EE3220 Quiz 2 Sample Solutions:

1. **What are the advantages of using the off-the-shelf soft core processor, such as MicroBlaze, over and above designing our own core, and what could be the other disadvantages? Now, compare the ISA of PicoBlaze and the ISA of MicroBlaze, you can use the corresponding instruction as an example.**

Ans:

Advantages:

Microblaze is a 32-bit soft processor core, which means that it isn't a hardware part of the FPGA.
Cost efficient
highly configurable
Supported by EDK.

Disadvantages:

Higher power consumption and decreased performance process compared to hard-core embedded processor

Comparison:

Function/Feature	PicoBlaze	MicroBlaze
Register size	8-bit	32-bit
Registers	16	32
Code storage	Internal block RAM	Internal block RAM/ external
Data storage space	64 bytes internal	0 to 4 Gb
Hardware multiply	n/a	Yes
Barrel shifter	n/a	Optional
Performance (MIPS)	44	85

PicoBlaze MicroBlaze Function/Feature Register size Registers Code storage Data storage space 8-bit 32-bit 16 32 Internal Int。

2. **Describe the development from ARMv7 to ARMv9-M, what new elements are added to the processor. Give 2 examples of Data Transfer Instructions in the ARM family and define what is Mnemonic.**

Ans: For example, you can answer ARMv7 implements a traditional ARM architecture with multiple modes, supports a virtual memory system architecture. ARMv9 extends the benefit of scalable vectors to many more use cases, provides hardware transactional memory supporting for the ARM architecture.

For the examples of Data Transfer instructions, single register load and store instructions,

multiple register load and store instructions.

You can select the examples from what you learned.

3. **What is the difference between branch instructions and jump instructions? Using ARM processor as an example, write a code segment to show the difference.**

Ans:

The branch needs a base register and a displacement to describe the to address. The newer jump only needs the (relative to the jump instruction) displacement for the destination address.

Once upon a time (on a different IBM computer), a big difference between a Jump and a Branch was instruction length and range of operation. A Jump was 2-bytes and could only move +/- 256 bytes from the current address (IIRC). A Branch was 4-bytes and had no such limitation

Sometimes, as in:
BRANCH RELATIVE AND SAVE
BRAS R1,I2 [RI]
Information from the current PSW, including the updated instruction address, is saved as link information at the first-operand location. Subsequently, the instruction address in the PSW is replaced by the branch address.
But not in:
BRANCH RELATIVE ON CONDITION
BRC M1,I2 [RI]
The instruction address in the current PSW is replaced by the branch address if the condition code has one of the values specified by M1; otherwise, normal instruction sequencing proceeds with the updated instruction address.

For the codes, you can write them from your comprehension.

4. Describe the key similarities and differences amongst Mbed, Keil Studio Cloud and ARM Development Studio. If you are one of the student members in the CityU Underwater Robotics Team, which one should we use and why?

Ans: You can answer this question based what impressed you more in learning. As an example: They are all development platform for build, code and debug ARM processor in other language such as C++. Some of them can be used online and they offer different functions.

5. **In this question, you are going to use Mbed to design a simple LED flashing. Write up the C program and describe the procedure to complete the whole design flow. Hint: you may need to use "mbed.h".**

For example:

```
#include  “mbed.h”
#BR 1000ms
int main()
{
DigitalOut led(LED1);
```

```
while(true){
led = !led;
ThisThread::sleep_for(BR);
            }
}
```

6. **Translate the following C program into Assembly program for the ARM processor. Explain what the meaning is of each assembly statement. What will happen if the first two lines of code are moved inside the main loop ()?**

Ans: As an example:

```
push    {r11, lr}
      mov     r11, sp
       sub     sp, sp, #16
       mov     r0, #1
       str     r0, [r11, #-4]
       mov     r1, #0
       str     r1, [sp, #8]
       ldr     r1, [sp, #8]
       str     r0, [sp]
      add     r0,#1
      add     r1,r0
```

If the first two lines of code is moved to the main loop(), the variables would be stored on the stack as opposed to the heap

7. **We have used the following two ARM processors in our course lab exercises, Cortex M-4 and Cortex A-9. Compare and describe their key similarities and key differences. How do you use Vivado and Vitis IDE to complete an SoC Design?**

Ans: For example, you can extend your answer based on this: Cortex M-4 is the microcontroller core, cortex A-9 is the processor core. The cortex A-9 core is a full processor core which would run a full operating system.

