17.1 Unit 18
Processor Organization

17.2 What Shall We Do?

- Let's design a simple processor to understand the entire flow from writing software to designing the hardware
  - This may not be the most advanced processor but the goal is to give you a fully working example from software to hardware

17.3 The Instruction Set (1)

- To start we will define the instruction set
- Let's make this a simple calculator-like processor that can perform at least the following 3 operations:
  - ___
  - ___
  - ___
- Goal is to evaluate simple arithmetic expressions: (7+4-5)&3
- Let's use ___-bit data values (i.e. all data operands will be ___-bits)
- To keep the number of bits needed to code an instruction to a minimum, let's use an ______________________ architecture where the _____ register is always one ________________
  - ADD 7 means: ________________
  - SUB 5 means: ________________

17.4 The Instruction Set (2)

- Let's assume the output of this computer is just 4 LED's to display a 4-bit binary number
- We'll provide some additional instructions to help us perform the calculations:
  - ____________________
  - ____________________
  - ____________________
- That leaves us with 6 total instructions
- How many bits do we need for the opcode of our instructions? __________
- If we want to store data/ constants in our instructions (e.g. ADD 7, SUB 5) how many additional bits do we need in our instruction? __________
- Instructions need ___ opcode + ___ data bits = ___-bits
  - Let's round up to 8-bits for each instruction
Compilation

- Consider the following "high-level" code
  - \((7 - 4 + 6) \& 3\)
- "Compile" it to an appropriate instruction sequence (i.e. assembly)
  - Assembly refers to the human readable syntax of each instruction

Now we need to convert to binary...

Defining the Machine Code

- Machine code refers to the representation of each instruction.
- We first need to define the actual opcodes so we can translate the assembly you wrote on the previous slide into binary for the hardware to execute.
- Before we do that, let's consider the hardware design as this will help us choose appropriate opcodes.

Instruction Set Summary

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADD k</td>
<td>ACC += k</td>
</tr>
<tr>
<td>SUB k</td>
<td>ACC -= k</td>
</tr>
<tr>
<td>AND k</td>
<td>ACC &amp; k</td>
</tr>
<tr>
<td>LOAD k</td>
<td>ACC = k</td>
</tr>
<tr>
<td>CLR</td>
<td>ACC = 0</td>
</tr>
<tr>
<td>OUT</td>
<td>OUT = ACC</td>
</tr>
</tbody>
</table>

Arithmetic and Logic Units

- Let's define and design the ALU that will perform the various operations our computer should support plus a few extra...

We will design what is inside this block. We just made up these code assignments and the various operations. Remember, we definitely need to support ADD, SUB, AND, and CLR (R=0).

Blank ALU To Complete

--- | --- | --- | --- | --- | --- | --- | --- | --- |
000 | 100 | 001 | 010 | 011 | 100 | 101 | 110 | 111 |

2-to-1, 4-bit wide mux

4-bit Binary Adder

We must have an input to provide the carry into the sum bits.

2-to-1, 4-bit wide mux
Control Logic

<table>
<thead>
<tr>
<th>R</th>
<th>FS[2:0]</th>
<th>S0</th>
<th>S1</th>
<th>S2</th>
<th>Ci</th>
<th>S3</th>
</tr>
</thead>
<tbody>
<tr>
<td>X+Y</td>
<td>000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>X-Y</td>
<td>001</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>X</td>
<td>010</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Y-X</td>
<td>011</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>X &amp; Y</td>
<td>100</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>unused</td>
<td>101</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>110</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>unused</td>
<td>111</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- \( S_0 = F_2'F_0 \)
- \( S_1 = F_2F_0' + F_1F_0 \)
- \( S_2 = F_2F_0' + F_2'F_1'F_0 \)
- \( C_i = F_1 + F_0 \)
- \( S_3 = F_1F_0 + F_2F_0 \)

Aside: Impacts of Coding (1)

- What if we changed the codes used for each operation?

