Final -- Fall 2019 Name: ____________________

CS33: Intro Computer Organization UID: ____________________

This is an open book, open notes exam, but you cannot share books/notes. Please follow the university guidelines in reporting academic misconduct.

Note that there is an ASCII Table at the end of this exam.

Please wait until everyone has their exam to begin. We will let you know when to start.

Good luck!

		Score	Points Possible
1	Warming up your cache		8
2	Multiple Choice		30
3	Variable Location		8
4	Streaming in Cache		8
5	Performance		10
6	Crossword		10
7	Virtual Memory		12
8	Forking		9
9	Overflow		5
The above add up to 100 points … below are challenges			
10	“I can’t even add one”		8
11	“Broken Bank”		8
12	“An obscure message”		8
			100 (124)

(advice: there might not be enough time to do all challenges, so prioritize carefully)

Question 1. Warming up your cache (8 pts)

In the broadest sense, a "cache" is a structure which can hold data to speedup up access to a (usually larger) structure. The memory hierarchy can be viewed as a series of caches, made up of different technologies. Some entity in the system needs to "manage" the cache -- ie. it needs to decide what is in the cache at any time, and possibly maintain consistency between different caches. Management can be done by the hardware, the compiler, the programmer, or even the operating system.

Fill in the blanks for each of the following questions. You may not use exactly the same combination of answers for each a,b,c, or d.

a) **<u>on-chip caches</u> are a cache for _____________,**

and they are managed by _____________

b) **_____________ are a cache for <u>on-chip cache,</u>**

and they are managed by _________________

c) **<u>Main memory</u> can be seen as a cache for _____________,**

and it is managed by the _____________

d) **_____________ is a cache for _____________,**

and it is managed by the <u>operating system</u>

Question 2. Multiple Choice (30 pts)
For each question, **mark all that apply**. Only answers that mark **all correct choices** will be considered correct.

1. In the context of multi-threading, which of the following are guaranteed to be thread-private:
(a) Static Value (eg. “static int” in a function)
(b) Values on Thread’s stack
(c) Values on Heap
(d) Value stored in register
(e) Global Value

2. Why could automatic function call inlining sometimes be bad for performance?
(a) It introduces extra dynamic instructions, potentially adding to the critical path length.
(b) It introduces extra static instructions, putting more strain on the instruction cache.
(c) Fewer call instructions reduce the instruction-level parallelism.
(d) Trick question, it is never harmful for performance.

3. What of the following are reason(s) why multi-threaded code might go slower than expected?
(a) More contention for data in the cache.
(b) Overhead of synchronizing shared variables.
(c) Overhead of starting/stopping threads.
(d) Space overhead from .bss/.data sections.

4. How is __sync_fetch_and_add() implemented?
(a) It uses a binary semaphore.
(b) It uses a pthread mutex.
(c) It uses a pthread barrier.
(d) It uses an atomic instruction.

5. Assuming no errors, which one of the following functions returns exactly twice?
(a) fork()
(b) execve()
(c) exit()
(d) wait()

6. What are the functions of the page table?
(a) Improves performance for data-intensive codes
(b) Provides Translation between virtual and physical addresses
(c) Improves cache locality

7. Why is address translation performed before accessing the cache?
(a) It improves the performance of the cache hierarchy
(b) It improves energy consumption
(c) It makes distinguishing addresses from different processes easier
(d) It improves temporal locality
(e) It improves spatial locality

8. Why is a cache tag smaller than the physical address?
(a) To reduce the number of bits that need to be stored in hardware
(b) To improve cache locality
(c) For correctness of the cache (eg. to handle the case where two processes with the same virtual address refer to different physical addresses)
(d) To prevent memory aliasing

9. You check your linux computer's logs and determine that there are 100 processes running, where each processes has 5 threads and 3 semaphore mutexes. How many page tables exist in the system:
(a) 1
(b) 3
(c) 5
(d) 100
(e) 300
(f) 500

10. Which of the following measures could prevent a return-oriented programming-based stack-overflow attack (like the one in the second part of the attack lab)?
(a) Marking the stack as read-only by setting appropriate bits in the process's page table.
(b) Address space layout randomization (eg. randomly moving the starting position of the stack)
(c) Making sure user input does not overrun the buffer where it is supposed to be stored.
(d) Making sure the compiler is using "stack canaries" (ie. set random value on the stack and make sure it is still the same).

