

Predication

Introduction

1. Some branches are difficult to predict even with sophisticated branch predictors
 - Compiler can help avoid some of these branches

Predication

1. Branch prediction: Guessing where program is going
 - No penalty if correct (0 instructions)
 - Huge penalty if incorrect (12-stage pipeline with 4 instructions/cycle -> 48)
2. Predication: Do work of both directions (taken/not taken)
 - Waste up to 50%
 - Throw away work from wrong path
3. Loop: Typically use branch prediction (more loops -> more predictable)
 - Using predication will waste many cycles and makes little sense
4. Function call/return: Predict since the branch is always taken
5. Large if-then-else: Predict
 - If length of branch is 100 cycles either way, we will waste these 100 cycles compared to the ~50 we wasted due to a branch miss in the above example
6. Small if-then-else: Predication unless we're certain the branch predictor will be highly accurate
 - If length of branch is 5 cycles either way, we always waste 5 cycles with predication
 - Branch prediction results in 50 wasted cycles in the case of a missed prediction
 - 100% of time we waste 5 cycles with predication -> 5
 - 10% of time we waste 50 cycles with branch prediction -> 5
 - This is the break-even point
 - If size of branch is smaller or branch prediction performs worse, this biases the result in favor of predication

If Conversion

1. Consider the following program:

```
if(cond)
{
    x = arr[i];
    y++;
}
else
{
    x = arr[j];
    y--;
}
```

2. This could be rewritten as follows:

```
x1 = arr[i];
x2 = arr[j];
y1 = y + 1;
y2 = y - 1;
x = cond ? x1 : x2; // Now we have two branches
y = cond ? y1 : y2; // Now we have two branches
```

3. Need some sort of conditional move instruction in hardware
 - MOV x, x1, cond

Conditional Move

1. MIPS
 - MOVZ Rd, Rs, Rt -> if(Rt == 0) Rd = Rs;
 - MOVN Rd, Rs, Rt -> if(Rt != 0) Rd = Rs;
2. x86
 - CMOVZ, CMOVNZ, CMOVGT, ... -> if(cc) Dst = Src;
 - Based on all of the condition codes

MovZ MovN Quiz

1. Consider the following program:

```
BEQZ R1, Else
ADDI R2, R2, 1
B End
Else: ADDI R3, R3, 1
End:
```

2. How do we rewrite this with conditional moves?

```
ADDI R4, R2, 1
ADDI R5, R3, 1
MOVN R2, R4, R1
MOVZ R3, R5, R1
```

MovZ MovN Performance

1. 80% branch accuracy, 40-instruction penalty on missed branch
 - When the branch is not taken we execute 3 instructions
 - When the branch is taken we execute 2 instructions
 - Assuming the branch is balanced, we execute an average of 2.5 instructions
 - $2.5 + 0.2 * 40 = 10.5$ instructions
2. If-conversion costs 4 instructions

MovZ MovN Performance Quiz

1. Consider the following program:

```
BEQZ R1, Else
ADDI R2, R2, 1
B End
Else: ADDI R3, R3, 1
End:
```

2. This can be rewritten as follows using conditional moves:

```
ADDI R4, R2, 1
ADDI R5, R3, 1
MOVN R2, R4, R1
MOVZ R3, R5, R1
```

3. Assumptions:
 - Branch is taken 50% of the time
 - 30-instruction misprediction penalty
4. If-conversion is better when prediction accuracy is below...
 - $2.5 + \text{Accuracy} * 30 = 4$
 - $\text{Accuracy} = (4 - 2.5) / 30 = 0.05$ (5%)

- If prediction accuracy is below 95%, predication is better

MOVc Summary

1. Require compiler support for if-conversion
 - Not fully backwards-compatible (old code won't use these instructions)
2. Removes hard-to-predict branches
3. More registers needed
 - Keep results from both parts
4. More instructions executed
 - Executing both paths
 - MOVc to select actual results
5. Full predication: Can change ISA such that all instructions are conditional
 - Requires extensive support in instruction set
 - Every instruction computes a result but may or may not store it

Full Predication Hardware Support

1. MOVcc
 - Separate opcode
2. Full Predication
 - Add condition bits to every instruction
 - Intel Itanium: (qp) ADD R1, R2, R3
 - qp: Qualifying predicate
 - Instruction has 41 bits, low 6 bits are 1-bit condition registers

Full Predication Example

1. Applying full predication to the example program above
 - MP.EQZ P1, P2, R1 (sets qualifying predicate bits)
 - (p2) ADDI R2, R2, 1
 - (p1) ADDI R3, R3, 1
2. Don't require additional registers
3. No overhead with moving between registers since we're storing directly in the result

Full Predication Quiz

1. Consider the following program:

```
BEQZ R2, ELSE
ADD R1, R1, 1
B DONE
ELSE: ADD R1, R1, -1
DONE:
```

2. Convert to full predication version:

```
MP.EQZ P1, P2, R2 (P1 -> R2 == 0, P2 -> R2 != 0)
(P2) ADD R1, R1, 1
(P1) ADD R1, R1, -1
```

3. Assuming
 - CPI = 0.5 without mispredictions
 - Misprediction penalty is 10 cycles
4. Do if-conversion if BEQZ prediction is < % correct
 - $0.5 * (2 + 3) / 2 + \text{Accuracy} * 10 = 3 / 2$

- $1.25 + \text{Accuracy} * 10 = 1.5$
- $\text{Accuracy} = 0.25 / 10 = 1 / 40 = 2.5\%$
- Require 97.5% accuracy

Conclusion

1. Branch prediction and predication handle control hazards
 - Need to consider how data hazards are handled while maintaining instruction flow in our pipeline