15 opcodes
• *Operate* instructions: ADD, AND, NOT
• *Data movement* instructions: LD, LDI, LDR, LEA, ST, STR, STI
• *Control* instructions: BR, JMP, RTI, TRAP
• some opcodes set/clear *condition codes*, based on result:
  ➢ N = negative, Z = zero, P = positive (> 0)

Data Types

• 16-bit 2’s complement integer

Addressing Modes

• non-memory addresses: *immediate, register*
• memory addresses: *PC-relative, indirect, base+offset*
LC-3 Overview: Instruction Set

Addressing Modes

• memory addresses: \textit{PC-relative, indirect, base+offset}
  
  ➢ \textit{PC-relative}: address directly in the instruction
    – Address: PC + \text{sext}(offset)
  
  ➢ \textit{Indirect}: Use data from Memory as Address
    – Address: M[ PC + \text{sext}(offset) ]
  
  ➢ \textit{Base+offset}: Use a different register as base
    – Address: \text{Base} + \text{sext}(offset)
  
  ➢ \textit{LEA}: To store addresses in Memory
    – Address: PC + \text{sext}(offset)
LC3: Control Instructions

Used to alter the sequence of instructions (by changing the Program Counter)

Conditional Branch
- branch is *taken* if a specified condition is true
  - signed offset is added to PC to yield new PC
- else, the branch is *not taken*
  - PC is not changed, points to the next sequential instruction

Unconditional Branch (or Jump)
- always changes the PC

TRAP
- changes PC to the address of an OS “service routine”
- routine will return control to the next instruction (after TRAP)