**ANÁLISIS DE ERRORES Y PROBLEMAS (Simulador de Arquitectura de Von Neumann)**

## ERRORES CRÍTICOS

### 1. Error en STORE con direcciones no válidas

**Ubicación:** executeInstruction() - caso STORE

**Código Actual:**

javascript

```
case 'STORE':
  const addr = parseInt(value);
  memory[addr] = acc.toString();
  highlight('mem-' + addr);
  displayMemory();
```

**Problema:**

- No valida si la dirección está dentro del rango (0-15)
- Si el programa tiene STORE 20, intentará acceder a memory[20] que no existe

**Ejemplo de Error:**

javascript

```
memory = [/* 16 posiciones */];
// STORE 20 → memory[20] = acc
// ❌ Acceso fuera de límites
```

**Solución:**

javascript

```
case 'STORE':
  const addr = parseInt(value);
  if (addr < 0 || addr >= memory.length) {
    setStatus(` ❌ ERROR: Dirección ${addr} fuera de rango (0-15)`);
```

```
    pc = 999; // Detener programa
    break;
  }
  memory[addr] = acc.toString();
  highlight('mem-' + addr);
  displayMemory();
  setStatus(` ⚡ EXECUTE: Guardar ACC (${acc}) en memoria[${addr}]`);
  break;
```

**2. Error en LOAD/ADD con direcciones de memoria**

**Ubicación:** executeInstruction() - casos LOAD y ADD

**Código Actual:**

javascript

```
case 'LOAD':
  if (value.match(/^\d+$/)) {
    acc = parseInt(value);
  } else {
    const addr = parseInt(value);
    acc = parseInt(memory[addr]);
    highlight('mem-' + addr);
  }
  break;
```

**Problema:**

- No distingue entre valores inmediatos grandes y direcciones de memoria
- LOAD 15 podría interpretarse como valor 15 o dirección [15]
- ADD 8 es ambiguo (¿suma 8 o suma memory[8]?)

**Ejemplo de Ambigüedad:**

javascript

```
LOAD 10   // ¿Cargar el número 10 o cargar desde memoria[10]?
ADD 5     // ¿Sumar 5 o sumar memory[5]?
```

**Solución:** Usar sintaxis explícita

javascript

```
case 'LOAD':
  if (value.startsWith('[') && value.endsWith(']')) {
    // LOAD [8] → Desde memoria
    const addr = parseInt(value.slice(1, -1));
    if (addr < 0 || addr >= memory.length) {
      setStatus(` ❌ ERROR: Dirección ${addr} inválida `);
      pc = 999;
      break;
    }
    acc = parseInt(memory[addr]);
    highlight('mem-' + addr);
    setStatus(` ⚡ EXECUTE: Cargar desde memoria[${addr}] = ${memory[addr]} → ACC `);
  } else {
    // LOAD 10 → Valor inmediato
    acc = parseInt(value);
    setStatus(` ⚡ EXECUTE: Cargar ${value} → ACC = ${acc} `);
  }
  break;
```

**Nuevo formato de instrucciones:**

LOAD 10 → Cargar valor 10

LOAD [8] → Cargar desde memoria posición 8

ADD 5 → Sumar 5

ADD [10] → Sumar el contenido de memoria[10]

**3. Manejo de INPUT sin validación**

**Ubicación:** waitForInput() y caso INPUT

**Código Actual:**

javascript

```
case 'INPUT':
  setStatus('⌨️ Esperando que ingreses un número...');
  const input = await waitForInput();
  acc = parseInt(input);
  document.getElementById('output').innerHTML += `<br>📩 Ingresaste: ${acc}`;
  break;
```

**Problema:**

- parseInt() puede devolver NaN si el usuario ingresa texto
- No valida números negativos muy grandes
- No maneja el caso cuando el usuario cancela

**Ejemplo de Error:**

javascript

```
Usuario ingresa: "abc"
parseInt("abc") = NaN
acc = NaN
// Todas las operaciones posteriores fallan
```

**Solución:**

javascript

```
case 'INPUT':
  setStatus('⌨️ Esperando que ingreses un número...');
  const input = await waitForInput();
  const numValue = parseInt(input);

  if (isNaN(numValue)) {
    setStatus('❌ ERROR: Debes ingresar un número válido');
    document.getElementById('output').innerHTML +=
      `<br>❌ Error: "${input}" no es un número válido`;
    pc--; // Repetir la instrucción INPUT
    break;
  }

  acc = numValue;
  document.getElementById('output').innerHTML += `<br>📩 Ingresaste: ${acc}`;
  setStatus(`⚡ EXECUTE: Número ingresado: ${acc}`);
  break;
```

