Unit 16
Computer Organization and Instruction Sets

You Can Do That...

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.)

Computer Organization

- Three primary sets of components
  - Processor
  - Memory
  - I/O (everything else)
- Tell us where things live?
  - Running code
  - Compiled program (not running)
  - Circuity to execute code
  - Source code file
  - Data variables
  - Data for the pixels being displayed on your screen
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

![Diagram of Processor, Memory, and I/O Interfaces]

Video Interface

- Processor
- Memory
- A/D/C
- FE
- 800
- 3FF
- WRITE
- 01

Keyboard Interface

- Processor
- Memory
- 400
- 61

FE may signify a white dot at a particular location

This could just as easily be the command and data register from the LCD shield...
Or the PORT/DDR registers.

Processor

- 3 Primary Components inside a processor
  - ALU
  - Registers
  - Control Circuitry
- Connects to memory and I/O via address, data, and control buses (bus = _______________)

![Diagram of Processor, Memory, and I/O Interfaces]

Arithmetic and Logic Unit (ALU)

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

![Diagram of ALU]

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

![Diagram of Processor, Memory, and Registers]
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: \( F = (X+Y) - (X\times Y) \)
  - 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: \( F = (X+Y) - (X\times Y) \)
  - Load X and Y into registers
  - ADD: \( R_0 + R_1 \) and store result in \( R_2 \)
  - MUL: \( R_0 \times R_1 \) 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 currently executing instruction
16.13 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

16.14 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

16.15 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 ____________)

16.16 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)
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
Historical Instruction Format Options

- Instruction sets limit the number of operands used in an instruction due to:
  - To limit the complexity of the _____________
  - So that when an instruction is coded to binary it can _____ in a certain # of bits
- Different instruction sets specify these differently
  - 3 operand instruction set (ARM, PPC) -> (32-bit processors)
    - Usually all 3 operands in registers
    - Format: ADD DST, SRC1, SRC2 (DST = SRC1 + SRC2)
  - 2 operand instructions (Intel / Motorola 68K)
    - Second operand doubles as source and destination
    - Format: ADD SRC1, S2/D (S2/D = SRC1 + S2/D)
  - 1 operand instructions (Low-End Embedded, Java Virtual Machine)
    - Implicit operand to every instruction usually known as the Accumulator (or ACC) register
    - Format: ADD SRC1 (ACC = ACC + SRC1)
- 0 operand instructions / stack architecture
  - Push operands on a stack: PUSH X, PUSH Y
  - ALU operation: ADD (Implicitly adds top two items on stack: X + Y & replaces them with the sum)

General Instruction Format Issues

- Consider the high-level code
  - \( F = X + Y - Z \)
  - \( G = A + B \)
- Simple embedded computers often use single operand format
  - Smaller data size (8-bit or 16-bit machines) means limited instruction size
- Modern, high performance processors (Intel, ARM) use 2- and 3-operand formats

<table>
<thead>
<tr>
<th>Three-Operand</th>
<th>Two-Operand</th>
<th>Single-Operand</th>
<th>Stack Arch.</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADD F,X,Y</td>
<td>MOVE F,X</td>
<td>LOAD X</td>
<td>PUSH Z</td>
</tr>
<tr>
<td>SUB F,Z</td>
<td>ADD F,Y</td>
<td>ADD Y</td>
<td>PUSH Y</td>
</tr>
<tr>
<td>ADD G,A,B</td>
<td>SUB F,Z</td>
<td>SUB Z</td>
<td>PUSH SUB</td>
</tr>
<tr>
<td>ADD G,A</td>
<td>MOVE G,A</td>
<td>STORE F</td>
<td>ADD</td>
</tr>
<tr>
<td>ADD G,B</td>
<td>ADD G,B</td>
<td>ADD B</td>
<td>ADD POP F</td>
</tr>
</tbody>
</table>

(+ More natural program style (+) Smaller instruction count
(+ Smaller size to encode each instruction

