1. **True/False.** Circle the appropriate choice.

   (a) **F** Registers and cache memory are the same thing.
   (b) **T** Putting `#include "foo.h"` in your C file is the equivalent of typing the contents of foo.h into your file.
   (c) **T** The EFLAGS register contains the zero flag and the sign flag
   (d) **F** The `.data` section of an x86 assembly program is used to specify the parameters for each procedure.
   (e) **T** EBP is a callee-saved register.
   (f) **F** \(2^{32} = 2G\)
   (g) **F** The difference between the SAR (arithmetic shift right) and SHR (logical shift right) operations is that SAR puts a one into the leftmost bit after a shift and SHR puts a zero.
   (h) **T** The instruction `cmp eax, ebx` (or `cmp %ebx, %eax` in AT&T syntax) computes the value of `EAX - EBX`.
   (i) **T** Given the declaration `int A[10][20];` in C, the following assembly code implements `A[i][j] = 1;` (assuming `A`, `i`, and `j` are all global variables).

<table>
<thead>
<tr>
<th>#Intel Syntax</th>
<th>#AT&amp;T Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>mov eax, OFFSET _A</td>
<td>movl $A,%eax</td>
</tr>
<tr>
<td>mov ecx, DWORD PTR _i</td>
<td>movl _i, %ecx</td>
</tr>
<tr>
<td>imul ecx, 20</td>
<td>imul $20,%ecx</td>
</tr>
<tr>
<td>add ecx, DWORD PTR _j</td>
<td>addl _j,%ecx</td>
</tr>
<tr>
<td>mov DWORD PTR [eax+ecx*4],1</td>
<td>movl $1,(%eax,%ecx,4)</td>
</tr>
</tbody>
</table>

   (j) **F** Every procedure in an x86 assembly program must be declared using `.globl`.

2. **Fill in the blanks provided on this sheet.**

   (a) Write in C a single statement that performs the equivalent of

   \[ y = x \times 100; \]

   but without using the multiply operator (*);

   **Answer:** \[ y = (x \ll 6) + (x \ll 5) + (x \ll 2); \]
(b) Given the C declarations,

```c
typedef struct cell {
    int value;
    struct cell *next;
} CELL;

CELL *p;
```

and assuming the value of `p` resides in EAX, write a single **assembly** instruction that implements the C statement

```c
p = p->next;
```
such that the new value of `p` is put into EAX.

Answer: `mov eax, DWORD PTR [eax+4]` (Intel) or `mov 4(%eax),%eax` (AT&T)


Given a number in EAX, write the assembly code to put the absolute value of that number in EAX.

**Answer:**

```assembly
#Intel Syntax                      #AT&T Syntax
cmp eax, 0                        cmpl $0,%eax
jge done                          jge done
mov ecx,0                         mov $0,%ecx  #subtract eax from 0
sub ecx,eax                        sub %eax,%ecx
mov eax,ecx                       mov %ecx,%eax
done:
```

Note that the actual negation could also be performed by doing

```assembly
xor eax,0xffffffff                xorl eax,$0xffffffff flip bits and add 1
add eax,1                        addl $1,%eax
```
or

```assembly
neg eax                           neg %eax #negates eax, not discussed in class
```


(a) Write a short C procedure, `int median(int x, int y, int z)` that returns the median (middle) value of `x`, `y`, and `z`. For example, `median(3, 6, 5)` would return 5.

**Answer:**

```c
int median(int x, int y, int z) {
    if (x >= y) {
```
if (z >= x)
    return x;
else if (y >= z)
    return y;
else
    return z;
}

else if (z >= y)
    return y;
else if (z >= x)
    return z;
else
    return x;
}

(b) Translate your C code for median into assembly, so that median can be called from C.

Answer:
.globl _median
_median:
push ebp
mov ebp,esp
mov eax,[ebp+8]
mov ecx,[ebp+12]
mov edx,[ebp+16]
cmp eax,ecx
j1 L3
cmp edx,eax
j1 L1
jmp done
L1:
cmp ecx,edx
j1 L2
mov eax,ecx
jmp done
L2:
mov eax,edx
jmp done
L3:
cmp edx,ecx
j1 L4
mov eax,ecx
jmp done
L4:
cmp edx,eax
j1 L5
mov eax,edx
jmp done
L5:
done:
pop ebp
ret

(c) Draw the state of the stack, including showing the values of ESP and EBP, right before the ret instruction of your assembly procedure is executed.
5. Put your answer in the blue book.

Write a C procedure `int count_bits(int n)` that returns the number of bits in `n` that are one. For example, `count_bits(7)` should return 3.

**Answer:**

```c
int count_bits(int a)
{
    int count = 0;
    while(a!=0) {
        if (a & 1)
            count++;
        a = a >> 1;
    }
    return count;
}
```