EECS 314 Extra Credit Test 1 Due: Wednesday February 6, 2002

Name: _____ Email: _____

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
000011 (3)	**00 0000 0000 0000 0001 0000 0000** (0x100 = word offset)	**jal 0x400** (0x400 = byte offset)
001101 (0xd)	**00000 01010 1101 1110 1111 1010** ($0) ($10) (0xdefa)	**li $10, 0xdefa** **(ori $10,$0,0xdefa)**
100101 (0x25)	**101001 00001 00000 00000 011010** ($sp) ($at) (26)	**lhu $at,26($sp)**
000100 (4)	**00000 00000 0000 0001 0000 0000** ($0) ($0) (0x0400 = word offset)	**b 0x400** **(beq $0,$0,0x400)**
000000 (0)	**00110 00000 00110 00000 100111** ($a2) ($0) ($a2) (0) (0x27)	**not $a2** **(nor $a2,$a2,$0)**
0x12740002 000100 (4) 10011 ($19) 10100 ($20) 0000 0000 0000 0010 (0x0002 = word offset)		**beq $19,$20,0x0008** (0x0008 = byte offset)

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=6; $ra=250; memory[8]=0xfedcba98; memory[12]=0x76543210;

Instruction	pc	$ra	$s3	$s4	memory[8]	memory[12]
jal 400	**400**	**2024**				
sll $s4, $s3, 2	**2024**			**48_{10}**		
sh $s4, 8($s4)	**2024**					**0x76540006**
move $ra, $s4	**2024**	**6**				
slt $s3, $s4, $ra	**2024**		**1**			
bgt $s3,$s4,40	**40**					
ori $s3,$s4,0x0001	**2024**		**7**			

Problem 3. (25%) Translate the following C code into MIPS. Please comment your code. Assume signed unless defined otherwise. **x** is $s0; **y** is $s1; **s** is $s2; **t** is $s3; **&r** is $s4; **p** is $s5; **d** is $s6.

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

register unsigned int x, y; register int s, **t; struct { int a[3]; short b; char c; } r, *p; char d;

(a) x += (x – y) + (s – 5);

```
add  $t0, $s2, -5   #t0 = (s-5)
subu $t1, $s0, $s1  #t1 = (x-y) (x and y are unsigned)
add  $t2, $t0, $t1  #t2 = ((x-y) + (s-5))
add  $s0, $s0, $t2  #x =  x + t2
```

(b) **t = d;

```
lw   $t1,0($s3)     #t1 = Memory($s3) = *s3
lw   $t2,0($t1)     #t2 = Memory($t1) = *t1
sw   $s6,0($t2)     #Memory($t2) = *t2 = s6
```

(c) x = (x >= 3)? y*3 : 0x40 + 3;

```
addi $t0, $0, 3     #$t0 = 3
slt  $t1, $s0,$t0   #if (x < 3) t1=1; else t1=0;
bne  $t1, $0, L1    #if (t1 != 0) goto L1;
add  $t2, $s1,$s1   #t2 = 2*y
add  $s0, $s1,$t2   #x = 3*y = y + 2y
j    L2
L1:  addi $s0, $0,0x43   #x = 0x43;
L2:
```

same as `blt $s0,$t0,L1`

(d) for(x=y; x < y+2; x+=8) { y *= 3; }

```
add  $s0, $s1, $0   # initialize x = y;
L2:  addi $t0, $s1, 2    # t0 = y + 2;
slt  $t1, $s0, $t0  # if (x < y+2) t1 = 1 else t1 = 0;
beq  $t1, $0, L1    # if ( t1 == 0) goto L1
add  $t2, $s1, $s1  # t2 = 2 * y;
add  $s1, $s1, $t2  # s1 = 3 * y = 2y + y;
addi $s0, $s0, 8    # x += 8;
j    L2
L1:
```

