Overview of Chapter 7

- Review from Chapter 1
- Datapath and Control Unit
- Register Transfer Operations
- Microoperations
- Register Transfer Structures
  - Multiplexer based
  - Bus based
  - Three state bus
  - Memory transfer
  - Other transfer
- Datapath
- ALU
- Shifter and Barrel Shifter
- Datapath Representation and Control
- Pipelined Datapath
Review from Chapter 1: Computer Diagram

- **CPU (Central Processing Unit):** Performs sequences of processing operations on data stored in memory and interacts with Input/Output.
- **Datapath:** Performs basic operation on data stored in registers as directed by Control.
- **Control:** Determines the sequence of data processing operations to be performed in the Datapath.
- **Memory:** An addressable repository for data.
- **Input/Output:** A collection of devices that store, display and convert information.

Datapath and Control Unit

- **Control:** signals that configure data transfers and establish operations to be performed.
- **Status:** signals that represent the state of data, such as overflow bits, "zero" tests, etc. These signals are tested to change the sequence of operations.
Register Transfer Operations

- **Registers** – a collection of binary storage flip-flops organized in a logical fashion.
- **Register Transfer Operations** – The movement and processing of data stored in registers
- **Three basic components:**
  - set of registers
  - operations
  - control of operations
- **Elementary Operations** -- load, count, shift, add, bitwise "OR", etc.
  - Elementary operations are called *microoperations*

Register Transfer

- **Register Notation**

<table>
<thead>
<tr>
<th>R</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>PC(H)</em></td>
<td><em>PC(L)</em></td>
<td>15</td>
<td>0</td>
<td>15</td>
<td>0</td>
<td>R2</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

  - Letters and numbers – denotes a register (ex. R2, PC, IR)
  - Parentheses ( ) – denotes a range of register bits (ex. R1(1), PC(7:0), AR(L))
  - Arrow (→) – denotes data transfer (ex. R1 ← R2, PC(L) ← R0)
  - Comma – separates parallel operations
  - Brackets [ ] – Specifies a memory address (ex. R0 ← M[AR], R3 M[PC] )
Conditional Transfer

- If \( (K1 = 1) \) then \( (R2 \leftarrow R1) \) is shortened to \( K1: (R2 \leftarrow R1) \)
  where \( K1 \) is a control variable specifying a conditional execution condition.
- Conditional execution is used to modify the sequence of microoperations.

Microoperations

- Logical Groupings:
  - Transfer -- move data from one set of registers to another.
  - Arithmetic -- perform arithmetic on data in registers.
  - Logic -- manipulate data or use bitwise logical operations.
  - Shift -- shift data in registers.

<table>
<thead>
<tr>
<th>Arithmetic operations (word-wise)</th>
<th>Logical operations (bitwise)</th>
</tr>
</thead>
<tbody>
<tr>
<td>+ Addition</td>
<td>( \lor ) Logical OR</td>
</tr>
<tr>
<td>− Subtraction</td>
<td>( \land ) Logical AND</td>
</tr>
<tr>
<td>* Multiplication</td>
<td>( \oplus ) Logical Exclusive OR</td>
</tr>
<tr>
<td>/ Division</td>
<td>( \lnot ) Not</td>
</tr>
</tbody>
</table>
Example Microoperations

- Add the content of R1 to the content of R2 and place the result in R1.
  \[ R1 \leftarrow R1 + R2 \]
- Multiply the content of R1 by the content of R6 and place the result in PC.
  \[ PC \leftarrow R1 \times R6 \]
- Exclusive OR the content of R1 with the content of R2 and place the result in R1.
  \[ R1 \leftarrow R1 \oplus R2 \]

Example Microoperations (Continued)

- Take the 1's Complement of the contents of R2 and place it in the PC.
  \[ PC \leftarrow \overline{R2} \]
- On condition K1 OR K2, Logical bitwise OR the content of R1 with the content of R3 and place the result in R1.
  \[ (K1 + K2): R1 \leftarrow R1 \lor R3 \]
- NOTE: "+" (as in \( K_1 + K_2 \)) and means “OR.” In \( R1 \leftarrow R1 + R3 \), + means “plus.”
## Control Expressions

- **The control expression** for an operation appears to the left of the operation and is separated from it by a colon.
- **Control expressions** specify the **logical conditions** for the operation to occur.
- **Control expression values** of:
  - Logic "1" -- the operation takes place.
  - Logic "0" -- the operation is inhibited.

