ECE 4750 PSET 2

Tim Yao (ty252)

Oct 10, 2015

1 PARCv1 Instruction Cache

1.a Categorizing Cache Misses

Addr	Instruction	Iteration 1	Iteration 2
	loop:		
0x108	addiu r1, r1, -1	compulsory	
0x10c	addiu r2, r2, 1		
0x110	j foo	compulsory	conflict
	...		
	foo:		
0x218	addiu r6, r6, 1	compulsory	conflict
0x21c	bne r1, r0, loop		

Figure 1: Cache Miss Type

1.b Average Memory Access Latency

Looking at iteration 2, we can see that there are 2 misses out of the 5 instructions. Therefore the miss rate for 64 iterations of the loop is 0.4.
The average memory access latency is:
AMAL = (Hit Time) + (Miss Rate $\times$ Miss Penalty)
AMAL = $1 + (0.4 \times 5)$
AMAL = 3 cycles
The AMAL is dominated by conflict misses, as shown by the miss chart above. Compulsory misses only occur on the first iteration of the loop.

1.c Set-Associativity

The cache performance will increase significantly, because there will no longer be conflict misses during the loop. With this new cache microarchitecture, only compulsory misses will be left.

2 Page-Based Memory Translation

2.a Two-Level Page Tables

The 16-bit virtual address is used as the following:

Bits 14-15	Bits 12-13	Bits 0-11
XX	XX	XXXXXXXXXXXX
Virtual Page Number		**Page Offset**
L1 Index	**L2 Index**	**Page Offset**

Figure 2: Virtual Address Usage

Page Tables:

	Page-Table Entry	
Paddr of PTE	**Valid**	**Paddr**
0xffffc	1	0xfffe0
0xffff8		
0xffff4		
0xffff0	1	0xfffb0
0xfffec	1	0x05000
0xfffe8	1	0x07000
0xfffe4		
0xfffe0		
0xfffdc		
0xfffd8		
0xfffd4		
0xfffd0		
0xfffcc		
0xfffc8		
0xfffc4		
0xfffc0		
0xfffbc	1	0x01000
0xfffb8	1	0x04000
0xfffb4	1	0x00000
0xfffb0		

Figure 3: Contents of Physical Memory with Page Tables

2.b Translation-Lookaside Buffer

Transaction Address	VPN	Page Offset	m/h	Total Num Mem Accesses	TLB Way 0		TLB Way 1	
					VPN	PPN	VPN	PPN
0xeff4	0xe	0xff4	m	3	-	-	-	-
0x2ff0	0x2	0xff0	m	3	0xe	0x07	-	-
0xeff8	0xe	0xff8	h	1	0xe	0x07	0x2	0x04
0x2ff4	0x2	0xff4	h	1	0xe	0x07	0x2	0x04
0xeffc	0xe	0xffc	h	1	0xe	0x07	0x2	0x04
0x2ff8	0x2	0xff8	h	1	0xe	0x07	0x2	0x04
0xf000	0xf	0x000	m	3	0xe	0x07	0x2	0x04
0x2ffc	0x2	0xffc	h	1	0xf	0x05	0x2	0x04
0xf004	0xf	0x004	h	1	0xf	0x05	0x2	0x04
0x3000	0x3	0x000	m	3	0xf	0x05	0x2	0x04
0xf008	0xf	0x008	h	1	0xf	0x05	0x3	0x01
0x3004	0x3	0x004	h	1	0xf	0x05	0x2	0x04
0xf00c	0xf	0x00c	h	1	0xf	0x05	0x2	0x04
0x3008	0x3	0x008	h	1	0xf	0x05	0x2	0x04

