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 loose potentially all points! [20 points]

```assembly
.syntax unified
.thumb
.global main
.type main, %function

main: @ int main() {
push {r4, lr} @ save callee safe registers
    @ uint32_t a=0x5, i, b;

    mov r1, #20 @ b = 4*a;
mov r4, #0 @ i = 0;

while: @ while(i<10)
    mov r0, #5 @ setup for function call
    bl gcd @ b = gcd(a, b)
mov r1, r0 @ store return into b

    add r4, r4, #1 @ i++;

    cmp r4, #10 @ while(i<10)
bne while

    bl print @ print(b) note that r0=r1!

pop {r4, lr} @ restore callee safe registers
bx lr @ return to where we came from
```
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](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]

```
.syntax unified
.thumb
.global gcd
.type gcd, %function

@ uint32_t gcd(uint32_t a, uint32_t b)
gcd:
    @ a = r0
    @ b = r1
    @ c = r2
    cmp r0, r1  @ if(a < b)
    bcs.n loop
    mov r2, r0  @ c = a
    mov r0, r1  @ a = b
    mov r1, r2  @ b = c

loop:
    udiv r2, r0, r1 @ c = a / b
    mls r2, r1, r2, r0 @ c = a - b * c
    cmp r2, #0  @ if(c == 0)
    beq.n end   @ return...
    mov r0, r1  @ a = b
    mov r1, r2  @ b = c
    b loop      @ while...

end:
    mov r0, r1
    bx  lr
```
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;

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]
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]

![AHB Bus timing diagram]

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

```assembly
pop {PC}                  0xbd00
pop {R1, R4, R7}          0xbc92
push {R1, R4, R7, LR}     0xb592
rors R0, R5, #10          0xea5f 20b5
ror R0, R5                0x41e8
```

6. Explain in simple english what the assembly function CMP does [5 points]

**CMP** subtracts a value, either an immediate or from a register (with optional shift) from a register and updates the status bits based on the result. It then discards the result.
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 \( \rightarrow \) strb r5, [r2, r5]

0x55555555 (think about this one!) \( \rightarrow \) the above twice (two 16-bit ops)

0xF8064013 \( \rightarrow \) strb r4, [r6, r3, LSL #1]