

Organización del Procesador

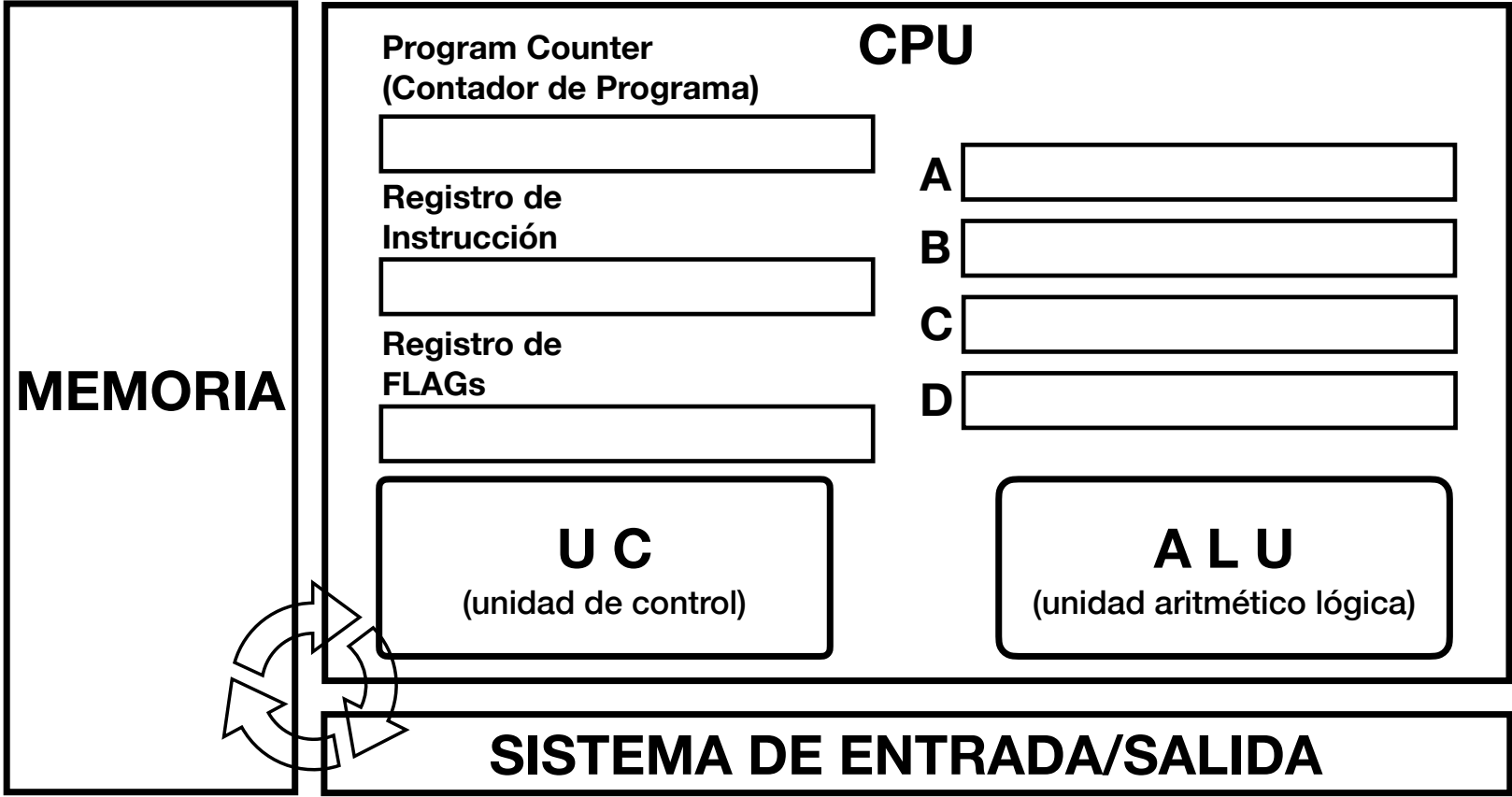
Microprogramas

Departamento de Computación - UNRC

El camino a recorrer

- Un poco de Historia y Sistemas Numéricos
- Introducción a la Electrónica
- Representación de Información
- Cómo computar utilizando la electricidad
- Evolución y funcionamiento abstracto de una computadora
- Assembly X86
- **Micro-programación (cómo fabricar un procesador)**
- Eficiencia
 - Pipelines
 - Memoria Caché
 - Memoria Virtual

Arquitectura von Neumann

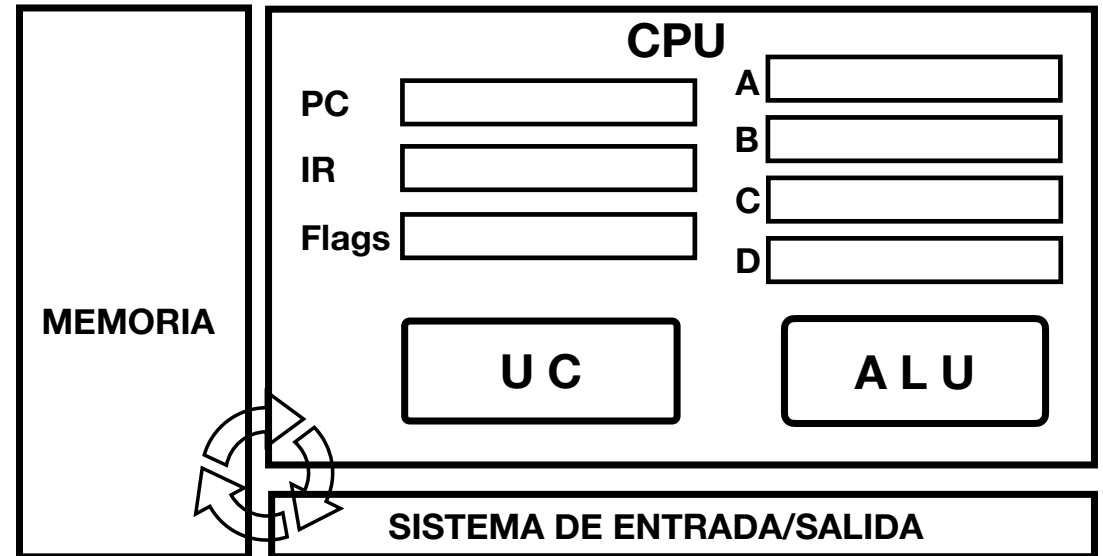


John von Neumann: (1903-1957) Físico Matemático Húngaro, Estadounidense. Participó activamente en el proyecto ENIAC y en el desarrollo de la EDVAC, donde propuso el concepto de almacenamiento de programas y datos en la memoria de la computadora, sentando las bases de la arquitectura de von Neumann, que se convirtió en el modelo predominante en las computadoras modernas.

