Announcements

• Program assignment 2 is posted
  – Due Friday, November 14
  – Keep the same group assignments
  – http://eecs.vanderbilt.edu/courses/cs231/CS_231_schedule.html

• Evaluation forms will be e-mailed after class
  – Return to me electronically by Monday 9am

Topics

• Questions and feedback

• Java bytecode

• Microinstructions
  – Tanenbaum 4.3.1 & 4.3.2
What is Program 2?

• Find the median value among five inputs
  – Input values are binary (method)
  – Sort the five values to identify the median (method)
  – Use the print method again to display the output in hexadecimal

• Suggestions
  – Each of you will be responsible for a code segment, but you will probably want to collectively discuss each part
  – It may be helpful to write the code in a high-level (pseudo) language and translate to assembly

Don’t wait until the last minute!!!

Questions and Feedback

• What is -128 in 8-bit 2’s complement?
  – 0x80 (this is hexadecimal notation)
  – The smallest negative value in 2’s complement is ALWAYS a 1 in the MSB followed by all zeroes
Questions and Feedback

- **What is -128 in 8-bit 2’s complement?**
  - 0x80 (this is hexadecimal notation)
  - The smallest negative value in 2’s complement is **ALWAYS** a 1 in the MSB followed by all zeroes

- **Thoughts on using the simulator?**
  - Compiling, debugging, etc.
  - Stepping through the program’s microcode

- **Anything else???
IJVM Instruction Set

<table>
<thead>
<tr>
<th>Hex</th>
<th>Mnemonic</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x10</td>
<td>BIPUSH byte</td>
<td>Push byte onto stack</td>
</tr>
<tr>
<td>0x59</td>
<td>DUP</td>
<td>Copy top word on stack and push onto stack</td>
</tr>
<tr>
<td>0xA7</td>
<td>GOTO offset</td>
<td>Unconditional branch</td>
</tr>
<tr>
<td>0x60</td>
<td>IADD</td>
<td>Pop two words from stack; push their sum</td>
</tr>
<tr>
<td>0x7E</td>
<td>IAND</td>
<td>Pop two words from stack; push Boolean AND</td>
</tr>
<tr>
<td>0x98</td>
<td>IFEO offset</td>
<td>Pop word from stack and branch if it is zero</td>
</tr>
<tr>
<td>0x9F</td>
<td>IF_JCMEQ offset</td>
<td>Pop two words from stack; branch if equal</td>
</tr>
<tr>
<td>0x84</td>
<td>INIC varnum, const</td>
<td>Add a constant to a local variable</td>
</tr>
<tr>
<td>0x15</td>
<td>ILOAD varnum</td>
<td>Push local variable onto stack</td>
</tr>
<tr>
<td>0x86</td>
<td>INVOKEVIRTUAL disp</td>
<td>Invoke a method</td>
</tr>
<tr>
<td>0x83</td>
<td>IOR</td>
<td>Pop two words from stack; push Boolean OR</td>
</tr>
<tr>
<td>0x36</td>
<td>IRETURN</td>
<td>Return from method with integer value</td>
</tr>
<tr>
<td>0x36</td>
<td>ISTORE varnum, disp</td>
<td>Pop word from stack and store in local variable</td>
</tr>
<tr>
<td>0x04</td>
<td>ISUB</td>
<td>Pop two words from stack; push their difference</td>
</tr>
<tr>
<td>0x13</td>
<td>LDGE_W index</td>
<td>Push constant from constant pool onto stack</td>
</tr>
<tr>
<td>0x00</td>
<td>NOP</td>
<td>Do nothing</td>
</tr>
<tr>
<td>0x57</td>
<td>POP</td>
<td>Delete word on top of stack</td>
</tr>
<tr>
<td>0x5F</td>
<td>SWAP</td>
<td>Swap the two top words on the stack</td>
</tr>
<tr>
<td>0xC4</td>
<td>WIDE</td>
<td>Prefix instruction; next instruction has a 16-bit index</td>
</tr>
</tbody>
</table>

Figure 4.11. The IJVM instruction set. The operands byte, const, and varnum are 1 byte. The operands disp, index, and offset are 2 bytes.

