Section 2: Performance & Instruction Encoding

CS 152
Spring 2009

TA: Scott Beamer

Problem 1

Suppose we have made the following measurements:

Frequency of FP operations = 25%
Average CPI of FP Operations = 4.0
Average CPI of other Instructions = 1.33
Frequency of FPSQR = 2%
CPI of FPSQR = 20

The designers have figured out two design alternatives. One will decrease the CPI of FPSQR to 2 and the other will decrease the average CPI of FP operations to 2.5. Which one will yield the greatest overall improvement in performance? You may assume that FPSQR is not counted as part of FP Operations. Hint: Consider the processor performance equation.

Using the Iron Law processor performance equation for this problem, you actually don't need to compute everything since many things stay constant. What does matter is which choices yields the greatest improvement, which in this case will be determined by largest decrease in CPI.

FPSQR
(frequency of FPSQR)(CPI of FPSQR) = (0.02)(20) = .02*20 = .4
(frequency of FPSQR)(CPI of improved FPSQR) = (0.02)(2) = .04
weighted CPI improvement = 0.4 - 0.04 = 0.36

FP Operations
(frequency of FP operations)(avg. CPI of FP operations) = (0.25)(4.0) = 1.0
(frequency of FP operations)(improved CPI of FP operations) = (0.25)(2.5) = .625
weighted CPI improvement = 1.0 - .675 = 0.375

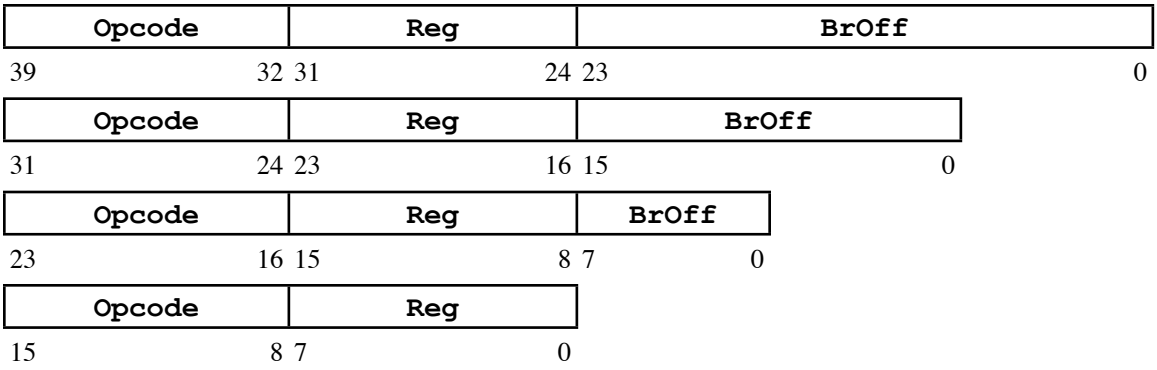
Even after speeding up FPSQR by a factor of 10, improving all FP operations will improve processor performance more because FPSQR are so rare in the code.

Problem 2

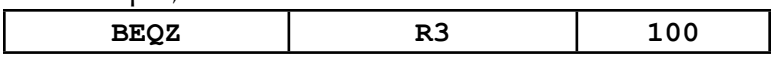
Ben Bitdiddle is designing a handheld device. However, because the device's storage capacity and battery life are limited, he needs to reduce the size of his code. Therefore, he decides to design variable-length instruction set formats for the handheld device to produce more compact code.

Part A

Ben is trying to decide whether it is worthwhile to have multiple offset lengths for branches. He considers the following formats for a branch instruction.



For example,



would mean if **R3** is zero, branch to location **100 + PC**.

He also has the following statistics reflecting the cumulative percentage of branch instructions that can be accommodated with the corresponding number of bits needed to encode the offset.

# Offset Magnitude Bits	Cumulative Branches	# Offset Magnitude Bits	Cumulative Branches
0	0.10%	11	96%
1	2.80%	12	96.80%
2	10.50%	13	97.40%
3	22.90%	14	98.10%
4	36.50%	15	98.50%
5	57.40%	16	99.50%
6	72.40%	17	99.50%
7	85.20%	18	99.80%
8	90.50%	19	100%
9	93.10%	20	100%
10	95.10%	21	100%

On average, how many bits is a branch instruction reduced by using this variable-length offset encoding, compared with the fixed 24-bit offset? Suppose branch instructions account for 10% of the static code. How much do the variable-sized branch offset encodings reduce the total code?

