UMass Boston CS444 f26 Review for Test1 J.H.DeBlois

0. TAKING TEST1:

Please show your id and turn off any digital device(s) you have brought with you. No notes are allowed. I'll write the last four of your UMB id on the test.

The two of you will be given a page with your test1 on it. You will also be given printed pages of any figures referred to in your questions.

Your test1 will have 4 sections:
Q1 Hello… and what did you find interesting in Chapter n? Chapter n? where n = {1, 2, 3, 4, 5, 12}. Student with lower last four in UMB id answers for the first chapter listed.
Q2 True/False: 8 statements
Q3 C code questions: a and b
Q4 OS Methods questions: a and b

The time allotted and points are:
Q1 2 mins, 8 points – if neither student has an answer, you get 0 points
Q2 8 mins, 8 T/F, 4 points each, 32 points
Q3 7 mins, 2 C code questions, 15 points each, 30 points
Q4 7 mins, 2 OS Methods questions, 15 points each, 30 points

The OS Methods questions are asking you to explain abstractions about how the operating system is working to assist you in running your program or helping other users run theirs.

You and your test partner will get the same score. On Q1, you each give your own prepared answer on the particular chapter.

You will write and/or draw your answers on the whiteboard. You will be able to discuss and make notes on the whiteboard and refer to them as you answer. For each question, you will give your answer verbally. Either partner can speak.

The test takes <u>about</u> 25 minutes. Time goes quickly. I will bring a clock and note the start time for each question. Please keep an eye on how time is progressing and give your answers within the time limit for each question.

I'll compute your score on my track sheet and tell you before you leave. Please arrive early. You can sit outside Swami's office while you wait. If you miss part of your test time, you will lose points.

1. Chapter 1: Introduction.
In Ch1, define an OS in relation to a computer (p1).
Know that the OS commands have to run in the CPU with higher privilege than the user tasks. Know that there are many kinds and sizes of OS. Define process, address space, files, directories, protection, input-output and the shell. Understand a system call in detail. Know the segments of a process (p57). Contrast UNIX to the Windows API. Know what a virtual machine is (p69-70) and how it does two levels of traps. Know the C code in section 1.8. Know the metric prefixes and the special definitions for memory. Know the ASCII code chart.

T/F examples:
1-Virtual memory cannot be bigger than physical memory.
2-When the OS runs in user mode, it has access to all the hardware and can execute any instruction the machine is capable of.
3-The Minux3 microkernel is about 15,000 lines of C code and 1400 lines of assembler. (p67)
4-Windows and Linux are in the 10s of millions of lines of code. (p3, Windows > 50 million, Linux > 20 million)
5-The memory hierarchy has four layers. (p25)

ANS: F, F, T, T, T.

OS Method question examples:
1-Figure 1-17, p52, Explain the 11 steps in making a read system call.
2-Figure 1-17, p52, Explain the 9 steps in making an exit system call.

Ans: see textbook, p52

C Code question examples:
1-Figure 1-19 A stripped down shell, p55: Notice the three system calls in the figure: exec, fork and exit. What type of system calls are they? (p55-56)

2-Figure 1-19 A stripped down shell, p55: Notice the three system calls in the figure: exec, fork and exit. What does exec do? (p55-56)

3-Figure 1-19 A stripped down shell, p55: Notice the three system calls in the figure: exec, fork and exit, what does fork do? (p55-56)

4-Figure 1-19 A stripped down shell, p55: After a fork, the parent and child share a copy of memory: when does the OS give the child process its own address space? (p55-56)

2. Chapter 2: Processes and Threads.
A process is an abstraction of a running program. Know how to draw

multiprogramming of several programs on a timeline. See accompanying diagram, Figure 2-21, and Figure 2-1. You will be required to quickly draw timelines for multiple running programs on the same set of axes. Know 4 ways for processes to get started (p88-89). Know their states. Know busy-waiting vs. interrupt processing. Know what threads offer. Know a race condition example (p120). Know how to mark a critical region in your code to get mutual exclusion when several processes can run the same code. Define and use a semaphore. Explain priority inversion by giving an example. Know scheduling algorithms.

