Midterm I
October 15th, 2008
CS162: Operating Systems and Systems Programming

General Information:
This is a closed book exam. You are allowed 1 page of hand-written notes (both sides). You have 3 hours to complete as much of the exam as possible. Make sure to read all of the questions first, as some of the questions are substantially more time consuming.

Write all of your answers directly on this paper. Make your answers as concise as possible. On programming questions, we will be looking for performance as well as correctness, so think through your answers carefully. If there is something about the questions that you believe is open to interpretation, please ask us about it!

<table>
<thead>
<tr>
<th>Problem</th>
<th>Possible</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>24</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>12</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>25</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>21</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>18</td>
<td></td>
</tr>
<tr>
<td>Total</td>
<td>100</td>
<td></td>
</tr>
</tbody>
</table>
Problem 1: Short Answer [24pts]

**Problem 1a[2pts]:** Give at least two reasons why the following implementation of a condition variable is incorrect (assume that MySemi is a semaphore initialized to 0):

```c
Wait() { MySemi.P(); }
Signal() { MySemi.V(); }
```

**Problem 1b[4pts]:** What is the difference between Mesa and Hoare scheduling for monitors? Include passing of locks between signaler and signalee, scheduling of CPU resources, and impact on programmer.

**Problem 1c[3pts]:** The SRTF algorithm requires knowledge of the future. Why is that? Name two ways to approximate the information required to implement this algorithm.

**Problem 1d[3pt]:** What is priority donation? What sort of information must the OS track to allow it to perform priority donation?
Problem 1e[3pts]: Above, we show the Readers-Writers example given in class. It used two condition variables, one for waiting readers and one for waiting writers. Suppose that all of the following requests arrive in very short order (while R1 and R2 are still executing):

Incoming stream: R1 R2 W1 R3 R4 R5 W3 R6 W4 R7 R8 W6 R9

In what order would the above code process the above requests? If you have a group of requests that are equivalent (unordered), indicate this clearly by surrounding them with braces ‘{}’. You can assume that the wait queues for condition variables are FIFO in nature (i.e. signal() wakes up the oldest thread on the queue). Explain how you got your answer.
Problem 1f[4pts]:
Suppose that you were to redesign the code in (1e). What is the minimum number of condition variables that we would need in order to handle the above requests in an order that guarantees that a read always returns the results of writes that have arrived before it but not after it? (Another way to say this is that the reads and writes occur in the order in which they arrive, while still allowing groups of reads that arrive together to occur simultaneously.) Provide a two or three sentence sketch of your scheme (do not try to write code!).

Problem 1g[3pts]: What are exceptions? Name two different types of exceptions and give an example of each type:

Problem 1h[2pts]: What was the problem with the Therac-25? Your answer should involve one of the topics of the class.
Problem 2: TRUE/FALSE [12 pts]
In the following, it is important that you EXPLAIN your answer in TWO SENTENCES OR LESS (Answers longer than this may not get credit!). Also, answers without an explanation GET NO CREDIT.

Problem 2a[2pts]: The kernel on a multiprocessor can use the local disabling of interrupts (within one CPU) to produce critical sections between the OSs on different CPUs.

True / False
Explain:

Problem 2b[2pts]: When designing a multithreaded application, you must use synchronization primitives to make sure that the threads do not overwrite each other's registers.

True / False
Explain:

Problem 2c[2pts]: A system that provides segmentation without paging can fragment the physical address space, forcing the operating system to waste physical memory.

True / False
Explain:

Problem 2d[2pts]: A user-level library implements each system call by first executing a “transition to kernel mode” instruction. The library routine then calls an appropriate subroutine in the kernel.

True / False
Explain:

Problem 2e[2pts]: The difference between processes and threads is purely historical.

True / False
Explain:

Problem 2f[2pts]: Round robin scheduling provides a latency improvement over FCFS scheduling for interactive jobs.