Remember you want to use enough bits to encode the offset, but no more than needed. For example, to find out how many branches are best served by the 16 bit offset, take the

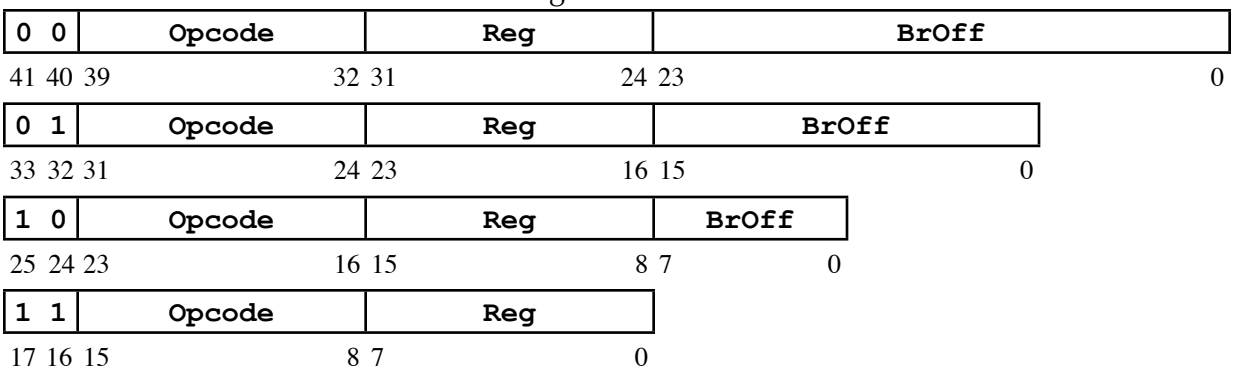
cumulative total for 16, but subtract how many could be satisfied with 8 bits (99.5% - 90.5% = 9%).

= (% of 0 offset)(24 bits saved) + (% of 8 offset)(16 bits saved) + (% of 16 offset)(8 bits saved) + (% of 24 bit offset)(0 bits saved)
= (0.001)(24) + (.905 - .001)(16) + (.995 - .905)(8) + (1-.995)(0)
= .024 + 14.464 + .72 + 0 = 15.208 bits saved on branches

Branches are 40 bits long, so this saves (15.208/40 = 38.02%) on branches. Since they are 10% of the code, this saves (10%*38.02% = 3.802%) in total code length.

Part B

In order to implement the above variable-length instruction encoding, Ben needs to specify each instruction's length. Since there are 4 possible instruction lengths, he decides to add 2 more bits as the instruction length field.



What is the overhead cost of adding these 2 bits? Describe two alternative encoding methods to specify the instruction length.

Average length of a branch instruction = (40 - 15.208) = 24.792 bits
Add 2 bits per branch is 2/24.792 = 8.067% increase in length of branch instruction

The cleanest alternate encoding would use 4 opcodes instead of 1 for branches, assuming the ISA hasn't used them all up. The opcode would determine its length, like (BR40, BR32, BR24…). Another way would be to use offsets that are 2 bits shorter, like 6 and 14. The one catch to this approach is the 0 offset can't be any shorter, so the instruction might still take 18 bits but this will not be noticeable since it is so rarely used. The 24 bit offset would seem to be implementable with only a 22 bit offset (making a 40 bit instruction), since for the data given it would be sufficient, but if the ISA requires it to allow up to 24 bits, that instruction could be 42 bits, and it also wouldn't matter because of how rare it is.

Part C

Ben decides to see if adding extra alignment/decoding hardware to support a smaller granularity of the branch offset field would be worth it. He considers branch offset sizes of

0, 4, 8, 12, 16, 20, 24. By how much does this encoding reduce the total static code (still assuming that branches account for 10% of the code)?

Using the same methodology as 2.A (abbreviated for sanity)
= (0 bit) + (4 bit) + (8 bit) + (12 bit) + (16 bit) + (20 bit) + (24 bit)
= .024 + 7.28 + 8.64 + .756 + .216 + .02 + 0 = 16.936 bits saved

This saves 42.35% on branches, and thus 4.235% on total code length. You can see that there are diminishing returns for using variable length encodings, and if you count the encoding costs from Part C it might not always be a clear win to use variable length.

Part D
What are some disadvantages of using variable-length instructions?

In summary: complexity. If the opcodes and operands can come from different parts of the instruction it requires more flexibility (and thus more hardware) to support decoding. The instructions must also be decoded serially, because at least some of the instruction must be read before it is known how long the instruction is. This also means it is hard to know where instruction boundaries are looking at arbitrary code. This instruction boundary ambiguity can make it hard to implement superscalar machines. Worse yet, with a variable length instruction, it could potentially straddle a page (or cache block) boundary, meaning it could cause 2 page (or cache) misses.