T/F:
1-PSW means Program Status Word.
2-When two processes are reading or writing some shared data the final result does not depend on who runs precisely when.

3-Locking the memory bus has the same effect as disabling interrupts.

Answers: T,F,F:
See page 126 in textbook:
"It is important to note that locking the memory bus is very different from disabling interrupts. Disabling interrupts than performing a read on a memory word followed by a write does not prevent a second processor on the bus from accessing the word between the read and the write. In fact, disabling interrupts on processor 1 has no effect at all on processor 2. The only way to keep processor 2 out of the memory until processor 1 is finished is to lock the bus, which requires a special hardware facility (basically, a bus line asserting that the bus is locked and not available to processors other than the one that locked it)."

3. Chapter 3. Memory Management.
Know what it means to have no memory abstraction. Be able to draw two programs in memory as shown in Figure 3-2c (p182). Note that the numbers on the right are for the first byte in the row, so the last byte would have a higher number, depending the row size.

See the picture of memory on the last page. Be able to declare a four byte int (named c), assign it value 6, store it at address 0. Little endian, if you can. Then declare an int pointer (named ip), assign it address 4, get the address of int c and store the address of int c as the data for int pointer ip. Repeat it for your own int and pointer.

Know how to read the output of hexdump -C on a file. Then know the abstraction called address space, the abstract memory for programs to use (p184). Be able to draw Figure 3-3 (pp186). What is swapping? How is free memory space managed?

What does virtual memory do for us (p192)? What translates virtual memory addresses to physical memory addresses? What are pages and what are page frames? Explain Fig 3-10 on p197. Why would we need

multi-level page tables? What is a page fault?
What is a page replacement algorithm? Describe the steps of page fault handling (pp233-234). Review your hw1 problem of how to draw addresses and data for different size pages.

T/F:
1-NULL is the integer three, as defined in stdio.h.
2-CPU utilization is a function of the number of processes in memory.
3-Race conditions can be solved by the use of critical regions and mutual exclusion.
4-For int *pa;, you can write pa=&a[0] and pa=a.

ANS: Consult Kernighan and Ritchie. See p176 for stdio.h contents, Also see textbook p75 for pointer handling. Also see textbook, p96 for CPU utilization. See textbook pp119-124 for race condtions.

Answers: F, T, T, T

OS Method question:
1-In Figure 1-9, p25, Typical Memory Hierarchy, there are four levels with sizes and access times. When would your code run at about a nanosecond per instruction? When would it take longer? How much longer on average for going to disk to write a file update? Define the units you used.

Ans: When it is doing operations locally. When it requires system calls. Disk operations require milliseconds. Nanosecond is a billionth of a sec. Microsecond is a millionth. Millisecond is a thousandth.

4. Chapter 4. File Systems.
Be sure to look at Figure 4-6. A simple program to copy a file, p271. Understand pp270-272 and be able to explain when the code is doing a system call and what the operating system is doing for the code.
Study I-nodes, p284-285.

T/Fs:
1-As shown in Figure 4-21, the virtual file system has two interfaces, one to user process running code and one to physical file system(s) with buffers.
2-Once block size is chosen by the designers of the file system, a method of keeping track of free blocks can be selected from common choices of linked-list or bitmap.

ANS:
First one is true – Be sure to know Figure 4-21 and the text that talks about it.

Second one, True. Review the figures in section 4.3.2, especially what is the i-node. Be sure you see why the two ways of implementation work.

OS method question:
1-In Fig 4-18, p288, explain why the link in black indicates a shared file. On the cs server your /home has a link to cs444. Your course directory is actually under /courses. Draw the picture and label the paths.

C code question:
1-Fig 4-6, p271, has a copy file program. Assume a copy is made. What changes are needed to the file to copy from the new file to another filename.

Ans: list which variables in the file go with the original file vs the new file, then substitute.

