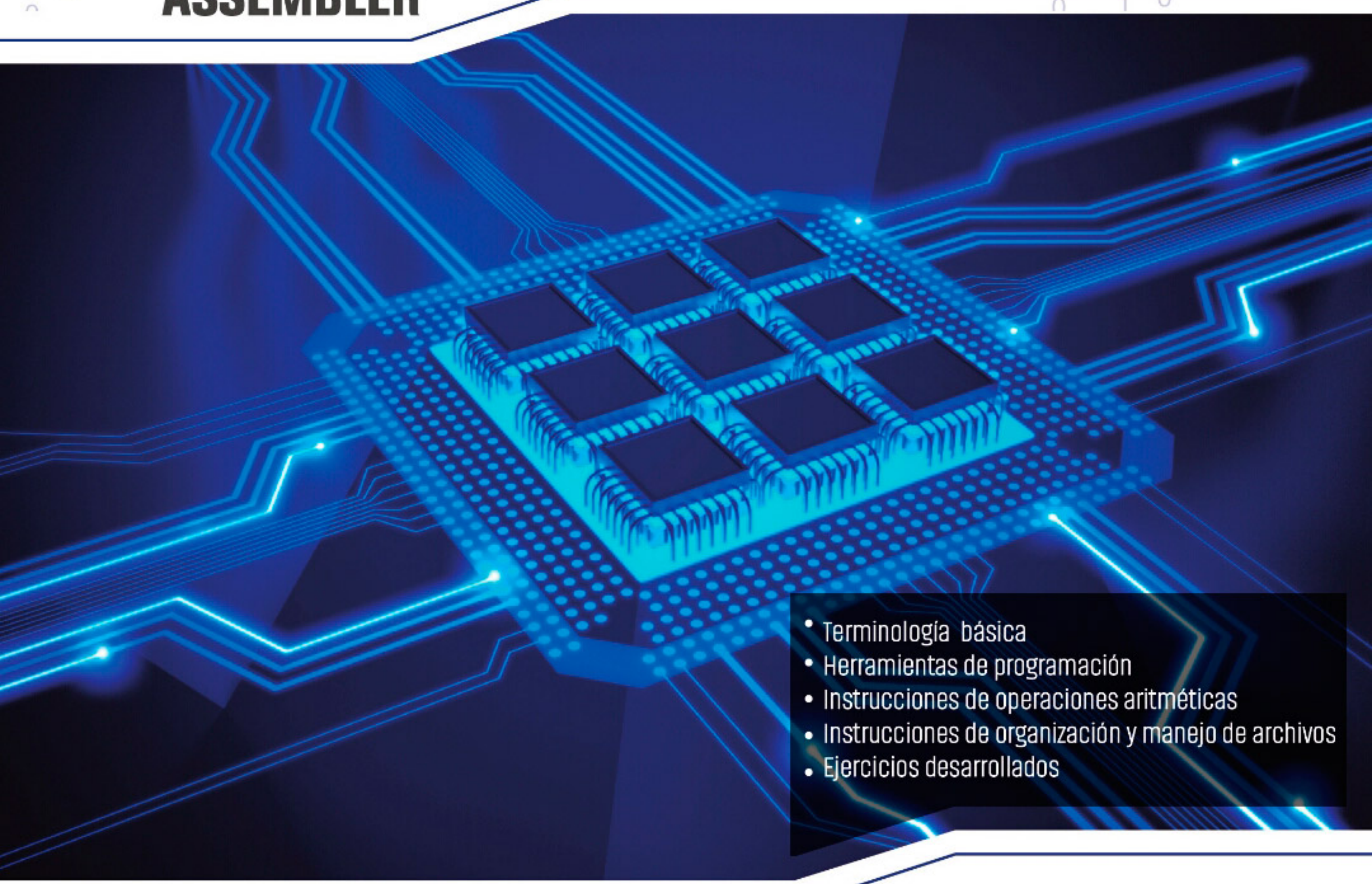


# LENQUAJE ENSAMBLADOR

Mg. Oswaldo Daniel Casazola Cruz

**ASSEMBLER**

- 
- Terminología básica
  - Herramientas de programación
  - Instrucciones de operaciones aritméticas
  - Instrucciones de organización y manejo de archivos
  - Ejercicios desarrollados





**Lenguaje ensamblador**

Autor: Oswaldo Daniel Casazola Cruz

© Derechos de autor registrados:

Empresa Editora Macro EIRL

© Derechos de edición, arte gráfico y diagramación reservados:

Empresa Editora Macro EIRL

Coordinación de edición:

Magaly Ramon Quiroz

Diseño de portada:

Brian Flores Uribe

Corrección ortográfica:

Yadira Cabello Villanueva

Diagramación:

Liyber Galindo

Edición a cargo de:

© Empresa Editora Macro EIRL

Av. Paseo de la República N.º 5613, Miraflores, Lima, Perú

☎ Teléfono: (511) 748 0560

✉ E-mail: [proyectoeditorial@editorialmacro.com](mailto:proyectoeditorial@editorialmacro.com)

🌐 Página web: [www.editorialmacro.com](http://www.editorialmacro.com)

Primera edición e-book: julio 2016

Disponible en: [macro.bibliotecasenlinea.com](http://macro.bibliotecasenlinea.com)

ISBN N.º 978-612-304-331-5

ISBN e-book N.º 978-612-304-489-3

Prohibida la reproducción parcial o total, por cualquier medio o método, de este libro sin previa autorización de la Empresa Editora Macro EIRL.



## OSWALDO DANIEL CASAZOLA CRUZ

Magíster en Ingeniería de Sistemas por la Universidad Nacional del Callao (UNAC). Fue presidente del Capítulo de Ingenieros Ambientales, Industriales, Sistemas, Economistas, Textiles, Informáticos y Computación del Consejo Departamental del Callao del Colegio de Ingenieros del Perú.

Realizó especialización en proyectos sociales de inversión pública: educación, salud y desarrollo de capacidades en la Pontificia Universidad Católica del Perú (PUCP), y en estadística para la investigación científica en la Universidad de San Martín de Porres (USMP). Imparte cátedra en universidades públicas y privadas, y es asesor y jurado de tesis. Sus investigaciones se centran en la optimización y el mejoramiento de sistemas complejos.







## DEDICATORIA

*A mi esposa Yemily y a mis hijas Camila y Luciana, quienes me regalaron su preciado tiempo para dárselo a este libro.*



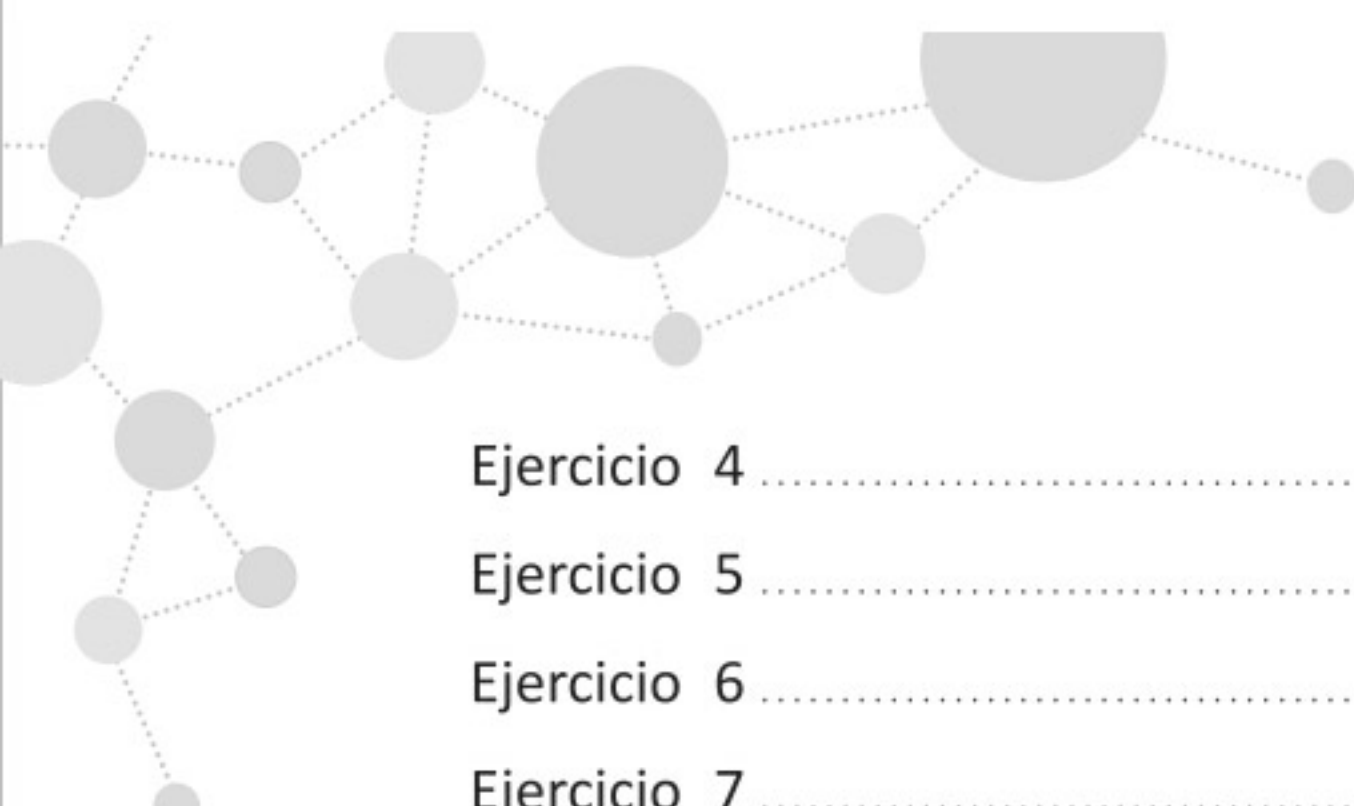






# ÍNDICE

Introducción.....	9
<b>CAPÍTULO 1: CONCEPTOS BÁSICOS .....</b>	<b>11</b>
1.1 Microprocesador.....	13
1.2 Microcontrolador.....	13
1.3 Arquitecturas de microprocesadores .....	21
<b>CAPÍTULO 2: HERRAMIENTAS DE PROGRAMACIÓN .....</b>	<b>25</b>
2.1 Lenguaje de bajo nivel.....	27
2.2 Interconexión de las unidades funcionales.....	27
2.3 Segmento.....	31
2.4 Interrupciones.....	33
2.5 Comentarios en lenguaje ensamblador.....	34
2.6 Transferencia de datos .....	36
<b>CAPÍTULO 3: OPERACIONES ARITMÉTICAS .....</b>	<b>39</b>
3.1 Suma y resta.....	41
Ejercicio 1 .....	41
Ejercicio 2 .....	43
Ejercicio 3 .....	47
3.2 Operandos lógicos.....	51
3.3 Multiplicación .....	51
3.4 División .....	52
<b>CAPÍTULO 4: ORGANIZACIÓN Y MANEJO DE ARCHIVOS .....</b>	<b>53</b>
4.1 La instrucción CMPS.....	55
4.2 La instrucción LOOP .....	56
4.3 Instrucciones de salto condicional .....	56
4.4 Operaciones CALL y RET .....	56
4.5 Guía de laboratorio usando MPLAB IDE 7.61 y PROTEUS 6.....	57
Ejercicio 1 .....	57
Ejercicio 2 .....	70
Ejercicio 3 .....	72



Ejercicio 4 .....	74
Ejercicio 5 .....	75
Ejercicio 6 .....	77
Ejercicio 7 .....	79
Ejercicio 8 .....	81
Ejercicio 9 .....	84
Ejercicio 10.....	84
Ejercicio 11.....	85
Ejercicio 12.....	87
<b>CAPÍTULO 5: EJERCICIOS DESARROLLADOS.....</b>	<b>93</b>
Ejercicio 1 .....	95
Ejercicio 2 .....	98
Ejercicio 3 .....	98
Ejercicio 4 .....	98
Ejercicio 5 .....	101
Ejercicio 6 .....	102
Ejercicio 7 .....	103
Ejercicio 8 .....	104
Ejercicio 9 .....	105
Ejercicio 10.....	106
Ejercicio 11.....	112
Ejercicio 12.....	123
Ejercicio 13.....	131
Ejercicio 14.....	139
Ejercicio 15.....	148
Anexos .....	151
Bibliografía.....	157



# INTRODUCCIÓN

Muchas veces, para los que inician el aprendizaje del lenguaje ensamblador, los términos «microprocesador» y «microcontrolador» son lo mismo; lo que genera muchos errores de programación y aplicación. Por ello, el presente libro permite la enseñanza del lenguaje ensamblador, con conceptos básicos que pretenden ubicarnos en el contexto de lenguaje máquina, que hace posible el uso de los equipos electrónicos en la actualidad. Cabe resaltar que la gran diferencia entre los dispositivos mencionados líneas arriba es su funcionalidad.

El capítulo uno permite la comprensión básica de los elementos que corresponden a los microcontroladores y las operaciones que realiza para llevar al lector hacia la terminología adecuada para adentrarse en el lenguaje ensamblador.

El capítulo dos presenta las herramientas de programación y los métodos de desplazamiento o direccionamiento, así como la transferencia de datos. Los capítulos tres y cuatro señalan la operación aritmética y lógica, y presentan ejercicios para tener una guía de apoyo en el aprendizaje de lenguaje ensamblador. En tanto, en el capítulo cinco se muestra la organización y el manejo de archivos con instrucciones y parámetros.

Finalmente, se presenta una guía de laboratorio o de prácticas, que servirá como apoyo para el aprendizaje de los estudiantes técnicos y universitarios, así como profesionales de las carreras de Ingeniería de Sistemas, Industrial, Eléctrica, Mecánica, Electrónica, Mecatrónica, Telecomunicaciones, Informática, y otras especialidades.





# capítulo 1

## CONCEPTOS BÁSICOS

### TEMAS:

- Microprocesador
- Microcontrolador
- Arquitecturas de microprocesadores

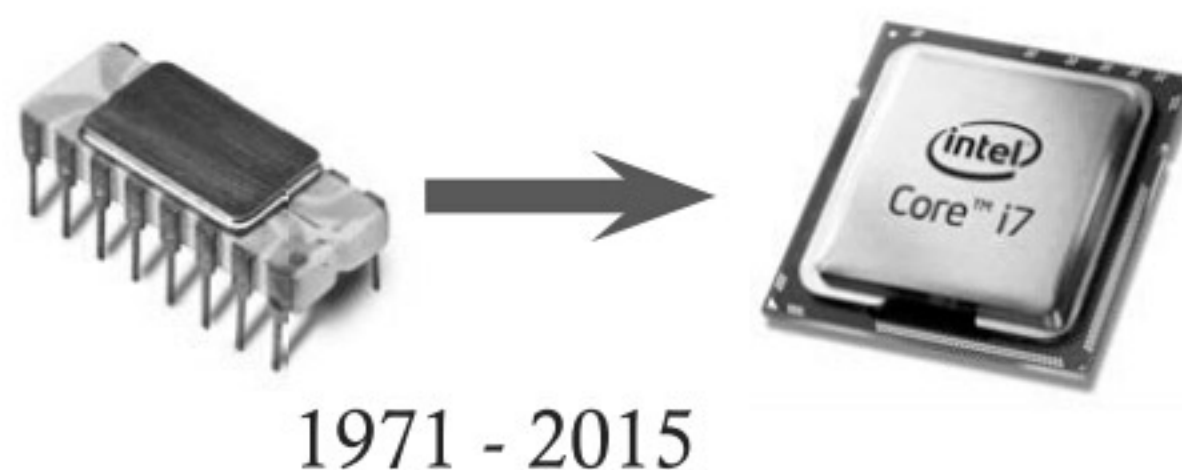






## 1.1 MICROPROCESADOR

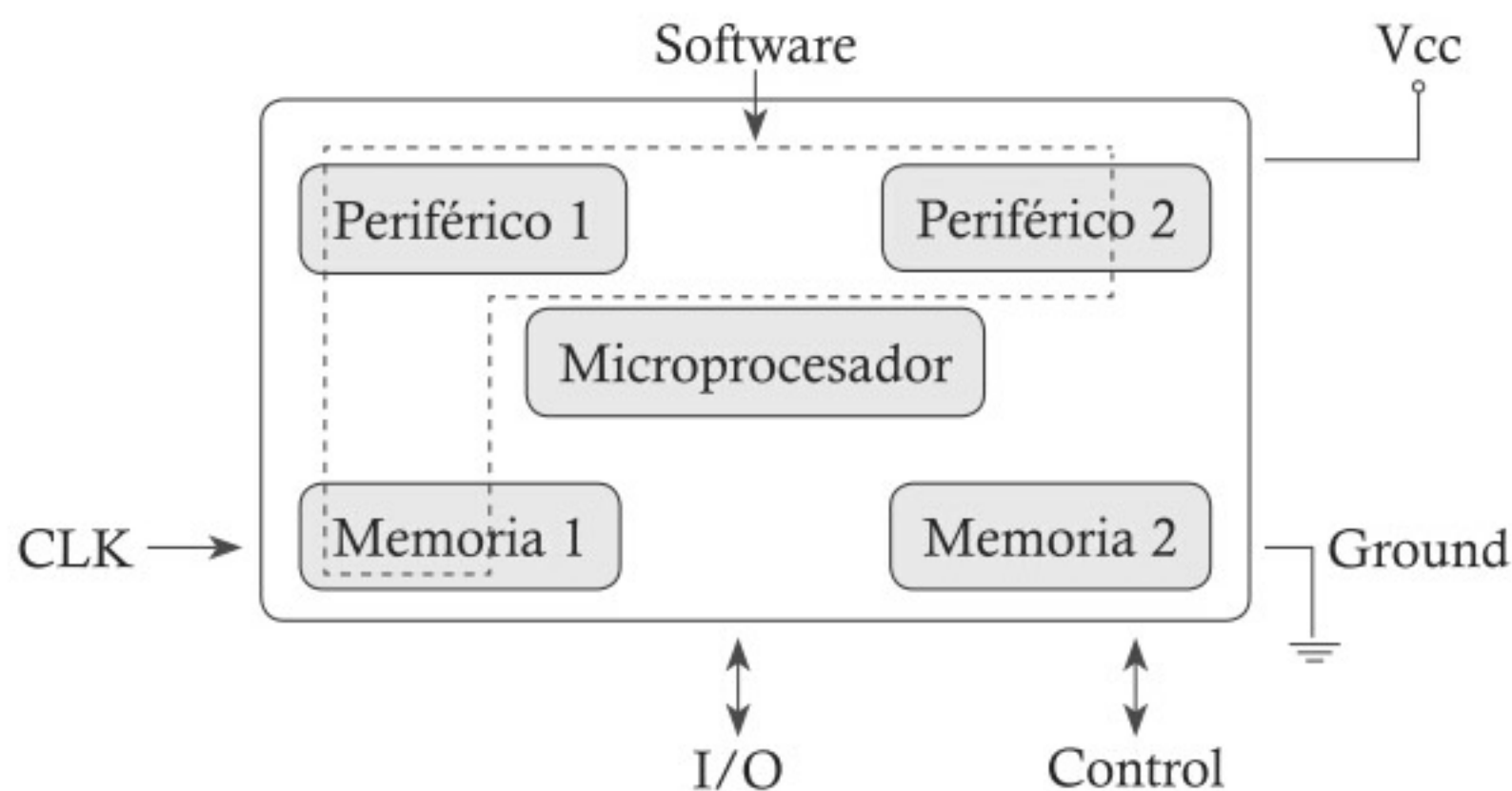
Es un circuito electrónico integrado para el cálculo y el control computacional, comúnmente conocido como la unidad central de procesamiento (CPU, por sus siglas en inglés *central processing unit*). Esta unidad realiza todas las operaciones aritméticas y lógicas sobre los datos; además, controla todos los procesos que se desarrollan en la computadora teniendo como componentes las partes lógicas unidad aritmético-lógica, unidad de control, memoria caché, registros de almacenamiento, unidad de ejecución y buses de datos control y dirección. Por ejemplo, para que se ejecute una instrucción, esta debe estar dentro de la unidad de control que enviará dicha instrucción por los buses de transmisión de datos para comunicarse con dispositivos periféricos. Los microprocesadores suelen tener forma de prisma achatado.



**Figura 1.1** Evolución de los microprocesadores

## 1.2 MICROCONTROLADOR

Es un circuito integrado programable que incluye las tres unidades funcionales de una computadora: CPU, memoria, y unidades de entrada y salida. Para ser utilizado son necesarios, básicamente, unos programas de control y un cristal de sincronización para controlar cualquier equipo electrónico: automóviles (control de frenos ABS), hornos microondas, lavadoras, teléfonos, etc., hasta convertirse en una computadora con un mínimo de dispositivos externos de apoyo. Para facilitar su uso, los microcontroladores distribuyen la velocidad y la flexibilidad.



**Figura 1.2** Esquema de un microcontrolador



### A. La memoria central

También es denominada memoria interna o memoria principal de la computadora. En esta memoria se almacenan los programas a ejecutarse. La información que almacena son las instrucciones que componen los programas que se utilizan, los datos que se ingresan y, finalmente, la información sobre el programa que realiza las funciones de control en las operaciones. Hay dos tipos de memoria: memoria RAM y memoria ROM.

### B. Memoria RAM (*random access memory*)

Está compuesta por chips alargados situados en la *mainboard* (tarjeta madre) contigua al procesador. Es una memoria de lectura y escritura de datos a gran velocidad. Es de carácter volátil, por tanto, depende del suministro eléctrico para no perder información. Es de acceso aleatorio: se puede acceder rápidamente a sus posiciones, pues no requiere de una lectura secuencial de los datos anteriores.

Al utilizar un programa, en primer lugar, se copia en la memoria RAM. Luego, el procesador lee una a una todas las instrucciones del programa. Finalmente, el procesador guarda en la memoria RAM todos los resultados de los cálculos. La memoria RAM determina cuántos programas puede ejecutar la computadora y a qué cantidad de datos puede acceder rápidamente.

### C. Memoria caché

La memoria caché (o memoria oculta) es pequeña y rápida. Está ubicada entre la memoria principal y el procesador de la computadora. Es más voluminosa y consume más energía que la memoria RAM, y tiene características similares a esta.

### D. Memoria ROM (*read only memory*)

En las computadoras, se denomina memoria de sistema básico de entrada/salida (BIOS, por sus siglas en inglés *basic input/output system*). Es un conjunto de chips con información sobre la configuración de los dispositivos internos de la computadora y sobre los dispositivos externos que están conectados para ordenar y controlar la entrada/salida de datos.

La memoria ROM no es volátil, es de lectura de datos y de acceso aleatorio; por ello, ya está grabada cuando se fabrica la computadora. Hay distintos tipos de memoria ROM: PROM, EPROM. Entre estas evoluciones, se pueden destacar estos hitos:

- ENIAC (*Electronic Numeric Integrator and Calculator*).
- KANM (*Electronic Discrete Variable Automatic Computer*).
- El CAMR 7030 (conocido como *Stretch*).
- El IBM 360/91.
- El JLMM 6600.
- Progresos.



Jhon Cocke propuso la segmentación superescalar donde el microprocesador puede ejecutar muchas instrucciones a la vez. Como consecuencia de esto, los nuevos microprocesadores han mejorado y evolucionado debido a la arquitectura de computadores más que el avance de la nanotecnología. Los primeros procesadores desperdiciaban el 90 % de sus componentes; ahora, siempre están trabajando lo que representan: microprocesadores rápidos y productivos.

### Funcionamiento

El microprocesador ejecuta instrucciones en varias fases:

- **Preselección (*prefetch*):** Prelectura de la instrucción desde la memoria principal.
- **Selección (*fetch*):** Envío de la instrucción al decodificador.
- **Decodificación (*decode*) de la instrucción:** Determinación del tipo de instrucción y, por tanto, lo que se debe hacer. Lectura de operandos (si los hay).
- **Ejecución (*execute*):** Lanzamiento de las máquinas de estado que llevan a cabo el procesamiento.
- **Escritura (*store*):** Resultados en la memoria principal o en los registros.

Cada fase se ejecuta en uno o varios ciclos de CPU, dependiendo de la arquitectura del procesador y según el grado de segmentación.

El microprocesador se conecta al cristal de cuarzo que oscila a ritmo constante, y determina la duración de estos ciclos o pulsos en un segundo siempre mayor al tiempo requerido para realizar la tarea individual (de un solo ciclo) de mayor coste temporal.

El *Firmware* es un software con la lógica de más bajo nivel que permite controlar los circuitos electrónicos de cualquier dispositivo que se encuentra grabado en una memoria tipo ROM; y, al estar integrado en la electrónica del dispositivo, es en parte hardware. Recibe los comandos externos hacia el dispositivo y ejecuta dichas órdenes.

Se puede ubicar en el software de diversos dispositivos periféricos y en cualquier circuito integrado, también se encuentra en el BIOS de una computadora para activarla desde su encendido y en el inicio del sistema operativo. Responde a eventos externos que permiten el intercambio de órdenes entre distintos dispositivos del sistema.

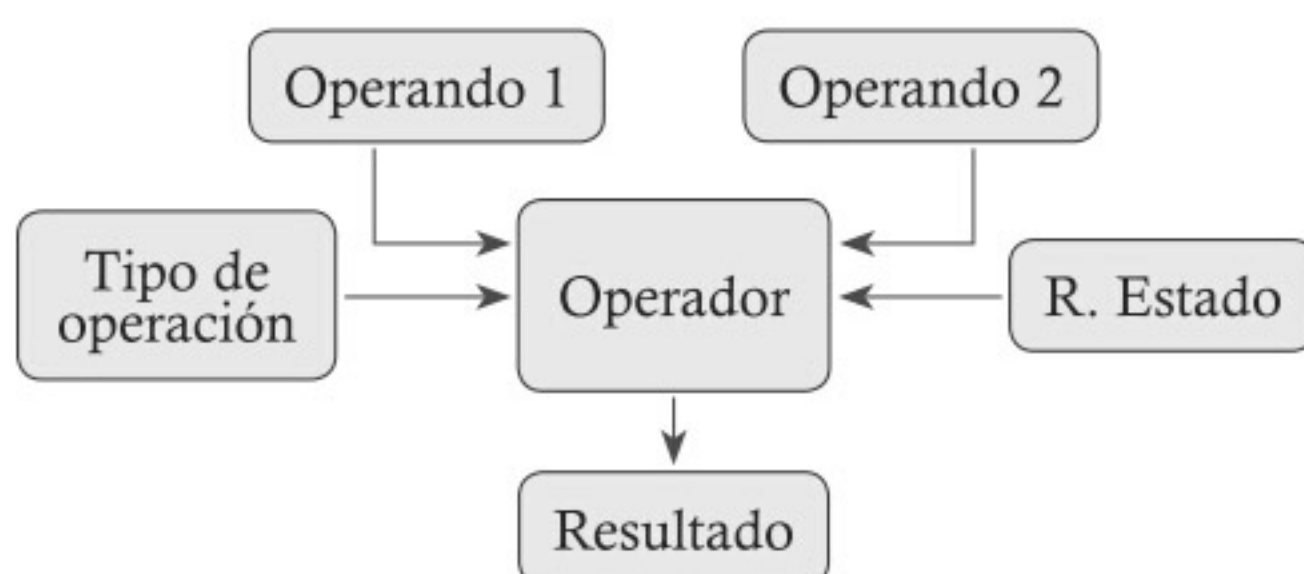
### E. Unidad aritmética lógica (*arithmetic logic unit* o ALU)

La ALU es un circuito digital que se encarga de realizar los cálculos de las siguientes operaciones: suma, resta, multiplicación, división, comparación (mayor que, menor que, igual a), y aquellas que trabajan con dígitos binarios (1-0, se conoce como operaciones lógicas: AND, NOR, NOT, NAND, OR, X-OR, etc.) entre dos números u operandos.

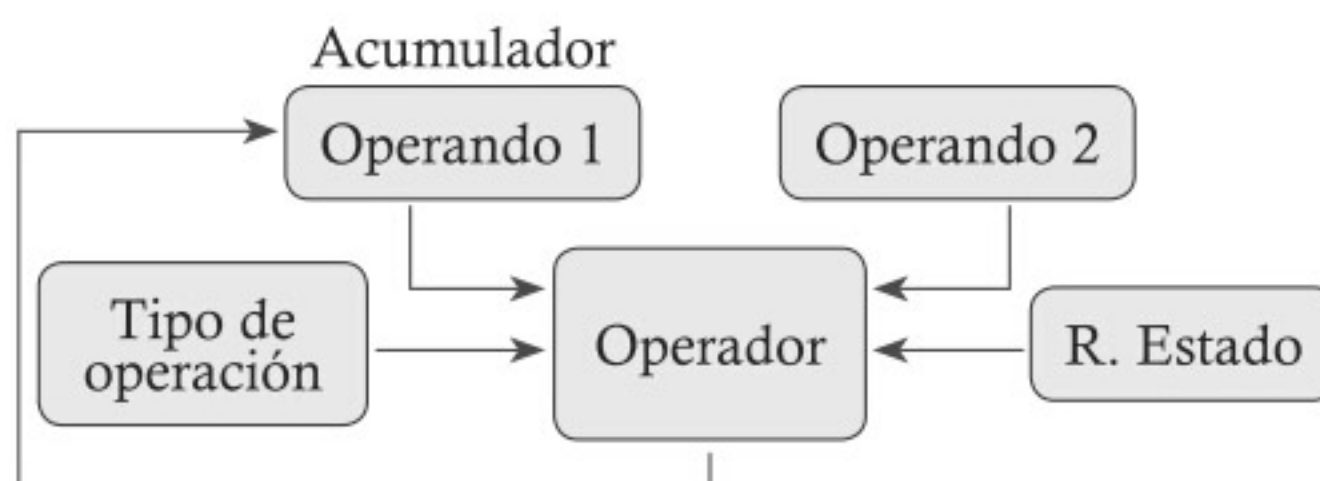


Para realizar dichas operaciones, y antes de ser llevados a la memoria o a las unidades de entrada y de salida, los operandos y los resultados se almacenan en el registro de trabajo o de entrada REN (solo para operandos) o en el registro acumulador (operando y resultado).

En la figura 1.3, se muestran los elementos de la ALU y los elementos que intervienen (tres registros). En la figura 1.4, se aprecia la ALU con acumulador.



**Figura 1.3** Símbolo esquemático ALU



**Figura 1.4** ALU con acumulador

## F. Componentes de la ALU

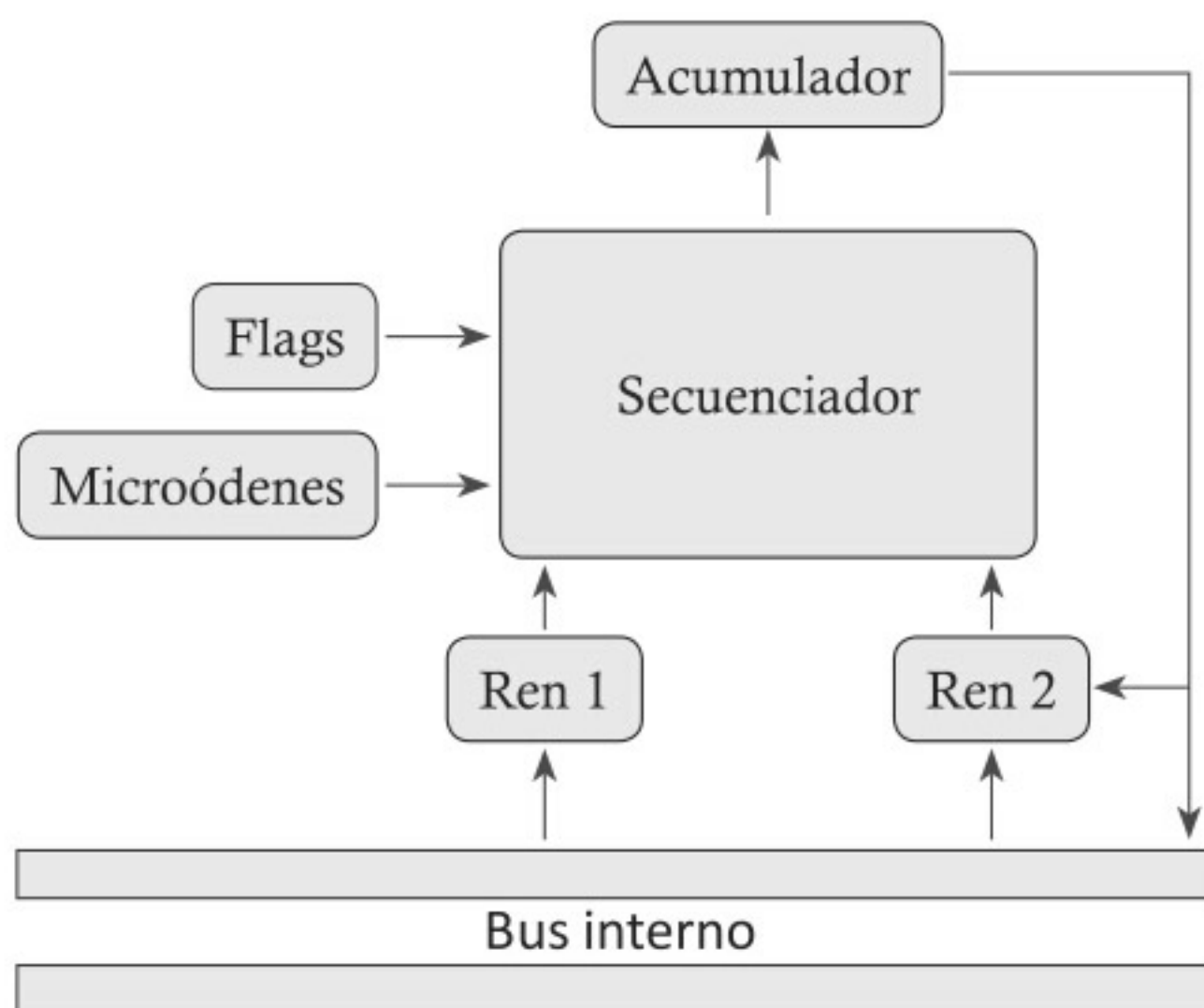
La ALU contiene el circuito operacional, los registros de entradas, el registro acumulador y un registro de estados, conjuntos de registros que permiten la realización de cada una de las operaciones solicitadas.

El circuito operacional presenta circuitos electrónicos que sirven para realizar las operaciones procedentes de los registros de entradas (o de trabajo). Están apoyados de un selector de operaciones dirigidas por microórdenes del secuenciador de la unidad de control para su ejecución. Los resultados de las operaciones se almacenan en el registro acumulador, y para realizar operaciones en cadena, se encuentra conectado con los registros de entradas.

Los resultados se envían a la memoria principal (RAM) o a algún periférico a través de un bus de datos del sistema.

Los registros de estado (*flag*), en estos registros de memoria, almacenan la última operación realizada para tenerla en cuenta en operaciones posteriores porque tiene que quedar constancia del resultado (cero, positivo o negativo). Los registros de estado más comunes son los siguientes:

- **Z (zero flag):** Cuando el resultado es cero.
- **N (negative flag):** Cuando el resultado es negativo.
- **V (overflow flag):** Cuando el resultado supera el número de bits que puede manejar la ALU.
- **P (parity flag):** Paridad del número de 1 en los datos.
- **I (interrupt flag):** Cuando se ha producido un error.
- **C (carry flag):** Acarreo de la operación realizada (registro de estado).



**Figura 1.5** Diagrama de la ALU



## G. Operaciones básicas

Se tienen las siguientes operaciones que la mayoría de ALU puede realizar:

- Operaciones lógicas (AND, NOT, OR, XOR).
- Operaciones aritméticas de números enteros.
- Operaciones de desplazamiento de bits (hacia la izquierda o derecha, con o sin extensión de signo) que pueden ser interpretadas como multiplicaciones o divisiones por 2.

## H. Operaciones complejas

Si se desea calcular un número elevado a un exponencial, se examinarán las siguientes opciones para implementar esta operación:

- Diseñar una ALU muy compleja que calcule el exponencial de cualquier número en un solo paso. Llamada cálculo en un solo ciclo de reloj.
- Diseñar una ALU compleja que calcule el exponencial con varios pasos. Llamada cálculo interactivo. Generalmente se confía en el control de una unidad de control compleja con microcódigo incorporado. Por lo tanto, cuanto más compleja sea la operación, se usará más espacio en el procesador, y más energía se disipará (volviendo más costosa la ALU).

## I. Instrucciones de la ALU

Las instrucciones que son capaces de entender y ejecutar un microprocesador se clasifican según su función. Estas son las siguientes:

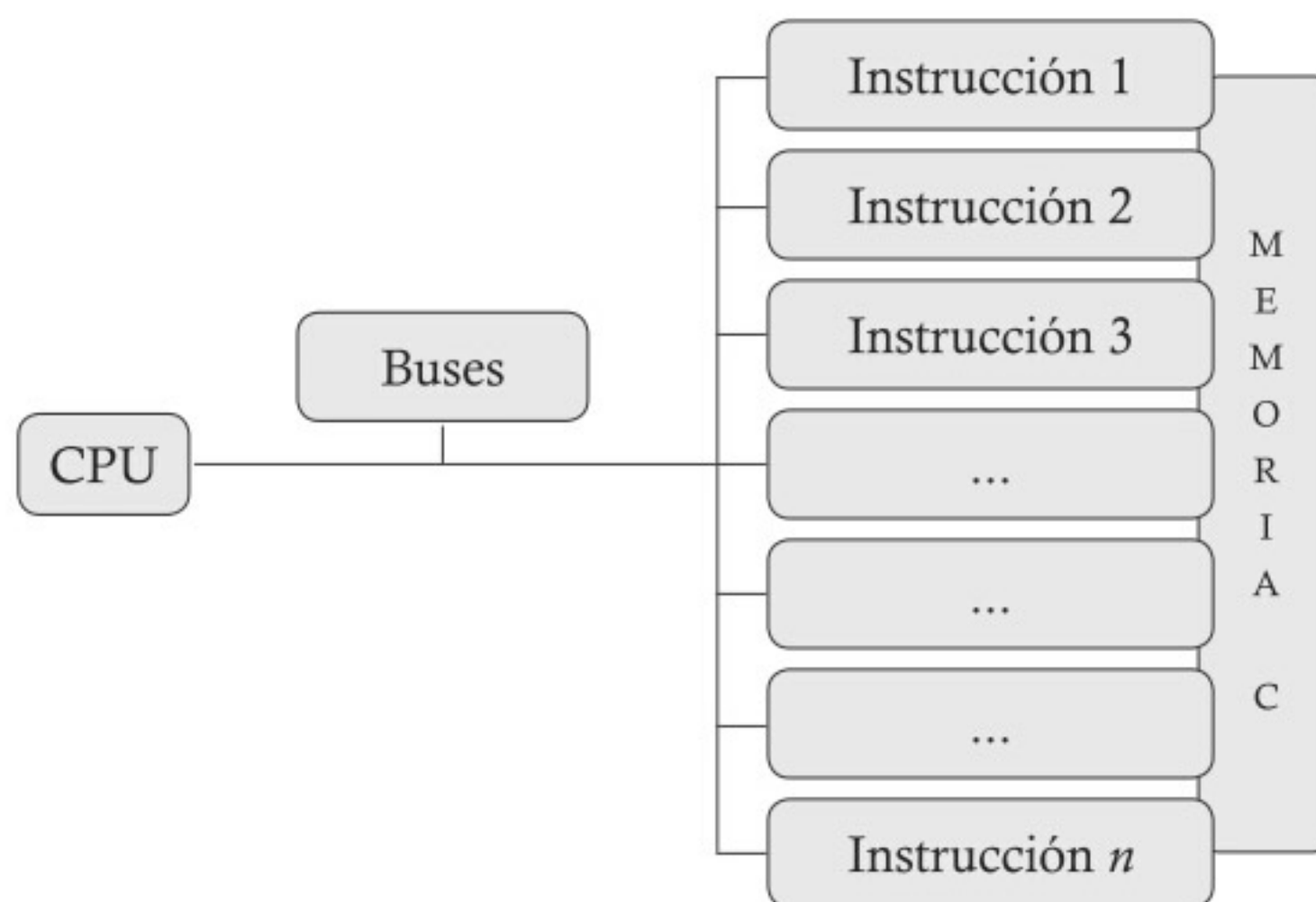
- **Instrucciones de cálculo:** Ejecutan ciertas operaciones aritméticas o ciertas operaciones lógicas, así como desplazamiento y rotación de bits.
- **Instrucciones de transferencia de datos:** Permiten el cambio de datos (elementos de entrada y salida) entre la memoria y los registros internos del microprocesador, o entre registros del mismo microprocesador.
- **Instrucciones de transferencia del control del programa:** Admiten romper la secuencia lineal del programa y saltar a otro punto del mismo.
- **Instrucciones de control:** Sirven para el control del propio microprocesador (activar interrupciones, pasar órdenes al coprocesador matemático, etc.).

Este ciclo de instrucciones inicia con la búsqueda de instrucciones, sigue con la decodificación de las instrucciones, la búsqueda de operandos, la ejecución de la instrucción y finaliza con el almacenamiento del resultado.



## Ejecución de las instrucciones

El microprocesador realizará las tareas que las instrucciones (almacenadas en la RAM) señalen. Véase la figura 1.6 para apreciar al detalle la descripción anterior:



**Figura 1.6** Ejecución de las instrucciones

## J. Velocidad

Los factores que determinan la velocidad de la CPU son los siguientes:

- El indicador de frecuencia (ciclos por segundo) de un microprocesador medido en hercios (hertz).
- El número de instrucciones que son necesarias para realizar una tarea.
- El número de instrucciones por ciclo (IPC o *instructions per cycle*).

## K. *Pipelining* o segmentación

Usado para aumentar el rendimiento de los microprocesadores y algunos sistemas electrónicos digitales. Son cálculos que son sincronizados con el reloj cada cierto tiempo para que el tramo con más carga o retardo computacional entre dos registros de reloj se reduzca. Por ello, a mayor tiempo o retraso entre registros, será menor la frecuencia máxima de trabajo; y a menor tiempo o retraso, mayor será la frecuencia de trabajo.

## L. Unidad de punto flotante (*floating point unit* o FPU)

La FPU también realiza operaciones aritméticas entre dos valores; no obstante, lo hace para números en representación del punto flotante. Esto es mucho más complicado



que la representación de complemento a dos, empleada en una típica ALU. Para realizar estos cálculos, una FPU tiene incorporados varios circuitos complejos. Esto incluye algunas ALU internas.

### M. Unidad de control

La unidad de control (*control unit* o CU) extrae (*to fetch*), decodifica y ejecuta las instrucciones almacenadas en la memoria principal, genera órdenes para los diversos componentes del microprocesador, temporiza las distintas operaciones para su ejecución en función de la instrucción o en la etapa de la instrucción que se esté ejecutando. Finalmente, se prepara para leer la siguiente posición de memoria y vuelve a extraer la siguiente instrucción.

Existen unidades de control cableadas (circuito de lógica secuencial, control de estado, lógica combinacional y emisión de reconocimiento de señales de control) y unidades de control microprogramadas (almacenadas en una micromemoria con acceso secuencial para su ejecución) para máquinas sencillas y máquinas más complejas, respectivamente.

Para realizar su función a cabalidad, la unidad de control consta de los siguientes elementos:

- **Contador de programa:** Almacena la dirección de memoria de la siguiente instrucción a ejecutar (toma la dirección de la primera instrucción e incrementa su valor en uno) o la dirección que indique la propia instrucción (si está en curso).
- **Registro de instrucciones:** Como se explicó líneas arriba, las operaciones y operandos se almacenan en registros, estos contienen la instrucción que se está ejecutando, llevan consigo el código de operación o las direcciones de memoria de estos operandos.
- **Decodificador:** Extrae, analiza y ejecuta el código de operación de la instrucción en curso para emitir las señales necesarias al resto de elementos del microprocesador a través del secuenciador.
- **Reloj:** Brinda la frecuencia constante mediante una sucesión de impulsos eléctricos o ciclos por cada instrucción. El reloj del sistema es quien sincroniza la velocidad medida en hercios de las operaciones dentro de la computadora.
- **Secuenciador:** Se denomina de esta manera porque genera microórdenes que, junto a los impulsos de reloj, permiten la ejecución secuencial de la instrucción (almacenada en el registro de instrucción).



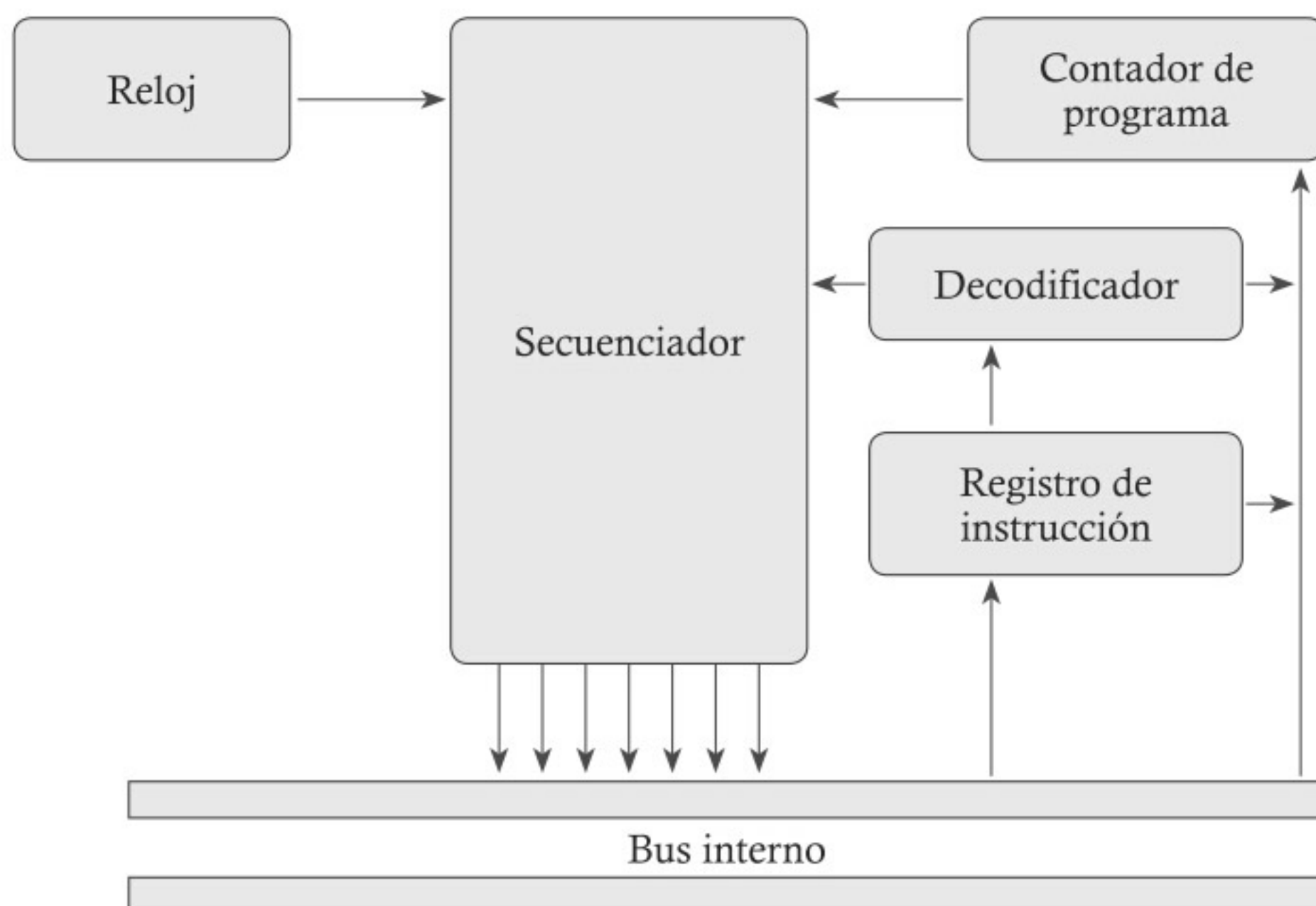


Figura 1.7 Diagrama CU

### 1.3 ARQUITECTURAS DE MICROPROCESADORES

La arquitectura de microprocesadores es el diseño conceptual y la estructura operacional fundamental de un sistema de computadora. Es decir, es un modelo y una descripción funcional de los requerimientos y las implementaciones de diseño para varias partes de una computadora, con especial interés en la forma en que la unidad central de proceso (CPU, por sus siglas en inglés) trabaja internamente y accede a las direcciones de memoria.

También suele definirse como la forma de seleccionar e interconectar componentes de hardware para crear computadoras según los requerimientos de funcionalidad, rendimiento y costo.

«La segmentación de instrucciones es similar al uso de una cadena de montaje de manufactura, pues el producto pasa por etapas de producción antes de tener el producto terminado. Cada etapa o segmento de la cadena está especializada en un área específica de la línea de producción y lleva a cabo siempre la misma actividad. Esta tecnología es aplicada en el diseño de procesadores eficientes. A estos procesadores se les conoce como *pipeline processors* que está compuesto por una lista de segmentos lineales y secuenciales en donde cada segmento lleva a cabo una tarea o un grupo de tareas computacionales» (Valdés, 2007).



Los datos del exterior son procesados en el sistema. Mediante múltiples operaciones, la computadora produce información que será usada externamente. La computadora realiza operaciones con los datos que tiene almacenados en la memoria, produce nuevos datos o información para uso externo.

Las arquitecturas y los conjuntos de instrucciones presentan la siguiente clasificación:

- **Almacenamiento de operandos en la CPU:** Ubicación de los operandos aparte de la memoria.
- **Número de operandos explícitos por instrucción:** Cantidad de operandos. Normalmente son 0, 1, 2 y 3.
- **Posición del operando:** Dirección de memoria o direccionamiento del operando dentro de los registros internos del CPU.
- **Operaciones:** Es el conjunto de instrucciones a realizarse con los operandos.

El núcleo o kernel (cuya raíz *kern* tiene origen germánico) se presenta en algunos sistemas operativos como núcleo ausente debido a que en la arquitectura de dichos sistemas operativos se tiene un solo modo de ejecución. Con esto en cuenta, el núcleo es solamente la parte primordial del sistema operativo que lo posea, pues, se encarga de administrar el acceso seguro, permisos y tiempos de los dispositivos de hardware interno o periférico; facilitando una interfaz uniforme a todo el hardware interconectado.

### 1.3.1. MODELOS DE ARQUITECTURA DE MICROPROCESADORES

El CISC (*complex instruction set computer*) es un modelo que presenta instrucciones de gran tamaño y operaciones complejas de los operandos ubicados en los registros internos. Este modelo es anterior a la arquitectura RISC.

Por ello, este modelo de arquitectura, por su complejidad, no permite fácilmente el paralelismo entre instrucciones, solucionando dicha dificultad al optimizar las instrucciones complejas mediante la conversión en múltiples pequeñas instrucciones RISC, mejorando el rendimientos de los sistemas CISC.

Los CISC son los primeros procesadores antes del desarrollo de los RISC. Este modelo de arquitectura permitió que la tecnología se abra paso desarrollando los primeros arquetipos como Motorola 68000, Zilog Z80 y toda la familia Intel x86, usada en la mayoría de las computadoras personales del mundo.

Es importante remarcar que la nomenclatura CISC se crea para diferenciarse de RISC, pues, los creadores de esta última arquitectura son quienes la señalan como compleja.

El RISC (*reduced instruction set computer*) se crea como necesidad a tener instrucciones reducidas pues los nuevos microprocesadores presentan instrucciones de tamaño fijo y con un reducido número de formatos, donde las instrucciones de carga y almacenamiento acceden a la memoria por datos. Dichos procesadores tienen muchos registros multipropósito.



La finalidad de la creación de la arquitectura RISC es reducir el acceso a la memoria. Dicha optimización solo se lograría realizando la segmentación y el paralelismo en la ejecución de instrucciones.

Actualmente, RISC es tendencia en la creación de microprocesadores como PowerPC, DEC Alpha, MIPS, ARM. Las microcomputadoras requerían que las tareas se realizarán en el menor tiempo posible, por ello, el diseño de CPU favorece, mediante la filosofía RISC, esta mejora mediante simple y pequeñas instrucciones, pero se basaron en CISC, como el x86, que es un equipo de escritorio, luego tradujeron las instrucciones complejas en nuevas instrucciones más simples que mejoren el tiempo de ejecución de este procesador.

El nuevo diseño del CPU ejecutaba las instrucciones ignorando las nuevas características incluidas que los diferenciaba del tradicional diseño; es decir, seguía siendo lenta pues el número total de acceso a memoria seguía siendo el mismo. La velocidad del procesador era más alta comparada con la de la memoria. Esto despertó el interés por generar una nueva técnica o arquitectura moderna de carga-almacenamiento que permita reducir el procesamiento de instrucciones dentro del CPU, que en consecuencia disminuya los accesos a memoria.

El *digital signal processor* (DSP) o sistema de procesamiento digital de señal es la arquitectura que en la actualidad es mayormente usada en telefonía móvil con proceso de señal digital.

Las señales que ingresan en el microprocesador ahora lo hacen en formato digital con secuencias o señales físicas que se transforman de lo que comúnmente se conoce como señal analógica a señal digital mediante transductores o convertidores. Este proceso que el sistema electrónico realiza se entiende como la aplicación de operaciones matemáticas que se representa de forma digital; después de estas operaciones matemáticas las señales en digital se reconvierten en señales físicas mediante transductores.

Se presenta esta arquitectura también, pues, un microcontrolador puede ser el core del sistema de procesamiento digital o procesador de propósito general, y que sirve en la actualidad como solución óptima a la carga computacional extremadamente intensa. Los fabricantes de DSP son Texas Instruments, con la serie TMS320; Motorola, con las series DSP56000, DSP56100, DSP56300, DSP56600 y DSP96000; Lucent Technologies (anteriormente AT&T), con las series DSP1600 y DSP3200; y Analog Devices, con las series ADSP2100 y ADSP21000.



## Aplicaciones

«El DSP se utiliza en diferentes aplicaciones: desde sistemas de radar militar hasta la electrónica de casa. Una primera tarea al momento de elegir un DSP es considerar la importancia, el costo, la integración, la facilidad de desarrollo, el consumo y otros factores para las necesidades de determinada aplicación en particular.

Asimismo es utilizado en las grandes aplicaciones con un gran volumen de producción como las de telefonías móviles y equipos de redes de telecomunicación, en donde el costo y la integración son de la mayor importancia. En sistemas portátiles, alimentados por baterías, el consumo es crítico por lo que los procesos internos deben ser óptimos. Los DSP mejoran los accesos a la memoria y contribuyen a disminuir el consumo de energía.

A pesar de que estas aplicaciones casi siempre implican el desarrollo de hardware y software a medida, el enorme volumen de producción justifica el esfuerzo extra para su desarrollo.

Una segunda clase de aplicaciones englobaría a aquellas que procesan un gran volumen de datos mediante algoritmos complejos. Ejemplos de este tipo incluye la exploración sonar y sísmica, donde el volumen de producción es bajo. Los algoritmos de este tipo son más exigentes, porque el diseño del producto es más complejo.

En algunos casos, más que diseñar el propio hardware y software, el sistema se construye a partir de placas de desarrollo de catálogo, y el software a partir de librerías de funciones ya existentes» (Alexandridis, 1993).

En la actualidad, para resolver problemas en diversos campos se utiliza, por ejemplo:

- Informática (módem, impresoras, teclados).
- Electrodomésticos (control de calefacción, microondas, lavadoras).
- Automotriz (inyección de combustible, ABS, alarmas).
- Audio y video (videograbadoras, reproductor de CD, sintonías digitales).
- Industria (automatismo de maniobra, control de temperatura, variadores de velocidad).
- Medicina (electrocardiógrafos).
- Usos militares.
- Domótica (edificios inteligentes).
- Burótica u ofimática.



# capítulo 2

## HERRAMIENTAS DE PROGRAMACIÓN

### TEMAS:

- Lenguaje de bajo nivel
- Interconexión de las unidades funcionales
- Segmento
- Interrupciones
- Comentarios en lenguaje ensamblador
- Transferencia de datos







## 2.1 LENGUAJE DE BAJO NIVEL

El lenguaje ensamblador se puede denominar «lenguaje de bajo nivel» o «lenguaje máquina». Este tipo de lenguaje consiste en una serie de instrucciones que la parte física de cualquier hardware es capaz de interpretar. Dicha interpretación se da mediante datos de tipo binario: valor 0 y valor 1; que físicamente, se materializan con tensiones comprendidas entre 0 y 4.0 voltios, y entre 4 y 5 voltios, respectivamente. Asimismo, una instrucción puede representarse de la siguiente forma: 101101100101001001000110.

Esta cadena de dígitos binarios (bits) puede indicar a la máquina o computadora (así como se entiende en nuestra lengua natural) lo siguiente: «Mueva el contenido ubicado en el espacio de memoria B a la posición de memoria C».

La instrucción anterior se podría escribir (para facilitar la comprensión) de la siguiente manera: «TRASLADAR POS-B POS-C».

El programador de lenguaje ensamblador debe conocer el hardware con que trabaja, ya que es necesario saber dónde indicar una posición de memoria determinada, un registro o cualquier otra parte de la máquina.

### Ventajas de los lenguajes ensambladores

Existe un renovado interés en el lenguaje ensamblador. Su uso conlleva a diferentes ventajas:

- Se requiere considerablemente menos memoria y tiempo de ejecución.
- Los programas residentes y rutinas de servicio de interrupción casi siempre son desarrollados en lenguaje ensamblador.
- Permite una comprensión de la arquitectura del hardware.

## 2.2 INTERCONEXIÓN DE LAS UNIDADES FUNCIONALES

La información se transmite en forma de señales eléctricas y, por lo tanto, circula a través de conductores eléctricos. Esta información se encuentra codificada en grupos de bits. Cada bit viaja de una unidad a otra por un conductor diferente al del resto de los bits. 8 bits necesitan 8 conductores, 16 bits necesitan 16 conductores, y así sucesivamente. Se ahorra tiempo porque circulan todos los bits de un mismo grupo a la vez, cada uno por su conductor, haciendo una transmisión en paralelo.

### A. Registros internos del procesador

Se usan para controlar las instrucciones en ejecución, manejar direccionamiento de la memoria y proporcionar capacidad aritmética. Los registros son direccionables por medio de un nombre. Los bits, por convención, se numeran de derecha a izquierda de la siguiente manera: 11 10 9 8 7 6 5 4 3 2 1 0.



## B. Registros de segmento

Un registro de segmento posee 16 bits de longitud y facilita un área de memoria para direccionamiento conocido como «segmento actual». Estos registros son CS, DS, SS, ES, FS y GS.

## C. Registros de propósito general

Los registros de propósito general AX, BX, CX y DX se pueden direccionar como una palabra o como una parte de un byte. El último byte de la izquierda es la parte «alta» y el último byte de la derecha es la parte «baja». Por ejemplo, el registro CX consta de una parte CH (alta) y una parte CL (baja). Puede referirse a cualquier parte por su respectivo nombre.

- **Registro AX:** Conocido como «acumulador principal». Usado para operaciones que implican entrada/salida, y la mayor parte de la aritmética. Por ejemplo, las instrucciones para multiplicar, dividir y traducir requieren el uso de AX. Algunas operaciones también generan un código más eficiente si se refieren al registro AX en lugar de referirse a otros registros.
- **Registro BX:** Conocido como «registro base», debido a que es el único registro de propósito general que puede ser índice para direccionamiento indexado. Es común usar el registro BX para cálculos.
- **Registro CX:** Conocido como «registro contador» es usado para el desplazamiento de bits, pues, mediante un valor puede controlar el número de veces que un ciclo se repite.
- **Registro DX:** Conocido como el «registro de datos». Algunas operaciones de entrada/salida requieren uso, y las operaciones de multiplicación y división con cifras grandes requieren el empleo de los registros DX y AX para que trabajen juntos. Se pueden usar los registros de propósito general para suma y resta de cifras de 8, 16 o 32 bits (Abel, 1996).

## D. Registro de apuntador de instrucciones (IP)

«El IP de 16 bits presenta el desplazamiento en la dirección hacia la siguiente instrucción a ejecutar. El IP, además, está asociado con el registro CS en lo concerniente a que el IP indica la instrucción actual dentro del segmento de código que se está ejecutando actualmente. Los procesadores 80386 y posteriores presentan un IP ampliado de 32 bits, el cual es llamado EIP» (Abel, 1996).

Por ejemplo, el registro CS contiene 26B4[0]H y el IP contiene 323H. Para encontrar la siguiente instrucción que ha de ser ejecutada, el procesador combina las direcciones en el CS y el IP:

- Segmento de dirección en el registro CS: 26B40H
- Desplazamiento de dirección en el registro IP: + 323H
- Dirección de la siguiente instrucción: 26E63H



### E. Registros apuntadores

«Los registros SP (apuntador de la pila) y BP (apuntador de base) están asociados con el registro SS y permiten al sistema ingresar datos en el segmento de la pila.

- **Registro SP:** El apuntador de pila de 16 bits se encuentra asociado con el registro SS y proporciona un valor de desplazamiento referido a la palabra actual que está siendo procesada en la pila. Los procesadores 80386 y posteriores tienen un apuntador de pila de 32 bits (registro ESP). El sistema maneja automáticamente estos registros» (Abel, 1996).

Por ejemplo, el registro SS contiene la dirección de segmento 25B2[0]H y el SP el desplazamiento 214H. Para encontrar la palabra actual que está siendo procesada en la pila, la computadora combina las direcciones en el SS y el SP:

Dirección de segmento: 25B2[0]H (p)  
 SP el desplazamiento: + 214H (q)  
 Dirección en la pila: 25D34H



**Figura 2.1** Registro de apuntadores

- **Registro BP:** El BP de 16 bits facilita la referencia de parámetros. Estos datos y direcciones son transmitidos vía pila. Los procesadores 80386 y posteriores tienen un BP ampliado de 32 bits denominado registro EBP.

### F. Registro índice

Los registros SI y DI están disponibles para direccionamiento indexado y para sumas y restas.

- **Registro SI:** El registro índice fuente de 16 bits es requerido por algunas operaciones con cadenas (de caracteres). En este contexto, el SI está asociado con el registro DS. Los procesadores 80386 y posteriores permiten el uso de un registro ampliado de 32 bits (el ESI).
- **Registro DI:** El registro índice destino también es requerido por algunas operaciones con cadenas de caracteres. En este contexto, el DI está asociado con el registro ES. Los procesadores 80386 y posteriores permiten el uso de un registro ampliado de 32 bits (el EDI).

### G. Registro de banderas

«De los 16 bits del registro de banderas, nueve son comunes a toda la familia de procesadores 8086, y sirven para indicar el estado actual de la máquina y el resultado del procesamiento. Muchas instrucciones que piden comparaciones y aritmética



cambian el estado de las banderas. Algunas instrucciones pueden realizar pruebas para determinar la acción subsecuente. Los bits de las banderas comunes son los siguientes:

- **OF (overflow o desbordamiento):** Indica el desbordamiento de un bit de orden alto (más a la izquierda) después de una operación aritmética.
- **DF (dirección):** Designa la dirección hacia la izquierda o hacia la derecha para mover o comparar cadenas de caracteres.
- **IF (interrupción):** Indica que una interrupción externa, como la entrada desde el teclado, sea procesada o, en su defecto, ignorada.
- **TF (trampa):** Permite la operación del procesador en modo de un solo paso. Los programas depuradores (como el debug) activan esta bandera de manera que se pueda avanzar en la ejecución de una sola instrucción a un tiempo, y así poder examinar el efecto de esa instrucción sobre los registros de memoria.
- **SF (signo):** Contiene el signo resultante de una operación aritmética (0 = positivo y 1 = negativo).
- **ZF (cero):** Indica el resultado de una operación aritmética o de comparación (0 = resultado diferente de cero y 1 = resultado igual a cero).
- **AF (acarreo auxiliar):** Contiene un acarreo externo del bit 3 en un dato de 8 bits para aritmética especializada.
- **PF (paridad):** Indica paridad (par o impar) de una operación en datos de 8 bits de bajo orden (más a la derecha).
- **CF (acarreo):** Contiene el acarreo de orden más alto (más a la izquierda) después de una operación aritmética. Lleva, además, el contenido del último bit en una operación de corrimiento o de rotación.

Las banderas están en el registro de banderas en las siguientes posiciones:

Núm. de bit	13	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Bandera					0	0	1	T	S	Z		A		P		C

**Figura 2.2** Posicionamiento de banderas

Las banderas más importantes para la programación en ensamblador son O, S, Z y C, para operaciones de comparación y aritméticas, y D, para operaciones de cadenas de caracteres» (Abel, 1996).



## 2.3 SEGMENTO

Un segmento es un área especial en un programa que inicia en un límite de un párrafo, es decir, en una localidad regularmente divisible entre 16 o 10 hexadecimal. Aunque un segmento puede estar ubicado casi en cualquier lugar de la memoria y, en modo real, puede ser hasta de 64K, solo necesita tanto espacio como el programa requiera para su ejecución.

Un segmento en modo real puede ser de hasta 64K. Además, se puede tener cualquier número de segmentos. Para direccionar un segmento en particular, solo basta cambiar la dirección en el registro del segmento apropiado. Los tres segmentos principales son los segmentos de código, los de datos y los de la pila.

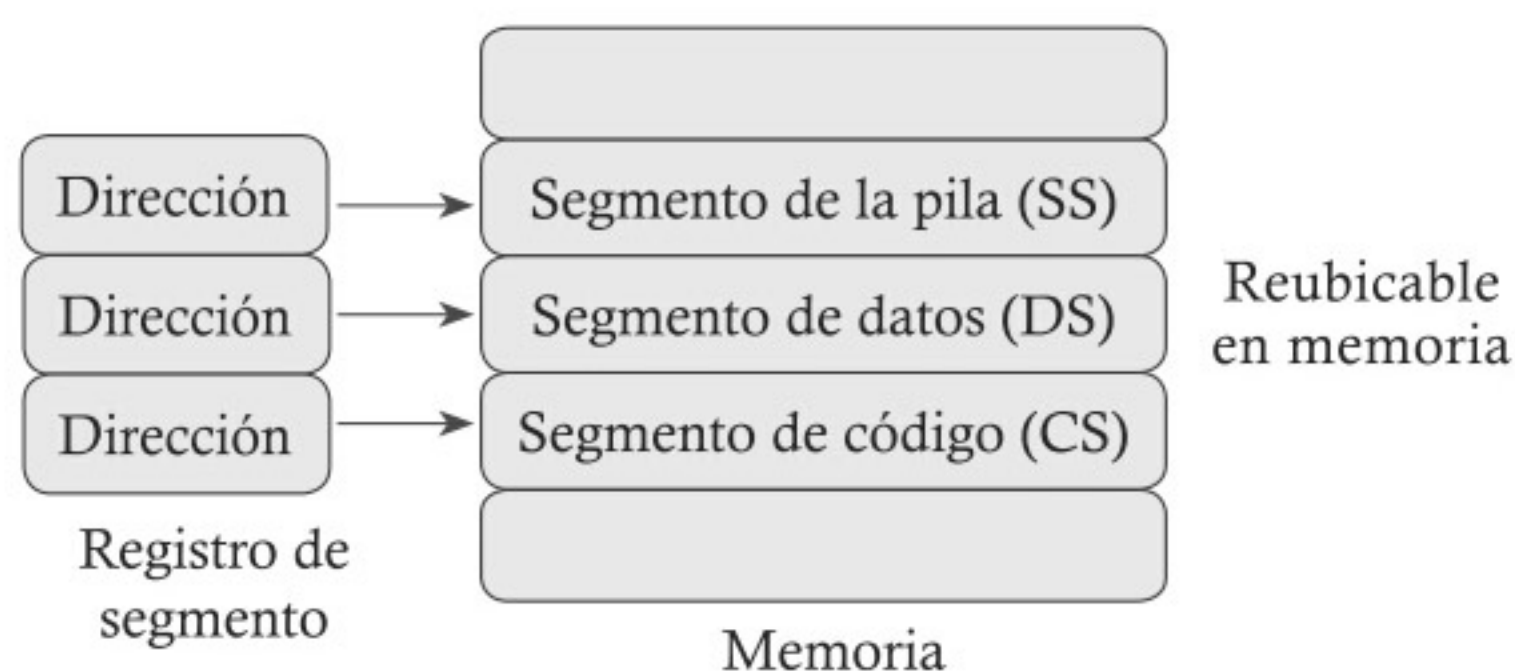
- **Segmento de código (*code segment* o **CS**):** Contiene las instrucciones de máquina que son ejecutadas. Por lo general, la primera instrucción ejecutable está en el inicio del segmento y el sistema operativo enlaza a esa localidad para iniciar la ejecución del programa. Como su nombre indica, el registro del CS direcciona el segmento de código. Si su área de código requiere más de 64K, su programa puede necesitar definir más de un segmento de código.
- **Segmento de datos (*data segment* o **DS**):** Contiene datos, constantes y áreas de trabajo definidos por el programa. El registro DS direcciona el segmento de datos. Si su área de datos requiere más de 64K, su programa puede necesitar definir más de un segmento de datos.
- **Segmento de pila (*stack segment* o **SS**):** Contiene los datos y direcciones que necesita guardar temporalmente o para uso de sus denominadas «subrutinas». El registro de segmento de la pila (SS) direcciona el segmento de la pila.

### A. Límites de los segmentos

Todos los registros de segmentos presentan la dirección inicial de cada segmento. La figura 2.3 presenta un esquema de los registros CS, DS y SS. Estos registros y segmentos no necesariamente se presentan en el orden mostrado. Otros registros de segmentos son el ES (segmento extra) y, en los antiguos procesadores 80386 y posteriores, los registros FS y GS, que contienen usos especializados.

El segmento inicia en un límite de párrafo, con una dirección por lo común divisible entre el 16 decimal o 10 hexadecimal. Por ejemplo, un segmento de datos inicia en la localidad de memoria 045F0H. En todos los demás casos, el último dígito hexadecimal de la derecha es cero, puesto que los diseñadores de computadora prefirieron no almacenar el dígito cero en el registro del segmento.





**Figura 2.3** Segmentos y registros

De esta manera, cuando se lee 054F0H, este se almacena como 054F, sobreentendiendo que el cero se ubica en la extrema derecha. También se permite el cero de la derecha encerrado entre corchetes, por ejemplo 054F[0].

## B. Desplazamiento

Se define como desplazamiento la distancia en bytes desde la ubicación de la dirección inicial del segmento, referido a los espacios de memoria, que presenta un conjunto de instrucciones. En un conjunto de instrucciones, todos los espacios de memoria están referidos a una dirección inicial de segmento. La distancia en bytes desde la dirección del segmento se define como el desplazamiento (*offset*).

Por ejemplo, «un desplazamiento de dos bytes (16 bits) puede estar en el rango de 0000H hasta FFFFH, o bien, desde cero hasta 65 535. Así, el primer byte del segmento de código tiene un desplazamiento 00, el segundo byte tiene un desplazamiento 01, etc., hasta el desplazamiento 65 535. Para referir cualquier dirección de memoria en un segmento, el procesador combina la dirección del segmento en un registro de segmento con un valor de desplazamiento» (Abel, 1996).

Por ejemplo, el registro DS contiene la dirección de segmento del segmento de datos en 044F[0]H y una instrucción hace referencia a una localidad con un desplazamiento de 0042H bytes dentro del segmento de datos.

Por lo tanto, la localidad real de memoria del byte referido por la instrucción es 04532H:

Dirección del segmento 044F[0]H	...	Desplazamiento 0042H
------------------------------------	-----	-------------------------

**Figura 2.4** Desplazamiento

Dirección del segmento DS:	044F[0]H
Desplazamiento:	<u>+0042H</u>
Dirección real:	04532H



Como se señala, un conjunto de instrucciones tiene uno o más segmentos, por lo tanto, pueden estar en cualquier orden, iniciar casi en cualquier lugar de la memoria, así también variar en tamaño.

### C. Métodos de direccionamiento

Los métodos de direccionamiento son reglas específicas que permiten interpretar o modificar el campo de dirección de la instrucción anterior al proceso de referenciar realmente al operando.

Los métodos de direccionamiento se diferencian por la manera de elegir los operandos durante la ejecución de un programa sobre los datos almacenados en los registros de la computadora o en palabras de memoria.

Las consideraciones para el uso de técnicas de modo de direccionamiento que usan las computadoras son:

- Brindar al programador flexibilidad y facilidades mediante herramientas como apuntadores a memoria, contadores para control de ciclo, indexación de datos y reubicación de datos.
- Disminuir el número de bits en el campo de direccionamiento de la instrucción.

La disponibilidad de los modos de direccionamiento proporciona al programador con experiencia en lenguaje ensamblador la flexibilidad para escribir programas más eficientes en relación con la cantidad de instrucciones y el tiempo de ejecución.

Para comprender los diferentes modos de direccionamiento, es importante entender el ciclo de operación básico de la computadora. La unidad de control de una computadora está diseñada para recorrer un ciclo de instrucciones que se divide en tres fases principales:

1. Búsqueda de la instrucción de la memoria
2. Decodificación de la instrucción
3. Ejecución de la instrucción

Aunque la mayoría de los modos de direccionamiento modifican el campo de dirección de la instrucción, existen dos modos que no necesitan el campo de dirección. Son los modos implícito e inmediato.

## 2.4 INTERRUPCIONES

Para realizar una acción diferente de un programa, es necesaria la suspensión de la ejecución de los procedimientos de dicho programa. Este proceso se denomina interrupción, pues, al terminar la ejecución de la interrupción, por lo general, se reasume o regresa a ejecutarse la rutina suspendida.



### A. Eventos de una interrupción

Cuando se realiza una interrupción, esta almacena en la pila el contenido del registro de banderas, el CS, y el IP. Por ejemplo, la dirección en la tabla de INT 05H (que imprime la que se encuentra en la pantalla cuando el usuario presiona Ctrl + PrtSC) es 0014H ( $05H \times 4 = 14H$ ). La operación extrae la dirección de cuatro bytes de la posición 0014H y almacena dos bytes en el IP y dos en el CS.

La dirección CS: IP, entonces, apunta al inicio de la rutina en el área del BIOS, que ahora se ejecuta. La interrupción regresa vía una instrucción IRET (regreso de interrupción), que saca de la pila el IP, CS y las banderas, y regresa el control a la instrucción que sigue al INT (Mano, 1994).

### B. Tipos de interrupciones

Las interrupciones se presentan de dos tipos: externas e internas. «Una interrupción externa es realizada por un dispositivo externo al procesador. Las dos líneas que pueden señalar interrupciones externas, son la línea de interrupción no enmascarable (NMI) y la línea de petición de interrupción (INTR).

La línea NMI reporta la memoria y errores de paridad de E/S. El procesador siempre actúa sobre esta interrupción, aun si emite un CLI para limpiar la bandera de interrupción en un intento por deshabilitar las interrupciones externas. La línea INTR reporta las peticiones desde los dispositivos externos, en realidad, las interrupciones 05H a la 0FH, para cronómetro, el teclado, los puertos seriales, el disco duro, las unidades de disco flexibles y los puertos paralelos.

Una interrupción interna ocurre como resultado de la ejecución de una instrucción INT o una operación de división que cause desbordamiento, ejecución en modo de un paso o una petición para una interrupción externa, como E/S de disco. Los programas por lo común utilizan interrupciones internas, que no son enmascarables, para acceder los procedimientos del BIOS y del DOS» (Abel, 1996).

## 2.5 COMENTARIOS EN LENGUAJE ENSAMBLADOR

El uso de comentarios en los programas mejora el entendimiento del mismo, se ubica a continuación de una instrucción, con un punto y coma (;), y puede contener cualquier carácter. Otra manera es por la directiva COMMENT.

; Toda esta línea es un comentario

ADD AX, BX ; Comentario en la misma línea que la instrucción



### A. Palabras reservadas

Las palabras reservadas son usadas bajo condiciones especiales y tienen que usarse para sus propios propósitos como MOV y ADD, END o SEGMENT, FAR y SIZE, @Data y @Model, y otras que se presentan en la sección de anexos de este libro.

### B. Identificadores

«Un identificador es un nombre que se emplea para nombrar elementos en el programa. Los dos tipos de identificadores son nombre, que se refiere a la dirección de un elemento de dato, y etiqueta, que se refiere a la dirección de una instrucción. Las mismas reglas se aplican tanto para los nombres como para las etiquetas. Un identificador puede usar los siguientes caracteres:

- |                           |                                                   |
|---------------------------|---------------------------------------------------|
| 1. Letras del alfabeto:   | Desde la A hasta la Z                             |
| 2. Dígitos:               | Desde el 0 al 9 (no puede ser el primer carácter) |
| 3. Caracteres especiales: | Signo de interrogación ( ? )                      |
|                           | Subrayado ( _ )                                   |
|                           | Signo de pesos ( \$ )                             |
|                           | Arroba ( @ )                                      |
|                           | Punto ( . ) (no puede ser el primer carácter)     |

El primer carácter de un identificador debe ser una letra o un carácter especial, excepto el punto, ya que el lenguaje ensamblador utiliza algunos símbolos especiales en palabras que inician con el símbolo @, debe evitar usarlo en sus definiciones» (Abel, 1996).

