EE3220 Embedded System Design

Quiz 2 – Suggested solution

Note: The marks will be given for individual condition, and this solution is only for your reference

1. **Suppose register i ($i \leq 12$) is initialized to have a value of i (e.g. r0 = 0, r1 = 1, and so on). Assume the main stack (MSP) is used. Also, in the interrupt handler, if LR = 0xFFFFFFF9, then the main stack (MSP) is used. If LR = 0xFFFFFFFD, then the process stack (PSP) is used. The program status register (PSR) = 0x00000020, PC = 0x08000020, and LR = 0x20008020, when the interrupt occurs. (10%)**

For example:
1. Finish current instruction (except for lengthy instructions)
2. Push context (8 32-bit words) onto current stack (MSP or PSP)
   xPSR, Return address, LR (R14), R12, R3, R2, R1, R0
3. Switch to handler/privileged mode, use MSP
4. Load PC with address of exception handler
5. Load LR with EXC_RETURN code, i.e. 0xFFFFFFF9 in this case
6. Load IPSR with exception number
7. Start executing code of interrupt handle

2. **Translate the following C statement into an assembly program, assuming 16-bit signed integers x, y, and z (signed short) are stored in 32-bit register r0, r1, r2? Please also describe in your own words the meaning of the assembly program. (10%)**

   x = x * y + z – x;

x, y, z are stored in the lower half word of r0, r1, r2.
```
MUL r3, r0,r1   ; r3 = r0*r1, i.e. tmp = x*y
ADD r1, r3, r2    ; r1 = r3 + r1, i.e. result = xy + z
SUB r1, r1, r0    ; r1 = r1 - r0, i.e. result = result - x
```
The final result is stored in r1. And r1 is named "result" and r3 is named "tmp" in the comments for convenience.

3. **Describe the key differences between Mbed and ARM Development Studio. As a University graduate who wants to startup a company, which one should we use and why? (10%)**

For some major difference:

Feature	**ARM Mbed Platform**	**ARM Development Studio**
Registration	Free (Open source)	Licensed
Devices supported	Mbed Enabled ARM microcontrollers	All ARM microcontrollers and processors
Technical support	Supported by 60 partners and a community of 200,000 developers	Supported by ARM Holdings company (over 5000 devices)
Simplicity	Simpler to use	Less simpler than Mbed
Mode of programming	Online compiler but also have an offline Desktop IDE and Mbed CLi	Offline using Desktop IDE using ARM compilers
Language	C/C++	Assembly, C/C++
Interoperability	Projects can be exported to many toolchains. We can export for uVision and import into ARM Dev. Studio projects.	ARM Dev. Studio can import uVision projects which can be exported from Mbe

4. **You are going to write an assembly program. (10%)**

- **To Add r1 to r3 and place the result in r5.**
- **To decrement r2 and check for zero, go to "LABEL".**
- **To multiply r0 by 5.**
- **To initialize r0 by 3.**
- **Then, call a sub-routine that will multiply r0 by 10.**

For first 4 lines:

```
Start    PUSH {r4, r5, lr}
         ADD   r4, r1, r2
         ADD   r5, r4, r3
         SUB   r5, r5, r2
         CMP   r5, #0
```

```
        BEQ     LABEL
        MOV     r4, #5
        MUL     r0, r0, r4
        MOV     r0, #3
        BL      sub
sub     MOV     r4, #10
        MUL     r0, r0, r4
        BX      lr
        END
```

This is only one example, you can finish this in your own ways.