5. Chapter 5. Input/Output.
Get the idea of I/O devices from section 5.1.1 and the list in Figure 5-1. With such different data rates, there is bound to be a need for buffering. There may be delays. Interrupts are very important. They need special attention when the OS is partially completing instructions. Controllers are in the devices themselves. Drivers are in the OS. See functions of the layers in Fig 5-17 (p369). With disks, know RAID and parity bits.
In a layered diagram, with hardware at the bottom, the U/I would be at the top. There is a big difference between command-line access and access via a window. Both ways select programs to run. Any OS will make processes and threads to run the program. For command-line access, the shell handles the U/I hardware. For access via a window, the windowing support software handles the U/I hardware. Both software types have to issue system calls to move from user space to kernel space and access the OS resources needed.

T/F:
1-If execution of instructions can be out of order and interrupted when partially complete, the only type of interrupt possible is precise interrupts.

Ans: See Section 5.1.5 pp347-351. Figure 5-5 illustrates the interrupts that can happen. Know precise and imprecise interrupts in the text and in Figure 5-6. The statement is false. Both are possible.

Another T/F:
2-No matter whether a CPU does or does not have memory-mapped I/O, it needs to address the device controllers to exchange data with them.

Ans: See p342. This is true.

OS Method question:
1-Fig 5-6 If a task must be swapped out, it's easy in (a). What could be done in (b)?

Ans: see p351. The program counter could be moved to point to the first instruction "not executed" as next instruction. Complete the processing of instructions from 304-324, then swap. Or use journaling information to back down.

C Code question;
1-What is the main loop of the X Window application program in Fig 5-34 doing?

Ans: While it is running, it gets the next event for the window and executes the code in the case for that event type.

6. Chapter 12. Operating System Design.
Tanenbaum and Bos admit there is no consensus on this subject, so they rely on their personal experience and well-known books. To design an OS which is very large, set your goals (abstractions, primitive operations, isolation and hardware management (p1042)). Strive for architectural coherence (p1048), and Corbato's "minimum of mechanism and maximum of clarity". Think about the interfaces you need to build. What is your user-interface paradigm (metaphor)? What is your execution paradigm (p1049)? What is your data paradigm? Implement using careful naming. A static structure allows fast access but might run out of space (p1062). Be sure to focus on desired outputs first.

T/F:
1-Every programmer contributes equally.

Ans: F. See p1080.

OS Method Question:
1-Describe how caching path/i-node combinations could save disk accesses when looking up a long absolute path.

Ans: Suppose you have to look up /usr/ast/grants/erc and the cache has /usr/ast/ as inode 26, you save 6 disk accesses.

C Code question:
1-Explain the algorithm implemented by the code on the left.

Ans: Algorithms have an order of steps. Here the steps are

generalized by init, do_something, read, do_something_else, …, exit.

7. WHAT IS INCLUDED. The test1 covers reading Chapters 1-5 and 12 from the Tanenbaum and Bos, Ed. 5 textbook, slides from class for chapters 1-5, 12 and huffman, your hw1, C code from class, grammar from Kernighan and Ritchie and the ASCII code chart.

Your hw1 covered huffman code prefix code tree and reading codes, metric prefixes and ASCII codes, tokens, operators and bit operations.

These are the names of the chapters studied:
Chapter 1. Introduction.
Chapter 2. Processes and Threads.
Chapter 3. Memory Management.
Chapter 4. File Systems.
Chapter 5. Input/Output.
Chapter 12. Operating System Design.
For Q1, **you need to know the name of each chapter – your test1 lists only the chapter number.**

These are the C code figures:
Fig 1-19, p55, A stripped down shell. True is defined as 1.
Intext, p75, C code in text for pointer understanding.
Fig 1-30, p77, Compiling – try command “which gcc”, is it /usr/bin/gcc?

Fig 2-9, p101, Outline of code for multi-threaded Web server. (a) Dispatcher thread. (b) Worker thread.
Fig 2-14, p108, Example program using threads.
Fig 2-23, p123, A proposed solution to the critical region problem.
Fig 2-27, p129, Producer Consumer with fatal race condition.
Fig 2-28, p131, Producer Consumer using semaphores.
Fig 2-40, p155, Bursts of CPU usage alternate with periods of waiting for I/O. (a) a CPU-bound process. (b) an I/O-bound process.
Fig 2-44, p164, A scheduling algorithm with four priority classes.

