Software Construction
Example CPU Simulator

Dhammika Elkaduwe

Department of Computer Engineering
Faculty of Engineering

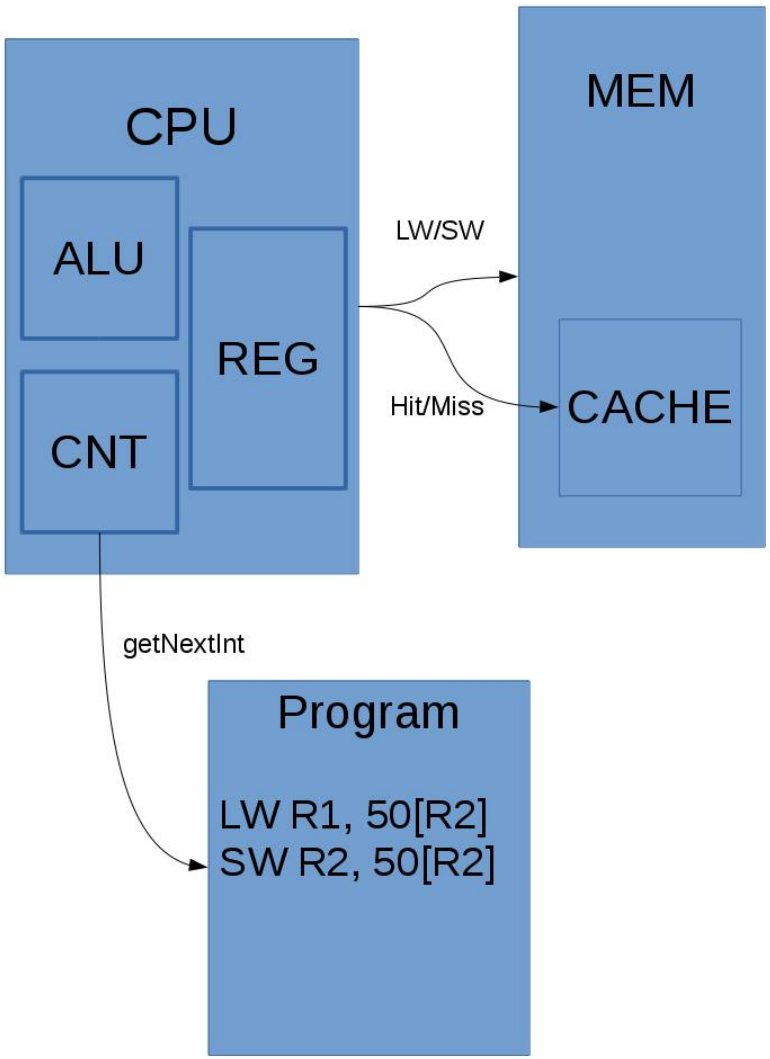
University of Peradeniya

ILOs

How to use the following concepts in a program;

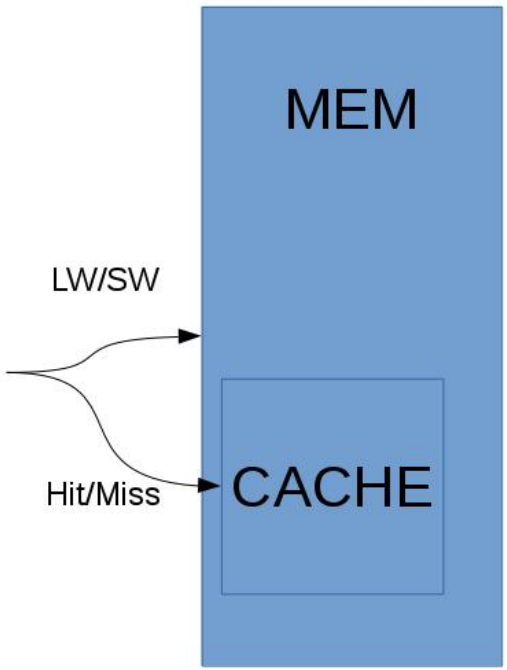
- interfaces
- classes
- abstractions
- testing

Specification



- Three main modules: memory, CPU and program
- Memory has a cache (which can be enabled)
- For simplicity:
  - Instructions are basically strings with *op code* and registers
  - Instructions are not stored in memory
  - memory only support *LW* and *SW* (no *LH, LB* etc)
  - Only few instructions (LW, SW, ADD, SUB, MUL, LI)
  - Only a data memory

Specification: Memory module



- Memory module only store data
- The size is determined at the time creating a machine (example: put things together like setting up a computer)
- Initially all memory locations are stored with zeros
- Trying to access memory address zero should throw an exception

How to start?

begin with defining interfaces!

```
import java.io.IOException;// IOExceptins

public interface MemInterface {
  int lw(String address) throws IOException;
  void sw(String address, int val) throws IOException;
  int cacheHits();
  int cacheMisses();
};
```

How to start?

begin with defining interfaces!

```
import java.io.IOException;// IOExceptins

public interface MemInterface {
  int lw(String address) throws IOException;
  void sw(String address, int val) throws IOException;
  int cacheHits();
  int cacheMisses();
};
```

