CIS 2107
Computer Systems and Low-Level Programming
Fall 2011
Final

December 15, 2011

Name: _____

Page	Points	Score
1	10	
2	10	
3	9	
4	10	
6	5	
7	10	
8	10	
9	6	
10	10	
11	10	
12	10	
Total:	100	

Instructions

The exam is closed book, closed notes. You may *not* use a calculator, cell phone, etc.

For each of the questions of this quiz, you can assume the following sizes for C data types:

type	**bytes**
char	1
short	2
int	4
long	8
float	4
double	8
void*	4

CIS 2107 Summary Card 28-jan-08

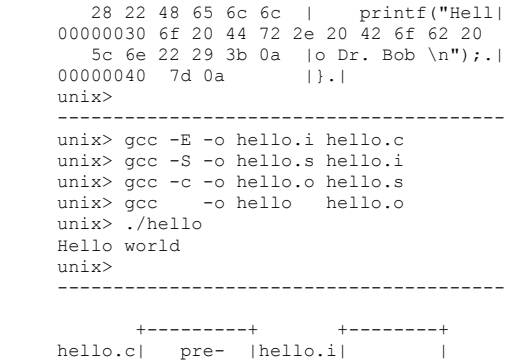
Unix

Compiling and executing a C program

```
unix> nano hello
#include <stdio.h>
int main()
{
    printf("Hello world\n");
}
^x
unix> gcc hello.cn
unix> ./a.out
Hello world
unix>
```

```
unix>hexdump -C hello.c
00000000 23 69 6e 63 6c 75 64 65 20 3c
   73 74 64 69 6f 2e  |#include <stdio.|
00000010 68 3e 0a 69 6e 74 20 6d 61 69
   6e 28 29 0a 7b 0a  |h>.int main().{.|
00000020 20 20 20 20 70 72 69 6e 74 66
   28 22 48 65 6c 6c  |    printf("Hell|
00000030 6f 20 44 72 2e 20 42 6f 62 20
   5c 6e 22 29 3b 0a  |o Dr. Bob \n");.|
00000040  7d 0a       |}.|
unix>
```

```
unix> gcc -E -o hello.i hello.c
unix> gcc -S -o hello.s hello.i
unix> gcc -c -o hello.o hello.s
unix> gcc    -o hello   hello.o
unix> ./hello
Hello world
unix>
```



gcc commands

```
gcc -E -o prog.i prog.c  preprocess only
gcc -S -o prog.s prog.c  plus compile
gcc -c -o prog.o prog.c  plus assemble
gcc    -o prog   prog.c  plus link
gcc -v prog.c            see detail
gcc -o prog one.c two.s three.o
```

Other Commands

```
hexdump -C file.c        hex dump
objdump -d prog.o        disassemble .o
objdump -d prog          disassemble exe
objdump -x prog.o        header contents
gdb prog                 debug prog
  x/20b main   examine 20 bytes of main
```

For help with any of the following, use "cmd --help" or "man cmd" or "info cmd"
as, cpp, gcc, gdb, help, info, ld, man, objdump, od

C Programming Language

Identifiers

1. Letters, numbers, underscores "_".
2. Begin with a letter (library routines use names beginning with "_").
3. Keywords reserved.
4. Declaration - interpretation of identifier, many times.
5. Definition - declares and reserves storage, only once.

keywords

```
auto      double   int       struct
break     else     long      switch
case      enum     register  typedef
char      extern   return    union
const     float    short     unsigned
continue  for      signed    void
default   goto     sizeof    volatile
do        if       static    while
```

type specifiers

```
char      int      unsigned  union
double    long     struct    unsigned
enum      short    typedef   void
float
```

Basic Data Types on x86 unix

1. char - 8-bit signed int, -128 to 127. ASCII character from 0 to 127.
2. short - 16-bit signed integer.
3. long - 32-bit signed integer.
4. int - 32-bit signed integer.
5. long long - 64-bit signed integer
6. float - 32-bit floating point number.
7. double - 64-bit floating point.
8. long double - 80-bit floating point.
9. void - nonexistant value

Data Type Modifiers

1. unsigned or signed (default)
2. const (never changed)
3. volatile (don't use optimization)

Storage Classes and Linkage of Objects
(An object is a named region of storage)

1. Internal (to a block or function)
   a. automatic - known only in block after def. No init. Discard on exit. (int i; auto int i;).
   b. static - known only in block after def. Init to zero. Preserved on exit. (static int i;)
2. External (to all blocks).
   a. external linkage (default). Known in file after def (1) or anywhere after extern (many). Init to zero. (int i; extern int i;)
   b. internal linkage using static keywork - known in file below def. or in file after extern. Init to zero. (static int i;)

Derived Types

1. arrays of objects of a given type
2. functions returning objects of type
3. pointers to objects of a given type
4. structures containing a sequence of objects of various types
5. unions containing any one of several objects of various types

Declaration examples.

```
int var;          32-bit signed int
int *var;         pointer to int
int var[3];       array of 3 ints
int *var[3];      array of 3 ptrs to ints
int *(var[3]);    array of 3 ptrs to ints
int (*var)[3];    ptr to array of 3 ints
int f(void);      function returns int
int *f(void);     function returns ptr
                     to int
int (*p)(void);   p is ptr to function
                     that returns int
void f(int *a);   arg to f is ptr to int
void f(int *);    same - "a" is a comment

int x[2][3][4]; /*definition*/
    x[1][1][1]  is int
    x[1][1]     is array of 4 ints
    x[1]        is a 3x4 array of ints
    x           is a 2x3x4 array if ints
    x[1][1], x[1], and x are pointers
```

