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>000000</td>
<td>00011 00000 00000 00000 001000</td>
<td>jr $v1</td>
</tr>
<tr>
<td></td>
<td>$3</td>
<td>jr $3</td>
</tr>
<tr>
<td>001101</td>
<td>00000 01010 0000 0000 1101 1110</td>
<td>ori $10, $0, 0xde</td>
</tr>
<tr>
<td></td>
<td>$0 $10 0xde</td>
<td></td>
</tr>
<tr>
<td>101000</td>
<td>11101 00000 0000 0000 0001 0001</td>
<td>sb $zero,17($sp)</td>
</tr>
<tr>
<td>page A-67</td>
<td>$29 $0 17</td>
<td>sb $0,17($29)</td>
</tr>
<tr>
<td>000100</td>
<td>10011 11001 0000 0111 0100 0001</td>
<td>beq $s3,$t9, 0x741</td>
</tr>
<tr>
<td></td>
<td>$19 $25 0x741</td>
<td>beq $19,$25, 0x741</td>
</tr>
<tr>
<td>000000</td>
<td>00000 00000 11111 00000 100000</td>
<td>clear $ra</td>
</tr>
<tr>
<td></td>
<td>$0 $0 $31</td>
<td>add $31,$0,$0</td>
</tr>
<tr>
<td>000000</td>
<td>00000 00000 10111 11110 0001 110000</td>
<td>srl $ra,$v0,3</td>
</tr>
<tr>
<td></td>
<td>x x x x v v v v v r r r r 3 3 3 3 3 x x x x x</td>
<td></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=4; $ra=250; memory[12]=0x18701914;

<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>jr $31</td>
<td>250</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>same as jr $ra</td>
<td>250</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>and $s3, $s3, $zero</td>
<td>2024</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>sw $s4, 8($s4)</td>
<td>2024</td>
<td></td>
<td></td>
<td>4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>li $s3,0x17761812</td>
<td>2028</td>
<td>0x17761812</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>slt $s3, $s3, $s4</td>
<td>2024</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>bne $s3,$s4,40</td>
<td>40</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>li $s4,0x1776</td>
<td>2024</td>
<td></td>
<td></td>
<td>0x1776</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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

\[
\begin{align*}
\text{and} & \quad \texttt{and } $t0, $t2, $t6 \quad \# t0 = x \& y \\
\text{sll} & \quad \texttt{sll } $t1, $t6, 5 \quad \# t1 = y << 5 \\
\text{add} & \quad \texttt{add } $t2, $t0, $t1 \quad \# x = t0 + t1
\end{align*}
\]

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

\[
\begin{align*}
\text{addi} & \quad \texttt{addi } $t0, 0, 3 \quad \# t0 = 3 \\
\text{slt} & \quad \texttt{slt } $t1, $t0, $t6 \quad \# if (3 < y ) t1=1; else t1=0; \\
\text{bne} & \quad \texttt{bne } $t1, 0, \text{L1} \quad \# if (t1 != 0) goto L1 \\
\text{add} & \quad \texttt{add } $t2, 0, 0x1999 \quad \# then x = 0x1999 \\
\text{j} & \quad \texttt{j } \text{L1} \\
\text{L1:} & \quad \texttt{add } $t2, $t0, $t6 \quad \# else x = y \\
\text{L2:} & \\
\end{align*}
\]

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

\[
\begin{align*}
\text{add} & \quad \texttt{add } $t2, $t6, 0 \quad \#\text{initialize } x=y; \\
\text{L2:} & \quad \texttt{slt } $t0, $t6, $t2 \quad \#\text{if } (y < x) t1=1; else t1=0; \\
\text{beq} & \quad \texttt{beq } $t0, 0, \text{L1} \quad \#\text{if } (t0 = 0) \text{ goto } L1 \\
\text{add} & \quad \texttt{add } $t6, $t6, -3 \quad \# y = y - 3 \\
\text{add} & \quad \texttt{add } $t2, $t2, 1 \quad \# \text{x ++} \\
\text{j} & \quad \texttt{j } \text{L2} \\
\text{L2:} & \\
\end{align*}
\]

