ARQUITECTURA DE COMPUTADORAS Y SISTEMAS OPERATIVOS (SI725)

Guía de Laboratorio de Assembler

Ciclo 2022-02

Índice General

Historial de Revisiones

Revisión	Fecha	Autor(es)	Descripción
1.00	10/03/2019	Geraldo Colchado	Versión inicial
1.01	31/03/2019	Geraldo Colchado	Corrección de palabra “sin” por “con” en página 18
1.02	05/04/2022	Geraldo Colchado	Corrección de “00000002” por “00000010” en página 5

1. Introducción

Esta guía ha sido diseñada para que sirva como una herramienta de aprendizaje y práctica para el curso de Arquitectura de Computadoras y Sistemas Operativos en la Universidad Peruana de Ciencias Aplicadas. En particular se focaliza en el tema "Assembler".

Se busca que el alumno resuelva paso a paso las indicaciones dadas en esta guía contribuyendo de esta manera a los objetivos de aprendizaje del curso, en particular con la comprensión del Assembler. Al finalizar el desarrollo de esta guía y complementando lo que se realizará en el correspondiente laboratorio, se espera que el alumno:

- Comprenda qué es el lenguaje de programación Assembler o Ensamblador.
- Comprenda y sepa utilizar las principales instrucciones de programación soportadas por el simulador de Assembler.
- Implemente algoritmos en el simulador de Assembler.

2. Materiales y Métodos

Como herramienta para la implementación de algoritmos en Assembler se utilizará un simulador de Assembler para una computadora con arquitectura de 8 Bits que simula un CPU x86 de Intel y que tiene 256 bytes de memoria RAM. Este simulador de Assembler utiliza una sintaxis de conjunto de instrucciones de la CPU basado en NASM (Netwide Assembler - https://www.nasm.us/).

El simulador está disponible en línea en este enlace https://schweigi.github.io/assembler-simulator/ desde el cual se puede escribir el programa en lenguaje de programación Assembler y ejecutarlo. Al finalizar esta guía el alumno puede consultar información complementaria del simulador en https://schweigi.github.io/assembler-simulator/instruction-set.html.

3. Lenguaje de Programación Assembler

3.1. Clasificación de Lenguajes de Programación

Los lenguajes de programación se clasifican en 3 tipos según se muestra en la Tabla 1.

Tabla 1: Clasificación de Lenguajes de Programación

Lenguajes de Alto Nivel
Lenguajes de Bajo Nivel
Lenguaje Máquina

El **Lenguaje Máquina** es el lenguaje de programación que entiende directamente el computador y utiliza el sistema de numeración binario (0 y 1) con el cual se escriben instrucciones (cadenas binarias) para que sean entendidas y ejecutadas por el CPU. Cada CPU, dependiendo del modelo y fabricante, puede tener definido

conjuntos de instrucciones que sólo funcionan en ese CPU y no en otros. Debido a lo anterior, no existe un estándar en los conjuntos de instrucciones que soportan los CPUs por lo que hace difícil a los programadores hacer funcionar un programa escrito para una CPU en otra CPU, además es muy fácil cometer errores en la escritura de los programas.

Debido a la complejidad de escribir programas mediante instrucciones en cadenas binarias (combinaciones de ceros y unos), se crearon los **Lenguajes de Bajo Nivel** que sustituyeron los ceros y unos por un lenguaje más parecido al de los seres humanos al cual se le denominó lenguaje de programación Assembler o Ensamblador. Para cada instrucción del conjunto de instrucciones de un CPU se definió su equivalente en lenguaje Assembler. Por ejemplo, la instrucción en Lenguaje Máquina "00000001" fue remplazada por la instrucción en Assembler "MOV" y "00000010" fue reemplazado por "ADD". Note que las instrucciones en Assembler están en inglés y significan MOVER y ADICIONAR respectivamente. Esto facilitó a los programadores la escritura de programas, sin embargo, el Assembler también depende completamente del conjunto de instrucciones de cada CPU y además es necesario traducir el programa escrito en Assembler a Lenguaje Máquina para que pueda ser entendido y ejecutado por el CPU.

Debido a que el Lenguaje Máquina y el Lenguaje Assembler dependen completamente del conjunto de instrucciones de cada CPU y los programas escritos en estos lenguajes no son portables a otros CPU se crearon los **Lenguajes de Alto Nivel** los cuales son independientes de la máquina (CPU) y sus instrucciones son muy similares al lenguaje humano lo que hace más fácil escribir programas y portarlos a otras máquinas (CPU) sin cambios como por ejemplo los lenguajes C, C++, C#, Java, Python, entre otros. Para que los programas puedan ser portables y ejecutados en cualquier CPU se hace necesario primero traducir el programa escrito en Lenguaje de Alto Nivel al Lenguaje Assembler específico de cada CPU y luego a su Lenguaje Máquina.

3.2. Simple 8-bit Assembler Simulator

Para aprender Assembler usaremos un simulador para una CPU x86 de Intel con arquitectura de 8 bits y que tiene 256 bytes de memoria RAM debido a que con 8 bits se puede direccionar a 2 elevado a 8 que es igual a 256 posiciones de memoria. La CPU de este simulador soporta el conjunto de instrucciones indicado en Tabla 2. Note que cada instrucción tiene un código, ejemplo "00000001" el cual le permite al CPU saber que tiene que ejecutar la instrucción "MOV" en el escenario "REG_TO_REG".