Arquitectura von Neumann

Fetch: La Unidad de control obtiene de la memoria la próxima instrucción que indica el *contador de programa* (**PC**) y la almacena en el Registro de Instrucción (**IR**). Finalmente actualiza el **PC** indicando la dirección de la próxima instrucción.

Decode: La Unidad de control decodifica la instrucción y obtiene de la memoria (si fuere necesario) la información que involucra dicha instrucción.



Execute: La ALU ejecuta (calcula) el resultado de la operación y lo almacena en un registro o memoria.

Ejemplo de tareas asociadas a la ejecución de una instrucción

```
ADD    eax, [L1]
```

- 1) Recuperar la instrucción de la memoria (al IR)
- 2) Decodificar cuál es la instrucción
- 3) Calcular los operandos (memoria efectiva)
- 4) Recuperar los operandos (en registros)
- 5) Ejecutar la instrucción
- 6) Guardar el resultado

Ejemplo de tareas asociadas a la ejecución de una instrucción

```
ADD    eax, [L1+4]
```

1) Recuperar la instrucción de la memoria (al IR)

- Configurar la dirección (MAR) con el PC
- Activar la Lectura de Memoria al IR
- Incrementar el PC

2) Decodificar cuál es la instrucción

- Decodificar ADD

3) Calcular los operandos (memoria efectiva)

- Calcular la dirección $L1+4$

4) Recuperar los operandos (en registros)

- Guardar en el RegistroA de la ALU el contenido de EAX
- Configurar la dirección (MAR) con $L1+4$ (calculado en 3)
- Activar la Lectura de Memoria al RegistroB de la ALU Ejecutar la instrucción

5) Ejecutar la instrucción

- Activar la ALU con la Operación correspondiente

6) Guardar el resultado

- Transferir al EAX el valor alojado en el registro de salida de la ALU

Control por Hardware vs. Microprogramada

CONTROL POR HARDWARE

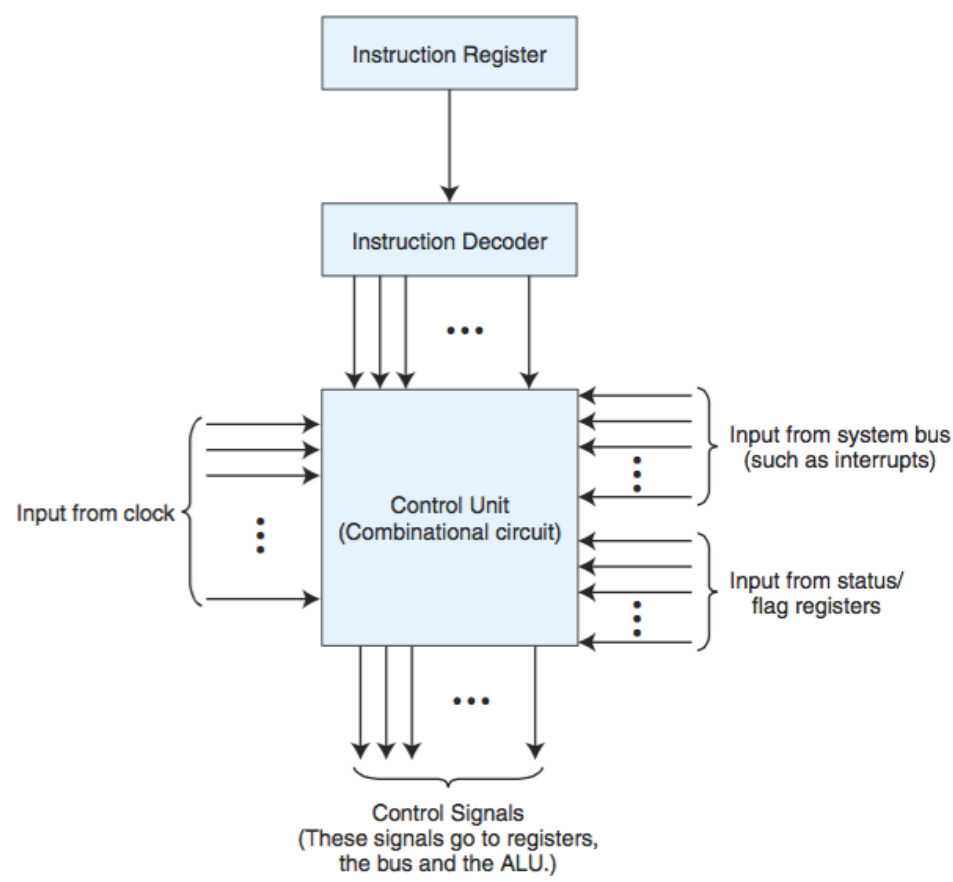
- **Diseño más rápido:** En el control por hardware, las señales de control son generadas directamente por circuitos lógicos específicos diseñados para cada instrucción y operación.
- **Menor flexibilidad:** Cambiar o agregar nuevas instrucciones o operaciones requerirá modificar directamente el hardware, lo que puede ser costoso y complejo.
- **Eficiencia en tiempo de ejecución:** el control por hardware puede ser más eficiente en términos de tiempo de ejecución para instrucciones comunes.
- **Mayor complejidad de diseño:** La implementación de un controlador por hardware para cada instrucción puede aumentar la complejidad del diseño del procesador.

