EE282
Computer Architecture

Lecture 6: Branch Prediction and ILP

October 16, 2001

Marc Tremblay
Computer Systems Laboratory
Stanford University
tremblay@csl.stanford.edu

Assignment

- Before Lecture on Thursday 10/18
  - Finish reading H&P Chapter 4

Today's Lecture

- More complex pipelining
  - the MIPS R4000 pipeline
  - Pentium 4
- Branch prediction
  - control hazards
  - static prediction
  - history table
  - multi-bit prediction
  - branch target table
  - two-level prediction
- Instruction-level parallelism
  - covert vs overt parallelism
  - parallelism in hardware and software
  - a question of *scheduling*

Example: R4000 Pipeline

- 8 Stages
  - F1 - select IP, start I$ access
  - F2 - complete I$ access
  - D - decode, register access, check I$ tag
  - A - ALU operation
  - M1 - start D$ access
  - M2 - complete D$ access
  - M3 - check D$ tag
  - W - write back result to register
- Memory access takes 3 cycles

Pipeline Diagram

Instr	1	2	3	4	5	6	7	8	9	10
i	F1	F2	D	A	M1	M2	M3	W		
i+1		F1	F2	D	A	M1	M2	M3	W	
I+2			F1	F2	D	A	M1	M2	M3	W
i+3				F1	F2	D	A	M1	M2	M3
i+4					F1	F2	D	A	M1	M2
i+5						F1	F2	D	A	M1
i+6							F1	F2	D	A
i+7								F1	F2	D

What are the bypass paths?

	A	B	C	D	E	F	G	H
From Stage								
To Stage								
To Instruction								

Characteristics of this pipeline

- How many comparators are needed to implement the forwarding decisions?
- What instruction sequences will still cause stalls?
- What is the branch delay?
- What is the load delay?

Example 2: Pentium 4

- Pipeline diagram (shown during lecture, available soon on website)

Branch Prediction

- Depending on use, some branches are very predictable
  - loops
    - TTT...TN
  - limit checks
    - almost always pass
- Some are not very predictable
  - data dependent dispatch with equally likely cases
- Types of predictors
  - static
  - history
  - multi-bit history
  - pattern

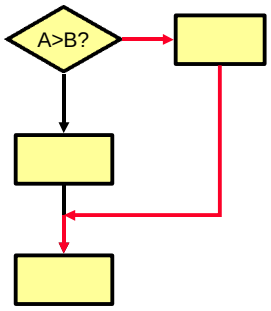
```
for(j=0;j<30;j++) {
   …
}


switch(mode) {
   case 1: …
   case 2: …
   default: …

…
if(a > limit) {
  …
}
```

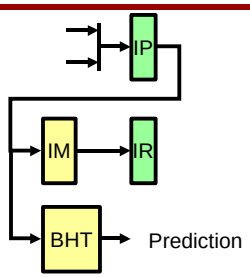
Static Prediction

- Assign a preferred direction to each branch
  - e.g.,
    - BNEZ_T (predict taken)
    - BNEZ_N (predict not taken)
- Base on
  - program analysis
    - loops tend to be taken
  - profiling of the program
    - but it may be data dependent



Dynamic Predictors

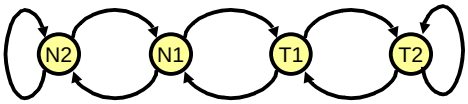
- Branch history table
  - indexed by IP
  - stores last direction each branch went
  - may indicate if last instruction at this address was a branch
  - table is a cache of recent branches



Multi-bit predictors

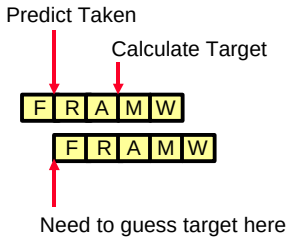
- A 'predict same as last' strategy gets two mispredicts on each loop
  - Predict NTTT…TTT
  - Actual TTTT…TTN
- Can do much better by adding *inertia* to the predictor
  - e.g., two-bit saturating counter
  - Predict TTTT…TTT

```
for(j=0;j<30;j++) {
   …
}
```



