High Performance SMIPS Processor

Jonathan Eastep

May 8, 2005

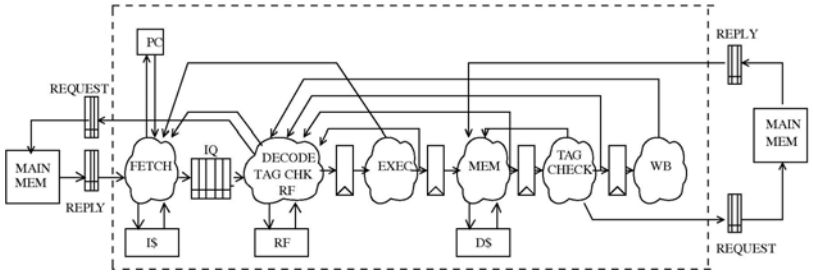
Objectives:

- Build a baseline implementation:
  - Single-issue, in-order, 6-stage pipeline
  - Full bypassing
  - ICache: blocking, direct mapped, 16KByte, 16bytes/line
  - DCache: blocking, direct-mapped, 32KByte, 16 bytes/line, write back/write allocate
  - Predict branches not taken and start decode early
  - Perfect L2 with 6 cycle latency
- Build a poweful ISA simulator/debugger with commit stage diffing against the BlueSpec model

Objectives: 4 Design-Point Study

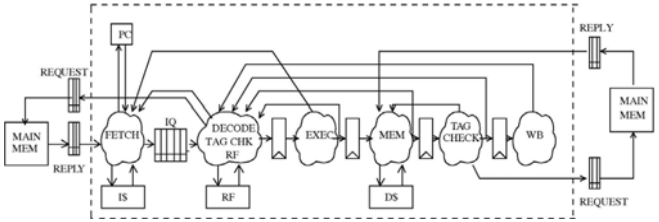
- A: Build a baseline, high performance, single-issue, in-order, pipelined SMIPS processor
- B: A + Incorporate early use of load data (before data cache tag check)
- C: B + simple predict backward branch taken, forward not taken
- D: B + 2-bit saturating counter branch prediction

Baseline RTL



Baseline 6-Stage Pipeline

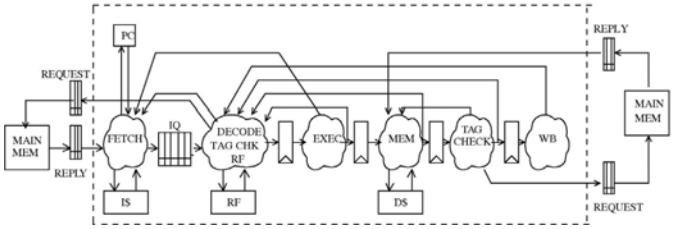
How Do We Do This In BlueSpec?



We represent each stage by a rule or a set of explicitly mutually exclusive rules

```
rule iCacheMissHandler( serviceICacheMiss() );
...
endrule
rule fetch ( !serviceICacheMiss() );
...
endrule
```

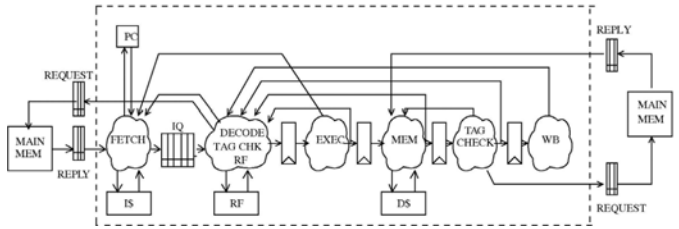
How Do We Do This In BlueSpec?



Same-cycle communication between rules accomplished using RWire

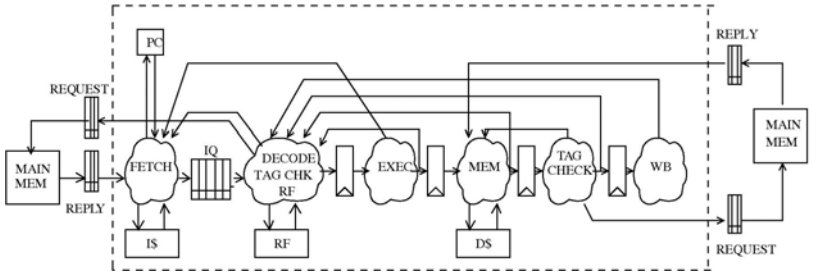
- i.e. iStream redirection for jump communicates the next pc to fetch in the same cycle that the jump is in decode
- i.e. bypassing values from later stages to decode

How Do We Do This In BlueSpec?



