Name: _____

Please put your name on all pages; we might split the pages!

Instructions:

1. This is an open-book, open-anything, but not open-friends exam! You are free to discuss the questions and ideas for answers with your friends and classmates, but make sure to answer the questions on your own.
2. You must **download** the exam today, **Thursday, 3-may-2012, before midnight!** At exactly midnight the exam document will disappear!
3. My preferred format is a typed document. You can submit this MS Word document with your answers written in, or a PDF if you prefer. If you absolutely have to handwrite some of the answers, it would be best if you scan and submit electronically.
4. The exam is **due** no later than **Sunday, 6-may-2012, 11:59 pm**.
5. **`submit haim 91-203-s2012-final <your exam file>`**.

1 (12)	
A	/6
B	/6
2	/10
3 (30)	
3.1	/10
3.2	/10
3.3	/10
4 (12)	
As0	/3
As1	/3
As2	/3
As3	/3
5 (36)	
Fun1	/3
Fun2	/3
Fun3	/3
As1	/3
Fun4	/3
Fun5	/3
Fun6	/3
As2	/3
Fun7	/3
Fun8	/3
Fun9	/3
As3	/3

Name: _____

Please put your name on all pages; we might split the pages!

Problem 1. (12 points):

Consider the following datatype definitions on an IA32 (x86) machine.

```
typedef struct {
   char c;
   short s;
   int i;
   float f;
   double *p;
}struct1;
```

```
typedef union {
   char c;
   short s;
   int i;
   float f;
   double *p;
}union1;
```

Using the template below (allowing a maximum of 32 bytes), indicate the allocation of data for a structure of type (A) `struct1` and (B) `union1`. Mark off and label the areas for each individual element using its designated symbol (there are 5 of them -- {c, p, i, d, s}). Mark with x the parts that are allocated, but not used (to satisfy alignment). Assume the alignment rule (Section 3.9.3, p. 248 in the text): data types of size *x* must be aligned on *x*-byte boundaries. Clearly indicate the right hand boundary of the data structure with a vertical line.

(A – 6 points)

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X													

How many bytes are allocated for an object of type `struct1`? **19 bytes**

Explain!: structs allocate the sum of the storage requirements of all fields within the struct

(B – 6 points)

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
X	X	X	X	X	X	X	X																								

How many bytes are allocated for an object of type `union1`? **8 bytes**

Explain!: unions allocate only enough bytes for the largest member within the field, in this case double which is 8 bytes long

Name: _____

Please put your name on all pages; we might split the pages!

Problem 2. (10 points):

Consider the following assembly representation of a function `foo` containing a `for` loop:

```
foo:                        // your comments here!
    pushl %ebp              // pushes %ebp onto the stack
    movl %esp,%ebp          // sets %ebp = %esp
    pushl %ebx              // pushes %ebx onto the stack
    movl 8(%ebp),%ebx       // %ebx = %ebp + 8
    leal 20(%ebx),%edx      // sets %edx to the address of %ebx +
                            // 20
    xorl %ecx,%ecx          // sets %ecx to zero by comparing it to
                            // itself
    cmpl %ebx,%ecx          // checks to see if %ecx > %ebx
    jge .L4                 // jumps to .L4 if greater or equal
.L6:
    leal 15(%ecx,%edx),%edx   // %edx = 15 + %ecx + %edx
    leal 13(%ecx),%eax        // %eax = %ecx + 13
    imull %eax,%edx           // %edx = (15 + %ecx + %edx) *
                              // (%ecx + 13 ) or %edx = %edx *
                              // %eax
    incl %ecx                 // ++%ecx
    cmpl %ebx,%ecx            // checks to see if %ecx > %ebx
    jl .L6                    // redoes loop if %ecx is less
                              // than %ebx
.L4:
    movl %edx,%eax            // sets %eax = %edx
    popl %ebx                 // pops %ebx from the stack
    movl %ebp,%esp            // sets %esp = %ebp
    popl %ebp                 // pops %ebp from the stack
    ret                       // returns to foo
```

1. Comment next to each assembly line what it does;
2. Fill in the blanks to provide the functionality of the loop:

```
int foo(int a)
{
    int i;
    int result = a;
    for( i = 0; i > a; i++ ) {
    ____________________;
    ____________________;
    }
    return result;

}
```

Name: _____

Please put your name on all pages; we might split the pages!

Problem 3 (30 Points)

(Please read carefully through the end before starting!) Ok, I've told you plenty times that I didn't like the confusing stack addressing scheme on the IA32.

