Full Name:_____

Mock Final Exam

Instructions:

- This is a mock final exam.
- This exam is OPEN BOOK. You may use any books or notes you like. However, laptop usage is not permitted.

Problem 1. (xxx points):

Assume we are running code on a 6-bit machine using two's complement arithmetic for signed integers. A "short" integer is encoded using 3 bits. Fill in the empty boxes in the table below. The following definitions are used in the table:

```
short sy = -3;
int y = sy;
int x = -17;
unsigned ux = x;
```

Note: You need not fill in entries marked with "–".

Expression	Decimal Representation	Binary Representation
Zero	0	
–	-6	
–		01 0010
ux		
y		
TMax		
$-$TMin		

Problem 2. (xxx points):

Consider the source code below, where `M` and `N` are constants declared with `#define`.

```
int mat1[M][N];
int mat2[N][M];

int sum_element(int i, int j)
{
  return mat1[i][j] +  mat2[i][j];
}
```

A. Suppose the above code generates the following assembly code:

```
sum_element:
        pushl %ebp
        movl %esp,%ebp
        movl 8(%ebp),%eax
        movl 12(%ebp),%ecx
        sall $2,%ecx
        leal 0(,%eax,8),%edx
        subl %eax,%edx
        leal (%eax,%eax,4),%eax
        movl mat2(%ecx,%eax,4),%eax
        addl mat1(%ecx,%edx,4),%eax
        movl %ebp,%esp
        popl %ebp
        ret
```

What are the values of `M` and `N`?

`M` =

`N` =

Problem 3. (xxx points):

Consider the following C functions and assembly code:

```
int fun1(int a, int b)
{
    if (a < b)
        return a;
    else
        return b;
}

int fun2(int a, int b)
{
    if (b < a)
        return b;
    else
        return a;
}
```

```
    pushl %ebp
    movl %esp,%ebp
    movl 8(%ebp),%edx
    movl 12(%ebp),%eax
    cmpl %eax,%edx
    jge .L9
    movl %edx,%eax
.L9:
    movl %ebp,%esp
    popl %ebp
    ret
```

Which of the functions compiled into the assembly code shown?

Problem 4. (xxx points):

This next problem will test your understanding of stack frames. It is based on the following recursive C function:

```
int silly(int n)
{
   volatile int v;
   v = n * 2 + 1;
   return v;

}
```

This yields the following machine code:

```
silly:
   pushl %ebp
   movl  %esp, %ebp
   subl  $16, %esp
   pushl %ebx
   movl  8(%ebp), %ebx
   addl  %ebx, %ebx
   addl  $1, %ebx
   movl  %ebx, -4(%ebp)
   movl  -4(%ebp), %eax
   movl -20(%ebp), %ebx
   movl %ebp, %esp
   popl %ebp
   ret
```

A. Is the variable `v` stored on the stack? If so, at what byte offset (relative to `%ebp`) is it stored?

B. Is the function argument `n` stored on the stack? If so, at what byte offset (relative to `%ebp`) is it stored?

C. What (if anything) is stored at `-20(%ebp)`? If something is stored there, why is it necessary to store it?

D. What (if anything) is stored at `-8(%ebp)`? If something is stored there, why is it necessary to store it?

Problem 5. (xxx points):

After watching the presidential election you decide to start a business in developing software for electronic voting. The software will run on a machine with a 1024-byte direct-mapped data cache with 64 byte blocks. You are implementing a prototype of your software that assumes that there are 7 candidates. The C-structures you are using are:

```
struct vote {
    int candidates[7];
    int valid;
};

struct vote vote_array[16][16];
register int i, j, k;
```

You have to decide between two alternative implementations of the routine that initializes the array `vote_array`. You want to choose the one with the better cache performance.
You can assume:

- `sizeof(int) = 4`
- `vote_array` begins at memory address 0
- The cache is initially empty.
- The only memory accesses are to the entries of the array `vote_array`. Variables `i`, `j` and `k` are stored in registers.

A. What percentage of the writes in the following code will miss in the cache?

```
for (i=0; i<16; i++){
    for (j=0; j<16; j++) {
        vote_array[i][j].valid=0;
    }
}

for (i=0; i<16; i++){
    for (j=0; j<16; j++) {
        for (k=0; k<7; k++) {
            vote_array[i][j].candidates[k] = 0;
        }
    }
}
```

Overall miss rate for writes to `vote_array`: _____ %

B. What percentage of the writes in the following code will miss in the cache?

```
for (i=0; i<16; i++){
    for (j=0; j<16; j++) {
        for (k=0; k<7; k++) {
            vote_array[i][j].candidates[k] = 0;
        }
        vote_array[i][j].valid=0;
    }
}
```

Miss rate for writes to `vote_array`: _____ %

Problem 6. (xxx points):

Consider the C program below. (For space reasons, we are not checking error return codes, so assume that all functions return normally.)

```
main() {

  if (fork() == 0) {
    if (fork() == 0) {
      printf("3");
    }
    else {
      pid_t pid; int status;
      if ((pid = wait(&status)) > 0) {
        printf("4");
      }
    }
  }
  else {
    if (fork() == 0) {
      printf("1");
      exit(0);
    }
    printf("2");
  }

  printf("0");

  return 0;
}
```

Out of the 5 outputs listed below, circle only the valid outputs of this program. Assume that all processes run to normal completion.

