Question 6 (30 points):

A binary optimizer is a program that analyses and transforms the binary representation of a program into a more efficient version of the same program. In this question your task is to write functions that analyze the binary of a RISC-V assembly program and identify opportunities for improvement. We are interested in the load-use dependence issue. In most modern microprocessors, whenever a load instruction is immediately followed by an instruction that uses the value in the destination register of the load, a delay of at least one cycle will be needed in the pipeline. For example:

```
lb      t3, 0(t4)
add     t0, t1, t3
```

To eliminate this delay, first we need to find the occurrences of such cases in the binary program. This is the task that we will solve in this question. Our goal will be to create an array of bytes called `dependent` where each byte corresponds to one instruction in the program[1]. Initially this array of bytes is already allocated, there is one byte for each instruction in the program, and the value of all the bytes is zero. After our functions are executed, the bytes that correspond to the immediate use of the value of a load instruction will contain the value `0xFF`.

In an actual binary optimizer we would have to analyze all instructions that use a register value. However, to simplify this question we will only look for loads that are immediately followed by R type instructions, also known as ALU instructions.

The binary for the program is already stored in memory with one word corresponding to each instruction. The end of the program is signalled by a sentinel value `0xFFFFFFFF`. You are asked to write the RISC-V assembly code for two functions: `Decode` and `LoadUse`. In both functions you must follow all the RISC-V register save/restore conventions.

The opcode for all types of load instruction is `0000011`. The opcode for all ALU (R-type) instructions is `0110011`. Load instructions are I-type instructions and ALU instructions are R-type instructions. The core instruction formats for RISC-V are shown in Figure **??**.

31 27 26 25	24 20	19 15	14 12	11 7	6 0	
funct7	rs2	rs1	funct3	rd	opcode	R-type
imm[11:0]		rs1	funct3	rd	opcode	I-type
imm[11:5]	rs2	rs1	funct3	imm[4:0]	opcode	S-type
imm[12|10:5]	rs2	rs1	funct3	imm[4:1|11]	opcode	B-type
imm[31:12]				rd	opcode	U-type
imm[20|10:1|11|19:12]				rd	opcode	J-type

Figure 1: RISC-V Core Instruction Formats.

[1]In a real compiler a bit vector is typically used for this purpose with one bit corresponding to each instruction. This question is using a whole byte to represent the immediate-use condition for each instruction to make the programming simpler.

1. (**10 points**) The function `Decode` has the following specification:

   parameters:

   - `a0`: binary representation of a single RISC-V assembly instruction

   return value:

   - `a0`:
     - `0`: if it is a load instruction
     - `1`: if it is an ALU instruction, also known as an R-type instruction
     - `-1`: if it is neither a load nor an ALU instruction
   - `a1`:
     - if it is a load instruction: five bits specifying `rd` register in the five least-significant bits of `a1`. All other bits of `a1` are zero.
     - if it is an ALU instruction: five bits specifying `rs1` in the five least-significant bits of `a1`. All other bits of `a1` are zero.
     - if it is another type of instruction: the value of `a1` is does not matter
   - `a2`:
     - if it is a load instruction: the value of `a2` does not matter
     - if it is an ALU instruction: five bits specifying `rs2` in the five least-significant bits of `a2`. All other bits of `a2` are zero.
     - if it is another type of instruction: the value of `a2` is does not matter

2. (**20 points**) Now you will write the `LoadUse` function that receives as parameters two memory addresses:

   - the address of the first instruction in a RISC-V assembly program; and
   - the address of the first byte of a preallocated array of bytes, called `dependent`, that already contains the value zero in each byte.

   `LoadUse` analyzes the binary RISC-V assembly program and writes the value `0xFF` in each byte of `dependent` whose position corresponds to an ALU instruction that uses the value of a register that was obtained by an immediate predecessor load instruction. Figure **??** provides an example illustrating the values that will be produced in the `dependent` array for a given RISC-V program. Your solution must be general and must work for any RISC-V program.

   The function `LoadUse` **must call** the function `Decode` to decode each instruction and find out if it is an instruction of interest. The specification of `LoadUse` is as follows:

   parameters:

   - `a0`: address of the first instruction of a RISC-V Assembly Program
   - `a1`: address of first element of the `dependent` array

   return value: None

```
Binary Code   RISC-V Assembly              dependent array
0x00050283    lb  t0, 0(a0)                   0x00
0x00058303    lb  t1, 0(a1)                   0x00
0x00628e33    add t3, t0, t1                  0xFF
0x00060383    lb  t2, 0(a2)                   0x00
0x006e7eb3    and t4, t3, t1                  0x00
0x0006af03    lw  t5, 0(a3)                   0x00
0x01eeffb3    and t6, t4, t5                  0xFF
0xfe0f06e3    beq t5, zero <label>            0x00
0xFFFFFFFF    # sentinel value
```

Figure 2: Example illustrating the values to be returned in the `dependent` array.

RISC-V code for `Decode`

Draft RISC-V code for `LoadUse`