Block Diagram of Microarchitecture (Mic-1)
Loops in Machine Language

High-level language:
for (int i = 3; i == 0; i--)
    BIPUSH 3
    ISTORE i
L1:    ILOAD i
    IFEQ L2
    IINC i -1
    GOTO L1
L2:    HALT

One byte per address line

<table>
<thead>
<tr>
<th>Byte Address</th>
<th>Bytecode</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0x10</td>
</tr>
<tr>
<td>1</td>
<td>0x03</td>
</tr>
<tr>
<td>2</td>
<td>0x36</td>
</tr>
<tr>
<td>3</td>
<td>0x01</td>
</tr>
<tr>
<td>4</td>
<td>0x15</td>
</tr>
<tr>
<td>5</td>
<td>0x01</td>
</tr>
<tr>
<td>6</td>
<td>0x99</td>
</tr>
<tr>
<td>7</td>
<td>0x00</td>
</tr>
<tr>
<td>8</td>
<td>0x09</td>
</tr>
<tr>
<td>9</td>
<td>0x84</td>
</tr>
<tr>
<td>A</td>
<td>0x01</td>
</tr>
<tr>
<td>B</td>
<td>0xFF</td>
</tr>
<tr>
<td>C</td>
<td>0xA7</td>
</tr>
<tr>
<td>D</td>
<td>0xFF</td>
</tr>
<tr>
<td>E</td>
<td>0xF8</td>
</tr>
<tr>
<td>F</td>
<td>0xFF</td>
</tr>
</tbody>
</table>

Mic-1 Simulator Window

• Microinstruction – current microinstruction being executed
• NextMicroinstruction – next microinstruction to be executed
• MPC – current location in the control store
• Registers – value stored in each microarchitecture register
Microcode for a “Mystery” IJVM Instruction

Each line is 1 execution cycle! Several tasks are combined.

H = LV
MAR = MBRU + H; rd
PC = PC + 1; fetch
H = MDR
PC = PC + 1; fetch
MDR = MBR + H; wr; goto Main1

Figure 4-17 contains the Mic-1 microprogram

Assignment and ALU Operations

| DEST = H |
| DEST = SOURCE |
| DEST = H |
| DEST = SOURCE |
| DEST = H + SOURCE |
| DEST = H + 1 |
| DEST = SOURCE + 1 |
| DEST = SOURCE - H |
| DEST = SOURCE - 1 |
| DEST = -H |
| DEST = H AND SOURCE |
| DEST = H OR SOURCE |
| DEST = 0 |
| DEST = 1 |
| DEST = -1 |

- Possible sources
  - Connected to B bus
  - MDR, PC, MBR, MBRU, SP, LV, CPP, TOS, or OPC

- Possible destinations
  - Connected to C bus
  - MAR, MDR, PC, SP, LV, CPP, TOS, OPC, or H

Figure 4-16. All permitted operations. Any of the above operations may be extended by adding “$\ll 8$” to them to shift the result left by 1 byte. For example, a common operation is $H = \text{MBR} \ll 8$. 
Invalid Operations

- MDR = SP + MDR
- H = H - MDR
- SP = H + MAR
- Etc.

Multiple assignments

- SP = MDR = SP + 1
- MDR = TOS = MBR
- Etc.
Memory Access

- **rd, wr**
  - uses MAR/MDR to access constant pool, current local variable frame and current operand stack

- **fetch**
  - uses PC/MBR to access method area

- Initiated at the end of the cycle
  - After the C bus is valid

- Data availability for rd and fetch
  - At the end of the next cycle

Branching

- **Unconditional**
  - goto *label*
  - Can explicitly name a successor for unconditional branch

- **Conditional**
  - Use ALU flags Z and N
    - Program Status Word
    - Set according to the result of the ALU operation
    - Ex: Z = TOS
  - If (Z) goto *L1*; else *L2*
  - If (N) goto *L1*; else *L2*

- Multiway branch for next opcode
  - goto (MBR OR *value*)
Next Address

- Opcode is the anchor for each IJVM instruction
  - First microinstruction for each IJVM instruction loaded at that hexadecimal address in control store

- Each microinstruction specifies its successor
  - In microcode, next address is implicitly the next microinstruction
  - Not necessarily at the next control store location

- Microassembler determines the remaining locations
  - For branches, the condition can only differ in MSB
  - Main1 loop does not reside at the first location because that is the opcode for NOP

Microinstruction Notation

- **B** field is source register
- **Mem** field for initiating memory operation
- **C** field sets one (or more) destination registers
- **ALU** field determines the ALU operation
- **JAM** field determines the branching
- **Addr** field explicitly names the successor of the current microinstruction
Function of the Microprogram

• Contains a Main loop
  – Fetches, decodes, and executes instructions from the program being interpreted

• Mic-1 Main loop
  Main1 \( PC = PC + 1; \) fetch; goto (MBR)
  – Increments the program counter
  – Initiates a fetch to get the next opcode/operand
  – Multiway branch to the current opcode

Microcode for a “Mystery” IJVM Instruction

\[
\begin{align*}
H &= LV \\
MAR &= MBRU + H; \text{ rd} \\
PC &= PC + 1; \text{ fetch} \\
H &= MDR \\
PC &= PC + 1; \text{ fetch} \\
MDR &= MBR + H; \text{ wr; goto Main1} \\
\end{align*}
\]

Microcode for IINC