Laboratorio 4: Arquitectura y Organización de Computadores

07 de noviembre 2025

1 Reglas Generales

Para la siguiente tarea se debe utilizar el Simulador RARS[1] para crear un programa que cumpla con los requerimientos de la sección 2. Se exigirá que el formato de los códigos se presente de la forma más limpia y ordenada posible. Deberá incluir un README con la identificación de los estudiantes que desarrollaron la tarea, además de cualquier supuesto utilizado.

2 Criptosistema del Núcleo Central

¡ALERTA MÁXIMA, AGENTE!

Gracias a tu sistema de rastreo secuencial del Lab 3, CyberSecure logró interceptar a varios miembros de la organización criminal. Sin embargo, ByteMaster ha descubierto algo mucho más grave:

La organización criminal está ejecutando su plan final: "PROTOCOLO GENESIS"
Este protocolo consiste en:

1. **Transferir todos sus fondos ilícitos** (millones de dólares en criptomonedas) a cuentas offshore inrastreables
2. **Liberar un malware autoreplicante** que destruirá las bases de datos de evidencia de CyberSecure
3. **Activar un sistema de borrado automático** que eliminará todas las pruebas digitales en 72 horas

La buena noticia: interceptaste los últimos mensajes encriptados que contienen:

- La estructura jerárquica de la organización
- Las claves de autenticación del servidor central

La mala noticia: están protegidos con algoritmos complejos de procesamiento y necesitas analizarlos en tiempo real usando programación de bajo nivel.

Tu misión: Desarrollar dos subsistemas críticos en ensamblador RISC-V para desmantelar el Protocolo Genesis antes de que sea demasiado tarde.

[1] Disponible en https://github.com/TheThirdOne/rars/releases

3 Subsistemas a Desarrollar

3.1 Desafío 1: Desciframiento de Jerarquía Criminal con Recursión

ByteMaster descubrió que la organización criminal usa un sistema de encriptación jerárquico basado en la Secuencia de Collatz modificada para ocultar los rangos de sus miembros.

3.1.1 ¿Qué es la Secuencia de Collatz?

Dado un número entero positivo `n`:

- Si `n` es par: `n = n / 2`
- Si `n` es impar: `n = 3n + 1`
- Se repite hasta llegar a 1

Ejemplo: n = 13

```
13 → 40 → 20 → 10 → 5 → 16 → 8 → 4 → 2 → 1
Pasos totales: 9
```

3.1.2 El Sistema de Rangos Criminales

Cada miembro tiene un ID numérico (entre 10 y 1000). La organización usa la siguiente codificación:

1. **Calcular la longitud de Collatz**: Número de pasos para llegar a 1
2. **Clasificar por rango según la longitud**:
   - **LÍDER**: Longitud $\geq$ 100 pasos
   - **OPERADOR**: Longitud entre 50-99 pasos
   - **INFORMANTE**: Longitud entre 20-49 pasos
   - **PEÓN**: Longitud $<$ 20 pasos
3. **Calcular el "valor máximo alcanzado"** durante toda la secuencia (este es el código de acceso)

Tu Tarea

Implementar un programa que:

1. Reciba una lista de IDs de miembros capturados
2. Calcule recursivamente la secuencia de Collatz para cada ID
3. Determine el rango de cada miembro

3.1.3 Entradas:

- Un entero `n` (cantidad de IDs, $5 \leq n \leq 20$)
- `n` enteros positivos representando los IDs ($10 \leq ID \leq 1000$)

3.1.4 Salida:

Para cada ID mostrar:

```
ID: [número]
Pasos: [longitud]
Rango: [LÍDER/OPERADOR/INFORMANTE/PEÓN]
```

3.1.5 Ejemplo:

Entrada:

```
n = 4
IDs: 27, 97, 871, 77
```

Proceso:

ID 27:

```
27 → 82 → 41 → 124 → 62 → 31 → 94 → 47 → 142 → 71 → 214 → 107 →
322 → 161 → 484 → 242 → 121 → 364 → 182 → 91 → 274 → 137 → 412 →
206 → 103 → 310 → 155 → 466 → 233 → 700 → 350 → 175 → 526 → 263
→ 790 → 395 → 1186 → 593 → 1780 → 890 → 445 → 1336 → 668 → 334 →
167 → 502 → 251 → 754 → 377 → 1132 → 566 → 283 → 850 → 425 →
1276 → 638 → 319 → 958 → 479 → 1438 → 719 → 2158 → 1079 → 3238 →
1619 → 4858 → 2429 → 7288 → 3644 → 1822 → 911 → 2734 → 1367 →
4102 → 2051 → 6154 → 3077 → 9232 → 4616 → 2308 → 1154 → 577 →
1732 → 866 → 433 → 1300 → 650 → 325 → 976 → 488 → 244 → 122 → 61
→ 184 → 92 → 46 → 23 → 70 → 35 → 106 → 53 → 160 → 80 → 40 → 20 →
10 → 5 → 16 → 8 → 4 → 2 → 1
Pasos: 111
Rango: Lider
```

ID 97:

```
97 → 292 → 146 → 73 → 220 → 110 → 55 → 166 → 83 → 250 → 125 →
376 → 188 → 94 → 47 → 142 → ... → 1
Pasos: 118
Rango: Lider
```

ID 871:

```
871 → ... → 1
Pasos: 178
Rango: Lider
```

ID 77:

```
77 → ... → 1
Pasos: 22
Rango: Informante
```

3.2 Desafío 2: Descifrador de Claves de Autenticación por Matriz de Bits

El servidor central de la organización usa un sistema de autenticación matricial donde cada clave es una cadena de 16 caracteres que debe pasar múltiples filtros de validación.

