Assembly x86

An 8088 assembly language program that calculates the expression `(a + 3) * b` and places the result in the accumulator register (AX). The program assumes that the values `a` and `b` are stored in memory locations `a` and `b`, respectively.

```
; Assuming a and b are stored in memory at locations a and b

section .data

a dw 5      ; example value for a

b dw 10     ; example value for b

section .text

global _start

_start:

    ; Load the value of a into AX

    mov ax, [a]


    ; Add 3 to the value of a (now in AX)

    add ax, 3


    ; Load the value of b into BX

    mov bx, [b]


    ; Multiply AX by BX (AX = (a + 3) * b)
```

```
    mul bx

   ; At this point, AX contains the result of (a + 3) * b

   ; The result is now in AX, which is the accumulator register

    ; In a real-world scenario, you would proceed with the rest of your program

    ; For example, to exit the program, you might invoke an interrupt to terminate execution

    ; Since this is an 8088 program, it doesn't include modern system calls for exiting

    ; Example exit code (platform-specific, usually part of DOS or BIOS interrupts):

    ; mov ah, 0x4C

    ; int 0x21

    ; Note: The actual mechanism to halt/exit depends on the specific environment being used

    ; For demonstration purposes, the program simply ends here.

section .bss

    ; BSS section for uninitialized data if needed
```

Explanation:

1. **Data Section (`section .data`)**:
    - This section contains the initialized data `a` and `b`. For this example, `a` is set to `5` and `b` is set to `10`.
2. **Text Section (`section .text`)**:
    - The code starts with the `_start` label, indicating the entry point of the program.
3. **Loading and Adding**:
    - `mov ax, [a]` loads the value of `a` from memory into the `AX` register.
    - `add ax, 3` adds `3` to the value in the `AX` register.
4. **Loading and Multiplying**:
    - `mov bx, [b]` loads the value of `b` from memory into the `BX` register.
    - `mul bx` multiplies the value in `AX` by the value in `BX`. The result of the multiplication is stored in `AX`.
5. **Result**:
    - After the multiplication, `AX` contains the result of the expression `(a + 3) * b`.