Imagen de la pantalla de un celular con letras

Descripción generada automáticamente con confianza baja

Tampico, Tamaulipas a **18** de **Septiembre** de **2022**

**Practicas**

**Equipo 1**

Nombres: **Gonzalez Saldivar Luis Roberto**

**Martinez Reyes Fernando**

Profesor: **Dr. García Ruiz Alejandro Humberto**

Asignatura: **Programación de Microprocesadores**

**8**vo. Semestre – Grupo “**I**”

2022-2

# Índice

[Índice 2](#_Toc114500335)

[Ejercicio 1. Área Trapecio 3](#_Toc114500336)

[Ejercicio 2. Sueldo 6](#_Toc114500337)

[Ejercicio 3. Punto Medio 9](#_Toc114500338)

### Ejercicio 1. Área Trapecio

**Descripción**: Realizar un programa en lenguaje ensamblador en Visual Studio, que sea capaz de calcular el área de un trapecio (sin importar que el resultado este entero por el momento).

**Introducción**:

Se requiere conocimiento de las siguientes instrucciones de lenguaje Ensamblador:

**MOV:**

Propósito: Transferencia de datos entre celdas de memoria, registros y acumulador.

Sintaxis:

MOV Destino,Fuente

Donde Destino es el lugar a donde se moverán los datos y fuente es el lugar donde se encuentran dichos datos.

Los diferentes movimientos de datos permitidos para esta instrucción son:

Destino: memoria. Fuente: acumulador

Destino: acumulador. Fuente: memoria

Destino: registro de segmento. Fuente: memoria/registro

Destino: memoria/registro. Fuente: registro de segmento

Destino: registro. Fuente: registro

Destino: registro. Fuente: memoria

Destino: memoria. Fuente: registro

Destino: registro. Fuente: dato inmediato

Destino: memoria. Fuente: dato inmediato

**ADD:**

Propósito: Adición de los operandos.

Sintaxis:

ADD destino, fuente

Suma los dos operandos y guarda el resultado en el operando destino.

**CALL:** La instrucción CALL sirve para hacer llamadas a funciones en ensamblador.

**MUL:** Multiplicación sin signo

Sintaxis: MUL fuente

El ensamblador asume que el multiplicando será del mismo tamaño que el del multiplicador, por lo tanto, multiplica el valor almacenado en el registro que se le da como operando por el que se encuentre contenido en AH si el multiplicador es de 8 bits o por AX si el multiplicador es de 16 bits.

Cuando se realiza una multiplicación con valores de 8 bits el resultado se almacena en el registro AX y cuando la multiplicación es con valores de 16 bits el resultado se almacena en el registro par DX:AX.