A. 2030401 B. 1234000 C. 2300140

D. 2034012 E. 3200410

Problem 7. (xxx points):

Consider the following C program. (For space reasons, we are not checking error return codes. You can assume that all functions return normally.)

```
int val = 10;

void handler(sig)
{
    val += 5;
    return;
}

int main()
{
    int pid;

    signal(SIGCHLD, handler);
    if ((pid = fork()) == 0) {
        val -= 3;
        exit(0);
    }
    waitpid(pid, NULL, 0);
    printf("val = %d\n", val);
    exit(0);
}
```

What is the output of this program? `val = _____`

Problem 8. (xxx points):

Consider an allocator that uses an implicit free list. Each memory block, either allocated or free, has a size that is a multiple of eight bytes. Thus, only the 29 higher order bits in the header and footer are needed to record block size, which includes the header and footer and is represented in units of bytes. The header (or footer) is 4-byte in size. The usage of the remaining 3 lower order bits is as follows:

- `bit 0` indicates the use of the current block: 1 for allocated, 0 for free.
- `bit 1` indicates the use of the previous adjacent block: 1 for allocated, 0 for free.
- `bit 2` is unused and is always set to be 0.

Five helper routines are defined to facilitate the implementation of `free(void *p)`. The functionality of each routine is explained in the comment above the function definition. Fill in the body of the helper routines the code section label that implement the corresponding functionality correctly.

```
/* given a pointer p to an allocated block, i.e., p is a
   pointer returned by some previous malloc()/realloc() call;
   returns the pointer to the header of the block*/
void * header(void* p)
{
  void *ptr;

  _______;
  return ptr;
}

A. ptr=p-1
B. ptr=(void *)((int *)p-1)
C. ptr=(void *)((int *)p-4)


/* given a pointer to a valid block header or footer,
   returns the size of the block */
int size(void *hp)
{
  int result;

  _______;
  return result;
}

A. result=(*hp)&(~7)
B. result=((*(char *)hp)&(~5))<<2
C. result=(*(int *)hp)&(~7)
```

```
/* given a pointer p to an allocated block,i.e. p is
   a pointer returned by some previous malloc()/realloc() call;
   returns the pointer to the footer of the block*/
void * footer(void *p)
{
  void *ptr;

  _______;
  return ptr;
}
```

A. ptr=p+size(header(p))-8
B. ptr=p+size(header(p))-4
C. ptr=(int *)p+size(header(p))-2

```
/* given a pointer to a valid block header or footer,
   returns the usage of the currect block,
   1 for allocated, 0 for free */
int allocated(void *hp)
{
  int result;

  ______;
  return result;
}
```

A. result=(*(int *)hp)&1
B. result=(*(int *hp)&0
C. result=(*(int *)hp)|1

```
/* given a pointer to a valid block header,
   returns the pointer to the header of previous block in memory */
void * prev(void *hp)
{
  void *ptr;

  ______;
  return ptr;
}
```

A. ptr = hp - size(hp)
B. ptr = hp - size(hp-4)
C. ptr = hp - size(hp-4) + 4

Problem 9. (xxx points):

Recall that the readers-writers problem permits multiple readers to perform reading simultaneously but a writer will exclude all other readers and writers. Below is a code snippet that attempts to solve the readers-writers problem. There are multiple reader threads, each of which invokes `do_read` function to perform some reading. There are also multiple writer threads, each of which invokes `do_write` function to perform some writing.

```
1: int readcnt; /*initialized to 0*/
2: sem_t mutex, w; /*Both initialized to 1*/
3:
4: void do_read()
5: {
6:   P(&mutex);
7:   readcnt++;
8:   V(&mutex);
9:   if (readcnt == 1)
10:     P(&w);
11:
12:  /* do some reading */
13:
14:  P(&mutex);
15:  readcnt--;
16:  V(&mutex);
17:  if (readcnt == 0)
18:     V(&w);
19:}
20:
21: void do_write(void)
22:{
23:  P(&w);
24:
25:  /*do some writing*/
26:
27:  V(&w);
28:}
```

Will the above code allow readers to perform reading while some writer is concurrently doing writing?

A. yes.
B. no.

If your answer is yes, give an example when this scenario happens.