- Eliminate read/write conflicts with ConfigReg
- Keep rules from becoming unwieldy using rule splitting and functions

Another look at the RTL



When do we have to stall?

- When there's an instruct in decode that wants to source a register that a load in MEM or TAG CHECK writes to
- We squash and stall when there's a branch taken

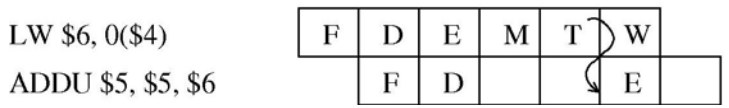
Vector-Vector Add Example

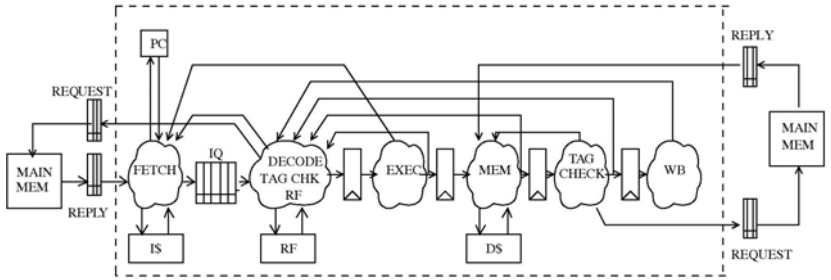
```
vvadd:
        LI      $2, 100
        LA      $3, vect1
        LA      $4, vect2
loop:
        LW      $5, 0($3)
        LW      $6, 0($4)
        ADDU    $5, $5, $6
        SW      $5, 0($3)
        ADDIU   $3, $3, 4
        SUBU    $2, $2, 1
        BNEZ    $2, loop
        ADDIU   $4, $4, 4
done:
        BEQ     $0, $0, done
        NOP
```

A look at the dynamic instruction stream...

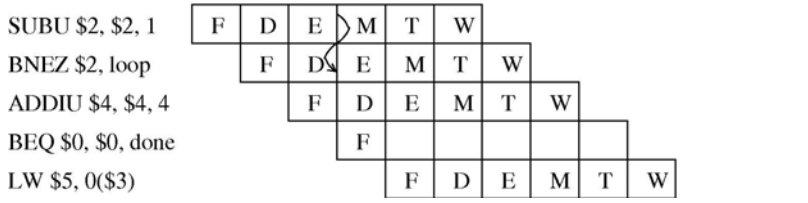
```
load hazard:
    LW      $6, 0($4)
    ADDU    $5, $5, $6

branch penalty:
    SUBU    $2, $2, 1
    BNEZ    $2, loop
    ADDIU   $4, $4, 4
--- BEQ     $0, $0, done
    LW      $5, 0($3)
```