**ERRORES LÓGICOS**

**4. Desbordamiento del Program Counter**

**Ubicación:** executeStep()

**Código Actual:**

javascript

```
async function executeStep() {
  if (pc >= memory.length || memory[pc] === 'HALT' || memory[pc] === '0') {
```

```
    setStatus('🏁 Programa terminado');
    setStep(0);
    return;
  }
  // ...
}
```

**Problema:**

- Si un programa no tiene HALT y termina en instrucciones válidas, continúa ejecutando '0' como instrucciones
- El PC puede incrementarse más allá de 15

**Solución:**

javascript

```
async function executeStep() {
  // Verificar límites primero
  if (pc >= memory.length) {
    setStatus('❌ ERROR: Program Counter fuera de límites');
    return;
  }

  // Verificar si hay instrucción válida
  if (memory[pc] === 'HALT') {
    setStatus('🛑 HALT: Programa terminado');
    setStep(0);
    return;
  }
```

```
  if (memory[pc] === '0' || memory[pc] === '') {
    setStatus('⚠️ Advertencia: No hay más instrucciones');
    setStep(0);
    return;
  }
  // ...
}
```

**5. Race Condition en executeAll()**

**Ubicación:** executeAll()

**Código Actual:**

javascript

```
async function executeAll() {
  while (pc < memory.length &&
         memory[pc] !== 'HALT' &&
         memory[pc] !== '0') {
    await executeStep();
    await sleep(200);
  }
}
```

**Problema:**

- Si el usuario hace clic múltiples veces en "Ejecutar Todo", se crean múltiples loops concurrentes
- No hay forma de detener la ejecución una vez iniciada

**Solución:**

javascript

```
let isRunning = false; // Variable global

async function executeAll() {
  if (isRunning) {
    setStatus('⚠️ Ya hay una ejecución en progreso');
    return;
  }

  isRunning = true;
  document.getElementById('run-btn').disabled = true;

  while (isRunning &&
      pc < memory.length &&
      memory[pc] !== 'HALT' &&
      memory[pc] !== '0') {
    await executeStep();
    await sleep(200);
  }

  isRunning = false;
  document.getElementById('run-btn').disabled = false;
}

function stopExecution() {
  isRunning = false;
  setStatus('⏹ Ejecución detenida por el usuario');
```

}

**HTML adicional:**

html

<button onclick="stopExecution()" id="stop-btn">⏹ Detener</button>

**6. Pérdida de datos en memory al hacer displayMemory()**

**Ubicación:** displayMemory()

**Código Actual:**

javascript

```
function displayMemory() {
  const memDiv = document.getElementById('memory');
  memDiv.innerHTML = '';
  for (let i = 0; i < memory.length; i++) {
    // ...
  }
}
```

**Problema:**

- Si memory.length es 0 o undefined, no muestra nada
- No verifica que memory sea un array válido

**Solución:**

javascript

```
function displayMemory() {
  const memDiv = document.getElementById('memory');
  memDiv.innerHTML = '';

  if (!Array.isArray(memory) || memory.length === 0) {
```

```
    memDiv.innerHTML = '<p style="color: red;">Error: Memoria no inicializada</p>';
    return;
  }

  for (let i = 0; i < memory.length; i++) {
    const cell = document.createElement('div');
    cell.className = 'memory-cell';
    cell.id = 'mem-' + i;
    cell.innerHTML = `
      <span class="memory-addr">[${i}]</span>
      <span class="memory-data">${memory[i] || '0'}</span>
    `;
    memDiv.appendChild(cell);
  }
}
```

🔧 **PROBLEMAS DE USABILIDAD**

**7. No se puede editar el programa**

**Problema:**

- Los usuarios solo pueden ejecutar programas predefinidos
- No hay forma de modificar o crear programas personalizados

**Solución:** Agregar un editor simple

```
html
<div class="program-editor">
  <h3>🖍️ Editor de Programa (Opcional)</h3>
  <textarea id="custom-program" rows="8" placeholder="Escribe tu programa aquí...
```

Ejemplo:

LOAD 10

ADD 5

OUT

HALT"></textarea>

<button onclick="loadCustomProgram()">📝 Cargar Programa Personalizado</button>

javascript

```
function loadCustomProgram() {
  const code = document.getElementById('custom-program').value.trim();
  if (!code) {
    setStatus('❌ El programa está vacío');
    return;
  }

  reset();
  const lines = code.split('\n').filter(line => line.trim() !== '');

  if (lines.length > 16) {
    setStatus('❌ El programa es demasiado largo (máximo 16 líneas)');
    return;
  }

  memory = [...lines];
  while (memory.length < 16) {
```

```
    memory.push('0');
  }

  displayMemory();
  setStatus('✅ Programa personalizado cargado');
}
```

