Full Name:_____

COMP2310/COMP6310, 2022

Final Exam

November 14, 2022

Instructions:

- The exam has a maximum score of 57 points.
- The problems vary in difficulty scale. The point value of each problem is indicated. Advice: Pile up the easy points quickly and then come back to the harder problems.
- Good luck!

1 (08):
2 (05):
3 (12):
4 (12):
5 (10):
6 (10):
TOTAL (57):

Problem 1. (8 points):

Condider the following assembly code for a C `for` loop:

```
loop:
        pushl %ebp
        movl %esp,%ebp
        movl 8(%ebp),%ecx
        movl 12(%ebp),%edx
        xorl %eax,%eax
        cmpl %edx,%ecx
        jle .L4
.L6:
        decl %ecx
        incl %edx
        incl %eax
        cmpl %edx,%ecx
        jg .L6
.L4:
        incl %eax
        movl %ebp,%esp
        popl %ebp
        ret
```

Based on the assembly code above, fill in the blanks below in its corresponding C source code. (Note: you may only use the symbolic variables `x`, `y`, and `result` in your expressions below — *do not use register names.*)

```
int loop(int x, int y)
{
   int result;

   for (_____________; _____________; result++ ) {

      ___________;

      ___________;
   }

   ___________;

   return result;
}
```

Problem 2. (5 points):

The following problem concerns basic cache lookups.

- The memory is byte addressable.
- Memory accesses are to **1-byte words** (not 4-byte words).
- Physical addresses are 12 bits wide.
- The cache is 4-way set associative, with a 2-byte block size and 32 total lines.

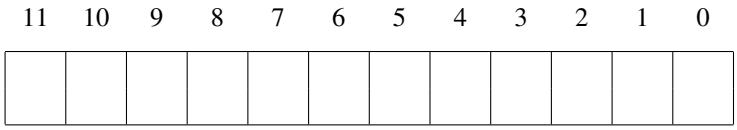
In the following tables, **all numbers are given in hexadecimal**. The contents of the cache are as follows:

4-way Set Associative Cache																
Index	Tag	Valid	Byte 0	Byte 1	Tag	Valid	Byte 0	Byte 1	Tag	Valid	Byte 0	Byte 1	Tag	Valid	Byte 0	Byte 1
0	29	0	34	29	87	0	39	AE	7D	1	68	F2	8B	1	64	38
1	F3	1	0D	8F	3D	1	0C	3A	4A	1	A4	DB	D9	1	A5	3C
2	A7	1	E2	04	AB	1	D2	04	E3	0	3C	A4	01	0	EE	05
3	3B	0	AC	1F	E0	0	B5	70	3B	1	66	95	37	1	49	F3
4	80	1	60	35	2B	0	19	57	49	1	8D	0E	00	0	70	AB
5	EA	1	B4	17	CC	1	67	DB	8A	0	DE	AA	18	1	2C	D3
6	1C	0	3F	A4	01	0	3A	C1	F0	0	20	13	7F	1	DF	05
7	0F	0	00	FF	AF	1	B1	5F	99	0	AC	96	3A	1	22	79

Part 1

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:

CO The block offset within the cache line
CI The cache index
CT The cache tag



Part 2

For the given physical address, indicate the cache entry accessed and the cache byte value returned **in hex**. Indicate whether a cache miss occurs.

If there is a cache miss, enter "-" for "Cache Byte returned".

Physical address: `3B6`

A. Physical address format (one bit per box)

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

B. Physical memory reference

Parameter	Value
Cache Offset (CO)	0x
Cache Index (CI)	0x
Cache Tag (CT)	0x
Cache Hit? (Y/N)	
Cache Byte returned	0x

Problem 3. (12 points):

3M decides to make Post-Its by printing yellow squares on white pieces of paper. As part of the printing process, they need to set the CMYK (cyan, magenta, yellow, black) value for every point in the square. 3M hires you to determine the efficiency of the following algorithms on a machine with a 2048-byte direct-mapped data cache with 32 byte blocks.
You are given the following definitions:

```
struct point_color {
    int c;
    int m;
    int y;
    int k;
};

struct point_color square[16][16];
register int i, j;
```

Assume:

- `sizeof(int) = 4`
- `square` begins at memory address 0
- The cache is initially empty.
- The only memory accesses are to the entries of the array `square`. Variables `i` and `j` are stored in registers.

A. What percentage of the writes in the following code will miss in the cache?

```
for (i=0; i<16; i++){
    for (j=0; j<16; j++) {
        square[i][j].c = 0;
        square[i][j].m = 0;
        square[i][j].y = 1;
        square[i][j].k = 0;
    }
}
```

Miss rate for writes to `square`: _____ %

B. What percentage of the writes in the following code will miss in the cache?

```
for (i=0; i<16; i++){
    for (j=0; j<16; j++) {
        square[j][i].c = 0;
        square[j][i].m = 0;
        square[j][i].y = 1;
        square[j][i].k = 0;
    }
}
```

Miss rate for writes to `square`: _____ %

C. What percentage of the writes in the following code will miss in the cache?

```
for (i=0; i<16; i++){
    for (j=0; j<16; j++) {
        square[i][j].y = 1;
    }
}
for (i=0; i<16; i++) {
    for (j=0; j<16; j++) {
        square[i][j].c = 0;
        square[i][j].m = 0;
        square[i][j].k = 0;
    }
}
```

Miss rate for writes to `square`: _____ %

Problem 4. (12 points):

This problem tests your understanding of cache conflict misses. Consider the following matrix transpose routine

```
typedef int array[2][2];

void transpose(array dst, array src) {
  int i, j;

  for (i = 0; i < 2; i++) {
    for (j = 0; j < 2; j++) {
      dst[j][i] = src[i][j];
    }
  }
}
```

running on a hypothetical machine with the following properties:

- `sizeof(int) == 4`.
- The `src` array starts at address 0 and the `dst` array starts at address 16 (decimal).
- There is a single L1 cache that is direct mapped and write-allocate, with a block size of 8 bytes.
- Accesses to the `src` and `dst` arrays are the only sources of read and write misses, respectively.

