# Tabla de Contenidos

# DEC

# Guía Técnica Completa: Sistema de Archivos RT-11 y Extractor

*Análisis Técnico Completo del Sistema de Archivos RT-11 y Documentación del Extractor*

**Documento Técnico**
RT-11 Extractor Project

*Preservación de Software Histórico*
Digital Equipment Corporation (DEC)
PDP-11 / RT-11 Operating System

Versión 1.0 - June 2025

*"Este documento forma parte de un esfuerzo de preservación del software histórico, permitiendo el acceso a miles de programas desarrollados para las minicomputadoras PDP-11 de Digital Equipment Corporation entre 1970 y 1990."*

# Tabla de Contenidos



# Introducción

RT-11 (Real-Time 11) fue uno de los sistemas operativos más importantes de la era de las minicomputadoras, desarrollado por Digital Equipment Corporation (DEC) para sus computadoras PDP-11. Este documento explica en detalle cómo funciona su sistema de archivos y cómo nuestro extractor decodifica las imágenes de disco RT-11.

## ¿Por qué es importante RT-11?

- **Simplicidad**: RT-11 fue diseñado para ser simple pero eficiente
- **Tiempo Real**: Optimizado para aplicaciones de tiempo real
- **Longevidad**: Se usó desde 1973 hasta los años 90
- **Preservación**: Miles de programas históricos están almacenados en discos RT-11

## Historia y Contexto

### El Ecosistema PDP-11

Las computadoras PDP-11 de DEC fueron revolucionarias en los años 70. RT-11 fue diseñado específicamente para:

- **Aplicaciones de tiempo real**: Control de procesos industriales
- **Sistemas embebidos**: Equipos médicos, instrumentos científicos
- **Desarrollo de software**: Compiladores, ensambladores, editores
- **Educación**: Universidades y laboratorios de investigación

### Características del Sistema de Archivos RT-11

RT-11 utiliza un sistema de archivos **plano** (flat filesystem) con estas características:

- **Sin subdirectorios**: Todos los archivos están en el directorio raíz
- **Nombres cortos**: Máximo 6 caracteres + 3 de extensión
- **Orientado a bloques**: Todo se organiza en bloques de 512 bytes
- **Directorio contíguo**: El directorio está en una ubicación fija del disco

## Estructura Física del Disco RT-11

### Anatomía de un Disco RT-11

```
+------------------+ Bloque 0 | Boot Block | Sector de arranque (opcional)
+------------------+ Bloque 1 | Home Block | Información del volumen
+------------------+ Bloque 2-5 | Reserved | Bloques reservados
+------------------+ Bloque 6+ | Directory | Directorio del sistema de
archivos | Segments | +------------------+ Variable | | | File Data |
```

```
Datos de los archivos | Area | | | +------------------+
```

### Tamaños de Bloque y Sector

RT-11 está diseñado alrededor del concepto de **bloques de 512 bytes**:

- **1 bloque = 512 bytes = 1 sector** (en la mayoría de dispositivos)
- **Todos los tamaños** se expresan en bloques
- **Alineación**: Todo está alineado a límites de bloque

### Tipos de Dispositivos Comunes

| Dispositivo | Descripción | Tamaño Típico | Sectores |
|---|---|---|---|
| RX01 | Diskette 8" | 256 KB | 77 tracks × 26 sectors |
| RX02 | Diskette 8" doble densidad | 512 KB | 77 tracks × 26 sectors × 2 |
| RK05 | Disco removible | 2.5 MB | 203 tracks × 12 sectors |
| RL01 | Disco removible | 5.2 MB | Variable |
| RL02 | Disco removible | 10.4 MB | Variable |

## El Home Block: Corazón del Sistema

### ¿Qué es el Home Block?

El **Home Block** (Bloque 1) contiene la información esencial sobre el volumen RT-11. Es como el "certificado de nacimiento" del disco.

