Answers to Training Exam in 2DT901

Instruction set: LEGv8 in all tasks
Total score: 50 credits

Grade F: 0–19, Grade E: 20–25, Grade D: 26–32,
Grade C: 33–38, Grade B: 39–44: Grade A: 45-50

1. Convert the following numbers to hexadecimal form.
   a. $1001\ 0111_2 = 97_{16}$
   b. $11101_2 = 0001\ 1101_2 = 1D_{16}$
   c. $133_{10} = 8*16 + 5 = 85_{16}$

2. Calculate the subtraction 1101 – 1001 by using the two-complementary method.
   1101 – 1001 = 1001 + (-1001). Use 2-complementary form for –1001:

   1-compl: switch all bits => 0110, then add one to get 2-compl: 0111. Add numbers using ripple-carry-algorithm:

   ```
     111
     ====
     1101
   +0111
   =====
  (1)0100 Ignore the overflow => result = 0100
   ```

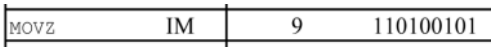
   Answer: 1101 – 1001 = **0100**

3. Below is a simple LEGv8 Assembly program. What number is stored in register X1 after the program has finished? Give the answer as a **hexadecimal number**!

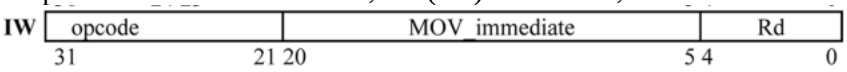
   ```
   MOVZ X1, #10
   MOVZ X2, #20
   ADD  X1, X1, X2 //Calc X1+X2, store result 30 in X1
   LSL  X1, X1, #3 //Shift left 3 pos -> multiply by 8 => X1=8*30=240
   ```

4. Translate the following bit patterns into LEGv8 Assembly instruction.

a. 110100101 00 0000000000100001 00001



Bit pattern identified as MOVZ, IM (IW) instruction,



Divide the bits in groups corresponding to the number of bits in the fields.

110100101 00 0000000000100001 00001

The two last bits in opcode field is for shift amount. In this case, opcode is MOVZ with shift 0, IMMEDIATE is $0000000000100001 = 33_{10}$, Rd = 00001 = 1

ASM instruction is `MOVZ X1, #33`

b. 11010010100000000000001011000010

This is also MOVZ, starts with same 9 bits. Next 2 bits is 00, this means left shift 0, because 00 => LSL #0, 01 => LSL #16, 10 => LSL #32, 11 => LSL #48

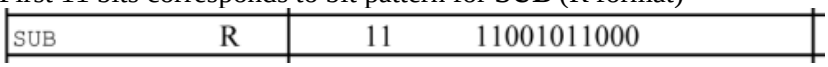
110100101 00 0000000000010110 00010

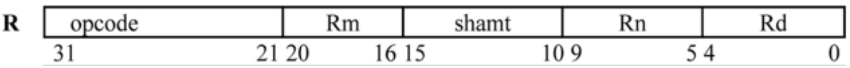
The immediate field in this case is 22_{10} and the last 5 bits gives Rd = 2. Thus, the Assembly instruction is:

`MOVZ X2, #22`

c. 11001011000 00010 000000 00001 00001

First 11 bits corresponds to bit pattern for **SUB** (R format)





Next 0 bits 00010 is Rm = 2_{10} => X2, next 6 bits SHAMT=6, ignore this.

Next 5 bits Rn = 00001 => X1, last 5 bits 00001=1=Rd => X1

ASM instruction: `SUB X1, X1, X2`

5. Explain the fundamental differences between the von Neumann and Harvard architectures.

   The Harvard architecture uses one memory for instructions and one memory for data. Advantages: The CPU can fetch instructions and do a read/write to data memory at the same time. Disadvantages: More expensive, needs more wires for address buses, more difficult to extend memory.

   The von Neumann architecture uses one memory, which is shared between data and instructions. Advantages: Less complex (and less expensive) hardware, easier to upgrade memory, disadvantage: One memory bus, so program instructions can not be fetched while data is read/written => lower performance.

6. Explain why data is often presented in hexadecimal and not binary form!

   Hexadecimal representation is a shorter form, where groups of four binary digits form one hexadecimal digit. This makes hex representation easier for humans to read, D3E is easier to read than 110100111110 and takes up less space on the screen.

7. Three fundamental components of a microprocessor are ALU, Registers and Buses (Data bus, Address bus, Control Bus). Explain the purpose of these three components.

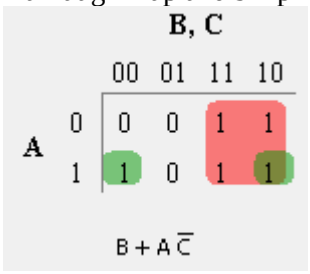
   ALU: Arithmetic/Logic Unit, performs all calculations and logical operations (like AND, OR) on bits in the registers.

   Resisters are used to store the data that the CPU is currently working with. Each register can contain one binary number of fixed length (usually 32 or 64 bits depending on architecture).

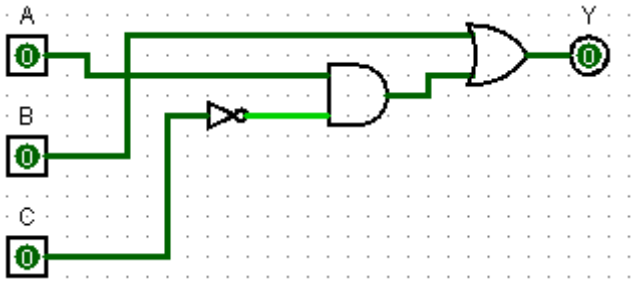
   Buses: Parallel wires, one for each bit, used to transfer data and control signals between CPU and memory. There are also serial buses, for example USB, where the bits are transferred in sequence, one bit at a time.

