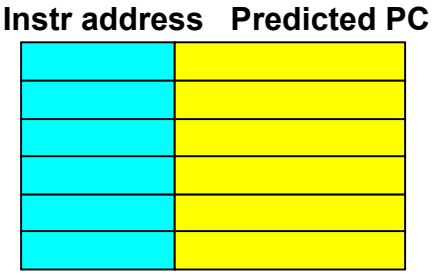
Dynamic Branch Prediction (Continued)

Branch Target Buffer

Branch prediction buffers contain prediction about whether the next branch will be taken (T) or not (NT), but it does not supply the target PC value. A Branch Target Buffer (BTB) does this.



BTB is a cache that holds

(instr addr, predicted PC)

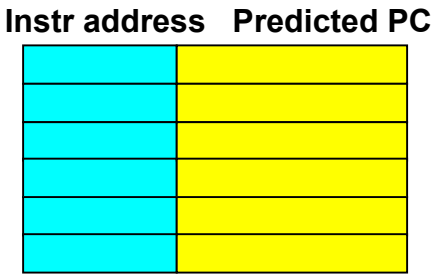
for every taken branch

The control unit looks up the branch target buffer during the "F" phase. The target PC is found out even before it is known to be a branch instruction.

BTB hit and miss

(BTB Hit) Implements *zero-cycle branches*

(BTB Miss) Target PC is computed and entered into the target buffer.



BTB is managed (by the control unit) as a regular cache. With a larger BTB there are fewer misses and the performance improves.

Predication

Predication mitigates the hassle of handling conditional branches in pipelined processors.

Example

```
If <condition>
    then <do this>
    else <do that>
end if
```

```
        If <condition> branch to L1
        do that;
        branch to L2
L1:     do this
L2:     exit
```

Using predication, we can translate it to

<p>	<do this: step 1>
<p>	<do this: step 2>
<not p>	<do that: step 1>
<not p>	<do that: step 2>

Each instruction is executed when a predicate is true.

Every instruction enters the pipeline, but results are suppressed if the predicate is false.

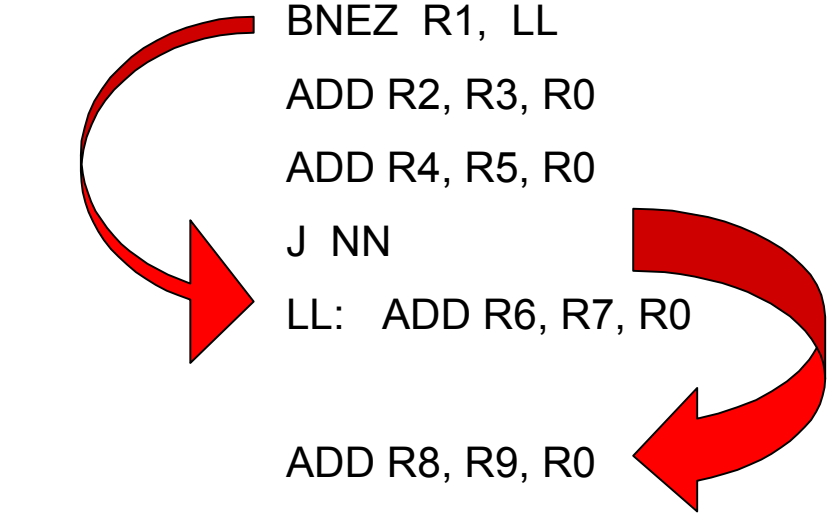
Predication eliminates branch prediction logic, and allows better bundling of instructions, and sometimes better parallelism. But it needs extra space in instructions.

Predication is used in Intel's IA-64 architecture, ARM and some newer processors

Examples

```
if (R1==0) {                    BNEZ  R1,  LL
    R2 = R3                     ADD R2, R3, R0
    R4 = R5                     ADD R4, R5, R0
 } else {                       J  NN
   R6 = R7                      LL:   ADD R6, R7, R0


   R8 = R9                      ADD R8, R9, R0
}                               NN:
```



CMOVZ R2, R3, R1 (conditional move: if R1=0 then R2=R3)

CMOVZ R4, R5, R1 (conditional move: if R1=0 then R4=R5)

CMOVN R6, R7, R1 (conditional move: if R1≠0 then R6=R7)

CMOVN R8, R9, R1 (conditional move: if R1≠0 then R8=R9)

More examples of predication:

```
if (R1 == R2)   {            CMEQ R1, R2, P2, P3
      R3 = R4                {if R1=R2 then set P2 else set P3)
} else  {           <P2>  ADD R3, R4, R0
      R5 = R6       <P3>  ADD R5, R6, R0
}
```

Instruction Level Parallelism

Instruction streams are *inherently sequential*. But superscalar processors are able to handle multiple instruction streams in parallel. To utilize the available parallelism, it is important to study techniques for extracting Instruction Level Parallelism (ILP). Superscalar processors rely on ILP for speedup.

