Question F6: Structs [10 pts]

For this question, assume a 64-bit machine and the following C struct definition.

```
typedef struct {
  char* title;     // title (e.g. "HW SW INTERFACE")
  char  dept[3];   // dept (e.g. "CSE")
  short num;       // course number (e.g. 351)
  int   enrolled;  // students enrolled
} course;
```

(A) How much memory, in bytes, does an instance of `course` use? How many of those bytes are *internal* fragmentation and *external* fragmentation? [6 pt]

`sizeof(course)`	Internal	External

(B) Assume that an instance **`course`** `c` is allocated on the stack and an array **`char`** `ar[]` is allocated 40 bytes below `c` (*i.e.* `&ar + 0x28 == (char*)&c`). Fill in the blanks below with the new ASCII characters stored in `c.dept` after the following loop is executed. Hint: recall that the values 0x30 to 0x39 correspond to the ASCII characters `'0'` to `'9'`. [4 pt]

```
for (int i = 0; i < 52; ++i) {
   ar[i] = i;
}
```

`c.dept[0]:`	'___'
`c.dept[1]:`	'___'
`c.dept[2]:`	'___'

19sp Final

1. Caches (15 points total)

You are using a byte-addressed machine with 64 KiB of Physical address space. You have a 2-way associative L1 data cache of total size 256 bytes with a cache block size of 16 bytes. It uses LRU replacement and write-allocate and write-back policies.

a) [2 pt] Give the number of bits needed for each of these:

Cache Block Offset: ___________ Cache Tag: _____________

b) [1 pt] How many **sets** will the cache have? _____________

c) [4 pts] Assume **i** and **j** are stored in registers, and that the array **x** starts at address 0x0. Give the miss rate (as a fraction or a %) for the following two loops, assuming that the cache starts out empty.

```
#define LEAP 2
#define SIZE 128
int x[SIZE];
... // Assume x has been initialized to contain values.
... // Assume the cache starts empty at this point.
for (int i = 0; i < SIZE; i += LEAP) {       // Loop 1
  x[i] = x[i] + i * i;
}
for (int j = 1; j < SIZE; j += LEAP) {       // Loop 2
  x[j] = x[j] + j * 2;
}
```

Miss Rate for Loop 1: ____________ **Miss** Rate for Loop 2: ____________

d) [8 pts] For each of the changes proposed below, indicate how it would affect the **miss rate** of each loop above in part c) *assuming that all other factors remained the same* as they were in the original problem. Circle one of: "increase", "no change", or "decrease" for each loop.

Change associativity from	Loop 1:	increase	/	no change	/	decrease
2-way to direct mapped:	Loop 2:	increase	/	no change	/	decrease
Change **LEAP** from	Loop 1:	increase	/	no change	/	decrease
2 to 4:	Loop 2:	increase	/	no change	/	decrease
Change cache size from	Loop 1:	increase	/	no change	/	decrease
256 bytes to 512 bytes:	Loop 2:	increase	/	no change	/	decrease
Change block size from	Loop 1:	increase	/	no change	/	decrease
16 bytes to 32 bytes:	Loop 2:	increase	/	no change	/	decrease

Question F8: Processes [18 pts]

(A) The following function prints out four numbers. In the following blanks, list three possible outcomes: [6 pt]

```c
void concurrent(void) {
   int x = 2, status;
   if (fork()) {
      x *= 2;
   } else {
      x -= 1;
   }
   if (fork()) {
      x += 1;
      wait(&status);
   } else {
      x -= 2;
   }
   printf("%d",x);
   exit(0);
}
```

(1) ____________________

(2) ____________________

(3) ____________________

(B) For each of the following types of synchronous exceptions, indicate whether they are intentional (I) or unintentional (U) AND whether they are recoverable (R), maybe recoverable (M) or not recoverable (N). [6 pt]

	I/U	R/M/N
Trap	_______	_______
Fault	_______	_______
Abort	_______	_______

(C) For the following scenarios, circle the outcome when the child process executes **exit(0)**. [6 pt]

Scenario:	Outcome for child:		
Parent is still executing.	Alive	Reaped	Zombie
Parent has called wait().	Alive	Reaped	Zombie
Parent has terminated.	Alive	Reaped	Zombie

Question 7: Virtual Memory **(30 total points)**

(a) Virtual memory is an extremely powerful abstraction with many benefits. For each benefit listed below, provide a brief (1-2 sentence) explanation of how VM accomplishes it.

i. (4 points) Protecting processes from one another.

ii. (4 points) Allow programs to use more memory than exists on the machine.

(b) Last month, I turned off my desktop computer, opened it up, added some more RAM (doubling it from 16GiB to 32GiB), closed it, and turned it back on.

i. (4 points) Did the size of the virtual address space change? Why or why not?

ii. (4 points) Did the size of the physical address space change? Why or why not?

iii. (4 points) After the RAM upgrade, I was able to have more programs open (and thus using memory) before performance began to degrade. Why did this happen? What was causing performance to drop after a certain number of programs were running? (1-2 sentences)

(c) Imagine the following system:

- 16-bit virtual addresses, 10-bit physical addresses.
- A page size of 16 bytes.
- 2-way set associative TLB with 8 total entries.
- All page table entries NOT in the initial TLB start as invalid.

i. (4 points) Compute the following quantities

Page offset bits: __________ PPN bits: __________

VPN bits: __________ TLB index bits: __________

ii. (6 points) The TLB has the following state:

Set	Tag	PPN	Valid	Tag	PPN	Valid
0	`0x1B2`	–	0	`0x283`	`0x3A`	1
1	`0x2FB`	`0x29`	1	`0x0E8`	`0x1D`	1
2	`0x004`	–	0	`0x346`	–	0
3	`0x3F4`	`0x1B`	1	`0x257`	`0x36`	1

Fill in the associated information for the following accesses to virtual addresses. (enter **n/a** if the answer cannot be determined):

Virtual Address	**TLB Hit?**	**Page Fault?**	**PPN**
`0x3A17`	____	____	____________________
`0x0123`	____	____	____________________

Question F10: Memory Allocation [18 pts]

(A) In the following code, briefly identify the TWO memory issues and their fixes. [6 pt]

```
int N = 32;
long* func(long src[]) {
   long* p = (long*) malloc(N * sizeof(long));
   for (int i = 0; i < N; i++) {
      p[i] += src[i];
   }
}
```

Error 1:
Fix 1:
Error 2:
Fix 2:

(B) We are using a dynamic memory allocator on a **64-bit machine** with an **explicit free list**, **16-byte boundary tags**, and **8-byte alignment**. Assume that a footer is always used. [6 pt]

Request	block addr	return value	block size	internal fragmentation in this block
`p = malloc(9);`	`0x628`	`0x`______	______ bytes	______ bytes

(C) Consider the C code shown here. Assume that the `malloc` call succeeds and that all variables are stored in memory (not registers). In the following groups of expressions, **circle the one** whose returned *value* (assume just before `return 0`) is **largest**. [6 pt]

```
#include <stdlib.h>
long glob = 10;
char* str = "351";

int main() {
    short* sp = malloc(8);
    int ONE = 1;
    free(sp);
    return 0;
}
```

Group 1:	`&sp`	`sp`	`&str`
Group 2:	`&glob`	`main`	`str`
Group 3:	`glob`	`ONE`	`*str`

End of Exam

Did you write your Student ID Number on the top-right corner of every odd page?