## **🔢 What is being done?**

The program multiplies:

go

CopyEdit

AC := AC × N

* **AC** initially contains the multiplicand (let's say Y).
* **N** is stored in memory (mult).
* **Multiplication is performed via repeated addition** (AC + AC + ... N times).

The program:

* Adds AC to a product register N times.
* Uses a **loop** that:  
  + Adds the current AC to prod.
  + Decreases N (stored in mult) by 1.
  + Tests if N == 0 using a BZ (Branch if Zero) instruction.
  + If not, it loops back.

## **🧠 Key Memory Locations**

| **Label** | **Purpose** |
| --- | --- |
| one | Constant value 1 |
| mult | Multiplier N |
| ac | Initial AC value (Y) |
| prod | Final product/result |

## **📜 Assembly Program Breakdown**

| **Line** | **Label** | **Instruction (Assembly)** | **Explanation** |
| --- | --- | --- | --- |
| 0 | one | 00...01 | Memory location holding constant 1 |
| 1 | mult | N | Multiplier N stored here |
| 2 | ac | 00...00 | AC backup (Y) |
| 3 | prod | 00...00 | To store the product (starts at 0) |
| 4 |  | ST ac | Store initial AC (Y) into ac |
| 5 | Loop | LD mult | Load N into AC |
| 6 |  | BZ exit | Exit if N == 0 |
| 7 |  | LD one | Load 1 |
| 8 |  | MOV DR, AC | Move 1 to DR |
| 9 |  | LD mult | Load current N again |
| 10 |  | SUB | N := N - 1 |
| 11 |  | ST mult | Store updated N |
| 12 |  | LD ac | Load Y (multiplicand) from backup |
| 13 |  | MOV DR, AC | Move Y to DR |
| 14 |  | LD prod | Load current product |
| 15 |  | ADD | Add DR (Y) to AC (product += Y) |
| 16 |  | ST prod | Store updated product |
| 17 |  | BRA Loop | Jump to start of loop |
| 18 | exit |  | End of program |

## **⚠️ Limitations of Accumulator-Based CPUs**

1. **Only one accumulator**:  
   * Intermediate results must go back and forth between memory and the accumulator.
2. **Few CPU registers**:  
   * Values like 1, N, Y, and product must be fetched repeatedly from memory.
3. **Slower execution**:  
   * More memory accesses = slower performance.
   * If these values could stay in dedicated **CPU registers**, the program would:  
     + Run **faster**
     + Be **shorter** (fewer instructions)

## **✅ Summary**

* This program multiplies two numbers via **repeated addition** using an accumulator and data register.
* It demonstrates the **single-address**, **register-scarce** design of accumulator-based CPUs.
* The architecture leads to **inefficient memory use**, which limits performance in data-heavy computations.