### Examples:

\[
\begin{align*}
X \ K1 & : R1 \leftarrow R1 + R2 \\
X \ K1 & : R1 \ (R1 + R2') + 1
\end{align*}
\]

- Variable K1 enables the add or subtract operation.
- If \( X = 0 \), then \( \bar{X} = 1 \) so \( X \ K1 = 1 \), activating the add of R1 and R2.
- If \( X = 1 \), then \( X \ K1 = 1 \), activating the add of R1 and the two's comp. of R2 (subtract).

## Arithmetic Microoperations

### From Table 7-3:

<table>
<thead>
<tr>
<th>Symbolic Designation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>R0 ← R1 + R2</td>
<td>Addition</td>
</tr>
<tr>
<td>R0 ← R1</td>
<td>Ones Complement</td>
</tr>
<tr>
<td>R0 ← R1 + 1</td>
<td>Two's Complement</td>
</tr>
<tr>
<td>R0 ← R2 + R1 + 1</td>
<td>R2 minus R1 (2's Comp)</td>
</tr>
<tr>
<td>R1 ← R1 + 1</td>
<td>Increment (count up)</td>
</tr>
<tr>
<td>R1 ← R1 – 1</td>
<td>Decrement (count down)</td>
</tr>
</tbody>
</table>

- Note that any register may be specified for source 1, source 2, or destination.
- These simple microoperations operate on the whole word -- except for 1's complement which is a bitwise operation.
Logical Microoperations

- From Table 7-4:

<table>
<thead>
<tr>
<th>Symbolic Designation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>R0 ← ¬R1</td>
<td>Bitwise NOT</td>
</tr>
<tr>
<td>R0 ← R1 ∨ R2</td>
<td>Bitwise OR (sets bits)</td>
</tr>
<tr>
<td>R0 ← R1 ∧ R2</td>
<td>Bitwise AND (clears bits)</td>
</tr>
<tr>
<td>R0 ← R1 ⊕ R2</td>
<td>Bitwise EXOR (complements bits)</td>
</tr>
</tbody>
</table>

Logical Microoperations (Continued)

- Let R1 = 10101010,
  and R2 = 11110000

- Then after the operation, R0 becomes:

<table>
<thead>
<tr>
<th>R0</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>01010101</td>
<td>R0 ← R1</td>
</tr>
<tr>
<td>11111010</td>
<td>R0 ← R1 ∨ R2</td>
</tr>
<tr>
<td>10100000</td>
<td>R0 ← R1 ∧ R2</td>
</tr>
<tr>
<td>01011010</td>
<td>R0 ← R1 ⊕ R2</td>
</tr>
</tbody>
</table>
Shift Microoperations

- From Table 7-5:
- Let \( R2 = 11001001 \)
- Then after the operation, \( R1 \) becomes:

<table>
<thead>
<tr>
<th>Symbolic Designation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>( R1 \leftarrow \text{sl} \ R2 )</td>
<td>Shift Left</td>
</tr>
<tr>
<td>( R1 \leftarrow \text{sr} \ R2 )</td>
<td>Shift Right</td>
</tr>
</tbody>
</table>

- Note: These shifts "zero fill". Sometimes a separate "link" bit can be used to provide the data shifted in, or to "catch" the data shifted out.
- Other shifts are possible (circular, arithmetic).

Register Transfer Structures

- **Multiplexer-Based Transfers** - Register inputs are connected to multiple sources via a multiplexer.
- **Bus-Based Transfers** - Register inputs are connected to a single bus driven by a multiplexer.
- **Three-State Bus** - Register inputs and outputs are connected to a single bus via tri-state drivers.
- **Register Cell Design** - designing a representative cell for the register
- **Memory Transfer** - Registers provide a source for Memory Addresses and a source or sink for Memory Data.
- **Other Transfer Structures** - Use multiple multiplexers, multiple busses, combinations of all the above, etc.
Multiplexer-Based Transfers

- Multiplexers connected to register inputs produce flexible transfer structures: (Note: Clocks are left off for clarity)

- The transfers are:

  K1: R0 ← R1
  K2 K1: R0 ← R2

![Figure 7-5a](image)

MUX-Based Transfers (Continued)

- Multiplexers connected to each register input produces a very flexible transfer structure:

- What transfers are possible with this structure? How many operations can occur in parallel?
Bus-Based Transfers

- A single input bus driven by a multiplexer limits the available transfers:
- What transfers can occur here?

