Lecture #13

****
Review -- 1 min
****
Branch prediction
Branch Target Buffer
  1-cycle or 0-cycle branches
  combine with branch pred. algorithms
Multiple issue: superscalar v. VLIW
    pressure ILP -- pipeline depth * issue rate
    7-deep times 5 issue = 35 parallel instructions needed
      (v. branch per 5 instructions)

****
Outline - 1 min
****
Speculative Execution
Improving ILP – reduce control dependencies
  Traces – combine basic blocks into straight line code
  Compiler -- trace scheduling
    ~loop unrolling
    (harder b/c harder taken/not taken decision
    and less regular code)
  HW – trace cache

  Change control dependencies into data dependencies
    Annulling instructions

Limits to ILP

****
Lecture - 20 min
****

Speculative Execution (HW Support)

Ties together
  o Dynamic branch prediction

- o register renaming
- o in-order commit

Advantages of HW v. Software

1. Disambiguate memory references
2. better branch prediction
3. binary compatibility

Implementation:

1. Branch Target Buffer
   --> don't stall on instructions
   (even if condition depends on pending instructions)
2. renaming + in-order commit via Reorder Buffer
   --> don't commit until branch direction known

Key point: look at reorder buffer

	State	Value
1. ADDD ...		
2. SUB ...		
3. BNEZ R1, LOOP	T	NT
4. MULTD ...		
5. ADD ...		

Increasing ILP in SW and HW

Many techniques – we'll look at a few; given tools you now have, should be possible to evaluate these and others.

1) Traces – make straight-line runs longer than a basic block
   SW: Trace Scheduling
   HW: Trace Cache
2) HW: Annulling instructions – replace control dependencies with data dependencies (e.g., don't change PC!!)

Traces

Taken branches are worse than untaken branches

1) if no BTB, cant start fetch until destination known
2) if BTB, misses cause delays

3) Even with BTB hit, on superscalar need to fetch from 2 (or more) different cache lines

Example: 4-way superscalar issue with 2nd instruction in window a taken branch
How would you want BTB to work?

- should "hit" in previous cycle ("next cycle, you've got a branch so don't take straight line code
- should tell you "fetch 2 instructions from straight line code and 2 from destination <addr>

Problem - - how do you fetch both from one cache? → stall

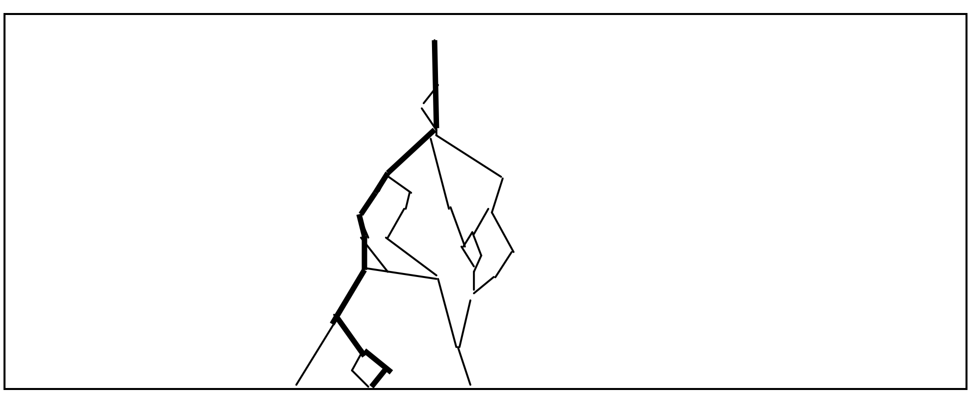
- generalizes into multiple branch prediction

Solution1: Software trace scheduling

Loop unrolling says -- common case is loop is taken
so copy code from after conditional branch
to before it

Can do same thing for non-loop conditional branches
called *Trace Scheduling*

picture: tree/graph of execution paths
Trace sceduling -- optimize for most likely path



Like loop unrolling, 2 steps
1. trace selection (~unrolling)
--> create straight line code down likely path

```
        example
                if A==0
                        // A1
                        <insr>; <insr>;...
                else
                        //A2
-->likely               <instr>; <instr>; ...
                        if B==0
                                // B1
-->likely                       <instr>...
                        else
                                //B2
                                <instr>...
                //C
                <instr>...


                BEQ R1, A1
        A2:     <instr>
                <instr>
                ...
                BNEZ R2, B2
        B1:     <instr>
                <instr>
                ...
        C:      <instr>
                ...


        A1:     <instr>
                ...
                j C

        B2:     <instr>
                ...
                j C
```

example: if untaken branch faster than taken branch make the straight line

2. trace compaction (~scheduling)
-->avoid hazards by moving instructions as early as
possible