El lenguaje ensamblador no discrimina mayúsculas y minúsculas. La longitud máxima de un identificador es de 31 caracteres (247 desde el MASM 6.0).

### C. Instrucciones

Un programa en lenguaje ensamblador consiste en un conjunto de enunciados. Hay dos tipos de enunciados:

- Instrucciones, tal como MOV y ADD, que el ensamblador traduce a código objeto.
- Directivas, que indican al ensamblador que realice una acción específica, como definir un elemento de dato.

### D. Operación

«La operación, que debe ser codificada, es con mayor frecuencia usada para la definición de áreas de datos y codificación de instrucciones. Para un elemento de datos, una operación como DB o DW define un campo, área de trabajo o constante» (Abel, 1996).



### E. Operando

Para una instrucción, el operando indica en donde realizar la acción. El operando de una instrucción puede tener una, dos o tal vez ninguna entrada.

## 2.6 TRANSFERENCIA DE DATOS

«La instrucción de transferencia de datos por excelencia es MOV destino, fuente (el contenido que se va a transferir). Esta instrucción, por lo tanto, va a permitir transferir información entre registros y memoria, memoria y registros, y entre los propios registros utilizando alguno de los diferentes modos de direccionamiento. Con la instrucción MOV se dirá que se pueden realizar todo tipo de movimientos teniendo en cuenta las siguientes restricciones:

1. No se puede realizar una transferencia de datos entre dos posiciones de memoria directamente, por ello, siempre que se quiera efectuarlas se tendrá que utilizar un registro intermedio que haga de puente.

Por ejemplo, para hacer la operación `DATO1 <-- DATO2`, la instrucción `MOV DATO2,DATO1` sería incorrecta. Lo correcto sería utilizar el registro DX, u otro, como puente y hacer:

```
MOV DX,DATO1
MOV DATO2,DX
```

2. Tampoco se puede hacer una transferencia directa entre dos registros de segmento. Por eso, como en el caso anterior, si fuera preciso, se utilizaría un registro como puente.
3. Asimismo, tampoco se puede cargar en los registros de segmento un dato utilizando direccionamiento inmediato, es decir, una constante; por lo que también habrá que recurrir a un registro puente cuando sea preciso.

Una instrucción útil, pero no imprescindible, es: `XCHG DATO1, DATO2`, que intercambia los contenidos de las posiciones de memoria o registros representados por DATO1 y DATO2.



Por ejemplo, si se quiere intercambiar los contenidos de los registros AX y BX, se puede hacer:

```
MOV AUX, AX  
MOV AX, BX  
MOV BX, AUX
```

En donde AUX es una variable auxiliar que hace de puente, o simplemente utilizar: XCHG AX, BX

La restricción que se muestra en esta operación es que no se pueden efectuar intercambios directamente entre posiciones de memoria ni tampoco entre registros de segmento» (Abel, 1996).







# capítulo 3

## OPERACIONES ARITMÉTICAS

### TEMAS:

- Suma y resta
- Operandos lógicos
- Multiplicación
- División







### 3.1 SUMA Y RESTA

Las instrucciones generales para suma y resta son las instrucciones ADD y SUB, respectivamente. El ejemplo siguiente utiliza el registro AX para sumar WORDAA a WORDBB:

```
WORDAA DW 133      ;DEFINE WORDAA
WORDBB DW 35 ;DEFINE WORDBB

MOV AX, WORDAA ;MUEVE WORDAA AL AX
ADD AX, WORDBB  ;SUMA WORDBB AL AX
MOV WORDBB, AX ;MUEVE AX A WORDBB
```

#### Ejercicio 1

Realizar un programa que permita la suma de 3 números hexadecimales y guardarlo en el registro AX en el depurador de Windows (debug). Ejemplo: AX=1+2+3; resultado: AX=6.

1. Primero, abra el depurador de Windows (debug) de la siguiente manera:

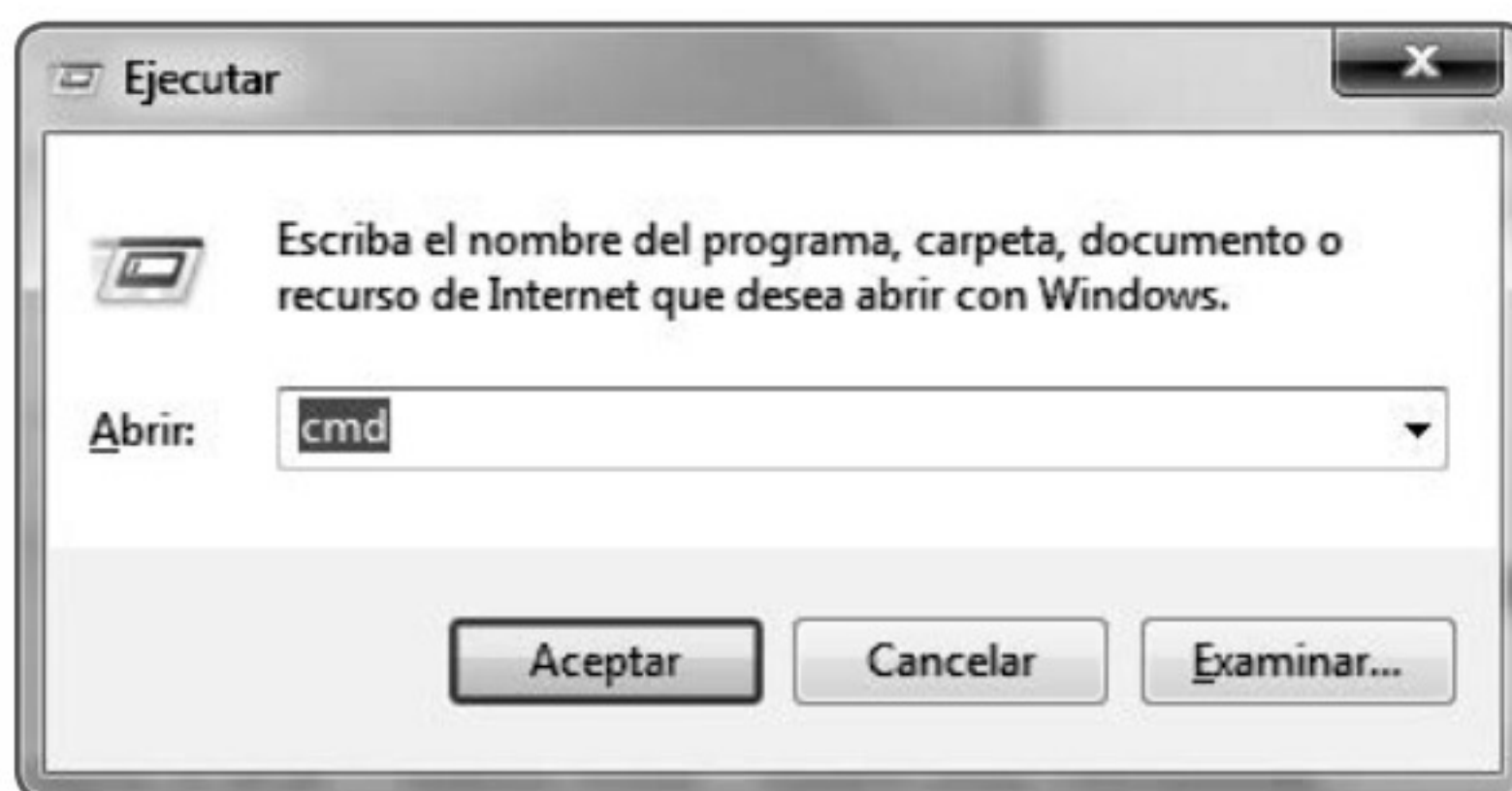


Figura 3.1 Ejecutar

- a. Abra la ventana **Ejecutar** con  + R y escriba **cmd** para abrir el símbolo del sistema.

Use **cd.TT.** las veces que sea necesario para llegar a la raíz C:.

- b. Escriba en la línea de comando la palabra **debug**.

2. Al empezar a ensamblar el programa, escriba **a100** (**a** = assembler y **100** = dirección de inicio del programa). Esto muestra la dirección lógica de la primera instrucción a introducir (1380:0100 = segmento:desplazamiento).

3. Luego, introduzca el siguiente código:

**mov ax,1** : Asigna al registro ax el valor 1, equivale en C++: ax=1.

**mov bx,2** : Asigna al registro bx el valor 2, equivale en C++: bx=2.

**add ax,bx** : Suma ambos registros y lo guarda en ax, equivale en C++: ax=ax+bx.

**mov bx,3** : Asigna al registro bx el valor 3, equivale en C++: bx=3.

**add ax,bx** : Suma ambos registros y guarda el resultado en ax; equivale en C++: ax=ax+bx.

**int 20** : Interrupción que termina el programa.

```

C:\Windows\system32\cmd.exe - debug
Microsoft Windows [Versión 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. Reservados todos los derechos.

C:\Users\user>cd..
C:\Users>cd..
C:\>debug
-a100
1380:0100 mov ax,1
1380:0103 mov bx,2
1380:0106 add ax,bx
1380:0108 mov bx,3
1380:010B add ax,bx
1380:010D int20
1380:010F
-t

```

Figura 3.2 cmd

4. Escriba **t** para ejecutar la primera instrucción y hacer correr el programa paso a paso:

```

C:\Windows\system32\cmd.exe - debug
Microsoft Windows [Versión 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. Reservados todos los derechos.

C:\Users\user>cd..
C:\Users>cd..
C:\>debug
-a100
1380:0100 mov ax,1
1380:0103 mov bx,2
1380:0106 add ax,bx
1380:0108 mov bx,3
1380:010B add ax,bx
1380:010D int20
1380:010F
-t
AX=0001  BX=0002  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=1380  ES=1380  SS=1380  CS=1380  IP=0103  NU UP EI PL NZ NA PO NC
1380:0106 add ax,bx

```

Figura 3.3 Ejecutar código ensamblador

La ventana muestra la primera instrucción que asigna ax=3 y también muestra todos los registros del microprocesador; y la última línea muestra la siguiente instrucción a ejecutar.



5. Escriba varias veces `t` hasta llegar a la última instrucción y muestre el resultado de la suma en el registro `ax`.

```

C:\Windows\system32\cmd.exe - debug
AX=0001 BX=0000 CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=1380 ES=1380 SS=1380 CS=1380 IP=0103  NU UP EI PL NZ NA PO NC
1380:0103 BB0200      MOV     BX,0002
-t
AX=0001 BX=0002 CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=1380 ES=1380 SS=1380 CS=1380 IP=0106  NU UP EI PL NZ NA PO NC
1380:0106 01D8      ADD     AX,BX
-t
AX=0003 BX=0002 CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=1380 ES=1380 SS=1380 CS=1380 IP=0108  NU UP EI PL NZ NA PE NC
1380:0108 BB0300      MOV     BX,0003
-t
AX=0003 BX=0003 CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=1380 ES=1380 SS=1380 CS=1380 IP=010B  NU UP EI PL NZ NA PE NC
1380:010B 01D8      ADD     AX,BX
-t
AX=0006 BX=0003 CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=1380 ES=1380 SS=1380 CS=1380 IP=010D  NU UP EI PL NZ NA PE NC
1380:010D CD20      INT     20

```

Figura 3.4 resultado: `ax=6`

## Ejercicio 2

Realizar la suma de dos números.

1. Abra la ventana **Ejecutar** con  + R y escriba `cmd` para abrir el símbolo del sistema.

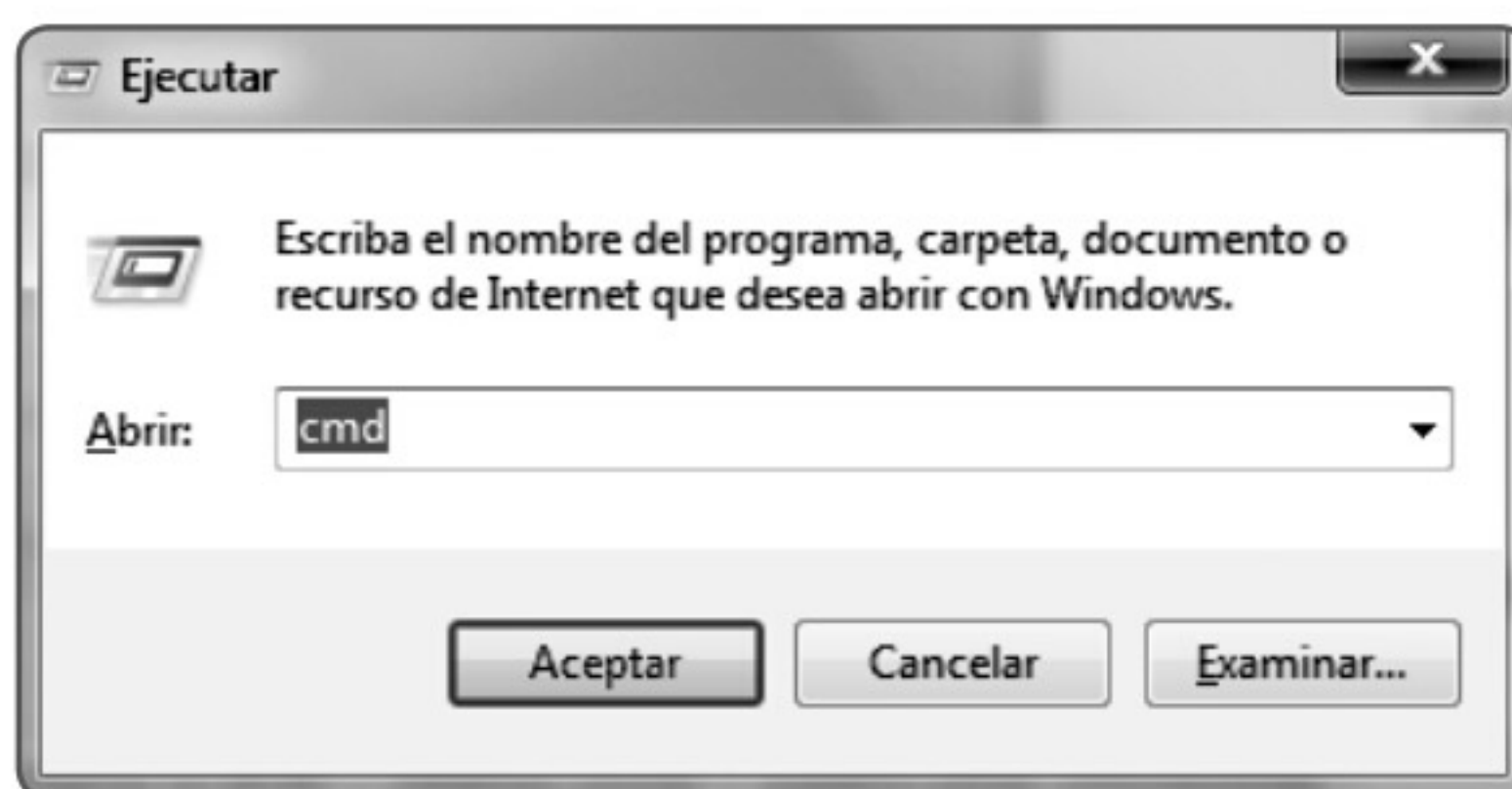


Figura 3.5 Ejecutar

2. Presione **Aceptar** y aparecerá la siguiente ventana:

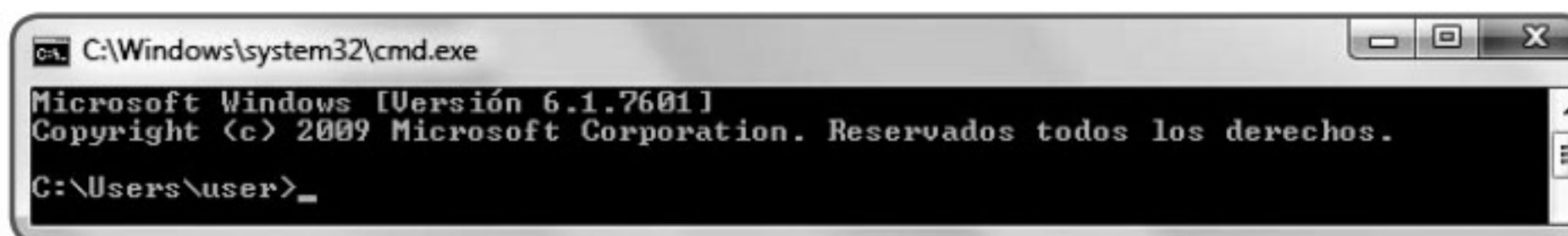


Figura 3.6 Pantalla cmd

3. Escriba **cd..** y presione **Enter** las veces que sea necesario para salir de las carpetas del sistema y poder llegar a la raíz C:

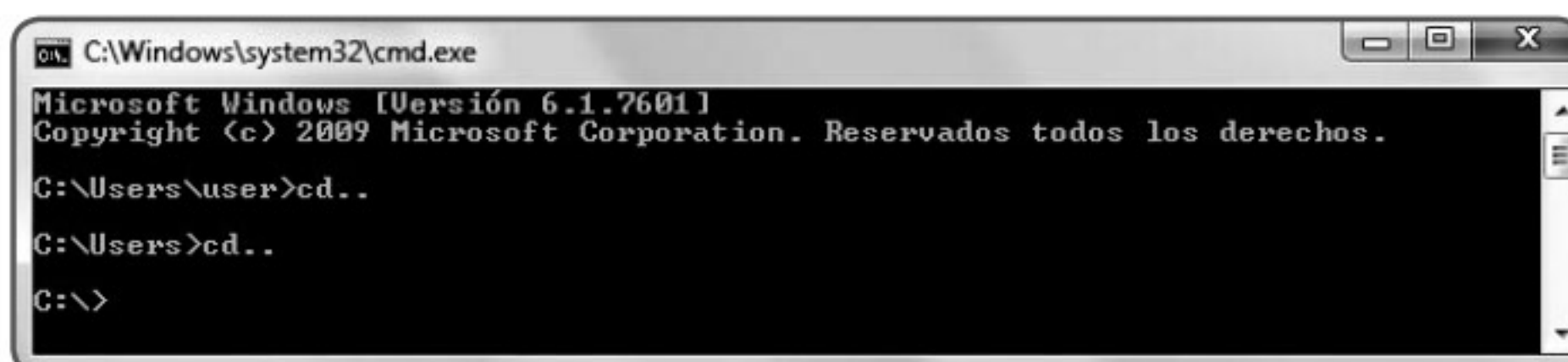


Figura 3.7 raíz\_c

4. Ingrese a debug, escriba en la línea de comando la palabra **debug**. A continuación, aparecerá lo siguiente:

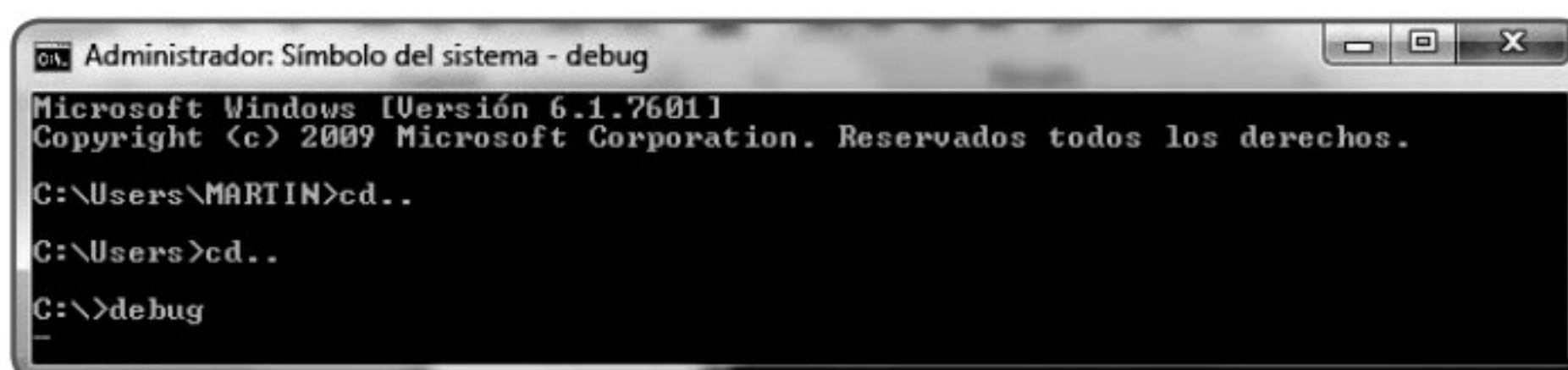


Figura 3.8 debug

5. Escriba **A100** y presione la tecla **Enter**.

**A** es una orden del debug que permite ensamblar instrucciones a partir de una dirección (normalmente se empieza en la dirección **0100h**). El debug generalmente utiliza números hexadecimales.



Figura 3.9 Ensamblar



6. Escriba las siguientes instrucciones en assembler:

La instrucción **MOV AX, 1000** ordena que AX tome el valor de 1.

La instrucción **MOV BX, 2000** ordena que BX tome el valor de 2.

La instrucción **ADD AX, BX** ordena que AX sea sumado con BX.

Por lo tanto, es lógico que AX tenga el valor 3000 luego de la instrucción ADD AX, BX, ya que se le sumó a AX el valor de BX.

```
MOV AX, 1
MOV BX, 2
ADD AX, BX
```



```
C:\Windows\system32\cmd.exe - debug
C:\Users>cd..
C:\>debug
-a100
1382:0100 mov ax,8
1382:0103 mov bx,2
1382:0106 sub ax,bx
```

Figura 3.10 Instrucciones

7. Fin del código.

Escriba **INT 20** y luego presione **Enter**.

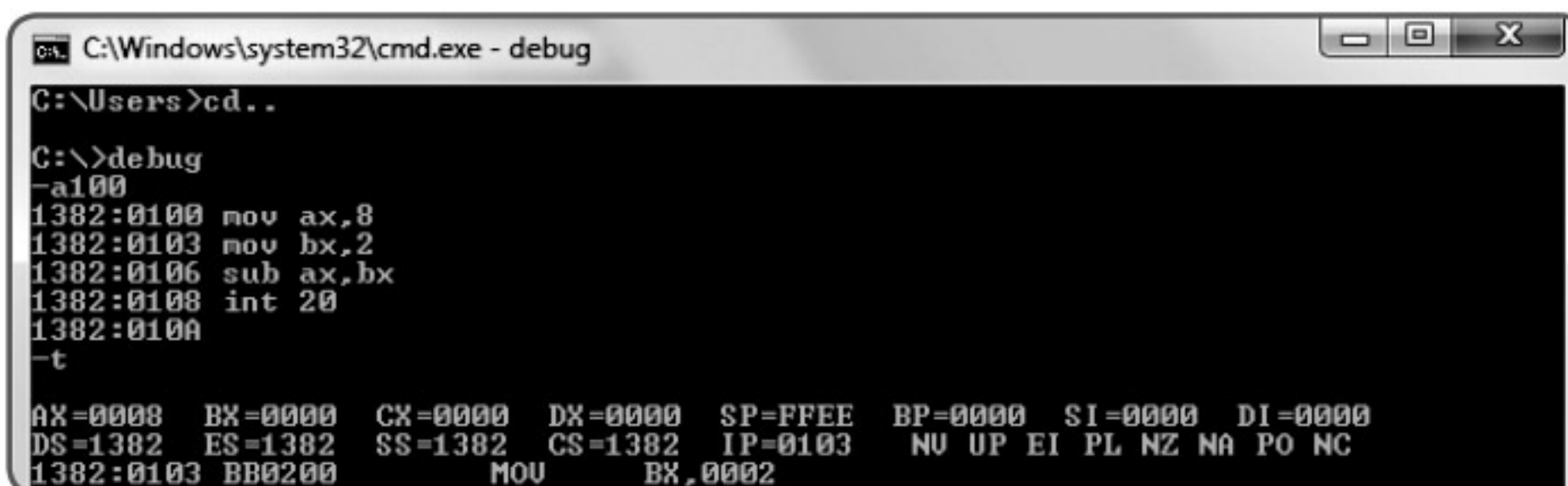


```
C:\Windows\system32\cmd.exe - debug
C:\Users>cd..
C:\>debug
-a100
1382:0100 mov ax,8
1382:0103 mov bx,2
1382:0106 sub ax,bx
1382:0108 int 20
1382:010A
```

Figura 3.11 int20

8. Mover al registro ax.

a. Escriba **t** y empiece con el código de la suma, donde primero debe mover el número 8 al registro ax.



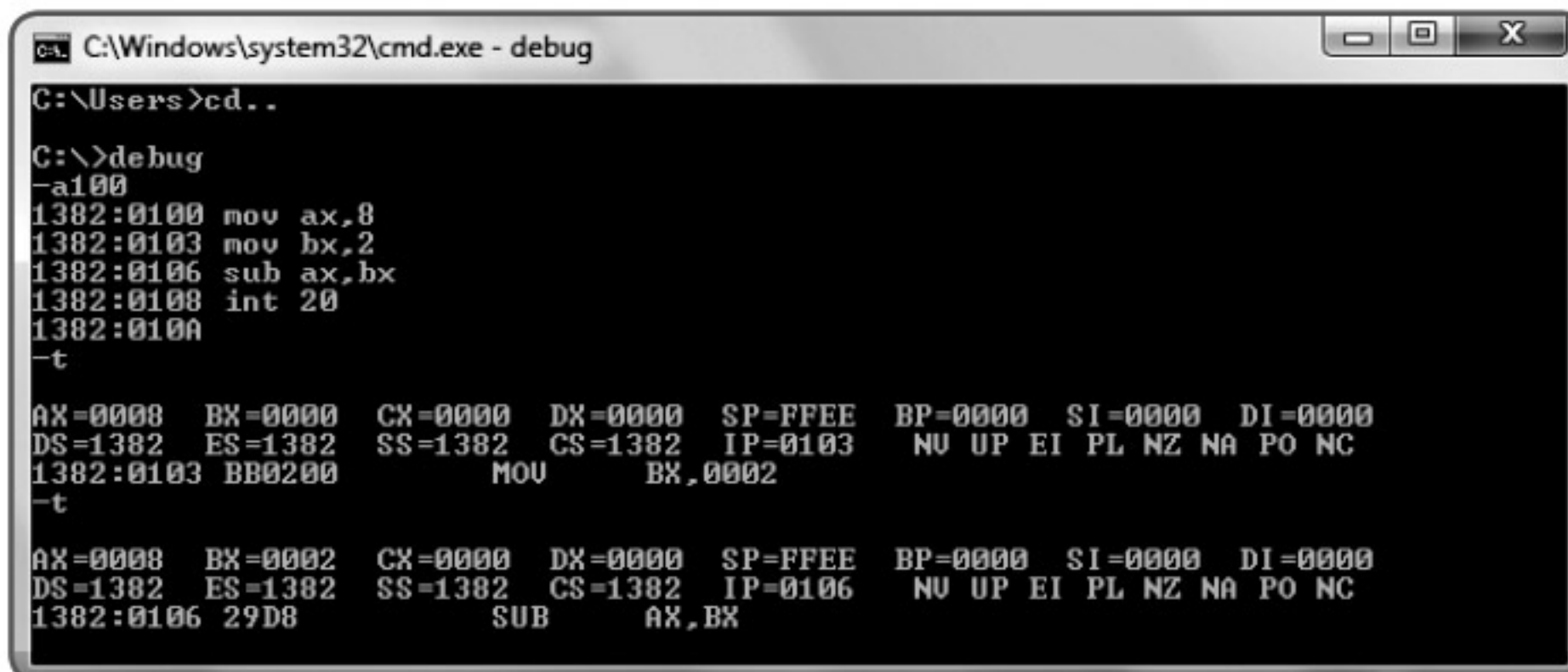
```
C:\Windows\system32\cmd.exe - debug
C:\Users>cd..
C:\>debug
-a100
1382:0100 mov ax,8
1382:0103 mov bx,2
1382:0106 sub ax,bx
1382:0108 int 20
1382:010A
-t
AX=0008 BX=0000 CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=1382 ES=1382 SS=1382 CS=1382 IP=0103 NU UP EI PL NZ NA PO NC
1382:0103 BB0200 MOV BX,0002
```

Figura 3.12 ax

El código de la suma donde se mueve el número 1 al registro ax.

b. Mover al registro bx.

El código de la suma donde se mueve el número 2 al registro bx.



```

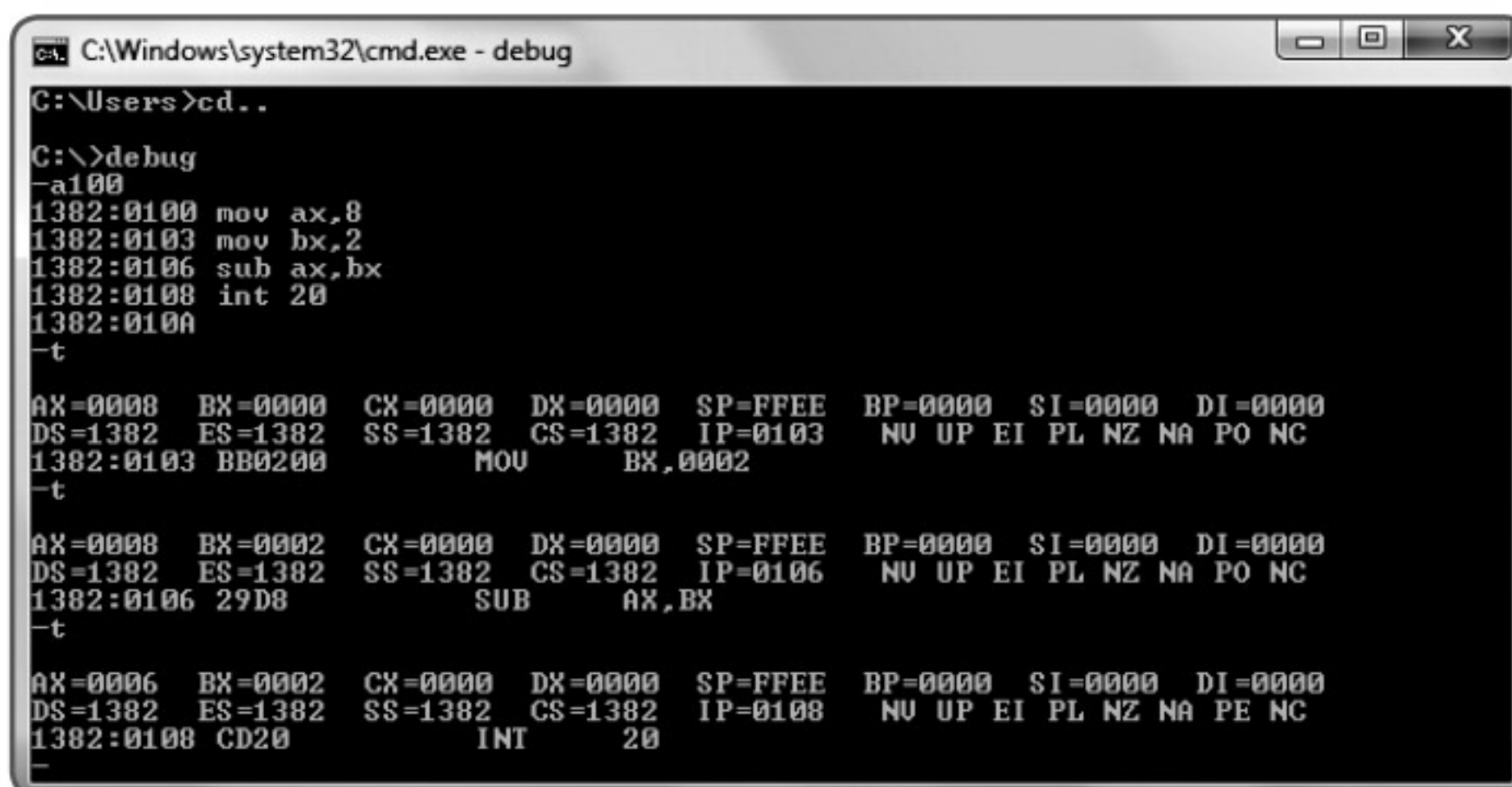
C:\Windows\system32\cmd.exe - debug
C:\Users>cd..
C:\>debug
-a100
1382:0100 mov ax,8
1382:0103 mov bx,2
1382:0106 sub ax,bx
1382:0108 int 20
1382:010A
-t
AX=0008 BX=0000 CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=1382 ES=1382 SS=1382 CS=1382 IP=0103  NU UP EI PL NZ NA PO NC
1382:0103 BB0200      MOV     BX,0002
-t
AX=0008 BX=0002 CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=1382 ES=1382 SS=1382 CS=1382 IP=0106  NU UP EI PL NZ NA PO NC
1382:0106 29D8      SUB     AX,BX

```

Figura 3.13 bx

c. Sumar ax con bx.

El código suma el registro ax con bx, suma el número 1 con 2, y da como resultado 3 en el primer registro ax.



```

C:\Windows\system32\cmd.exe - debug
C:\Users>cd..
C:\>debug
-a100
1382:0100 mov ax,8
1382:0103 mov bx,2
1382:0106 sub ax,bx
1382:0108 int 20
1382:010A
-t
AX=0008 BX=0000 CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=1382 ES=1382 SS=1382 CS=1382 IP=0103  NU UP EI PL NZ NA PO NC
1382:0103 BB0200      MOV     BX,0002
-t
AX=0008 BX=0002 CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=1382 ES=1382 SS=1382 CS=1382 IP=0106  NU UP EI PL NZ NA PO NC
1382:0106 29D8      SUB     AX,BX
-t
AX=0006 BX=0002 CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=1382 ES=1382 SS=1382 CS=1382 IP=0108  NU UP EI PL NZ NA PE NC
1382:0108 CD20      INT     20

```

Figura 3.14 suma ax con bx



### Ejercicio 3

Realizar la resta de dos números.

1. Abra la ventana **Ejecutar** con  + R y escriba **cmd** para abrir el símbolo del sistema.

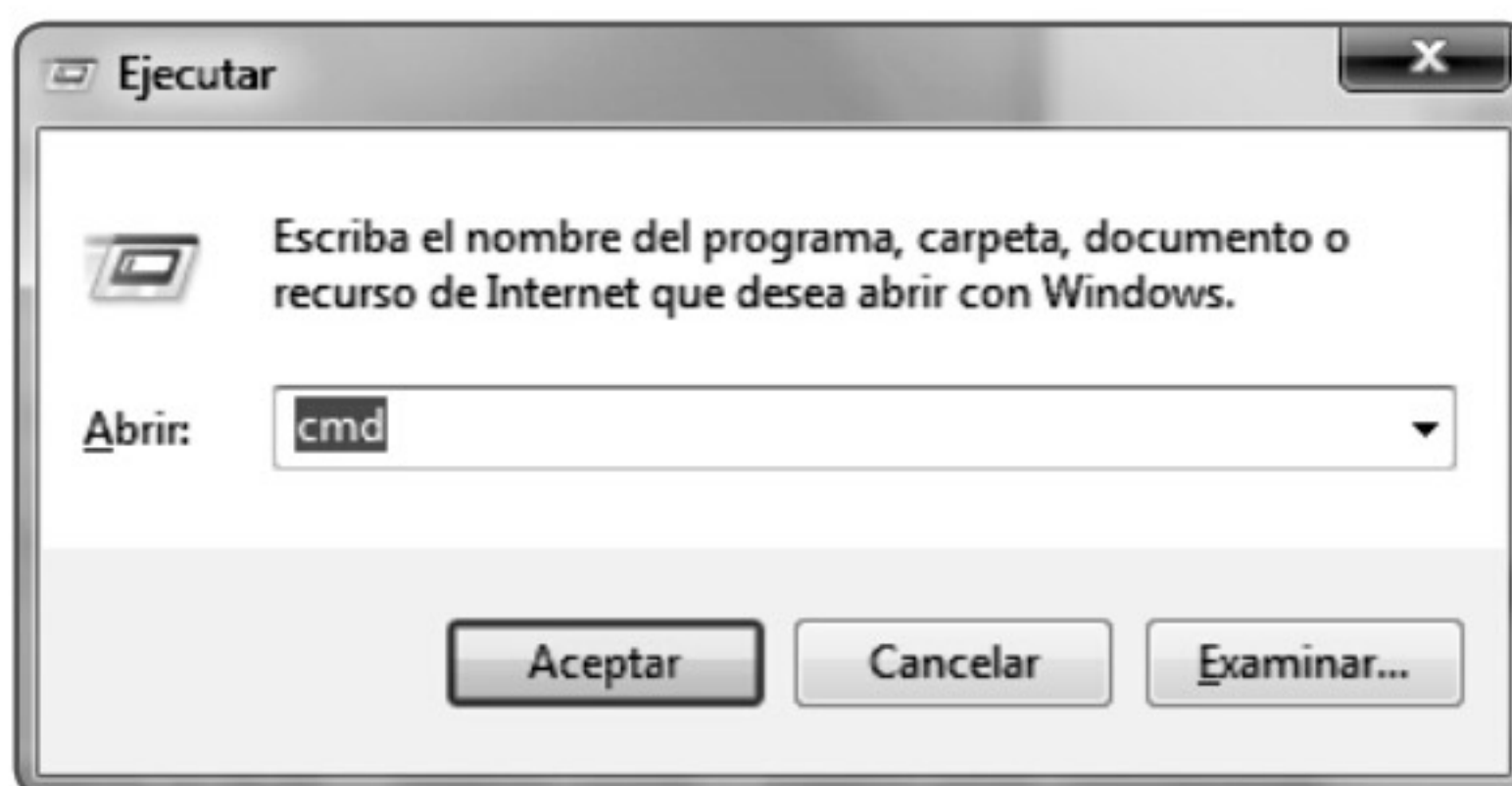


Figura 3.15 Ejecutar

2. Presione **Aceptar** y aparecerá la siguiente ventana:



Figura 3.16 Pantalla cmd

3. Escriba **cd..** y presione **Enter** las veces que sea necesario para salir de las carpetas del sistema y poder llegar a la raíz C:.

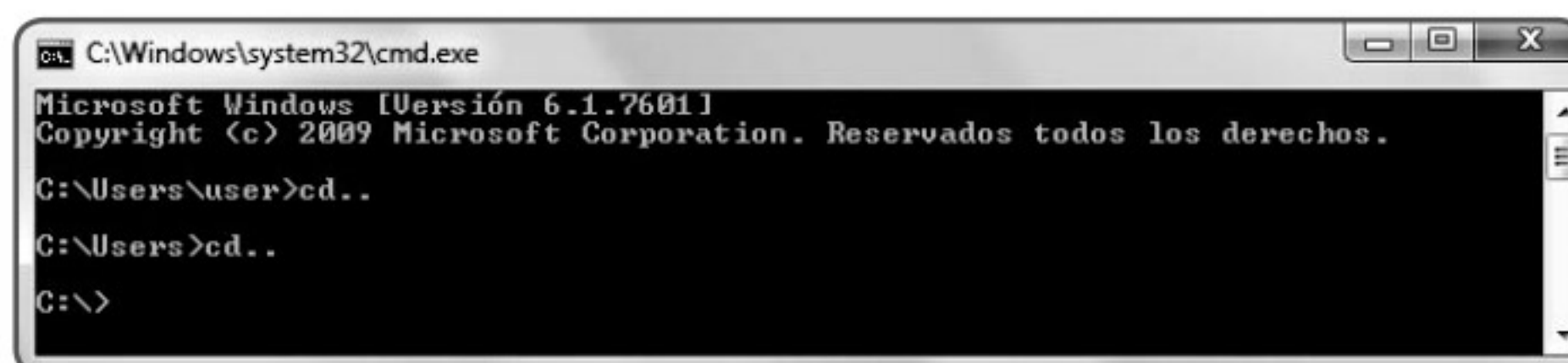
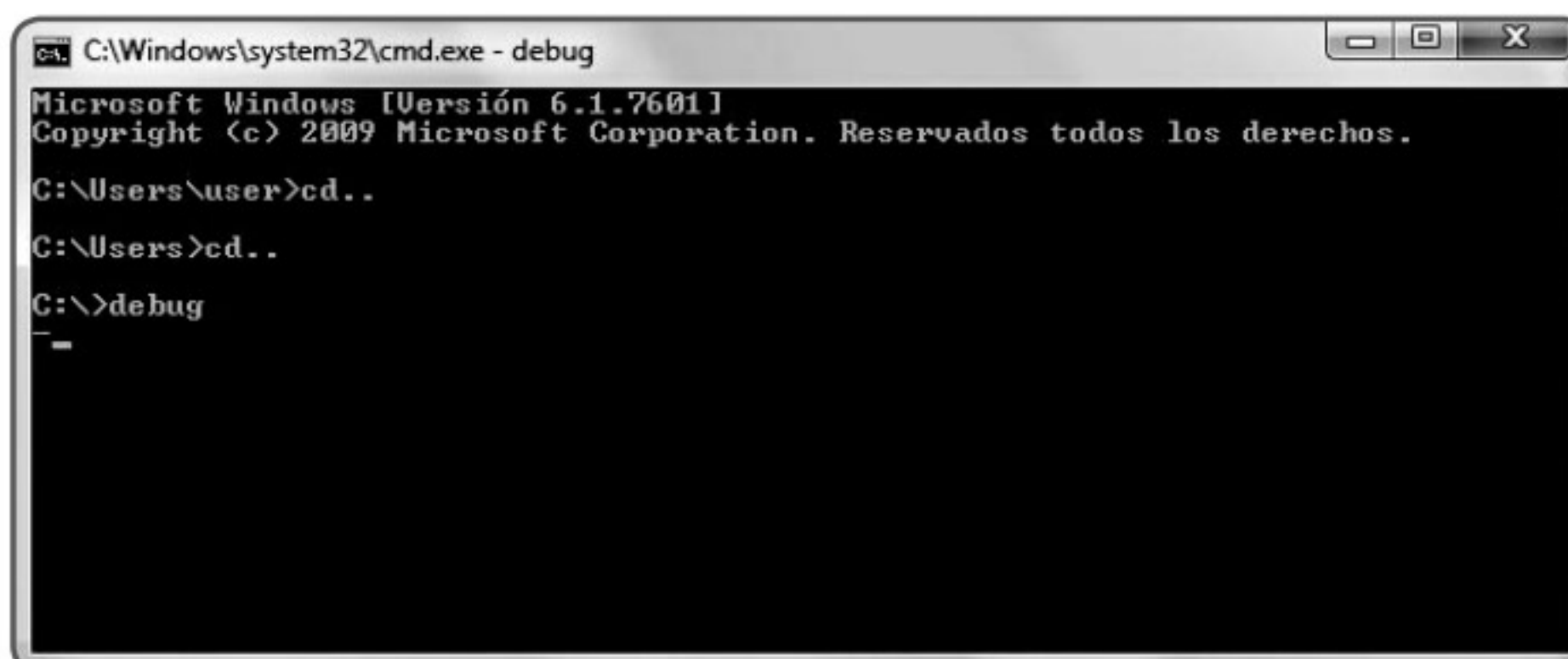


Figura 3.17 raíz\_c

4. Ingrese a debug, para esto, escriba en la línea de comando la palabra **debug**. A continuación, aparecerá lo siguiente:



```
C:\Windows\system32\cmd.exe - debug
Microsoft Windows [Versión 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. Reservados todos los derechos.
C:\Users\user>cd..
C:\Users>cd..
C:\>debug
-
```

Figura 3.18 debug

5. Escriba las siguientes instrucciones en assembler:

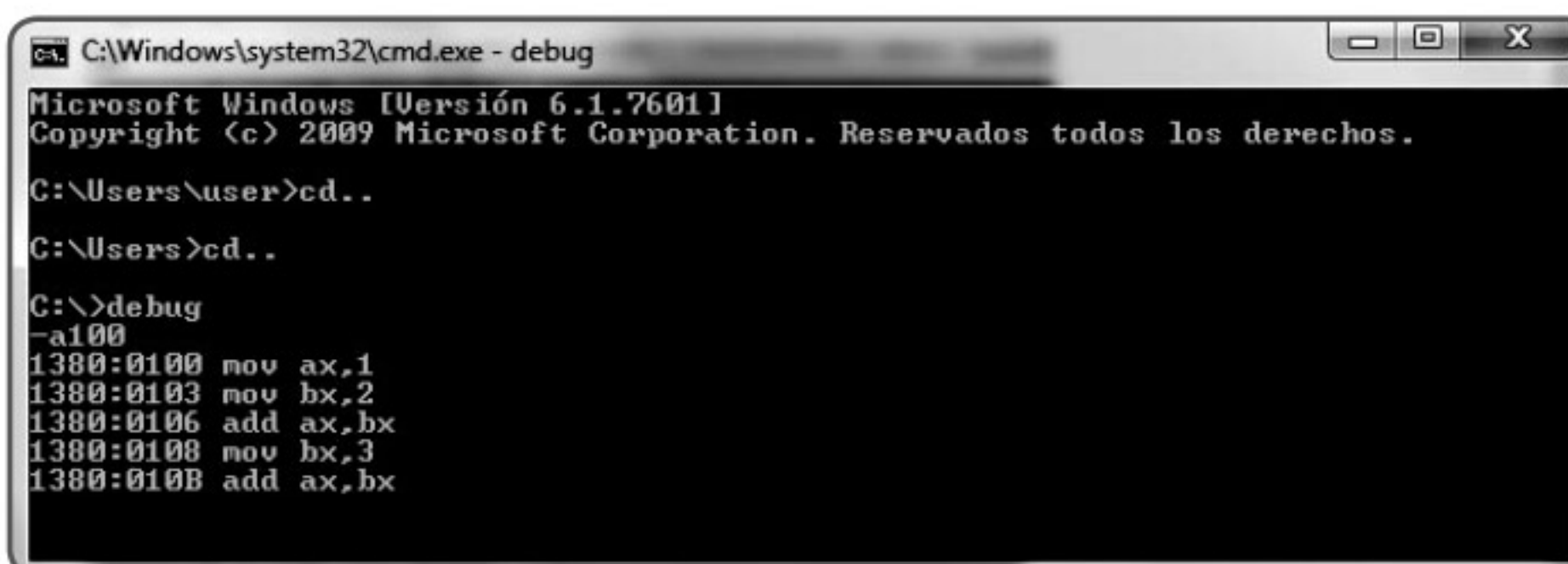
La instrucción **MOV AX, 8** ordena que AX tome el valor de 8.

La instrucción **MOV BX, 2** ordena que BX tome el valor de 2.

La instrucción **SUB AX, BX** ordena que AX sea restado con BX.

Por lo tanto, es lógico que AX tenga el valor 8 luego de la instrucción SUB AX, BX, ya que se le sumó a AX el valor de BX.

```
MOV AX, 8
MOV BX, 2
SUB AX, BX
```



```
C:\Windows\system32\cmd.exe - debug
Microsoft Windows [Versión 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. Reservados todos los derechos.
C:\Users\user>cd..
C:\Users>cd..
C:\>debug
-a100
1380:0100 mov ax,1
1380:0103 mov bx,2
1380:0106 add ax,bx
1380:0108 mov bx,3
1380:010B add ax,bx
```

Figura 3.19 resta



## 6. Fin del código.

Escriba **INT 20** y luego presione **Enter**.

INT 20: Esta sentencia llama a la interrupción 20.

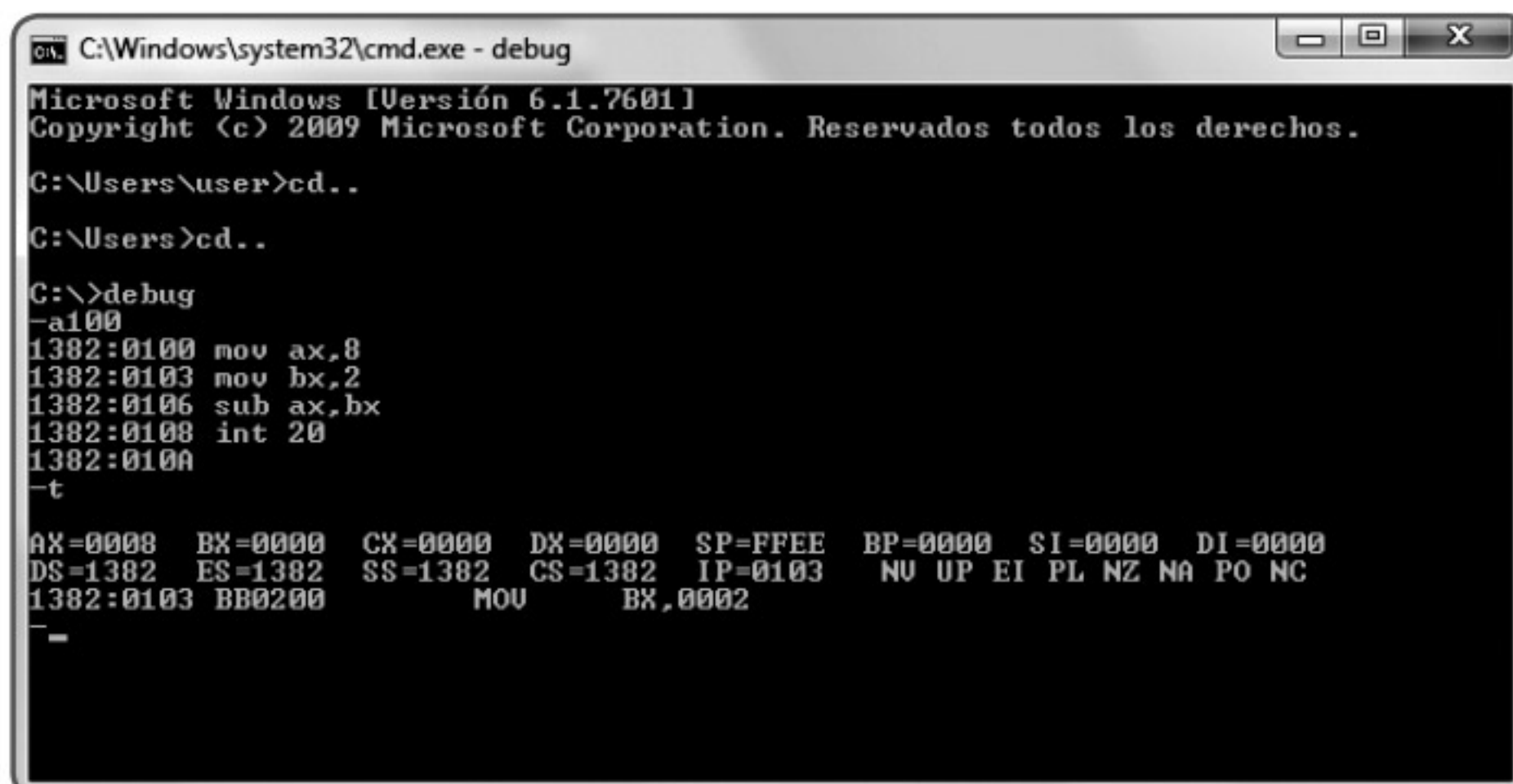


```
C:\Windows\system32\cmd.exe - debug
Microsoft Windows [Versión 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. Reservados todos los derechos.
C:\Users\user>cd..
C:\Users>cd..
C:\>debug
-a100
1382:0100 mov ax,8
1382:0103 mov bx,2
1382:0106 sub ax,bx
1382:0108 int 20
1382:010A
```

Figura 3.20 Interrupción

## 7. Mover al registro ax.

- a. Escriba **t**, y empiece con el código de la resta donde, primero, mueva el número 8 al registro ax.

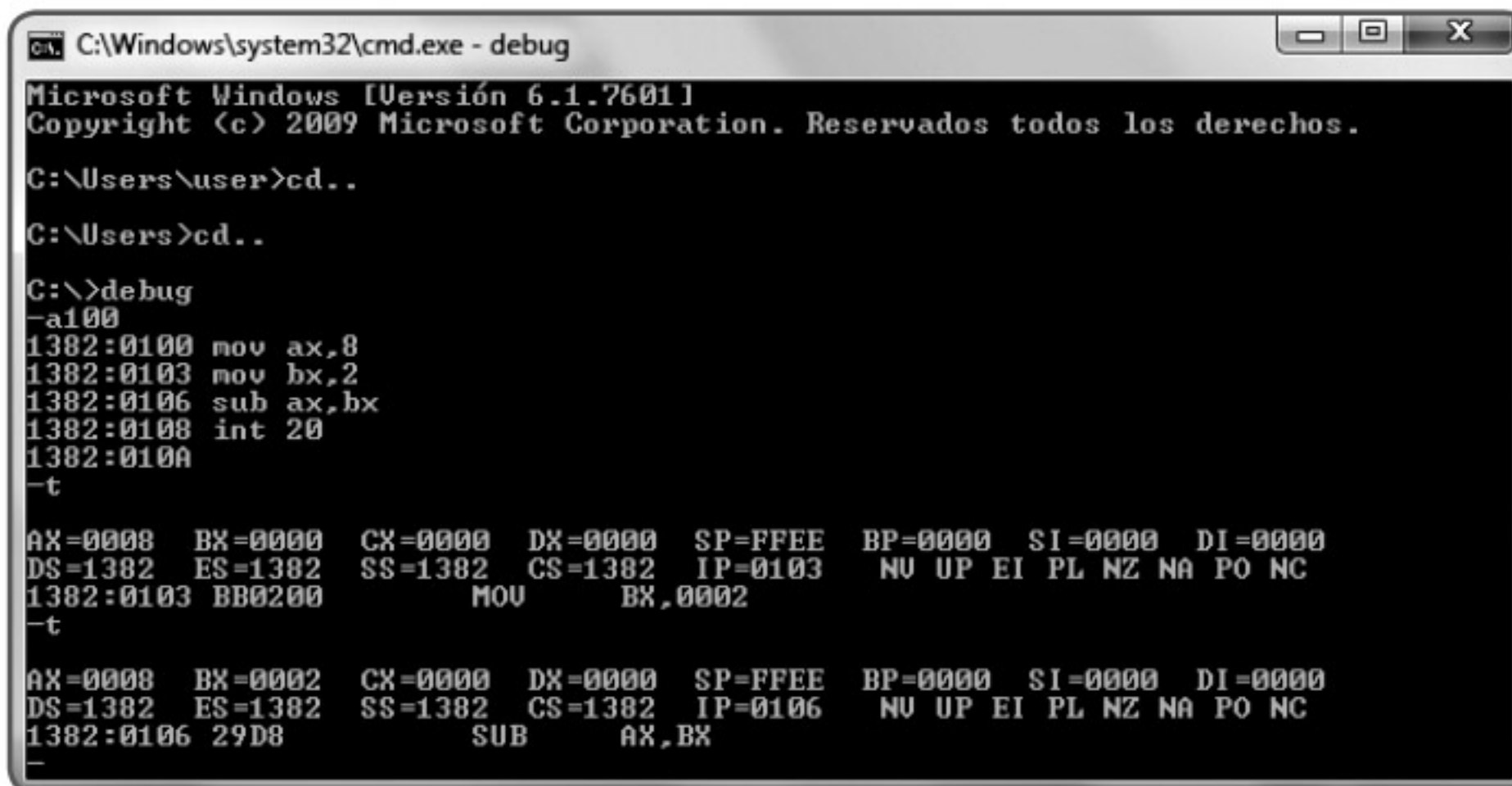


```
C:\Windows\system32\cmd.exe - debug
Microsoft Windows [Versión 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. Reservados todos los derechos.
C:\Users\user>cd..
C:\Users>cd..
C:\>debug
-a100
1382:0100 mov ax,8
1382:0103 mov bx,2
1382:0106 sub ax,bx
1382:0108 int 20
1382:010A
-t
AX=0008 BX=0000 CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=1382 ES=1382 SS=1382 CS=1382 IP=0103 NU UP EI PL NZ NA PO NC
1382:0103 BB0200 MOV BX,0002
-
```

Figura 3.21 resta\_ax

b. Mover al registro bx.

Escriba `t` y, luego, el código de la suma donde se mueve el número 2 al registro bx.



```

C:\Windows\system32\cmd.exe - debug
Microsoft Windows [Versión 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. Reservados todos los derechos.

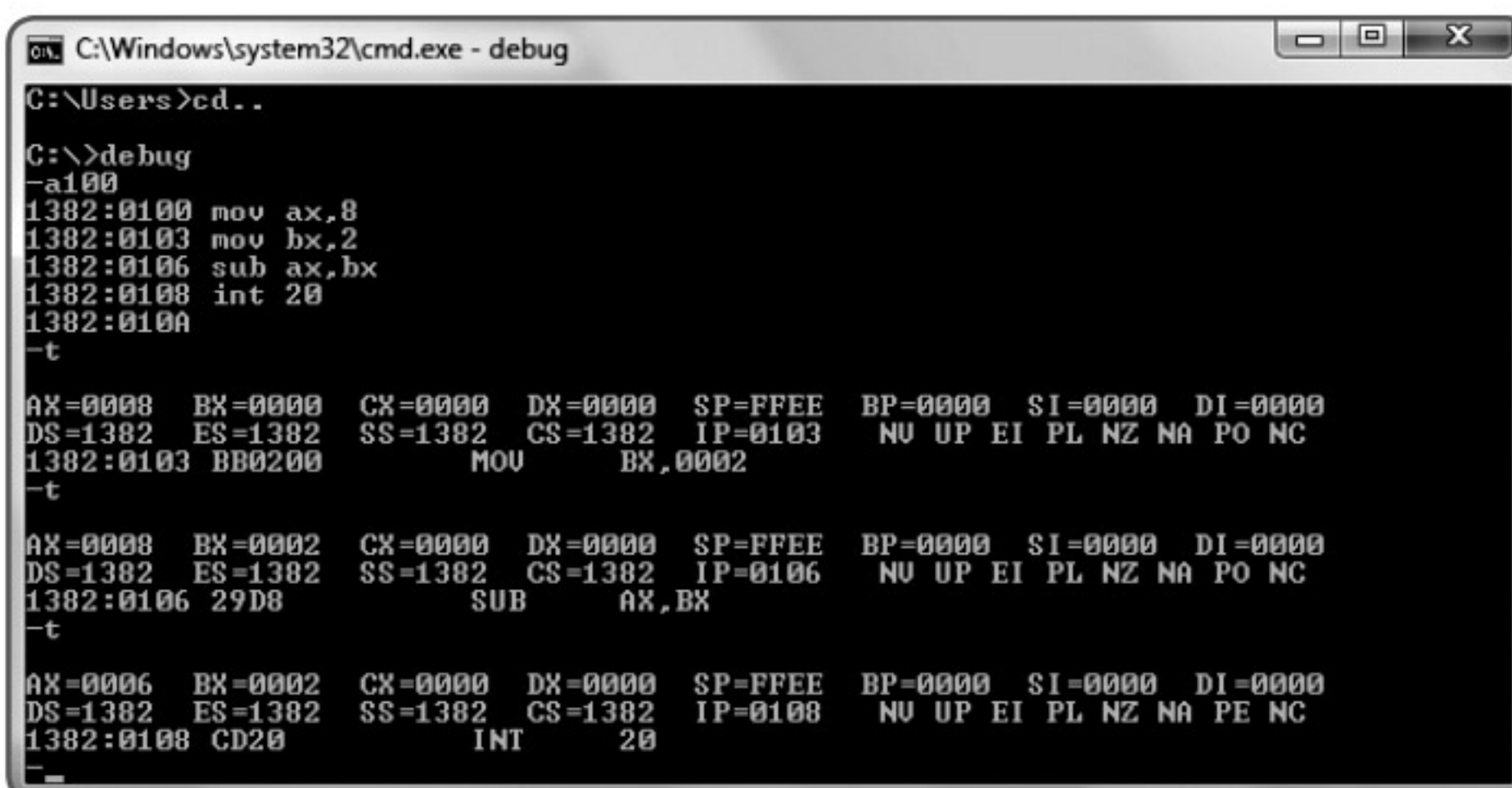
C:\Users\user>cd..
C:\Users>cd..
C:\>debug
-a100
1382:0100 mov ax,8
1382:0103 mov bx,2
1382:0106 sub ax,bx
1382:0108 int 20
1382:010A
-t
AX=0008 BX=0000 CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=1382 ES=1382 SS=1382 CS=1382 IP=0103  NU UP EI PL NZ NA PO NC
1382:0103 BB0200      MOV     BX,0002
-t
AX=0008 BX=0002 CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=1382 ES=1382 SS=1382 CS=1382 IP=0106  NU UP EI PL NZ NA PO NC
1382:0106 29D8      SUB     AX,BX

```

Figura 3.22 resta\_bx

c. Restar ax con bx.

Luego, el código de la suma donde se resta el registro ax con bx, se resta el número 8 con 2, y da como resultado 6 en el primer registro ax.



```

C:\Windows\system32\cmd.exe - debug
C:\Users>cd..
C:\>debug
-a100
1382:0100 mov ax,8
1382:0103 mov bx,2
1382:0106 sub ax,bx
1382:0108 int 20
1382:010A
-t
AX=0008 BX=0000 CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=1382 ES=1382 SS=1382 CS=1382 IP=0103  NU UP EI PL NZ NA PO NC
1382:0103 BB0200      MOV     BX,0002
-t
AX=0008 BX=0002 CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=1382 ES=1382 SS=1382 CS=1382 IP=0106  NU UP EI PL NZ NA PO NC
1382:0106 29D8      SUB     AX,BX
-t
AX=0006 BX=0002 CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=1382 ES=1382 SS=1382 CS=1382 IP=0108  NU UP EI PL NZ NA PE NC
1382:0108 CD20      INT     20

```

Figura 3.23 resta ax con bx



### 3.2 OPERANDOS LÓGICOS

La lógica booleana es usada en el diseño de circuitos similar en la lógica de programación. Las instrucciones de la lógica booleana son AND, OR, XOR, TEST y NOT, pueden usarse para poner bits en 0 o en 1 y para manejar datos ASCII con propósitos aritméticos.

- **AND:** Si ambos bits son 1, establece el resultado en 1; sino, el resultado 0.
- **OR:** Si cualquier bit es 1, el resultado es 1. Si ambos son 0, el resultado es 0.
- **XOR:** Si ambos bits son iguales, el resultado es 0; si no, el resultado es 1.

Las operaciones siguientes AND, OR y XOR ilustran los mismos valores de bits como operandos:

	AND	OR	XOR
	0101	0101	0101
	<u>0011</u>	<u>0011</u>	<u>0011</u>
Resultado:	0001	0111	0110

La instrucción NOT convierte los bits ceros en unos, y los unos en ceros.

### 3.3 MULTIPLICACIÓN

Existen dos instrucciones para la multiplicación: la instrucción MUL (datos sin signo) y la instrucción IMUL (multiplicación entera con signo). Ambas instrucciones afectan las banderas de acarreo y de desbordamiento.

Las operaciones de multiplicación básicas son byte por byte, palabra por palabra, y palabras dobles por palabras dobles.

#### A. Byte por byte

Para multiplicar dos números de un byte, el multiplicando está en el registro AL y el multiplicador es un byte en memoria o en otro registro. Para la instrucción MUL DL, la operación multiplica el contenido del AL por el contenido del DL. El producto generado se ubica en el registro AX. La operación ignora y borra cualquier información que pueda estar en el AH.

#### B. Palabra por palabra

Cuando se quiere multiplicar dos números de una palabra, el multiplicando está en el registro AX y el multiplicador es una palabra en memoria o en otro registro. Para la instrucción MUL DX, la operación multiplica el contenido del AX por el contenido del DX. El producto generado es una palabra doble que necesita dos registros. La parte de orden alto (más a la izquierda) en el DX y la parte de orden bajo (más a la derecha) en el AX. La operación ignora y borra cualquier información que está en el DX.



### C. Palabra doble por palabra doble

Si se quiere multiplicar dos números de palabras dobles, el multiplicando está en el registro EAX y el multiplicador es una palabra doble en memoria o en otro registro. El producto es generado en el par EDX:EAX y la operación ignora y borra cualquier información que ya está en el EDX.

## 3.4 DIVISIÓN

«Para la división, la instrucción DIV (dividir) maneja datos sin signo y la IDIV (división entera) maneja datos con signo. Usted es responsable de seleccionar la instrucción apropiada. Las operaciones de multiplicación básicas son byte entre byte, palabra entre palabra y palabras dobles entre palabras dobles.

### A. Palabra entre palabra

En esta operación, el dividendo está en el AX y el divisor es un byte en memoria o en otro registro. Después de la división, el residuo está en el AH y el cociente está en el AL. Un cociente de un byte es muy pequeño, sin signo, un máximo de +255 (FFH) y con signo +127 (7FH). La operación tiene un uso limitado.

### B. Palabra doble entre palabra

En esta operación, el dividendo está en el par DX:AX y el divisor es una palabra en memoria o en otro registro. Después de la división, el residuo está en el DX y el cociente está en el AX. El cociente de una palabra permite, para datos sin signo, un máximo de +32, 767 (FFFFH) y, con signo, +16, 383 (7FFFH)» (Abel, 1996).



# capítulo 4

## ORGANIZACIÓN Y MANEJO DE ARCHIVOS

### TEMAS:

- La instrucción CMPS
- La instrucción LOOP
- Instrucciones de salto condicional
- Operaciones CALL y RET
- Guía de laboratorio usando MPLAB IDE 7.61 y PROTEUS 6







## 4.1 LA INSTRUCCIÓN CMPS

Para comparar el contenido de una localidad de memoria, se usa la instrucción CMPS, que permite el incremento o disminución de los registros SI y DI en 1 para bytes, en 2 para palabras, y en 4 para palabras dobles; dependiendo de la bandera de dirección.

La operación establece las banderas AF, CF, OF, PF, SF y ZF. Algunas derivaciones de CMPS son las siguientes:

- **CMPSB:** Compara bytes.
- **CMPSD:** Compara palabras dobles.
- **CMPSW:** Compara palabras.

A continuación, se muestra la codificación del uso del CMPS y sus derivaciones:

```

TITLE                                P12CMPST (COM) Uso de CMPS para operaciones en cadena
                                .MODEL      SMALL
                                .CODE
                                ORG         100H
BEGIN:                            JMP             SHORT      MAIN
;-----
--
NOM1          DB          'Assemblers'          ;Elementos de datos
NOM2          DB          'Assemblers'
NOM3          DB          10 DUP ( ' ' )
;-----
--
MAIN          PROC        NEAR                  ;Procedimiento principal
pal
                                CLD              ;Izquierda a derecha
                                MOV             CX, 10          ;Iniciar para 10 bytes
                                LEA             DI, NOM2
                                LEA             SI, NOM1
                                REPE           CMPSB           ;Compare NOM1:NOM2
                                JNE            G20             ;No es igual, saltarlo
                                MOV            BH, 01          ;Igual, fijar BH
G20:
                                MOV             CX, 10          ;Iniciar para 10 bytes
                                LEA             DI, NOM3
                                LEA             SI, NOM2
                                REPE           CMPSB           ;Compare NOM2:NOM3
                                JE             G30             ;Igual, salir
                                MOV            BL, 02          ;No es igual, fijar BL
G30:
                                MOV             AX, 4C00H        ;Salir a DOS
                                INT             21H
MAIN          ENDP
END            BEGIN

```



## 4.2 LA INSTRUCCIÓN LOOP

La instrucción LOOP disminuye 1 de CX en cada iteración. Para ello, requiere un valor inicial en el registro CX, el control pasa a la dirección del operando; pero si el valor en el CX es cero, el control pasa a la instrucción que sigue.

La distancia debe ser un salto corto, desde -128 hasta +127 bytes; si excede el límite, el ensamblador envía el mensaje «salto relativo fuera de rango».

El siguiente programa muestra el funcionamiento de la instrucción LOOP.

Page 60,132

```

TITLE                P08LOOP (COM) Instrucción LOOP
                    .MODEL  SMALL
                    .CODE
                    ORG      100H

MAIN                PROC      NEAR
                    MOV       AX,01          ;Iniciación de AX,
                    MOV       BX,01          ;BX y
                    MOV       CX,01          ;CX a 01
                    MOV       CX,10          ;Iniciar

A20:
                    ADD       AX, 01          ;Número de iteraciones
                    ADD       BX, AX         ;Sumar 01 a AX
                    SHL       DX, 1          ;Sumar AX a BX
                    LOOP      A20            ;Multiplicar por dos a DX
                    MOV       AX, 4C00H      ;Iterar si es diferente de cero
                                          ;Salida a DOS

MAIN                ENDP

END                MAIN

```

Existen dos variaciones de la instrucción LOOP: LOOPE/LOOPZ y LOOPNE/LOOPNZ.

## 4.3 INSTRUCCIONES DE SALTO CONDICIONAL

Las instrucciones de salto condicional transfieren el control dependiendo de las configuraciones en el registro de banderas.

## 4.4 OPERACIONES CALL Y RET

«La instrucción CALL transfiere el control a un procedimiento llamado, y la instrucción RET regresa del procedimiento llamado al procedimiento original que hizo la llamada» (Abel, 1996).



## 4.5 GUÍA DE LABORATORIO USANDO MPLAB IDE 7.61 Y PROTEUS 6

La presente guía permite obtener conocimientos avanzados sobre el lenguaje ensamblador para el uso de los laboratorios de cómputo, y descubrir que el IDE MPLAB permite crear un proyecto, depurar y generar un archivo para el simulador Proteus.

### Ejercicio 1

El presente ejercicio propone realizar un programa de encendido de 8 ledes en el puerto B mediante el cambio de un interruptor en el puerto A.

#### Encendido de ledes

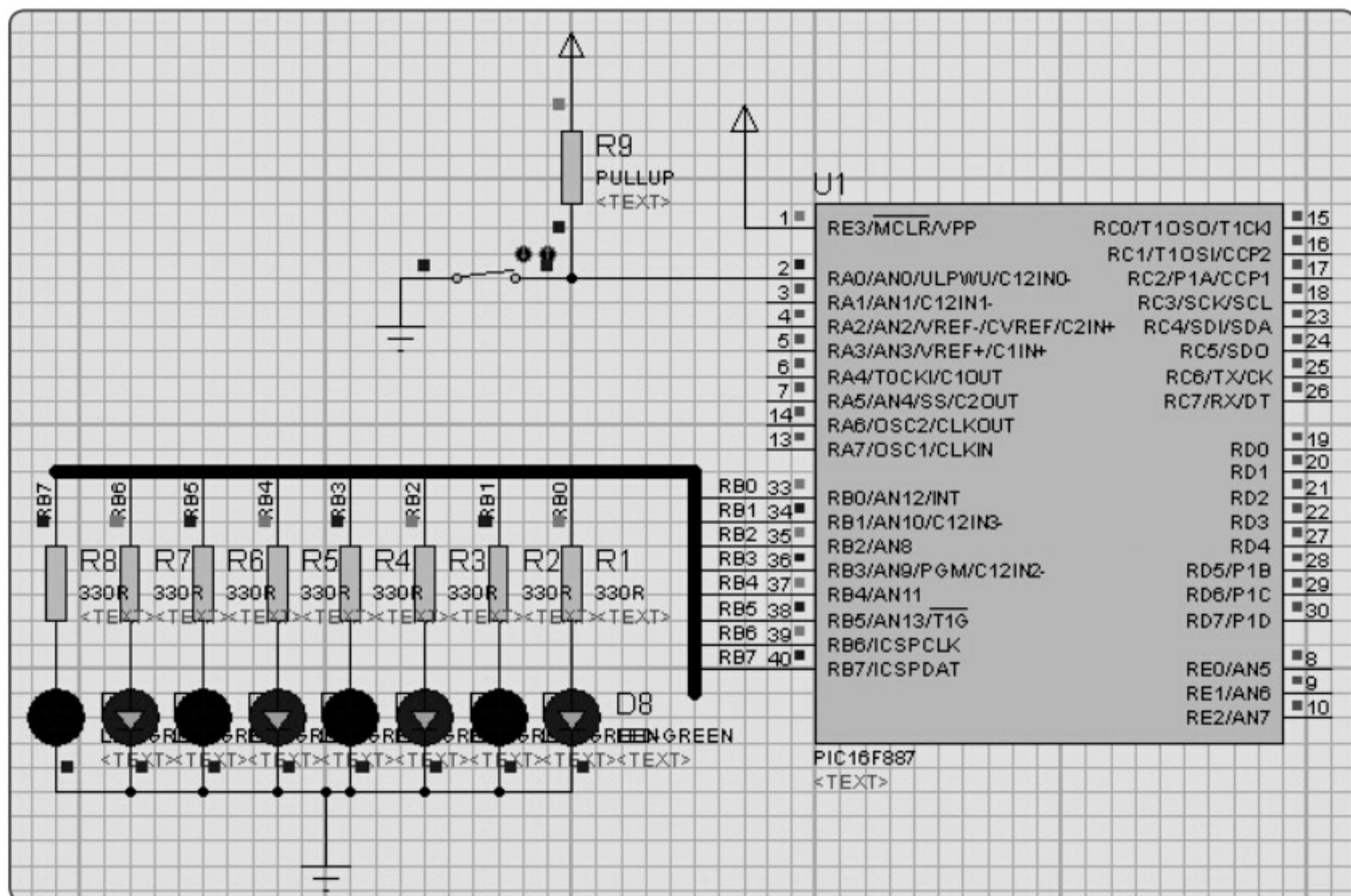
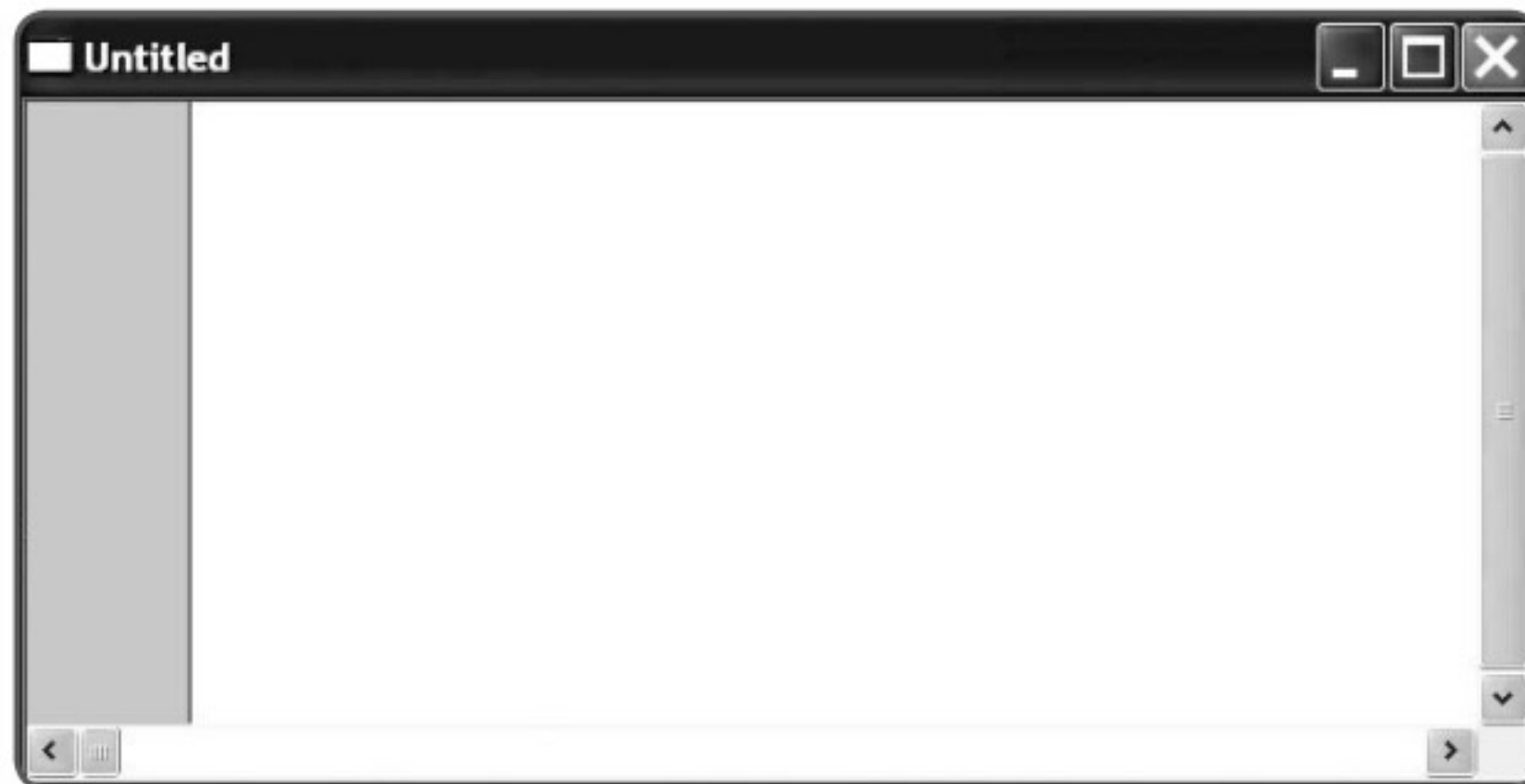


Figura 4.1 Diseño del circuito

## Creación de un proyecto nuevo

Crear un archivo .asm en el ambiente del MPLAB.

