COMPUTADORAS DE BOLSILLO

FX-785P/FX-790P

INTRODUCCION AL LENGUAJE DE ENSAMBLE

CASIO

COMPUTADORAS DE BOLSILLO

FX-785P/FX-790P

INTRODUCCION AL LENGUAJE DE ENSAMBLE

CASIO®

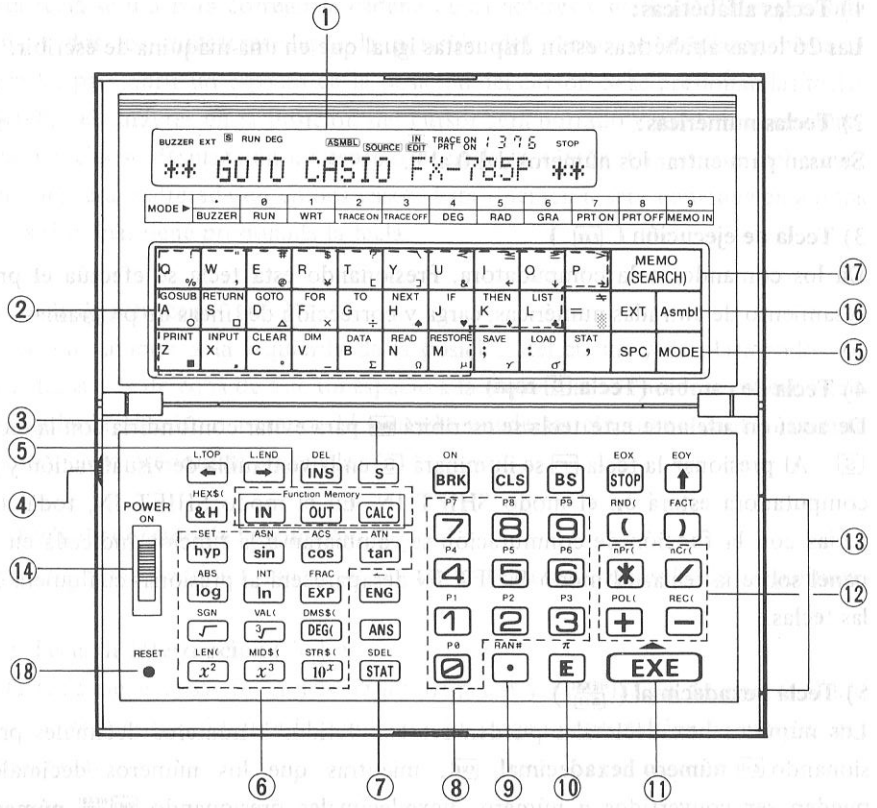
Introducción

En este manual estudiaremos el principio de operación de la computadora. Con este propósito imaginaremos una computadora inexistente de una estructura muy simple. (A la que nos referiremos de aquí en adelante como la computadora hipotética.) Esta unidad tiene la función de imitar la operación de la computadora hipotética. Es capaz de ejecutar programas al entrarlos en lenguaje de ensamble y traducirlos a lenguaje de máquina. Por lo tanto, esta unidad le permitirá aprender el principio de las operaciones de la computadora, que es común a todas las computadoras, desde las manuales a las computadoras de propósitos generales a escala completa.

Tabla de contenidos

1 Guía general

1-1 Nombres de los componentes



① Ventanilla de visualización
② Teclas alfabéticas
③ Tecla de cambio
④ Contraste de visualización
⑤ Teclas de la memoria de funciones
⑥ Teclas de funciones
⑦ Tecla de entrada de datos estadísticos
⑧ Teclas numéricas
⑨ Tecla de la coma decimal
⑩ Tecla del exponente
⑪ Tecla de ejecución
⑫ Teclas de comandos de cálculo
⑬ Conector para los equipos periféricos
⑭ Interruptor de encendido
⑮ Tecla de modo
⑯ Tecla de montaje
⑰ Tecla de apuntes/búsqueda
⑱ Botón RESET (de reposición)

* Los contenidos visualizados corresponden a la FX-785P.

1-2 Funciones de los componentes

1) **Teclas alfabéticas:**
Las 26 letras alfabéticas están dispuestas igual que en una máquina de escribir.

2) **Teclas numéricas:**
Se usan para entrar los números del 0 al 9.

3) **Tecla de ejecución (EXE)**
Dá los comandos a la computadora. Presionando esta tecla se efectúa el procesamiento de entradas numéricas, carga y corrección de líneas de programas.

4) **Tecla de cambio (Tecla S roja)**
De aquí en adelante este tecla se escribirá SHIFT para evitar confundirla con la tecla S . Al presionar la tecla SHIFT se iluminará S en la ventanilla de visualización y la computadora estará en el modo SHIFT IN. En el modo SHIFT IN, todas las teclas con la función de conmutación se cambiarán a la función indicada en el panel sobre la tecla. El modo SHIFT IN desaparecerá al presionar cualquiera de las teclas.

5) **Tecla hexadecimal (HEX$(&H)**
Los números hexadecimales pueden ser convertidos a números decimales presionando &H **número hexadecimal** EXE, mientras que los números decimales pueden ser convertidos a números hexadecimales presionando SHIFT HEX$(**número decimal**) EXE .

6) **Teclas de movimiento del cursor (LTOP LEND)**
Estas teclas se usan para mover el cursor hacia adelante y hacia atrás. Si se presiona SHIFT LTOP, el cursor se moverá al principio de la línea, mientras que si se presiona SHIFT LEND, éste se moverá a la derecha del último carácter.

7) **Tecla de inserción/borrado (DEL INS)**
Esta tecla se usa para corregir la cadena de caracteres visualizada. Si se presiona INS , todos los caracteres desde la posición del cursor se desplazan hacia la derecha para abrir un espacio en la posición del cursor. Si se presionan las teclas SHIFT DEL , el carácter en la posición del cursor será borrado y todos los caracteres a la derecha se desplazarán un espacio hacia la izquierda. La posición del cursor permanecerá inalterada en ambos casos. Esta operación será contínua en ambos casos si se mantiene presionada la tecla.

8) **Tecla de retroceso (BS)**
Borra un carácter a la izquierda de la posición del cursor y desplaza todos los caracteres a la derecha de éste un espacio a la izquierda. El cursor se mueve hacia la izquierda al mismo tiempo. El borrado puede ser efectuado contínuamente manteniendo presionada esta tecla.

9) **Tecla de borrado de la pantalla (CLS)**
Borra la visualización y mueve el cursor al extremo izquierdo de la pantalla.

10) **Tecla de interrupción (BRK)**
Esta tecla tiene varias funciones de interrupción y cancelación de errores. También, si se presiona durante la condición de apagado automático la energía será restaurada.

11) **Tecla de detención (STOP)**
Si se presiona esta tecla durante el recorrido (scrolling) de la visualización, éste se detendrá temporalmente. El recorrido volverá a comenzar al presionar la tecla EXE .

12) **Tecla de apuntes/búsqueda (MEMO SEARCH)**
Esta se usa para visualizar el programa fuente en el área fuente.

13) **Tecla de ensamble (Asmbl)**
Visualiza el menú de ensamble. Ver el Capítulo 10 sobre su método de uso.

14) **Tecla de modo ([MODE])**

Esta tecla especifica el modo de la computadora y se usa en combinación con las teclas [.], [0] ~ [9]. Los siguientes modos relativos se usan en este manual.

[MODE] [.] . . . . Activa y desactiva el zumbador de entrada de teclas. Cuando el zumbador está activado, el símbolo "BUZZER" aparece en la parte superior izquierda de la pantalla.

[MODE] [0] . . . . Fija el modo RUN.

[MODE] [1] . . . . Fija el modo WRT.

[MODE] [2] . . . . Visualiza el símbolo "TRACE ON" y efectúa el seguimiento de la ejecución de un programa objeto (Ver 10-5 en el Capítulo 10).

[MODE] [3] . . . . Cancela el modo de seguimiento (TRACE).

[MODE] [7] . . . . Visualiza el símbolo "PRT ON" y fija el modo de salida de la impresora.

[MODE] [8] . . . . Cancela el modo de salida de la impresora.

2 Método de notación numérica

En primer lugar se resumirán los ítems necesarios relativos a las notaciones binaria y hexadecimal, las cuales se usarán con posterioridad.

- **Notación binaria**

Toda la información, como por ejemplo los programas y datos, se expresa en notación binaria. En notación binaria sólo se utilizan dos números, 0 y 1. Los números expresados en notación binaria se denominan números binarios, y cada dígito de un número binario es llamado **bit** ("dígito binario"). Por lo tanto, la información expresada por un número binario de ***n*** dígitos se puede decir que consta de ***n*** **bits**.

Abajo se indican los números decimales del 0 al 10, que son los que usamos con mayor frecuencia, en notación binaria junto con la importancia de cada dígito del número binario.

Número decimal	Número binario
0	0
1	1
2	10
3	11
4	100
5	101
6	110
7	111
8	1000
9	1001
10	1010

Importancia de cada dígito de un número binario	Número decimal correspondiente
2^0	1
2^1	2
2^2	4
2^3	8
2^4	16
2^5	32
2^6	64
2^7	128
2^8	256
2^9	512
2^{10}	1024
2^{11}	2048
2^{12}	4096
2^{13}	8192
2^{14}	16384
2^{15}	32768
2^{16}	65536

Por ejemplo, el número decimal correspondiente al número binario 10101010 será $2^7 + 2^5 + 2^3 + 2^1 = 170$.

- **Notación hexadecimal**

Aunque la notación binaria es conveniente para usarla con computadoras, a menudo se usa la notación hexadecimal cuando se requiere de un gran número de dígitos para expresar un número, ya que es más fácil de leer. En notación hexadecimal se utilizan 16 caracteres, 0 ~ 9 y A ~ F, y los números expresados en este tipo de notación son denominados números hexadecimales. La tabla siguiente muestra la correspondencia entre los números hexadecimales y los números decimales.

Números hexadecimales	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
Números decimales	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

En esta computadora se pueden efectuar conversiones hexadecimal y decimal usando la tecla HEX$ &H.

Como $2^4 = 16$, un número binario de 4 dígitos (4 bits) corresponde a un número hexadecimal de 1 dígito. Por lo tanto podemos expresar un número binario usando menos dígitos al expresarlo en notación hexadecimal, mientras aun mantenemos la estructura binaria.

La notación hexadecimal se puede efectuar separando los números binarios en grupos de 4 dígitos a partir de los dígitos de menor orden y convirtiendolos en un dígito hexadecimal por grupo, como se muestra en la tabla de la derecha. Si los dígitos de mayor orden son insuficientes, basta con agregar ceros.

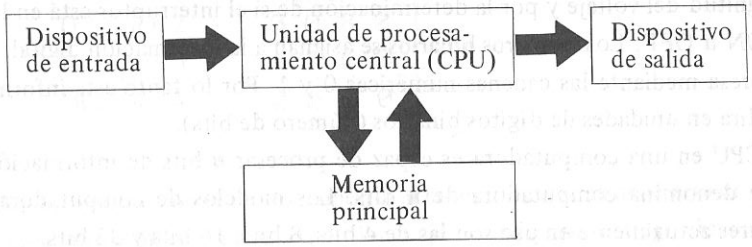
Número binario de 4 dígitos	Número hexadecimal
0 0 0 0	0
0 0 0 1	1
0 0 1 0	2
0 0 1 1	3
0 1 0 0	4
0 1 0 1	5
0 1 1 0	6
0 1 1 1	7
1 0 0 0	8
1 0 0 1	9
1 0 1 0	A
1 0 1 1	B
1 1 0 0	C
1 1 0 1	D
1 1 1 0	E
1 1 1 1	F

Ejemplo:

	Números binarios	**Números hexadecimales**
Sumar	1 0 1 1 0 0 0 1	B 1
	B 1	
	0 1 0 1 1 1 1 1 0 0 1 1 0 1 1 1	5 F 3 7
	5 F 3 7	

3 Configuración de la computadora y principio de operación

Una computadora que lleve a cabo una serie de operaciones generalmente está compuesta de las cuatro secciones siguientes.



La unidad de procesamiento central, que se abrevia **CPU**, es el corazón del sistema de la computadora. Las computadoras actuales están diseñados con un "**sistema de programa almacenado**". Los programas y los datos se almacenan en la **memoria principal**. La CPU recupera esta información en orden sucesivo y la opera de acuerdo a las instrucciones. Aunque en las primeras computadoras se requería de máquinas separadas para tipos de trabajos diferentes, en las computadoras actuales, la CPU misma es altamente avanzada y puede ejecutar variadas tareas con programas almacenados en la memoria principal. La máquina que conforma la computadora misma es denominade "**hardware**" y los programas que dan las instrucciones de operación a la computadora son denominados "**software**".

Para permitir la interacción entre la CPU y los seres humanos se utilizan los dispositivos de entrada y salida.

El **dispositivo de entrada** se usa para leer los programas y datos en la CPU, un ejemplo típico es el teclado. Es decir, juega el mismo papel que para los seres humanos juegan los ojos y oídos.

El **dispositivo de salida** sirve para transmitir los resultados de los cálculos de la CPU hacia el exterior, por ejemplo la **CRT**, una visualización de cristal líquido o una impresora. Estos dispositivos son equivalentes a la boca y manos en los seres humanos. El nombre general para los dispositivos de entrada y salida es "**dispositivos de entrada/salida**" **(I/O)**.

Además de estos dispositivos se usa un **dispositivo de almacenamiento secundario** para almacenar la información que no puede ser mantenida en la memoria principal y también sirve para almacenar programas y datos. Para los dispositivos de almacenamiento secundario se usan **discos magnéticos** y **cintas**. En el caso de los humanos, éstos serían apuntes y notas.

En muchas de las computadoras actuales, la información cuantitativa se procesa en forma digital. La unidad más pequeña de información digital se determina por la magnitud del voltaje y por la determinación de si el interruptor está en la posición ON u OFF. Los **números binarios** se asignan a la información digital, la que se expresa mediante las cadenas numéricas 0 y 1. Por lo tanto esta información se medirá en unidades de dígitos binarios (número de bits).

Si la CPU en una computadora es capaz de procesar *n* bits de información a la vez, se denomina computadora de **n bits**. Los modelos de computadoras más populares actualmente en uso son las de 4 bits, 8 bits, 16 bits y 32 bits.

La memoria principal se divide en secciones pequeñas denominadas **celdas** de acuerdo a un número fijo de bits. Las instrucciones y datos de un programa se almacenan en estas celdas en la forma de **patrones de bits** binarios. Las celdas se distinguen asignando una dirección a cada celda en la forma de un valor entero para mostrar la posición de ella. Normalmente, las direcciones están en orden numérico comenzando desde 0.

Dirección 0	1100010000000011
Dirección 1	0110010000010000
Dirección 2	0000000010000000
Dirección 3	1010101111001101
Dirección 4	0000000000000000
	⋮

(Disposición interna de la memoria principal con 16 bits en cada celda.)

La verdadera forma de un programa consiste simplemente de patrones de bits almacenados en cada celda como se muestra en el diagrama. Como la CPU opera con un cierto tipo de patrón de bit es precisamente lo que se determina al designar la CPU. Este es el único lenguaje que la CPU entiende y es denominado "**lenguaje de máquina**".

Sin embargo, sólo los patrones de bits son muy dificiles de leer, es por ésto que a menudo se usan números hexadecimales correspondientes a los contenidos de cada celda en las notaciones.

Dirección 0	C403
Dirección 1	6410
Dirección 2	0080
Dirección 3	ABCD
Dirección 4	0000
	⋮

(Cada celda se compone de 16 bits.)

Para dar las instrucciones de operación a la computadora, es necesario en primer lugar, crear programas expresados en lenguaje de máquina y luego almacenarlos en la memoria principal. Sin embargo, como ésto es una tarea muy difícil para los seres humanos, los programas se expresan agregando nombres nemotécnicos a cada programa en lenguaje de máquina sugiriendo los contenidos de las instrucciones. El lenguaje de máquina expresado mediante nemotécnicos es llamado **lenguaje de ensamble**.

Si usamos el lenguaje de ensamble para mostrar el programa y los datos almacenados en la memoria principal, éstos aparecerán de la manera siguiente.

Dirección 0	LD :1:3	Instrucciones de la computadora
Dirección 1	WRITE:1:16	
Dirección 2	HJ :0:128	
Dirección 3	CONST:ABCD	Datos (Hexadecimal ABCD)

Si expresamos programas de esta manera mediante lenguaje de ensamble, uno puede decir que hará el programa simplemente mirándolo (en la medida que se entienda la sintaxis del lenguaje de ensamble) y ésto facilitará en gran medida la creación de programas. Sin embargo, la CPU no puede comprender el programa escrito en lenguaje de ensamble, por lo que será necesario convertir este programa a su correspondiente en lenguaje de máquina. Este proceso se llama "**ensamble**".

El ensamble es un trabajo monótono en que el lenguaje de ensamble es convertido a lenguaje de máquina, tomando las instrucciones una a una de acuerdo a una tabla de correspondencia de lenguaje. Esto resultó en la creación de programas para efectuar este trabajo con la computadora. El programa para convertir el lenguaje de ensamble en lenguaje de máquina se llama "**ensamblador**".

*Como cada instrucción escrita en lenguaje de ensamble corresponde a una operación individual de la CPU, será necesario crear un programa que parcialice el procedimiento en pasos muy pequeños para llevar a cabo un cálculo. Ahora se conciben lenguajes de programación que se aproximan al nivel del lenguaje humano para mejorar la productividad de los programas. FORTRAN y COBOL son lenguajes bien conocidos y están en el grupo de los lenguajes de alto nivel.