11. You are programming a video game about the apocalypse, and you are using multiple processes. What could be the negative consequences of forking children and not cleaning them up when they die?
(a) The dead children wake up a horde of **daemons**, whose only mission is to cause **page faults**.
(b) While you are playing, the grim **reaper** comes (init process), and cleans up your dead children for you (which you find disappointing).
(c) The dead children rise again as **zombies**, at first they seem harmless and inactive, but eventually they take up so much space (in memory) that they crash the game.
(d) The dead children become **ghosts**, but are prevented by a barrier (pthread barrier) from entering the afterlife.

----- malloc-lab questions ------

12. In malloclab, we provided code for an implicit list allocator. Some students improved this code by creating a linked list of free blocks. Why did this increase the performance of the allocator?
(a) Each step in traversing a linked list is significantly faster than moving from block to block in the implicit list.
(b) The implicit list was longer; it had to include every block in memory, but the linked list could just include the free blocks.
(c) The compiler knows how to optimize the code for a linked list by unrolling loops, but wasn't able to do this for the implicit list.
(d) Having a linked list made coalescing significantly faster.
(e) Relying on the linked list reduced internal fragmentation

[I threw out 13/14 because they were bad questions]

15. Another optimization in malloc lab is to replace the linked list with a binary search tree. Why did this increase the performance of the allocator?
(a) It's faster to traverse each link of a tree compared to a list.
(b) The linked list had to include every free block in memory, but the tree only had to include the blocks which were approximately the correct size.
(c) The compiler knows how to optimize the code for a tree by function-call inlining, but wasn't able to do this for the explicit list.
(d) Having a binary search tree made finding the best fit significantly faster
(e) Relying on the tree reduced internal fragmentation

Question 3. Where's that variable? (8pts)

Consider the following linked-list traversal function, where all linked_list items have been allocated dynamically (by calling malloc).

```
line:
0     struct linked_list;
1
2     long long int total=10000;
3
4     typedef struct linked_list {
5       struct linked_list* next;
6       long long int value;
7     } linked_list;
8
9     long long int* traverse(linked_list* root) {
10      linked_list* current = root;
11      int total=0;
12
13      while(current->next) {
14          total += current->value;
15          current = current->next;
16      }
17      return &total;
18    }
```

1. **Variables can be in global memory, the stack, the heap, or in registers. What is the most likely location of each of the following variables? (be as specific as possible) (6pts)**
   a. **total on line 2** _________________

   b. **root on line 9** ________________

   c. ***root on line 9** _________________

   d. **current on line** 15 ________________

   e. **total on line 17** ________________

   f. **&total on line 17** ________________

2. **Suppose that traverse is compiled into an object file, traverse.o. Which of the above variables (a-f) would have a symbol in this ELF file? (2pts)**

   ____

Question 4. Streaming in Cache (8pts)
Suppose we have two programs on an x86-64 CPU which issues the following series of addresses.

Program A	Program B
0x0000	0x0000
0x0100	0x0004
0x0010	0x0008
0x0110	0x000C
0x0020	0x0010
0x0120	0x0014
0x0030	0x0018
0x0130	0x001C
0x0040	0x0020
0x0140	0x0024
…	0x0028
	...

Suppose that on our CPU, our cache hierarchy has only one L1 data cache, where a cache block is 64 bytes, and the cache has **only one** block. Also assume the caches are initially empty.

1. If requests are for two byte words, what is the cache miss rate of each program? (4pts)

   Program A: _____________

   Program B: ____________

2. If requests are for four byte words, what is the cache miss rate of each program? (2pts)

   Program A: _____________

   Program B: ____________

3. Suppose you had to design the simplest-possible cache to support both programs, which cache parameters would you choose to both minimize cache miss rate and reduce hardware complexity: (2pts)
   a. 1 Block/Set, 16 Blocks (ie. direct mapped)
   b. 2 Blocks/Set, 16 Blocks
   c. 16 Blocks/Set, 16 Blocks (ie. fully associative)

Question 5. Performance Analysis (10 pts)
Consider the following function for computing the product of an array of integers. We have unrolled the loop by a factor of 3. (for simplicity, assume n is divisible by 3)

```
int aprod(int a[], int n) {
  int i, x, y, z;
  int r = 1;
  for (i = 0; i < n-2; i+= 3) {
    x = a[i];
    y = a[i+1];
    z = a[i+2];
    r = r * x * y * z;  // Product
  }
```

For the line labeled “Product”, we can use parentheses to create 5 different associations of the computation, as follows:

```
r = ((r * x) * y) * z; // A1
r = (r * (x * y)) * z; // A2
r = r * ((x * y) * z); // A3
r = r * (x * (y * z)); // A4
r = (r * x) * (y * z); // A5
```

We express the performance of the function in terms of the number of cycles per element (CPE). Intuitively, CPE is the number of cycles per processing each array element.