### Estructura del Home Block

```
Offset Tamaño Descripción ------ ------ ----------- +0 2 Palabra de validación (opcional) +2 2 Cluster size (tamaño de cluster) +4 2 Primer bloque del directorio +6 2 Versión del sistema +8 2 Identificación del volumen +10 12 Nombre del propietario +22 490 Información adicional (variable)
```

### Ejemplo de Análisis del Home Block

```
def analizar_home_block(data): """ Analiza el contenido de un Home Block
RT-11 """ # Los primeros bytes pueden contener información del sistema
words = struct.unpack('<256H', data) # 256 palabras de 16 bits # Buscar el
bloque de inicio del directorio for i, word in enumerate(words[:50]): if 2
<= word <= 50: # Rango razonable para directorio primer_directorio = word
break else: primer_directorio = 6 # Valor por defecto return {
'primer_directorio': primer_directorio, 'tamaño_disco': len(data_completa)
// 512, 'formato': 'RT-11 estándar' }
```

## Directorio RT-11: Organización de Archivos

### Concepto de Segmentos de Directorio

El directorio RT-11 está organizado en **segmentos** de 1024 bytes (2 bloques). Cada segmento puede contener múltiples entradas de archivo.

### Estructura de un Segmento de Directorio

```
+------------------+ 0-9 bytes | Header | Cabecera del segmento (5
palabras) | (5 words) | +------------------+ 10+ bytes | Entry 1 | Primera
entrada de archivo | (14+ bytes) | +------------------+ | Entry 2 |
Segunda entrada de archivo | (14+ bytes) | +------------------+ | ... |
Más entradas... +------------------+ | End Marker | Marcador de fin de
segmento +------------------+
```

### Cabecera del Segmento (10 bytes)

```
Palabra Descripción ------- ----------- 0 Número total de segmentos 1
Número del siguiente segmento (0 = último) 2 Segmento más alto usado 3
Bytes extra por entrada 4 Bloque de inicio del área de datos
```

### Ejemplo de Cabecera

```
Segmento 3: total_segs=31, next=2, highest=3, extra=0, start_blk=68
```

Esto significa:

- **total_segs=31**: Hay 31 segmentos de directorio en total
- **next=2**: El próximo segmento a leer es el segmento 2

- **highest=3**: El segmento más alto usado es el 3
- **extra=0**: No hay bytes extra en las entradas
- **start_blk=68**: Los datos de archivos empiezan en el bloque 68

## Estructura de una Entrada de Archivo

### Formato Estándar de Entrada (14 bytes)

```
Offset Tamaño Descripción ------ ------ ----------- +0 2 Palabra de estado (status) +2 2 Nombre archivo (parte 1, RADIX-50) +4 2 Nombre archivo (parte 2, RADIX-50) +6 2 Extensión archivo (RADIX-50) +8 2 Longitud en bloques +10 2 Job/Channel (información del trabajo) +12 2 Fecha de creación
```

### Palabras de Estado: El ADN del Archivo

La palabra de estado define qué tipo de entrada es:

| Valor (Octal) | Valor (Hex) | Significado |
|---|---|---|
| 0 | 0x0000 | Archivo permanente normal |
| 400 | 0x0100 | Archivo tentativo (temporal) |
| 1000 | 0x0200 | Área no utilizada |
| 2000 | 0x0400 | Archivo permanente |
| 4000 | 0x0800 | Marcador de fin de segmento |
| 102000 | 0x8400 | Archivo permanente protegido |

### Interpretación de Estados

```
def interpretar_estado(status_word): """ Interpreta la palabra de estado de una entrada RT-11 """ if status_word & 0x800: # 4000 octal return "fin_de_segmento" elif status_word & 0x400: # 2000 octal if status_word & 0x8000: # 100000 octal return "permanente_protegido" else: return "permanente" elif status_word & 0x200: # 1000 octal return "no_utilizado" elif status_word & 0x100: # 400 octal return "tentativo" else: return "desconocido"
```

# Codificación RADIX-50: El Alfabeto de RT-11

## ¿Qué es RADIX-50?

RADIX-50 es un sistema de codificación que permite almacenar 3 caracteres en una palabra de 16 bits. Fue inventado por DEC para ahorrar espacio en memoria y disco.

