CS-537: Midterm Exam (Fall 2010)
Memory Gone Bad

Please Read All Questions Carefully!

There are 13 total numbered pages.

Please put your NAME and student ID on THIS page, and JUST YOUR student ID (but NOT YOUR NAME) on every other page. Why are you doing this? So I can grade the exam without knowing who you are, of course.

Name and Student ID: ____

Student ID: ______________

Grading Page

	Points	Total Possible
Q1		20
Q2		20
Q3		20
Q4		20
Q5		20
Q6		20
Total		120

You are about to take an exam in which some memory has gone bad. Don't worry, it's not your memory I am talking about (well, at least, I hope not!). Rather, it's the computer's memory, in a new PC from **Iron Systems**.

As it turns out, the Iron Systems PC has cheap, unreliable memory, in which a bit occasionally gets flipped. A **bit flip** takes one single bit in the system memory and changes it to the opposite of its current value, thus turning a 0 into a 1, or a 1 into a 0. As you can imagine, this makes the OS of the system, which stores a lot of its important data in memory, not behave correctly (sometimes).

Bit flips are challenging to handle because they can change the value of a variable that the OS (or a user program) is using. For example, imagine a simple integer, set to the value 1. In memory, this integer might look like this:

```
31 30 29 28 27 26 .... 04 03 02 01 00
 0  0  0  0  0  0       0  0  0  0  1
```

In the diagram, the integer is laid out in memory with the high-order (31st) bit on the left and the low-order (0th) bit on the right. A single bit flip, of say the 30th bit, can dramatically change the value of the integer:

```
31 30 29 28 27 26 .... 04 03 02 01 00
 0  1  0  0  0  0       0  0  0  0  1
```

Now the integer is set to $2^{30} + 1$, instead of plain old 1! Quite a change in value, from just one bit getting flipped.

All questions will center around **Iron OS**, a new operating system that I just made up for the purposes of this exam. Most questions will describe how Iron OS works, and then ask you to determine what happens when memory goes bad, or something like that.

Good luck! Trust in your memory; it will serve you well, or at least better than the Iron PC's memory is serving poor old Iron OS.

1. **Iron Scheduling.**

   The Iron OS has a scheduler that tries to emulate a multi-level feedback queue (MLFQ) as we saw in class. Iron OS keeps track of important MLFQ information in a simply array of structures in memory (`allpri`):

```
struct priority {
  unsigned int priority; // current priority of job
  unsigned int timeLeft; // time left at this priority (in clock ticks)
};

// all priorities stored in this array
struct priority      allpri[MAX_PROCS];
```

   The `priority` variable tracks the current priority level of each job; in the Iron MLFQ scheduler, there are five levels, and thus priority can be set to 4 (highest priority) through 0 (lowest).

   The `timeLeft` variable tracks how much time (in clock ticks) a process has at a given priority. That is, each time the timer interrupt goes off, timeLeft (for the currently running job) is decremented by 1; when it reaches 0, the job's priority level is decreased (as per MLFQ rules).

   (a) Before getting into the corruption scenario, describe how MLFQ works. What are its major goals as a scheduling policy?

   Now let's consider corruption. It turns out that corruption can affect both the `priority` field as well the `timeLeft` field.

   (b) What do you think will happen if a bit gets flipped and sets the priority **too high** (i.e., higher than it is supposed to be)?

(c) What do you think will happen if a bit gets flipped and sets the priority **too low**?

Here is the code that deals with the priority. In fact, all this code does is get the priority for the process ID (pid) of interest; later (not shown), the priority is put it into a list of all priorities, which is then sorted, in order to determine which process runs next.

```
int getPriority(int pid) {
    return allpri[pid].priority;
}
```

(d) Assuming the priority value was changed in memory, how could you rewrite `getPriority()` so that it returns a valid priority? (if not the correct one)

(e) Discuss the limits of such an approach. Does your new routine solve the memory corruption problem?

Finally, let's think about what happens if a bit in the `timeLeft` field gets flipped.

(f) What if the timeLeft field gets flipped to a **much higher number**?

(g) What if the timeLeft field gets flipped to a **lower number**?

(h) Which is worse for the system? Why?

2. **Iron Segmentation.**

   An early version of the Iron PC uses **segmentation** in order to place segments of a virtual address space in memory. Unfortunately, a single bit in one of the base or limit registers has been flipped! Your job is to figure out which one, in each of the following examples.