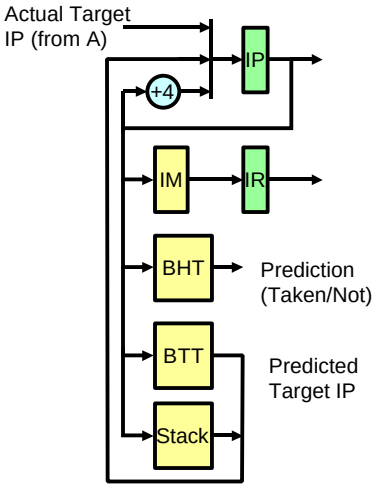
Branch Target Tables

- Need to know where to go if the prediction is 'taken'
  - predict the target along with the direction
- May use different target prediction strategy for different types of branches
  - subroutine returns



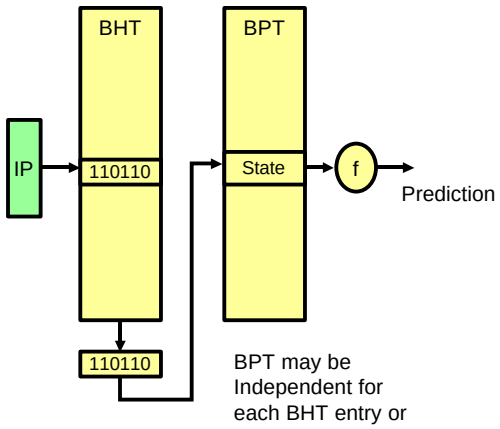
Branch Target Prediction (2)

- Use current IP to index a cache of next IPs
- Use a push-down stack to record subroutine return addresses
- The ISA can give *hints* about where you're going
- Compaq Alpha has 4 instructions with identical ISA behavior
  - JMP, JSR, RET, JSR_COROUTINE
  - specify predictor's use of stack
  - include *hint* of target address
  - JMP R31, (R3), hint



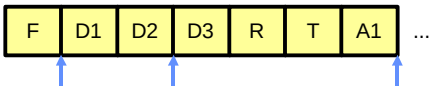
Branch Pattern Tables (Two-Level Predictors)

- History gives a pattern of recent branches
  - e.g., TTNTTNTTN
  - what comes next?
- Predict next branch by looking up history of branches for a particular pattern
- Two-level predictor
  - first level - find history (pattern)
  - 2nd level - predict branch for that pattern
- Correlating predictors



Branch Performance

Consider a modern pipeline with a long decode stage



Predict

Discover its a branch

Resolve direction and calculate target

Penalty for mispredicted branch is _____

If 10% of instructions are branches what is CPI
With no prediction?
With 70% accurate prediction (static)
With 85% accurate prediction (2-bit)
With 95% accurate prediction (2-level)

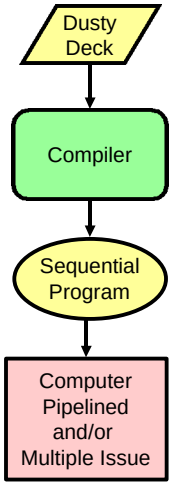
Alternatives to Prediction

- Predication
  - guard instructions with predicate, cancel if false
  - when is this a good idea?
    - What length conditional segment?
- Delay slots
  - make branch delay explicit
  - exposes implementation in the ISA
    - makes life difficult for future implementations
  - compiler tries to fill delay slot with useful instruction (possibly predicated)
- ISA support
  - *hints* to the predictor
    - let the compiler pass along the information it has
  - separate the components of the branch
    - target address calculation
      - prepare to branch (Tera)
    - determining direction
    - actually branching

To Unity and Beyond (Below 1 CPI)

- No reason to limit pipelines to process 1 instruction per cycle
- Can predict next several (2-6) instructions and execute them *simultaneously*
- Need to resolve data dependencies
- Several approaches
  - VLIW - compiler schedules instructions
  - Multi-issue - issue instructions in order, but in parallel
  - Superscalar - issue instructions out of order

Parallelism
Overt vs Covert (1 of 2)



