EECS 322 Test 1 Wednesday February 14, 2001

Name: ______________________________

Problem 1 (18%). Assemble the following machine instructions into **binary**, use spaces to separate fields, and **registers** in their symbolic form ($ra NOT $31). Assume absolute jump addresses.

Field 1	Fields 2 and etc	instruction
000000	00011 00000 00000 00000 001000 $3	jr $v1 jr $3
001101	00000 01010 0000 0000 1101 1110 $0 $10 0xde	ori $10, $0, 0xde
101000 page A-67	11101 00000 0000 0000 0001 0001 $29 $0 17	sb $zero,17($sp) sb $0,17($29)
000100	10011 11001 0000 0111 0100 0001 $19 $25 0x741	beq $s3,$t9, 0x741 beq $19,$25, 0x741
000000	00000 00000 11111 00000 100000 $0 $0 $31	clear $ra add $31,$0,$0
000000	0 0 0 0 0 0 0 0 1 0 1 1 1 1 1 0 0 0 1 1 0 0 0 0 1 0 x x x x x v v v v v r r r r r 3 3 3 3 3 x x x x x x	srl $ra,$v0,3

Problem 2 (7%).
Assume each part is **independent**. Assume absolute jump & branch addresses (no pc relative).
Fill in only registers that changed!
What is the value of the register or memory contents **after** the execution of the instruction.
Assume pc = 2020; $s3=12; $s4=4; $ra=250; memory[12]=0x18701914;

instruction	pc	$ra	$s3	$s4	memory[8]	memory[12]
jr $31 same as jr $ra	250					
and $s3, $s3, $zero	2024		0			
sw $s4, 8($s4)	2024					4
li $s3,0x17761812	2028		0x17761812			
slt $s3, $s3, $s4	2024		0			
bne $s3,$s4,40	40					
li $s4,0x1776	2024			0x1776		

pseudo-instruction	machine instruction
li $reg,small16	addi $reg,$0,small16
li $reg,big32	ori $reg,$reg,lower_half(big32) lui $reg,upper half(big32)

Problem 3. (25%) Translate the following C code into MIPS. Please comment your code.
Assume **x** is $t2; **y** is $t6; **p** is $a1 and points to integers; **s** is $a3 and points to unsigned char.
***No** pseudo-assembler instructions allowed. Points will be taken off for assembler syntax errors.*

(a) x = (x & y) + (y << 5);

```
        and  $t0,$t2,$t6        # t0=x & y
        sll  $t1,$t6,5          # t1=y << 5
        add  $t2,$t0,$t1        # x= t0 + t1
```

(b) x = (y <= 3)? 0x1999 : y;

```
                                # !(y ≤ 3) → (y !≤ 3) → y>3 → 3<y
                                # → bne(3<y)
        addi $t0,$0,3           # t0=3
        slt  $t1,$t0,$t6        # if (3 < y ) t1=1; else t1=0;
        bne  $t1,$0,L1          # if (t1 != 0) goto L1
        addi $t2,$0,0x1999      # then x = 0x1999
        j    L2
L1:     add  $t2,$0,$t6         # else x = y
L2:
```

(c) for(x=y; x > y; x++) { y = y – 3; }

```
                                # !(x>y) → !(y<x) → beq(y<x)
        add  $t2,$t6,$0         #initialize x=y;
L2:     slt  $t0,$t6,$t2        #if (y < x) t1=1; else t1=0;
        beq  $t0,$0,L1          #if (t0 = = 0) goto L1
        addi $t6,$t6,–3         #y = y – 3
        addi $t2,$t2,1          #x ++
        j    L2
L2:
```

(d) s[3] = p[3];

```
# alternative solution (best)
lw   $t0,12($a1)    # t0=*((char )p +3*4)
sb   $t0,3($a3)     # *(s+3) = t0
```

```
addi $t0,$0,3       # byte offset  = int offset * sizeof(int)
add  $t1,$t0,$t0    # t1 = t0 * 2
add  $t3,$t1,$t1    # t3 = t1 * 2  (why did I not use $t2 as a temp?)
add  $t4,$a1,$t3    # t4 = p + t3 = p + t0*4 = p + 3*4
lw   $t5,0($t4)     # t5 = *t4 = *(p+t3)
add  $t7,$a3,3      # t7 = s + 3
sb   $t5,0($t7)     # *t7 = t5
```