Tabla 2: Conjunto de Instrucciones de la CPU del simulador

Código de Instrucción en Lenguaje Máquina			**Instrucción en Lenguaje Assembler**	
Binario (8 bits)	**Equivalente en Hexadecimal**	**Equivalente en Decimal**	**Instrucción**	**Escenario**
00000000	00	0	HLT	
00000001	01	1	MOV	REG_TO_REG
00000010	02	2	MOV	ADDRESS_TO_REG
00000011	03	3	MOV	REGADDRESS_TO_REG
00000100	04	4	MOV	REG_TO_ADDRESS
00000101	05	5	MOV	REG_TO_REGADDRESS
00000110	06	6	MOV	NUMBER_TO_REG
00000111	07	7	MOV	NUMBER_TO_ADDRESS
00001000	08	8	MOV	NUMBER_TO_REGADDRESS

00001010	0A	10	ADD	REG_TO_REG
00001011	0B	11	ADD	REGADDRESS_TO_REG
00001100	0C	12	ADD	ADDRESS_TO_REG
00001101	0D	13	ADD	NUMBER_TO_REG
00001110	0E	14	SUB	REG_FROM_REG
00001111	0F	15	SUB	REGADDRESS_FROM_REG
00010000	10	16	SUB	ADDRESS_FROM_REG
00010001	11	17	SUB	NUMBER_FROM_REG
00010010	12	18	INC	REG
00010011	13	19	DEC	REG
00010100	14	20	CMP	REG_WITH_REG
00010101	15	21	CMP	REGADDRESS_WITH_REG
00010110	16	22	CMP	ADDRESS_WITH_REG
00010111	17	23	CMP	NUMBER_WITH_REG
00011110	1E	30	JMP	REGADDRESS
00011111	1F	31	JMP	ADDRESS
00100000	20	32	JC, JB, JNAE	REGADDRESS
00100001	21	33	JC, JB, JNAE	ADDRESS
00100010	22	34	JNC, JNB, JAE	REGADDRESS
00100011	23	35	JNC, JNB, JAE	ADDRESS
00100100	24	36	JZ, JE	REGADDRESS
00100101	25	37	JZ, JE	ADDRESS
00100110	26	38	JNZ, JNE	REGADDRESS
00100111	27	39	JNZ, JNE	ADDRESS
00101000	28	40	JA, JNBE	REGADDRESS
00101001	29	41	JA, JNBE	ADDRESS
00101010	2A	42	JNA, JBE	REGADDRESS
00101011	2B	43	JNA, JBE	ADDRESS
00110010	32	50	PUSH	REG
00110011	33	51	PUSH	REGADDRESS
00110100	34	52	PUSH	ADDRESS
00110101	35	53	PUSH	NUMBER
00110110	36	54	POP	REG
00110111	37	55	CALL	REGADDRESS
00111000	38	56	CALL	ADDRESS
00111001	39	57	RET	
00111100	3C	60	MUL	REG
00111101	3D	61	MUL	REGADDRESS
00111110	3E	62	MUL	ADDRESS
00111111	3F	63	MUL	NUMBER
01000000	40	64	DIV	REG
01000001	41	65	DIV	REGADDRESS
01000010	42	66	DIV	ADDRESS
01000011	43	67	DIV	NUMBER
01000110	46	70	AND	REG_WITH_REG
01000111	47	71	AND	REGADDRESS_WITH_REG
01001000	48	72	AND	ADDRESS_WITH_REG
01001001	49	73	AND	NUMBER_WITH_REG
01001010	4A	74	OR	REG_WITH_REG
01001011	4B	75	OR	REGADDRESS_WITH_REG
01001100	4C	76	OR	ADDRESS_WITH_REG
01001101	4D	77	OR	NUMBER_WITH_REG
01001110	4E	78	XOR	REG_WITH_REG
01001111	4F	79	XOR	REGADDRESS_WITH_REG
01010000	50	80	XOR	ADDRESS_WITH_REG
01010001	51	81	XOR	NUMBER_WITH_REG
01010010	52	82	NOT	REG
01011010	5A	90	SHL	REG_WITH_REG
01011011	5B	91	SHL	REGADDRESS_WITH_REG
01011100	5C	92	SHL	ADDRESS_WITH_REG
01011101	5D	93	SHL	NUMBER_WITH_REG
01011110	5E	94	SHR	REG_WITH_REG
01011111	5F	95	SHR	REGADDRESS_WITH_REG
01100000	60	96	SHR	ADDRESS_WITH_REG
01100001	61	97	SHR	NUMBER_WITH_REG

Mas adelante en esta guía aprenderemos en detalle a utilizar algunas de las instrucciones soportadas por la CPU del simulador. Por ahora es importante saber que la CPU del simulador soporta las instrucciones con la funcionalidad indicada en Tabla 3 lo cual es la base para empezar a escribir programas en Assembler.

Tabla 3: Funcionalidad de Instrucciones de la CPU del simulador

Tipo de Instrucción	Instrucción	Funcionalidad
Copiar	MOV	**MOV**E: Copia un valor desde un origen hacia un destino
Operaciones Matemáticas	ADD	**ADD**: Suma dos números
	SUB	**SUB**TRACT: Resta dos números
	MUL	**MUL**TIPLY: Multiplica dos números
	DIV	**DIV**IDE: Divide dos números
	INC	**INC**REMENT: Incrementa un número en 1
	DEC	**DEC**REMENT: Decrementa un número en 1
Operaciones Lógicas	AND	Aplica operador lógico **AND** a dos números (0, 1)
	OR	Aplica operador lógico **OR** a dos números (0, 1)
	XOR	Aplica operador lógico **XOR** a dos números (0, 1)
	NOT	Aplica operador lógico **NOT** a un número (0, 1)
Operaciones de desplazamiento de bits	SHL	**SH**IFT **L**EFT: Desplaza todos los 8 bits (0, 1) una posición a la izquierda
	SHR	**SH**IFT **R**IGHT: Desplaza todos los 8 bits (0, 1) una posición a la derecha
Comparar	CMP	**C**O**MP**ARE: Compara dos números si son iguales
Saltar sin condición	JMP	**J**U**MP**: Salta a una dirección de memoria específica para ejecutar la instrucción alojada
Saltar con condición	JC, JB, JNAE	**J**UMP: Salta a una dirección de memoria específica, si se cumple una condición, para ejecutar la instrucción alojada
	JNC, JNB, JAE	
	JZ, JE	
	JNZ, JNE	
	JA, JNBE	
	JNA, JBE	
Operaciones para pila	PUSH	**PUSH**: Pone un valor en la pila
	POP	**POP**: Saca un valor de la pila
Operaciones para subrutina/función	CALL	**CALL**: Llama a una subrutina/función
	RET	**RET**URN: Sale de una subrutina/función y retorna al lugar desde donde fue llamada
Detener	HLT	**HALT**: Detiene la ejecución del programa

