ECE 2035 A	Programming Hw/Sw Systems	Spring 2015
5 problems, 8 pages	Final Exam	29 April 2015

Instructions: This is a closed book, closed note exam. Calculators are not permitted. If you have a question, raise your hand and I will come to you. Please work the exam in pencil and do not separate the pages of the exam. For maximum credit, show your work.
Good Luck!

Your Name (***please print***) ____

This exam will be conducted according to the Georgia Tech Honor Code. I pledge to neither give nor receive unauthorized assistance on this exam and to abide by all provisions of the Honor Code.

Signed ____

1	2	3	4	5	6	total
20	20	24	21	20	40	145

Problem 1 (20 points) **Optimization**

Perform at least **five** standard compiler optimizations on the following C code fragment by writing the optimized version (in C) to the right. *Assume* ***f*** *and* ***g*** *are pure functions that each return an integer with no side effects to other data structures.*

```
int mycode(int w, int z) {
  int x = 256;
  int y = 1;
  while (y<x+z)                              ⇨
    {
      if (x)
        z = f(w*x, y, z+w*x);
      else
        z = g(z+w*x, y, w*x);
      printf("y:%d, z:%d\n",y,z);
      y += z;
    }
  while (x>0)
    printf(%d\n", g(y, --x, z);
  return y;
}
```

Briefly describe which standard compiler optimizations you applied:

1.

2.

3.

4.

5.

Problem 2 (2 parts, 20 points) **Conditionals: Compound Predicates**

Part A (8 points) Consider the following MIPS code fragment. The comment indicates which variable each register holds. These variables are of type `int` and are initialized elsewhere.

Label	Instruction	Comment
		`# $2: I, $3: C, $9: Count, $8: temp`
	`slt  $8, $3, $0`	
	`bne  $8, $0, Next`	
	`slti $8, $3, 26`	
	`beq  $8, $0, Next`	
	`addi $9, $9, 1`	
`Next:`	`addi $2, $2, 1`	

What is the equivalent C code fragment? For maximum credit, use a compound logical predicate wherever possible.

Part B (12 points) Turn this C code fragement into the equivalent MIPS code. Assume $1 holds A, $2 holds B, $3 holds C and $4 holds D. *For maximum credit, include comments and use a minimal number of instructions.*

```
if (A && B)
  C = C | D;
else
  C = C & D;
D = C * 8;
```

Label	Instruction	Comment

Problem 3 (3 parts, 24 points) **Associative Sets and 3D Arrays**

Part A (8 points) Suppose we have an associative set of **125** (key, value) pairs implemented as a **sorted singly linked list**. An application performs **1500** lookups of various keys: **1200** of the lookups find the key in the list and **300** lookups fail to find the key. The keys that are found are distributed throughout the list so that each position is equally likely to be where a key is found.

What is the average number of key comparisons that would be needed for a lookup in this list implementation? (Show work. Note: you may not have to use all data provided.)

number of comparisons: ____________________________

Part B (8 points) Suppose the associative set is reimplemented as an **open hash table**. The same **125** (key, value) pairs are stored in the hash table and are evenly distributed across **25** buckets, each implemented as an **unsorted singly linked list**. An application performs the same **1500** lookups in which **1200** find the key being searched for and **300** do not. The keys that are found are distributed throughout the bucket lists so that each bucket and each position in the bucket lists is equally likely to be where a key is found.

What is the average number of key comparisons that would be needed for a lookup in this hash table implementation? (Show work. Note: you may not have to use all data provided.)

number of comparisons: ____________________________

Part C (8 points) Suppose we have a video snippet containing `L` image frames, where each frame has width `w` and height `h` pixels. Complete the following procedure which sets a pixel at position `(x, y)` in frame number `f` to `Color`, where `y` gives the row and `x` gives the column, with (0, 0) at the top lefthand corner of the image frame, as in Project 3. Assume `L`, `w` and `h` are globally defined. `VideoPixels` is a pointer to the base of the video pixel array containing all `L` image frames in a contiguous linear sequence starting with the first pixel in the first row of frame 0 and ending with the last pixel in the last row of frame `L-1`.

```
void SetPixel(int x, int y, int f, uint32_t* VideoPixels, uint32_t Color){



}
```

Problem 4 (4 parts, 21 points) **Garbage Collection**

Below is a snapshot of heap storage. Values that are pointers are denoted with a "$". The heap pointer is **$6188**. The heap has been allocated contiguously beginning at **$6000**, with no gaps between objects.

addr	value	addr	value	addr	value	addr	value	addr	value	addr	value
6000	8	6032	12	6064	0	6096	16	6128	12	6160	0
6004	33	6036	28	6068	4	6100	$6052	6132	$6120	6164	0
6008	$6132	6040	$6120	6072	$6132	6104	$6016	6136	$6016	6168	16
6012	16	6044	80	6076	8	6108	5	6140	72	6172	$6016
6016	$6100	6048	16	6080	24	6112	148	6144	20	6176	0
6020	$6172	6052	0	6084	$6172	6116	8	6148	6046	6180	0
6024	25	6056	$6100	6088	4	6120	32	6152	8	6184	0
6028	30	6060	0	6092	80	6124	$6080	6156	26	6188	0

Part A (10 points) Suppose the stack holds a local variable whose value is the memory address **$6080**. No other registers or static variables currently hold heap memory addresses. List the addresses of all objects in the heap that are *not* garbage.

Addresses of

Non-Garbage Objects: ______________________________

Part B (3 points) If a reference counting garbage collection strategy is being used, what would be the reference count of the object at address **$6016**?

Reference count of object at $6016 = ______________________________

Part C (5 points) If the local variable whose value is the address $**6080** is popped from the stack, which addresses from Part A will be reclaimed by mark and sweep garbage collection strategy, but ***not*** by a reference counting strategy? If none, write "none."