Trace scheduling harder, less common than loop unrolling
o harder to tell by inspection if branch taken/not taken
(need high prob. branch to make it worth while)
o less regular code -- harder to move around

Technique 2: HW Trace Cache

1) As you execute instructions, feed them into a buffer
2) Store buffer of sequentially executed instructions in TC indexed by <start address, taken/not pattern>
3) BTB provides <start addr, taken/not pattern> → instructions from TC (may allow partial match)
4) Feed this set of "speculative straight line" instructions to issue unit

****
Admin - 3 min
****

Project Proposal: Feb 26

Friday – HW2b due at START OF CLASS

Next Wednesday – Doug Burger guest lecture

****
Lecture - 24 min
****

Annulling Instructions

Tech trends:

branches getting more expensive v. regular instructions
- -- superscaler/pipelined --> more cycles delay
- -- superscaler --> more instructions per cycle lost

Idea -- change control dependence into a data dependence

Conditional instruction evaluates condition (e.g. register === 0)
- if true -- do instruction
- if false -- no effect (no register update; no exception)

example: conditional move
```
    BNEZ R1, L
    MOV R2, R3
  L: ....
```
becomes
```
    CMOVZ R2, R3, R1
```

Most vendors have added to new instruction
- CMOV -- Alpha, MIPS, PowerPC, SPARC
- annul any following instr -- PA-RISC

(-->probably a good idea)

Limits to ILP

Motivation: in first lecture -- electrical engineers --> 35% performance

per yer via better processes
Past few years -- 55% per year -- the 20% comes from taking advantage of better ILP

pipelining (SPARC1, MIPS R2000)
superpipelining (Alpha 21064, Mips R4000)
superscaler (IBM Power, Ultrasparc, Alpha)
...

What will it take to get more ILP?
Where does it end?

ILP Studies

Goal: Find out how much parallelism
Difficulty: depends on

1. Benchmarks used
2. hardware assumptions
3. compiler assumptions
   (e.g. no conditional execution instruction)

This study

Aggressive HW model

1. Infinite Functional Units (1 cycle latency for all)
2. Infinite 1 cycle cache
3. Inifinite renaming registers
   -- not WAR, WAW hazards
4. Perfect branch and jump prediction
5. Infinite instruction issue
6. perfect alias analysis in memory

Picture: you have trace of entire program
(you know whether all branches are taken/not taken before program starts running)
load the trace into an infinite tomasulo engine

at every cycle, execute all instructions that no longer have data hazards

Notice, this is *really* aggressive, but, surprizingly, we can imagine a machine that does even better – HW2b shows a technique that executes instructions before data dependency clears.

Also, standard MIPS compiler (could imagine a compiler that does high-level optimizations to get rid of unnecessary calculations, etc.)
As we restrict hardware, compiler limitations may become more important

Start with aggressive hardware then make more realistic assumptions.

SLIDES:

- upper limit to ILP (fig 4.38 p 319)
  - ideal processor
- window (fig 4.40 p 323)
  - dependencies (issue)^2

    Think about logic for renaming registers
    This is part of why machines will issue "2FP, 2 integers, and a branch, but not 5 integer instructions – limit dependencies to check)

- branch and jump prediction (fig 4.42)
  - "ideal processor" – results of all branches known before first instruction executed!

    For restricted cases, rule is that if you mispredict you have to stop until dependency resolved (no other penalty, just restricts parallelism)

- register impact (fig 4.44 p328)

limit number of renaming registers

surprizingly much impact from name dependencies – need to look at lots of live variables to get full parallelism
(today's processors <32)

alias (4.46 p 330)

conclude: need either sophisticated compiler analysis or dynamic HW

realistic (vy aggressive) (4.48 p 332)

- <= 64 instructions issued per clock
  - of any type (e.g. all Loads? OK)
- selective predictor with 1K entries; 16-entry return predictor
- perfect disambiguation of memory refs (HW)
- register renaming w/ 64 integer and 64 FP
- vary window size
- perfect cache; 1-cycle latencies for all instructions (XXX not realistic)
-

Bottom line:

FP limited by how many instructions we can issue
Integer: limited by branch prediction

Case study: powerPC604 and PPro

<slides>

****
Summay - 1 min
****