5. **Which of the following is equivalent to "PUSH {r7}"? You can further explain the meaning of it and explain what will happen when "POP {r7}". (5%)**

- 1) SP = SP-4, and then memory[SP] = r7
- 2) SP = SP+4, and then memory[SP] = r7
- 3) memory[SP] = r7 and then SP = SP-4
- 4) memory[SP] = r7, and then SP = SP+4

6. **What does "ALIGN 8, 5" mean? Show the data memory layout assuming that the data memory starts at 0x20000000. (10%)**

```
                AREA        myData, Data
                ALIGN 4
a               DCB         1
b               DCB         2
c               DCB         3

                ALIGN 8,5
d               DCB         5
```

The data align is 8 and the offset is 5

0x20000000	1
0x20000004	2
0x20000008	3
0x2000000D	5

7. **Describe the key differences between Cortex-M3 and Cortex-M4**

For example: There is no floating point number supported in M3 while in M4 floating point number is supported and optional. The DSP in M4 is hardware based while that in M3 is software based.

8. A) Compile this program using https://godbolt.org/, "armv7-a clang 11.0.0"and set optimization level using flags -O0. Now compile this program and generate the assembly code. Explain what is happening in this assembly program. (10%)

```
int main(void){
   int number, count;

   number = 123456;
   count = 0;

   while (number){
      number = number/10;
      count++;
   }

   while(1);
}
```

B) Now change the optimization level to flags -O1. Explain what has happened. How will you change the C program, in order to generate the correct program that can get the valid "count"? (5%)

Here is an example for basic content of this program. You can finish it in your own ways. If you finish main part, you will get full marks.

A. The explanation is in the comments

```
main:
      sub    sp, sp, #12  ;; reserve 12 bytes for the main function
      mov    r0, #0   ;;; initialize count to 0
      str    r0, [sp, #8] ;; push r0 to the stack
      mov    r1, #8768    ;; part of "number"
      orr    r1, r1, #114688 ;;; part of "number"
      str    r1, [sp, #4]
      str    r0, [sp]
;;; push r1 and r0 to stack, r0 and r1 combined to gorm the number
      b      .LBB0_1
.LBB0_1:                                @ =>This Inner Loop Header: Depth=1
      ldr    r0, [sp, #4]     ;; load the part of number
      cmp    r0, #0
      beq    .LBB0_3
      b      .LBB0_2
.LBB0_2:                                @   in Loop: Header=BB0_1 Depth=1
      ldr    r0, [sp, #4]
      ldr    r1, .LCPI0_0
```

```
        smull   r2, r3, r0, r1
        asr     r0, r3, #2
        add     r0, r0, r3, lsr #31
        str     r0, [sp, #4]
        ldr     r0, [sp]
        add     r0, r0, #1
        str     r0, [sp]
        b       .LBB0_1
.LBB0_3:
        b       .LBB0_4
.LBB0_4:                                @ =>This Inner Loop Header: Depth=1
        b       .LBB0_4
.LCPI0_0:
        .long   1717986919                      @ 0x66666667
```

B. The code generated is as follows:

```
main:
.LBB0_1:                                @ =>This Inner Loop Header: Depth=1
        b       .LBB0_1
```

That is, it does nothing. The main reason is that, although there are some variables which require value assignment, but they are not used, therefore, the compiler deletes them for optimization.

Solution: We can encapsulate the computation into a function:

```
int cal(int number)
{
   int count = 0;
    while (number){
      number = number/10;
      count++;
   }
   return count;
}

int main(void){
   int number, count;
   number = 123456;
   count = cal(number);

}
```

9. What are the values for r1 and r2 after the following operations? (10%)

```
LDR     r0, 0x12345678

RBIT    r1, r0

REV     r2, r0
```

r1 = 0x1E6A2C48
r2 = 0x78563412

10. **Suppose r0 = 0x20008000, and the memory layout is as follows. What is the value of r1 after running LDR r1, [r0] if the system is little endian? What is the value if the system uses the big-endian? (5%) Describe the procedure of Nested Vector Interrupt Controller (NVIC) in handling the interrupts and exceptions. (5%)**

Address	Data
0x20008007	0x79
0x20008006	0xCD
0x20008005	0xA3
0x20008004	0xFD
0x20008003	0x0D
0x20008002	0xEB
0x20008001	0x2C
0x20008000	0x1A

0x0DEB2C1A little

0x1A2CEB0D big
NVIC manages and prioritizes external interrupts.