1. Abra el programa MPLAB, seleccione **File > New** y aparecerá una ventana sin título.



**Figura 4.2** Entorno de desarrollo MPLAB

2. En **Untitled**, digite el siguiente programa:

```
;PROGRAMA
; *****
; Tenemos que colocar un interruptor en RA0.
; Cuando RA0 = 1 en el Puerto B, entonces, el valor hexadecimal AA
; y cuando RA0 = 1, el valor hexadecimal 55.
; *****
LIST    p=16F887      ; Tipo de microcontrolador
INCLUDE P16F887.INC   ; Define los SFRs y bits del
                      ; P16F887

__CONFIG _CONFIG1      ; _CP_OFF&_WDT_OFF&_XT_OSC
                      ; parámetros de Configuración

errorlevel    -302    ; Deshabilita mensajes de
                      ; Advertencia por cambio
                      ; Bancos
; *****
; INICIO DEL PROGRAMA
ORG    0x00    ; Comienzo del programa (Vector de Reset)

; SETEO DE PUERTOS

BANKSEL    TRISB    ; selecciona el banco conteniendo
```



```

    CLRF    TRISB    ; puerto B configurado como salida
    BANKSEL ANSEL
    CLRF    ANSEL    ; configura puertos con entradas digitales
    CLRF    ANSELH    ; configura puertos con entradas digitales
    BANKSEL    PORTB    ; selecciona el puerto B como salida
    CLRF    PORTB
    CLRF    PORTA
; *****
; DESARROLLO DEL PROGRAMA
; *****

LOOP
    BTFSS    PORTA,0; prueba del bit 0 del puerto A
    GOTO     NUEVO_VALOR
    MOVLW    B'10101010'    ; mueve 0xAA al registro W
    MOVWF    PORTB    ; pasa el valor al puerto B
    GOTO     LOOP

NUEVO_VALOR
    MOVLW    B'01010101'    ; mueve 0x55 al registro W
    MOVWF    PORTB    ; pasa el valor al puerto B
    GOTO     LOOP

    END                ; fin del programa

```

3. Cree una carpeta de trabajo para este primer ejercicio.
4. Guarde el documento en la carpeta de trabajo y nómbrelo como **p1\_led.asm**.
5. A continuación, usando el MPLAB, seleccione el menú **Project > Wizard**, que permite crear el proyecto. Presione el botón **Siguiente >**.



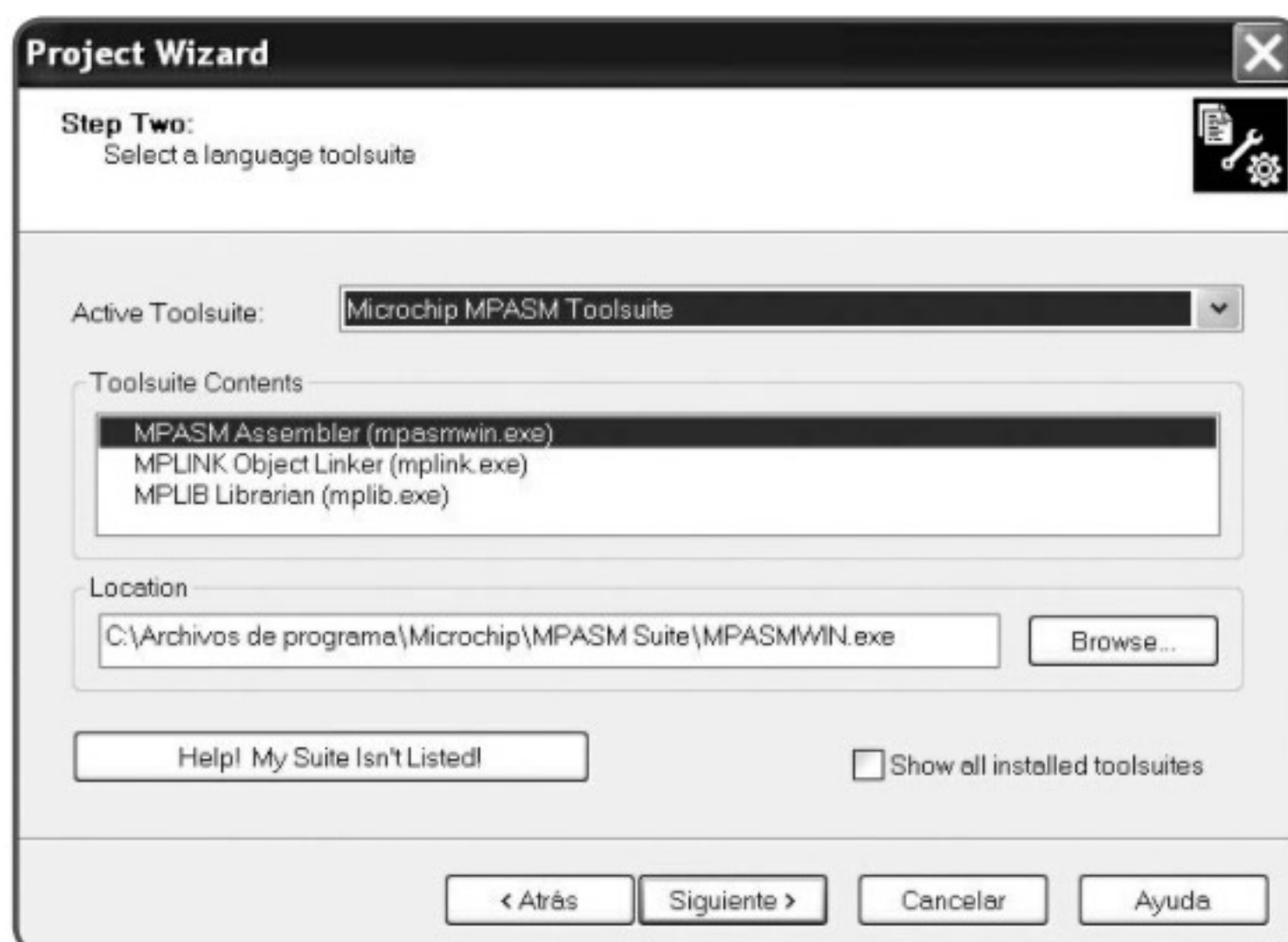
**Figura 4.3** Pantalla de bienvenida

6. Esta ventana desplegable permite seleccionar el microcontrolador que se va a utilizar. Para este ejercicio, use el 16F887. Presione el botón **Siguiente >**.



**Figura 4.4** Seleccionar dispositivo

7. Es necesario tener los contenidos del Microchip MPASM Toolsuite (en caso de no tenerlo seleccionado, tendrá que ubicarlo en la carpeta MPASM Suite, en la carpeta Microchip). Presione el botón **Siguiente >**.



**Figura 4.5** Revisar contenido Microchip MPASM



8. Recuerde que creó su carpeta. Aquí debe ubicarla con el botón **Browse....** Seleccione su directorio de trabajo para crear el directorio del proyecto.

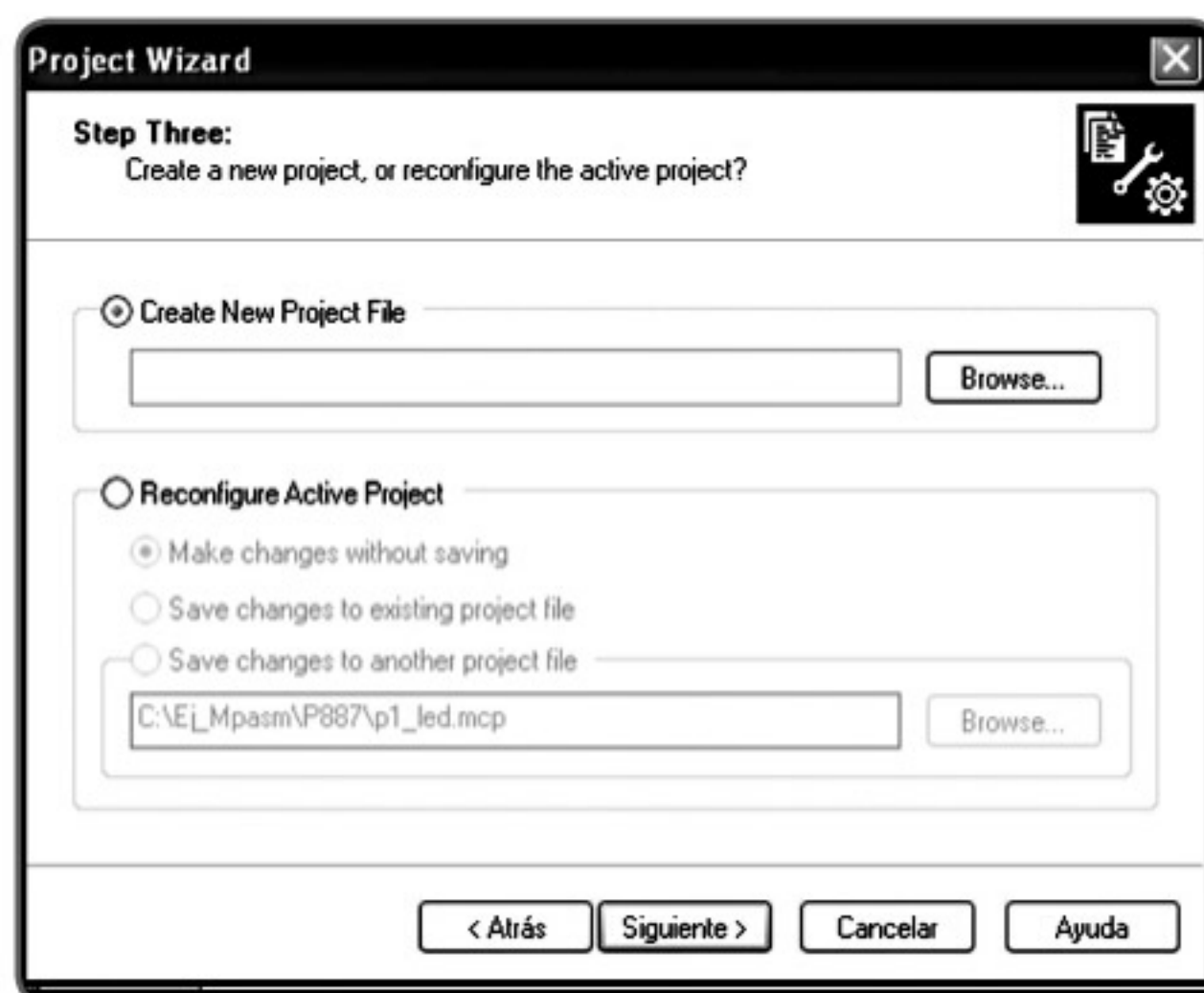


Figura 4.6 Crear Project File

9. Ingrese el nombre del proyecto **p1\_led** y seleccione el botón **Guardar**.

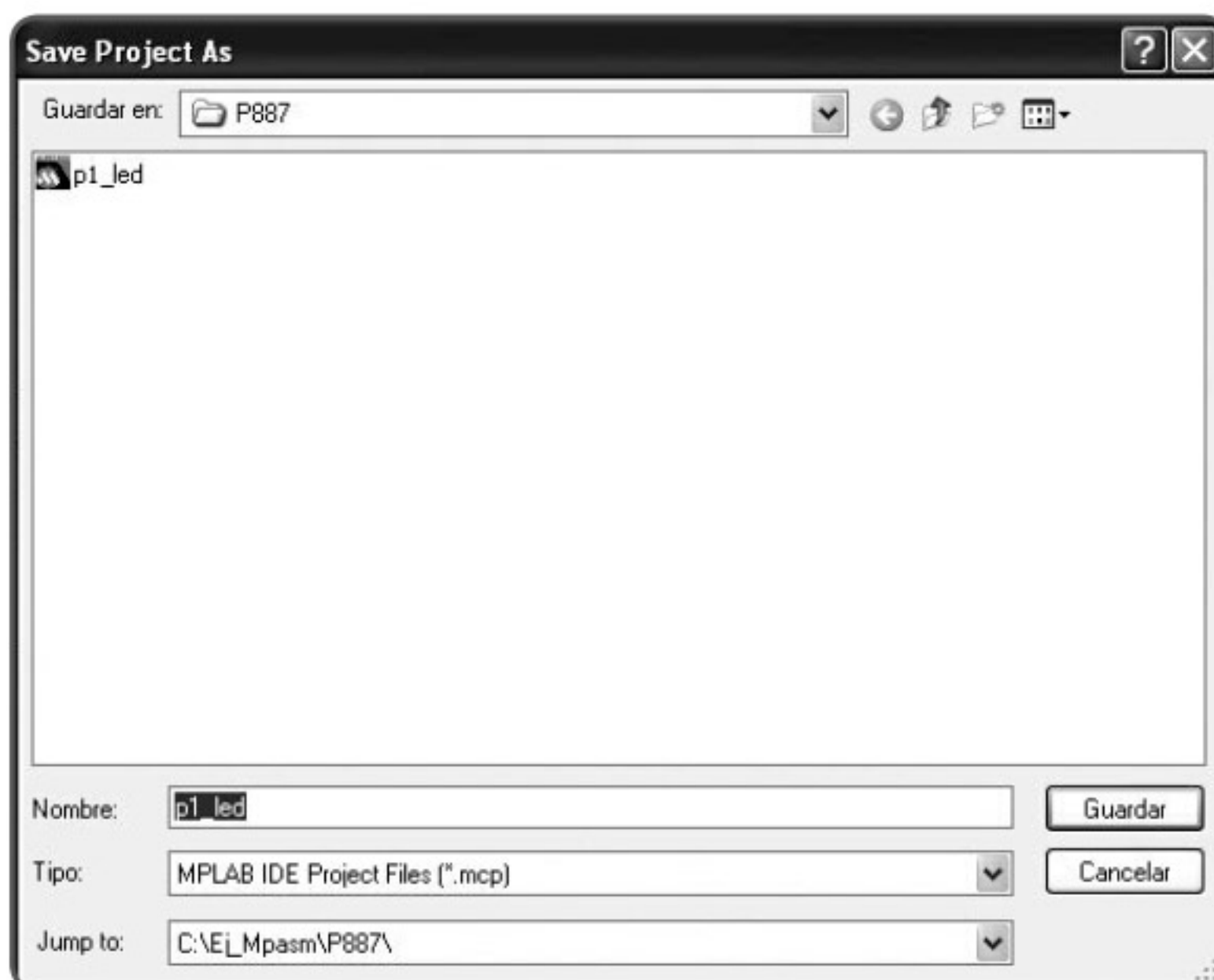


Figura 4.7 Guardar proyecto como

10. Ahora, se ve la dirección donde se encuentra ubicado el proyecto. Haga clic en el botón **Siguiente >**.

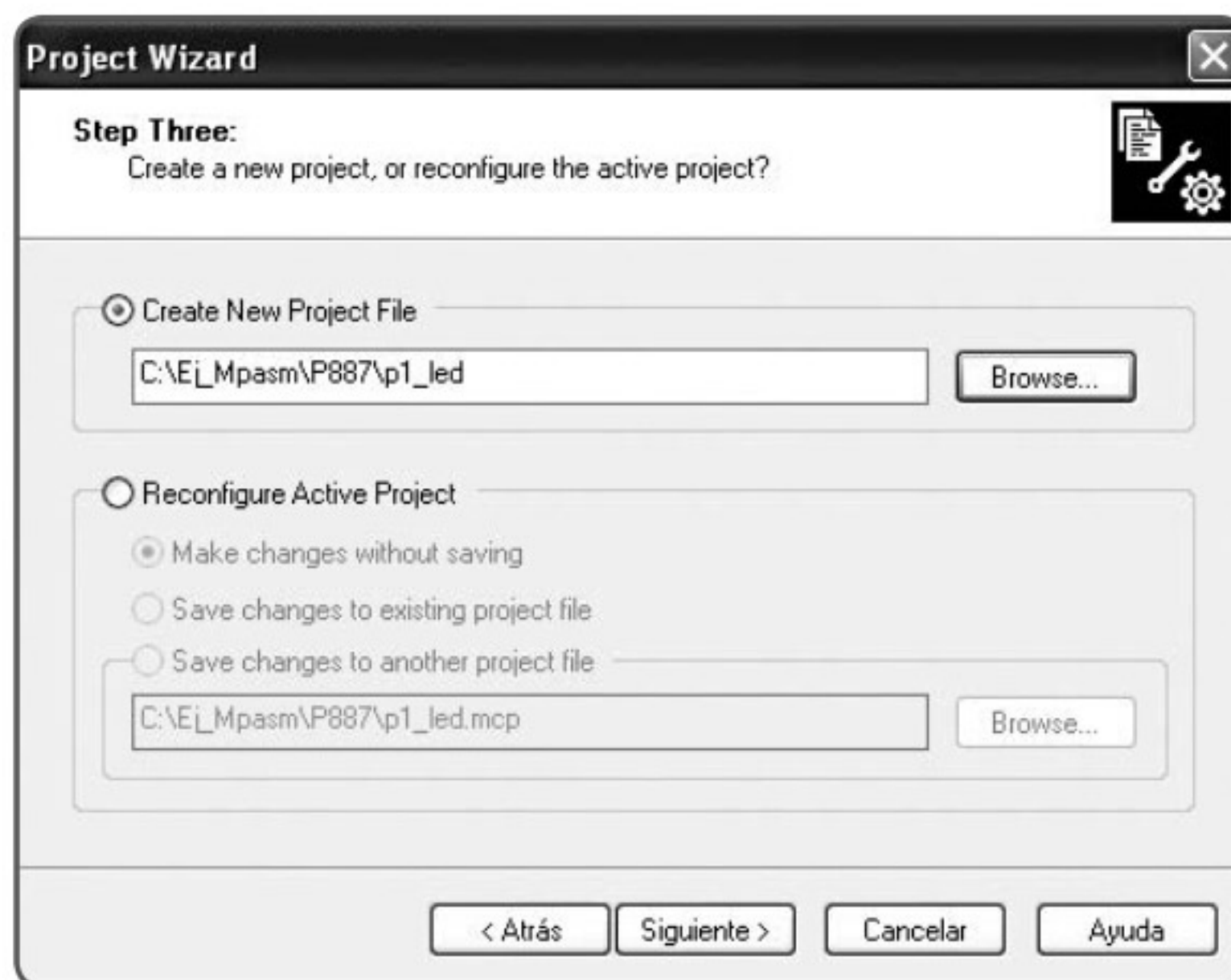


Figura 4.8 Ubicación del proyecto

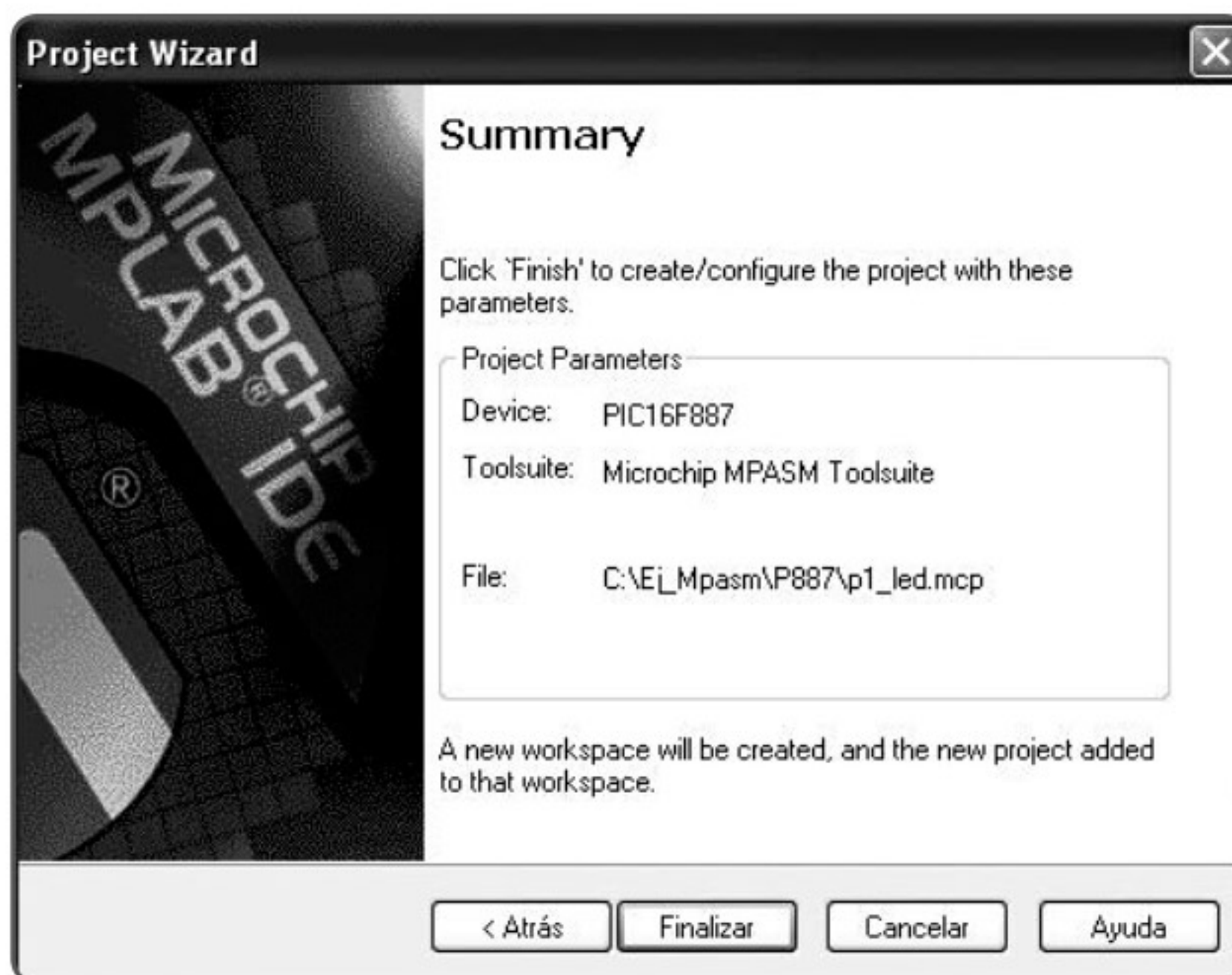
11. Ubique el archivo en la columna izquierda, selecciónelo y haga clic en **Add** para agregarlo. Presione el botón **Siguiente >**.



Figura 4.9 Agregar archivo de programación

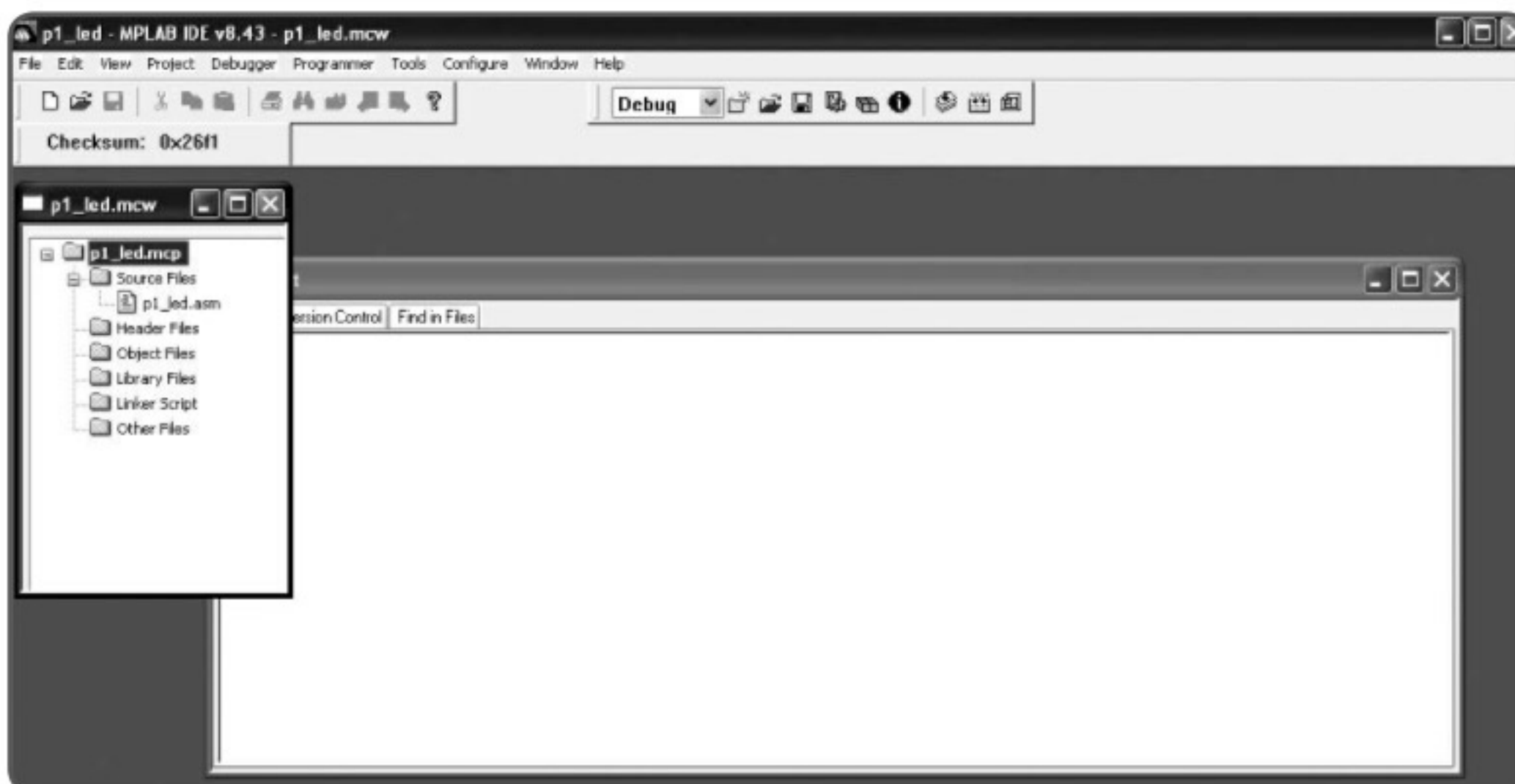


12. Se muestra el resumen de la creación del proyecto y sus parámetros. Presione el botón **Finalizar**.



**Figura 4.10** Resumen del proyecto

13. Finalmente, se muestra el IDE MPLAB.



**Figura 4.11** IDE MPLAB

14. Seleccione **Debugger > Select Tool > MPLAB SIM** y se habilitarán los íconos que permiten la ejecución paso a paso.



**Figura 4.12** Herramientas del Debugger del proyecto

15. Seleccione **Configure > Configuration Bits** y observe los valores de configuración que deben coincidir con los ingresados con la directiva `_config` en el programa.

Address	Value	Field	Category	Setting
2007	3FF1	OSC	Oscillator	XT oscillator: Crystal/resonator on RA6/OSC2/CLKIN and RA7/
		WDT	Watchdog Timer	WDT disabled
		PUT	Power Up Timer	PURT disabled
		MCLR	Master Clear Enable	RE3/MCLR pin function is MCLR
		CP	Code Protect	Program memory code protection is disabled
		CPD	Data EE Read Protect	Data memory code protection is disabled
		BOREN	Brown Out Reset Select	BOR enabled
		IESO	Internal External Swi	Internal/External Switchover mode is enabled
		FCMEN	Monitor Clock Fail-se	Fail-Safe Clock Monitor is enabled
		LVP	Low Voltage Program	RB3/PGM pin has PGM function, low voltage programming enabl
2008	3FFF	BOR4V	Brown Out Reset Select	Brown out Reset set to 4.0V
		WRT	Flash Program Memory	Write protection off

**Figura 4.13** Registro de funciones

16. Para compilar el programa `-asm`, seleccione **Project > Build All**. Si la compilación es exitosa, aparece el archivo `.asm` con una flecha verde que señala la primera instrucción a ejecutarse. En caso de errores de compilación, debe corregirlos antes de continuar. Haciendo doble clic en el error lo conduce al sitio del error.



**Figura 4.14** Build All

Una vez compilado el programa, se observa cómo funciona cada uno de los registros.



- a. Con **View > Special Function Registers**, abra los registros de funciones especiales SFR y muéstrelas al lado derecho de la pantalla.

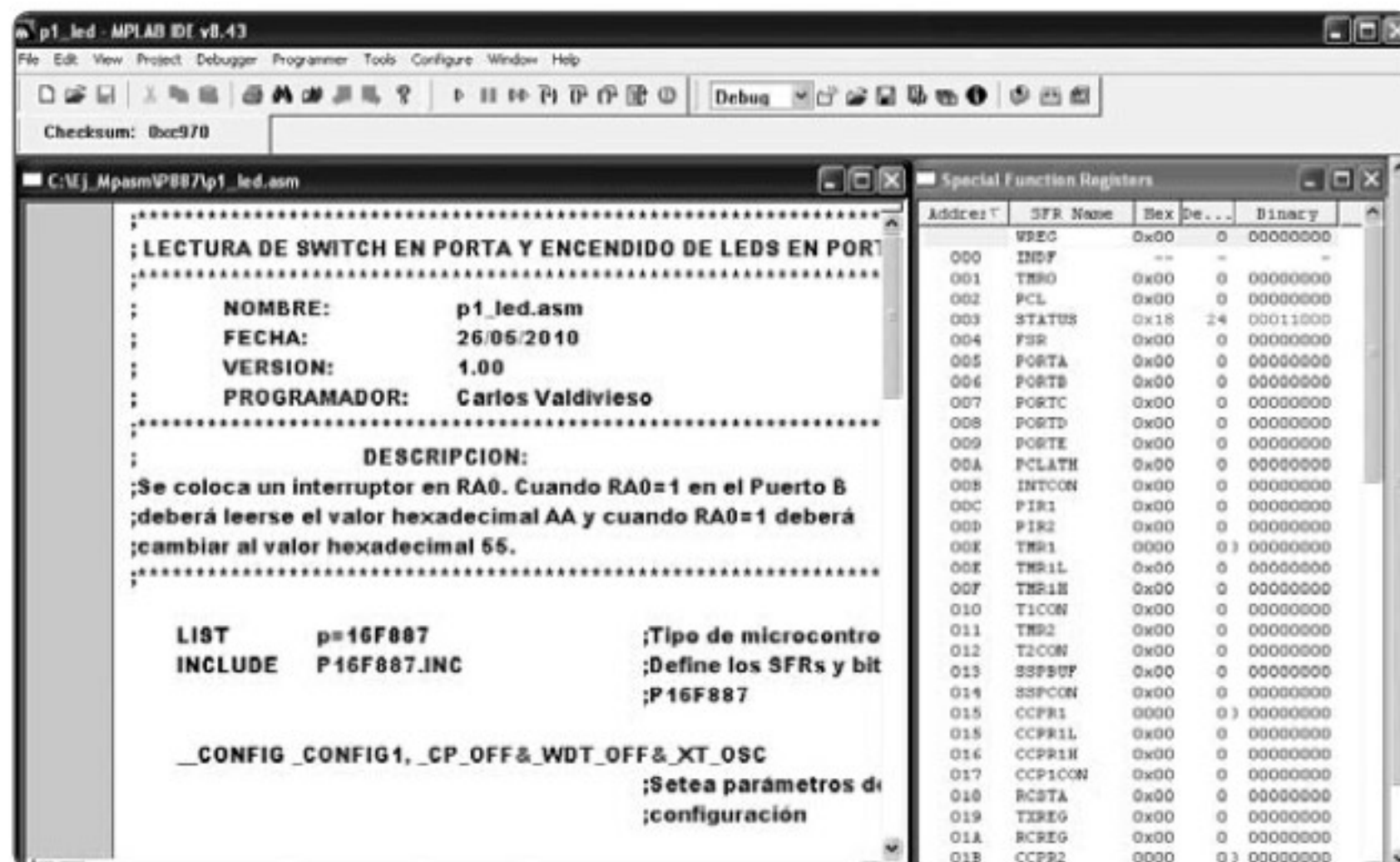


Figura 4.15 Registro de funciones

- b. Con el depurador, ejecuta paso a paso el código **Debugger > Step into** o su ícono equivalente o **F8**. Vea, en los registros SFR, los cambios que sufren de acuerdo con la ejecución de cada instrucción.
- c. Analice el comportamiento de cada uno de los íconos del MPLAB SIM con **Debugger > Select Tool > MPLAB SIM**.
- d. Con **View > File Registers**, observe el contenido de los cuatro bancos de datos del 16F887.

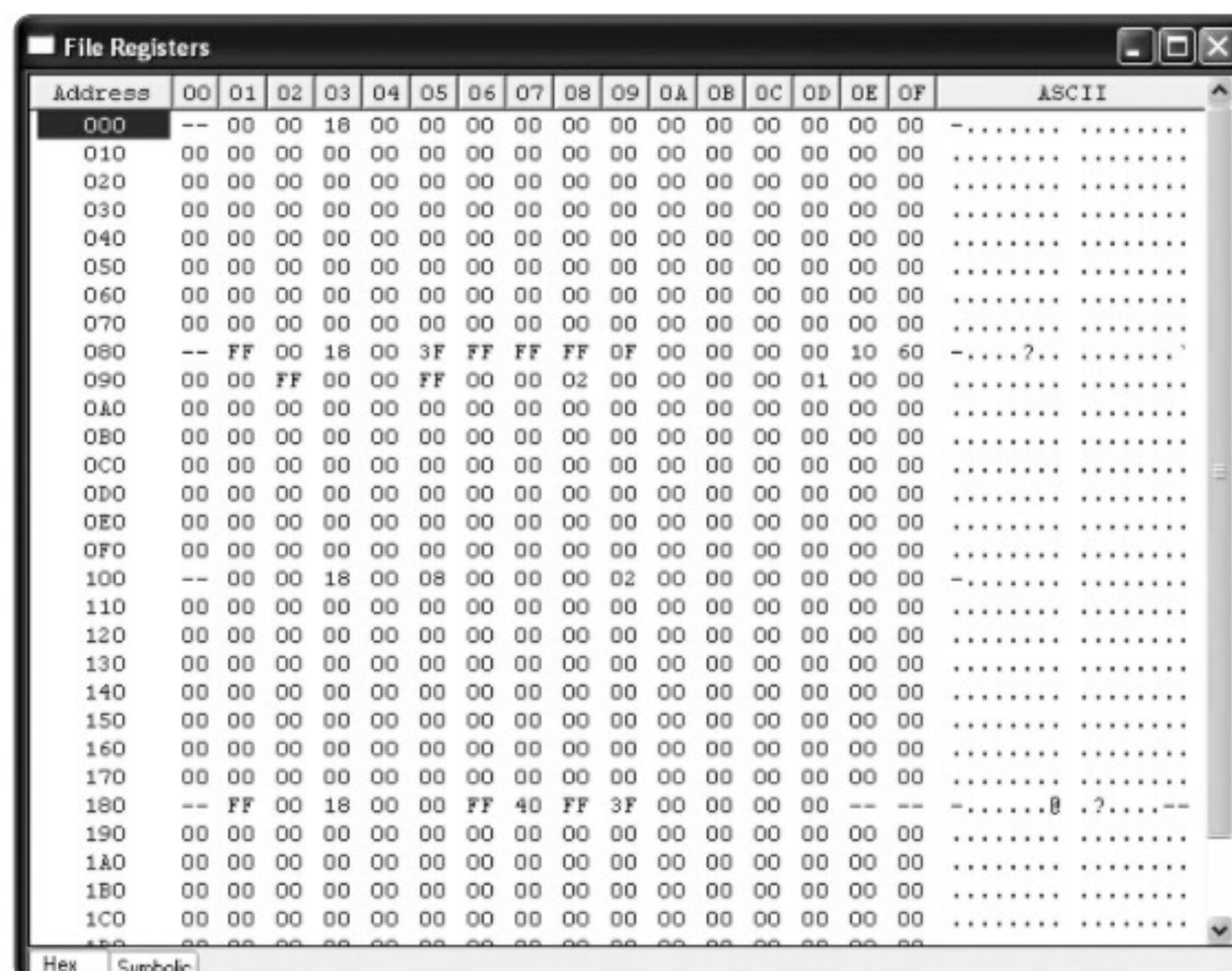
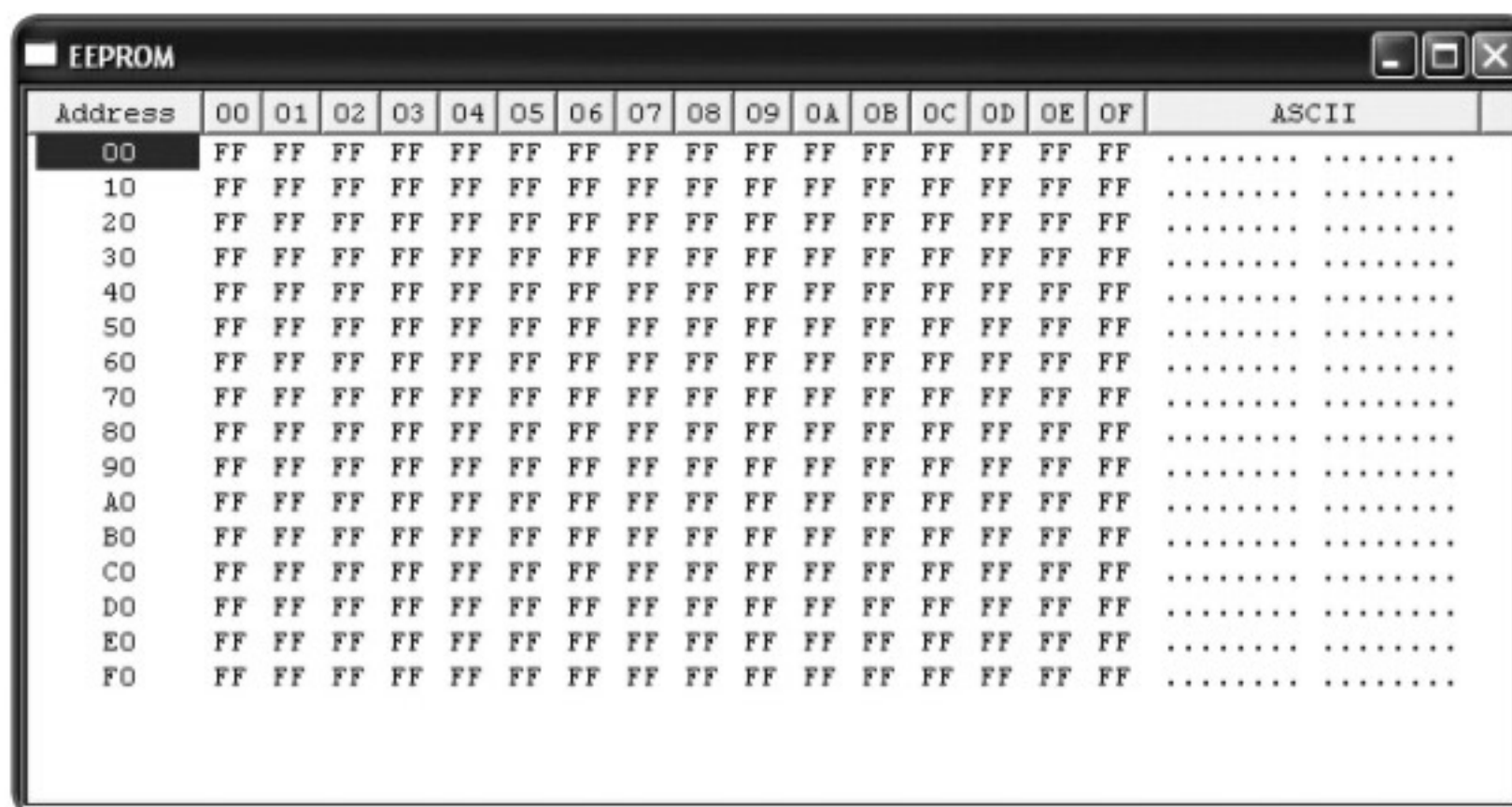


Figura 4.16 Archivo de registro



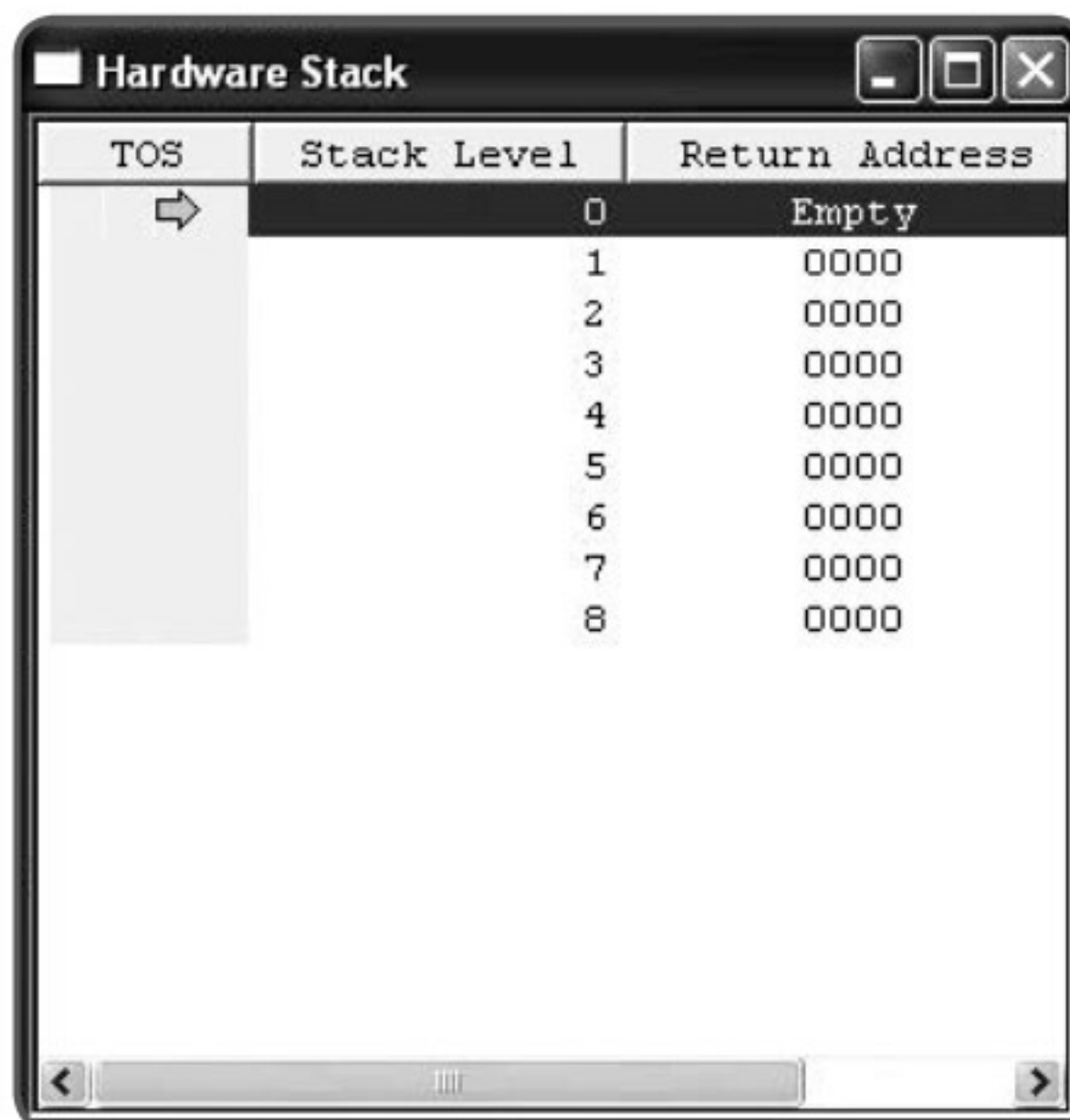
- e. Con **View > EEPROM**, observe el contenido de las 256 posiciones de memoria EEPROM del 16F887.



Address	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	ASCII
00	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	.....
10	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	.....
20	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	.....
30	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	.....
40	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	.....
50	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	.....
60	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	.....
70	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	.....
80	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	.....
90	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	.....
A0	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	.....
B0	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	.....
C0	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	.....
D0	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	.....
E0	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	.....
F0	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	.....

**Figura 4.17** EEPROM

- f. Con **View > Hardware Stack**, observe los 8 niveles de stack disponibles en los microcontroladores de la gama media.

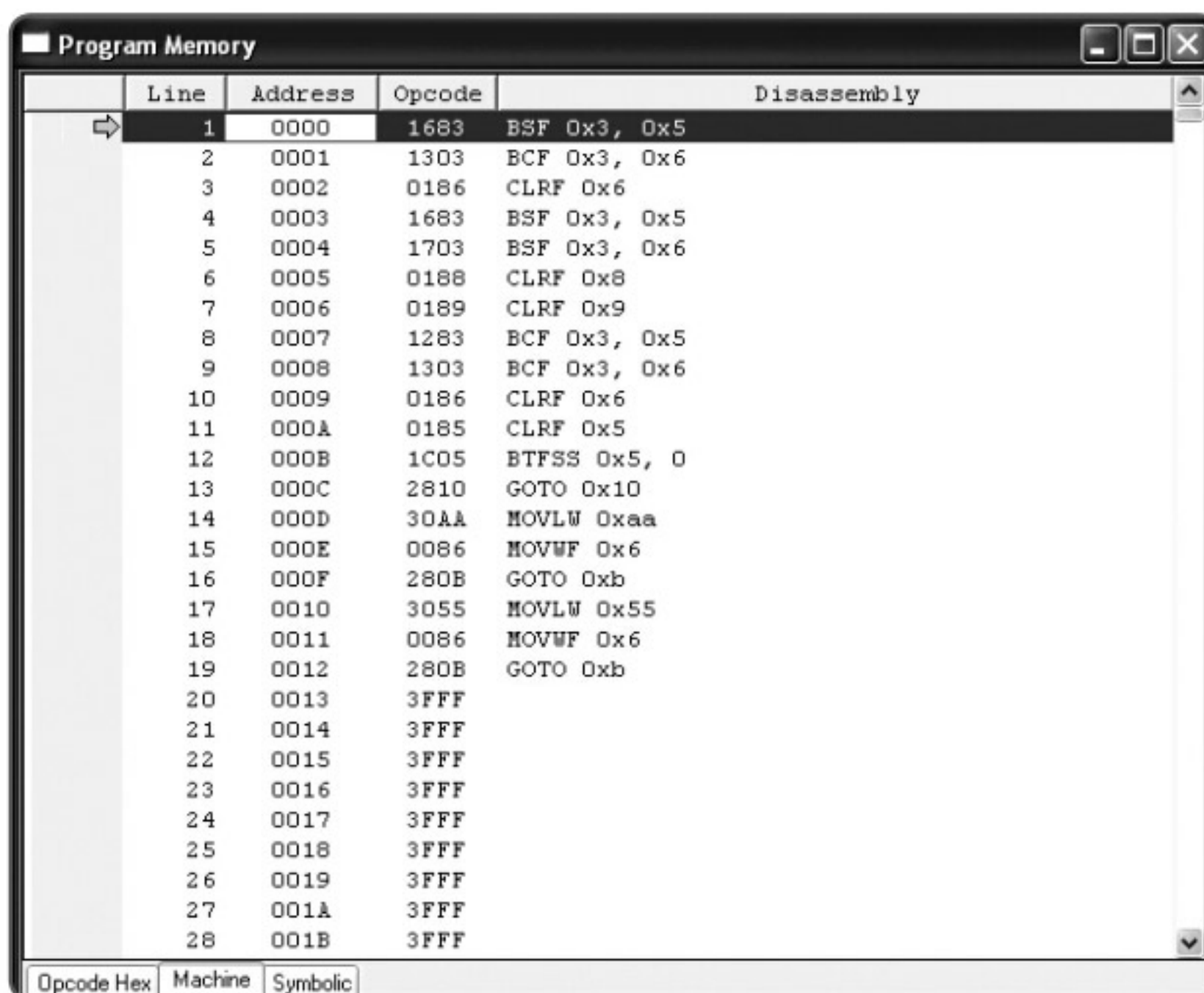


TOS	Stack Level	Return Address
⇒	0	Empty
	1	0000
	2	0000
	3	0000
	4	0000
	5	0000
	6	0000
	7	0000
	8	0000

**Figura 4.18** Hardware Stack



- g. Con **View > Program Memory**, vea a detalle de cada una de las instrucciones en la memoria de programa.



Line	Address	Opcode	Disassembly
1	0000	1683	BSF 0x3, 0x5
2	0001	1303	BCF 0x3, 0x6
3	0002	0186	CLRF 0x6
4	0003	1683	BSF 0x3, 0x5
5	0004	1703	BSF 0x3, 0x6
6	0005	0188	CLRF 0x8
7	0006	0189	CLRF 0x9
8	0007	1283	BCF 0x3, 0x5
9	0008	1303	BCF 0x3, 0x6
10	0009	0186	CLRF 0x6
11	000A	0185	CLRF 0x5
12	000B	1C05	BTFSS 0x5, 0
13	000C	2810	GOTO 0x10
14	000D	30AA	MOVLW 0xaa
15	000E	0086	MOVWF 0x6
16	000F	280B	GOTO 0xb
17	0010	3055	MOVLW 0x55
18	0011	0086	MOVWF 0x6
19	0012	280B	GOTO 0xb
20	0013	3FFF	
21	0014	3FFF	
22	0015	3FFF	
23	0016	3FFF	
24	0017	3FFF	
25	0018	3FFF	
26	0019	3FFF	
27	001A	3FFF	
28	001B	3FFF	

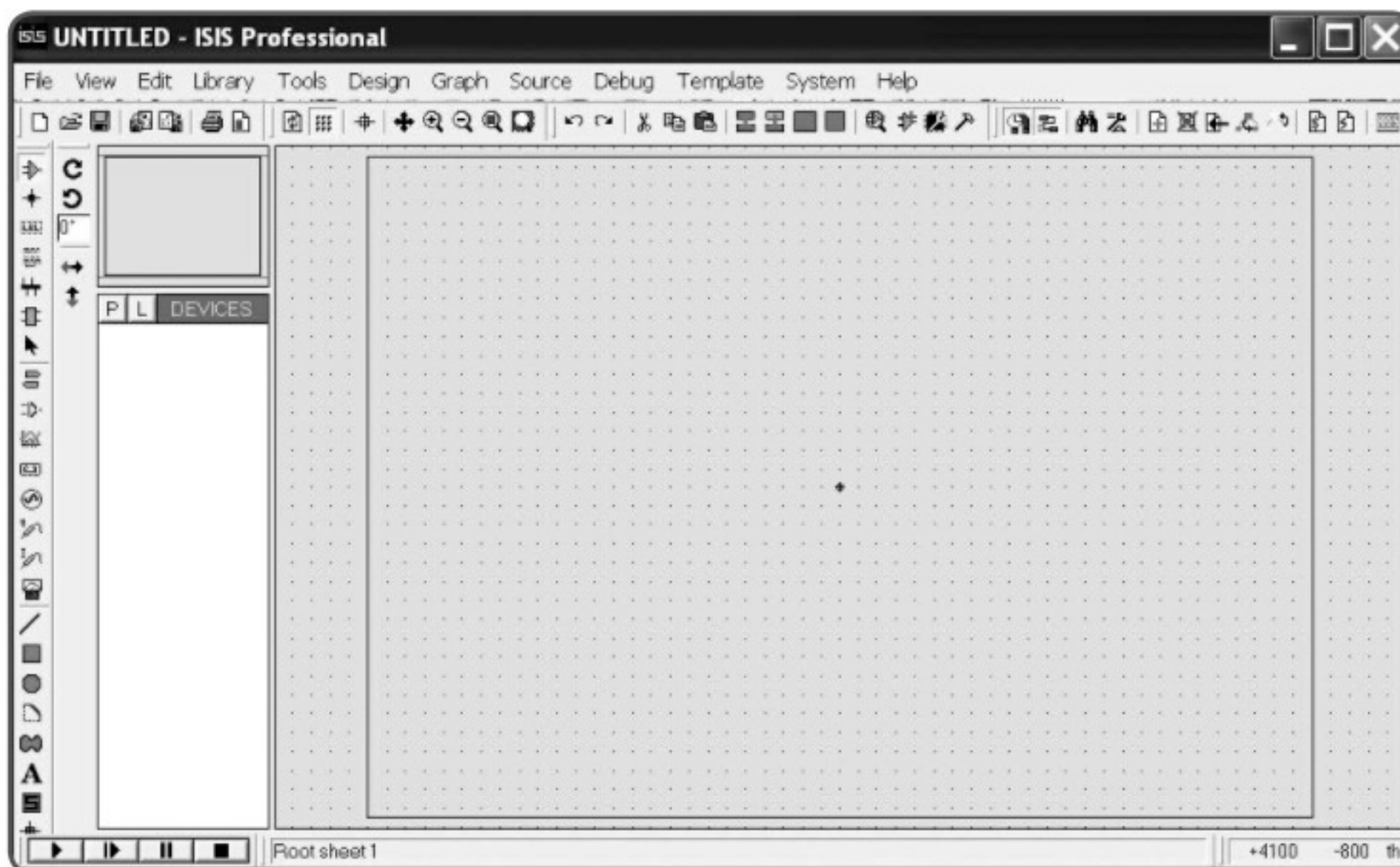
**Figura 4.19** Memoria del programa

Como resultado de la compilación, se genera un archivo con extensión **.hex** que es el ejecutable que se usa para la simulación en PROTEUS y con la tableta de control PIC.

- Se conecta el programador **Pic Start Plus** en el puerto serial.
- Ahora, diríjase a **Programmer > Select Programmer > PICSTART Plus**.
- Luego, **Programmer > Enable Programmer**.
- Con **Programmer > Program**, se programa el PIC (se puede observar la oscilación del led).

## Simulación en PROTEUS

Para minimizar los costos que se requieren para obtener los dispositivos (PIC), Protobar, cableado y energía existe el programa PROTEUS que permite la simulación del comportamiento del PIC, usando el archivo de extensión **.hex**.



**Figura 4.20** PROTEUS – ISIS Professional

1. Seleccione **Component** y, luego, **P (Pick Device)** para seleccionar los componentes a utilizar.
2. En **Keywords**, ingrese el nombre del PIC **16F887** y, con un doble clic, aparecerá dicho elemento en el ambiente de trabajo. Continúe ubicando el **switch**, **res 330R** y **led green** en el ambiente de trabajo.
3. Guarde la hoja de ISIS con el nombre **led1**.
4. Arrastre el mouse y, con un clic sostenido, ubique los componentes y únalos con buses.



5. El resultado se presenta en la siguiente gráfica:

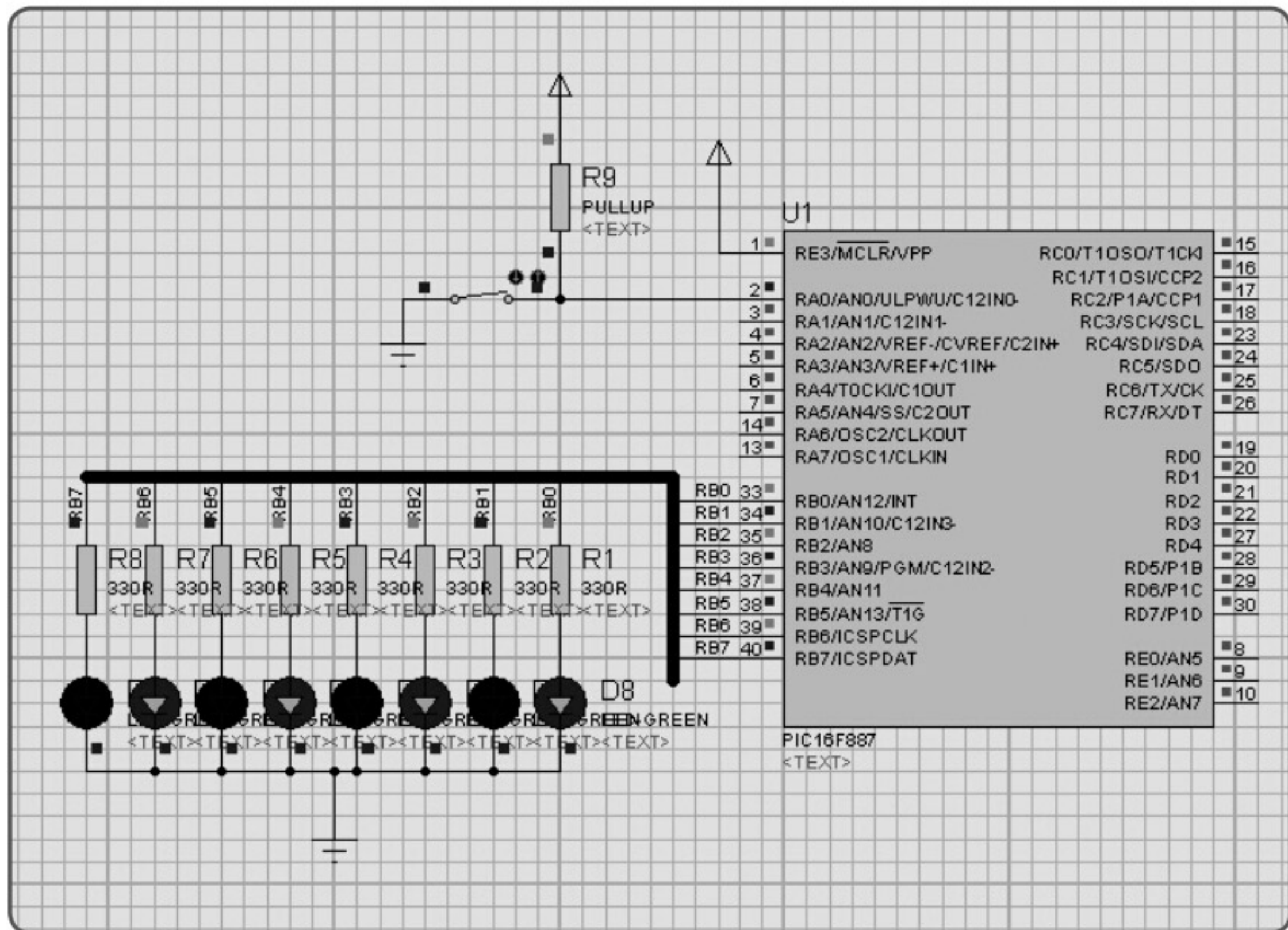


Figura 4.21 Diseño del PIC

- Ubique el microcontrolador y, sobre la imagen de dicho dispositivo, haga doble clic. En la ventana que se abre seleccione, en la línea de **Program File**, el ícono de la carpeta. Aparece un directorio de búsqueda en donde tiene que dirigirse al archivo **p1\_led.hex** de su archivo anterior MPLAB ubicado en su carpeta de trabajo. Luego, presione **OK**. A continuación, ya puede realizar la simulación. (Véase la ventana **Edit Component**, pág. 90).
- En el margen inferior izquierdo, ubique el botón **PLAY**, presiónelo para dar inicio a la simulación.
- Verifique paso a paso el encendido y apagado de los ledes, con ello culmina la simulación.

Los siguientes ejercicios solo contienen el código assembler para MPLAB y el diseño del circuito en PROTEUS.



## Ejercicio 2

Mostrar el barrido de luces ledes con un 16F84A.

```

LIST P=16F84A,      ; usar PIC 16F84A
#include <p16f84A.inc>
__CONFIG _CP_OFF&_PWRTE_ON&_WDT_OFF&_XT_OSC

PDel0      equ      0C
PDel1      equ      0D

        ORG      0
        BSF      STATUS,5          ; activa la pagina 1
        MOVLW    B'00000'          ; carga 00000 en W
        MOVWF    TRISA             ; puerto a todos salidas
        MOVLW    B'00000000'       ; carga 00000000 en W
        MOVWF    TRISB             ; puerto b todos salidas
        BCF      STATUS,5          ; volvemos a la pagina 0
        CLRF     PORTB             ; ponemos a cero el puerto b

INICIO    ; etiqueta
        BSF      PORTB,0           ; prende RB0
        BCF      STATUS,0          ; limpia el carry de STATUS,C

REPETIR
        IZQ
        CALL     DEMORA            ; demora de 100ms
        RLF      PORTB,1           ; rota el contenido de portb a la derecha
        BTFSS    PORTB,7           ; hasta que prenda RB7, luego se salta
        GOTO     IZQ               ; una linea
        DER
        CALL     DEMORA            ; demora de 100 ms
        RRF      PORTB,1           ; rota el contenido de portb a la izquierda
        BTFSS    PORTB,0           ; hasta que prenda RB0, luego salta
        GOTO     DER               ; una linea
        GOTO     REPETIR           ; repite el ciclo
        GOTO     INICIO            ; va a inicio

;-----
;   La demora a sido generada con el programa PDEL
;   Descripcion: Delay 100000 ciclos - 100 ms
;-----

DEMORA
        movlw    .110              ; 1 set numero de repeticion (B)

        movwf    PDel0             ; 1 |

```



```

PLoop1      movlw      .181      ; 1 set numero de repeticion (A)
            movwf      PDel1      ; 1 |
PLoop2      clrwdt      ; 1 clear watchdog

            clrwdt      ; 1 ciclo delay
            decfsz     PDel1, 1    ; 1 + (1) es el tiempo 0 ? (A)
            goto      PLoop2      ; 2 no, loop
            decfsz     PDel0, 1    ; 1 + (1) es el tiempo 0 ? (B)
            goto      PLoop1      ; 2 no, loop

PDelL1      goto      PDelL2      ; 2 ciclos delay
PDelL2      goto      PDelL3      ; 2 ciclos delay
PDelL3      clrwdt      ; 1 ciclo delay
            return      ; 2+2 Fin.
;-----
END

```

### Diseño en PROTEUS

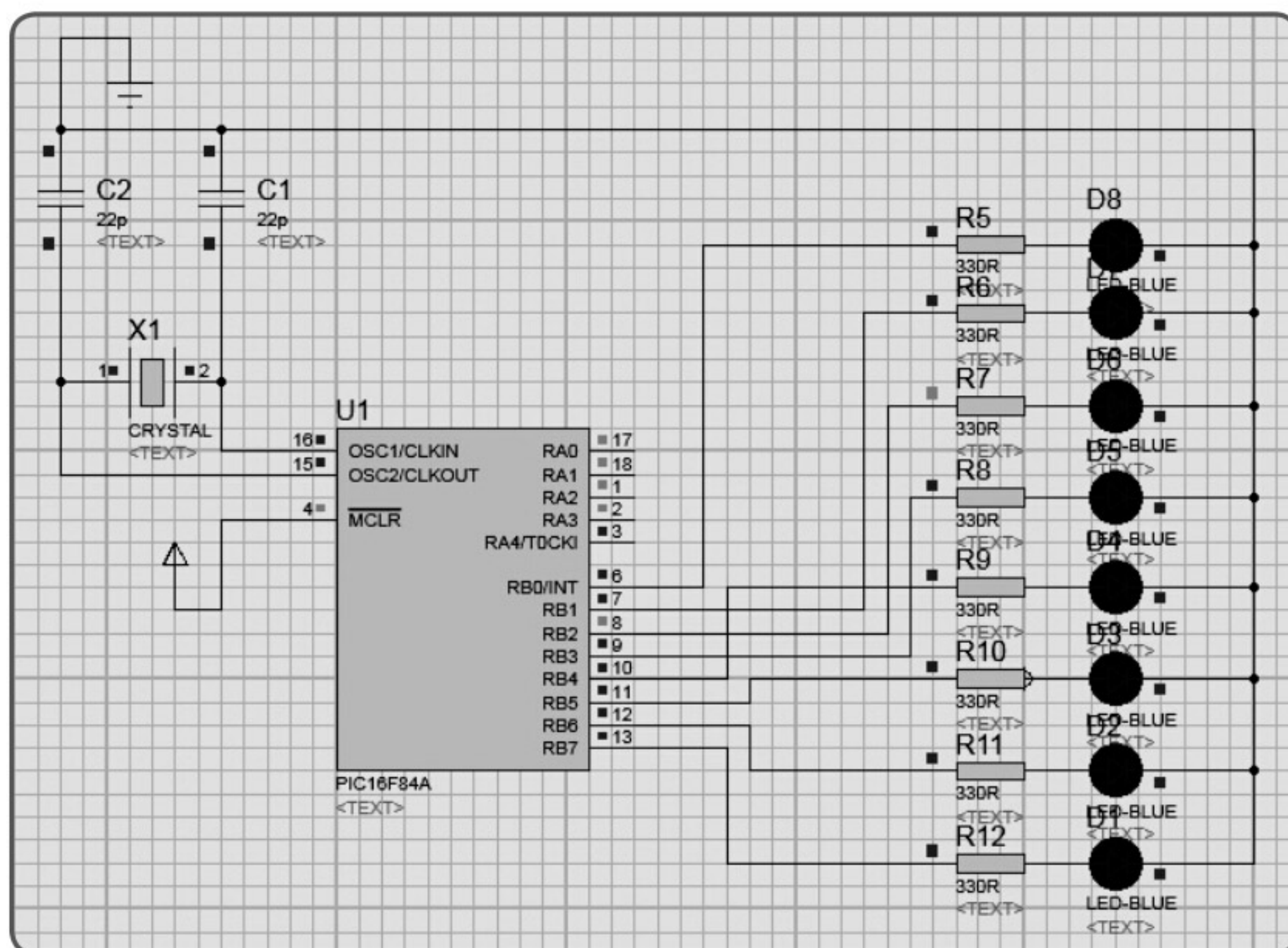


Figura 4.22 Diseño del PIC2

### Ejercicio 3

Realizar un semáforo con PIC 16F84A.

```
#include <p16f84A.inc>
__CONFIG _CP_OFF & _PWRTE_ON & _WDT_OFF & _XT_OSC
; Asignacion de variables utilizadas en demora
CONT equ 0Ch
CONT2 equ 0Dh
CONT3 equ 0Eh
CONT4 equ 0Fh

ORG 0000h
    bsf          STATUS,5      ; Cambiamos al banco 1 Switch to Bank 1
    movlw        0000h         ; Ponemos los pines del puerto A
    movwf        TRISA         ; como salidas.
    bcf          STATUS,5      ; Volvemos al Banco 0.

LUZROJA    movlw    b'0000010' ; Encendemos el LUZROJA poniendo pri-
mero
el Valor, por 20 segundos.
    movwf        PORTA
                call RETARDO
LUZAMARILLA movlw    b'00000110' ; Encendemos LUZROJA Y LUZ AMARILLA.
    movwf        PORTA
                call RETARDO1
                movlw b'00000000'

RETARDO ; Retardo de 20s
                movlw d'10'
                movwf CONT4

CICLO4    movlw d'100'
                movwf CONT3

CICLO3    movlw d'98'
                movwf CONT2

CICLO2    movlw d'67'
                movwf CONT

CICLO    decfsz CONT,1
                goto CICLO
                decfsz CONT2,1
                goto CICLO2
                decfsz CONT3,1
```



```

        goto CICLO3
        decfsz CONT4,1
        goto CICLO4
        Return

RETARDO1; Reatardo de 5s
        movlw d'100'
        movwf CONT3

CICLO6   movlw d'100'
        movwf CONT2

CICLO5   movlw d'166'
        movwf CONT

CICLO4   decfsz CONT,1
        goto CICLO4
        decfsz CONT2,1
        goto CICLO5
        decfsz CONT3,1
        goto CICLO6
        Return

END

```

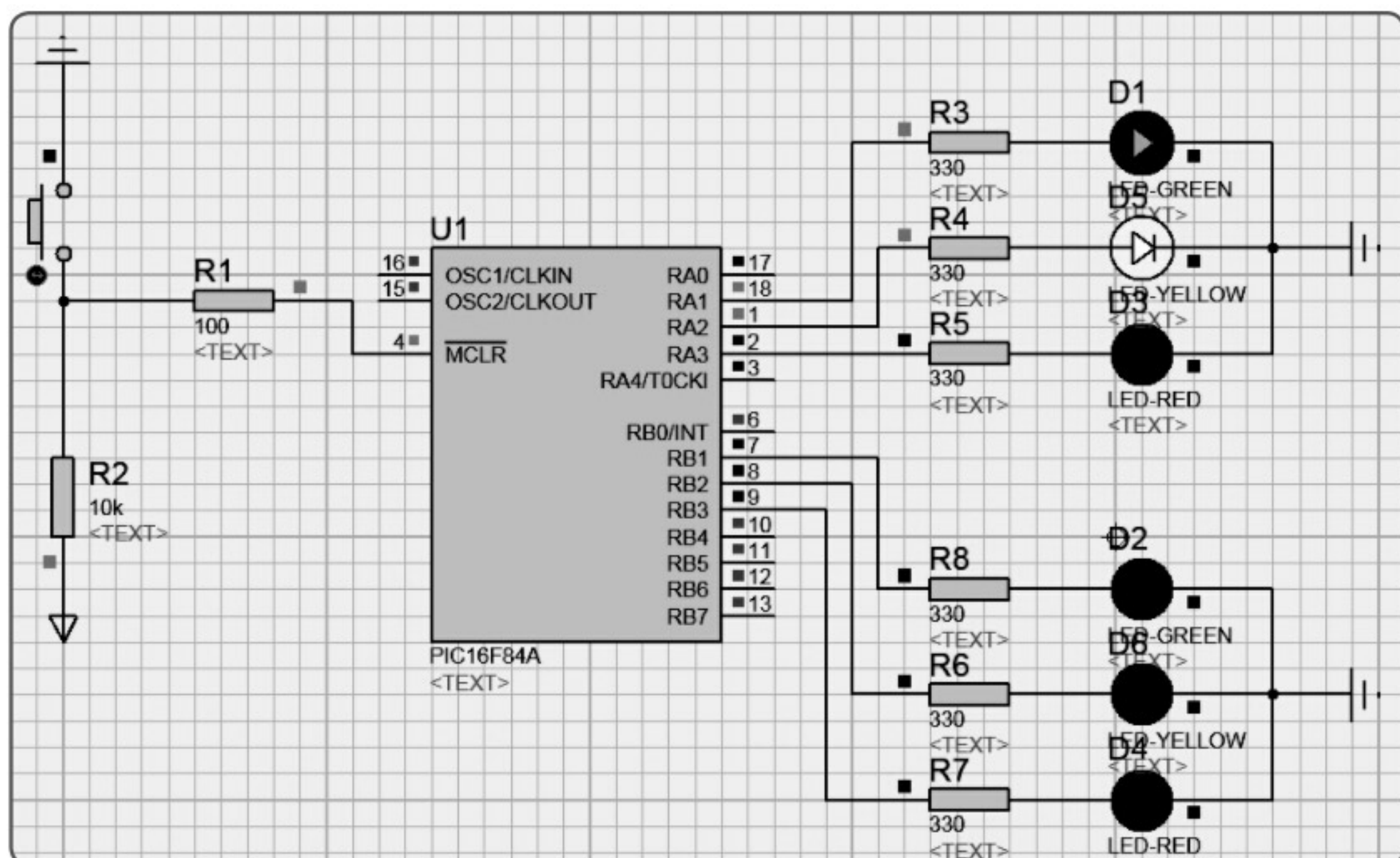


Figura 4.23 Semáforo



## Ejercicio 4

Realizar un programa con 8 ledes que encienda los bits impares incluyendo el 0 y salida por el puerto B.

```

LIST P=16F84A           ;comando que indica el pic usado.
RADIX      HEX          ;los valores en hexadecimal

STATUS     EQU    0x03   ;direcciona al registro de STATUS
PTOB EQU    0x06         ;direcciona al puerto B

        ORG    0x00
        BSF    STATUS,5   ;seleccionar el banco 1
        MOVLW  0x00       ;carga w con el valor 00h
        MOVWF  PTOB       ;para habilitar el puerto B como salida
        BCF    STATUS,5   ;seleccionar el banco 0

        CLRF   PTOB       ;limpia el puerto B
CICLO     MOVLW  0xAB      ;cargamos registro w con el numero ABh
        MOVWF  PTOB       ;enviamos el registro w al puerto B
        GOTO   CICLO      ;ir a ciclo
        END             ;fin del programa

```

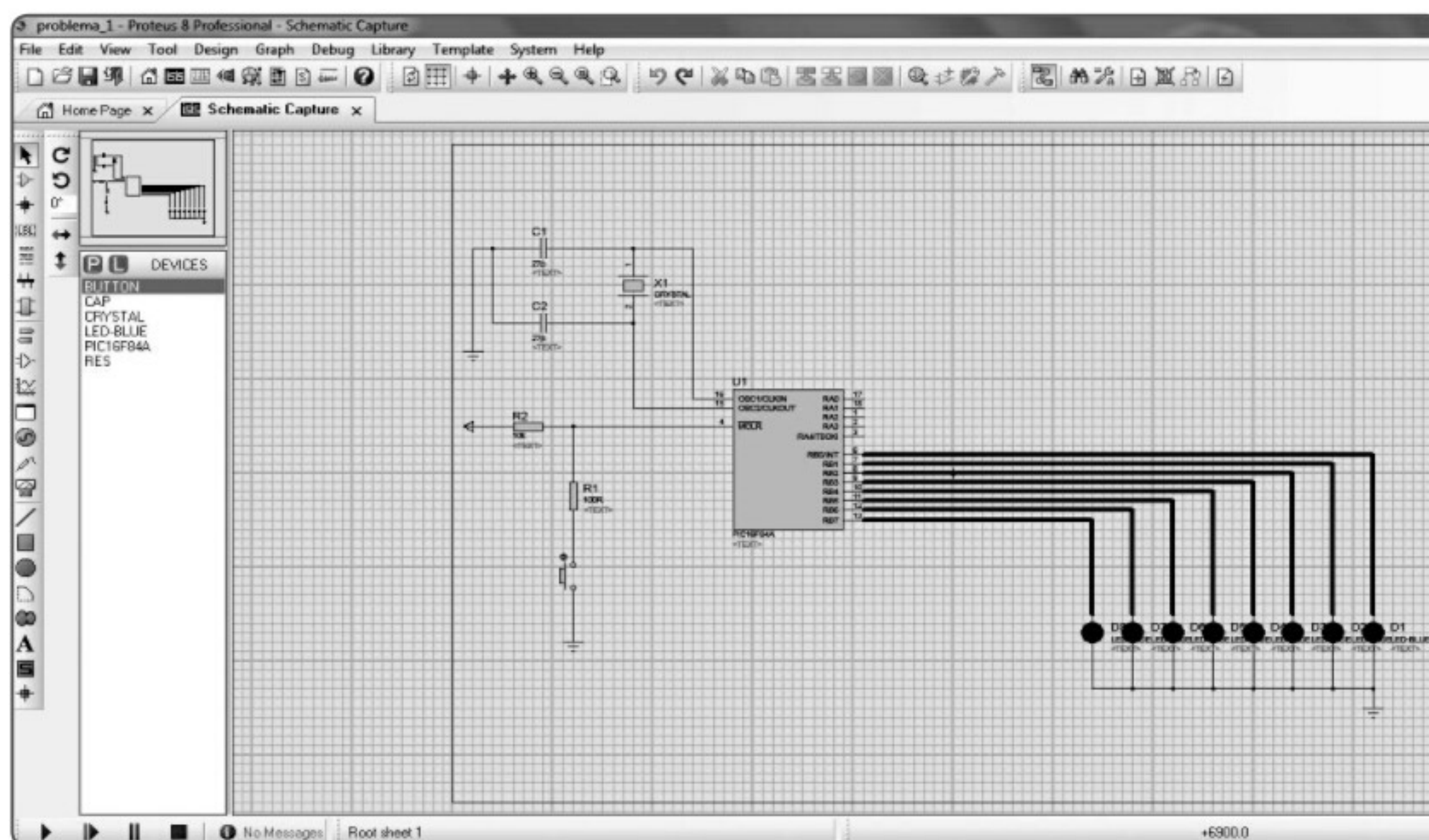


Figura 4.24 Ledes



**Ejercicio 5**

Realizar un programa que permita mostrar un contador en forma ascendente mediante un display, salida puerto B.

```

;-----

;|
LIST          p = 16F84          ;Usar el PIC16F84A-04/P
RADIX         HEX                ;Todo en hexadecimal
;|
;|          VARIABLES
w             equ    0            ;Cuando d=0 el destino es w
f             equ    1            ;Cuando d=1 el destino es f
PUERTO B      equ    0x06         ;Dirección del Puerto B
PUERTO A      equ    0x05
ESTADO        equ    0x03         ;Dirección del Estado
PCL           equ    0x02         ;Dirección de PCL
Aux1          equ    0x0C         ;Direcciones de ocupados para
Aux2          equ    0x0D         ;la subrutina de retardo
Aux3          equ    0x0E
cuenta        equ    0x20         ;Dirección del registro que lleva el conteo
ORG           0x00               ;Dirección de inicio
BSF           ESTADO,5           ;Pasarse al Banco 1
CLRF          PUERTO B          ;Establecer el Puerto B como de salida
CLRF          PUERTO A
BCF           ESTADO,5           ;Volver al banco 0
;|
PROGRAMA PRINCIPAL
BSF           PUERTO A,0         ;Activar el display 1
Ciclo1        CLRF    cuenta      ;Inicializar el contador
Ciclo2        MOVF    cuenta,w     ;Pasar a W el contenido de con-
tador

CALL          Tabla              ;Llamar a la tabla dependiendo de W
MOVWF         PUERTO B           ;Mandar al Puerto B el valor obtenido
CALL          Retardo            ;Llamar la subrutina de retardo
INCF          cuenta,f           ;Incrementar al contador
MOVLW        0x0A                ;Mover b'1010' a W
XORWF         cuenta,w           ;Hacer ope XOR cuenta con W
BTFSS        ESTADO,2           ;El contador es igual a 10?
GOTO          Ciclo2            ;No, seguir con el conteo
GOTO          Ciclo1            ;Si, ir a inicilizar el contador

Retardo
MOVLW        0x0D                ;14
MOVWF         Aux1
Uno
MOVLW        0x48                ;72
MOVWF         Aux2
Dos
MOVLW        0x7A                ;0xF7
MOVWF         Aux3

```



```

Tres      CLRWDT      ;Limpiar el reloj del Perro guardian
          DECFSZ Aux3,f ;Decrementar Aux3 -> Aux3-1
          GOTO Tres
          DECFSZ Aux2,f ;Decrementar Aux2 -> Aux2-1
          GOTO Dos
          DECFSZ Aux1,f ;Decrementar Aux1 -> Aux1-1
          GOTO Uno
          GOTO Sig

Sig        CLRWDT      ;Limpiar el reloj del Perro guardian
          RETURN

;|
          TABLA DE LOS DÍGITOS (0-9)
          ;B'gfedcba'

Tabla      ADDWF PCL,f
          RETLW B'00111111' ; 0
          RETLW B'00000110' ; 1
          RETLW B'01011011' ; 2
          RETLW B'01001111' ; 3
          RETLW B'01100110' ; 4
          RETLW B'01101101' ; 5
          RETLW B'01111100' ; 6
          RETLW B'00000111' ; 7
          RETLW B'01111111' ; 8
          RETLW B'01100111' ; 9
          END
          ;Fin del programa

```

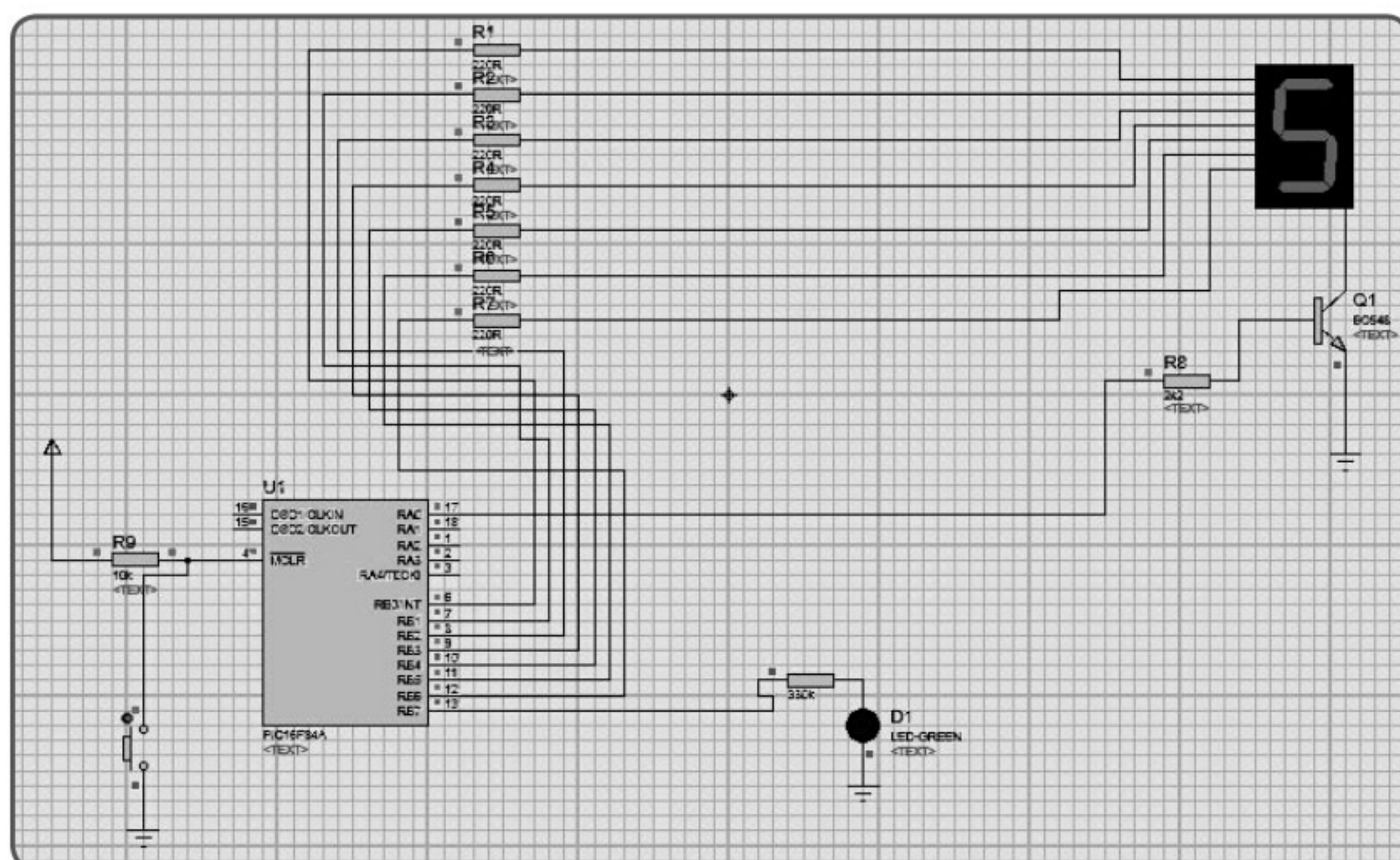


Figura 4.25 Pantalla digital



## Ejercicio 6

Proponer un circuito que muestre en un display los números primos, puerto B como salida.