Fig 4-6, p271, A simple program to copy a file.

Fig 5-34. A skeleton of an X Window application program.

Fig 12-1, p1049, (a) Algorithmic code. (b) Event-driven code.
Fig 12-5, p1062, Code for searching the process table for a given PID.
Fig 12-7, p1073, Ways to count bits in a byte.

We also studied endian.c and program2.c.
You need to know pointers, files, bit operations, fwrite and grammar

from Kernighan and Ritchie and the huffman slides. You need to understand that an address can show up as data at the address of a pointer variable.

These are the additional Chapter 1 figures that pretty much summarize the textbook:

Fig 1-1, p2, Where the Operating System fits in.
Fig 1-5, p12, A Multiprogamming system.
Fig 1-6, p21, Some of the components of a simple personal computer.
Fig 1-7, p22, a) 3-stage pipeline, b) superscalar CPU
Fig 1-8, p25, (a) A quad-core chip with a shared L2 cache (Intel). (b) A quad-core chip with separate L2 caches (AMD).
Fig 1-9, p25, A typical memory hierarchy.
Fig 1-10, p28, Structure of a Disk Drive.
Fig 1-11, p32, a) Steps in starting an I/O device and getting an interrupt. b) Interrupt processing involves taking the interrupt, running the interrupt handler and returning to the user program.
Fig 1-13, p41, Process tree
Fig 1-14, p43, A file system for a university department.
Fig 1-17, p52, The 10 steps in making the system call read(fd, buffer, nbytes).
Fig 1-18, p54, Some of the major POSIX system calls.
Fig 1-20, p57, Processes have three segments.
Fig 1-23, p62, The corresponding Win32 API calls.
Fig 1-27, p69, The client-server model over a network.
Fig 1-28, p70, The structure of VM/370 with CMS, 1979.
Fig 1-31, p81, Principal metric prefixes (and the text regarding prefixes for talking about memory size)

These are the additional OS method figures:

Fig 2-1, p87, (a) Multiprogramming four programs. (b) Conceptual model of four independent, sequential processes. (c) Only one program is active at once.
Fig 2-2, p93, States of a process with transitions.
Fig 2-4, p95, Some of the fields of a typical process-table entry.
Fig 2-6, p96, CPU utilization as a function of # of processes in memory.
Fig 2-11. Per process vs per thread items, p104.
Fig 2-21, p120, Two processes want to access shared memory at the same time.
Fig 2-22, p122, Mutual exclusion using critical regions.

Fig 3-8, p194, The position and function of the MMU.
Fig 3-9, p195,Page Table.
Fig 3-10, p197, Operation of MMU.
Fig 3-11, p198, A typical page table entry.
Fig 3-13, p204, Multi-level page tables.
Fig 3-21, p220, Summary of page replacement algorithms.
Fig 3-27, p235, An instruction causing a page fault.

```
Fig 4-8, p274, Hierarchical Directories.
Fig 4-10, p279, A possible file system layout.
Fig 4-15, p284, An example i-node (with all the blocks of a file).
Fig 4-17, p287, Two ways of handling long file names. (a) In0line.
(b) In a heap.
Fig 4-18, p288, File system containing a shared file.
Fig 4-21, p299, Position of the virtual file system.
Fig 4-23, p303, The dashed curve (left-hand scale) gives the data
rate of a disk. The solid curve (right-hand scale) gives the disk-
space efficiency.  All files are 4KB.
Fig 4-27, p310, A file system to be dumped.
Fig 4-35, p329, A UNIX i-node.
Fig 4-36, p329, The steps in looking up /usr/ast/mbox.

Fig 5-1, p339, Some typical device, network and bus data rates.
Fig 5-3, p343, (a) A single-bus architecture. (b) A dual-bus
architecture.
Fig 5-4, p345, Operation of a DMA transfer.
Fig 5-5, p348, How an interrupt happens.
Fig 5-6, p351, Precise and Imprecise interrupts.
Fig 5-11, p357, Layers of the I/O software system.
Fig 5-12, p360, Logical positioning of device drivers.
Fig 5-14, p363, With/without a standard driver interface.
Fig 5-17, p369, Layers of the I/O system and main functions of each
layer.
Fig 5-22, p376, Shortest seek first disk scheduling algorithm.
Fig 5-26, p388, RAID levels 0 to 6 with parity shaded.
Fig 5-23, p405, Clients and servers in the M.I.T. X Window System.

Fig 12-2, p1054, One possible design for a modern layered OS.
Fig 12-3, p1056, Client-server computing based on a microkernel.
```

