Motivation

• Now that you have some understanding...
  – Of how hardware is designed and works
  – Of how software can be used to control hardware
• We will look at how to improve efficiency of computer systems and software so that...
  – ...we can start to understand why HW companies create the structures they do (multicore processors)
  – ...we can begin to intelligently take advantage of the capabilities the HW gives us
  – ...we can start to understand why SW companies deal with some of the issues they do (efficiencies, etc.)
### Input / Output

- Processor performs reads and writes to communicate with I/O devices just as it does with memory.
  - I/O devices have locations (i.e. _________) that contain data that the processor can access.
  - These registers are assigned unique ____________ just like memory.

### Processor

- 3 Primary Components inside a processor:
  - ALU
  - Registers
  - Control Circuitry

- Connects to memory and I/O via **address, data, and control** buses (bus = ____________).

### Arithmetic and Logic Unit (ALU)

- Executes arithmetic operations like addition and subtraction along with logical operations (AND, OR, etc.).

### Registers

- Some are for general use by software:
  - Registers provide ______________ storage locations within the processor (to avoid having to read/write slow memory).

- Others are required for specific purposes to ensure proper operation of the hardware.
General Purpose Registers

- Registers available to software instructions for use by the ________________
- Instructions use these registers as inputs (_______ locations) and outputs (___________ locations)

What if we didn’t have registers?

- Example w/o registers: \( F = (X+Y) - (X*Y) \)
  - Requires an ADD instruction, MULtiply instruction, and SUBtract Instruction
  - w/o registers
    - ADD: Load X and Y from memory, store result to memory
    - MUL: Load X and Y again from mem., store result to memory
    - SUB: Load results from ADD and MUL and store result to memory
  - 9 memory accesses

What if we have registers?

- Example w/ registers: \( F = (X+Y) - (X*Y) \)
  - Load X and Y into registers
  - ADD: \( R_0 + R_1 \) and store result in \( R_2 \)
  - MUL: \( R_0 * R_1 \) and store result in \( R_3 \)
  - SUB: \( R_2 - R_3 \) and store result in \( R_4 \)
  - Store \( R_4 \) back to memory
  - 3 total memory access

Other Registers

- Some bookkeeping information is needed to make the processor operate correctly
- Example: Program Counter (PC)
  - Recall that the processor must fetch instructions from memory before decoding and executing them
  - PC register holds the address of the next instruction to fetch
Fetching an Instruction

- To fetch an instruction
  - PC contains the address of the instruction
  - The value in the PC is placed on the address bus and the memory is told to read
  - The PC is incremented, and the process is repeated for the next instruction

Control Circuitry

- Control circuitry is used to __________ the instruction and then generate the necessary signals to complete its execution
- Controls the ALU
- __________ registers to be used as source and destination locations (using __________)

Control Circuitry

- Assume 0x0201 is machine code for an ADD instruction of R2 = R0 + R1
- Control Logic will...
  - select the registers (R0 and R1)
  - tell the ALU to add
  - select the destination register (R2)
**DESIGN OF A SIMPLE INSTRUCTIONS SET AND PROCESSOR**

**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

**Instruction Sets**

- Defines the software _____________ of the processor and memory system
- Instruction set is the ___________ the HW processor can understand and the SW is composed with
  - Usually the compiler is the one that translates the software
- Most assembly/machine instructions fall into one of three categories
  - ________________
  - ________________ (to and from memory)
  - ________________ (branch, subroutine call, etc.)

**Instruction Set Architecture (ISA)**

- 2 approaches
  - __________ = __________ instruction set computer
    - ________________ vocabulary
    - More work per instruction, slower clock cycle
  - __________ = __________ instruction set computer
    - Small, basic, but ____________ vocabulary
    - Less work per instruction, faster clock cycle
    - Usually a simple and small set of instructions with regular format facilitates building faster processors
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: ___________

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/constant 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

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
Arithmetic and Logic Units

- Let's use the ALU we designed in a previous unit...

![ALU diagram]

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).

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>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>X-Y</td>
<td>001</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>X</td>
<td>010</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Y-X</td>
<td>011</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>X &amp; Y</td>
<td>100</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>d</td>
<td>1</td>
</tr>
<tr>
<td>unused</td>
<td>101</td>
<td>d</td>
<td>d</td>
<td>d</td>
<td>d</td>
<td>d</td>
</tr>
<tr>
<td>0</td>
<td>110</td>
<td>d</td>
<td>1</td>
<td>0</td>
<td>d</td>
<td>1</td>
</tr>
<tr>
<td>unused</td>
<td>111</td>
<td>d</td>
<td>d</td>
<td>d</td>
<td>d</td>
<td>d</td>
</tr>
</tbody>
</table>

- S0 = F1 • F0
- S1 = F1 • F0'
- S2 = F1 • F0
- Ci = F0
- S3 = F2

Completed ALU

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>Instruction Set Summary</th>
</tr>
</thead>
</table>
- ADD k (ACC += k)
- SUB k (ACC -= k)
- AND k (ACC & k)
- LOAD k (ACC = k)
- CLR (ACC = 0)
- OUT (OUT = ACC)

<table>
<thead>
<tr>
<th>Instruc.</th>
<th>OPCODE</th>
<th>Op./Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADD k</td>
<td>000</td>
<td>R = X + Y</td>
</tr>
<tr>
<td>SUB k</td>
<td>001</td>
<td>R = X - Y</td>
</tr>
<tr>
<td>AND k</td>
<td>010</td>
<td>R = X</td>
</tr>
<tr>
<td>LOAD k</td>
<td>011</td>
<td>R = Y - X</td>
</tr>
<tr>
<td>CLR</td>
<td>100</td>
<td>R = X &amp; Y</td>
</tr>
<tr>
<td>OUT</td>
<td>101</td>
<td>Unused</td>
</tr>
<tr>
<td>010</td>
<td>110</td>
<td>R = 0</td>
</tr>
<tr>
<td>011</td>
<td>111</td>
<td>Unused</td>
</tr>
</tbody>
</table>

![Machine Code Format diagram]
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 = ______

### Opcode

<table>
<thead>
<tr>
<th>Instruc. OPCODE</th>
<th>Opcode (3-bits)</th>
<th>Constant (4-bits)</th>
<th>Chosen Instruction Format</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADD 000</td>
<td>ACC = ACC + C</td>
<td></td>
<td></td>
</tr>
<tr>
<td>OUT 001</td>
<td>OUT = ACC</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LOAD 010</td>
<td>ACC = C</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SUB 011</td>
<td>ACC = ACC - C</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AND 100</td>
<td>ACC = ACC &amp; C</td>
<td></td>
<td></td>
</tr>
<tr>
<td>- 101</td>
<td>Unused</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CLR 110</td>
<td>ACC = 0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>- 111</td>
<td>Unused</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Processor Datapath

- Now let's consider the processor data path

Sample Execution of SUB 11

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)
- 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:
  - ( (4&14) + (5&3) - (6&11) + (8&13) )
- Try to use as few instructions as you can