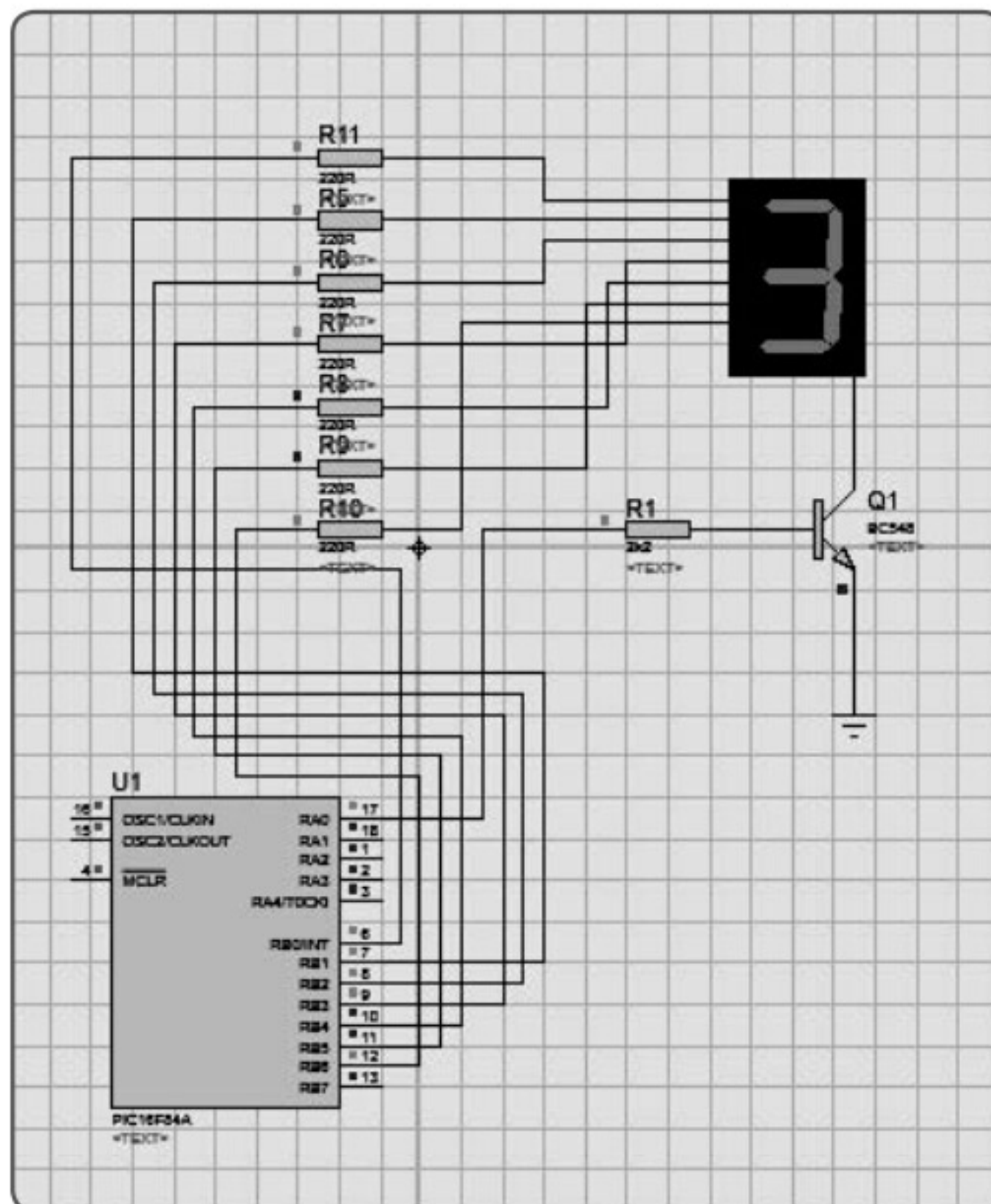


Figura 4.26 Números primos

```

;+-----+
;|  ENCABEZADO      |
;+-----+
+
LIST p = 16F84      ;Usar el PIC16F84A-04/P
RADIX      HEX      ;Todo en hexadecimal
;+-----+
;|  VARIABLES        |
;+-----+
w    equ    0        ;Cuando d=0 el destino es w
f    equ    1        ;Cuando d=1 el destino es f
PUERTO B    equ    0x06 ;Dirección del Puerto B
PUERTO A    equ    0x05
ESTADO      equ    0x03 ;Dirección del Estado
PCL    equ    0x02    ;Dirección de PCL
Aux1    equ    0x0C    ;Direcciones de ocupados para

```



```

Aux2 equ    0x0D                ;la subrutina de retardo
Aux3 equ    0x0E
cuenta      equ    0x20          ;Dirección del registro que lleva el conteo
;+-----+
;|          CONFIGURACIÓN DEL PUERTO B COMO SALIDA          |
;+-----+

        ORG        0x00          ;Dirección de inicio
        BSF        ESTADO,5      ;Pasarse al Banco 1
        CLRF       PUERTOB       ;Establecer el Puerto B como de salida
        CLRF       PUERTOA
        BCF        ESTADO,5      ;Volver al banco 0

;+-----+
;|          PROGRAMA PRINCIPAL                              |
;+-----+

        BSF        PUERTOA,0      ;Activar el display 1

Ciclo1          CLRF    cuenta      ;Inicializar el contador
Ciclo2          MOVF    cuenta,w     ;Pasar a W el contenido de contador
                CALL    Tabla        ;Llamar a la tabla dependiendo de W
                MOVWF   PUERTOB      ;Mandar al Puerto B el valor obtenido
                CALL    Retardo      ;Llamar la subrutina de retardo
                INCF    cuenta,f     ;Incrementar al contador
                MOVLW   0x0A         ;Mover b'1010' a W
                XORWF   cuenta,w     ;Hacer ope XOR cuenta con W
                BTFSS   ESTADO,2     ;¿El contador es igual a 10?
                GOTO    Ciclo2       ;No, seguir con el conteo
                GOTO    Ciclo1       ;Si, ir a inicializar el contador

;+-----+
;|          RUTINA DE RETARDO                              |
;+-----+

Retardo        MOVLW   0x0D          ;14
                MOVWF   Aux1
Uno            MOVLW   0x48          ;72
                MOVWF   Aux2
Dos           MOVLW   0x7A          ;0xF7          ;247
                MOVWF   Aux3
Tres          CLRWDT                ;Limpiar el reloj del Perro guardian
                DECFSZ  Aux3,f       ;Decrementar Aux3 -> Aux3-1
                GOTO    Tres
                DECFSZ  Aux2,f       ;Decrementar Aux2 -> Aux2-1
                GOTO    Dos
                DECFSZ  Aux1,f       ;Decrementar Aux1 -> Aux1-1
                GOTO    Uno
                GOTO    Sig

```



```

Sig CLRWDT                ;Limpiar el reloj del Perro guardian
RETURN

;+-----+
;|          TABLA DE LOS NUMEROS PRIMOS          |
;+-----+
;B'gfedcba'
Tabla ADDWF PCL,f
RETLW B'01011011'        ; 2
RETLW B'01001111'        ; 3
RETLW B'01101101'        ; 5
RETLW B'00000111'        ; 7

END                        ;Fin del programa

```

## Ejercicio 7

Proponer un circuito en el que se muestren las letras de la palabra HOLA.

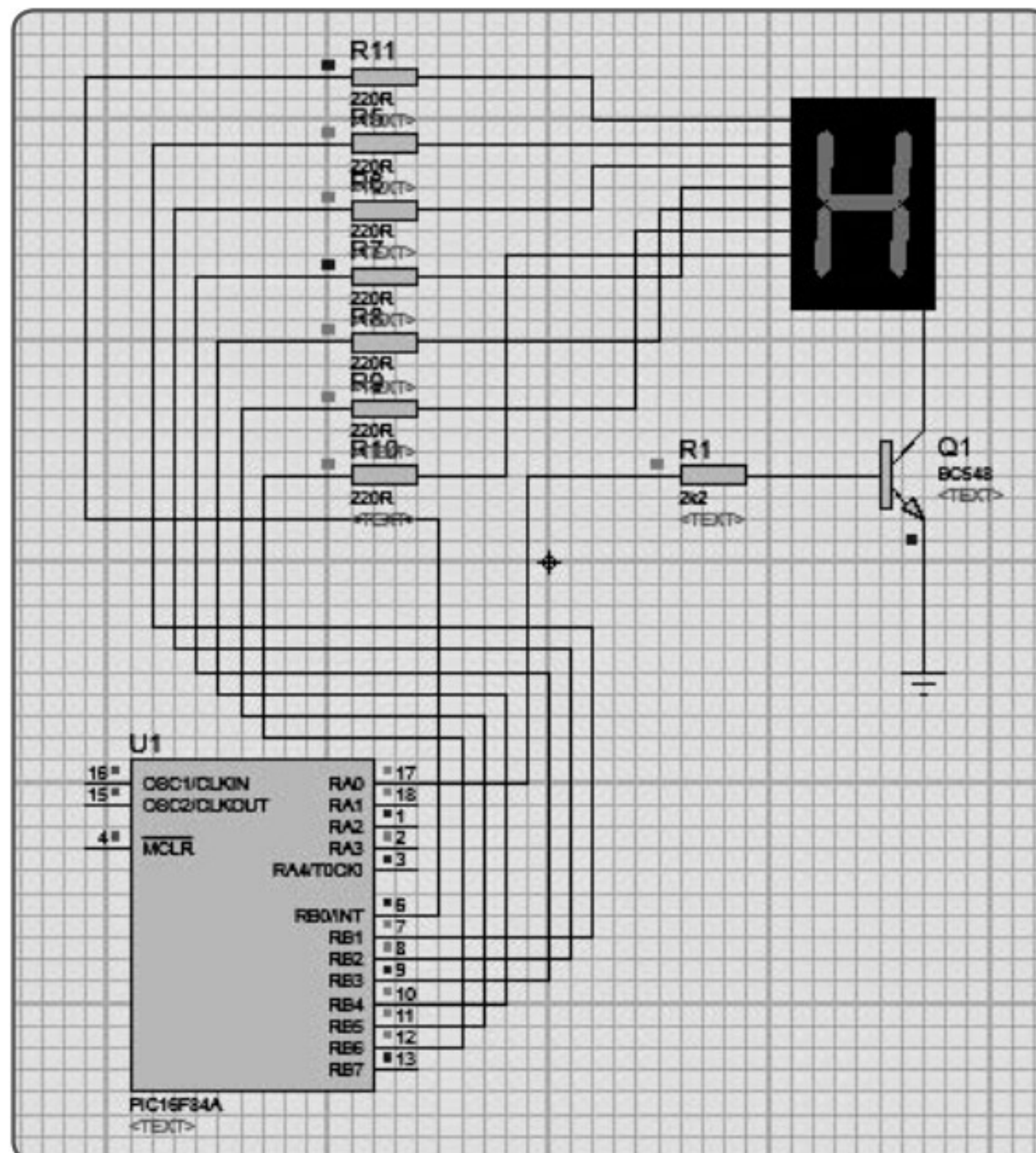


Figura 4.27 Circuito HOLA



```

;+-----+
;|          ENCABEZADO          |
;+-----+
LIST p = 16F84          ;Usar el PIC16F84A-04/P
RADIX      HEX          ;Todo en hexadecimal
;+-----+
;|          VARIABLES          |
;+-----+
w          equ          0          ;Cuando d=0 el destino es w
f          equ          1          ;Cuando d=1 el destino es f
PUERTOB    equ          0x06       ;Dirección del Puerto B
PUERTOA    equ          0x05
ESTADO     equ          0x03       ;Dirección del Estado
PCL        equ          0x02       ;Dirección de PCL
Aux1       equ          0x0C       ;Direcciones de ocupados para
Aux2       equ          0x0D       ;la subrutina de retardo
Aux3       equ          0x0E
cuenta     equ          0x20       ;Dirección del registro que lleva el
conteo
;+-----+
;|  CONFIGURACIÓN DEL PUERTO B COMO SALIDA  |
;+-----+
          ORG          0x00          ;Dirección de inicio
          BSF          ESTADO,5      ;Pasarse al Banco 1
          CLRF         PUERTOB       ;Establecer el Puerto B como de salida
          CLRF         PUERTOA
          BCF          ESTADO,5      ;Volver al banco 0
;+-----+
;|          PROGRAMA PRINCIPAL          |
;+-----+
          BSF          PUERTOA,0     ;Activar el display 1

Ciclo1     CLRF         cuenta        ;Inicializar el contador
Ciclo2     MOVF         cuenta,w      ;Pasar a W el contenido de con-
tador
          CALL         Tabla          ;Llamar a la tabla dependiendo de W
          MOVWF        PUERTOB       ;Mandar al Puerto B el valor obtenido
          CALL         Retardo        ;Llamar la subrutina de retardo
          INCF         cuenta,f      ;Incrementar al contador
          MOVLW        0x0A          ;Mover b'1010' a W
          XORWF        cuenta,w      ;Hacer ope XOR cuenta con W
          BTFSS        ESTADO,2      ;¿El contador es igual a 10?
          GOTO         Ciclo2        ;No, seguir con el conteo
          GOTO         Ciclo1       ;Si, ir a inicilizar el contador
;+-----+
;|          RUTINA DE RETARDO          |
;+-----+
Retardo    MOVLW        0x0D          ;14
          MOVWF        Aux1

```



```

Uno      MOVLW    0x48          ;72
          MOVWF    Aux2
Dos      MOVLW    0x7A          ;0xF7      ;247
          MOVWF    Aux3
Tres     CLRWDT                ;Limpiar el reloj del Perro guardian
          DECFSZ   Aux3,f      ;Decrementar Aux3 -> Aux3-1
          GOTO     Tres
          DECFSZ   Aux2,f      ;Decrementar Aux2 -> Aux2-1
          GOTO     Dos
          DECFSZ   Aux1,f      ;Decrementar Aux1 -> Aux1-1
          GOTO     Uno
          GOTO     Sig
Sig      CLRWDT                ;Limpiar el reloj del Perro guardian
          RETURN

;+-----+
;|          TABLA DE LAS LETRAS(H-O-L-A)
;+-----+
;B'gfedcba'
Tabla    ADDWF    PCL,f
          RETLW    B'01110110' ; H
          RETLW    B'00111111' ; O
          RETLW    B'00111000' ; L
          RETLW    B'01110111' ; A

          END                                     ;Fin del programa

```

## Ejercicio 8

Proponer un circuito que vaya de izquierda a derecha y viceversa.

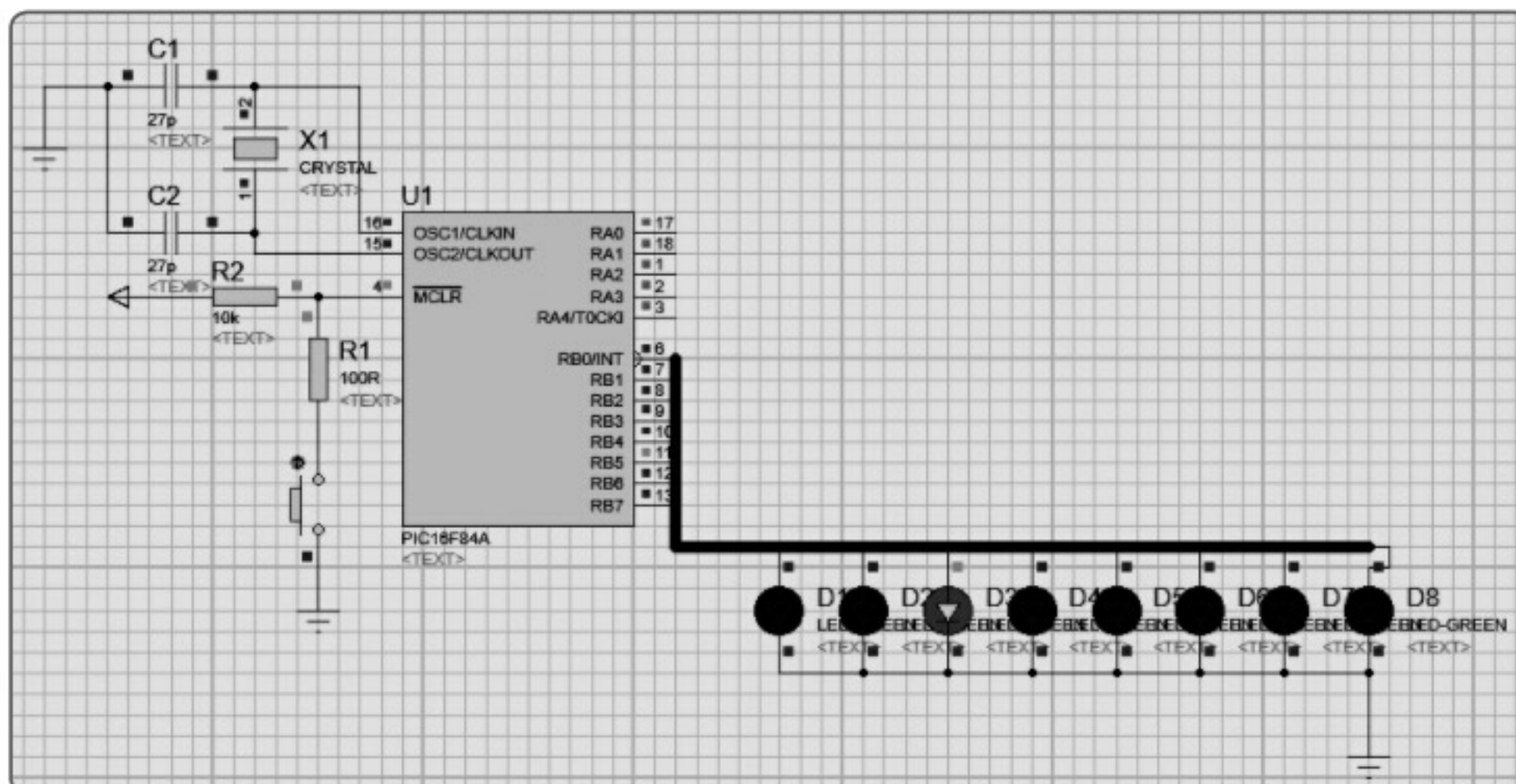


Figura 4.28 Circuito led ida y vuelta



```

;+-----+
;|          ENCABEZADO          |
;+-----+
;          LIST          p = 16F84          ;Usar el PIC16F84A-04/P
;          RADIX          HEX          ;Todo en hexadecimal
;+-----+
;|          VARIABLES          |
;+-----+
w          equ          0          ;Cuando d=0 el destino es w
f          equ          1          ;Cuando d=1 el destino es f
PUERTOB    equ          0x06          ;Dirección del Puerto B
ESTADO     equ          0x03          ;Dirección del Estado
PCL        equ          0x02          ;Dirección de PCL
Aux1       equ          0x0C          ;Direcciones de ocupados para
Aux2       equ          0x0D          ;la subrutina de retardo
Aux3       equ          0x0E
cuenta     equ          0x20          ;Dirección del registro que lleva
el conteo
;+-----+
;|  CONFIGURACIÓN DEL PUERTO B COMO SALIDA  |
;+-----+
ORG        0x00          ;Dirección de inicio

BSF        ESTADO,5      ;Pasarse al Banco 1
CLRWF     PUERTOB        ;Establecer el Puerto B como de salida
BCF        ESTADO,5      ;Volver al banco 0

;+-----+
;|          PROGRAMA PRINCIPAL          |
;+-----+
Ciclo1     CLRWF          cuenta          ;Inicializar el conta-
dor
Ciclo2     MOVF           cuenta,w        ;Pasar a W el contenido de con-
tador
MOVWF      CALL          Tabla            ;Llamar a la tabla dependiendo de W
PUERTOB    ;Mandar al Puerto B el valor obtenido
CALL       Retardo       ;Llamar la subrutina de retardo
INCF       cuenta,f      ;Incrementar al conta-
dor
MOVLW     0x10            ;Mover b'00010000' a W 16 decimal
XORWF     cuenta,w        ;Hacer ope XOR cuenta con W
BTFSS     ESTADO,2        ;¿El contador es igual a 16?
GOTO      Ciclo2          ;No, seguir con el cciclo
GOTO      Ciclo1          ;Si, ir a inicilizar el ciclo

;+-----+
;|          RUTINA DE RETARDO          |
;+-----+
Retardo    MOVLW          0x0D          ;14
MOVWF     Aux1

```



```

Uno      MOVLW  0x48          ;72
          MOVWF  Aux2
Dos      MOVLW  0x7A          ;0xF7      ;247
          MOVWF  Aux3
Tres     CLRWD  T            ;Limpiar el reloj del Perro guardian
          DECFSZ Aux3,f      ;Decrementar Aux3 -> Aux3-1
          GOTO   Tres
          DECFSZ Aux2,f      ;Decrementar Aux2 -> Aux2-1
          GOTO   Dos
          DECFSZ Aux1,f      ;Decrementar Aux1 -> Aux1-1
          GOTO   Uno
          GOTO   Sig
Sig      CLRWD  T            ;Limpiar el reloj del Perro guardian
          RETURN

```

```

;+-----+
;|          TABLA DE LA SECUENCIA          |
;+-----+

```

```

Tabla    ADDWF  PCL,f
          RETLW  B'10000000'    ; 0x80
          RETLW  B'01000000'    ; 0xC0
          RETLW  B'00100000'    ; 0xE0
          RETLW  B'00010000'    ; 0xF0
          RETLW  B'00001000'    ; 0xF8
          RETLW  B'00000100'    ; 0xFC
          RETLW  B'00000010'    ; 0xFE
          RETLW  B'00000001'    ; 0xFF
          RETLW  B'0000a0010'   ; 0x7F
          RETLW  B'00000100'    ; 0x3F
          RETLW  B'00001000'    ; 0x1F
          RETLW  B'00010000'    ; 0x0F
          RETLW  B'00100000'    ; 0x07
          RETLW  B'01000000'    ; 0x03
          RETLW  B'10000000'    ; 0x01
          RETLW  B'00000000'    ; 0x00

```

```

END                                     ;Fin del programa

```

## Ejercicio 9

Realizar un programa que permita intercambio entre 0Eh y 0Fh datos (6F y F6).

```

LIST      P=16F84A                ;comando que indica el pic usado.
RADIX     HEX                    ;los valores de hexadecimal
STATUS    EQU      0X03          ;direcciona al registro de STATUS
AUX        EQU      0X0D          ;direccion de memoria 0DH
DIR1       EQU      0X0E          ;direccion de memoria 0EH
DIR2       EQU      0X0F          ;direccion de memoria 0FH
ORG        0X00
CLRF      DIR1                  ;limpia la direccion 0EH
CLRF      DIR2                  ;limpia la direccion 0FH
CLRF      AUX                   ;limpia la direccion 0DH
MOVLW     0X6F                  ;cargar el reg w con el num 3Fh
MOVWF     DIR1                  ;almacenar el reg w en la dir 0Eh
MOVWF     AUX                   ;almacenar reg w en aux 0Dh
MOVLW     0XF6                  ;cargar reg w con el numero F3h
MOVWF     DIR2                  ;almacenar el reg w en la dir 0Fh
MOVWF     DIR1                  ;almacenar el reg w en la dir 0Eh
MOVWF     AUX                   ;cargar reg w con dato de dir 0Dh
MOVWF     DIR2                  ;almacenar reg w en la dir 0Fh
END                          ;fin del programa

```

## Ejercicio 10

Realizar un programa que permita hacer la operación suma con el dato 05h PTOA (entrada) PTOB (salida). porta=porta + 05h.

```

;PROGRAMA 11
LIST      P=16F84A
RADIX     HEX
STATUS    EQU      0X03
PTOA      EQU      0X05
PTOB      EQU      0X06
ORG        0X00
BSF       STATUS,5
MOVLW     0X1F
MOVWF     PTOA
MOVLW     0X00
MOVWF     PTOB

```



```

BCF      STATUS,5
CLRF     PTOA
CLRF     PTOB

CICLO          MOVLW    0X05
                ADDWF   PTOA,0
                MOVWF   PTOB
                GOTO    CICLO
                END

; FIN DEL PROGRAMA

```

### Ejercicio 11

Proponer un circuito que tenga el puerto B como salida conectando un motor a pasos en sentido del reloj. Que inicie la rotación cuando se presione el bit 0 del puerto A, y si se presiona cuando está rotando, este deberá parar, es decir, el bit será de arranque y paro.

```

; PROGRAMA A0C
LIST P = 16F84

RADIX      HEX

W    EQU    0
F    EQU    1

PUERTOA    EQU    0X05
PUERTOB    EQU    0X06

ESTADO     EQU    0X03
PCL        EQU    0X02
AUX1       EQU    0X0D
AUX2       EQU    0X0E
NPASO      EQU    0X20
ORG 0X00
BSF  ESTADO,5

MOVLW      0X00
MOVWF      PUERTOB
MOVLW      0X0F
MOVWF      PUERTOA
BCF  ESTADO,5

;INICIO

```

```

INICIO      CLRf    NPASO                ;BORRAR CONTENIDO DE NPASO
TEST        BTFSS  PUERTOA,0            ;EL BIT 0 DEL PUERTO A ES 1?
GOTO INICIO ;NO, IR A INICIO
GOTO        GIRO      ;SI, IR A GIRO
                                ;FIN

;INICIO
GIRO MOVf    NPASO,W                ;PASAR A W EL CONTENIDO DE CONTADOR
CALL        TABLAD                ;LLAMAR LA TABLA DE PASOS
MOVWF       PUERTOB                ;MANDAR AL PUERTO B EL VALOR OBTENIDO
CALL RETARDO                ;LLAMAR SUBROUTINA DE RETARDO
                                ;INCREMENTAR, NPASO = NPASO + 1
                                ;MOVER B'00000100' A W
                                ;HACER OPE XOR CUENTA CON W
                                ;¿EL CONTADOR ES IGUAL A 4?
                                ;NO,VA A CICLO
                                ;SI, REGRESA AL TESTEO
                                ;FIN

;+-----+
;|                RUTINA DE RETARDO                |
;+-----+

RETARDO     MOVLW   .33                ; 1 SET NUMBER OF REPETITIONS (B)
MOVWF       AUX1; 1 |
PLOOP1      MOVLW   .60                ; 1 SET NUMBER OF REPETITIONS (A)
MOVWF       AUX2; 1 |
PLOOP2      CLRWDT                    ; 1 CLEAR WATCHDOG
CLRWDT      ; 1 CYCLE DELAY
DECFSZ      AUX2,1                    ; 1 + (1) IS THE TIME OVER? (A)
            GOTO    PLOOP2            ; 2 NO, LOOP
DECFSZ      AUX1,1                    ; 1 + (1) IS THE TIME OVER? (B)
            GOTO    PLOOP1            ; 2 NO, LOOP
CLRWDT      ; 1 CYCLE DELAY
RETURN      ; 2+2 DONE

;FIN

TABLAD      ADDWF   PCL,F
            RETLW   B'00001001'      ;09
            RETLW   B'00000011'      ;03
            RETLW   B'00000110'      ;06
            RETLW   B'00001100'      ;0C
            END
;FIN DEL PROGRAMA

```



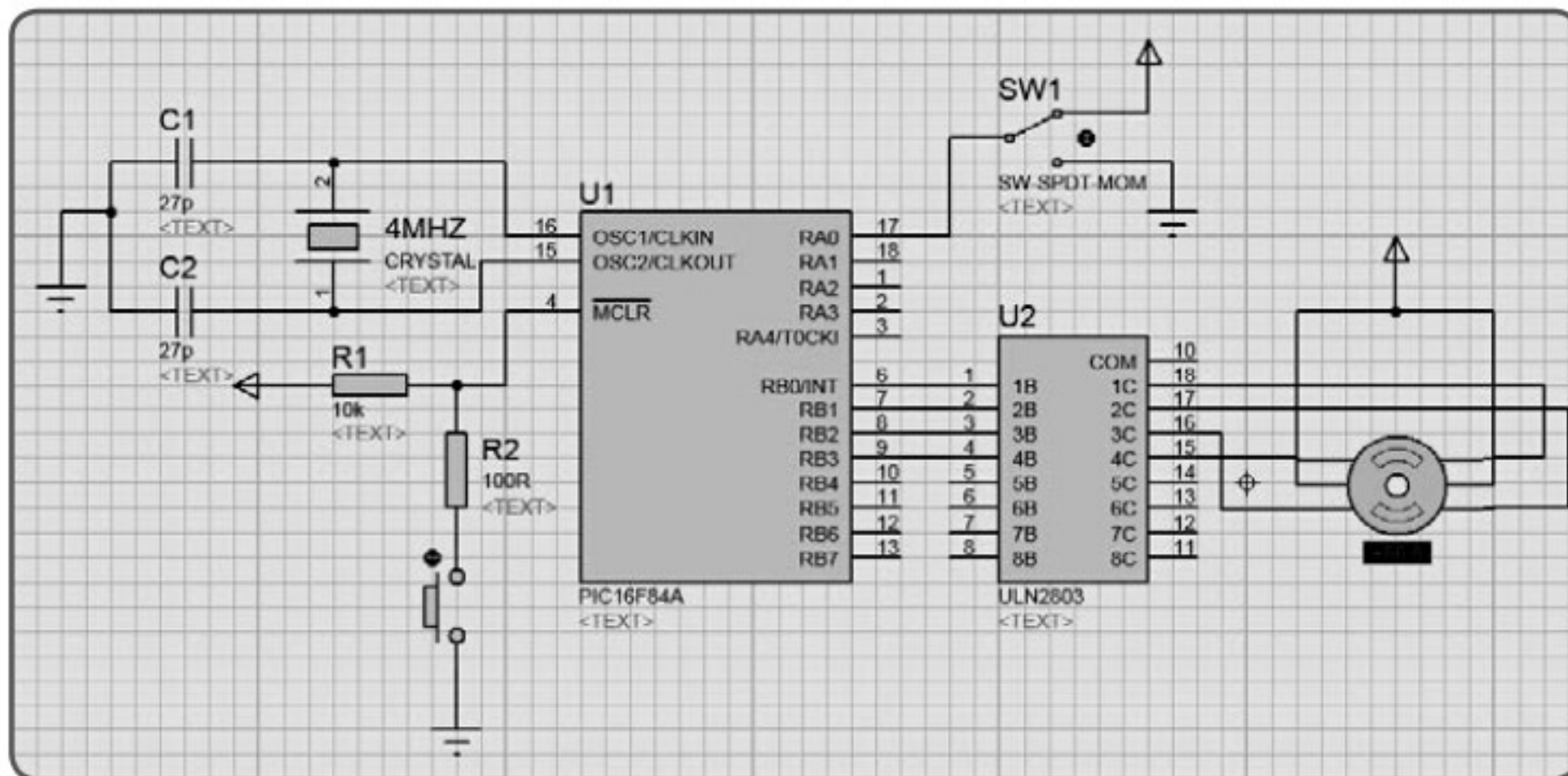


Figura 4.29 Circuito reloj

## Ejercicio 12

Proponer un circuito de incremento y decremento (bit 0 y bit 1 respectivamente); PTOB salida conectando un motor a pasos, PTOA entrada.

```

; PROGRAMA A0E
LIST P = 16F84          ;usar el pic16f84a-04/p
RADIX      HEX          ;todo en hexadecimal

W    EQU    0           ;cuando d=0 el destino es w
F    EQU    1           ;cuando d=1 el destino es f
PUERTOA    EQU    0X05   ;dirección del puerto a
PUERTOB     EQU    0X06   ;dirección del puerto b
ESTADO      EQU    0X03   ;dirección del estado
PCL    EQU    0X02       ;dirección de pcl
AUX1    EQU    0X0D
AUX2    EQU    0X0E
NPASO    EQU    0X20     ;dirección del reg que lleva el conteo de los
pasos

ORG 0X00               ;dirección de inicio
BSF ESTADO,5           ;pasarse al banco 1
MOVLW     0X00
MOVWF     PUERTOB       ;establecer el puerto b como de salida
MOVLW     0X0F
MOVWF     PUERTOA       ;puerto a como de entrada ra0-ra3

```



```

BCF  ESTADO,5      ;volver al banco 0

;INICIO

INICIO  CLRF      NPASO      ;borrar contenido de npaso
TEST    BTFSS     PUERTO A,0 ;el bit 0 del puerto a es 1?
        GOTO      INICIO     ;no, ir a inicio
        BTFSC     PUERTO A,1 ;si, el bit 1 del puerto a es 0?
        GOTO      VELMAX     ;no, gira a vel max
        BTFSC     PUERTO A,2 ;si, el bit 2 del puerto a es 0?
        GOTO      VELMIN     ;no, gira a vel min
        GOTO      INICIO     ;si, vuelve al testeo

;FIN

;INICIO
VELMAX  MOVF      NPASO,W    ;pasar a w el contenido de contador
        CALL      TABLAD     ;llamar la tabla de pasos
        MOVWF     PUERTO B   ;mandar al puerto b el valor obtenido
        CALL      RETARDO1   ;llamar la subrutina de retardo
        INCF      NPASO,F    ;incrementar npaso = npaso + 1
        MOVLW     0X04       ;mover b'00000100' a w
        XORWF     NPASO,W    ;hacer ope xor cuenta con w
        BTFSS     ESTADO,2   ;¿el contador es igual a 4?
        GOTO      TEST      ;no, va a ciclo
        GOTO      INICIO     ;si, regresa al testeo

;fin

;INICIO
VELMIN  MOVF      NPASO,W    ;pasar a w el contenido de contador
        CALL      TABLAD     ;llamar la tabla de pasos
        MOVWF     PUERTO B   ;mandar al puerto b el valor obtenido
        CALL      RETARDO    ;llamar la subrutina de retardo1
        INCF      NPASO,F    ;incrementar npaso = npaso + 1
        MOVLW     0X04       ;mover b'00000100' a w
        XORWF     NPASO,W    ;hacer ope xor cuenta con w
        BTFSS     ESTADO,2   ;¿el contador es igual a 4?
        GOTO      TEST      ;no, va a ciclo
        GOTO      INICIO     ;si, regresa al testeo

;FIN

RETARDO  MOVLW     .33        ; 1 set number of repetitions (b)
        MOVWF     AUX1       ; 1 |
PLOOP1   MOVLW     .60        ; 1 set number of repetitions (A)
        MOVWF     AUX2       ; 1 |
PLOOP2   CLRWD     CLRWD     ; 1 clear watchdog
        CLRWD     CLRWD     ; 1 cycle delay
        DECFSZ    AUX2,1     ; 1 + (1) is the time over? (a)
        GOTO      PLOOP2    ; 2 no, loop

```



```

    DECFSZ AUX1,1          ; 1 + (1) is the time over? (b)
    GOTO     PLOOP1        ; 2 no, loop
    CLRWDT              ; 1 cycle delay
    RETURN              ; 2+2 done
;FIN

RETARDO1 MOVLW    .15      ; 1 set number of repetitions (b)
        MOVWF AUX1        ; 1
PLOOP1   MOVLW    .30      ; 1 set number of repetitions (a)
        MOVWF AUX2        ; 1
PLOOP2   CLRWDT          ; 1 clear watchdog
        CLRWDT            ; 1 cycle delay
        DECFSZ AUX2,1      ; 1 + (1) is the time over? (a)
        GOTO     PLOOP2    ; 2 no, loop
        DECFSZ AUX1,1      ; 1 + (1) is the time over? (b)
        GOTO     PLOOP1    ; 2 no, loop
        CLRWDT            ; 1 cycle delay
        RETURN            ; 2+2 done
;FIN

TABLAD   ADDWF    PCL,F
        RETLW    B'00001001' ;09
        RETLW    B'00000011' ;03
        RETLW    B'00000110' ;06
        RETLW    B'00001100' ;0C
END

;FIN DEL PROGRAMA

```

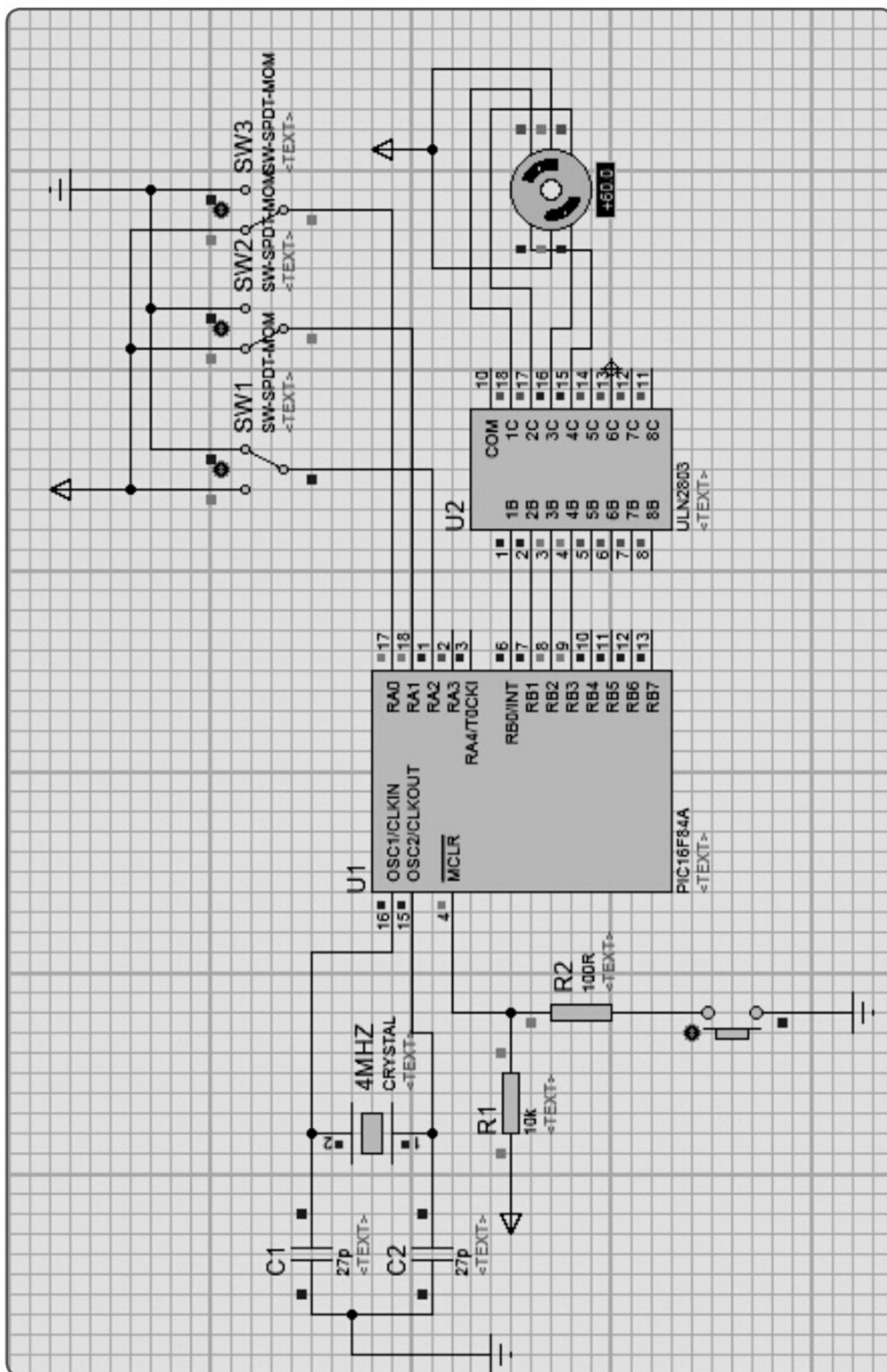
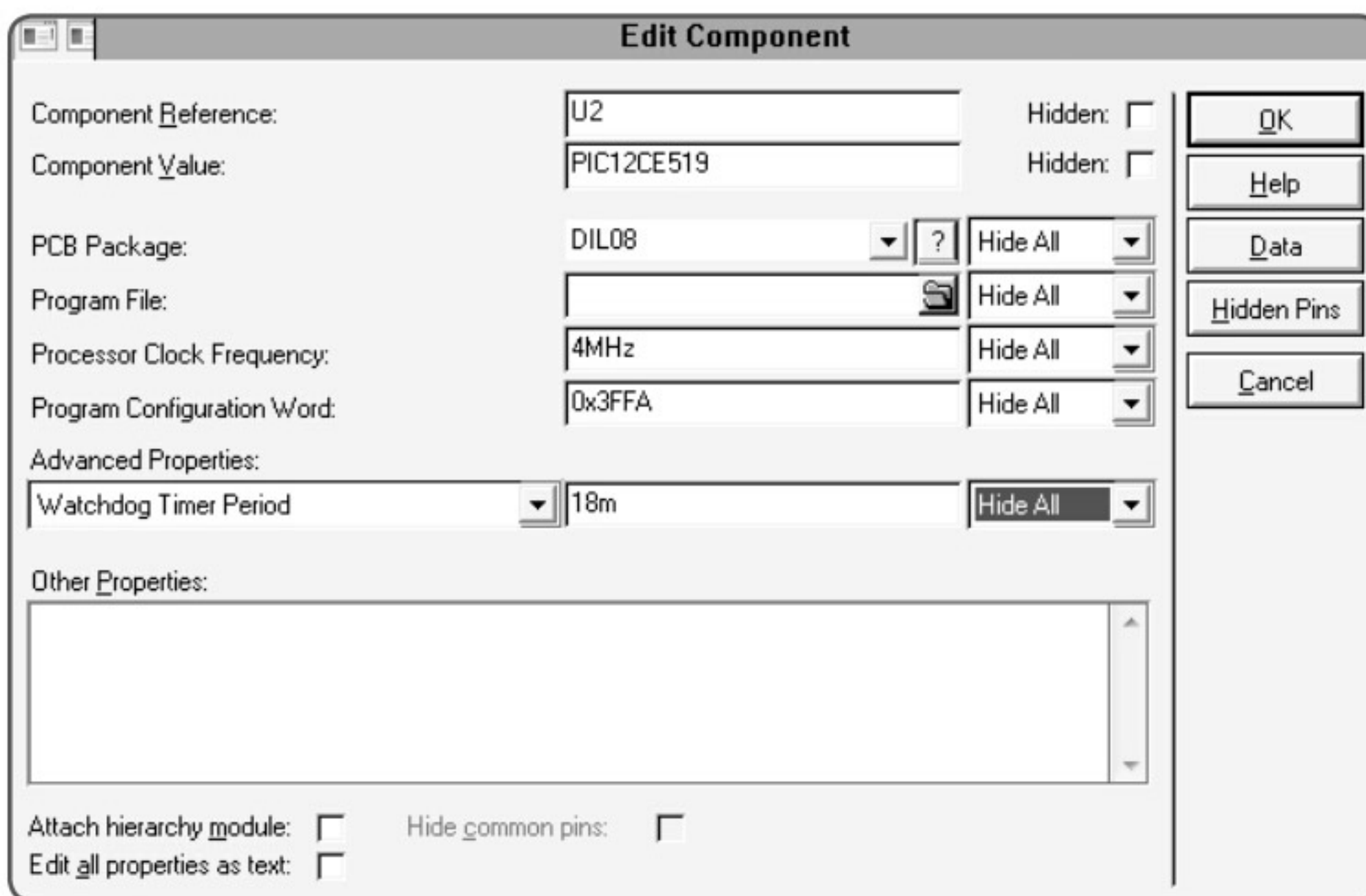


Figura 4.30 Incremento y decremento





**Edit Component**

Component Reference: U2 Hidden: ☐

Component Value: PIC12CE519 Hidden: ☐

PCB Package: DIL08 ? Hide All

Program File: Hide All

Processor Clock Frequency: 4MHz Hide All

Program Configuration Word: 0x3FFA Hide All

Advanced Properties:

Watchdog Timer Period 18m Hide All

Other Properties:

Attach hierarchy module: ☐ Hide common pins: ☐

Edit all properties as text: ☐

Buttons: OK, Help, Data, Hidden Pins, Cancel





# capítulo 5

## EJERCICIOS DESARROLLADOS



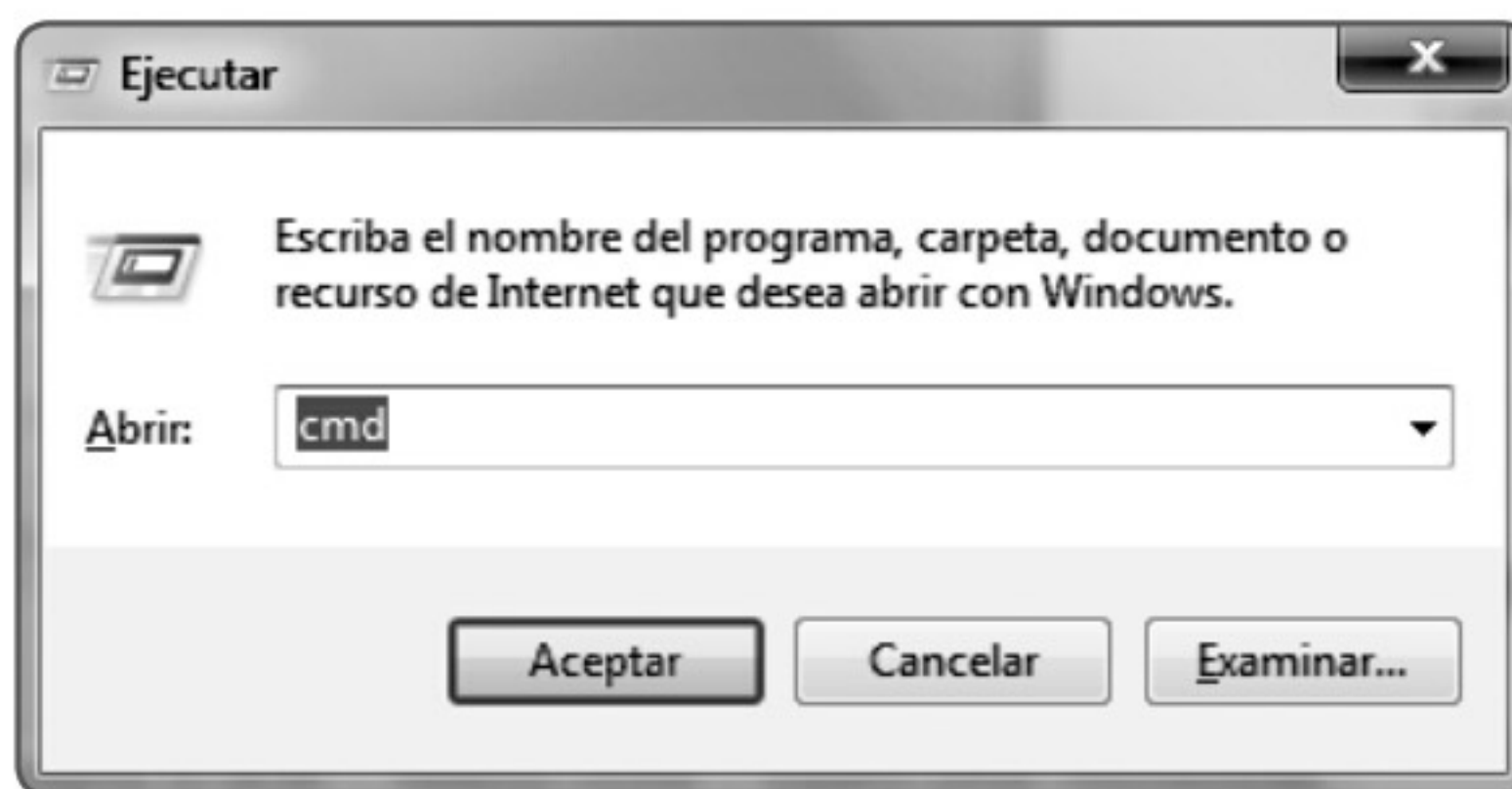




**Ejercicio 1**

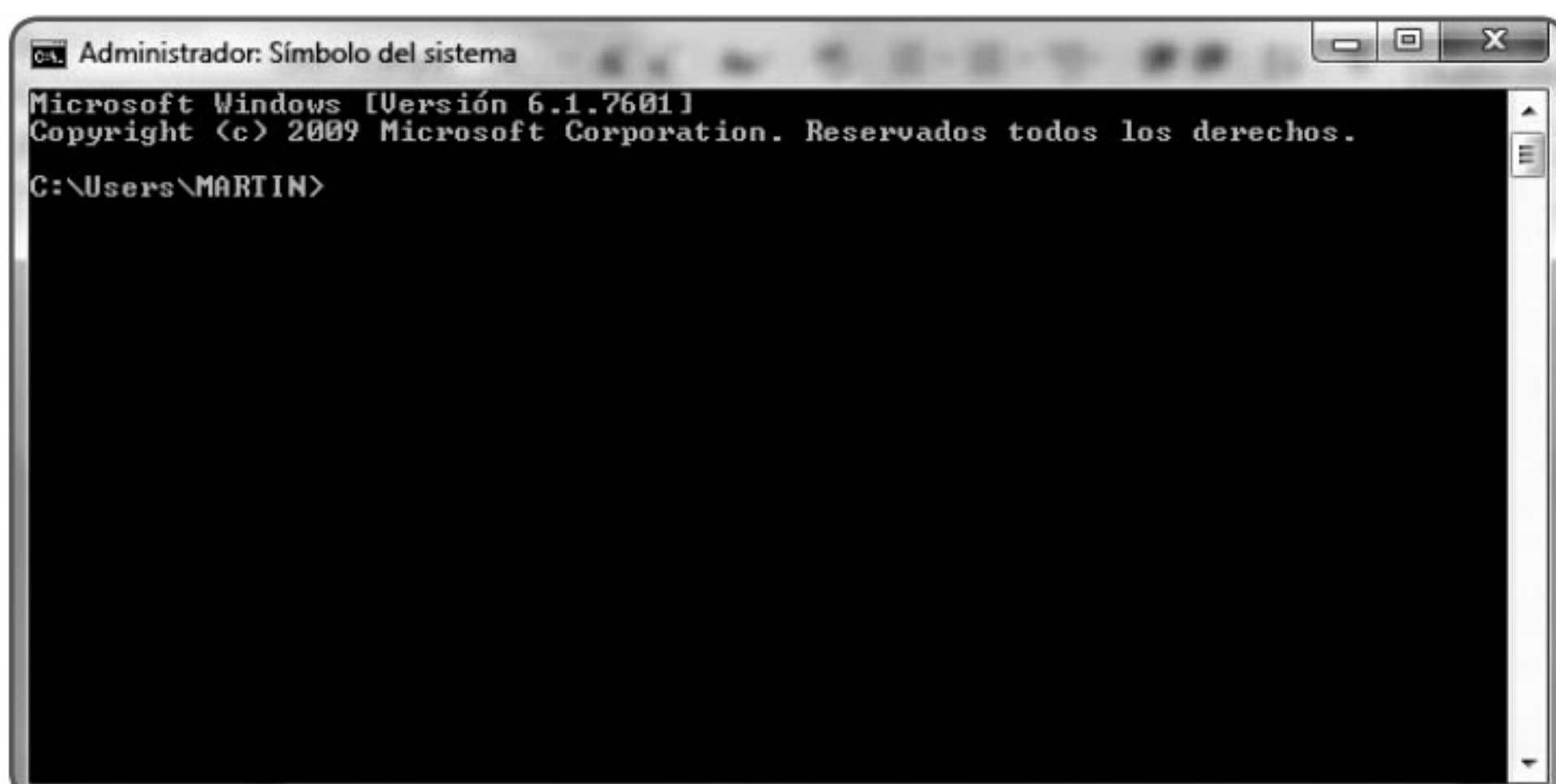
Realizar un programa que permita la suma de 3 números hexadecimales y guardarlo en el registro AX del depurador de Windows (debug). Ejemplo: AX=1+2+3, resultado: AX=6.

1. Primero, abrir el depurador de Windows (debug) de la siguiente manera:
  - a. Abra la ventana **Ejecutar** con  + R y escriba **cmd** para abrir el símbolo del sistema.



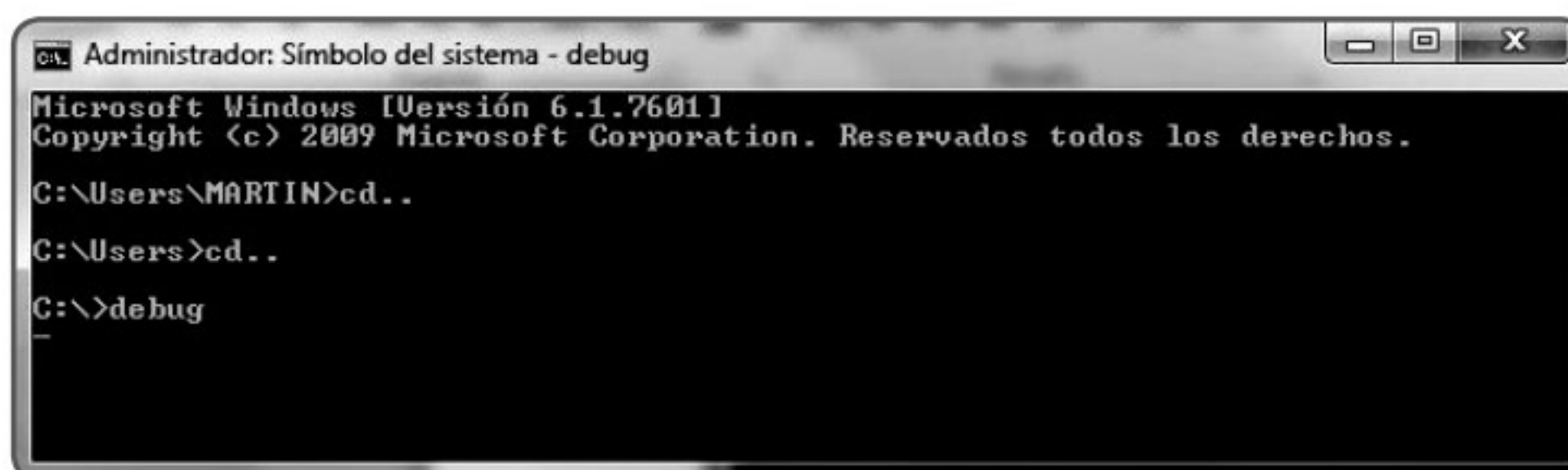
**Figura 5.1** Ventana Ejecutar

- b. Use **cd..** las veces necesarias para llegar a la raíz C:



**Figura 5.2** cd..

2. Nuevamente, busque la raíz del programa. Utilice la instrucción **cd..**, repitiendo el comando cuantas veces sea necesario para llegar al disco C (raíz del programa) y, luego, escriba la instrucción **debug**.



```
Administrador: Símbolo del sistema - debug
Microsoft Windows [Versión 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. Reservados todos los derechos.
C:\Users\MARTIN>cd..
C:\Users>cd..
C:\>debug
```

Figura 5.3 raíz

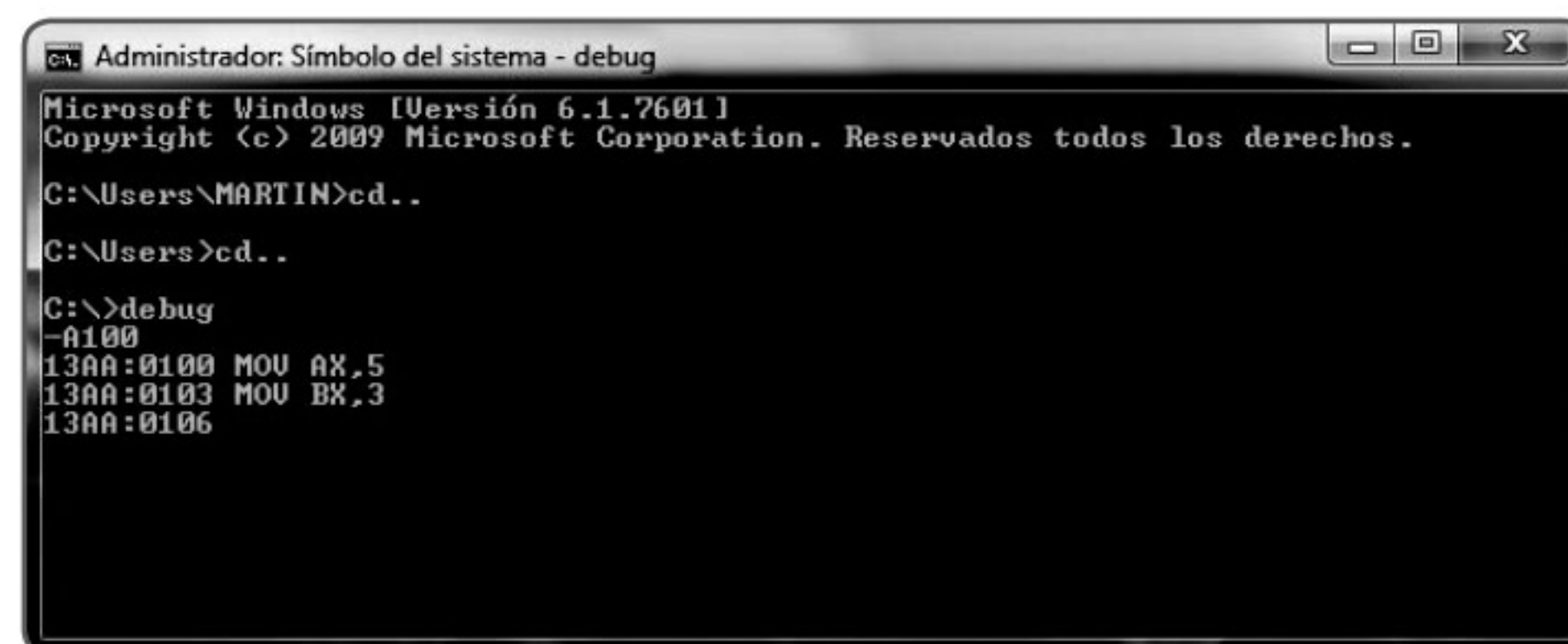
3. Asigne un direccionamiento de memoria con la instrucción **A100** y, luego, presione **Enter**.



```
Administrador: Símbolo del sistema - debug
Microsoft Windows [Versión 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. Reservados todos los derechos.
C:\Users\MARTIN>cd..
C:\Users>cd..
C:\>debug
```

Figura 5.4 Direccionamiento de memoria

4. Con la instrucción **MOV**, asigne un número para A y B.



```
Administrador: Símbolo del sistema - debug
Microsoft Windows [Versión 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. Reservados todos los derechos.
C:\Users\MARTIN>cd..
C:\Users>cd..
C:\>debug
-A100
13AA:0100 MOV AX,5
13AA:0103 MOV BX,3
13AA:0106
```

Figura 5.5 raíz



5. Con la instrucción **ADD**, sume A y B.

```

Administrador: Símbolo del sistema - debug
Microsoft Windows [Versión 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. Reservados todos los derechos.
C:\Users\MARTIN>cd..
C:\Users>cd..
C:\>debug
-A100
13AA:0100 MOV AX,5
13AA:0103 MOV BX,3
13AA:0106 ADD AX,BX
13AA:0108 _

```

Figura 5.6 suma\_ab

6. Para finalizar escriba la instrucción **INT 20**, luego, presione **Enter** dos veces.

```

Administrador: Símbolo del sistema - debug
Microsoft Windows [Versión 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. Reservados todos los derechos.
C:\Users\MARTIN>cd..
C:\Users>cd..
C:\>debug
-A100
13AA:0100 MOV AX,5
13AA:0103 MOV BX,3
13AA:0106 ADD AX,BX
13AA:0108 INT 20
13AA:010A _

```

Figura 5.7 int20

7. Para ver el programa línea por línea escriba la instrucción **t**.

```

Administrador: Símbolo del sistema - debug
C:\Users>cd..
C:\>debug
-A100
13AA:0100 MOV AX,5
13AA:0103 MOV BX,3
13AA:0106 ADD AX,BX
13AA:0108 INT 20
13AA:010A
-T
AX=0005 BX=0000 CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=13AA ES=13AA SS=13AA CS=13AA IP=0103 NU UP EI PL NZ NA PO NC
13AA:0103 BB0300 MOV BX,0003
-T
AX=0005 BX=0003 CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=13AA ES=13AA SS=13AA CS=13AA IP=0106 NU UP EI PL NZ NA PO NC
13AA:0106 01D8 ADD AX,BX
-T
AX=0008 BX=0003 CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=13AA ES=13AA SS=13AA CS=13AA IP=0108 NU UP EI PL NZ NA PO NC
13AA:0108 CD20 INT 20
_

```

Figura 5.8 mostrar\_resultado

## Ejercicio 2

Realizar un programa que permita la suma de 5 y 3. Ejemplo:  $AX=5+3$ , resultado:  $AX=8$ .

```

Copyright (c) 2009 Microsoft Corporation. Reservados todos los derechos.
C:\Users\MARTIN>DEBUG
-A100
13AA:0100 MOV AX,6
13AA:0103 MOV BX,3
13AA:0106 ADD AX,BX
13AA:0108 INT 20
13AA:010A
-T
AX=0006 BX=0000 CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=13AA ES=13AA SS=13AA CS=13AA IP=0103  NU UP EI PL NZ NA PO NC
13AA:0103 BB0300      MOV     BX,0003
-T
AX=0006 BX=0003 CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=13AA ES=13AA SS=13AA CS=13AA IP=0106  NU UP EI PL NZ NA PO NC
13AA:0106 01D8      ADD     AX,BX
-T
AX=0009 BX=0003 CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=13AA ES=13AA SS=13AA CS=13AA IP=0108  NU UP EI PL NZ NA PE NC
13AA:0108 CD20      INT     20
-

```

Figura 5.9 resultado:  $AX=8$

## Ejercicio 3

Realizar un programa que permita la suma de 6 y 3. Ejemplo:  $AX=6+3$ , resultado:  $AX=9$ .

```

Microsoft Windows [Versión 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. Reservados todos los derechos.
C:\Users\MARTIN>cd..
C:\Users>cd..
C:\>debug
-A100
13AA:0100 MOV AX,6
13AA:0103 MOV BX,2

```

Figura 5.10 resultado:  $AX=9$

## Ejercicio 4

Realizar un programa que permita la resta de 6 y 2. Ejemplo:  $AX=6-2$ , resultado:  $AX=4$ .

