Question 1: Structs

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?

`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'.

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

`c.dept[0]`: '_____'

`c.dept[1]`: '_____'

`c.dept[2]`: '_____'

Question 2: Caching

We have 256 KiB of RAM and a 4-KiB L1 data cache that is 2-way set associative with 32-byte blocks and random replacement, write-back, and write allocate policies.

(A) Calculate the TIO address breakdown:

Tag bits	Index bits	Offset bits

(B) The code snippet below accesses two arrays of doubles. Assuming i is stored in a register and the cache starts *cold*, give the memory access pattern (read or write to which elements/addresses) and compute the **miss rate**.

```
#define SIZE 128
double src[SIZE]; // &src = 0x08000 (physical addr)
double dst[SIZE]; // &dst = 0x0E000 (physical addr)
for (int i = 0; i < SIZE; i += 1) {
  dst[i] = src[i];
  src[i] = i;
}
```

Per Iteration:	Access 1:	Access 2:	Access 3:
(circle) →	R / W to	R / W to	R / W to
(fill in) →	_____[i]	_____[i]	_____[i]
			Code Miss Rate: _______

(C) For each of the proposed (independent) changes, draw ↑ for "increased", — for "no change", or ↓ for "decreased" to indicate the effect on the **miss rate from Part B** for the code above:

Use float instead __________ Double the cache size __________

Half the associativity __________ No-write allocate __________

(D) Assume it takes 160 ns to get a block of data from main memory. If our L1 data cache has a hit time of 5 ns and a miss rate of 5%, what is our average memory access time (AMAT)?

ns

Question 3: Processes

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

```
void concurrent(void) {

 int n = 5;
 if (fork()) {
   n++;
   if (fork()) {
     n++;
     wait();
   }
   printf("%d, ", n);
   exit(0);
 } else {
   printf("%d, ", n);
 }
 printf("%d, ", n);
 exit(0);
}
```

(1) ____________________

(2) ____________________

(3) ____________________

(B) For the following examples of exception causes, write "**S**" for synchronous or "**A**" for asynchronous from the perspective of the user process.

System call __________ Divide by zero _________

Segmentation fault ________ Key pressed _________

(C) Fill in the following blanks with "**A**" for always, "**S**" for sometimes, and "**N**" for never if the following would be different when **context switching** to a *different* process?

Process ID _____ Program _____ PTBR _____ Condition Codes _______

(D) Is the following statement True or False? Provide a *brief* justification: a single process can execute multiple programs simultaneously.

Circle one: True / False

Justification:

Question 4: Virtual Memory

Our system has the following setup:

- 15-bit virtual addresses and 2 KiB of RAM with 256-byte pages
- A 4-entry fully-associative TLB with LRU replacement
- A PTE contains bits for valid (V), dirty (D), read (R), write (W), and execute (X)

(A) Compute the following values:

page offset width ______ # of TLB sets ______

# of virtual pages ______ minimum width of PTBR ______

(B) Assuming that the TLB is in the state shown (permission bits: 1 = allowed, 0 = disallowed), give example addresses that will fulfill the following scenarios:

TLBT	**PPN**	**Valid**	**D**	**R**	**W**	**X**
`0x20`	`0xc`	1	0	1	0	0
`0x7f`	`0xa`	1	0	1	1	0
`0x7e`	`0xf`	1	0	1	1	0
`0x04`	`0xe`	1	0	1	1	1

A value in `%rip` that causes a TLB Hit and no exception:

A *write* address that causes a TLB Hit and segmentation fault:

Question 5: Memory Allocation

```
1 #include <stdlib.h>
2 float pi = 3.14;
3
4 int main(int argc, char *argv[]) {
5   int year = 2019;
6   int* happy = malloc(sizeof(int*));
7   happy++;
8   free(happy);
9   return 0;
10 }
```

(A) Consider the C code shown above. Assume that the malloc call succeeds and happy and year are stored in memory (not in a register). Fill in the following blanks with "<" or ">" or "UNKNOWN" to compare the *values* returned by the following expressions just before return 0.

&year _________ &main

happy _________ &happy

&pi _________ happy

(B)] The code above has two memory-related errors. Use the line numbers in the code to describe what the errors are and where they occur.

Error #1:

Error #2:

(C) (Not related to code at top of page) Give one advantage that next fit placement policy has over a first fit placement policy in an implicit free list implementation.

(D) List two reasons why it would be hard to write a garbage collector for the C programming language.

Reason #1:

Reason #2:

Question 6: Java vs. C

This is an open-ended question, just make sure to avoid repeating the same point but worded differently or listing the same point but worded in opposite ways for both questions.

(A) Describe two distinct ways or things that you think C does better than Java.

1.

2.

(B) Describe two distinct ways or things that Java does better than C.

1.

2.