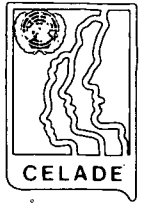


3221 (7899) c.1

Centro Latinoamericano de Demografía



Documentos de Seminario

LENGUAJE DE ENSAMBLE

DS/4
100
1975

CURSO LATINOAMERICANO DE
PROCESAMIENTO DE DATOS (PED)
APLICADO A LAS CIENCIAS SOCIALES



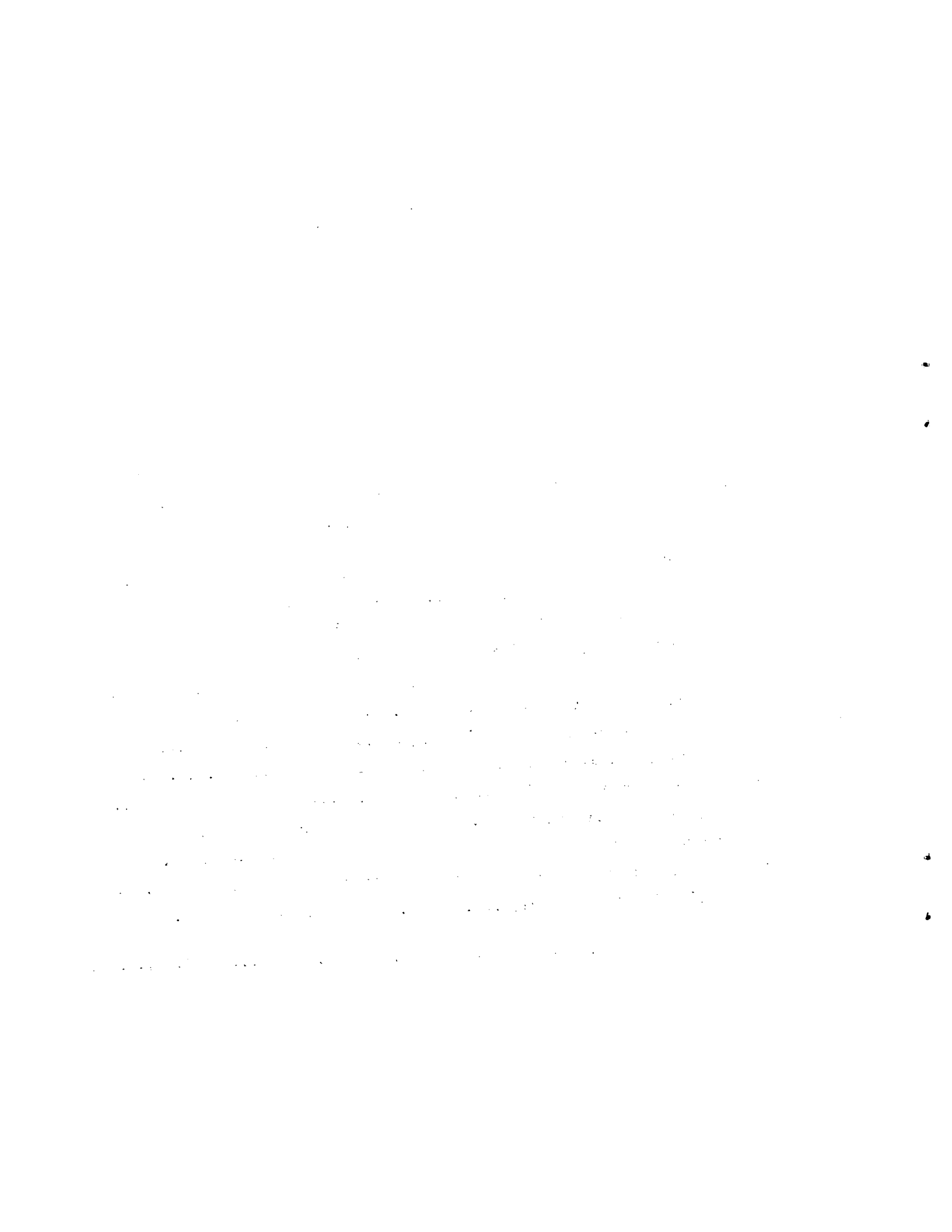
•
•

•
•



INDICE

	<u>Página</u>
I. INTRODUCCION.....	1
1. Lenguaje de máquina y lenguaje de ensamblaje:.....	1
II. UN ENSAMBLADOR.....	11
1. Definición.....	11
2. El lenguaje de ensamblaje del Sistema IBM/360/370.....	12
3. Aritmética de punto fijo.....	20
4. Ensamblado y pseudo-instrucciones.....	54
5. Instrucciones lógicas.....	79
6. Códigos mnemotécnicos ampliados.....	101
7. Aritmética decimal.....	103
8. Instrucciones de BIFURCACION.....	119
9. Subrutinas y subprogramas.....	128
10. Instrucciones nuevas de assembler para el Sistema/370.....	134
11. Entrada/Salida de información (Input-Output).....	144
12. Definiciones de macros.....	154
BIBLIOGRAFIA.....	180



