Computer Organization

- What does a computer consist of?
  - Electrons
  - IC chips with millions of transistors
  - Circuit theory
  - Keyboard, mouse, display
  - Windows or MacOS
  - Java programming language
  - Applications like iTunes
  - The internet
  - ...
- What level of abstraction describes a computer?

Von Neumann Architecture

- Abstraction of a programmable computer
  - Applies to almost all computers
- Key elements
  - Input and output to communicate information
  - Memory to hold information
  - Processor to perform operations
    - Arithmetic-Logic unit for data operations
    - Control unit to get instructions or data
  - Bus to move information between memory and processor
- Stored Program
  - Fundamental concept – the sequence of operations (the program) is just another set of data
  - This is what makes the computer easily programmed to do different things
Von Neumann Architecture

- History
  - Vacuum tubes → Transistors → Integrated circuits
  - Mainframes, minicomputers, microcomputers
  - VLSI, Moore's Law
  - Stack based architectures, RISC versus CISC
  - The network
- Alternatives and enhancements to Von Neumann architecture
  - Distributed/parallel systems
  - Array processing, Single Instruction Multiple Data (SIMD)
  - Caches, look-ahead, pipelining

Input and Output

- External memory
  - Significantly slower access
  - Disk, Flash memory, CDROM, tape, network, display, keyboard, mouse, printers
  - Orders of magnitude difference
- Disk drive access time
  - Directly addressable - but unequal access time
  - Seek time - to move head to track
  - Latency - spin till sector addressable
  - Transfer - pass under head, transfer to memory
- I/O buffered by controllers and drivers
Memory

- Random access memory
  - Typical unit is byte (8 bits)
  - View as sequence of bytes, each addressable
  - Equal time access to all memory cells
- Operations to transfer to processor
  - Load from memory to processor
  - Store from processor to memory

Processor

- Arithmetic/Logic Unit (ALU)
  - One or more data registers (R1, R2, ...) for operands and results
  - Status register for condition codes (CC)
  - Connected to ALU circuits to perform various arithmetic (add, subtract, ...) and logic (and, or, ...)
- Control Unit
  - Finite state machine directs operation
    - Decodes operation, operands, etc., from machine language format
  - Selects next instruction
    - Instruction register (IR), program counter (PC)
Machine Language

- Data transfer
  - LOAD/STORE from/to main memory and registers
  - MOVE between registers
- I/O between memory and devices
  - IN, OUT, READ, WRITE
- Arithmetic
  - ADD, SUB, MULT, DIV, SHIFT
  - Operands are registers or memory cells
- Comparison
  - Compare values and set condition code
  - EQ, LT, GT
- Control and branch
  - Default is sequential execution
  - JUMP, JUMPGT, ... - set PC absolutely or on a condition
  - HALT to end execution

Processing Model

- Machine cycle is fetch-decode-execute
- Fetch
  - Copy instruction indicated by PC to IR
  - Increment PC
- Decode
  - Decoder circuits interpret IR contents
  - Fetch operands as indicated
- Execute
  - Perform the instruction
  - May update data register
  - May update condition codes
  - May update PC
Duck Machine

- A simple single register machine
- 16 bit words
  - 4 bits instruction
  - 12 bits address
- One register $R$
- Program counter $PC$
- Condition code $CC$ (values $GT$, $EQ$, $LT$)

Duck Machine Instructions

- Each instruction (except halt) has one memory address as its operand

<table>
<thead>
<tr>
<th>Opcode</th>
<th>Instruction</th>
<th>Opcode</th>
<th>Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>Load from memory</td>
<td>1000</td>
<td>Jump to address</td>
</tr>
<tr>
<td>0001</td>
<td>Store to memory</td>
<td>1001</td>
<td>Jump if GT</td>
</tr>
<tr>
<td>0010</td>
<td>Clear memory</td>
<td>1010</td>
<td>Jump if EQ</td>
</tr>
<tr>
<td>0011</td>
<td>Add register to memory</td>
<td>1011</td>
<td>Jump if LT</td>
</tr>
<tr>
<td>0100</td>
<td>Increment memory</td>
<td>1100</td>
<td>Jump if not EQ</td>
</tr>
<tr>
<td>0101</td>
<td>Subtract register from memory</td>
<td>1101</td>
<td>Input to address</td>
</tr>
<tr>
<td>0110</td>
<td>Decrement memory</td>
<td>1110</td>
<td>Output from address</td>
</tr>
<tr>
<td>0111</td>
<td>Compare memory to register</td>
<td>1111</td>
<td>Halt</td>
</tr>
</tbody>
</table>
Example DM Programs

- Add two numbers
  
  1101000001000000
  1101000001000001
  0000000001000000
  0011000001000001
  0001000001000010
  1110000001000010
  1111000000000000

- Larger of two numbers
  
  1101000001000000
  1101000001000001
  0000000001000001
  0111000001000000
  1001000000000111
  1110000001000001
  1000000000001000
  1110000001000010
  1111000000000000

Machine Coding

- Low level von Neumann architecture provides general purpose computation
  - Theoretically possible to compute any problem with Duck Machine
- Programming in machine code is tedious, error prone
- Higher levels of abstraction needed to make programming easier
System Software

- Acts as intermediary between user and machine
  - Provides access to system resources
  - Manages system resources
- User's view is of system resources and the way access is provided
  - This creates a level of abstraction
  - User sees a virtual machine with appropriate functionality
  - User does not need to see the actual implementation of the machine

Types of System Software

- Language processors
  - Assemblers
  - Compilers
- Memory managers
  - Virtual memory
- File systems
  - Hierarchical directory structures
- Process scheduling
  - Shared use
- Utilities
  - Editors, browsers, debuggers, ...