El lenguaje BASIC que emplea esta computadora también es uno de los lenguajes de alto nivel. Un programa escrito en uno de estos lenguajes de alto nivel se ejecuta en la CPU después de que un programa traductor llamado compilador o interprete lo convierte a lenguaje de máquina. Aun hoy día, cuando existen tantos lenguajes de alto nivel, es importante comprender el lenguaje de ensamble o ensamblador para capacitar a la computadora para que efectúe un trabajo detallado y a la vez haga un uso efectivo de la memoria.

4 Computadora hipotética y simulador

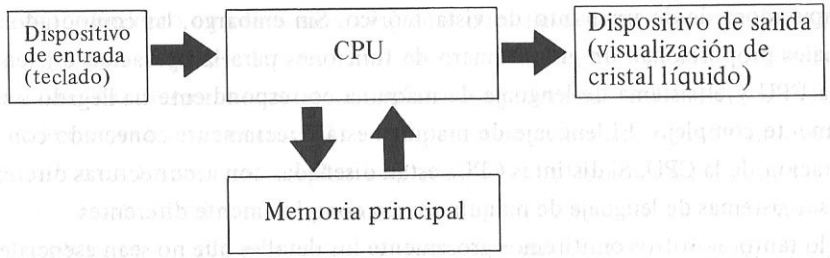
Como se explicó en la sección previa, será necesario aprender programación en lenguaje de máquina o lenguaje de ensamble para comprender la operación de la computadora desde un punto de vista teórico. Sin embargo, las computadoras actuales proporcionan un gran número de funciones para la operación eficiente de la CPU y el sistema de lenguaje de máquina correspondiente ha llegado a ser altamente complejo. El lenguaje de máquina está directamente conectado con la operación de la CPU. Si distintas CPUs están diseñadas con arquitecturas diferentes, sus sistemas de lenguaje de máquina serán completamente diferentes.

Por lo tanto, nosotros omitiremos grosamente los detalles que no sean esenciales, y estudiaremos los principios de operación de la computadora y de programación en lenguaje de máquina asumiendo una computadora hipotética que tiene la arquitectura de lenguaje de máquina necesaria sólo para la operación de una computadora básica.

Cuando una computadora A es capaz de imitar la operación de una computadora B, generalmente se dice que la computadora A es un simulador de la computadora B. Y la computadora tiene las funciones de simulación de la computadora hipotética. También tiene las funciones de ensamblador para traducir programas escritos en lenguaje de ensamble para la computadora hipotética en lenguaje de máquina. Como de aquí en adelante esta computadora será usada como un simulador de la computadora hipotética y no como una computadora BASIC, la llamaremos simplemente **"simulador"**. El lenguaje para expresar el programa simulador será llamado simplemente **"lenguaje de ensamble"**.

5 Hardware del simulador

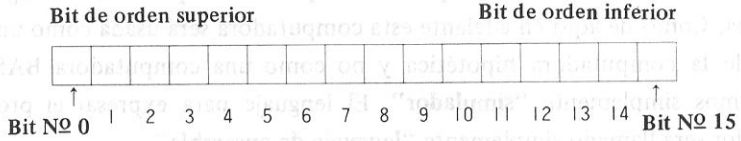
La configuración básica del simulador es la que se muestra a continuación.



*Una impresora exclusiva puede ser usada como dispositivo de salida y una grabadora de cintas cassette como dispositivo de almacenamiento secundario.

- **Memoria principal**

La memoria principal del simulador está dividida en celdas de 16 bits llamadas "**palabras**" y cada palabra está provista de una "**dirección**". Los bits que configuran una palabra están dispuestos desde la izquierda (bit de orden superior) a la derecha (bit de orden inferior) en orden numérico de 0 a 15.

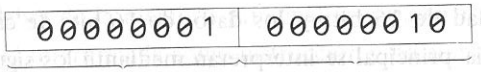


En la memoria principal se pueden almacenar un total de 2048 (2^{11}) palabras y se dispone de direcciones con valores enteros desde 0 a 2047 para estas palabras. Aunque para especificar la dirección de una palabra bastan 11 bits, en la práctica las direcciones se especifican con 16 bits.

Las 256 palabras sucesivas comenzando desde la del múltiplo entero de 256 (incluyendo el 0) son denominadas individualmente bloque de almacenamiento. Las 256 palabras comenzando desde 256 × dirección N (N = 0 ~ 7) son llamadas bloque de almacenamiento Número N. Por lo tanto se pueden obtener hasta ocho bloques de almacenamiento (Número de bloque de almacenamiento 0 ~ Número 7). Los 8 bits de orden superior de los 16 bits de especificación indican el bloque de almacenamiento (0 ó 7) y los 8 bits de orden inferior indican la dirección de la palabra en el bloque de almacenamiento.

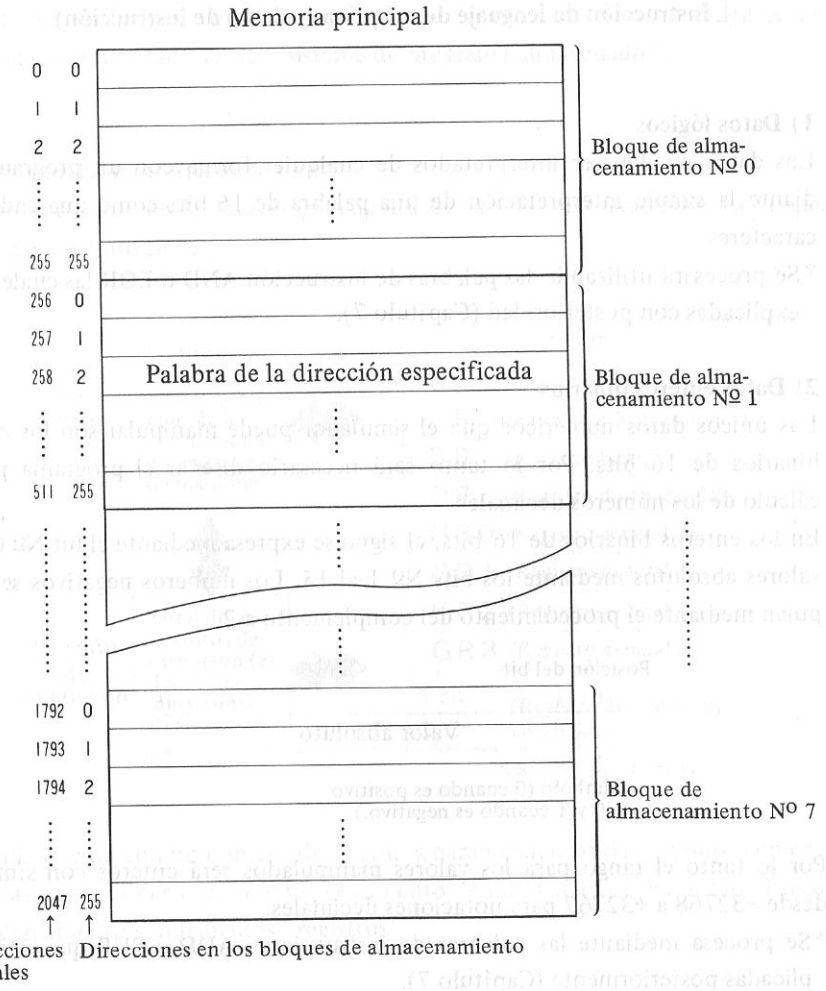
*El número de bloques de almacenamiento (1 ~ 8) que pueden ser asegurados cambiará dependiendo del número restante de bytes (octetos).

Especifica la dirección 258 entre todas las direcciones



El bloque de almacenamiento Nº 1 se especifica mediante los 8 bits de orden superior.

Los 8 bits de orden inferior especifican la Dirección 2 en el bloque de almacenamiento.



- **Interpretación de palabras**

El simulador es una unidad de 16 bits y los datos de 16 bits de cada palabra almacenada en la memoria principal se interpretan mediante los siguientes tres métodos.

Datos lógicos
Datos enteros binarios
Instrucción de lenguaje de máquina (palabra de instrucción)

1) Datos lógicos

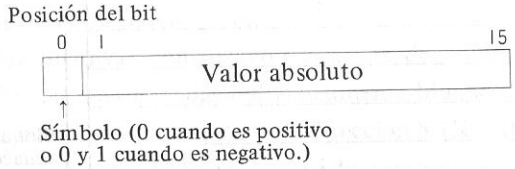
Los datos pueden ser interpretados de cualquier forma con un programa mediante la simple interpretación de una palabra de 16 bits como una cadena de caracteres.

* Se procesará utilizando las palabras de instrucción AND o EOR las cuales serán explicadas con posterioridad (Capítulo 7).

2) Datos enteros binarios

Los únicos datos numéricos que el simulador puede manipular son los enteros binarios de 16 bits. Por lo tanto será necesario diseñar el programa para el cálculo de los números decimales.

En los enteros binarios de 16 bits, el signo se expresa mediante el bit Nº 0 y los valores absolutos mediante los bits Nº 1 al 15. Los números negativos se manipulan mediante el procedimiento del complemento a 2.



Por lo tanto el rango para los valores manipulados será enteros con símbolos desde -32768 a +32767 para notaciones decimales.

* Se procesa mediante las palabras de instrucciones ADD o SUB que serán explicadas posteriormente (Capítulo 7).

3) Instrucción de lenguaje de máquina (palabra de instrucción)

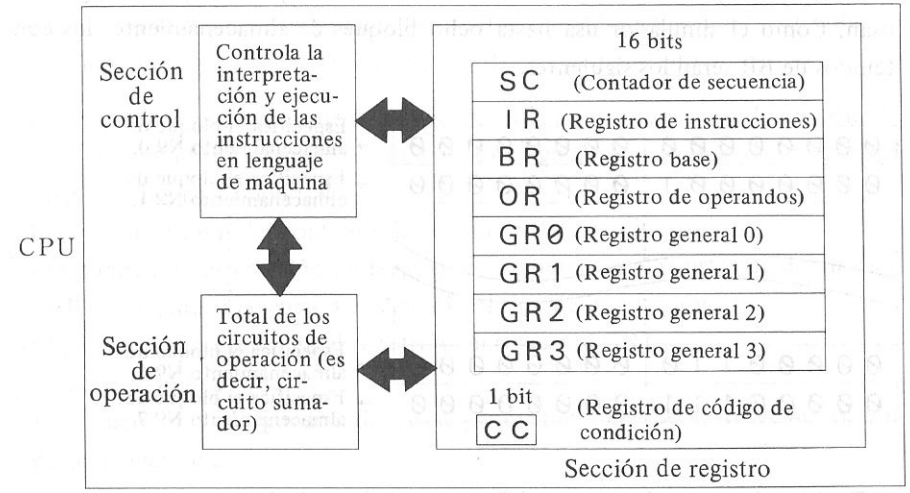
Si una palabra de la memoria principal se carga en el registro de instrucciones de la CPU, será interpretada como instrucción y no como dato.

Nota:

Dependiendo del programa, un dato simple puede ser considerado como un dato lógico, dato entero binario o una palabra de instrucción. Este tipo de licencias es una de las principales características de la programación en lenguaje de máquina en computadoras del "**sistema de programa almacenado**".

- **Configuración interna de la CPU**

La configuración de las partes principales de la CPU del simulador es la que aparece a continuación.



La CPU cuenta con un número de circuitos para almacenar datos temporalmente para las operaciones o para el procesamiento de instrucciones. Estos circuitos de almacenamiento son llamados "**registros**".

1) Contador de secuencia (Sequence Counter, símbolo SC)

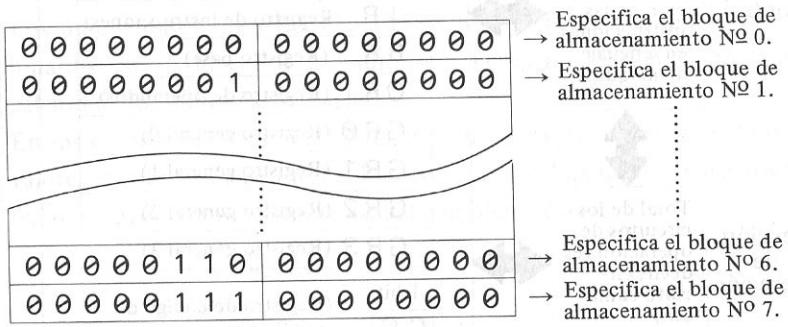
Este es un registro de 16 bits para almacenar la dirección de la memoria principal en la que se encuentra almacenada la próxima instrucción en lenguaje de máquina a ser ejecutada. La dirección de comienzo de la ejecución se inicializa en el SC antes de la ejecución de un programa.

2) Registro de instrucción (Instruction Register, símbolo IR)

Este es un registro de 16 bits para almacenar la instrucción en lenguaje de máquina que está siendo ejecutada. No se puede hacer referencia a este registro en el programa.

3) Registro base (Base Register, símbolo BR)

Este es un registro de 16 bits que especifica los 8 bits de orden superior (bloque de almacenamiento) cuando se están especificando las direcciones (16 bits) en la memoria principal. Los 8 bits de orden inferior siempre se fijan en 0 y no se usan. Como el simulador usa hasta ocho bloques de almacenamiento, los contenidos de BR serán los siguientes.



4) Registro de operando (Operand Register, símbolo OR)

Los cálculos tales como la suma y resta se llevan a cabo mediante la operación de los contenidos (operando) de las palabras en la memoria principal con los contenidos del registro general. Este OR es un registro de 16 bits para almacenar temporalmente los datos recuperados desde la memoria principal en ese momento. No se puede hacer referencia a este registro dentro de un programa.

5) Registro general (General Register, símbolo GR)

Este es un registro de 16 bits que puede ser usado como un registro índice (símbolo XR, que se explicará con posterioridad) para la modificación de dirección, y como registro de operación para las operaciones aritméticas tales como la suma y resta. Se llama registro general debido a sus múltiples usos y está a su vez compuesto de cuatro registros GR0, GR1, GR2 y GR3. Aunque los cuatro registros pueden ser usados para operaciones aritméticas, sólo GR0 puede ser usado como registro índice.

6) Registro de código de condición (Condition Code Register, símbolo CC)

Este es un registro de 1 bit que almacena los bits de código (bit Nº 0) de los resultados cuando se lleva a cabo una suma o resta. El contenido de este registro se mantiene hasta que se lleva a cabo la próxima suma o resta.

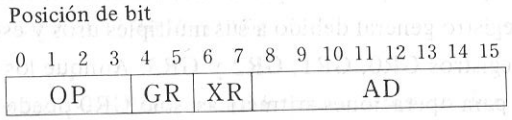
La recuperación, interpretación y ejecución de las instrucciones en lenguaje de máquina desde la memoria principal se controlan mediante la sección de control, cuyo flujo de procesamiento básico se muestra a continuación.

a) Considera el contenido de la dirección en la memoria principal especificada por el contador de secuencia (SC) como lenguaje de máquina y lo carga en el registro de instrucción (IR).
b) Aumenta el valor del contador de secuencia (SC) en uno.
c) Decodifica la instrucción en lenguaje de máquina en el registro de instrucción (IR) y prepara la ejecución de acuerdo al tipo de instrucción.
d) Ejecuta la instrucción en el registro de instrucción (IR).
e) Vuelve al paso a).

*Los pasos c) y d) pueden diferir dependiendo del tipo instrucción en lenguaje de máquina.

- **Configuración de la instrucción en lenguaje de máquina**

Cada instrucción en lenguaje de máquina (palabra de instrucción) tiene la siguiente configuración de 16 bits.



Cada sección de configuración interna de 16 bits que tiene semántica se denomina "**campo**". Como se muestra en el diagrama anterior, una palabra de instrucción se divide en cuatro campos que tienen los siguientes nombres y funciones.

1) Campo OP (Campo de código de instrucción)

Especifica un código de instrucción (tipo de instrucción). Una código de instrucción se expresa con 4 bits y este simulador está dotado con los siguientes 14 tipos de códigos de instrucción. (Ver en el Capítulo 7 para los detalles.) Cada código de instrucción es denominado por su nemotécnica.

Visualización binaria	Hexa-decimal	Instrucción	Nemotécnicas
0000	0	Halt and jump (Detención y salto)	HJ
0001	1	Jump if GR is not zero (Salto si el GR no es cero)	JNZ
0010	2	Jump on condition (Salto condicional)	JC
0011	3	Jump to subroutine (Salto a subrutina)	JSR
0100	4	Shift (Desplazamiento)	SFT
0101	5	Read (Lectura)	READ
0110	6	Write (Escritura)	WRITE
1000	8	Load address immediate (Carga de la dirección contigua)	LAI
1010	A	Add (Suma)	ADD
1011	B	Substract (Resta)	SUB
1100	C	Load (Carga)	LD
1101	D	Store (Almacenamiento)	ST
1110	E	And (Y)	AND
1111	F	Exclusive or (O exclusivo)	EOR

2) Campo GR (Campo de registro general)

Especifica el número del registro general que será el objeto de la instrucción. Sin embargo, los contenidos del campo GR serán descartados en el caso de una instrucción de "Detención y salto" y una condición de decisión para el salto se adoptará en el caso de una instrucción de "Salto condicional". (Ver en el Capítulo 7 para los detalles.)

3) Campo XR (Campo de registro índice)

Especifica el número del registro índice (GR1, GR2, GR3) para modificar una dirección. No obstante, si la dirección no está siendo modificada se especificará 0 en el campo XR. Esto también indica la dirección del desplazamiento en el caso de una instrucción de desplazamiento. (Ver en el Capítulo 7 para los detalles.)