## El Alfabeto RADIX-50

```
Índice: 01234567890123456789012345678901234567890123456789 Chars: "
ABCDEFGHIJKLMNOPQRSTUVWXYZ$.?0123456789"
```

Los caracteres válidos son:

- **Espacio** (índice 0)
- **A-Z** (índices 1-26)
- **$, ., ?** (índices 27-29)
- **0-9** (índices 30-39)

## Algoritmo de Decodificación

```
def decodificar_radix50(word): """ Decodifica una palabra de 16 bits
RADIX-50 a 3 caracteres """ RADIX50_CHARS = '
ABCDEFGHIJKLMNOPQRSTUVWXYZ$.?0123456789' if word == 0: return ' ' # Tres
espacios # Extraer los 3 caracteres c3 = word % 40 c2 = (word // 40) % 40
c1 = (word // 1600) % 40 # Convertir índices a caracteres resultado =
RADIX50_CHARS[c1] + RADIX50_CHARS[c2] + RADIX50_CHARS[c3] return resultado
```

## Ejemplo de Decodificación de Nombre de Archivo

Un archivo llamado "HELLO.TXT" se almacena así:

1. "**HEL**" → RADIX-50 → `word1`

2. "**LO** " → RADIX-50 → `word2` (con espacio al final)

3. "**TXT**" → RADIX-50 → `word3`

```
# Ejemplo con valores reales word1 = 0x48C5 # "HEL" word2 = 0x4C4F # "LO " word3 = 0x5458 # "TXT" parte1 = decodificar_radix50(word1) # "HEL" parte2 = decodificar_radix50(word2) # "LO " parte3 = decodificar_radix50(word3) # "TXT" nombre = (parte1 + parte2).strip() # "HELLO" extension = parte3.strip() # "TXT" archivo_completo = f"{nombre}.{extension}" # "HELLO.TXT"
```

## Formato de Fechas RT-11

### Estructura de la Fecha (16 bits)

RT-11 almacena fechas en una sola palabra de 16 bits:

```
Bits Contenido ---- --------- 0-4 Año base (0-31, desde 1972) 5-9 Día (1-31) 10-13 Mes (1-12) 14-15 Era (0-3, cada era = 32 años)
```

### Algoritmo de Decodificación de Fecha

```
def decodificar_fecha_rt11(date_word): """ Decodifica una fecha RT-11 de 16 bits """ if date_word == 0: return None # Extraer campos año_base = date_word & 0x1F # bits 0-4 dia = (date_word >> 5) & 0x1F # bits 5-9 mes = (date_word >> 10) & 0x0F # bits 10-13 era = (date_word >> 14) & 0x03 # bits 14-15 # Calcular año real año = año_base + 1972 + (era * 32) # Validar y corregir valores inválidos if dia == 0: dia = 1 if mes == 0: mes = 1 # Verificar rango válido if 1 <= mes <= 12 and 1 <= dia <= 31 and 1972 <= año <= 2099: return f"{año:04d}-{mes:02d}-{dia:02d}" return None
```

### Ejemplos de Fechas

| Valor Hex | Binario | Año | Mes | Día | Fecha |
|---|---|---|---|---|---|
| 0x3425 | 0011010000100101 | 1977 | 1 | 5 | 1977-01-05 |
| 0x8C8D | 1000110010001101 | 1985 | 4 | 13 | 1985-04-13 |

## Tipos de Archivo y Estados

### Clasificación por Extensión

RT-11 identifica tipos de archivo principalmente por su extensión:

| Extensión | Descripción | Uso Típico |
|---|---|---|
| .SAV | Programa ejecutable | Aplicaciones, utilidades |
| .OBJ | Archivo objeto | Código compilado no enlazado |
| .MAC | Código fuente MACRO | Programas en ensamblador |
| .FOR | Código fuente FORTRAN | Programas científicos |
| .REL | Biblioteca relocatable | Bibliotecas de código |
| .SYS | Archivo del sistema | Drivers, monitores |
| .TMP | Archivo temporal | Archivos de trabajo |
| .BAK | Copia de respaldo | Respaldos automáticos |
| .DAT | Archivo de datos | Datos de aplicaciones |
| .TXT | Archivo de texto | Documentación |

## Estados de Archivo

#### Archivos Permanentes

- **Estado**: Normal (0x0000) o Marcado (0x0400)
- **Características**: Archivo completamente escrito y cerrado
- **Extracción**: Siempre se extraen