   Assume the following about the segmentation system. The address space is a tiny 1 KB in size; physical memory is 16 KB. The top bit of a virtual address determines which of two segments a reference is in; segment 0 grows in the positive direction, whereas segment 1 grows in the negative direction. Hmm, does this seem familiar?

   In this first bit flip scenario, we see the following:

```
Segment 0 base  (grows positive) : 0x8400
Segment 0 limit                  : 0x100 (note: this is in hex)
Segment 1 base  (grows negative) : 0x0c00
Segment 1 limit                  : 0x000
```

   We also know that **virtual address 0x095** should translate to **physical address 0x0495**.

   (a) Given that we know how VA 0x095 should be translated, which segment base register has the wrong value? (describe why you think so)

   (b) What should the correct value be?

   We also find out that the **virtual address 767 (decimal) should cause a segmentation violation**, but **768 (decimal) should not** (i.e., 768 is a valid virtual address).

   (c) Which segmentation register (seg 0 base, seg 0 limit, seg 1 base, seg 1 limit) determines whether these two virtual addresses (767, 768) are valid? (describe your reasoning)

   (d) What should the correct value of that register be?

   (e) Given the correct value, what physical address should 768 translate to?

3. **Iron Multi-level Paging.**

   At some point, some Iron OS mastermind decided to change the Iron VM system to use **multi-level paging** instead of that simple segmentation system. Assume a **15-bit virtual address** (i.e., a 32 KB AS), a **page size of 32 bytes**, and **4KB of physical memory** (i.e., there are 12 bits in the physical address). Assume further that we are using a simple **two-level page table**. A **page-directory base register** points to the physical frame where the page directory resides. The format of each page directory entry (PDE) or page-table entry is pretty similar: a single valid bit followed by 7 bits of a page frame number. Hmm, where have you seen these assumptions before? Assuming this setup, you are given access to the following memory dump. The dump shows the contents of each physical page, from byte 0 of the page on the left to byte 31 on the right.

```
page   0: 1a 16 1a 10 17 09 06 11 16 1e 12 0c 07 10 1a 0c 15 06 1d 17 10 00 12 16 18 1c 00 17 0d 08 1e 02
page   1: 0c 08 14 15 18 1c 14 1b 01 16 00 10 08 04 1e 1d 09 03 1a 1d 0c 17 1d 08 0a 0b 05 0d 17 1d 03 13
page   2: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
page   3: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
page   4: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
page   5: 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f b1 7f 7f 7f 7f 7f 7f
page   6: 7f 7f 7f bb 7f 7f 7f e1 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f fa 7f 7f 7f 7f 7f 7f 7f 7f 7f da ea 7f
page   7: 17 00 17 07 1e 0f 1e 09 1d 09 02 0f 0d 0b 03 1b 06 0d 0c 01 14 06 0a 10 0d 0f 1e 0f 1d 1a 13 03
page   8: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
page   9: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
page  10: 15 02 0b 1d 13 00 08 15 0a 0f 18 11 18 12 18 08 15 12 0e 17 0f 0f 1b 19 17 11 05 04 09 11 1a 11
page  11: 0f 05 15 0d 05 1b 0c 08 16 1c 11 16 02 04 0f 15 09 07 08 02 0e 14 13 0a 0d 04 09 0e 17 16 1c 01
page  12: 0e 10 1e 04 14 0b 0f 06 14 07 0e 01 1e 0f 0e 16 0c 1b 00 19 0e 19 1d 1e 05 15 03 04 02 09 00 1a
page  13: 0c 1b 16 0f 14 11 17 1a 0f 1b 06 01 18 0a 0d 02 0d 02 03 0b 12 07 0c 07 07 07 0b 10 0c 19 11 14
page  14: 19 05 15 03 0c 09 1e 01 1b 10 02 1e 01 0d 02 16 03 06 16 0a 1c 0a 16 01 0e 00 0a 09 16 0d 15 01
