SET C

CPTS 360: Systems Programming — Fall 2023
Exam 1
Time Limit: 45 Minutes

This exam has 15 questions worth a total of 100 points.
This document contains a total of 16 pages.

The problems are of varying difficulty. The point value of each problem is indicated on the right.

Last Name: ____________________________________

First Name: ____________________________________

Instructions:

- Make sure that your exam is not missing any sheets, then clearly write your full name on the front page.
- This exam is closed-book, closed-note (except for 1-page double-sided handwritten note sheets).
- You may not use any electronic devices.
- You can use basic/scientific calculators.
- Write your answers in the space provided below the problem.
- If you make a mess, clearly indicate your final answer.
- For MCQ questions, clearly mark/circle the correct answer.

Do not start the exam until instructed.

Good Luck!

Question	Points	Score
1	2	
2	2	
3	2	
4	2	
5	2	
6	10	
7	10	
8	5	
9	5	
10	10	
11	10	
12	10	
13	10	
14	10	
15	10	
Total:	100	

Do not write anything on this page.

1. Consider the following C declaration: **2 Points**

   `int* foo, bar, foobar;`

   True or False: `int** foo`: A pointer to an `int`.

   A. True

   B. False

2. The following statement prints address of `foo`: `printf("%d\n", *bar);` **2 Points**

   A. True

   B. False

3. Reference array elements in succession is Temporal Locality. **2 Points**

   A. True

   B. False

4. Local linker symbols are not local program variables. **2 Points**

   A. True

   B. False

5. System calls are Asynchronous Exceptions. **2 Points**

   A. True

   B. False

6. Consider the following code snippets that flips `src` matrix to `dst` matrix: **10 Points**

```
typedef int array[2][2];
void toggle(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];
    }
  }
}
```

Assume this code runs on a machine with the following properties:

- The system uses 8-bit memory addresses.
- `sizeof(int)` = 4.
- The `src` array starts at address 0_D=00000000_B
- The `dst` array starts at address 64_D=01000000_B
- A single data cache is direct-mapped with a block size of 8 bytes.
- The cache has a total size of 16 data bytes.
- The cache is initially empty.
- Accesses to the `src` and `dst` arrays are the only sources of read and write misses.

For each row and column, indicate whether the access to `src[row][col]` and `dst[row][col]` is a "Hit" or a "Miss" (see next page).

Show cache contents for each step as the code iterates over the `src` and `dst` arrays.

Note:

- Write either "Hit" or "Miss". Do NOT use short forms ("h/m").
- A decimal-to-binary conversion table is provided on Page 16 for your convenience.

src array	col=0	col=1
row=0		
row=1		

dst array	col=0	col=1
row=0		
row=1		

Solution: All entries in `dst` will me “Miss”. All entries except `src[1][1]` will be “Miss”.

7. Consider the following code:

```
int x[2][128];
int i;
int sum = 0;
for (i = 0; i < 128; i++) {
  sum += x[0][i] * x[1][i];
}
```

Assume we execute this program under the following conditions:

- `sizeof(int)` = 4.
- The cache is initially empty.
- Array `x` begins at memory address `0x0` and is stored in row-major order.
- The only memory accesses are to the entries of the array `x`. All other variables are stored in registers.

(a) What is the miss rate for the following setup? **5 Points**

512 bytes cache, direct-mapped, with 16-byte cache blocks

Note: you can assume any-size memory address for the array `x` indices.

> **Solution:** Assume the cache is 512-bytes, direct-mapped, with 16-byte cache blocks. What is the miss rate? In this case, each access to `x[1][i]` conflicts with previous access to `x[0][i]`, so the miss rate is 100%.

(b) Can we reduce the miss rate in the previous setup with a larger cache size (1024 bytes)? **5 Points**

Justify your answer.

> **Solution:** If we double the cache size, then the entire array fits in the cache, so the only misses are the cold (compulsary) misses for each new block. Since each block holds four array items, the miss rate is 25%.

8. What do the following section mean in the ELF file format? Briefly Explain **5 Points**

   (i) `.text`
   (ii) `.data`
   (iii) `.rodata`
   (iv) `.bss`
   (v) `.symtab`

Solution: See CSAPP 7.4

9. Consider a system with the following properties:

- The memory is byte addressable.
- Memory accesses are to 1-byte words.
- Addresses are 12 bits wide.
- The cache is two-way set associative, with a 4-byte block size and four sets.

