ECE411 MP3 CP4 Report

Group: ZerotoOne

Progress Report:
By checkpoint 4, we successfully added a memory mapped performance unit that user can access. We record 10 parameters, including:

1. the number of misses in the branch prediction
2. Total number of branches
3. miss rate in the L1-instruction cache
4. Total number of L1instruction access
5. miss rate in the L1-data cache
6. Total number of L1 data cache
7. miss rate in the L2 cache
8. Total number of L2 cache
9. number of instruction-data access conflict in the arbiter
10. total number of stall cycles.

We also implemented a single unit eviction write buffer. The buffer immediately responds to a write request if the buffer is unoccupied, holds the data for several cycles and then finds the next available cycle that is not serving read operation to write the evicted data back to the lower level of memory. One eviction write buffer is placed between the L1Dcache and arbiter, the other eviction write buffer is placed between the L2cache and the physical memory. During the test we find a few write back operations successfully performed between the L1Dcache and the arbiter but did not find any write back requests between the L2cache and the physical memory.

The static branch predictor was already implemented before this checkpoint. It always assume the branch will not be taken, and there will be a 3 cycle punishment if the result calculated in the ex stage shows that it should take the branch.

During this checkpoint, we have done additional corner-tests on branch prediction and data dependency. We found a critical bug when the two parameters in the EX stage depend on both the data in MEM and WE stage, and also the MEM data is fetched from memory. When this scenario happens, our cpu fails to forward both of them. We fixed it by adding an additional forwarding unit in the EX pipe which is not affected by

Contribution:
Haichuan Xu: implementation of eviction write buffer and debugging
Haiyang Zhang: implementation of eviction buffer and debugging
Yuan Ma: implementation of the performance unit and debugging

Roadmap:
By the final checkpoint, we expect to finish the implementation of all the advanced cache features that we want:

1. dynamic branch prediction unit which utilizes branch history tables and branch target buffer. It also features tournament predictors which switches between the local history and the global history to make predictions
2. Pipeline the L2 cache to improve the max frequency.
3. Victim cache
4. Memory prefetch unit in L2 based on memory access pattern

Testing:

1. Data dependency test
   We modified out factorial code in mp0 to further test data dependency. We intentionally add several lines of code that require data dependence. The test code with comments are provided in the appendix. For the corner tests, we tested the scenario where an instruction' parameter(s) depend(s) two previous instructions. See appendix I.

2. Eviction buffer and Cache testing
   To test functionality of eviction buffer, we intentionally replaced dirty data from L1 to L2. We found that the cache performance is not significantly improved. The reason for this is that we have built an highly inclusive cache where the data in L1 are all in L2 so there is fewer chances for evicting delay already. See appendix II.

APPENDIX I

```
factorial.s:
.align 4
.section .text
.globl _start
        # This piece of code is aim to calculate the factorial of an input
        # integer. This program is using 32-unsigned data type for both the input
        # and the output number. This constraints the input number to be strictly under
        # or equal 12.

        # input and output
        # Please modify the number in memory address(input_integer) to calculate the different
        # ingeger's factorial. The input default value is 5.
        # When the program 'halt,' the following memory(output_integer) should contain the
correct
        # output as well as x3. the data in (output_integer) is set to zero while calculating.
_start:
        #load the input number to X1
        lw x1, input_integer
        lui x2, 1
        lui x3, 1
```

```
        srli x2, x2, 12   #WB to EX forwarding
        srli x3, x3, 12   #WB to EX forwarding
        blt x1, x2, halt
        beq x1, x2, load_output   # if the input is one, than it should be done
multiply_new_number:
        andi x4, x2, 0          #MEM to EX forwarding
        addi x4, x4, 1
        add x9, x0, x2          #intentionally add dependency
        addi x10, x0, 1
        add x2, x9, x10         #both parameters based on prev two instrs both MEM and WB
should forward data to EX stage
        addi x5, x3, 0
keep_multiplying:
        addi x4, x4, 1
        blt x2, x4, iteration_check
        andi x6, x6, 0
add_one_more_multiplicant:
        add x6, x0, x6 #intentionally add dependency
        add x6, x0, x6
        addi x6, x6, 1  # same parameter based on pre two instrs, MEM and WB forwarding
conflict

        blt x5, x6, keep_multiplying
        addi x3, x3, 1
        beq x0, x0, add_one_more_multiplicant

iteration_check:
        bge x2, x1, load_output # iteration done
        beq x0, x0, multiply_new_number # start a new iteration that times a new number
load_output:
        la  x6, output_integer # store the output to destination memory
        sw  x3, 0(x6)
halt:           # Infinite loop to keep the processor
        add x2, x0, 0   #intentionally add dependency
        lw x1, 0(x2)
        add x2, x2, x1  #both parameters based on prev two instrs both MEM and WB should
forward data to EX stage
        beq x0, x0, halt  # from trying to execute the data below.

.section .rodata
        # Please modify the following input_integer number to calculate different
        # ingeger's factorial. The input default value is 5.
```

```
	# When the program 'halt' the following memory(output_integer) should contain the correct
	# output. the output is set to zero while calculating.
input_integer:  .word   5
output_integer:    .word          0
```