4. Ejercicios resueltos

4.1. Programa para sumar dos números – Versión 1 – Sin variables

Si revisamos en la Tabla 3, existe la instrucción ADD que permite sumar dos números. Sin embargo, no es suficiente para implementar el programa en Assembler. El siguiente programa 1 nos permite sumar los números decimales 7 y 8 en Assembler, el cual iremos explicando paso a paso y en el proceso explicaremos algunos conceptos como memoria RAM y registros de CPU del simulador.

Programa 1: Sumar dos números versión 1

```
; Sumar dos números versión 1
MOV A, 7        ; Copia número 7 a registro A
ADD A, 8        ; Adiciona número 8 a registro A
HLT             ; Detiene la ejecución del programa
```

Primero se debe ingresar al simulador y copiar el programa 1 anterior en el recuadro "Code" tal como se muestra en la Figura 1.

Figura 1: Programa 1 – Sumar dos números versión 1

Luego se debe presionar el botón "Assemble" para poder traducir las instrucciones en Lenguaje Assembler a Lenguaje Máquina y cargarlas en la memoria RAM del simulador tal como se muestra en la Figura 2.

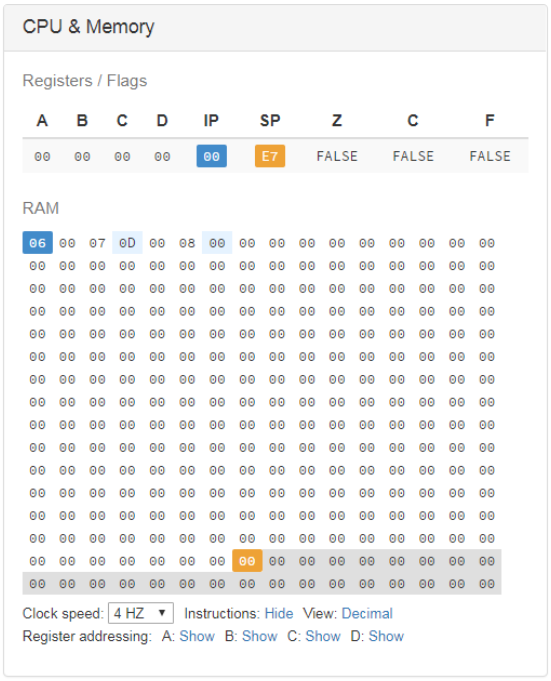


Figura 2: Memoria RAM de Programa 1

La memoria RAM de este simulador tiene 256 posiciones de memoria que se muestran como una matriz de 16 columnas x 16 filas y están numeradas desde la posición 0 hasta la 255. En cada celda se puede almacenar un número binario de hasta 8 bits, es decir hasta 2 elevado a la 8 de números diferentes que equivale a 256 números que van desde el número 0 hasta el 255 inclusive. Para poder facilitar la visualización y por temas de espacio, en este simulador todos los números binarios de 8 bits se han convertido a su equivalente en hexadecimal.

La primera línea del programa 1 no es tomada en cuenta por el traductor porque es un comentario. Los comentarios empiezan con punto y coma, todo lo que está a la derecha del punto y coma no es tomado en cuenta:

```
; Sumar dos números versión 1
```

Para cargar la primera instrucción del programa 1 en la memoria RAM:

```
MOV A, 7        ; Copia número 7 a registro A
```

El traductor tiene que buscar el código de instrucción en lenguaje máquina equivalente obtenido de la Tabla 2 (Conjunto de Instrucciones de la CPU del simulador) que corresponde a la instrucción "MOV" en el escenario siguiente:

Código de Instrucción en Lenguaje Máquina			Instrucción en Lenguaje Assembler	
Binario (8 bits)	**Equivalente en Hexadecimal**	**Equivalente en Decimal**	**Instrucción**	**Escenario**
00000110	06	6	MOV	NUMBER_TO_REG

Y lo carga en el inicio de la memoria RAM utilizando el código de instrucción en lenguaje máquina en hexadecimal "06" que es colocado en la primera celda de 8 bits (ó 1 byte) de la memoria RAM tal como se muestra en la Figura 3, luego coloca en la segunda celda de memoria RAM el valor "00" que es el identificador del registro de la CPU "A". La CPU de este simulador maneja 4 registros denominados A, B, C, D que tienen los identificadores en hexadecimal "00", "01", "02", "03" respectivamente y que sirven para almacenar números para realizar operaciones más rápido que usando la memoria RAM dado que están dentro de la CPU. Luego en la tercera celda de memoria RAM se coloca el valor "07" que equivale al número 7 en hexadecimal. Finalmente se actualiza el registro de la CPU "IP" o Instruction Pointer con el valor en hexadecimal "00" que significa la dirección de la primera celda de memoria RAM donde se debe iniciar la ejecución del programa.