An 8-bit machine with an **8-bit address bus** can reference a total of **256 bytes** of memory (2^8 = 256). If we display **4 bytes of data per line**, a 256-byte memory space requires exactly **64 lines** (256 \4 = 64). Each line represents a block of 4 bytes, so the address increments by 4 (in binary) for each line moving up. Below is the diagram sorted in reverse order with address `00000000` at the bottom. Each byte contains 8 bits, represented by an underline (_) and a space, with three spaces separating bytes.

Prompt:Prompt:
draw address and memory for an 8-bit machine with an 8 bit address bus, lable all addresses in binary, show 4 data bytes on a line with one underline and one space for each bit and three spaces between the bits, write out all 64 lines completely, listing addresses in reverse order so address 00000000 is at the bottom.

Please also translate addresses to hex. Picture is on next page.

Address	Data (4 Bytes per line: 8 bits per byte)			
11111100	_ _ _ _ _ _ _ _	_ _ _ _ _ _ _ _	_ _ _ _ _ _ _ _	_ _ _ _ _ _ _ _
11111000	_ _ _ _ _ _ _ _	_ _ _ _ _ _ _ _	_ _ _ _ _ _ _ _	_ _ _ _ _ _ _ _
11110100	_ _ _ _ _ _ _ _	_ _ _ _ _ _ _ _	_ _ _ _ _ _ _ _	_ _ _ _ _ _ _ _
11110000	_ _ _ _ _ _ _ _	_ _ _ _ _ _ _ _	_ _ _ _ _ _ _ _	_ _ _ _ _ _ _ _
11101100	_ _ _ _ _ _ _ _	_ _ _ _ _ _ _ _	_ _ _ _ _ _ _ _	_ _ _ _ _ _ _ _
11101000	_ _ _ _ _ _ _ _	_ _ _ _ _ _ _ _	_ _ _ _ _ _ _ _	_ _ _ _ _ _ _ _
11100100	_ _ _ _ _ _ _ _	_ _ _ _ _ _ _ _	_ _ _ _ _ _ _ _	_ _ _ _ _ _ _ _
11100000	_ _ _ _ _ _ _ _	_ _ _ _ _ _ _ _	_ _ _ _ _ _ _ _	_ _ _ _ _ _ _ _
11011100	_ _ _ _ _ _ _ _	_ _ _ _ _ _ _ _	_ _ _ _ _ _ _ _	_ _ _ _ _ _ _ _
11011000	_ _ _ _ _ _ _ _	_ _ _ _ _ _ _ _	_ _ _ _ _ _ _ _	_ _ _ _ _ _ _ _
11010100	_ _ _ _ _ _ _ _	_ _ _ _ _ _ _ _	_ _ _ _ _ _ _ _	_ _ _ _ _ _ _ _
11010000	_ _ _ _ _ _ _ _	_ _ _ _ _ _ _ _	_ _ _ _ _ _ _ _	_ _ _ _ _ _ _ _