Type names - declare an object and omit the object name. Used in casts, function declarations, and sizeof

```
type name         declaration

int               int var;
int *             int *var;
int *[3]          int *var[3];
int (*)[]         int (*var)[];
int (int)         int fun(int);
int *(void)       int *fun(void);
int (*[])(void)   int (*var[])(void);
```

Signed type conversions

```
char -> short -> long-> float -> dou
```

```
Integer constants (suffix u, U, l, I
  123    32-bit int
  123UL  32-bit unsigned long
  0100   100 octal is 64 decimal
  0x100  100 hexadecimal is 256 deci

Floating constants (suffix f, F, l,
  12.3   64-bit double
  12.3F  32-bit float
  12.3L  80-bit long double
  123e-01 == 12.3

Character and String Constants
  'x'       8-bit character constant
  'a' == '\101' == 'x41' == 61
  "string"  array of 7 char ('\0' ac

Enumeration Constants
  enum day {sun=1, mon, tues, wed}
```

C Operators in decreasing precedence

```
       () (grp)  [] (sub)
struct -> (mbr)   . (mbr)
unary   ! (not)   ~ (1's)  ++ (inc)
unary  -- (dec)   + (pos)   - (neg)
unary   * (ind)   & (adr)
unary    (type)   sizeof
arith   / (div)   * (mul)   % (mod)
arith   + (add)   - (sub)
shift  << (lft)  >> (rgt)
relat   < (ls)   <= (le)    > (gt)
relat   >= (ge)  == (eq)   != (ne)
bit     & (and)   ^ (exor)  | (or)
logic  && (and)  || (or)
if      ? (true)  : (false)
assign  =        +=        -=
assign *=        r-1       /=
assign %=        &=        ^=
assign |=       <<=       >>=
stmt    ,
```

Operators

1. struct.member, union.member
2. ptr_to_struct->member
3. ptr_to_union->member
3. struct->member <==> (*struct).mer
4. *&var <==> var
5. exp1<<exp2 <==> exp1*(2**exp2) if no overflow
6. exp1>>exp2 <==> exp1/(2**exp2) if positive and no overflow
7. for negative exp1, exp1>>exp2 may fill with 0 (unsigned) or 1 (sign

Typical Program Parts

```
#include ...;
#define ...;
function prototypes;
external declarations and definitior
int main(argc, *argv[] {...}
ret-type funct-name(arg-decl) {...}
ret-type funct-name(arg-decl) {...}
```

Typical function parts (including main)

```
ret-type funct-name(arg-decl)
{
    internal definitions and decl.;
    statement;
    statememt;
    return expression;
}
```

Composition of Declarations & Statements

```
tokens: identifiers, keywords, constants
       string literals, operators, other
white space: sp, ht, vt, nl, ff, comment
declaration: legal arrangement of tokens
statement: legal arrangement of tokens
```

Types of Statements

```
labeled-stmt.        label: stmt;
expression-stmt.     expression;
compound-statement.  {stmt; stmt; stmt;}
selection-stmt.      if, switch
iteration-stmt.      while, do, for
jump-stmt.           goto, continue,
                     break, return,
```

while, for, do, if

```
while (exp) {...}
for (exp; exp; exp) {...}
do {...} while (exp);
if (expression) {...} else {...}
exp ? nonzerostmt : zerostmt;
```

switch

```
switch (intexp) {
  case int1: stmts;
  case int2: stmts; break;
  default:   stmts;
}
```

Jump statements and statement labels

```
if (exp) goto error;
...
error: stmts;

continue;  /*end this iter; start next*/
break;     /*end iteration or switch  */
return exp;/*return to stmt after call*/
```

C Preprocessor

```
#include "filename"   /* local dir   */
#include <filename>   /* sys defined */
#define  name  replacement text
#define forever for (;;) {}
#define max(A,B) ((A) > (B) ? (A) : (B))
        max(i++, j++); /* bad idea*/
#if #ifdef #ifndef #elif #else #endif

#if !defined(ABC)
  #define ABC
#else
  #undef ABC
#endif
```

typedef - new names for old data type

```
typedef int Length; Length a, b, c;
void (*signal(int signum,
             void (*func)(int)))(int);
typedef void Sigfun(int);
Sigfun *signal(int signum, Sigfun *fun);
```

Structures and Unions

```
struct OptStructName {
   declaration1;
   declaration2;
} OptVar1, OptVar2;
struct StructName Var3, Var4;

struct point {
   int x;
   int y;
} xy, *pxy
xy.x = 5; xy.y=10;
pxy = &xy;
pxy->x = 20; pxy->y = 30;

typedef struct Flt {  /*little endian*/
  unsigned int fract:23;
  unsigned int exp:8;
  unsigned int sign:1;
} flt;
union Tstflt {
  flt  also3;
  float f;
} tstflt;
flt a3 = {0x400000,128,0};
testflt.also3=a3;
print("%12.10f\n", tstflt.f);
```

Binary Trees

```
struct tnode {
   int data;
   struct tnode *left;
   struct tnode *right;
}

struct tnode *node = (struct tnode *)
   malloc(sizeof(struct tnode));
node->data = 1;
node->left = node->right = null;

typedef struct tnode Treenode;
typedef struct tnode *Treeptr;
Treeptr node = (Treeptr)
               malloc(sizeof(Treenode));
```