True / False
Explain:
Problem 3: Atomic Synchronization Primitives [25 pts]

In class, we discussed a number of atomic hardware primitives that are available on modern architectures. In particular, we discussed “test and set” (TSET), SWAP, and “compare and swap” (CAS). They can be defined as follows (let “expr” be an expression, “&addr” be an address of a memory location, and “M[addr]” be the actual memory location at address addr):

<table>
<thead>
<tr>
<th>Test and Set (TSET)</th>
<th>Atomic Swap (SWAP)</th>
<th>Compare and Swap (CAS)</th>
</tr>
</thead>
<tbody>
<tr>
<td>TSET(&amp;addr) {</td>
<td>SWAP(&amp;addr, expr) {</td>
<td>CAS(&amp;addr, expr1, expr2) {</td>
</tr>
<tr>
<td>int result = M[addr];</td>
<td>int result = M[addr];</td>
<td>if (M[addr] == expr1) {</td>
</tr>
<tr>
<td>M[addr] = 1;</td>
<td>M[addr] = expr;</td>
<td>M[addr] = expr2;</td>
</tr>
<tr>
<td>return (result);</td>
<td>return (result);</td>
<td>return true;</td>
</tr>
<tr>
<td>}</td>
<td>}</td>
<td>}</td>
</tr>
</tbody>
</table>

Both TSET and SWAP return values (from memory), whereas CAS returns either true or false. Note that our &addr notation is similar to a reference in c++, and means that the &addr argument must be something that can be stored into (an “lvalue”). For instance, TSET could be used to implement a spin-lock acquire as follows:

```c
int lock = 0; // lock is free
// Later: acquire lock
while (TSET(lock));
```

CAS is general enough as an atomic operation that it can be used to implement both TSET and SWAP. For instance, consider the following implementation of TSET with CAS:

```c
TSET(&addr) {
   int temp;
   do {
      temp = M[addr];
   } while (!CAS(addr,temp,1));
   return temp;
}
```

Problem 3a[3pts]:
Show how to implement a spinlock acquire with a single while loop using CAS instead of TSET. You must only fill in the arguments to CAS below:

```c
// Initialization
int lock = 0; // Lock is free

// acquire lock
while ( !CAS(           ,          ,         ) );
```
Problem 3b[2pts]:
Show how SWAP can be implemented using CAS. Don’t forget the return value.

```
SWAP(addr, reg1) {

}
```

Problem 3c[3pts]:
With spinlocks, threads spin in a loop (busy waiting) until the lock is freed. In class we argued that spinlocks were a bad idea because they can waste a lot of processor cycles. The alternative is to put a waiting process to sleep while it is waiting for the lock (using a blocking lock). Contrary to what we implied in class, there are cases in which spinlocks would be more efficient than blocking locks. Give a circumstance in which this is true and explain why a spinlock is more efficient.
An object such as a queue is considered “lock-free” if multiple processes can operate on this object simultaneously without requiring the use of locks, busy-waiting, or sleeping. In this problem, we are going to construct a lock-free FIFO queue using the atomic CAS operation. This queue needs both an Enqueue and Dequeue method.

We are going to do this in a slightly different way than normally. Rather than Head and Tail pointers, we are going to have “PrevHead” and Tail pointers. PrevHead will point at the last object returned from the queue. Thus, we can find the head of the queue (for dequeuing). If we don’t have to worry about simultaneous Enqueue or Dequeue operations, the code is straightforward (ignore the null-pointer exception for the Dequeue() operation for now):

```java
// Holding cell for an entry
class QueueEntry {
    QueueEntry next = null;
    Object stored;

    QueueEntry(Object newobject) {
        stored = newobject;
    }
}

// The actual Queue (not yet lock free!)
class Queue {
    QueueEntry prevHead = new QueueEntry(null);
    QueueEntry tail = prevHead;

    void Enqueue(Object newobject) {
        QueueEntry newEntry = new QueueEntry(newobject);
        QueueEntry oldtail = tail;
        tail = newEntry;
        oldtail.next = newEntry;
    }

    Object Dequeue() {
        QueueEntry oldprevHead = prevHead;
        QueueEntry nextEntry = oldprevHead.next;
        prevHead = nextEntry;
        return nextEntry.stored;
    }
}
```

**Problem 3d [3pts]:**
For this non-multithreaded code, draw the state of a queue with 2 queued items on it:
Problem 3e[3pts]:
For each of the following potential context switch points, state whether or not a context switch at that point could cause incorrect behavior of Enqueue(); Explain!

```java
void Enqueue(Object newobject) {
    QueueEntry newEntry = new QueueEntry(newobject);
    QueueEntry oldtail = tail;
    tail = newEntry;
    oldtail.next = newEntry;
}
```