1. Ingrese a cmd.

```

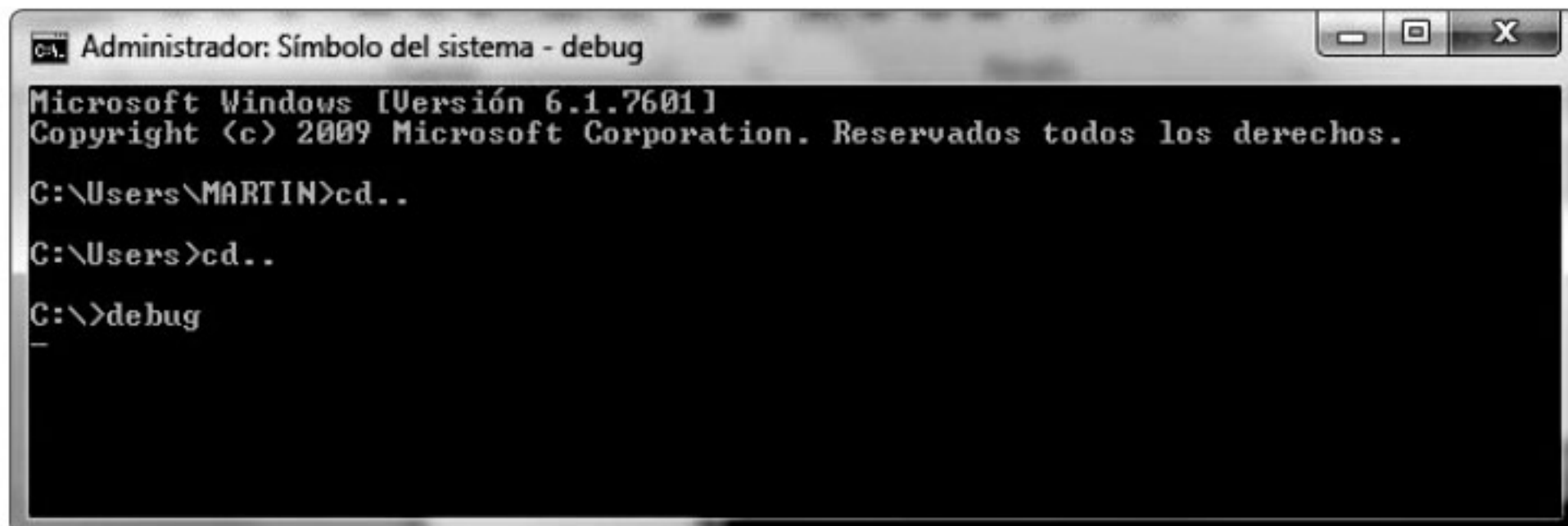
Microsoft Windows [Versión 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. Reservados todos los derechos.
C:\Users\MARTIN>

```

Figura 5.11 Ingresar cmd



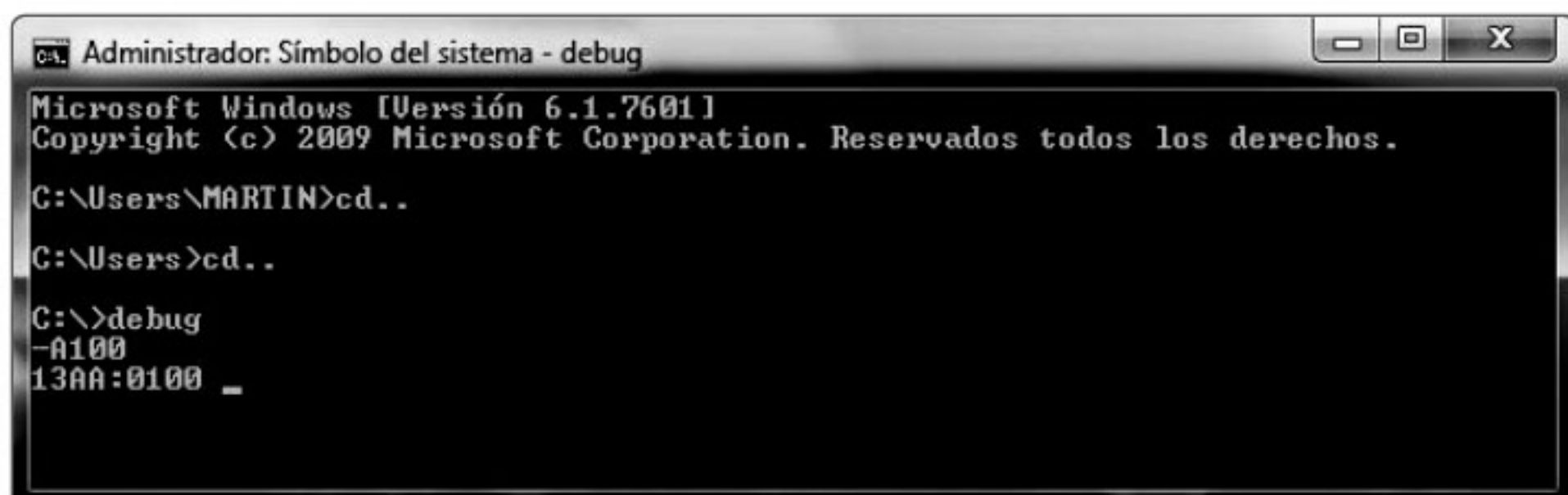
2. Debe ir a la raíz del programa, por lo tanto, utilice la instrucción **cd..** y luego la instrucción **debug**.



```
Administrador: Símbolo del sistema - debug
Microsoft Windows [Versión 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. Reservados todos los derechos.
C:\Users\MARTIN>cd..
C:\Users>cd..
C:\>debug
```

Figura 5.12 raiz&degub

3. Asigne un direccionamiento de memoria con la instrucción **A100** y luego presione **Enter**.



```
Administrador: Símbolo del sistema - debug
Microsoft Windows [Versión 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. Reservados todos los derechos.
C:\Users\MARTIN>cd..
C:\Users>cd..
C:\>debug
-A100
13AA:0100 _
```

Figura 5.13 insA100

4. Con la instrucción **MOV**, asigne un número para A y B.



```
Administrador: Símbolo del sistema - debug
Microsoft Windows [Versión 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. Reservados todos los derechos.
C:\Users\MARTIN>cd..
C:\Users>cd..
C:\>debug
-A100
13AA:0100 MOV AX,6
13AA:0103 MOV BX,2
```

Figura 5.14 AsignarAB

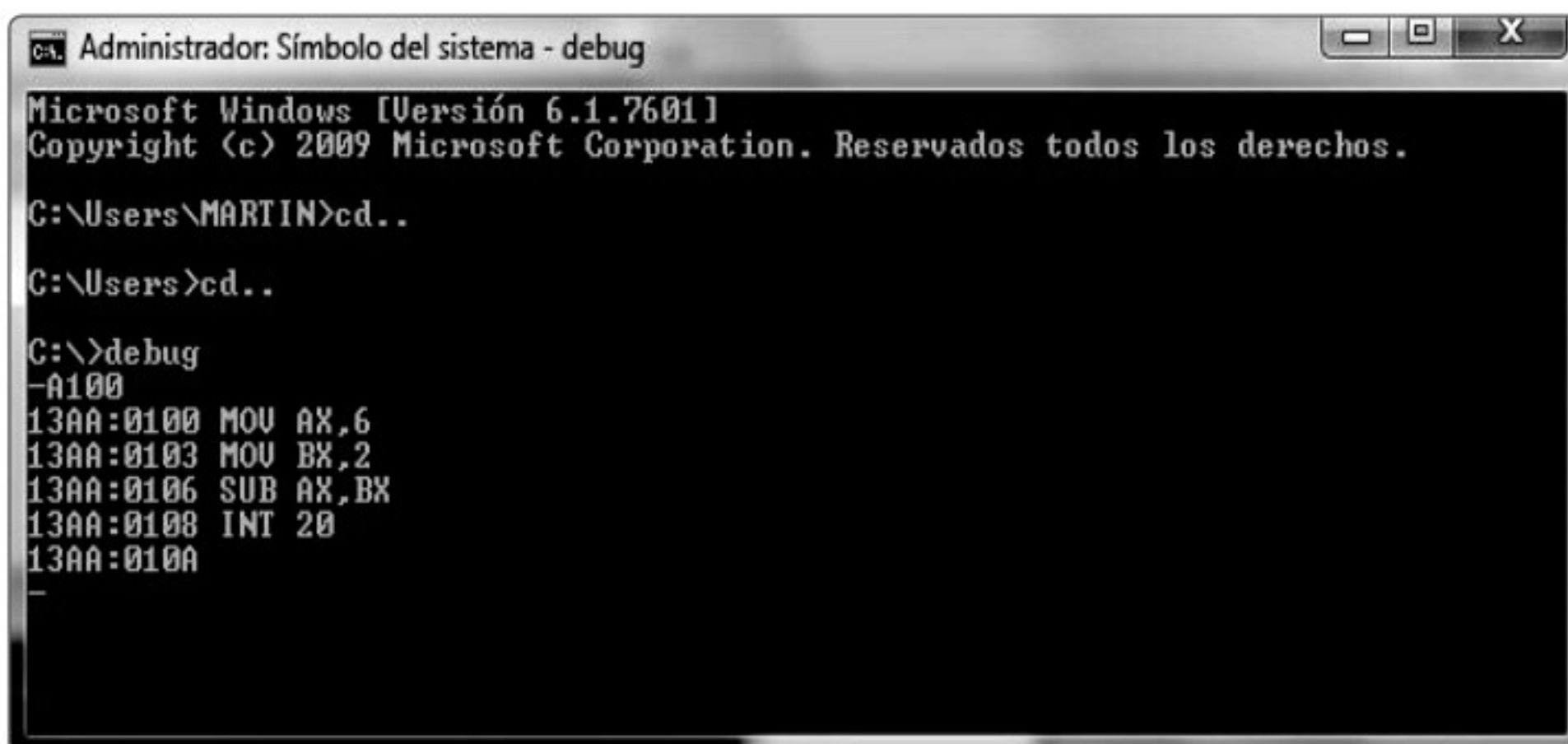
5. Para restar, utilice la instrucción **SUB**.



```
Administrador: Símbolo del sistema - debug
Microsoft Windows [Versión 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. Reservados todos los derechos.
C:\Users\MARTIN>cd..
C:\Users>cd..
C:\>debug
-A100
13AA:0100 MOV AX,6
13AA:0103 MOV BX,2
13AA:0106 SUB AX,BX
13AA:0108
```

Figura 5.15 InsSUB

6. Para darle una pausa, escriba la instrucción **INT 20**. Luego, presione **Enter** dos veces.

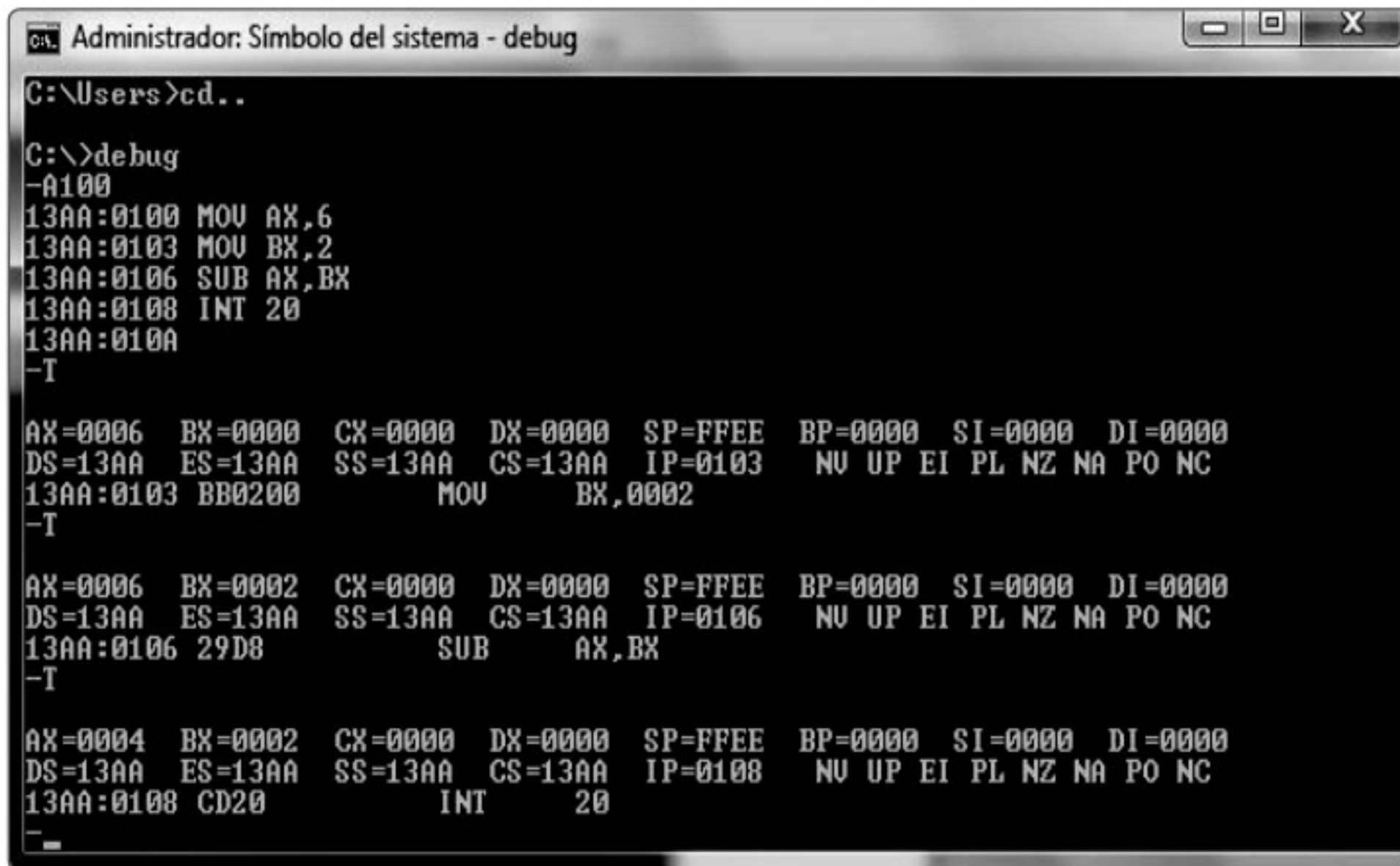


```
Administrador: Símbolo del sistema - debug
Microsoft Windows [Versión 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. Reservados todos los derechos.
C:\Users\MARTIN>cd..
C:\Users>cd..
C:\>debug
-A100
13AA:0100 MOV AX,6
13AA:0103 MOV BX,2
13AA:0106 SUB AX,BX
13AA:0108 INT 20
13AA:010A
```

Figura 5.16 finalizarINT



7. Para ver el programa línea por línea, escriba la instrucción **t**.



```

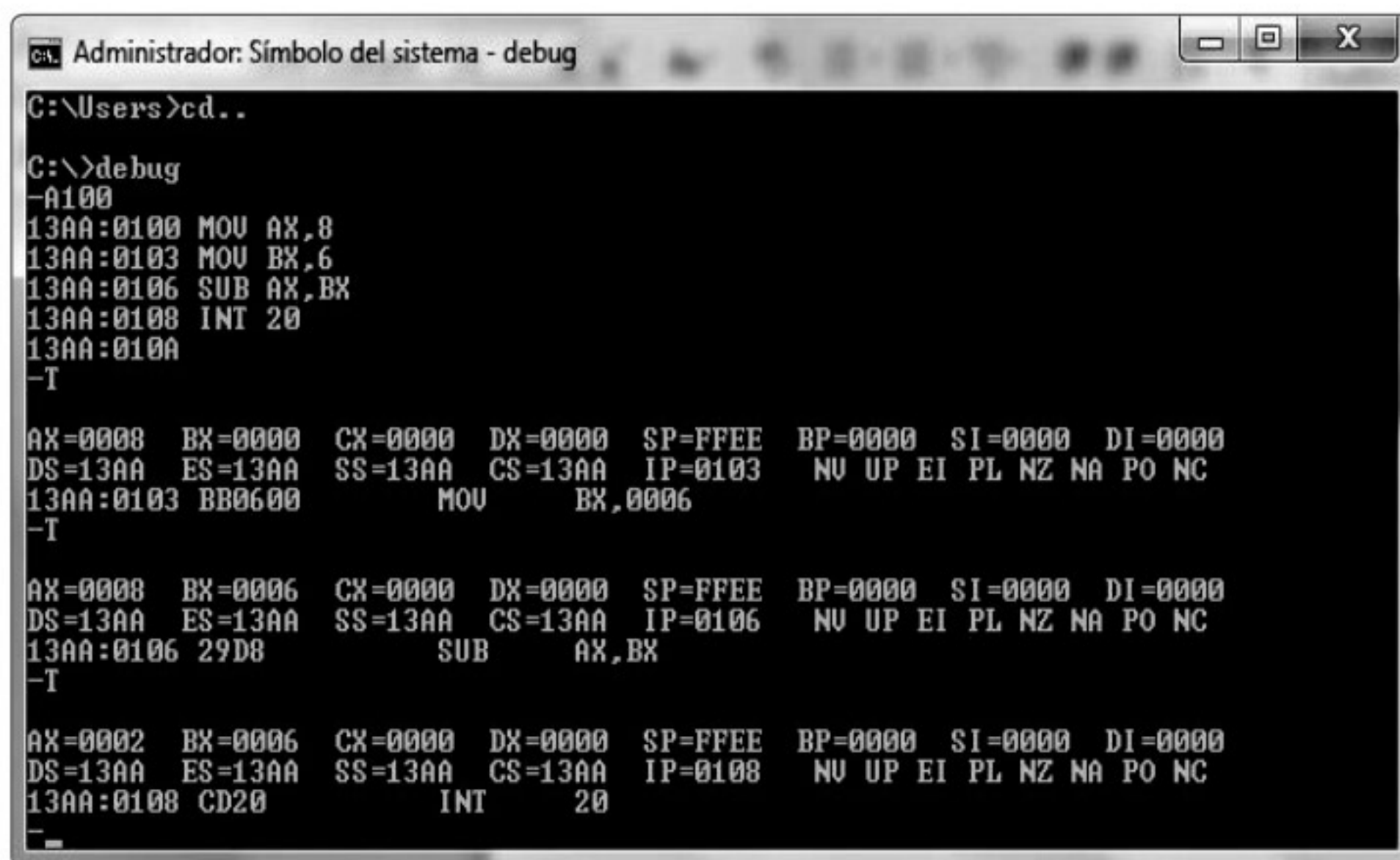
C:\Users>cd..
C:\>debug
-A100
13AA:0100 MOV AX,6
13AA:0103 MOV BX,2
13AA:0106 SUB AX,BX
13AA:0108 INT 20
13AA:010A
-T
AX=0006 BX=0000 CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=13AA ES=13AA SS=13AA CS=13AA IP=0103  NU UP EI PL NZ NA PO NC
13AA:0103 BB0200      MOV     BX,0002
-T
AX=0006 BX=0002 CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=13AA ES=13AA SS=13AA CS=13AA IP=0106  NU UP EI PL NZ NA PO NC
13AA:0106 29D8      SUB     AX,BX
-T
AX=0004 BX=0002 CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=13AA ES=13AA SS=13AA CS=13AA IP=0108  NU UP EI PL NZ NA PO NC
13AA:0108 CD20      INT     20
-

```

Figura 5.17 resultado: 4

## Ejercicio 5

Realizar un programa que permita la resta de 8 y 6. Ejemplo:  $AX=8-6$ , resultado:  $AX=2$ .



```

C:\Users>cd..
C:\>debug
-A100
13AA:0100 MOV AX,8
13AA:0103 MOV BX,6
13AA:0106 SUB AX,BX
13AA:0108 INT 20
13AA:010A
-T
AX=0008 BX=0000 CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=13AA ES=13AA SS=13AA CS=13AA IP=0103  NU UP EI PL NZ NA PO NC
13AA:0103 BB0600      MOV     BX,0006
-T
AX=0008 BX=0006 CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=13AA ES=13AA SS=13AA CS=13AA IP=0106  NU UP EI PL NZ NA PO NC
13AA:0106 29D8      SUB     AX,BX
-T
AX=0002 BX=0006 CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=13AA ES=13AA SS=13AA CS=13AA IP=0108  NU UP EI PL NZ NA PO NC
13AA:0108 CD20      INT     20
-

```

Figura 5.18 resultado: 2

## Ejercicio 6

Realizar un programa que permita la suma de 2, 3 y 4. Ejemplo:  $AX=2+3+4$ , resultado:  $AX=9$ .

1. Abra la ventana **Ejecutar** con  + R y escriba **cmd** para abrir el símbolo del sistema.

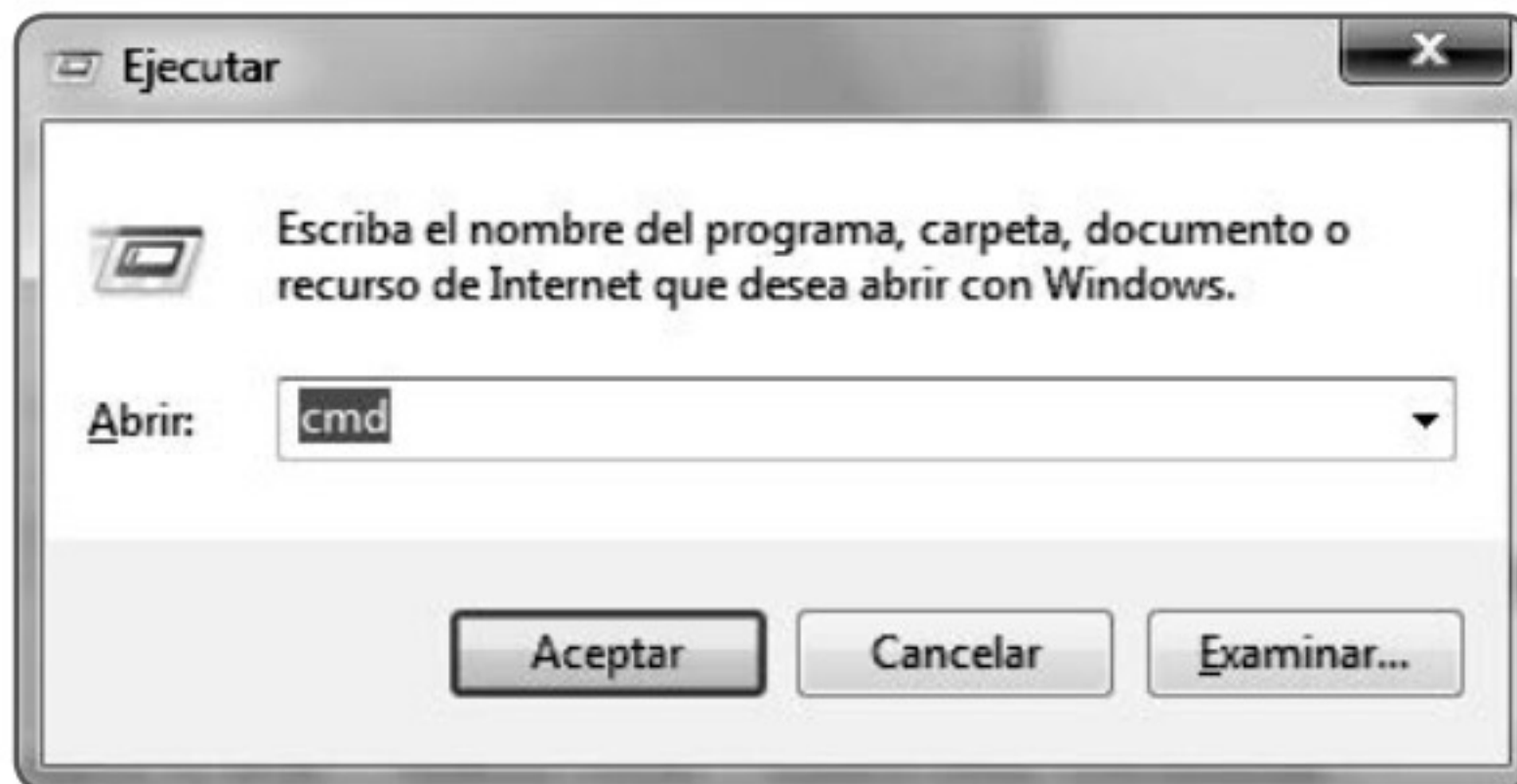


Figura 5.19 vista\_ejecutar

2. Ingrese a **cmd**.



Figura 5.20 vista\_cmd

3. Digite los datos.

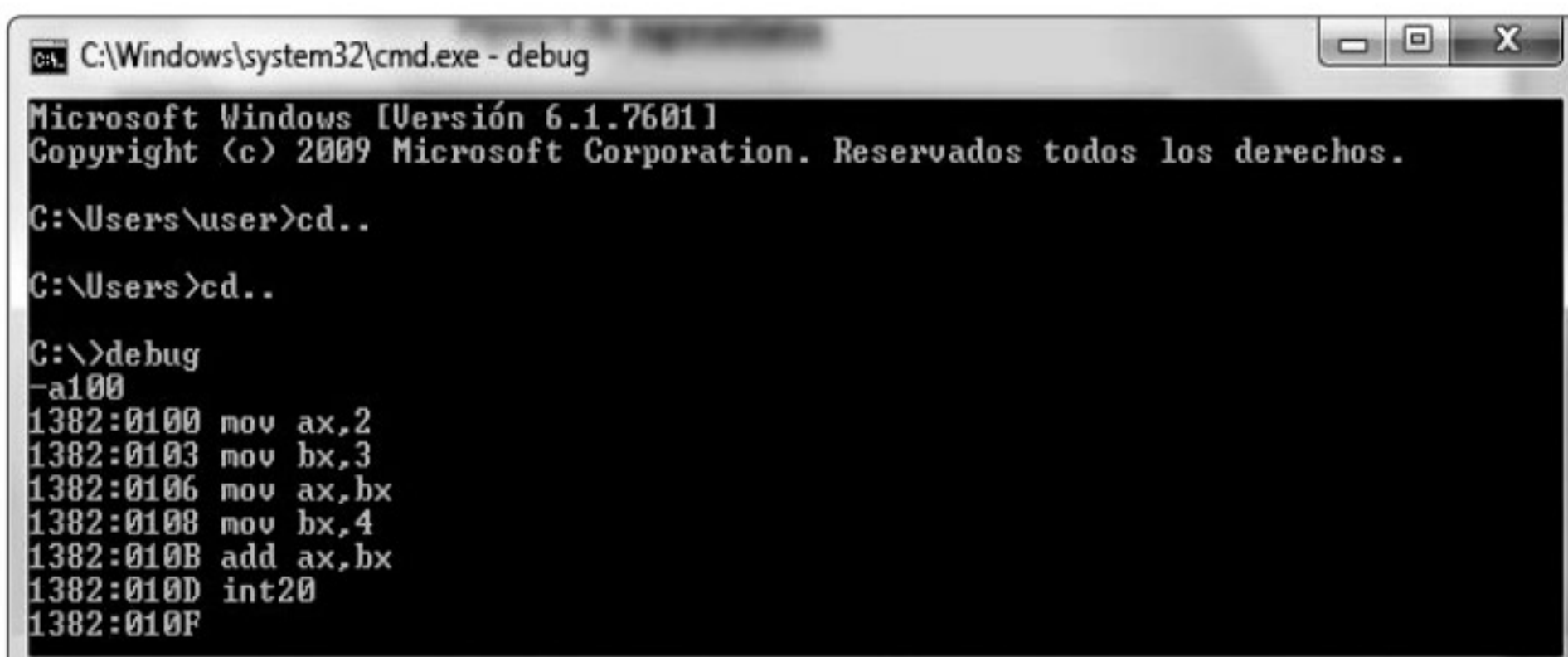


Figura 5.21 ingrese datos



4. Observe los resultados.

```

C:\Windows\system32\cmd.exe - debug
AX=0002 BX=0000 CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=1382 ES=1382 SS=1382 CS=1382 IP=0103  NU UP EI PL NZ NA PO NC
1382:0103 BB0300      MOV     BX,0003
-t
AX=0002 BX=0003 CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=1382 ES=1382 SS=1382 CS=1382 IP=0106  NU UP EI PL NZ NA PO NC
1382:0106 01D8      ADD     AX,BX
-t
AX=0005 BX=0003 CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=1382 ES=1382 SS=1382 CS=1382 IP=0108  NU UP EI PL NZ NA PE NC
1382:0108 BB0400      MOV     BX,0004
-t
AX=0005 BX=0004 CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=1382 ES=1382 SS=1382 CS=1382 IP=010B  NU UP EI PL NZ NA PE NC
1382:010B 01D8      ADD     AX,BX
-t
AX=0009 BX=0004 CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=1382 ES=1382 SS=1382 CS=1382 IP=010D  NU UP EI PL NZ NA PE NC
1382:010D CD20      INT     20

```

Figura 5.22 resumen

## Ejercicio 7

Realizar un programa que permita la suma de 1, 2 y 3. Ejemplo:  $AX=1+2+3$ , resultado:  $AX=6$ .

```

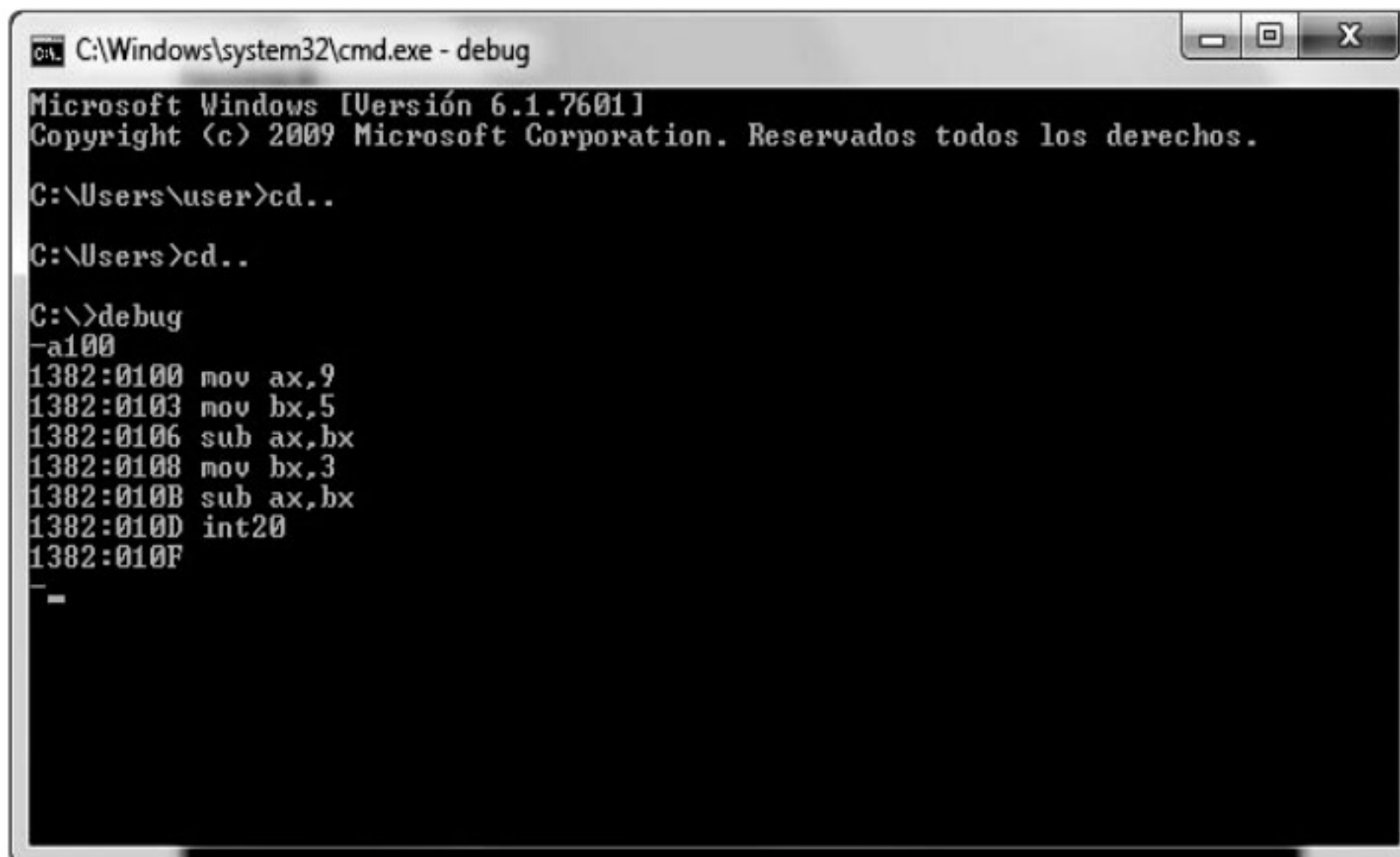
C:\Windows\system32\cmd.exe - debug
AX=0001 BX=0000 CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=1380 ES=1380 SS=1380 CS=1380 IP=0103  NU UP EI PL NZ NA PO NC
1380:0103 BB0200      MOV     BX,0002
-t
AX=0001 BX=0002 CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=1380 ES=1380 SS=1380 CS=1380 IP=0106  NU UP EI PL NZ NA PO NC
1380:0106 01D8      ADD     AX,BX
-t
AX=0003 BX=0002 CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=1380 ES=1380 SS=1380 CS=1380 IP=0108  NU UP EI PL NZ NA PE NC
1380:0108 BB0300      MOV     BX,0003
-t
AX=0003 BX=0003 CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=1380 ES=1380 SS=1380 CS=1380 IP=010B  NU UP EI PL NZ NA PE NC
1380:010B 01D8      ADD     AX,BX
-t
AX=0006 BX=0003 CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=1380 ES=1380 SS=1380 CS=1380 IP=010D  NU UP EI PL NZ NA PE NC
1380:010D CD20      INT     20

```

Figura 5.23 resultado: 6

## Ejercicio 8

Realice un programa que permita la resta de 9, 5 y 3. Ejemplo:  $AX=9-5-3$ , resultado:  $AX=1$ .

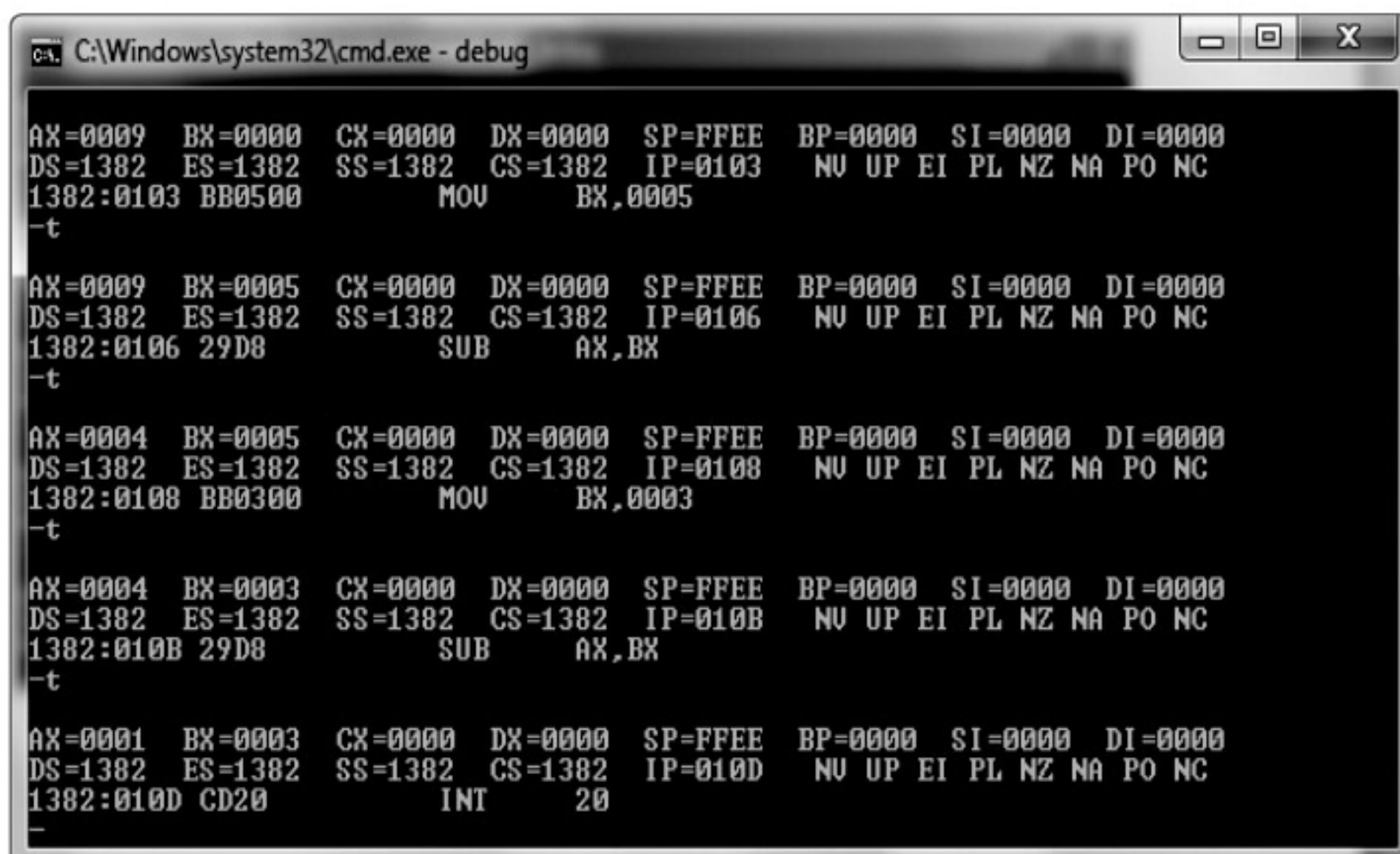


```

C:\Windows\system32\cmd.exe - debug
Microsoft Windows [Versión 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. Reservados todos los derechos.

C:\Users\user>cd..
C:\Users>cd..
C:\>debug
-a100
1382:0100 mov ax,9
1382:0103 mov bx,5
1382:0106 sub ax,bx
1382:0108 mov bx,3
1382:010B sub ax,bx
1382:010D int20
1382:010F
-
  
```

Figura 5.24 ingreseDatos



```

C:\Windows\system32\cmd.exe - debug
AX=0009 BX=0000 CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=1382 ES=1382 SS=1382 CS=1382 IP=0103  NU UP EI PL NZ NA PO NC
1382:0103 BB0500      MOV     BX,0005
-t
AX=0009 BX=0005 CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=1382 ES=1382 SS=1382 CS=1382 IP=0106  NU UP EI PL NZ NA PO NC
1382:0106 29D8      SUB     AX,BX
-t
AX=0004 BX=0005 CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=1382 ES=1382 SS=1382 CS=1382 IP=0108  NU UP EI PL NZ NA PO NC
1382:0108 BB0300      MOV     BX,0003
-t
AX=0004 BX=0003 CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=1382 ES=1382 SS=1382 CS=1382 IP=010B  NU UP EI PL NZ NA PO NC
1382:010B 29D8      SUB     AX,BX
-t
AX=0001 BX=0003 CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=1382 ES=1382 SS=1382 CS=1382 IP=010D  NU UP EI PL NZ NA PO NC
1382:010D CD20      INT     20
-
  
```

Figura 5.25 resultado: 1



## Ejercicio 9

Realizar un programa que permita la resta de 8, 4 y 2. Ejemplo:  $AX=8-4-2$ , resultado:  $AX=2$ .

```

C:\Windows\system32\cmd.exe - debug
Microsoft Windows [Versión 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. Reservados todos los derechos.

C:\Users\user>cd..
C:\Users>cd..
C:\>cd..
C:\>debug
-a100
1382:0100 mov ax,8
1382:0103 mov bx,4
1382:0106 sub ax,bx
1382:0108 mov bx,2
1382:010B sub ax,bx
1382:010D int20
1382:010F
-

```

Figura 5.26 Datos953

```

C:\Windows\system32\cmd.exe - debug
AX=0008 BX=0000 CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=1382 ES=1382 SS=1382 CS=1382 IP=0103 NU UP EI PL NZ NA PO NC
1382:0103 BB0400 MOV BX,0004
-t
AX=0008 BX=0004 CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=1382 ES=1382 SS=1382 CS=1382 IP=0106 NU UP EI PL NZ NA PO NC
1382:0106 29D8 SUB AX,BX
-t
AX=0004 BX=0004 CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=1382 ES=1382 SS=1382 CS=1382 IP=0108 NU UP EI PL NZ NA PO NC
1382:0108 BB0200 MOV BX,0002
-t
AX=0004 BX=0002 CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=1382 ES=1382 SS=1382 CS=1382 IP=010B NU UP EI PL NZ NA PO NC
1382:010B 29D8 SUB AX,BX
-t
AX=0002 BX=0002 CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=1382 ES=1382 SS=1382 CS=1382 IP=010D NU UP EI PL NZ NA PO NC
1382:010D CD20 INT 20
-

```

Figura 5.27 resultado: 2

## Ejercicio 10

Empleando un display de 7 segmentos y con el PIC16F84A, realizar el desplazamiento de ledes hacia la derecha con un retardo de 200 ms. Realizar el listado `.asm`, simular y programar el PIC.

```

;-----
; Código en assembler:
;-----
LIST p=16F84A
INCLUDE <P16F84A.INC>
__CONFIG _CP_OFF & _WDT_OFF & _PWRTE_ON & _XT_OSC
CBLOCK 1x0c ; Incrustación de la librería en el Registro CONTADOR
CONTADOR
ENDC
#DEFINE DISPLAY PORTB; Definimos el Display en el puerto B
ORG 0
BSF STATUS,RP0 ;Banco de memoria
CLRF DISPLAY
BCF STATUS,RP0
INICIO      CLRF CONTADOR
            MOVF CONTADOR,W      ; Movemos el valor de Contador a W
            CALL SIETE_SEGMENTOS
            MOVWF DISPLAY
            CALL Retardo_200ms

CONTAR
            INCF CONTADOR, F
            MOVLW d'7'
            SUBWF CONTADOR, W
            BTFSC STATUS, C
            GOTO INICIO
            MOVF CONTADOR, W
            CALL SIETE_SEGMENTOS
            MOVWF DISPLAY
            CALL Retardo_200ms
            GOTO CONTAR

SIETE_SEGMENTOS
ADDWF PCL, F
TABLA
            RETLW b'00000001'
            RETLW b'00000010'
            RETLW b'00000100'
            RETLW b'00001000'
            RETLW b'00010000'
            RETLW b'00100000'
            RETLW b'01000000'

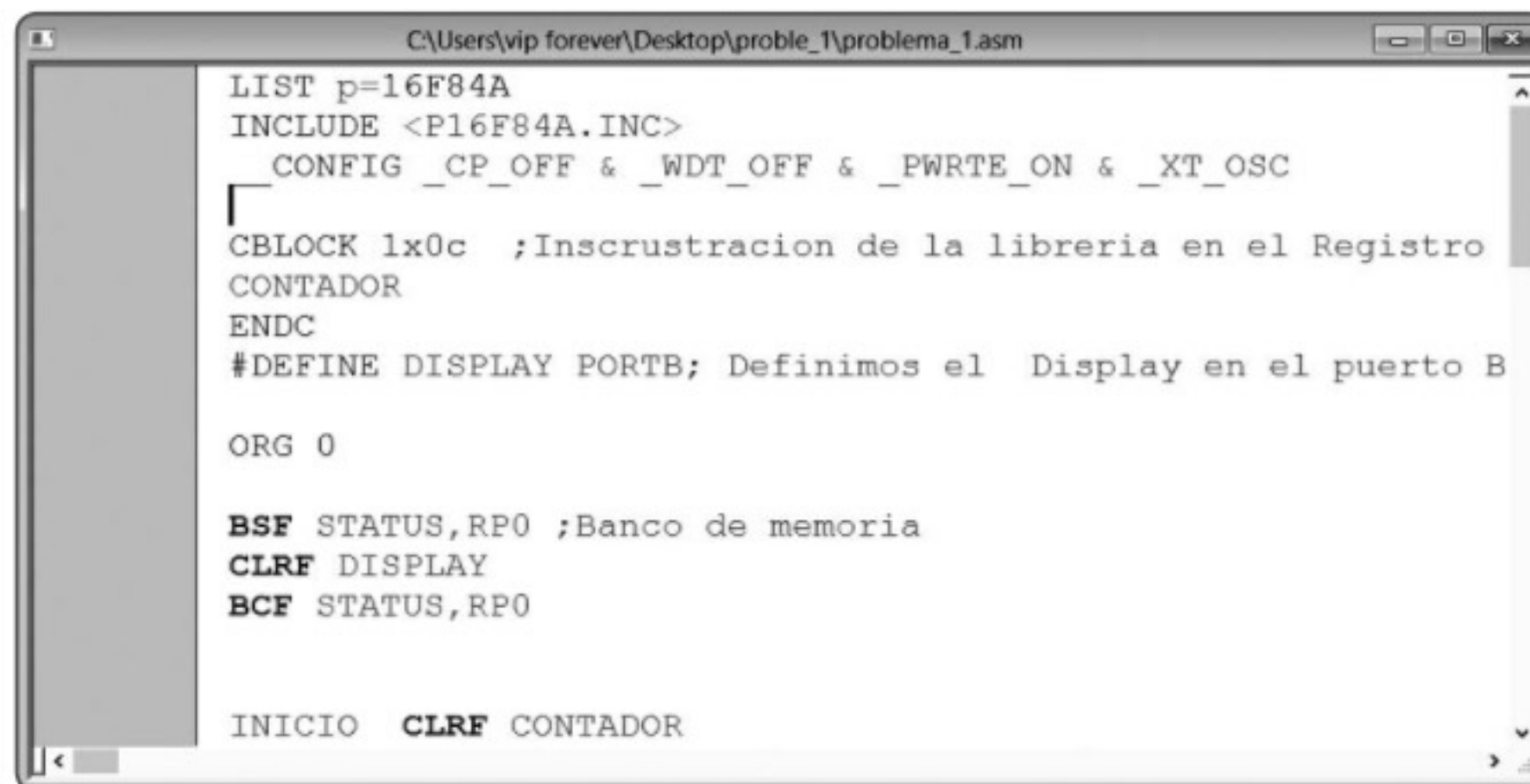
INCLUDE <Retardos.inc>
END

```



Abra el programa MPLAB, al hacer clic en **New** se abrirá una ventana donde escribirá los códigos correspondientes para el funcionamiento del programa.

Cargando el programa en PROTEUS:



```

C:\Users\vip forever\Desktop\proble_1\problema_1.asm
LIST p=16F84A
INCLUDE <P16F84A.INC>
_CONFIG _CP_OFF & _WDT_OFF & _PWRTE_ON & _XT_OSC

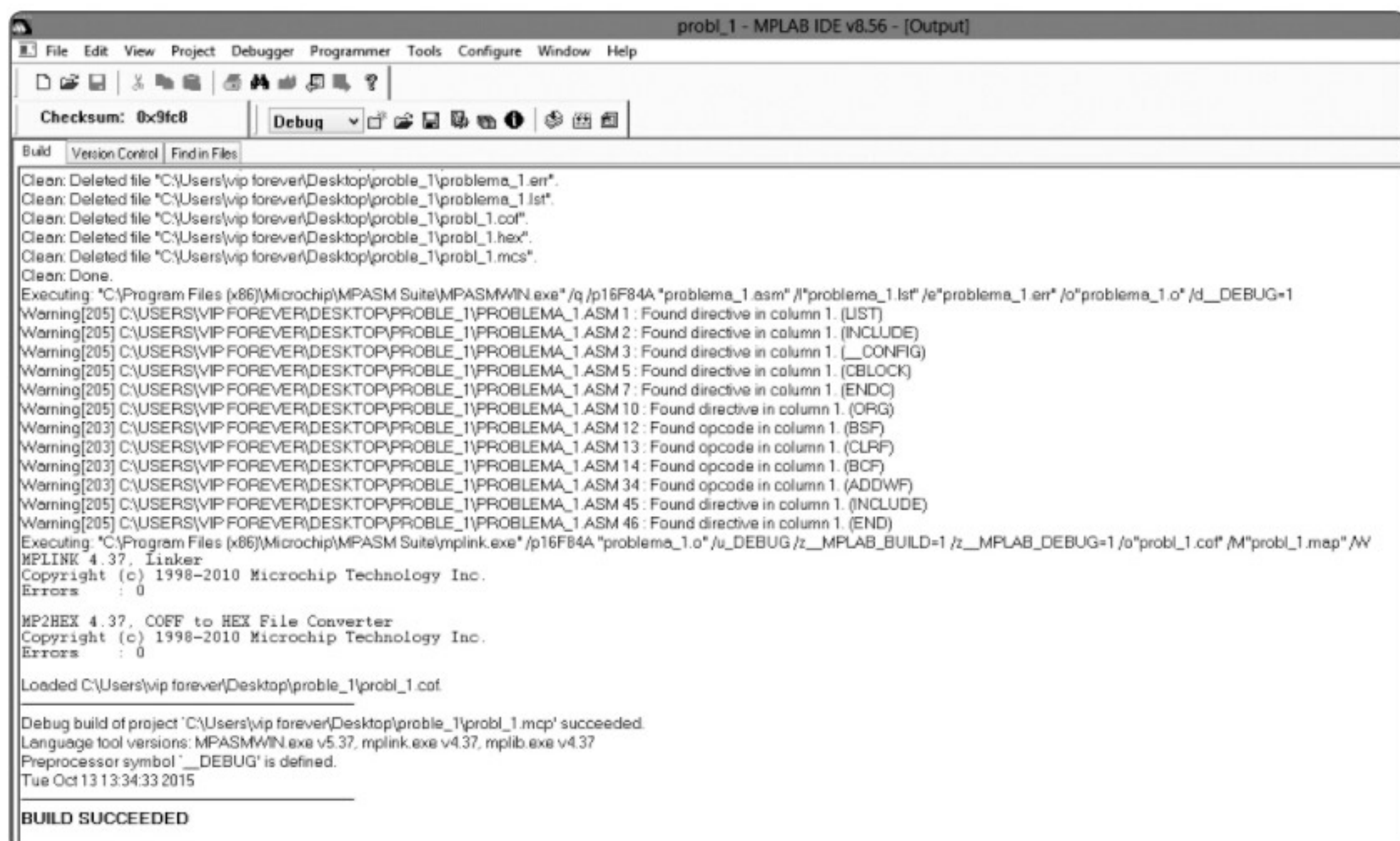
CBLOCK 1x0c ;Inscrustracion de la libreria en el Registro
CONTADOR
ENDC
#DEFINE DISPLAY PORTB; Definimos el Display en el puerto B

ORG 0

BSF STATUS,RP0 ;Banco de memoria
CLRF DISPLAY
BCF STATUS,RP0

INICIO CLRF CONTADOR
  
```

Figura 5.28 Mplabcodigo



```

probi_1 - MPLAB IDE v8.56 - [Output]
File Edit View Project Debugger Programmer Tools Configure Window Help
Checksum: 0x9fc8 Debug
Build Version Control Find in Files
Clean: Deleted file "C:\Users\vip forever\Desktop\proble_1\problema_1.err".
Clean: Deleted file "C:\Users\vip forever\Desktop\proble_1\problema_1.lst".
Clean: Deleted file "C:\Users\vip forever\Desktop\proble_1\probi_1.cof".
Clean: Deleted file "C:\Users\vip forever\Desktop\proble_1\probi_1.hex".
Clean: Deleted file "C:\Users\vip forever\Desktop\proble_1\probi_1.mcs".
Clean: Done.
Executing: "C:\Program Files (x86)\Microchip\MPASM Suite\MPASMWIN.exe" /q /p16F84A "problema_1.asm" /l"problema_1.lst" /e"problema_1.err" /o"problema_1.o" /d__DEBUG=1
Warning[205] C:\USERS\VIP FOREVER\DESKTOP\PROBLE_1\PROBLEMA_1.ASM 1: Found directive in column 1. (LIST)
Warning[205] C:\USERS\VIP FOREVER\DESKTOP\PROBLE_1\PROBLEMA_1.ASM 2: Found directive in column 1. (INCLUDE)
Warning[205] C:\USERS\VIP FOREVER\DESKTOP\PROBLE_1\PROBLEMA_1.ASM 3: Found directive in column 1. (_CONFIG)
Warning[205] C:\USERS\VIP FOREVER\DESKTOP\PROBLE_1\PROBLEMA_1.ASM 5: Found directive in column 1. (CBLOCK)
Warning[205] C:\USERS\VIP FOREVER\DESKTOP\PROBLE_1\PROBLEMA_1.ASM 7: Found directive in column 1. (ENDC)
Warning[205] C:\USERS\VIP FOREVER\DESKTOP\PROBLE_1\PROBLEMA_1.ASM 10: Found directive in column 1. (ORG)
Warning[203] C:\USERS\VIP FOREVER\DESKTOP\PROBLE_1\PROBLEMA_1.ASM 12: Found opcode in column 1. (BSF)
Warning[203] C:\USERS\VIP FOREVER\DESKTOP\PROBLE_1\PROBLEMA_1.ASM 13: Found opcode in column 1. (CLRF)
Warning[203] C:\USERS\VIP FOREVER\DESKTOP\PROBLE_1\PROBLEMA_1.ASM 14: Found opcode in column 1. (BCF)
Warning[203] C:\USERS\VIP FOREVER\DESKTOP\PROBLE_1\PROBLEMA_1.ASM 34: Found opcode in column 1. (ADDWF)
Warning[205] C:\USERS\VIP FOREVER\DESKTOP\PROBLE_1\PROBLEMA_1.ASM 45: Found directive in column 1. (INCLUDE)
Warning[205] C:\USERS\VIP FOREVER\DESKTOP\PROBLE_1\PROBLEMA_1.ASM 46: Found directive in column 1. (END)
Executing: "C:\Program Files (x86)\Microchip\MPASM Suite\mplink.exe" /p16F84A "problema_1.o" /u__DEBUG /z__MPLAB_BUILD=1 /z__MPLAB_DEBUG=1 /o"probi_1.cof" /M"probi_1.map" /W
MPLINK 4.37, Linker
Copyright (c) 1998-2010 Microchip Technology Inc.
Errors : 0
MP2HEX 4.37, COFF to HEX File Converter
Copyright (c) 1998-2010 Microchip Technology Inc.
Errors : 0
Loaded C:\Users\vip forever\Desktop\proble_1\probi_1.cof
Debug build of project "C:\Users\vip forever\Desktop\proble_1\probi_1.mcp" succeeded.
Language tool versions: MPASMWIN.exe v5.37, mplink.exe v4.37, mplib.exe v4.37
Preprocessor symbol "__DEBUG" is defined.
Tue Oct 13 13:34:33 2015
BUILD SUCCEEDED
  
```

Figura 5.29 Mplabcodigo



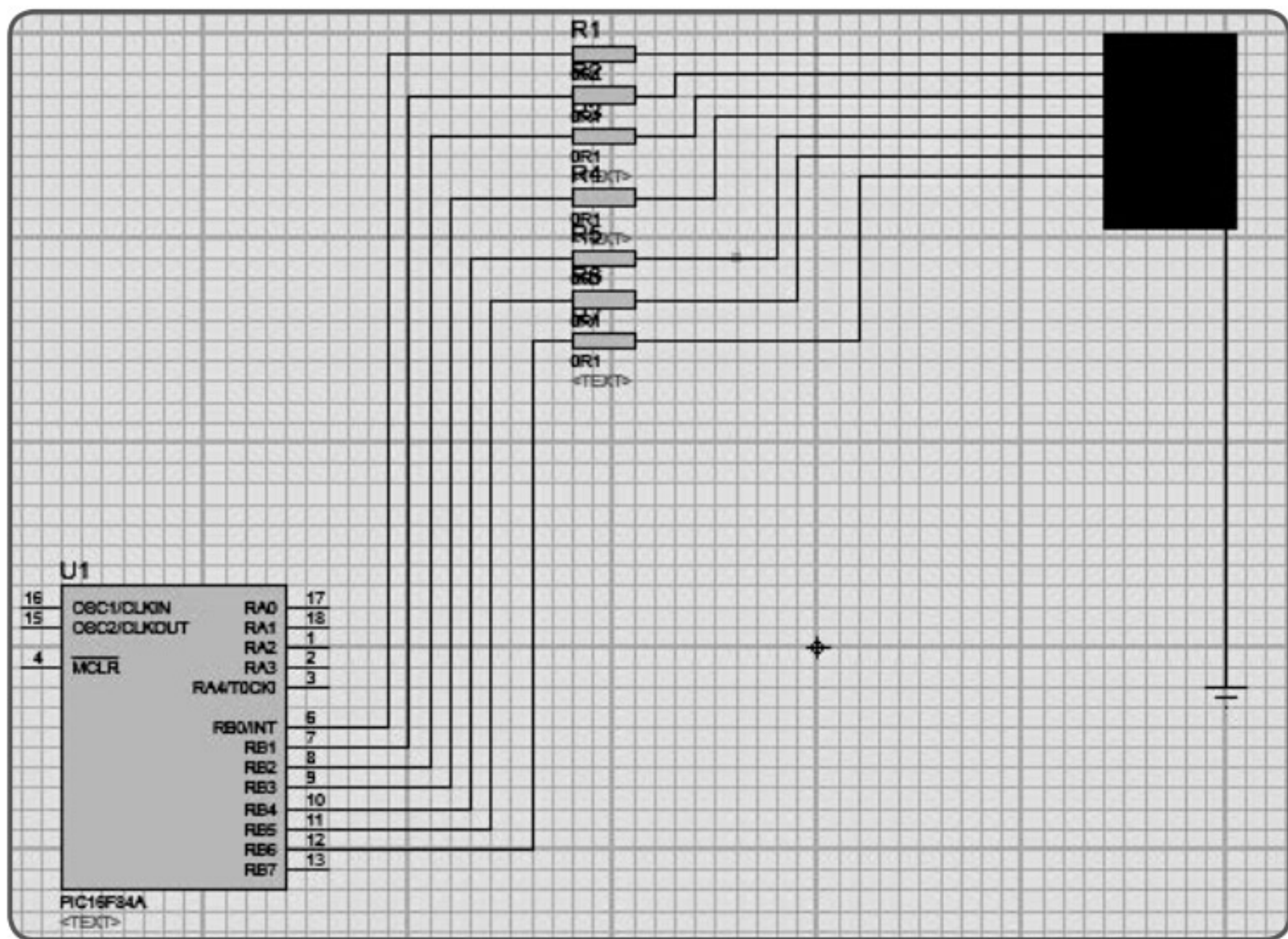
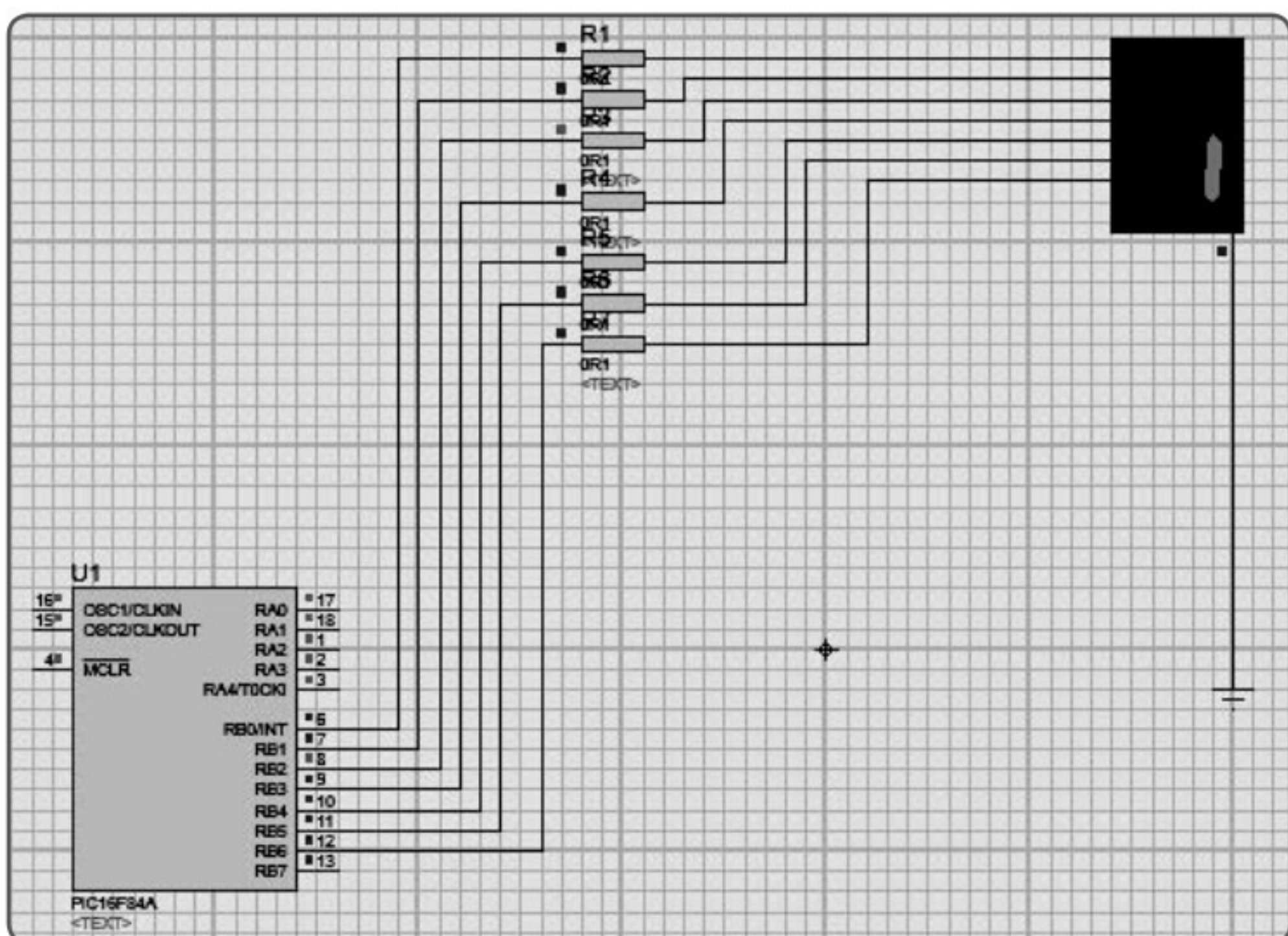


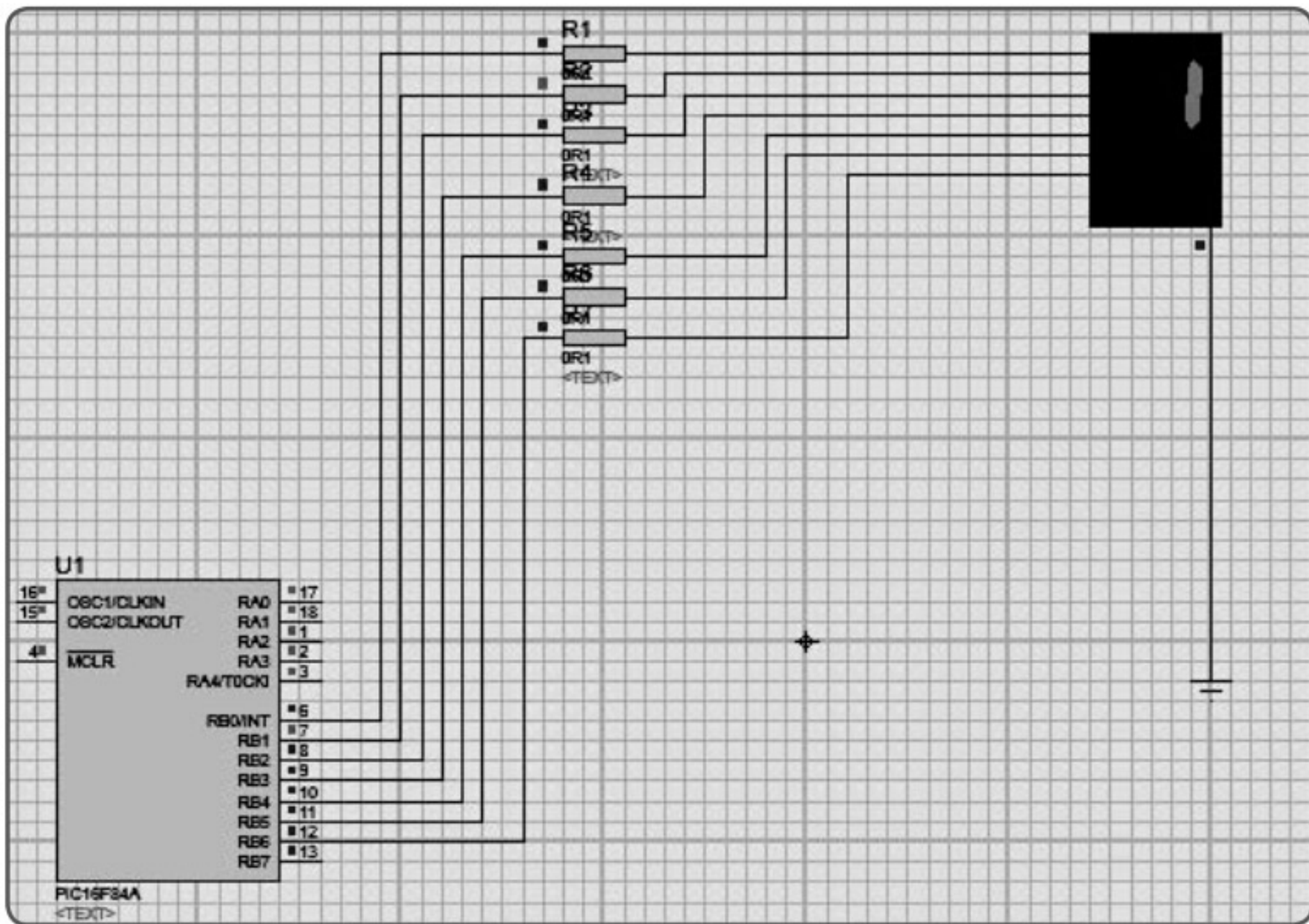
Figura 5.30 Display

### Paso 1





## Paso 2



## Paso 3

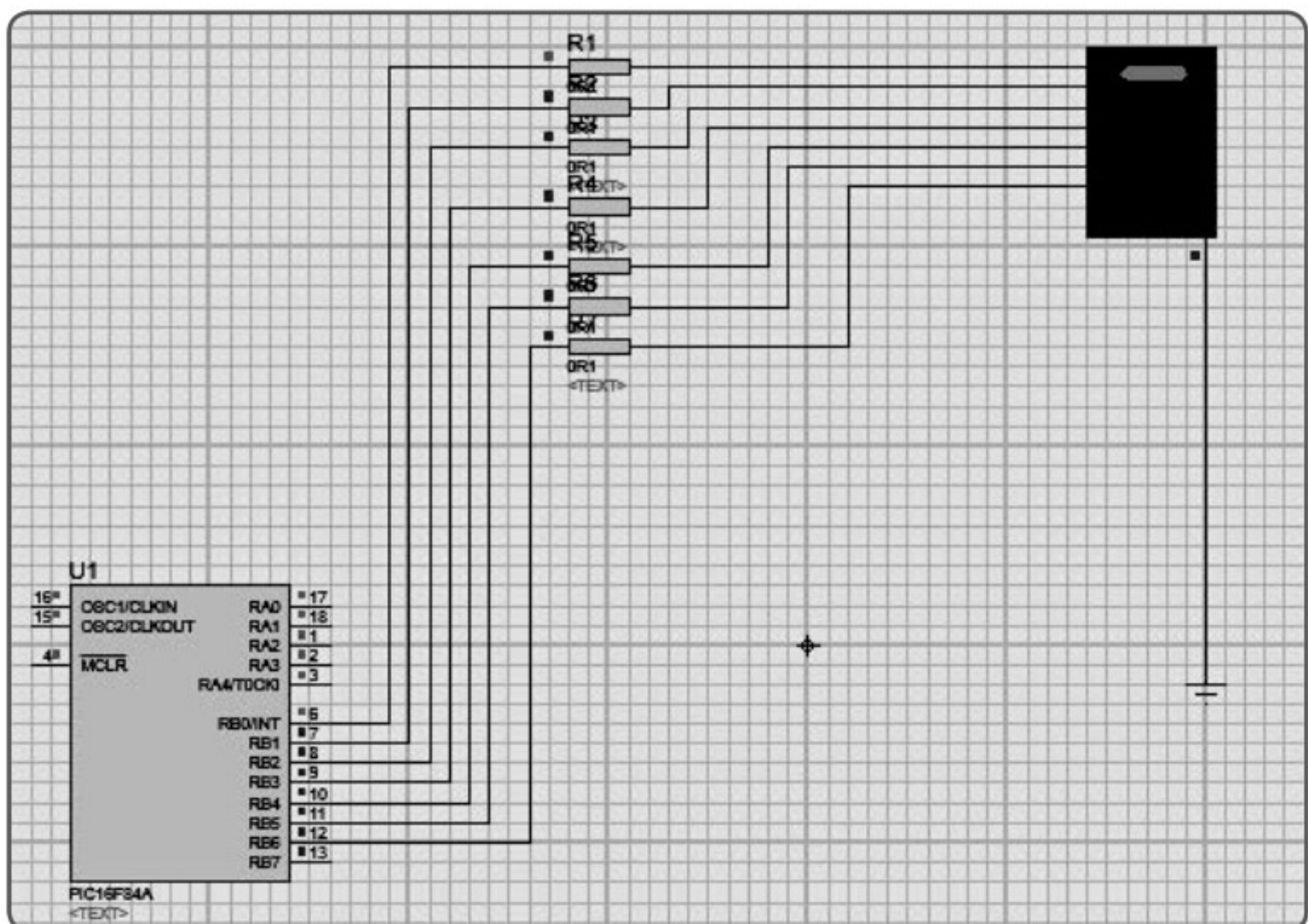
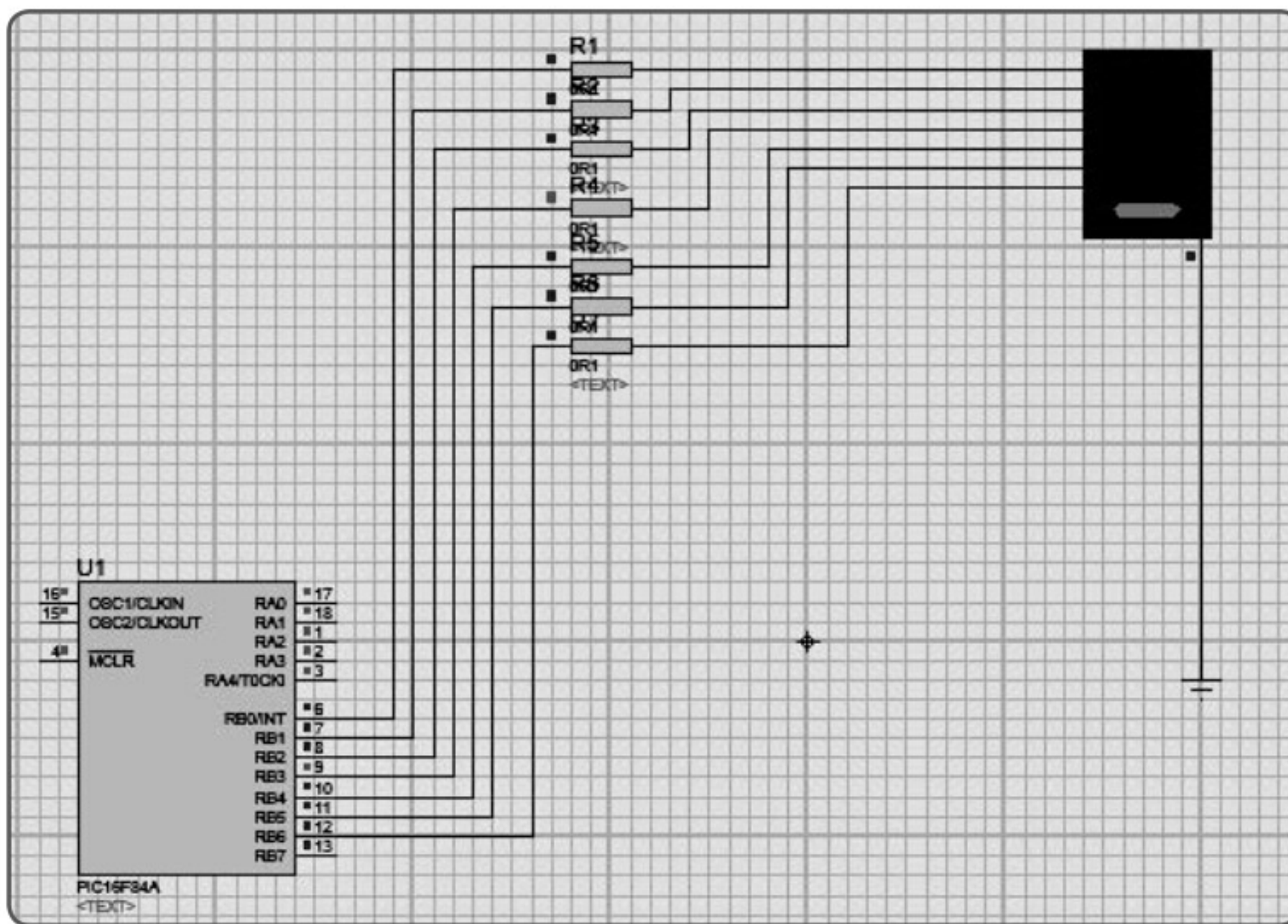


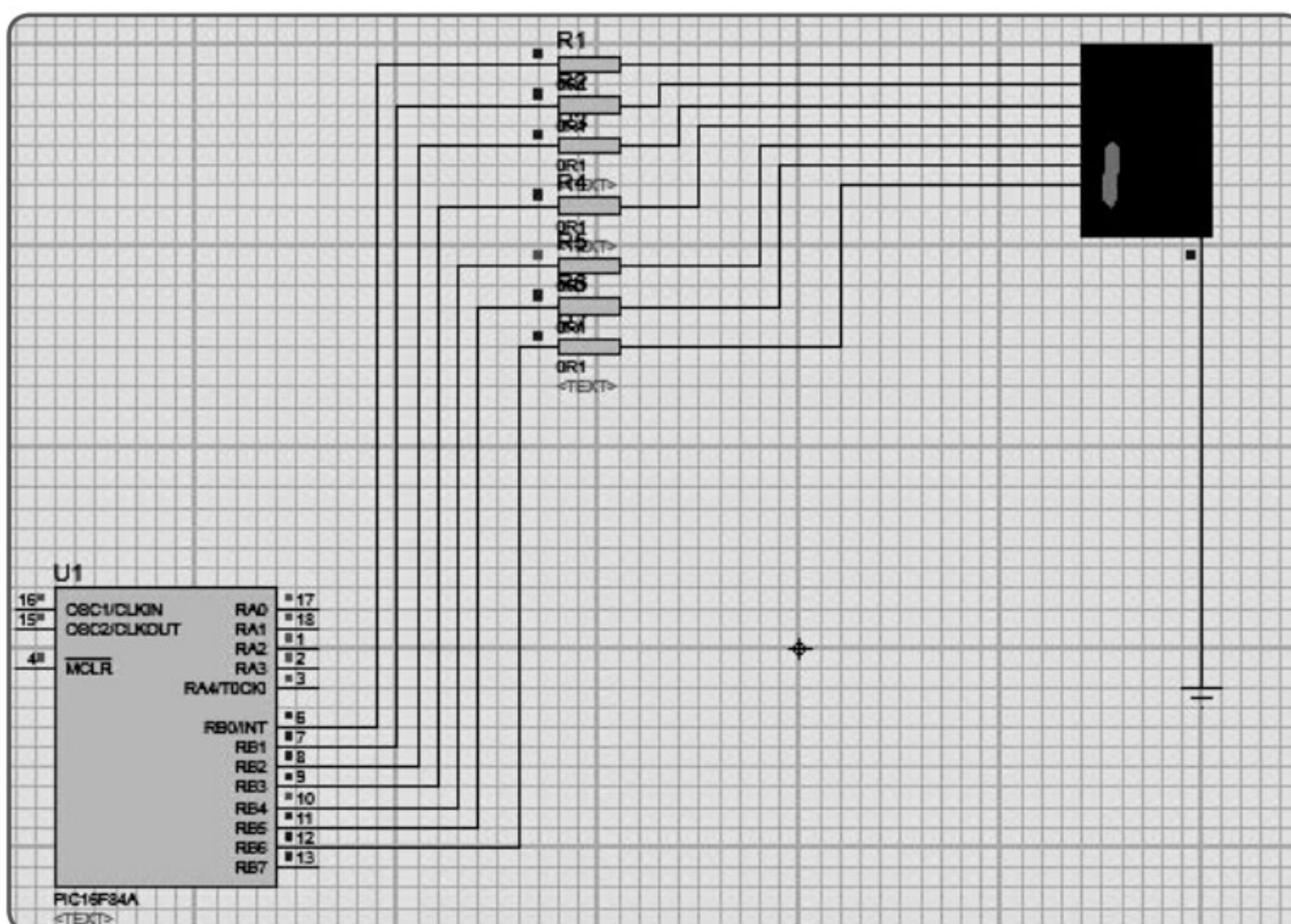
Figura 5.31 Movimientos de ledes



## Paso 1

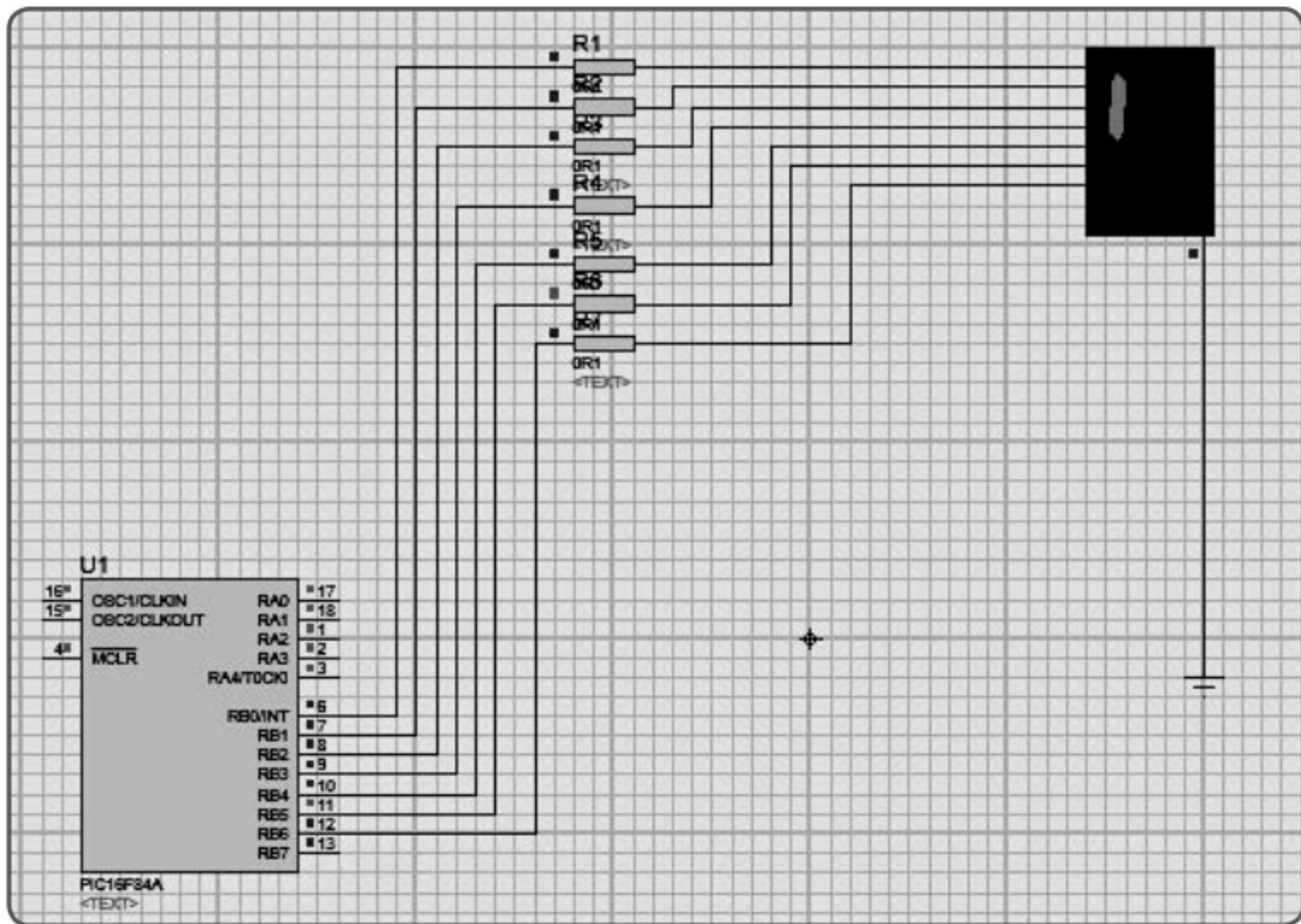


## Paso 2





## Paso 3



## Paso 4

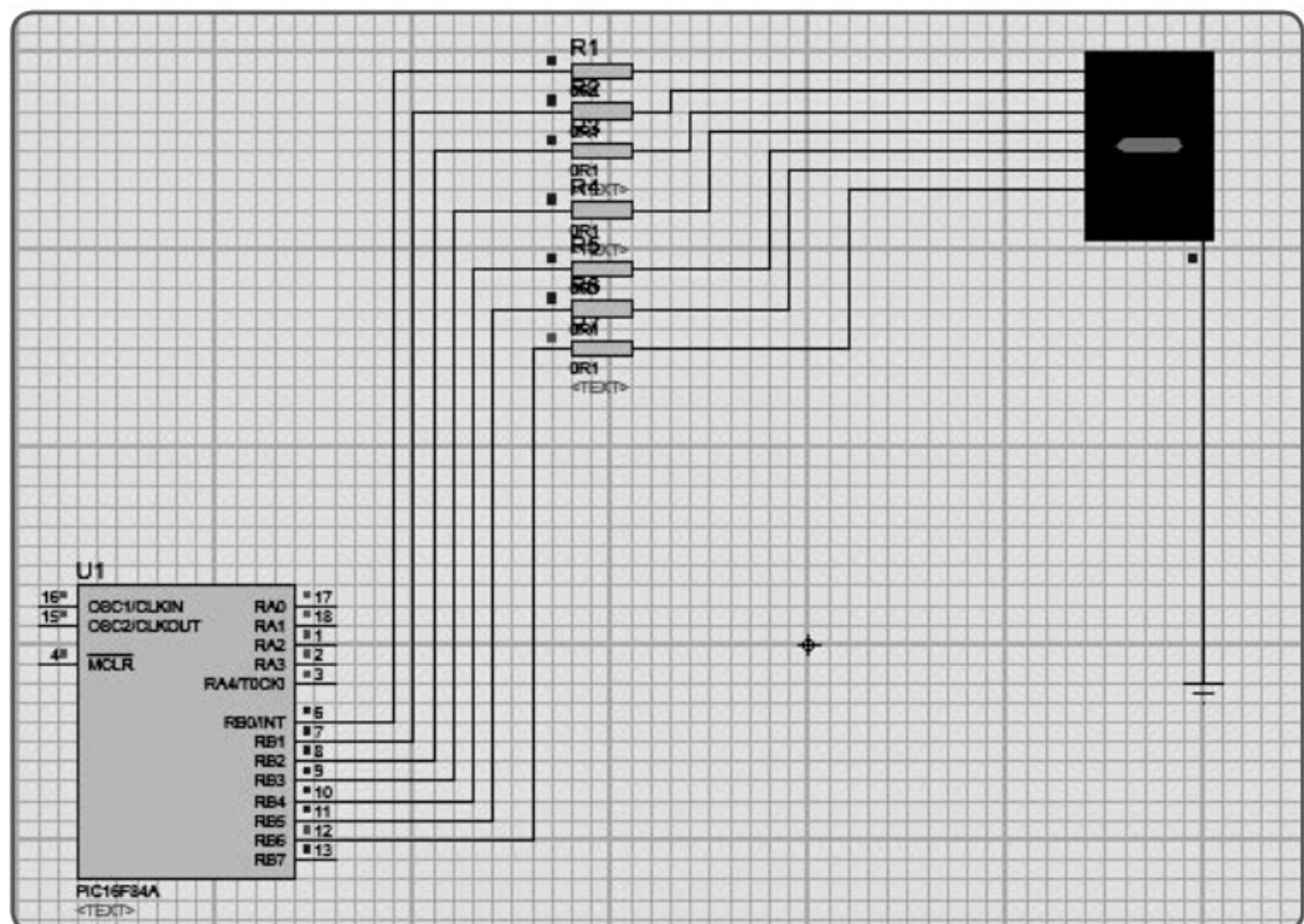


Figura 5.32 Movimientos de ledes



## Ejercicio 11

Realizar el desplazamiento de un juego de luces ledes de izquierda a derecha, con intermitencia de 5 veces conectado al puerto B, usando el PIC16F84A; realizar el programa y simular. Grabar el PIC y comprobar con el entrenador.

```
;-----  
; Código en assembler:  
;-----  
LIST P=16F84A  
INCLUDE      <P16F84A.INC>  
__CONFIG    __CP_OFF &      __WDT_OFF & __PWRTE_ON & __XT_OSC  
            CBLOCK 0X0C  
            Contador  
            ENDC  
            CONT      EQU 0X0C  
            #DEFINE   LUCES PORTB  
            org 0  
            BSF STATUS, RP0  
            CLRF LUCES  
            BCF STATUS, RP0  
  
            INICIO  
            MOVLW B'10000000'  
            MOVWF LUCES  
            CALL Retardo_100ms  
  
            MOVLW B'01000000'  
            MOVWF LUCES  
            CALLA Retardo_100ms  
  
            MOVLW B'00100000'  
            MOVWF LUCES  
            CALL Retardo_100ms  
  
            MOVLW B'00010000'  
            MOVWF LUCE  
            CALL Retardo_100ms  
  
            MOVLW B'00001000'  
            MOVWF LUCES  
            CALL Retardo_100ms  
  
            MOVLW B'00000100'  
            MOVWF LUCES  
            CALL Retardo_100ms
```



```
MOVLW B'00000010'  
MOVWF LUCES  
CALL Retardo_100ms
```

```
MOVLW B'00000001'  
MOVWF LUCES
```

```
;*****IZQUIERDA*****
```

```
MOVLW B'00000001'  
MOVWF LUCES  
CALL Retardo_100ms
```

```
MOVLW B'00000010'  
MOVWF LUCES  
CALL Retardo_100ms
```

```
MOVLW B'00000100'  
MOVWF LUCES  
CALL Retardo_100ms
```

```
MOVLW B'00001000'  
MOVWF LUCES  
CALL Retardo_100ms
```

```
MOVLW B'00010000'  
MOVWF LUCES  
CALL Retardo_100ms
```

```
MOVLW B'00100000'  
MOVWF LUCES  
CALL Retardo_100ms
```

```
MOVLW B'01000000'  
MOVWF LUCES  
CALL Retardo_100ms
```

```
MOVLW B'10000000'  
MOVWF LUCES  
CALL Retardo_100ms
```

```
MOVLW B'101'  
MOVWF CONT
```

```
CALL PARPADEO
```

```
GOTO INICIO
```

```

;*****Indicador de conducción*****
PARPADEO

MOVLW B'11111111'
MOVWF LUCES
CALL Retardo_100ms

MOVLW B'00000000'
MOVWF LUCES
CALL Retardo_200ms

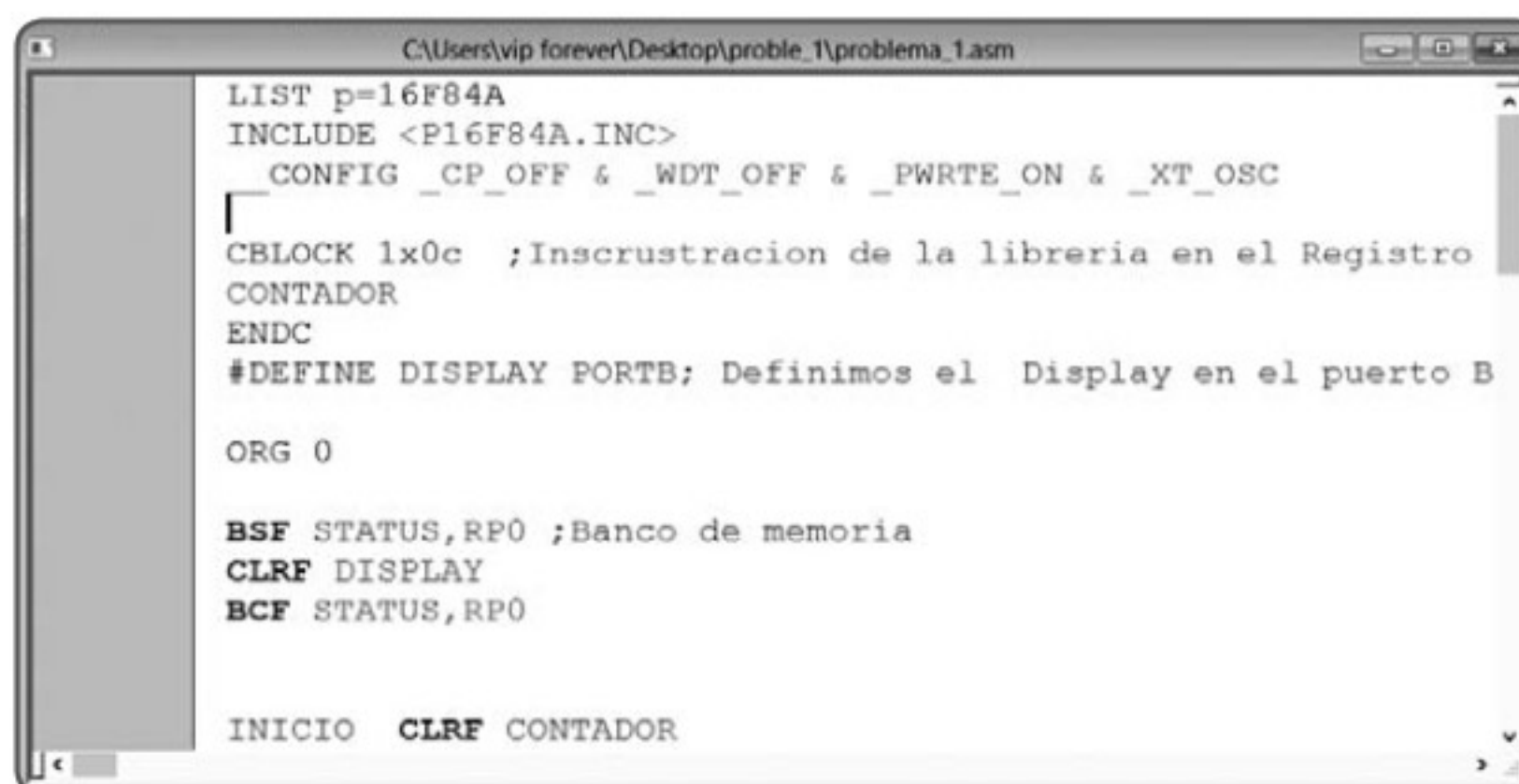
DECFSZ CONT
GOTO PARPADEO
RETURN

;*****Importar librería*****

INCLUDE<RETARDOS.INC>
END
;*****