Sistema de Validación Multi-Capa

Una clave es válida si pasa TODAS estas verificaciones:

3.2.1 CAPA 1: Análisis de Composición

La clave debe contener:

- **Exactamente 4 letras mayúsculas** (A-Z)
- **Exactamente 4 letras minúsculas** (a-z)
- **Exactamente 4 dígitos** (0-9)
- **Exactamente 4 caracteres especiales del conjunto:** {!, @, #, $, %, &, *, +}

3.2.2 CAPA 2: Validación de Paridad Posicional

Dividir la clave en 4 bloques de 4 caracteres cada uno:

```
Bloque 0: posiciones 0-3
Bloque 1: posiciones 4-7
Bloque 2: posiciones 8-11
Bloque 3: posiciones 12-15
```

Para cada bloque calcular su checksum de paridad:

- Sumar los valores ASCII de los 4 caracteres
- El checksum es válido si la suma es impar

La clave es válida si **AL MENOS** 3 de los 4 bloques tienen checksum impar

3.2.3 CAPA 3: Código Hash de Verificación

Si pasa todas las capas, calcular el hash simplificado:

```
hash = 0
para cada carácter en la clave:
    hash = (hash * 31 + ASCII(caracter)) MOD 65536
```

El hash debe ser mayor a 10000 para ser aceptado.

3.3 Entrada

- Una cadena de exactamente **16 caracteres** ASCII

3.4 Salida

Si pasa todas las capas:

```
CLAVE ACEPTADA
```

Si falla alguna capa:

```
CLAVE RECHAZADA
Capa fallida: [número de capa]
```

3.5 Ejemplo "CLAVE RECHAZADA":

Entrada:

```
Clave: "ABCD1234efgh!@#$"
```

Análisis (NO ES NECESARIO MOSTRAR ESTO)

CAPA 1:

- Mayúsculas: A,B,C,D $\rightarrow$ 4
- Minúsculas: e,f,g,h $\rightarrow$ 4
- Dígitos: 1,2,3,4 $\rightarrow$ 4
- Especiales: !,@,#,$ $\rightarrow$ 4

CAPA 2:

```
Bloque 0 "ABCD": 65+66+67+68 = 266 (par)
Bloque 1 "1234": 49+50+51+52 = 202 (par)
Bloque 2 "efgh": 101+102+103+104 = 410 (par)
Bloque 3 "!@#$": 33+64+35+36 = 168 (par)
Bloques válidos: 0/4 → FALLA (necesita 3)
```

Salida:

```
CLAVE RECHAZADA
Capa fallida: 2 (PARIDAD POSICIONAL)
```

3.6 Ejemplo "CLAVE VALIDADA":

Entrada:

```
Clave: "aB3!cD5#eF7@gH9%"
```

Análisis (NO ES NECESARIO MOSTRAR ESTO)

(Asume que esta clave pasa todas las capas)

Salida:

```
CLAVE VALIDADA
```

4 README

Debe contener como mínimo:

- Nombre, RoI y Paralelo de los integrantes.
- Especificación de los algoritmos y desarrollo realizado.
- Supuestos utilizados
- **Para el Desafío 1**: Explicación detallada de cómo se implementó la recursión
- **Para el Desafío 2**: Explicación de las estructuras de datos utilizada (mas que nada características de lo que hicieron)
- Instrucciones de compilación y ejecución en RARS

5 Consideraciones

- Se deberá trabajar de a pares. Se deberá entregar en Aula a mas tardar el día 21 de noviembre de 2025 a las 23:59 horas. Se descontarán 5 puntos por cada hora o fracción de atraso. Las copias serán evaluadas con nota 0 en el promedio de las tareas.
- La tarea debe realizarse en RARS. Se recomienda que se familiarice rápidamente con la plataforma, y ante cualquier duda consulte con sus compañeros o directamente con los ayudantes lo antes posible. El único responsable si no acude a alguien para resolver sus dudas a tiempo es usted.
- La entrega considera dos archivos, `subsistema1.asm` y `subsistema2.asm`, junto con el README. Los archivos deberán ser comprimidos y enviados juntos en un archivo .zip de nombre LAB4_ROL1_ROL2.
- Si no se entrega README, o si su programa no funciona, la nota es 0 hasta la recorrección.
- Una vez entregadas las notas de la tarea existirá un plazo de 3 días para apelar. Transcurrido este plazo las notas no podrán ser modificadas.