**DIV:** División sin signo

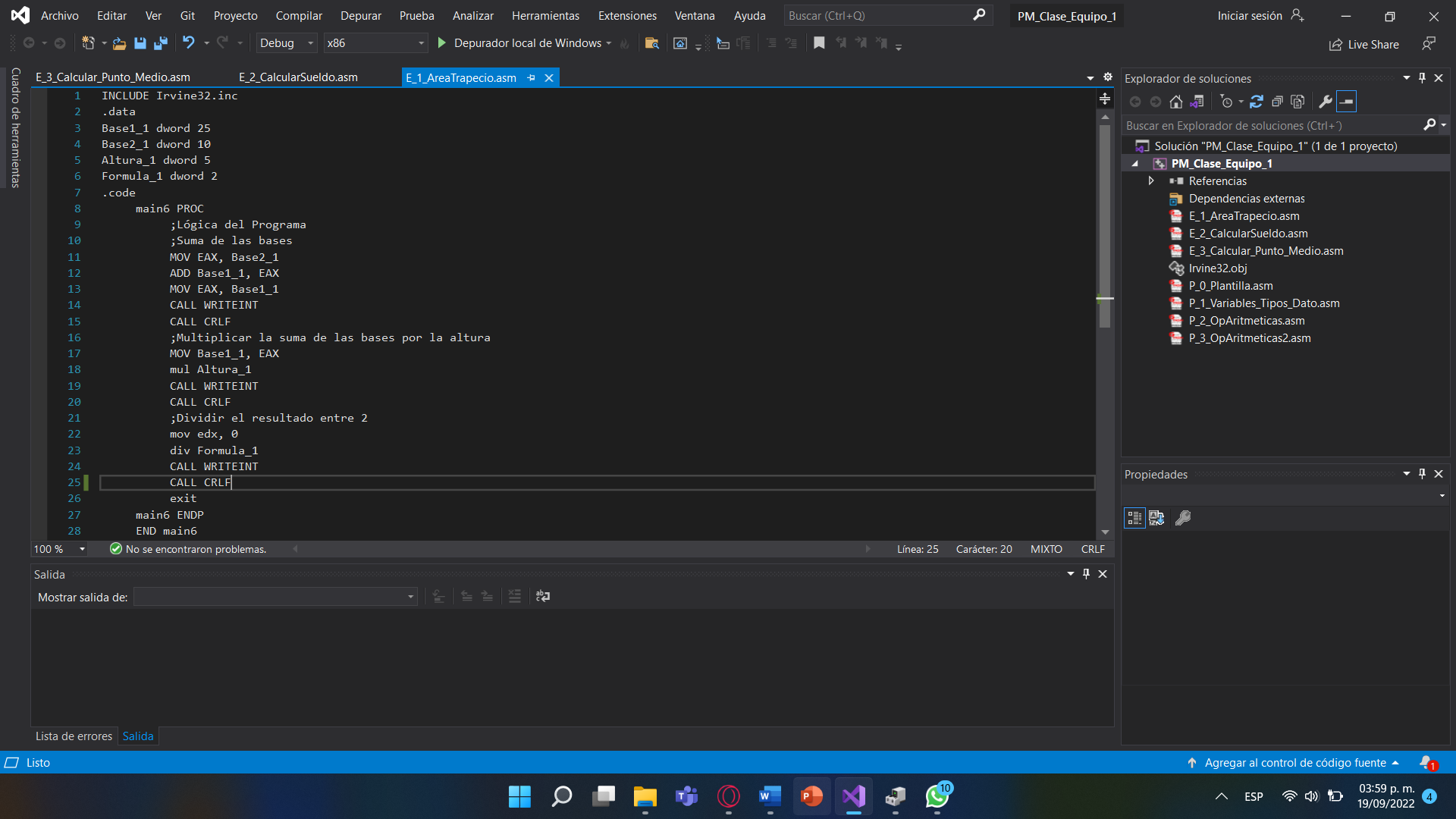
Sintaxis: DIV fuente

El divisor puede ser un byte o palabra y es el operando que se le da a la instrucción.

Si el divisor es de 8 bits se toma como dividendo el registro de 16 bits AX y si el divisor es de 16 bits se tomará como dividendo el registro par DX:AX, tomando como palabra alta DX y como baja AX.

Si el divisor fue un byte el cociente se almacena en el registro AL y el residuo en AH, si fue una palabra el cociente se guarda en AX y el residuo en DX.

**Desarrollo:**



Primero en DATA, se inicializan las variables que se usarían en el programa, la primera es Base1\_1 con valor de 25, Base\_2\_1 con valor de 10, Altura\_1 con valor de 2 y Formula\_1 con valor de 2 (esta última variable se utilizará para dividir entre 2 ), después en el .CODE

En la línea 11 se transferirá el valor de Base2\_1 a EAX.

En la línea 12 se sumará el Valor de EAX (la base 2) con Base1\_1 y se guardará en Base1\_1.

En la línea 13 se moverá el valor de base1\_1 (la suma) a EAX.

En la línea 14 se Imprime el resultado en consola con la Instrucción CALL WRITEINT, dándole así un valor positivo y entero.

En la línea 15 se realiza un salto de línea con la instrucción CALL CRLF.

En la línea 17 se mueve el valor de EAX a la variable Base1\_1 (esto cn el fin de guardar el valor de la suma).

En la línea 18 se multiplica EAX por el valor de Altura\_1.

En la línea 19 se imprime el resultado en valor entero.

En la línea 20 se realiza un salto de línea.