CONTROL MICROPROGRAMADO

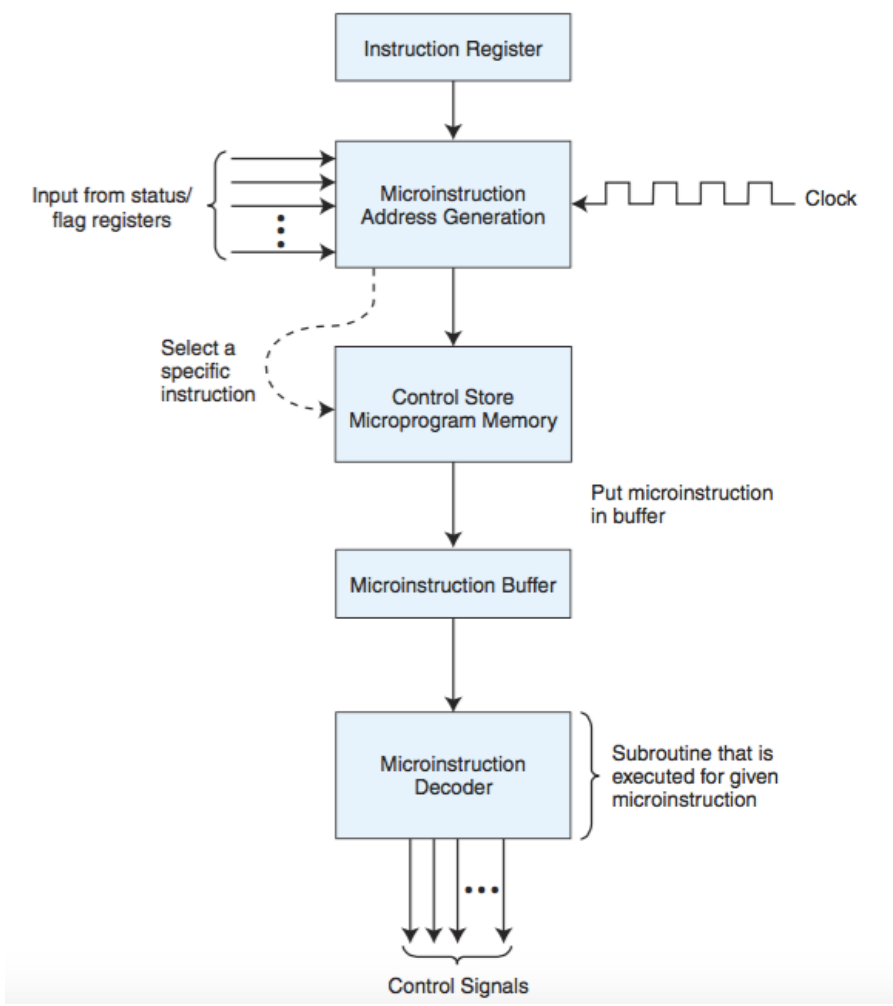
- **Diseño más flexible:** En el control microprogramado, las instrucciones se implementan utilizando una secuencia de microinstrucciones almacenadas en una memoria especial llamada microprograma. Esto facilita la modificación y la adición de nuevas instrucciones o operaciones sin cambiar el hardware físico.
- **Mayor tiempo de ejecución:** el control microprogramado puede ser más lento en comparación con el control por hardware.
- **Facilidad de actualización:** Las modificaciones y mejoras en las instrucciones se pueden realizar actualizando el microprograma sin necesidad de alterar el hardware subyacente.
- **Menos complejidad de diseño:** El control microprogramado puede simplificar el diseño del hardware al reducir la cantidad de circuitos de control necesarios.

