Laboratorio 2

Carlos Alberto Medina Castillo

PROFESOR

FREDY ALEXANDER RIVERA VELEZ

Arquitectura de computadores y Laboratorio

Descripción:

El proyecto se enfoca en la creación de un programa en lenguaje ensamblador MIPS que utiliza la técnica de codificación digrama con un diccionario estático. Esta técnica es clave en la compresión de datos, mejorando la eficiencia de almacenamiento y transmisión.
Se diseño un diccionario estático que contiene símbolos individuales y digramas. El codificador digrama busca y codifica secuencias de dos símbolos en función de este diccionario. Si la secuencia está presente, se codifica con su índice; de lo contrario, codificamos el primer símbolo y continuamos la búsqueda. Este proceso se repite para la compresión de datos.

Planteamiento:

A continuación, se presentan algunas decisiones de diseño del sistema y la justificación de estas, en primer lugar, se decidió optar por codifica el proyecto en diferentes archivos fuente .asm y no desarrollarlo en un único archivo como usualmente se suele hacer, esta decisión se tomó con la intención de tener una mayor organización del proyecto y poder definir responsabilidades a cada archivo de código, en el que se encuentran métodos específicos de cada funcionalidad. En la Fig. 1 se muestra la estructura del proyecto “Encoder” y en la Fig. 2 del proyecto “Decoder”.

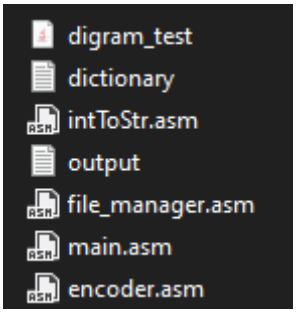


Fig. 1. Estructura proyecto Encoder

Descripción de responsabilidades de cada archivo de código.

Nombre archivo	Descripción
digram_test.java	Archivo de entrada que se pretende codificar
dictionary.txt	Archivo .txt que contiene el diccionario de digramas
intToStr.asm	Archivo que contiene la función Itoa (int to ASCII), la cual convierte un entero a código ASCII.
ouput.txt	Archivo de salida que contiene el mensaje codificado
file_manager.asm	Responsable de la manipulación de archivos

main.asm	Archivo de entrada del sistema, desde donde se hacen llamados a las demás funciones
encoder.asm	Responsable de la codificación del mensaje

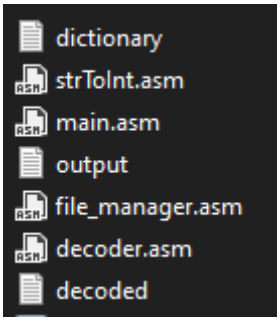


Fig. 2. Estructura proyecto decoder

Nombre archivo	**Descripción**
decoded.txt	Archivo de salida. contiene el mensaje decodificado
dictionary.txt	Archivo .txt que contiene el diccionario de digramas
strToInt.asm	Archivo que contiene la función atoi (ASCII to int), la cual convierte un carácter ASCII a su representación decimal.
ouput.txt	Archivo de salida que contiene el mensaje codificado
file_manager.asm	Responsable de la manipulación de archivos
main.asm	Archivo de entrada del sistema, desde donde se hacen llamados a las demás funciones
decoder.asm	Responsable de la decodificación del mensaje

Para la creación del diccionario se tuvo en cuenta las principales palabras claves del lenguaje de programación Java, tomando como referencia el sitio web https://www.w3schools.com/java/default.asp , Además se obtuvo archivos de código fuentes de aplicaciones desarrolladas en este lenguaje del sitio web Github.com, teniendo como fuente de datos esta información se procedió a desarrollar un algoritmo escrito en lenguaje de programación Python para detectar aquellos digramas que aparecían mas veces en los archivos de entrada Fig. 3, por lo que se tuvieron en

cuenta aquellos digramas que aparecían una mayor cantidad de veces, los cuales fueron agregados al diccionario.