Point 1:

Point 2:

Point 3:

Problem 3f[4pts]:
Rewrite code for Enqueue(...), using the CAS() operation, such that it will work for any number of simultaneous Enqueue and Dequeue operations. You should never need to busy wait. **Do not use locking (i.e. don’t use a test-and-set lock).** The solution is tricky but can be done in a few lines. We will be grading on conciseness. Do not use more than one CAS() or more than 10 lines total (including the function declaration at the beginning). **Hint: wrap a do-while around vulnerable parts of the code identified above.**

```java
void Enqueue(Object newobject) {
    QueueEntry newEntry = new QueueEntry(newobject);

    // Insert code here
```
Problem 3g[3pts]:
For each of the following potential context switch points, state whether or not a context switch at that point could cause incorrect behavior of Dequeue(); Explain!

Object Dequeue() {
    QueueEntry oldprevHead = prevHead;
    QueueEntry nextEntry = oldprevHead.next;
    prevHead = nextEntry;
    return nextEntry.stored;
}

Point 1:

Point 2:

Point 3:

Problem 3h[4pts]:
Rewrite code for Dequeue(), using the CAS() operation, such that it will work for any number of simultaneous Enqueue and Dequeue operations. You should never need to busy wait. Do not use locking (i.e. don’t use a test-and-set lock). The solution can be done in a few lines. We will be grading on conciseness. Do not use more than one CAS() or more than 10 lines total (including the function declaration at the beginning). Hint: wrap a do-while around vulnerable parts of the code identified above.

Object Dequeue() {
    // Insert code here
}
Problem 4: Deadlock[21 pts]

Problem 4a[5pts]:  
The figure at the right illustrates a 2D mesh of network routers. Each router is connected to each of its neighbors by two network links (small arrows), one in each direction. Messages are routed from a source router to a destination router and can stretch through the network (i.e. consume links along the route from source to destination). Messages can cross inside routers. 

Assume that no network link can service more than one message at a time, and that each message must consume a continuous set of channels (like a snake). Messages always make progress to the destination and never wrap back on themselves. The figure shows two messages (thick arrows).

Assume that each router or link has a very small amount of buffer space and that each message can be arbitrarily long. Show a situation (with a drawing) in which messages are deadlocked and can make no further progress. Explain how each of the four conditions of deadlock are satisfied by your example. *Hint: Links are the limited resources in this example.*

Problem 4b[3pts]:  
Define a routing policy that avoids deadlocks in the network of (4a). Name one of the four conditions that is no longer possible, given your routing policy. Explain.

Problem 4c[3pts]:  
Suppose that each router node contains sufficient queue space to hold complete messages (assume infinite space, if you like). Why is it impossible for deadlocks such as in (4a) to occur? Name one of the four conditions that is no longer possible, given infinite queue space in the router. Explain.
**Problem 4d[4pts]:**
Suppose that we have the following resources: A, B, C and threads T1, T2, T3, T4. The total number of each resource is:

<table>
<thead>
<tr>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
</tr>
<tr>
<td>B</td>
</tr>
<tr>
<td>C</td>
</tr>
<tr>
<td>12</td>
</tr>
<tr>
<td>9</td>
</tr>
<tr>
<td>12</td>
</tr>
</tbody>
</table>

Further, assume that the processes have the following maximum requirements and current allocations:

<table>
<thead>
<tr>
<th>Thread ID</th>
<th>Current Allocation</th>
<th>Maximum</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>A</td>
<td>B</td>
</tr>
<tr>
<td>T1</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>T2</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>T3</td>
<td>5</td>
<td>4</td>
</tr>
<tr>
<td>T4</td>
<td>2</td>
<td>1</td>
</tr>
</tbody>
</table>

Is the system in a safe state? If “yes”, show a non-blocking sequence of thread executions. Otherwise, provide a proof that the system is unsafe. Show all steps, intermediate matrices, etc.
Problem 4e[3pts]:
Assume that we start with a system in the state of (4d). Suppose that T1 asks for 2 more copies of resource A. Can the system grant this if it wants to avoid deadlock? Explain.

