Topics

- Datapath registers
- Memory operations
- Microinstructions
- Control Section
- Datapath timing

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 H/W necessary to direct the datapath
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 (Miscellaneous)

- **OPC**
  - Name not given
  - A temporary (scratch) register without a preassigned use
- **H**
  - Holding register
  - Attached to the left input of the ALU
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
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

**Procedure A** is active

After **Procedure A** calls **Procedure B**

---

Stack Example

After **Procedure A** calls **Procedure B**

After **Procedure B** calls **Procedure C**
Stack Example

After Procedure B calls Procedure C

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

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 \textbf{words}
- PC contains a \textbf{byte address}

Block Diagram of Microarchitecture (Mic-1)

Control Section
Part of CPU containing the H/W necessary to direct the datapath
**Fetch-Decode-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
Control Store (ROM)

- 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

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)
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

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

- Cycle 1 starts here
- Shifter output stable
- Registers loaded instantaneously from C bus and memory on rising edge of clock
- Clock cycle 1
- New MPC used to load MIR with next microinstruction here
- Clock cycle 2
- Set up signals to drive data path
- ALU and shifter
- MPC available here
- Drive H and B bus
- Propagation from shifter to registers