Homework 4: Memory [Gavin McRoy]

Due date: 11:59PM Nov. 8

1. Consider an array declared in C as "double a[100];". How many 64-byte cache lines are required to hold the complete array? [4pts]

You could get the entire array in 13 lines

2. Consider the byte address 0x002468ac. What is the value modulo 64? (That is, what is the offset of this address within a 64-byte block?) [4pts]

0x0000002C

3. Consider the byte address 0x002468ac. What is the value shifted to the right by 6 bits? (That is, what is the block address corresponding to this byte address when using 64-byte blocks?) [4pts]

0x000091A2

4. Consider matrix transpose written in C. Which array is exhibiting spatial locality: array "a", "b", or both? (Note that NROWS and NCOLS could each be relatively large compared to the size of the cache.) [4pts]

```
for(i=0; i<NROWS; i++){
  for(j=0; j<NCOLS; j++){
    b[i][j] = a[j][i];
  }
}
```

I'm pretty sure b is exhibiting spatial locality and a is not

5. Consider a 4 GB byte-addressable main memory (32-bit address) with a level-1 data cache that is eight-way set-associative, 32 KB in size, with 64-byte block size. [12pts]

a) How many total blocks are there in cache?

Cache Memory / Block Size = 32KB / 64 = 512

b) How many sets are there?

Each set has 64 blocks , 8 sets so 8 * 64 = 512

c) Show how the main memory address is partitioned into fields for the cache access and give the bit lengths of these fields.

Memory Address Length = 32 Bits

Index Bits = 6 Bits

Offset Bits = 6

Memory Address Bit Length = 20 (32 - 6+6)

6. Consider a direct-mapped data cache design in which a 32-bit address is divided into these three fields: 20-bit tag, 9-bit index, and 3-bit offset. [16pts]

a) How large is a line in the number of bytes?

2^3 = 8

b) How many lines are in the cache?

2^9 = 512

c) How large is the cache in the number of bytes?

numLines * sizeOfLine = 512 * 8 = 4096

d) For the following segment of code written in C, where "sum" and the array "a" are typed as 4-byte integers, what is the miss rate?

(Assume the variable "sum" and the loop index "i" are register-allocated by the compiler within the body of the loop and thus do not cause data cache accesses within the loop.)

```
for(i=0;i<4096;i++){
  sum = sum + a[i];
}
```

8 Byes for a block, so integers are 4 bytes, 2 integers per block. With each access there is one hit and one miss. so 2048 misses, 4096 accesses, miss rate is roughly 50%?

7. Assume a 256-byte main memory and a four-line cache with four bytes Per line. The cache is initially empty. For the byte address reference stream (reads) given below circle which of the references are hits for the different cache placement schemes. Also, show the final contents of the cache. (The byte addresses are in decimal.) [16pts]

a) direct-mapped

Red are the cache hits

0, 16, 1, 31, 2, 32, 3, 17, 4, 18

Is this right or not what you were expecting?

Line	Content
0	[16 17 18 19]
1	[4 5 6 7]
2	<Empty>

3	28 29 30 31

b) fully-associative with first-in-first-out replacement

0, 16, 1, 31, 2, 32, 3, 17, 4, 18

Bank #	Tag	Content
0	00001	4 5 6 7
1	00100	16 17 18 19
2	00111	28 29 30 31
3	01000	32 33 34 35

8. Consider a computer with a paged logical address space with 16 pages and each page is 2 Kbytes. The logical address space is mapped into a 512-Kbyte of physical memory space. (12pts)

i) Draw the fields in the logical and physical address and show the number of bits of each field.

Logical Address:

Logical Page Number 8 = 2^3 so it requires 3 bits

offset is 2kb so 2^11 means it requires 11 bits.

Physical Address:

Page Frame Number 512kb / 2kb = 256 = 2^8 so it requires 8 bits

Offset is 2kb so 2^11 means it requires 11 bits

ii) Draw the page table of a process and show the number of entries in the table and number of bits per entry.

Page Number	Valid Bit	Bit Frame #
0	1	8
1	1	8
2	1	8
3	1	8
4	1	8
5	1	8
6	1	8
7	1	8

iii) Populate the page table for process, namely A, which is currently running on the CPU. Several pages of process A is in the physical memory as follows:

	…
#frame 10	Page 2 of Process A
#frame 11	Page 4 of Process A
#frame 12	Page 1 of Process A
#frame 13	Page 7 of Process A
	…

Page Number	Valid Bit	Bit Frame #
0	0	
1	1	0000 1100 (12)
2	1	0000 1010 (10)
3	0	
4	1	0000 1011 (11)
5	0	
6	0	
7	1	0000 1101 (13)

9. Consider paged virtual memory systems. Assume a page size of 256 bytes (2^8), and that processes in this system can have a maximum virtual address space of 64K bytes (2^{16}). The system is currently configured with 8K (2^{13}) bytes of physical memory. (12pts)

i) How many pages are in the virtual address space?

64KB / 256 = 2^16 / 2^8 = 2^8 = 256

ii) How many pages are in the physical address space?

8kb / 256 = 2^13/2^8 = 2^5 = 32

iii) A user process generates the virtual address 12,345 (0011000000111001 in binary). Explain how the system establishes the corresponding physical address assuming that the hardware memory management unit and transfer lookaside buffer (TLB) is used.

The highest 8 bits 0011 0000 indicate page number, while the lowest 8 bits indicate 0011 1001 the pages offset.

MMU looks into TLB for an entry with a page number corresponding to the lowest 8 bits. If an entry is found and valid bit is set to 1, the field frame # indicates a physical page frame. The actual physical address is frame # and the lowest 8 bits concat together. Else MMU just looks up frame # in the process table.

10. Consider a paged virtual memory system with a physical memory that can only contain 4 pages. Assume the execution of a program generates the following

address trace

a b c d d e f f b e

where *a*, *b*, *c*, *d*, *e*, and *f* are the pages referenced and the page frames are initially empty. (16pts)

i). How many page faults occur with first-in-first-out Page Replacement?

Time	1	2	3	4	5	6	7	8	9	10
Request	*a*	*b*	*c*	*d*	*d*	*e*	*f*	*f*	*b*	*e*
	a	a	a	a	a	b	b	c	d	d
		b	b	b	b	c	c	d	f	f
			c	c	c	d	d	f	e	e
				d	d	f	f	e	b	b
Fault?	Yes	Yes	Yes	Yes	No	Yes	No	Yes	Yes	No

ii). How many page faults occur with LRU page Replacement

Time	1	2	3	4	5	6	7	8	9	10
Request	*a*	*b*	*c*	*d*	*d*	*e*	*f*	*f*	*b*	*e*
	a	a	a	a	a	b	c	d	d	d
		b	b	b	b	c	d	f	f	f
			c	c	c	d	f	b	e	b
				d	d	f	b	e	b	e
Fault?	Yes	Yes	Yes	Yes	No	Yes	No	Yes	No	No