Computer Organization  
CS 231-01

Mic-1 Microinstructions

Dr. William H. Robinson  
November 1, 2004

http://eecs.vanderbilt.edu/courses/cs231/

Topics

“There is no one giant step that does it. It's a lot of little steps.”

– Peter A. Cohen  
(American Investment Banker)

• Administrative stuff
  – Homework #3 posted
  – Program #2 posted (Later today)

• Mic-1 Microinstructions
• Improving the Mic-1

Programming Teams

• Weblink from class schedule page
  – Keep the same teams for all assignments

• Team 1: Noor, Ted, Tamim
• Team 2: Michael, David F., Nikhil
• Team 3: Theodor, Brandon, Jon
• Team 4: Brett, Jeb, John W.
• Team 5: Dante’, Clay, William
• Team 6: John L., Lauren, David R.

Control Store (ROM)

• Memory that holds the microprogram
• Contains 512 words, each a 36-bit microinstruction
• Each microinstruction specifies its successor
  – Not executed in order stored in control store
• Accessing the microprogram
  – MicroProgram Counter holds address for next microinstruction
  – MicroInstruction Register holds the current microinstruction
Microinstruction Notation

- Could use the 36-bit words
  - Cumbersome method
  - Easier to abstract to pseudocode

- Things to be familiar with
  - Register names & functions
  - MAR/MDR & PC/MBR functions
  - IJVM memory model

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

<table>
<thead>
<tr>
<th>F2</th>
<th>F1</th>
<th>ENA</th>
<th>ENVA</th>
<th>INC</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>DEST = H</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>DEST = SOURCE</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>DEST = SOURCE</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>DEST = SOURCE</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>DEST = A – B</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>DEST = A – B</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>DEST = A – B</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>DEST = A – B</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>DEST = A AND B</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>DEST = A OR B</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>DEST = 0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>DEST = 1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>DEST = 1</td>
</tr>
</tbody>
</table>

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)

Program Status Word

ISA register containing status flags (p. 310)

• N – Set when the result is **Negative**
• Z – Set when the result is **Zero**
• V – Set when the result caused an **Overflow**
• C – Set when the result caused a **Carry out of MSB**
• P – Set when the result had even **Parity**

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 0, 1, 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

<table>
<thead>
<tr>
<th>Bits</th>
<th>9</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>NEXT_ADDRESS</strong></td>
<td><strong>J</strong></td>
<td><strong>M</strong></td>
<td><strong>AM</strong></td>
<td><strong>Z</strong></td>
<td><strong>C</strong></td>
</tr>
<tr>
<td><strong>ALU</strong></td>
<td><strong>E</strong></td>
<td><strong>N</strong></td>
<td><strong>AB</strong></td>
<td><strong>V</strong></td>
<td><strong>O</strong></td>
</tr>
<tr>
<td><strong>C</strong></td>
<td><strong>M</strong></td>
<td><strong>P</strong></td>
<td><strong>R</strong></td>
<td><strong>X</strong></td>
<td><strong>E</strong></td>
</tr>
<tr>
<td><strong>Mem</strong></td>
<td><strong>B</strong></td>
<td><strong>A</strong></td>
<td><strong>C</strong></td>
<td><strong>G</strong></td>
<td><strong>L</strong></td>
</tr>
<tr>
<td><strong>Addr</strong></td>
<td><strong>D</strong></td>
<td><strong>SP</strong></td>
<td><strong>16</strong></td>
<td><strong>none</strong></td>
<td></td>
</tr>
</tbody>
</table>

**B** registers:
- 0 = MDR
- 1 = LR
- 2 = MR
- 3 = MBR
- 4 = SP

**Mem** registers:
- 0 = MDR
- 1 = LR
- 2 = MR
- 3 = MBR
- 4 = SP

**ALU** operations:
- 0 = ADD
- 1 = SUB
- 2 = AND
- 3 = OR
- 4 = XOR
- 5 = MUL
- 6 = DIV
- 7 = SUBL
- 8 = SUBL
- 9 = SHL
- 10 = SHR

**JAM** values:
- 0 = CONTINUE
- 1 = NEXT
- 2 = IMMEDIATE
- 3 = RELATIVE
- 4 = JUMP

**Addr**:
- 0 = CONTINUE
- 1 = NEXT
- 2 = IMMEDIATE
- 3 = RELATIVE
- 4 = JUMP

- **N** – Set when the result is Negative
- **Z** – Set when the result is Zero
- **V** – Set when the result caused an Overflow
- **C** – Set when the result caused a Carry out of MSB
- **P** – Set when the result had even Parity
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

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

Microcode for IINCl

---

Stepping Through Microinstructions

- Main loop is executed
- Multiway branch to BIPUSH microinstruction sequence
Stepping Through Microinstructions

Step: 2
Instruction: $SP = MAR = SP + 1$

- Stack Pointer and Memory Address Register updated
- Next byte is fetched into Memory Buffer Register

Step: 3
Instruction: $PC = PC + 1; \text{fetch}$

- Program Counter is updated
- Initiate a fetch operation

Stepping Through Microinstructions

Step: 4
Instruction: $MDR = TOS = MBR; \text{wr}; \text{goto Main1}$

- Operand is stored in Top Of Stack and Memory Data Register
- Next byte is fetched into Memory Buffer Register

Summary

- IJVM instructions are executed by a series of microinstructions that each take one cycle
- Microinstruction notation references the structures within the Mic-1 microarchitecture
- Memory operations (rd, fetch) are available at the end of the next cycle