The contents of the cache are as follows, with all addresses, tags, and values given in hexadecimal notation:

Let us use the following symbols:

CO The cache block offset
CI The cache set index
CT The cache tag

Set index	Tag	Valid	Byte 0	Byte 1	Byte 2	Byte 3
0	00	1	40	41	42	43
	83	1	FE	97	CC	D0
1	00	1	44	45	46	47
	83	0	—	—	—	—
2	00	1	48	49	4A	4B
	40	0	—	—	—	—
3	FF	1	9A	C0	03	FF
	00	0	—	—	—	—

(a) By labeling the diagram (CO, CI, CT), indicate the fields that would be use to determine the following for the 12-bit address. **3 Points**

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

Solution: Address fields: CT: [11–4], CI: [3–2], CO: [1–0]

(b) For each of the following memory accesses, indicate if it will be a cache hit or miss when carried out *in sequence* as listed. **2 Points**

Operation	Address	Hit? **(Clearly write YES or NO)**
Read	`0x834`	
Write	`0x836`	
Read	`0xFFD`	

Solution: 0x834 No
0x836 Yes
0xFFD Yes

10. Consider the following C code files (`foo.c` and `bar.c`). **10 Points**

```c
// foo.c

#include <stdio.h>

void f(void);

int y = 15212;
int x = 15213;
int main() {
  f();
  printf("x = 0x%x y = 0x% \n", x, y);
  return 0;
}
```

```c
// bar.c

double x;
void f() {
  x = -0.0;
}
```

Assume `double`s are 8 bytes and `int`s are 4 bytes. Further, the address of `y`: `0xcd1038` and the address of `x`: `0xcd103c`. The code is compiled using the following command:

```
gcc -o foobar -Wl,--allow-multiple-definition foo.c bar.c
```

What is the output of the executable `foobar`? Explain your answer.

Solution: x=0x0, y=0x3b6c.

As y is not overwritten by x.

11. Revise the following code so that the array a can be scanned with stride-1 reference pattern. **10 Points**

```
#define N 4096

int prod(int a[N][N][N])
{
  int i, j, k, product = 1;
  for (i = N-1; i >= 0; i--) {
    for (j = N-1; j >= 0; j--) {
      for (k = N-1; k >= 0; k--) {
        product *= a[j][k][i];
      }
    }
  }
  return product;
}
```

Solution: Change `a[i][j][k]` or change loop order `(j, k, i)`.

12. Does this function have a good locality with respect to the array a? Justify your answer. **10 Points**

```
#define M 4096
#define N 1024

int sum_array_rows(int a[M][N])
{
  int i, j, sum = 0;
  for (i = 0; i < M; i++)
    for (j = 0; j < N; j++)
      sum += a[i][j];
  return sum;
}
```

Solution: Yes (due to stride-1 reference pattern).

Decimal	Binary	Decimal	Binary	Decimal	Binary
0	00000000	43	00101011	86	01010110
1	00000001	44	00101100	87	01010111
2	00000010	45	00101101	88	01011000
3	00000011	46	00101110	89	01011001
4	00000100	47	00101111	90	01011010
5	00000101	48	00110000	91	01011011
6	00000110	49	00110001	92	01011100
7	00000111	50	00110010	93	01011101
8	00001000	51	00110011	94	01011110
9	00001001	52	00110100	95	01011111
10	00001010	53	00110101	96	01100000
11	00001011	54	00110110	97	01100001
12	00001100	55	00110111	98	01100010
13	00001101	56	00111000	99	01100011
14	00001110	57	00111001	100	01100100
15	00001111	58	00111010	101	01100101
16	00010000	59	00111011	102	01100110
17	00010001	60	00111100	103	01100111
18	00010010	61	00111101	104	01101000
19	00010011	62	00111110	105	01101001
20	00010100	63	00111111	106	01101010
21	00010101	64	01000000	107	01101011
22	00010110	65	01000001	108	01101100
23	00010111	66	01000010	109	01101101
24	00011000	67	01000011	110	01101110
25	00011001	68	01000100	111	01101111
26	00011010	69	01000101	112	01110000
27	00011011	70	01000110	113	01110001
28	00011100	71	01000111	114	01110010
29	00011101	72	01001000	115	01110011
30	00011110	73	01001001	116	01110100
31	00011111	74	01001010	117	01110101
32	00100000	75	01001011	118	01110110
33	00100001	76	01001100	119	01110111
34	00100010	77	01001101	120	01111000
35	00100011	78	01001110	121	01111001
36	00100100	79	01001111	122	01111010
37	00100101	80	01010000	123	01111011
38	00100110	81	01010001	124	01111100
39	00100111	82	01010010	125	01111101
40	00101000	83	01010011	126	01111110
41	00101001	84	01010100	127	01111111
42	00101010	85	01010101	128	10000000

Table 1: Decimal to Binary Conversion Table.