Figure 4: TLB Contents Over Time

3 Impact of Cache Access Time and Replacement Policy

3.a Miss Rate Analysis

Transaction Address	**tag**	**idx**	**m/h**	**L0**	**L1**	**L2**	**L3**	**L4**	**L5**	**L6**	**L7**
0x024	0x0	0x2	m	-	-	-	-	-	-	-	-
0x030	0x0	0x3	m			0x0					
0x07c	0x0	0x7	m				0x0				
0x070	0x0	0x7	h								0x0
0x100	0x2	0x0	m								
0x110	0x2	0x1	m	0x2							
0x204	0x4	0x0	m		0x2						
0x214	0x4	0x1	m	0x4							
0x308	0x6	0x0	m		0x4						
0x110	0x2	0x1	m	0x6							
0x114	0x2	0x1	h		0x2						
0x118	0x2	0x1	h								
0x11c	0x2	0x1	h								
0x410	0x8	0x1	m								
0x110	0x2	0x1	m		0x8						
0x510	0xa	0x1	m		0x2						
0x110	0x2	0x1	m		0xa						
0x610	0xc	0x1	m		0x2						
0x110	0x2	0x1	m		0xc						
0x710	0xe	0x1	m		0x2						
Number of Misses = 16											
Miss Rate = 0.8											

Figure 5: Direct-Mapped Cache Contents Over Time

Transaction				Set 0		Set 1		Set 2		Set 3	
Address	**tag**	**idx**	**m/h**	**Way 0**	**Way 1**	**Way 0**	**Way 1**	**Way 0**	**Way 1**	**Way 0**	**Way 1**
0x024	0x0	0x2	m	-	-	-	-	-	-	-	-
0x030	0x0	0x3	m					0x0			
0x07c	0x1	0x3	m							0x0	
0x070	0x1	0x3	h								0x1
0x100	0x4	0x0	m								
0x110	0x4	0x1	m	0x4							
0x204	0x8	0x0	m			0x4					
0x214	0x8	0x1	m		0x8						
0x308	0xc	0x0	m				0x8				
0x110	0x4	0x1	h	0xc							
0x114	0x4	0x1	h								
0x118	0x4	0x1	h								
0x11c	0x4	0x1	h								
0x410	0x10	0x1	m								
0x110	0x4	0x1	h				0x10				
0x510	0x14	0x1	m								
0x110	0x4	0x1	h				0x14				
0x610	0x18	0x1	m				0x18				
0x110	0x4	0x1	h								
0x710	0x1c	0x1	m								
Number of Misses = 12											
Miss Rate = 0.6											

Figure 6: Two-Way Set-Associative Cache Contents Over Time with LRU Replacement

Transaction				Set 0		Set 1		Set 2		Set 3	
Address	**tag**	**idx**	**m/h**	**Way 0**	**Way 1**	**Way 0**	**Way 1**	**Way 0**	**Way 1**	**Way 0**	**Way 1**
0x024	0x0	0x2	m	-	-	-	-	-	-	-	-
0x030	0x0	0x3	m					0x0			
0x07c	0x1	0x3	m							0x0	
0x070	0x1	0x3	h								0x1
0x100	0x4	0x0	m								
0x110	0x4	0x1	m	0x4							
0x204	0x8	0x0	m			0x4					
0x214	0x8	0x1	m		0x8						
0x308	0xc	0x0	m				0x8				
0x110	0x4	0x1	h	0xc							
0x114	0x4	0x1	h			0x4					
0x118	0x4	0x1	h								
0x11c	0x4	0x1	h								
0x410	0x10	0x1	m								
0x110	0x4	0x1	h				0x10				
0x510	0x14	0x1	m								
0x110	0x4	0x1	m			0x14					
0x610	0x18	0x1	m								
0x110	0x4	0x1	m				0x18				
0x710	0x1c	0x1	m								
Number of Misses = 14											
Miss Rate = 0.7											

Figure 7: Two-Way Set-Associative Cache Contents Over Time with FIFO Replacement

3.b Sequential Tag Check then Memory Access

Component	Delay Equation	Delay(τ)
addr_reg_M0	1	1
tag_decoder	3+2×2	7
tag_mem	10 + [(4+27)/16]	12
tag_cmp	3+2[log2(26)]	13
tag_and	2-1	1
data_decoder	3+2×3	9
data_mem	10 + [(8+128)/16]	19
rdata_mux	3[log2(4)] + [32/8]	10
rdata_reg_M1	1	1
Total		**73**
addr_reg_M0	1	1
tag_decoder	3+2×2	7
tag_mem	10 + [(4+27)/16]	12
tag_cmp	3+2[log2(26)]	13
tag_and	2-1	1
data_decoder	3+2×3	9
data_mem	10 + [(8+128)/16]	19
Total		**62**

Figure 8: Critical Path and Cycle Time for 2-Way Set-Associative Cache with Serialized Tag Check before Data Access

The reason that the 2-way set-associative microarchitecture is slower than the direct-mapped microarchitecture is the need for the tag check result to go through the data_decoder. It happens that the data_decoder's delay is relatively significant (9τ).

