CSCI2400 Final Practice Problems

December 12, 2013

1 TLB and Virtual Memory

The following problem concerns the way virtual addresses are translated into physical addresses.

- The memory is byte addressable.
- Memory accesses are to **1-byte words** (not 4-byte words).
- Virtual addresses are 10 bits wide.
- Physical addresses are 14 bits wide.
- The page size is 64 bytes.
- The TLB is 2-way set associative with 8 total entries.
- The L1 Cache is direct mapped, with a 4-byte block size and 64 total bytes.

In the following tables, **all numbers are given in hexadecimal and the left-most value is byte #0 and the right-most byte is byte #3, where applicable.**. The contents of the TLB, a portion of the page tables, and the 16 entries of the Cache are as follows:

TLB			
Index	Tag	PPN	Valid
0	1	0d	1
	-	–	0
1	1	18	1
	3	0c	1
2	2	16	1
	-	–	0
3	2	0e	1
	-	–	0

Page Table		
VPN	PPN	Present
000	019	1
001	001	1
002	03f	1
003	020	1
004	00d	1
005	018	1
007	001	1
008	015	1
009	000	1
00a	016	1
00b	00e	1
00c	03b	1
00d	00c	1
00f	039	1

Cache			
Index	Valid	Tag	Data
0	1	3b	BF3A02F3
1	1	4a	E8E7BA4F
2	1	12	23033CCA
3	1	16	7AFB27EE
4	1	2f	8F9F64E8
5	1	3e	EA13BEFD
6	1	2b	FEA8AAA6
7	1	4c	BD501308
8	1	0d	4D011D8E
9	0	1b	7EFEB6ED
10	1	2b	860DFCB3
11	1	15	9D769441
12	1	3a	62DA7A7D
13	1	5b	C8D747DD
14	1	6f	CA8DC445
15	1	7a	90FAAF41

1. The box below shows the format of a virtual address. Indicate (by labeling the diagram) the fields (if they exist) that would be used to determine the following: (If a field doesn't exist, don't draw it on the diagram.)

VPO	The virtual page offset	*TLBI*	The TLB index
VPN	The virtual page number	*TLBT*	The TLB tag

9	8	7	6	5	4	3	2	1	0

2. The box below shows the format of a physical address. Indicate (by labeling the diagram) the fields that would be used to determine the following: *PPO* (The physical page offset) and *PPN* (The physical page number).

11	10	9	8	7	6	5	4	3	2	1	0

3. For the given virtual addresses, indicate the TLB entry accessed and the physical address. **Indicate whether the TLB misses and whether the entry is or is not in the page table.** If the physical page number and address can not be determined, write "N/A". Then if a physical address exists indicate the cache translation parts, if its a cache hit, and a value if applicable. If any part can't be determined just write "N/A".

Virtual address: `0x28e`

(a) Virtual address format (one bit per box)

9	8	7	6	5	4	3	2	1	0

(b) Address translation

Parameter	Value
VPN	0x
TLB Index	0x
TLB Tag	0x
TLB Hit? (Y/N)	
In Page Table? (Y/N)	
PPN	0x

(c) Physical address format (one bit per box)

11	10	9	8	7	6	5	4	3	2	1	0

(d) Cache Translation

Parameter	Value
PPN	0x
Block Offset	0x
Cache Index	0x
Cache Tag	0x
Cache Hit? (Y/N)	
Value	0x

Virtual address: `0x300`

(a) Virtual address format (one bit per box)

9	8	7	6	5	4	3	2	1	0

(b) Address translation

Parameter	Value
VPN	0x
TLB Index	0x
TLB Tag	0x
TLB Hit? (Y/N)	
In Page Table? (Y/N)	
PPN	0x

(c) Physical address format (one bit per box)

11	10	9	8	7	6	5	4	3	2	1	0

(d) Cache Translation

Parameter	Value
PPN	0x
Block Offset	0x
Cache Index	0x
Cache Tag	0x
Cache Hit? (Y/N)	
Value	0x

2 Signals

```
pid_t pid;
int counter = 0;

void handler1(int sig) {
    counter ++;
    printf("counter = %d\n", counter);
    fflush(stdout); /* Flushes the printed string to stdout */
    kill(pid, SIGUSR1);
}
void handler2(int sig) {
    counter += 3;
    printf("counter = %d\n", counter); exit(0);
}

int main() {
    signal(SIGUSR1, handler1);

    if ((pid = fork()) == 0) {
        signal(SIGUSR1, handler2);
        kill(getppid(), SIGUSR1);
        while(1) {};
    } else {
        pid_t p;
        int status;

        if ((p = wait(&status)) > 0) {
            counter += 2;
            printf("counter = %d\n", counter);
        }
    }
}
```

1. What is the output of this program?