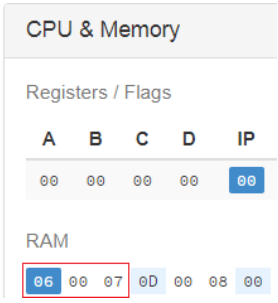


Figura 3: Primera instrucción de programa 1

Para cargar la segunda instrucción del programa 1 en la memoria RAM:

```
ADD A, 8        ; Adiciona número 8 a registro A
```

El traductor tiene que buscar el código de instrucción en lenguaje máquina equivalente obtenido de la Tabla 2 (Conjunto de Instrucciones de la CPU del simulador) que corresponde a la instrucción "ADD" en el escenario siguiente:

Código de Instrucción en Lenguaje Máquina			Instrucción en Lenguaje Assembler	
Binario (8 bits)	**Equivalente en Hexadecimal**	**Equivalente en Decimal**	**Instrucción**	**Escenario**
00001101	0D	13	ADD	NUMBER_TO_REG

Y lo carga en la siguiente celda disponible (cuarta celda) de la memoria RAM utilizando el código de instrucción en lenguaje máquina en hexadecimal "0D" tal como se muestra en la Figura 4, luego coloca en la celda siguiente el valor "00" que es el identificador del registro de la CPU "A" y finalmente en la celda siguiente coloca el valor "08" que equivale al número 8 en hexadecimal.

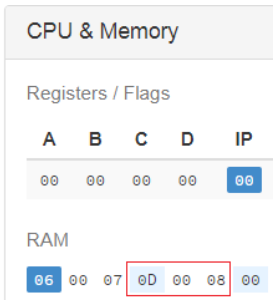


Figura 4: Segunda instrucción de programa 1

Para cargar la tercera instrucción del programa 1 en la memoria RAM:

```
HLT             ; Detiene la ejecución del programa
```

El traductor tiene que buscar el código de instrucción en lenguaje máquina equivalente obtenido de la Tabla 2 (Conjunto de Instrucciones de la CPU del simulador) que corresponde a la instrucción "HLT":

Código de Instrucción en Lenguaje Máquina			Instrucción en Lenguaje Assembler	
Binario (8 bits)	**Equivalente en Hexadecimal**	**Equivalente en Decimal**	**Instrucción**	**Escenario**
00000000	00	0	HLT	

Y lo carga en la siguiente celda disponible (sétima celda) de la memoria RAM utilizando el código de instrucción en lenguaje máquina en hexadecimal "00" tal como se muestra en la Figura 5.

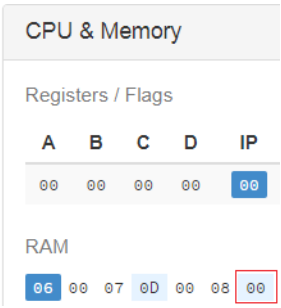


Figura 5: Tercera instrucción de programa 1

Para ejecutar el programa en lenguaje máquina, ya cargado en memoria RAM, se puede hacer presionando el botón "Run" para ejecutar todo el programa de una vez o presionando el botón "Step" para ejecutar instrucción por instrucción o paso a paso de la memoria RAM. Los botones se muestran en la Figura 6.

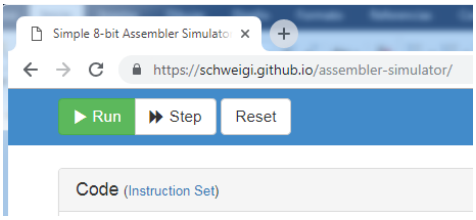


Figura 6: Botones para ejecutar programa

En nuestro caso usaremos el botón "Step". Al presionar el botón "Step" se ejecuta la primera instrucción de la memoria RAM en lenguaje máquina (`06 00 07`) que equivale en lenguaje assembler (`MOV A, 7`) y que copia el valor "07" al registro A y actualiza el registro IP Instruction Pointer con "03" en hexadecimal (dirección de cuarta celda de memoria) que corresponde a la posición de memoria con dirección 3 en decimal considerando que las direcciones de memoria empiezan en 0 en decimal hasta 255 en decimal. Los resultados se pueden observar en la Figura 7.

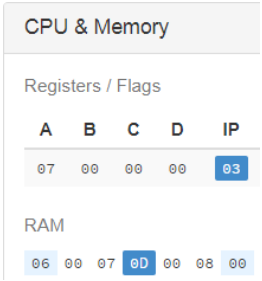


Figura 7: Step 1 de programa 1

Al presionar nuevamente el botón "Step" se ejecuta la segunda instrucción de la memoria RAM en lenguaje máquina (`0D 00 08`) que equivale en lenguaje assembler (`ADD A, 8`) y que adiciona o suma el valor "08" con el valor anterior del registro A que era "07" quedando el registro A con el valor de "0F" en hexadecimal que equivale a 15 en decimal. Se actualiza el registro IP Instruction Pointer con "06" en hexadecimal (dirección de sétima celda de memoria). Los resultados se pueden observar en la Figura 8.

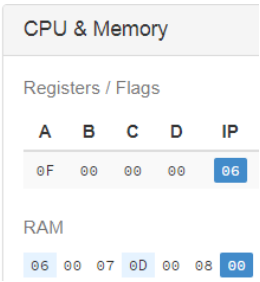


Figura 8: Step 2 de programa 1

Al presionar nuevamente el botón "Step" se ejecuta la tercera instrucción de la memoria RAM en lenguaje máquina (`00`) que equivale en lenguaje assembler (`HLT`) y que le informa al CPU que finalice o detenga la ejecución del programa. Ningún registro de la CPU es cambiado luego de ejecutar `HLT`.

Si no se está familiarizado con la numeración en hexadecimal se puede seleccionar el enlace "Decimal" dentro del recuadro "CPU & Memory" tal como se muestra en la Figura 9 y se convierten todos los valores de Hexadecimal a Decimal o viceversa.

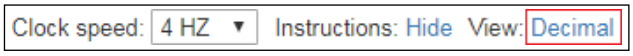


Figura 9: Visualizar en Decimal o Hexadecimal

En la Figura 10 se muestran los valores de los registros de la CPU y memoria RAM en números decimales.