4) Campo AD (Campo de dirección)

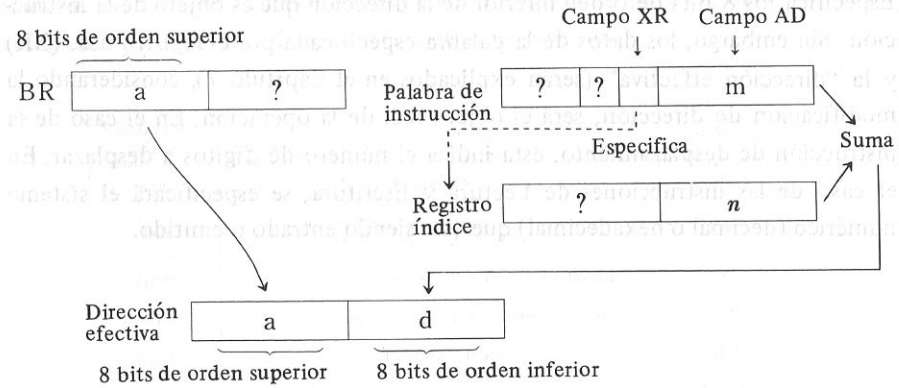
Especifica los 8 bits de orden inferior de la dirección que es objeto de la instrucción. Sin embargo, los datos de la palabra especificada por el registro base (BR) y la "dirección effectiva" (serán explicados en el Capítulo 7), considerando la modificación de dirección, será el objeto real de la operación. En el caso de la instrucción de desplazamiento, ésta indica el número de dígitos a desplazar. En el caso de las instrucciones de Lectura y Escritura, se especificará el sístema numérico (decimal o hexadecimal) que está siendo entrado o emitido.

6 Modificación de dirección y direccion efectiva

La CPU interpreta la palabra de instrucción en la memoria principal y lleva a cabo el intercambio de datos con las palabras de las direcciones especificadas según las instrucciones y también controla los saltos a la dirección especificada. En el caso de la dirección objeto en este simulador, sin embargo, la operación es ejecutada por la "**dirección efectiva**" obtenida mediante un método fijo.

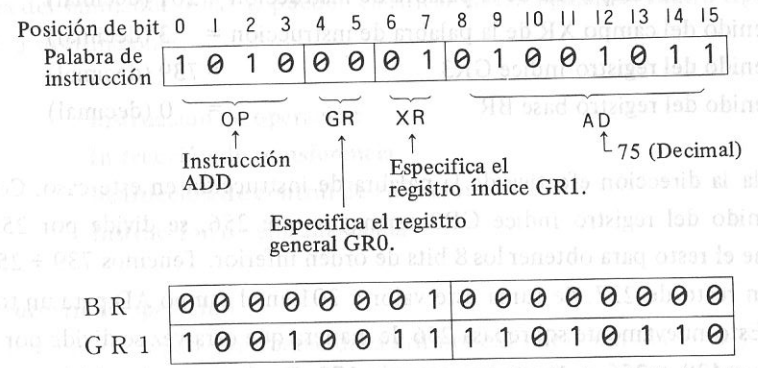
La dirección efectiva de una palabra de instrucción está compuesta de 16 bits. De éstos, los 8 bits de orden superior siempre usan los contenidos de los 8 bits de orden superior del registro base BR. Los 8 bits de orden inferior de la dirección efectiva se fijan como la suma del valor del campo AD (8 bits) de la palabra de instrucción y el valor de los 8 bits de orden inferior de los registros índice (GR1, GR2, GR3) especificados en el campo XR. En este caso, si la suma sobrepasa de 256, se usará el resto después de dividir por 256. Si el valor del campo XR es 0, el valor del campo AD serán los 8 bits de orden inferior de la dirección efectiva.

El diagrama siguiente muestra el método de cálculo de la dirección efectiva.



A continuación se presentan varios métodos para el método de cálculo de la dirección efectiva.

Ejemplo 1:



El contenido de GR1 es 37846 (decimal). Sus 8 bits de orden inferior tienen como contenido 214, lo que representa el resto después de dividir por 256.
(37846 ÷ 256 = 147 con un resto de 214)

Calcula la dirección efectiva de esta instrucción ADD cuando los contenidos de los registros involucrados son como se muestra en el diagrama. BR se transforma en el bloque de almacenamiento Nº 1 ya que su contenidos es 256 (decimal). Suma el 214 de los 8 bits de orden inferior del GR1 al 75 (decimal) en el campo AD de la palabra de instrucción.

214 + 75 = 289 = 256 + 33

Divide la suma por 256 y asigna el resto 33 (decimal) a los 8 bits de orden inferior de la dirección efectiva. Así, la dirección efectiva es la Dirección 33 en el bloque de almacenamiento Nº 1. La dirección entre todas las direcciones será 289 (256 + 33).

Ejemplo 2:

Contenido del campo AD de la palabra de instrucción = 43 (decimal)
Contenido del campo XR de la palabra de instrucción = 2 (decimal)
Contenido del registro índice GR2 = 3 (decimal)
Contenido del registro base BR = 0 (decimal)

Esto indica que la dirección efectiva de esta palabra de instrucción es 46 (= 43 + 3). Como en este caso BR es 0, se trata del bloque de almacenamiento Nº 0, y la dirección en este bloque concuerda con la dirección en las direcciones totales.

Ejemplo 3:

Contenido del campo AD de la palabra de instrucción = 201 (decimal)
Contenido del campo XR de la palabra de instrucción = 3 (decimal)
Contenido del registro índice GR3 = 739 (decimal)
Contenido del registro base BR = 0 (decimal)

Calcula la dirección efectiva de la palabra de instrucción en este caso. Como el contenido del registro índice GR3 es mayor que 256, se divide por 256 y se obtiene el resto para obtener los 8 bits de orden inferior. Tenemos 739 ÷ 256 = 2 con un resto de 227. Se suma este valor a 201 en el campo AD para un total de 428. Esto nuevamente sobrepasa 256 de manera que otra vez se divide por 256 y tenemos 428 ÷ 256 = 1 con un resto de 172. Por lo tanto, los 8 bits de orden inferior de la dirección efectiva serán 172 (decimal). Como el BR es 0 en este caso, se trata del bloque de almacenamiento Nº 0 y la dirección efectiva entre las direcciones totales será 172.

El cambiar los efectos reales de la especificación de dirección mediante el campo AD de acuerdo a los contenidos del registro índice especificado en el campo XR es denominado "**modificación de dirección**". Si se especifica un número 1, 2 ó 3 (decimal) para el campo XR, se llevará a cabo la modificación de dirección. Por otro lado, si se especifica el valor 0 para el campo XR, ésto significa que la "modificación de dirección no se llevará a cabo".

Ejemplo 4:

Contenido del campo AD de la palabra de instrucción = 38 (decimal)
Contenido del campo XR de la palabra de instrucción = 0
Contenido del registro base BR = 0
No se llevará cabo la modificación de dirección en este caso y la dirección efectiva será la dirección 38.

7 Instrucción en lenguaje de máquina del simulador

Se dispone de 14 palabras de instrucción para las instrucciones en lenguaje de máquina del simulador. Ellas se pueden clasificar en los siguientes cuatro tipos de acuerdo a sus funciones.

Instrucción de operación
Instrucción de transferencia
Instrucción de control de ejecución
Instrucción de entrada/salida

1) Instrucción de operación

Da las instrucciones para las operaciones aritméticas tales como la suma y resta, y para las operaciones lógicas tales como el producto lógico.

2) Instrucción de transferencia

Da las instrucciones para la transferencia de información entre los registros generales y la memoria principal.

3) Instrucción de control de la ejecución

Controla la secuencia de ejecución de las instrucciones en lenguaje de máquina. Normalmente la instrucción en la dirección $(n + 1)$ se ejecuta a continuación de una instrucción en la dirección n. Este flujo puede ser cambiado mediante una instrucción de control de ejecución. Esta es una instrucción en lenguaje de máquina equivalente a GOTO, GOSUB e IF ~ THEN en lenguaje BASIC.

4) Instrucción de entrada/salida (I/O)

Da las instrucciones para la entrada de valores numéricos desde el dispositivo de entrada (teclado) y para la salida de valores numéricos al dispositivo de salida (visualización de cristal líquido).

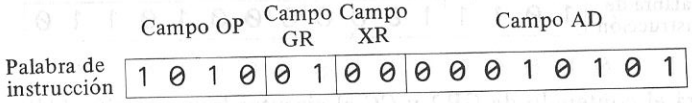
Clasificación	Nombre	Nemo-técnicas	Código de instrucción (hexadecimal)	Código de instrucción (binario)
Operación	Add (Suma)	ADD	A	1010
	Substract (Resta)	SUB	B	1011
	Shift (Desplazamiento)	SFT	4	0100
	And (Y)	AND	E	1110
	Exclusive or (O exclusivo)	EOR	F	1111
Transfe-rencia	Load (Carga)	LD	C	1100
	Store (Almacenamiento)	ST	D	1101
	Load address immediate (Carga de dirección contigua)	LAI	8	1000
Control de ejecución	Jump if GR is not zero (Salto si GR es distinto de cero)	JNZ	1	0001
	Jump on condition (Salto condicional)	JC	2	0010
	Jump to subroutine (Salto a subrutina)	JSR	3	0011
	Halt and Jump (Detención y salto)	HJ	0	0000
I/O	Read (Lectura)	READ	5	0101
	Write (Escritura)	WRITE	6	0110

Ahora explicaremos las funciones de las palabras de instrucciones respectivas.

- **Instrucción ADD (Código de instrucción binario 1010)**

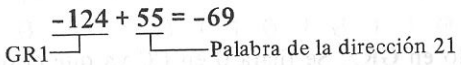
> Suma el contenido del registro general especificado por el campo GR al contenido de la palabra en memoria principal especificada por la dirección efectiva. Estos contenidos se consideran enteros binarios con código de 16 bits. Luego, esta instrucción fija el resultado en el registro general. En este caso se descarta el desbordamiento. Esta instrucción también fija el bit de código (bit Nº 0) de los resultados de la suma en el registro de código de condición (CC). El contenido de la palabra permanece inalterado.

Ejemplo:
Asumamos que tenemos la siguiente instrucción ADD.
(En este caso BR = 0)



¿Cuál será el contenido de GR1 y CC si se ejecuta la instrucción ADD cuando el contenido de GR1 es -124 y el contenido de la palabra en la dirección 21 es +55?

Respuesta:
El campo GR de la palabra de instrucción será 1, el campo XR será 0 y no habrá modificación de dirección. Así, el valor 21 del campo AD se transforma en la dirección efectiva.



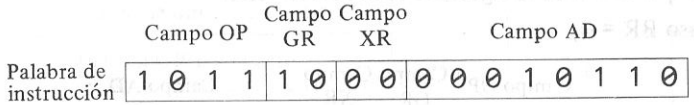
El resultado -69 se fija en GR1. Como este resultado es negativo, el bit de código es 1 y este mismo valor se fija en CC.

- **Instrucción SUB (Código de instrucción binario 1011)**

Resta el contenido de la palabra en la memoria principal especificada por la dirección efectiva del contenido del registro general especificado por el campo GR. Estos contenidos son considerados enteros binarios con códigos de 16 bits. Luego, esta instrucción fija el resultado en el registro general. En este caso se descarta el desbordamiento. El bit de código (bit Nº 0) del resultado de la resta se fija en registro de código condicional (CC). El contenido de la palabra permanece inalterado.

Ejemplo:

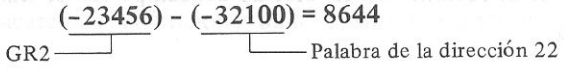
Asumamos que tenemos la siguiente instrucción SUB. (BR = 0)



¿Cuál será el contenido de GR2 y CC al ejecutar la instrucción SUB cuando el contenido de GR2 es -23456 y el contenido de la palabra en la dirección 22 es -32100?

Respuesta:

El campo GR de la palabra de instrucción es 2, el campo XR es 0 y no hay modificación de dirección. El valor 22 del campo AD es la dirección efectiva.



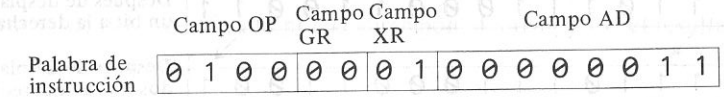
El resultado 8644 será fijado en GR2. Se fijará 0 en CC ya que el resultado es positivo y el bit de código es 0.

- **Instrucción SFT (Código de instrucción binario 0100)**

Desplaza los contenidos (dígitos) de un registro general especificado por el campo GR, excluyendo el bit de código (bit Nº 0), a la derecha o izquierda en el número de bits especificados por el campo AD. La dirección de desplazamiento se especifica mediante el campo XR y será hacia la derecha cuando el valor del campo XR sea 0 y a la izquierda cuando este valor sea 1. Después de un desplazamiento a la izquierda se entrará un 0 en la posición de bit vacía, y un bit igual al código de bit se entrará después de un desplazamiento a la derecha. No hay dirección efectiva.

Ejemplo:

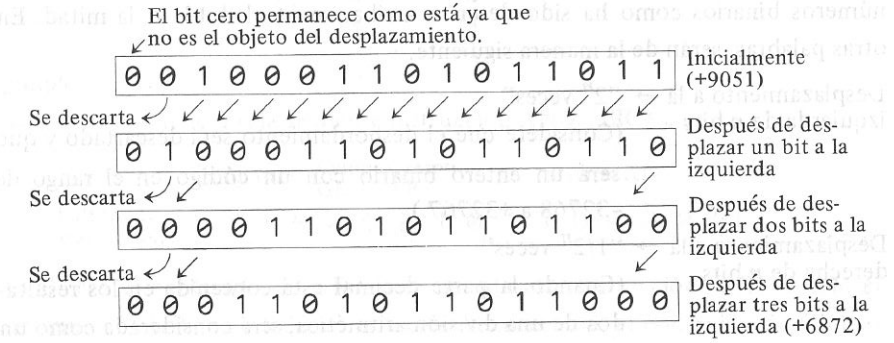
Asumamos que tenemos la siguiente instrucción SFT.



¿Cuál será el contenido de GR0 si se ejecuta la instrucción SFT cuando el contenido de GR0 es +9051?

Respuesta:

Como el valor del campo XR es 1 y el del campo AD es 3, se efectuará un desplazamiento de 3 bits hacia la izquierda.



Por lo tanto, el contenido de GR0 será +6872 como resultado del desplazamiento.

Ejemplo:

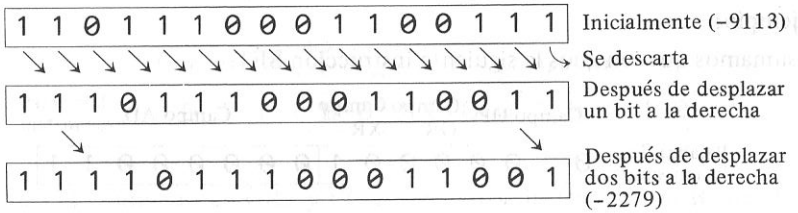
¿Cuál será el contenido de GR3 después de ejecutar la siguiente instrucción SFT si su contenido inicial es -9113?

	Campo OP	Campo GR	Campo XR	Campo AD
Palabra de instrucción	0 1 0 0	1 1	0 0	0 0 0 0 0 0 1 0

Respuesta:

El campo GR de la palabra de instrucción es 3 y el objeto del desplazamiento son los contenidos de GR3. Como el valor del campo XR es 0 y el del campo AD es 2, se efectuará un desplazamiento de dos bits a la derecha.

En las posiciones vacías se entrará el contenido del bit de código (bit Nº 0).



Por lo tanto, como resultado del desplazamiento el contenido de GR3 será -2279.

Nota:

Como en el caso del desplazamiento de dígitos a la derecha e izquierda en los números decimales, podemos considerar que los desplazamientos de bits en los números binarios como ha sido descrito arriba, serán el doble o la mitad. En otras palabras, serán de la manera siguiente.

Desplazamiento a la izquierda de *n* bits → "2^n veces"
(Considere que el desbordamiento será descartado y que será un entero binario con un código en el rango de -32768 a +32767.)

Desplazamiento a la derecha de *n* bits → "$1/2^n$ veces"
(Cuando la parte decimal está contenida en los resultados de una división aritmética, será considerada como un número codificado y la respuesta será el entero máximo que no exceda el resultado.)

Como el objeto del desplazamiento son los bits Nº 1 al 15, que no incluyen el bit de código, el desplazamiento se efectuará con el valor numérico como un "entero binario con código". Como +9051 se desplaza tres bits a la derecha en el primer ejemplo anterior, el resultado será 72408; $(+9051) \times 2^3 = +9051 \times 8$. Con el rango de un entero binario de 16 bits con un código en el resultado será +6872 (72408 − 65536).

En el segundo ejemplo, como -9113 se desplaza dos bits a la derecha, el resultado será -2278,25; $(-9113) \times 1/2^2 = (-9113) \times 1/4$. Como el entero máximo que no excede este valor es -2279, éste será la respuesta.

- **Instrucción AND (Código de instrucción binario 1110)**

> Obtiene el producto lógico por bit del contenido del registro general especificado por el campo GR y el contenido de la palabra en la memoria principal especificada por la dirección efectiva, y fija el resultado en el registro general.