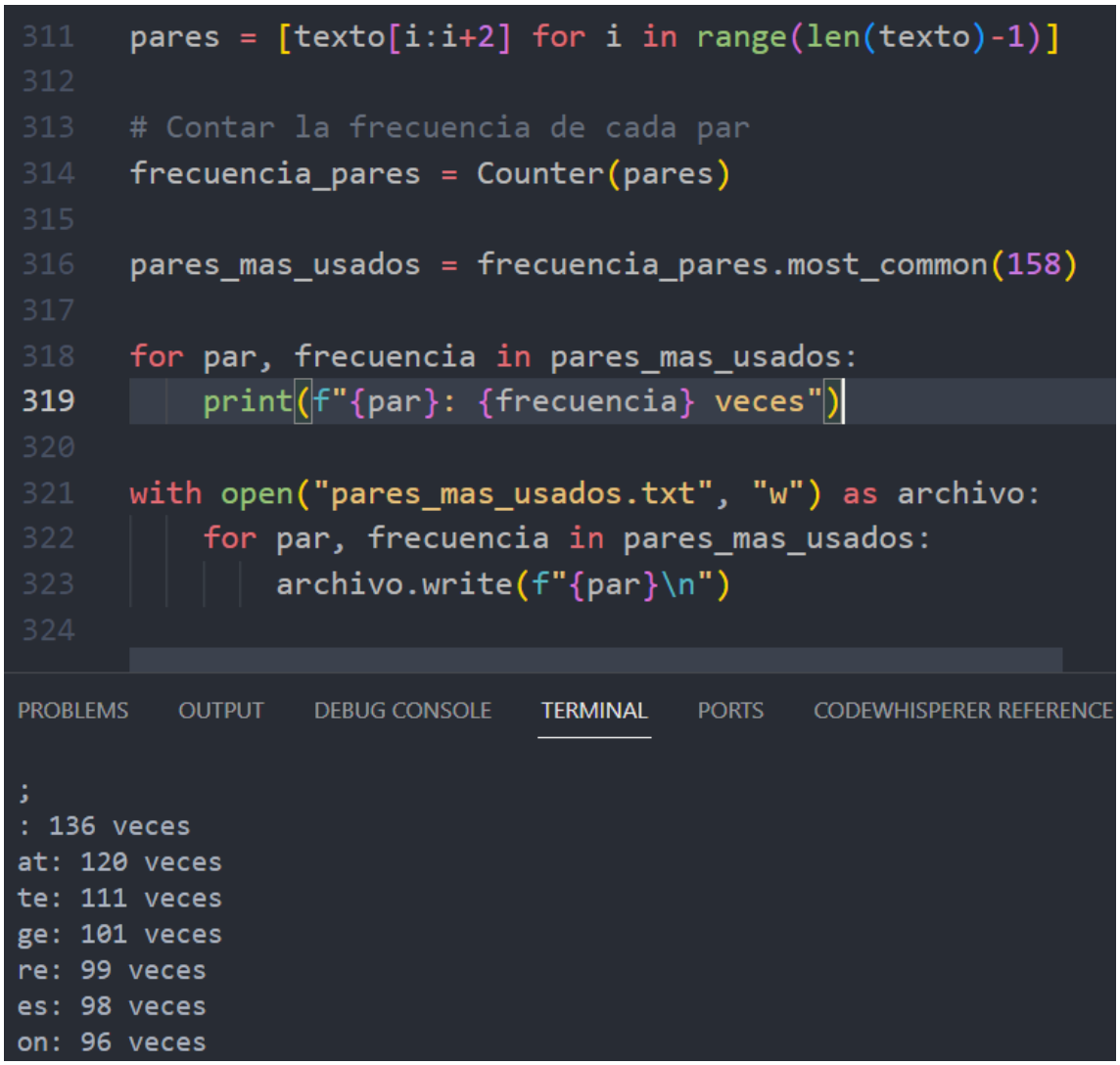


Fig. 3 Algoritmo para detectar digramas con mayor cantidad de apariciones

Pseudocodigo:

Decodificador:

Función decodificarMensaje(mensaje):
mensajeTemporal = mensaje
diccionarioTemporal = diccionarioOriginal
índice = 1
mensajeDecodificado = []

Para cada caracter en mensajeTemporal:

```
        valorCaracter = convertirAEntero(siguienteCaracter(mensajeTemporal))

        Si valorCaracter es igual a retornoCarro:
            índice = 13
            agregarALista(mensajeDecodificado, índice)
            avanzar(mensajeTemporal, 8)  # Continuar al siguiente caracter
            índice = 1

        Si valorCaracter es igual a saltoDeLinea:
            índice = 10
            agregarALista(mensajeDecodificado, índice)
            avanzar(mensajeTemporal, 8)  # Continuar al siguiente caracter
            índice = 1

        # Loop en el diccionario
        Mientras índice < valorCaracter:
            Si siguienteCaracter(diccionarioTemporal) es igual a retornoCarro:
                avanzar(diccionarioTemporal, 2)
            Sino:
                Si siguienteCaracter(diccionarioTemporal) es igual a saltoDeLinea:
                    avanzar(diccionarioTemporal, 2)
                Sino:
                    avanzar(diccionarioTemporal, 3)

            índice = índice + 1

        Si índice es igual a valorCaracter:
            valorEncontrado = siguienteCaracter(diccionarioTemporal)
            agregarALista(mensajeDecodificado, valorEncontrado)
            avanzar(diccionarioTemporal, 1)

            Si siguienteCaracter(diccionarioTemporal) es igual a retornoCarro:
                agregarALista(mensajeDecodificado, retornoCarro)
                avanzar(diccionarioTemporal, 2)
            Sino:
                Si siguienteCaracter(diccionarioTemporal) no es igual a
saltoDeLinea:
                    agregarALista(mensajeDecodificado,
siguienteCaracter(diccionarioTemporal))
                    avanzar(diccionarioTemporal, 1)

        Si valorCaracter es igual a 0:
            Salir del bucle  # Fin del mensaje
```

```
        avanzar(mensajeTemporal, 8)
        ReiniciarValores()

    Fin del bucle principal

    ReiniciarValores()
    avanzar(diccionarioTemporal, 1)  # Saltar el último carácter del diccionario
    volver al bucle principal

    Fin del bucle principal

    La lista mensajeDecodificado contiene el mensaje decodificado

Función siguienteCaracter(cadena):
    caracter = primer carácter en cadena
    avanzar(cadena, 1)
    Devolver caracter

Función avanzar(cadena, n):
    Mover el puntero en cadena n posiciones hacia adelante

Función agregarALista(lista, valor):
    Agregar valor al final de lista

Función convertirAEntero(cadena):
    Convertir cadena a entero y devolver el valor

Función ReiniciarValores():
    Restablecer los valores de índice y otros registros
```