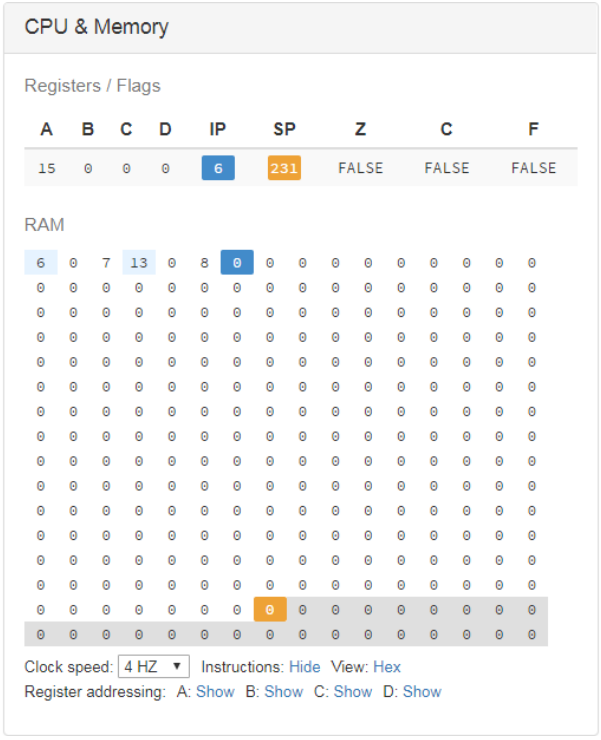


Figura 10: Resultados de ejecución de programa 1 en numeración decimal

4.2. Programa para sumar dos números – Versión 2 – Con variables

En la segunda versión del programa para sumar dos números usaremos variables. Las variables en assembler se definen con la palabra "DB". El siguiente programa 2 nos permite sumar los números decimales 7 y 8 en Assembler utilizando variables.

Programa 2: Sumar dos números versión 2

```
; Sumar dos números versión 2
        JMP start       ; Salta a etiqueta start

nume1:   DB 7           ; Variable número 1
nume2:   DB 8           ; Variable número 2

start:
        MOV A, [nume1] ; Copia contenido de nume1 a registro A
        ADD A, [nume2] ; Adiciona contenido de nume2 a registro A
        HLT            ; Detiene la ejecución del programa
```

En este programa 2 se muestran tres etiquetas (*labels*) con los nombres “nume1”, “nume2” y “start”. Luego de copiar el programa 2 en el simulador y presionar el botón “Assemble” se traduce el programa en assembler a lenguaje máquina el cual es cargado en la memoria RAM tal como se muestra en la Figura 11.

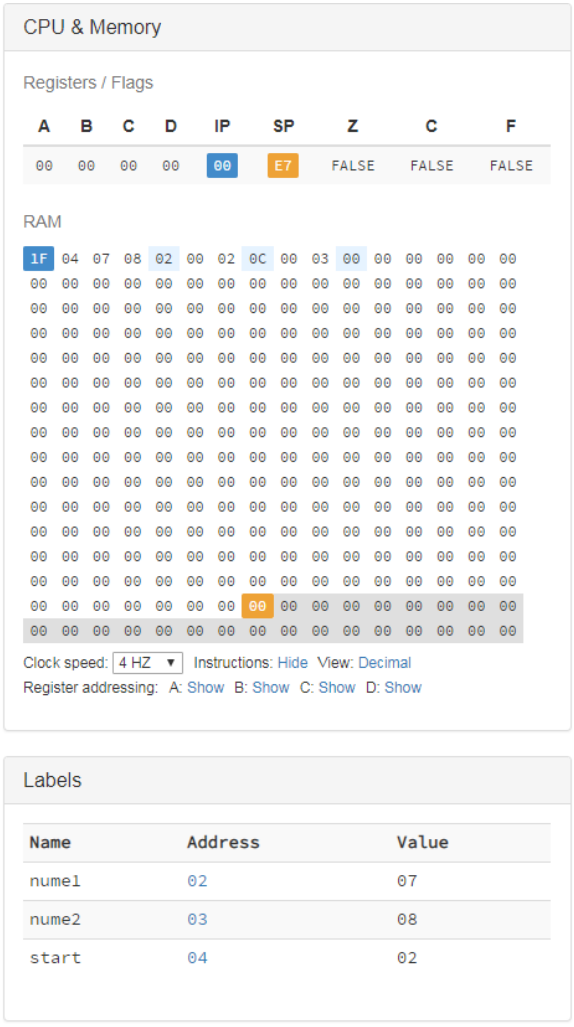


Figura 11: Memoria RAM y etiquetas (Labels) de programa 2

El traductor descarta la primera línea del programa 2 porque es un comentario y procesa la primera instrucción:

```
JMP start
```

El traductor tiene que buscar el código de instrucción en lenguaje máquina equivalente obtenido de la Tabla 2 (Conjunto de Instrucciones de la CPU del simulador) que corresponde a la instrucción "JMP" en el escenario siguiente:

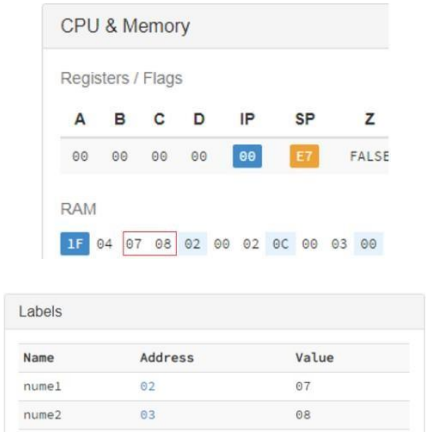
Código de Instrucción en Lenguaje Máquina			Instrucción en Lenguaje Assembler	
Binario (8 bits)	**Equivalente en Hexadecimal**	**Equivalente en Decimal**	**Instrucción**	**Escenario**
00011111	1F	31	JMP	ADDRESS

El traductor detecta que "start" es una etiqueta o *label* y lo deja pendiente porque no sabe aún en qué dirección de memoria está. Copia en la memoria en la primera celda "1F" tal como se muestra en la Figura 11 y reserva la siguiente celda de memoria para la dirección del label "start".