page  15: 7f 7f 7f 7f 7f 7f 7f 7f 7f ff 7f 7f e4 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f d9 7f 7f
page  16: 7f 7f 7f 7f aa 7f 7f 96 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f
page  17: 16 18 0d 0a 0c 00 15 0a 1a 0c 17 14 03 17 05 00 14 09 1e 00 09 04 15 12 1e 1a 00 1b 19 1b 0c 16
page  18: 16 12 08 1a 01 13 0f 19 03 1a 0a 0f 06 02 0d 05 0d 05 02 0c 0c 0a 03 15 19 18 0c 05 02 07 0f 0a
page  19: 11 01 15 11 13 03 09 05 1e 18 01 12 19 16 05 1a 18 17 08 11 11 15 17 0f 0f 1e 14 04 01 0c 07 16
page  20: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
page  21: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
page  22: 0c 0c 1c 14 15 02 1c 15 08 1a 14 11 15 1c 12 09 1a 06 09 16 0b 12 06 0a 1b 06 0a 1a 18 13 10 05
page  23: 7f b7 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f
page  24: 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f c6 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f a4 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f
page  25: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
page  26: 17 15 02 09 0c 0f 0f 0e 08 17 01 11 1c 06 0e 1d 0c 15 15 0a 12 10 0c 1a 0c 1a 12 0a 1a 0b 1e 03
page  27: 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f db 7f 7f 7f 7f 7f 7f ee 7f 7f 7f 7f
page  28: 18 12 00 00 07 1b 19 1b 00 1d 04 0c 17 06 02 06 06 0b 1c 15 02 01 08 08 06 0f 18 17 01 1d 19 0b
page  29: 1b 1c 07 02 0a 13 0a 18 1b 12 00 04 03 1d 01 0d 02 1b 13 0b 17 08 0f 15 14 1e 1a 1a 17 01 02 06
page  30: 1e 1e 09 19 00 04 05 05 0e 07 1e 16 0c 17 03 14 01 1a 06 1a 18 18 05 09 19 06 0e 05 17 08 0e 00
page  31: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
page  32: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
page  33: 7f 7f 7f 7f 87 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f c8 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f
page  34: 7f 7f 7f 7f 7f 7f 7f f8 7f 7f eb 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f
page  35: 08 07 1e 06 10 0f 16 01 1e 0d 1a 05 09 19 1d 10 05 18 10 06 07 01 05 0b 15 0f 10 1c 0c 18 0c 1e
page  36: 05 11 0c 0d 06 14 0e 1e 14 12 0c 0f 14 0e 1d 11 07 14 1a 1d 01 18 00 1b 15 0b 0a 01 06 1a 00 0d
page  37: 1d 1a 03 0e 0c 1b 1a 00 1e 1c 18 15 0e 0b 09 18 03 00 0f 04 0e 0f 1b 1a 0d 18 00 0a 07 0f 1b 1e
page  38: 7f 7f 7f bf 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f fb 7f 7f 7f 7f 7f b3 7f 7f 7f 7f fd 7f 7f 7f
page  39: 7f 7f 9e 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 8a 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f cc 7f f4 7f
page  40: 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f a5 7f 7f 7f 7f 7f 7f 7f 7f 7f 91 7f 7f 7f 7f 7f 7f 7f 7f 7f cf
page  41: 14 07 1d 07 0e 02 05 11 01 0e 01 1e 0e 0c 02 14 1b 02 1d 08 11 0d 11 17 1e 13 14 03 00 09 18 0b
page  42: 0e 03 09 09 17 1c 05 1c 0f 0d 01 16 17 14 19 17 0f 06 15 18 17 04 02 1d 14 08 01 1a 04 1c 15 03
page  43: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
page  44: 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 9a 7f 7f 7f 7f 7f 7f 7f 7f 7f e6 7f 7f 7f 7f 7f 7f 7f 7f
page  45: 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 8c 7f b9 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f
page  46: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
page  47: 1d 17 10 19 09 05 1b 1b 1a 0c 1a 0f 1e 1b 18 03 0a 06 0a 07 0f 0f 11 05 1e 11 0f 05 06 1a 17 19
page  48: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
page  49: 02 19 1e 1a 19 05 0f 11 08 0c 04 0a 19 1d 1e 0b 12 04 18 06 01 13 07 1b 03 08 11 09 1a 13 04 12
page  50: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
