Nazareth Colmenares. 20-10017

Gaston Garcia. 19-10668

Universidad Simón Bolívar

Organización del Computador.

Profesor: Fernando Lovera

Proyecto 1 Septiembre – Diciembre 2025

Caja Registradora.

Este Proyecto implementa un Sistema de caja registradora en MIPS que procesa códigos numéricos, gestiona inventario y genera reportes de venta. Se tiene como motivación crear una solución eficiente que maneje las operaciones básicas de un punto de venta mientras mantiene un registro preciso del inventario.

Diseño de Estructuras de Datos:

- **Inventario:** Conjunto de etiquetas que hacen referencia a cuatro palabras: código numérico, stock, parte entera del precio, parte decimal del precio, también, el inventario se almacena en memoria estática comenzando en 0x10010000 con estructura fija de 32 bytes por producto:

    1. 4 bytes: Código UPC
    2. 4 bytes: Stock disponible
    3. 4 bytes: Precio parte entera
    4. 4 bytes: Precio parte decimal
    5. 16 bytes: Nombre del producto

- **Compra Actual**: Lista enlazada que almacena los productos siendo procesados en la transacción actual. Cada nodo referencia un producto del inventario y almacena la cantidad.

```
# Offset   Contenido        Tamaño    Descripción
# 0        Next             4 bytes   Puntero al siguiente nodo
# 4        Código           4 bytes   UPC del producto
# 8        Cantidad         4 bytes   Cantidad comprada
# 12       Parte Entera     4 bytes   Precio total parte entera
# 16       Parte Decimal    4 bytes   Precio total parte decimal
# 20       Nombre           16 bytes  Nombre del producto (16 chars)
```

Funciones Implementadas:

1. ILCompraActual: Inicializa la lista
2. ACCompraActual: Agrega nuevo producto a la cabeza
3. RLCompraActual: Recorre la lista hasta el último elemento
4. ELCompraActual: Elimina el último elemento de la lista

- **Ventas del Día**: Lista enlazada que guarda todos los productos vendidos y cuantos de esos productos fueron vendidos, permitiendo generar el reporte de cierre.

Dificultades Encontradas u otras decisiones:

- **Captación del código:** A la hora de que el programa lea el código, lo lee como una cadena de caracteres, en vez de como un entero. La razón es, porque nos permite reutilizar el algoritmo para captar el código. Esto debido a que el código puede ser solo números o operaciones.
- **Creación de listas enlazadas:** Para la creación de rutinas de inicialización, agregar elementos, recorrer listas y eliminar elementos, hicimos los siguientes pasos:
  Inicialización de las listas:
  1. Para inicializar la lista, creamos dos palabras**:** raíz y cabeza. La raíz guarda la dirección del primer elemento en el heap de la lista y la cabeza guarda la dirección en el heap donde puede ser agregado un nuevo elemento.
  2. Reservamos mediante **sbrk** una palabra en el heap, esa palabra va a tener la dirección en el heap del primer elemento.

  Agregar un elemento a la lista: La primera palabra de un elemento, será siempre el apuntador al siguiente elemento. Los elementos poseen 36 bytes y un elemento siempre se agrega al final o de ultimo en la lista.

  1. Guardar en un registro la dirección a la que apunte la cabeza.
  2. Mediante sbrk reservar 36 bytes: Los primeros 4 bytes son un apuntador al siguiente elemento, después 4 bytes para el código del producto, 4 bytes para la cantidad de ese producto que se está comprando, 4 bytes

3. para la parte entera del precio, 4 bytes para la parte decimal del precio, y 16 bytes para el nombre del producto.
4. Se actualiza la cabeza de la lista: se coloca en el apuntador del anterior elemento la dirección del elemento agregado.
5. Se van guardando los datos en el espacio reservado del heap.

Recorrer la lista:

1. Guardar los registros, llamémoslos $1 y $2: En $1, guardamos la dirección en la que apunta la raíz y en $2, guardamos la dirección a la que apunta $1.
2. Un ciclo en el que iteramos sobre $2, verificando si este es cero. Si no lo es, colocamos en $1 la primera palabra a la que apunta $1 (next de un elemento). Y luego, colocamos en $2, lo mismo. Si es cero es porque el ultimo elemento no apunta a ningún elemento y recorrimos la lista.

- **Gestión de Precios:** Separar precios en parte entera y decimal para evitar problemas de precisión con números flotantes en MIPS. Las operaciones aritméticas manejan manualmente el "carry" entre decimales y enteros.
- **Manejo de Cadenas y conversión:**
  Problema: Convertir strings de entrada a valores numéricos para códigos UPC y operadores.
  Solución: Implementación de ReadString que:
  1. Convierte caracteres ASCII a valores numéricos
  2. Maneja base decimal con multiplicaciones sucesivas
  3. Detecta fin de línea (carácter 10)
- **Sincronización entre Listas:**
  Problema: Transferencia de datos entre compra actual y cierre del día.
  Solución: Implementación de RLActual2Cierre que:
  1. Recorre lista actual
  2. Busca en lista de cierre
  3. Actualiza existentes o agrega nuevos

Nazareth Colmenares. 20-10017

Gaston Garcia. 19-10668

Universidad Simón Bolívar

Posibles mejoras futuras:

- Implementar tabla hash para búsquedas más eficientes en inventario grande.
- Agregar persistencia de datos para mantener inventario entre ejecuciones.
- Implementar manejo de archivos para el extra credit.

El programa se encuentra completamente operativo y cumple con todos los requerimientos especificados, gestionando eficientemente el inventario, procesando compras con operaciones de multiplicación y eliminación, calculando totales con precisión y manejando errores de entrada, mientras que las principales lecciones aprendidas incluyen la importancia del diseño previo de estructuras de datos, donde la decisión de nodos de 36 bytes demostró ser óptima, el manejo creativo de las limitaciones de MIPS mediante aritmética entera-decimal separada, y la aplicación de principios como modularidad y abstracción en bajo nivel, desarrollando competencias técnicas en gestión de memoria y punteros, así como profesionales en análisis de requerimientos y documentación.