3.c Parallel Read Hit Path

Component	Delay Equation	Delay(τ)
addr_reg_M0	1	1
addr_mux	3[log2(2)] + [5/8]	4
data_decoder	3+2×3	9
data_mem	10 + [(8+128)/16]	19
rdata_mux	3[log2(4)] + [32/8]	10
rdata_reg_M1	1	1
Total		**44**

Figure 9: Critical Path and Cycle Time for Direct Mapped Cache with Parallel Read Hit

Component	Delay Equation	Delay(τ)
addr_reg_M0	1	1
addr_mux	3[log2(2)] + [4/8]	4
data_decoder	3+2×2	7
data_mem	10 + [(4+128)/16]	19
rdata_mux	3[log2(4)] + [32/8]	10
way_mux	3[log2(2)] + [32/8]	7
rdata_reg_M1	1	1
Total		**49**

Figure 10: Critical Path and Cycle Time for 2-Way Set-Associative Cache with Parallel Read Hit

INSERT SOME SHIT HERE

3.d Pipelined Write Hit Path

Component	**Delay Equation**	**Delay(τ)**
addr_reg_M0	1	1
tag_decoder	3+2×3	9
tag_mem	10 + [(8+26)/16]	13
tag_cmp	3+2[log2(25)]	13
tag_and	2-1	1
wen_and	2-1	1
wen_reg_M1	1	1
Total		**39**

Figure 11: Critical Path and Cycle Time for Direct Mapped Cache with Pipelined Write Hit

Component	**Delay Equation**	**Delay(τ)**
addr_reg_M0	1	1
tag_decoder	3+2×2	7
tag_mem	10 + [(4+27)/16]	12
tag_cmp	3+2[log2(26)]	13
tag_and	2-1	1
wen_and	2-1	1
wen_reg_M1	1	1
Total		**36**

Figure 12: Critical Path and Cycle Time for 2-Way Set-Associative Cache with Pipelined Write Hit

INSERT SOME SHIT HERE

3.e Average Memory Access Latency

4 Two-Cycle Pipelined Integer ALU and Multiplier

4.a Part 4.A Control and Data Hazard Latencies

Jump Resolution Latency = 2
Branch Resolution Latency = 4
ALU-use Delay Latency = 2
Load-use Delay Latency = 3

4.b Resolving Data Hazards with Software Scheduling

```
1           bne r1, r0, done
2           lw r5, 0(r2)
3           lw r6, 0(r3)
4           addiu r8, r4, 4
5           nop
6           addu r7, r5, r6
7           nop
8           sw r7, 0(r8)
9           ...
10  done:   addiu r10, r9, 1
```

By swapping the addu and addiu instructions, we can avoid 1 nop for the dependence between the addu and the lw instructions. Regardless of how the addiu and addu instructions are scheduled, there will always be a nop right before the sw instruction due to the dependence.

	Dynamic Transaction		Cycle 1	2	3	4	5	6	7	8	9	10	11	12	13
1	`bne`	`r1, r0, done`	F	D	X0	X1	M	W							
2	`lw`	`r5, 0(r2)`		F	D	X0	X1	M	W						
3	`lw`	`r6, 0(r3)`			F	D	X0	X1	M	W					
4	`addiu`	`r8, r4, 4`				F	D	X0	X1	M	W				
5	`nop`						F	D	X0	X1	M	W			
6	`addu`	`r7, r5, r6`						F	D	X0	X1	M	W		
7	`nop`								F	D	X0	X1	M	W	
8	`sw`	`r7, 0(r8)`								F	D	X0	X1	M	W

Figure 13: Software Scheduling Pipeline

Instruction 6 (addu)'s D stage depends on instruction 2 (lw)'s M stage and instruction 3 (lw)'s M stage. Instruction 7 (sw)'s D stage depends on instruction 4 (addiu)'s X1 stage and instruction 6 (addu)'s X1 stage.