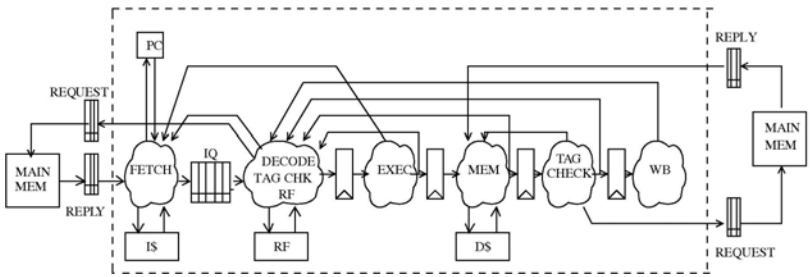
Load Hazard



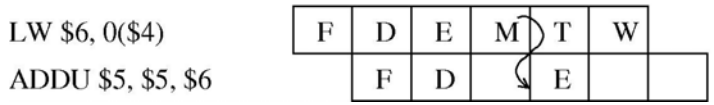


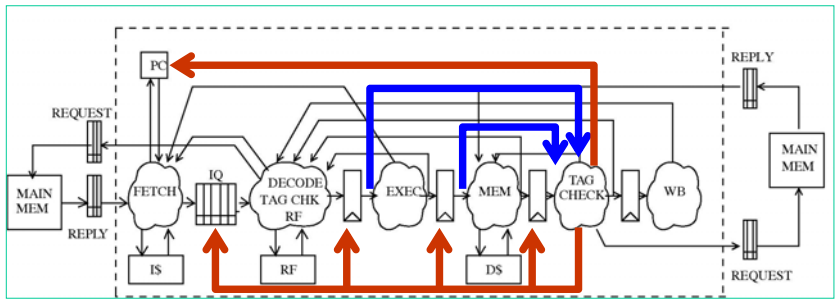
Branch Penalty





Load Hazard

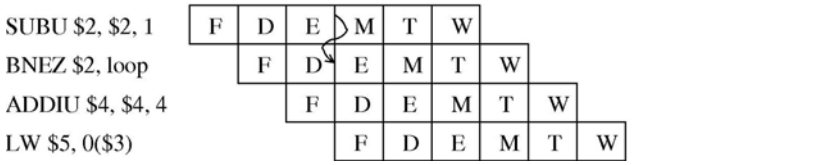


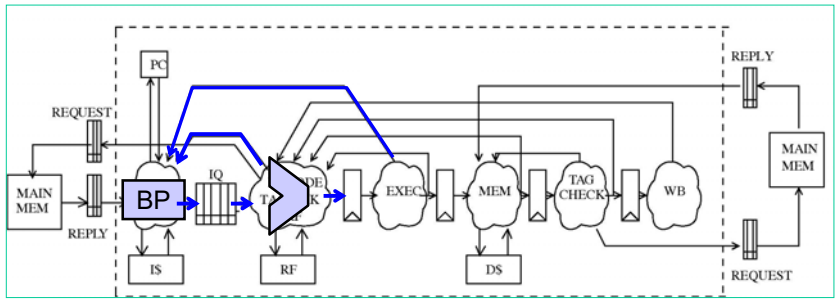


Cycle Time, Area, and IPC

- Without early use of load data:
  - PR area: 632,471um^2
  - PR timing: 3.68ns, 272MHz
  - IPC (vvadd): .47
- With early use of load data:
  - PR area: 672,744um^2
  - PR timing: 3.81ns, 263MHz
  - IPC (vvadd): .70

Branch Prediction

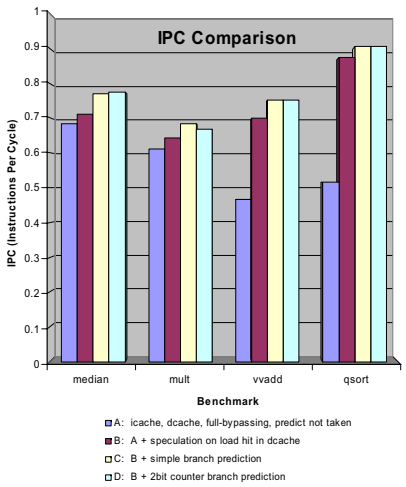


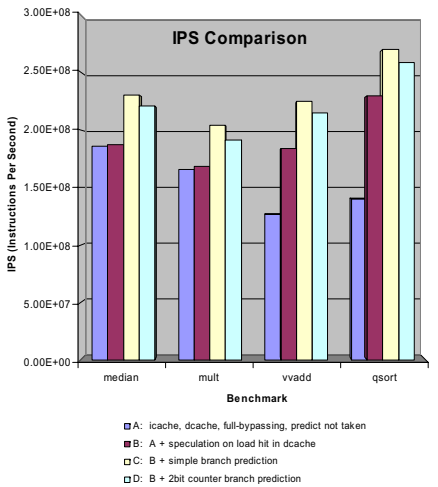


Cycle Time, Area, and IPC

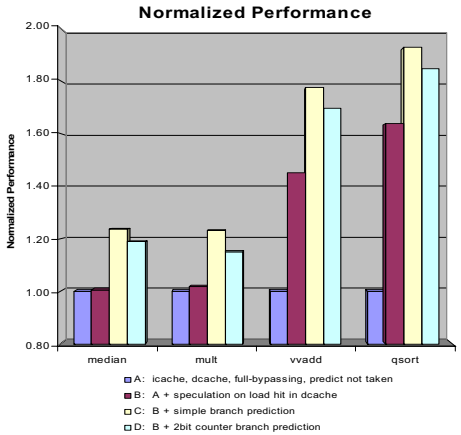
- With early use of load data and predict backward taken, forward not:
  - PR area: 638,645um^2
  - PR timing: 3.35ns, **298MHz**
  - IPC (vvadd): .75
  - IPC (qsort): **.905**
- With early use of load data and 2-bit saturating counter branch prediction (1024 2-bit entries):
  - PR area: 766,642um^2
  - PR timing: 3.51ns, 285MHz
  - IPC (vvadd): .75
  - IPC (qsort): .906

IPC vs. IPS





Normalized Performance



Summary

Take away points:

- It's not IPC that matters in the end, it's IPS when the ISA is fixed in your comparison
- To be fair, the benchmarks available didn't really push the branch prediction to illustrate when the bht is useful. I'm looking into that.
- IPC of .75-.90 on an in-order, single-issue machine with a clock of ~300MHz in TSMC .15um…in BlueSpec!

Questions?