Problem 4f[3pts]:
Assume that we start with a system in the state of (4d). What is the maximum number of additional copies of resources (A, B, and C) that T1 can be granted in a single request without risking deadlock? Explain.
Problem 5: Address Translation [18 pts]

Problem 5a[3pts]:
Suppose we have a 32-bit processor (with 32-bit virtual addresses) and 8 KB pages. Assume that it can address up to 2 TB (terabytes) of DRAM; 1TB = 1024 GB = (1024)^2 MB. Assume that we need 4 permissions bits in each page table entry (PTE), namely Valid (V), Writable (W), Accessed (A), and Dirty (D). Show the format of a PTE, assuming that each page should be able to hold an integer number of PTEs. If you have extra bits in the PTE, you can mark them as “unused”. Explain.

Problem 5b[5pts]:
Assume that we wish to build a two-level page table for the processor from (5a) in which each piece of the page table consumes exactly a page (no more, no less). We may end up wasting space as a result. Draw and label a figure showing how a virtual address gets mapped into a physical address. Show the format of the page table (complete with access checks), the virtual address, and physical address. Minimize pieces of the page table that consume less than a page (and thus waste space).
Problem 5c[3pts]:
Consider a multi-level memory management scheme using the following format for virtual addresses:

<table>
<thead>
<tr>
<th>Virtual seg # (2 bits)</th>
<th>Virtual Page # (6 bits)</th>
<th>Offset (12 bits)</th>
</tr>
</thead>
</table>

Virtual addresses are translated into physical addresses of the following form:

<table>
<thead>
<tr>
<th>Physical Page # (8 bits)</th>
<th>Offset (12 bits)</th>
</tr>
</thead>
</table>

Page table entries (PTE) are 16 bits in the following format, stored in big-endian form in memory (i.e. the MSB is first byte in memory):

<table>
<thead>
<tr>
<th>Physical Page # (8 bits)</th>
<th>Kernel</th>
<th>Nocache</th>
<th>0</th>
<th>0</th>
<th>Dirty</th>
<th>Use</th>
<th>Writeable</th>
<th>Valid</th>
</tr>
</thead>
</table>

1) How big is a page? Explain.

2) What is the maximum amount of virtual memory supported by this scheme? Explain.

3) What is the maximum amount of physical memory supported by this scheme? Explain.

Problem 5d[7pts]: Assume the memory translation scheme from (5c). Use the Segment Table and Physical Memory table given on the next page to predict what will happen with the following load/store instructions. Addresses are virtual. The return value for a load is an 8-bit data value or an error, while the return value for a store is either "ok" or an error. If there is an error, make sure to say which error. Possibilities are: "bad segment" (invalid segment), "segment overflow" (address outside range of segment), or "access violation" (page invalid, or attempt to write a read only page). A few answers are given:

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Result</th>
<th>Instruction</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>Load [0xC1015]</td>
<td>0x57</td>
<td>Store [0x52002]</td>
<td></td>
</tr>
<tr>
<td>Store [0x43045]</td>
<td>ok</td>
<td>Load [0x04013]</td>
<td></td>
</tr>
<tr>
<td>Store [0xC1016]</td>
<td>Access violation</td>
<td>Store [0x81015]</td>
<td></td>
</tr>
<tr>
<td>Load [0xD2002]</td>
<td></td>
<td>Store [0x03010]</td>
<td></td>
</tr>
<tr>
<td>Store [0xD2031]</td>
<td></td>
<td>Load [0x13035]</td>
<td></td>
</tr>
</tbody>
</table>
### Segment Table (Max Segment=3)

<table>
<thead>
<tr>
<th>Seg #</th>
<th>Page Table Base</th>
<th>Max Page Entries</th>
<th>Segment State</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0x02030</td>
<td>0x20</td>
<td>Valid</td>
</tr>
<tr>
<td>1</td>
<td>0x01020</td>
<td>0x10</td>
<td>Valid</td>
</tr>
<tr>
<td>2</td>
<td>0x01040</td>
<td>0x40</td>
<td>Invalid</td>
</tr>
<tr>
<td>3</td>
<td>0x04000</td>
<td>0x20</td>
<td>Valid</td>
</tr>
</tbody>
</table>