El producto lógico de p y q (aquí se expresará como $p \wedge q$) es la siguiente operación lógica.

p	q	$p \wedge q$
0	0	0
0	1	0
1	0	0
1	1	1

Ejemplo:

Asumamos que tenemos la siguiente instrucción AND. (BR = 0)

	Campo OP	Campo GR	Campo XR	Campo AD
Palabra de instrucción	1 1 1 0	0 0	0 1	0 1 1 0 0 0 0 0

¿Cuál será el contenido de GR0 si se ejecuta esta instrucción AND cuando el contenido de GR0 es ABCD en notación hexadecimal, el contenido de GR1 es -2 y el contenido de la palabra en la dirección 94 es 00FF en notación hexadecimal?

Respuesta:
Como el valor del campo XR es 1, la dirección efectiva será determinada con la modificación de dirección mediante el registro índice GR1.

Dirección efectiva **96 + (-2) = 94** (El registro base BR es 0.)
Valor del campo AD — Valor de GR1

Luego se obtiene el producto lógico por bit del registro general GR0 y la palabra en la dirección 94 de la memoria principal.

GR0 (ABCD en hexadecimal)	1	0	1	0	1	0	1	1	1	1	0	0	1	1	0	1
Dirección 94 (00FF en hexadecimal)	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
↓ Producto lógico																
	0	0	0	0	0	0	0	0	1	1	0	0	1	1	0	1

Por lo tanto, el contenido de GR0 será 00CD en notación hexadecimal.

Mediante la preparación de una palabra en la memoria principal con la instrucción AND y obteniendo el producto lógico de este dato y el registro general de esta manera, será posible entrar en el registro general sólo las posiciones de bit en que está escrito el dato 1 y enmascarar las otras posiciones de bit. Esto se denomina una "operación de enmascaramiento" y los datos usados "datos de máscara". En el ejemplo anterior, los ocho bits de orden superior de GR0 están enmascarados con el dato de máscara 00FF y sólo se entran los 8 bits de orden inferior.

- **Instrucción EOR (Código de instrucción binario 1111)**

> Obtiene el O exclusivo (la suma lógica exclusiva) por bit del contenido del registro general especificado por el campo GR y el contenido de la palabra en la memoria principal especificada por la dirección efectiva, y fija el resultado en el registro general.

El O exclusivo de p y q (aquí se expresará como p ⊕ q) es la siguiente operación lógica.

p	q	p ⊕ q
0	0	0
0	1	1
1	0	1
1	1	0

Ejemplo:
Asumamos que tenemos la siguiente instrucción EOR (BR = 0)

	Campo OP				Campo GR		Campo XR		Campo AD							
Palabra de instrucción	1	1	1	1	0	0	0	0	0	1	1	0	0	0	0	0

¿Cuál será el contenido de GR0 si se ejecuta la instrucción EOR cuando el contenido de GR0 es A50F en notación hexadecimal y el contenido en la dirección 96 es FFFF en notación hexadecimal?

Respuesta:
El valor del campo XR es 0 y no hay modificación de dirección. Por lo tanto la dirección efectiva será 96. Luego obtenemos el O exclusivo por bit de GR0 y la palabra en la dirección 96.

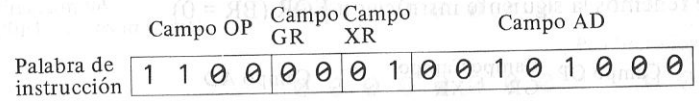
GR0 (A50F en hexadecimal)	1	0	1	0	0	1	0	1	0	0	0	0	1	1	1	1
Dirección 96 (FFFF en hexadecimal)	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
↓ O exclusivo																
	0	1	0	1	1	0	1	0	1	1	1	1	0	0	0	0

Por lo tanto el contenido de GR0 será 5AF0 en notación hexadecimal.
Los 0s y 1s en los bits del GR0 son invertidos. (Para detalles, refiérase a la 11-3 en el Capítulo 11.)

- **Instrucción LD (Código de instrucción binario 1100)**

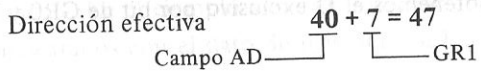
Transfiere los contenidos de la palabra en memoria principal especificada mediante la dirección efectiva al registro general especificado por el campo GR.
El contenido de la palabra permanece inalterado.

Ejemplo:
Asumamos que tenemos la siguiente instrucción LD. (BR = 0)



¿Cuál será el contenido de GR0 si esta instrucción LD se ejecuta cuando el contenido de GR0 es 29, el contenido de GR1 es 7 y el contenido de la palabra de la dirección 47 es 112?

Respuesta:
Como el valor del campo XR es 1, la dirección efectiva será determinada con la modificación de dirección mediante el registro índice GR1.

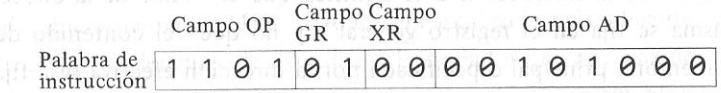


Por consiguiente, el contenido de la palabra en la dirección 47, 112, será transferido a GR0 y el contenido de GR0 cambiará a 112. El contenido de la dirección 47 continúa siendo 112.

- **Instrucción ST (Código de instrucción binario 1101)**

Almacena los contenidos del registro general especificado por el campo GR en la palabra de la memoria principal especificada mediante la dirección efectiva. El contenido del registro general permanece inalterado.

Ejemplo:
Asumamos que tenemos la siguiente instrucción ST. (BR = 0)



¿Cuál será el contenido en la dirección 40 si la instrucción ST se ejecuta cuando el contenido de GR1 es 25 y el contenido de la palabra en la dirección 40 es 516?

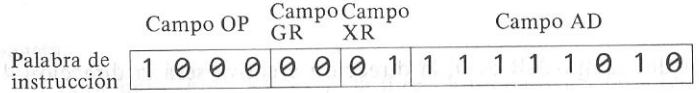
Respuesta:
Como el valor del campo XR es 0, la dirección efectiva será la dirección 40 sin modificación. El contenido 25 de GR1 será almacenado en la palabra de la dirección 40, haciendo así el contenido de esta palabra 25. El contenido de GR1 continúa siendo 25.

- **Instrucción LAI (Código de instrucción binario 1000)**

> Los 8 bits de orden inferior de la dirección efectiva se fijan a los 8 bits de orden inferior del (bits del 8 al 15) del registro general especificado por el campo GR, y al mismo tiempo los 8 bits de orden superior (bits del 0 al 7) se fijan a 0. Esta instrucción se usa para hacer el contenido del registro general especificado un valor numérico de 0 a 255.

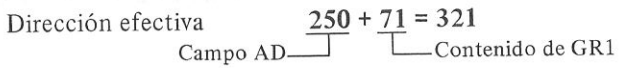
*El "inmediato" de la instrucción LAI significa que el "valor de la dirección efectiva misma se fija en el registro general", y no que "el contenido de la palabra en memoria principal especificada por la dirección efectiva será fijado en el registro general".

Ejemplo:
Asumamos que tenemos la siguiente instrucción LAI. (BR = 0)



¿Cuál será el contenido de GR0 si esta instrucción LAI es ejecutada cuando el contenido de GR0 es 84 y el contenido de GR1 es 71?

Respuesta:
Como el valor del campo XR es 1, la dirección efectiva será determinada con modificación de dirección por el registro índice GR1.



Los 8 bits de orden inferior de la dirección efectiva.

1 con resto de 65; 321 ÷ 256.

Por lo tanto, el valor 65 en los 8 bits de orden inferior de la dirección efectiva será fijado en GR0.

- **Instrucción JNZ (Código de instrucción binario 0001)**

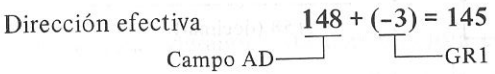
> Los 8 bits de orden inferior de la dirección efectiva serán almacenados en el contador de secuencia (SC) y la ejecución salta a la dirección efectiva si el contenido del registro general especificado por el campo GR no es 0. Si éste es 0, se ejecutará la instrucción de la próxima dirección.

*Como los 8 bits de orden inferior de la dirección efectiva están almacenados en el contador de secuencia (SC), el salto sólo es posible dentro del mismo bloque de almacenamiento. JSR, la que será explicada más adelante, es la única instrucción en lenguaje de máquina del simulador que es capaz de cambiar el valor del registro base (BR) y manipular direcciones en un bloque de almacenamiento diferente.

Ejemplo:
Asumamos que tenemos la siguiente instrucción. (BR = 0)

¿Cuál será la dirección de la próxima instrucción a ser ejecutada si se ejecuta esta instrucción JNZ cuando el contenido de GR0 es 125 y el contenido de GR1 es -3.

Respuesta:
Como el valor de campo XR es 1, la dirección efectiva será determinada con modificación de dirección por el registro índice GR1.



Como el contenido de GR0 no es 0, la ejecución saltará a la dirección 145 si se ejecuta esta instrucción.

- **Instrucción JC (Código de instrucción binario 0010)**

Se efectuarán las siguientes operaciones según los contenidos especificados por los campos GR (bits Nº 4 y 5).

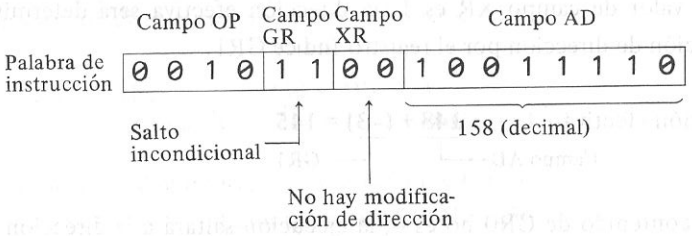
Bit Nº 4	Bit Nº 5	Operación
0	0	Simplemente avanza a la próxima instrucción.
0	1	Salta a la dirección efectiva cuando el contenido del registro de código de condición (CC) es 1.
1	0	Salta a la dirección efectiva cuando el contenido del registro de código condición (CC) es 0.
1	1	Salta incondicionalmente a la dirección efectiva.

*Como los 8 bits de orden inferior de la dirección efectiva serán almacenados en el contador de secuencia (SC) cuando se ejecute el salto, el salto sólo es posible a direcciones dentro del mismo bloque de almacenamiento. La instrucción JSR, que será explicada con posterioridad, se debe usar para saltar a un bloque de almacenamiento diferente.

*Como el registro de código de condición (CC) sólo será fijado para sumas y restas, se debe tener cuidado al hacer saltos con un valor del CC.

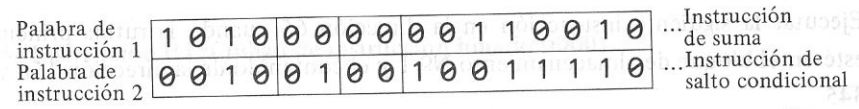
Ejemplo:

Use la siguiente instrucción para hacer un salto incondicional a la dirección 158. (BR = 0)



Ejemplo:

Verificar si el contenido de GR0 es negativo y haga un salto a 158 si lo es. Si se determina que la constante 0 está almacenada en la dirección 50 de la memoria principal, este salto se puede llevar a cabo ejecutando las siguientes instrucciones consecutivamente.



La constante 0 de la dirección 50 se suma a GR0 mediante la instrucción de suma de la palabra de instrucción 1. Aunque el contenido de GR0 permanece inalterado, el registro de código de condición (CC) será fijado por el bit de código de GR0. Esto permitirá el salto a la dirección 158 si el contenido de CC en la palabra de instrucción 2 es 1 (es decir, si GR0 es negativo).

- **Instrucción JSR (Código de instrucción binario 0011)**

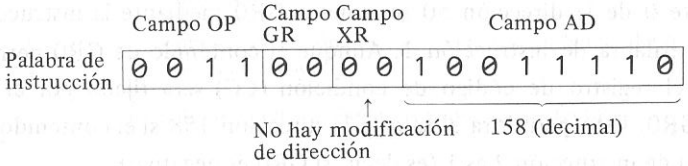
Almacena el contenido de la palabra en memoria principal especificada mediante la dirección efectiva en el contador de secuencia (SC) y el registro base (BR) después de fijar el "contenido + 1" del contador de secuencia (SC) en el registro general especificado por el campo GR. Los 8 bits de orden inferior del registro base (BR) se fijan a 0. Esta instrucción permite el salto a cualquier dirección opcional en la memoria principal y es la única instrucción que puede cambiar el contenido del registro base (BR). Saltos a una subrutina de un bloque de almacenamiento diferente y el retorno de dichas subrutinas se llevan a cabo con esta instrucción.

*Cuando se está operando esta computadora en lenguaje BASIC, el programa puede ser dividido en varias secciones, y las respectivas secciones pueden ser almacenadas en las áreas de programas P0 a P9. Similarmente, al usar esta computadora como un simulador, los programas que requieren una serie de procesos pueden ser almacenados en cada uno de los dos bloques de almacenamiento. El programa que juega el papel principal es llamado rutina principal y el otro subrutina. El salto a un bloque de almacenamiento diferente equivale

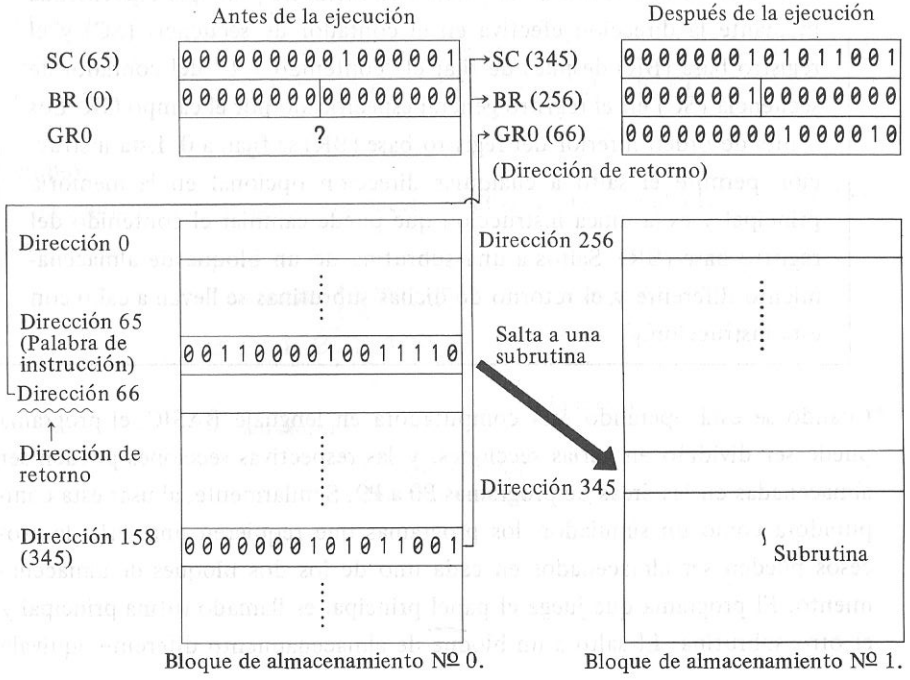
a GOTO #*n* y GOSUB #*n* en BASIC. Sin embargo, en BASIC el retorno a la subrutina principal se efectúa automáticamente mediante RETURN, mientras que en el caso del simulador se debe usar la instrucción JSR para retornar a la rutina principal.

Ejemplo:

Ejecutar la siguiente instrucción en la dirección 65 cuando la rutina principal esté en el bloque de almacenamiento Nº 1 y el contenido de la dirección 158 sea 345.



Abajo se muestra el salto desde la dirección 345 en el bloque de almacenamiento Nº 1 a una subrutina. Para retornar desde la subrutina, se debe usar el valor de GR0, ya que la dirección de retorno (dirección 66) será almacenada en GR0. (Ver en el Capítulo 11-6 para los detalles.)



Nota:

Como la memoria principal del simulador está compuesta de 8 bloques de almacenamiento (2048 palabras), el simulador visualizará un error "fuera del rango de direcciones" si la dirección efectiva especificada por la instrucción JSR excede de 2047.

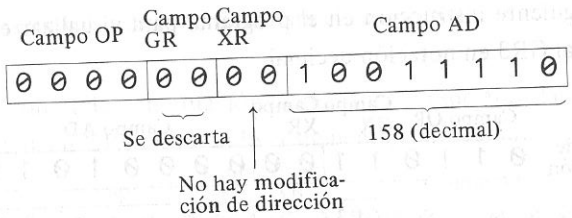
- **Instrucción HJ (Código de instrucción binario 0000)**

> Almacena la dirección efectiva en el contador de secuencia (SC) y detiene la ejecución del programa en lenguaje de máquina. La ejecución de la instrucción comenzará de nuevo desde la dirección indicada por el contador de secuencia (SC) si se da la instrucción de recomienzo. El contenido del campo GR será descartado.

* Si se ejecuta la instrucción HJ, el simulador detiene la ejecución del programa y retorna a la "pantalla de menú del simulador". La ejecución recomenzará con "Go" en la "pantalla del menú del simulador". Aparte de la instrucción HJ, la ejecución del programa puede ser detenida presionando la tecla BRK durante la ejecución del programa en lenguaje de máquina y se visualiza la "pantalla de menú del simulador". Ver en el Capítulo 10, "Operación fundamental del simulador" para los detalles en relación a la ejecución de programas en lenguaje de máquina usando el simulador.

Ejemplo:

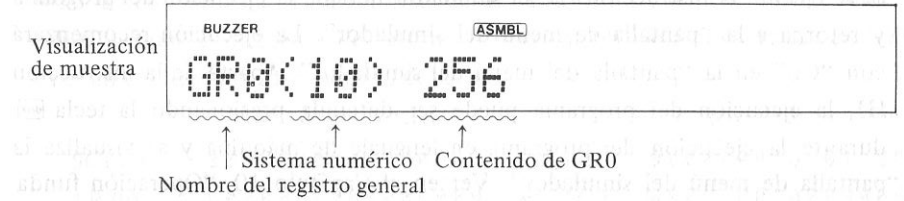
Los contenidos del contador de secuencia (SC) serán fijados en 158 y la ejecución se detendrá si se ejecuta la siguiente instrucción HJ. Si se da una instrucción de recomienzo, la ejecución recomenzará desde la dirección 158.



