Overview

- Branch Target Buffers
- RISC-V Infrastructure
- Final Project
Two-stage Pipeline with BTB

- BTB: Branch Target Buffer
- At fetch: Use BTB to predict next PC
- At execute: Update BTB with correct next PC
  - Only if instruction is a branch (iType == J, Jr, Br)

March 4, 2016  http://csg.csail.mit.edu/6.375

Next Address Predictor: Branch Target Buffer (BTB)

- BTB remembers recent targets for a set of control instructions
  - Fetch: looks for the pc and the associated target in BTB; if pc in not found then ppc is pc+4
  - Execute: checks prediction, if wrong kills the instruction and updates BTB (only for branches and jumps)

Even small BTBs are effective
Next Addr Predictor interface

interface AddrPred;
  method Addr nap(Addr pc);
  method Action update(Redirect rd);
endinterface

Two implementations:
Simple PC+4 predictor
Predictor using BTB

Simple PC+4 predictor

module mkPcPlus4(AddrPred);
  method Addr nap(Addr pc);
    return pc + 4;
  endmethod

  method Action update(Redirect rd);
  endmethod
endmodule
BTB predictor

module mkBtb(AddrPred);
    RegFile#(BtbIndex, Addr) ppcArr <- mkRegFileFull;
    RegFile#(BtbIndex, BtbTag) entryPcArr <- mkRegFileFull;
    Vector#(BtbEntries, Reg#(Bool))
        validArr <- replicateM(mkReg(False));
    function BtbIndex getIndex(Addr pc) = truncate(pc>>2);
    function BtbTag getTag(Addr pc) = truncateLSB(pc);
    method Addr nap(Addr pc);
        BtbIndex index = getIndex(pc);
        BtbTag tag = getTag(pc);
        if(validArr[index] && tag == entryPcArr.sub(index))
            ppcArr.upd(index, redirect.nextPc);
        endmethod
    method Action update(Redirect redirect);
        if(redirect.taken)
            begin
                let index = getIndex(redirect.pc);
                let tag = getTag(redirect.pc);
                validArr[index] <= True;
                entryPcArr.upd(index, tag);
                ppcArr.upd(index, redirect.nextPc);
            end
        else if(tag == entryPcArr.sub(index))
            validArr[index] <= False;
        endmethod
endmodule

BTB predictor update method

redirect input contains a pc, the correct next pc and whether the branch was taken or not (to avoid making entries for not-taken branches)

method Action update(Redirect redirect);
    if(redirect.taken)
        begin
            let index = getIndex(redirect.pc);
            let tag = getTag(redirect.pc);
            validArr[index] <= True;
            entryPcArr.upd(index, tag);
            ppcArr.upd(index, redirect.nextPc);
        end
    else if(tag == entryPcArr.sub(index))
        validArr[index] <= False;
endmethod
Multiple Predictors: BTB + Branch Direction Predictors

- **Next Addr Pred**
  - Need next PC immediately
  - Instr type, PC relative targets available

- **Br Dir Pred**
  - Correct mispred
  - Simple conditions, register targets available

- **Correct mispred**
  - Complex conditions available

- **Write Back**
  - Mispred instrs must be filtered

---

**RISC-V Processor**

SCE-MI Infrastructure
RISC-V Interface

interface Proc;
  method Action hostToCpu(Addr startpc);
  method ActionValue#(CpuToHostData) cpuToHost;
  interface MemInit iMemInit;
  interface MemInit dMemInit;
endinterface

typedef struct {
  CpuToHostType c2hType;
  Bit#(16) data;
} CpuToHostData deriving(Bits, Eq);

typedef enum {
  ExitCode, PrintChar, PrintIntLow, PrintIntHigh
} CpuToHostType deriving(Bits, Eq);
RISC-V Interface: cpuToHost

- Write mtohost CSR: csrw mtohost, rs1
  - rs1[15:0]: data
    - 32-bit Integer needs two writes
  - rs1[17:16]: c2hType
    - 0: Exit code
    - 1: Print character
    - 2: Print int low 16 bits
    - 3: Print int high 16 bits

```cpp
typedef struct {
    CpuToHostType c2hType;
    Bit#(16) data;
} CpuToHostData deriving (Bits, Eq);
```

RISC-V Interface: Others

- hostToCpu
  - Tells the processor to start running from the given address
- iMemInit/dMemInit
  - Used to initialize iMem and dMem
  - Can also be used to check when initialization is done
  - Defined in MemInit.bsv
SceMi Interface

Load Program

Bypass this step in simulation
Load Program

- Simulation: load with mem.vmh (fixed file name)
  - Copy <test>.riscv.vmh to mem.vmh