Linked Lists

```
struct llist {
    struct llist *next;
    int    value;
}
for (p = head; p != NULL; p = p->next)
```

Printf format specifiers (after %)

```
i,d ints       x hex        e,f,g double
u   uint       c char       p pointer
o   octal      s string
```

Examples

```
for (i=1, j=n; i>n; ++i, --j) {...}

struct tnode *node = (struct tnode *)
   malloc(sizeof(struct tnode));

for (p = head; p != NULL; p = p->next)

int factorial(int i) {
    return i<=1 ? 1 : i*factorial(i-1);
}
```

ANSI C Standard Library

Input and Output <stdio.h>

```
FILE *f, char *s, *t, int c, i, j

f=fopen(s, t)         i=fclose(f)
i=fprintf(f, s, ...) i=printf(s, ...)
i=sprintf(s, t, ...) i=fscanf(f, s, ...)
i=scanf(s, ...)       i=sscanf(s, t, ...)
i=fgetc(f)            t=fgets(s, i, f)
i=fputc(c, f)         i=fputs(s, f)
i=getc(f)             i=getchar(void)
t=gets(s)             i=putc(j, f)
i=putchar(j)          i=puts(s)
i=ungetc(j, f)
```

Character Class Tests <ctype.h>

```
int i, char c

i=isalnum(c)  i=isalpha(c)  i=iscntrl(c)
i=isdigit(c)  i=isgraph(c)  i=islower(c)
i=isprint(c)  i=isspace(c)  i=isupper(c)
```

string functions (incomplete) <string.h>

```
char *s, *t, const char *cs, *ct
size_t n, int c, i

s=strcpy(s,ct)       s=strncpy(s,ct,n)
s=strcat(s,ct)       s=strncat(s,ct,n)
i=strcmp(cs,ct)      i=strncmp(cs,ct,n)
s=strchr(ct,c)       s=strrchr(ct,c)
s=strstr(cs,ct)      n=strlen(cs)
s=strerror(n)
```

Mathematical Functions (math.h)

```
double x, y, z, int n
z=sin(x)      z=cos(x)      z=tan(x)
z=asin(x)     z=acos(x)     z=atan(x)
z=atan2(y,x)  z=sinh(x)     z=cosh(x)
z=tanh(x)     z=exp(x)      z=log(x)
z=log10(x)    z=pow(x,y)    z=sqrt(x)
z=ciel(x)     z=floor(x)    z=fabx(x)
```

Utility Functions <stdlib.h>

```
double atof(const char *s)
int  atoi(const char *s)
long atol(const char *s)
int  rand(void)
void stand(unsigned int seed)
void *calloc(size_t nobj, size_t size)
void *malloc(size-t size)
void free(void *p)
void exit(int status)
int  abs(int n)
int  labs(long n)
```

as Assembler (info as)

AT&T vs. Intel Syntax

```
.intel_syntax noprefix
.att_syntax prefix

AT&T               Intel

push $4            push 4
clr  %eax          clr  eax
jump *%eax         jump eax
addl $4, %eax      add  eax, 4
movb movw          mov  (and examine
movl movq                operands)
movb $10,achar     mov  byte ptr acha

AT&T addr.  SEC:DISP(BASE,INDEX,SCAI
Intel addr. SEC:(BASE+INDEX*SCALE+DI
```

Assembler (as) Syntax

```
1. /* this is a "multiline" comment
2. #  this is a "line" comment
3. Statements end at '\n' or ';'
4. Symbols (A-Z), (a-z), (0-9), (_.$
5. symbol: - define symbolic address
6. .symbol - assembly directive
7. symbol  - op code or address
```

Assembler Operands

```
zero, one, and two operand instructi
  setc            # set the carry bi
  incl wage       # increment long w
  movb $5,x(%eax) # x[%eax] = 5;

operands
  1000  - mem. addr. 1000 contains c
  beta  - addr. beta contains oper.
  $55   - immediate operand 55
  %eax  - 32-bit eax reg. contains c
  (%eax)- eax contains addr. of oper
  beta(%eax,%ebx,4) - address
     beta + %eax + 4*ebx contains op
```

section - block of addresses at 0000

```
as predefined sections text, data, b
  .text      contains code
  .data      initialized data (int i
  .bss       uninitialized data (int
named sections with .section directi
  .section .rodata
  .section .note.GNU-stack,"",@progb
other sections
  undefined  value is 0, ld will def
  .comment   .ident text goes here
as address=(SECTION)+(OFFSET INTO SE
.comm   defines symbol in .bss secti
```

symbols

```
symbol=expression       #same as .se
.set symbol expression  #same as =
.Lsymbol is local symbol (ld never s
1:, 2:, 3: are local labels, 1f, 2b
"." is location counter. myadr: .lon
```

symbol attributes (plus name)

```
value                   # usually 32 bits
type includes section   # .text, etc
.type name,@function    # for functions
.type name,@object      # for data
.size name,exp          # .size myint, 4
.size main, .-main      # length of main
```