Control por Hardware vs. Microprogramada

CONTROL POR HARDWARE

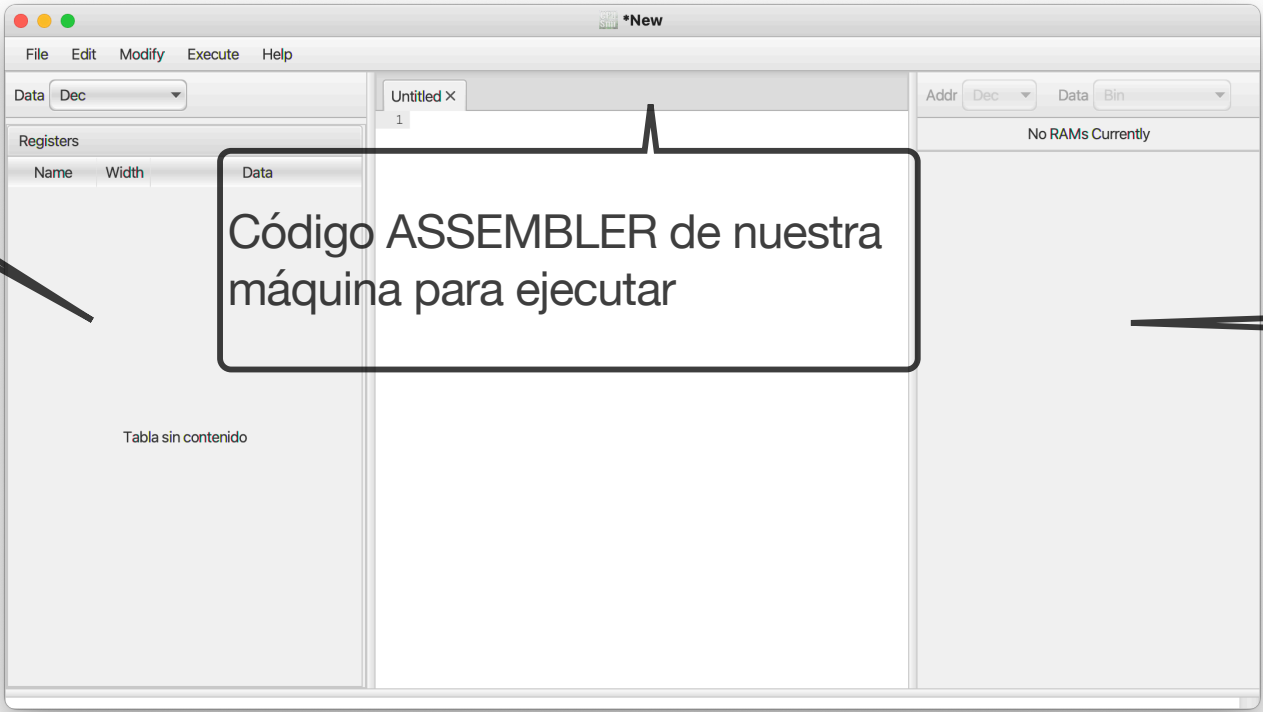


CONTROL MICROPROGRAMADO



Implementando nuestro propio procesador con CPU-SIM

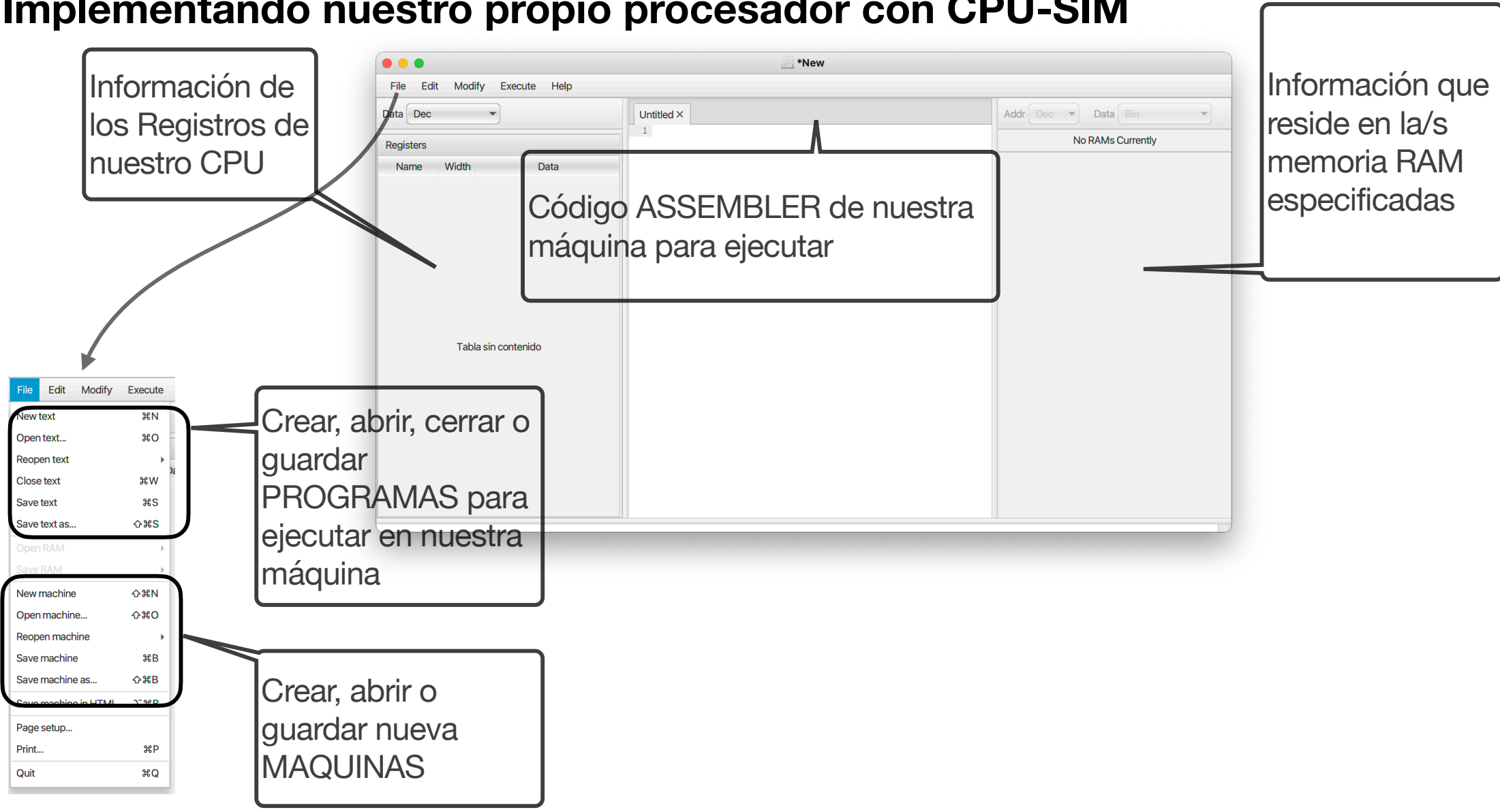
Información de los Registros de nuestro CPU