Three-State Bus

- The 3-input MUX can be replaced by 3-state buffers. Transfers are still limited:
- What transfers are allowed here?
Register Cell Design – need and what is it?

- In Chapter 7: Register transfers introduced
- There may be multiple transfers into a register
- In Chapter 8: Multiplier ASM output development involves defining control signals for a set of register transfers:
  - Collect together all transfers into a given register
  - Define control signals that cause each distinct transfer
  - Find an equation for each control signal in terms of inputs and ASM state
- Given the above, we need to design the register
- This design is done if possible by:
  - Designing a representative cell for the register
  - Connecting copies of the cell together to form the register
  - Applying appropriate “boundary conditions” to end cells
- Register cell design is the first step of the above process

Foundations for Register Cell Design

- Assuming that the register has a “hold” function when no transfers occur, use a D flip-flop plus multiplexer register with a LOAD control.
- The LOAD for the cell is the “OR” of all of the control signals under which a transfer into the register occurs.
- The D-input for the cell can be designed using a K-map for a small number of transfers and a multiplexer + encoder or CAFÉ for a large number of transfers
Example 1: Register Cell Design

- Register A has the following transfers into it:
  - CX: A <- B v A
  - CY :A <- B XOR A

- Design register cell A_i for A.
  - What is the LOAD_i input for A_i?
    - CX + CY
  - What is the D_i input for A_i?
    - CX(B+A) + CY(B EOR A)

- K-map method can be used to minimize the logic or it can be implemented using mux and decoder

Memory Transfer

- Memory operations require:
  - ADDRESS

- And require:
  - DATA (write operations),

- Or provide:
  - DATA (read operations)

- Typically:
  - There can be more than one memory address source in a system.
  - There can be more than one data source or data sink in a system.
  - Some structure of buses and multiplexers is needed to access the memory.
Other Transfer Structures

- Fast systems require that parallel operations occur within the same clock.
- Parallel operations imply "resources" required to move the data.
- SO:
  - Multiple buses are used, and
  - Multiplexers are used to select input sources.
- THIS REQUIRES MORE HARDWARE!

Other Transfer Structures (Continued)

- What transfers does this system allow?

![Diagram of transfer structure]
Datapath

- Unit that can perform multiple functions and store results – often contains functional units (such as ALU, shifter) and register file
- Example Figure 7-9
ALU, Shifter and Barrel Shifter

- **ALU**
  - Can perform multiple operations such as add, subtract, and, or, EOR, increment, decrement, transfer – see table 7-8 for an example set of operations
  - Realization
    - Mux based: figure 7-15
    - Further reduction of logic possible using logic minimization methods

- **Shifter, Barrel Shifter**
  - Can be used for one or multiple bit shifts
  - Mux based realizations given in Figures 7-16 and 7-17

N-bit ALU (Fig 7-10)
An Arithmetic Circuit

B input logic for one stage of Arithmetic Unit

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>S_1</td>
<td>S_0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

(a) Truth table

(b) Map Simplification:
Y_i = B_iS_0 + B_iS_1
4-bit Arithmetic Unit

1-stage of logic unit

(a) Logic Diagram

(b) Function Table
1 stage of ALU

4-bit basic shifter
4-bit Barrel Shifter

Datapath Representation and Control

- **Datapath representation**
  - See Figure 7-18
  - Control signals (for function unit) are shown – see Table 7-10

- **Control**
  - Control variables and functions of datapath shown in Figure 7-19 and Table 7-11
Pipelined Datapath

- A method of improving performance
- Tradeoff between latency and throughput
  - Increases clock frequency
  - Many clock cycles to complete the job