11101100	_ _ _ _ _ _ _ _	_ _ _ _ _ _ _ _	_ _ _ _ _ _ _ _	_ _ _ _ _ _ _ _
11001000	_ _ _ _ _ _ _ _	_ _ _ _ _ _ _ _	_ _ _ _ _ _ _ _	_ _ _ _ _ _ _ _
11000100	_ _ _ _ _ _ _ _	_ _ _ _ _ _ _ _	_ _ _ _ _ _ _ _	_ _ _ _ _ _ _ _
11000000	_ _ _ _ _ _ _ _	_ _ _ _ _ _ _ _	_ _ _ _ _ _ _ _	_ _ _ _ _ _ _ _
10111100	_ _ _ _ _ _ _ _	_ _ _ _ _ _ _ _	_ _ _ _ _ _ _ _	_ _ _ _ _ _ _ _
10111000	_ _ _ _ _ _ _ _	_ _ _ _ _ _ _ _	_ _ _ _ _ _ _ _	_ _ _ _ _ _ _ _
10110100	_ _ _ _ _ _ _ _	_ _ _ _ _ _ _ _	_ _ _ _ _ _ _ _	_ _ _ _ _ _ _ _
10110000	_ _ _ _ _ _ _ _	_ _ _ _ _ _ _ _	_ _ _ _ _ _ _ _	_ _ _ _ _ _ _ _
10101100	_ _ _ _ _ _ _ _	_ _ _ _ _ _ _ _	_ _ _ _ _ _ _ _	_ _ _ _ _ _ _ _
10101000	_ _ _ _ _ _ _ _	_ _ _ _ _ _ _ _	_ _ _ _ _ _ _ _	_ _ _ _ _ _ _ _
10100100	_ _ _ _ _ _ _ _	_ _ _ _ _ _ _ _	_ _ _ _ _ _ _ _	_ _ _ _ _ _ _ _
10100000	_ _ _ _ _ _ _ _	_ _ _ _ _ _ _ _	_ _ _ _ _ _ _ _	_ _ _ _ _ _ _ _
10011100	_ _ _ _ _ _ _ _	_ _ _ _ _ _ _ _	_ _ _ _ _ _ _ _	_ _ _ _ _ _ _ _
10011000	_ _ _ _ _ _ _ _	_ _ _ _ _ _ _ _	_ _ _ _ _ _ _ _	_ _ _ _ _ _ _ _
10010100	_ _ _ _ _ _ _ _	_ _ _ _ _ _ _ _	_ _ _ _ _ _ _ _	_ _ _ _ _ _ _ _
10010000	_ _ _ _ _ _ _ _	_ _ _ _ _ _ _ _	_ _ _ _ _ _ _ _	_ _ _ _ _ _ _ _
10001100	_ _ _ _ _ _ _ _	_ _ _ _ _ _ _ _	_ _ _ _ _ _ _ _	_ _ _ _ _ _ _ _
10001000	_ _ _ _ _ _ _ _	_ _ _ _ _ _ _ _	_ _ _ _ _ _ _ _	_ _ _ _ _ _ _ _
10000100	_ _ _ _ _ _ _ _	_ _ _ _ _ _ _ _	_ _ _ _ _ _ _ _	_ _ _ _ _ _ _ _
10000000	_ _ _ _ _ _ _ _	_ _ _ _ _ _ _ _	_ _ _ _ _ _ _ _	_ _ _ _ _ _ _ _
01111100	_ _ _ _ _ _ _ _	_ _ _ _ _ _ _ _	_ _ _ _ _ _ _ _	_ _ _ _ _ _ _ _
01111000	_ _ _ _ _ _ _ _	_ _ _ _ _ _ _ _	_ _ _ _ _ _ _ _	_ _ _ _ _ _ _ _
01110100	_ _ _ _ _ _ _ _	_ _ _ _ _ _ _ _	_ _ _ _ _ _ _ _	_ _ _ _ _ _ _ _
01110000	_ _ _ _ _ _ _ _	_ _ _ _ _ _ _ _	_ _ _ _ _ _ _ _	_ _ _ _ _ _ _ _
01101100	_ _ _ _ _ _ _ _	_ _ _ _ _ _ _ _	_ _ _ _ _ _ _ _	_ _ _ _ _ _ _ _
01101000	_ _ _ _ _ _ _ _	_ _ _ _ _ _ _ _	_ _ _ _ _ _ _ _	_ _ _ _ _ _ _ _