page  51: 04 0d 16 02 0e 0c 1c 04 1a 11 0f 1b 0e 18 00 16 1b 07 11 02 12 0a 08 1d 09 03 0c 0e 03 0c 08 16
page  52: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
page  53: 0a 0e 19 15 05 1c 11 18 02 07 1a 12 16 1c 0a 14 12 12 0b 11 19 11 16 07 0b 01 04 11 1c 07 0e 1e
page  54: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
page  55: 19 0d 07 02 04 06 1d 16 0d 1d 02 1e 0d 0c 1b 0a 0f 06 17 11 0c 1c 08 18 12 13 11 0c 00 07 0f 09
page  56: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
page  57: 0a 0e 18 1d 1e 13 0f 0a 00 02 00 1b 07 0e 17 02 13 06 1c 1a 0c 11 1e 05 03 1c 0a 17 1c 0e 14 1e
page  58: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
page  59: 19 00 14 08 1b 07 1d 06 1b 13 13 00 12 04 0e 04 12 1c 15 19 04 1b 1e 1b 14 19 18 00 0e 06 1c 0a
page  60: 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 9c 7f 7f 7f 7f 7f 7f 7f 7f
page  61: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
page  62: 00 15 0d 0e 0d 13 11 05 09 16 15 18 1c 08 10 0b 0f 06 03 03 1e 05 11 17 1e 16 1a 08 0d 11 00 10
page  63: 0b 02 0e 1e 18 1a 1a 13 0d 0f 10 04 03 08 11 03 18 0e 0f 0c 02 19 11 0e 01 0d 0d 11 12 1b 07 07
page  64: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
page  65: 19 06 10 06 01 05 0e 16 0b 0a 1c 02 18 01 1e 0d 02 09 00 08 06 1b 16 07 0a 13 18 14 18 04 0e 18
page  66: 7f 7f 7f 7f 7f 7f 7f 7f 7f 92 7f 7f 7f 7f 7f af 7f 7f 7f 7f 7f 7f 7f ec 7f 7f 7f a9 7f 7f 7f 7f
page  67: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
page  68: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
page  69: 0c 0e 11 17 04 01 1e 17 12 01 03 14 0d 09 1c 04 0b 05 14 1c 13 0e 0f 0c 07 18 1a 17 18 1e 0a 0c
page  70: 09 09 14 07 13 1b 1a 09 0e 0f 08 0a 1e 00 04 14 02 09 18 1c 0b 06 1b 13 0f 0a 0a 09 17 0e 06 1b
page  71: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
page  72: 01 10 16 10 11 18 11 07 0d 0e 00 0f 0e 19 03 13 1b 05 02 0e 0a 08 11 19 18 17 13 1a 1a 16 0a 0a
page  73: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
page  74: 04 10 1c 04 1e 13 16 15 17 06 12 07 03 09 1c 1d 0f 13 05 08 14 08 17 19 0d 0c 05 07 19 08 02 16
page  75: 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 80 7f 7f 7f df 7f c5 7f 7f 7f 7f 7f f9 7f 7f 93 7f 7f 7f 7f 7f 8d
page  76: 04 06 0e 06 1a 16 15 15 0e 17 03 1b 1a 10 1e 06 05 10 1c 19 1d 18 02 02 19 01 0a 17 00 11 13 18
page  77: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
page  78: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
page  79: 04 01 0a 0b 14 11 13 13 13 1e 0e 01 05 06 16 08 1e 17 19 0d 11 12 1a 08 12 13 1e 15 19 18 0b 16
page  80: 0b 16 11 00 14 0c 12 1d 08 01 1c 11 0b 17 02 06 01 02 0c 19 14 1e 04 17 03 14 0e 03 07 14 07 0b
page  81: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
page  82: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
page  83: 7f 7f 7f 7f f0 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f
page  84: 02 0d 0f 00 0f 1c 04 0b 06 10 16 14 04 16 13 1d 13 02 15 0a 01 18 11 0d 11 0e 18 1e 0a 16 1c 0b
page  85: 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f be 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f
page  86: 1d 14 0f 0a 16 00 1e 04 0d 00 0e 09 03 15 1b 00 06 0d 05 1b 11 0e 18 0a 16 0a 0b 0c 10 07 08 00
page  87: 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f d4 7f 7f 7f 7f 7f 7f 7f 7f 7f
page  88: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
page  89: 0e 08 06 07 1b 10 07 19 12 1b 0e 0f 1d 0d 00 02 05 1d 0b 12 17 13 18 02 00 0b 02 07 17 0b 17 03
page  90: 15 04 0a 11 19 1c 10 1e 09 02 16 02 1b 10 0d 14 01 0e 1b 04 0e 16 07 02 04 08 0f 1c 0b 10 18 12
