## 4 Eliminate the Branch

**Predicated execution**

- instructions are executed conditionally
  - set a condition
  - test a condition & execute the instruction if the condition is true
  - if the condition is false, don't write the instruction's result in the register file
  - i.e., instruction execution is ***predicated*** on the condition
- replaces conditional branch (expensive if mispredicted)
  - can fetch & execute instructions on both branch paths
    - all instructions depend on the same condition
  - eliminates branch latencies by changing a **control hazard** to a **data hazard**
- increases **instruction level parallelism (ILP)**: the number of independent instructions (see the next slide)
- hard to fit into an existing ISA



## Instruction Level Parallelism

**Instruction level parallelism (ILP)**

- instructions executing at the same time
  - from instruction overlap in a pipeline
  - from issuing & executing instructions in parallel (later, with multiple instruction issue)
- can be exploited when instructions are independent of each other, for example,
  - two instructions are **independent** if their operands are different
  - an example of independent instructions, i.e., ILP

```
ld r1, 0(r2)
or r7, r3, r8
```

  - two instructions are **dependent** if one of the source operands of the second is computed by the first
  - an example of dependent instructions, i.e., **no** ILP

```
ld r1, 0(r2)
or r7, r1, r8
```

- important for executing instructions in parallel and hiding latencies

## 4 Eliminate the Branch

An example of predicated execution using a **conditional move**:

```
    bnez R1, L     ⇒       cmovz R2, R3, R1
    mov R2, R3           L:
L:
```

Have eliminated an expensive branch instruction

An example that executes both paths:

if (A>B) then X = 4; else X = 0

(condition in R1; A in R2; B in R3; X in R4; 4 in R5)

```
sgt R1, R2, R3       !set the condition
add R4, R0, R0       !execute the false path
cmovlbs R4, R5, R1   !conditionally move the
                       true condition
```



## 4 Eliminate the Branch

**Advantages** of predicated execution

+ no branch hazard
+ creates straightline code; therefore better prefetching of instructions

  **prefetching** = fetch instructions before you need them to hide cache miss latency
+ more independent instructions, therefore better code scheduling

**Disadvantages** of predicated execution

- if the condition is false, true path instructions are still executed
- may be hard to add predicated instructions to an existing instruction set
- additional register pressure
- instructions cannot generate exceptions because you might not execute that path
- good branch prediction might get the same effect

## Today's Branch Prediction Strategy

**Static and dynamic branch prediction work together**

**Predicting**

- gshare
  - MIPS R12000 (2K entries, 11 bits of PC, 8 bits of history)
  - UltraSPARC-3 (16K entries, 14 bits of PC, 12 bits of history)
- correlated branch prediction
  - Pentium III (512 entries, 2-bit)
  - Pentium Pro (4 history bits)
- combined branch prediction
  - Alpha 21264 has a combination of local (1K entries, 10 history bits) & global (4K entries) predictors
- 2 bits/every 2 instructions in the I-cache (UltraSPARC-1)

**BTB**

- 512 entries, 4-way set associative (Pentium Pro)
- 32 entries, 2-way set associative (Pentium Pro)
  - no BTB; target address is calculated (R10000, Alpha 21164, UltraSPARC-3)
- next address every 4 instructions in the I-cache (Alpha 21264, UltraSPARC)
  - "address" = I-cache entry & set



## Today's Branch Prediction Strategy

**Return address stack** (Alpha 21264, R10000, Pentium Pro, UltraSPARC-3)

**Resume cache** (R10000, UltraSPARC-3)

**Predicated execution** (Alpha 21264 (conditional move), IA-64: Itanium (full predication))