Datapath Components – Subtractors, Two’s Complement, Overflow, ALUs, Register Files

Subtractor

- Can build subtractor as we built carry-ripple adder
- Mimic subtraction by hand
- Compute borrows from columns on left
  - Use full-subtractor component:
    - wi is borrow by column on right, wo borrow from column on left

Subtractor Example: Color Space Converter – RGB to CMY

- Color
  - Often represented as weights of three colors: red, green, and blue (RGB)
    - Perhaps 8 bits each, so specific color is 24 bits
      - White: R=11111111, G=11111111, B=11111111
      - Black: R=00000000, G=00000000, B=00000000
      - Other colors: values in between, e.g., R=00111111, G=00000000, B=00001111 would be a reddish purple
    - Good for computer monitors, which mix red, green, and blue lights to form all colors
  - Printers use opposite color scheme
    - Because inks absorb light
Datapath Components

Subtractor Example: Color Space Converter – RGB to CMY

- Printers must quickly convert RGB to CMY
  - C=255-R, M=255-G, Y=255-B
  - Use subtractors as shown

Try to save colored inks
- Expensive
- Imperfect – mixing C, M, Y doesn’t yield good-looking black

Solution: Factor out the black or gray from the color, print that part using black ink
- e.g., CMY of (250,200,200) = (200,200,200) + (50,0,0).
  - (200,200,200) is a dark gray – use black ink

Datapath Components

Subtractor Example: Color Space Converter – RGB to CMYK

- Call black part K
  - (200,200,200): K=200
  - (Letter “B” already used for blue)
- Compute minimum of C, M, Y values
  - Use MIN component designed earlier, using comparator and mux, to compute K
  - Output resulting K value, and subtract K value from C, M, and Y values
- Ex: Input of (250,200,200) yields output of (50,0,0,200)

Representing Negative Numbers: Two’s Complement

- Negative numbers common
- How represent in binary?
- Signed-magnitude
  - Use leftmost bit for sign bit
    - So -5 would be: 1101 using four bits
    - 10001001 using eight bits
- Better way: Two’s complement
  - Big advantage: Allows us to perform subtraction using addition
  - Thus, only need adder component, no need for separate subtractor component!
Before introducing two's complement, let's consider ten's complement. Complements for each base ten number shown to right. Complement is the number that when added results in 10.

- Nice feature of ten's complement: Instead of subtracting a number, adding its complement results in an answer exactly 10 too much. So just drop the 1 - results in subtracting using addition only.

**Example:**

46 + 7 \[= 53\] (10 too much)

Adding the complement results in an answer exactly 10 too much – dropping the tens column gives the right answer.

Two's complement is easy to compute:

- Just invert bits and add 1.

- Hold on!
  - Sure, adding the ten's complement achieves subtraction using addition only.
  - But don't we have to perform subtraction to determine the complement in the first place?
  - True - but in binary, two's complement can be computed easily.

**Example:**

- Two's complement of 011 is 101, because 011 + 101 = 1000.
- Could compute complement of 011 as 1000 - 011 = 101.
- Easier method: Just invert all the bits, and add 1.
  - The complement of 011 is 1001 + 1 = 1011 -- it works!

---

**Datapath Components Two’s Complement**

- What is the 4-bit binary two’s complement representation for the decimal number -5?
  1. -0101
  2. 1101
  3. 1010
  4. 1011

- What is the 5-bit binary two’s complement representation for the decimal number -5?
  1. -00101
  2. 10101
  3. 10011
  4. 11011
How many bits are needed to represent the number -12 in binary?

1. 3
2. 4
3. 5
4. Impossible!

What is the 5-bit binary two’s complement representation for the decimal number 7?

1. 00111
2. 10111
3. 11001
4. Two’s complement can only represent negative numbers

What is the decimal equivalent the two’s complement binary number 111?

1. 1
2. 7
3. -1
4. -3

What is the decimal equivalent the two’s complement binary number 1000?

1. 0
2. 1
3. 8
4. -8
Datapath Components

Two's Complement

- What is the decimal equivalent the two's complement binary number 0101?
  1. 5
  2. -5
  3. -11
  4. Nothing

Datapath Components

Two's Complement

- What is the decimal equivalent the two's complement binary number 111111111111?
  1. 1
  2. -1
  3. Who cares?
  4. I don’t know!

Datapath Components

Two's Complement Subtractor Built with an Adder

- Using two's complement
  \[ A - B = A + (-B) = A + \text{two's complement of } B = A + \text{invert_bits}(B) + 1 \]
  - So build subtractor using adder by inverting B’s bits, and setting carry in to 1

Datapath Components

Adder/Subtractor

- Adder/subtractor: control input determines whether add or subtract
  - Can use 2x1 mux – sub input passes either B or inverted B
  - Alternatively, can use XOR gates – if sub input is 0, B’s bits pass through; if sub input is 1, XORs invert B’s bits
Datapath Components

Overflow

- Sometimes result can’t be represented with given number of bits
  - Either too large magnitude of positive or negative
  - e.g., 4-bit two’s complement addition of 0111+0001 (7+1=8). But 4-bit two’s complement can’t represent number >7
    - 0111+0001 = 1000 WRONG answer, 1000 in two’s complement is -8, not +8
  - Adder/subtractor should indicate when overflow has occurred, so result can be discarded

Datapath Components

Overflow: Detecting Overflow: Method 1

- Assuming 4-bit two’s complement numbers, can detect overflow by detecting when the two numbers’ sign bits are the same but are different from the result’s sign bit
  - If the two numbers’ sign bits are different, overflow is impossible
    - Adding a positive and negative can’t exceed largest magnitude positive or negative
  - Simple circuit
    - overflow = a3b3’s3 + a3b3s3’
    - Include “overflow” output bit on adder/subtractor