page  91: 12 13 07 04 17 10 0d 0e 18 19 0c 17 00 1b 00 1e 1e 12 1b 14 02 15 1e 16 06 0d 1a 18 06 19 0a 00
page  92: 7f 7f 7f 7f 8b 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f d0 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f
page  93: 12 04 12 0f 1c 00 00 1c 17 1a 0a 09 00 12 1d 1a 12 0e 07 06 04 1c 1b 1c 1b 02 14 11 1a 12 0d 08
page  94: 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f ca 7f 7f 7f
page  95: 18 1d 1b 14 10 02 1b 16 0b 07 0b 0f 19 0b 04 10 0a 17 1c 09 09 01 06 1d 02 1c 08 1e 0d 15 1e 11
page  96: 7f 7f 7f 7f 7f 7f 7f 7f 7f c1 7f 7f 7f 7f 7f 7f 7f 81 7f 7f 7f 7f 7f 7f ed 7f 7f 7f 7f 7f 7f 7f
page  97: 1e 02 09 06 0e 03 00 0a 06 1a 0d 17 11 1a 02 00 15 1c 0a 0a 11 0f 17 01 04 08 07 0c 10 18 07 1d
page  98: 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 9d 7f 7f 7f
page  99: 90 86 f3 e5 a6 d5 a7 85 cb 98 97 7f e2 d3 d7 9b 7f dc c2 bc a2 ad e8 f1 f7 e0 a1 ac a8 de 8f f5
page 100: 0f 19 02 10 02 0f 1d 02 14 0e 18 10 14 16 09 06 12 11 1e 13 0c 00 18 13 11 1a 00 06 13 0b 1c 1b
page 101: 7f 8e 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f
page 102: 11 0c 0b 08 0f 04 05 04 00 14 07 00 01 07 15 06 13 0b 0e 1b 07 09 11 1c 10 07 15 17 03 16 0d 01
page 103: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
page 104: 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f fe 7f 7f 7f 7f 7f
page 105: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
page 106: 09 0d 0a 02 08 11 03 03 03 07 1c 17 1d 13 1e 1c 1a 1a 0c 09 0d 04 03 06 1b 05 14 10 1c 0d 0a 10
page 107: 16 0f 10 1a 0d 14 05 14 02 1d 15 13 17 0c 0e 09 15 19 1e 15 0e 05 0e 13 19 13 12 19 1e 06 1d 0d
page 108: 0a 15 0d 0f 15 0d 09 04 0b 08 17 16 12 0b 14 08 16 13 01 1e 1b 1c 0b 07 19 11 1a 0f 14 10 06 19
page 109: 04 0e 0d 0f 15 15 15 1d 18 17 0d 19 1d 15 02 14 0f 06 0b 18 14 0c 15 16 0d 1b 0d 05 1a 17 16 18
page 110: 17 04 0e 0b 1b 12 12 1d 0d 06 17 1e 03 11 17 13 0f 15 1d 18 1c 1d 02 1a 1e 18 06 05 13 14 12 13
page 111: 07 1b 17 0c 0a 11 12 05 0f 1d 0e 0c 1c 1e 1d 01 1a 06 0a 1b 03 04 08 06 0a 16 13 04 17 1c 12 05
page 112: 00 1b 08 0f 05 00 19 00 10 14 0c 08 1e 05 01 1c 00 1b 10 1d 00 08 1b 17 0d 01 1e 15 00 12 05 05
page 113: 7f 7f 7f 7f 7f 7f 7f 7f 7f fc dd 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f
page 114: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
page 115: 7f 7f 7f 7f 7f d6 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f
page 116: 0c 0a 15 1d 1a 0f 0c 14 06 1e 08 06 10 11 03 06 04 03 0b 04 0f 1c 14 0c 04 07 09 0c 02 11 1d 0a
page 117: 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f b5 7f 7f 7f 7f 7f 7f 7f 7f 7f
page 118: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
page 119: 7f 7f 7f 7f 7f ef 7f 7f 7f 7f 7f 7f 7f a3 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f
page 120: 18 17 0a 0a 1c 09 0f 17 04 0f 07 1b 1b 06 01 11 14 13 0b 02 16 05 03 07 1a 13 0e 02 1b 0c 03 05
page 121: 17 02 11 0f 1c 10 1b 00 0f 18 00 17 14 18 0e 04 01 0f 19 0c 01 17 19 00 0f 1a 03 0a 1b 03 0b 0c
page 122: 16 08 0e 0e 13 08 1e 0e 01 05 0a 19 0e 04 17 1b 14 04 15 0b 04 17 18 0a 1b 1d 1a 0e 10 17 17 07
page 123: 08 08 08 01 0e 1b 08 16 04 1d 0d 05 0f 1d 1a 08 11 1b 01 01 0a 00 0e 10 11 09 05 1e 1c 0c 05 0c
page 124: 07 06 02 15 17 1a 15 1d 08 15 02 04 01 10 0d 0c 1b 0a 13 17 1c 16 1c 1d 18 0c 1d 1a 16 00 08 12
page 125: 1c 07 07 17 02 0a 02 02 07 09 1e 1a 10 02 1b 0b 18 0a 1e 0e 06 1c 18 05 08 0a 08 1c 09 13 09 09
page 126: 02 06 07 00 19 0f 1b 05 14 1d 0c 0d 0e 17 03 0c 1e 18 1a 0d 01 1e 09 17 0f 12 06 1e 18 03 19 16
page 127: 15 0f 19 0c 06 01 0f 16 0d 09 1b 04 13 1b 15 09 1e 19 10 18 17 00 0b 0e 09 04 0c 0f 05 09 01 17
```

