1. Rewrite the following program in UAL assembly, given its C code. Assume that `gcd()` and `print()` are ABI compliant functions that calculate the greatest common divider and print the variable respectively. Make sure to annotate your assembly code, or you will lose potentially all points! [20 points]

```c
int main() {
    uint32_t a=0x5, b, i;
    b = 4*a;
    while(i<10) {
        b = gcd(a, b);
        i++;
    }
    print(b);
}
```

2. Write the ABI compliant function `uint32_t gcd(uint32_t a, uint32_t b)` from above in UAL (see http://www.cs.mtu.edu/~shene/COURSES/cs201/NOTES/chap04/gcd.html for a simple algorithm). Assume the target is a Cortex-M3 system. Annotate your assembly code, or you will lose points! [15 points]
3. Assume that the memory and registers are initialize to all zero. Write down the memory content at the end of the execution of the two small programs. Assume nothing has been done to the Cortex-M3 this code is running on, and the system is coming out of a reset. [20 points, 10 each part]

3. a)

```
BASE_EMC = 0x74000000;
uint32_t *a = (uint32_t*)BASE_EMC;
*a = 0x76543210;
*(a-1) = 0xfedcba98;
```

<table>
<thead>
<tr>
<th>Address</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x74000003</td>
<td></td>
</tr>
<tr>
<td>0x74000002</td>
<td></td>
</tr>
<tr>
<td>0x74000001</td>
<td></td>
</tr>
<tr>
<td>0x74000000</td>
<td></td>
</tr>
<tr>
<td>0x73FFFFFF</td>
<td></td>
</tr>
<tr>
<td>0x73FFFFFE</td>
<td></td>
</tr>
<tr>
<td>0x73FFFFFD</td>
<td></td>
</tr>
<tr>
<td>0x73FFFFFC</td>
<td></td>
</tr>
</tbody>
</table>

3. b)

```
mov   r2, #100
movw  r1, #50
movt  r1, #51
strh  r1, [r2, 3]
str   r1, [r2], 1
strb  r1, [r2, 1]
strb  r2, [r2, 2]
```

<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>
</tbody>
</table>
4. Draw the AHB Bus timing diagram for the store instructions shown in exercise 3b). Assume there are no wait states to access the memory locations. Use the diagram below, and the notation from Lecture 5 to complete the diagram. [10 points]

5. Translate the following Thumb-2 assembly instructions into hexadecimal OP Codes using the ARMv7-M Architecture Reference Manual section A7.7. [10 points]

```
pop {PC}
pop {R1, R4, R7}
push {R1, R4, R7, LR}
rors R0, R5, #10
ror R0, R5
```

6. Explain in simple english what the assembly function CMP does [5 points]
7. Translate the following OPCodes into Thumb-2 Assembly instructions. Use the ARMv7-M Architecture Reference Manual (Section A5.2 and A5.3 will be very helpful for this). [10 points]

0x5555

0x55555555 (think about this one!)

0xF8064013