El traductor procesa estas 2 líneas de programa y se da cuenta que son 2 etiquetas o *labels* "nume1" y "nume2" y que en cada etiqueta se define una variable con un valor numérico decimal inicializado.

```
nume1:  DB 7            ; Variable número 1
nume2:  DB 8            ; Variable número 2
```

Luego copia el valor de la primera variable "07" (7 en hexadecimal) en la siguiente celda disponible y copia el valor de la segunda variable "08" (8 en hexadecimal) en la siguiente celda disponible y registra en una tabla de etiquetas o *Labels* la dirección de memoria o *address* donde almacenó los valores o *value* de ambas variables tal como se muestra a continuación:



El traductor procesa esta línea de programa y encuentra la etiqueta o *label* "start":

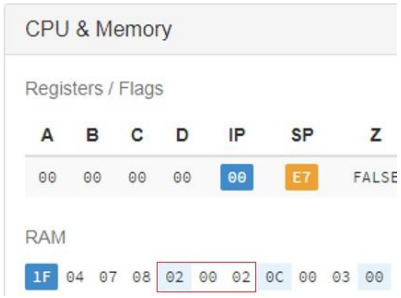
```
start:
```

Luego encuentra esta instrucción y tiene que buscar el código de instrucción en lenguaje máquina equivalente obtenido de la Tabla 2 (Conjunto de Instrucciones de la CPU del simulador) que corresponde a la instrucción "MOV" en el escenario siguiente:

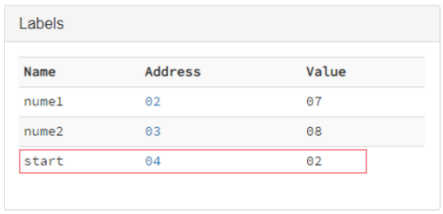
```
MOV A, [nume1] ; Copia contenido de nume1 a registro A
```

Código de Instrucción en Lenguaje Máquina			Instrucción en Lenguaje Assembler	
Binario (8 bits)	**Equivalente en Hexadecimal**	**Equivalente en Decimal**	**Instrucción**	**Escenario**
00000010	02	2	MOV	ADDRESS_TO_REG

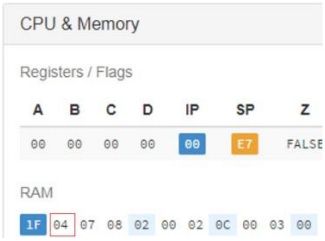
Y como nume1 es una etiqueta o *label* entonces consulta la tabla de *Labels* y obtiene la dirección o *address* "02". Finalmente se copian los valores indicados en recuadro rojo en la memoria:



Adicionalmente se registra en la tabla de etiquetas o *Labels* a "start" con su dirección o *address* "04" y valor o *value* respectivo "02" tal como se muestra a continuación:



Y se actualiza la dirección o *address* de *label* "start" en la celda siguiente a la primera instrucción "1F" ("JMP") tal como se muestra en recuadro rojo a continuación:

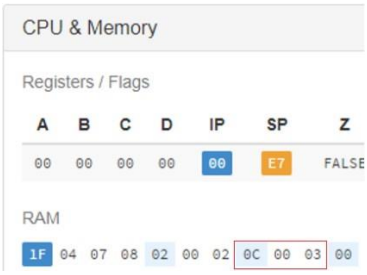


Luego encuentra esta instrucción y tiene que buscar el código de instrucción en lenguaje máquina equivalente obtenido de la Tabla 2 (Conjunto de Instrucciones de la CPU del simulador) que corresponde a la instrucción "ADD" en el escenario siguiente:

```
ADD A, [nume2] ; Adiciona contenido de nume2 a registro A
```

Código de Instrucción en Lenguaje Máquina			Instrucción en Lenguaje Assembler	
Binario (8 bits)	**Equivalente en Hexadecimal**	**Equivalente en Decimal**	**Instrucción**	**Escenario**
00001100	0C	12	ADD	ADDRESS_TO_REG

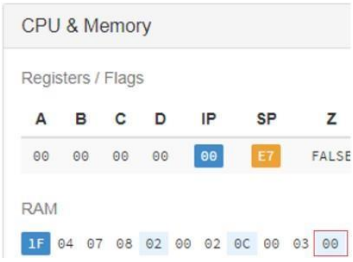
Y como nume2 es una etiqueta o *label* entonces consulta la tabla de *Labels* y obtiene la dirección o *address* "03". Finalmente se copian los valores indicados en recuadro rojo en la memoria:



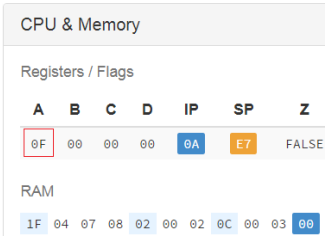
Luego encuentra esta instrucción y tiene que buscar el código de instrucción en lenguaje máquina equivalente obtenido de la Tabla 2 (Conjunto de Instrucciones de la CPU del simulador) que corresponde a la instrucción "HLT":

Código de Instrucción en Lenguaje Máquina			Instrucción en Lenguaje Assembler	
Binario (8 bits)	**Equivalente en Hexadecimal**	**Equivalente en Decimal**	**Instrucción**	**Escenario**
00000000	00	0	HLT	

Y copia en la siguiente celda de memoria disponible el valor "00" tal como se muestra a continuación en recuadro rojo:



Ejecute paso a paso y analice qué se realiza en cada instrucción ejecutada. Una vez ejecutado todo el programa obtendrá la suma "0F" en hexadecimal que equivale a 15 en decimal en el registro A tal como se muestra a continuación en recuadro rojo:



4.3. Programa para sumar dos números – Versión 3 – Con función

En la tercera versión del programa para sumar dos números usaremos una función para realizar la suma. El siguiente programa 3 nos permite sumar los números decimales 7 y 8 en Assembler utilizando una función.

