1. (15 points) Given 8-bit words
   a) (5 points) What is the binary representation of $9D$?

   1001 1101

   b) (5 points) If $B4$ is in 2's complement form, then what is its decimal value?

   \[
   \begin{align*}
   131 & \rightarrow \text{ B4 } \\
   127 & \rightarrow \text{ 4B } \\
   126 & \rightarrow \text{ 4C = 64 + 12 = 76, so -76 }
   \end{align*}
   \]

   c) (5 points) If $E7$ is in signed-magnitude form, then what is its decimal value?

   \[
   1110 0111 \rightarrow -110 0111 = -67h = -103
   \]

2. (30 points) Convert the following assembly language mnemonics to machine code.

   a) (15 points) For an 8x86 in Intel form, add cl, [si + bp + 729h]. The opcode for this add is $02$. To facilitate partial credit, I suggest that you provide the mode, reg, r/m, and displacement separately before combining them.

   \[
   \begin{array}{cccc}
   \text{Op} & \text{mod} & \text{reg} & \text{Displacement} \\
   \hline
   02h & 10 & 001 & 010 29h 07h = 028A907h
   \end{array}
   \]

   b) (15 points) For a MIPS, OR $t7$, $a2$, $s3$. OR has opcode 0, and func 0x25. To facilitate partial credit, I suggest that you provide the encoding for each part of the instruction separately before combining them.

   \[
   \begin{array}{ccccccc}
   \text{Op} & \text{rs} & \text{rt} & \text{rd} & \text{shift} & \text{func} \\
   \hline
   0000.00 & 00.110 & 1.0011 & 0111.1 & 000.00 & 10.0101 = 00D37825
   \end{array}
   \]

3. (25 points) Using the instruction times listed in Appendix A.10, p. 367, fill in the machine cycles for each instruction and then write the timing expression for the following code in term of N. N’s address is stored in the ! $5

   - BGN # 2
   - LDX# 0
   - LOOP: LDA ! $4
   - ADA# 1
   - OUTW+ ! 6
   - AOC* ! $5 ; N
   - JLT LOOP
   - FIN # 2
   - RTN

   Cycles: 2 points for each correct cycle time, 7 points for timing expression

   \[
   T = 11 + (16 \times N)
   \]
4. (82 points) Write an Intel register neutral function for a min function that will set the int pointed to by minValue to the minimum value found in the array. Please use the 32 bit registers, e.g. eax, not ax. Comments are unnecessary but would be helpful. The signature of min() is void min(int *array, int arrayCount, int *minValue);

You should assume the following declarations:

array        dd 1        
dd 5        
dd -9        
dd 18        
arrayCount   dd 4        
minValue     dd 0

a) (17 points) Provide the Intel assembly code you would write to place the parameters on the stack, call min(), and return the stack to its proper state after calling min(). You must use arrayCount, and not 4.

```
<table>
<thead>
<tr>
<th>Pts</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>push minValue</td>
</tr>
<tr>
<td>3</td>
<td>mov eax, arrayCount</td>
</tr>
<tr>
<td>3</td>
<td>mov eax, [eax]</td>
</tr>
<tr>
<td>2</td>
<td>push eax,</td>
</tr>
<tr>
<td>2</td>
<td>push array</td>
</tr>
<tr>
<td>2</td>
<td>call min</td>
</tr>
<tr>
<td>3</td>
<td>add esp, 12</td>
</tr>
</tbody>
</table>
```