Vivado would initially be used for elaborate design, import IPs, synthesis, implement and verified into the SOC. The synthesis model would then be transferred over the Vitis IDE to be programmed with the program it would be running.

8. **A processor with 8-bit words for multiple-precision addition of two 32-bit unsigned numbers: 1F C6 24 7B + 00 57 ED 4B. Describe how the data are placed in the memory, and then write a program / Pseudocode to execute the addition, using add and add with carry instructions.**

```
org 00h                    /* starting address
mov r2,#4                  /* number is 32 bit , so we need to perform addition 4 times (32/8=4).
stored to register r2
mov r0,#30h          /* starting address of first number (LSB is at 30H and MSB at 33H ) is
stored at register r0
mov r1,#40h          /* starting address of second number (LSB is at 40H and MSB at 43H ) is
stored at register r1
clr c                      /* carry bit is cleared for LSB addition
```

```
loop:                   /* starting point of loop
mov a ,@r0          /* content of address in register r0 is transferred to accumulator(a) .
mov r3,a              /* accumulator content is transferred to r3 for addition in future.
mov a, @r1         /* content of address in register r1 is transferred to accumulator(a) .
addc a,r3             /* a and r3 are added through carry (means carry is also added)
mov @r1,a         /* result is stored in memory location corresponds to 2nd number (over
written)
inc r0                 /* memory locarion for first number increased
inc r1                 /* memory locarion for second number increased
djnz r2,loop     /* decrement r2 and checked if zero (in effect loop runs 4 times)
```

9. **Describe what is stacking and unstacking when Interrupt occurs. What is the relationship with thread (the main program) and the exceptional handler?**

Ans:

For stacking steps:

Stacking refers to pushing of context (address) onto a current stack which operates under LIFO or last in first out and the stack grows towards the smaller address.

For unstacking step:

No "return from interrupt" instruction

Use regular instruction instead

BX LR - Branch to address in LR by loading PC with LR contents

POP ···, PC - Pop address from stack into PC

··· with a special value EXC_RETURN loaded into the PC to trigger exception handling processing

BX LR used if EXC_RETURN is still in LR

If EXC_RETURN has been saved on stack, then use POP.

For the relationship, you can select the content from Lecture Note 7. For example: The vector table stores the starting address of exceptional handler. When the main program is interrupted by exceptions, it will finish the current instruction, push the context and switch to handler mode, load the necessary registers, such as looking for the starting address of exceptional handler in vector table and load PC with such address.

10. In the following given Assembly program, describe what this program does, and translate it to the approximate C program. Can we get an optimized version of this assembly code running on a processor?

```
fun1(int):
        sub     sp, sp, #4
        str     r0, [sp]
        ldr     r0, [sp]
        mul     r1, r0, r0
        sub     r0, r1, r0
        add     sp, sp, #4
```

```
        bx      lr
main:
        push    {r11, lr}
        mov     r11, sp
        sub     sp, sp, #16
        mov     r0, #0
        str     r0, [r11, #-4]
        mov     r1, #5
        str     r1, [sp, #8]
        ldr     r1, [sp, #8]
        str     r0, [sp]
        mov     r0, r1
        bl      fun1(int)
        str     r0, [sp, #4]
        ldr     r0, [sp]
        mov     sp, r11
        pop     {r11, lr}
        bx      lr
```

Ans:

```
int fun1(int a){
    return a*a - a;
}

int main(void){
    int a = 5;
    int b = fun1(a);

    return 0;
}
```

----------**END**----------