1. (10 points) Design a stack to my liking: in this stack, you would go "up" by adding, "down" by subtracting. You will need to allocate enough space for at least a few frames (see below for the content of each frame) so that you don't run into "stack overflow" problems.
2. (10 points) Write a new pair of `push/pop` procedures to manage this new stack.
3. (10 points) Write and new pair of `call/return` procedures. Our pair will **always** save/restore **all** registers.

pushl *source*

```
addl -4, %esp; //moves stack pointer to make room
movl source, (%esp); //moves source to the top of the stack
```

popl *destination*

```
movl (%esp), desination; //move top to destination
add 4, %esp; //decrement stack pointer
```

call *label*

```
pushl %ebp
movl %esp, %ebp
pushl %ebx  // ret address
push all registers in order
jmp label
```

return

```
popl all registers in reverse order
movl -4(%ebp), %ebx
movl %ebp, %esp
popl %ebp
jmp (%ebx) //return
```

Name: _____

Please put your name on all pages; we might split the pages!

Problem 4. (12 points – 3 points each):

Match each of the assembler routines on the left with the equivalent C function on the right: write in the appropriate box on the left the number `n` of the matching `cn` on the right).

```
as0: [=c4]
    pushl %ebp
    movl %esp,%ebp
    movl 8(%ebp),%eax
    addl $32,%eax
    testl %eax,%eax
    jge .L4
    sarl $5, %eax
.L4:
    subl $5,%eax
    movl %ebp,%esp
    popl %ebp
ret
```

```
as1: [=c3]
    pushl %ebp
    movl %esp,%ebp
    movl 8(%ebp),%eax
    sall $5,%eax
    testl %eax,%eax
    jge .L4
    addl 8(%ebp),%eax
.L4:
    movl %ebp,%esp
    popl %ebp
    ret
```

```
as2: [=c6]
    pushl %ebp
    movl %esp,%ebp
    movl 8(%ebp),%eax
    testl %eax,%eax
    jge .L4
    addl $32,%eax
.L4:
    sarl $5,%eax
    movl %ebp,%esp
    popl %ebp
    ret
```

```
as3: [=c7]
    pushl %ebp
    movl %esp,%ebp
    movl 8(%ebp),%eax
```

```
int c1(int x)
{
   return 32 * x;
}

int c2(int x)
{
   return 33 * x;
}

int c3(int x)
{
   return (x + 32)/3
}


int c4(int x)
{
   return (x + 32)-5
}

int c5(int x)
{
   return x/32;
}

int c6(int x)
{
   return (x + 32)/3
}

int c7(int x)
{
   return (x>>32)+5;
}

int c8(int x)
{
   return (x>>32);
}
```

Name: _____

Please put your name on all pages; we might split the pages!

`shrl $32,%eax`
`addl $5, %eax`
`movl %ebp,%esp`
`popl %ebp`
`ret`	

Name: _____

Please put your name on all pages; we might split the pages!

Problem 5. (36 points – 3 points each):

For each function/routine (C or Assembly) below write the most computationally efficient equivalent (Assembly or C) routine/function.

C	Assembly

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

```
fun1:
        pushq   %rbp
        movq    %rsp, %rbp
        movl    %edi, -4(%rbp)
        movl    %esi, -8(%rbp)
        movl    -4(%rbp), %eax
        cmpl    -8(%rbp), %eax
        jge     .L1
        cmpl    $0, -4(%rbp)
        jne     .L2
.L1:
        movl    -4(%rbp), %eax
        movl    %eax, -12(%rbp)
        jmp     .L3
.L2:
        movl    -8(%rbp), %eax
        movl    %eax, -12(%rbp)
.L3:
        movl    -12(%rbp), %eax
        leave
        ret
```

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

```
fun2:
        pushq   %rbp
        movq    %rsp, %rbp
        movl    %edi, -4(%rbp)
        movl    %esi, -8(%rbp)
        movl    -8(%rbp), %eax
        cmpl    -4(%rbp), %eax
        jge     .L1
        cmpl    $0, -8(%rbp)
        je      .L1
        movl    -8(%rbp), %eax
        movl    %eax, -12(%rbp)
        jmp     .L2
.L1:
        movl    -4(%rbp), %eax
        movl    %eax, -12(%rbp)
.L2:
        movl    -12(%rbp), %eax
        leave
```

Name: _____

Please put your name on all pages; we might split the pages!

	ret

int fun3(int a)
{
  unsigned ua = (unsigned) a;
  if (a >= 0)
  return a;
  else
  return ua;
}
	fun3:
	  pushq %rbp
	  movq %rsp, %rbp
	  movl %edi, -20(%rbp)
	  movl -20(%rbp), %eax
	  movl %eax, -4(%rbp)
	  cmpl $0, -20(%rbp)
	  js .L1
	  movl -20(%rbp), %eax
	  movl %eax, -24(%rbp)
	  jmp .L2
	.L1:
	  movl -4(%rbp), %eax
	  movl %eax, -24(%rbp)
	.L2:
	  movl -24(%rbp), %eax
	  leave
	  ret

int IAfun1(int a, int b)
{
  if(a > b)
  return a;
  else
  return b;
}
	pushl %ebp
	movl %esp,%ebp
	movl 8(%ebp),%edx
	movl 12(%ebp),%eax
	cmpl %eax,%edx
	jle .L9
	movl %edx,%eax
	.L9:
	movl %ebp,%esp
	popl %ebp
	ret

int fun4(int *ap, int *bp)
{
  int a = *ap;
  int b = *bp;
 	
  return (a+3*b+b*b);
}
	fun4:
	  pushq %rbp
	  movq %rsp, %rbp
	  movq %rdi, -24(%rbp)
	  movq %rsi, -32(%rbp)
	  movq -24(%rbp), %rax
	  movl (%rax), %eax
	  movl %eax, -8(%rbp)
	  movq -32(%rbp), %rax
	  movl (%rax), %eax
	  movl %eax, -4(%rbp)