- **Instrucción WRITE (Código de instrucción binario 0110)**

Visualiza el contenido del registro general especificado por el campo GR con un sistema numérico especificado por el campo AD. El sistema numérico especificado será decimal o hexadecimal. Siempre se especificará un 0 para el campo XR. No hay dirección efectiva y el contenido del registro general permanece inalterado aun si se ejecuta esta instrucción.

*Cuando se interpreta una instrucción WRITE, el simulador visualiza el nombre del registro general especificado mediante el campo GR y el sistema numérico especificado por el campo AD, como así también el contenido del registro general con su sistema numérico. Luego el simulador estará en el estado de espera de entrada de teclas hasta que se presione la tecla [EXE].

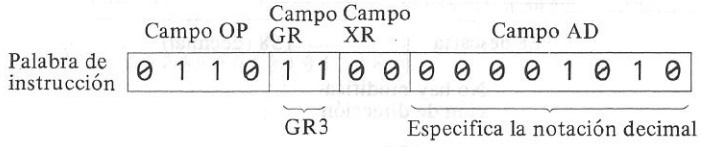


*Si se presiona la tecla [EXE], el simulador suma 1 al contenido del contador de secuencia (SC) y comienza la lectura de la próxima instrucción.

*Si se escribe un valor distinto de 10 (decimal) ò 16 (decimal) en el campo AD, o si se especifica un valor distinto de 0 para el campo XR, se producirá un error (Error de operando).

Ejemplo:

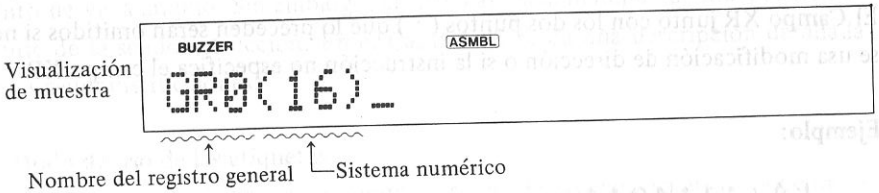
Ejecutar la siguiente instrucción en el programa para visualizar el contenido del registro general GR3 en notación decimal.



- **Instrucción READ (Código de instrucción binario 0101)**

Almacena una entrada de valor numérico desde el teclado en el registro general especificado por el campo GR. Este valor numérico puede ser decimal o hexadecimal, y el sistema numérico será especificado por el campo AD. Para el campo XR siempre estará especificado 0 y no habrá dirección efectiva.

*Cuando se interpreta una instrucción READ, el simulador visualiza el nombre de registro general especificado por el campo GR y el sistema numérico especificado por el campo AD. Luego el simulador permanecerá en el estado de espera de entrada de teclas hasta que se presione la tecla [EXE].



*Entre los valores numéricos de acuerdo al sistema numérico visualizado y presione la tecla [EXE]. Cuando se presiona la tecla [EXE], el simulador interpreta el valor de entrada de acuerdo al sistema numérico del campo AD y, si es correcto, almacena este valor en el registro general.

*Si el sistema numérico no es correcto, o si el rango de entrada (0000 ~ FFFF en notación hexadecimal) está excedido, en la etapa de interpretación del valor de entrada el simulador retornará al estado de espera de entrada y solicitará una nueva entrada.

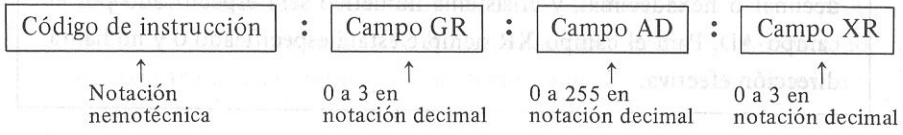
*Si en el campo AD se escribe un valor distinto de 10 (decimal) ó 16 (decimal), se producirá un error de operando. También se producirá un error de operando si se especifica un valor distinto de 0 para el campo XR.

Ejemplo:

Entre un valor numérico en notación hexadecimal desde el teclado y almacénelo en el registro general GR1 ejecutando la siguiente instrucción.

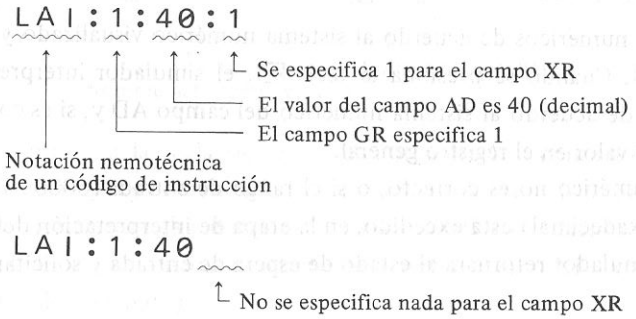
8 Reglas de notación nemotécnica

En lenguaje de ensamble, los códigos de instrucción de cada instrucción en lenguaje de máquina se escriben con nemotécnicas. Los contenidos del campo GR, campo XR y campo AD se escriben de la manera siguiente.



Nótese que lo que se muestra arriba difiere de la configuración interna de una palabra de instrucción real, la información del campo XR viene al último.
En el lenguaje de ensamble del simulador se usan dos puntos (:) para separar las informaciones de los campos.
El Campo XR junto con los dos puntos (:) que lo preceden serán omitidos si no se usa modificación de dirección o si la instrucción no especifica el campo XR.

Ejemplo:



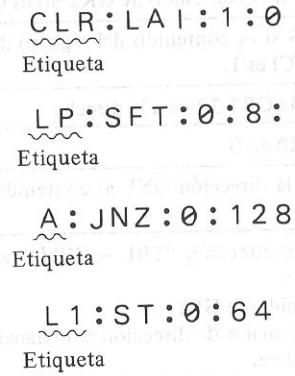
- **Seudoinstrucción y etiqueta**

Un programa escrito en lenguaje de ensamble se convierte mediante el ensamblador en un programa en lenguaje de máquina que puede ser ejecutado por la CPU. Además de las instrucciones que corresponden con las instrucciones de máquina, hay algunas instrucciones en lenguaje de ensamble, éstas no se convierten a lenguaje de máquina, sino que están disponibles sólo para el ensamblador. Estas instrucciones son denominadas "seudoinstrucciones" o "comandos de instrucción del ensamblador".
En lenguaje de ensamble también se pueden usar direcciones y constantes en un programa agregándoles una "etiqueta". Como esta etiqueta será convertida a una dirección o constante en lenguaje de máquina por el ensamblador durante el ensamble, ésta también puede ser considerada una seudoinstrucción desde un punto de vista amplio. Sin embargo, en este caso será manipulada independientemente de la seudoinstrucción. En el Capítulo 9 se da una descripción detallada de las seudoinstrucciones.

- **Modo de uso de las etiquetas**

En un programa en lenguaje de ensamble se puede agregar una etiqueta de hasta tres caracteres al principio de un código de instrucción. Para la etiqueta se usan cadenas de caracteres compuestas de caracteres numéricos y caracteres alfabéticos en mayúscula. Sin embargo, el primer carácter debe ser alfabético en mayúscula. La etiqueta y el código de instrucción se separan con dos puntos.

Ejemplo:



En este simulador se asume que cada línea del programa comienza desde la izquierda con una etiqueta, cuando se está ensamblando un programa escrito en lenguaje de ensamble. Por lo tanto será necesario escribir dos puntos (:) al comienzo de los códigos de instrucción incluso cuando no se usa etiqueta.

Ejemplo:

```
:LAI:0:0
```

↳ Los dos puntos (:) no se pueden omitir aunque no se use etiqueta.

```
LAI:0:0
```

↳ Esto será considerado como una etiqueta durante el ensamble.

Si se agrega una cierta etiqueta (exceptúando las seudoinstrucciones que serán explicadas en el Capítulo 9), ella puede ser manipulada exactamente igual que las direcciones en la memoria principal cuando se almacena una instrucción. Una etiqueta definida puede ser usada para especificar el campo AD en una instrucción.

- **Ejemplos de notación y significados de las instrucciones**

Notaciones nemotécnicas	Significado
:HJ:0:BGN	Fija la dirección BGN en el contador de secuencia (SC) y detiene la ejecución. En este simulador se da la "pantalla de menú del simulador". (Ver en el Capítulo 10-3.)
:JNZ:2:J1	Salta a la dirección J1 si el contenido de GR2 no es 0.
:JC:1:NG	Salta a la dirección NG si es contenido del registro de código de condición (CC) es 1.
:SFT:1:3:0	Desplaza el contenido de GR1 3 bits a la direcha.
:LAI:0:0	Fija el contenido de GR0 en 0.
:ADD:1:ONE	Suma el contenido de la dirección ONE al contenido de GR1.
:ADD:2:TBL:1	Suma el contenido de la dirección "TBL + (GR1)" al contenido de GR2. * (GR1) indica el contenido de GR1. * Se lleva a cabo modificación de dirección para hacer de GR1 un registro índice.

Notaciones nemotécnicas	Significado
:LD:0:WK	Almacena el contenido de la palabra en la dirección WK en GR0.
:ST:1:SAV	Almacena el contenido de GR1 en la dirección SAV.
:AND:2:MSK	Obtiene el producto lógico del contenido de GR2 y el contenido de la palabra en la dirección MSK y lo almacena en GR2.
:EOR:3:AL1	Obtiene el O exclusivo del contenido de GR3 y el contenido de la palabra en la dirección AL1 y lo almacena en GR3.
:READ:0:10	Entra un valor en notación decimal desde el teclado y lo almacena en GR0.
:WRITE:1:16	Visualiza el contenido de GR1 en notación hexadecimal de 4 dígitos.

9 Seudoinstrucciones

Para el lenguaje de ensamble de este simulador hay cinco tipos de seudoinstrucciones además de los 14 tipos de lenguajes de máquina. Los códigos de instrucción se indican mediante las siguientes nemotécnicas.

START
END
RESV
CONST
ADCON

Las seudoinstrucciones sólo dan instrucciones al ensamblador y no son convertidas a lenguaje de máquina.

- **Seudoinstrucción START** (Comienzo de programa/subprograma)

Formato: Etiqueta : START : *n*

Función:
Se debe escribir siempre al comienzo de un programa en lenguaje de ensamble, pero la etiqueta puede ser omitida. "*n*" es un número decimal que especifica la dirección de comienzo del almacenamiento cuando se almacena un programa en la memoria principal usando el lenguaje de máquina que ha sido ensamblado.
La etiqueta de la seudoinstrucción START se usa como una entrada desde un programa diferente cuando se están usando programas múltiples, junto con indicar la palabra al comienzo del programa que comienza con esta instrucción.
La referenciación es posible desde un programa diferente cuando se usan programas múltiples, junto con indicar la palabra al comienzo del programa que comienza con esta instrucción. La referenciación también es posible desde un programa diferente escribiendo la etiqueta sobre el operando (objeto) de la seudoinstrucción ADCON.

Ejemplo:

: START : 32	Especifica que el almacenamiento del programa comienza en la dirección 32.
BGN : LD : 1 : M	Por lo tanto, la etiqueta BGN en una instrucción próxima significa 32.

- **Seudoinstrucción END** (Fin de programa/subprograma)

Formato: : END : *n*

Función:
Se debe escribir siempre al final de un programa en lenguaje de ensamble. Nunca use una etiqueta antes de esta seudoinstrucción END ya que ésto producirá un error durante el ensamble. "*n*" es un número decimal o un nombre de etiqueta que especifica una dirección para el comienzo de la ejecución del programa. Puede ser omitido junto con los dos puntos (:) que le preceden.

Ejemplo 1:

: END : BGN	Esta indica que el programa termina aquí y que la dirección para el comienzo de la ejecución del programa es BGN.

Ejemplo 2:

: END	La dirección para el comienzo de la ejecución no necesita ser especificada. La seudoinstrucción END se escribe de esta manera al final de una subrutina.

- **Seudoinstrucción RESV** (Area reservada)

Formato: Etiqueta : RESV : *n*

Función:
Se escribe un número decimal para *n*. Esta seudoinstrucción se usa para reservar área en la memoria principal para una sucesión de *n* palabras. Sin embargo, el almacenamiento de programas no cambiará los contenidos del área reservada. La etiqueta indica la dirección de la primera palabra del área reservada. Esta puede ser omitida.

Ejemplo:

```
TBL:RESV:12 ·····
```

Reserva área en la memoria principal para 12 palabras sucesivas. La dirección de esta área se determina mediante la posición de la instrucción escrita en el programa. La dirección de la primera palabra en el área puede ser referenciada mediante el nombre de etiqueta TBL.

- **Seudoinstrucción CONST** (Define constantes)

Formato: Etiqueta `:CONST:`*h*

Función:

Se escribe un número hexadecimal de 4 dígitos para *h*. Se reserva una palabra en la memoria principal y el número hexadecimal escrito para *h* se almacena en la memoria principal como una constante.

La etiqueta indica la dirección de la palabra en la que se almacena la constante *h*. Esta puede ser omitida.

Ejemplo:

```
AL1:CONST:FFFF···
```

Reserva una palabra en la memoria principal y fija la constante hexadecimal FFFF en esta palabra. La dirección de esta palabra puede ser referenciada con el nombre de etiqueta AL1.

- **Seudoinstrucción ADCON** (Define dirección de constante)

Formato: Etiqueta `:ADCON:`*n*

Función:

Se escribe un nombre de etiqueta o un número decimal para *n*. Se reserva una palabra en la memoria principal y el valor de la dirección indicada por el nombre de la etiqueta se convierte en la dirección de constante. Si *n* es el nombre de etiqueta y este nombre de etiqueta está definido en el mismo programa, el valor de la dirección indicada será la constante de dirección. Si *n* es el nombre de etiqueta y este nombre de etiqueta no está definido en el mismo programa, la constante de dirección será determinada mediante el uso de la etiqueta de la seudoinstrucción START de un programa diferente. La etiqueta antes de la seudoinstrucción ADCON indica la dirección de una palabra reservada para almacenar la constante de dirección.

Ejemplo 1:

```
ADR:ADCON:128···
```

El número 128 será fijado en la dirección ADR como una constante de dirección.

Ejemplo 2:

```
     :START:32
        ⋮
SBR:ADCON:MUL····
        ⋮
     :END   :BGN
MUL:START:300
        ⋮
     :END
```

El nombre de etiqueta MUL representa la dirección 300 de un programa diferente y se almacena en la dirección SBR como una constante de dirección.

Como en la dirección SBR se almacenará la dirección de un programa diferente, será posible almacenar la dirección de retorno en GR0 y saltar a la subrutina que comienza desde la dirección MUL, ejecutando la instrucción de salto a subrutina ":JSR:0:SBR".

10 Operación fundamental del simulador

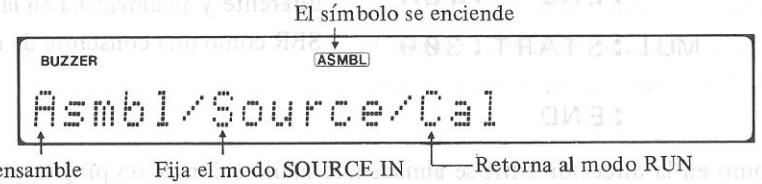
Ahora usaremos este simulador y trataremos de dominar la "operación fundamental" de la entrada de programas escritos en lenguaje de ensamble y ejecutar el programa en lenguaje de máquina después del ensamble.
Como aquí se entregan explicaciones detalladas para un ejemplo de programa, ejecute el programa y confirme cada procedimiento.

10-1 Creación de un programa fuente

Un programa escrito en lenguaje de ensamble es llamado programa fuente. El ensamblador es el que convierte el programa fuente a un programa en lenguaje de máquina que la CPU puede ejecutar. Este es llamado "**programa objeto**" o simplemente "objeto" en relación al programa fuente.
Primero crearemos un programa fuente usando este simulador. El simulador esta diseñado de manera que el programa fuente escrito en lenguaje de ensamble
Ahora prepararemos un programa fuente usando el simulador.
Esta computadora está diseñada de manera que el programa fuente escrito en lenguaje de ensamble se almacena en el "área fuente". Por lo tanto la escritura del programa fuente se efectúa presionando en primer lugar la tecla [Asmbl] y visualizando el menú de comienzo de la simulación. Luego se presiona la tecla [S] y se fija el modo SOURCE IN. Como no se desea que otros datos estén en el "Area fuente", ejecute inicialmente el comando NEW✳en el modo WRT ([MODE][1]).

- **Visualización del menú de ensamble**

Si se presiona la tecla [Asmbl] en el modo RUN o en el modo WRT, la visualización aparecerá como se muestra abajo.



Llamaremos a esta visualización una "pantalla del menú de ensamble". Con esta visualización son posibles las siguientes operaciones de teclas.

Operación	Función
[A]	Ensambla en programa fuente escrito en lenguaje de ensamble.
[S]	Fija el modo SOURCE IN (para entrar y corregir los programas fuente).
[C]	Retorna al modo RUN.
[BRK]	Retorna al modo RUN.

Como un ejemplo, asumiremos que el programa a ser entrado sirve para calcular el mayor divisor común de dos valores (aquí usaremos los valores decimales 16 y 24).