Operand Addressing

- Most modern processors use a ___________ architecture
  - Load operands from memory into a register
  - Perform operations on registers and put results back into other registers
  - Store results back to memory
- Because ALU instructions only access registers, the CPU design can be simpler and thus faster
- Older designs
  - Register/Memory Architecture (Intel)
    - Operands of ALU instruc. can be in a reg. or mem.
  - Memory/Memory Architecture (DEC VAX)
    - Operands of ALU instruc. Can be in memory
    - ADD addrDst, addrSrc1, addrSrc2

Addressing Modes

- Addressing modes refer to how an instruction specifies **where** the operands are
  - Can be in a ___________, ____________ location, or a ____________ that is part of the instruction itself (aka. ____________ value)
- Most RISC processors: All data operands for arithmetic instructions must be in a ___________
  - This allows the hardware to be simpler and faster
- But what about something like: \( r8 = r8 + A[i] \) (A[i] is in mem.)
  - Intel instructions would allow: ADD r8,A[i]
    - A[i] is read from memory AND added to r8 in a single instruction
  - Other processors requires all data to be in a register before performing an arithmetic or logic operation (aka Load/Store Architecture)
    - Must use a separate instruction to read data from memory into a register
    - LOAD r9, A(i)
    - ADD r8, r9 (r8 = r8 + r9)
Load/Store Addressing

- When we load or store from/to memory how do we specify the address to use?
  - Note: Everything is a pointer at the instruction level
- Option 1: Direct Addressing
  - Address must be a constant: LOAD r8, (0xa140)
    - 0xa140 is just a made up address where we will assume A[0] lives
  - Would have to translate to:
    - LOAD r8, (0xa140)
    - LOAD r9, (0xa144)
    - LOAD r10, (0xa144)
    - ...

Option 2: Indirect Addressing
- Put address in a register: r9 = 0xa140
- LOAD uses contents of reg. as the address
- Then we can increment the address to prepare for next iteration
- Loop:
  - LOAD r8, (r9)
  - ADD r9, _____, _____
  - repeat

Picoblaze

- Picoblaze (aka KCPSM6) is an 8-bit soft-processor
  - It is not "hard" in that there is __________ you can buy with just a Picoblaze processor
  - It is "soft" in that the processor design is given as __________
  - It is intended to be integrated with other hardware designs and used to execute software to control those other hardware designs
  - The whole system can then be implemented on a chip or FPGA
Picoblaze Internals

• _______ registers named _______
  – Each register stores an 8-bit value
• PC is 12-bits allowing it to handle programs of up to ______ instructions

Picoblaze ALU

- ADD
- in1 in2
- out

PC

s0-sf (8-bits each)

Picoblaze Control

Normal Processor Bus Topology

• Most processors talk to memory and I/O devices over a common bus

Normal Processor Bus Topology Diagram

- Processor
- Memory
- Video Interface
- FE may signify a white dot at a particular location
- Keyboard Interface

PicoBlaze Processor Bus Topology

• Picoblaze has a separate:
  – ___________ memory / bus
  – ___________ memory / bus
  – _______ bus

PicoBlaze Processor Bus Topology Diagram

PICOBLAZE INSTRUCTION SET

PicoBlaze Instruction Set Diagram
SAMPLE ARITHMETIC/LOGIC INSTRUCTIONS

Performing operations on our data

ADD Instruction

- Adds a register value with a constant or two register values
  - add sx, _________ // sx = sx + ______
  - add sx, _________ // sx = sx + ______

- Example: add s3, 01
  - Performs register s3 = s3 + 1
  - Example: add s3, sb
    - Performs register s3 = s3 + sb

Derived from the KCPSM6 Manual

SUB Instruction

- Subtracts a register value with a constant or two register values
  - sub sx, constant // sx = sx - const.
  - sub sx, sy // sx = sx - sy

- Example: sub s3, 01
  - Performs register s3 = s3 - 1
- Example: sub s3, sb
  - Performs register s3 = s3 - sb

Derived from the KCPSM6 Manual

AND Instruction

- Subtracts a register value with a constant or two register values
  - and sx, constant // sx = sx & const.
  - and sx, sy // sx = sx & sy