b) (65 points) Provide all the Intel assembly code for the min() function.

```
<table>
<thead>
<tr>
<th>Pts</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>min: push ebp</td>
</tr>
<tr>
<td>3</td>
<td>mov ebp, esp</td>
</tr>
<tr>
<td>2</td>
<td>push eax</td>
</tr>
<tr>
<td>2</td>
<td>push ebx</td>
</tr>
<tr>
<td>2</td>
<td>push ecx</td>
</tr>
<tr>
<td>4</td>
<td>mov ecx, [ebp + 8] ; ecx = arr</td>
</tr>
<tr>
<td>3</td>
<td>mov eax, [ecx] ; eax = arr[0]</td>
</tr>
<tr>
<td>4</td>
<td>mov [ebp + 16], eax ; *minValue = eax</td>
</tr>
<tr>
<td>3</td>
<td>mov ebx, 1 ; i = 1</td>
</tr>
<tr>
<td>5</td>
<td>L1    cmp ebx, [ebp + 12] ; i &lt; arrayCount</td>
</tr>
<tr>
<td>2</td>
<td>jge done</td>
</tr>
<tr>
<td>5</td>
<td>mov eax, [ecx + 4 * ebx] ; eax = ar[i]</td>
</tr>
<tr>
<td>4</td>
<td>cmp eax, [ebp + 16] ; ar[i] &lt; *minValue</td>
</tr>
<tr>
<td>2</td>
<td>jge L2</td>
</tr>
<tr>
<td>4</td>
<td>mov [ebp + 16], eax ; *minValue = eax</td>
</tr>
<tr>
<td>4</td>
<td>L2    add ebx, 1 ; i++</td>
</tr>
<tr>
<td>2</td>
<td>jmp L1</td>
</tr>
<tr>
<td>3</td>
<td>done pop ecx</td>
</tr>
<tr>
<td>2</td>
<td>pop ebx</td>
</tr>
<tr>
<td>2</td>
<td>pop eax</td>
</tr>
<tr>
<td>2</td>
<td>pop ebp</td>
</tr>
<tr>
<td>1</td>
<td>ret</td>
</tr>
</tbody>
</table>
```
5. (50 points)
a) (39 points) Write a "register-neutral" CUSP subroutine for strcmp(). The C declaration is:
int strcmp (const char *str1, const char *str2); This function starts comparing the first character of each string. If they are equal to each other, it continues with the following pairs until the characters differ or until a terminating '0' character is reached. It returns an integral value indicating the relationship between the strings: A zero value indicates that both strings are equal. A value greater than zero indicates that the first character that does not match has a greater value in str1 than in str2; And a value less than zero indicates the opposite. Comments are unnecessary, but would be helpful.

```
4   strcmp: bgn# 1
1   pshx
3   ldx# 0
4   loop: ldc* ! 3
3   sta ! 0
3   ldc* ! 4
2   jeq done
3   cma ! 0
2   jne done
3   adx# 1
2   jmp loop
4   done: sba ! 0
1   popx
3   fin# 1
1   rtn
```

b) (11 points) Given the following, write the CUSP code to call the subroutine strcat, and correct the stack pointer.
s1: .char 'Hello'
   .word 0 ; terminate with '0'
s2: .char 'Hi'
   .word 0 ; terminate with '0'

```
3   psh# s1
3   psh# s2
2   jsr strcmp
3   ads# 2
```

6. (30 points) CUSP is definitely a CISC CPU. Suggest 3 changes to the instruction set and addressing modes that would bring it more in line with a RISC CPU. Assume we still have seven registers, ACC, PC, XR, SP, FP, flags, and IR, but they may be re-wired.

1. Make all instructions the same length, 24 bits.
2. Allow the ACC, XR, and FP to be used interchangeably.
3. Eliminate the addressing modes that take extra time. Have only three addressing modes: immediate, direct, and register.
4. Have the only direct memory access limited to LOAD and STORE instructions.
5. Follow branches and direct memory accesses with delay slots.
6. Since the most operate instructions will be replaced with register addressing mode, the number of opcodes may be reduced to 32.
10 points for each answer up to 30 points.
7. (25 points) In a few sentences, describe how compilers and operating systems facilitate machine independent programming.

**Compilers permit high level languages to ignore the underlying machine by providing a translator from the language to the machine dependent encoding. Operating systems provide well defined system calls that provide a consistent abstraction for the hardware.**

8. (25 points)
a) (15 points) In a sentence of two explain how modern preemptive operating systems regulate application access to the CPU?

Modern operating systems control a timer interrupt that can cause an unavoidable interrupt that passes CPU usage to the operating systems.

b) (10 points) Before preemptive operating systems, there was cooperative multitasking in which applications voluntarily surrendered the CPU to the operating system. In one sentence, explain how an application could surrender control of the CPU to the operating system.

The application would call a subroutine (system call) that is part of the operating system.

9. (20 points) CPU Design

The Intel 8086 described in the text could only access one megabyte of RAM. Now they can access more. Suggest the changes you make so that a later version of x86 CPUs could access four gigabytes of RAM. Beware, this requires multiple changes in the CPU design.

Have the segment registers shifted to the left 16 bits instead of four bits. This would give you a 32-bits address value which is capable of holding 4 billion different values. The MAR, and the address bus would both have to be expanded to handle the enlarged addresses.