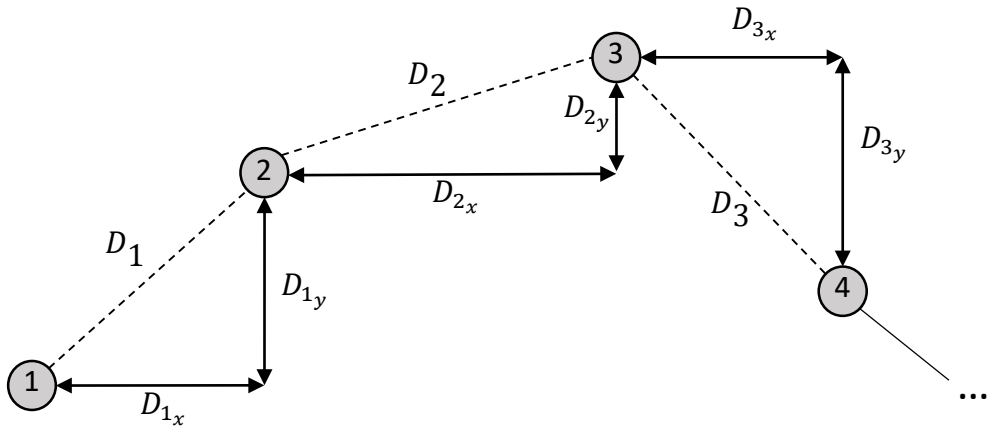
Consider the following problem:

We want to calculate the distance of the path connecting 50 points spread in the Cartesian plane. The points are connected in order of their index by a line segment. For each line segment connecting two points the x and y components are given in the Memory unit. Each component occupies 1 Byte of the Memory and therefore is represented as an 8-bit unsigned integer:



		MEMORY	ADDRESS
D_1	D_{1x}	CB_H	00_H
	D_{1y}	65_H	01_H
D_2	D_{2x}	BD_H	02_H
	D_{2y}	21_H	03_H
D_3	D_{3x}	$E8_H$	04_H
	D_{3y}	$F1_H$	05_H
D_4	D_{4x}	$7A_H$	06_H
	D_{4y}	14_H	07_H
. . .	. . .	. . .	. . .
D_{49}	D_{49x}	$E7_H$	60_H
	D_{49y}	$8E_H$	61_H

Using the following RISC-V assembly instruction, we are going to write the code that calculates the distance of the path connecting these points in order of $1.2.3.\cdots.50$ and stores the result in on the general purpose registers of out CPU.

$$D_1 = \sqrt{\left(D_{1_x}\right)^2 + (D_{1_y})^2}$$

$$Distance = \sum (D_1 + D_2 + \cdots)$$

Format	Name	Pseudocode
`ADD rd, rs1, rs2`	Add	`rd ← rs1 + rs2`
`ADDI rd, rs1, imm`	Add Immediate	`rd ← rs1 + imm`
`LBU* rd, offset(rs1)`	Load Byte Unsigned	`rd ← u8[rs1 + offset]`
`BLT rs1, rs2, offset`	Branch Less Than	`if rs1 < rs2 then pc ← pc + offset`
`MUL rd, rs1, rs2`	Multiply	`rd ← rs1 × rs2`
`FCVT.S.WU** rd, rs1`	Convert Unsigned Int to Single Floating Point	`f{rd} ← u32[rs1]`
`FADD rd, rs1, rs2`	Float Add	`rd ← rs1 + rs2`
`FSQRT rd, rs1`	Float Square Root	`rd ← √rs1`

*__LBU:__ Loads an 8-bit value from memory and zero-extends this to XLEN bits before storing it in register rd.

**__FCVT.S.WU:__ Converts a 32-bit unsigned integer, in integer register rs1 into a floating-point number in floating-point register rd.

To solve this problem, we are going to consider two pointer variables that point to x and y components independently. Using these 2 pointers we are going to iterate over our data memory and calculate each partial distance D_i and accumulate the result in a general purpose register along the way:

	MEMORY	ADDRESS(D)	
X pointer →	**XX**	**0**	x_1
Y pointer →	**XX**	**1**	
Next X pointer →	**XX**	**2**	y_1
	XX	**3**	
	XX	**4**	x_2
	XX	**5**	
	XX	**6**	y_2
	XX	**7**	
	XX	**8**	x_3
	XX	**9**	
	XX	**10**	y_3
	XX	**11**	
	XX	**12**	x_4
	XX	**13**	
	XX	**14**	y_4
	XX	**15**	

```
        ADD         X10, X0, X0      // Initialize x-pointer to 0
        ADDI        X11, X0, 1       // Initialize y-pointer to 1
        ADDI        X12, X0, 98      // Initialize X12 to 98 – End Index
        FADD        FX2, FX2, FX0    // Initialize FX2 to zero for storing result
LOOP:   LBU         X1, 0(X10)       // Load 1 Byte from x-pointer and load into x1 as D1x
        ADDI        X10, X10, 2      // Increase x-pointer by 2
        LBU         X2, 0(X11)       // Load 1 Byte from y-pointer and load into x2 as D1y
        ADDI        X11, X11, 2      // Increase y-pointer by 2
        MUL         X3, X1, X1       // Calculate the second power – (D1x)^2
        MUL         X4, X2, X2       // Calculate the second power – (D1y)^2
        ADD         X5, X3, X4       // (D1x)^2 + (D1y)^2
        FVCT.S.WU   FX5, X5          // Convert unsigned int in X5 to floating point FX5
        FSQRT       FX1, FX5         // Calculate the square root
        FADD        FX2, FX2, FX1    // Add the result to the previous sum
        BLT         X11, X12, LOOP   // Branch to label LOOP if not completed
```