- Example: and s3, 01
  - Performs register s3 = s3 & 1
- Example: and s3, sb
  - Performs register s3 = s3 & sb

Derived from the KCPSM6 Manual
DATA TRANSFER INSTRUCTIONS

Getting data in and out of our processor

**LOAD Instruction**

- Loads a register value with a constant
  - `load s3, constant` // \( sx = \text{const.} \)

**Example:** `load s3, 05`
- Performs register \( s3 = 05 \)

**STORE Instruction**

- Writes (stores) data from a processor register into memory at a given address
  - `store sx, const_addr`
  - `store sx, (sy)`

**Example:** `store s3, 20`
- Stores data from s3 to memory address 20

**Example:** `store s3, (sf)`
- Stores data in s3 using the value in reg. sf as the mem. address

Derived from the KCPSM6 Manual
Output Instruction

• Writes (stores) data from a processor register onto the I/O bus for the given port_id (I/O address)

  – output sx, const_addr // out_port = sx
  // port_id = const_addr

  – output sx, (sy) // out_port = sx
  // port_id = sy

• Example: output s3, 40
  – Outputs data in s3 and sets the port_id (I/O address) to 40

• Example: output s3, (sf)
  – Outputs data in s3 and uses the value in sf as the port_id (I/O address)

Derived from the KCPSM6 Manual

Input Instruction

• Reads (loads) data from an I/O register at the given port_id (I/O address) into a processor register

  – input sx, const_addr // sx = in_port = sx
  // port_id = const_addr

  – input sx, (sy) // sx = in_port
  // port_id = sy

• Example: input s3, 40
  – Reads the data at I/O port address 40 and places the data into processor reg. s3

• Example: input s3, (sf)
  – Uses the contents of sf as the I/O port address and reads the data into processor reg. s3

Derived from the KCPSM6 Manual

COMPARE Instruction

• Compares a register value with a constant or two register values by performing subtraction and updating the condition codes based on the result [if it is Negative (C) or Zero (Z)]

  – compare sx, constant // sx <=> const.

  – compare sx, sy // sx <=> sy

• Example: compare s3, 17
  – Performs register s3-17

• Example: compare s3, sf
  – Performs register s3-sf

Before: 16 s3
- 17

After: C,Z

Before: 85 s3
- 85 sf

After: C,Z

Derived from the KCPSM6 Manual
JUMP Instruction

• Jumps (changes the PC) to a new instruction if the given condition is true, or continues sequentially if condition is false
  - `jump const_addr` // PC=const_addr
  - `jump Z, const_addr` // if(z) PC=const_addr
  - `jump NZ, const_addr` // if(!z) PC=const_addr
  - `jump {C,NC}, const_addr`

• Example: `jump Z, 100`
  - Sets PC=100 only if Z=1, else PC++

• Example: `jump NC, 100`
  - Sets PC=100 only if C=0, else PC++

Derived from the KCPSM6 Manual

Picoblaze Assembly 1

• Suppose a button is attached to the Picoblaze responding to PORT_ID=4
• Suppose an LED is attached to the Picoblaze responding to PORT_ID=12
• Turn on the LED when the button is pressed (i.e. btn => 0) and off when not pressed (i.e. btn => 1)

```
while(1) {
  if(btn == 0) // pressed
    LED = 1;
  else
    LED = 0;
}
```

```
L1: input s1, __ // read button
    s2, s1  // OR w/ (1 << 6)
output s2, 20  // set ADCSRA
L2: input s3, 20  // get ADCSRA
    s3, s1  // OR w/ (1 << 6)
    s3, 0   // compare
    s2, L2  // jump if btn=1
input s4, 21  // get ADCH
load s5, 0   // load
load s6, 80  // load
L3: fetch s7, (s6)//
    s7, (s5)//
    s7, (s5)//
    s7, (s5)//
    store s7, (s6)//
    add s5, 1   //
    add s6, 1   //
    compare s5, 10 //
    s5, 10  //
    jump L2  //
    L1      // Done, goto top
```