```

Abra el programa MPLAB. Al hacer clic en **New**, se abre una ventana donde debe escribir los códigos correspondientes para el funcionamiento del programa.



```

C:\Users\vip forever\Desktop\proble_1\problema_1.asm
LIST p=16F84A
INCLUDE <P16F84A.INC>
_CONFIG _CP_OFF & _WDT_OFF & _PWRTE_ON & _XT_OSC

CBLOCK 1x0c ;Inscrustracion de la libreria en el Registro
CONTADOR
ENDC
#define DISPLAY PORTB; Definimos el Display en el puerto B

ORG 0

BSF STATUS,RP0 ;Banco de memoria
CLRF DISPLAY
BCF STATUS,RP0

INICIO CLRF CONTADOR

```

**Figura 5.33** Código Assm



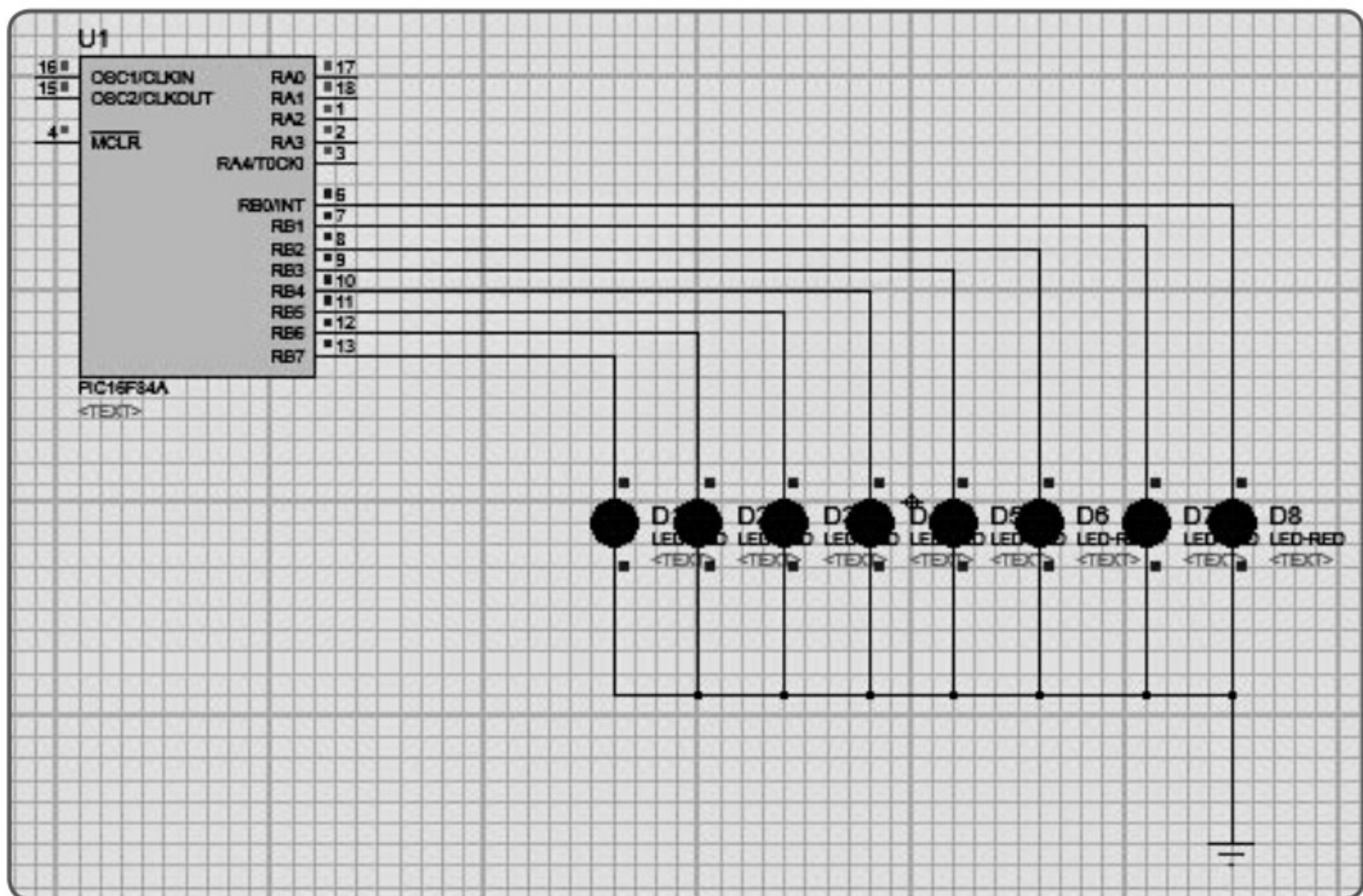
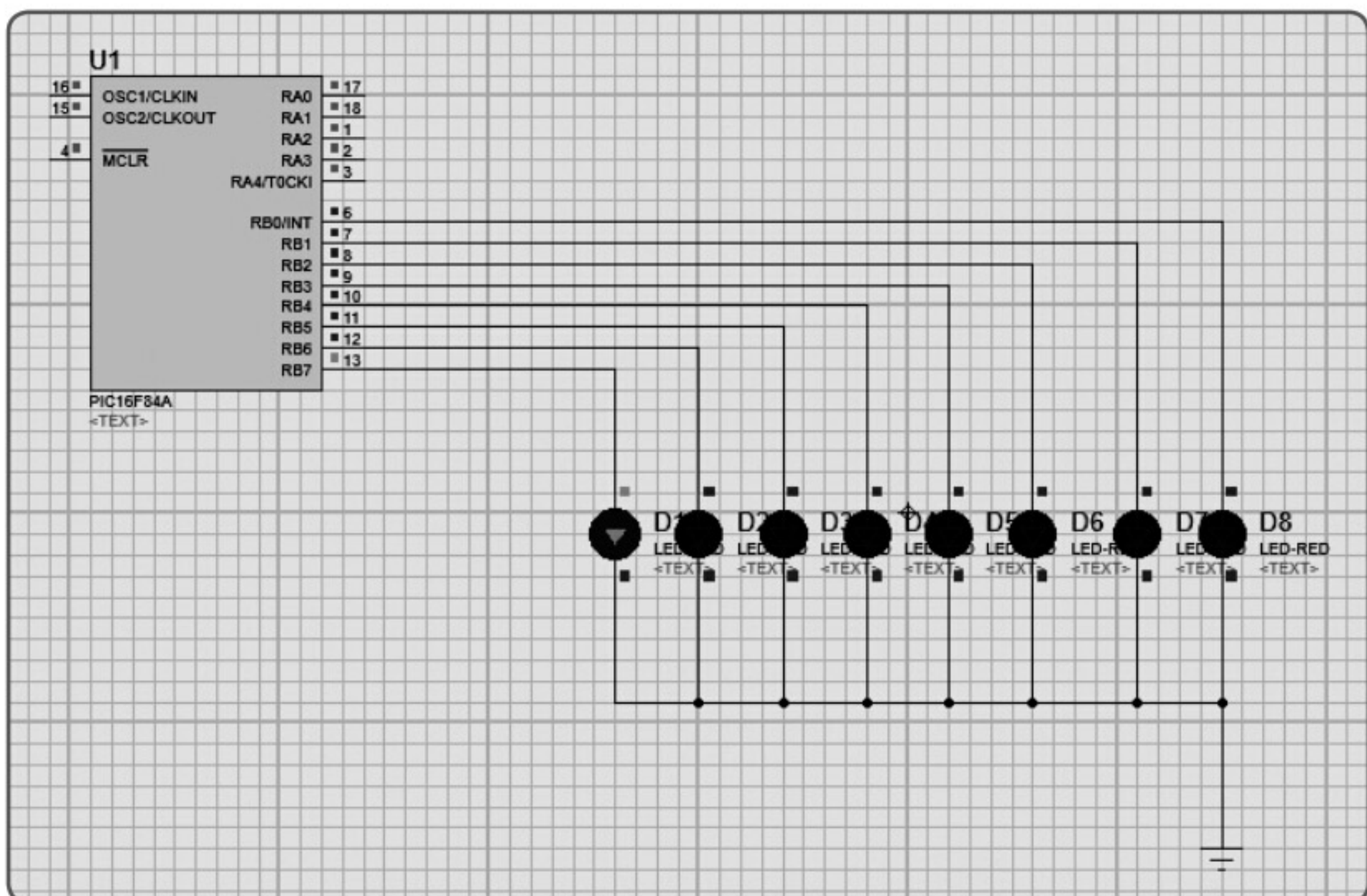


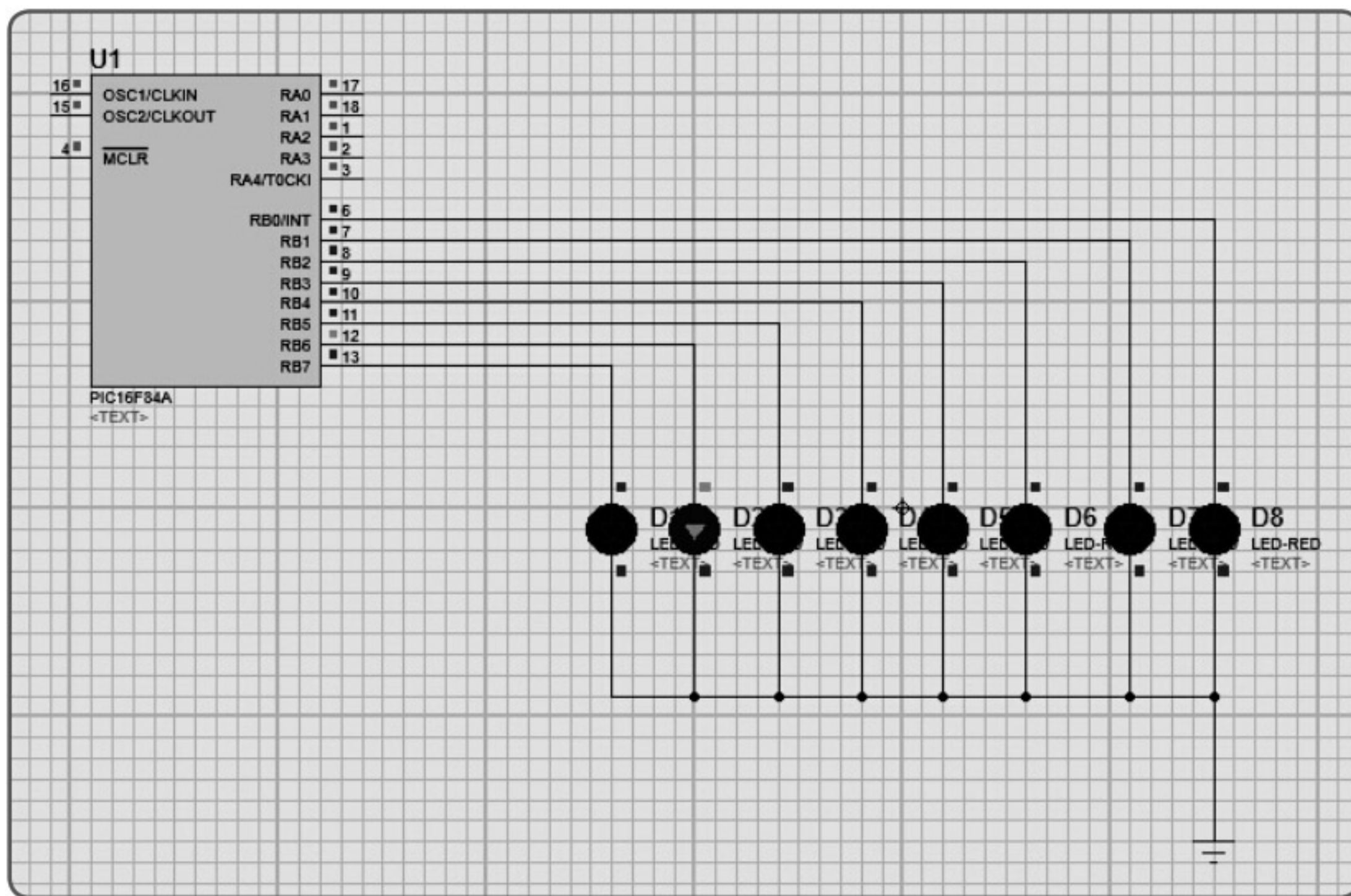
Figura 5.34 PROTEUSLED

## Paso 1

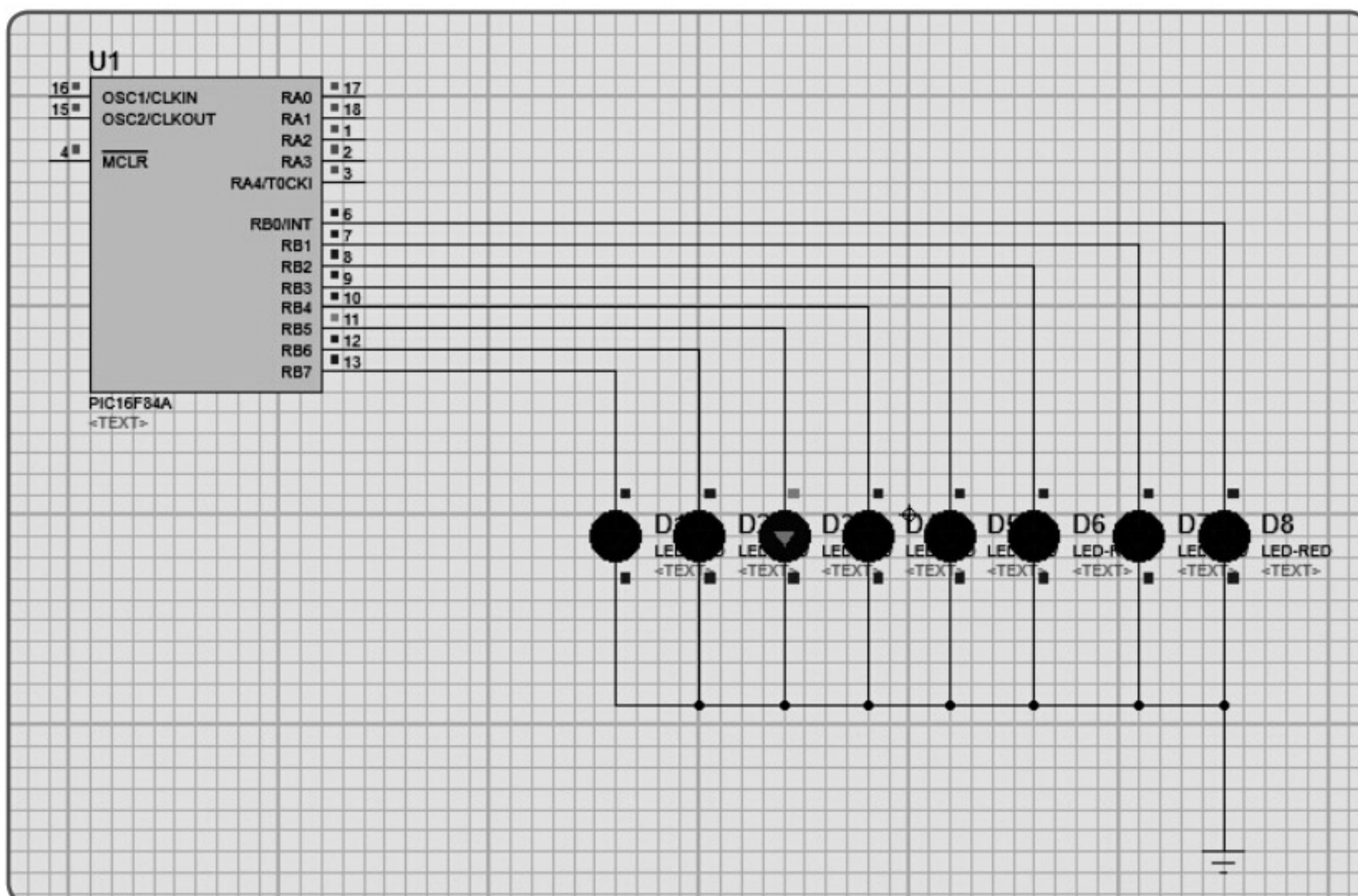




## Paso 2

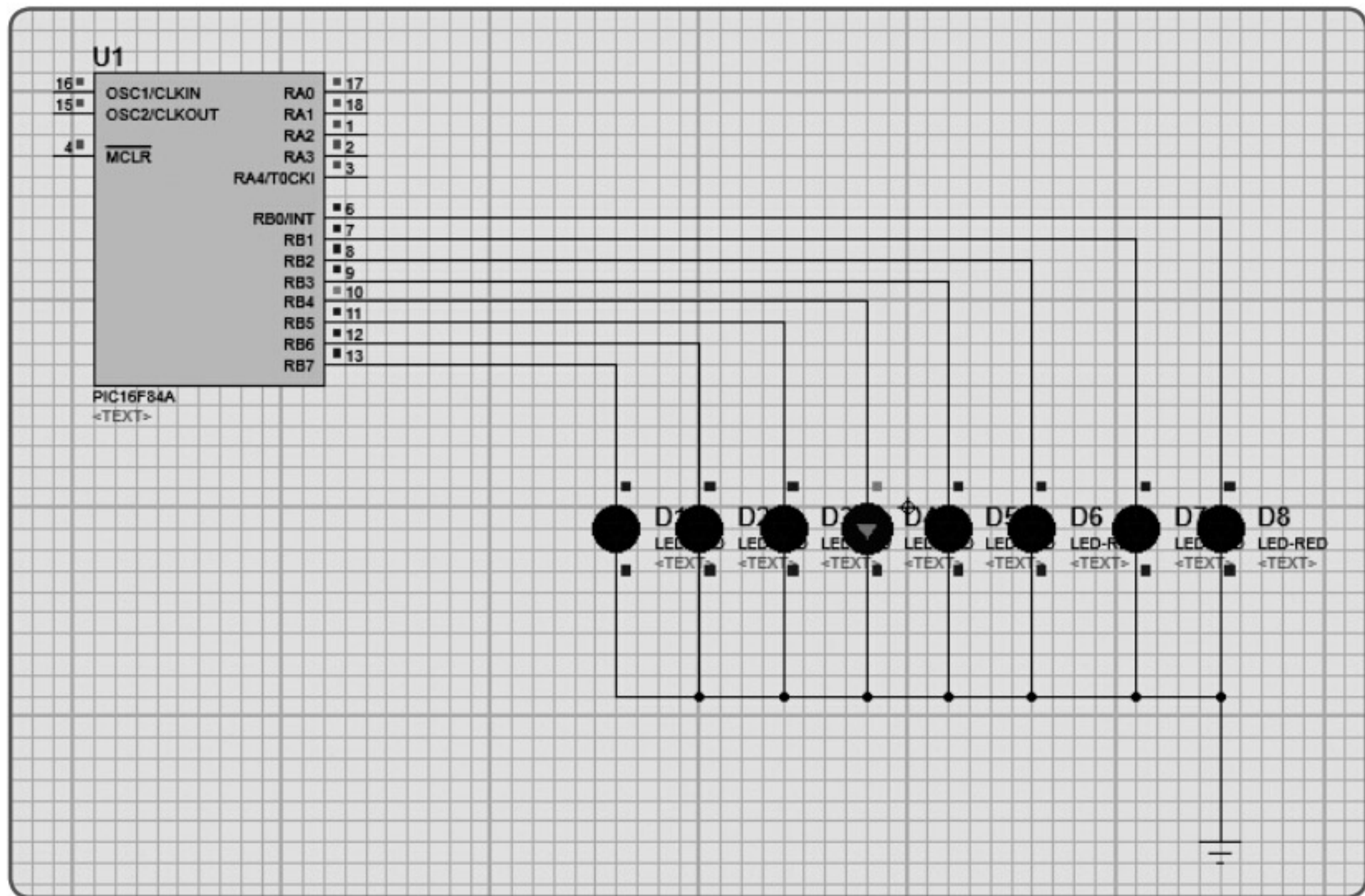


## Paso 3

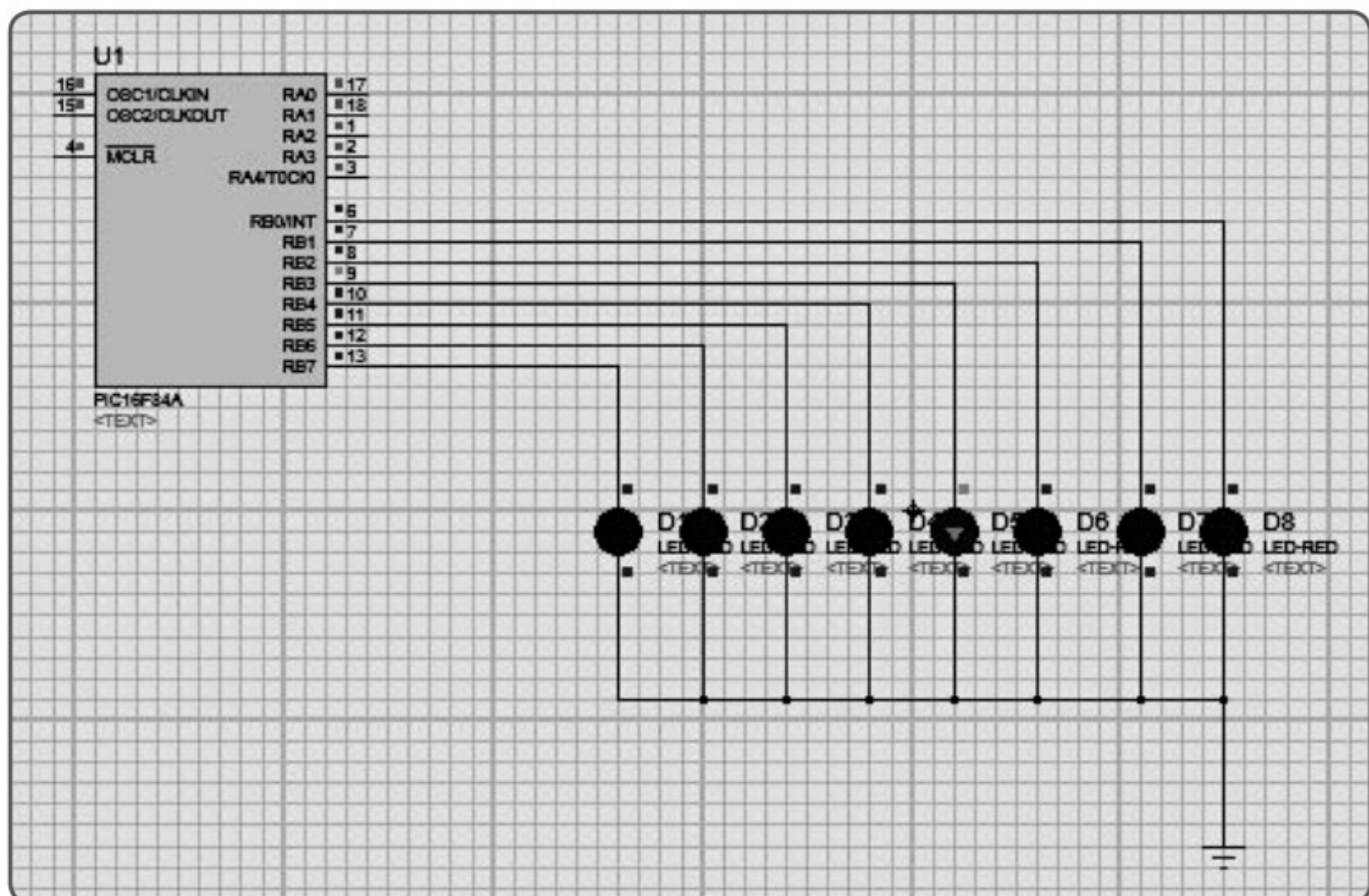




## Paso 4

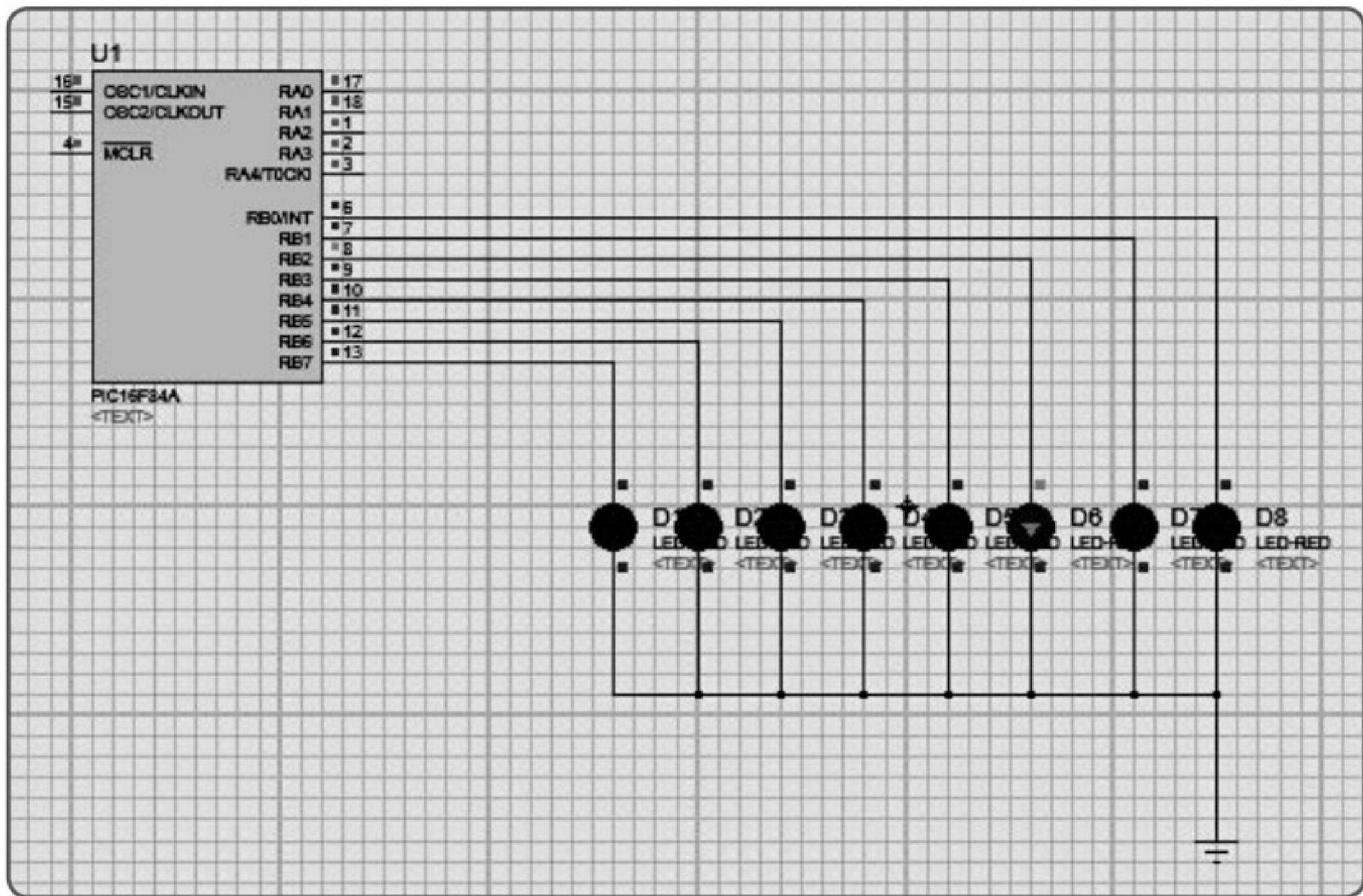


## Paso 5

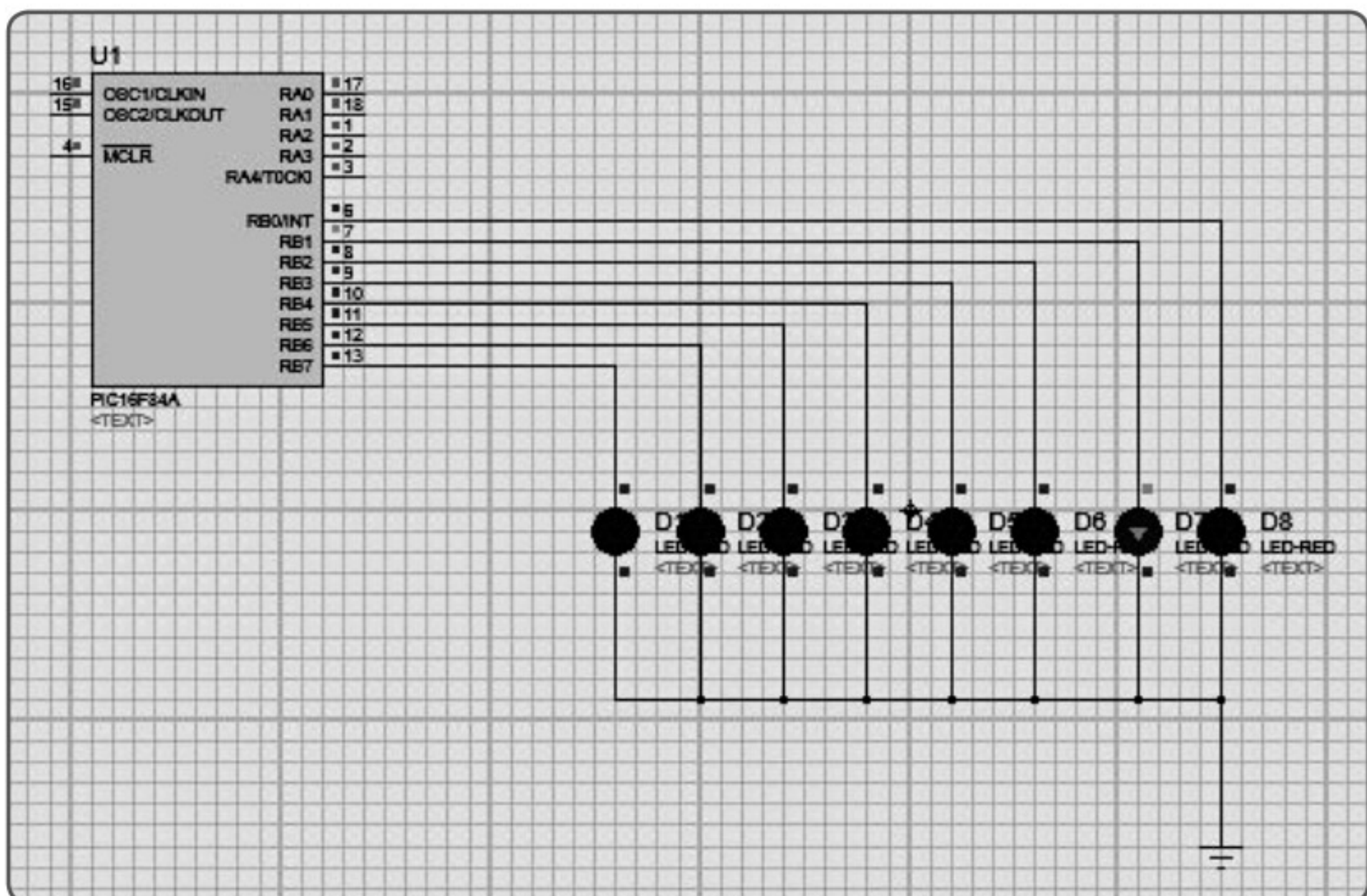




## Paso 6

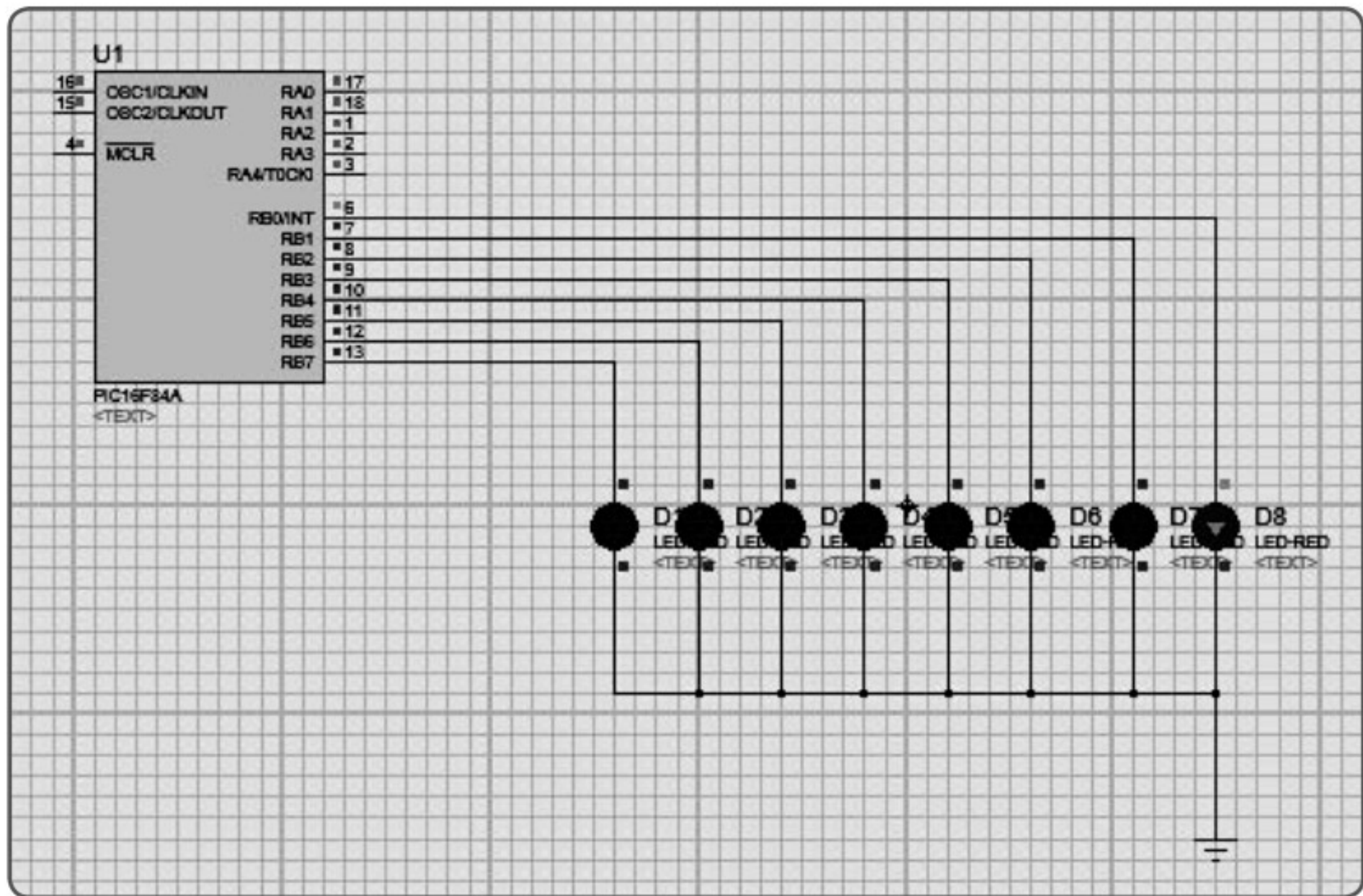


## Paso 7

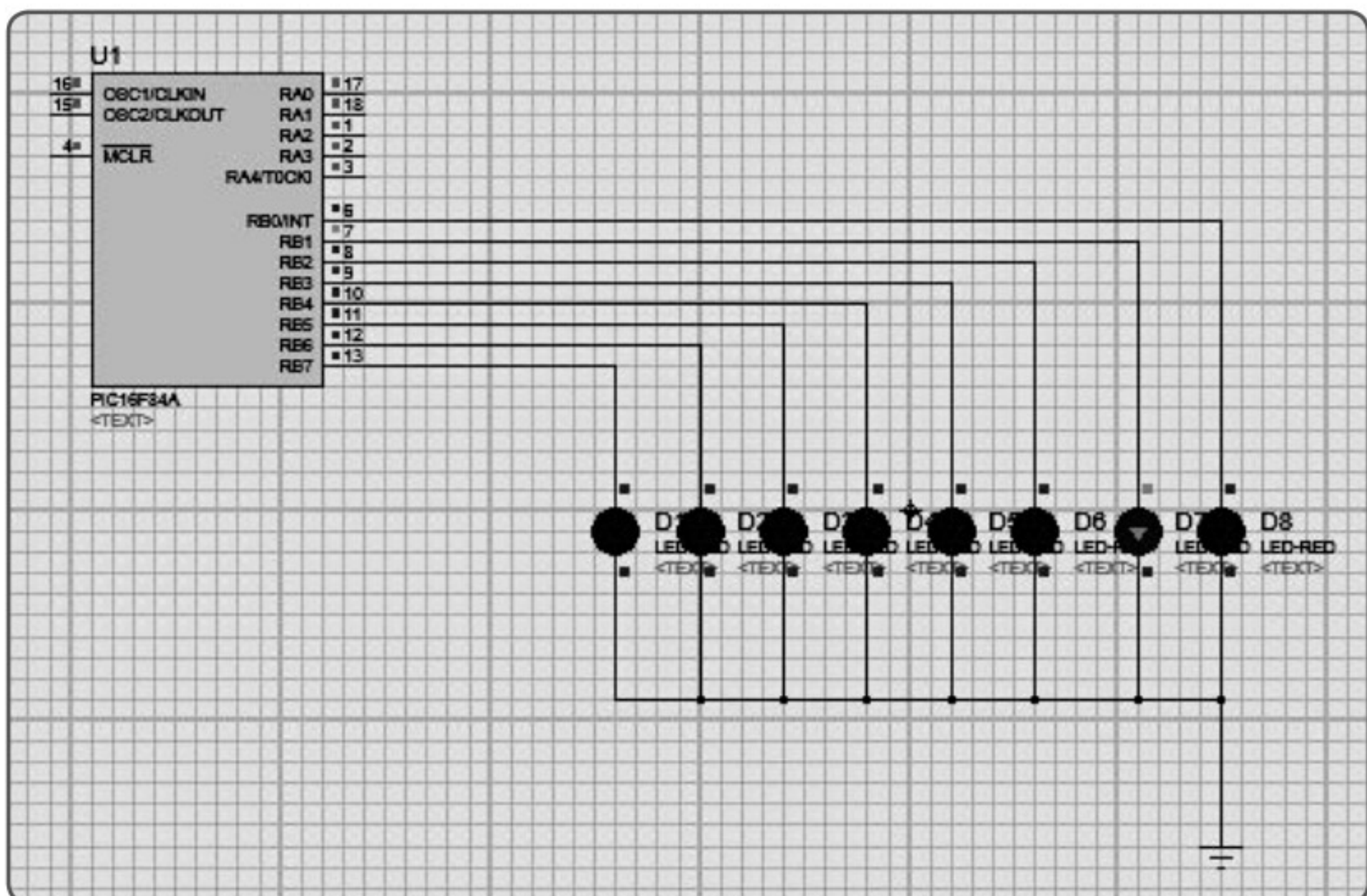




## Paso 8

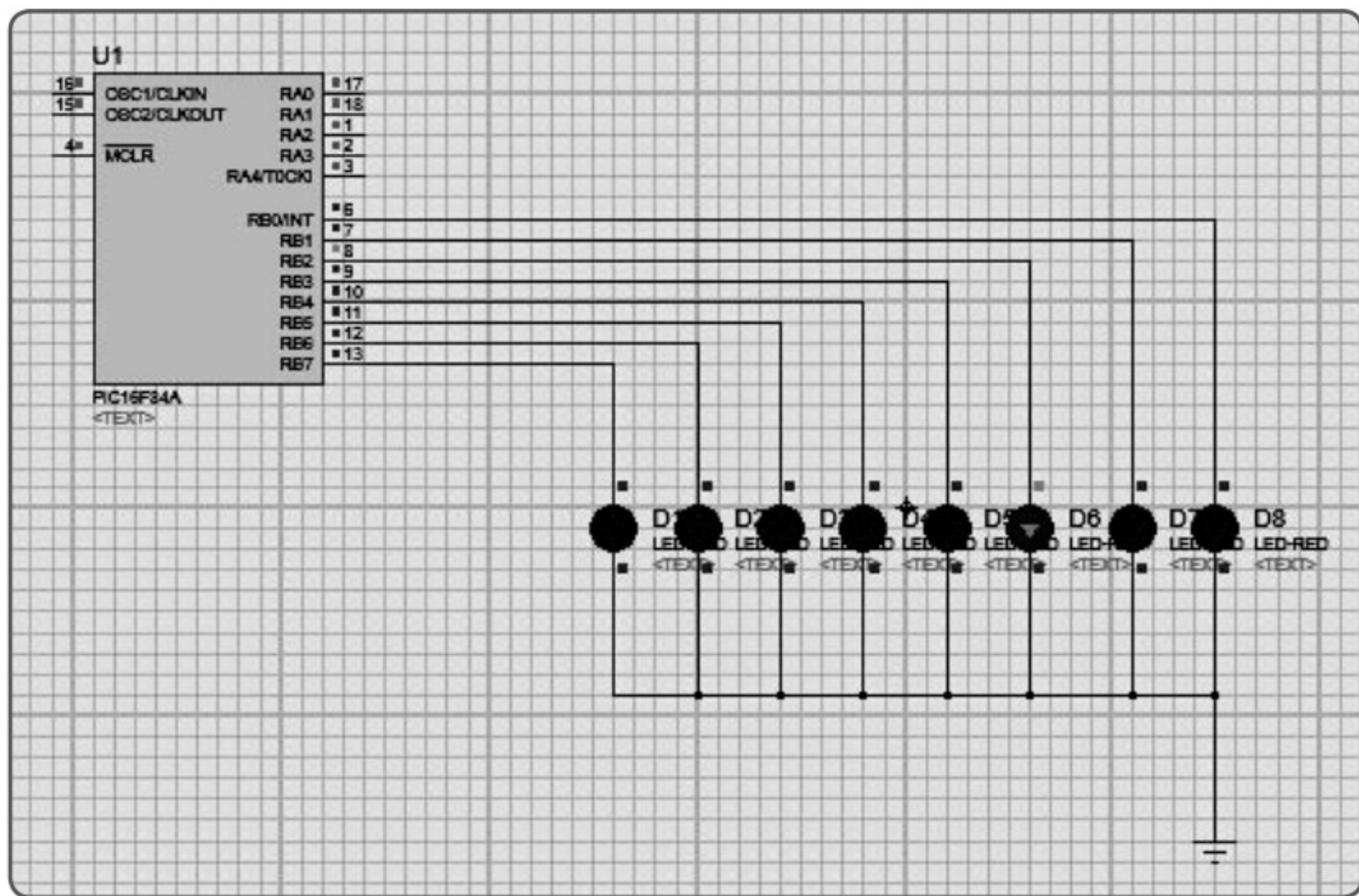


## Paso 9

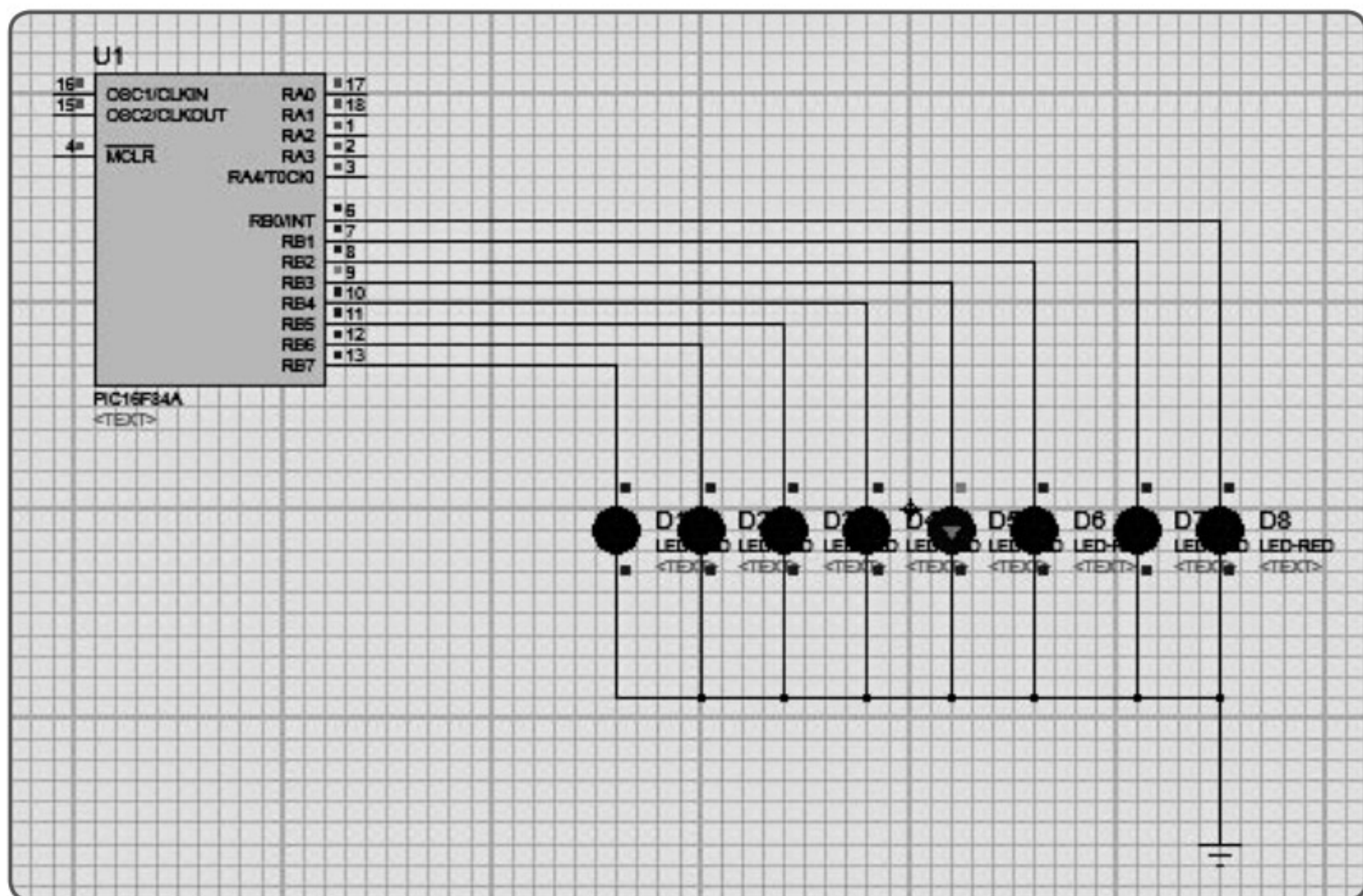




## Paso 10

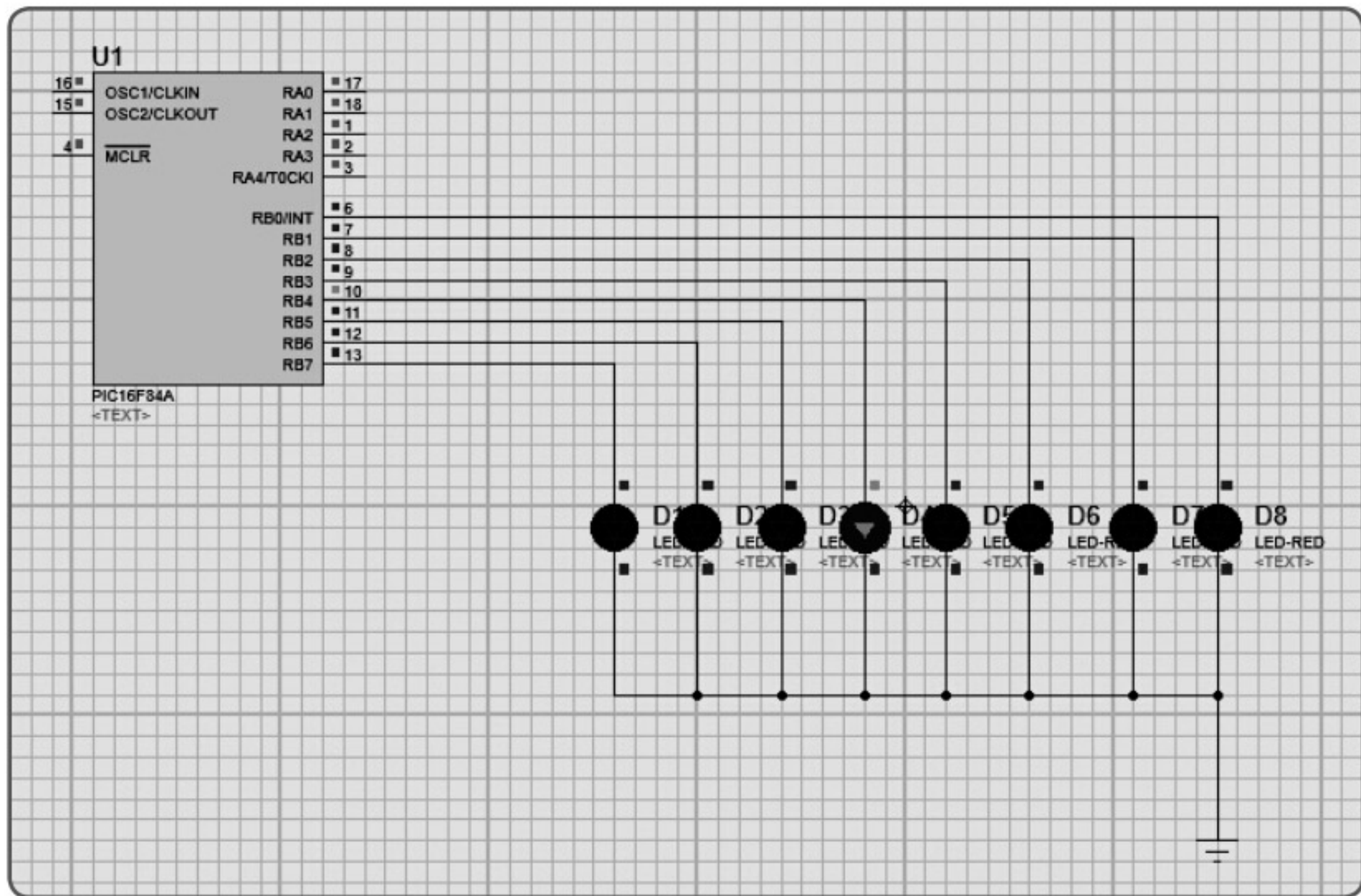


## Paso 11

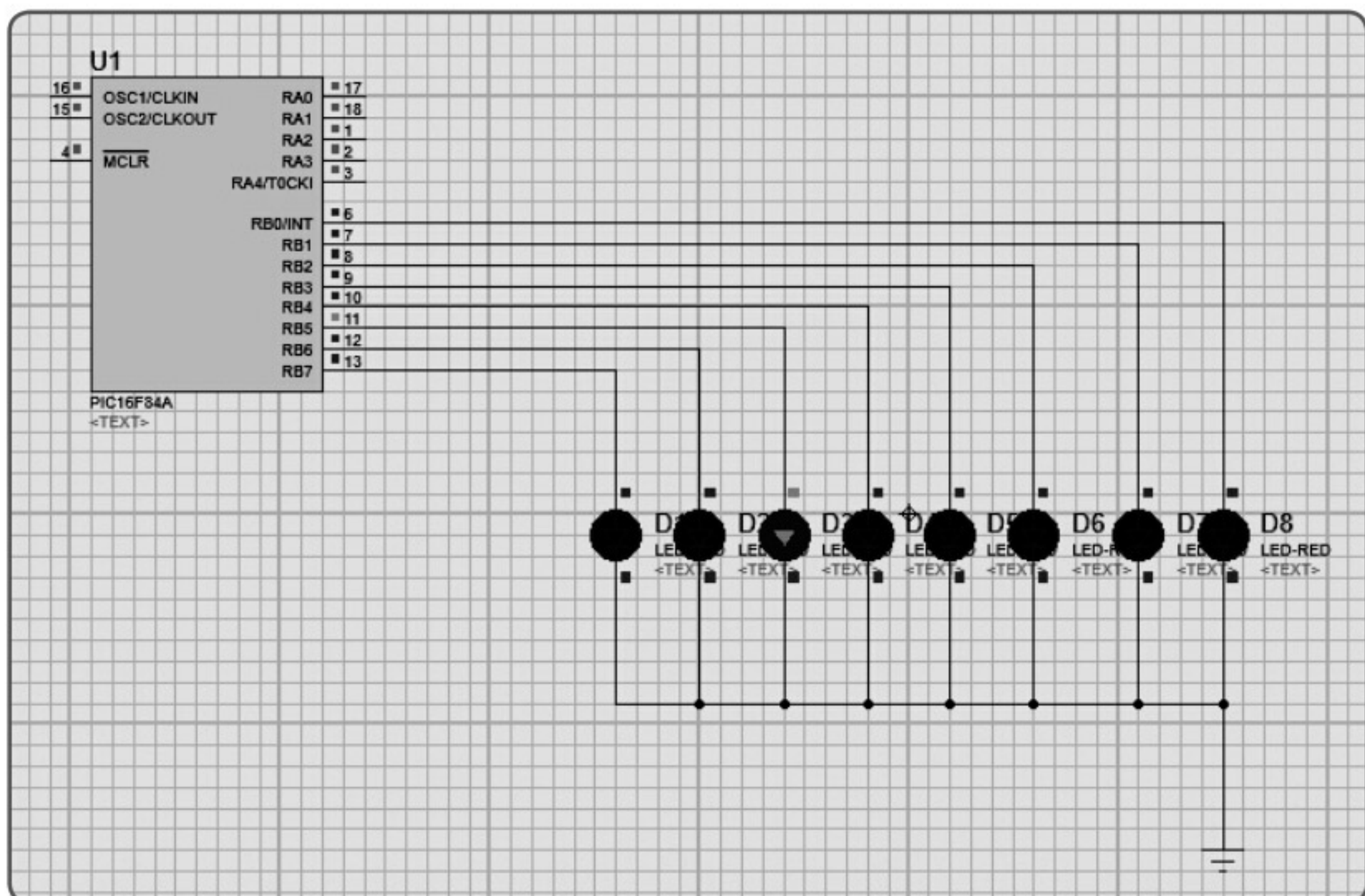




## Paso 12

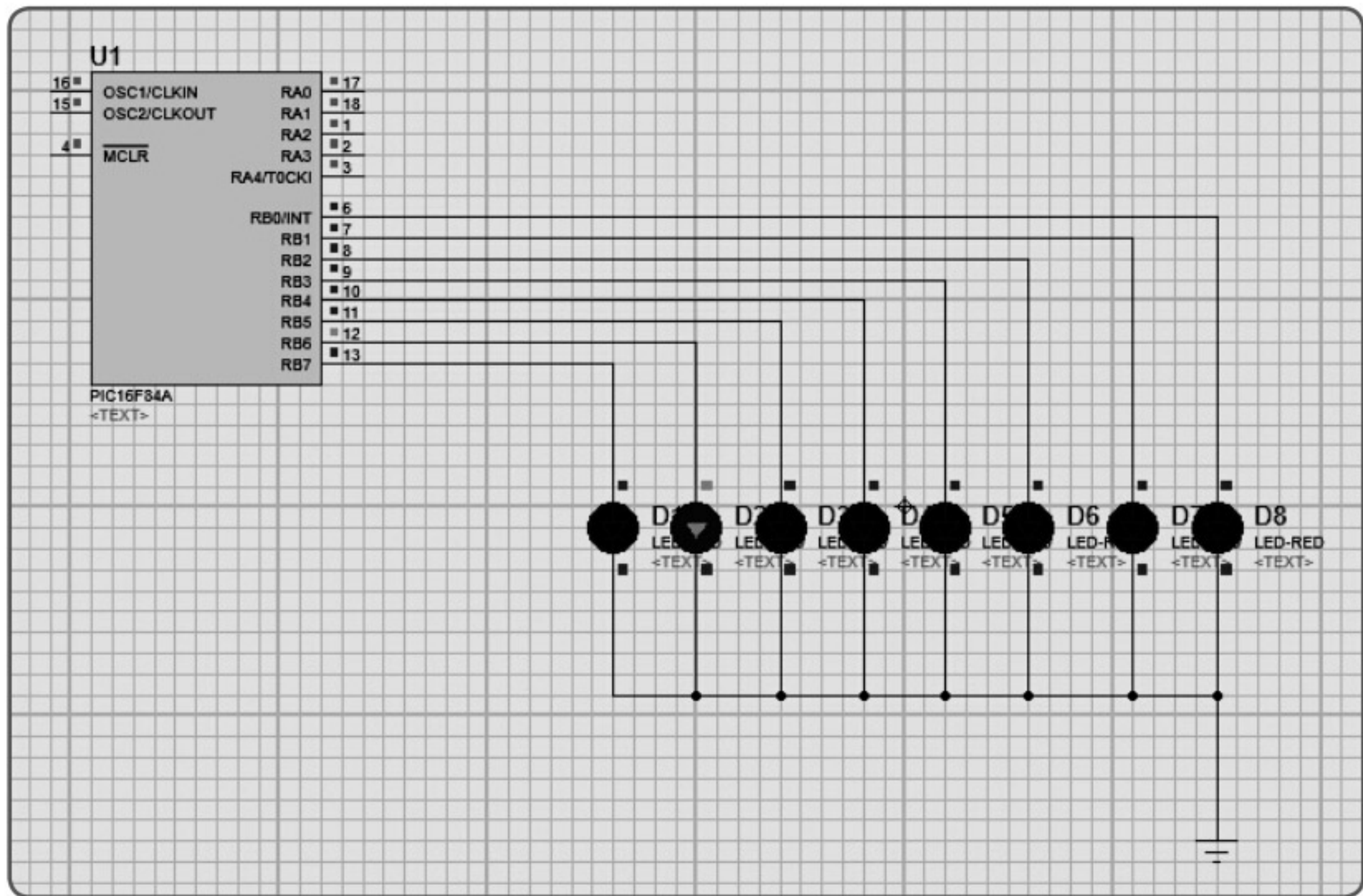


## Paso 13

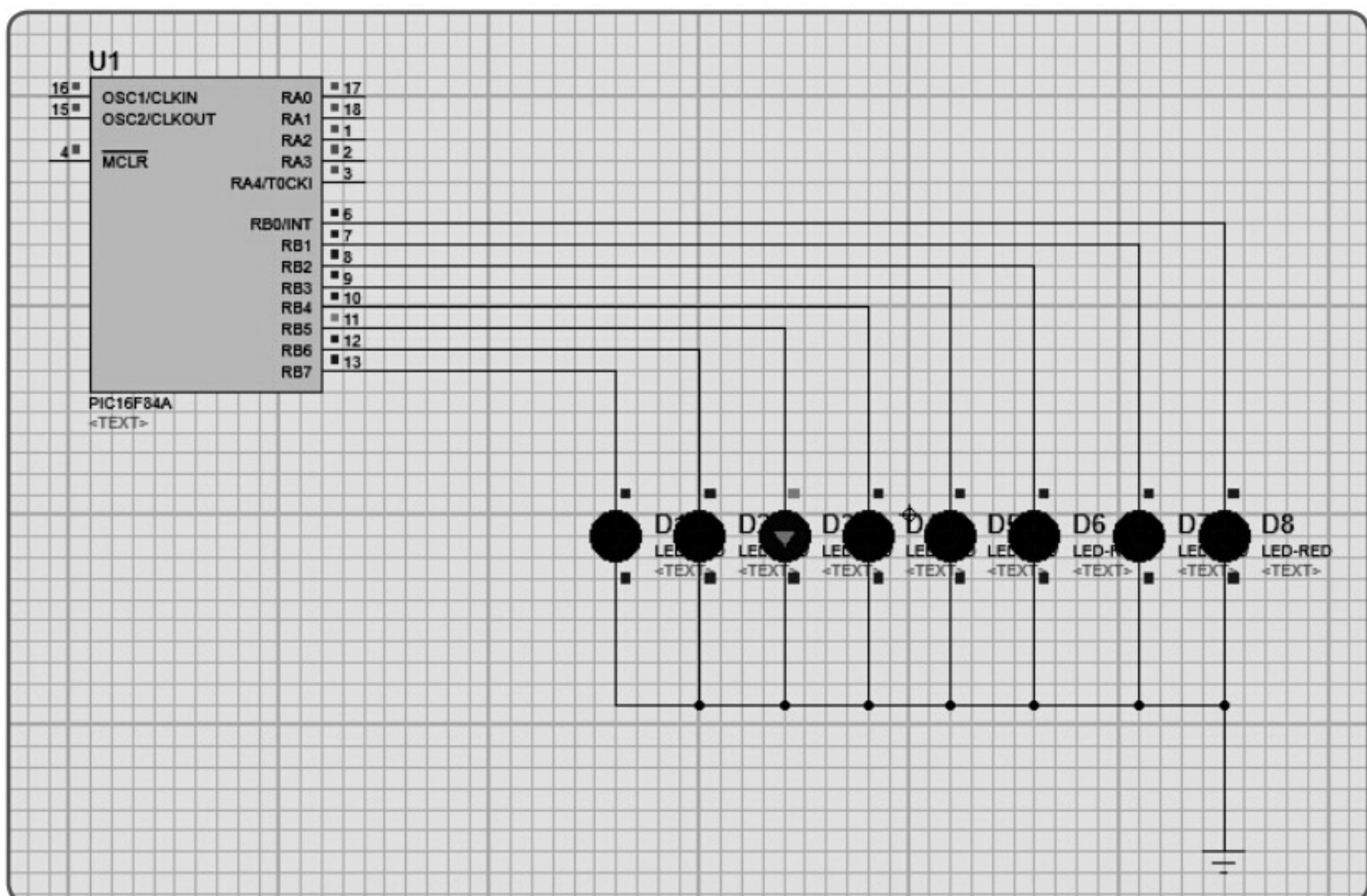




## Paso 14



## Paso 15





## Paso 16

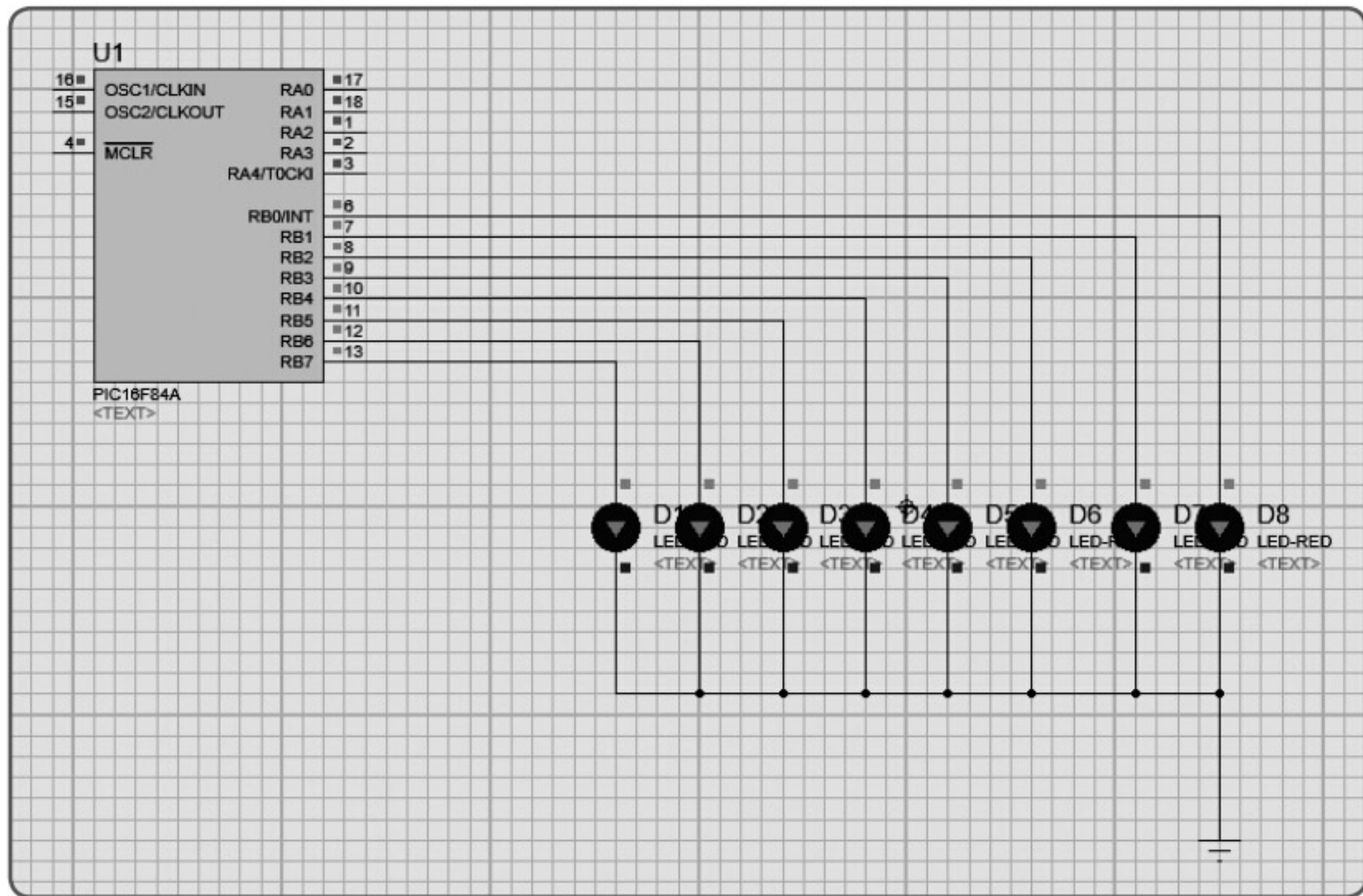


Figura 5.35 Simulación\_Code

## Ejercicio 12

Realizar un contador automático con un display. El contador debe realizar el llenado en

```

;*****
✓ Código en assembler:
;*****
LIST P=16F84A
INCLUDE <P16F84A.INC>
__CONFIG      _CP_OFF &      _WDT_OFF & _PWRTE_ON & _XT_OSC
CBLOCK 0X0C
Contador
ENDC
#define Display PORTB
org 0

BSF STATUS, RP0

```

```
CLRF Display
BCF STATUS, RP0
```

```
INICIO
```

```
MOVLW B'10111111'
MOVWF Display
CALL Retardo_200ms
```

```
MOVLW B'100000110'
MOVWF Display
call Retardo_200ms
```

```
MOVLW B'11011011'
MOVWF Display
CALL Retardo_200ms
```

```
MOVLW B'11001111'
MOVWF Display
CALL Retardo_200ms
```

```
MOVLW B'11100110'
MOVWF Display
CALL Retardo_200ms
```

```
MOVLW B'11101101'
MOVWF Display
CALL Retardo_200ms
```

```
MOVLW B'11111101'
MOVWF Display
CALL Retardo_200ms
```

```
MOVLW B'10000111'
MOVWF Display
CALL Retardo_200ms
```

```
MOVLW B'11111111'
MOVWF Display
CALL Retardo_200ms
```

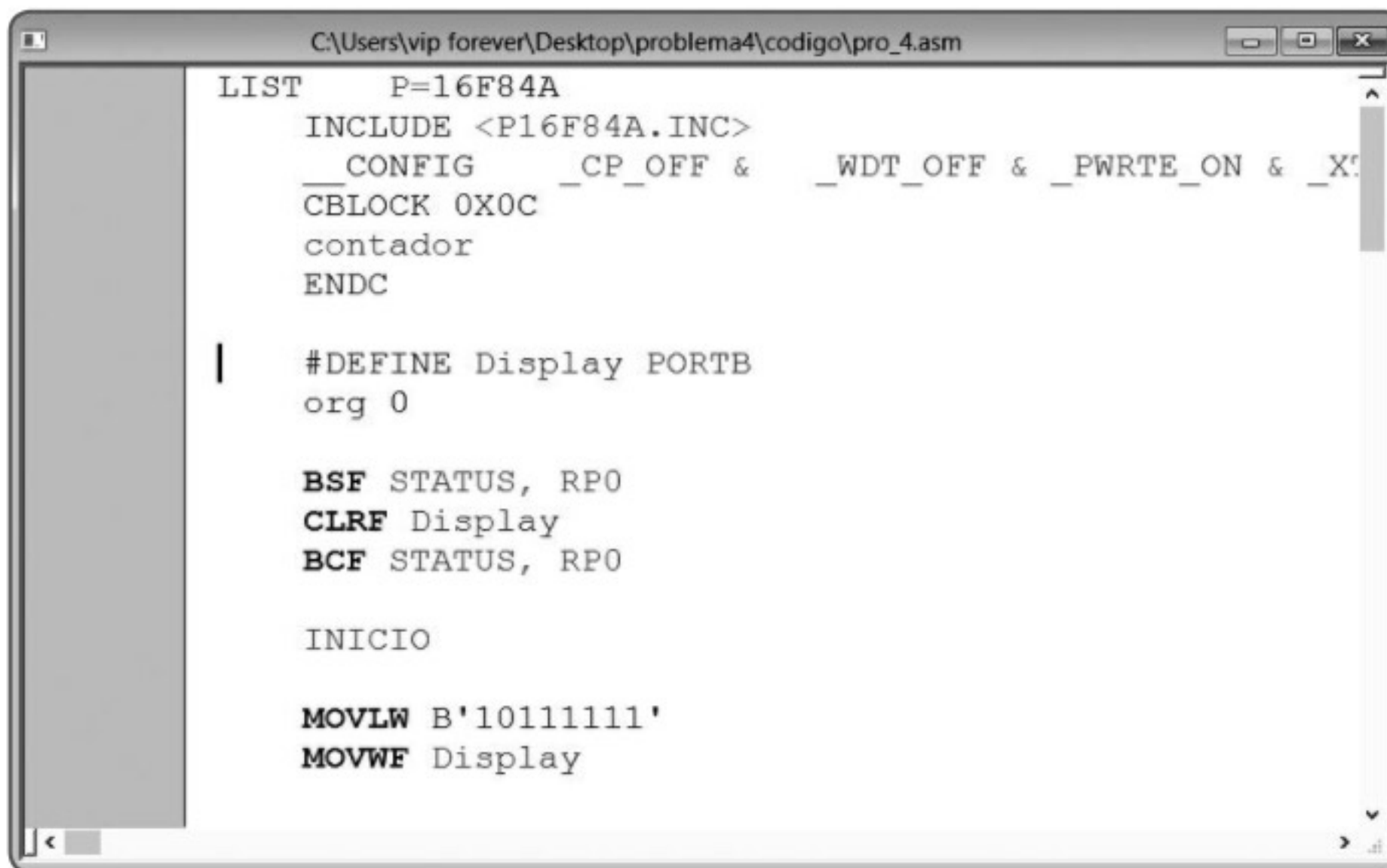
```
MOVLW B'11100111'
MOVWF Display
CALL Retardo_200ms
```

```
INCLUDE<RETARDOS.INC>
END
```



forma creciente con un retardo de 200 milisegundos.