Start Processor

- Starting PC 0x200
Print & Exit

Get reg
c2hType:
1,2,3: print
0: Exit
Data == 0
  PASSED
Data != 0
  FAILED

Final Project
Overview

- Groups of 2-3 students
- Each group assigned to a graduate mentor in our group
- Groups meet individually with Arvind, mentor and me
- Weekly reports due before the meeting
  - Email to 6.375-admin@mit.edu and mentor

Schedule

<table>
<thead>
<tr>
<th>Week</th>
<th>Date</th>
<th>Deliverable</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Tuesday, March 15</td>
<td>Preliminary Proposal</td>
</tr>
<tr>
<td>0</td>
<td>Wednesday, March 16</td>
<td>Project Idea Presentation</td>
</tr>
<tr>
<td>1</td>
<td>Week of March 28</td>
<td>Final Proposal, High-Level Design and Test Plan</td>
</tr>
<tr>
<td>2</td>
<td>Week of April 4</td>
<td>Microarchitectural Description</td>
</tr>
<tr>
<td>3</td>
<td>Week of April 11</td>
<td>Implementation Status and Planned Exploration</td>
</tr>
<tr>
<td>4</td>
<td>Week of April 18</td>
<td>First Synthesis Results</td>
</tr>
<tr>
<td>5</td>
<td>Week of April 25</td>
<td>Simulation Demonstration</td>
</tr>
<tr>
<td>6</td>
<td>Week of May 2</td>
<td>FPGA Demonstration</td>
</tr>
<tr>
<td>7</td>
<td>Wednesday, May 11</td>
<td>Final Report, Final Presentation</td>
</tr>
</tbody>
</table>

Figure 1: Schedule of Deliverables
Project Considerations

- Design a complex digital system
- Choose an application that could benefit from hardware acceleration or FPGAs
- Application should be well understood
  - Find/implement reference software code
  - Look at past year projects on the website

FPGA IPs and Resources

- Many Xilinx related IPs are available in the BSV library
  - $BLUESPECDIR/BSVSource/Xilinx
  - BRAMs, DRAM, Clock generators/buffers, LED controller, HDMI controller, LCD controller
- Can wrap Verilog libraries/IPs in BSV code using importBVI
**BRAMs on FPGAs**

- Fast, small, on-chip distributed RAM on FPGA
  - 1 cycle access latency
  - 36Kbits × 1500 (approx) = ~6.75MB total
  - Up to 2 ports

![BRAM Diagram]

---

**BRAMs in BSV Library**

- 2 Ported BRAM server: mkBRAM2Server()
- Large FIFOs: mkSizedBRAMFIFO()
- Large sync FIFOs: mkSyncBRAMFIFO()
- Primitive BRAM: mkBRAMCore2()

```hs
import BRAM::*;
BRAMConfigure cfg = defaultValue;
cfg.memorySize = 1024*32; //define custom memorySize
// instantiate 32K x 16 bits BRAM module
BRAM2Port#(UInt#(15), Bit#(16)) bram <- mkBRAM2Server (cfg);
rule doWrite;
  bram.portA.request.put( BRAMRequest{
      write: True,
      responseOnWrite: False,
      address: 15’h01
      datain: data } );
```

March 4, 2016
DRAM on FPGA

- Large capacity (1GB on VC707)
- Longer access latency, especially random access
- BSV library at $BLUESPECDIR/BSVSource/
  Xilinx/XilinxVC707DDR3.bsv
  Misc/DDR3.bsv
  - Not officially in documentation
  - Example code will be given as part of Lab 6

![Diagram of DRAM on FPGA with DRAM, Off-chip FPGA, DRAM Controller IP, BSV Wrapper, and DDR3_Use](image)

DRAM Request/Response

- 512-bit wide user interface
- DDR Request:
  - Write: write or read
  - Byteen: byte enable mask. Which of the 8-bit bytes in the 512-bits will be written
  - Address: DRAM address for 512-bit words
  - Data: data to be written
- DDR Response:
  - Bit #(512) read data

![Diagram of DRAM Request/Response with 512-bit wide user interface](image)
**Indirect Memory Access**

- Host CPU load/stores data from host DRAM to PCIe device (FPGA)
  - Low bandwidth, consumes CPU cycles
  - Used in SceMi: ~50MB/s

**Diagram:**
- Host CPU
- Host DRAM
- Bus
- FPGA
- FPGA DRAM

---

**Direct Memory Access (DMA)**

- Host CPU sets up DMA engine
- DMA engine performs data transfer
  - High bandwidth, minimal CPU involved: 1-4 GB/s
  - Not supported by SceMi

**Diagram:**
- Host CPU
- Host DRAM
- Bus
- FPGA
- DMA Eng
- FPGA DRAM
Connectal
A SceMi Alternative

- Open source hardware/software co-design library
  - Generates glue logic between software/hardware
  - Supports DMA

- [https://github.com/cambridgehackers/connectal](https://github.com/cambridgehackers/connectal)

- Guest lecture next Wed on this