8.

a. Karnaugh map and simplified expression:



b. Implementation with logic gates:



9. The code below shows a simple LEGv8 program.

```
MOVZ X1, #5
MOVZ X2, #10
loop:
ADDI  X1, X1, #5
SUBIS X2, X2, #2
B.GE  loop
```

a. Which number is stored in X1 after the program finishes?
Give your answer in decimal form.

The number stored in X1 is 35_{10}

b. How many times does the loop repeat?
Explain your answer!

The loop is repeated 6 times, because the counter in X2 starts at 10 and is subtracted by 2 for each step in the loop. When the counter reaches 0, the B.GE is evaulated to false, and the loop stops.

c. Why is the instruction SUBIS and not SUBI used?

The B.GE statemens uses flags to check if the condition is true or false. The instructions ending with S set flags, but not the instructions without S. Thus, SUBIS sets flags but SUBI does not affect the flags.

10. In the following program, four numbers are inserted into registers.

```
MOVZ X0, #17
MOVZ X1, #23
MOVZ X2, #15
MOVZ X3, #18
```

a. Write a subroutine that takes the four registers X0, X1, X2, X3 and return their average number, with return value in register X0.

```
average:
ADD X0, X0, X1
ADD X0, X0, X2
ADD X0, X0, X3
//Div by 4 is same as right shift 2 steps
LSR X0, X0, #2
//Return to point where subroutine was called
BR LR
```

b. Call the subroutine from the main program.

```
MOVZ X0, #17
MOVZ X1, #23
MOVZ X2, #15
MOVZ X3, #18
BL   average
```

c. How is the Link Register (LR) used when subroutines are called?

When a subroutine is called with BL, the PC is copied to the LR register, so the program can continue to where it was called after the subroutine has finished. The subroutine has to end with the instruction BR LR (Branch to Register) or BX LR (as it is called in the ARMv6m Thumb version)

11. When subroutines are called within subroutines, the Link Register must be saved on the stack before subroutine calls. Explain why!

    When the first subroutine is called, the PC is copied to LR. When the nested subroutine call is made, the LR is once again overwritten with the new PC, and then the original return value of the PC is lost. This can be solved by saving the PC on stack (push) before the nested subroutine call is made, and then store it back again from the stach to the PC (pop) after the nested subroutine call is finished.

12. Virtual memory is a memory management technique used by operating systems to provide the illusion of having more available memory than is physically present in a computer system. It allows programs to operate as if they have access to a large, contiguous block of memory, even if the physical memory is limited.

    The concept of virtual memory involves the use of a combination of physical memory (RAM) and disk storage. The operating system divides the virtual address space used by a program into fixed-size blocks called pages. These pages are then mapped to either physical memory or disk storage.

    When a program requests data from memory, the operating system checks if the requested data is present in physical memory. If it is, the data is retrieved directly from RAM. However, if the requested data is not currently in physical memory, the operating system retrieves it from disk storage, a process known as a page fault."

13. A very large hexadecimal number can not be inserted into a register using the MOVZ instruction. For example, the instruction MOVZ X1, 0x3f4c01 fails when the program is translated to machine code.

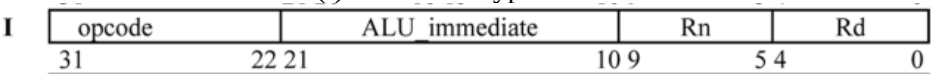
    Write a LEGv8 program that solves this problem!

    The LEGv8 instruction set uses 16 bits for immmediate operator in the MOVZ instruction, so 4 hexadecimal digits can be inserted at once. Start with inserting the higher four bits (003f) into X1, shift them 16 bits, gives the number 3f0000 in X1. Then, move the lower bits 4c01 with keep into X1. The full program is:

    ```
    //Insert 0x3f4c01 into X1
    MOVZ X1, #0x3f, LSL #16
    MOVK X1, #0x4c01
    ```

14. When adding an immediate value to a register, the instruction ADDI can be used, in the form ADDI Xn, Xn, #IMMEDIATE, for example ADDI X1, X1, #12. What number is the largest possible immediate number possible in this instruction? You have to **motivate your answer**!

ADDI is an IMMEDIATE (I) instruction type. The format for I is:



As we can see, the bits from 21 to 9 => **12 bits** is used for immediate value. The number of different binary number is 2^{12}=4096, so **immidiate constants in the interval 0-4095 are possible to add using ADDI instruction.**

15. Find the error in the following LEGv8 program:

```
MOVZ X1, #11
MOVZ X2, #2
SUB  X1, X1, #4
SUB  X1, X1, X2
```

The third instruction must be an SUBI instruction, because the SUB instruction can only take registers as parameters!

16. Write a LEGv8 program that calculates the sum $1^2+3^2+5^2+\ldots 99^2$

```
MOVZ X1, #0 //Used to store sum os squares
MOVZ X2, #99 //Counter
loop:
MOV X3, X2
MUL X3, X3, X3 //Multiply by itself
ADD X1, X1, X3 //Add current square to sum
SUBIS X2, X2, #1 //Decrement counter
B.GE loop
```

Note: All instructions in the reference sheet are valid LEGv8 instructions, although the simulator we used does not implement them all! So instructions such as MUL, SDIV, UDIV, … are available and can be used on all exams!