The Big Picture

- Build machine model of execution
  - for Java and C programs
  - by examining language features
  - and deciding how they are implemented by the machine

What is required
- design an ISA into which programs can be compiled
- implement the ISA in the hardware simulator

Our approach
- examine code snippets that exemplify each language feature in turn
- look at Java and C, passing to dig deeper when C is different from Java
- design and implement ISA as needed

The simulator is an important tool
- machine execution is hard to visualize without it
- this visualization is really our WHOLE POINT here

In the Lab ...

- write C program to determine Endianness
  - prints "Little Endian" or "Big Endian"
- get comfortable with Unix command line and tools (important)
- compile and run this program on two architectures
  - IA32: i586 ugrad.cs.ubc.ca
  - Sparc: gallano.ugrad.cs.ubc.ca
- you can tell what type of arch you are on
  - ./arch

Simple Machine simulator
- load code into Eclipse and get it to build
- write and test MainMemory.java
- additional material available on the web page at lab time

The Main Memory Class

- The SM213 simulator has two main classes
  - CPU implements the fetch-execute cycle
  - MainMemory implements memory

The first step in building our processor
- implement 6 main internal methods of MainMemory

CPU
- fetch
- execute

MainMemory
- isAligned
- byteToInteger
- integerToBytes
- getset

Interlude
A Quick C Primer

The Code You Will Implement

- Convert a sequence of four bytes into a Big Endian integer.
  - @param byteArray value of byte at base address plus 3
  - @param byteArray value of byte at base address plus 2
  - @param byteArray value of byte at base address plus 1
  - @param byteArray byte at base address
  - @return Big Endian integer formed by these four bytes
  - public int bytesToInteger (UnsignedByte byteArray) {
  - return byteArray + byteArray + byteArray + byteArray;
  - }

- Fetch a sequence of bytes from memory.
  - @param address address of the first byte to fetch
  - @param length length of bytes to fetch
  - protected int readBytes (int address, int length) {
  - return byteArray[address] + byteArray[address+1] + ... + byteArray[address+length-1];
  - }

- Store a sequence of bytes into memory.
  - @param address address of the first byte
  - @param bytes array of UnsignedByte values
  - protected void set (int address, UnsignedByte[] bytes) {
  - byteArray[address] = byteArray[address+1] = ... = byteArray[address+length-1] = bytes[0];
  - }

- Get an array of UnsignedByte from four bytes.
  - @param byteArray value of byte at base address plus 3
  - @param byteArray value of byte at base address plus 2
  - @param byteArray value of byte at base address plus 1
  - @param byteArray byte at base address
  - @return an array of UnsignedByte
  - public UnsignedByte[] integerToBytes (int i) {
  - return byteArray + byteArray + byteArray + byteArray;
  - }

- Get an array of UnsignedByte from four bytes.
  - @param byteArray value of byte at base address plus 3
  - @param byteArray value of byte at base address plus 2
  - @param byteArray value of byte at base address plus 1
  - @param byteArray byte at base address
  - @return an array of UnsignedByte
  - public unsignedByte[] integerToByte (int i) {
  - return byteArray + byteArray + byteArray + byteArray;
  - }

Memory and Integers

- Memory is byte addressed
  - every byte of memory has a unique address, numbered from 0 to N
  - N is huge: billions is common these days (2-16 GB)

- Integers can be declared at different sizes
  - byte is 1 byte, 8 bits, 2 hexadecimals
  - short is 2 bytes, 16 bits, 4 hexadecimals
  - int or word or long is 4 bytes, 32 bits, 8 hexadecimals
  - long long is 8 bytes, 64 bits, 16 hexadecimals

Integers in memory
- reading or writing an integer requires specifying a range of byte addresses

Binary, Hex, and Decimal Refresher

Hexadecimal notation
- number starts with “0x,” each digit is base 16 not base 10
- e.g.: 0x23 = 2*16^1 + 3*16^0
- a convenient way to describe numbers when binary format is important
- each hex digit (hexit) is stored by 4 bytes: (0|0x4 + (1|0x1) + (2|0x3) + (3|0x2)x)

Examples
- 0x10 in binary? 7 decimal?
- 0x23 in binary? in decimal?
- 1101 1000 1001 0110 in hex? in decimal?
- 102 in binary? in hex?

Making Integers from Bytes

- Our first architectural decisions
  - assembling memory bytes into integer registers
  - Consider 4-byte memory word and 32-bit register
    - it has memory addresses i, i+1, and i+3
    - we’ll just say it’s “at address i is 4 bytes long”
    - e.g., the word at address 4 is in bytes 4, 5, 6 and 7.
  - Big or Little Endian (end means where start from, not finish)
    - we could start with the BIG END of the number (most everyone but Intel)
    - or we could start with the LITTLE END (Intel x86, some others)

CPU
- fetch
- execute

Interleave
A Quick C Primer

Numbers and Memory

Aligned or Unaligned Addresses
- we could allow any number to address a multi-byte integer
- or we could require that addresses be aligned to integer size boundary

-Aligned or Unaligned Addresses
- we could allow any number to address a multi-byte integer
- or we could require that addresses be aligned to integer size boundary

Memory is byte addressed
- every byte of memory has a unique address, numbered from 0 to N
- N is huge: billions is common these days (2-16 GB)

Integers can be declared at different sizes
- byte is 1 byte, 8 bits, 2 hexadecimals
- short is 2 bytes, 16 bits, 4 hexadecimals
- int or word or long is 4 bytes, 32 bits, 8 hexadecimals
- long long is 8 bytes, 64 bits, 16 hexadecimals

Integers in memory
- reading or writing an integer requires specifying a range of byte addresses

Question

Which of the following statement(s) is/are true
- [A] 6 = 110, is aligned for addressing a short
- [B] 6 = 110, is aligned for addressing a long
- [C] 20 = 10100 is aligned for addressing a long
- [D] 20 = 10100 is aligned for addressing a long long (i.e., 8-byte int)
Determine Endianness of a Computer

Which of the following statements are true

• [A] The Java constants 16 and 0x10 are exactly the same integer
• [B] 16 and 0x10 are different integers
• [C] Neither
• [D] I don’t know

Which of these are true

• [A] The Java constants 16 and 0x10 are exactly the same integer
• [B] 16 and 0x10 are different integers
• [C] Neither
• [D] I don’t know

What is the Big-Endian integer value at address 4 below?

• [A] 0x1c04b673
• [B] 0xc1406b37
• [C] I don’t know

What is the value of \( i \) after this Java statement executes?

\[
i = (\text{byte}(0x8b)) \ll 16;
\]

• [A] 0x8b
• [B] 0x0000008b
• [C] 0x00000008b
• [D] None of these
• [F] I don’t know