(e) *(p + y) = *(s + 2 + x) + 2;

```
addi $t0,$t2,2      # t0 = x + 2
add  $t1,$a3,$t0    # t1 = s + t0
lbu  $t3,0($t1)     # t3 = *t1
addi $t3,$t3,2      # t3 += 3
add  $t4,$t6,$t6    # t4 = 2*y
add  $t5,$t4,$t4    # t5 = 4*y
add  $t7,$a1,$t5    # t7 = p + t5
sw   $t3,0($t7)     # *t7 = t3;
```

```
#alternative solution (best)
                       #
add  $t0,$a3,$t2       # t1 = s + x
lbu  $t0,2($t0)        # t0 = *(t0+2)
addi $t0,$t0,2         # t0 += 2
                       #
sll  $t1,$t6,2         # t1 = y<<2 = 4*y
add  $t1,$a1,$t1       # t1 = p + t1
sw   $t0,0($t1)        # *t1 = t0;
```

Problem 4. (25%) Translate the following code and add **comments**

No *pseudo-assembler instructions allowed.   Points will be taken off for assembler syntax errors.*

```
void f(int x, int y)  {
     register int  w=0x1960;

     w = f(y, x + 5);
     if (y < x )  { return w; }

     return x;
}
```

(a) Write the prolog

```
     .text
f:   addi $sp,$sp, -16   #allocate space for w, and $ra
     sw   $ra,12($sp)
     sw   $s2,8($sp)     # needed for w
     sw   $s1,4($12)     # needed for $a1 on recursion
     sw   $s0,0($sp)     # needed for $a0 on recursion
     addi $s2,$0,0x1960  # w = 0x1960
```

(b) Write the body

Remember $a0 and $a1 are destroyed by the previous recursive jal.
Must use $s0, $s1 or restore by a load.

```
     add  $s0,$a0,$0     # save argument x in $s0
     add  $s1,$a1,$0     # save argument y in $s1
     add  $a0,$s1,$0     # $a0 = y
     add  $a1,$s0,5      # $a1 = x + 5
     jal  f              # $v0 = f($a0,$a1)
     # x and y cannot be $a0 & $a1 since they as not saved
                         # !(x < y) → beq(x < y)
     slt  $t0,$s0,$s1    # if (y < x) t0=1; else t0=0;
     beq  $t0,$0,L1      # if (t0 = = 0 ) goto L1;
     add  $v0,$v0,$0     # return w = return of jal f
     j    f_epilog
L1:  add  $v0,$s0,$0     # return x;
                         # jump or fall through to epilog
```

(c) Write the epilog

```
f_epilog:
     lw   $s0,0($sp)     #restore caller
     lw   $s1,4($sp)
     lw   $s2,8($sp)
     lw   $ra,12($sp)
     addi $sp,$sp,16     #must match entry allocation
     jr   $ra
```

Problem 5. (10%) Translate the following global variables

```
(a)  int x[3] = { 0x1920, 0x1930 };
```

x: name of variable is the label

```
          .data
     x:   .word     0x1920, 0x1930, 0    # "3" words allocated
                                         # BUT only "2" are  initialized!
```

```
(b)  struct keyword {
          short     *p,  x;
          unsigned char    t[2];
          struct treenode  *left;
     }   fp;
```

```
          .data
     fp:                                # NECESSARY LABEL for accessing struct!
     fp_p:     .word     0              # short *p;
     fp_x:     .half     0              # short x;
     fp_t:     .byte     0,0            # unsigned char t[2];
     fp_left:  .word     0              # struct treenode *left;
```

Problem 6. (15%) Given the following instruction sequence in the table below.

Assume the (alu and slt instructions are 5 clocks); (loads 10 clocks); (stores 20 clocks); (jumps 2 clocks); (branches 5 fall through/10 for branch);

(a) Show the **best** case timing path through the code showing annotations and total. #all 4 paths
(b) Show the **worst** case timing path through the code showing annotations and total. #all 4 paths

(c) What values will make this code execute the worst case?

```
# Must state $a0,$a1, $a2. Any other register is wrong!
$a0 = any value, $a1= any value, $a2=any value
```

instruction	best case				worst case			
slt $t0,$a0,$a1	5	5	5	5	5	5	5	5
bne $t0,$zero,L1	10	10	5	5	10	10	5	5
addi $t2,$zero,5			5	5			5	5
L1: beq $a2,$zero,L2	10	5	10	5	10	5	10	5
addi $a1,$a1,3		5		5		5		5
L2: addi $s1,$zero,10	5	5	5	5	5	5	5	5
Total Time	30	30	30	30	30	30	30	30