Guest Lecture

- **Dr. Robinson will be on travel**
  - Friday, October 8, 2004

- **Dr. Larry Dowdy will cover the lecture**
  - Lecture topics will remain on schedule
  - Responsible for material presented
  - Homework assignment will be collected

Mic-1 Simulator for Class

Java-based simulator which implements the Mic-1 microarchitecture described in Chapter 4

- [http://www.ontko.com/mic1/](http://www.ontko.com/mic1/)

*Programming assignments will use this!!!*
Block Diagram of Microarchitecture (Mic-1)

Datapath
Part of CPU containing ALU, its inputs, and its outputs

Purpose
Implement the ISA level above it (macro-architecture)

Control Section
Part of CPU containing the HW necessary to direct the datapath

Datapath of IJVM

Example ISA: IJVM
- Subset of Java Virtual Machine (JVM)
- Only integer instructions

Example microarchitecture
- 32-bit architecture (4 bytes/word)
- Microprogram (in ROM)
- Loop to fetch, decode, and execute instructions

Datapath Registers
- Memory Address Register
- Memory Data Register
- Program Counter
- Memory Buffer Register
- Stack Pointer
- Local Variable pointer
- Constant Pool Pointer
- Top Of Stack
- OPC
- Holding register

Datapath Timing
- Has a finite propagation time
  - Signals travel along wires, through transistors, etc.
- Implicit clock subcycles
  - Set up control signals to drive datapath
  - Register loaded onto B bus
  - ALU and shifter operate
  - Result propagates along C bus to registers
Timing Diagram for One Datapath Cycle

Single-Stage Datapath

- Making the design work requires:
  - Rigid timing
  - Long clock cycle
  - Known minimum propagation time through ALU
  - Fast load of registers from C Bus

- ALU runs continuously
  - Inputs and outputs aren’t valid until right “subcycle”

- Only explicit signals driving datapath
  - Falling edge and rising edge of clock

Datapath Registers

- Only accessible at the microarchitecture level by the microprogram
- Maintains the state of the computer
- Usually hold an address or a data value corresponding to an ISA variable of the same name

Datapath Registers (Memory Control)

- MAR
  - Memory Address Register
  - Contains word addresses for memory references

- MDR
  - Memory Data Register
  - Contains data words to/from memory references

- PC
  - Program Counter
  - Contains address of next instruction to be executed

- MBR
  - Memory Buffer Register
  - 8-bit register used to read a single byte from memory
**Memory Operation**
- Memory Address Register
- Memory Data Register
- Program Counter
- Memory Buffer Register

- Registers driven by control signals
  - WRITE, READ, FETCH

- Two ways to access memory
  - 32-bit port (word addressable) (MAR, MDR)
  - 8-bit read-only port (byte addressable) (PC, MBR)

- Actual memory is byte oriented

**Mapping the MAR to Address Bus**

**Placing MBR onto B Bus**
- **Unsigned**
  - MBR value in low-order 8 bits
  - Zeros placed in upper 24 bits
  - Used to index into a table
  - Used to assemble a 16-bit integer from 2 consecutive (unsigned) bytes in instruction stream

- **Signed**
  - Use MBR as a value between -128 and +127
  - Sign extension duplicates leftmost bit in MBR

**Datapath Registers (Stack)**
- **SP**
  - Stack Pointer
  - Contains the address of the highest word in the local variable frame

- **LV**
  - Local Variable pointer
  - Contains the address of the first location in the local variable frame

- **CPP**
  - Constant Pool Pointer
  - Contains the address of the first word of the constant pool

- **TOS**
  - Top Of Stack
  - Contains the value of highest word on memory stack
Using the Stack for Local Variables

- Programming languages support the concept of procedures
  - Local variables can be accessed within the procedure but not after the procedure has returned
  - Where are they stored?

- Simple solution: Absolute address
  - Problem occurs with recursive procedures
  - Data would be overwritten

- Use an area of memory called the “stack”
  - Local variable frames are placed on top of each other when procedures are called

Stack Example

After Procedure A calls Procedure B

After Procedure B calls Procedure C

After Procedure B and Procedure C return, then Procedure A calls Procedure D

Stack Example
IVJM Memory Model

- **Constant pool**
  - Cannot be written by IJVM program
  - Consists of constants, strings, and pointers to other areas of memory

- **Local variable frame**
  - Allocated to store variables for the lifetime of an invoked procedure

- **Operand stack**
  - Separate area above local variable frame
  - Used to hold operands during arithmetic computation

- **Method area**
  - Contains the program
  - Treated as a byte array

Parts of IJVM Memory

- CPP, LV, and SP registers are pointers to **words**
- PC contains a **byte address**

Block Diagram of Microarchitecture (Mic-1)

Fetch-Decide-Execute

1. Fetch the next instruction from memory into the instruction register.
2. Change the program counter to point to the following instruction.
3. Determine the type of instruction just fetched.
4. If the instruction uses a word in memory, determine where it is.
5. Fetch the word, if needed, into a CPU register.
6. Execute the instruction.
7. Start over and begin executing the following instruction.
Mic-1 Sequencer

- Determines which control signal should be enabled on each cycle
- Steps through sequence of operations needed to execute one ISA instruction
- For each cycle, produces state of every control signal in system
- Also, produces address of microinstruction to be executed next

Microinstructions

- Control the data path
- Five functional groups:
  - 9 control writing data from C bus to registers
  - 9 control driving registers onto B bus
  - 8 control ALU and shifter functions
  - 2 indicate memory read or write (MAR/MDR) (*not shown*)
  - 1 indicates memory fetch (PC/MBR) (*not shown*)

Reduce Control Signals for B Bus

- **B Bus**
  - Connected to all other registers except MAR
  - MBR has two enables for signed and unsigned versions
  - Becomes the right input to the ALU
  - Only want one register enabled
- Use a 4:16 Decoder

Microinstructions

- Control the data path
- Five functional groups:
  - 9 control writing data from C bus to registers
  - 4 control driving registers onto B bus
  - 8 control ALU and shifter functions
  - 2 indicate memory read or write (MAR/MDR) (*not shown*)
  - 1 indicates memory fetch (PC/MBR) (*not shown*)
Microinstruction Format

- Use two additional sets of signals
  - 9 for address of next microinstruction
  - 3 to determine how next microinstruction is selected (JAM)

- Total of 36 control signals (bits)

Control Signals for ALU Functions

<table>
<thead>
<tr>
<th>F_x</th>
<th>F_y</th>
<th>ENA</th>
<th>ENB</th>
<th>INV A</th>
<th>INC</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>A</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>B</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>A + B</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>A - 1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>B - A</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>-A</td>
</tr>
</tbody>
</table>

Figure 4.2. Useful combinations of ALU signals and the function performed

Determining Next Microinstruction

- Memory that holds the microprogram
- Contains 512 words, each a 36-bit microinstruction
- Each microinstruction specified its successor
  - Not executed in order stored in control store
- Accessing the microprogram
  - MicroProgram Counter similar to memory address register
  - MicroInstruction Register similar to memory data register
Determining Next Microinstruction

• When JAM bits are all zeros
  – NEXT_ADDRESS field is the next microinstruction

• When JAMN or JAMZ are set
  – The value of NEXT_ADDRESS
  – The value of NEXT_ADDRESS with high-order bit ORed with 1

• WHEN JMPC is set
  – MBR is bitwise ORed with 8 low-order bits of NEXT_ADDRESS field

Summary

• The microarchitecture level implements the ISA level above it

• The Mic-1 microarchitecture contains a datapath and control section

• Microinstructions are used to execute each ISA instruction