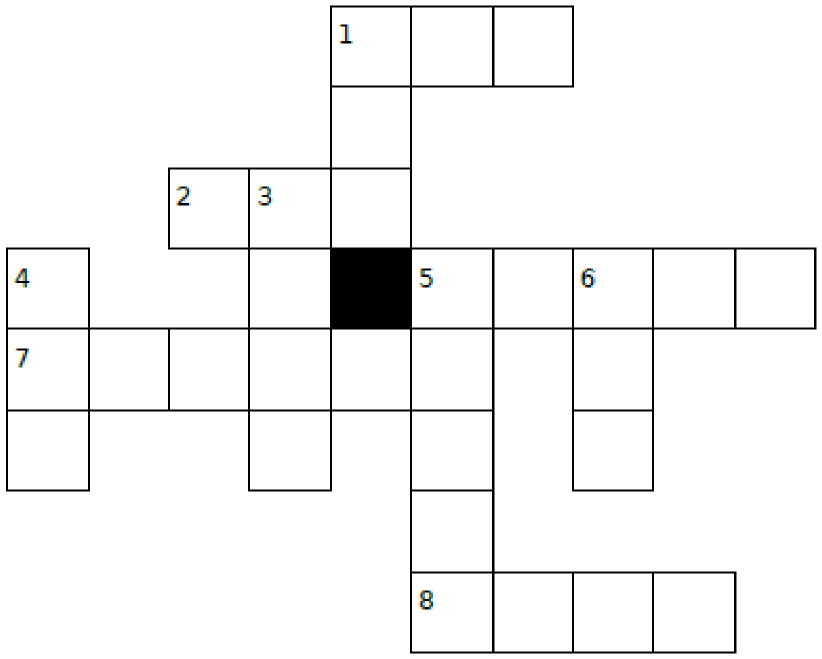
We measured the CPE for 5 versions of the function. Assume integer multiplication has a latency of 4 cycles and an issue time of 1 cycle. Assume all other instructions take one cycle.

The following table shows some values of the CPE, and other values are missing. The measured CPE values are those that were actually observed. “Theoretical CPE” means that performance that would be achieved if the only limiting factor were the latency and issue time of the integer multiplier.

Version	Measured CPE	Theoretical CPE
A1	4.0	
A2	2.67	
A3		4/3 = 1.33
A4	1.67	
A5		8/3 = 2.67

1. Fill in the missing entries above. For the missing values of the measured CPE, you can use the values from other versions that would have the same computational behavior. For the values of the theoretical CPE, you can determine the number of cycles that would be required for an iteration considering only the latency and issue time of the multiplier, and then divide by 3.

Question 6. ISA Fun!!!!!111 (10pts)



Down:
1. / rdi, rsi, __, rcx, r8, r9
3. / smallest unit of data in address space
4. / stack pointer
5. / cheap multiply by 2^n
6. / interface between functions at assembly level

Across:
1. / register where data is returned into
2. / bl, bx, ebx, ...
5. / subtracting from rsp grows this
7. / what a jmp* usually corresponds to in C
8. / tmax plus one

Question 7. MM-You **(12pts)**

Assume:

- Memory accesses are to 4-byte words.
- Virtual addresses are 20 bits wide.
- Physical addresses are 16 bits wide.
- The page size is 4096 bytes.
- The TLB is 4-way set associative with 16 total entries.