`bge $s0,$t0,L1`

(e) r.c = r.b + r.a[x]; `/* Offsets: within struct: 0:a[0], 4:a[1], 8:a[2], 12:b, 14:c */`

```
lh   $t1,12($s4)    #$t1 = r.b = *(&r+12) load short
sll  $t2, $s0, 2    #$t2 = x << 2= x*4 = x*sizeof(int)
add  $t4,$s4,$t2    #$t4 = &r + (x << 2)
lw   $t2,0($t4)     #$t2 = r.a[x] (load int)
add  $t3,$t1,$t2    #$t3 = $t1 + $t2
sb   $t3, 14($t4)   #r.c = $t3 (store char)
```

(e) p–>c = p–>b + p–>a[x]; **/* In C/C++, same as (*p).c = (*p).b + (*p).a[x]; */**

```
lh   $t1,12($s5)    #$t1 = p->b = (*p).b (load short)
sll  $t2,$s0,2      #$t2 = x << 2= x*4 = x*sizeof(int)
add  $t4,$s5,$t2    #$t4 = p + (x << 2)
lw   $t2,0($t4)     #$t2 = p->a[x] (load int)
add  $t3,$t1,$t2    #$t3 = $t1 + $t2
sb   $t3, 14($t4)   #p->c = $t3 (store char)
```

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

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

```
char *strcpy(char *s, char *t)  {
     register int  i;

     i=0;
     while((s[i] = t[i]) != '\0') i++; /* note: "=" NOT "= =" */

     return s;
}
```

(a) Write the prolog

```
#prolog is empty because:
```

- $t0..$t3 registers are only used (by convention these registers are not required to be saved).
- Function does not call another function, therefore no need to save $ra.

(b) Write the body

```
     add  $t0, $0, $0    #i = 0;
L1:
     add  $t1, $a1, $t0  #address of t[i] is in $t1
     lb   $t2, 0($t1)    #$t2 = (char) t[i]
     add  $t3, $a0, $t0  #address of s[i] is in $t3
     sb   $t2, 0($t3)    #s[i] = t[i]
     beq  $t2, $0, L2    #if (temp=='\0') goto L2
     addi $t0, $t0,1     #i = i+1
     j    L1
L2:
```

The best way to do this is to write the pseudo code and then translate into MIPS. The pseudo code for the above function would be

```
i=0;
L1:   temp = *(t+i);
      *(s+i) = temp;
if (temp = = '\0') goto L2
      i = i + 1;
      goto L1;

L2:    return s;
```

(c) Write the epilog

```
     add  $v0,$a0,0      #return s
     jr   $ra            #return to caller
```

Problem 5. (10%) Translate the following global variables and assign the location counter beginning at 4000

(a)
```
short x=0x1914; short *s=&x; short **p = &s;
```

Location Counter (Decimal)	Assembler definitions
4000	x: .half 0x1914
4002	s: .word x
4006	p: .word s
4010	

(b)
```
struct keyword {
        char      **argv;
        int       (*daytab)[13];
        int       *(montab[13]);
        void      (*strcpy)();
        struct    keyword *next;
    }   fp;
```

Location counter (Decimal)	Assembler definitions	
4000	fp: .word 0	#char **argv
4004	.word 0	# int (*daytab)[13]
4008	.word 0,0,0,0,0,0,0,0,0,0,0,0,0	# int *(montab[13])
4060	.word 0	# void (*strcpy)()
4064	.word 0	# struct keyword *next

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 4 fall through/8 for branch);

(a) How many different timing paths? 4_____
(b) Show the **best** case timing path through the code showing annotations and total.
(c) Show the **worst** case timing path through the code showing annotations and total.
(d) What values will make this code execute the worst case? ($a0 < 0 , $a2 != 1)

instruction	best case			worst case
slti $t0,$a0,3	5	5	5	5
slt $t1,$a0,$t0	5	5	5	5
beq $t1,$0,L1	8	4	8	4
addi $t2,$zero,5		5		5
L1: beq $a2,$t1,L2	8	8	4	4
addi $a1,$a1,3			5	5
L2: addi $s1,$zero,10	5	5	5	5
Total Time	31	32	32	33

(best case)