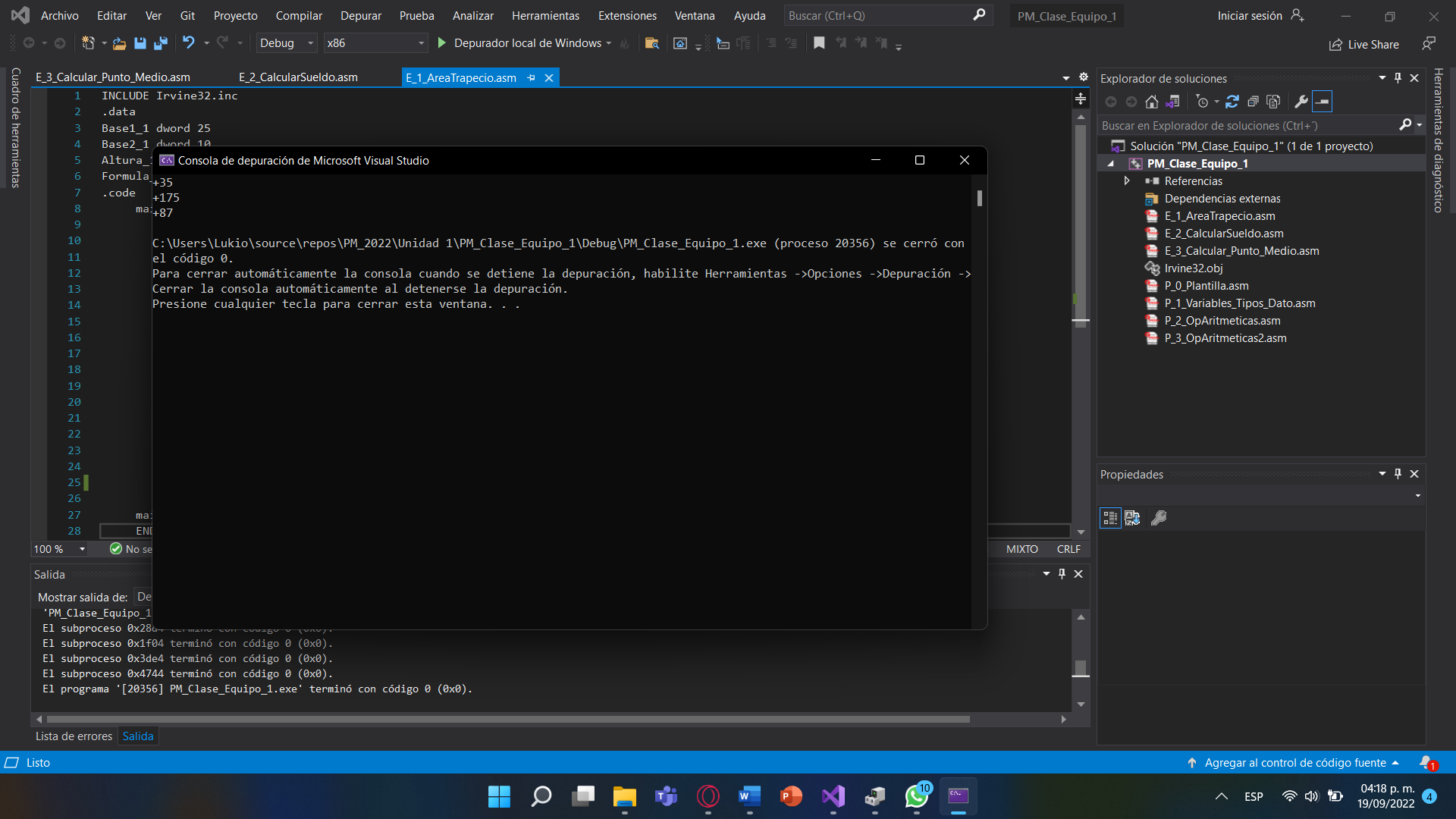
En la línea 22 se limpia EDX con la instrucción mov edx,0 dandole el valor 0 a edx, con esto evitamos el sobreflujo, y que la división se realice correctamente.

En la línea 23 realizamos la división con div, dividiendo el valor de EAX (la multiplicación de suma de bases x altura) entre Formula\_1, que este caso fue nuestra variable auxiliar para dividir entre 2.

En la línea 24 se imprime el resultado en valor entero.

En la línea 25 se realiza un salto de línea.

**Resultados:**



### Ejercicio 2. Sueldo

**Descripción**: Realizar un programa en lenguaje ensamblador en Visual Studio, que sea capaz de calcular el sueldo de un trabajador multiplicando el sueldo por las horas (sin limitantes, sueldo u horas exageradas).