### Physical Memory

<table>
<thead>
<tr>
<th>Address</th>
<th>+0</th>
<th>+1</th>
<th>+2</th>
<th>+3</th>
<th>+4</th>
<th>+5</th>
<th>+6</th>
<th>+7</th>
<th>+8</th>
<th>+9</th>
<th>+A</th>
<th>+B</th>
<th>+C</th>
<th>+D</th>
<th>+E</th>
<th>+F</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00000</td>
<td>0E</td>
<td>0F</td>
<td>10</td>
<td>11</td>
<td>12</td>
<td>13</td>
<td>14</td>
<td>15</td>
<td>16</td>
<td>17</td>
<td>18</td>
<td>19</td>
<td>1A</td>
<td>1B</td>
<td>1C</td>
<td>1D</td>
</tr>
<tr>
<td>0x00010</td>
<td>1E</td>
<td>1F</td>
<td>20</td>
<td>21</td>
<td>22</td>
<td>23</td>
<td>24</td>
<td>25</td>
<td>26</td>
<td>27</td>
<td>28</td>
<td>29</td>
<td>2A</td>
<td>2B</td>
<td>2C</td>
<td>2D</td>
</tr>
<tr>
<td>0x01010</td>
<td>40</td>
<td>41</td>
<td>42</td>
<td>43</td>
<td>44</td>
<td>45</td>
<td>46</td>
<td>47</td>
<td>48</td>
<td>49</td>
<td>4A</td>
<td>4B</td>
<td>4C</td>
<td>4D</td>
<td>4E</td>
<td>4F</td>
</tr>
<tr>
<td>0x01020</td>
<td>40</td>
<td>03</td>
<td>41</td>
<td>01</td>
<td>30</td>
<td>01</td>
<td>31</td>
<td>03</td>
<td>00</td>
<td>03</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
</tr>
<tr>
<td>0x01030</td>
<td>00</td>
<td>11</td>
<td>22</td>
<td>33</td>
<td>44</td>
<td>55</td>
<td>66</td>
<td>77</td>
<td>88</td>
<td>99</td>
<td>AA</td>
<td>BB</td>
<td>CC</td>
<td>DD</td>
<td>EE</td>
<td>FF</td>
</tr>
<tr>
<td>0x01040</td>
<td>10</td>
<td>01</td>
<td>11</td>
<td>03</td>
<td>31</td>
<td>03</td>
<td>13</td>
<td>00</td>
<td>14</td>
<td>01</td>
<td>15</td>
<td>03</td>
<td>16</td>
<td>01</td>
<td>17</td>
<td>00</td>
</tr>
<tr>
<td>0x02030</td>
<td>10</td>
<td>01</td>
<td>11</td>
<td>00</td>
<td>12</td>
<td>03</td>
<td>67</td>
<td>03</td>
<td>11</td>
<td>03</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
</tr>
<tr>
<td>0x02040</td>
<td>02</td>
<td>20</td>
<td>03</td>
<td>30</td>
<td>04</td>
<td>40</td>
<td>05</td>
<td>50</td>
<td>01</td>
<td>60</td>
<td>03</td>
<td>70</td>
<td>08</td>
<td>80</td>
<td>09</td>
<td>90</td>
</tr>
<tr>
<td>0x02050</td>
<td>10</td>
<td>00</td>
<td>31</td>
<td>01</td>
<td>10</td>
<td>03</td>
<td>31</td>
<td>01</td>
<td>12</td>
<td>03</td>
<td>30</td>
<td>00</td>
<td>10</td>
<td>00</td>
<td>10</td>
<td>01</td>
</tr>
<tr>
<td>0x04000</td>
<td>30</td>
<td>00</td>
<td>31</td>
<td>01</td>
<td>11</td>
<td>01</td>
<td>33</td>
<td>03</td>
<td>34</td>
<td>01</td>
<td>35</td>
<td>00</td>
<td>43</td>
<td>38</td>
<td>32</td>
<td>79</td>
</tr>
<tr>
<td>0x04010</td>
