Princeton University
COS 217: Introduction to Programming Systems
Spring 2003 Final Exam Answers

The exam was a three-hour exam. It was closed-book. However the students were allowed to use a self-generated one-page summary sheet.

Question 1 Part (a)

```
     Binary   Octal   Hexadecimal

30 = 11110    36      1E
17 = 10001    21      11
```

Question 1 Part (b)

```
      Signed Magnitude   1's Complement   2's Complement

 31 = 00011111           00011111         00011111
-31 = 10011111           11100000         11100001
 32 = 00100000           00100000         00100000
-32 = 10100000           11011111         11100000
```

Question 1 Part (c)

$(a - b) \bmod 2^{32}$
$= (a - b + 2^{32}) \bmod 2^{32}$
$= (a - b + 2^{32} - 1 + 1) \bmod 2^{32}$
$= (a + (2^{32} - 1 - b) + 1) \bmod 2^{32}$
$= (a + \text{onescomplement}(b) + 1) \bmod 2^{32}$
$= (a + \text{twoscomplement}(b)) \bmod 2^{32}$

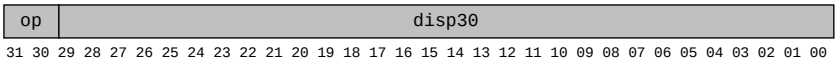
Question 2

```
!-----------------------------------------------
        .section ".rodata"
!-----------------------------------------------

pcFormat:
        .asciz "%d\n"

!-----------------------------------------------
        .section ".data"
!-----------------------------------------------

a:
        .word 1
        .word 2
        .word 3
        .word 4

!-----------------------------------------------
        .section ".text"
!-----------------------------------------------

        .global main
main:
        save %sp, -96, %sp
```

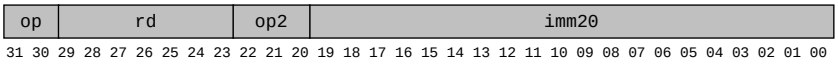
```
        ! printf("%d\n", f(4));
        mov  4, %o0
        call f
        nop
        mov %o0, %o1
        set pcFormat, %o0
        call printf
        nop
        ret
        restore

        .global f
f:
        save %sp, -96, %sp

        ! if (x <= 2) goto else1;
        cmp  %i0, 2
        ble  else1
        nop

        ! return f(x-1) + f(x-2);
        sub  %i0, 1, %o0
        call f
        nop
        mov  %o0, %l1
        sub  %i0, 2, %o0
        call f
        nop
        add  %l1, %o0, %i0
        ret
        restore

else1:

        ! return a[x];
        set  a, %l0
        sll  %i0, 2, %l1
        ld   [%l0 + %l1], %i0
        ret
        restore
```

Question 3 Part (a)

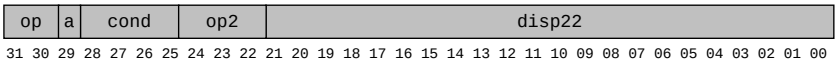
Format 1 (call)



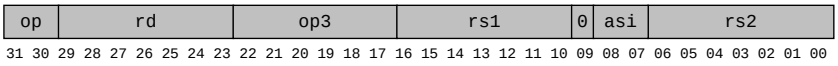
Format 2a (sethi, nop)



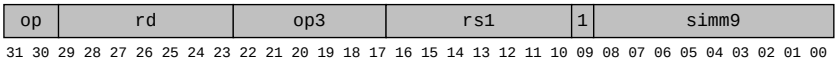
Format 2b (branches)



Format 3a (remaining instructions, three registers)



Format 3b (remaining instructions, two registers and one immediate)



Question 3 Part (b)

-256 <= N <= 255

Question 3 Part (c)

30 bits are available for the target of a call instruction. The call instruction can jump forward up to $2^{29} - 1$ instructions, and backward up to 2^{29} instructions.

Question 3 Part (d)

22 bits are available for the target of a branch instruction. The branch instruction can jump forward up to $2^{21} - 1$ instructions, and backward up to 2^{21} instructions.

Question 3 Part (e)

```
Assembly Code               Machine Language

sethi 0x12345, %r33         00 0100001 100 00010010001101000101
or    %g0, 0x67, %g1        10 0000001 000010 0000000 1 001100111
sll   %g1, 4, %g1           10 0000001 100101 0000001 1 000000100
or    %r33, %g1, %r33       10 0100001 000010 0100001 0 00 0000001
or    %r33, 0x8, %r33       10 0100001 000010 0100001 1 000001000
```

Question 4 Part (a)

Accessing data in registers is faster than accessing data in memory.

