Dartmouth College

COSC 051: Computer Architecture

Homework 7 Report

Professor: Sean Smith

Student: Amittai Wekesa

Part A: Caching (50 points) I decided to implement my solution in C++ because it's relatively faster than Python / Java and I do not have to worry about deep-level memory management. I implemented and compiled my program using the `C++11` standard and tested with GNU's compiler, `g++`. Either way, it shouldn't be an issue to compile on `clang`.

I implemented a program, in caching.cpp that simulates a cache structure and, given a trace file and caching mode, simulates caching with that mode.

I wrote a `Makefile` and a test program testing.sh that runs the different tests provided in the Homework specification.

To test, run `make test`.

To simply build the brogram (maybe to test other trace files), simply run `make`.

Check out testing.out for sample output.

"Usage: ./caching [trace file] [associativity]"

NOTE: The associativity modes are:

```
0 -> direct mapping

1 -> 2-way associative

2 -> 4-way associative

3 -> Fully associative
```

Problem 1 (10 Points)

My caching program was able to match the professor's output on sample.txt:

```
Direct Mapping
 1: addr 0x22222210; tag 222010; looking in set 33, miss! line 0 empty, adding there.
 2: addr 0x33333310; tag 333010; looking in set 49, miss! line 0 empty, adding there.
 3: addr 0x44444410; tag 444010; looking in set 1, miss! line 0 empty, adding there.
 4: addr 0x55555510; tag 555010; looking in set 17, miss! line 0 empty, adding there.
 5: addr 0x66666610; tag 666010; looking in set 33, miss! line 0 evicted, adding there.
 6: addr 0x22222210; tag 222010; looking in set 33, found it in line 1; hit!
 7: addr 0x33333310; tag 333010; looking in set 49, found it in line 1; hit!
 8: addr 0x8fe01030; tag e01030; looking in set 3, miss! line 0 empty, adding there.
 9: addr 0x8fe01031; tag e01031; looking in set 3, miss! line 0 evicted, adding there.
10: addr 0xbffff8fc; tag fff074; looking in set 15, miss! line 0 empty, adding there.
2 hits, 8 misses, 10 addresses.

Two-Way Associative Mapping
 1: addr 0x22222210; tag 222010; looking in set 1, miss! line 0 empty, adding there.
 2: addr 0x33333310; tag 333010; looking in set 17, miss! line 0 empty, adding there.
 3: addr 0x44444410; tag 444010; looking in set 1, miss! line 0 evicted, adding there.
 4: addr 0x55555510; tag 555010; looking in set 17, miss! line 0 evicted, adding there.
 5: addr 0x66666610; tag 666010; looking in set 1, miss! line 0 evicted, adding there.
 6: addr 0x22222210; tag 222010; looking in set 1, found it in line 2; hit!
 7: addr 0x33333310; tag 333010; looking in set 17, found it in line 2; hit!
 8: addr 0x8fe01030; tag e01030; looking in set 3, miss! line 0 empty, adding there.
 9: addr 0x8fe01031; tag e01031; looking in set 3, miss! line 1 evicted, adding there.
10: addr 0xbffff8fc; tag fff074; looking in set 15, miss! line 0 empty, adding there.
2 hits, 8 misses, 10 addresses.

Four-Way Associative Mapping
 1: addr 0x22222210; tag 222010; looking in set 1, miss! line 0 empty, adding there.
 2: addr 0x33333310; tag 333010; looking in set 1, miss! line 0 evicted, adding there.
 3: addr 0x44444410; tag 444010; looking in set 1, miss! line 0 evicted, adding there.
 4: addr 0x55555510; tag 555010; looking in set 1, miss! line 0 evicted, adding there.
 5: addr 0x66666610; tag 666010; looking in set 1, miss! line 0 evicted, adding there.
 6: addr 0x22222210; tag 222010; looking in set 1, found it in line 4; hit!
 7: addr 0x33333310; tag 333010; looking in set 1, miss! line 0 evicted, adding there.
 8: addr 0x8fe01030; tag e01030; looking in set 3, miss! line 0 empty, adding there.
 9: addr 0x8fe01031; tag e01031; looking in set 3, miss! line 1 evicted, adding there.
10: addr 0xbffff8fc; tag fff074; looking in set 15, miss! line 0 empty, adding there.
1 hits, 9 misses, 10 addresses.

Fully Associative Mapping
 1: addr 0x22222210; tag 222010; looking in set 0, miss! line 16 empty, adding there.
 2: addr 0x33333310; tag 333010; looking in set 0, miss! line 16 evicted, adding there.
 3: addr 0x44444410; tag 444010; looking in set 0, miss! line 16 evicted, adding there.
 4: addr 0x55555510; tag 555010; looking in set 0, miss! line 16 evicted, adding there.
 5: addr 0x66666610; tag 666010; looking in set 0, miss! line 16 evicted, adding there.
 6: addr 0x22222210; tag 222010; looking in set 0, found it in line 64; hit!
 7: addr 0x33333310; tag 333010; looking in set 0, miss! line 16 evicted, adding there.
 8: addr 0x8fe01030; tag e01030; looking in set 0, miss! line 48 evicted, adding there.
 9: addr 0x8fe01031; tag e01031; looking in set 0, miss! line 49 evicted, adding there.
10: addr 0xbffff8fc; tag fff074; looking in set 0, miss! line 52 evicted, adding there.
1 hits, 9 misses, 10 addresses.
```