Week 6  CIS 212  Spring 2006  13

Week 6  CIS 212  Spring 2006  14
Compilers and Interpreters

- Language translators are tools that translate code in one language (the source code) to code in another language (the target code).
- Compilers typically translate to machine code:
  - C, C++, Pascal, Fortran
  - The Java compiler translates to Java byte code, the code for the Java Virtual Machine
  - The java virtual machine interprets and executes byte code
- Interpreters execute the program as the source code is analyzed:
  - Scheme, Prolog, Perl, Python, Shell
- Either may use intermediate code during translation.

Assembly Language

- Writing programs in binary machine code is extremely difficult.
- Assembly language provides mnemonics to make writing and reading programs easier:
  - Assembly language is specific to a machine architecture
- An assembler is a tool that translates assembly language to machine code:
  - Usually one to one correspondence between assembly language instructions and machine instructions
  - May have some constructs that generate several instructions
  - Labels associate symbolic names to data and program locations
  - Easier to change and maintain code.
Assembly Language

- Pseudo operations
  - .BEGIN .END
delineate program
  - .DATA
create and initialize data
  - .BEGMACRO .ENDMACRO
define macros

- Example program

Example program

```
.BEGIN
IN X
IN Y
LOAD X
ADD Y
STORE RES
OUT RES
HALT
X: .DATA 0
Y: .DATA 0
RES: .DATA 0
.END
```

Duck Machine Assembly Language

- Mnemonics for binary opcodes
- Address argument is symbolic (from label)
- Any instruction may be labeled

<table>
<thead>
<tr>
<th>ASM</th>
<th>Instruction</th>
<th>ASM</th>
<th>Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>LOAD</td>
<td>Load from memory</td>
<td>JUMP</td>
<td>Jump to address</td>
</tr>
<tr>
<td>STORE</td>
<td>Store to memory</td>
<td>JUMPGT</td>
<td>Jump if GT</td>
</tr>
<tr>
<td>CLEAR</td>
<td>Clear memory</td>
<td>JUMPEQ</td>
<td>Jump if EQ</td>
</tr>
<tr>
<td>ADD</td>
<td>Add register to memory</td>
<td>JUMPLT</td>
<td>Jump if LT</td>
</tr>
<tr>
<td>INC</td>
<td>Increment memory</td>
<td>JUMPNEQ</td>
<td>Jump if not EQ</td>
</tr>
<tr>
<td>SUB</td>
<td>Subtract register from memory</td>
<td>IN</td>
<td>Input to address</td>
</tr>
<tr>
<td>DEC</td>
<td>Decrement memory</td>
<td>OUT</td>
<td>Output from address</td>
</tr>
<tr>
<td>COMPARE</td>
<td>Compare memory to register</td>
<td>HALT</td>
<td>Halt</td>
</tr>
</tbody>
</table>
Instruction Labels

- Print maximum of two input numbers

```
.BEGIN
IN   X
IN   Y
LOAD X
COMPARE Y -- set CC
JUMPGT YGT -- test CC
XGT: OUT X
     JUMP LAST
YGT: OUT Y
LAST: HALT
X: .DATA 0
Y: .DATA 0
.END
```

Loopying

- Compute sum from 1 to N

```
.BEGIN
IN   N
LOOP: LOAD I
      COMPARE N
      JUMPEQ LAST
      INC  I
      LOAD RES
      ADD  I
      STORE RES
      JUMP LOOP
LAST: OUT RES
      HALT
N:   .DATA 0
I:   .DATA 1
RES: .DATA 1
.END
```
General Patterns

- Expressions
  - Load value
  - Perform arithmetic operations
  - Store temporary results if necessary
- Assignment
  - Evaluate expression - value is in register
  - Store to variable
- Selection
  - Compare to set condition code
  - Jump on condition to then
    - <else part>
      - Jump to next
    - then:
      - <then part>
      - next:

- Repetition
  - loop: compare to set condition codes
    - Jump on condition to next
      - <body of loop>
        - Jump to loop
    - next:

- Basic statement forms of a structured programming language (like Java) can be represented with patterns in assembly
Assemblers

- Assemblers translate source (assembly language) into object (binary) code
- Two Pass Assembler
  - Pass 1 - Associate symbolic labels with address offsets
    - Count from beginning of file
    - Build symbol table
    - Report duplicate labels as error
  - Pass 2 - Translate assembly code to binary
    - Symbolic opcodes to binary
    - Symbolic addresses to binary
    - Report unknown addresses and opcodes as errors

Symbol Tables

- Core element of all language translators
- Two symbol tables in an assembler
  - Symbolic address to binary address
  - Symbolic opcode to binary opcode
- Format of table
  - Elements are pairs: symbol and binary value
    - Retrieved by symbol as key
  - Implementation affects performance
    - List is O(n) time
    - Sorted array is O(log n) time
    - Hash table is O(1) time