Programa 3: Sumar dos números versión 3

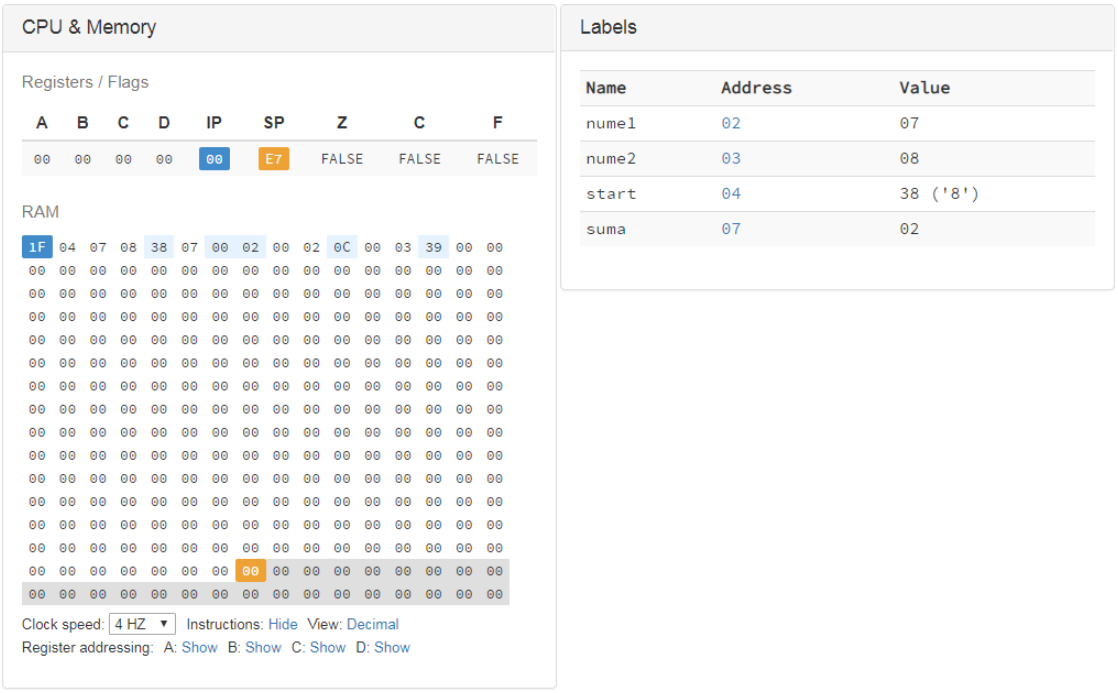
```
; Sumar dos números versión 3
        JMP start       ; Salta a etiqueta start

nume1:   DB 7           ; Variable número 1
nume2:   DB 8           ; Variable número 2

start:
        CALL suma       ; Llama a función con etiqueta suma
        HLT             ; Detiene la ejecución del programa

suma:
        MOV A, [nume1] ; Copia contenido de nume1 a registro A
        ADD A, [nume2] ; Adiciona contenido de nume2 a registro A
        RET             ; Sale de función
```

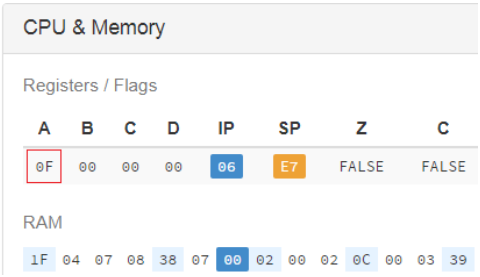
Copie el programa 3 en el simulador y presione el botón "Assemble" para traducir a lenguaje máquina y cargar en memoria RAM el programa tal como se muestra a continuación:



Analice cada instrucción colocada en memoria RAM y justifique cada valor colocado en cada celda de memoria.

Ejecute paso a paso y analice qué se realiza en cada instrucción ejecutada, note que se utiliza el registro de CPU "SP" *Stack Pointer* o puntero de pila cuando se llama a la función suma para almacenar la dirección "06" de la siguiente instrucción a ejecutar luego de completar la función suma. Una vez ejecutado todo el programa

obtendrá la suma "0F" en hexadecimal que equivale a 15 en decimal en el registro A tal como se muestra a continuación en recuadro rojo:



4.4. Programa para obtener el mayor de dos números

El siguiente programa 4 nos permite obtener el mayor de dos números decimales 15 y 20 en Assembler.

Programa 4: Obtener el mayor de dos números

```
; Obtiene el mayor de 2 números en registro B

        JMP start               ; Salta a etiqueta start

nume1:   DB 15                  ; Variable número 1
nume2:   DB 20                  ; Variable número 2

start:
        MOV A, [nume1]          ; Copia contenido nume1 a registro A
        CMP A, [nume2]          ; Compara nume1 y nume2 si son =
        JBE .menor_igual        ; If nume1<=nume2 salta a .menor_igual
        MOV B, A                ; Else nume1 es el mayor
        JMP end                 ; Salta a etiqueta end

.menor_igual:
        MOV B, [nume2]          ; nume2 es el mayor
        JMP end                 ; Salta a etiqueta end

end:
        HLT                     ; Detiene la ejecución del programa
```

En este programa se ha utilizado la instrucción en assembler "CMP" que permite comparar dos números y establecer el Flag "Z" Zero en TRUE si son iguales. "CMP" se utiliza siempre antes de una instrucción de saltar con condición como la usada en este programa "JBE". En la figura 12 primera columna "*Instruction*" se muestran todas las instrucciones para saltar con condición (*conditional jumps*), en la segunda columna "*Description*" se muestra en qué caso se utiliza y en la tercera columna "*Condition*" se muestran los valores que toman los Flags de CPU "C" Carry y "Z" Zero. En el programa 4 se ha usado la instrucción "JBE" (**J**ump if **B**elow or **E**qual) que es para "<=" con "C = TRUE".