In the following tables, all numbers are given in hexadecimal. The contents of the TLB and the page table for the first 32 pages are as follows:

TLB			
Index	Tag	PPN	Valid
0	03	B	1
	07	6	0
	28	3	1
	01	F	0
1	31	0	1
	12	3	0
	07	E	1
	0B	1	1
2	2A	A	0
	11	1	0
	1F	8	1
	07	5	1
3	07	3	1
	3F	F	0
	10	D	0
	32	0	0

Page Table					
VPN	PPN	Valid	VPN	PPN	Valid
00	7	1	10	6	0
01	8	1	11	7	0
02	9	1	12	8	0
03	A	1	13	3	0
04	6	0	14	D	0
05	3	0	15	B	0
06	1	0	16	9	0
07	8	0	17	6	0
08	2	0	18	C	1
09	3	0	19	4	1
0A	1	1	1A	F	0
0B	6	1	1B	2	1
0C	A	1	1C	0	0
0D	D	0	1D	E	1
0E	E	0	1E	5	1
0F	D	1	1F	3	1

1. Your job is to be the MMU, and translate the following three consecutive accesses (they are performed one after the other in the program). Fill in as much information as you can.

Address	TLB Hit?	Page Fault?	Physical Address
1CBAD			
1C0AF			
1DACE			

Question 8. Wrong Utensils (9pts)

Examine the following ridiculous programs and determine what they print.

```
void doit() {
  if (fork() == 0) {
    fork();
    printf("spoon\n");
    return;
  }
  return;
}
int main() {
  doit();
  printf("spoon\n");
  exit(0);
}
```
	1. How many times is “spoon” printed?

```
void doit() {
  if (fork() == 0) {
    fork();
    printf("spoon\n");
    exit(0);
  }
  return;
}
int main() {
  doit();
  printf("spoon\n");
  exit(0);
}
```
	2. How many times is “spoon” printed?

```
int spork = 1;
int main() {
  if (fork() == 0) {
    spork--;
    exit(0);
  } else {
    wait(NULL); //waits for one child
    spork++;
    printf("spork = %d\n", spork);
  }

exit(0);
}
```
	3. What is printed (what is the value of spork)?

Please don't ask us why we make you do this.

Question 9. Comment on CS33 (5pts)

The professor uses a simple C program to compute the final grades in the course. In an array of structs, he maintains each students' name, comment, and final_grade. The comment field is supplied by the student to let the professor know what they think of the course.

```
typedef struct student {
  char username[50];
  char comment[16];
  int final_grade; // 100==A+, 0==F-
} student_t;

student_t users[250];  //well, we had a few drops… only the strong
survive!

void string_copy(char* dest, char* src) {
  int i = 0;
  while(src[i] != 0) { //keep going till we get to the end of the string
    dest[i]=src[i];
    i++;
  }
  dest[i]=0; // don't forget the null character!
}

int set_class_comment(int id, char* comment_from_file) {
  string_copy(users[id].comment, comment_from_file);
}
```

One of the naughty LA's gives you access to a snippet of the source code. At the end you can see the code that copies your comment into the users array.

1. **Write a comment that will ensure you get a good grade! (I would NOT rely on the professor's kindness or weakness to flattery if I were you) Also, try not to set off any alarm bells by setting a grade past 100.**

Question 10. I can't even add one (8 pts)

If you can believe it, the following two source files can be compiled separately and linked with no warnings whatsoever from the compiler or linker. (thanks for nothing dennis ritchie)

src1.c

```c
#include <stdio.h>

typedef struct my_struct {
  char a;
  short b;
  int c;
} my_struct;

my_struct x;

void add1();

int main(int argc, char**argv) {
  x.a=0;
  x.b=0;
  x.c=0;

  add1();
  printf("%x, %x, %x\n",x.a,x.b,x.c);
}
```

src2.c

```c
#include <stdio.h>

typedef struct my_struct {
  int c;
  short b;
  char a;
} my_struct;

my_struct x;

void add1() {
  x.a+=1;
  x.b+=1;
  x.c+=1;
}
```

1. **How large is my_struct in src1? How large is my_struct in src2? (1pts)**

2. **What is printed out? (7pts)**

Question 11. Broken Bank (8pts)

Consider the following code for a multi-threaded account managing software. You can assume integer overflows do not occur.

```
struct account {
      int balance; // in dollars
      sem_t sem; // mutex for this account, initialized to 1
};

struct account accounts[NUM_ACCOUNTS];

// transfers money from one account to another
// return value of 0 means no transfer happened, 1 means successful
Int transfer_dollar(int id_from, int id_to, int amount) {
      // no need to transfer to ourselves
      if(id_from == id_to) return 0;

      // Lock accounts
      P(&accounts[id_from].sem);
      P(&accounts[id_to].sem);

      // make sure there is money to transfer
      if (accounts[id_from].balance < amount) {
            V(&accounts[id_to].sem);
            V(&accounts[id_from].sem);
            return 0;
      }
      // do transfer
      accounts[id_from].balance-=amount;
      accounts[id_to].balance+=amount;

      // Unlock accounts
      V(&accounts[id_to].sem);
      V(&accounts[id_from].sem);
      return 1;
}
```