Lista del programa fuente

Etiqueta	Código de instrucción	Operando	Explicación
	:START	:32	(Especifica la dirección 32 como la dirección de almacenamiento.)
L1	:LD	:0:M	(Almacena GR0 en M.)
	:SUB	:0:N	(Fija M-N en GR0.)
	:JNZ	:0:L2	(Salta a L2 si M-N ≠ 0.)
	:HJ	:0:L1	(Detiene el programa y fija el SC en L1.)
L2	:JC	:2:L3	(Salta a L3 si M-N > 0.)
	:LD	:0:N	(Almacena N en GR0.)
	:SUB	:0:M	(Fija N-M en GR0.)
	:ST	:0:N	(Almacena el contenido de GR0 en N.)
	:JC	:3:L1	(Lleve a cabo un salto incondicional a L1.)
L3	:ST	:0:M	(Almacena el contenido de GR0 en M.)
	:JC	:3:L1	(Lleva a cabo un salto incondicional a L1.)
M	:CONST	:0018	(Fija 24 como un hexadecimal en la dirección M.)
N	:CONST	:0010	(Fija 16 como un hexadecimal en la dirección N.)
	:END	:L1	(Fin del programa. La dirección de ejecución es L1.)

- **Escribiendo el programa fuente**

Cuando entre este programa no olvide entrar la marca dos puntos (:) de separación. Ponga el ":" al comienzo aun si no hay etiqueta. Tampoco olvide presionar la tecla [EXE] después de entrar cada línea.

Operación de teclas	
[MODE] [1] NEW [*] [EXE]	Borra los datos en el "Area fuente" (modo "WRT").
[Asmbl]	Visualiza el menú de ensamble
[S]	Especifica el mode SOURCE IN (se enciende el símbolo " [SOURCE] [IN] ").
:START:32 [EXE]	
L1:LD:0:M [EXE]	
:SUB:0:N [EXE]	
:JNZ:0:L2 [EXE]	
:HJ:0:L1 [EXE]	
L2:JC:2:L3 [EXE]	
:LD:0:N [EXE]	
:SUB:0:M [EXE]	
:ST:0:N [EXE]	
:JC:3:L1 [EXE]	
L3:ST:0:M [EXE]	
:JC:3:L1 [EXE]	
M:CONST:0018 [EXE]	
N:CONST:0010 [EXE]	
:END:L1 [EXE]	

Ahora ya ha sido almacenado el programa fuente en el "Area fuente". Ahora presione las teclas [MODE] [⊘] y retorne al modo RUN.

- **Visualizando el programa fuente**

Con el propósito de confirmación visualizaremos los contenidos escritos. Ejecutando el comando LIST∗1 en el modo RUN o en el modo WRT, se pueden visualizar los contenidos del "Area fuente" en el siguiente formato de "programa fuente".

Etiqueta : Código de instrucción : Campo GR : Campo AD : Campo XR

Tres caracteres	Cinco caracteres	Un carácter	Tres caracteres	Según el programa fuente

El programa fuente recién entrado puede ser visualizado de la manera siguiente presionando las teclas [SHIFT] [LIST] [*] [1] [EXE] . Como la visualización se detendrá en cada línea, presionar la tecla [EXE] para avanzar a la próxima línea.

Operación	Visualización
[MODE] [⊘]	Ready P0
LIST∗1 [EXE]	:START:32
[EXE]	L1 :LD :0:M
[EXE]	:SUB :0:N
[EXE]	:JNZ :0:L2
[EXE]	:HJ :0:L1
[EXE]	L2 :JC :2:L3
[EXE]	:LD :0:N
[EXE]	:SUB :0:M
[EXE]	:ST :0:N
[EXE]	:JC :3:L1
[EXE]	L3 :ST :0:M
[EXE]	:JC :3:L1
[EXE]	M :CONST:0018
[EXE]	N :CONST:0010
[EXE]	:END :L1
[EXE]	Ready P0

* El programa fuente puede ser visualizado mediante los comandos ya mencionado LIST∗ y LIST∗ (0) (la cifra entre paréntesis puede ser omitida). En el segundo caso, el programa fuente se visualiza contínuamente segundo a segundo con el número de línea agregado. Presione la tecla [STOP] si desea detener la visualización temporalmente para verificar los contenidos. Presione la tecla [EXE] cuando desee continuar la visualización.

- **Editando el programa fuente** (Corrección, inserción, borrado)

Refiérase al capítulo 7 en el MANUAL DEL PROPIETARIO, ya que la manipulación es exactamente igual que en el caso de los datos de apuntes en el BANCO DE DATOS.

10-2 Ensamble

El simulador tiene funciones de ensamble para ensamblar el programa fuente escrito en lenguaje de ensamble y convertirlo en el objeto. Para efectuar el ensamble, presione la tecla [A] después de presionar la tecla [Asmbl] .

Si se presiona la tecla [Asmbl] después de escribir el programa fuente en el "Area fuente", el símbolo "ASMBL" se enciende en la ventanilla de visualización superior. Luego se crea el objeto a partir del programa fuente presionando la tecla [A] . Se reservará un bloque en el almacenamiento principal para crear el objeto correspondiente a la capacidad de área libre (número de bytes (octetos) restantes)) restante al momento de presionar la tecla [A] .

Número de bytes (octetos) restantes	Bloque de almacenamiento reservado
0 ~ 512 bytes (octetos)	Area libre insuficiente para crear el objeto. Se visualiza un "Error 1".
513 ~ 1024 bytes (octetos)	Reserva un bloque de almacenamiento (256 palabras) y ejecuta el ensamble.
1025 ~ 1536 bytes bytes (octetos)	Reserva dos bloques de almacenamiento (512 palabras) y ejecuta el ensamble.
⋮	
Más de 4.097 bytes (octetos)	Reserva ocho bloques de almacenamiento (2.048 palabras) y ejecuta el ensamble.

Sin embargo, si hay etiquetas de programa se requerirán cinco bytes (octetos) por cada etiqueta.

La entrada de dirección es efectiva en el rango de direcciones reservadas (0 ~ 2.047 en el caso de ocho bloques de almacenamiento).

El simulador también posee la función de detectar y visualizar errores durante el ensamble. Los mensajes de error durante la ejecución del ensamble difieren de los mensajes de error que aparecen durante la ejecución de un programa en BASIC y son los que aparecen abajo.

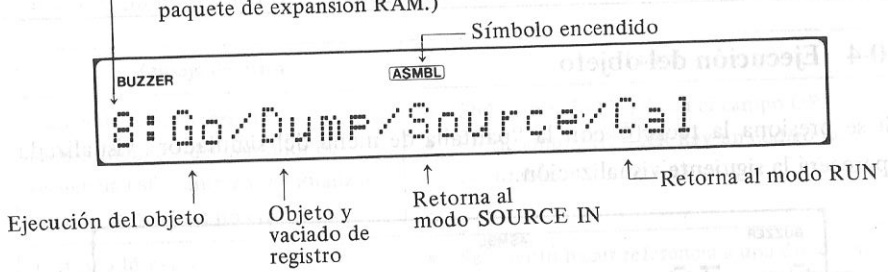
Mensaje de error	Causa
Error 1	• Incapaz de reservar el bloque de almacenamiento para crear el objeto. • Area insuficiente para la tabla de rótulos.
Label error (Error de etiqueta) (También se visualiza el número de registro de la línea con el error)	• Se declararon etiquetas duplicados. • Se ha agregado una etiqueta a la seudo-instrucción END. • El carácter cabeza del rótulo no es una letra alfabética en mayúscula. • La etiqueta excede de tres caracteres.
Op-code error (Error de código-operando) (También visualiza el número de registro del registro con error)	• El deletreo del código de instrucción está errado. • El código de instrucción no está escrito.
Operand error (Error de operando) (También visualiza el número de registro del registro con error)	• El operando (contenidos del campo GR y del campo AD) no está escrito. • El operando (contenidos del campo GR y del campo XR) no está escrito. • El operando (contenidos del campo AD) excede de tres caracteres). • No hay un rótulo de referencia con el operando (contenidos del campo AD).
Source error (Error de fuente)	• No hay seudoinstrucciones para START (comienzo) o END (fin). • Otros.

Si se visualiza un mensaje de error, corrija el programa fuente correspondiente al error después de fijar el modo SOURCE IN ([Asmbl] [S]). Si el ensamblador se ejecuta correctamente, GR0, GR1, GR2 y GR3 serán 0 (borrados).
Si se produce un error siga los siguientes procedimientos.

1. Si se produce un Error 1
   a) Detenga el ensamble y fije el "modo RUN" mediante la tecla [BRK].
   b) Fije el "modo SOURCE IN" mediante la tecla [EXE].
2. Sie se produce un error de etiqueta, un error de Op-code (código de operando) o un error de operando.
   a) Detenga temporalmente el ensamble y fije el "modo RUN" mediante la tecla [BRK].
   b) Continúe el ensamble mediante la tecla [EXE] y fije el modo "SOURCE IN" después de terminar.

10-3 Pantalla de menú del simulador

Si el ensamble se ejecuta correctamente, la visualización aparecerá como se muestra abajo con el símbolo "[ASMBL]" encendido.

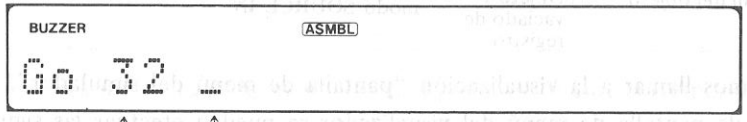


Deberemos llamar a la visualización "pantalla de menú del simulador". En el estado de pantalla de menú del visualizador se pueden efectuar las siguientes operaciones de teclas.

Operación	Función
[G]	Cambia a la "pantalla de ejecucción del objeto" (se explica en el Capítulo 10-4).
[D]	Cambia a la "pantalla de menú del vaciado" (se explica en el Capítulo 10-6).
[S]	Retorna al modo SOURCE IN.
[C]	Retorna al modo "RUN".
[MODE] [•]	Pone el sonido de zumbador de entrada de teclas en las posiciones ON y OFF. (Cuando en la posición ON, se enciende el símbolo "BUZZER".)
[MODE] [2]	Activa el modo de seguimiento. (Se enciende el símbolo "TRACE ON".)
[MODE] [3]	Cancela el modo de seguimiento.
[MODE] [7]	Activa el modo de impresión. (Se enciende el símbolo "PRT ON".)
[MODE] [8]	Cancela el modo de impresión.

10-4 Ejecución del objeto

Si se presiona la tecla [G] con la "pantalla de menú del simulador" visualizada, aparecerá la siguiente visualización.



Llamaremos a esta visualización la "**pantalla de ejecución del objeto**".
Si se presiona la tecla [EXE] con esta pantalla visualizada, el objeto será ejecutado a partir de la dirección de comienzo de ejecución.
Para cambiar la dirección de comienzo de la ejecución, entrar un nuevo número de dirección o etiqueta correspondiente y presionar la tecla [EXE]. La dirección de comienzo de la ejecución será entonces cambiada y la visualización retornará a la "pantalla de ejecución del objeto".

Operación	Visualización	
	Go 32 _	
[4] [0]	Go 32 40_	(Cambia a la dirección 40.)
[EXE]	Go 40 _	
[L] [1]	Go 40 L1_	(Cambia a la dirección de la etiqueta L1.)
[EXE]	Go 32 _	
[EXE] (Ejecución del objeto)	8:Go/Dump/Source/Cal	(Retorna a la "pantalla de menú del simulador.")

Si el programa de muestra se ejecuta como arriba, la visualización retornará a la "pantalla de menú del simulador" y la ejecución se detendrá mediante la instrucción HJ sin visualizaciones ya que no hay instrucciones READ o WRITE. Para observar la ejecución en proceso, será necesario utilizar el "seguimiento de objeto" (que será explicado en el Capítulo 10-5).

- **Mensajes de error durante la ejecución del objeto**

Como el objeto se ejecutará bajo el control de este simulador, se visualizarán los siguientes mensajes de error si se produce un error durante la ejecución.

Mensaje de error	Causa
Bad code (Código errado) (Los contenidos del contador de secuencia (SC) también se visualizan cuando se produce un error.)	• Código errado (7 ó 9) en el campo OP. • En el campo GR o XR hay un código no ejecutable.
Out of address (Fuera de la dirección)	• Se intentó hacer referencia a una dirección que no estaba reservada.
Overflow (Desbordamiento) (Los contenidos del contador de secuencia (SC) también se visualizan cuando se produce un error.)	• Se produjo un desbordamiento durante la ejecución de una instrucción ADD o SUB. * En este caso se puede resumir la ejecución presionando la tecla [EXE].

Presionar la tecla [BRK] para cancelar el error. Luego la visualización retornará a la "pantalla de menú del simulador".

10-5 Seguimiento del objeto

Si el objeto se ejecuta en el modo de seguimiento (presionar [MODE] [2] y se encenderá el símbolo "TRACE ON"), la ejecución se detendrá después de visualizar los contenidos del objeto y del registro general en el siguiente formato para cada instrucción. Presionar la tecla [EXE] para avanzar la ejecución y visualizar la próxima instrucción.

Dirección : Objeto ␣ Contenido ␣ Contenido ␣ Contenido ␣ Contenido

(Decimal de 3 dígitos)	(Hexadecimal de 4 dígitos)	GR0 (Decimal)	GR1 (Decimal)	GR2 (Decimal)	GR3 (Decimal)

(␣ indica un espacio de un carácter)

*Si la visualización sobrepasa de 24 dígitos, se desplazará hacia la izquierda de manera que los caracteres de la derecha queden a la vista. Presionar la tecla [STOP] para detener temporalmente el desplazamiento y presionar la tecla [EXE] para resumirlo.

Como se puede efectuar el seguimiento del objeto de esta manera, nosotros ejecutaremos el seguimiento del programa de muestra.

Operación	Ejemplos de visualización
	BUZZER ASMBL 8:Go/Dump/Source/Cal
[MODE] [2]	BUZZER ASMBL TRACE ON 8:Go/Dump/Source/Cal
[G]	BUZZER ASMBL TRACE ON Go 32_
[EXE]	BUZZER ASMBL TRACE ON 32:C02B 8 0 0 0
[EXE]	BUZZER ASMBL TRACE ON 33:B02C 0 0 0 0
[EXE]	BUZZER ASMBL TRACE ON 34:1024 0 0 0 0

(Se omiten las operaciones subsiguientes.)

*Como las visualizaciones diferirán dependiendo del tipo de información en la computadora, las visualizaciones no necesariamente serán como las que aparecen arriba.

La mera ejecución del seguimiento de un objeto de esta manera es inefectiva, ya que sólo se visualizarán en orden sucesivo valores "sin sentido" almacenados en el registro general. Los cambios en el registro serán fácilmente comprendidos si la ejecución se sigue después de inicializar los contenidos del registro presionando la tecla [A] en el menú de ensamble. Para cancelar el modo TRACE, presione [MODE] [3] . El símbolo "TRACE ON" se apagará.

10-6 Vaciado del objeto y registro

El sacar los contenidos de la memoria principal o del registro a un dispositivo de salida es lo que se llama "**vaciado**".

Si se presiona la tecla [D] con el "pantalla de menú del simulador" se visualizará lo siguiente.

BUZZER ASMBL
Dump:Object/Register

Llamaremos a esta visualización "**pantalla de menú del vaciado**". Con esta "pantalla de menú del vaciado" son posibles las siguientes operaciones de teclas.

Operación	Función
[O]	Vacía los contenidos del objeto.
[R]	Vacía los contenidos del registro.

- **Vaciado del objeto**

Si se presiona la tecla [O] con la "pantalla de menú del vaciado", el objeto será vaciado. Como las direcciones de comienzo del vaciado y fin del vaciado serán pedidas, éntre los valores o etiquetas de dirección respectivas.

Operación	Visualización
	Dump:Object/Register
[O]	from _
	↑ Visualiza el cursor y está en el estado de espera de la dirección de comienzo del vaciado.
[L][1] (Entra la etiqueta)	from L1_
[EXE]	from 32 to _
	↑ Estado de espera de entrada de la dirección de fin del vaciado.
[4][2] (Entra el valor de la dirección)	from 32 to 42_

Si se presiona la tecla [EXE] después de especificar las direcciones de comienzo y de fin del vaciado, el objeto será vaciado en el siguiente formato dentro del rango especificado.

```
****:****  *****  ** ***
```

- `****` Dirección (decimal)
- `****` Objeto (hexadecimal)
- `*****` Código de instrucción (nemotécnico)
- `**` Campo XR
- `**` Campo GR
- `***` Campo AD (decimal)

Durante una instrucción READ, el código de instrucción será visualizado agregado al campo GR y durante una instrucción WRITE el objeto también será agregado al campo y visualizado.

Ahora procederemos a vaciar el objeto del programa de muestra desde la dirección 32 a la dirección 42.

Operación	Visualización
[EXE]	32:C02B LD 00 43
[EXE]	33:B02C SUB 00 44
[EXE]	34:1024 JNZ 00 36
[EXE]	35:0020 HJ 00 32
[EXE]	36:2829 JC 20 41
[EXE]	37:C02C LD 00 44
[EXE]	38:B02B SUB 00 43
[EXE]	39:D02C ST 00 44
[EXE]	40:2C20 JC 30 32
[EXE]	41:D02B ST 00 43
[EXE]	42:2C20 JC 30 32
[EXE]	8:Go/Dump/Source/Cal

- **Vaciado del registro**

Si se presiona la tecla [R] con la "pantalla de menú del vaciado", el registro será vaciado.

Los registros serán vaciados en el orden de registro base (BR), registros generales (GR0, GR1, GR2, GR3), contador de secuencia (SC) y registro de código de condición (CC) en el siguiente formato.