Reserving and defining storage

```
   .byte   74, 0B01001010  # 74 2 times
   .byte   0x4A, 0x4a, 'J  # 74 3 times
a: .byte   1       # char   a = 1;
b: .value  2       # short  b = 2;
c: .long   4       # int    c = 4;
d: .long   4       # long   d = 4;
e: .float  4.0     # float  e = 4.0;
f: .double 8.0     # double f = 8.0;
g: .ascii  "abc"   # char g[3]={97,98,99}
h: .asciz  "abc"   # char h[3]= "abc";
i: .=.+100         # char i[100];
j: .comm symbol,length,align #j in .bss
```

More Assembler Directives

```
.align 4          # addr%4==0
.align 4,,15      # addr%4==0, max 15
.align 4,0,15     # same, but fill with 0
.p2align 4,,7     # addr%2^4==0, max 7
.file  file.c     # may disappear
.fill  repeat, size, value
.globl symbol     # visible to ld
.ident "comment" # to comment section
```

expressions

result is abs. number or offset into section (text, data, bss, absolute)

operators -, ~, *, /, %, <<, >>, |, &, ^, !, +, -, ==, <>, !=, <, >, >=, <=, &&, || (true = -1, false = 0)

386-486-Pentium-Core 2-Athlon

386-486-Pentium "General" Registers

```
8 32-bit regs:   %eax, %ebx, %ecx, %edx,
                 %edi, %esi, %ebp, %esp
8 16-bit regs:    %ax,  %bx,  %cx,  %dx,
                  %di,  %si,  %bp,  %sp
8  8-bit regs:    %ah,  %al,  %bh,  %bl,
                  %ch,  %cl,  %dh,  %dl
1 32-bit eflags:
1 32-bit eip:
```

Additional 386-486-Pentium Registers

```
6 16-bit segment: %cs,  %ds,  %ss,  %es,
                  %fs,  %gs
4 32-bit ctrl:    %cr0, %cr1, %cr2, %cr3
6 32-bit debug:   %db0, %db1, %db2, %db3,
                  %db6, %db7
2 32-bit test:    %tr6, %tr7
8 80-bit flt pt:  %st=%st(0), %st(1),
                  %st(2), ..., %st(7)
```

AMD and Pentium 4 64-bit Registers

```
8 64-bit regs:   %rax  %rbx  %rcx  %rdx
                 %rdi  %rsi  %rbp  %rsp
8 64-bit exten:  %r8   %r9   %r10  %r11
                 %r12  %r13  %r14  %r15
8 32-bit exten:  %r8d  %r9d  %r10d %r11d
                 %r12d %r13d %r14d %r15d
8 16-bit exten:  %r8w  %r9w  %r10w %r11w
                 %r12w %r13w %r14w %r15w
8  8-bit exten:  %r8b  %r9b  %r10b %r11b
                 %r12b %r13b %r14b %r15b
8 64-bit SSE     %xmm0 %xmm1 %xmm2 %xmm3
                 %xmm4 %xmm5 %xmm6 %xmm7
```

24 General Registers

```
bit  31            16 15     8 7      0
    +----------------+--------+--------+
%eax|           %ax->|   %ah  |   %al  |
    +----------------+--------+--------+
%ebx|           %bx->|   %bh  |   %bl  |
    +----------------+--------+--------+
%ecx|           %cx->|   %ch  |   %cl  |
    +----------------+--------+--------+
%edx|           %dx->|   %dh  |   %dl  |
    +----------------+--------+--------+
%edi|                |       %di       |
    +----------------+-----------------+
%esi|                |       %si       |
    +----------------+-----------------+
%ebp|                |       %bp       |
    +----------------+-----------------+
%esp|                |       %sp       |
    +----------------+-----------------+
```

Flag Register (low order 16 bits)

```
Carry flag ----------------------------+
Parity flag ---------------------+     |
Aux carry flag ---------------+  |     |
Zero flag ----------------+   |  |     |
sign flag ---------------+|   |  |     |
trap flag ------------+  ||   |  |     |
                      |  ||   |  |     |
       15             v  vv   v  v     v
      +-----------------------------------+
      | | | | | | | | | | | | | | | | |
      +-----------------------------------+
         ^ ^ ^ ^ ^ ^   7             0
         | | | | | |
         | | | | | +-- interrupt enable
         | | | | +------ Direction flag
         | | | +--------- Overflow flag
         | +-+----------- priv level
         +--------------- nested task
```

Calling Conventions

```
caller saves %eax, %ecx, %edx if wanted
callee saves %ebx, %esi, %edi if used

fn: pushl %ebp      #save callers %ebp
    movl  %esp,%ebp #create my %ebp
    ...
    movl  %ebp,%esp #restore %esp
    popl  %ebp      #rest. callers %ebp
    ret
```

Stack Frame

```
          +----------------+
          | ...            |
          +----------------+
          |callers storage |
          +----------------+
%ebp-12 -->|my arg 2        |<--+
          +----------------+   |
%ebp-8  -->|my arg 1        |   |
          +----------------+   |
%ebp-4  -->|return to caller|  my
          +----------------+ stack
my %ebp -->|callers %ebp    | frame <--+
          +----------------+   |      |
%ebp+4  -->|my storage      |   |      |
          +----------------+   |      b
%ebp+8  -->|my storage      |<--+-%esp o
          +----------------+          o
push arg2  |callees arg 2   |<--- %esp k
          +----------------+          |
push arg1  |callees arg 1   |<--- %esp |
          +----------------+          |
call       |return addr     |<--- %esp-+
          +----------------+
push %ebp  |my saved %ebp   |<--- %esp
          +----------------+     %ebp
movl %esp, |                |
     %ebp  +----------------+
```