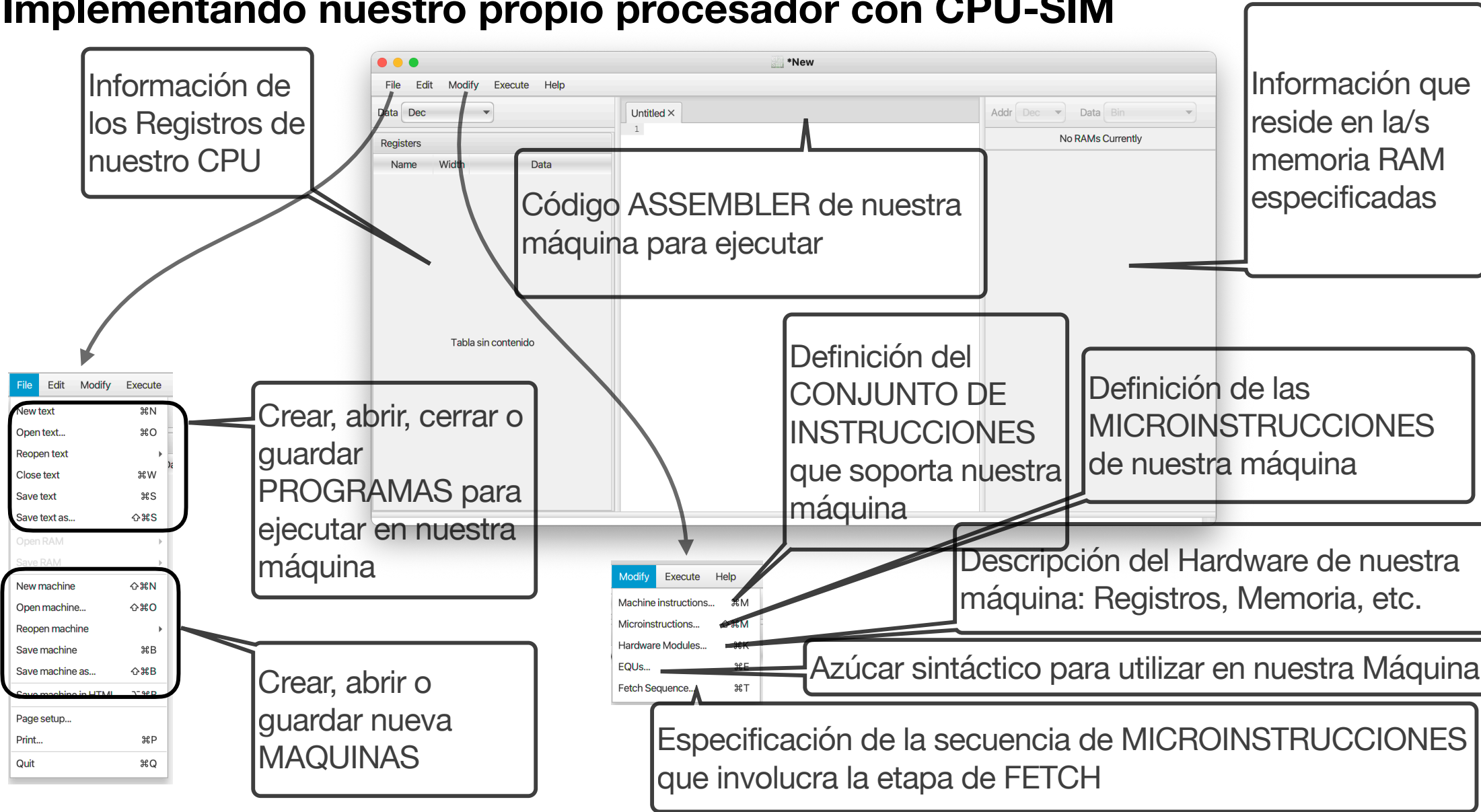
Código ASSEMBLER de nuestra máquina para ejecutar

Información que reside en la/s memoria RAM especificadas

Implementando nuestro propio procesador con CPU-SIM



Implementando nuestro propio procesador con CPU-SIM



Implementando nuestro propio procesador con CPU-SIM - Ejemplo WOMBAT

- Arquitectura de 16 bits
- 1 Registro de Propósito General ACC
- Program Counter (PC), Registro de Instrucción (IR), Interfaz con la memoria (MAR y MDR) y un registro de 3 Flags (STATUS)
- 12 Instrucciones:
 - **load**: lee de la memoria el valor almacenado en la dirección pasada como parámetro y la guarda en ACC
 - **store**: similar a load pero almacena el valor en la memoria.
 - **read**: lee por consola un valor almacenándolo en ACC
 - **write**: escribe en la consola el valor almacenado en ACC
 - **add, subtract, multiply, divide**: suma/resta/multiplica/divide(entera) el valor almacenado en la dirección pasada como operando al ACC y guarda el resultado en el ACC
 - **jump**: salto incondicional a la instrucción que reside en la dirección que indica el operando
 - **jmpz**: salto a la instrucción que reside en la dirección que indica el operando cuando $ACC = 0$
 - **jmpn**: salto a la instrucción que reside en la dirección que indica el operando cuando $ACC < 0$
- 128 bytes de memoria RAM

Implementando nuestro propio procesador con CPU-SIM - Ejemplo WOMBAT

CPU
SIM

Edit Modules

Type of Module: Register

| name | width | initial value | read-only |
|--------|-------|---------------|--------------------------|
| pc | 12 | 0 | <input type="checkbox"/> |
| acc | 16 | 0 | <input type="checkbox"/> |
| ir | 16 | 0 | <input type="checkbox"/> |
| mar | 12 | 0 | <input type="checkbox"/> |
| mdr | 16 | 0 | <input type="checkbox"/> |
| status | 3 | 0 | <input type="checkbox"/> |

New

Delete

Duplicate

Properties...

?