Problem 2 (40 Points)

Is the conventional wisdom right for caching?

Simulating on **long-trace**, I interestingly got the best performace on **directly mapped** caching and worse performance with increasing associativity.

Fully associative mapping turned out zero matches on long-trace. I think this is understandable because – with only a single working set, the only possible matches are when an address recurs within 64 (the number of lines in the set) fetches after it's initial occurrence. On the other hand, opportunistic hits are more likely even more than 64 fetches after an address occurred because some other addresses will be mapped to other sets in the cache.

```
Direct Mapping
6026 hits, 6340052 misses, 6346078 addresses.

Two-Way Associative Mapping
5687 hits, 6340391 misses, 6346078 addresses.

Four-Way Associative Mapping
3629 hits, 6342449 misses, 6346078 addresses.

Fully Associative Mapping
0 hits, 6346078 misses, 6346078 addresses.
```

Part B: OPLI (50 points) Extend your Y86 to implement **OPLI** operations; *isubl*, *iaddl*, *iandl*, and *ixorl*.

Revised FSM FLOW-CHART

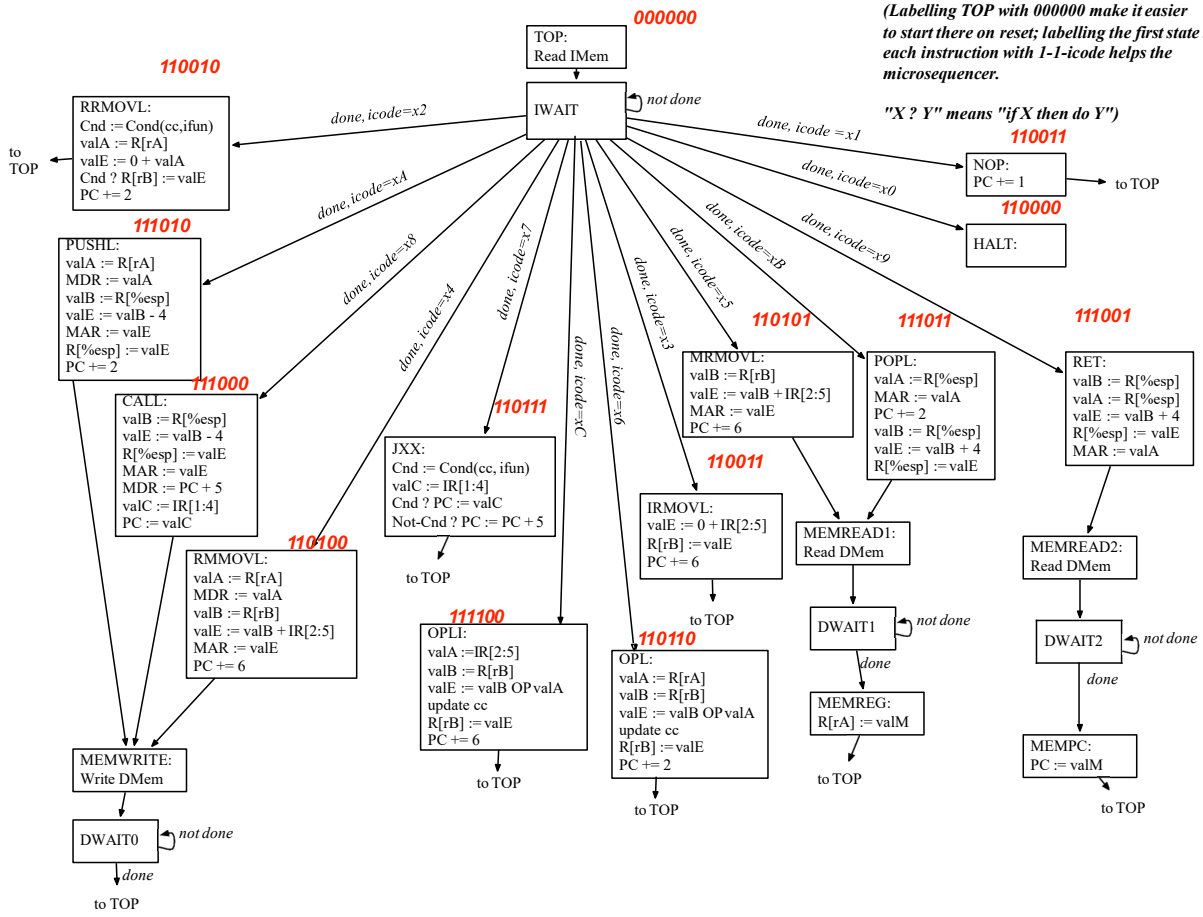

Figure 1: FSM Flowchart

Here is a link to my revised FSM Flowchart: flowchart

state				rom0: microinstruction																																rom1: input to microsequencer									
decimal	hex	binary	name	lmemreq	pcincsel		valcsel	srcasel	srcbsel	dstesel	dstereq		dstmreq	aluasel		alubsel	aluopsel	ldcc	marsel	ldmar	mdrsel	ldmdr	dmemreq	dmemwrite	newpcsel		ldpc	for future use								sel		ValN						for futu	
0	0x00	000000	TOP	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	1	0	0
1	0x01	000001	IWAIT	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	1	0	0
2	0x02	000010	MEMWRITE	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	1	1	0	0
3	0x03	000011	DWAIT0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
4	0x04	000100	DWAIT1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1	0	0	0	0	0
5	0x05	000101	DWAIT2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1	0	0	1	0	0