#### Archivos Tentativios

- **Estado**: 0x0100 (400 octal)
- **Características**: Archivo abierto para escritura, no cerrado
- **Causa**: Programa terminó antes de cerrar el archivo
- **Extracción**: Opcional (pueden estar incompletos)

#### Archivos Protegidos

- **Estado**: 0x8400 (102000 octal)
- **Características**: Archivo protegido contra escritura
- **Uso**: Archivos críticos del sistema
- **Extracción**: Normal, pero marcados como protegidos

# Proceso de Extracción: Cómo Funciona Nuestro Software

## Fase 1: Carga y Validación de la Imagen

```
def cargar_imagen(ruta_imagen): """ Carga y valida una imagen de disco RT-11 """ # 1. Cargar archivo completo en memoria with open(ruta_imagen, 'rb') as f: datos_imagen = f.read() # 2. Validar tamaño mínimo if len(datos_imagen) < 512 * 10: raise Error("Imagen demasiado pequeña para RT-11") # 3. Calcular número de bloques num_bloques = len(datos_imagen) // 512 # 4. Detectar tipo de dispositivo por tamaño tipo_dispositivo = detectar_dispositivo(len(datos_imagen)) return datos_imagen, num_bloques, tipo_dispositivo
```

## Fase 2: Análisis del Home Block

```
def analizar_home_block(datos_imagen): """ Analiza el Home Block (bloque 1) de la imagen """ # Leer bloque 1 (512 bytes desde offset 512) home_data = datos_imagen[512:1024] # Buscar información del directorio words = struct.unpack('<256H', home_data) # Encontrar bloque de inicio del directorio primer_directorio = 6 # Por defecto for word in words[:50]: if 2 <= word <= 50: primer_directorio = word break return { 'primer_directorio': primer_directorio, 'valido': True }
```

## Fase 3: Lectura del Directorio Multi-Segmento

```
def leer_directorio_completo(datos_imagen, primer_bloque): """ Lee todos los segmentos del directorio RT-11 """ archivos = [] segmento_actual = 1 segmentos_visitados = set() while segmento_actual and segmento_actual not in segmentos_visitados: # Calcular offset del segmento offset_segmento = (primer_bloque + segmento_actual - 1) * 512 # Leer datos del segmento (1024 bytes = 2 bloques) datos_segmento = datos_imagen[offset_segmento:offset_segmento + 1024] # Procesar segmento archivos_segmento, siguiente = procesar_segmento(datos_segmento) archivos.extend(archivos_segmento) # Marcar como visitado y continuar segmentos_visitados.add(segmento_actual) segmento_actual = siguiente return archivos
```

## Fase 4: Procesamiento de Segmento Individual

```
def procesar_segmento(datos_segmento): """ Procesa un segmento individual del directorio """ # Leer cabecera (10 bytes = 5 palabras) header = struct.unpack('<5H', datos_segmento[:10]) total_segs, next_seg, highest_seg, extra_bytes, start_block = header # Calcular tamaño de entrada tamaño_entrada = 14 + extra_bytes archivos = [] offset = 10 # Después de la cabecera while offset <= 1024 - tamaño_entrada: # Leer entrada entrada_data = datos_segmento[offset:offset + tamaño_entrada] entrada = struct.unpack('<7H', entrada_data[:14]) status, word1, word2, word3, longitud, job_chan, fecha = entrada # Verificar fin de segmento if status & 0x800: # Marcador de fin break # Procesar solo archivos válidos if status & 0x400 or status & 0x100: # Permanente o tentativo archivo = procesar_entrada_archivo(entrada, offset) if archivo: archivos.append(archivo) offset += tamaño_entrada return archivos, next_seg if next_seg > 0 else None
```