1. **This code contains a possible deadlock when *multiple* threads call transfer_dollar simultaneously, depending on the inputs. Which input scenario would cause this problem, and give a possible solution.**

Question 12: An obscure message (8pts)
You're enjoying a relaxing winter break in Lake Tahoe, a short reprieve from classes and monotonous CS homework. There are no computers for miles, yes!!! After a long day of skiing while relaxing by a fire, out of nowhere a small envelope is slipped underneath the door of your chalet. When you open it, you find the following x86_64 program in assembly.

You notice it contains calls to "print_char" which is a function that prints the register in rdi. That's interesting, perhaps it prints a message?

```
00000000000008cb <main>:
 8cb:   48 83 ec 08             sub    $0x8,%rsp
 8cf:   bf 62 00 00 00          mov    $0x62,%edi
 8d4:   e8 a1 ff ff ff          callq  87a <print_char>
 8d9:   ba 00 00 00 00          mov    $0x0,%edx
 8de:   be 00 00 00 00          mov    $0x0,%esi
 8e3:   48 8d 3d 56 07 20 00    lea    0x200756(%rip),%rdi   # 201040 <sem>  (sem var is global)
 8ea:   e8 41 fe ff ff          callq  730 <sem_init@plt> #Hint:initial semaphore val in 3rd arg
 8ef:   b9 00 00 00 00          mov    $0x0,%ecx
 8f4:   48 8d 15 98 ff ff ff    lea    -0x68(%rip),%rdx        # 893 <func1>
 8fb:   be 00 00 00 00          mov    $0x0,%esi
 900:   48 8d 3d 59 07 20 00    lea    0x200759(%rip),%rdi        # 201060 <tid>
 907:   e8 e4 fd ff ff          callq  6f0 <pthread_create@plt> #thread function in 3rd arg
 90c:   48 8d 3d 2d 07 20 00    lea    0x20072d(%rip),%rdi        # 201040 <sem>
 913:   e8 f8 fd ff ff          callq  710 <sem_wait@plt>
 918:   bf 65 00 00 00          mov    $0x65,%edi
 91d:   e8 58 ff ff ff          callq  87a <print_char>
 922:   bf 0a 00 00 00          mov    $0xa,%edi
 927:   e8 4e ff ff ff          callq  87a <print_char>
 92c:   c7 05 f6 06 20 00 01    movl   $0x1,0x2006f6(%rip)        # 20102c <wait_var>
 933:   00 00 00
 936:   be 00 00 00 00          mov    $0x0,%esi
 93b:   48 8b 3d 1e 07 20 00    mov    0x20071e(%rip),%rdi        # 201060 <tid>
 942:   e8 f9 fd ff ff          callq  740 <pthread_join@plt>
 947:   bf 63 00 00 00          mov    $0x63,%edi
 94c:   e8 29 ff ff ff          callq  87a <print_char>
 951:   bf 73 00 00 00          mov    $0x73,%edi
 956:   e8 1f ff ff ff          callq  87a <print_char>
 95b:   bf 0a 00 00 00          mov    $0xa,%edi
 960:   e8 15 ff ff ff          callq  87a <print_char>
 965:   e8 e6 fd ff ff          callq  750 <fork@plt>
 96a:   bf 33 00 00 00          mov    $0x33,%edi
 96f:   e8 06 ff ff ff          callq  87a <print_char>
 974:   b8 00 00 00 00          mov    $0x0,%eax
 979:   48 83 c4 08             add    $0x8,%rsp
```

```
0000000000000893 <func1>:
 893:   48 83 ec 08             sub    $0x8,%rsp
 897:   bf 38 00 00 00          mov    $0x38,%edi
 89c:   81 c7 41 01 00 00       add    $0x141,%edi
 8a2:   40 0f be ff             movsbl %dil,%edi
 8a6:   e8 cf ff ff ff          callq  87a <print_char>
 8ab:   48 8d 3d 8e 07 20 00    lea    0x20078e(%rip),%rdi        # 201040 <sem>
 8b2:   e8 69 fe ff ff          callq  720 <sem_post@plt>
 8b7:   8b 05 6f 07 20 00       mov    0x20076f(%rip),%eax        # 20102c <wait_var>
 8bd:   85 c0                   test   %eax,%eax
 8bf:   74 f6                   je     8b7 <func1+0x24>
 8c1:   b8 00 00 00 00          mov    $0x0,%eax
 8c6:   48 83 c4 08             add    $0x8,%rsp
 8ca:   c3                      retq
```

