ICS Homework 14

May 25, 2022

1 Organization

1.1 Virtual Memroy 1

Which type of address is used in each of following scenarios, **virtual** or **physical** address?

The address of variables in C program	
The address stored in a C pointer	
The address of a C pointer	
The address used in looking up L1 cache	
The value in CR3	
The address in L2 PTE	
The address in L4 PTE	
The value in PC register	

1.2 Virtual Memory 2

The lookup of L1 cache usually consists of three steps:

1. locating the set
2. comparing the tag of each cache line in the set
3. returning bytes from cache or loading 1 value from next level memory system.

As the lookup uses physical address, the hardware can only start the cache lookup after address translation is completed. How to make cache lookup and address translation parallelized?

Show the requirements to the parameters of your paging and cache system.

2 Address Translation

This problem concerns the way virtual addresses are translated into physical addresses. Below are the specifications of the system on which the translation occurs:

- The main memory is byte addressable.
- The memory accesses are to **1-byte** words (not 4-byte words).
- The system uses **one-level** page table.
- Virtual addresses are **25 bits** wide.
- PPN is **9 bits** wide.
- The amount of PTE is 2^{14}.
- The TLB is **4-way** set associative with **16** total entries.

2.1

The VPO bits	_____
The physical address bits	_____
The page size	_____
The number of bits for TLBT	_____

The contents of the TLB and first 16 entries of the page table are given below. All numbers are in hexadecimal. According to the illustration and answer the questions. Please fill in the blanks.

Set	Tag	PPN	Valid	Tag	PPN	Valid
0	0CD	09	1	AA1	00	1
	3E0	62	0	C4C	48	1
1	312	45	0	010	75	1
	987	3A	1	D39	3F	0
2	038	E3	0	0A7	13	1
	18B	52	1	49B	11	0
3	6C0	42	0	075	50	0
	013	39	1	0F2	0D	0

TLB: 4 sets, 16 entries, 4-way set associative

VPN	PPN	Valid	VPN	PPN	Valid
00	39	1	08	OD	0
01	52	1	09	45	0
02	E3	0	0A	13	1
03	00	1	0B	3A	1
04	11	0	0C	09	1
05	50	0	0D	42	0
06	62	0	0E	3F	0
07	75	1	0F	48	1

Page Table: Only the first 16 PTEs are shown

2.2

Please translate virtual address to physical address and get the data from cache if hit (otherwise entering '–')

Parameter	Value
Virtual Address	0x014F2D3
VPN	0x____
TLB Index	0x____
TLB Tag	0x____
TLB Hit? (Y/N)	__
Page Fault? (Y/N)	__
PPN	0x____
Physical Address	0x____

Parameter	Value
Virtual Address	0x07C01A5
VPN	0x____
TLB Index	0x____
TLB Tag	0x____
TLB Hit? (Y/N)	__
Page Fault? (Y/N)	__
PPN	0x____
Physical Address	0x____

3 Locking

What are problems in the following simple implementation of lock?

```
1 typedef struct __lock_t {
2     int flag;
```

```
} lock_t;

void init (lock_t *mutex) {
    /* 0 -> lock is available 1 -> held */
    mutex->flag = 0;
}

void lock (lock_t *mutex) {
    /* spin-wait (do nothing) */
    while (mutex->flag == 1) ;
    /* now SET it! */
    mutex->flag = 1;
}

void unlock(lock_t *mutex) { mutex->flag = 0; }
```

4 Semaphore

Let p denote the number of producers, c the number of consumers, and n the buffer size in units of items. Consider the following buffer implementation. For each of the following scenarios, indicate whether the **mutex** semaphore is necessary or not to implement function **sbuf_insert** and **sbuf_remove**.

```
typedef struct {
  int *buf;     /* Buffer array */
  int n;        /* Maximum number of slots */
  int front;    /* buf[(front+1)%n] is first item */
  int rear;     /* buf[rear%n] is last item */
  sem_t mutex; /* Protects accesses to buf */
  sem_t slots; /* Counts available slots */
  sem_t items; /* Counts available items */
} sbuf_t;
```

A. $p = 1, c = 1, n > 1$
B. $p = 1, c = 1, n = 1$
C. $p > 1, c > 1, n = 1$