Name: _____

Please put your name on all pages; we might split the pages!

	movl -4(%rbp), %eax
	addl $3, %eax
	imull -4(%rbp), %eax
	addl -8(%rbp), %eax
	leave
	ret

```
int fun5(int *ap, int *bp)
{
   int b = *bp;
   *bp = *bp + *ap;
   return a+b;
}

/* went off of something more
like this*/
int fun5(int *ap, int *bp)
{
   int b = *bp;
   int a = *bp + *ap;
   return a+b;
}
```

```
fun5:
        pushq   %rbp
        movq    %rsp, %rbp
        movq    %rdi, -24(%rbp)
        movq    %rsi, -32(%rbp)
        movq    -32(%rbp), %rax
        movl    (%rax), %eax
        movl    %eax, -8(%rbp)
        movq    -32(%rbp), %rax
        movl    (%rax), %edx
        movq    -24(%rbp), %rax
        movl    (%rax), %eax
        leal    (%rdx,%rax), %eax
        movl    %eax, -4(%rbp)
        movl    -8(%rbp), %edx
        movl    -4(%rbp), %eax
        addl    %edx, %eax
        leave
        ret
```

```
int fun6(int *ap, int *bp)
{
   int a = *ap;
   *bp = *bp - *ap;
   return a-b;
}

/* went off of something more
like this*/
int fun6(int *ap, int *bp)
{
   int a = *ap;
   itn b = *bp - *ap;
   return a-b;
}
```

```
fun6:
        pushq   %rbp
        movq    %rsp, %rbp
        movq    %rdi, -24(%rbp)
        movq    %rsi, -32(%rbp)
        movq    -24(%rbp), %rax
        movl    (%rax), %eax
        movl    %eax, -8(%rbp)
        movq    -32(%rbp), %rax
        movl    (%rax), %edx
        movq    -24(%rbp), %rax
        movl    (%rax), %eax
        movl    %edx, %ecx
        subl    %eax, %ecx
        movl    %ecx, %eax
        movl    %eax, -4(%rbp)
        movl    -4(%rbp), %edx
        movl    -8(%rbp), %eax
```

Name: _____

Please put your name on all pages; we might split the pages!

	`subl %edx, %eax`
	`leave`
	`ret`

	`pushl %ebp`
	`movl %esp,%ebp`
	`movl 8(%ebp),%edx`
	`movl 12(%ebp),%eax`
	`movl %ebp,%esp`
	`movl (%edx),%edx`
	`movl (%edx),%edx`
	`addl %edx,(%eax)`
	`movl %edx,%eax`
	`popl %ebp`
	`ret`

`int fun7(int a)`
`{`
`  return a * 17;`
`}`
	`fun7:`
	`  pushq %rbp`
	`  movq %rsp, %rbp`
	`  movl %edi, -4(%rbp)`
	`  movl -4(%rbp), %edx`
	`  movl %edx, %eax`
	`  sall $4, %eax`
	`  addl %edx, %eax`
	`  leave`
	`  ret`

`int fun8(int a)`
`{`
`  return a * 34;`
`}`
	`fun8:`
	`  pushq %rbp`
	`  movq %rsp, %rbp`
	`  movl %edi, -4(%rbp)`
	`  movl -4(%rbp), %eax`
	`  leal (%rax,%rax), %edx`
	`  movl %edx, %eax`
	`  sall $4, %eax`
	`  leal (%rdx,%rax), %eax`
	`  leave`
	`  ret`

`int fun9(int a)`
`{`
`  return a * 68;`
	`fun9:`
	`  pushq %rbp`
	`  movq %rsp, %rbp`

Name: _____

Please put your name on all pages; we might split the pages!

}	movl %edi, -4(%rbp)
	movl -4(%rbp), %eax
	leal 0(,%rax,4), %edx
	movl %edx, %eax
	sall $4, %eax
	leal (%rdx,%rax), %eax
	leave
	ret

	pushl %ebp
	movl %esp,%ebp
	movl 8(%ebp),%eax
	sall $5,%eax
	subl 8(%ebp),%eax
	addl %eax,%eax
	sarl $1, %eax
	movl %ebp,%esp
	popl %ebp
	ret