**Introducción**:

Se requiere conocimiento de las siguientes instrucciones de lenguaje Ensamblador:

**MOV:**

Propósito: Transferencia de datos entre celdas de memoria, registros y acumulador.

Sintaxis:

MOV Destino,Fuente

Donde Destino es el lugar a donde se moverán los datos y fuente es el lugar donde se encuentran dichos datos.

Los diferentes movimientos de datos permitidos para esta instrucción son:

Destino: memoria. Fuente: acumulador

Destino: acumulador. Fuente: memoria

Destino: registro de segmento. Fuente: memoria/registro

Destino: memoria/registro. Fuente: registro de segmento

Destino: registro. Fuente: registro

Destino: registro. Fuente: memoria

Destino: memoria. Fuente: registro

Destino: registro. Fuente: dato inmediato

Destino: memoria. Fuente: dato inmediato

**CALL:** La instrucción CALL sirve para hacer llamadas a funciones en ensamblador.

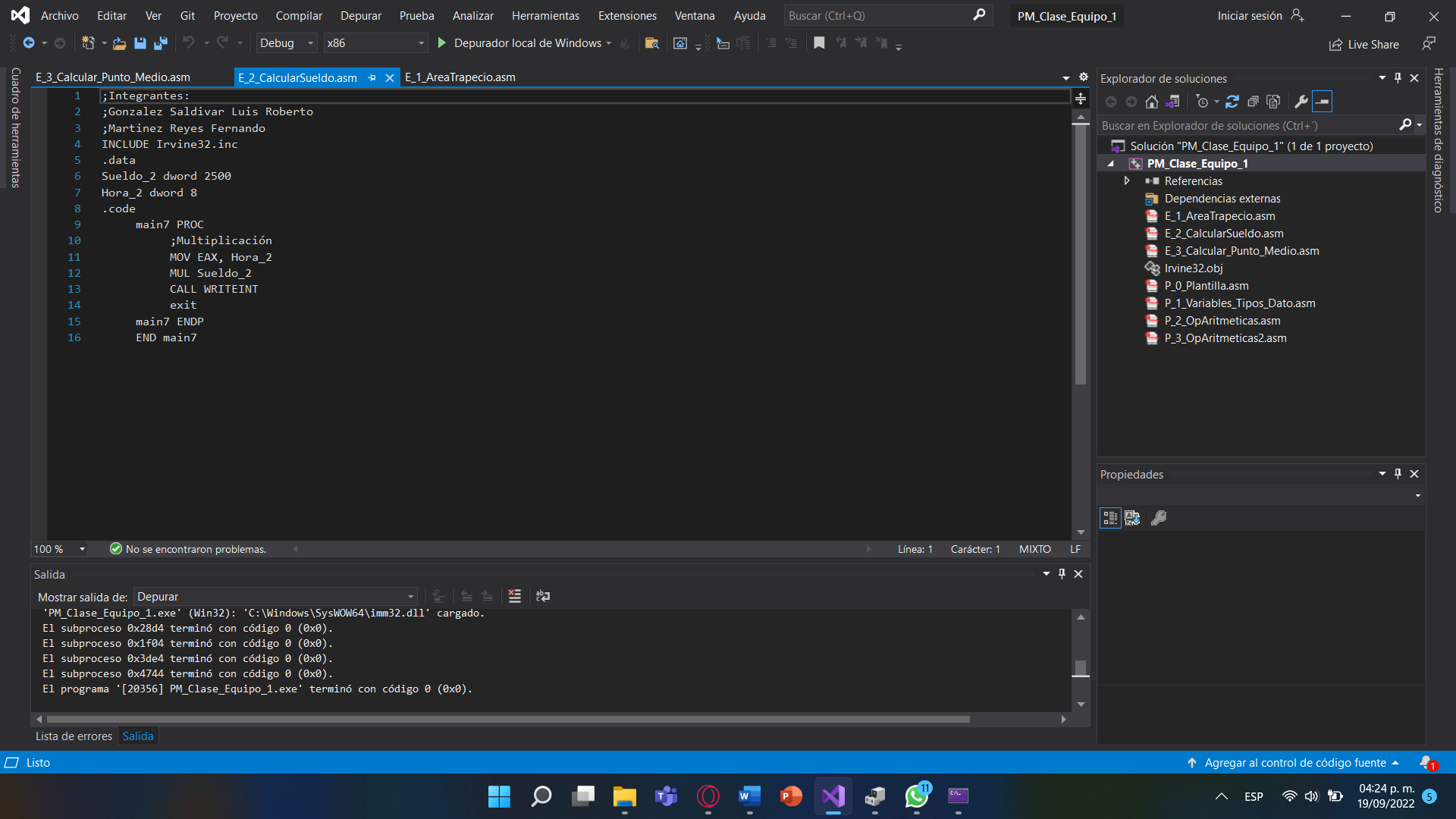