<td>50</td>
<td>28</td>
<td>84</td>
<td>19</td>
<td>71</td>
<td>69</td>
<td>39</td>
<td>93</td>
<td>75</td>
<td>10</td>
<td>58</td>
<td>20</td>
<td>97</td>
<td>49</td>
<td>44</td>
<td>59</td>
</tr>
<tr>
<td>0x04020</td>
<td>23</td>
<td>03</td>
<td>20</td>
<td>03</td>
<td>00</td>
<td>01</td>
<td>62</td>
<td>08</td>
<td>99</td>
<td>86</td>
<td>28</td>
<td>03</td>
<td>48</td>
<td>25</td>
<td>34</td>
<td>21</td>
</tr>
<tr>
<td>0x10000</td>
<td>AA</td>
<td>55</td>
<td>AA</td>
<td>55</td>
<td>AA</td>
<td>55</td>
<td>AA</td>
<td>55</td>
<td>AA</td>
<td>55</td>
<td>AA</td>
<td>55</td>
<td>AA</td>
<td>55</td>
<td>AA</td>
<td>55</td>
</tr>
<tr>
<td>0x10010</td>
<td>A5</td>
<td>5A</td>
<td>A5</td>
<td>5A</td>
<td>A5</td>
<td>A5</td>
<td>A5</td>
<td>A5</td>
<td>A5</td>
<td>A5</td>
<td>A5</td>
<td>A5</td>
<td>A5</td>
<td>A5</td>
<td>A5</td>
<td>5A</td>
</tr>
<tr>
<td>0x11000</td>
<td>00</td>
<td>11</td>
<td>22</td>
<td>33</td>
<td>44</td>
<td>55</td>
<td>66</td>
<td>77</td>
<td>88</td>
<td>99</td>
<td>AA</td>
<td>BB</td>
<td>CC</td>
<td>DD</td>
<td>EE</td>
<td>FF</td>
</tr>
<tr>
<td>0x11010</td>
<td>11</td>
<td>22</td>
<td>33</td>
<td>44</td>
<td>55</td>
<td>66</td>
<td>77</td>
<td>88</td>
<td>99</td>
<td>AA</td>
<td>BB</td>
<td>CC</td>
<td>DD</td>
<td>EE</td>
<td>FF</td>
<td>00</td>
</tr>
<tr>
<td>0x11020</td>
<td>22</td>
<td>33</td>
<td>44</td>
<td>55</td>
<td>66</td>
<td>77</td>
<td>88</td>
<td>99</td>
<td>AA</td>
<td>BB</td>
<td>CC</td>
<td>DD</td>
<td>EE</td>
<td>FF</td>
<td>00</td>
<td>11</td>
</tr>
<tr>
<td>0x31000</td>
<td>01</td>
<td>12</td>
<td>23</td>
<td>34</td>
<td>45</td>
<td>56</td>
<td>67</td>
<td>78</td>
<td>89</td>
<td>9A</td>
<td>AB</td>
<td>BC</td>
<td>CD</td>
<td>DE</td>
<td>EF</td>
<td>00</td>
</tr>
<tr>
<td>0x31010</td>
<td>02</td>
<td>13</td>
<td>24</td>
<td>35</td>
<td>46</td>
<td>57</td>
<td>68</td>
<td>79</td>
<td>8A</td>
<td>9B</td>
<td>AC</td>
<td>BD</td>
<td>CE</td>
<td>DF</td>
<td>F0</td>
<td>01</td>
</tr>
<tr>
<td>0x31020</td>
<td>03</td>
<td>01</td>
<td>25</td>
<td>36</td>
<td>47</td>
<td>58</td>
<td>69</td>
<td>7A</td>
<td>8B</td>
<td>9C</td>
<td>AD</td>
<td>BE</td>
<td>CF</td>
<td>E0</td>
<td>F1</td>
<td>02</td>
</tr>
<tr>
<td>0x31030</td>
<td>04</td>
<td>15</td>
<td>26</td>
<td>37</td>
<td>48</td>
<td>59</td>
<td>70</td>
<td>7B</td>
<td>8C</td>
<td>9D</td>
<td>AE</td>
<td>BF</td>
<td>D0</td>
<td>E1</td>
<td>F2</td>
<td>03</td>
</tr>
</tbody>
</table>
[ This page intentionally left blank ]