Abra el programa MPLAB. Al hacer clic en **New**, se abre una ventana donde debe escribir los códigos correspondientes para el funcionamiento del programa.



```
LIST      P=16F84A
INCLUDE <P16F84A.INC>
__CONFIG  _CP_OFF & _WDT_OFF & _PWRTE_ON & _X
CBLOCK 0X0C
contador
ENDC

| #DEFINE Display PORTB
org 0

BSF STATUS, RP0
CLRF Display
BCF STATUS, RP0

INICIO

MOVLW B'10111111'
MOVWF Display
```

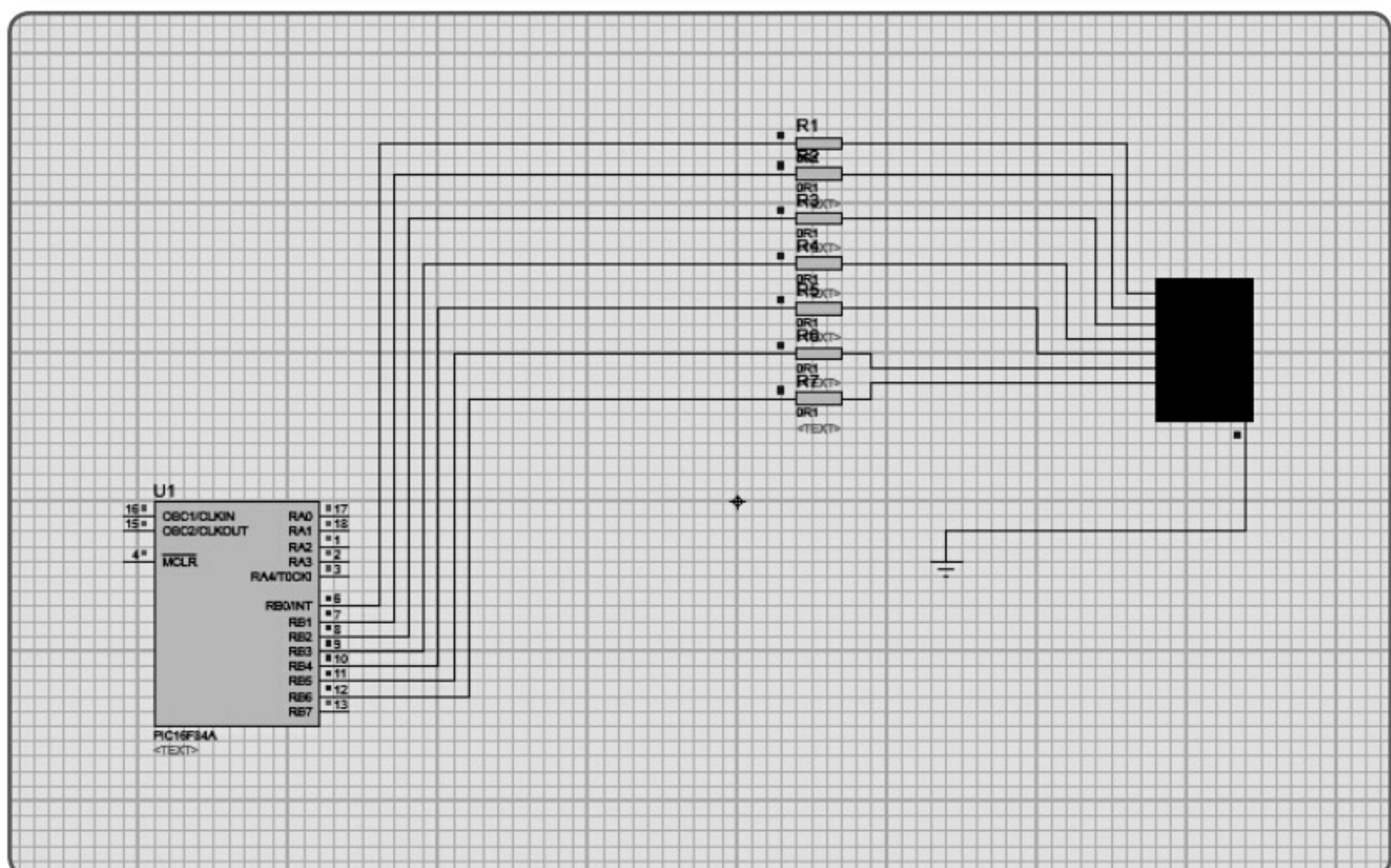
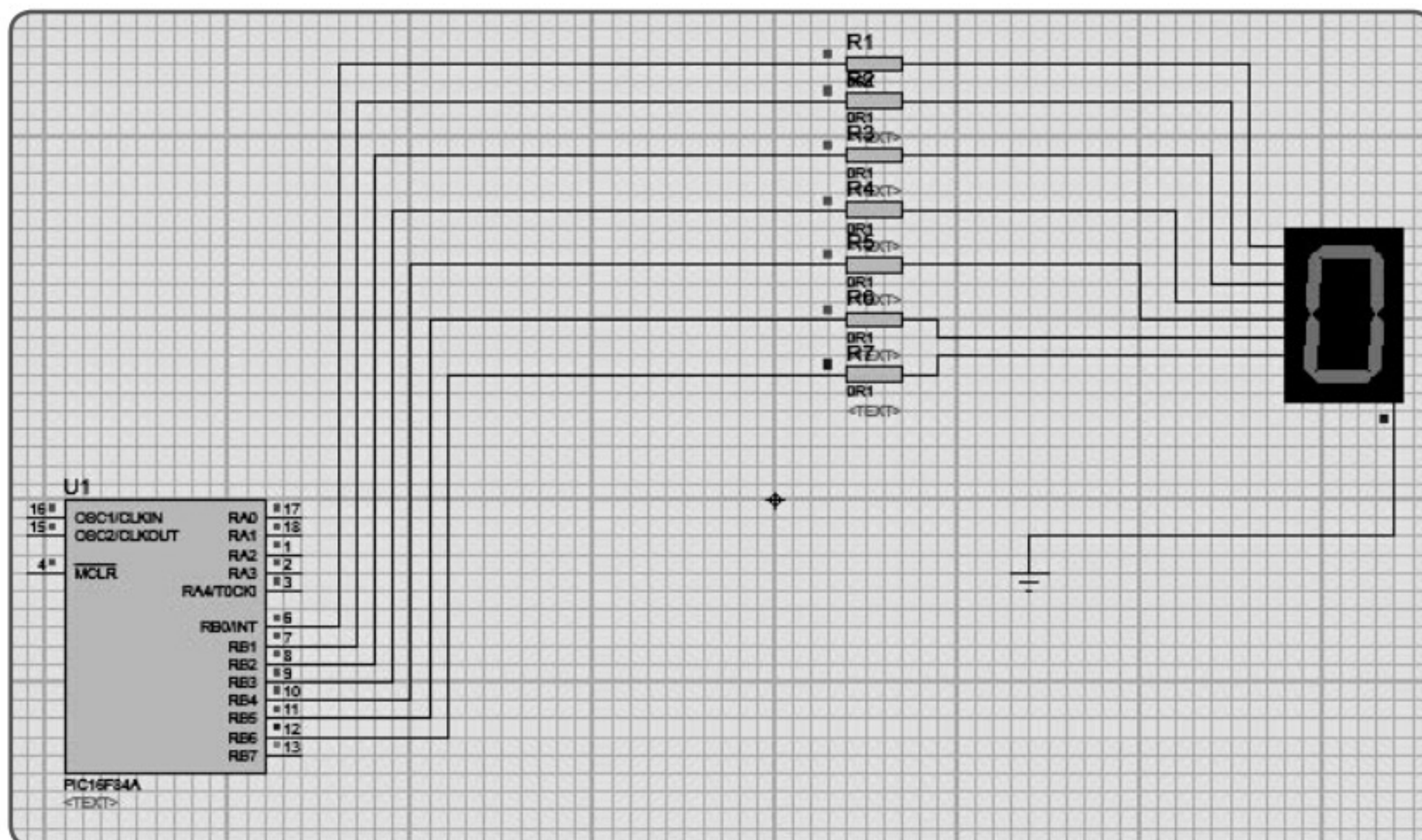


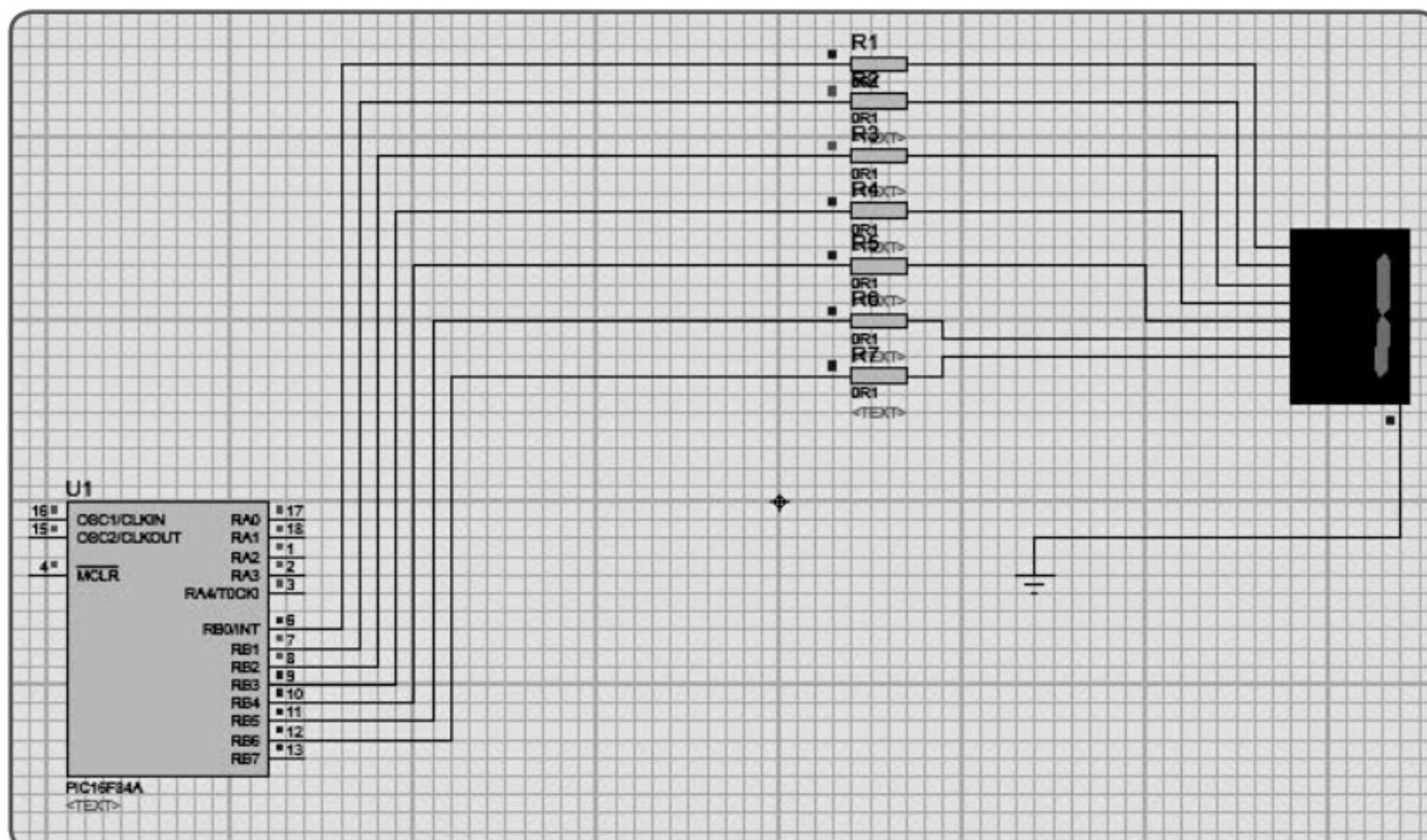
Figura 5.36 Simulación\_Code



## Paso 1

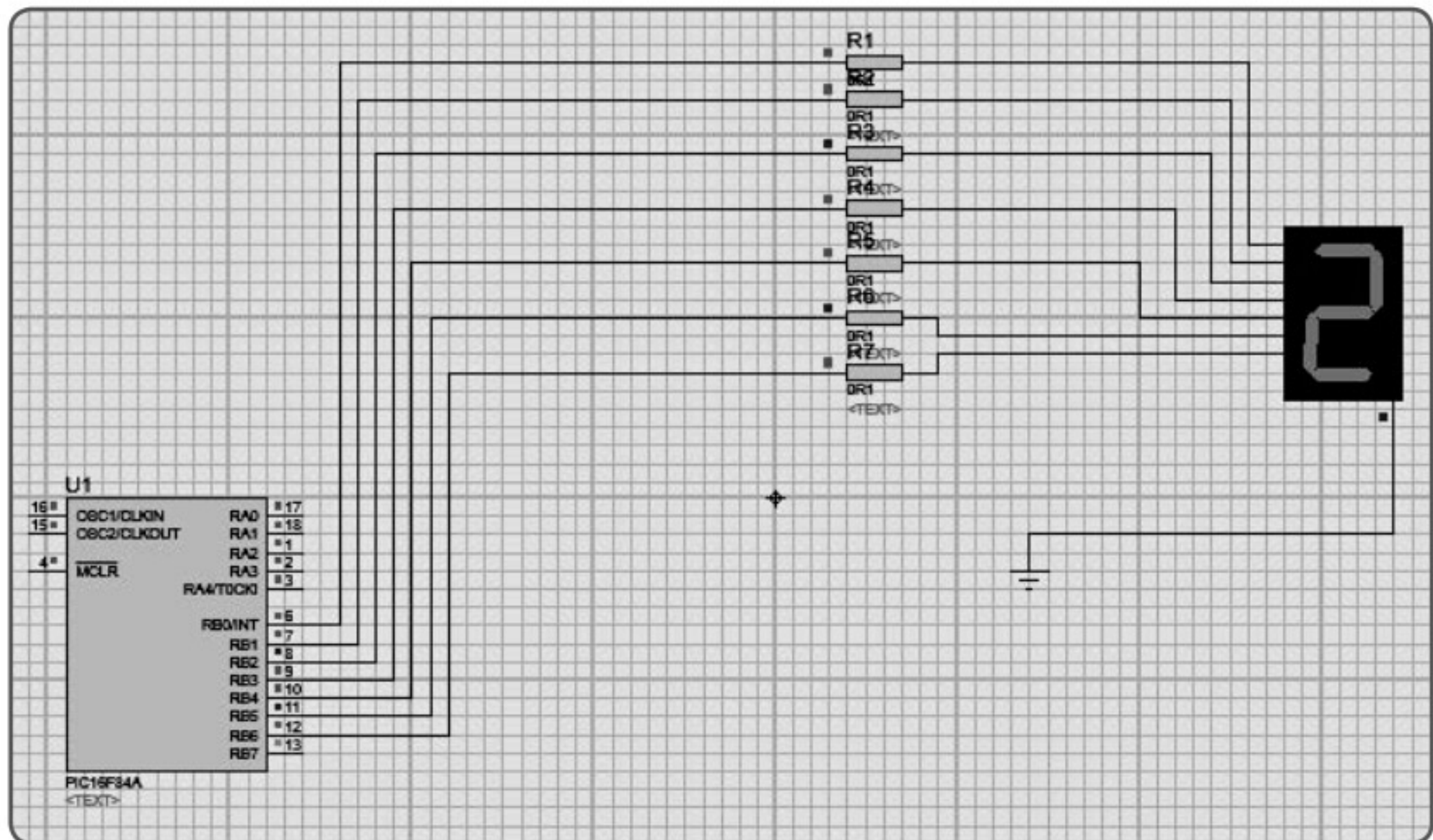


## Paso 2

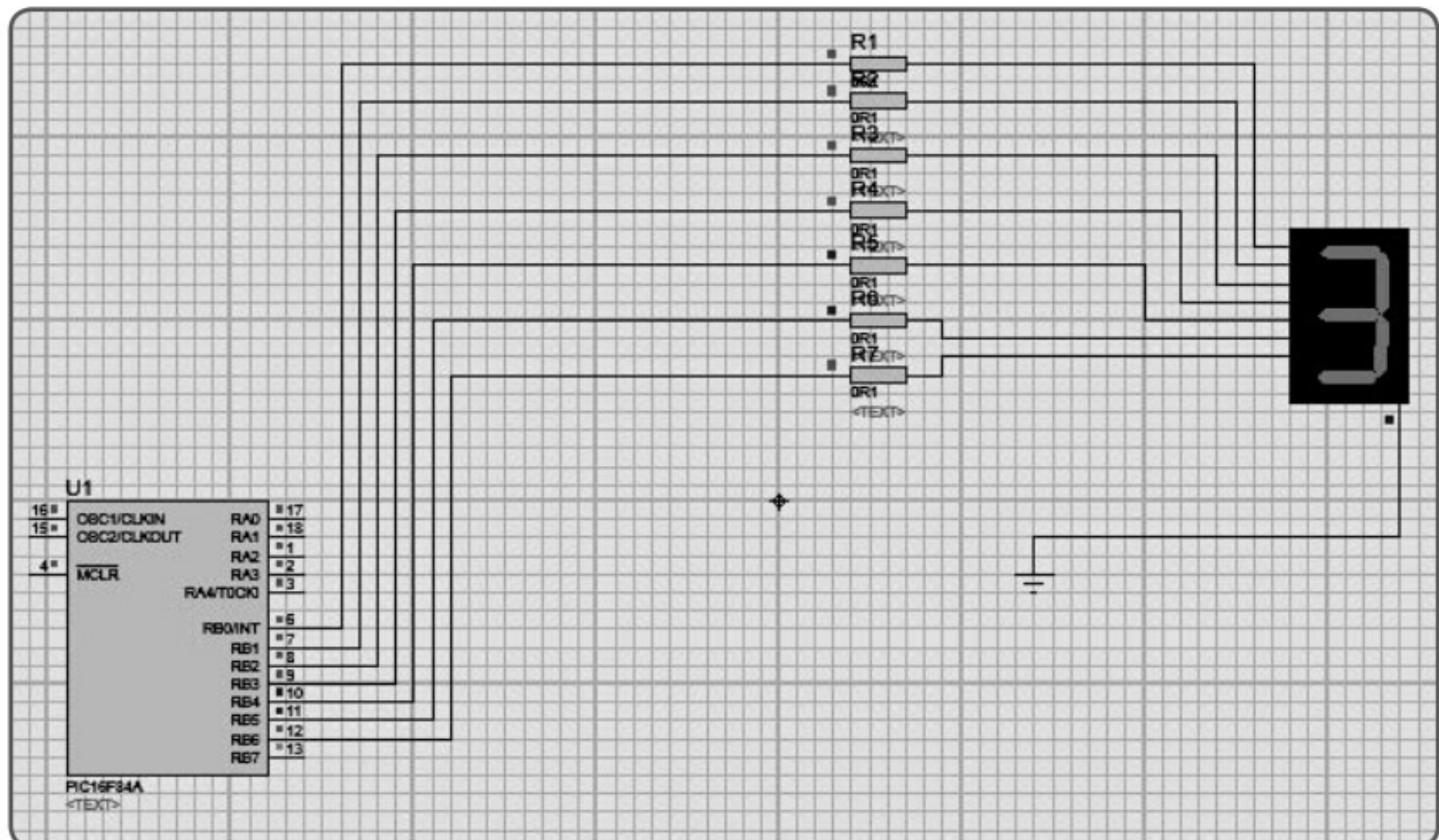




## Paso 3

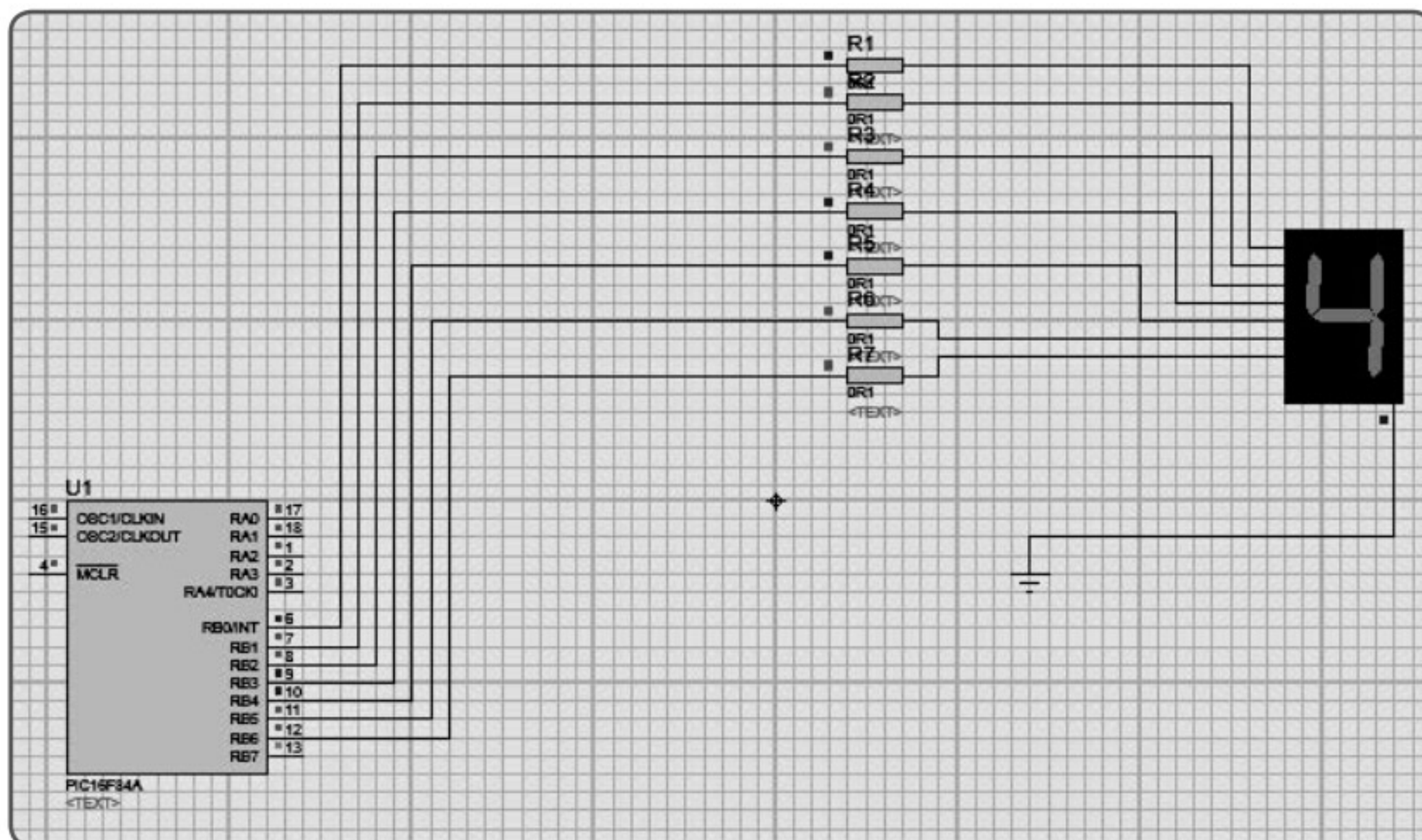


## Paso 4

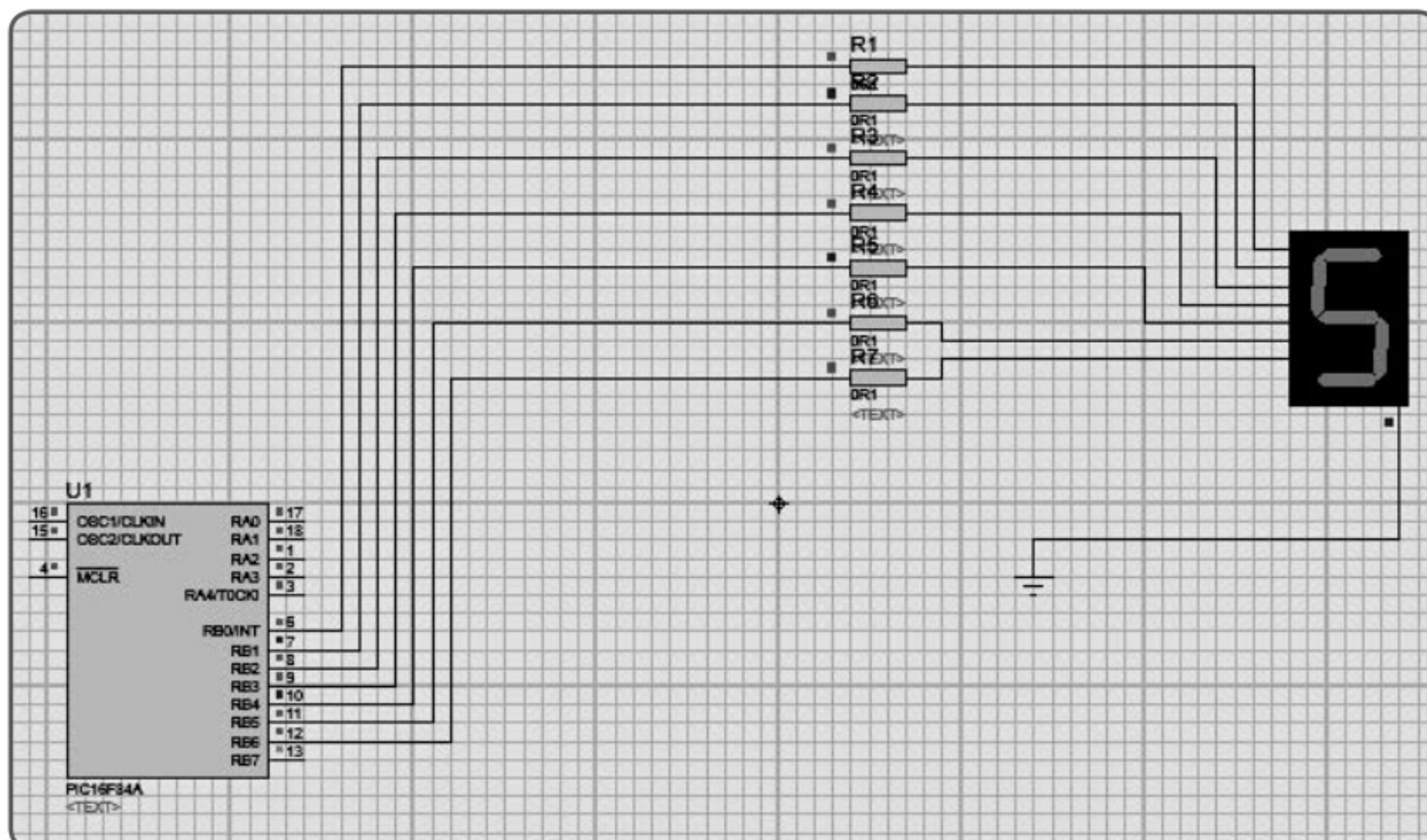




## Paso 5

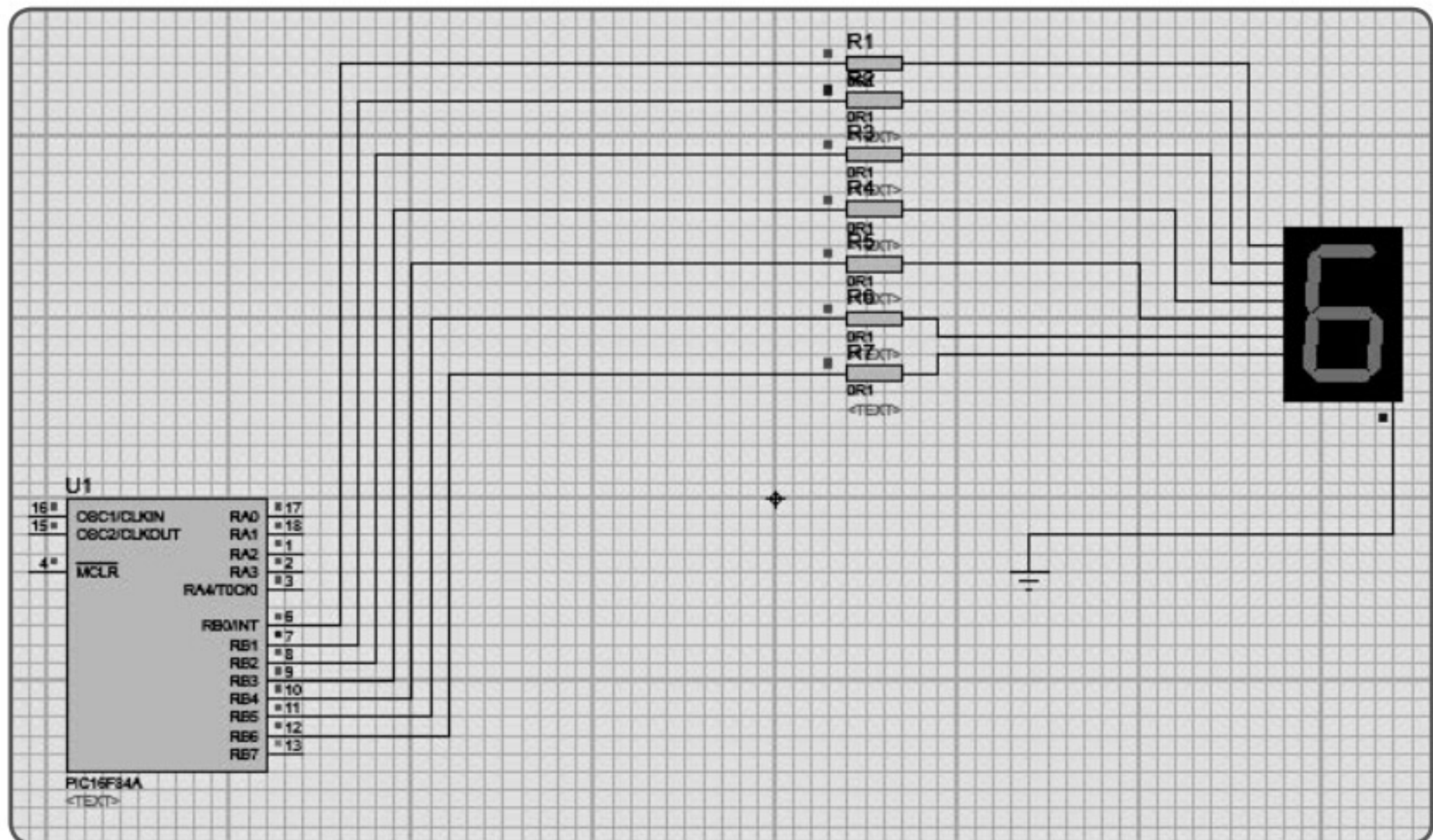


## Paso 6

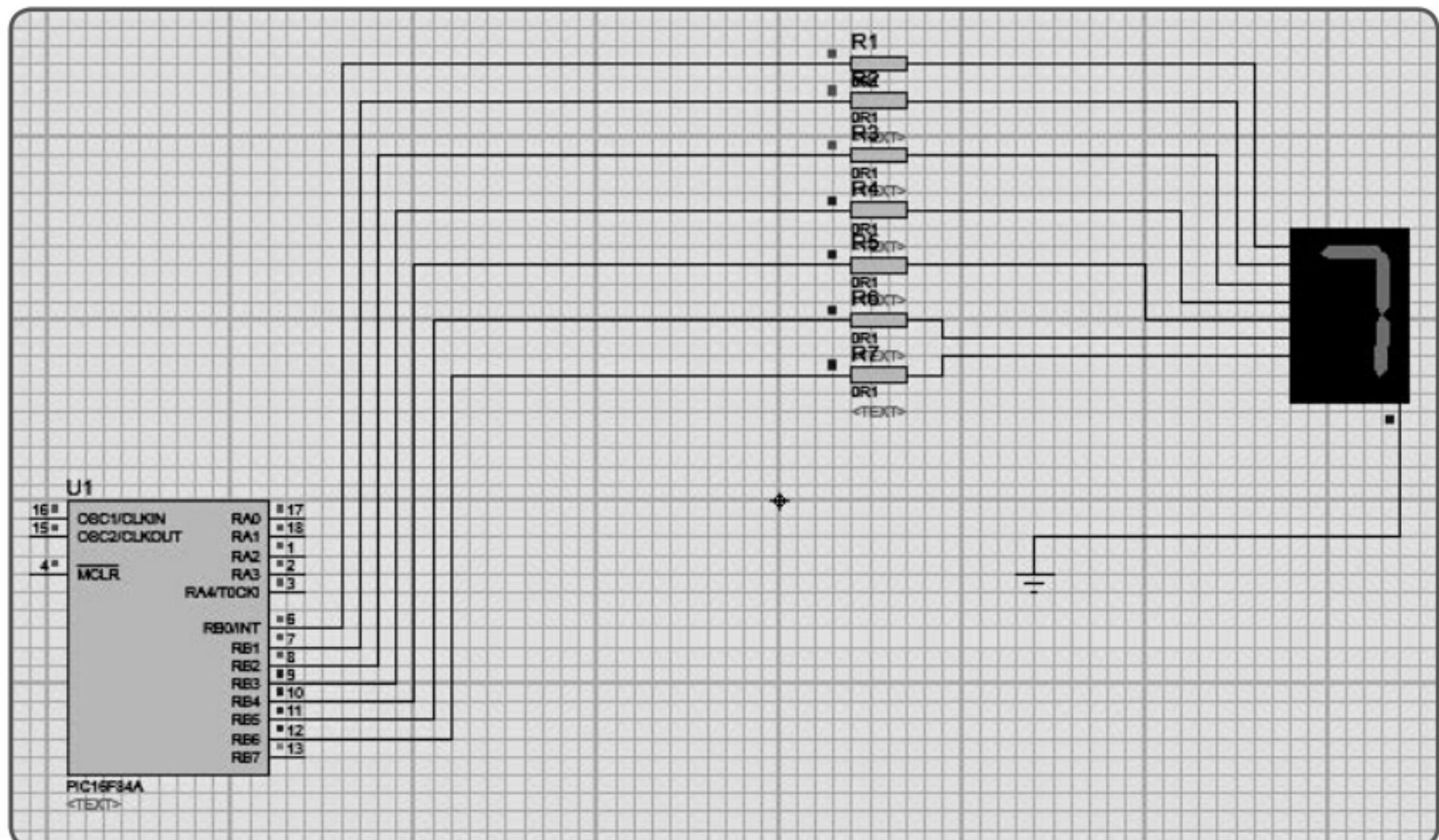




## Paso 7

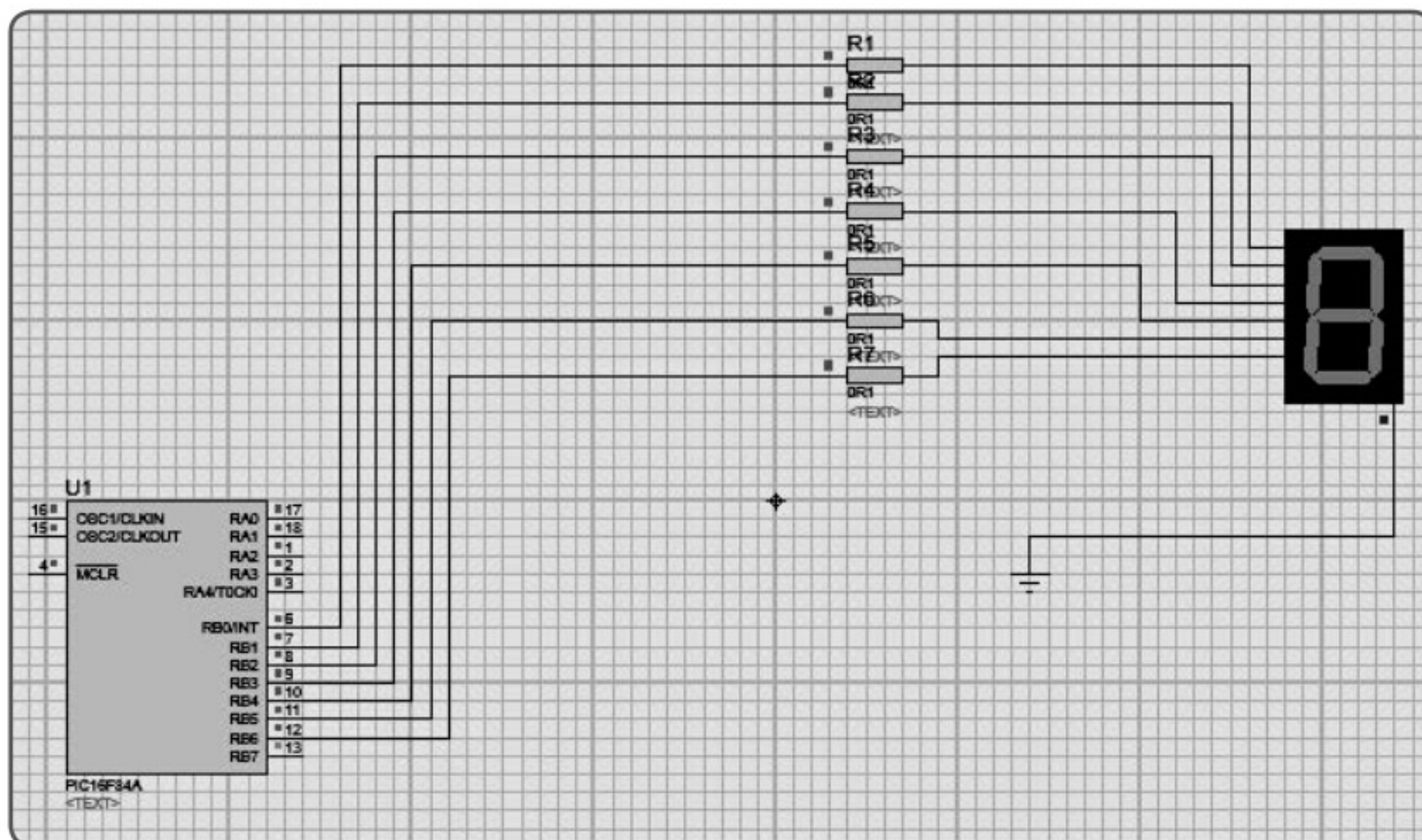


## Paso 8





## Paso 9



## Paso 10

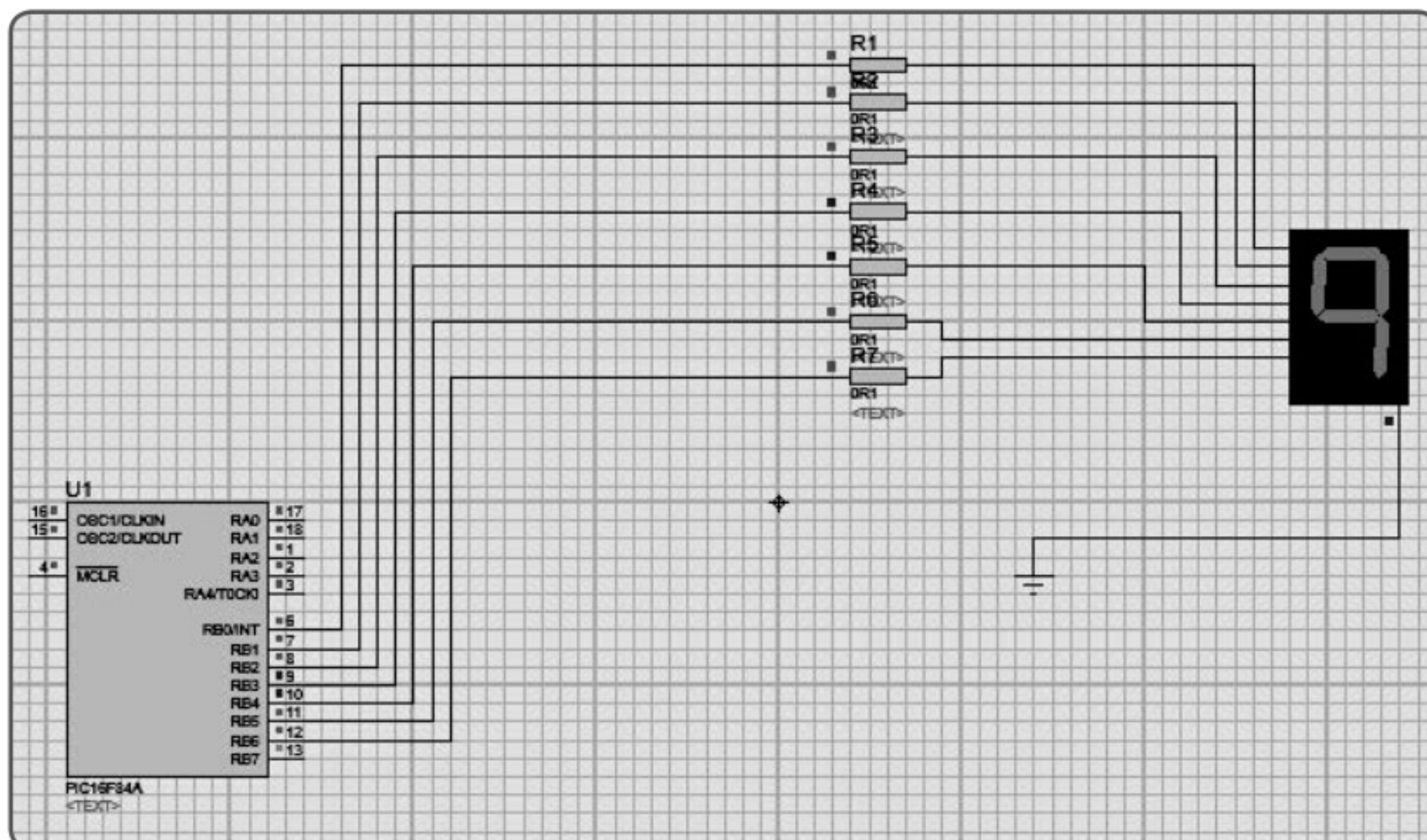


Figura 5.37 DisplayAscendente



**Ejercicio 13**

Realizar un juego de luces cuya secuencia dependa de unos interruptores conectados en el PORTA.

- Si el valor leído es 00001, la secuencia de luces del PORTB será 8 ledes parpadeantes.
- Si el valor es 00010, los ledes realizarán un recorrido de derecha a izquierda.
- Si el valor es 00100, los ledes realizarán una secuencia de izquierda a derecha.

```
;*****
```

```
Código en assembler:
```

```
;*****
```

```
LIST      P=16F887
INCLUDE   P16F887.INC
```

```
CONTA EQU 0X2A
CONTB EQU 0X2B
ORG 0
```

```
BSF STATUS,RP0 ;BANCO 0+1=1
BSF STATUS,RP1 ;BANCO 1+2=3
CLRF ANSEL
BCF STATUS,RP1 ;BANCO 3-2=1
CLRF TRISA
```

```
MOVLW B'11111111'
MOVWF TRISA
```

```
CLRF TRISB
BCF STATUS,RP0 ;BANCO 0 1-1=0
CLRF PORTB
```

```
INICIO
BTFSS PORTA,0
GOTO A ; NO ES 1
;ES 1
MOVLW B'11111111'
MOVWF PORTB
CALL RETARDO
CLRF PORTB
CALL RETARDO
GOTO INICIO
```

```
A
BTFSS PORTA,1
```

```
GOTO G ; NO ES 1
```

```
;ES 1
BSF PORTB,7
CALL RETARDO
```

```
BCF PORTB,7
BSF PORTB,6
CALL RETARDO
```

```
BCF PORTB,6
BSF PORTB,5
CALL RETARDO
```

```
BCF PORTB,5
BSF PORTB,4
CALL RETARDO
```

```
BCF PORTB,4
BSF PORTB,3
CALL RETARDO
```

```
BCF PORTB,3
BSF PORTB,2
CALL RETARDO
```

```
BCF PORTB,2
BSF PORTB,1
CALL RETARDO
```

```
BCF PORTB,1
BSF PORTB,0
CALL RETARDO
```

```
BCF PORTB,0
GOTO INICIO
```

```
G
```

```

BTFSS PORTA,2
GOTO INICIO;NO ES 1
;ES 1

```

```

BSF PORTB,0
CALL RETARDO
BCF PORTB,0
BSF PORTB,1
CALL RETARDO

```

```

BCF PORTB,1
BSF PORTB,2
CALL RETARDO

```

```

BCF PORTB,2
BSF PORTB,3
CALL RETARDO

```

```

BCF PORTB,3
BSF PORTB,4
CALL RETARDO

```

```

BCF PORTB,4
BSF PORTB,5
CALL RETARDO

```

```

BCF PORTB,5

```

```

BSF PORTB,6
CALL RETARDO

```

```

BCF PORTB,6
BSF PORTB,7
CALL RETARDO

```

```

BCF PORTB,7

```

```

GOTO INICIO

```

```

RETARDO
MOVLW d'255'
MOVFW CONTA

```

```

DECA
MOVLW d'255'
MOVFW CONTB
DECFSZ CONTA
GOTO DECB
RETURN
DECB
DECFSZ CONTB
GOTO DECB
GOTO DECA

```

```

END

```

```

C:\Users\vip forever\Desktop\pro_7\codigo\c_7.asm
LIST          P=16F887
INCLUDE       P16F887.INC

CONTA EQU 0X2A
CONTB EQU 0X2B

ORG 0

BSF STATUS,RP0 ;BANCO 0+1=1
BSF STATUS,RP1 ;BANCO 1+2=3
CLRF ANSEL
BCF STATUS,RP1 ;BANCO 3-2=1
CLRF TRISA
MOVLW B'11111111'
MOVWF TRISA
CLRF TRISB
BCF STATUS,RP0 ;BANCO 1-1=0

CLRF PORTB

```

Figura 5.38 MPLABPORTA



Abra el programa MPLAB. Al hacer clic en **New**, se abre una ventana donde debe escribir los códigos correspondientes para el funcionamiento del programa.



Figura 5.39 Build

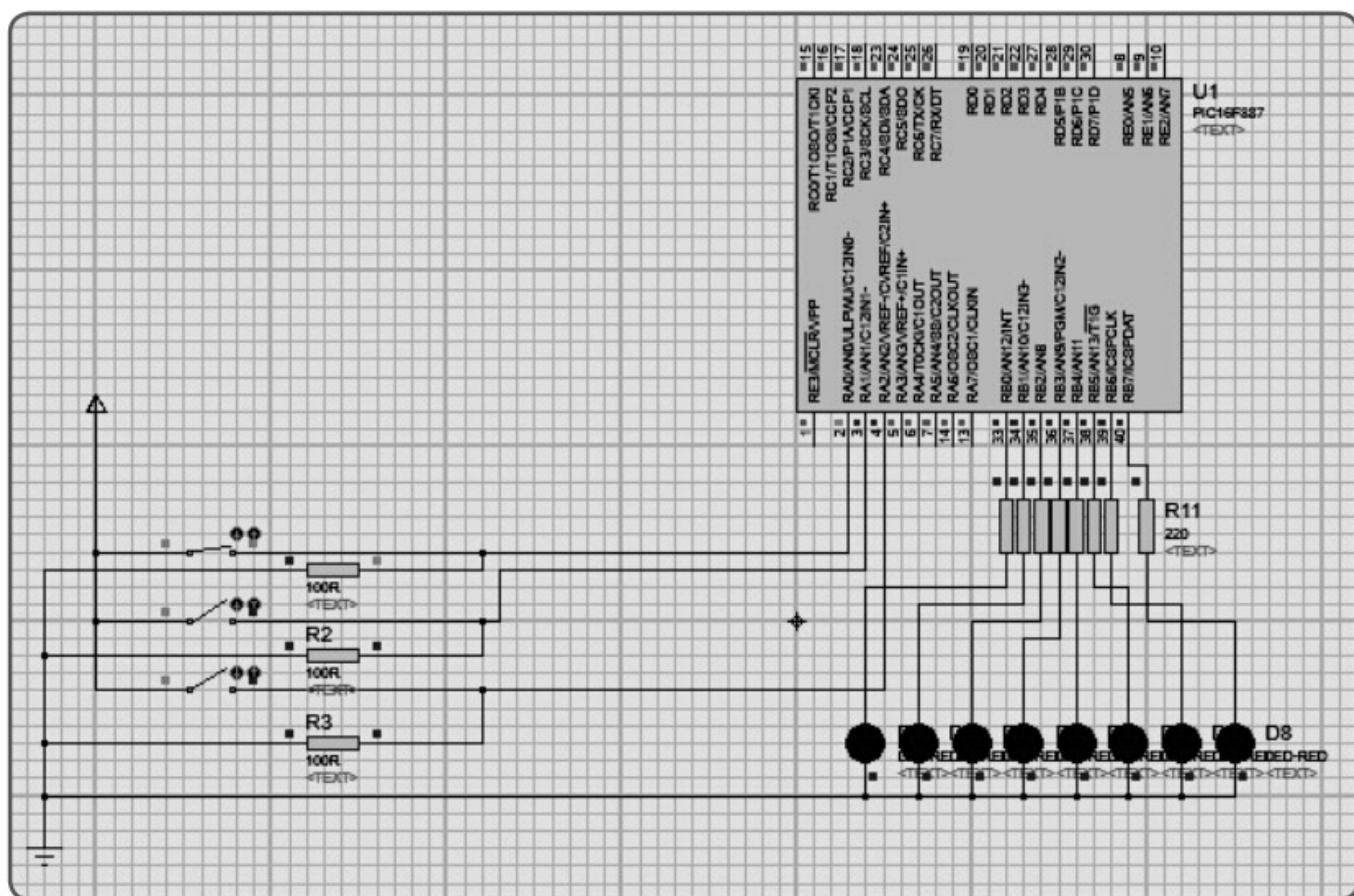
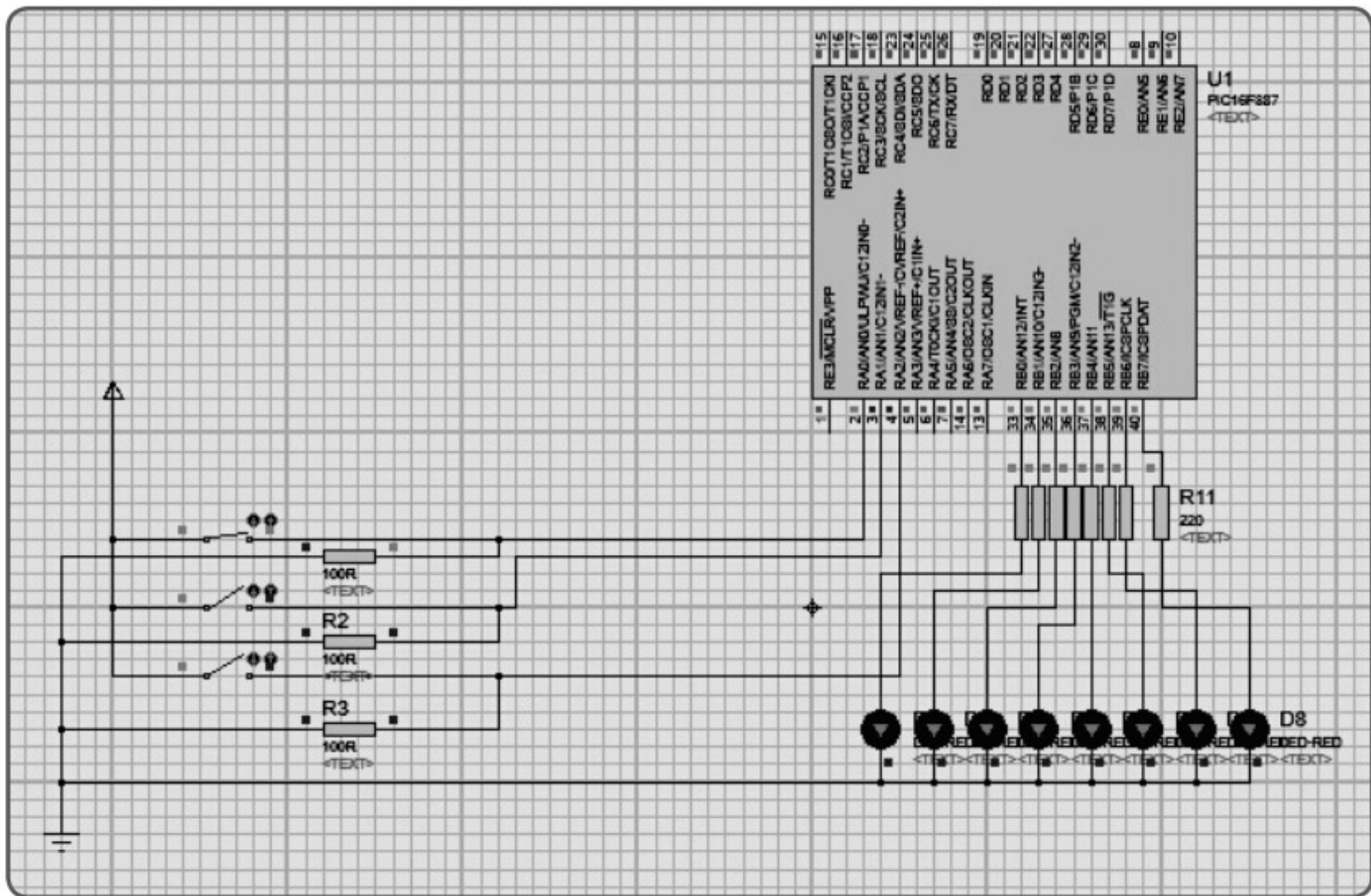


Figura 5.40 LEDAPAGADO







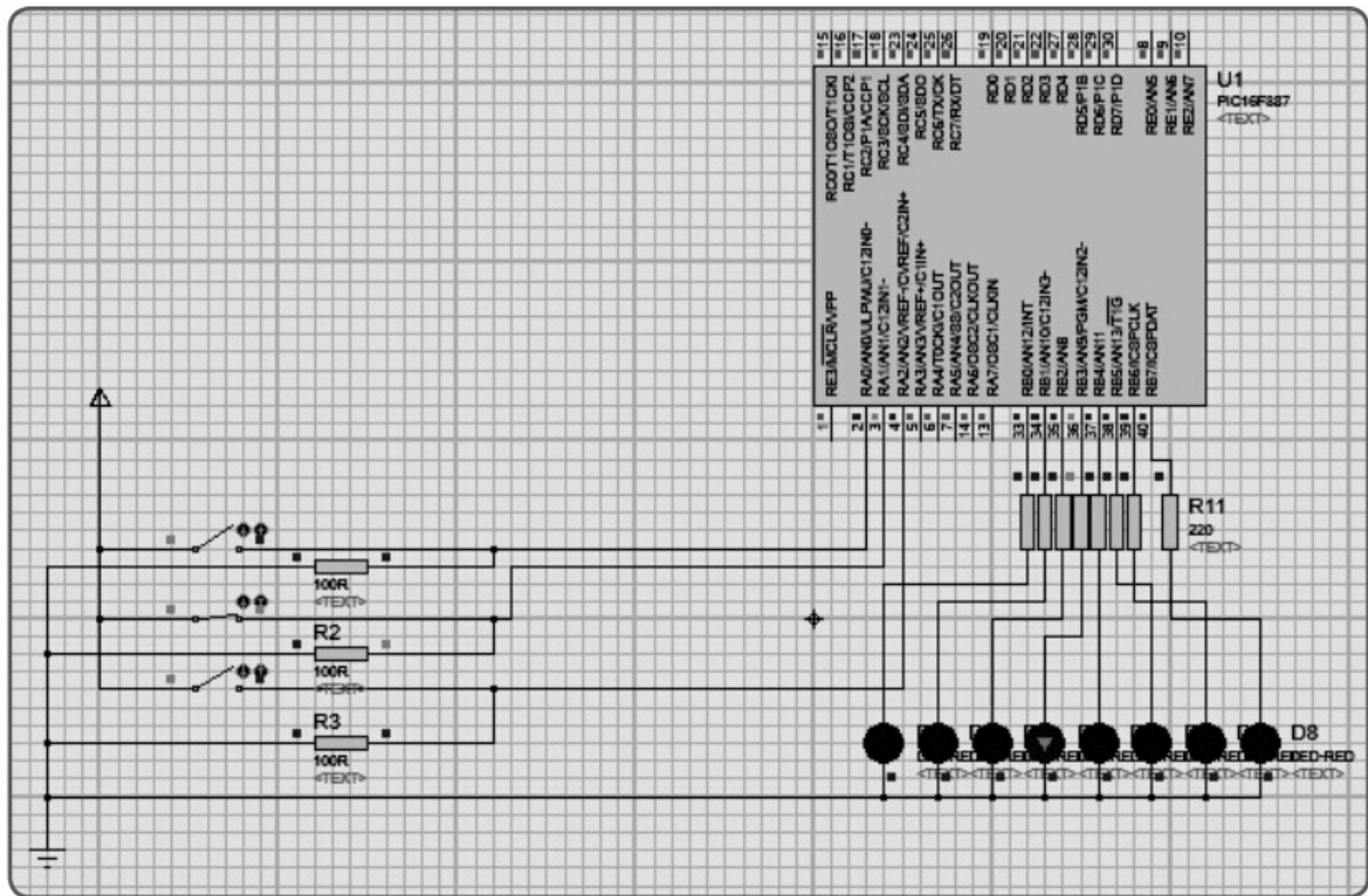




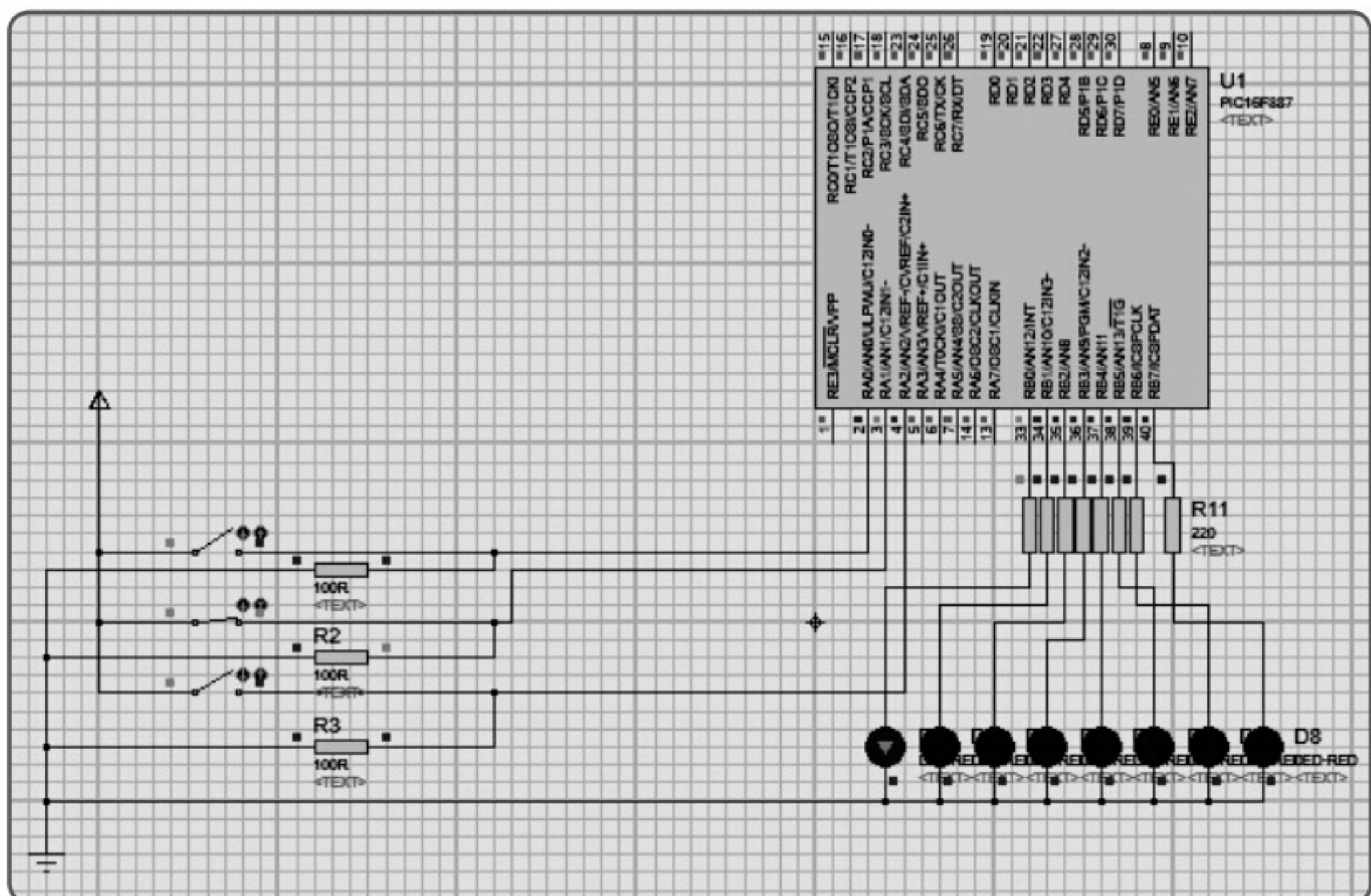
The diagram shows a PIC16F887 microcontroller (U1) interfaced with an 8255 PPI. The PIC is connected to a 5V supply and ground. Its I/O lines are connected to the 8255's control and data buses. The 8255 is configured with R1 (220 ohm) on its output enable pin. The 8255's output lines are connected to an 8-pin D-sub connector (D8) labeled 'D8'.



## Paso 6

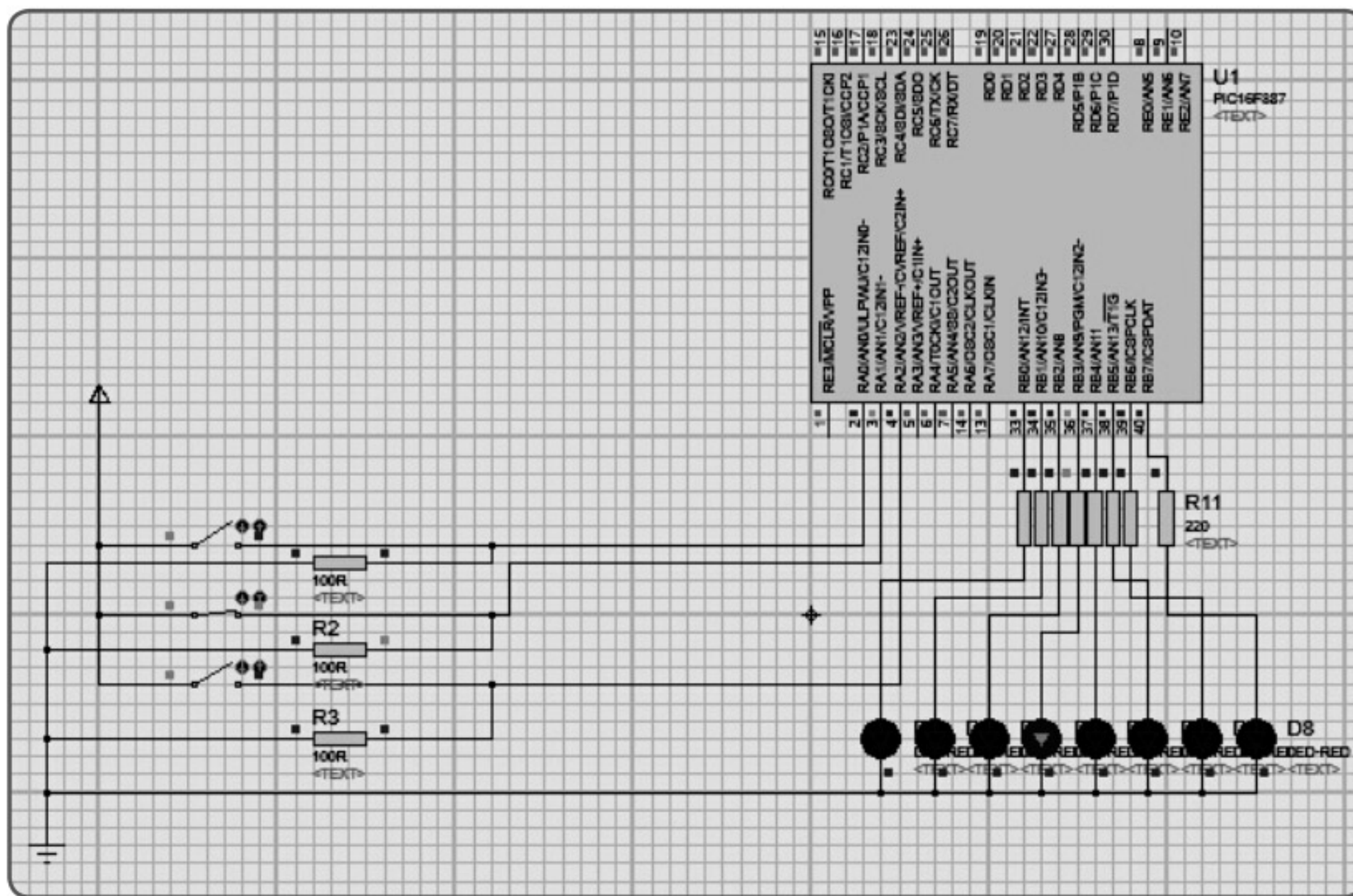


## Paso 7

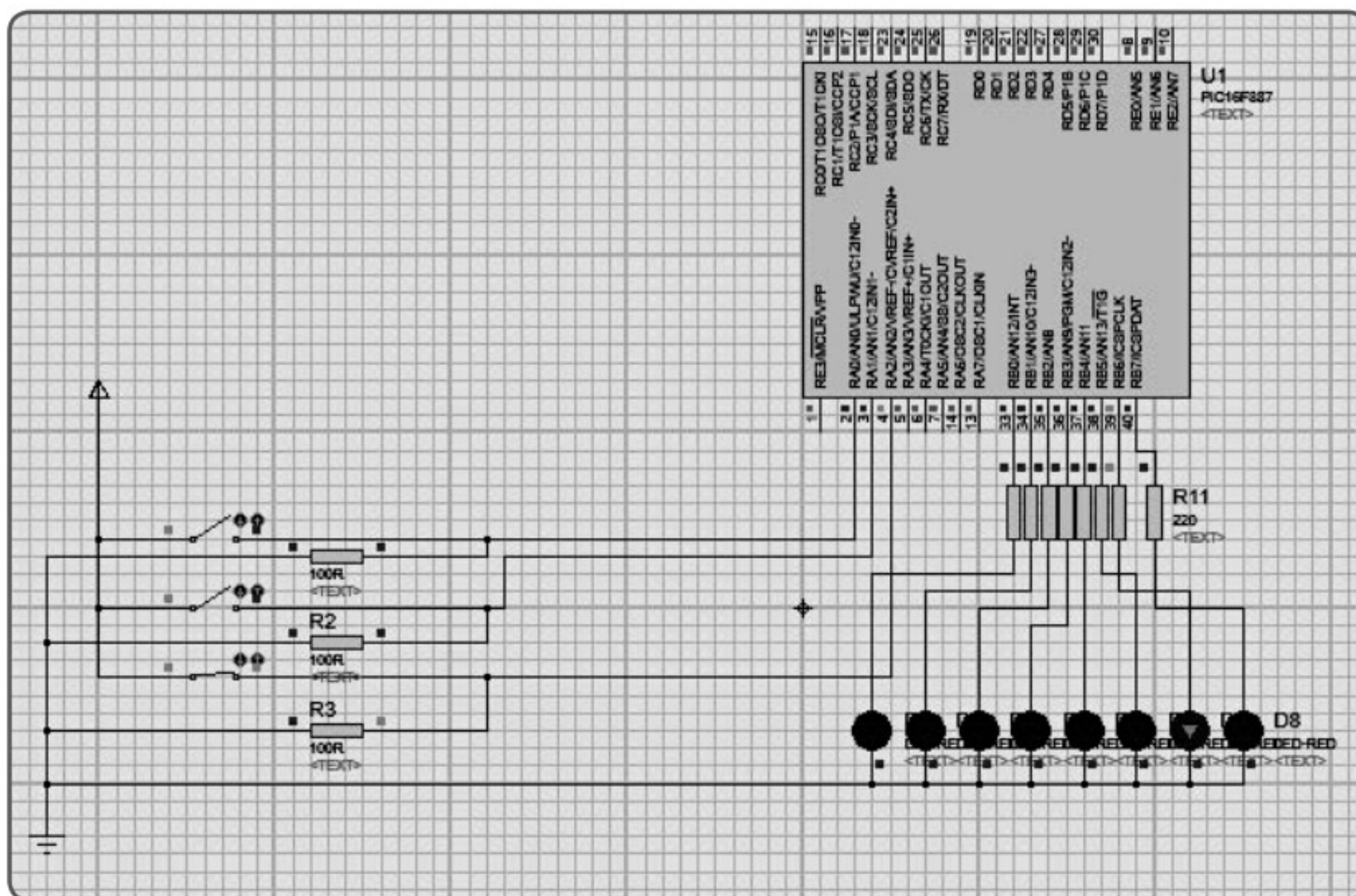




### Paso 8



## Paso 9





## Paso 10

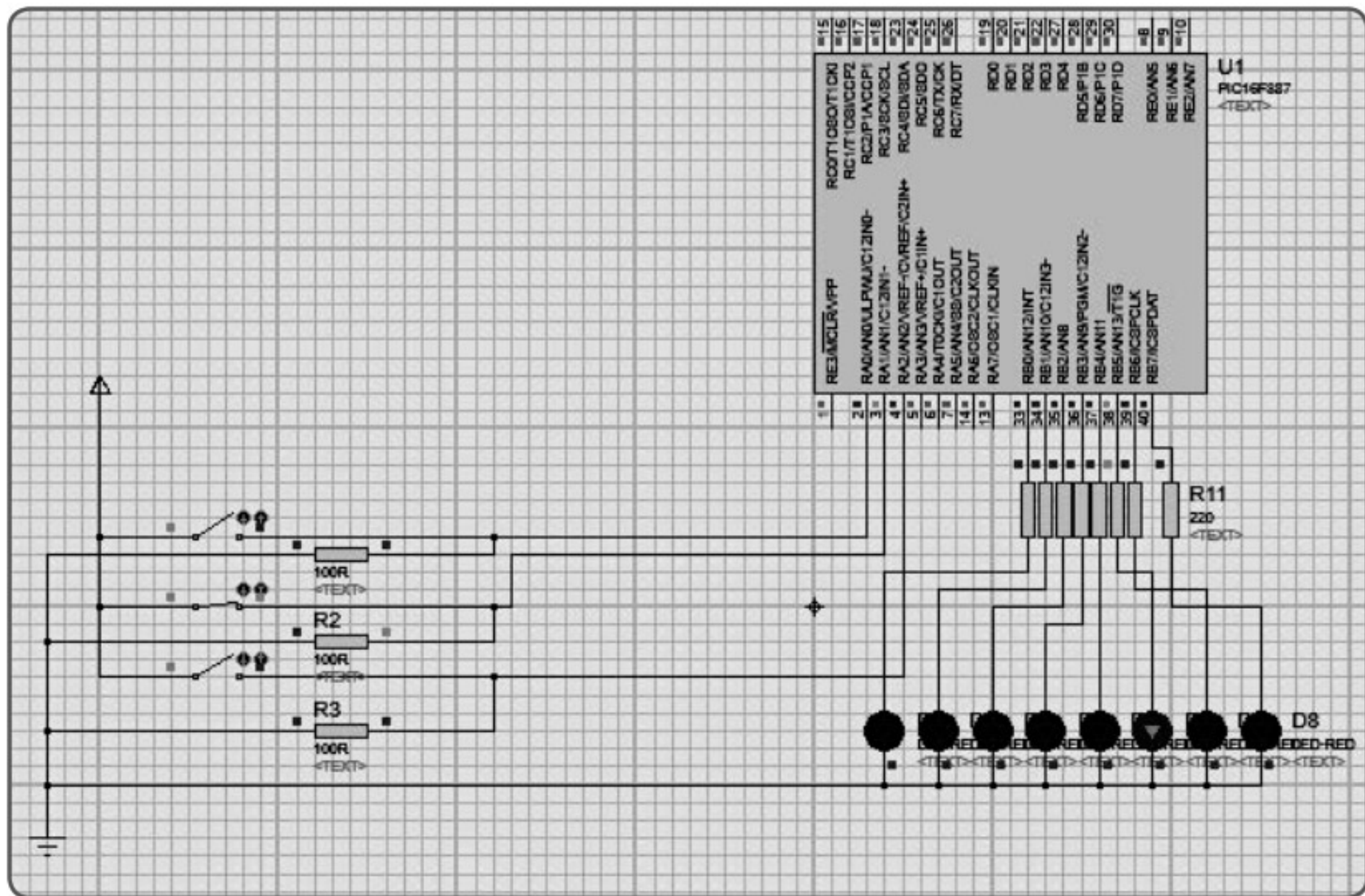


Figura 5.42 SimulandoLED

## Ejercicio 14

Realizar una cuenta atrás mediante un display, de forma que empiece en 9, 8, 7, 6, 5, 4, 3, 2, 1, 0 y, en ese momento, se ilumine de forma parpadeante el led central del display 5 veces (utilizar un decrementador dcfsz).

```

;-----
Código en assembler:
;-----
LIST P=16F84A
INCLUDE <P16F84A.INC>
__CONFIG      _CP_OFF &      _WDT_OFF & _PWRTE_ON & _XT_OSC
CBLOCK 0X0C
contador
ENDC

VAR                                EQU 0X0C
#define Display PORTB
org 0

```



```
BSF STATUS, RP0
CLRF Display
BCF STATUS, RP0

INICIO

MOVLW B'11100111'
MOVWF Display

CALL Retardo_200ms

MOVLW B'11111111'
MOVWF Display

call Retardo_200ms

MOVLW B'10000111'
MOVWF Display

CALL Retardo_200ms

MOVLW B'11111101'
MOVWF Display

CALL Retardo_200ms

MOVLW B'11101101'
MOVWF Display

CALL Retardo_200ms
MOVLW B'11100110'
MOVWF Display

CALL Retardo_200ms

MOVLW B'1001111'
MOVWF Display
CALL Retardo_200ms
```

```
MOVLW B'11011011'
MOVWF Display
CALL Retardo_200ms
MOVLW B'10000110'
MOVWF Display

CALL Retardo_200ms

MOVLW B'101'
MOVWF VAR

CALL INDICADOR

GOTO INICIO

;*****indicador_de_conduccion****
INDICADOR

MOVLW B'11000000'
MOVWF Display

CALL Retardo_200ms

MOVLW B'10000000'
MOVWF Display

CALL Retardo_200ms

DECFSZ VAR
GOTO INDICADOR
RETURN

;*****importar libreria*****

INCLUDE<RETARDOS.INC>
END
```



Abra el programa MPLAB. Al hacer clic en **New**, se abre una ventana donde debe escribir los códigos correspondientes para el funcionamiento del programa.



```

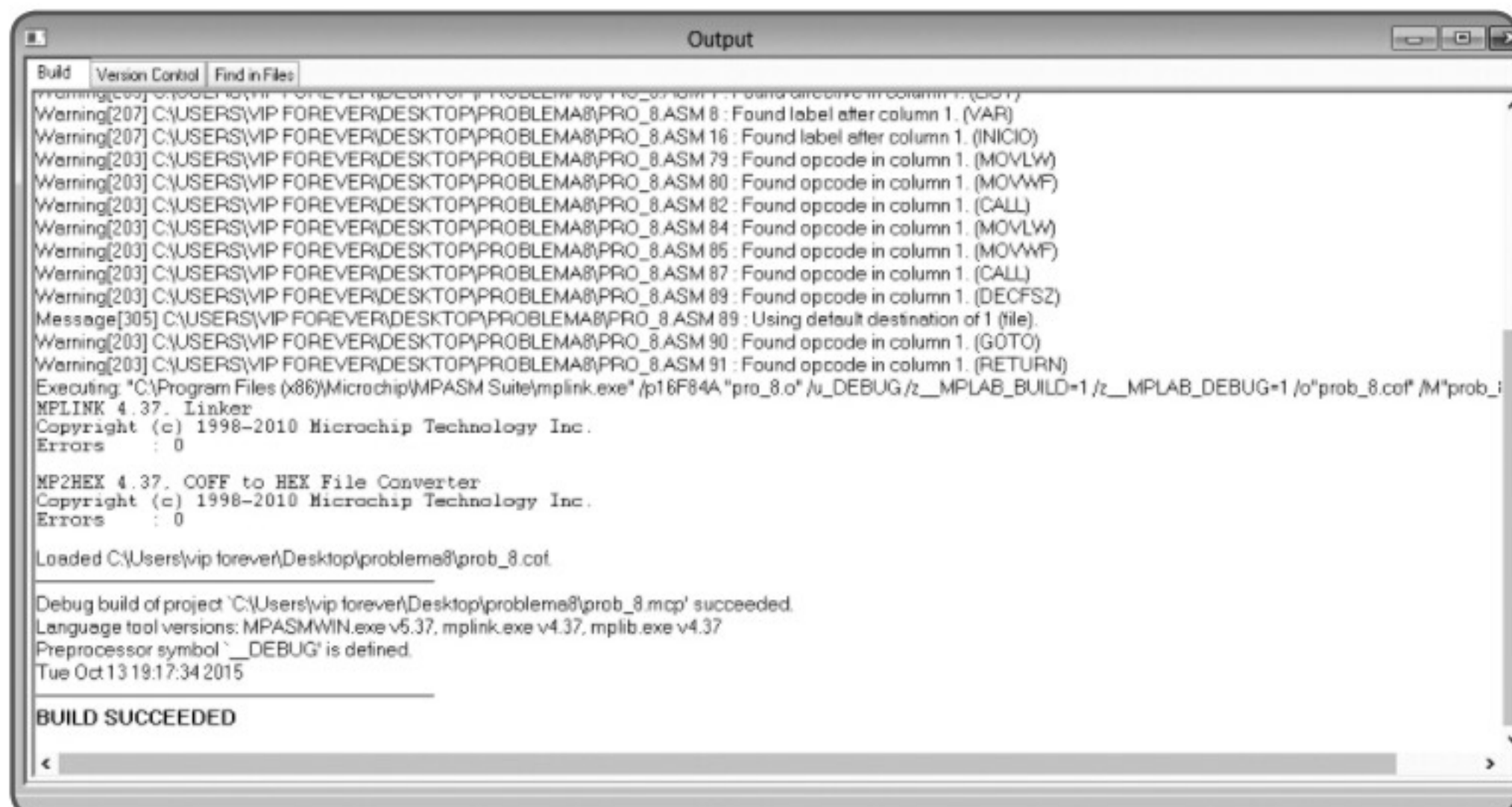
LIST      P=16F84A
INCLUDE <P16F84A.INC>
__CONFIG  _CP_OFF &  _WDT_OFF &  _PWRTE_ON &  _XT_OSC
CBLOCK 0X0C
contador
ENDC

|
VAR      EQU 0X0C
#DEFINE Display PORTB
org 0

BSF STATUS, RP0
CLRF Display
BCF STATUS, RP0

INICIO
  