**MUL:** Multiplicación sin signo

Sintaxis: MUL fuente

El ensamblador asume que el multiplicando será del mismo tamaño que el del multiplicador, por lo tanto, multiplica el valor almacenado en el registro que se le da como operando por el que se encuentre contenido en AH si el multiplicador es de 8 bits o por AX si el multiplicador es de 16 bits.

Cuando se realiza una multiplicación con valores de 8 bits el resultado se almacena en el registro AX y cuando la multiplicación es con valores de 16 bits el resultado se almacena en el registro par DX:AX.

**Desarrollo:**



En el DATA se declaran y asignan valores a sus variables

En la línea 6 Se declara la variable Sueldo\_2 y se asigna un valor de 2500

En la línea 7 Se declara la variable Hora\_2 y se asigna un valor de 8

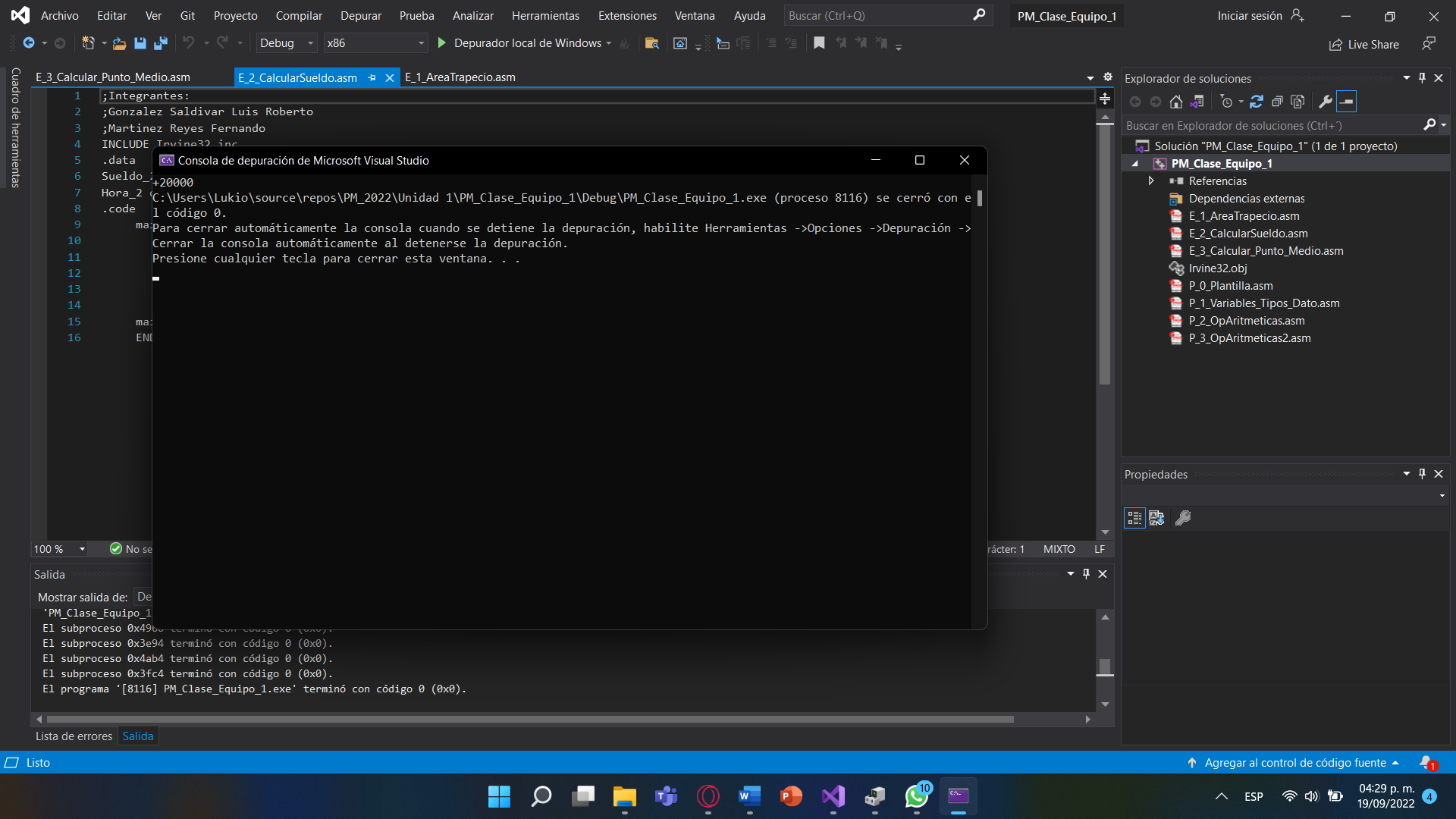
En el CODE se pondrá la lógica del programa.