Prefixes

```
Section override prefixes
    `cs', `ds', `ss', `es', `fs', `gs'
Operand/Address size prefixes
    `data16', `addr16', data32, addr32
lock and wait prefixes
rep, repe, repne for string instructions
     (%ecx times)
rex family of prefixes (extended regs)
```

Instruction Abbreviations

```
X = b, w, l, or q
mem = label(%base, %index, 1|2|4|8)
%reg = one of 24 general registers
XX = bw, bl, bq, wl, wq, lq
$imm = immediate operand
```

Data Movement (source -> dest)

```
movX   %reg, %reg    movX   %reg, mem
movX    mem, %reg    movX   $imm, mem
movX   $imm, %reg

movsXX %reg, %reg    movsXX mem, %reg
movzXX %reg, %reg    movzXX mem, %reg

pushl  mem           popl   mem
pushl  %reg          popl   %reg
pushl  imm

exch   %reg,%reg     exch   mem,%reg

leal   mem,%reg
```

Convert (sign extend) instructions

```
cbtw  %al  to %ax
cwtl  %ax  to %eax
cltd  %eax to %edx,%eax
```

Classic Two Operand (source+dest->de

(add, adc, and, xor, or, sbb, sub, c

```
addX %reg,%reg       adcX %reg,%reg
addX $imm,%reg       adcX $imm,%reg
addX mem,%reg        adcX mem,%reg
addX %reg,mem        adcX %reg,mem
addX $imm,mem        adcX $imm,mem

subX %reg,%reg       sbbX %reg,%reg
subX $imm,%reg       sbbX $imm,%reg
subX mem,%reg        sbbX mem,%reg
subX %reg,mem        sbbX %reg,mem
subX $imm,mem        sbbX $imm,mem

andX %reg,%reg       orX %reg,%reg
andX $imm,%reg       orX $imm,%reg
andX mem,%reg        orX mem,%reg
andX %reg,mem        orX %reg,mem
andX $imm,mem        orX $imm,mem

xorX %reg,%reg       cmpX %reg,%reg
xorX $imm,%reg       cmpX $imm,%reg
xorX mem,%reg        cmpX mem,%reg
xorX %reg,mem        cmpX %reg,mem
xorX $imm,mem        cmpX $imm,mem

testX %reg,%reg      testX $imm,%reg
```

Classic 1 Operand (op dest->dest)

```
incX %reg            decX %reg
incX mem             decX mem

notX %reg            negX %reg
notX mem             negX mem
```

Shift and Rotate Instructions

(rol, ror, rcl, rcr, shl=sal, shr, s

If no $imm, shift amount in %cl

```
salX $imm,%reg       sarX $imm,%reg
salX $imm,mem        sarX $imm,mem

shlX $imm,%reg       shrX $imm,%reg
shlX $imm,mem        shrX $imm,mem

rolX $imm,%reg       rorX $imm,%reg
rolX $imm,mem        rorX $imm,mem

rclX $imm,%reg       rcrX $imm,%reg
rclX $imm,mem        rcrX $imm,mem
```

Unsigned Multiply and Divide

oper = %reg or mem

for divide, result is remainer,quoti

```
mulb oper   %ax       = %al * oper
mulw oper   %dx,%ax   = %ax * oper
mull oper   %edx,%eax = %eax * oper
divb oper   %ah,%al   = %ax / oper
divw oper   %dx,%ax   = %dx,%ax / op
divl oper   %edx,%eax = %edx,%eax /
```

Signed Multiply and Divide

```
oper = %reg or mem
for divide, result is remainder,quotient

imulb oper   %ax       = %al * oper
imulw oper   %dx,%ax   = %ax * oper
imull oper   %edx,%eax = %eax * oper

imulX %reg,%reg
imulX &mem,%reg
imulX $imm,%reg

idivb oper   %ah,%al  = %ax / oper
idivw oper   %dx,%ax  = %dx,%ax / oper
idivl oper  %edx,%eax = %edx,%eax / oper
```

Jump, Call, and Ret Instructions

```
jmp  label    #uses %eip relative addr.
jmp  *%reg    #jump to addr in reg
jmp  *mem     #jump to addr in mem loc

call pushes %eip (return addr) on stack
call label    #uses %eip relative addr.
call *%reg    #function addr in reg
call *mem    #function addr in mem loc
ret           #pop stack into %eip
```

Condition Codes

```
CF:  Carry flag (unsigned overflow if 1)
ZF:  Zero flag, set to 1 if result is 0
SF:  Sign flag, equals result high bit
OF:  Overflow flag (unsigned overflow)
```

Conditional Jump instructions, jCC label

```
Use after cmp, add, sub, etc. (not mov)
cmpl a,b; jg label; jumps if b-a>0 (b>a)

Simple Condition Code Tests

jo  label           jno label
jz  label           jnz label
js  label           jns label
jo  label           jn0 label

signed              unsigned

jg  (jnle) label    ja  (jnbe) label
jge (jnl)  label    jae (jnb)  label
jl  (jnge) label    jb  (jnae) label
jle (jng)  label    jbe (jna)  label
```

Set Instructions (setCC reg8)

```
setCC reg8;  movzbl reg8,reg32;

Set Individual Condition Code Bits

seto  label         setno label
setz  label         setnz label
sets  label         setns label
seto  label         setno label

signed              unsigned

setg (setnle) reg8  seta (setnbe) reg8
setge (setnl) reg8  setae (setnb) reg8
setl (setnge) reg8  setb (setnae) reg8
setle (setng) reg8  setbe (setna) reg8
```

