9 VLIW [60 points]

You are the human compiler for a VLIW processor whose specifications are as follows:

- There are a total of 7 functional units: 3 load units, 1 store unit, 1 addition unit, 1 multiplication unit, and 1 branch unit.
- The VLIW processor can **only** execute assembly operations listed in Table 1. The table shows the instructions that each functional unit can execute and each instructon's semantics. Note that the `load_inc`/`store_inc` instructions automatically increment the address source register r_{src2} by 1, *after* data is loaded/stored.
- All assembly operations have a 1-cycle latency (including `load`, `load_inc`, `store`, and `store_inc`).
- This machine has 32 registers (`r0`, `r1`, ..., `r31`).
- The registers are read at the rising edge and written at the falling edge of the clock.
- The memory is word-addressable (1 word = 4 bytes).
- The VLIW processor operates at 1 GHz.

Functional Unit Type	**Operation (in assembly notation)**	**Semantics**
load	`load` r_{dst}, [r_{src1}, r_{src2}, #offset]	r_{dst} := MEM[r_{src1} + r_{src2} + #offset]
	`load_inc` r_{dst}, [r_{src1}, r_{src2}, #offset]	r_{dst} := MEM[r_{src1} + r_{src2} + #offset] r_{src2} := r_{src2} + 1
store	`store` [r_{src1}, r_{src2}, #offset], r_{src3}	MEM[r_{src1} + r_{src2} + #offset] := r_{src3}
	`store_inc` [r_{src1}, r_{src2}, #offset], r_{src3}	MEM[r_{src1} + r_{src2} + #offset] := r_{src3} r_{src2} := r_{src2} + 1
addition	`add` r_{dst}, r_{src1}, r_{src2}	r_{dst} := r_{src1} + r_{src2}
multiplication	`mult` r_{dst}, r_{src1}, r_{src2}	r_{dst} := r_{src1} × r_{src2}
branch	`bne` r_{src1}, #offset, TARGET	branch to TARGET if r_{src1} is not equal to #offset
(any of the above)	`NOP`	Functional unit is idle for one cycle

Table 1: Assembly operations of the target VLIW processor. `#offset` indicates an immediate value.

Figure 1 shows the C code and its equivalent assembly code for the application that we will execute in this VLIW processor. Assume that N **is an even positive integer** throughout this question.

In the assembly code, registers `r29`, `r30`, and `r31` hold the base addresses of the C-code arrays `A`, `B`, and `C`, respectively. Register `r0` is initialized with 0 and register `r1` is initialized with 1.

C code

```
// An integer is 4 bytes long
int A[N+1];
int B[N+1];
int C[N+1];
... // code to initialize A and B
for (int i = 1; i <= N; i++)
    C[i] = C[i-1] * A[i] + B[i];
```

Assembly code

```
LOOP:
    (v1) load_inc   r2, [r31, r0, #0]  // r2 := [r31 + r0 + #0]; r0 := r0 + 1
    (v2) load       r3, [r29, r1, #0]  // r3 := [r29 + r1 + #0]
    (v3) load       r4, [r30, r1, #0]  // r4 := [r30 + r1 + #0]
    (v4) mult       r5, r2, r3         // r5 := r2 * r3
    (v5) add        r6, r5, r4         // r6 := r5 + r4
    (v6) store_inc  [r31, r1, #0], r6  // [r31 + r1 + #0] := r6; r1 := r1 + 1
    (v7) bne        r1, #N, LOOP       // branch to LOOP if r1 not equal to #N
```

Figure 1: C and assembly codes. ($v1$) .. ($v7$) are instruction labels.

(a) [30 points] Your goal in this question is to statically schedule the instructions in Figure 1 to the VLIW processor specified above. Table 2 (on the next page) represents the occupancy of each functional unit during the execution of the assembly code in Figure 1.

For the assembly code given in Figure 1, **fill in Table 2** with the appropriate VLIW instructions.

In your solution, **minimize the number of VLIW instructions**, and ensure that each instruction is scheduled to execute **as soon as possible**. Table 2 should only contain assembly operations supported by the VLIW processor, as described in Table 1.

	Functional Unit						
VLIW Instruction	**Load**	**Load**	**Load**	**Store**	**Mult**	**Add**	**Branch**
1 LOOP:	load_inc r2, [r31, r0, #0]	load r3, [r29, r1, #0]	load r4, [r30, r1, #0]	NOP	NOP	NOP	NOP
2	NOP	NOP	NOP	NOP	mult r5, r2, r3	NOP	NOP
3	NOP	NOP	NOP	NOP	NOP	add r6, r5, r4	NOP
4	NOP	NOP	NOP	store_inc [r31, r1, #0], r6	NOP	NOP	NOP
5	NOP	NOP	NOP	NOP	NOP	NOP	bne r1, #N, LOOP
6							
7							
8							
9							
10							
11							
12							
13							
14							
15							

Table 2

(b) [15 points] What is the ratio between the number of useful operations and the number of VLIW instructions in your code? A useful operation refers to any assembly operation that is *not* a `NOP`.

$\frac{7}{5}$ useful operations per VLIW instruction.

Explanation.
There are a total of 7 assembly operations (excluding `NOP`s) composing 5 VLIW instructions.

(c) [15 points] What is the execution time (in cycles) of the VLIW processor when executing the sequence of instructions in Table 2, as a function of the loop counter N? Show your work.

Execution time $= 5 \times N$.

Explanation.
A single iteration of the loop takes 5 clock cycles to execute. Since the loop repeats N times, the total execution time is equal to $5 \times N$.