En la línea 11 se transfiere el valor de Hora\_2 a EAX

En la línea 12 se multiplica EAX por Sueldo\_2

En la línea 13 se imprime el resultado en consola, dando así un valor entero y positivo.

**Resultados:**



### Ejercicio 3. Punto Medio

**Descripción**: Realizar un programa en lenguaje ensamblador en Visual Studio, que sea capaz de calcular el punto medio de 2 coordenadas (el resultado en entero).

**Introducción**:

Se requiere conocimiento de las siguientes instrucciones de lenguaje Ensamblador:

**MOV:**

Propósito: Transferencia de datos entre celdas de memoria, registros y acumulador.

Sintaxis:

MOV Destino,Fuente

Donde Destino es el lugar a donde se moverán los datos y fuente es el lugar donde se encuentran dichos datos.

Los diferentes movimientos de datos permitidos para esta instrucción son:

Destino: memoria. Fuente: acumulador

Destino: acumulador. Fuente: memoria

Destino: registro de segmento. Fuente: memoria/registro

Destino: memoria/registro. Fuente: registro de segmento

Destino: registro. Fuente: registro

Destino: registro. Fuente: memoria

Destino: memoria. Fuente: registro

Destino: registro. Fuente: dato inmediato

Destino: memoria. Fuente: dato inmediato

**ADD:**

Propósito: Adición de los operandos.

Sintaxis:

ADD destino, fuente

Suma los dos operandos y guarda el resultado en el operando destino.