OK

Cancel

CPU
SIM

Edit Modules

Type of Module: RAM

| name | length | cellSize |
|------|--------|----------|
| Main | 128 | 8 |

New

Delete

Duplicate

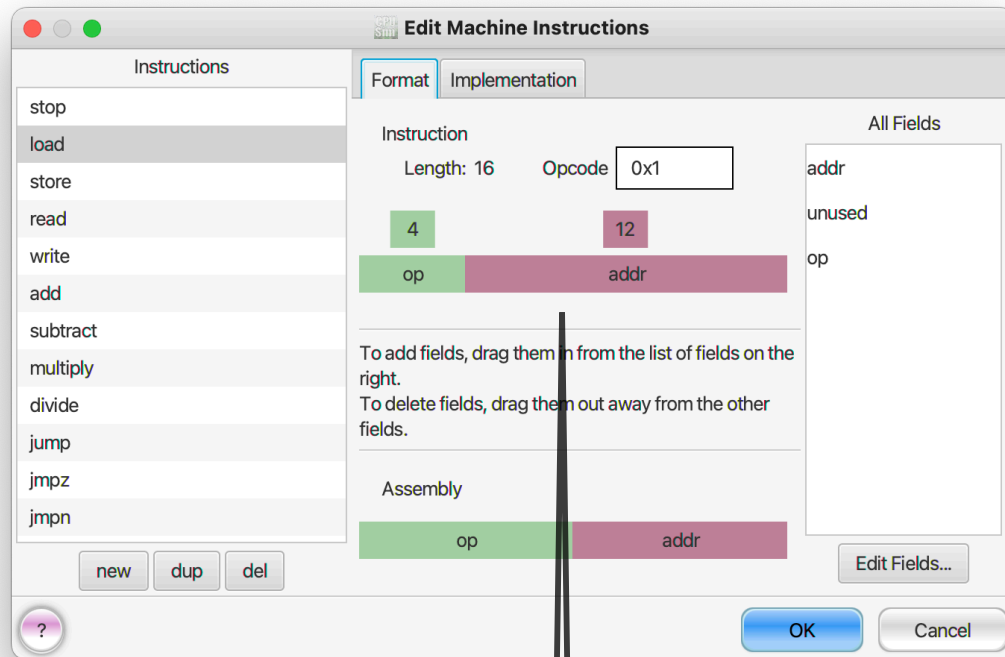
Properties...

?