  ![Overflow Examples](image)

  (a) 0111 1000 + 0001
  - Overflow: 1000 in two’s complement is -8, not +8

  (b) 1111 0111 + 0100
  - Overflow: 1011 in two’s complement is -7

  (c) 1000 1111 + 1011
  - No overflow: 1000 in two’s complement is -8

Datapath Components

Overflow: Detecting Overflow: Method 2

- Even simpler method: Detect difference between carry-in to sign bit and carry-out from sign bit
  - Yields simpler circuit: overflow = c3 xor c4

  ![Overflow Examples](image)

  (a) 1111 0111 + 0001
  - Overflow: 1000 in two’s complement is -8, not +8

  (b) 1110 0000 + 0010
  - Overflow: 1010 in two’s complement is -6

  (c) 1101 0000 + 0110
  - No overflow: 1011 in two’s complement is -7

Datapath Components

Arithmetic-Logic Unit (ALU)

- **ALU**
  - Component that can perform any of various arithmetic (add, subtract, increment, etc.) and logic (AND, OR, etc.) operations, based on control inputs

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Operation</th>
<th>Sample Output if A=00001111, B=00000000</th>
</tr>
</thead>
<tbody>
<tr>
<td>x y z</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0 0 0</td>
<td>A + B</td>
<td>A=00001000</td>
</tr>
<tr>
<td>0 0 1</td>
<td>A - B</td>
<td>B=00000100</td>
</tr>
<tr>
<td>0 1 0</td>
<td>A + 1</td>
<td>B=00000100</td>
</tr>
<tr>
<td>0 1 1</td>
<td>A OR B</td>
<td>A=00001111</td>
</tr>
<tr>
<td>1 0 0</td>
<td>A AND B</td>
<td>B=00000001</td>
</tr>
<tr>
<td>1 0 1</td>
<td>A OR B</td>
<td>A=00001111</td>
</tr>
<tr>
<td>1 1 0</td>
<td>A XOR B</td>
<td>B=00000011</td>
</tr>
<tr>
<td>1 1 1</td>
<td>NOT A</td>
<td>B=00000001</td>
</tr>
</tbody>
</table>

- **Motivation:**
  - Suppose want multi-function calculator that not only adds and subtracts, but also increments, ANDs, ORs, XORs, etc.
Datapath Components

Arithmetic-Logic Unit (ALU)

- More efficient design uses ALU
  - ALU design not just separate components multiplexed (same problem as previous slide),
  - Instead, ALU design uses single adder, plus logic in front of adder's A and B inputs
  - Logic in front is called an arithmetic-logic extender
  - Extender modifies the A and B inputs such that desired operation will appear at output of the adder

Datapath Components

Arithmetic-Logic Extender in Front of ALU

xyz = 000: Want S = A + B - just pass a to ia, b to ib, and set cin = 0
xyz = 001: Want S = A - B - pass a to ia, b' to ib, and set cin = 1
xyz = 010: Want S = A + 1 - pass a to ia, set ib = 0, and set cin = 1
xyz = 011: Want S = A - set a to ia, set ib = 0, and set cin = 0
others: likewise

Based on above, create logic for ia(x, y, z, a, b) and ib(x, y, z, a, b) for each abext, and create logic for cin(x, y, z), to complete design of the AL-extender component

Datapath Components

Register Files

- \( M \times N \) Register File
  - Provides efficient access to \( M \times N \)-bit-wide registers
  - If we have many registers but only need access one or two at a time, a register file is more efficient
  - Ex: Above-mirror display (earlier example), but this time having 16 32-bit registers
    - Too many wires, and big mux is too slow
**Datapath Components**

**Register Files: Timing Diagram**

- Can write one register and read one register each clock cycle
- May be same register

<table>
<thead>
<tr>
<th>Cycle</th>
<th>W_data</th>
<th>W_addr</th>
<th>W_en</th>
<th>R_data</th>
<th>R_addr</th>
<th>R_en</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>9</td>
<td>x</td>
<td>x</td>
<td>177</td>
<td>x</td>
<td>555</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>2</td>
<td>x</td>
<td>2</td>
<td>9</td>
<td>555</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>x</td>
<td>x</td>
<td>2</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>2</td>
<td>x</td>
<td>2</td>
<td>9</td>
<td>555</td>
</tr>
<tr>
<td>5</td>
<td>3</td>
<td>x</td>
<td>x</td>
<td>3</td>
<td>3</td>
<td></td>
</tr>
</tbody>
</table>

- clk: Cycle 1, 2, 3, 4, 5, 6
- W_data, W_addr, W_en: Write data, address, enable
- R_data, R_addr, R_en: Read data, address, enable

**Register File**:

```
<table>
<thead>
<tr>
<th>周期1</th>
<th>周期2</th>
<th>周期3</th>
<th>周期4</th>
<th>周期5</th>
<th>周期6</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>A</td>
<td>B</td>
<td>C</td>
<td>D</td>
<td>E</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>2</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
</tr>
<tr>
<td>3</td>
<td>13</td>
<td>14</td>
<td>15</td>
<td>16</td>
<td>17</td>
</tr>
<tr>
<td>4</td>
<td>19</td>
<td>20</td>
<td>21</td>
<td>22</td>
<td>23</td>
</tr>
<tr>
<td>5</td>
<td>25</td>
<td>26</td>
<td>27</td>
<td>28</td>
<td>29</td>
</tr>
<tr>
<td>6</td>
<td>31</td>
<td>32</td>
<td>33</td>
<td>34</td>
<td>35</td>
</tr>
</tbody>
</table>
```