Floating Point (8 80-bit registers)

```
float a, b, c;      double d, e, f;
a = b + c;          d = e + f;

flds  b  #s=short   fldl  e  #l = long
flds  c  #p=pop     fldl  f  #p = pop
faddp %st,%st(1)    faddp %st,%st(1)
fstps a             fstpl d

Floating Point Condition Codes. After
"fnstsw %ax; andb $69,%ah", %ah = ($0,
$1, $64 or $69) for (>, <, =, or NAN)
```

String Instructions

```
%esi points to source, %edi to dest.
%esi and %edi incremented by 1, 2, or 4
  if DF (direction flag) == 0 (cld)
  decrement if DF == 1 (std)
movs  move string,    movX (%esi),(%edi)
cmps  compare string, cmpX (%esi),(%edi)
stos  store string,   movX    %a?,(%edi)
lods  load string,    movX (%edi),%a?
scas  scan string,    cmpX (%edi),%a?

put count in %ecx and repeat instruction
using rep, repe, and repne prefix
```

Read Time Stamp Counter

```
rdtsc - cycles since boot in %edx, %ecx
```

Counting and Number Representation

Signed and unsigned 8-bit bytes

hex	binary	unsigned	2's comp
00	00000000	0	0
01	00000001	1	1
02	00000010	2	2
03	00000011	3	3
...	...	...	...
7E	01111110	126	126
7F	01111111	127	127
80	10000000	128	-128
81	10000001	129	-127
...	...	...	...
FE	11111110	254	-2
FF	11111111	255	-1

Signed and unsigned 32-bit integers

hex	unsigned	2's comp
00000000	0	0
00000001	1	1
00000002	2	2
00000003	3	3
...	...	...
7FFFFFFE	2,147,483,646	2,147,483,646
7FFFFFFF	2,147,483,647	2,147,483,647
80000000	2,147,483,648	-2,147,483,648
80000001	2,147,483,649	-2,147,483,647
...	...	...
FFFFFFFD	4,294,967,294	-3
FFFFFFFE	4,294,967,294	-2
FFFFFFFF	4,294,967,295	-1

ASCII

10	16	ch	10	16	ch	10	16	ch	10	16	ch
0	00	\0	32	20	SP	64	40	@	96	60	`
1	01	SOH	33	21	!	65	41	A	97	61	a
2	02	STX	34	22	"	66	42	B	98	62	b
3	03	ETX	35	23	#	67	43	C	99	63	c
4	04	EOT	36	24	$	68	44	D	100	64	d
5	05	ENQ	37	25	%	69	45	E	101	65	e
6	06	ACK	38	26	&	70	46	F	102	66	f
7	07	BEL	39	27	'	71	47	G	103	67	g
8	08	BS	40	28	(	72	48	H	104	68	h
9	09	HT	41	29	)	73	49	I	105	69	i
10	0A	\n	42	2A	*	74	4A	J	106	6A	j
11	0B	\v	43	2B	+	75	4B	K	107	6B	k
12	0C	\f	44	2C	,	76	4C	L	108	6C	l
13	0D	\r	45	2D	-	77	4D	M	109	6D	m
14	0E	SO	46	2E	.	78	4E	N	110	6E	n
15	0F	SI	47	2F	/	79	4F	O	111	6F	o
16	10	DLE	48	30	0	80	50	P	112	70	p
17	11	DC1	49	31	1	81	51	Q	113	71	q
18	12	DC2	50	32	2	82	52	R	114	72	r
19	13	DC3	51	33	3	83	53	S	115	73	s
20	14	DC4	52	34	4	84	54	T	116	74	t
21	15	NAK	53	35	5	85	55	U	117	75	u
22	16	SYN	54	36	6	86	56	V	118	76	v
23	17	ETB	55	37	7	87	57	W	119	77	w
24	18	CAN	56	38	8	88	58	X	120	78	x
25	19	EM	57	39	9	89	59	Y	121	79	y
26	1A	SUB	58	3A	:	90	5A	Z	122	7A	z
27	1B	ESC	59	3B	;	91	5B	[	123	7B	{
28	1C	FS	60	3C	<	92	5C	\	124	7C	|
29	1D	GS	61	3D	=	93	5D	]	125	7D	}
30	1E	RS	62	3E	>	94	5E	^	126	7E	~
31	1F	US	63	3F	?	95	5F	_	127	7F	DE

Escape sequences in ASCII

null or zero	NUL	\0	= \000 = \x00 =	0
audible alert	BEL	\a	= \007 = \x07 =	7
backspace	BS	\b	= \010 = \x08 =	8
horizontal tab	HT	\t	= \011 = \x09 =	9
newline (LF)	NL	\n	= \012 = \x0a =	10
vertical tab	VT	\v	= \013 = \x0b =	11
formfeed	FF	\f	= \014 = \x0c =	12
carriage ret	CR	\r	= \015 = \x0d =	13
double quote	"	\"	= \042 = \x22 =	34
single quote	'	\'	= \047 = \x27 =	39
question mark	?	\?	= \077 = \x3f =	63
backslash	\	\\	= \134 = \x5c =	92
octal number	ooo	\ooo		
hex number	hh	\xhh		

