FINAL EXAMINATION

Course: **COMPUTER SYSTEM PROGRAMMING II**	
Time: **60 minutes** Lecturer(s): **Dinh Dien**	Term: 1 – Academic year: **2021-2022** Code: 20CTT-B
Student name: Bùi Lê Gia Cát	Student ID: 20125071

Instructions:

- Make sure that your exam is not missing any sheets, then write your full name on the front.
- Write your answers in the space provided below the problem. If you make a mess, clearly indicate your final answer.
- The problems are of varying difficulty. The point value of each problem is indicated. Pile up the easy points quickly and then come back to the harder problems

Problem 1 (20pts).

Consider the following C declarations:

```
typedef struct
{
    long start;
    char buf[3];
    short code;
    int raw[2];
}OldSensorData;
```

```
typedef struct
{
 char buf[5];
 short code[2];
 float sense;
 double data;
} NewSensorData;
```

Using the templates below (allowing a maximum of 24 bytes), indicate the allocation of data for structs of type OldSensorData NewSensorData. Mark off and label the areas for each individual element(arrays may be labeled as a single element). Cross hatch the parts that are allocated, but not used (to satisfy alignment). Assume the Linux alignment rules discussed in class. Clearly indicate the right hand boundary of the data structure with a vertical line.

OldSensorData:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
	start																						

NewSensorData:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23

For example:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
xx	code		xxxxx		start					data			xxxxxxxxxxxxxxxx						raw				

Short explanation:

.....

.....

.....

.....

Problem 2 (20pts).

Consider the source code below, where M and N are constants declared with #define.

```
int array1[M][N];
int array2[N][M];
void copy(int i, int j)
{
     array1[i][j] = array2[j][i];
}
```

Suppose the above code generates the following assembly code:

```
copy:
pushl %ebp
movl %esp,%ebp
pushl %ebx
movl 8(%ebp),%ecx
movl 12(%ebp),%eax
leal 0(,%eax,4),%ebx
leal 0(,%ecx,8),%edx
subl %ecx,%edx
addl %ebx,%eax
sall $2,%eax
movl array2(%eax,%ecx,4),%eax
movl %eax,array1(%ebx,%edx,4)
popl %ebx
movl %ebp,%esp
popl %ebp
ret
```

What is the value of M, N?

.....

.....

.....

.....

.....

.....

.....

Problem 3 (20pts).

Procedure foo and main has the following disassembled forms on an 64-bit machine:

```
000000000000066a <foo>:
66a: 48 83 ec 38              sub $0x38,%rsp
67c: 31 c0                    xor %eax,%eax
67e: 48 c7 04 24 02 00 00     movq $0x2,(%rsp)
686: 48 c7 44 24 08 03 00     movq $0x3,0x8(%rsp)
68f: 48 c7 44 24 10 05 00     movq $0x5,0x10(%rsp)
698: 48 c7 44 24 18 07 00     movq $0x7,0x18(%rsp)
6a1: 83 e7 03                 and $0x3,%edi
6a4: 48 8b 04 fc              mov (%rsp,%rdi,8),%rax
6b8: 48 83 c4 38              add $0x38,%rsp
6bc: c3                       retq
```

```
00000000000006c2 <main>:
6c2: 48 83 ec 08              sub $0x8,%rsp
6c6: bf 03 00 00 00           mov $0x3,%edi
6cb: b8 00 00 00 00           mov $0x0,%eax
6d0: e8 95 ff ff ff           callq 66a <foo>
6d5: 48 98                    cltq
6d7: 48 83 c4 08              add $0x8,%rsp
6db: c3                       retq
6dc: 0f 1f 40 00              nopl 0x0(%rax)
```

Please complete the stack diagram on the following page.

- To help you get started, we have given you the first two rows.
- Write the actual values (for example: 1 instead of %eax), or **Unused**
- Before calling foo
  - The address of %rsp = 0x7FFFFFFFDE88**,**
  - The return address in main function called foo = 0x6d5

Stack address

0x7FFFFFFFDE88	0x6d5
0x7FFFFFFFDE80	Unused
0x7FFFFFFFDE78	Unused
0x7FFFFFFFDE70	Unused
0x7FFFFFFFDE68	0x7
0x7FFFFFFFDE60	0x5
0x7FFFFFFFDE58	0x3
0x7FFFFFFFDE50	0x2

Short explanation

.....

.....

.....

.....

.....

.....

.....

.....

.....

Problem 4 (20pts).

The following table gives the parameters for a number of different caches, where m is the number of physical address bits, C is the cache size (number of data bytes), B is the block size in bytes, and E is the number of lines per set. For each cache, determine the number of cache sets (S), tag bits (t), set index bits (s), and block offset bits (b).

Cache	m	C	B	E	S	t	s	b
1.	32		8	1		21		3
2.	32		8	1				
3.	32	2048			128			2
4.	32	1024	2		64			
5.	32			2		23	4	

Short explanation:

.....

.....

.....

.....

.....

.....

Problem 5.

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 13 bits wide.
- The cache is 2-way set associative, with a 4 byte line size and 16 total lines.

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

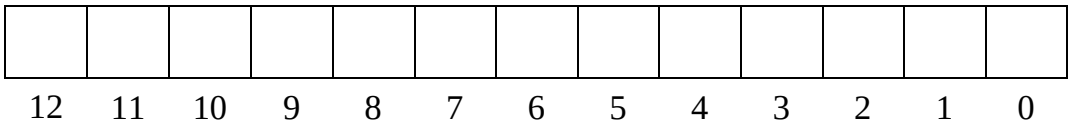
2-way Set Associative Cache												
Index	Tag	Valid	Byte 0	Byte 1	Byte 2	Byte 3	Tag	Valid	Byte 0	Byte 1	Byte 2	Byte 3
0	09	1	86	30	3F	10	00	0	99	04	03	48
1	45	1	60	4F	E0	23	38	1	00	BC	0B	37
2	EB	0	2F	81	FD	09	0B	0	8F	E2	05	BD
3	06	0	3D	94	9B	F7	32	1	12	08	7B	AD
4	C7	1	06	78	07	C5	05	1	40	67	C2	3B
5	71	1	0B	DE	18	4B	6E	0	B0	39	D3	F7
6	91	1	A0	B7	26	2D	F0	0	0C	71	40	10
7	46	0	B1	0A	32	0F	DE	1	12	88	88	37

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



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: **0x1BDC**	
Parameter	**Value**
Byte offset	
Cache Index	
Cache Tag	
Cache Hit? (Y/N)	
Cache Byte return	

Physical address: **0x016A**	
Parameter	**Value**
Byte offset	
Cache Index	
Cache Tag	
Cache Hit? (Y/N)	
Cache Byte return	

Short explanation:

.....

.....

.....

.....

.....