We just made up these code assignments and the various operations. Remember, we definitely need to support ADD, SUB, AND, and CLR (R=0).
Defining the Machine Code Format

- Using the ALU design, can you suggest opcodes for the various instructions?
  - The accumulator (ACC) will be connected to the result of the ALU
  - But should the ACC be connected to the X or Y input of the ALU?
    • Important: We achieve Load by passing __ through the ALU to the ACC, so we need the constant to come in on X (so _____ cannot)

<table>
<thead>
<tr>
<th>Opcode</th>
<th>Op./Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>R = X + Y</td>
</tr>
<tr>
<td>001</td>
<td>R = X - Y</td>
</tr>
<tr>
<td>010</td>
<td>R = X</td>
</tr>
<tr>
<td>011</td>
<td>R = Y - X</td>
</tr>
<tr>
<td>100</td>
<td>R = X &amp; Y</td>
</tr>
<tr>
<td>101</td>
<td>Unused</td>
</tr>
<tr>
<td>110</td>
<td>R = 0</td>
</tr>
<tr>
<td>111</td>
<td>Unused</td>
</tr>
</tbody>
</table>

**Instruc. OPCODE Op./Result**
ADD 000
OUT 001
LOAD 010
SUB 011
ADD 100
CLR 110
AND 111

**Processor Datapath**

- Now let's consider the processor data path

**Assembler**

- Now translate the assembly you found from a few slides back to machine code and show it as 2 hex digits per instruction
- The "high-level" code was
  – (7 - 4 + 6) & 3
- "Compile" it to an appropriate instruction sequence (i.e., assembly)
  – CLR = ______
  – ADD 7 = ______
  – SUB 4 = ______
  – ADD 6 = ______
  – AND 3 = ______
**A Problem**

- Write assembly for:
  - \((-((7 \& 3) + (6 \& 5)))\)

**A Solution**

- Let's modify our processor as follows:
  - Add _______________ for temporary storage: ________________
  - Could add more but we'll keep it simple
  - A new instruction to save the_____ to a register: SAVE Rx (_______________)
  - Update ALU instructions to be able to specify a _________________ rather than just a constant
    - ADD Rx (ACC = ACC + Rx)
    - SUB Rx (ACC = ACC - Rx)
    - AND Rx (ACC = ACC & Rx)
    - LOAD Rx (ACC = Rx)
  - Update the instruction format to use the leftover bit to indicate whether the operand is a constant or should come from a register

**Updated Assembly**

- Write assembly for:
  - \((-((7 \& 3) + (6 \& 5)))\)
- New assembly & machine code

**Updated Processor Datapath**
More Practice (On Own Time)

- Write assembly for:
  - \( (48\&14) + (583) - (6\&11) + (8\&13) \)
- Try to use as few instructions as you can

<table>
<thead>
<tr>
<th>Instruction</th>
<th>OPCODE</th>
<th>Op./Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADD</td>
<td>00D</td>
<td>ACC = ACC + C/R</td>
</tr>
<tr>
<td>OUT</td>
<td>001</td>
<td>OUT = ACC</td>
</tr>
<tr>
<td>LOAD</td>
<td>010</td>
<td>ACC = X</td>
</tr>
<tr>
<td>SUB</td>
<td>011</td>
<td>ACC = ACC - C/R</td>
</tr>
<tr>
<td>AND</td>
<td>100</td>
<td>ACC = ACC &amp; C/R</td>
</tr>
<tr>
<td>CLR</td>
<td>110</td>
<td>ACC = 0</td>
</tr>
<tr>
<td>SAVE Rx</td>
<td>111</td>
<td>Rx = ACC</td>
</tr>
</tbody>
</table>

New Instruction Format

Opcode (3-bits) | C/R | Constant (4-bits) | Unused (3-bits) | I/1
---|---|---|---|---

ADD 7

<table>
<thead>
<tr>
<th>Instruction</th>
<th>OPCODE</th>
<th>Op./Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADD</td>
<td>000</td>
<td>ACC = ACC + C/R</td>
</tr>
<tr>
<td>OUT</td>
<td>001</td>
<td>OUT = ACC</td>
</tr>
<tr>
<td>LOAD</td>
<td>010</td>
<td>ACC = X</td>
</tr>
<tr>
<td>SUB</td>
<td>011</td>
<td>ACC = ACC - C/R</td>
</tr>
<tr>
<td>AND</td>
<td>100</td>
<td>ACC = ACC &amp; C/R</td>
</tr>
<tr>
<td>CLR</td>
<td>110</td>
<td>ACC = 0</td>
</tr>
<tr>
<td>SAVE Rx</td>
<td>111</td>
<td>Rx = ACC</td>
</tr>
</tbody>
</table>

ADD R0

SAVE R1