6	0x06	000110	MEMREAD1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1	0	0	0	0
7	0x07	000111	MEMREAD2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1	0	1	0	0
8	0x08	001000	MEMREG	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
9	0x09	001001	MEMPC	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
48	0x30	110000	HALT	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
49	0x31	110001	NOP	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
50	0x32	110010	RRMOVL	0	0	1	0	0	0	0	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
51	0x33	110011	IRMOVL	0	1	1	1	0	0	0	0	1	0	0	1	1	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
52	0x34	110100	RMMOVL	0	1	1	1	0	0	0	0	0	0	0	1	0	0	0	0	1	0	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	1	0	0	0
53	0x35	110101	MRMOVL	0	1	1	1	0	0	0	0	0	0	0	1	0	0	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1	0	0	0	1	1	0	0	0
54	0x36	110110	OPL	0	0	1	1	0	0	0	0	1	0	0	0	0	1	1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
55	0x37	110111	jXX	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
56	0x38	111000	CALL	0	1	0	0	0	1	1	0	1	0	1	1	0	0	0	0	1	1	1	0	0	0	1	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	1	0	0	0
57	0x39	111001	RET	0	0	0	0	1	1	1	0	1	0	1	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1	1	1	0	0
58	0x3A	111010	PUSHL	0	0	1	0	0	1	1	0	1	0	1	1	0	0	0	0	1	0	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	1	0	0	0
59	0x3B	111011	POPL	0	0	1	0	1	1	1	0	1	0	1	0	0	0	0	1	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1	0	0	0	1	1	0	0	0
54	0x36	111100	OPLI	0	1	1	1	0	0	0	0	1	0	0	1	0	1	1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
61	0x3D	111101		0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
62	0x3E	111110		0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
63	0x3F	111111		0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 2: FSM matrix

To test my operations, I wrote two programs: decr.ys and incr.ys that manipulate ASCII by repeatedly `isubl`~ing and `iaddl`~ing with $0x01$ and printing the values to DDR.

The result is the ASCII representations being printed to the TTY.