**8. Falta feedback visual en errores**

**Problema:**

- Los errores solo aparecen en el status bar
- No hay indicadores visuales claros de errores

**Solución:**

javascript

```
function setStatus(msg, type = 'info') {
  const statusDiv = document.getElementById('status');
  statusDiv.textContent = msg;

  // Cambiar color según tipo
  statusDiv.style.background = type === 'error' ? '#ffebee' :
                               type === 'warning' ? '#fff3e0' :
                               type === 'success' ? '#e8f5e9' :
                               'white';
  statusDiv.style.color = type === 'error' ? '#c62828' :
                          type === 'warning' ? '#e65100' :
                          type === 'success' ? '#2e7d32' :
                          '#333';
```

```
}

// Uso:
setStatus('❌ ERROR: Dirección inválida', 'error');
setStatus('⚠️ Advertencia: No hay más instrucciones', 'warning');
setStatus('✅ Programa cargado correctamente', 'success');
```

**PROBLEMAS DE RENDIMIENTO**

**9. Re-renderizado innecesario de memoria**

**Problema:**

- displayMemory() reconstruye TODO el DOM cada vez
- Es ineficiente si solo cambió una celda

**Solución Optimizada:**

```
javascript
function updateMemoryCell(index, value) {
  const cell = document.getElementById('mem-' + index);
  if (cell) {
    cell.querySelector('.memory-data').textContent = value;
  }
}

// Usar en STORE:
case 'STORE':
  const addr = parseInt(value);
  memory[addr] = acc.toString();
  updateMemoryCell(addr, acc.toString()); // Solo actualiza una celda
```

```
  highlight('mem-' + addr);
  break;
```

**10. Acumulación de event listeners en INPUT**

**Problema:**

- Cada vez que se ejecuta INPUT, se agrega un nuevo event listener
- Si el usuario cancela, los listeners quedan huérfanos

**Solución:**

javascript

```
let inputHandler = null; // Variable global

function waitForInput() {
  return new Promise((resolve, reject) => {
    const input = document.getElementById('input-field');

    // Limpiar listener anterior si existe
    if (inputHandler) {
      input.removeEventListener('keypress', inputHandler);
    }

    inputHandler = (e) => {
      if (e.key === 'Enter') {
        const value = input.value.trim();
        if (value !== '') {
          input.value = '';
          input.removeEventListener('keypress', inputHandler);
```

```
            inputHandler = null;
            resolve(value);
          } else {
            setStatus(' ⚠️ Debes ingresar un valor', 'warning');
          }
        }
      };

      input.addEventListener('keypress', inputHandler);
      input.focus();

      // Timeout de 60 segundos
      setTimeout(() => {
        if (inputHandler) {
          input.removeEventListener('keypress', inputHandler);
          inputHandler = null;
          reject('Timeout: No se ingresó ningún valor');
        }
      }, 60000);
    });
}
```

**RESUMEN DE ERRORES**

| # | Tipo | Severidad | Descripción |
|---|---|---|---|
| 1 | Crítico | 🔴 Alta | STORE sin validación de límites |
| 2 | Lógico | 🟡 Media | Ambigüedad LOAD/ADD inmediato vs memoria |
| 3 | Crítico | 🔴 Alta | INPUT no valida NaN |
| 4 | Lógico | 🟡 Media | PC puede desbordarse |
| 5 | Concurrencia | 🟠 Media-Alta | Race condition en executeAll() |
| 6 | Validación | 🟢 Baja | displayMemory sin validar array |
| 7 | Usabilidad | 🟡 Media | No se pueden editar programas |
| 8 | UI/UX | 🟢 Baja | Falta feedback visual de errores |
| 9 | Rendimiento | 🟢 Baja | Re-renderizado innecesario |
| 10 | Memoria | 🟡 Media | Event listeners huérfanos |