4.c Resolving Data Hazards with Stalling

	Dynamic Transaction		Cycle 1	2	3	4	5	6	7	8	9	10	11	12	13	14
1	`bne`	`r1, r0, done`	F	D	X0	X1	M	W								
2	`lw`	`r5, 0(r2)`		F	D	X0	X1	M	W							
3	`lw`	`r6, 0(r3)`			F	D	X0	X1	M	W						
4	`addu`	`r7, r5, r6`				F	D	D	D	X0	X1	M	W			
5	`addiu`	`r8, r4, 4`					F	F	F	D	X0	X1	M	W		
6	`sw`	`r7, 0(r8)`								F	F	D	X0	X1	M	W

Figure 14: Hardware Stalling Pipeline

Instruction 6 (addu)'s D stage depends on instruction 2 (lw)'s M stage and instruction 3 (lw)'s M stage.
Instruction 7 (sw)'s D stage depends on instruction 4 (addiu)'s X1 stage and instruction 6 (addu)'s X1 stage.

4.d New Stall Signal

```
ostall_load_use_X_rs_D
  =    val_D && rs_en_D && (val_X0 || val_X1) && (rf_wen_X0 || rf_wen_X1)
    && ((inst_rs_D == rf_waddr_X0) || (inst_rs_D == rf_waddr_X1))
    && ((rf_waddr_X0 != 0) || (rf_waddr_X1 != 0))
    && ((op_X0 == LW) || (op_X1 == LW))
ostall_load_use_X_rt_D
  =    val_D && rt_en_D && (val_X0 || val_X1) && (rf_wen_X0 || rf_wen_X1)
    && ((inst_rt_D == rf_waddr_X0) || (inst_rt_D == rf_waddr_X1))
    && ((rf_waddr_X0 != 0) || (rf_waddr_X1 != 0))
    && ((op_X0 == LW) || (op_X1 == LW))
ostall_alu_use_X_rs_D
  =    val_D && rs_en_D && val_X0 && rf_wen_X0
    && (inst_rs_D == rf_waddr_X0)
    && (rf_waddr_X0 != 0)
ostall_alu_use_X_rt_D
  =    val_D && rt_en_D && val_X0 && rf_wen_X0
    && (inst_rt_D == rf_waddr_X0)
    && (rf_waddr_X0 != 0)
stall_D
  =    val_D && !squash_D
    && ((ostall_load_use_X_rs_D || ostall_load_use_X_rt_D) ||
       (ostall_alu_use_X_rs_D || ostall_alu_use_X_rt_D))
```

4.e Resolving Control Hazards with Scheduling and Speculation

	Dynamic Transaction		Cycle 1	2	3	4	5	6	7	8	9	10
1	`bne`	`r1, r0, done`	F	D	X0	X1	M	W				
2	`lw`	`r5, 0(r2)`		F	D	X0	-	-	-			
3	`lw`	`r6, 0(r3)`			F	D	-	-	-	-		
4	`addu`	`r7, r5, r6`				F	-	-	-	-	-	
5	`addiu`	`r10, r9, 1`					F	D	X0	X1	M	W

Figure 15: Squashing Pipeline

	Dynamic Transaction		Cycle 1	2	3	4	5	6	7	8	9	10
1	`bne`	`r1, r0, done`	F	D	X0	X1	M	W				
2	`lw`	`r5, 0(r2)`		F	D	X0	X1	M	W			
3	`lw`	`r6, 0(r3)`			F	D	-	-	-	-		
4	`addu`	`r7, r5, r6`				F	-	-	-	-	-	
5	`addiu`	`r10, r9, 1`					F	D	X0	X1	M	W

Figure 16: Branch Delay Slot Pipeline