APPENDIX II

```
load_test:
.align 4
.section .text
.globl _start
_start:
# word-align store_test: Cache hit/miss && LRU check
la x1, R0S3W0
la x2, R0S3W3
la x3, R1S3W0
la x4, R2S3W1
la x8, TESTSIG
lui x7, 0 # Initializing register for testing
lw x8, 0(x8) # cache miss: loading test signal
#TEST CASE ONE: write to cache with miss
sw x8, 0(x1) # cache miss;
lw x5, 0(x1)
bne x8, x5, badend
addi x7, x7, 1
#TEST CASE TWO: write to cache with hit
sw x8, 0(x2) #cache hit for line1;
lw x5, 0(x2)
bne x8, x5, badend
addi x7, x7, 1
#TEST CASE THREE: replace LRU
sw x8, 0(x3) # cache miss;
```

```
lw x5, 0(x3)
bne x8, x5, badend
sw x8, 0(x4) # cache miss and repalce line1;
lw x5, 0(x4)
bne x8, x5, badend
sw x8, 4(x3) # cache hit because line2 is not replaced; expected value=0x600D1301
lw x5, 4(x3)
bne x8, x5, badend
lw x5, 0(x2) # reload replaced line1 and check if the replaced value is changed in pmem
bne x8, x5, badend
addi x7, x7, 1
#TEST CASE FOUT: half-word-align load word
sw x8, 2(x1)
lw x5, 2(x1)
bne x8, x5, badend
addi x7, x7, 1
#TEST CASE FIVE: half-word-align load half-wordla x2, R0S6W0
lw x5, 2(x2) # cache miss for a new set;
srli x9, x5, 16
slli x9, x9, 16
slli x10, x8, 16
srli x10, x10, 16
add x9, x9, x10 # load expected value to x9
sh x8, 2(x2) # write to the half-word
lw x5, 2(x2)
bne x5, x9, badend
addi x7, x7, 1
#TEST CASE SIX: half-word-algin load byte
la x2, R0S6W2
lw x5, 2(x2) # cache hit
srli x9, x5, 8
slli x9, x9, 8
slli x10, x8, 24
srli x10, x10, 24
add x9, x9, x10 # load expected value to x9
sb x8, 2(x2) # write to the byte
lw x5, 2(x2)
bne x5, x9, badend
addi x7, x7, 1
#TEST CASE SEVEN: byte-align load word
la x1, R0S3W5
sw x8, 1(x1)
lw x5, 1(x1) # cahce hit
```

```
bne x5, x8, badend
addi x7, x7, 1
#TEST CASE EIGHT: byte-align load half-word
la x1, R0S3W6
lw x5, 1(x2) # cache hit;
srli x9, x5, 16
slli x9, x9, 16
slli x10, x8, 16
srli x10, x10, 16
add x9, x9, x10 # load expected value to x9
sh x8, 1(x2)
lw x5, 1(x2)
bne x5, x9, badend
addi x7, x7, 1
#TEST CASE NINE: byte-align load byte
la x1, R0S6W7
slli x9, x8, 24
srli x9, x9, 24
sb x8, 3(x1)
lbu x5, 3(x1) # cache hit; expected value=0xFFFFFFF0
bne x5, x9, badend
addi x7, x7, 1
goodend:
jal x0, goodend
badend:
jal x0, badend
.section .rodata
TESTSIG: .word 0x600DFFFF
.balign 256 #align to the start of a new set round
.zero 96 #pad to Set 3
R0S3W0: .word 0x600D0300
R0S3W1: .word 0x600D0301
R0S3W2: .word 0x600D0302
R0S3W3: .word 0x600D0303
R0S3W4: .word 0x600D0304
R0S3W5: .word 0x600D0305
R0S3W6: .word 0x600D0306
R0S3W7: .word 0x600D0307
.zero 64 #pad to Set 6
R0S6W0: .word 0x600DF0F0
R0S6W1: .word 0x600DF0F0
R0S6W2: .word 0x600DF0F0
R0S6W3: .word 0x600DF0F0
```

```
R0S6W4: .word 0xF0F0600D
R0S6W5: .word 0xF0F0600D
R0S6W6: .word 0xF0F0600D
R0S6W7: .word 0xF0F0600D
.balign 256 #align to the start of a new set round
.zero 96 #pad to Set 3
R1S3W0: .word 0x600D1300
R1S3W1: .word 0x600D1301
R1S3W2: .word 0x600D1302
R1S3W3: .word 0x600D1303
R1S3W4: .word 0x600D1304
R1S3W5: .word 0x600D1305
R1S3W6: .word 0x600D1306
R1S3W7: .word 0x600D1307
.zero 64 #pad to Set 6
R1S6W0: .word 0xF0F0F0F0
R1S6W1: .word 0xF0F0F0F0
R1S6W2: .word 0xF0F0F0F0
R1S6W3: .word 0xF0F0F0F0
R1S6W4: .word 0xF0F0F0F0
R1S6W5: .word 0xF0F0F0F0
R1S6W6: .word 0xF0F0F0F0
R1S6W7: .word 0xF0F0F0F0
.balign 256 #align to the start of a new set round
.zero 96 #pad to Set 3
R2S3W0: .word 0x600D2300
R2S3W1: .word 0x600D2301
R2S3W2: .word 0x600D2302
R2S3W3: .word 0x600D2303
R2S3W4: .word 0x600D2304
R2S3W5: .word 0x600D2305
R2S3W6: .word 0x600D2306
R2S3W7: .word 0x600D2307
.zero 64 #pad to Set 6
R2S6W0: .word 0xF0F0F0F0
R2S6W1: .word 0xF0F0F0F0
R2S6W2: .word 0xF0F0F0F0
R2S6W3: .word 0xF0F0F0F0
R2S6W4: .word 0xF0F0F0F0
R2S6W5: .word 0xF0F0F0F0
R2S6W6: .word 0xF0F0F0F0
R2S6W7: .word 0xF0F0F0F0
```