Addresses: ______________________________

Part D (3 points) What benefit does old-new space (copying) garbage collection provide that a mark and sweep garbage collection strategy does not provide?

Benefit: ______________________________

Problem 5 (2 parts, 20 points) **MIPS and C programming**

Part A (5 points) Write a single MIPS instruction that is equivalent to the following MIPS fragment.

Original:	Equivalent MIPS statement:
`addi $1, $0, 0xFF`	
`sll $1, $1, 16`	
`lw $4, 0($8)`	
`and $4, $1, $4`	
`srl $4, $4, 16`	

Part B (15 points) Consider a singly linked list whose elements are `Student_t` structs defined as:

```
typedef struct STUDENT
{
  struct STUDENT* next; // Next pointer for linked list
  char* fname;
  char* mname;
  char* lname;
  double average;
  char  letterGrade;
} Student_t;

Student_t* head;
Student_t* tail;
```

The global variables `head` and `tail` are initially `NULL` and they hold the head and tail of the list, respectively. Complete the C function `AddToList` below that adds the student record `s` to the end of the linked list pointed to by `head` and `tail`. This list might or might not be empty. Be sure to update `head` and `tail` properly. (The list is unsorted.)

```
void AddToList(Student_t* s)
{




}
```

Problem 6 (40 points) Activation Frames

The function `Bar` (below left) calls function `Foo` after completing `code block 1`. Write MIPS assembly code that properly calls `Foo`. Include all instructions between `code block 1` and `code block 2`. Symbolically label all required stack entries and give their values if they are known (below right).

```
int Bar() {
   int         A[] = {25, 36, 49};
   int         B =  3;
   int         *P;

   (code block 1)

   P = &B;
   A[2] = Foo(A, P, *P);

   (code block 2)

}
```

Bar's FP 9900	XXX	XXX
9896	A[2]	49
9892	A[1]	36
9888	A[0]	25
9884	B	3
SP 9880	P	
9876		
9872		
9868		
9864		
9860		
9856		

label	instruction	comment
		# compute &B
		# update P
		# allocate activation frame
		# preserve bookkeeping info
		# push inputs
	jal Foo	# call Foo
		# restore bookkeeping info
		# read return value
		# store return value in A[2]
		# deallocate activation frame

MIPS Instruction Set (core)

instruction	***example***	***meaning***
	arithmetic	
add	add $1,$2,$3	$1 = $2 + $3
subtract	sub $1,$2,$3	$1 = $2 - $3
add immediate	addi $1,$2,100	$1 = $2 + 100
add unsigned	addu $1,$2,$3	$1 = $2 + $3
subtract unsigned	subu $1,$2,$3	$1 = $2 - $3
add immediate unsigned	addiu $1,$2,100	$1 = $2 + 100
set if less than	slt $1, $2, $3	if ($2 < $3), $1 = 1 else $1 = 0
set if less than immediate	slti $1, $2, 100	if ($2 < 100), $1 = 1 else $1 = 0
set if less than unsigned	sltu $1, $2, $3	if ($2 < $3), $1 = 1 else $1 = 0
set if < immediate unsigned	sltui $1, $2, 100	if ($2 < 100), $1 = 1 else $1 = 0
multiply	mult $2,$3	Hi, Lo = $2 * $3, 64-bit signed product
multiply unsigned	multu $2,$3	Hi, Lo = $2 * $3, 64-bit unsigned product
divide	div $2,$3	Lo = $2 / $3, Hi = $2 mod $3
divide unsigned	divu $2,$3	Lo = $2 / $3, Hi = $2 mod $3, unsigned
	transfer	
move from Hi	mfhi $1	$1 = Hi
move from Lo	mflo $1	$1 = Lo
load upper immediate	lui $1,100	$1 = 100 x 2^16
	logic	
and	and $1,$2,$3	$1 = $2 & $3
or	or $1,$2,$3	$1 = $2 | $3
and immediate	andi $1,$2,100	$1 = $2 & 100
or immediate	ori $1,$2,100	$1 = $2 | 100
nor	nor $1,$2,$3	$1 = not($2 | $3)
xor	xor $1, $2, $3	$1 = $2 ⊕ $3
xor immediate	xori $1, $2, 255	$1 = $2 ⊕ 255
	shift	
shift left logical	sll $1,$2,5	$1 = $2 << 5 (logical)
shift left logical variable	sllv $1,$2,$3	$1 = $2 << $3 (logical), variable shift amt
shift right logical	srl $1,$2,5	$1 = $2 >> 5 (logical)
shift right logical variable	srlv $1,$2,$3	$1 = $2 >> $3 (logical), variable shift amt
shift right arithmetic	sra $1,$2,5	$1 = $2 >> 5 (arithmetic)
shift right arithmetic variable	srav $1,$2,$3	$1 = $2 >> $3 (arithmetic), variable shift amt
	memory	
load word	lw $1, 1000($2)	$1 = memory [$2+1000]
store word	sw $1, 1000($2)	memory [$2+1000] = $1
load byte	lb $1, 1002($2)	$1 = memory[$2+1002] in least sig. byte
load byte unsigned	lbu $1, 1002($2)	$1 = memory[$2+1002] in least sig. byte
store byte	sb $1, 1002($2)	memory[$2+1002] = $1 (byte modified only)
	branch	
branch if equal	beq $1,$2,100	if ($1 = $2), PC = PC + 4 + (100*4)
branch if not equal	bne $1,$2,100	if ($1 ≠ $2), PC = PC + 4 + (100*4)
	jump	
jump	j 10000	PC = 10000*4
jump register	jr $31	PC = $31
jump and link	jal 10000	$31 = PC + 4; PC = 10000*4