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

\[
\begin{align*}
\text{addi} & \quad \texttt{addi } $t0, 0, 3 \quad \#\text{byte offset = int offset } \times \text{sizeof(int)} \\
\text{add} & \quad \texttt{add } $t1, $t0, $t0 \quad \# t1 = t0 \times 2 \\
\text{add} & \quad \texttt{add } $t3, $t1, $t1 \quad \# t3 = t1 \times 2 \quad (\text{why did I not use } $t2 \text{ as a temp?}) \\
\text{add} & \quad \texttt{add } $t4, $a1, $t3 \quad \# t4 = p + t3 = p + t0 \times 4 = p + 3 \times 4 \\
\text{lw} & \quad \texttt{lw } $t5, 0($t4) \quad \# t5 = *t4 = *(p+t3) \\
\text{add} & \quad \texttt{add } $t7, $a3, 3 \quad \# t7 = s + 3 \\
\text{sb} & \quad \texttt{sb } $t5, 0($t7) \quad \# *t7 = t5
\end{align*}
\]

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

\[
\begin{align*}
\text{add} & \quad \texttt{add } $t0, $t2, 2 \quad \# t0 = x + 2 \\
\text{add} & \quad \texttt{add } $t1, $a3, $t0 \quad \# t1 = s + t0 \\
\text{lbu} & \quad \texttt{lbu } $t3, 0($t1) \quad \# t3 = *t1 \\
\text{add} & \quad \texttt{add } $t4, $t3, 2 \quad \# t3 = t3 + 3 \\
\text{add} & \quad \texttt{add } $t4, $t6, $t6 \quad \# t4 = 2 \times y \\
\text{add} & \quad \texttt{add } $t5, $t4, $t4 \quad \# t5 = 4 \times y \\
\text{add} & \quad \texttt{add } $t7, $a1, $t5 \quad \# t7 = p + t5 \\
\text{sw} & \quad \texttt{sw } $t3, 0($t7) \quad \# *t7 = t3;
\end{align*}
\]

#alternative solution (best)
\[
\begin{align*}
\text{lw} & \quad \texttt{lw } $t0, 12($a1) \quad \# t0 = *(char )p + 3 \times 4 \\
\text{sb} & \quad \texttt{sb } $t0, 3($a3) \quad \# *(s + 3) = t0
\end{align*}
\]

#alternative solution (best)
\[
\begin{align*}
\text{add} & \quad \texttt{add } $t0, $t2, 2 \quad \# t0 = x + 2 \\
\text{add} & \quad \texttt{add } $t1, $a3, $t0 \quad \# t1 = s + t0 \\
\text{lbu} & \quad \texttt{lbu } $t3, 0($t1) \quad \# t3 = *t1 \\
\text{add} & \quad \texttt{add } $t4, $t3, 2 \quad \# t3 = t3 + 3 \\
\text{add} & \quad \texttt{add } $t4, $t6, $t6 \quad \# t4 = 2 \times y \\
\text{add} & \quad \texttt{add } $t5, $t4, $t4 \quad \# t5 = 4 \times y \\
\text{add} & \quad \texttt{add } $t7, $a1, $t5 \quad \# t7 = p + t5 \\
\text{sw} & \quad \texttt{sw } $t3, 0($t7) \quad \# *t7 = t3;
\end{align*}
\]
Problem 4. (25%) Translate the following code and add comments.

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

```c
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

```assembly
.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

```assembly
    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
    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

```assembly
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
```

Remember $a0 and $a1 are destroyed by the previous recursive jal. Must use $s0, $s1 or restore by a load.
Problem 5. (10%) Translate the following global variables

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

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

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?

<table>
<thead>
<tr>
<th>instruction</th>
<th>best case</th>
<th>worst case</th>
</tr>
</thead>
<tbody>
<tr>
<td>slt $t0,$a0,$a1</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>bne $t0,$zero,L1</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>addi $t2,$zero,5</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>L1: beq $a2,$zero,L2</td>
<td>10</td>
<td>10</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</td>
<td>5</td>
</tr>
<tr>
<td>Total Time</td>
<td>30</td>
<td>30</td>
</tr>
</tbody>
</table>