Covert approach

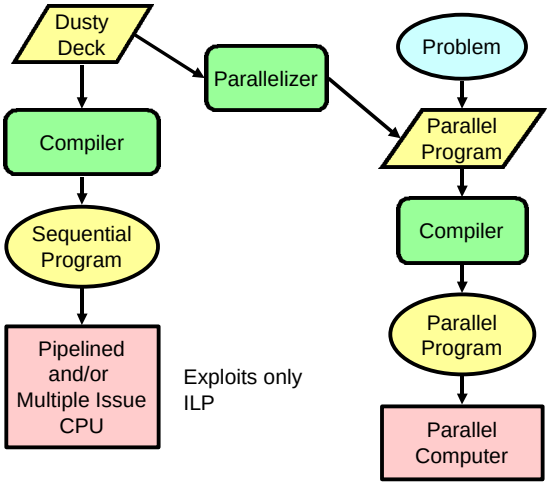
Programmer doesn't see parallelism, just speed

Compiler may discover parallelism (e.g., in loops) and/or reorganize code to remove dependencies

Code generated usually has *sequential semantics* for compatibility with ISA

Hardware executes in parallel while preserving sequential semantics

Parallelism
Overt vs Covert

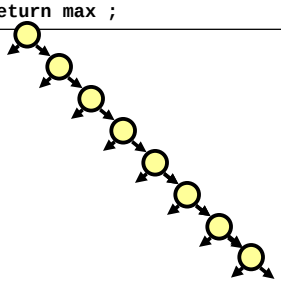


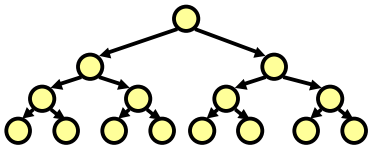
Problem vs Program

Find the maximum element of an array, a

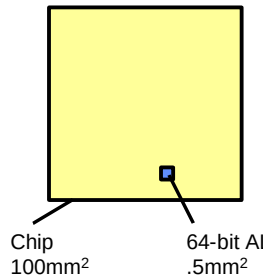
```
max = a[0] ;
for(i=1;i<n;i++) {
  if(a[i] > max) max = a[i] ;
}
return max ;
```

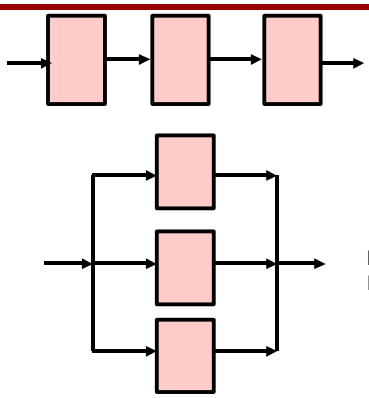
```
for(i=1;i<=logn;i++) {
  m = 1<<(logn-i) ; p = i&1 ; q = p^1 ;
  forall(j=0;j<m;j++){
    a[j,p] = max(a[2*j,q],a[2*j+1,q]);
  }
}
return a[0,p] ;
```





Parallelism and Hardware





Technology gives us *lots* of function units

They get only slightly faster each year

The wires get slower

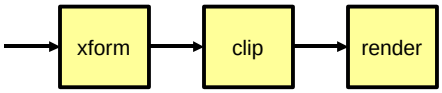
Pipeline or replicate at bit, word, vector, subroutine levels

Parallelism and Software

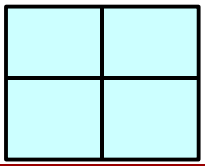
Independent Operations (ILP)

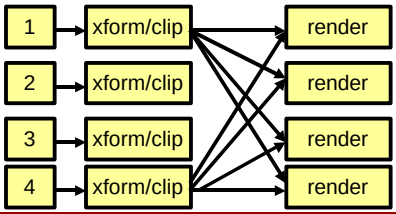
```
a = (b + c) * (d + e + f) ;
```

Function decomposition



Domain decomposition





How Are We Going to Get There?

- Microarchitecture support for lowering the CPI < 1.0
  - Issuing more than 1 instruction per cycle
  - Speculation beyond going past a branch
  - Latency issues
  - Memory hierarchy needed to support aggressive compute engines
    - Caches
    - TLBs
    - Memory banks
    - Etc.