QUESTION 1

Please type your full name below, whereby you pledge on your honor that you will neither give nor receive any unauthorized assistance on this examination.

For the toolbar, press ALT+F10 (PC) or ALT+FN+F10 (Mac).

Paragraph Arial 10pt

Alan Biju Palayil

P 3 WORDS POWERED BY TINY

0 points Saved

QUESTION 2

The access time of DRAM is approximately _______ times that of on-board (L1) SRAM.

- ○ 200-500
- ◉ 50-100
- ○ 500-1000
- ○ 5-10

2 points Saved

QUESTION 3

What is a viable compiler-level optimization to help improve memory throughput?

- ◉ evicting unneeded pages from the TLB
- ○ mapping variables to registers
- ○ mapping addresses to cache slots
- ○ reducing array stride-length

2 points Saved

QUESTION 4

Consider the following structure declarations:

```
struct foo {
    char c[3];
    int i;
};

struct bar {
    char c1[9];
    struct foo f[3];
    char c2[3];
    int i;
};
```

Given that a `char` is 1 byte and an `int` is 4 bytes, how many bytes of memory would a single `struct bar` variable occupy, considering memory alignment?

Given that a `char` is 1 byte and an `int` is 4 bytes, how many bytes of memory would a single `struct bar` variable occupy, considering memory alignment?

Ans = 37

If we replace the two `ints` in the foo/bar declarations with `doubles` (a double is 8 bytes), how many bytes of memory would the updated `struct bar` occupy, considering memory alignment?

Ans = 53

6 points Saved

QUESTION 5

Accessing elements in a lengthy array with a small stride length demonstrates ________.

- ○ good temporal locality
- ○ poor temporal locality
- ◉ good spatial locality
- ○ poor spatial locality

2 points Saved

QUESTION 6

What cache-hit policy is typically paired with write-around?

- ○ write-allocate
- ◉ write-through
- ○ write-absorb
- ○ write-back

2 points ✓ Saved

QUESTION 7

An important advantage to using the write-back caching policy is that ________.

- ○ cache and DRAM are kept in sync at all times
- ◉ data being written to the memory hierarchy doesn't need to occupy cache space
- ○ the implementation is simpler compared to a write-through cache
- ○ write-hits may avoid accessing DRAM

2 points ✓ Saved

QUESTION 8

Keeping the total cache (payload) size fixed, ________ is most likely to improve the overall hit rate for the following code.

```
int accum = 0;
int arr[1000] = { ... };
for (int i=0; i<1000; i+=2) {
  accum += arr[i] * arr[i+1];
}
```

- ○ increasing the associativity
- ○ decreasing the block size
- ◉ increasing the block size
- ○ decreasing the associativity

3 points ✓ Saved

QUESTION 9

In caches that are further removed from the CPU (i.e., closer to DRAM), we are more likely to see _______.

- (•) higher cache complexity and increased hit time
- ( ) decreased hit time and hit rate
- ( ) lower associativity and block size
- ( ) higher associativity and smaller block sizes

2 points Saved

QUESTION 10

Consider a 2-way set-associative cache with 4-byte blocks and 8 total lines, whose contents are shown below:

Cache						
Index	Tag	Valid	Byte 0	Byte 1	Byte 2	Byte 3
0	1E0C	1	45	1C	86	F1
	0658	0	3B	FB	98	0B
	19B5	1	BF	14	E1	4B
	1348	1	81	11	E1	A5
1	004D	0	3A	6A	FE	93
	15BE	1	E6	1D	9D	81
	0197	1	40	5D	60	36
	0417	1	FB	08	E5	BF

Match each 16-bit memory address below with the result of looking that address up in the cache (note that a given answer may be used more than once).

D. **F062**	A. Hit, data=**0x11**
C. **CDA8**	B. Hit, data=**0x9D**
E. **ADF7**	C. Hit, data=**0xBF**
F. **026E**	D. Hit, data=**0x86**
F. **9A71**	E. Hit, data=**0x81**
F. **289C**	F. Miss

8 points Saved

QUESTION 11

Consider the following function which processes two arrays:

```
void procArr(int rows, int cols, int A[rows][cols], int B[rows][cols]) {
  int i, j, res;
  res = 0;
  for (i=0; i<rows; i++) {
    for (j=0; j<cols; j++) {
      res += A[i][j] + B[i][cols-j-1];
    }
  }
}
```

If the function is called with two arrays A and B where rows=3 and cols=3, fill in the tables below indicating which accesses in the corresponding array locations result in cache hits (H), misses (M), and evictions (E). Base your answer on the following cache parameters and assumptions:

- the cache is direct-mapped, with 8-byte blocks and 4 total lines
- all variables other than the arrays are mapped to registers
- an `int` is 4 bytes wide
- the cache is initially empty
- array A begins at address `0x601040` and array B at address `0x602000`

A	0	1	2
0	Miss	Hit	Eviction
1	Hit	Eviction	Eviction
2	Miss	Hit	Eviction

B	0	1	2
0	Hit	Eviction	Miss
1	Eviction	Eviction	Miss
2	Hit	Eviction	Eviction

8 points Saved

QUESTION 12

The lifetime of objects stored in the heap is _______

- ○ LIFO
- ○ permanent
- ○ FIFO
- ◉ arbitrary

2 points Saved

QUESTION 13

The malloc, free, and realloc functions in the C DMA library are ________.

