**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.

<table>
<thead>
<tr>
<th>Field 1</th>
<th>Fields 2 and etc</th>
<th>Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>000011</td>
<td>00 0000 0000 0000 0001 0000 0000 (0x100 = word offset)</td>
<td>jal 0x400 (0x400 = byte offset)</td>
</tr>
<tr>
<td>001101</td>
<td>00000 01010 1101 1110 1111 1010 ($0) ($10) (0xdefa)</td>
<td>li $10, 0xdefa (ori $10, $0, 0xdefa)</td>
</tr>
<tr>
<td>100101</td>
<td>101001 00001 00000 00000 011010 ($sp) ($at) (26)</td>
<td>lhu $at,26($sp)</td>
</tr>
<tr>
<td>000100</td>
<td>00000 0000 0000 0001 0000 0000 (0x0400 = word offset)</td>
<td>b 0x400 (beq $0, $0, 0x400)</td>
</tr>
<tr>
<td>000000</td>
<td>00110 00000 00110 00000 100111 ($a2) ($0) ($a2) (0) (0x27)</td>
<td>not $a2 (nor $a2, $a2, $0)</td>
</tr>
</tbody>
</table>

**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;

<table>
<thead>
<tr>
<th>Instruction</th>
<th>pc</th>
<th>$ra</th>
<th>$s3</th>
<th>$s4</th>
<th>memory[8]</th>
<th>memory[12]</th>
</tr>
</thead>
<tbody>
<tr>
<td>jal 400</td>
<td>400</td>
<td>2024</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>sll $s4, $s3, 2</td>
<td>2024</td>
<td></td>
<td></td>
<td>4810</td>
<td>0x76540006</td>
<td></td>
</tr>
<tr>
<td>sh $s4, 8($s4)</td>
<td>2024</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>move $ra, $s4</td>
<td>2024</td>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>slt $s3, $s4, $ra</td>
<td>2024</td>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>bgt $s3,$s4,40</td>
<td>40</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ori $s3,$s4,0x0001</td>
<td>2024</td>
<td></td>
<td>7</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Problem 3. (25%) Translate the following C code into MIPS. Please comment your code. Assume
signed unless defined otherwise. \texttt{x} is $s0; \texttt{y}$ is $s1; \texttt{s}$ is $s2; \texttt{t}$ is $s3; \texttt{r}$ is $s4; \texttt{p}$ is $s5; \texttt{d}$ is $s6.$

\textbf{No pseudo-assembler instructions allowed. Points will be taken off for assembler syntax errors.}

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

(a) $x += (x - y) + (s - 5);$

\begin{verbatim}
add $t0, $s2, -5 \hspace{1cm} #t0 = (s-5)
subu $t1, $s0, $s1 \hspace{1cm} #t1 = (x-y) \hspace{1cm} (x \text{ and } y \text{ are unsigned})
add $t2, $t0, $t1 \hspace{1cm} #t2 = ((x-y) + (s-5))
add $s0, $s0, $t2 \hspace{1cm} #x = x + t2
\end{verbatim}

(b) $**t = d;$

\begin{verbatim}
lw $t1,0($s3) \hspace{1cm} #t1 = Memory($s3) = *s3
lw $t2,0($t1) \hspace{1cm} #t2 = Memory($t1) = *t1
sw $s6,0($t2) \hspace{1cm} #Memory($t2) = *t2 = s6
\end{verbatim}

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

\begin{verbatim}
addi $t0, $0, 3 \hspace{1cm} #$t0 = 3
same as slt $t1, $s0,$t0 \hspace{1cm} #if (x < 3) t1=1; else t1=0;
bne $t1, $0, L1 \hspace{1cm} #if (t1 != 0) goto L1;
add $t2, $s1,$s1 \hspace{1cm} #t2 = 2*y
add $s0, $s1,$t2 \hspace{1cm} #x = 3*y = y + 2y
j L2
\end{verbatim}

L1: addi $s0, $0,0x43 \hspace{1cm} #x = 0x43;

L2:

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

\begin{verbatim}
add $s0, $s1, $0 \hspace{1cm} # initialize x = y;
L2: addi $t0, $s1, 2 \hspace{1cm} # t0 = y + 2;
bge $s0,$t0,L1 \hspace{1cm} # if (x < y+2) t1=1 else t1=0;
slt $t1, $s0,$t0 \hspace{1cm} #if (t1 != 0) goto L1;
beq $t1, $0, L1 \hspace{1cm} #if ( t1 == 0) goto L1
add $t2, $s1,$s1 \hspace{1cm} # t2 = 2 * y;
add $s1, $s1, $t2 \hspace{1cm} # s1 = 3 * y = 2y + y;
addi $s0, $s0, 8 \hspace{1cm} # x += 8;
j L2
\end{verbatim}

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 */

\begin{verbatim}
lh $t1,12($s4) \hspace{1cm} #$t1 = r.b = *(r+12) load short
sll $t2, $s0, 2 \hspace{1cm} #$t2 = x << 2= x*4 = x*sizeof(int)
add $t4,$s4,$t2 \hspace{1cm} #$t4 = &r + (x << 2)
lw $t2,0($t4) \hspace{1cm} #$t2 = r.a[x] (load int)
add $t3,$t1,$t2 \hspace{1cm} #$t3 = $t1 + $t2
sb $t3, 14($t4) \hspace{1cm} #r.c = $t3 (store char)
\end{verbatim}

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

\begin{verbatim}
lh $t1,12($s5) \hspace{1cm} #$t1 = p->b = (*p).b (load short)
sll $t2, $s0, 2 \hspace{1cm} #$t2 = x << 2= x*4 = x*sizeof(int)
add $t4,$s5,$t2 \hspace{1cm} #$t4 = p + (x << 2)
lw $t2,0($t4) \hspace{1cm} #$t2 = p->a[x] (load int)
add $t3,$t1,$t2 \hspace{1cm} #$t3 = $t1 + $t2
sb $t3, 14($t4) \hspace{1cm} #p->c = $t3 (store char)
\end{verbatim}
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;

<table>
<thead>
<tr>
<th>Location Counter (Decimal)</th>
<th>Assembler definitions</th>
</tr>
</thead>
<tbody>
<tr>
<td>4000</td>
<td>x: .half 0x1914</td>
</tr>
<tr>
<td>4002</td>
<td>s: .word x</td>
</tr>
<tr>
<td>4006</td>
<td>p: .word s</td>
</tr>
<tr>
<td>4010</td>
<td></td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Location counter (Decimal)</th>
<th>Assembler definitions</th>
</tr>
</thead>
<tbody>
<tr>
<td>4000</td>
<td>fp: .word 0</td>
</tr>
<tr>
<td></td>
<td>#char **argv</td>
</tr>
<tr>
<td>4004</td>
<td>.word 0</td>
</tr>
<tr>
<td></td>
<td># int (*daytab)[13]</td>
</tr>
<tr>
<td>4008</td>
<td>.word 0,0,0,0,0,0,0,0,0,0,0,0,0,0</td>
</tr>
<tr>
<td></td>
<td># int *(montab[13])</td>
</tr>
<tr>
<td>4060</td>
<td>.word 0</td>
</tr>
<tr>
<td></td>
<td># void (*strcpy)()</td>
</tr>
<tr>
<td>4064</td>
<td>.word 0</td>
</tr>
<tr>
<td></td>
<td># struct keyword *next</td>
</tr>
</tbody>
</table>

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? ($a0 < 0, a2 != 1$)

<table>
<thead>
<tr>
<th>instruction</th>
<th>best case</th>
<th>worst case</th>
</tr>
</thead>
<tbody>
<tr>
<td>slti $t0,a0,3</td>
<td>5 5 5</td>
<td>5</td>
</tr>
<tr>
<td>slt $t1,a0,t0</td>
<td>5 5 5</td>
<td>5</td>
</tr>
<tr>
<td>beq $t1,$0,L1</td>
<td>8 4 8</td>
<td>4</td>
</tr>
<tr>
<td>addi $t2,$zero,5</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>L1: beq $a2,$t1,L2</td>
<td>8 8 4</td>
<td>4</td>
</tr>
<tr>
<td>addi $a1,$a1,3</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>L2: addi $s1,$zero,10</td>
<td>5 5 5</td>
<td>5</td>
</tr>
<tr>
<td>Total Time</td>
<td>31 32 32 33</td>
<td></td>
</tr>
</tbody>
</table>

(best case)