**CALL:** La instrucción CALL sirve para hacer llamadas a funciones en ensamblador.

**DIV:** División sin signo

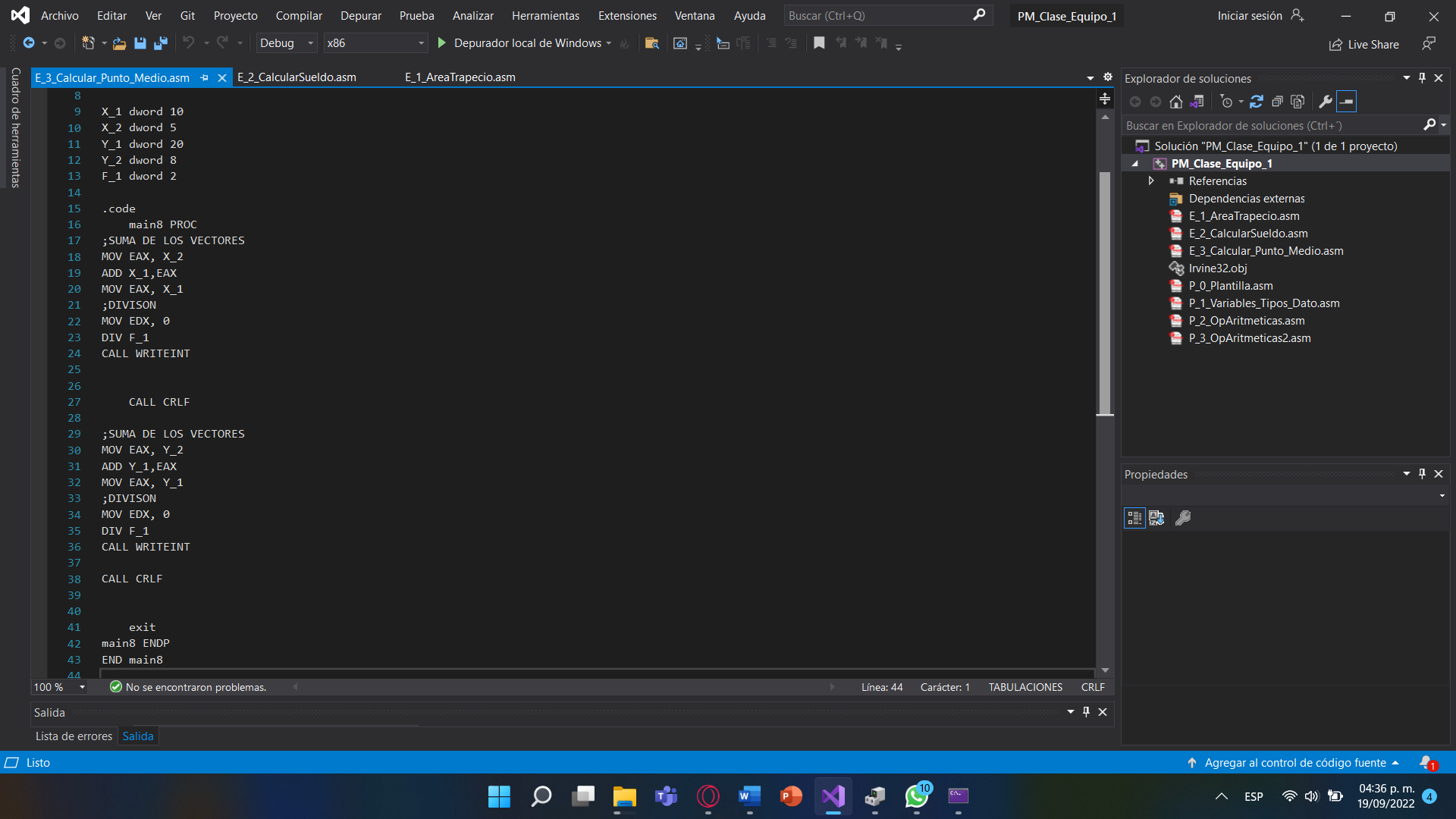
Sintaxis: DIV fuente

El divisor puede ser un byte o palabra y es el operando que se le da a la instrucción.

Si el divisor es de 8 bits se toma como dividendo el registro de 16 bits AX y si el divisor es de 16 bits se tomará como dividendo el registro par DX:AX, tomando como palabra alta DX y como baja AX.

Si el divisor fue un byte el cociente se almacena en el registro AL y el residuo en AH, si fue una palabra el cociente se guarda en AX y el residuo en DX.

**Desarrollo:**



En el DATA se declaran y asignan valores a sus variables:

En la línea 9 Se declara la variable X\_1 y se asigna un valor de 10.

En la línea 10 Se declara la variable X\_2 y se asigna un valor de 5.

En la línea 11 Se declara la variable Y\_1 y se asigna un valor de 20

En la línea 12 Se declara la variable Y\_2 y se asigna un valor de 8.

En la línea 13 Se declara la variable F\_1 y se asigna un valor de 2.

En el CODE se pondrá la lógica del programa:

En la línea 18 se transfiere a EAX el valor de X\_2 con la instrucción MOV.

En la línea 19 se suma el valor de EAX + X\_1 con la instrucción ADD (el resultado se guarda en X\_1).

En la línea 20 se transfiere el valor de X\_1 a EAX con la instrucción MOV.

En la línea 22 con la instrucción MOV se limpia EDX asignándole valor 0 (esto con el fin de evitar sobre flujo, y que la división se realice correctamente).

En la línea 23 se realiza la división del valor de EAX entre F\_1.

En la línea 24 se imprime el resultado en pantalla (siendo positivo y entero).

En la línea 27 se realiza un salto de línea.

En la línea 30 se transfiere a EAX el valor de Y\_2 con la instrucción MOV.

En la línea 31 se suma el valor de EAX + Y\_1 con la instrucción ADD (el resultado se guarda en Y\_1).

En la línea 32 se transfiere el valor de Y\_1 a EAX con la instrucción MOV.