### Fase 5: Decodificación de Entrada Individual

```
def procesar_entrada_archivo(entrada, offset): """ Procesa una entrada individual de archivo """ status, word1, word2, word3, longitud, job_chan, fecha = entrada # Decodificar nombre usando RADIX-50 parte1 = decodificar_radix50(word1) parte2 = decodificar_radix50(word2) parte3 = decodificar_radix50(word3) nombre = (parte1 + parte2).strip() extension = parte3.strip() # Validar nombre if not nombre: return None # Construir nombre completo nombre_completo = f"{nombre}.{extension}" if extension else nombre # Decodificar fecha fecha_str = decodificar_fecha_rt11(fecha) if fecha else None # Determinar tipo de archivo tipo_archivo = determinar_tipo_archivo(status) return { 'nombre': nombre_completo, 'tamaño_bloques': longitud, 'tamaño_bytes': longitud * 512, 'estado': status, 'tipo': tipo_archivo, 'fecha_creacion': fecha_str, 'job_channel': job_chan, 'offset': offset }
```

### Fase 6: Cálculo de Posiciones de Archivo

```
def calcular_posiciones_archivos(archivos): """ Calcula las posiciones físicas de los archivos en el disco """ # Agrupar por segmento por_segmento = {} for archivo in archivos: seg = archivo['segmento'] if seg not in por_segmento: por_segmento[seg] = [] por_segmento[seg].append(archivo) # Calcular posiciones para cada segmento for segmento, lista_archivos in por_segmento.items(): # Ordenar por offset en el directorio lista_archivos.sort(key=lambda x: x['offset']) # Calcular posición inicial bloque_actual = archivo['bloque_inicio_segmento'] for archivo in lista_archivos: archivo['bloque_inicio'] = bloque_actual bloque_actual += archivo['tamaño_bloques']
```

### Fase 7: Extracción de Archivos

```
def extraer_archivo(datos_imagen, archivo, directorio_salida): """ Extrae un archivo individual de la imagen """ # Calcular posición física offset_inicio = archivo['bloque_inicio'] * 512 tamaño_bytes = archivo['tamaño_bloques'] * 512 # Leer datos del archivo datos_archivo = datos_imagen[offset_inicio:offset_inicio + tamaño_bytes] # Crear archivo de salida ruta_salida = directorio_salida / archivo['nombre'] # Manejar conflictos de nombres contador = 1 ruta_original = ruta_salida while ruta_salida.exists(): stem = ruta_original.stem suffix = ruta_original.suffix ruta_salida = ruta_original.parent / f"{stem}_{contador}{suffix}" contador += 1 # Escribir archivo with open(ruta_salida, 'wb') as f: f.write(datos_archivo) # Aplicar fecha original si está disponible if archivo['fecha_creacion']: aplicar_fecha_original(ruta_salida, archivo['fecha_creacion']) return True
```

## Casos Especiales y Manejo de Errores

### Detección de Bloques Dañados

```
def detectar_bloques_dañados(datos_imagen): """ Detecta bloques potencialmente dañados """ bloques_dañados = [] for i in range(0, len(datos_imagen), 512): bloque = datos_imagen[i:i+512] # Bloque completamente en ceros (sospechoso) if bloque == b'\x00' * 512: bloques_dañados.append(i // 512) # Bloque con patrón repetitivo (sospechoso) elif len(set(bloque)) < 5: bloques_dañados.append(i // 512) return bloques_dañados
```

### Recuperación de Archivos Parciales

```
def extraer_con_recuperacion(datos_imagen, archivo): """ Extrae un archivo intentando recuperar bloques dañados """ datos_archivo = b'' bloques_leidos = 0 bloques_fallidos = 0 for i in range(archivo['tamaño_bloques']): bloque_num = archivo['bloque_inicio'] + i offset = bloque_num * 512 try: # Intentar leer bloque bloque = datos_imagen[offset:offset + 512] # Verificar si el bloque parece válido if es_bloque_valido(bloque): datos_archivo += bloque bloques_leidos += 1 else: # Bloque dañado - rellenar con ceros datos_archivo += b'\x00' * 512 bloques_fallidos += 1 except IndexError: # Bloque más allá del final de la imagen datos_archivo += b'\x00' * 512 bloques_fallidos += 1 return datos_archivo, bloques_leidos, bloques_fallidos
```