Conditional jumps

Let the instruction pointer do a conditional jump to the defined address. See the table below for the available conditions.

Instruction	Description	Condition	Alternatives
JC	Jump if carry	Carry = TRUE	JB, JNAE
JNC	Jump if no carry	Carry = FALSE	JNB, JAE
JZ	Jump if zero	Zero = TRUE	JB, JE
JNZ	Jump if no zero	Zero = FALSE	JNE
JA	>	Carry = FALSE && Zero = FALSE	JNBE
JNBE	not <=	Carry = FALSE && Zero = FALSE	JA
JAE	>=	Carry = FALSE	JNC, JNB
JNB	not <	Carry = FALSE	JNC, JAE
JB	<	Carry = TRUE	JC, JNAE
JNAE	not >=	Carry = TRUE	JC, JB
JBE	<=	C = TRUE or Z = TRUE	JNA
JNA	not >	C = TRUE or Z = TRUE	JBE
JE	=	Z = TRUE	JZ
JNE	!=	Z = FALSE	JNZ

Figura 12: Instrucciones para saltar con condición (Jump). Fuente: https://schweigi.github.io/assembler-simulator/instruction-set.html

Copie el programa 4 en el simulador y presione el botón "Assemble" para traducir a lenguaje máquina y cargar en memoria RAM el programa. Analice cada instrucción colocada en memoria RAM y justifique cada valor colocado en cada celda de memoria.

Ejecute paso a paso y analice qué se realiza en cada instrucción ejecutada. Una vez ejecutado todo el programa obtendrá el mayor de los números en el Registro de la CPU "B" con valor 14 en hexadecimal que equivale al número 20 en decimal tal como se muestra a continuación en recuadro rojo:

Cambie en el programa 4 los valores de las variables nume1 y nume2 por estos valores, presione el botón "Assemble" y ejecute paso a paso. Analice y valide los resultados.

```
nume1:  DB 37                ; Variable número 1
nume2:  DB 20                ; Variable número 2
```

Cambie en el programa 4 los valores de las variables nume1 y nume2 por estos valores, presione el botón "Assemble" y ejecute paso a paso. Analice y valide los resultados.

```
nume1:  DB 20                ; Variable número 1
nume2:  DB 20                ; Variable número 2
```

4.5. Programa para calcular la potencia de un número

El siguiente programa 5 nos permite calcular la potencia del número decimal 2 elevado a 7 en Assembler.

Programa 5: Calcular la potencia de un número

```
; Calcula potencia de 2 elevado a la 7 = 128 en registro A
        JMP start       ; Salta a etiqueta start

nume:   DB 2            ; Variable número
pote:   DB 7            ; Variable potencia

start:
        MOV A, [nume] ; Copia contenido de nume a registro A
        CALL potencia ; Llama a función con etiqueta potencia
        HLT           ; Detiene la ejecución del programa

potencia:
        MOV B, 1        ; Inicializa registro B en valor 1 para contar
.loop:
        MUL [nume]      ; Multiplica contenido de nume por registro A
        INC B           ; Aumenta en 1 el contador
        CMP B, [pote]   ; Compara registro B con potencia si son =
        JB .loop        ; If B<pote salta a .loop

        RET             ; Sale de función
```

En el programa 5 se ha usado la instrucción "JB" (**J**ump if **B**elow) que es para "<" con "C = TRUE" según Figura 12.

Copie el programa 5 en el simulador y presione el botón "Assemble" para traducir a lenguaje máquina y cargar en memoria RAM el programa. Analice cada instrucción colocada en memoria RAM y justifique cada valor colocado en cada celda de memoria.

Ejecute paso a paso y analice qué se realiza en cada instrucción ejecutada. Una vez ejecutado todo el programa obtendrá la potencia de 2 elevado a la 7 en el Registro de la CPU "A" con valor 80 en hexadecimal que equivale al número 128 en decimal tal como se muestra a continuación en recuadro rojo:

5. Ejercicios propuestos

Elabore un programa en assembler para cada uno de los ejercicios propuestos que incluya comentario en cada instrucción y ejecútelo en el simulador de assembler.

5.1. Suma de números naturales

Se desea calcular la suma de los números naturales que existen en un rango [a..b]. Por ejemplo, si a = 5 y b = 10, se deberá calcular la sumatoria = 5 + 6 + 7 + 8 + 9 + 10 = 45.

5.2. Resolver fórmula matemática

Se desea resolver la siguiente fórmula matemática: ((6*5) + (20-10) - (10-5)) / 7 = 5

Sugerencia 1: Utilice los registros de la CPU A, B, C, D para ir almacenando los resultados parciales de la fórmula.

Sugerencia 2: Utilice la pila que se almacena en la memoria RAM y las instrucciones en assembler "PUSH" y "POP" para poner y sacar valores de la pila para ir resolviendo la fórmula por resultados parciales.

5.3. Tabla de verdad de AND

Se desea comprobar la tabla de verdad del operador lógico AND:

A	B	A AND B
0	0	0
0	1	0
1	0	0
1	1	1

Sugerencia: Utilice los registros de CPU A y B para almacenar los valores de cada fila de la tabla de verdad y luego ejecute la instrucción "AND" y valide el Flag Z de la CPU que cambia de valor.

5.4. Tabla de verdad de OR

Se desea comprobar la tabla de verdad del operador lógico OR:

A	B	A OR B
0	0	0
0	1	1
1	0	1
1	1	1

Sugerencia: Utilice los registros de CPU A y B para almacenar los valores de cada fila de la tabla de verdad y luego ejecute la instrucción "OR" y valide el Flag Z de la CPU que cambia de valor.

5.5. Menor de tres números

Se desea obtener el menor de tres números decimales.