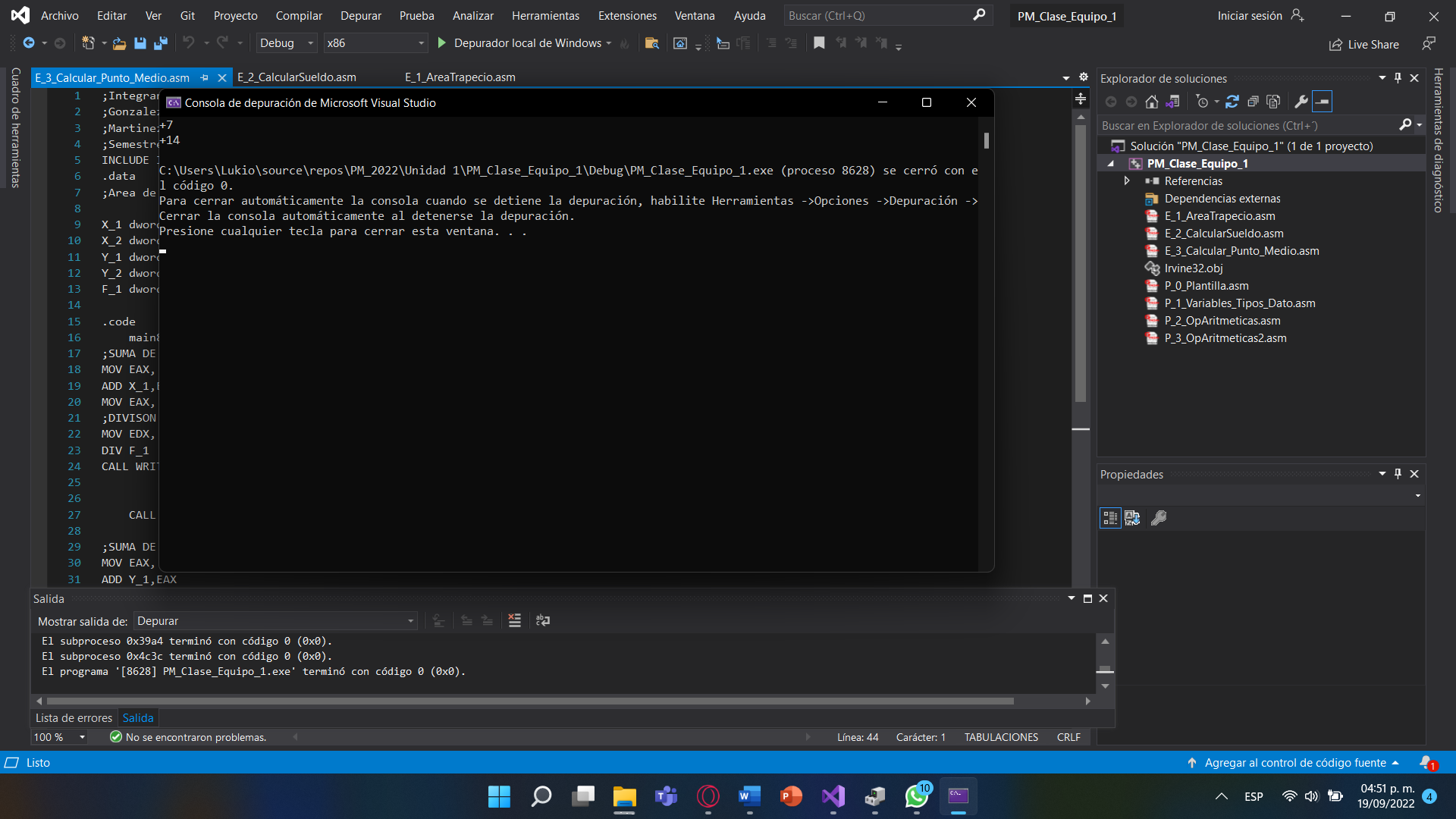
En la línea 34 con la instrucción MOV se limpia EDX asignándole valor 0 (esto con el fin de evitar sobre flujo, y que la división se realice correctamente).

En la línea 35 se realiza la división del valor de EAX entre F\_1.

En la línea 36 se imprime el resultado en pantalla (siendo positivo y entero).

En la línea 38 se realiza un salto de línea.

**Resultados:**



El primer resultado es la primer coordenada del punto medio, y el segundo resultado es la segunda coordenada del punto medio.