You are told that the **page directory** is located within **page 99** (decimal).

Assume we are translating the virtual address 0x778e, which is a valid virtual address for the process in question. The memory dump above has no bits flipped to some incorrect value; rather, we will consider cases where things go wrong and discuss exactly which bit in memory must be flipped to cause the "wrong" thing to occur.

(a) Before flipping any bits, please translate the virtual address 0x778e; show both the resulting physical address and value that is read from memory if accessing said address. Be detailed and show your work!

Now we're going to talk about what would happen if some bits get flipped in memory. In this case, you are an analyst, and are trying to separate rumor from truth.

(b) You hear about about a case where bits in the **page directory** get flipped and cause all translations to be invalid. Is this possible? If so, what bits need to be flipped? If not, why not?

(c) You also hear about a case where bits are flipped in **page table entries** such that a load to **any** virtual address always returns the value 0x00. Is this possible? If so, how could it happen? If not, why not?

(d) Finally, you hear about a problem where a bit flip leads to a physical address that is greater than 12 bits long. Is this possible? If so, how could it happen? If not, why not?

4. **Iron TLB.**

One version of the Iron system has a TLB. In this case, the TLB structure looks like this:

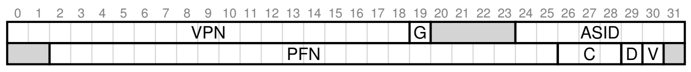

Figure 1: A ~~MIPS~~ Iron TLB Entry.

The VPN and PFN fields should be self-explanatory, as should the V field (valid); the ASID field (an address-space identifier field); and the D field (dirty); ignore any other fields.

On this system, Iron OS has a software-managed TLB. Thus, the OS is responsible for installing the correct translation when a TLB miss occurs. When finished with the update to the TLB, the OS returns from a trap, and the hardware retries the instruction.

Unfortunately, just before Iron OS updates the TLB, sometimes a bit gets flipped and thus the wrong translation ends up in the TLB! For each of the following fields, both (1) **describe what the field is used** for and (2) **explain what you think would happen if a bit gets flipped** in said field just before the OS installs the entry in the TLB:

(a) VPN:

i. What is the VPN field for?

ii. What would happen if a bit in the VPN got flipped?

(b) PFN: (same questions)

i. What is the PFN field for?

ii. What would happen if a bit in the PFN got flipped?

(c) ASID: (same questions)

i. What is the ASID field for?

ii. What would happen if a bit in the ASID got flipped?

(d) Valid: (same questions)

i. What is the Valid bit for?

ii. What would happen if the valid bit got flipped?

(e) Dirty: (same questions)

i. What is the Dirty bit for?

ii. What would happen if the dirty bit got flipped?

5. **Iron Page Replacement.**

   The Iron OS page replacement algorithm uses **reference** bits to approximate LRU.

   (a) Why does Iron OS need to approximate LRU? Why not just implement LRU directly?

   (b) How does a scheme with reference bits work? Describe.

   (c) Assume some reference bits get flipped by accident. What problem(s) would arise? Would the system work correctly despite these bit flips?

   (d) The page table also has **presence** bits. Describe how presence bits work, and why they are needed.

   (e) Assume some presence bits get flipped by accident. What problem(s) would arise? Would the system work correctly despite these bit flips?

6. **Towards a Real Iron OS.**

   In this last question, we discuss how we might actually deal with memory corruption, in specific with the page tables of the system.

   To make the problem more tractable, assume the following:

   - A bit flip only occurs once per day, to some random bit in one of your page tables (thus, the bit flip never happens to code, which is put in higher quality memory that has extra protection against such flips).
   - This version of Iron OS (without any special protection) uses a simple **linear page table** (per process) and a **software-managed TLB**.

   How would you change your page table structure so that you can both detect and recover from a bit flip in memory? How would your TLB handler access this data structure to work correctly despite bit flips? (the more details here, the better)