Powers of 2 and 16

n	2^n	16^n
0	1	1
1	2	16
2	4	256
3	8	4,096
4	16	65,536
5	32	1,048,576
6	64	16,777,216
7	128	268,435,456
8	256	4,294,967,296
9	512	68,719,476,736
10	1,024	1,099,511,627,776

Decimal-Binary-Hexadecimal (Hex)

10	2	16	10	2
0	0000	0	8	1000
1	0001	1	9	1001
2	0010	2	10	1010
3	0011	3	11	1011
4	0100	4	12	1100
5	0101	5	13	1101
6	0110	6	14	1110
7	0111	7	15	1111

Floating Point Numbers

	len	sign	exp	hide	fr
float	32	1	8	1	
double	64	1	11	1	
long double	80	1	15	-	

```
exponent bias is 127, 1023, or 16383

      31 30    23 22
      +-+--------+------------------
float |s|  exp   |1. + 23-bit fracti
      +-+--------+------------------
```

Recognizing Floating Point Numbers

```
if e==0:
    denormalized
    f=(-1)^S * 2^(1-bias)*0.M
else if e == all 1s:
    if M == 0:
        if S == 0:
            +infinity
        else:
            -infinity
    else:
        NaN
else:
     normalized
     (-1)^S * 2^(e-bias)*1.M
```

```
#define bit_get(p,m) ((p) & (m))
#define bit_set(p,m) ((p) |= (m))
#define bit_clr(p,m) ((p) &= ~(m))
#define bit_flip(p,m) ((p) ^= (m))
#define bit_write(c,p,m)
   (c ? bit_set(p,m) : bit_clear(p,m
#define BIT(x) (0x01 << (x))
#define LONGBIT(x)
  ((unsigned long)0x00000001 << (x))
```

```
struct one32bitlong  {
     unsigned bit:1;
     unsigned nibble:4;
     unsigned dozen:12;
     unsigned fifteen:15;
};
```

1. Please place the letter for each in the answer line.

A L1	H BSS	O flash	V preprocessor
B loader	I SRAM	P sector	W symbol table
C nanoseconds	J compiler	Q heap	X CPU
D cylinder	K track	R magnetic disk	Y EBP
E L2	L EAX	S assembler	Z milliseconds
F data	M seconds	T stack	
G DRAM	N instruction pointer or program counter	U locality	

(1 point) (a) The auto storage class (used for local variables in a function) uses this area of memory.

(a) _____

(1 point) (b) Translates a high level language program into assembly language.

(b) assembler

(1 point) (c) Data structure created by the assembler during pass 1.

(c) _____

(1 point) (d) Roughly one million times slower, cheaper, and larger than main memory or cache.

(d) flash

(1 point) (e) Technology used for main memory.

dram

(e) _____

(1 point) (f) Tracks are partitioned into these (typically 512 bytes each).

(f) _____

(1 point) (g) Malloc and free manage this area of memory.

(g) _____

(1 point) (h) Stores each bit in a cell composed of six transistors.

(h) _____

(1 point) (i) One of the concentric rings on the surface of a disk.

(i) _____

(1 point) (j) Technology used to create solid state disks.

(j) _____

question 1 continues . . .

(The answers are repeated at the top of this page for your convenience. They are the same as the answers on the previous page.)

A L1	H BSS	or program counter	U locality
B loader	I SRAM	O flash	V preprocessor
C nanoseconds	J compiler	P sector	W symbol table
D cylinder	K track	Q heap	X CPU
E L2	L EAX	R magnetic disk	Y EBP
F data	M seconds	S assembler	Z milliseconds
G DRAM	N instruction pointer	T stack	

(1 point) (k) Area of memory used for uninitialized global variables.

(k) _____

(1 point) (l) Tendency for programs to access multiple objects in a block.

(l) _____

(1 point) (m) Translates assembly language programs into machine language.

(m) _____

(1 point) (n) Area of memory used for initialized global variables.

(n) _____

(1 point) (o) In a collection of disk platters, a set of tracks equidistant from the center of the platter.

(o) _____

(1 point) (p) Processor register that contains the address of the next instruction to be executed.

(p) _____

(1 point) (q) Contains the return value of functions which return ints.

(q) _____

(1 point) (r) The larger but slower cache. Still much faster than main memory.

(r) _____

(1 point) (s) Time it takes to read from disk.

(s) _____

(1 point) (t) Time to read from on-CPU cache.

(t) _____

(2 points) 2. What is $111101011010_2 + 1011111011_2$ in base 2?

	1	1	1	1	0	1	0	1	1	0	1	0_2
+			1	0	1	1	1	1	1	0	1	1_2

(2 points) 3. What is $5696C1B_{16} + DA778_{16}$ in base 16?

	5	6	9	6	C	1	B_{16}
+			D	A	7	7	8_{16}

(5 points) 4. The hex value 0x411a0000 is stored in a 32-bit C float variable. What floating point number does this value represent? It's perfectly OK if your answer includes fractions. (For example, if the correct answer were 2.75, you could also write "2 and $\frac{3}{4}$".)

(10 points) 5. **Some bit operations.** If we have `char x = 0x5C, y = 0xA9;`, what is the result of the following operations? Your answer must be in the form of exactly two hex digits[1].

(a) x|y

(a) _____

(b) x||y

(b) _____

(c) x<<2

(c) _____

(d) ~x

(d) _____

(e) ~~x

(e) _____

(f) x&0x0F

(f) _____

(g) x^y

(g) _____

[1]Ignore the possibility of promotion to 32-bit ints. Behave as though we're living in the land of 8-bit arithmetic.