```

**Figura 5.43** ventana\_new



```

Build  Version Control  Find in Files
Warning[207] C:\Users\vip forever\Desktop\problema8\pro_8.asm 7: Found opcode in column 1. (BSF)
Warning[207] C:\Users\vip forever\Desktop\problema8\pro_8.asm 8: Found label after column 1. (VAR)
Warning[207] C:\Users\vip forever\Desktop\problema8\pro_8.asm 16: Found label after column 1. (INICIO)
Warning[203] C:\Users\vip forever\Desktop\problema8\pro_8.asm 79: Found opcode in column 1. (MOVLW)
Warning[203] C:\Users\vip forever\Desktop\problema8\pro_8.asm 80: Found opcode in column 1. (MOVWF)
Warning[203] C:\Users\vip forever\Desktop\problema8\pro_8.asm 82: Found opcode in column 1. (CALL)
Warning[203] C:\Users\vip forever\Desktop\problema8\pro_8.asm 84: Found opcode in column 1. (MOVLW)
Warning[203] C:\Users\vip forever\Desktop\problema8\pro_8.asm 85: Found opcode in column 1. (MOVWF)
Warning[203] C:\Users\vip forever\Desktop\problema8\pro_8.asm 87: Found opcode in column 1. (CALL)
Warning[203] C:\Users\vip forever\Desktop\problema8\pro_8.asm 89: Found opcode in column 1. (DECFSZ)
Message[305] C:\Users\vip forever\Desktop\problema8\pro_8.asm 89: Using default destination of 1 (file).
Warning[203] C:\Users\vip forever\Desktop\problema8\pro_8.asm 90: Found opcode in column 1. (GOTO)
Warning[203] C:\Users\vip forever\Desktop\problema8\pro_8.asm 91: Found opcode in column 1. (RETURN)
Executing: "C:\Program Files (x86)\Microchip\MPASM Suite\mplink.exe" /p16F84A "pro_8.o" /u_DEBUG /z_MPLAB_BUILD=1 /z_MPLAB_DEBUG=1 /o"prob_8.cof" /M"prob_8.mcp"
MPLINK 4.37. Linker
Copyright (c) 1998-2010 Microchip Technology Inc.
Errors      : 0

MP2HEX 4.37. COFF to HEX File Converter
Copyright (c) 1998-2010 Microchip Technology Inc.
Errors      : 0

Loaded C:\Users\vip forever\Desktop\problema8\prob_8.cof

Debug build of project 'C:\Users\vip forever\Desktop\problema8\prob_8.mcp' succeeded.
Language tool versions: MPASMWIN.exe v5.37, mplink.exe v4.37, mplib.exe v4.37
Preprocessor symbol '_DEBUG' is defined.
Tue Oct 13 19:17:34 2015

BUILD SUCCEEDED
  
```

**Figura 5.44** Build



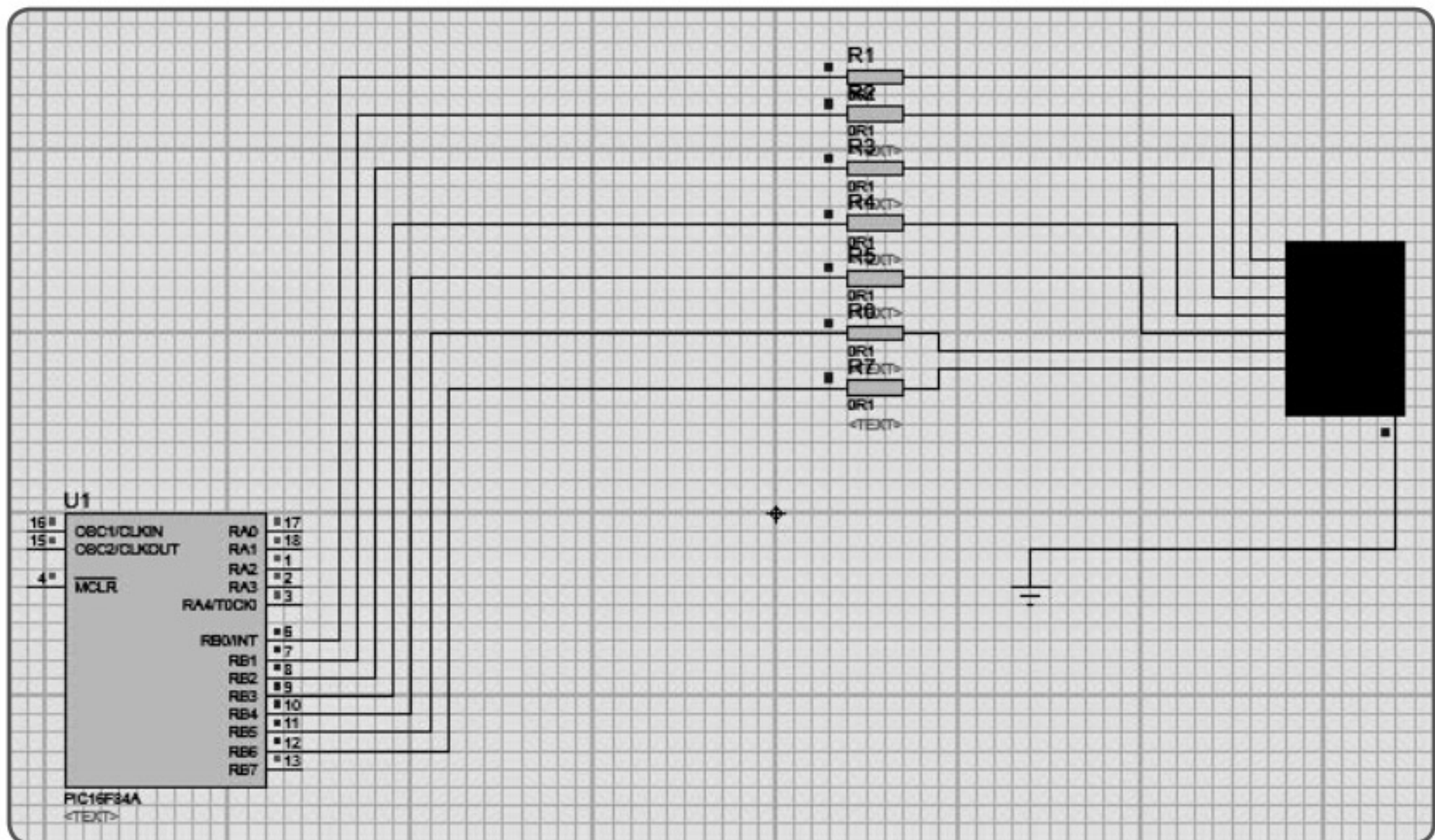
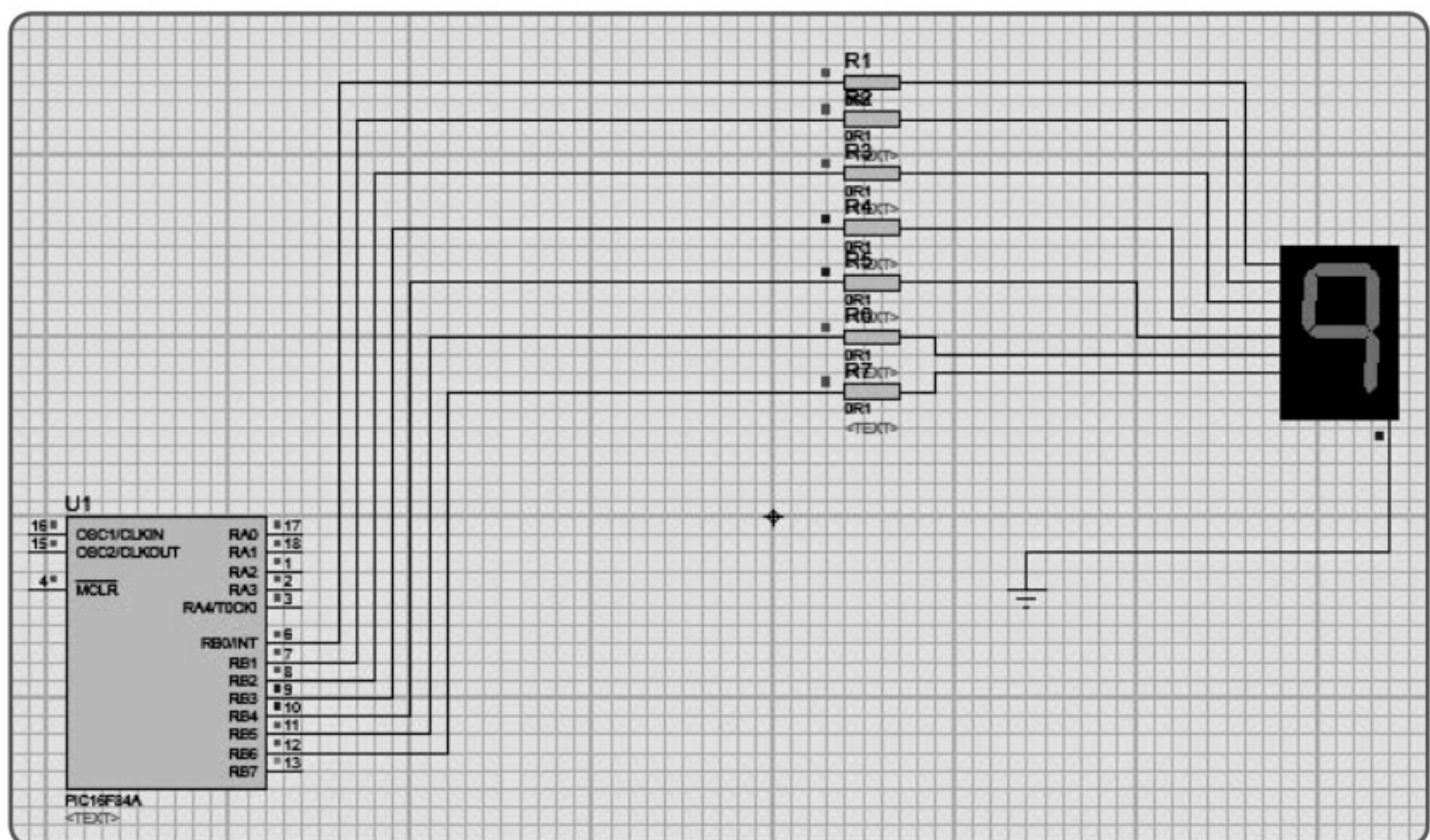


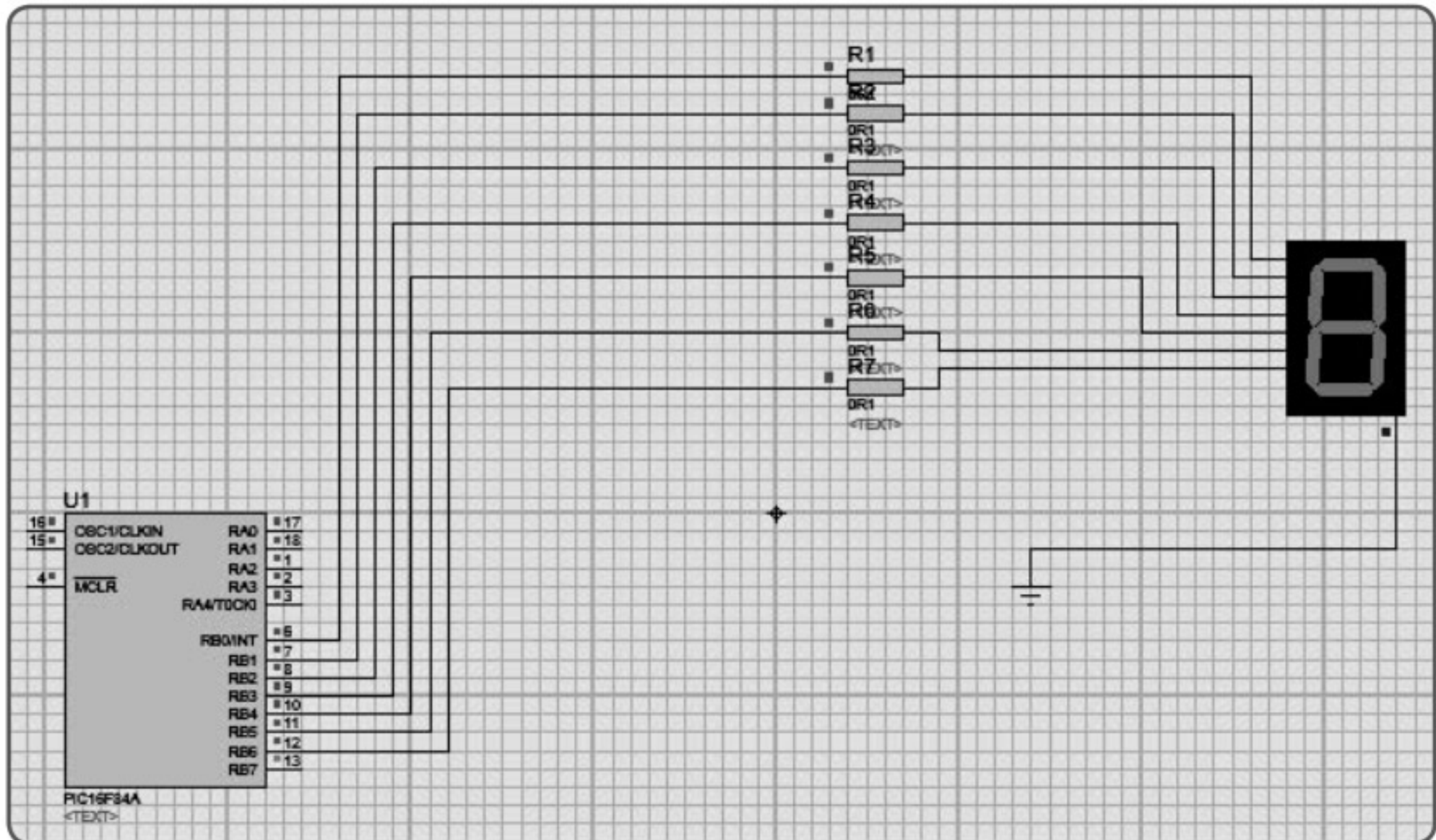
Figura 5.45 Build cargando el programa en PROTEUS

### Paso 1

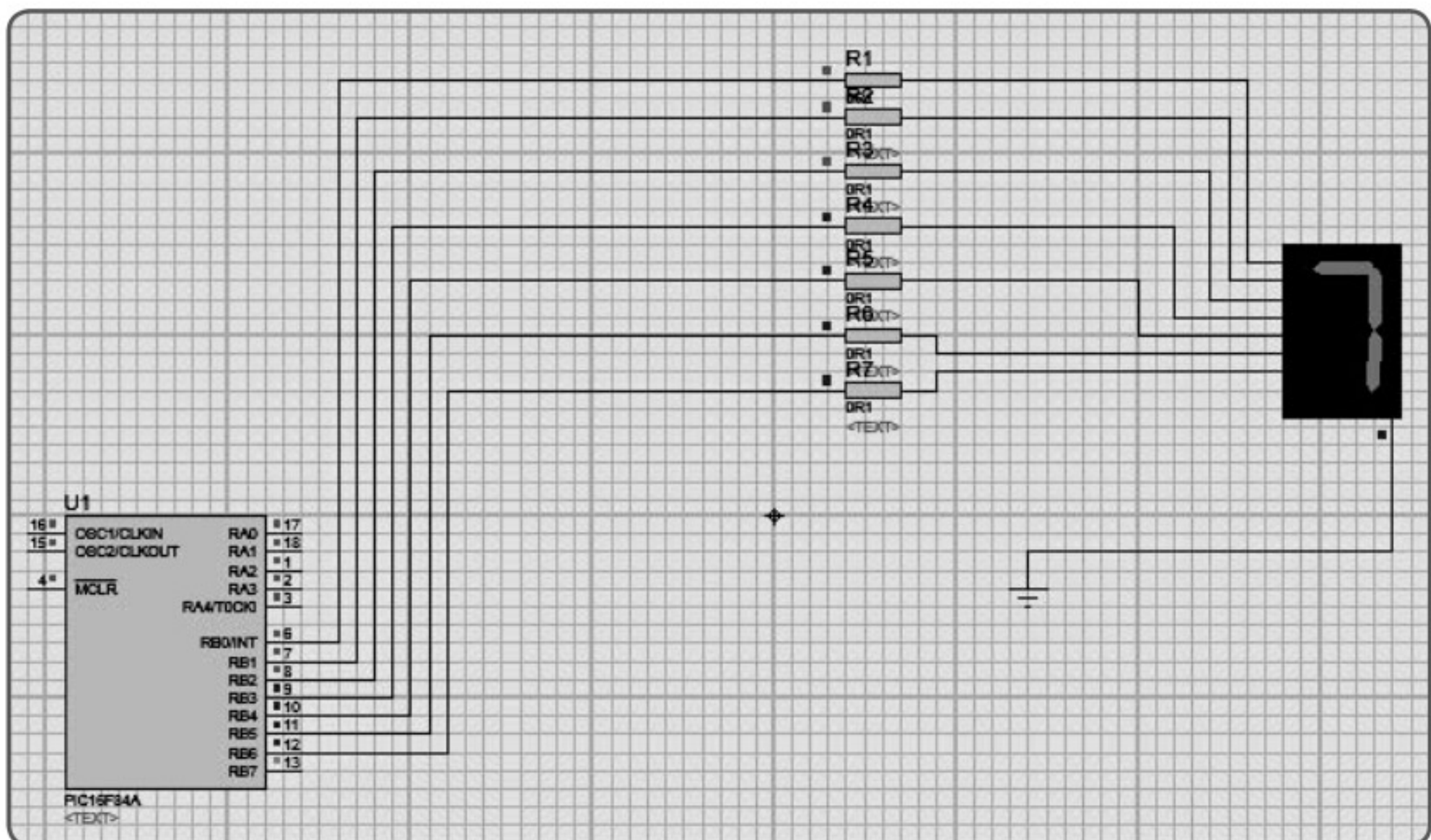




## Paso 2

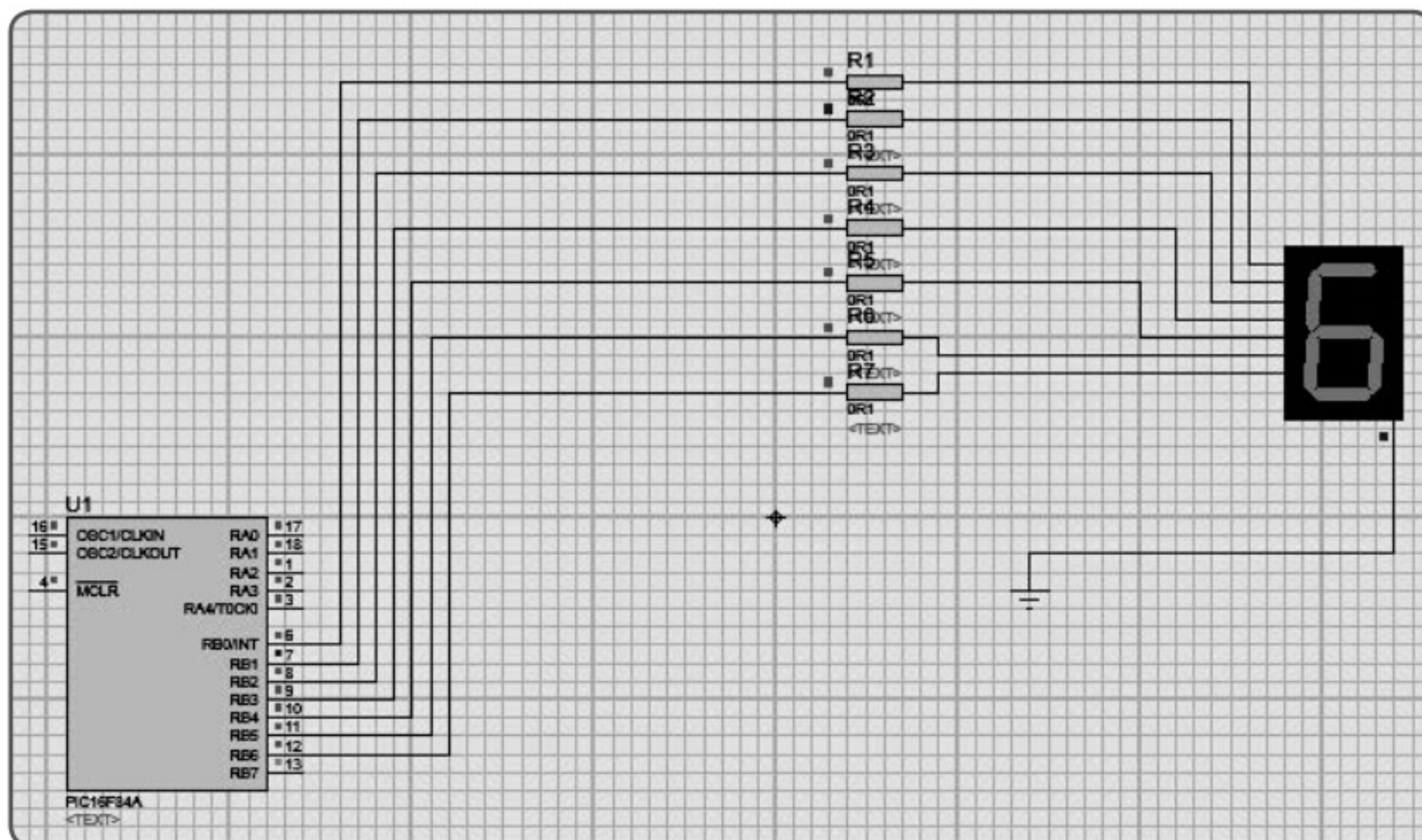


## Paso 3

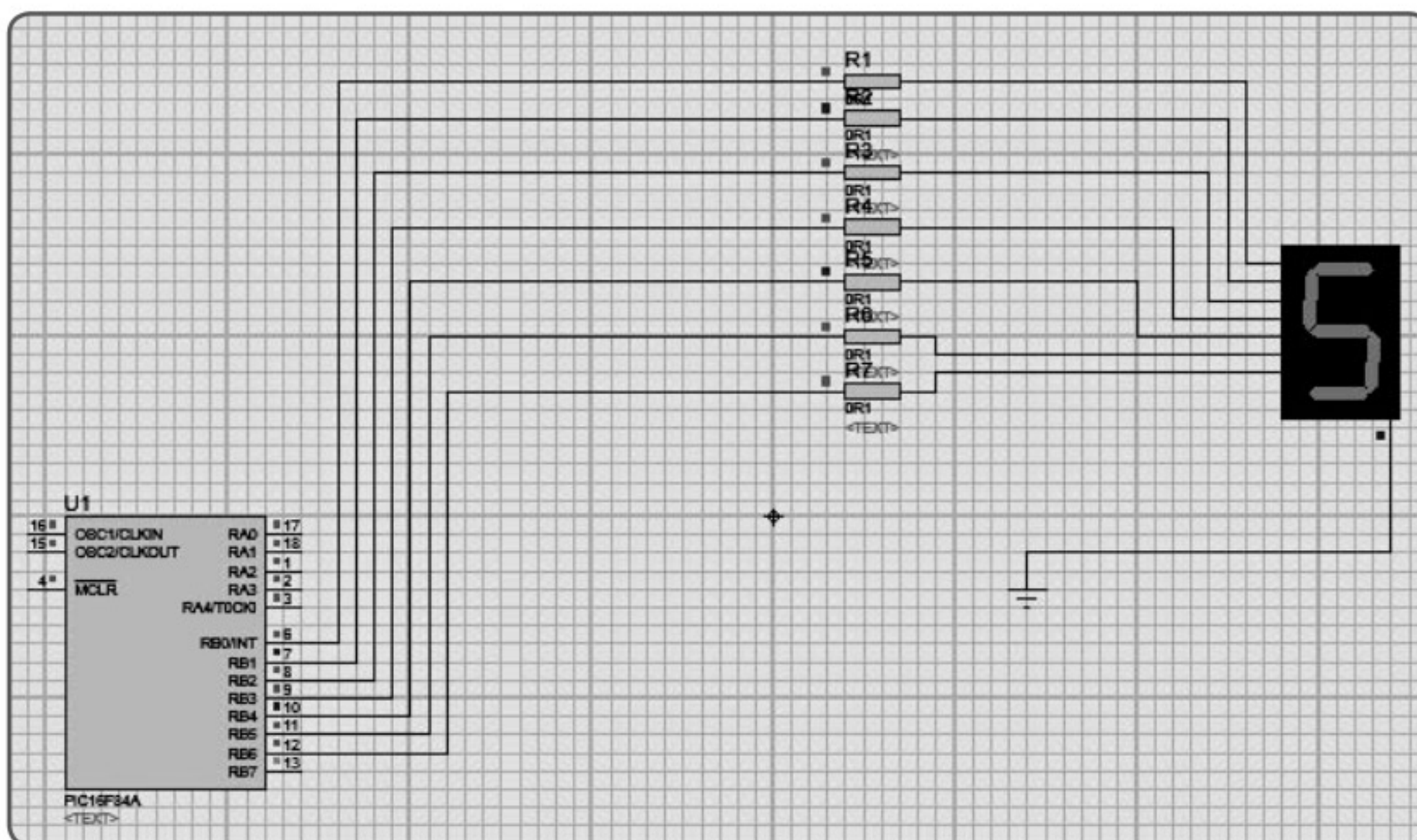




## Paso 4

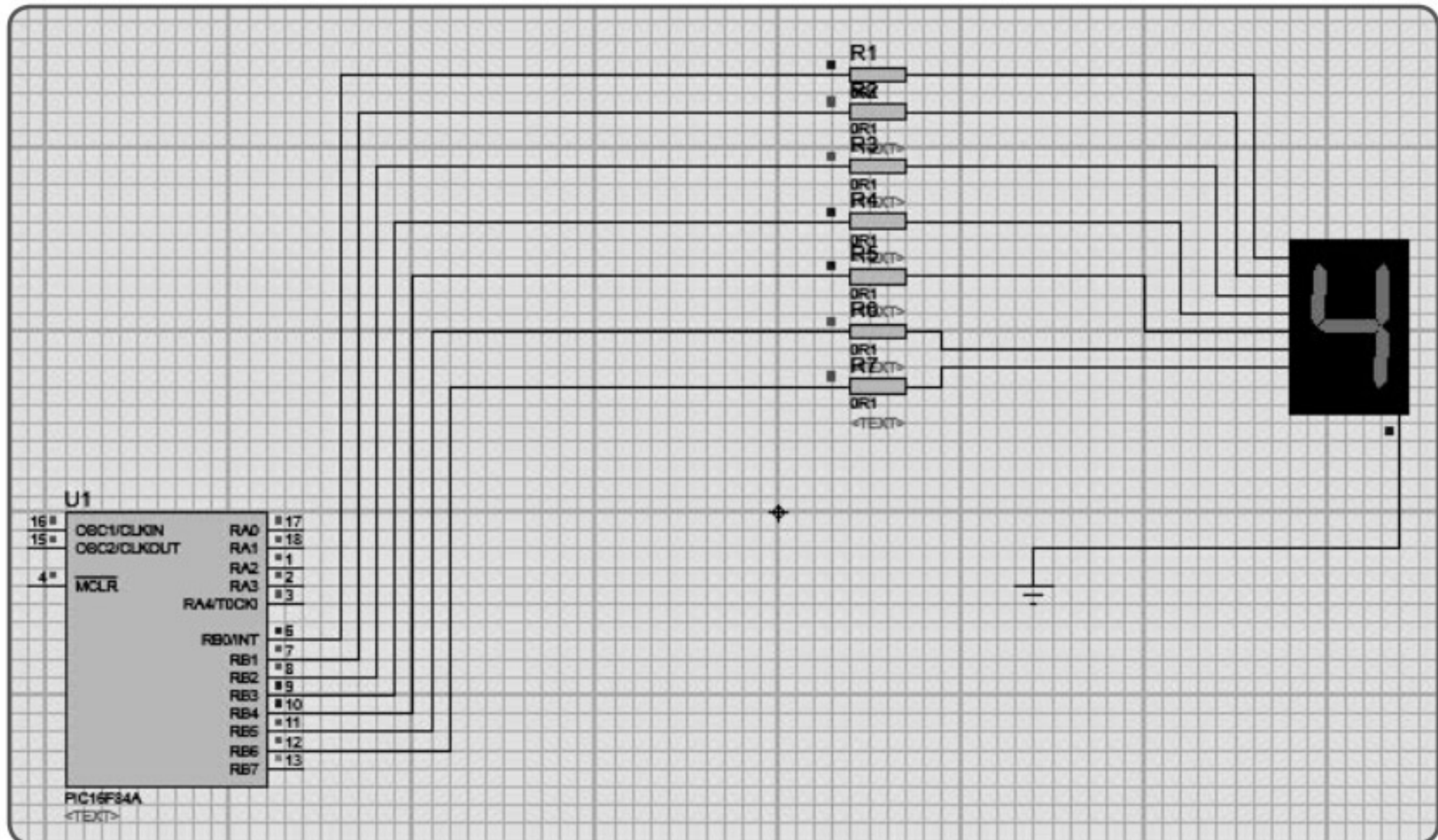


## Paso 5

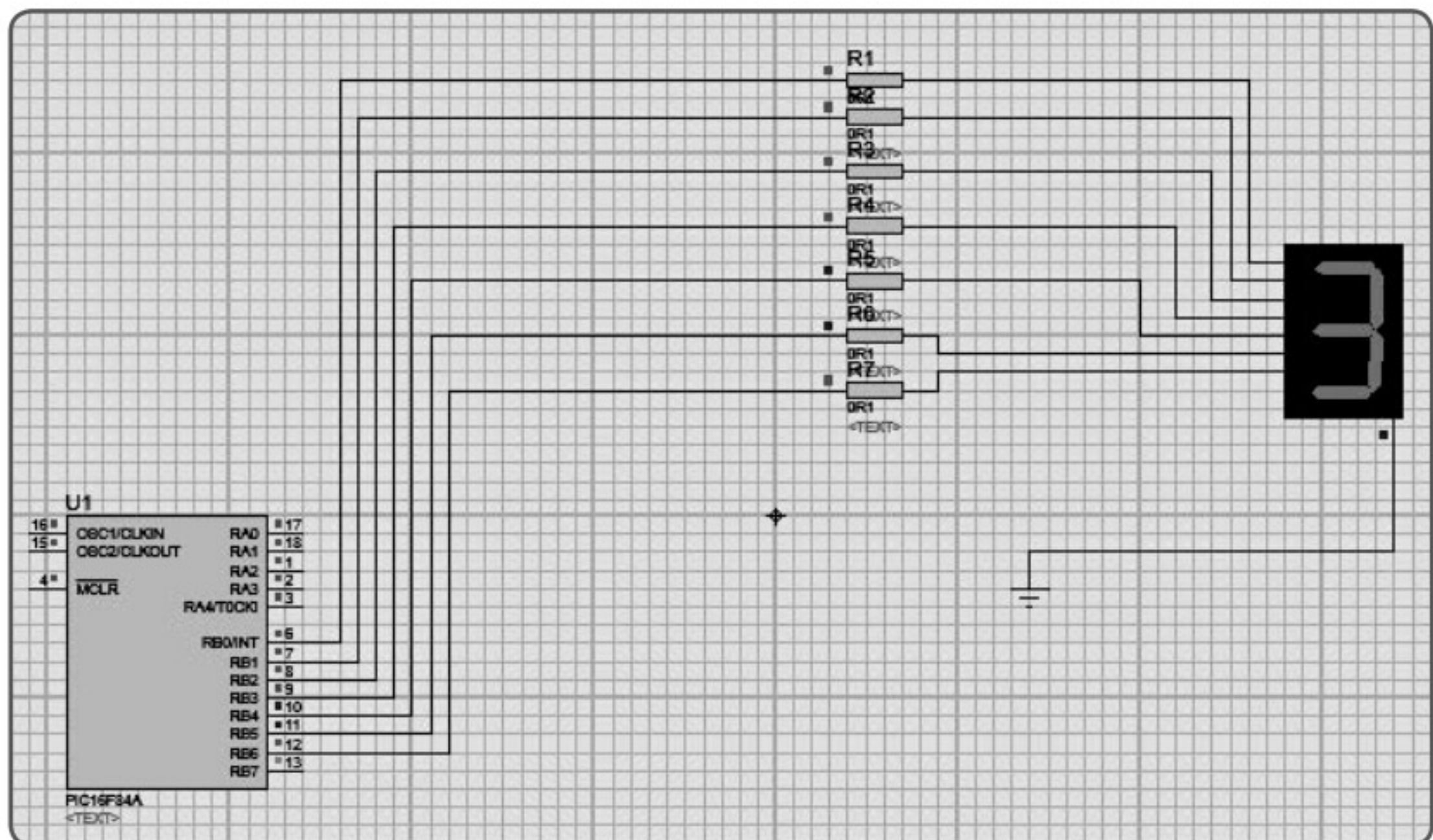




## Paso 6

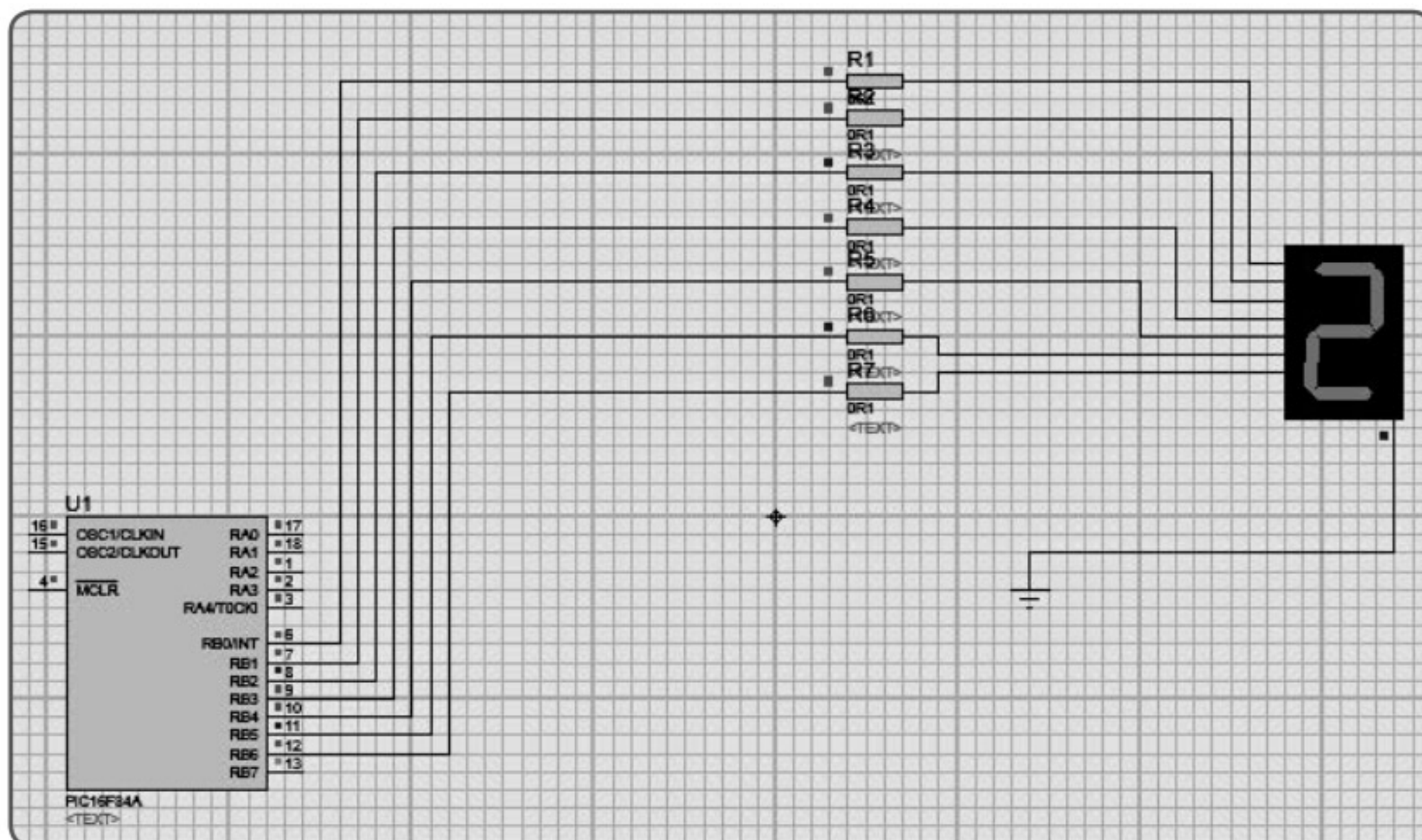


## Paso 7

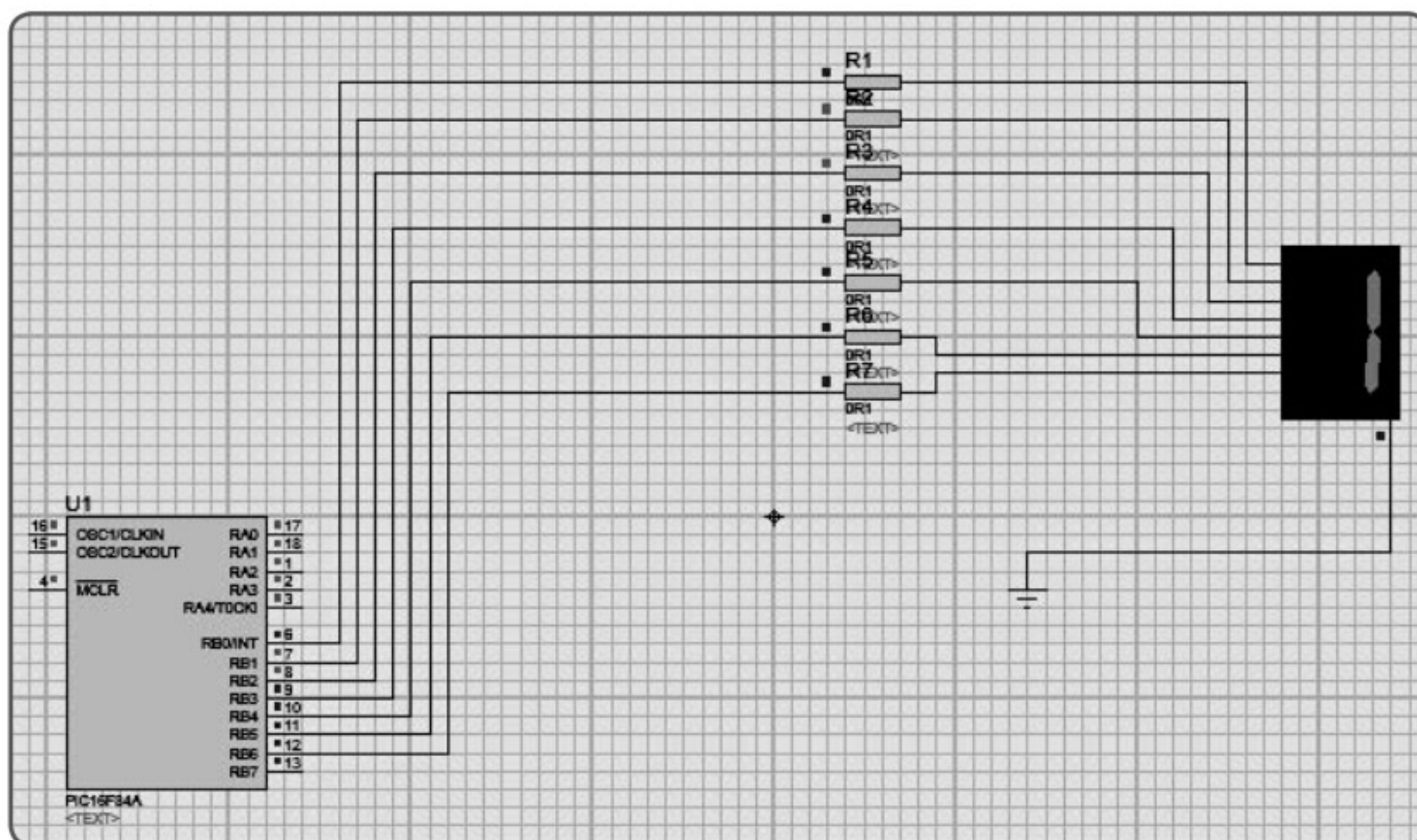




## Paso 8



## Paso 9





## Paso 10

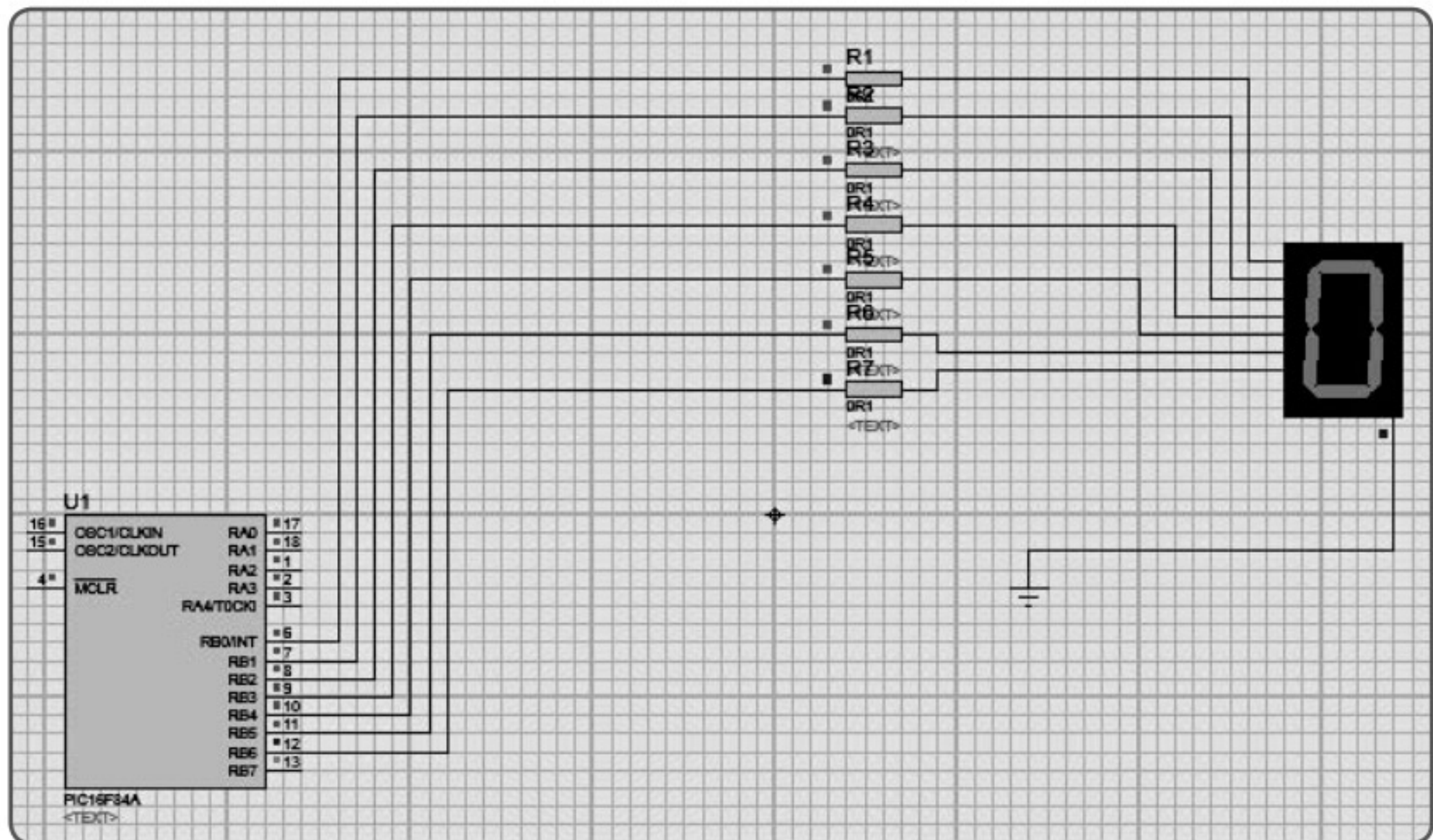


Figura 5.46 Build cargando el programa en PROTEUS

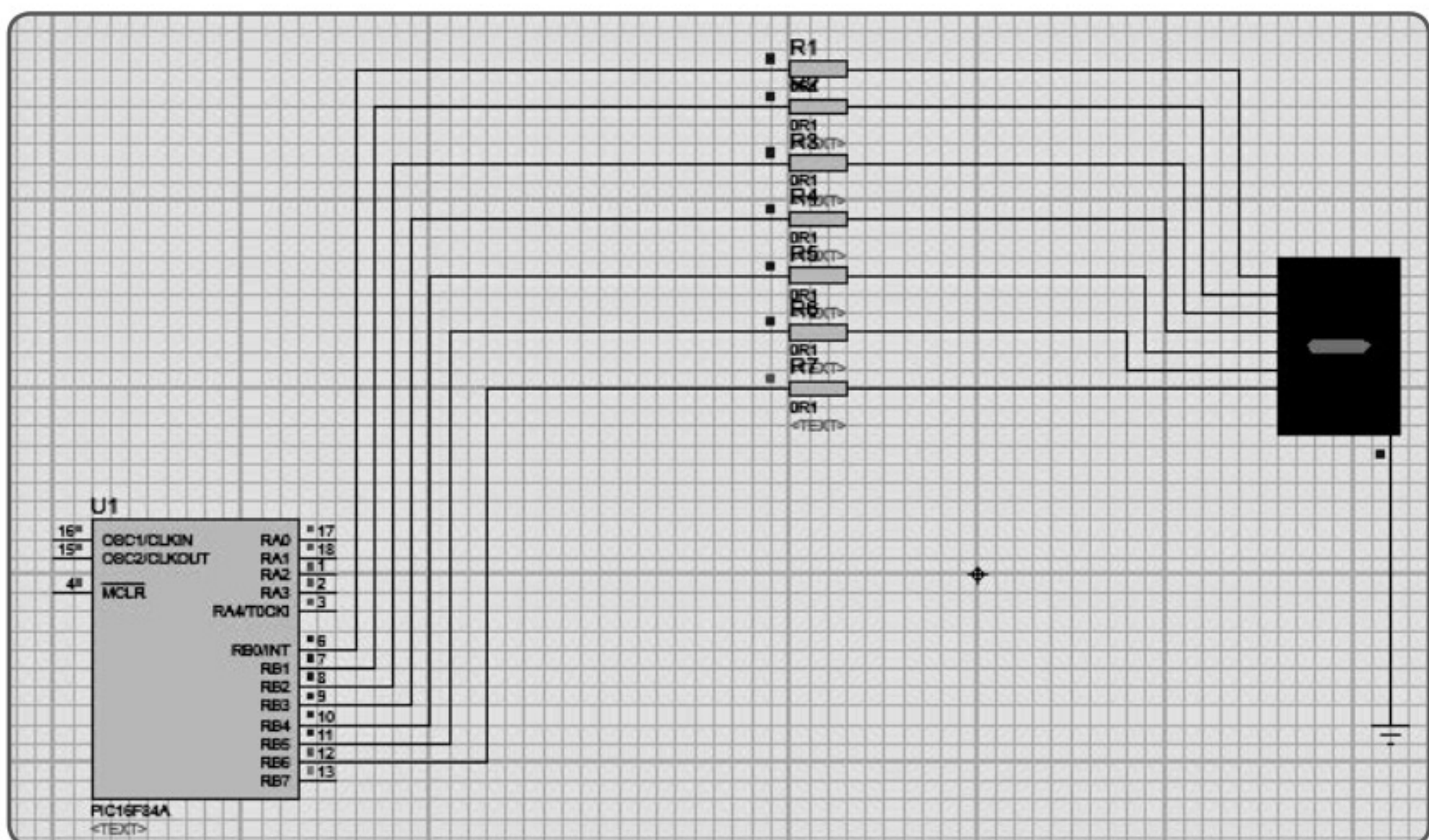


Figura 5.47 ModeloProteus



## Ejercicio 15

Realizar el control de un depósito de agua, de forma que se tiene 4 sensores conectados al port a y, dependiendo del nivel, se visualizará mediante una barra de 4 ledes conectada al port b, además, al llegar al nivel mínimo se tendrá un led que indicará la puesta en marcha de una bomba de agua. Al llegar al nivel máximo, esta bomba se para y se ilumina de forma parpadeante otro led.

```

;-----
Código en assembler:
;-----
LIST P=16F887
INCLUDE <P16F887.INC>
ORG 0
BSF STATUS, RP0
BSF STATUS, RP1
CLRF ANSEL
BCF STATUS, RP1
CLRF TRISA
MOVLW b'11111111'
MOVWF TRISA
CLRF TRISB
CLRF TRISD
BCF STATUS, RP0
CLRW

```

```

CODIGO
INICIO
MOVLW b'00001111'
ANDWF PORTA, W
MOVWF PORTB
BTFSS PORTA, 3
GOTO PRENDE
BCF PORTD, 7
GOTO INICIO
PRENDE
BSF PORTD, 7
GOTO INICIO
END

```

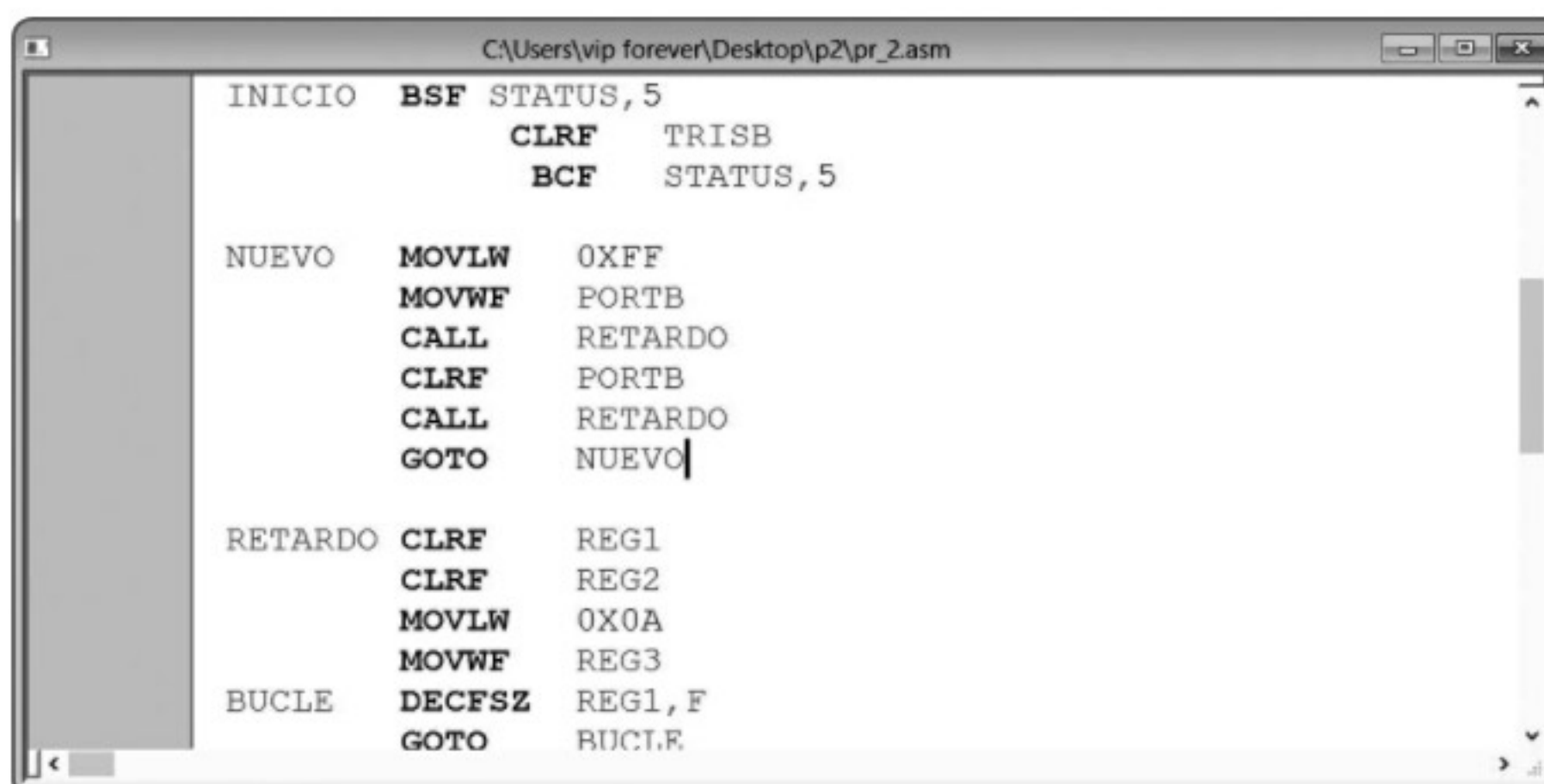


Figura 5.48 ModeloProteus



Abra el programa MPLAB, al hacer clic en **New** se abrirá una ventana donde se observará los códigos correspondientes para el funcionamiento del programa.

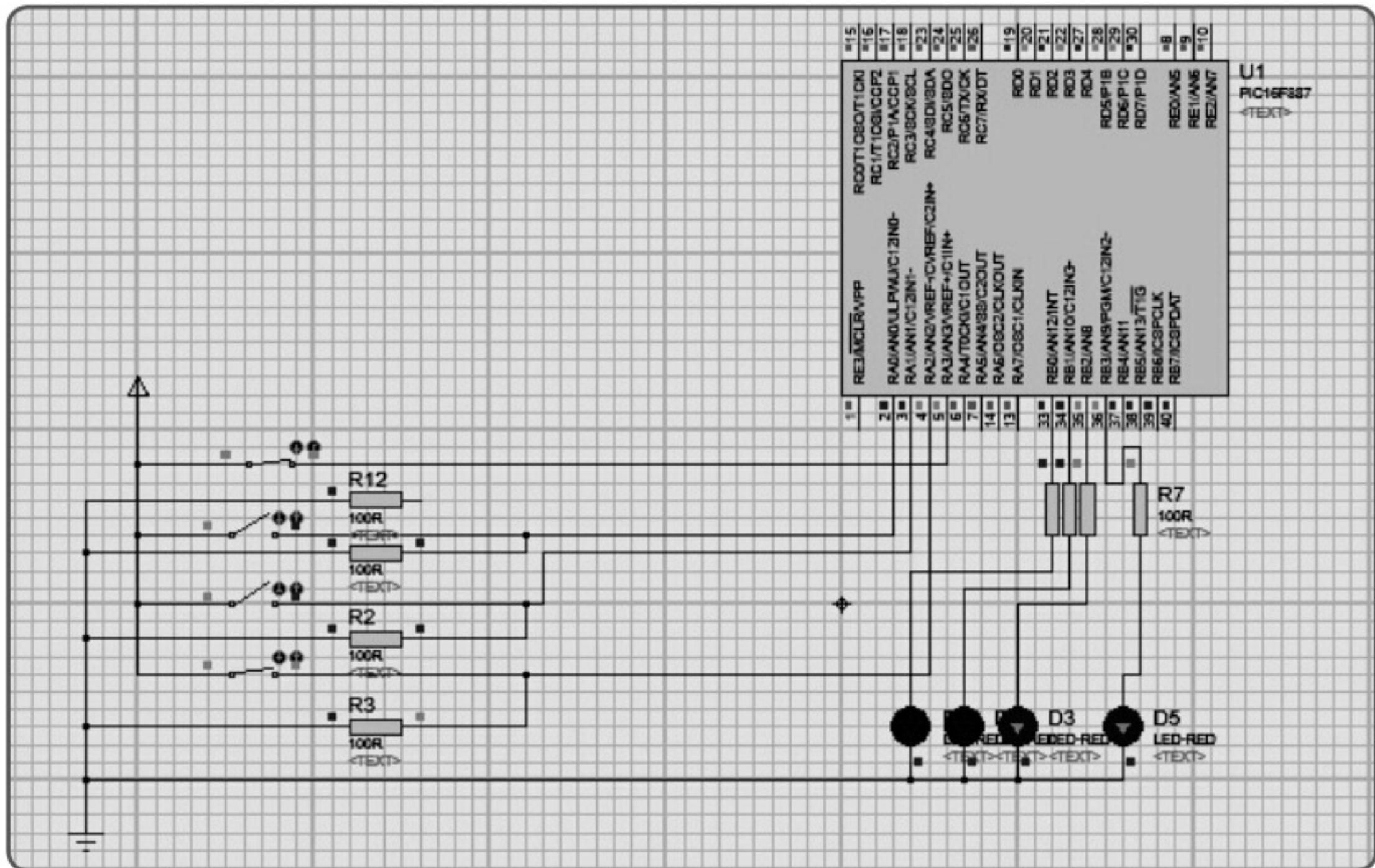


Figura 5.49 Model\_ProteusA

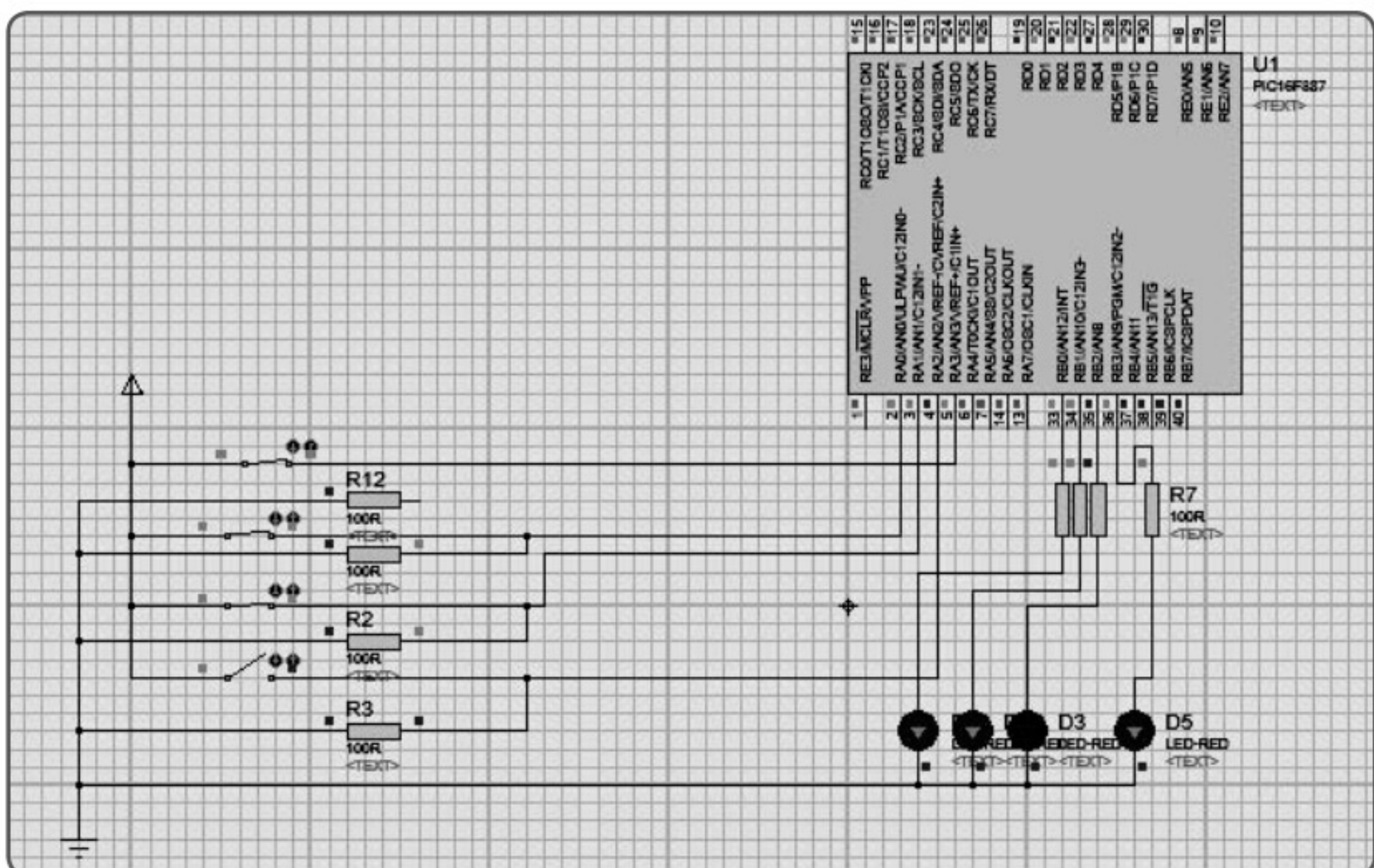
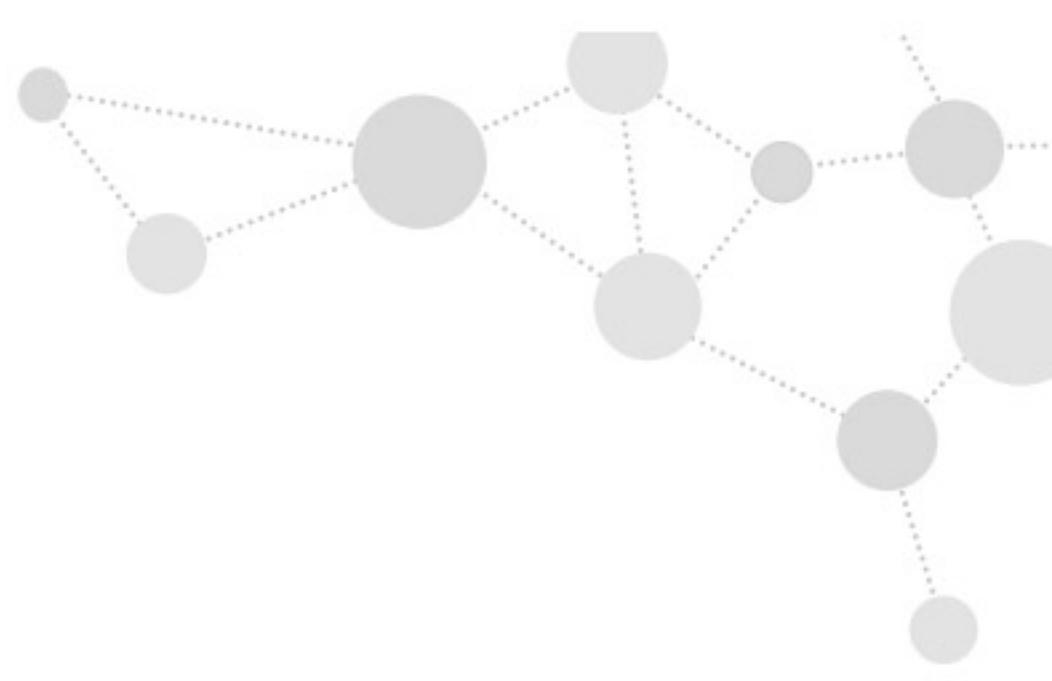


Figura 5.49 Model\_ProteusA









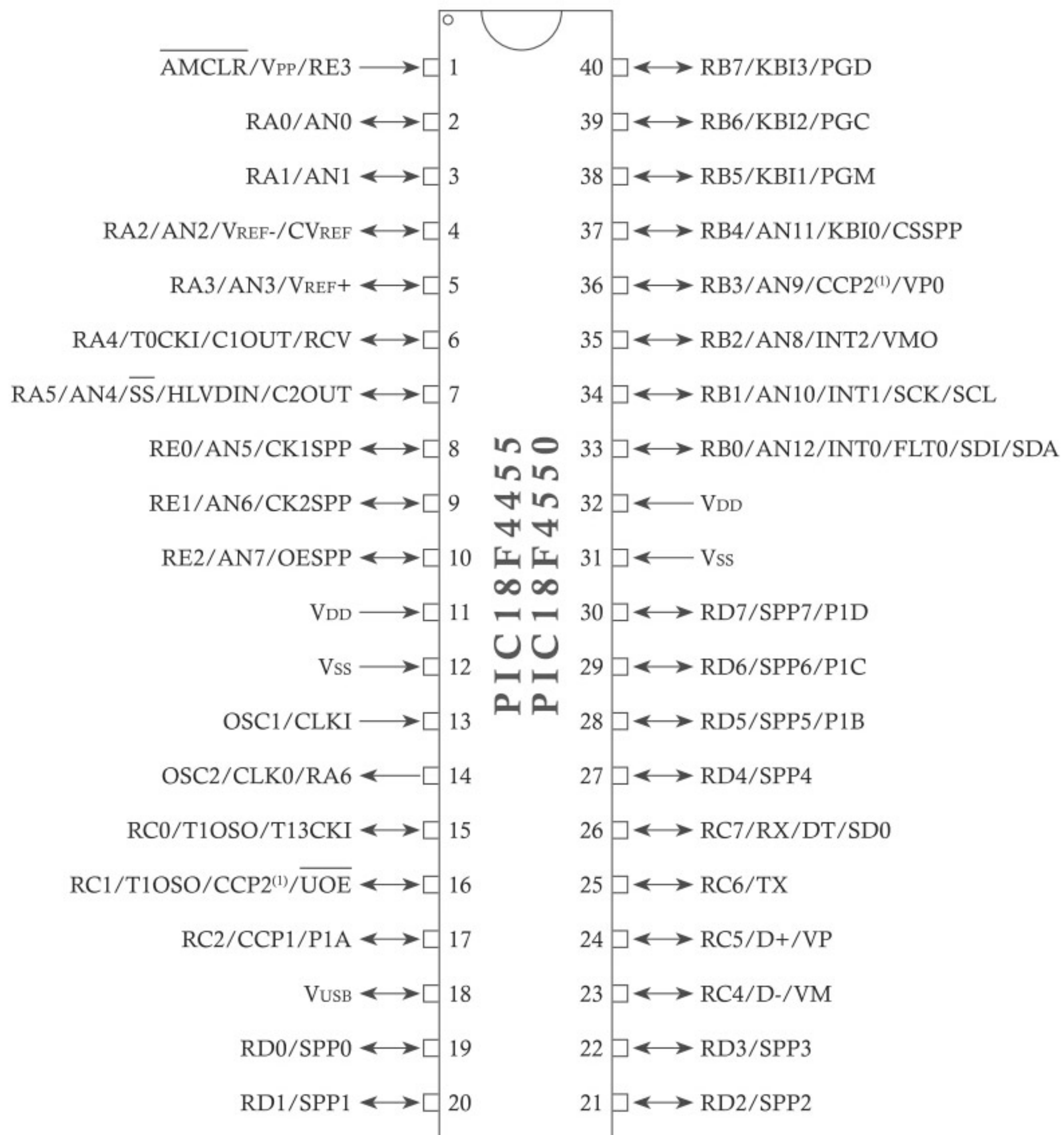
# ANEXOS





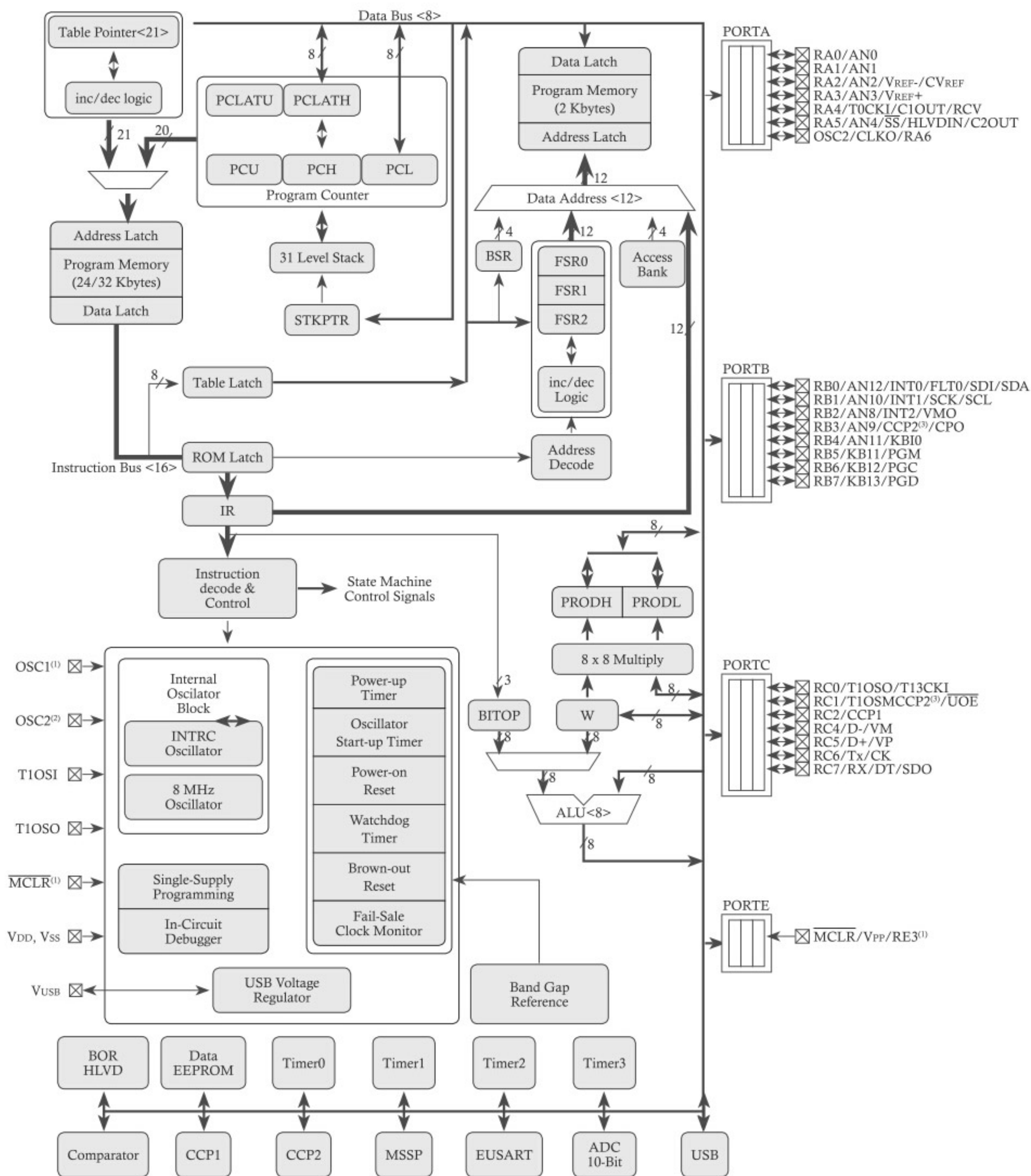


## Anexo 1: PIC 18F4455





## Anexo 2: Diagrama de bloques del PIC18f4550





### Anexo 3: Código assembler

TRANSFERENCIA				Flags								
Nombre	Comentario	Código	Operación	O	D	I	T	S	Z	A	P	C
MOV	Mover (copiar)	MOV Dest, Fuente	Dest.=Fuente									
XCHG	Intercambiar	XCHG Op1, Op2	Op1:=Op2, Op2:=Op1									
STC	Set the carry (Carry = 1)	STC	CF:=1									1
CLC	Clear Carry (Carry=0)	CLC	CF:=0									0
CMC	Complementar Carry	CMC	CF:=¬CF									±
STD	Setear dirección	STD	DF:=1 (interpreta strings de arriba hacia abajo)		1							
CLD	Limpiar dirección	CLD	DF:=0 (interpreta strings de abajo hacia arriba)		0							
STI	Flag de interrupción en 1	STI	IF:=1			1						
CLI	Flag de interrupción en 0	CLI	IF:=0			0						
PUSH	Apilar en la pila	PUSH Fuente	DEC SP, [SP]:=Fuente									
PUSHF	Apilar los flags	PUSHF	O, D, I, T, S, Z, A, P, C 286+: También NT, IOPL									
PUSHA	Apila los registros generales	PUSHA	AX, CX, DX, BX, SP, BP, SI, DI									
POP	Desapila de la pila	POP Dest	Destino:=[SP], INC SP									
POPF	Desapila a los flags	POPF	O, D, I, T, S, Z, A, P, C 286+: También NT, IOPL	±	±	±	±	±	±	±	±	±
POPA	Desapila a los reg. general	POPA	DI, SI, BP, SP, BX, DX, CX, AX									
CBW	Convertir Byte a Word	CBW	AX=AL (con signo)									
CWD	Convertir Word a Doble	CWD	DX:AX:=AX (con signo)	±				±	±	±	±	±
CWDE	Conv. Word a Doble Exten.	CWDE 386	EAX:=AX (con signo)									
IN i	Entrada	IN Dest, Puerto	AL/AX/EAX:=byte/word/double del puerto especifi.									
OUT i	Salida	OUT Puerto, Fuente	Byte/word/double del puerto especifi. :=AL/AX/EAX									

i para más información ver especificaciones de la instrucción Flags: ±=Afectado por esta instrucción ?=Indefinido luego de esta instrucción

ARITMÉTICOS				Flags								
Nombre	Comentario	Código	Operación	O	D	I	T	S	Z	A	P	C
ADD	Suma	ADD Dest, Fuente	Dest:=Dest+ Fuente	±				±	±	±	±	±
ADC	Suma con acarreo	ADC Dest, Fuente	Des:=Dest+ Fuente +CF	±				±	±	±	±	±
SUB	Resta	SUB Dest, Fuente	Dest:=Dest- Fuente	±				±	±	±	±	±
SBB	Resta con acarreo	SBB Dest, Fuente	Dest:=Dest- (Fuente +CF)	±				±	±	±	±	±
DIV	División (sin signo)	DIV Op	Op=byte: AL:=AX / Op AH:=Resto	?				?	?	?	?	?
DIV	División (sin signo)	DIV Op	Op=word: AX:=DX:AX / Op DX:=Resto	?				?	?	?	?	?
DIV 386	División (sin signo)	DIV Op	Op=doublew.: EAX:=EDX:EAX / Op	?				?	?	?	?	?
IDIV	División entera con signo	IDIV Op	Op=byte: AL:=AX / Op AH:=Resto	?				?	?	?	?	?
IDIV	División entera con signo	IDIV Op	Op=word: AX:=DX:AX / Op DX:=Resto	?				?	?	?	?	?
IDIV 386	División entera con signo	IDIV Op	Op=doublew.: EAX:=EDX:EAX / Op EDX:=Resto	?				?	?	?	?	?
MUL	Multiplicación (sin signo)	MUL Op	Op=byte: AX:=AL*Op si AH=0 •	±				?	?	?	?	±
MUL	Multiplicación (sin signo)	MUL Op	Op=word: DX:AX:=AX*Op si DX=0 •	±				?	?	?	?	±
MUL 386	Multiplicación (sin signo)	MUL Op	Op=double: EDX:EAX:=EAX*Op si EDX=0 •	±				?	?	?	?	±
IMUL i	Multiplic. entera con signo	IMUL Op	Op=byte: AX:=AL*Op si AL es suficiente •	±				?	?	?	?	±
IMUL	Multiplic. entera con signo	IMUL Op	Op=word: DX:=AX:=AX*Op si AX es suficiente •	±				?	?	?	?	±
IMUL 386	Multiplic. entera con signo	IMUL Op	Op=double: EDX:EAX:=EAX*Op si EAX es sufi. •	±				?	?	?	?	±
INC	Incrementar	INC Op	Op:=Op+1 (El Carry no resulta afectado !)	±				±	±	±	±	
DEC	Decrementar	DEC Op	Op:=Op-1 (El Carry no resulta afectado !)	±				±	±	±	±	
CMP	Comparar	CMP Op1, Op2	Op1-Op2	±				±	±	±	±	±
SAL	Desplazam. aritm. a la izq.	SAL Op, Cantidad		i				±	±	?	±	±
SAR	Desplazam. aritm. a la der.	SAR Op, Cantidad		i				±	±	?	±	±
RCL	Rotar a la izq. c/acarreo	SCL Op, Cantidad		i								±
RCR	Rotar a la derecha c/acarreo	RCR Op, Cantidad		i								±
ROL	Rotar a la izquierda	ROL Op, Cantidad		i								±
ROR	Rotar a la derecha	ROR Op, Cantidad		i								±

i para más información ver especificaciones de la instrucción entonces CF:=0, OF:=0 sino CF:=1, OF:=1

LÓGICOS				Flags								
Nombre	Comentario	Código	Operación	O	D	I	T	S	Z	A	P	C
NEG	Negación (Complemento a 2)	NEG Op	Op:=0-Op si Op=0 entonces CF:=0 sino CF:=1	±				±	±	±	±	±
NOT	Invertir cada bit	NOT Op	Op:=¬Op (invierte cada bit)									
AND	'Y' (And) lógico	AND Dest,Fuente	Dest:=Dest ∧ Fuente	0				±	±	?	±	0
OR	'O' (Or) lógico	OR Dest,Fuente	Dest:=Dest ∨ Fuente	0				±	±	?	±	0
XOR	'O' (Or) exclusivo	XOR Dest,Fuente	Dest:=Dest (xor) Fuente	0				±	±	?	±	0
SHL	Desplazam. lógico a la izq.	SHL Op, Cantidad		i				±	±	?	±	±
SHR	Desplazam. lógico a la der.	SHR Op, Cantidad		i				±	±	?	±	±

Consiga la última versión gratuita de [jegerlehner.ch/intel](http://jegerlehner.ch/intel) Esta página puede ser libremente distribuida sin costo alguno si no es modificada. Todos los derechos reservados.







# BIBLIOGRAFÍA

- Abel, Peter (1996). *Lenguaje ensamblador*. 3ª ed. México: Prentice Hall.
- Alexandridis, N. (1993). *Design of Microprocessor-Based Systems*. Texas: Prentice Hall.
- Gopal, K. (2007). *IBM 360 Assembler Language Programming*. New York: John Wiley & Sons.
- Lapsley, P; Bier, J; Shoham, A y Lee, E. (2006). *DSP Processor Fundamentals: Architectures and Features*. California: Design Technology, Inc.
- Mandado, E.(2009). *Electrónica y microcontroladores PIC*. Barcelona: Marcombo.
- Morris, M. (1994). *Arquitectura de computadoras*. Los Ángeles: Pearson.
- Valdés, F. (2007). *Microcontroladores: fundamentos y aplicaciones con PIC*. Barcelona: Marcombo.