Luckily, you brought your textbook with you on vacation (why ?), and were able to lookup the following function signatures:

int sem_init(sem_t **sem*, int *pshared*, unsigned int *value*);
int sem_wait(sem_t *sem);
int pthread_create(pthread_t *thread, const pthread_attr_t *attr,
void *(*start_routine) (void *), void *arg);
int pthread_join(pthread_t *thread*, void ***value_ptr*);
int sem_post(sem_t *sem);

1. Which Functions contain Loops (if any)? (1 pts)

2. Which line (instruction address) in func1 accesses a shared variable that is not a synchronization variable? (1pt)

2. Assuming that there are no errors from system calls. what can be printed when this program is run? (6 pts)

ASCII TABLE

Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char
0	0	*[NULL]*	32	20	*[SPACE]*	64	40	@	96	60	`
1	1	*[START OF HEADING]*	33	21	!	65	41	A	97	61	a
2	2	*[START OF TEXT]*	34	22	"	66	42	B	98	62	b
3	3	*[END OF TEXT]*	35	23	#	67	43	C	99	63	c
4	4	*[END OF TRANSMISSION]*	36	24	$	68	44	D	100	64	d
5	5	*[ENQUIRY]*	37	25	%	69	45	E	101	65	e
6	6	*[ACKNOWLEDGE]*	38	26	&	70	46	F	102	66	f
7	7	*[BELL]*	39	27	'	71	47	G	103	67	g
8	8	*[BACKSPACE]*	40	28	(	72	48	H	104	68	h
9	9	*[HORIZONTAL TAB]*	41	29	)	73	49	I	105	69	i
10	A	*[LINE FEED]*	42	2A	*	74	4A	J	106	6A	j
11	B	*[VERTICAL TAB]*	43	2B	+	75	4B	K	107	6B	k
12	C	*[FORM FEED]*	44	2C	,	76	4C	L	108	6C	l
13	D	*[CARRIAGE RETURN]*	45	2D	-	77	4D	M	109	6D	m
14	E	*[SHIFT OUT]*	46	2E	.	78	4E	N	110	6E	n
15	F	*[SHIFT IN]*	47	2F	/	79	4F	O	111	6F	o
16	10	*[DATA LINK ESCAPE]*	48	30	0	80	50	P	112	70	p
17	11	*[DEVICE CONTROL 1]*	49	31	1	81	51	Q	113	71	q
18	12	*[DEVICE CONTROL 2]*	50	32	2	82	52	R	114	72	r
19	13	*[DEVICE CONTROL 3]*	51	33	3	83	53	S	115	73	s
20	14	*[DEVICE CONTROL 4]*	52	34	4	84	54	T	116	74	t
21	15	*[NEGATIVE ACKNOWLEDGE]*	53	35	5	85	55	U	117	75	u
22	16	*[SYNCHRONOUS IDLE]*	54	36	6	86	56	V	118	76	v
23	17	*[ENG OF TRANS. BLOCK]*	55	37	7	87	57	W	119	77	w
24	18	*[CANCEL]*	56	38	8	88	58	X	120	78	x
25	19	*[END OF MEDIUM]*	57	39	9	89	59	Y	121	79	y
26	1A	*[SUBSTITUTE]*	58	3A	:	90	5A	Z	122	7A	z
27	1B	*[ESCAPE]*	59	3B	;	91	5B	[	123	7B	{
28	1C	*[FILE SEPARATOR]*	60	3C	<	92	5C	\	124	7C	|
29	1D	*[GROUP SEPARATOR]*	61	3D	=	93	5D	]	125	7D	}
30	1E	*[RECORD SEPARATOR]*	62	3E	>	94	5E	^	126	7E	~
31	1F	*[UNIT SEPARATOR]*	63	3F	?	95	5F	_	127	7F	*[DEL]*