A. Suppose the cache has a total size of 16 data bytes (i.e., the block size times the number of sets is 16 bytes) and that the cache is initially empty. Then for each `row` and `col`, indicate whether each access to `src[row][col]` and `dst[row][col]` is a hit (h) or a miss (m). For example, reading `src[0][0]` is a miss and writing `dst[0][0]` is also a miss.

`dst` array		
	col 0	col 1
row 0	m	
row 1		

`src` array		
	col 0	col 1
row 0	m	
row 1		

B. Repeat part A for a cache with a total size of 32 data bytes.

`dst` array		
	col 0	col 1
row 0	m	
row 1		

`src` array		
	col 0	col 1
row 0	m	
row 1		

Problem 5. (10 points):

Consider the C program below. (For space reasons, we are not checking error return codes, so assume that all functions return normally.)

```
int main () {
    if (fork() == 0) {
        if (fork() == 0) {
            printf("3");
        }
        else {
            pid_t pid; int status;
            if ((pid = wait(&status)) > 0) {
                printf("4");
            }
        }
    }
    else {
        printf("2");
        exit(0);
    }
    printf("0");
    return 0;
}
```

For each of the following strings, circle whether (Y) or not (N) this string is a possible output of the program.

A. `32040` Y N

B. `34002` Y N

C. `30402` Y N

D. `23040` Y N

E. `40302` Y N

Problem 6. (10 points):

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 16 bits wide.
- Physical addresses are 14 bits wide.
- The page size is 1024 bytes.
- The TLB is 4-way set associative with 16 total entries.

In the following tables, **all numbers are given in hexadecimal**. The contents of the TLB and the page table for the first 32 pages are as follows:

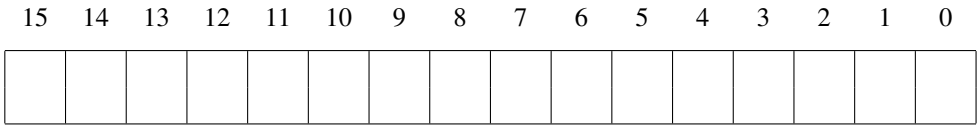
TLB			
Index	Tag	PPN	Valid
0	8	7	1
	F	6	1
	0	3	0
	1	F	1
1	1	E	1
	2	7	0
	7	3	0
	B	1	1
2	0	0	0
	C	1	0
	F	8	1
	7	6	1
3	8	4	0
	3	5	0
	0	D	1
	2	9	0

Page Table					
VPN	PPN	Valid	VPN	PPN	Valid
00	2	0	10	1	1
01	5	1	11	3	0
02	7	1	12	9	0
03	9	0	13	7	1
04	F	1	14	D	1
05	3	1	15	5	0
06	B	0	16	E	1
07	D	1	17	6	0
08	7	1	18	1	0
09	C	0	19	0	1
0A	3	0	1A	8	1
0B	1	1	1B	C	0
0C	0	1	1C	0	0
0D	D	0	1D	2	1
0E	0	0	1E	7	0
0F	1	0	1F	3	0

Part 1

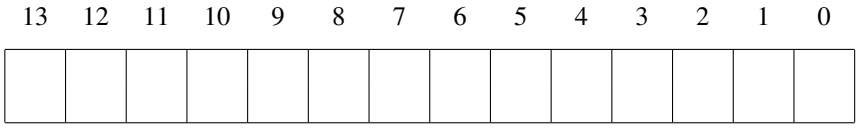
A. The box below shows the format of a virtual address. Indicate (by labeling the diagram) the fields (if they exist) that would be used to determine the following: (If a field doesn't exist, don't draw it on the diagram.)

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



B. 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
PPN The physical page number



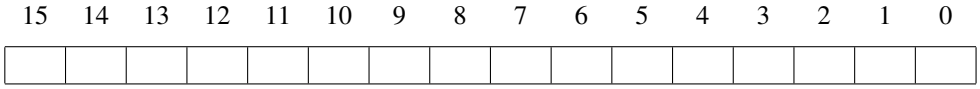
Part 2

For the given virtual addresses, indicate the TLB entry accessed and the physical address. Indicate whether the TLB misses and whether a page fault occurs.

If there is a page fault, enter "-" for "PPN" and leave part C blank.

Virtual address: `2F09`

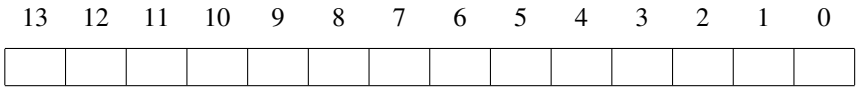
A. Virtual address format (one bit per box)



B. Address translation

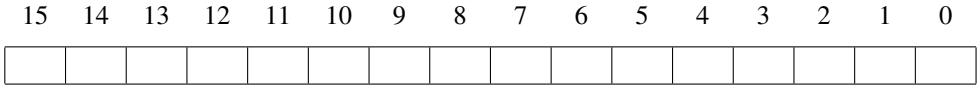
Parameter	Value
VPN	0x
TLB Index	0x
TLB Tag	0x
TLB Hit? (Y/N)	
Page Fault? (Y/N)	
PPN	0x

C. Physical address format (one bit per box)



Virtual address: `0C53`

A. Virtual address format (one bit per box)



B. Address translation

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

C. Physical address format (one bit per box)