Question 4 Part (b)

```
set   0x7fffffff, %l0
addcc %l0, 1, %l0
```

Question 4 Part (c)

```
mov  16, %o0
call malloc
nop
set  4321, %l0
st   %l0, [%o0]
mov  1, %l0
st   %l0, [%o0+4]
mov  2, %l0
st   %l0, [%o0+8]
mov  3, %l0
st   %l0, [%o0+12]
```

Question 5 Part (a)

Data Section

Offset	Machine Code in Hexadecimal	Comment
0	0x43	C .ascii “COS”
1	0x4F	O
2	0x53	S
3	0x00	.skip 2
4	0x00	
5	0x00	.align 4
6	0x00	
7	0x00	
8	0x00	.word 1
9	0x00	
10	0x00	
11	0x01	
12	0x43	C .ascii “COT”
13	0x4F	O
14	0x54	T
15	0x00	.align 2
16	0x00	.half 0
17	0x00	
18	0x00	.half 3
19	0x03	
20	0x00	.half 4
21	0x04	
22	0x05	.byte 5
23	0x54	T .asciz “TON”
24	0x4F	O
25	0x4E	N
26	0x00	‘\0’

Text Section

Offset	Machine Instruction in Binary	Comment
0	10 01010 000000 01000 0 00000000 01001	add %r8, %r9, %r10
4	00 01000 100 ????????????????????	sethi %hi(var2), %r8
8	10 01000 000010 01000 1 ?????????????	or %r8, %lo(var2), %r8
12	01 00000000000000000000000000000110	call doit
16	00 0 1010 010 11111111111111111101	bg label1
20	00 10001 100 ????????????????????	sethi %hi(var1), %r17
24	10 10001 000010 10001 1 ?????????????	or %r17, %lo(var1), %r17
28	10 00000 111000 11111 1 0000000001000	jmpl %i7+8, %g0
32	11 10000 000000 01010 1 0000000001010	ld [%r10+10], %r16
36	01 ??????????????????????????????	call printf
40	10 00000 111000 01111 1 0000000001000	jmpl %o7+8, %g0

Question 5 Part (b)

Label	Section	Offset	Type	Label Sequence #
var1	data	8	local	0
label1	text	4	local	1
doit	text	36	local	2
var2	data	12	local	3
var3	data	16	local	4
printf	?	?	global	5

Question 5 Part (c)

Offset	Relocation Type	Label Sequence #
4	HI22	3
8	LO10	3
20	HI22	0
24	LO10	0
36	DISP30	5

Question 6 Part (a)

A system call executes a trap instruction to switch the CPU to supervisor mode; a user-level call does not.

Question 6 Part (b)

The operating system provides a special mechanism for system calls (that is, supervisor mode) to prohibit ordinary users from executing privileged instructions and accessing privileged memory.

Question 6 Part (c)

The stdio module buffers data for efficiency; it is faster to read and write large quantities of data infrequently rather than small quantities of data frequently.

Question 6 Part (d)

This is the FILE data type shown in the Kernighan and Ritchie textbook (p. 176):

```
typedef struct _iobuf {
   int cnt;    /* characters left */
   char *ptr;  /* next character position */
   char *base; /* location of buffer */
   int flag;   /* mode of file access */
   int fd;     /* file descriptor */
} FILE;
```

Question 6 Part (e)

This is the definition of the fopen function shown in the Kernighan and Ritchie textbook (p. 177). It is a reasonable subset of a typical "real" fopen function. A correct answer to the exam question would be some reasonable subset of the Kernighan and Ritchie function.

```
#define BUFSIZ 1024
#define OPEN_MAX 20  /* Max #files open at once */
#define PERMS 0666   /* RW for owner, group, others */

enum _flags {
   _READ  = 01,  /* file open for reading */
   _WRITE = 02,  /* file open for writing */
```

```
   _UNBUF = 04,  /* file is unbuffered */
   _EOF   = 010, /* EOF has occurred on this file */
   _ERR   = 020  /* error occurred on this file */
};

FILE _iob[OPEN_MAX] = {  /* File descriptor table */
   { 0, (char *) 0, (char *) 0, _READ, 0 },           /* stdin */
   { 0, (char *) 0, (char *) 0, _WRITE, 1 },          /* stdout */
   { 0, (char *) 0, (char *) 0, _WRITE | _UNBUF, 2 }  /* stderr */
};

#define stdin  (&_iob[0])
#define stdout (&_iob[1])
#define stderr (&_iob[2])

FILE *fopen(char *name, char *mode)
{
   int fd;
   FILE *fp;

   if (*mode != 'r' && mode != 'w' && mode != 'a')
      return NULL;

   for (fp = _iob; fp < _iob + OPEN_MAX; fp++)
      if ((fp->_flag & (_READ | _WRITE)) == 0)
         break;  /* found free slot */
   if (fp >= _iob + OPEN_MAX)  /* no free slots */
      return NULL;

   if (*mode == 'w')
      fd = creat(name, PERMS);
   else if (*mode == 'a') {
      if ((fd = open(name, O_WRONLY, 0)) == -1)
         fd = creat(name, PERMS);
      lseek(fd, 0L, 2);
   } else
      fd = open(name, O_RDONLY, 0);

   if (fd == -1)
      return NULL;

   fp->fd = fd;
   fp->cnt = 0;
   fp->base = NULL;
   fp->flag = (*mode == 'r') ? _READ : _WRITE;
   return fp;
}
```