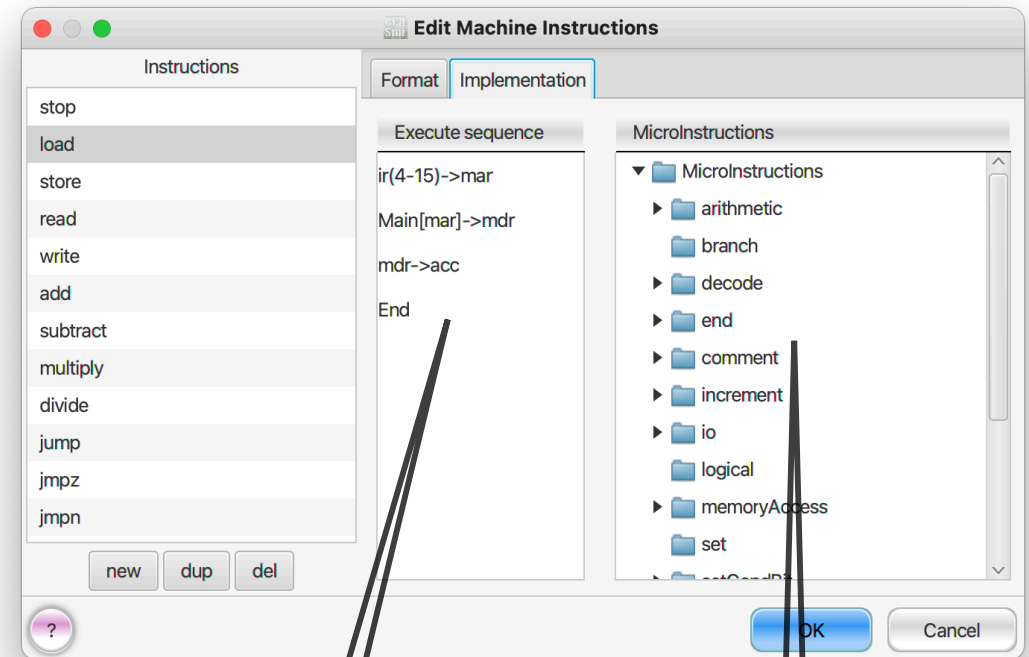
OK

Cancel

Implementando nuestro propio procesador con CPU-SIM - Ejemplo WOMBAT



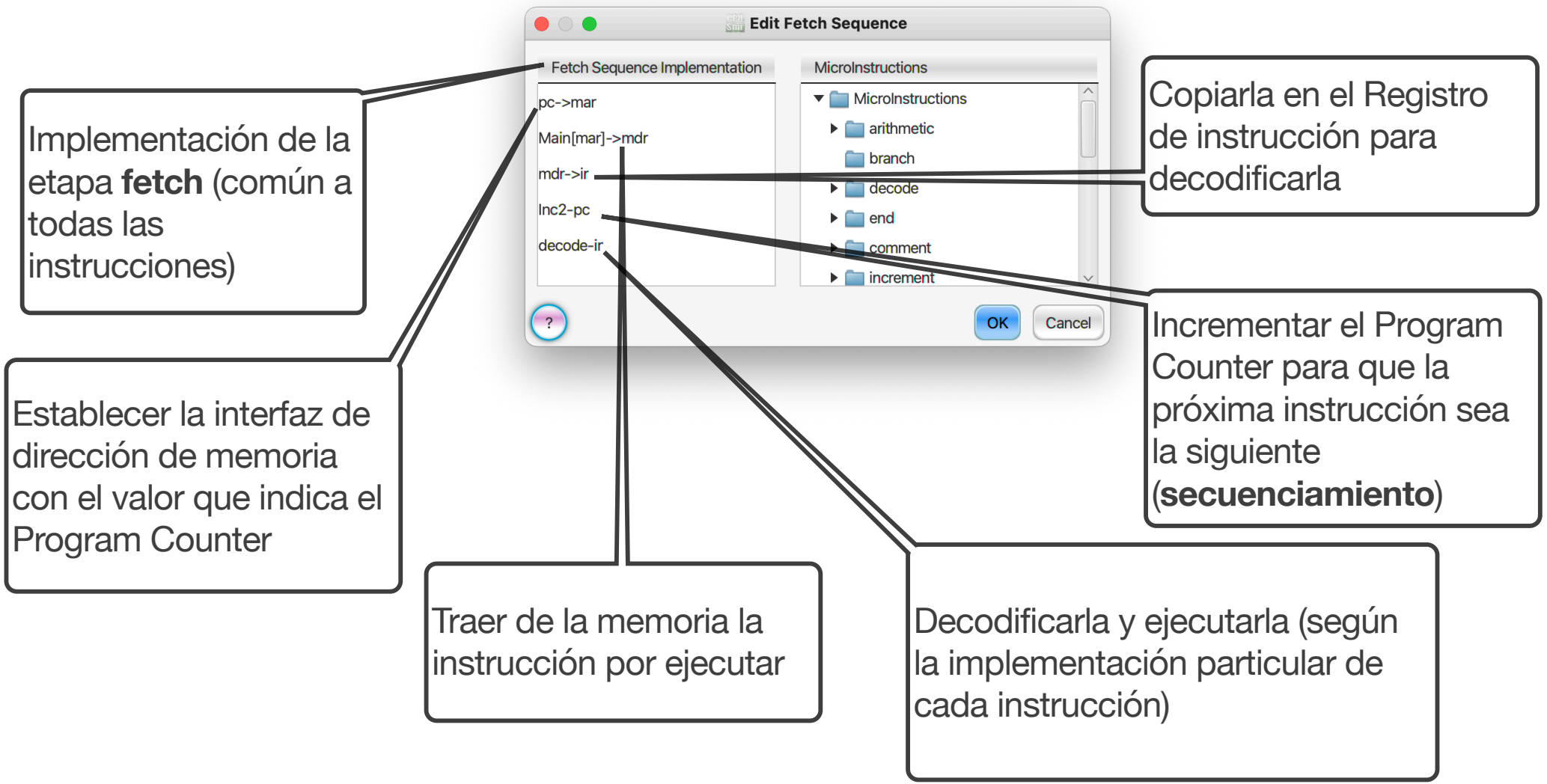
Configuración de las instrucciones



Implementación de las Instrucciones en términos de microInstrucciones. Recordar que el **Fetch** es común a toda instrucción por ejecutar (sólo implementamos el **Decode** y **Execute**).

Conjunto de microinstrucciones disponibles. Podemos especificar (construir hardware) nuevas si es necesario.

Implementando nuestro propio procesador con CPU-SIM - Ejemplo WOMBAT



Implementando nuestro propio procesador con CPU-SIM - Ejemplo WOMBAT

Tipos de Microinstrucciones,
por ejemplo entre Registros

CPU
Sim

Edit Microinstructions

Type of Microinstruction: TransferRtoR

| name | source | srcStartBit | dest | destStartBit | numBits |
|---------------|--------|-------------|------|--------------|---------|
| pc->mar | pc | 0 | mar | 0 | 12 |
| mar->pc | mar | 0 | pc | 0 | 12 |
| ir(4-15)->mar | ir | 4 | mar | 0 | 12 |
| mdr->ir | mdr | 0 | ir | 0 | 16 |
| mdr->acc | mdr | 0 | acc | 0 | 16 |
| acc->mdr | acc | 0 | mdr | 0 | 16 |
| ir(4-15)->pc | ir | 4 | pc | 0 | 12 |

New

Delete

Duplicate

Nombre de la microinstrucción,
que luego son utilizados para
implementar las instrucciones
que provee la máquina.

Especificación de que registros afecta (origen y destino)
desde y hasta qué bits copia. Simula una conexión física
(hardware) entre los mismos.

Implementando nuestro propio procesador con CPU-SIM - Ejemplo WOMBAT

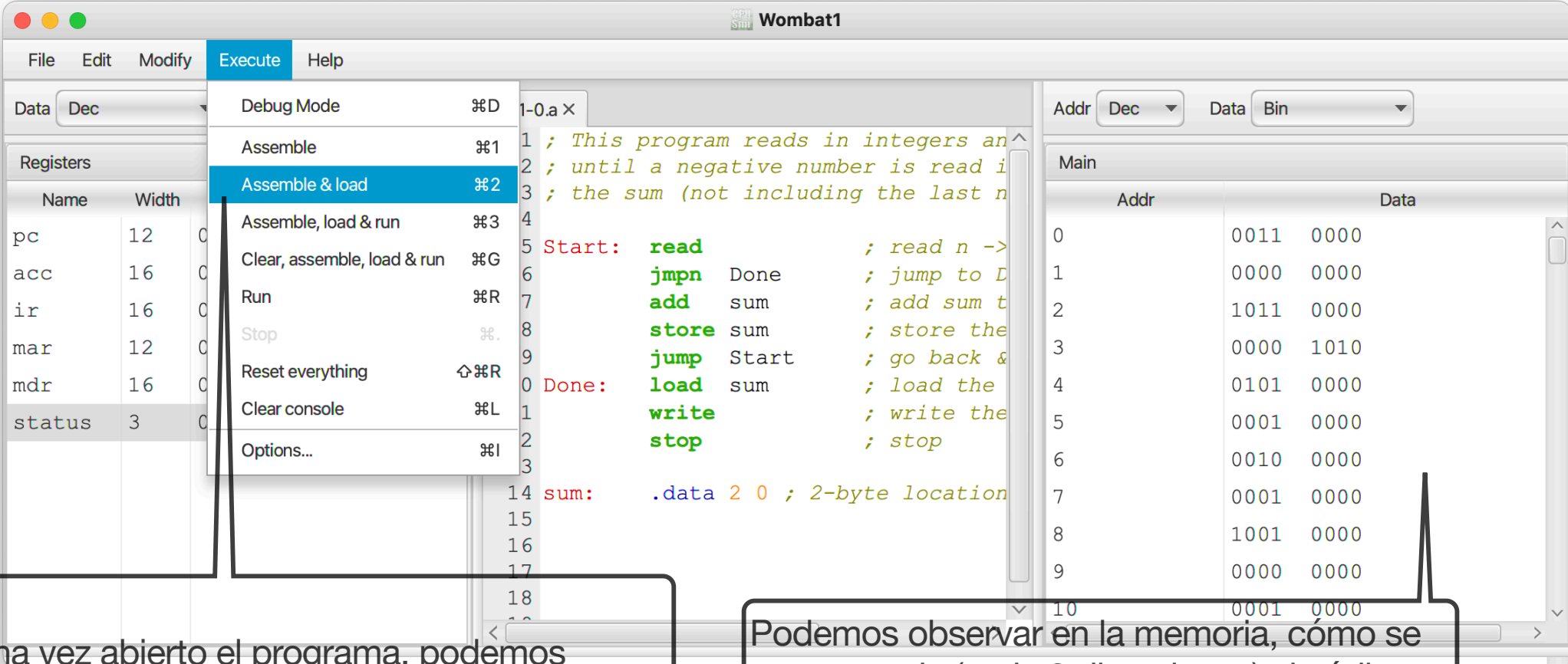
PROGRAMA ESCRITO EN ASSEMBLER PARA WOMBAT 1

```
; Este programa lee números enteros y los suma entre sí  
; hasta que se lee un número negativo.  
; Luego, muestra la suma (sin incluir el último número).
```

```
Start:  read           ; lee n -> acc  
        jmpn  Done      ; salta a Done si n < 0.  
        add   sum       ; suma sum junto con acc  
        store sum       ; guarda el resultado en sum  
        jump  Start     ; vuelve y lee el próximo número  
Done:   load  sum       ; lee en el acc el valor final de sum  
        write           ; lo escribe por consola  
        stop           ; termina
```

```
sum:    .data 2 0 ; 2-byte reserva de memoria para sum inicializado con 0
```