### Validación de Integridad

```
def validar_sistema_archivos(archivos): """ Valida la integridad del sistema de archivos """ errores = [] advertencias = [] # Verificar solapamiento de archivos archivos_ordenados = sorted(archivos, key=lambda x: x['bloque_inicio']) for i in range(len(archivos_ordenados) - 1): actual = archivos_ordenados[i] siguiente = archivos_ordenados[i + 1] fin_actual = actual['bloque_inicio'] + actual['tamaño_bloques'] if fin_actual > siguiente['bloque_inicio']: errores.append(f"Solapamiento: {actual['nombre']} y {siguiente['nombre']}") # Verificar nombres válidos for archivo in archivos: if not es_nombre_valido_rt11(archivo['nombre']): advertencias.append(f"Nombre inválido: {archivo['nombre']}") # Verificar tamaños razonables for archivo in archivos: if archivo['tamaño_bloques'] > 65535: advertencias.append(f"Tamaño sospechoso: {archivo['nombre']}") return errores, advertencias
```

## Ejemplos Prácticos

### Ejemplo 1: Decodificar un Nombre de Archivo Manualmente

Supongamos que encontramos estas tres palabras en una entrada de directorio:

```
Word 1: 0x4845 (HE) Word 2: 0x4C4C (LL) Word 3: 0x4F54 (OT)
```

Proceso de decodificación:

```
# Decodificar cada palabra RADIX-50 word1 = 0x4845 parte1 =
decodificar_radix50(word1) # 0x4845 = 18501 decimal # c1 = 18501 // 1600 =
11 -> 'L' # Error en cálculo # Cálculo correcto: # c1 = 18501 // 1600 = 11
-> RADIX50_CHARS[11] = 'K' # c2 = (18501 // 40) % 40 = 462 % 40 = 22 ->
'V' # c3 = 18501 % 40 = 21 -> 'U' # Vamos a usar el método correcto: def
ejemplo_decodificacion(): words = [0x4845, 0x4C4C, 0x4F54] for i, word in
enumerate(words): resultado = decodificar_radix50(word) print(f"Word {i+1}
(0x{word:04X}): '{resultado}'")
```

### Ejemplo 2: Interpretar una Fecha RT-11

Fecha encontrada: `0x3425`

```
def ejemplo_fecha(): date_word = 0x3425 # 13349 decimal # Extraer campos
bit a bit año_base = date_word & 0x1F # 13349 & 31 = 5 dia = (date_word >>
5) & 0x1F # (13349 >> 5) & 31 = 417 & 31 = 1 mes = (date_word >> 10) &
0x0F # (13349 >> 10) & 15 = 13 & 15 = 13 (¡error!) era = (date_word >> 14)
& 0x03 # (13349 >> 14) & 3 = 0 # mes = 13 es inválido, corregir a 1 if mes
> 12: mes = 1 año = año_base + 1972 + (era * 32) # 5 + 1972 + 0 = 1977
print(f"Fecha: {año:04d}-{mes:02d}-{dia:02d}") # 1977-01-01
```

### Ejemplo 3: Analizar una Cabecera de Segmento

Datos de cabecera encontrados:

```
Bytes: 1F 00 02 00 03 00 00 00 44 00
```

Interpretación:

```
def ejemplo_cabecera(): datos = bytes.fromhex("1F00020003000000440")
header = struct.unpack('<5H', datos) total_segs, next_seg, highest_seg,
extra_bytes, start_block = header print(f"Total segmentos: {total_segs}")
# 31 print(f"Siguiente segmento: {next_seg}") # 2 print(f"Segmento más
alto: {highest_seg}") # 3 print(f"Bytes extra: {extra_bytes}") # 0
print(f"Bloque de inicio: {start_block}") # 68
```

## Apéndices

### Apéndice A: Tabla Completa RADIX-50

| Índice | Carácter | Binario | Descripción |
|---|---|---|---|

| | | | |
|---|---|---|---|
| 0 | espacio | 000000 | Carácter de relleno |
| 1-26 | A-Z | 000001-011010 | Letras mayúsculas |
| 27 | $ | 011011 | Símbolo de dólar |
| 28 | . | 011100 | Punto (separador) |
| 29 | ? | 011101 | Interrogación |
| 30-39 | 0-9 | 011110-100111 | Dígitos |