Si se presiona la tecla [EXE] después de completar el vaciado dentro del rango especificado, la visualización retornará a la "pantalla de menú del simulador".

```
***:␣****␣*****␣_
```

- `***` Nombre del registro
- `****` Contenido del registro (hexadecimal de 4 dígitos)
- `*****` Contenido del registro (Decimal)
- `_` Cursor

Operación	Ejemplos de visualización
	Dump:Object/Register
R	BR : 0100 256 _
⌀	BR : 0100 256 0_
EXE	BR : 0000 0 _
EXE	GR0: 0000 0 _
EXE	GR1: FFFF -1 _
⌀	GR1: FFFF -1 0_
EXE	GR1: 0000 0 _
EXE	GR2: 0008 8 _
⌀	GR2: 0008 8 0_
EXE	GR2: 0000 0 _
EXE	GR3: 0000 0 _
EXE	SC : 0020 32 _
EXE	CC : 0000 0 _
EXE	8:Go/Dump/Source/Cal

Notas:

1) Los cambios en el registro base (BR) sólo son posibles para los 8 bits de orden superior (bits del Nº 0 al Nº 7). Los 8 bits de orden inferior (bits del Nº 8 al Nº 15) serán descartados automáticamente.
2) En el registro de código de condición (CC) sólo es posible efectuar cambios con 0 ó 1. Cualquier otro valor que se entre será fijado como 0 si es un número par y como 1 si es un número impar.

* Los contenidos de los diversos registros no serán necesariamente los mismos que en el ejemplo anterior, ya que diferirán dependiendo del tipo de información en la computadora en ese momento.
Sin embargo, GR0 a GR3 estarán en 0 inmediatamente después del ensamble.

Los cambios en el registro pueden ser efectivamente seguidos presionando la tecla A en el menú de ensamble y ejecutando el ensamble de esta manera, y también ejecutando el "seguimiento de la ejecución del objeto" después de inicializar los contenidos de cada registro. Usaremos como ejemplo el programa siguiente.

Operación	Visualización
	BUZZER ASMBL 8:Go/Dump/Source/Cal
MODE 2	BUZZER ASMBL TRACE ON 8:Go/Dump/Source/Cal
G	BUZZER ASMBL TRACE ON Go 32 _
EXE	BUZZER ASMBL TRACE ON 32:C02B 24 0 0 0
EXE	BUZZER ASMBL TRACE ON 33:B02C 8 0 0 0
EXE	BUZZER ASMBL TRACE ON 34:1024 8 0 0 0
EXE	BUZZER ASMBL TRACE ON 36:2829 8 0 0 0
EXE	BUZZER ASMBL TRACE ON 41:D02B 8 0 0 0
EXE	BUZZER ASMBL TRACE ON 42:2C20 8 0 0 0
EXE	BUZZER ASMBL TRACE ON 32:C02B 8 0 0 0
EXE	BUZZER ASMBL TRACE ON 33:B02C -8 0 0 0
EXE	BUZZER ASMBL TRACE ON 34:1024 -8 0 0 0
EXE	BUZZER ASMBL TRACE ON 36:2829 -8 0 0 0

EXE
BUZZER ASMBL TRACE ON
37:C02C 16 0 0 0

EXE
BUZZER ASMBL TRACE ON
38:B02B 8 0 0 0

EXE
BUZZER ASMBL TRACE ON
39:D02C 8 0 0 0

EXE
BUZZER ASMBL TRACE ON
40:2C20 8 0 0 0

EXE
BUZZER ASMBL TRACE ON
32:C02B 8 0 0 0

EXE
BUZZER ASMBL TRACE ON
33:B02C 0 0 0 0

EXE
BUZZER ASMBL TRACE ON
34:1024 0 0 0 0

EXE
BUZZER ASMBL TRACE ON
8:Go/Dump/Source/Cal

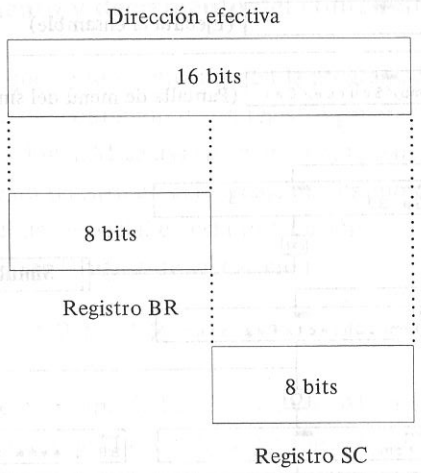
MODE 3
BUZZER ASMBL
8:Go/Dume/Source/Cal

(Cancela el modo de seguimiento)

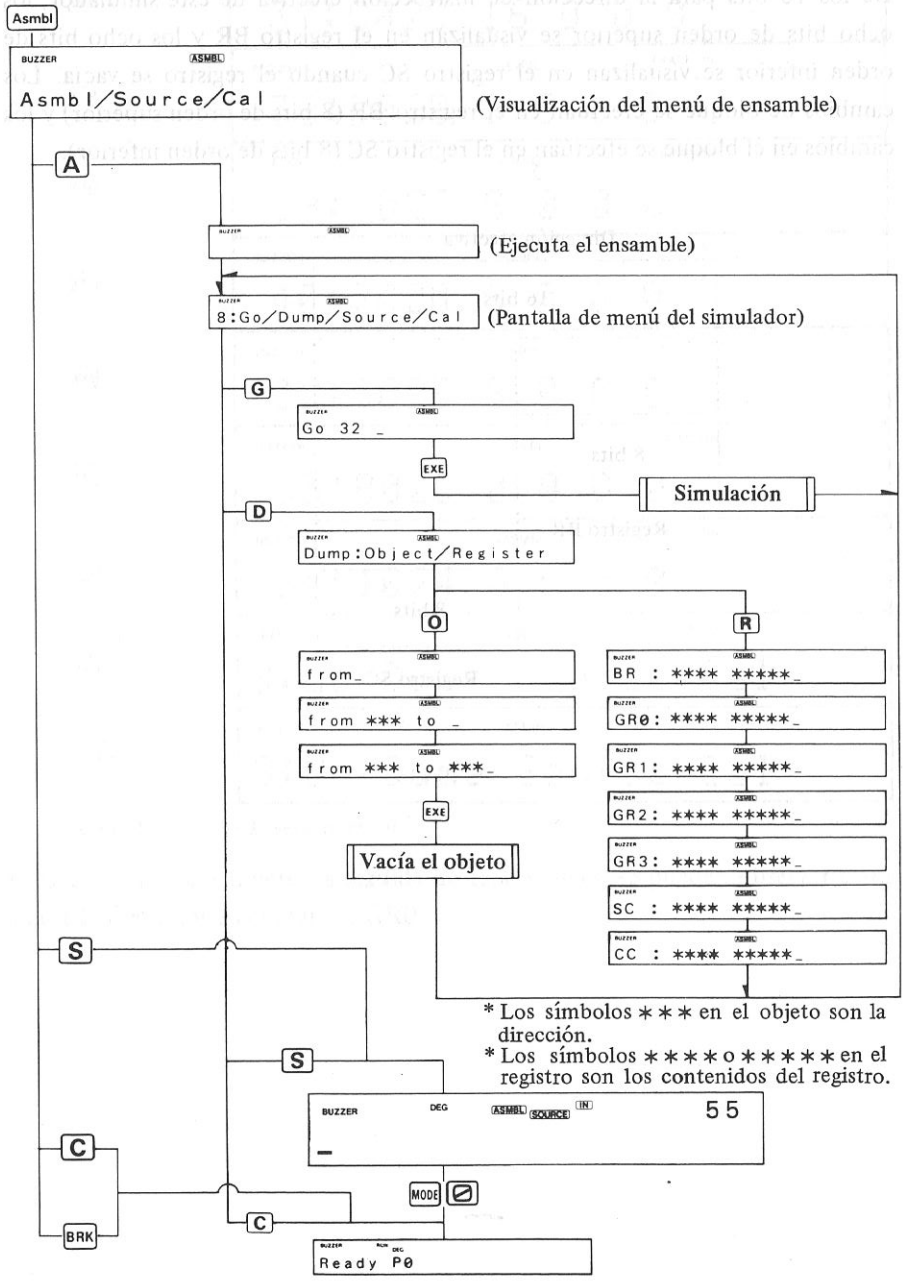
Si la ejecución del objeto es seguida de esta manera, se pueden observar claramente los cambios en el registro GR0.

- **Precauciones para el vaciado de registros**

De los 16 bits para la dirección de instrucción efectiva de este simulador, los ocho bits de orden superior se visualizan en el registro BR y los ocho bits de orden inferior se visualizan en el registro SC cuando el registro se vacía. Los cambios de bloque se efectuan en el registro BR (8 bits de orden superior) y los cambios en el bloque se efectúan en el registro SC (8 bits de orden inferior).



10-7 Resumen de las operaciones fundamentales del simulador



* Los símbolos ＊＊＊ en el objeto son la dirección.
* Los símbolos ＊＊＊＊o＊＊＊＊＊ en el registro son los contenidos del registro.

11 Técnicas fundamentales de programación en lenguaje de ensamble

En este capítulo introduciremos algunas técnicas fundamentales de programación en lenguaje ensamble en este simulador. Como cada instrucción en lenguaje de ensamble sólo ejecuta un proceso muy simple, un número de instrucciones debe ser apilado cuando la CPU ejecuta un trabajo de magnitud. La técnica que usted aprenderá en este capítulo será útil como un "componente" cuando se encuentre escribiéndo un programa más o menos largo en longuaje de ensamble.

11-1 Incremento y decremento del contenido de un registro

Una operación que se usa a menudo en la programación en lenguaje de máquina es el incremento del contenido de un registro general en un cierto número.
Como la instrucción LAI se usa frecuentemente para incrementar o decrementar el contenido de un registro en el lenguaje de ensamble de este simulador, primero aprenderemos el uso especial de esta instrucción.
Por ejemplo, nótese la siguiente instrucción.

```
LAI    :2:1:2
```

Esto indica que el campo GR es 2, el campo AD es 1 y el campo XR es 2. Como el campo XR está especificado, la dirección efectiva será calculada con la modificación de dirección por el registro índice GR2.

Dirección efectiva = 1 + (contenido de GR)
(8 bits de orden inferior) ← Valor del campo AD

Con la instrucción LAI, los 8 bits de orden inferior (bit Nº 8 al 15) de la dirección efectiva se almacenan en el registro general especificado por el campo GR. Como GR2 está especificado como el registro general en este caso, será posible incrementar el contenido de GR2 usando esta instrucción como se muestra abajo.

GR2 ← 8 bits de orden inferior de [(contenidos de GR2) + 1]

Sin embargo, la instrucción LAI sólo puede ser manipulada dentro del rango de 0 ~ 255 como los 8 bits de orden inferior de la dirección efectiva. Los valores sobre 255 se dividen por 256 y el resto se fija como los 8 bits de orden inferior. Ejecutar la siguiente instrucción asumiendo que el contenido presente del registro general GR2 es 4.

```
LAI     :2:255:2
```

Los 8 bits de orden inferior de la dirección efectiva presente pueden ser calculados de la manera siguiente:

Dirección efectiva = 255 + 4 = 259

(8 bits de orden inferior) — Valor del campo AD — Contenido de GR2

Sin embargo, como 259 es mayor que 255, el resto 3 en la siguiente fórmula será usado como los 8 bits de orden inferior de la dirección efectiva.

259 ÷ 256 = 1 con resto de 3

Ejecutando la instrucción en esta manera, el contenido del registro general GR2 será 3, con el valor original 4 disminuído en 1.

Por lo tanto, será posible usar este método para incrementar o decrementar los contenidos de los registros generales GR1, GR2 y GR3 en 1.

También, cambiando el valor asignado al campo AD, será posible variar el contenido del registro general correspondiente a ese valor. En la tabla siguiente se muestran algunos ejemplos.

Instrucción	Significado
LAI :*n*:1:*n*	Incrementa GR*n* en 1
LAI :*n*:2:*n*	Incrementa GR*n* en 2
LAI :*n*:255:*n*	Decrementa GR*n* en 1
LAI :*n*:254:*n*	Decrementa GR*n* en 2

(*n* = 1, 2 ó 3)

Cuando usted desee incrementar el contenido de GR*n* en un cierto número, usted puede simplemente escribir ese valor en el campo AD. Cuando usted desee decrementar el contenido de GR*n*, reste el valor decrementado desde 256 y escriba el resultado en el campo AD. (Por ejemplo, si usted desea decrementar en 3, escriba 253 (256 − 3 = 253) en el campo AD).

Sin embargo, este método no puede ser usado con GR0 ya que la modificación de dirección por el registro índice está siendo usada. Como los 8 bits de orden inferior de la dirección efectiva también se usan, se debería notar que las variaciones en el valor sólo serán posibles dentro del rango de 0 ~ 255.

Ejemplo:

Si el siguiente programa fuente se ensambla y el objeto es ejecutado, el contenido 5 del registro general GR1 será decrementado en 1 y la ejecución se detendrá cuando el contenido llegue a 0.

```
        :START:32          (Crea el objeto desde la dirección 32.)
BGN:LAI   :1:5             (Fija el valor inicial 5 en GR1.)
LP  :WRITE:1:10            (Visualiza el contenido de GR1 en
                            notación decimal.)
    :LAI   :1:255:1        (Decrementa el contenido de GR1 en 1.)
    :JNZ   :1:LP           (Salta a LP si GR1 ≠ 0.)
    :HJ    :0:BGN          (Fija el contador de secuencia en
                            BGN y se detiene.)
    :END   :BGN            (Termina el programa.)
```

Operación

```
Asmbl  Asmbl/Source/Cal
A      8:Go/Dump/Source/Cal
G      Go 32 _
EXE    GR1 (10) 5
EXE    GR1 (10) 4
EXE    GR1 (10) 3
EXE    GR1 (10) 2
EXE    GR1 (10) 1
EXE    8:Go/Dump/Source/Cal
```

Como el contenido de GR1 es 0, la ejecución es detenida por la instrucción HJ y la visualización vuelve a la "pantalla de menú del simulador".

Aquí se debe notar que la instrucción "WRITE : 1 : 10" se ejecuta cinco veces. Si se programa como se muestra abajo, generalmente será posible repetir el proceso escrito entre dos instrucciones LAI un número especificado de veces usando GR1 como contador (registro para contar la frecuencia).

```
      :LAI    :1: Frecuencia
LP    : Proceso a ser repetido
        .
        .
        .
      :LAI    :1:255:1
      :JNZ    :1:LP
```

(Sin embargo, como se usa la instrucción LAI, la frecuencia se limita al rango de 1 ~ 256.)

11-2 Usando el área de trabajo

Existirán frecuentes casos en que un cierto número de palabras serán reservadas en la memoria principal para usarlas en guardar cálculos intermedios o guardar el contenido de un registro temporalmente. Esa parte de la memoria principal usada para este propósito es llamada "**área de trabajo**". Esta puede ser considerada equivalente a la "variable" en un programa en BASIC. Ahora estudiaremos el uso del área de trabajo mediante ejemplos.

Ejemplo: Transferencia de valores entre registros generales

Aunque no existe una instrucción en el lenguaje de máquina del simulador para efectuar la transferencia directa de los contenidos entre dos registros generales, puede surgir la necesidad de intercambiar los contenidos de los registros generales GR0 y GR1. Esto puede ser logrado mediante el área de trabajo reservada en la memoria principal.

El siguiente programa sirve para intercambiar los contenidos de GR0 y GR1.


```
      :START:32
BGN:READ  :0:10   } Entra un valor decimal en el registro.
      :READ  :1:10 }
SWP:ST    :0:WK1  (Almacena el contenido de GR0 en WK1.)
      :ST    :1:WK0  (Almacena el contenido de GR1 en WK0.)
      :LD    :0:WK0  (Carga el contenido de WK0 en GR0.)
      :LD    :1:WK1  (Carga el contenido de WK1 en GR1.)
OUT:WRITE :0:10   } Saca el contenido del registro
      :WRITE :1:10 } en notación decimal.
      :HJ    :0:BGN  (Detiene el programa.)
WK0:RESV  :1      } Area de trabajo.
WK1:RESV  :1      }
      :END   :BGN
```

En este trabajo, el intercambio se ejecuta por las cuatro líneas de instrucciones desde la línea con la etiqueta SWP. Este intercambio se logra usando las dos áreas de trabajo (etiquetas WK0 y WK1) e invirtiendo la orden de almacenamiento y carga.

Si este programa se ensambla y el objeto se ejecuta, se visualizará lo siguiente.

Operación

```
[Asmbl]  Asmbl/Source/Cal
[A]      8:Go/Dump/Source/Cal
[G]      Go 32_
[EXE]    GR0 (10) _
[1]      GR0 (10) 1_
[EXE]    GR1 (10) _
[2]      GR1 (10) 2_
[EXE]    GR0 (10) 2
[EXE]    GR1 (10) 1
[EXE]    8:Go/Dump/Sorce/Cal
```

De la operación anterior se puede notar que si se entra el valor 1 en GR0 desde el teclado y el valor 2 se entra en GR1, el valor final de GR0 será 2 y el valor final de GR1 será 1 mediante la operación de intercambio.

Ejemplo: Resultados de cálculos intermedios

El incremento del contenido de un registro general de a 2 veces, 4 veces, 8 veces, ... (en general 2^n veces) se puede lograr llevando a cabo el desplazamiento hacia la izquierda usando la instrucción SFT. Luego, ¿qué debemos hacer cuando deseamos múltiplos distintos de estos?