Example of Superscalar Processing

instr	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*
1 (integer)	**F**	**D**	**X**	**M**	**W**				
2 (FP)	**F**	**D**	**X**	**M**	**W**				
3 (integer)		**F**	**D**	**X**	**M**	**W**			
4 (FP)		**F**	**D**	**X**	**M**	**W**			
5			**F**	**D**	**X**	**M**	**W**		
6			**F**	**D**	**X**	**M**	**W**		

If N instructions are issued per cycle then the ideal CPI is 1/N. However, the probability of hazards increases, and it makes the CPI lower than 1/N.

For example, by scheduling multiple unrelated instructions in parallel, ILP improves, and the instruction throughput also improves.

ILP can be improved at run time, or at compile time. Run time methods of bundling unrelated instructions rely on the control unit, and increases the cost of the machine.

Very Large Instruction Word (VLIW) Processors

In VLIW, the compiler packages a number of operations from the instruction stream into one large instruction word.

Integer	integer	FP	FP	memory	memory	branch

Instruction size may be as large as 100-150 bits.

Needs better methods for identifying parallelism in the instruction stream.

EPIC utilizes this idea in the IA-64 specifications.

Hardware Speculation

Superscalar machines often remain under-utilized.

Hardware speculation helps improve the utilization of multiple issue processors, and leads to better speed-up.

Speculative Execution

Execute codes before it is known that it will be needed.

Schedule instructions based on speculation

Save the result in a Re-Order Buffer (ROB)

Commit the results when they are correct,

otherwise discard them.

Example 1

```
even = 0;  odd= 0;  i=0;
while (i < N) {
k := i*i
      if (i/2*2 == i)        even = even + k
            else       odd = odd + k
      i= i+1
}
```

The Strategy

To improve ILP using speculation, until the outcome of branch is known, evaluate both (even + k) and (odd + k) possibly in parallel, on a two-issue machine, and save them in ROB

Problems and Solutions

What if a speculatively executed instruction causes an exception and the speculation turns out to be false? It is counterproductive! Consider this:

if (x > 0) z = y / x;

Suppose x = 0. The program speculatively executes y/x causing an exception! This leads to the failure of a correct program!

A set of status bits called poison bits are attached to result registers. Poison bits are set by speculative instructions when they cause exceptions, but exception handling is disabled. The poison bits cause an exception when the speculation is correct.

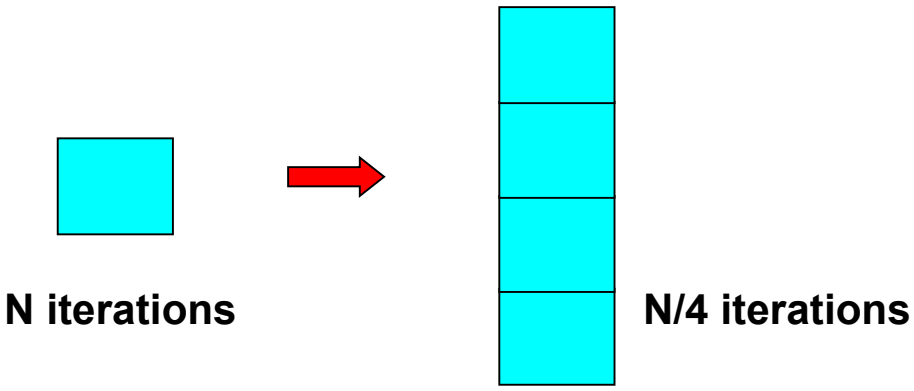
Compiler Support for better ILP

Loop Unrolling

Consider the following program on the MIPS processor.

loop:	R1 := M[i];	1
	R2 := R1+99;	3
	M[i] := R2;	5
	i := i-1;	6
	if (i ≠ 0) then goto loop	8
	branch delay slot	9

If the branch penalty is 1 cycle, then every iteration of the loop takes 9 cycles. Unrolling of the loop unfolds additional parallelism.



The Unrolled Loop

	Before optimization	***After Optimization***
loop:	R1 := M[i];	R1:= M[i];
	R2 := R1+99;	R3:= M[i-1];
	M[i] := R2;	R5:= M[i-2];
	R3 := M[i-1];	R7:= M[i-3];
	R4 := R3+99;	R2:= R1+99;
	M[i-1] := R4;	R4:= R3+99
	R5 := M[i-2];	R6:= R5+99
	R6 := R5+99;	R8:= R7+99
	M[i-2] := R6;	M[i]:= R2
	R7 := M[i-3];	M[i-1]:=R4
	R8 := R7+99;	M[i-2]:=R6
	M[i-3] := R8;	M[i-3]:=R8
	i := i - 4;	i:= i - 4
	if (i≠0) the goto loop;	if (i≠0) the goto loop;

- Estimate the performance improvement now.
- Branches may marginally degrade performance.
- Easy to schedule on superscalar architectures.