Instructions You will have eighty minutes to complete the actual open-book, open-note exam. Electronic devices will be allowed only to consult notes or books from local storage; network use will be prohibited. The actual exam will be a little shorter than this practice exam.

Fall 2018 exam will not have a question of this form. The first two questions refer to the following declarations and function:

typedef struct {
    int a[3];
    short b, c;
} stuff;

stuff s[32][32];

int sum(int mode, stuff s[32][32]) {
    int i, j, a = 0;

    for (i = 0; i < 32; i++)
        for (j = 0; j < 32; j++) {
            a += s[i][j].a[mode];
            if (mode)
                a += s[i][j].b + s[i][j].c;
            else
                a += s[i][j].b;
        }

    return a;
}
For each question below, assume a 4kB direct-mapped cache that uses 32-byte blocks, the cache is initially empty, and local variables are in registers. Also assume that the array \( a \) is at the address \( 0xA0000 \) in memory.

1. Fall 2018 exam will not have a question of this form. For each of first six memory accesses via \( a \) in \( \text{sum}(0, \ a) \), what is the accessed element, what is the accessed address, and is the access a cache hit or miss?

<table>
<thead>
<tr>
<th>Access expression</th>
<th>Access address</th>
<th>Hit or miss</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \text{sum}(0, \ a) )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( \text{sum}(1, \ a) )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( \text{sum}(2, \ a) )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( \text{sum}(3, \ a) )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( \text{sum}(4, \ a) )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( \text{sum}(5, \ a) )</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

2. Fall 2018 exam will not have a question of this form. What is the expected cache-miss rate of \( \text{sum}(0, \ a) \)?

3. Fall 2018 exam will not have a question of this form. For each of first six memory accesses via \( a \) in \( \text{sum}(2, \ a) \), what is the accessed element, what is the accessed address, and is the access a cache hit or miss?

<table>
<thead>
<tr>
<th>Access expression</th>
<th>Access address</th>
<th>Hit or miss</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \text{sum}(2, \ a) )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( \text{sum}(3, \ a) )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( \text{sum}(4, \ a) )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( \text{sum}(5, \ a) )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( \text{sum}(6, \ a) )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( \text{sum}(7, \ a) )</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

4. Fall 2018 exam will not have a question of this form. What is the expected cache-miss rate of \( \text{sum}(2, \ a) \)?
The next two questions refer to a directory that contains the following files.

```c
#include <stdio.h>

int z();
int w();

int main () {
  printf("%d\n", z() + w());
  return 0;
}
```

5. Cross out the files above that turn out to be irrelevant to the result of `make && ./a.out` (in the sense that erasing the file content would not change the output).

6. What does `make && ./a.out` print? In case you don’t compute the output correctly, to improve opportunities for partial credit, list specific functions that are called and the value that each call returns.
7. The output further below is the partial result of using `readelf` on the shared library produced by `gcc -shared` on a source file. Cross out the files among the four shown below that could not have been the source file that lead to this output.

```
extern int a;
int w();

int q() {
  return a + w();
}
```

```
extern int a;
int q();

int w() {
  return a + q();
}
```

```
extern int q;
int a();

int w() {
  return q + a();
}
```

```
extern int w;
int a();

int q() {
  return w + a();
}
```

Program Headers:
```
<table>
<thead>
<tr>
<th>Type</th>
<th>Offset</th>
<th>VirtAddr</th>
<th>PhysAddr</th>
<th>FileSiz</th>
<th>MemSiz</th>
<th>Flags</th>
<th>Align</th>
</tr>
</thead>
<tbody>
<tr>
<td>LOAD</td>
<td>0x0000000000000000</td>
<td>0x0000000000000000</td>
<td>0x0000000000000000</td>
<td>0x0000000000000794</td>
<td>0x0000000000000794</td>
<td>R E</td>
<td>200000</td>
</tr>
<tr>
<td>LOAD</td>
<td>0x0000000000000df0</td>
<td>0x0000000000200df0</td>
<td>0x0000000000200df0</td>
<td>0x0000000000000240</td>
<td>0x0000000000000248</td>
<td>RW</td>
<td>200000</td>
</tr>
<tr>
<td>DYNAMIC</td>
<td>0x0000000000000e10</td>
<td>0x0000000000002e10</td>
<td>0x0000000000002e10</td>
<td>0x0000000000000000</td>
<td>0x00000000000002e10</td>
<td>RW</td>
<td>8</td>
</tr>
</tbody>
</table>
```

Dynamic section at offset 0xe10 contains 24 entries:
```
[..] ...
```

Relocation section `'.rela.dyn'` at offset 0x480 contains 9 entries:
```
<table>
<thead>
<tr>
<th>Offset</th>
<th>Info</th>
<th>Type</th>
<th>Sym. Value</th>
<th>Sym. Name + Addend</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x000000200df0</td>
<td>000000000008</td>
<td>R_X86_64_RELATIVE</td>
<td>6b0</td>
<td></td>
</tr>
<tr>
<td>0x000000200df8</td>
<td>000000000008</td>
<td>R_X86_64_RELATIVE</td>
<td>670</td>
<td></td>
</tr>
<tr>
<td>0x000000200de0</td>
<td>000000000008</td>
<td>R_X86_64_RELATIVE</td>
<td>200e08</td>
<td></td>
</tr>
<tr>
<td>0x000000200df0</td>
<td>000000000006</td>
<td>R_X86_64_GLOB_DAT</td>
<td>0000000000000000 _ITM_deregisterTMClone + 0</td>
<td></td>
</tr>
<tr>
<td>0x000000200df8</td>
<td>000300000006</td>
<td>R_X86_64_GLOB_DAT</td>
<td>0000000000000000 <strong>gmon_start</strong> + 0</td>
<td></td>
</tr>
<tr>
<td>0x000000200de0</td>
<td>000400000006</td>
<td>R_X86_64_GLOB_DAT</td>
<td>0000000000000000 a + 0</td>
<td></td>
</tr>
<tr>
<td>0x000000200df0</td>
<td>000600000006</td>
<td>R_X86_64_GLOB_DAT</td>
<td>0000000000000000 _Jv_RegisterClasses + 0</td>
<td></td>
</tr>
<tr>
<td>0x000000200df8</td>
<td>000700000006</td>
<td>R_X86_64_GLOB_DAT</td>
<td>0000000000000000 _ITM_registerTMCloneTa + 0</td>
<td></td>
</tr>
<tr>
<td>0x000000200e00</td>
<td>000800000006</td>
<td>R_X86_64_GLOB_DAT</td>
<td>0000000000000000 __cxa_finalize + 0</td>
<td></td>
</tr>
</tbody>
</table>
```

Relocation section `'.rela.plt'` at offset 0x558 contains 3 entries:
```
<table>
<thead>
<tr>
<th>Offset</th>
<th>Info</th>
<th>Type</th>
<th>Sym. Value</th>
<th>Sym. Name + Addend</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00000201018</td>
<td>000300000007</td>
<td>R_X86_64_JUMP_SLO</td>
<td>0000000000000000 <strong>gmon_start</strong> + 0</td>
<td></td>
</tr>
<tr>
<td>0x00000201020</td>
<td>000500000007</td>
<td>R_X86_64_JUMP_SLO</td>
<td>0000000000000000 q + 0</td>
<td></td>
</tr>
<tr>
<td>0x00000201028</td>
<td>000800000007</td>
<td>R_X86_64_JUMP_SLO</td>
<td>0000000000000000 __cxa_finalize + 0</td>
<td></td>
</tr>
</tbody>
</table>
```

[..]
8. What are all of the possible outputs of the following program?
In case you don’t list all of the possible outputs correctly, to improve opportunities for partial credit, show how you arrived at your answer by sketching one or more process graphs.

```c
#include "csapp.h"

int main() {
    pid_t pid1, pid2;
    int status;
    char buffer[1];

    pid1 = Fork();
    write(1, "F", 1);

    if (pid1 == 0) {
        exit(6);
    }

    pid2 = Fork();
    write(1, "S", 1);

    if (pid2 == 0) {
        exit(7);
    }

    Waitpid(pid2, &status, 0);
    buffer[0] = WEXITSTATUS(status) + '0';
    Write(1, buffer, 1);

    Waitpid(pid1, &status, 0);
    buffer[0] = WEXITSTATUS(status) + '0';
    Write(1, buffer, 1);

    return 0;
}
```
9. What are all of the possible outputs of the following program?
Again, to improve opportunities for partial credit, show how you arrived at your answer
by sketching one or more process graphs.

```c
#include "csapp.h"

int main() {
    int fds[2];
    char buffer[1];

    Pipe(fds);

    if (Fork() == 0) {
        if (Fork() == 0) {
            Write(1, "2", 1);
            Write(fds[1], "5", 1);
            return 0;
        }
        Write(1, "1", 1);
        Read(fds[0], buffer, 1);
        return 0;
    }
    Write1, "1", 1);
    Read(fds[0], buffer, 1);
    return 0;
}

Wait(NULL);

Write1, "3", 1);
Write(fds[1], "4", 1);

return 0;
}
```
10. Consider a memory system with 16 bit virtual addresses, 16 bit physical addresses with a page size of 256 bytes. In the page table below, some entries are not listed; assume that those entries are all marked as invalid. For each of the following virtual addresses, indicate its physical address or indicate that it is a page fault.

<table>
<thead>
<tr>
<th>Virtual address</th>
<th>Physical address or page fault</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x3111</td>
<td></td>
</tr>
<tr>
<td>0x4161</td>
<td></td>
</tr>
<tr>
<td>0x00aa</td>
<td></td>
</tr>
<tr>
<td>0x0880</td>
<td></td>
</tr>
<tr>
<td>0x2198</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>VPN</th>
<th>PPN/valid?</th>
<th>VPN</th>
<th>PPN/valid?</th>
<th>VPN</th>
<th>PPN/valid?</th>
<th>VPN</th>
<th>PPN/valid?</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00</td>
<td>0x00 / 0</td>
<td>0x01</td>
<td>0x08 / 0</td>
<td>0x02</td>
<td>0x10 / 0</td>
<td>0x03</td>
<td>0x18 / 0</td>
</tr>
<tr>
<td>0x04</td>
<td>0x20 / 0</td>
<td>0x05</td>
<td>0x28 / 0</td>
<td>0x06</td>
<td>0x30 / 0</td>
<td>0x07</td>
<td>0x38 / 0</td>
</tr>
<tr>
<td>0x08</td>
<td>0x40 / 0</td>
<td>0x09</td>
<td>0x48 / 0</td>
<td>0x0a</td>
<td>0x50 / 0</td>
<td>0x0b</td>
<td>0x58 / 0</td>
</tr>
<tr>
<td>0x0c</td>
<td>0x60 / 0</td>
<td>0x0d</td>
<td>0x68 / 0</td>
<td>0x0e</td>
<td>0x70 / 0</td>
<td>0x0f</td>
<td>0x78 / 0</td>
</tr>
<tr>
<td>0x10</td>
<td>0x80 / 0</td>
<td>0x11</td>
<td>0x88 / 0</td>
<td>0x12</td>
<td>0x90 / 0</td>
<td>0x13</td>
<td>0x98 / 0</td>
</tr>
<tr>
<td>0x14</td>
<td>0xa0 / 0</td>
<td>0x15</td>
<td>0xaa / 0</td>
<td>0x16</td>
<td>0xb0 / 0</td>
<td>0x17</td>
<td>0xb8 / 0</td>
</tr>
<tr>
<td>0x18</td>
<td>0xc0 / 0</td>
<td>0x19</td>
<td>0xc8 / 0</td>
<td>0x1a</td>
<td>0xd0 / 0</td>
<td>0x1b</td>
<td>0xd8 / 0</td>
</tr>
<tr>
<td>0x1c</td>
<td>0xe0 / 0</td>
<td>0x1d</td>
<td>0xe8 / 0</td>
<td>0x1e</td>
<td>0xf0 / 0</td>
<td>0x1f</td>
<td>0xf8 / 0</td>
</tr>
<tr>
<td>0x20</td>
<td>0x01 / 0</td>
<td>0x21</td>
<td>0x02 / 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

7
For the following two questions, a word is defined to be 16 bytes, each cell in a diagram represents a word, and an underlined number \( N \) is a shorthand for \( N \) times 16.

Assume that an allocator produces word-aligned payload pointers, uses a word-sized header and footer for each allocated block, uses a 2-word prolog block and a 1-word terminator block, coalesces unallocated blocks, and is confined to 18 words of memory that is initially filled with 0s. Show a header in a diagram as a value for the block size over a 0 or 1 to indicate the block’s allocation status; draw a footer as just the block size.

The left-hand column below contains a sequence of \texttt{malloc} and \texttt{free} calls that are handled by the allocator. Fill in the left-hand column by showing relevant header and footer values just after each step on the left. The first row of the left column is blank so that you can show the initial state of memory in the first row of the right column.

11. Show the state of memory after each step for an allocator that uses a \texttt{first-fit} allocation strategy, where the allocator searches from the start of an \textit{implicit} free list.

\begin{verbatim}
p1 = malloc(6)  
p2 = malloc(4)  
free(p1)  
p3 = malloc(2)  
p4 = malloc(2)  
free(p2)  
free(p4)
\end{verbatim}
12. Show the state of memory after each step for an allocator that uses a best-fit allocation strategy, where the allocator finds the smallest unallocated block that matches the requested allocation size.

```
p1 = malloc(5)                   
p2 = malloc(1)                   
free(p1)                   
p3 = malloc(3)                   
free(p2)                   
free(p3)                   
```
Answers

1. Access expression | Access address | Hit or miss
--- | --- | ---
\(s[0][0].a[0]\) | 0xA0000 | miss
\(s[0][0].b\) | 0xA000c | hit
\(s[0][1].a[0]\) | 0xA0010 | hit
\(s[0][1].b\) | 0xA001c | hit
\(s[0][2].a[0]\) | 0xA0020 | miss
\(s[0][2].b\) | 0xA002c | hit

2. 25%

3. Access expression | Access address | Hit or miss
--- | --- | ---
\(s[0][0].a[2]\) | 0xa0008 | miss
\(s[0][0].b\) | 0xa000c | hit
\(s[0][0].c\) | 0xa000e | hit
\(s[0][1].a[2]\) | 0xa0018 | hit
\(s[0][1].b\) | 0xa001c | hit
\(s[0][1].c\) | 0xa001e | hit

4. 16.7%

5. Cross out \(x.c\)

6. Output: \(7; y(na)\) returns 5, \(z()\) returns 5, \(w()\) returns 2

7. Cross out all except \(2.c\)

8. Four possible outputs: FFSS76, FSFS76, FSSF76, FSS7F6

9. Two possible outputs: 123, 213
10. Virtual address | Physical address or page fault
---|---
0x3111 | 0x8911
0x4161 | page fault
0x00aa | page fault
0x0880 | 0x4080
0x2198 | 0x0998

11. \[ \begin{array}{cccccc}
\text{p1 = malloc(6)} & 2 & 2 & 2 & 2 & 15 \\
\text{p2 = malloc(4)} & 2 & 2 & 2 & 2 & 8 \\
\text{free(p1)} & 2 & 2 & 2 & 2 & 8 \\
\text{p3 = malloc(2)} & 2 & 2 & 2 & 2 & 4 \\
\text{p4 = malloc(2)} & 2 & 2 & 2 & 2 & 4 \\
\text{free(p2)} & 2 & 2 & 2 & 2 & 4 \\
\text{free(p4)} & 2 & 2 & 2 & 2 & 4
\end{array} \]

Shading above is not required in an answer.
12.

\[
\begin{align*}
p1 &= \text{malloc}(5) \quad 2 & 2 & 7 & 1 & 0 \quad 7 & 8 & 0 & 8 & 0 \\
p2 &= \text{malloc}(1) \quad 2 & 2 & 7 & 1 & 0 \quad 7 & 3 & 3 & 5 & 5 & 0 \\
\text{free}(p1) \quad 2 & 2 & 7 & 1 & 0 \quad 7 & 3 & 3 & 5 & 5 & 0 \\
p3 &= \text{malloc}(3) \quad 2 & 2 & 7 & 1 & 0 \quad 7 & 3 & 3 & 5 & 5 & 0 \\
\text{free}(p2) \quad 2 & 2 & 10 & 1 & 0 \quad 10 & 5 & 5 & 0 \\
\text{free}(p3) \quad 2 & 2 & 15 & 0 & 1 \quad 15 & 0 & 1
\end{align*}
\]

*Shading above is not required in an answer.*