01100100	_ _ _ _ _ _ _ _	_ _ _ _ _ _ _ _	_ _ _ _ _ _ _ _	_ _ _ _ _ _ _ _
01100000	_ _ _ _ _ _ _ _	_ _ _ _ _ _ _ _	_ _ _ _ _ _ _ _	_ _ _ _ _ _ _ _
01011100	_ _ _ _ _ _ _ _	_ _ _ _ _ _ _ _	_ _ _ _ _ _ _ _	_ _ _ _ _ _ _ _
01011000	_ _ _ _ _ _ _ _	_ _ _ _ _ _ _ _	_ _ _ _ _ _ _ _	_ _ _ _ _ _ _ _
01010100	_ _ _ _ _ _ _ _	_ _ _ _ _ _ _ _	_ _ _ _ _ _ _ _	_ _ _ _ _ _ _ _
01010000	_ _ _ _ _ _ _ _	_ _ _ _ _ _ _ _	_ _ _ _ _ _ _ _	_ _ _ _ _ _ _ _
01001100	_ _ _ _ _ _ _ _	_ _ _ _ _ _ _ _	_ _ _ _ _ _ _ _	_ _ _ _ _ _ _ _
01001000	_ _ _ _ _ _ _ _	_ _ _ _ _ _ _ _	_ _ _ _ _ _ _ _	_ _ _ _ _ _ _ _
01000100	_ _ _ _ _ _ _ _	_ _ _ _ _ _ _ _	_ _ _ _ _ _ _ _	_ _ _ _ _ _ _ _
01000000	_ _ _ _ _ _ _ _	_ _ _ _ _ _ _ _	_ _ _ _ _ _ _ _	_ _ _ _ _ _ _ _
00111100	_ _ _ _ _ _ _ _	_ _ _ _ _ _ _ _	_ _ _ _ _ _ _ _	_ _ _ _ _ _ _ _
00111000	_ _ _ _ _ _ _ _	_ _ _ _ _ _ _ _	_ _ _ _ _ _ _ _	_ _ _ _ _ _ _ _
00110100	_ _ _ _ _ _ _ _	_ _ _ _ _ _ _ _	_ _ _ _ _ _ _ _	_ _ _ _ _ _ _ _
00110000	_ _ _ _ _ _ _ _	_ _ _ _ _ _ _ _	_ _ _ _ _ _ _ _	_ _ _ _ _ _ _ _
00101100	_ _ _ _ _ _ _ _	_ _ _ _ _ _ _ _	_ _ _ _ _ _ _ _	_ _ _ _ _ _ _ _
00101000	_ _ _ _ _ _ _ _	_ _ _ _ _ _ _ _	_ _ _ _ _ _ _ _	_ _ _ _ _ _ _ _
00100100	_ _ _ _ _ _ _ _	_ _ _ _ _ _ _ _	_ _ _ _ _ _ _ _	_ _ _ _ _ _ _ _
00100000	_ _ _ _ _ _ _ _	_ _ _ _ _ _ _ _	_ _ _ _ _ _ _ _	_ _ _ _ _ _ _ _
00011100	_ _ _ _ _ _ _ _	_ _ _ _ _ _ _ _	_ _ _ _ _ _ _ _	_ _ _ _ _ _ _ _
00011000	_ _ _ _ _ _ _ _	_ _ _ _ _ _ _ _	_ _ _ _ _ _ _ _	_ _ _ _ _ _ _ _
00010100	_ _ _ _ _ _ _ _	_ _ _ _ _ _ _ _	_ _ _ _ _ _ _ _	_ _ _ _ _ _ _ _
00010000	_ _ _ _ _ _ _ _	_ _ _ _ _ _ _ _	_ _ _ _ _ _ _ _	_ _ _ _ _ _ _ _
00001100	_ _ _ _ _ _ _ _	_ _ _ _ _ _ _ _	_ _ _ _ _ _ _ _	_ _ _ _ _ _ _ _
00001000	_ _ _ _ _ _ _ _	_ _ _ _ _ _ _ _	_ _ _ _ _ _ _ _	_ _ _ _ _ _ _ _
00000100	_ _ _ _ _ _ _ _	_ _ _ _ _ _ _ _	_ _ _ _ _ _ _ _	_ _ _ _ _ _ _ _
00000000	_ _ _ _ _ _ _ _	_ _ _ _ _ _ _ _	_ _ _ _ _ _ _ _	_ _ _ _ _ _ _ _