(h) x&&1

(h) _____

(i) -x

(i) _____

(j) x-y

(j) _____

(k) !!x

(k) _____

(l) x<<1

(l) _____

(m) x&y

(m) _____

(n) x^y^y

(n) _____

(o) x|0

(o) _____

(5 points) 6. If I have the following:

```
int main(void)
{
  int a=10;
  int b=20;

  int *p=&a;
  int *q=p;
  char *cp = (char*)q;

  (*p)--;
  q--;
  cp--;
```

and memory is laid out like this:

cp	1000	
q	1004	
p	1008	
b	1012	
a	1016	

what do you see if you print:

(a) a

(a) _____

(b) &a

(b) _____

(c) b

(c) _____

(d) p

(d) _____

(e) *p

(e) _____

(f) &p

(f) _____

(g) q

(g) _____

(h) *q

(h) _____

(i) cp

(i) _____

(j) &cp

(j) _____

points: _____
out of a possible 5

exam continues ...

(10 points) 7. Use the following code to answer the questions. Data sizes are specified on the cover of the exam.

```
1  struct Stuff {
2    int x;
3    int *p;
4    int A[10];
5  };
6
7  int main(void)
8  {
9    struct Stuff s;
10   int A[10];
11   int x, y;
12   char str[24];
13
14   x=10;
15   y=20;
16   A[0]=30;
17   strcpy(str, "almost quitting time");
18   s.x=40;
19   s.p=&y;
20   s.A[0]=50;
21
22   func01(A);
23   func02(str);
24   func03(str);
25   func04(s);
26
27   return 0;
28 }
29
30 void func01(int arr[]) {
31   arr[0]=3333;
32 }
33
34 void func02(char *s) {
35   strcpy(s, "yeah, winter break");
36 }
37
38 void func03(char *s) {
39   s = malloc(40);
40   strcpy(s, "how many more pages is this thing?");
41 }
42
43 void func04(struct Stuff s) {
44   s.x=4444;
45   *(s.p)=2222;
46   s.A[0]=5555;
47   s.p=malloc(sizeof(int));
48   *(s.p)=2020;
49 }
```

(a) How many bytes are passed to the function `func01( )`?

(a) _____

(b) How many bytes are passed to the function `func02( )`?

(b) _____

(c) How many bytes are passed to the function `func04( )`?

(c) _____

What is the value of each of the following after `func04( )` has been called?

(d) x

(d) _____

(e) y

(e) _____

(f) A[0]

(f) _____

(g) s.x

(g) _____

(h) s.A[0]

(h) _____

(i) *(s.p)

(i) _____

(j) str (What's the string?)

(j) _____

points: _____
out of a possible 10

exam continues ...

(10 points) 8. Write a C function which is passed an unsigned int x. The function returns 1 if there are an odd number of 1s in x's binary representation or 0 otherwise.

9. Given the C function:

```
int func(int x, int y) {
  int t;

  ...

  return x+y-t;
}
```

Immediately before `func( )` is called, *i.e.*, immediately before the instruction `call func`, `%ebp` contains the value 1000_{10} and `%esp` contains the value 960_{10}. Before `func( )` exits (more precisely, just before the `leave` and `return` instuctions are executed), what is:

(1 point) (a) stored in `%ebp`?

(a) _____

(1 point) (b) stored in `%esp`?

(b) _____

(1 point) (c) the location of x?

(c) _____

(1 point) (d) the location of y?

(d) _____

(1 point) (e) the most likely location of `t`?

(e) _____

(1 point) (f) the location of the return value?

(f) _____

(10 points) 10. Write a C function equivalent to the following assembly (no credit for an answer containing inline assembly).

```
1           .section .text
2           .globl mystery
3           .type mystery, @function
4   mystery:
5           pushl %ebp
6           movl  %esp, %ebp
7           xorl  %eax, %eax
8           xorl  %ecx, %ecx
9           movl  8(%ebp), %edx
10  begin:
11          cmpl 12(%ebp), %ecx
12          jge  done
13          addl (%edx, %ecx, 4), %eax
14          incl %ecx
15          jmp  begin
16  done:
17          movl  %ebp, %esp
18          popl  %ebp
19          ret
```

(10 points) 11. Implement the function `void reverse(int A[ ], int len)`, which reverses the order of `A[ ]`, an array of `len` items. Do not use the `[ ]` operator. No credit will be given for solutions which use the `[ ]` operator, or which declare `len` or more elements of temporary storage.

```
void reverse(int A[ ], int len) {
```

(10 points) 12. A common way of storing a spreadsheet is comma-separated text. For example, the following line in a spreadsheet:

apple	banana	cherry	some fruit beginning with d

could be stored as "apple, banana, cherry, some fruit beginning with d". Write the function `char **split(char *s)` which is passed `s`, which is a string of comma-separated values, and returns an array of string containing the values in the line terminated by a NULL pointer. Using our current example, we'd return:

w[0]	apple
w[1]	banana
w[2]	cherry
w[3]	some fruit beginning with d
w[4]	`NULL`

`split( )` should return `NULL` on failure. Hint: if there are n commas in `s`, there will be $n + 1$ words. You may use any function in the Standard C Library.

points: _____
out of a possible 10

question 12 continues . . .

(extra space)