This is the definition of the _fillbuf function given in the Kernighan and Ritchie textbook (p. 178). The fread function (shown below) calls it.

```
int _fillbuf(FILE *fp)
{
   int bufsize;

   if ((fp->flag & (_READ | _EOF | _ERR)) != _READ)
      return EOF;
   bufsize = (fp->flag & _UNBUF) ? 1 : BUFSIZE;
   if (fp->base == NULL)  /* no buffer yet */
      if ((fp->base = (char *)malloc(bufsize)) == NULL)
         return EOF;  /* can't get buffer */

   fp->ptr = fp->base;
   fp->cnt = read(fp->fd, fp->ptr, bufsize);
   if (--fp->cnt < 0) {
      if (fp->cnt == -1)
         fp->flag |= _EOF;
      else
         fp->flag |= _ERR;
```

```
      fp->cnt = 0;
      return EOF;
   }
   return (unsigned char) *fp->ptr++;
}
```

This is a definition of fread, designed so it calls the _fillbuf function shown above. A correct answer to the exam question would be a reasonable subset of this function. Note: The function also calls the standard fgetpos and fsetpos functions. Note: The function is untested.

```
size_t fread(void *ptr, size_t size, size_t nitems, FILE *fp)
{
   size_t i, j;
   fpos_t fpos;

   for (i = 0; i < nitems; i++) {
      fgetpos(fp, &fpos);  /* get the current file position */
      for (j = 0; j < size; j++) {
         if (fp->cnt == 0)  /* no bytes in the buffer */
            if (_fillbuf(fp) == EOF)  /* get more bytes */
               break;
         *ptr = *(fp->ptr);
         ptr++;
         fp->ptr++;
         fp->cnt--;
      }
      if (j < size)  /* not able to read a complete object */
         fsetpos(fp, &fpos);  /* set the file position to the spot
                                 immediately after the final object
                                 that was completely read */
   }

   return i;
}
```

Question 7 Part (a)

"Virtual memory" is the "memory" that a process accesses. The operating system maps the process's virtual memory addresses to physical memory addresses.

Question 7 Part (b)

Using virtual memory, the operating system can:

- Run many programs at once, each with its own distinct physical memory.
- Control access to physical memory so one program cannot access or modify the memory of another.
- Run a program that uses more virtual memory than the computer has available in physical memory.

Question 7 Part (c)

If virtual memory is larger than physical memory, the operating system temporarily stores some virtual memory pages on disk; "swapping" is the movement of pages between memory and disk.

Question 7 Part (d)

When a program references memory that is not available in physical memory, the operating system:

- Swaps the "victim" page from physical memory to disk.
- Changes the "victim" page's status to "invalid" in the page table.
- Swaps the desired page from disk to physical memory.
- Changes the desired page's status to "valid" in the page table.

Question 7 Part (e)

A process's "working set" is the small region of memory that it accesses frequently.

Question 7 Part (f)

"Locality of reference" is the principle that states that most memory references are nearby previous ones.

Question 7 Part (g)

"Thrashing" is the excessive swapping that occurs when the cumulative size of the working set exceeds the capacity of physical memory.

Question 7 Part (h)

Fragment #1 is likely to execute faster because it accesses the elements of the array in the order in which they are stored in memory, thus minimizing swapping.

Question 8 Part (a)

```
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char *argv[])
{
   int iBase = atoi(argv[1]);
   int iExponent = atoi(argv[2]);
   int iPower = iBase;
   int i;

   for (i = 2; i <= iExponent; i++)
      iPower *= iBase;

   printf("%d\n", iPower);
   return 0;
}
```

Question 8 Part (b)

```
#include <stdio.h>
#include <unistd.h>

#define MAX_DIGITS_PER_INT 10

int main(int argc, char *argv[])
{
   char pcBase[MAX_DIGITS_PER_INT];
   char pcExponent[MAX_DIGITS_PER_INT];
   char pcPower[MAX_DIGITS_PER_INT];
   int iPid;
   int piPipeFd[2];

   printf("Enter x:  ");
   scanf("%s", pcBase);
   printf("Enter y:  ");
   scanf("%s", pcExponent);

   pipe(piPipeFd);

   fflush(NULL);
   iPid = fork();
   if (iPid == 0)
   {
      char *ppcArgv[4];
      close(piPipeFd[0]);
      close(1);
      dup(piPipeFd[1]);
      close(piPipeFd[1]);
      ppcArgv[0] = "power";
      ppcArgv[1] = pcBase;
      ppcArgv[2] = pcExponent;
      ppcArgv[3] = NULL;
      execvp(ppcArgv[0], ppcArgv);
      perror(argv[0]);
      exit(1);
   }

   close(piPipeFd[1]);
   close(0);
   dup(piPipeFd[0]);
   close(piPipeFd[0]);
   scanf("%s", pcPower);
   wait(NULL);
   printf("x to the y power:  %s\n", pcPower);
   return 0;
}
```