- ○ system calls
- ○ kernel level functions
- ○ implemented by the MMU
- ◉ user level functions

2 points Saved

QUESTION 14

Which of the following are included when calculating internal fragmentation of a DMA implementation? Pick all that apply.

- ☐ unreachable allocated blocks
- ☐ payload
- ☑ padding
- ☑ header & footer

2 points Saved

QUESTION 15

What might be a reasonable approach to managing external fragmentation in a DMA?

- ○ always allocating in the largest available free block
- ○ preferring many small free blocks over fewer large ones
- ○ requiring that all allocated blocks be the same size
- ◉ preferring fewer large free blocks over many small ones

2 points Saved

QUESTION 16

A segregated storage based DMA runs the risk of ________

- ◉ creating massive amounts of external fragmentation due to unused blocks
- ○ having very poor "free" API throughput
- ○ creating massive amounts of internal fragmentation due to block metadata
- ○ having very poor "malloc" API throughput

2 points Saved

QUESTION 17

Consider the following DMA implementation:

```
#define ALIGNMENT 8
#define ALIGN(size) (((size) + (ALIGNMENT-1)) & ~0x7)
#define SIZE_T_SIZE (ALIGN(sizeof(size_t)))

void *malloc(size_t size)
{
  size_t blk_size = ALIGN(size + SIZE_T_SIZE);
  size_t *header  = find_fit(blk_size);
  if (header) {
    *header |= 1;
  } else {
    header = sbrk(blk_size);
    *header = blk_size | 1;
  }
  return (char *)header + SIZE_T_SIZE;
}


void *find_fit(size_t size) {
  size_t *header = heap_start();
  while ((void *)header < heap_end()) {
    if (!(*header & 1) && *header >= size) {
      return header;
    }
    header = (size_t *)((char *)header + (*header & ~1L));
  }
  return NULL;
}
void free(void *ptr)
{
  size_t *header = ptr - SIZE_T_SIZE;
  *header &= ~1L;
}
```

Assume that:

- `sizeof(size_t)` = 4
- `heap_start` and `heap_end` return the start and end addresses of the heap, respectively

Given the following program that makes use of the DMA defined above:

```
1 main() {
2   void *p0 = malloc(50);
3   void *p1 = malloc(50);
4   free(p0);
5   void *p2 = malloc(25);
6   void *p3 = malloc(50);
7   free(p1);
8   void *p4 = malloc(75);
9   void *p5 = malloc(25);
10 }
```

Compute the following heap statistics:

- Maximum aggregate payload: 175 bytes
- Maximum heap size: 275 bytes
- Total internal fragmentation immediately after line 6: 125 bytes
- Peak memory utilization: 63 %

8 points Saved

QUESTION 18

You recently joined a software startup that's building the latest and greatest smart kitchen scale. As part of the software architecture discussion, you are asked to help decide between platforms to deploy onto the smart scales. One platform uses automatic reference counting, while the other uses a tracing garbage collector. Memory on the smart scales is limited, and responsiveness is critical. With this limited information, which platform would you recommend and why?

For the toolbar, press ALT+F10 (PC) or ALT+FN+F10 (Mac).

Paragraph Arial 10pt

Automatic reference counting is comparetively better than tracing garbage collector because reference counts can be used as a source of data for other runtime optimizations.The main advantage of reference counting over tracing garbage collection is that objects are incrementally recovered as soon as they can no longer be referred, with no extended collection periods and a well-defined lifespan for each object. In real-time environments or on devices with limited memory, maintaining responsiveness is crucial. When the number of living objects exceeds the amount of memory available, tracing garbage collection cycles are triggered too frequently, and extra space is required to complete the task. The accuracy of reference counting does not deteriorate when the total amount of vacant space decreases. Reference counting when used for memory management techniques can efficiently control non-memory resources such as operating system elements, which are often much more limited than memory.

P 145 WORDS POWERED BY TINY

5 points Saved

QUESTION 19

In which of the following scenarios will a stdio output stream be flushed?

- [ ] When the process `fork`s a new child
- [x] When the buffer is filled
- [x] When the stream is line-buffered, and a newline character is written to the stream
- [ ] When a child process that shares the open file underlying the stream performs a write operation on the stream
- [x] When the process writing to the stream terminates normally (e.g., via `exit`)
- [ ] When multiple write operations take place on the stream
- [ ] When the process calls `open` on the same file underlying the stream again
- [x] When the process calls `fflush` on the stream

4 points Saved

QUESTION 20

For which of the following situations/applications is the C standard I/O library likely not well suited?

- [ ] Reading/Writing regular text files
- [x] For the implementation of a robust, low-level I/O library
- [x] Reading/Writing character special files (e.g., the network)
- [x] When used in combination with I/O system calls (e.g., `read`, `write`) on the same underlying file descriptors
- [ ] Reading/Writing binary data in regular files

3 points Saved

QUESTION 21

Consider the following program, which makes use of the buffered stdio library:

```
main() {
    int nread;
    char buf[80];
    FILE *infile = fopen("fox.txt", "r");

    if (fork() == 0) {
        nread = fread(buf, 1, 10, infile);
        write(1, buf, nread);
    } else {
        wait(NULL);
        nread = fread(buf, 1, 10, infile);
        write(1, buf, nread);
    }
}
```

What is the output when this program is run if the file "fox.txt" contains the following text?

`the quick brown fox jumps over the lazy dog`

- (•) the quick brown fox
- ( ) the quick
- ( ) brown fox brown fox
- ( ) the quick the quick

3 points Saved