Implementation memory module

see RAM.java

```
public class RAM implements MemInterface {

  int cacheHits, cacheMiss;
  int sizeOfRAM;

  Map<String, Integer> memory;

  public RAM(int sizeInMB) {
    cacheHits = cacheMiss = 0;
    sizeOfRAM = sizeInMB * 1024 * 1024;//size in bytes
    memory = new HashMap<String, Integer>();
}
```

Implementation memory module ...

see RAM.java

```
public int cacheHits() { return this.cacheHits; }
public int cacheMisses() { return this.cacheMiss; }

private boolean withinMemory(String address) throws
    IOException {
  int addr;
  try {
    addr = Integer.parseInt(address);
  } catch(NumberFormatException e) {
    throw new IOException();
  }

  return (addr > 0) && (addr < this.sizeOfRAM);
}
```

Implementation memory module ...

see RAM.java

```
public int lw(String address) throws IOException {
  if(!withinMemory(address)) throw new IOException();

  Integer val = memory.get(address);

  if(val == null) return 0;
  else return (int) val;
}

public void sw(String address, int val) throws IOException {
  if(!withinMemory(address)) throw new IOException();

  memory.put(address, new Integer(val));
}
```

Register file

Following is the specification for registers:

- there are 8 general purpose registers
- named as *R0, R1, R2 ... R7*
- out of which *R0* is read-only and hardwired to value 0.
- All registers are 32bits
- Except *R0* all other registers can be written/read. *R0* can only be read.

Registers definition

see RegBase.java, ReadWriteReg.java

```
abstract class RegBase {
  protected int value;
  abstract public int readReg() throws IOException;
  abstract public void writeReg(int value) throws
      IOException;
}
```

```
class ReadWriteReg extends RegBase {

  public ReadWriteReg() { value = 0; }

  public int readReg() throws IOException { return value; }

  public void writeReg(int value) throws IOException {
      this.value = value;}

}
```

Registers definition

see ReadOnlyReg.java

```
    class ReadOnlyReg extends RegBase {

      public ReadOnlyReg() { value = 0; }

      public int readReg() throws IOException { return 0; }

      public void writeReg(int value) throws IOException {
        throw new IOException("Can not write to a readonly reg");
      }
}
```

Things to note from the above code

- Note the use of abstract class
- Note the use of throwing exceptions
- Register file is a map from *String* to a *RegBase*.
- Note how the *CPUReg* functions are throwing the exceptions via the call stack.

Test the register file

see TestRegFile.java

```
public static void main(String [] args)
throws IOException {
   CPUReg registers = new CPUReg();

   System.out.println("R0 = " + registers.readReg("R0"));

   System.out.println("Write 5 to R1");
   registers.writeReg("R1", 5);
   System.out.println("R1 = " + registers.readReg("R1"));

   try {
     registers.writeReg("R0", 5);
   } catch(IOException e) {
     System.out.println("Goot the exception " + e);
     return;
   }
   System.out.println("****** Bad test failed");
```

Controller

Following instructions are available;

Syntax	Semantics
ADD <Reg1> <Reg2> <Reg3>	$Reg1 \leftarrow Reg2 + Reg3$
SUB <Reg1> <Reg2> <Reg3>	$Reg1 \leftarrow Reg2 - Reg3$
MUL <Reg1> <Reg2> <Reg3>	$Reg1 \leftarrow Reg2 * Reg3$
LI <Reg1> val	$Reg1 \leftarrow val$

Things to note about the instruction format

- takes at most 3 registers
- separated by spaces (not commas)
- different instructions take different number of arguments

Instructions

see Instruction.java

```
import java.io.IOException;

interface Instruction {
  public void execute(String [] agrs, CPUReg regFile)
  throws IOException;
}
```

Controller

see Controller.java

```
Map<String, Instruction> cnt;

public Controller() {

  cnt = new HashMap<String, Instruction>();

  class ADD implements Instruction {
    public void execute(String [] args, CPUReg regFile)
    throws IOException {
      regFile.writeReg(args[1],
      regFile.readReg(args[2]) + regFile.readReg(args[3]));
    }
  }
  cnt.put("ADD", new ADD());
  // do that for all instructions
```

Controller ...

see Controller.java

```
public void executeInstruction(String [] args, CPUReg
    regFile)
throws IOException {

  Instruction inst = cnt.get(args[0]);
  if(inst == null) throw new IOException("Cannot find
      instruction");

  inst.execute(args, regFile);
}
```

Testing the CPU

see TestCPU.java

```
static String [] prog = {
  "LI R1 10",
  "LI R2 20",
  "ADD R3 R1 R2",
  "MUL R3 R3 R3"
};
public static void main(String [] args) {
  try {
    CPUReg regFile = new CPUReg();
    Controller cnt = new Controller();
    for(int i=0; i < prog.length; i++) {
      String [] cmd = prog[i].split(" ");
      cnt.executeInstruction(cmd, regFile);
    }

    System.out.println("R3 = " + regFile.readReg("R3"));
  } catch(IOException e) { System.out.println(e); }
```

To do

- Change the code a bit and cause an exception
- Modify the exceptions throwing so that more information about the exception can be found.
- Program memory:
  - Program memory is where the instructions are stored
  - Define a suitable interface for the instruction memory
  - Implement it to read the instructions from a file.