Se pueden considerar varios métodos, uno de ellos es usar el múltiplo de 2^n en combinación. En este caso crearemos un programa para aumentar el contenido del registro general GR0 en 10 veces usando el hecho que 10 = 2 + 8.

```
      :START:32
BGN:READ  :0:10      (Entra el valor decimal n en GR0.)
      :SFT  :0:1:1   (Dobla (2 x n) el contenido de GR0 desplazando un bit hacia la izquierda.)
      :ST   :0:WK    (Almacena el resultado intermedio (2 x n) en WK.)
      :SFT  :0:2:1   (Aumenta el contenido de GR0 en ocho veces (8 x n) desplazándolo dos bits más hacia la izquierda.)
      :ADD  :0:WK    (Suma WK (2 x n) a a GR0 (8 x n) y lo almacena en GR0.)
      :WRITE:0:10    (Saca el contenido de GR0 (10 x n) en notación decimal.)
      :HJ   :0:BGN   (Detiene el programa.)
WK    :RESV :1       (Area de trabajo)
      :END  :BGN
```

El contenido de GR0 se dobla mediante la instrucción SFT en la 3ª línea. Esta se guarda temporalmente en el área de trabajo WK y luego se desplaza dos bits más hacia la izquierda mediante la instrucción SFT en la 5ª línea, así el contenido de GR0 es 8 veces el contenido original. Se puede lograr un aumento de 10 veces sumando el valor de WK.

A continuación se muestra el ejemplo de la ejecución de un objeto después del ensamble.

Operación

[Asmbl]	Asmbl/Source/Cal
[A]	8:Go/Dump/Source/Cal
[G]	Go 32 _
[EXE]	GR0 (10) _
[1][1][2][5]	GR0 (10) 1125_
[EXE]	GR0 (10) 11250
[EXE]	8:Go/Dump/Source/Cal

Otro método de uso del área de trabajo es guardar temporalmente el contenido actual de un registro general en el área de trabajo para usar el registro para un cierto proceso y retornar a este contenido después de completar el proceso.

```
      :
      :ST   :3:WK    (Guarda el contenido de GR3 en WK temporalmente.)
  Proceso usando GR3
      :LD   :3:WK    (Retorna el contenido desde WK a GR3.)
      :
WK    :RESV :1       (Area de trabajo)
```

11-3 Usando la operación lógica

En esta sección estudiaremos el uso de la instrucción de producto lógico, AND y la instrucción de O exclusivo, EOR.

- **Instrucción de enmascaramiento**

Un ejemplo típico del uso de la instrucción AND es la recuperación de información desde una posición de bit específica de un registro general. Este tipo de operación se llama generalmente, "operación de enmascaramiento" ya que toda la información en las posiciones de bit innecesarias se hacen "invisibles" fijandolas en 0.

La operación de enmascaramiento se efectúa almacenando inicialmente un dato de 1 palabra (generalmente llamado dato de máscara), con la posición de bit requerida como 1 y todas las otras posiciones como 0, en la memoria principal. Luego obteniendo el AND del registro general y el dato máscara.

Ejemplo:
En esta operación, reservaremos los bits numerados en par (los bits 0, 2, 4, 6, 8, 10, 12 y 14) en el registro general GR1 y fijaremos todos los otros bits en 0. Luego aparecerá el dato de máscara para una palabra como se muestra abajo.

```
1010101010101010
```
(AAAA en notación hexadecimal)

A continuación se muestra un ejemplo de un programa usando la operación de enmascaramiento.

```
      :START:32
BGN:READ :1:16    (Entra un valor hexadecimal en GR1.)
    :AND  :1:MSK  (Operación de enmascaramiento)
    :WRITE:1:16   (Saca los contenidos de GR1 en notación
                   hexadecimal.)
    :HJ   :0:BGN  (Detiene el programa.)
MSK:CONST:AAAA    (Dato de máscara)
    :END  :BGN
```

A continuación se muestra un ejemplo de la ejecución de un objeto después del ensamble.

Operación

Asmbl	Asmbl/Source/Cal
A	8:Go/Dump/Source/Cal
G	Go 32 _
EXE	GR1 (16) _
F 5 A C	GR1 (16) F5AC
EXE	GR1 (16) A0A8
EXE	8:Go/Dump/Source/Cal

El dato hexadecimal F5AC fijado en GR1 aparecerá como se muestra abajo si se expresa como una cadena de bits binarios.

```
1111010110101100
```


Si sólo se han reservado los bits en las posiciones numeradas par y todas las otras se han fijado en 0, éste aparecerá como se muestra abajo.

```
1010000010101000
 A    0    A    8
```
(notación hexadecimal)

- **Complemento a 2**

En este simulador, un entero negativo se manipula como un "complemento de 2". El "complemento de 2" de una palabra de 16 bits significa el estado cuando se agrega 1 a todos los bits en el cual se invierten los 0s y 1s.
La instrucción EOR se usa para invertir los bits para obtener el "complemento de 2" en lenguaje de ensamble. A continuación se muestra un programa para obtener el "complemento de 2" del contenido del registro general GR0.

Ejemplo

```
      :START:32
BGN:READ :0:10    (Entra un valor numérico decimal en GR0.)
    :EOR  :0:AL1  (Invierte todos los bits de GR0.)
    :ADD  :0:ONE  (Agrega 1 a GR0.)
    :WRITE:0:10   (Saca los contenidos de GR0 en notación
                   decimal.)
    :HJ   :0:BGN  (Detiene el programa.)
AL1:CONST:FFFF    (Dato para invertir los bits)
ONE:CONST:0001    (Dato para sumar 1)
    :END  :BGN
```

A continuación se muestra un ejemplo de la ejecución de un objeto después del ensamble.

Operación

Asmbl	Asmbl/Source/Cal
A	8:Go/Dump/Source/Cal
G	Go 32 _
EXE	GR0 (10) _
1 2 3	GR0 (10) 123_
EXE	GR0 (10) -123
EXE	8:Go/Dump/Source/Cal
G	Go 32 _
EXE	GR0 (10) _
− 4 5	GR0 (10) -45
EXE	GR0 (10) 45
EXE	8:Go/Dump/Source/Cal

En el ejemplo de ejecución anterior, el "complemento de 2" está siendo obtenido para **123** y **-45**. Si se visualiza en notación decimal, se producirá la inversión de signo correspondiente. La razón para que esta inversión de signo sea posible mediante el O exclusivo del dato hexadecimal **FFFF**, en el cual todos los bits son 1, es que el O exclusivo (expresado por ⊕), 1 ⊕ 1 = 0 y 0 ⊕ 1 = 1.

La instrucción ADD se usa para sumar 1 al contenido de GR0 después de la inversión de bits. La instrucción LAI previamente mencionada no es capaz de incrementar GR0 en 1 y esta instrucción no es adecuada cuando es necesario agregar 1 a GR1, GR2 ó GR3 de esta manera, como un entero con signo de 16 bits.

11-4 Comparación

Cuando se esta comparando el tamaño de dos números, A y B, es normal verificar el signo aritmético (más, 0 ó menos) después de calcular A − B. Con la programación en lenguaje de ensamble del simulador, esto se ejecuta mediante combinaciones de la instrucción SUB, de la instrucción JC y de la instrucción JNZ.

Ejemplo:
En el programa siguiente, el valor fijado en el registro general GR0 se compara con el número decimal 50 (0032 en notación hexadecimal).

El contenido de GR0 es el número decimal 100 cuando el contenido de GR0 es > 50.
Permanece inalterado si el contenido de GR0 = 50.
El contenido de GR0 es 1 cuando el contenido de GR0 es < 50.

El procesamiento se divide de acuerdo a cada caso.

```
      :START:32
BGN:READ  :0:10    (Entra un valor decimal en GR0.)
    :ST   :0:WK    (Guarda el contenido de GR0 temporalmente.)
    :SUB  :0:FTY   (Compara con 50.)
    :JC   :1:NG    (Salta a NG si es menor que 50.)
    :JNZ  :0:POS   (Salta a POS si es mayor que 50.)
ZER:LD    :0:WK    (Retorna a GR0 si el valor original es igual
                    a 50.)
    :JC   :3:OUT   (Salta a OUT.)
NG  :LAI  :0:1     (Fija 1 en GR0.)
    :JC   :3:OUT   (Salta a OUT.)
POS:LAI   :0:100   (Fija 100 en GR0.)
OUT:WRITE:0:10     (Saca el contenido de GR0 en notación
                    decimal.)
    :HJ   :0:BGN   (Detiene el programa.)
WK  :RESV :1       (Area de trabajo para guardar temporalmente)
FTY:CONST:0032     (Dato de comparación, 50 en notación
                    decimal)
    :END  :BGN
```

Ahora mostraremos un ejemplo de ejecución de un objeto después del ensamble. Notar que el procesamiento está dividido en cada caso cuando 51, 50 y 49 están fijados en GR0.

Operación

Asmbl	Asmbl/Source/Cal
A	8:Go/Dump/Source/Cal
G	Go 32 _
EXE	GR0 (10) _
5 1	GR0 (10) 51_
EXE	GR0 (10) 100
EXE	8:Go/Dump/Source/Cal
G	Go 32 _
EXE	GR0 (10) _
5 0	GR0 (10) 50_
EXE	GR0 (10) 50
EXE	8:Go/Dump/Source/Cal
G	Go 32 _
EXE	GR0 (10) _
4 9	GR0 (10) 49_
EXE	GR0 (10) 1
EXE	8:Go/Dump/Source/Cal

Se resta 50 del contenido de GR0 mediante " SUB:0:FTY " y el resultado se almacena en el registro de código de condición (CC) de acuerdo con el signo. (CC será 0 cuando el signo es positivo ó 0, y 1 cuando es negativo.) Como el contenido de GR0 cambiará, " ST:0:WK " se ejecuta en la línea inmediatamente anterior para guardar temporalmente el contenido de GR0 en el área de trabajo WK.

Si en este momento CC es 1, la ejecución saltará a la dirección NG mediante la instrucción de salto condicional " JC:1:NG ". Esta dirección se saltará cuando el contenido de CC es 0, " JNZ:0:POS " entonces decide si el resultado es positivo y, si es así, saltará a la dirección POS. Como el resultado es 0, GR0 será devuelto a su valor original mediante " LD:0:WK ".

Al completar el procesamiento en la dirección ZER y en la dirección NG, la ejecución saltará a la dirección OUT mediante la instrucción de salto incondicional " JC:3:OUT ".

11-5 Operación de tablas

Ahora crearemos un programa en lenguaje de ensamble para procesar los "datos en formato de tablas". Como los "datos en formato de tablas" están normalmente almacenados en áreas sucesivas de la memoria principal, estos datos se llaman **tabla.**

Ejemplo:

Crearemos un programa para obtener el total de cinco datos en la tabla que comienza en la dirección TBL. Si usamos la modificación de dirección mediante el registro índice en la operación de tabla, el programa puede ser acortado. En el programa que se muestra a continuación, el total se fija en el registro general GR0 usando GR1 como registro índice.

```
      :START:32
BGN   :LAI   :0:0        (Borra GR0.)
      :LAI   :1:4        (Fija 4 en GR1.)
LP    :ADD   :0:TBL:1    (Suma usando el registro índice.)
      :LAI   :1:255:1    (Decrementa GR1 en 1.)
      :JNZ   :1:LP       (Vuelve al ciclo si GR1 no es 0.)
      :ADD   :0:TBL      (Suma el contenido de la dirección TBL
                          al final.)
      :WRITE:0:10        (Saca un valor en notación decimal.)
      :HJ    :0:BGN      (Detiene el programa.)
TBL   :CONST:0008   }
      :CONST:0002   }
      :CONST:0006   }  Tabla
      :CONST:0001   }
      :CONST:0003   }
      :END   :BGN
```

Fija el valor inicial de 4 en el registro índice GR1. Como el contenido de GR1 se decrementa de a 1 a la vez mediante la instrucción LAI cada vez que el ciclo que comienza desde la dirección LP se hace, la dirección efectiva del " ADD:0: TBL:1 " cambia de la manera siguiente y los datos de la tabla se suman en orden sucesivo al GR0 desde el dato final.

Dirección TBL+4
Dirección TBL+3
Dirección TBL+2
Dirección TBL+1

La ejecución escapa del ciclo en el momento que el contenido de GR1 llega a 0. Como los datos de la dirección TBL aún permanecen, éstos se suman mediante " ADD:0: TBL ". El total de la suma de los cinco datos, 20, en la tabla se fijan en GR0 de esta manera.
El uso de las variables arreglo en BASIC corresponde a este tipo de operación de tabla.

11-6 Creando y usando subrutinas

La instrucción más difícil del simulador es la instrucción de salto de subrutina JSR. Sin embargo, como JSR es la única instrucción capaz de cambiar el contenido del registro base (BR), es muy importante para programar con lenguaje de ensamble y dominar el uso de esta instrucción.

Ejemplo:
Crearemos una subrutina (con la etiqueta DIV) que divida el contenido del registro general GR2 por el contenido del registro general GR3 y, después de fijar el cuociente en GR2 y el resto en GR3, retornar a la rutina principal. Para el almacenamiento de programas, pondremos la rutina principal en el bloque de almacenamiento Nº 0 y la subrutina en el bloque de almacenamiento Nº 1.


```
        :START:32
BGN     :READ :2:10     (Entra un valor decimal en GR2.)
        :READ :3:10     (Entra un valor decimal en GR3.)
        :JSR  :0:SB     (Fija la dirección de retorno en GR0 y salta
                         a una subrutina.)
        :WRITE:2:10     (Saca el cuociente de GR2 en notación
                         decimal.)
        :WRITE:3:10     (Saca el resto de GR3 en notación decimal.)
        :HJ   :0:BGN    (Se detiene.)
SB      :ADCON:DIV      (Define una dirección de subrutina para la
                         etiqueta SB.)
        :END  :BGN

DIV:START:256
        :ST   :0:SAV    (Almacena la dirección de retorno en la
                         dirección SAV.)
        :ST   :2:A
        :ST   :3:B
        :LD   :3:A
        :LAI  :2:0
LP      :SUB  :3:B
        :JC   :1:ADJ
        :ADD  :2:ONE
        :JC   :3:LP
ADJ:ADD  :3:B
RET:JSR  :0:SAV
SAV:RESV :1
A   :RESV :1
B   :RESV :1
ONE:CONST:0001
    :END
```

En la rutina principal almacenada desde la dirección 32 en el bloque de almacenamiento Nº 0, los valores se entran en GR2 y GR3, y los valores retornados desde la subrutina están siendo entregados.

El salto a la subrutina se ejecuta mediante la instrucción " JSR:0:SB ". La dirección de retorno (dirección 35 en la cual " WRITE:2:10 " está almacenado es la dirección de retorno) desde la subrutina se fija en el registro general GR0 mediante esta instrucción y el valor de la dirección definida mediante la etiqueta SB se almacena en el contador de secuencia. Al mismo tiempo, el contenido del registro base (BR) se fija en la dirección 0100 (notación hexadecimal) del bloque de almacenamiento Nº 1 como 256, la cual es la primera dirección de la subrutina, que está difinida en la etiqueta SB mediante la seudoinstrucción ADCON.

A continuación explicaremos el proceso de la subrutina desde la dirección 256. Aunque el registro general GR0 no se usa en una subrutina, la dirección de retorno en GR0 se guarda en el área de trabajo SAV mediante " ST:0:SAV " con una finalidad de seguridad. (SAV de la etiqueta significa guardar.) "Guardando la dirección de retorno" de esta manera debería ser recordado cuando se usa una subrutina con lenguaje de máquina en este simulador. En la subrutina, el contenido de GR2 primero se transfiere a GR3 a través del área de trabajo A y el valor original de GR3 se almacena en el área de trabajo B. Este proceso se ejecuta para almacenar el resto de GR3 cuando se está retornando a la rutina principal.

La ejecución se repite en el ciclo de división (comenzando desde la dirección LP) de esta manera, usando GR2 como contador. En el ciclo, el valor del área de trabajo B se resta del valor de GR3. El ciclo continuará mientras el resultado sea positivo ó 0, y el número de veces del ciclo (GR2) será el cuociente. Si el valor de GR3 – B se hace negativo, la ejecución saldrá del ciclo mediante el salto condicional " JC:1:ADJ " y saltará a la dirección ADJ (etiqueta de ajuste). El excesivo valor de B es corregido una vez aquí mediante " ADD:3:B " y el valor del resto correcto se fija en GR3.

" JSR:0:SAV " de la dirección RET es el retorno desde la subrutina a la rutina principal. La dirección de retorno guardada en el área de trabajo SAV se fija en el contador de secuencia (SC) y el salto se hace a la dirección de retorno (dirección 35) en el bloque de almacenamiento Nº 0, exactamente la misma que para el salto a subrutina.

Nota:

El incremento del contenido de GR2 en 1 se ejecuta mediante " ADD:2:ONE " en el ciclo de división de la subrutina. Como el contenido de GR2 estará limitado desde 0 a 255, si esta instrucción se sustituye con " LAI:2:1:2 ", esta sustitución no será adecuada. (Ver el ejemplo de ejecución que aparece a continuación.)

Abajo se muestr a un ejemplo de la ejecución de un objeto después del ensamble.

Operación

Asmbl	Asmbl/Source/Cal
A	8:Go/Dump/Source/Cal
G	Go 32 _
EXE	GR2 (10) _
23456	GR2 (10) 23456_
EXE	GR3 (10) _
78	GR3 (10) 78_
EXE	GR2 (10) 300
EXE	GR3 (10) 56
EXE	8:Go/Dump/Source/Cal

El cálculo en este ejemplo es 23456 ÷ 78 = 300 con un resto de 56.

Indice

S

T

U

V

W

MO0901O909C Printed in Japan