Problem 1: ARM Assembly and Addressing Modes (30 points).

(a) Assume that memory is initialized to zero and the code shown below executes.

```
mov r2, #100
movw r1, #255
movt r1, #15
strb r1, [r2, 2]!
str r1, [r2], 1
strh r2, [r2, -3]
```

Fill out the following table with the memory contents in hex after executing the prior lines of code. Show how you arrived at your solution by annotating what each line of code does. Note that each memory location is shown as a single byte.

<table>
<thead>
<tr>
<th>Address</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td></td>
</tr>
<tr>
<td>101</td>
<td></td>
</tr>
<tr>
<td>102</td>
<td></td>
</tr>
<tr>
<td>103</td>
<td></td>
</tr>
<tr>
<td>104</td>
<td></td>
</tr>
<tr>
<td>105</td>
<td></td>
</tr>
<tr>
<td>106</td>
<td></td>
</tr>
<tr>
<td>107</td>
<td></td>
</tr>
</tbody>
</table>

(b) Assume r3=0x<YOUR-UMID>, r1=0x00001000, and all other registers and memory locations are initialized to zero. After executing the following code:

```
str r3, [r1, 1]
ldrb r5, [r1], #2
orr r5, r5, #0x0f
strh r3, [r1, #-4]!
ldr r3, [r1]
```

What are the values of registers r1, r3, and r5? Show how you arrived at your solution.

```
UMID =
 r1 =
 r3 =
 r5 =
```
Problem 2: Assembly and C (10 pts). Write C code that does the same thing as the following ARM assembly language code. Your C code must not be longer than three lines.

```
movw r0, #0030
movt r0, #2008
ldr r1, [r0]
add r1, r1
str r1, [r0]
```

Problem 3: Assembly, C, the EABI, and Toolchains (30 pts). Consider the following C program that computes and prints out factorials for 0-9:

```
#include <stdio.h>

int factorial(int n) {
    if (n == 0)
        return 1;
    return n * factorial (n - 1);
}

int main () {
    int i;
    int n;
    for (i = 0; i < 10; ++i) {
        n = factorial (i);
        printf("factorial(%d) = %d\n", i, n);
    }
    return 0;
}
```

(a) Rewrite the `factorial` function in ARM EABI-compliant assembly language, including parameter passing and return values. Annotate your assembly language to show how it works. Ensure that the assembly code you provide can be directly entered into a .s file, assembled, and linked with a C file that calls the `factorial` routine to create an executable.

(b) Provide the exact set of commands needed to assemble the assembly language code, compile the C code, and link the two together to create an executable.
Problem 4: Memory Access from C (10 pts). Assume that memory is initialized to zero and the following code executes:

```c
BASE_EMC = 0x74000000;
uint32_t *x = (uint32_t*)BASE_EMC;
*x = 0x01234567;
*(x-1) = 0x89abcdef;
```

Fill out the following table with the memory contents in hex after executing the prior lines of code:

<table>
<thead>
<tr>
<th>Base Addr</th>
<th>00</th>
<th>01</th>
<th>02</th>
<th>03</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x74000004</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x74000000</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x73FFFFFC</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x73FFFFF8</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Problem 5: Memory Bus (20 pts). Draw a timing diagram that shows the AHB-Lite memory write cycle associated with the C instructions shown below. Assume that x is of type `int32_t`, x = 0x74000000, and that the first of the two instructions experiences a single wait state. Annotate the timing diagram in hex with the values being transferred on the bus.

```c
*x = 0x01234567;
*(x-1) = 0x89abcdef;
```