**CÓDIGO CORREGIDO - PARCHE COMPLETO**

javascript

```
// Variables globales adicionales
let isRunning = false;
let inputHandler = null;


// executeStep() mejorado
async function executeStep() {
  if (pc >= memory.length) {
    setStatus('❌ ERROR: Program Counter fuera de límites', 'error');
```

```
    return;
}

if (memory[pc] === 'HALT') {
    setStatus('🛑 HALT: Programa terminado', 'success');
    setStep(0);
    return;
}

if (!memory[pc] || memory[pc] === '0') {
    setStatus('⚠️ No hay más instrucciones', 'warning');
    setStep(0);
    return;
}

setStep(1);
ir = memory[pc];
highlight('mem-' + pc);
highlight('pc-box');
highlight('ir-box');
setStatus(`🔍 FETCH: Leyendo "${ir}" desde posición ${pc}`);
updateDisplay();
await sleep(800);

pc++;
updateDisplay();
```

```
  setStep(2);
  setStatus(` 🔧 DECODE: Interpretando "${ir}"`);
  await sleep(800);

  setStep(3);
  await executeInstruction(ir);
  await sleep(800);

  setStep(0);
}

// executeInstruction() mejorado (casos principales)
async function executeInstruction(instruction) {
  const parts = instruction.split(' ');
  const op = parts[0];
  const value = parts[1];

  highlight('acc-box');

  switch(op) {
    case 'LOAD':
      if (value.startsWith('[') && value.endsWith(']')) {
        const addr = parseInt(value.slice(1, -1));
        if (addr < 0 || addr >= memory.length) {
          setStatus(` ❌ ERROR: Dirección ${addr} inválida`, 'error');
```

```
            pc = 999;
            break;
        }
        acc = parseInt(memory[addr]) || 0;
        highlight('mem-' + addr);
        setStatus(` ⚡ EXECUTE: Cargar desde memoria[${addr}] → ACC = ${acc}`);
    } else {
        acc = parseInt(value);
        setStatus(` ⚡ EXECUTE: Cargar ${value} → ACC = ${acc}`);
    }
    break;

case 'ADD':
    if (value.startsWith('[') && value.endsWith(']')) {
        const addr = parseInt(value.slice(1, -1));
        if (addr < 0 || addr >= memory.length) {
            setStatus(` ❌ ERROR: Dirección ${addr} inválida`, 'error');
            pc = 999;
            break;
        }
        const oldAcc = acc;
        acc += parseInt(memory[addr]) || 0;
        highlight('mem-' + addr);
        setStatus(` ⚡ EXECUTE: ${oldAcc} + memoria[${addr}] = ${acc}`);
    } else {
        const oldAcc = acc;
```

```
      acc += parseInt(value);
      setStatus(` ⚡ EXECUTE: ${oldAcc} + ${value} = ${acc}`);
    }
    break;

  case 'STORE':
    const addr = parseInt(value);
    if (addr < 0 || addr >= memory.length) {
      setStatus(` ❌ ERROR: Dirección ${addr} fuera de rango`, 'error');
      pc = 999;
      break;
    }
    memory[addr] = acc.toString();
    updateMemoryCell(addr, acc.toString());
    highlight('mem-' + addr);
    setStatus(` ⚡ EXECUTE: Guardar ACC (${acc}) en memoria[${addr}]`);
    break;

  case 'INPUT':
    try {
      setStatus('⌨️ Esperando que ingreses un número...');
      const input = await waitForInput();
      const numValue = parseInt(input);

      if (isNaN(numValue)) {
        setStatus(` ❌ ERROR: "${input}" no es un número válido`, 'error');
```

```
            document.getElementById('output').innerHTML +=
                `<br>❌ Error: "${input}" no es un número`;
            pc--; // Repetir INPUT
            break;
        }

        acc = numValue;
        document.getElementById('output').innerHTML +=
            `<br>📥 Ingresaste: ${acc}`;
        setStatus(`⚡ EXECUTE: Número ingresado: ${acc}`);
    } catch (error) {
        setStatus(`❌ ERROR: ${error}`, 'error');
        pc = 999;
    }
    break;

case 'OUT':
    document.getElementById('output').innerHTML +=
        `<br>📤 <strong>RESULTADO: ${acc}</strong>`;
    setStatus(`⚡ EXECUTE: Mostrar resultado: ${acc}`);
    break;

case 'HALT':
    setStatus('🛑 HALT: Programa terminado', 'success');
    pc = 999;
```

```
            break;

        default:
            setStatus(` ❌ ERROR: Instrucción desconocida "${op}"` , 'error');
            pc = 999;
            break;
    }

    updateDisplay();
}

// Función auxiliar para actualizar una celda
function updateMemoryCell(index, value) {
    const cell = document.getElementById('mem-' + index);
    if (cell) {
        cell.querySelector('.memory-data').textContent = value;
    }
}

// setStatus mejorado
function setStatus(msg, type = 'info') {
    const statusDiv = document.getElementById('status');
    statusDiv.textContent = msg;

    statusDiv.style.background = type === 'error' ? '#ffebee' :
                                 type === 'warning' ? '#fff3e0' :
```

```
                type === 'success' ? '#e8f5e9' :
                'white';
    statusDiv.style.color = type === 'error' ? '#c62828' :
                type === 'warning' ? '#e65100' :
                type === 'success' ? '#2e7d32' :
                '#333';
}
```

**CONCLUSIÓN**

**Total de errores encontrados:** 10

**Distribución:**

- Críticos: 2
- Altos: 1
- Medios: 4
- Bajos: 3