## Apéndice B: Códigos de Estado RT-11

| Valor Octal | Valor Hex | Bits Set | Descripción |
|---|---|---|---|
| 0 | 0x0000 | ninguno | Archivo permanente normal |
| 400 | 0x0100 | bit 8 | Archivo tentativo |
| 1000 | 0x0200 | bit 9 | Área no utilizada |
| 2000 | 0x0400 | bit 10 | Archivo permanente |
| 4000 | 0x0800 | bit 11 | Fin de segmento |
| 100000 | 0x8000 | bit 15 | Bit de protección |
| 102000 | 0x8400 | bits 10,15 | Permanente protegido |

## Apéndice C: Extensiones de Archivo Comunes

| Extensión | Descripción | Herramienta Típica |
|---|---|---|
| .SAV | Programa ejecutable | LINK |
| .OBJ | Archivo objeto | MACRO |
| .REL | Biblioteca relocatable | LIBR |
| .MAC | Código fuente MACRO | EDIT |
| .FOR | Código fuente FORTRAN | F4P |
| .LST | Listado de compilación | MACRO |
| .MAP | Mapa de enlazado | LINK |
| .STB | Tabla de símbolos | MACRO |
| .TMP | Archivo temporal | varios |
| .BAK | Copia de respaldo | EDIT |

## Apéndice D: Tipos de Dispositivo RT-11

| Código | Nombre | Descripción | Tamaño Típico |
|---|---|---|---|
| DX | RX01 | Diskette 8" simple | 256 KB |
| DY | RX02 | Diskette 8" doble | 512 KB |
| DK | RK05 | Disco removible | 2.5 MB |
| DL | RL01/RL02 | Disco removible | 5-10 MB |
| DB | RK06/RK07 | Disco grande | 14-28 MB |
| DM | RM02/RM03 | Disco masivo | 67-256 MB |

### Apéndice E: Herramientas de Diagnóstico

```
def diagnosticar_imagen_rt11(ruta_imagen): """ Realiza un diagnóstico completo de una imagen RT-11 """ with open(ruta_imagen, 'rb') as f: datos = f.read() print(f"=== Diagnóstico de {ruta_imagen} ===") print(f"Tamaño total: {len(datos):,} bytes") print(f"Bloques totales: {len(datos) // 512}") # Analizar Home Block home_data = datos[512:1024] print(f"\\nHome Block (bloque 1):") print(f" Primeros 16 bytes: {home_data[:16].hex()}") # Buscar directorio primer_dir = encontrar_directorio(datos) print(f"\\nDirectorio encontrado en bloque: {primer_dir}") # Analizar primer segmento offset_dir = primer_dir * 512 segmento_data = datos[offset_dir:offset_dir + 1024] header = struct.unpack('<5H', segmento_data[:10]) print(f"\\nPrimer segmento:") print(f" Total segmentos: {header[0]}") print(f" Siguiente: {header[1]}") print(f" Más alto: {header[2]}") print(f" Bytes extra: {header[3]}") print(f" Bloque inicio: {header[4]}") # Contar archivos archivos = escanear_directorio_rapido(datos, primer_dir) print(f"\\nArchivos encontrados: {len(archivos)}") for archivo in archivos[:5]: # Mostrar primeros 5 print(f" {archivo['nombre']} ({archivo['tamaño_bloques']} bloques)") if len(archivos) > 5: print(f" ... y {len(archivos) - 5} más")
```

## Conclusión

El sistema de archivos RT-11 representa una elegante solución de los años 70 para el almacenamiento eficiente en minicomputadoras. Su diseño simple pero robusto ha permitido que miles de programas históricos sobrevivan hasta hoy.

Nuestro extractor RT-11 implementa todas estas especificaciones con precisión, permitiendo la recuperación confiable de software histórico. Al entender estos detalles técnicos, podemos:

1. **Preservar historia**: Recuperar software que de otra manera se perdería

2. **Depurar problemas**: Diagnosticar imágenes de disco dañadas

3. **Optimizar extracción**: Manejar casos especiales y errores

4. **Educar**: Enseñar sobre sistemas históricos de computación

La simplicidad de RT-11 es su fortaleza: sin la complejidad de sistemas modernos, cada byte tiene un propósito claro y documentado, haciendo posible esta ingeniería inversa precisa.

*Documento técnico generado para el proyecto RT-11 Extractor*

*Versión 1.0 - Diciembre 2024*