MIPSim

MIPS Processor Simulator Project

Will Drach & Braun Lake

Installation & Running

The code is bundled with a Makefile. You can simply run “make” to build the code. Once the code is built, it is run with the following arguments

```
./run_sim <program_location> <i size> <d size> <block size> [--write_through/-wt]
```

For example, if I wanted to run the file “examples/Program2File.txt” with an i cache size of 256 bytes, a d cache size of 128 bytes, a block size of 16, and write-through enabled, you would do the following

```
./run_sim examples/Program2File.txt 256 128 16 -wt
```

Simply omit the “-wt” flag to run in write-back mode.

There are 4 examples in the “examples” folder, the 2 given program files, and the 2 simple load/store examples. You can modify these or run any file on your filesystem if you pass the correct location.

Self Assessment

Pipeline works with instruction and data caches with write through and write back. The caches are simulated and no data is actually stored in a cache (as we did not see any reason to make another array that just contains the same data as our “memory” array). There are lots of sources for little differences in speed (branch data hazards, write-back discrepancies, etc.), but in general it should be close (within about 1500 clock cycles).

Programs

Program 1

Results

I size	D size	block size	WB/WT	I hit rate	d hit rate	CPI	total cycles
0	0					1.28	606987
128	256	1	WT	99.66	74.9	2.14	1013508
128	256	1	WB	99.66	74.9	1.93	915606
128	256	4	WT	99.78	93.45	2.02	955555
128	256	4	WB	99.78	93.45	1.59	754335
128	256	16	WT	99.79	97.81	3.22	1527161
128	256	16	WB	99.79	97.81	1.59	753504
64	1024	1	WT	97.61	98.98	1.72	814286
64	1024	1	WB	97.61	98.98	1.5	712565
64	1024	4	WT	99.16	99.74	1.78	845248
64	1024	4	WB	99.16	99.74	1.38	656148
64	1024	16	WT	93.57	99.54	4.49	2128232
64	1024	16	WB	93.57	99.54	3.04	1442567

Analysis

Overall, we see some interesting results here. Write back is generally better than write through, and larger caches are generally better than smaller caches (not a big surprise). You will also notice that having a big block size is generally not a good idea for data, especially when we see the 16 line blocks reducing hit rates. This is because bigger block size means less blocks in the cache, which depending on the program means more collisions. It seems as though, from our

tested data, a block size of 4, i cache of 64, and d cache of 1024 did the best. However, if we were to mix and match based on hit rate, a d cache of 1024 with a block size 4, and an i cache of 128 with a block size of 16 would be the best bet for maximizing efficiency.

Program 2

Results

I size	D size	block size	WB/WT	I hit rate	d hit rate	CPI	total cycles
0	0					1.19	14446
64	512	1	WT	64.14	81.93	4.62	59096
64	512	1	WB	64.14	81.93	4.47	54354
64	512	4	WT	80.86	94.75	3.69	44834
64	512	4	WB	80.86	94.75	3.33	40439
64	512	16	WT	85.9	94.51	5.75	69936
64	512	16	WB	85.9	94.51	4.91	59628
128	256	1	WT	98.86	73.79	1.66	20138
128	256	1	WB	98.86	73.79	1.62	19641
128	256	4	WT	99.67	84.35	1.57	19063
128	256	4	WB	99.67	84.35	1.55	18865
128	256	16	WT	90.57	57.56	6.22	75557
128	256	16	WB	90.57	57.56	6.6	80173
256	128	1	WT	98.86	73.79	1.66	20138
256	128	1	WB	98.86	73.79	1.62	19647
256	128	4	WT	99.68	84.24	1.57	19084
256	128	4	WB	99.68	84.24	1.56	18901
256	128	16	WT	99.88	57.33	3.56	43305
256	128	16	WB	99.88	57.33	3.76	45666

Analysis

This cache simulation shows some different results than the last one, particularly when looking at the danger of having a cache that is too small. The small i cache simply destroyed the CPI, so it is important in this case to have a larger i cache. We also run into the issue of large block sizes again, particularly in the test of a 256 byte d cache with the block size of 16, which saw a drop of 27% compared to a block size of 4. In this case, the best test ran was with the I/D caches having 128 and 256 bytes respectively. This was also very similar with those sizes flipped, which was interesting. Overall, it seems like the bigger the cache the better. If we could mix and match tests, a 512 byte d cache with a block size of 4 and a 256 byte i cache with a block size of 16. It seems as though the large i cache with a large block size is actually quite good, which means there's probably large sections before branches that are repeated over and over.

Combined Results

Overall, we saw pretty much what was expected. We can make 3 pretty significant conclusions from this data, which is a 16 line block for instructions and a 4 line block for data is best, and the bigger the cache, the better. Write through and write back are seemingly dependent on what data you are working with. In Program 1, write back gave significant improvements to the CPI. In Program 2, we didn't see this improvement, with some trials having very similar results. Due to the increased complexity involved in write back, it would be better to increase cache size and not do a write back cache in this case. If you had some combination, you would have to determine the trade offs and the space needs of a write back cache. Altogether, nothing about these result is surprising. For the instructions, you are pretty likely to have 16 instructions back to back without a branch, meaning larger block size benefits you. For data, it's much more likely to have 3 or 4 variables within a function that you play around with, and anything more than that ends up wasting time and valuable cache space. In both cases, big caches means less misses which means a lower CPI.

Lessons Learned

To say that we learned a lot during this project would be an extreme understatement. There is simply so much that you think you know about how a processor works, but building it out just solidifies all of those little issues that arise during the design process. For example, we move branches up from branching in the EX stage to branching in the ID stage. There's a good reason for this, as this combined with MIPS' delayed branches means that you don't have to insert any noops after a branch. However, this causes additional data hazards because there is not functionality to forward to the ID stage. In order to adjust for this, we had to insert stalls based on data hazards and forwarding conditions, which greatly increased the CPI and overall cycles needed. This is just one example of how making a small improvement has a huge effect

on other stages of the pipeline, and there were many more. Some other things include whether to load things into cache on a write-through, how much data to write on a write back, forwarding rt from EX but rs from MEM, the fact that rd is not always the destination register, etc. There's simply so much more in here that is not quite covered by having a basic knowledge of a processor, but is equally important in the end result.