**Q4)**

When calculating the distance   
𝐷  
D using division in ARM assembly, the choice between SDIV (Signed Divide) and UDIV (Unsigned Divide) depends on the nature of the operands and the intended result. Let’s explore why one might be preferred over the other and whether it matters in a specific context.  
  
Signed vs. Unsigned Division  
SDIV (Signed Divide): This instruction performs division where both the numerator and the denominator are treated as signed integers. It considers the sign of the numbers and produces a signed result.  
  
UDIV (Unsigned Divide): This instruction performs division where both the numerator and the denominator are treated as unsigned integers. It treats the numbers as non-negative and produces an unsigned result.  
  
When to Use Each:  
Nature of Operands:  
  
Signed Division (SDIV): Use SDIV when you need to handle negative numbers and you want the result to consider the sign of the operands. For example, if you’re working with coordinates, differences, or other quantities where negative values are possible and meaningful, SDIV would be appropriate.  
Unsigned Division (UDIV): Use UDIV when both operands are non-negative, and you do not expect negative values. For example, when working with indices, sizes, or counts that should logically be non-negative, UDIV is more suitable.  
Result Interpretation:  
  
The result of SDIV can be negative if the numerator or denominator is negative, reflecting the true mathematical outcome of division with signed numbers.  
The result of UDIV is always non-negative, which is appropriate for cases where the division outcome should not be negative.  
Relevance to Distance Calculation:  
When calculating a distance   
𝐷  
D, whether you use SDIV or UDIV depends on the context:  
  
Distance Calculation (e.g., in a geometric or positional sense): Distance is typically a non-negative quantity. If you’re calculating distances, you usually deal with non-negative values, so UDIV would generally be appropriate. For instance, if you’re dividing positive values or zero to compute distance or magnitudes, UDIV is suitable.  
  
Distance with Signed Differences: If your distance calculation involves signed differences (e.g., subtracting one coordinate from another, where the result could be negative), but you need the absolute value, you might want to handle signs properly. In this case, you could use SDIV if you are considering signed integers and then take the absolute value if necessary.  
  
Conclusion:  
Use UDIV if you’re working with non-negative integers (which is common for distance calculations where distances themselves are always positive or zero).

Use SDIV if your division involves signed integers and the result must reflect possible negative values.  
In the context of calculating distances where negative values are not expected or meaningful, UDIV is typically used. Using SDIV would not be necessary unless you are dealing with signed values and need to ensure the sign is properly accounted for in some specific way.

**Q5)**

In ARM assembly, when a branch with link (BL) instruction is executed, it performs a branch to a subroutine and saves the return address into the link register (LR). Here's a detailed explanation of how this works:  
  
What is Saved into the LR Register?  
BL Instruction:  
  
When the BL Convert instruction is executed, the ARM processor branches to the Convert subroutine. At the same time, the address of the instruction immediately following the BL instruction (the return address) is saved into the LR register.  
For example, if the address of the BL Convert instruction is 0x1000, and the BL instruction is located at 0x1000, then the address of the instruction right after BL (i.e., 0x1004) is saved into LR. This saved address is where the processor will return after the Convert subroutine finishes execution.  
  
Command to Return from the Subroutine  
Returning from the Subroutine:  
  
To return from a subroutine and resume execution at the instruction following the BL instruction, the BX LR (Branch and Exchange) or MOV PC, LR (Move to Program Counter) instruction is typically used. Both commands achieve the same goal of branching to the address contained in LR, which is the return address.  
BX LR Example:  
  
assembly  
Copy code  
BX LR  
This instruction branches to the address contained in the LR register. It also performs an exchange of the instruction set if needed, which is useful for switching between ARM and Thumb instruction sets.  
MOV PC, LR Example:  
  
assembly  
Copy code  
MOV PC, LR  
This instruction directly moves the value in the LR register into the PC (Program Counter), effectively causing a branch to the return address.  
Summary  
Saved into LR: The address of the instruction immediately following the BL instruction (the return address).  
Command to Return: BX LR or MOV PC, LR.

**Q7)**

Sure, let's recalculate using the offset 0x0dc instead of 0x2e4.  
  
Calculation of Effective Address  
Instruction Address:  
  
The instruction ldr r1, [pc, #0xdc] is located at address 0x00000410.  
PC-relative Address Calculation:  
  
The PC is 8 bytes ahead of the current instruction address for PC-relative addressing.  
Therefore, pc for this instruction is 0x00000410 + 8 = 0x00000418.  
Effective Address Calculation:  
  
The effective address is computed by adding the offset #0xdc to the PC value.  
Effective address = 0x00000418 + 0x0dc = 0x000005f4.  
Data Loaded into R1:  
  
The value loaded into R1 is the data at the memory address 0x000005f4.  
General Expression  
For the instruction ldr r1, [pc, #0xdc] and pc = 0x00000410, the effective address is 0x000005f4.  
  
So, the data content loaded into R1 is the value at memory address 0x000005f4.

What is the content there?

**Q8)**

MAIN unused is 3f804

Insert photo taken @ 13:19 on 300824

**Q9)**

.text?(Check lecture notes)

Insert photo accordingly

**Q10)**

43b

Insert photo taken @ 13:28 on 300824

Q6)?

Q1-Q3??

Q3) – 7.4 Lab manual + chatgpt

Q1)-Q2) – Chatgpt + lecture notes

Q6) - – Chatgpt + lecture notes