Codificador:

```
Función codificarMensaje(mensaje):
    mensajeTemporal = mensaje
    diccionarioTemporal = diccionarioOriginal
    contador = 1
    mensajeCodificado = []

    Para cada caracter en mensajeTemporal:
        primerCaracterMensaje = siguienteCaracter(mensajeTemporal)
        segundoCaracterMensaje = siguienteCaracter(mensajeTemporal)
```

```
    Si primerCaracterMensaje es igual a retornoCarro:
        contador = 13
        agregarALista(mensajeCodificado, contador)
        siguienteCaracter(mensajeTemporal)  # Avanza una posición
        contador = 1

    Si primerCaracterMensaje es igual a saltoDeLinea:
        contador = 10
        agregarALista(mensajeCodificado, contador)
        siguienteCaracter(mensajeTemporal)  # Avanza una posición
        contador = 1

    Si primerCaracterMensaje es igual a finDeMensaje:
        Si segundoCaracterMensaje es igual a finDeMensaje:
            Continuar  # No hacer nada
        Sino:
            Continuar  # No hacer nada

    primerCaracterDiccionario = siguienteCaracter(diccionarioTemporal)
    segundoCaracterDiccionario = siguienteCaracter(diccionarioTemporal)

    Si primerCaracterMensaje es igual a primerCaracterDiccionario:
        Si segundoCaracterMensaje es igual a retornoCarro:
            contador = 13
        Sino:
            Si segundoCaracterMensaje es igual a segundoCaracterDiccionario:
                contador = 1
            Sino:
                contador = 0  # No coinciden
    Sino:
        contador = 0  # No coinciden

    Si contador es mayor que 0:
        banderaDígrafoEncontrado = Verdadero
        AumentarContador(mensajeCodificado)

Si contador es mayor que 0:
    Continuar  # No hacer nada

Sino:
    ReiniciarValores()

Si segundoCaracterMensaje es igual a retornoCarro:
```

```
            Avanzar(diccionarioTemporal, 2)
        Sino:
            Si segundoCaracterDiccionario es igual a saltoDeLinea:
                Avanzar(diccionarioTemporal, 2)
            Sino:
                Avanzar(diccionarioTemporal, 3)

        AumentarContador(contador)

        Si contador es menor o igual que tamañoDiccionario:
            Continuar  # Volver al bucle interno

        Si banderaDígrafoEncontrado es igual a Falso:
            ReiniciarValores()
            buscarDígrafo = Verdadero

        Continuar  # Volver al bucle principal

    ReiniciarValores()

    Si segundoCaracterMensaje no es igual a finDeMensaje:
        Continuar  # No hacer nada

    Fin de la función

    Función siguienteCaracter(cadena):
        caracter = primer carácter en cadena
        Avanzar(cadena, 1)
        Devolver caracter

    Función Avanzar(cadena, n):
        Mover el puntero en cadena n posiciones hacia adelante

    Función agregarALista(lista, valor):
        Agregar valor al final de lista

    Función AumentarContador(contador):
        Agregar contador al final de mensajeCodificado
```

Simulación del sistema:
Para la simulación del sistema se almacenaron los digramas en el archivo dictionary.txt y se definió como archivo de entrada código fuente de java, el cual fue cargado en digram_test.java Fig. 4.

```
public class Main {CRLF
    // Static methodCRLF
    static void myStaticMethod() {CRLF
        System.out.println("Static methods can be called without creating objects");CRLF
    }CRLF
CRLF
    // Public methodCRLF
    public void myPublicMethod() {CRLF
        System.out.println("Public methods must be called by creating objects");CRLF
    }CRLF
CRLF
    // Main methodCRLF
    public static void main(String[] args) {CRLF
        myStaticMethod(); // Call the static methodCRLF
        // myPublicMethod(); This would compile an errorCRLF
CRLF
        Main myObj = new Main(); // Create an object of MainCRLF
        myObj.myPublicMethod(); // Call the public method on the objectCRLF
    }CRLF
}
```