Implementando nuestro propio procesador con CPU-SIM - Ejemplo WOMBAT



Registers

| Name | Width | Value |
|--------|-------|-------|
| pc | 12 | 0 |
| acc | 16 | 0 |
| ir | 16 | 0 |
| mar | 12 | 0 |
| mdr | 16 | 0 |
| status | 3 | 0 |

```
1 ; This program reads in integers and calculates their sum
2 ; until a negative number is read in
3 ; the sum (not including the last negative number)
4
5 Start: read          ; read n -> ir
6       jmpn Done      ; jump to Done if negative
7       add  sum        ; add sum to acc
8       store sum      ; store the sum in memory
9       jump Start     ; go back & read more numbers
10
11 Done: load sum       ; load the sum from memory
12       write          ; write the sum to the console
13       stop          ; stop
14
15 sum: .data 2 0 ; 2-byte location for the sum
```

Main

| Addr | Data |
|------|-----------|
| 0 | 0011 0000 |
| 1 | 0000 0000 |
| 2 | 1011 0000 |
| 3 | 0000 1010 |
| 4 | 0101 0000 |
| 5 | 0001 0000 |
| 6 | 0010 0000 |
| 7 | 0001 0000 |
| 8 | 1001 0000 |
| 9 | 0000 0000 |
| 10 | 0001 0000 |

Una vez abierto el programa, podemos cargarlo a memoria y ejecutar (si está sintácticamente correcto)

Podemos observar en la memoria, cómo se corresponde (cada 2 direcciones) el código y operando de cada una de las instrucciones. ¿ en qué dirección está **sum**? (ayuda **add** tiene el opcode = 5)

Implementando nuestro propio procesador con CPU-SIM - Ejemplo WOMBAT

The screenshot shows the Wombat1 CPU simulator interface. The 'Execute' menu is open, showing options like 'Debug Mode', 'Assemble', and 'Assemble & load'. The main window displays assembly code for a program that reads integers and calculates a sum. The memory window on the right shows a table of addresses and data, with the instruction at address 5 highlighted.

| Addr | Dec | Data | Bin |
|------|-----|------|------|
| 0 | | 0011 | 0000 |
| 1 | | 0000 | 0000 |
| 2 | | 1011 | 0000 |
| 3 | | 0000 | 1010 |
| 4 | | 0101 | 0000 |
| 5 | | 0001 | 0000 |
| 6 | | 0010 | 0000 |
| 7 | | 0001 | 0000 |
| 8 | | 1001 | 0000 |
| 9 | | 0000 | 0000 |
| 10 | | 0001 | 0000 |

Una vez abierto el programa, podemos cargarlo a memoria y ejecutar (si está sintácticamente correcto)

Podemos observar en la memoria, cómo se corresponde (cada 2 direcciones) el código y operando de cada una de las instrucciones. ¿ en qué dirección está **sum**? (ayuda **add** tiene el opcode = 5)

Implementando nuestro propio procesador con CPU-SIM - Ejemplo WOMBAT

Podemos ejecutarlo en **Debug mode** para observar la ejecución de cada instrucción o a nivel de microinstrucciones.

nivel de microinstrucciones

Consola de interacción, para el ejemplo de WOMBAT, instrucciones **read y write**

The screenshot shows the CPU-SIM Wombat1 application window. The menu bar includes File, Edit, Modify, Execute, and Help. The toolbar contains buttons for Go, Step by Instr, Step by Micro, Backup one Instr, Backup one Micro, and Start Over. The Fetch sequence dropdown is set to pc->mar, with Main[mar]->mdr and mdr->ir shown below it. A callout points to this area, stating 'nivel de microinstrucciones'. On the left, the Registers panel shows: acc (16, 0), ir (16, 12288), mar (12, 0), mdr (16, 12288), and status (3, 0). The main assembly window displays code for 'W1-0.a' with instructions like read, jmpn, add, store, jump, load, write, and stop. A callout points to the 'Debug mode' text in the first callout, stating 'Podemos ejecutarlo en Debug mode para observar la ejecución de cada instrucción o a nivel de microinstrucciones.' On the right, the Main memory window shows a table of addresses and data. A callout points to the bottom console area, stating 'Consola de interacción, para el ejemplo de WOMBAT, instrucciones read y write'. The console area at the bottom is highlighted in yellow and contains the text 'Enter Inputs, the first of which must be an Integer:'.

| Addr | Data |
|------|-----------|
| 12 | 0100 0000 |
| 13 | 0000 0000 |
| 14 | 0000 0000 |
| 15 | 0000 0000 |
| 16 | 0000 0000 |
| 17 | 0000 0000 |
| 18 | 0000 0000 |
| 19 | 0000 0000 |
| 20 | 0000 0000 |
| 21 | 0000 0000 |
| 22 | 0000 0000 |
| 23 | 0000 0000 |
| 24 | 0000 0000 |

Enter Inputs, the first of which must be an Integer: