Some Review Notes and Practice Questions for Final Exam

This document contains study notes and questions to help you prepare for the final exam. The exam will cover all material we discussed throughout the semester. Here, the emphasis is on more recent material. Please go back to prior sets of review notes for earlier material. In addition, you are expected to review your class and discussion notes, on-line and textbook readings, as well as your homework and programming assignments.

Some of the following questions/topics are to get you to think about the material covered in the course. In some cases, you will need to reason about the material covered in lectures, assignments, and book chapters to determine the answers. Don’t panic if you don’t know the answers to everything.

The exam will be closed book and notes.

Review Points

• Memory Layout and Starting a Program
  We discussed the division of memory in Linux, and how the data can be global/static, dynamic or stack data. The dynamic data grows up and the stack used for function calls grows down.
  Understand what happens to programs as they get assembled, loaded and executed. Know the goals of the assembler and how references (labels) end up encoded in binary knowing where the program and data are loaded in memory.
  What does an object module (i.e., object file) typically contain?

• Optimizing Programs
  Why are compilers often limited in optimizing our programs?
  Describe at least 3-4 machine-independent program optimizations. Understand how you might apply such optimizations.

• Memory
  What is meant by “random access” in the term random access memory?
  What is the primary use of Dynamic RAM versus Static RAM.
  What is the purpose of a hierarchical memory system based on the use of caches?
  Explain what is meant by spatial and temporal locality. Give an example of each in instructions and in data.
  Important factors in cache design are miss ratio, miss penalty, and hit time. What is meant by each of these?
  How can the memory system know whether a particular address referenced by the processor is in the cache, since the memory items stored in the cache are not stored in the relative
order they are in memory, and the cache holds only a small fraction of the items available in memory? How does direct mapped caching work?

What is a virtual address space for a program and how does it differ from physical representation of a program?

What are some benefits of virtual memory?

How can a virtual address be partitioned into a page offset and a virtual page number?

Describe the form and purpose of the page table? How can the page table be used to translate virtual addresses to physical addresses?

In a virtual memory system, virtual addresses are translated to physical addresses at what time? That is, when is a virtual address bound to a physical address?

What’s the purpose of dynamic memory allocation?

What’s the difference between an implicit list implementation of memory blocks and an explicit list? Which one is more efficient and why? What are possible implementations for the explicit list? What’s the purpose of each?

What are possible strategies for allocating free memory blocks? What’s the purpose of each?

What’s the purpose of coalescing memory blocks? Describe different coalescing policies.

• I/O

What does Amdahl’s law state and how is it useful in system design?

What is the difference between memory-mapped I/O and the use of special I/O instructions?

Three methods for managing I/O are polling (programmed I/O), interrupt-driven I/O, and direct memory access. How does each work?

What is an interrupt? What causes interrupts? How are they handled by the processor and the operating system?

What role does the operating system play in input/output?

What is a process?

What is a bus?

The wires (lines) of a bus are partitioned into functional groups. What are they?

Explain the need for bus arbitration. Describe one arbitration method.

Practice Questions

1. Suppose we have a 64KByte cache and a 4-word block (assuming 4 bytes per word). Assume a 32-bit word address. (Show your work.)

   (a) How many entries are there in the cache?

   (b) Using a direct address mapping scheme: How is the address partitioned into tag, cache index, and word offset? Suppose an address is 0000 0000 0000 1111 0011 0101 1100 1000. What cache entry would this be in, if it is in cache?
2. Suppose we have a 16 KB cache and an 8-word block (assuming 4 bytes per word). Assume a 32-bit address is used for the byte-addressable memory.

(a) How many blocks are there in the cache?
(b) Using a direct mapping scheme, how is the 32-bit address partitioned into tag, cache index, word-offset, and byte-offset within the block?

Suppose an address is

```
0000 0000 0000 0011 1100 0000 1111 0100
```

Which cache entry will this block be assigned to?

(c) A cache, in addition to storing words from memory, must include storage for tags and valid and dirty bits. How much extra storage will be needed?

3. For each of the following, choose the right answer.

- In a direct-mapped cache, finding out whether a memory reference is in the cache involves:
  A. searching the entire cache.
  B. examining the tag at one location in the cache.
  C. checking the page table for the reference.
  D. looking at the dirty bit associated with that memory address.

- In an interrupt-driven I/O interface, the processor is interrupted:
  A. at the end of every bus cycle.
  B. when the DMA transfer is complete.
  C. when the I/O device completes the I/O command.
  D. at the end of a polling period.

4. Consider an operating system with 24-bit virtual addresses for byte-accessed memory. Physical addresses are 18 bits long and pages are of size 2 KB. The system uses 1 KB direct-mapped cache. The cache uses 4 words per block. The word size is 4 bytes. Draw a diagram showing the virtual-to-physical mapping through the cache. Make sure to label the width of all fields and lines.

5. Consider we have a list of two free memory blocks of size 2000 and 1000, in that order. Given a series of allocate requests, define a performance metric to be the time or number of requests successfully allocated until / before we fail to satisfy a given allocate request. Compare the performance of First Fit, Best Fit, and Worst Fit, for the following sequence of allocate requests. What conclusion can you draw?

- 800, 300, 800, 950.
- 700, 1500.
- 1100, 300, 800, 650.
6. (i) Consider a sequence of address references given as word addresses:

   1, 4, 9, 5, 20, 17, 19, 5

   Assuming a direct-mapped cache with 4-word blocks that is initially empty and that has a total size of 16 words, label each reference in the list as a hit or a miss and show the final contents of the cache.

<table>
<thead>
<tr>
<th>cache block index</th>
<th>contents from addresses</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

   **Direct Mapped Cache**

   (ii) Assume that instead of direct-mapped translation, we use a so-called *fully associative* cache. Under this scheme, any one of the main memory blocks can be mapped into any one of the cache blocks. If a new block is brought into the cache due to a miss, then it is placed in the first empty cache block. If there is no empty cache block, then the new block replaces the least recently used (LRU). Repeat part (i) for this fully associative cache.

<table>
<thead>
<tr>
<th>cache block index</th>
<th>contents from addresses</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

   **Fully Associative Cache**

7. Consider a byte-addressed memory system which has the following characteristics. Give the size (i.e., width in bits) of the fields labelled (i) to (v) of the page table and addresses shown below. Take $1K=2^{10}$, and 1 byte=8 bits. Each page table entry contains either a disk address or a frame number, which requires the same number of bits. Ignore the “valid” and other management bits. Assume a page fault does not occur.

<table>
<thead>
<tr>
<th>Virtual address</th>
<th>32 bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>Page size</td>
<td>2 K bytes</td>
</tr>
<tr>
<td>Physical address</td>
<td>36 bits</td>
</tr>
</tbody>
</table>
8. Consider a memory-mapped access to a keyboard. The keyboard controller has two registers: a control register whose bit 1 (i.e., the bit at position one, assuming the least-significant bit is at position zero) can be set to 1 to enable interrupts; and a data register which contains the ASCII code of the key which got pressed (and at that time, the keyboard sends an interrupt signal if interrupts were enabled). Assume the control and data registers are accessed using physical memory addresses 0xFFFF0000 and 0xFFFF0004, respectively.

(i) Write low-level Intel-like instruction(s) to enable keyboard interrupt by setting bit 1 to 1 in the keyboard’s control register (and other bits to zero). (Don’t worry about the exact syntax, pseudo-code is fine.)

(ii) Write low-level Intel-like instruction(s) to read the character from the keyboard’s data register at the time of interrupt.

(iii) Which part of the system (software and/or hardware) typically executes these instructions?