Fig. 4 Algoritmo escrito en java

Al realizar el proceso de codificación el sistema tiene como salida la codificación del mensaje, donde cada decimal representa la posición del digrama en el diccionario, en la Fig.5 se muestra una parte de la salida resultante del mensaje anterior.

```
142 ¨ 119 ¨ 124 ¨ 2 ¨ 175 ¨ 121 ¨ 86 ¨ 2 ¨ 177 ¨ 100 ¨ 234 ¨ 13
¨ 10 ¨ 219 ¨ 18 ¨ 18 ¨ 128 ¨ 127 ¨ 123 ¨ 151 ¨ 208 ¨ 129 ¨ 82 ¨
71 ¨ 13 ¨ 10 ¨ 219 ¨ 107 ¨ 108 ¨ 124 ¨ 2 ¨ 157 ¨ 158 ¨ 2 ¨ 80
¨ 92 ¨ 152 ¨ 108 ¨ 124 ¨ 48 ¨ 56 ¨ 168 ¨ 71 ¨ 194 ¨ 234 ¨ 13 ¨
10 ¨ 219 ¨ 219 ¨ 217 ¨ 107 ¨ 146 ¨ 17 ¨ 130 ¨ 87 ¨ 17 ¨ 178 ¨ 100
¨ 87 ¨ 79 ¨ 81 ¨ 11 ¨ 4 ¨ 152 ¨ 108 ¨ 124 ¨ 2 ¨ 208 ¨ 129 ¨
82 ¨ 206 ¨ 2 ¨ 154 ¨ 81 ¨ 2 ¨ 69 ¨ 72 ¨ 2 ¨ 154 ¨ 189 ¨ 156
¨ 2 ¨ 90 ¨ 120 ¨ 168 ¨ 190 ¨ 2 ¨ 70 ¨ 122 ¨ 108 ¨ 100 ¨ 74 ¨ 2
¨ 82 ¨ 69 ¨ 103 ¨ 134 ¨ 86 ¨ 4 ¨ 225 ¨ 13 ¨ 10 ¨ 219 ¨ 95 ¨
13 ¨ 10 ¨ 13 ¨ 10 ¨ 219 ¨ 18 ¨ 18 ¨ 2 ¨ 51 ¨ 117 ¨ 155 ¨ 151
¨ 208 ¨ 129 ¨ 82 ¨ 71 ¨ 13 ¨ 10 ¨ 219 ¨ 142 ¨ 119 ¨ 124 ¨ 2 ¨
157 ¨ 158 ¨ 2 ¨ 80 ¨ 92 ¨ 51 ¨ 117 ¨ 155 ¨ 70 ¨ 48 ¨ 184 ¨ 168
¨ 71 ¨ 194 ¨ 234 ¨ 13 ¨ 10 ¨ 219 ¨ 219 ¨ 217 ¨ 107 ¨ 146 ¨ 17 ¨
```

Fig. 5 Salida mensaje codificado

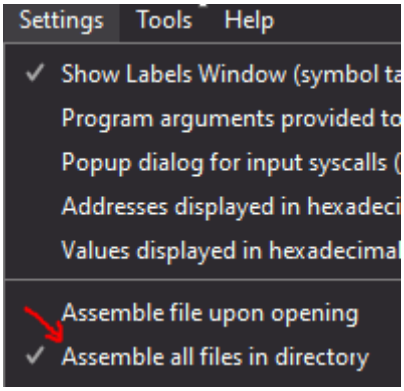
La salida anterior fue cargada en el proyecto Decoder, el cual tomara cada decimal almacenado en el mensaje codificado e realizara iteraciones hasta el valor de ese decimal en el diccionario para hallar su valor correspondiente, dando como salida el mensaje decodificado el cual se almacena en decoded.txt. Fig. 6.

```
public class Main {
  // Static method
  static void myStaticMUhod() {
    System.out.println("Static methods can be called without creating objects");
  }

  // Public method
  public void myPublicMethod() {
    System.out.println("Public methods must be called by creating objects");
  }

  // Main method
  public static void main(String[] args) {
    myStaticMethod(); // Call the static method
    // myPublicMethod(); This would compile an error

    Main myObj = new Main(); // Create an object of Main
    myObj.myPublicMethod(); // Call the public method on the object
  }
}
```

Fig. 6 Mensaje decodificado

Observaciones:

- Para la ejecución del programa, se requiere poseer todos los archivos en descritos anteriormente en en el mismo directorio de cada proyecto Encoder y Decoder.
- El sistema tiene como salidas archivo con extensión .txt, y como entrada archivos con esta misma extensión exceptuando el mensaje a codificar cuya extensión del archivo es .java.
- El tamaño máximo de lectura y escritura que soporta el sistema es archivos de 4096 bytes o 4kb.
- Para que MARS detecte todos los archivos de código fuente.asm que se encuentran en el directorio, debe habilitar la opción “Assemble all files in directory” desde el menú Settings, así:



Conclusiones:

La estructura organizativa del proyecto en varios archivos ha facilitado el desarrollo y el mantenimiento del código, mejorando la claridad y la gestión de responsabilidades. El potencial de aplicación de esta técnica va más allá de lo académico, ya que puede ser relevante en escenarios prácticos que requieran una gestión eficiente de datos, como la transmisión de información a través de redes y el almacenamiento en dispositivos con limitaciones de espacio.