I. INTRODUCCION

1. Lenguaje de máquina y lenguaje de ensamble

Para programar en el lenguaje de máquina de un computador es necesario conocer: características internas del computador como: capacidad de memoria, representación de los datos, sistema de direccionamiento, etc., formato de las instrucciones, significado y uso de cada uno de los operandos, códigos de error, etc. Todo lo anterior hace que dicha programación sea bastante lenta, con muchas posibilidades de error y difícil detección de los mismos.

A pesar de las desventajas enunciadas se verá a continuación una máquina simplificada en la que se hará uso del lenguaje de máquina o lenguaje absoluto dado que su utilización permite comprender más claramente lo que ocurre internamente cuando se está programando en un lenguaje de alto nivel, al mismo tiempo que sirve como introducción a los lenguajes de ensamble que constituyen el nivel de lenguaje inmediatamente superior al de máquina.

Respecto a los lenguajes de ensamble, si bien es cierto que presentan en programación un grado de dificultad menor que el lenguaje de máquina, distan bastante de ser la solución ideal para el programador, sin embargo, en toda instalación de computador o grupo de procesamiento de datos es necesario que hayan uno o dos especialistas en este tipo de lenguajes, pues cualquiera que sea el lenguaje de alto nivel que se utilice, en algún momento será necesario entrar a analizar en forma más detallada lo que ha ocurrido internamente en un proceso para ver qué causas han originado detenciones anormales del mismo.

A. Una máquina simplificada

La máquina que se utilizará será un computador de segunda generación, el ER-56 de la Standard Elektrik Lorenz. Para el objetivo que se persigue se podría haber inventado un computador con un cierto número de instrucciones que diera la posibilidad de ejemplificar las características principales del funcionamiento o forma de operación de un computador al realizar un proceso. Se ha elegido el ER-56 pues es un computador muy simple y muy fácil de programar, aún en lenguaje de máquina.

a) Características principales del computador

Memoria de trabajo: 3000 palabras (celdas)

Palabra: 7 dígitos decimales

Dirección de cada palabra: 1000 a 3999

Unidad aritmética

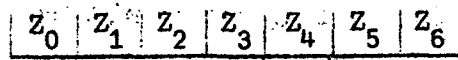
Acumulador: 14 dígitos decimales

Aritmética de punto fijo y de punto flotante

Registros: 10 (capacidad para 4 dígitos decimales)

Instrucción: 7 dígitos decimales (una palabra)

Formato



dígitos de
parámetro

dígitos de
operación

dígito
de
índice

Los dígitos de parámetro indican:

i) La dirección de la celda de memoria en la que está contenido el dato, o la dirección donde se almacenará un dato ($Z_0 \dots Z_3 = n$).

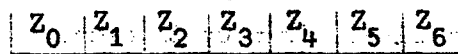
ii) Cualquier variable desde 0000 hasta 9999 que actuará como uno de los operandos en una operación ($Z_0 \dots Z_3 = p$).

El dígito de índice representa uno de los diez registros de índice ($j_i = 0, \dots, 9$) que tiene el computador.

Los dígitos de operación o código de operación indican la función que se va a realizar.

b) Representación de la información

i) Punto fijo (entera)



Z_0 = Signo (1 = + , 2 = -)

$Z_1 \dots Z_6$ = Mantisa, palabra simple

$Z_1 \dots Z_{13}$ = Mantisa, palabra doble

Desde el punto de vista aritmético, el computador interpreta el número como menor que uno, presumiendo la coma decimal ubicada siempre inmediatamente después del dígito Z_0 . Si se utiliza doble palabra se trabaja con dos celdas contiguas.

ii) Punto flotante (real)



Z_0 = Signo

$Z_1 \dots Z_{11}$ = Mantisa

$Z_{12} Z_{13}$ = Característica

Para obtener la característica se normaliza el dato, esto es, se transforma en el producto de una fracción decimal, en que los dígitos significativos están inmediatamente a continuación de la coma decimal, por una potencia de 10. El exponente de 10 se suma a la constante 50 y el resultado es la característica.

Ejemplo 1.

Dato original	=	Dato normalizado	=	Representación interna
17,385	=	$0,17385 \cdot 10^2$	=	11738500000052
0,017385	=	$0,17385 \cdot 10^{-1}$	=	11738500000049
-17385,0	=	$-0,17385 \cdot 10^5$	=	21738500000055

c) El lenguaje de máquina (absoluto)

Primer conjunto de instrucciones

7200000 indica que se trabajará en punto flotante

7900000 pone fin al proceso

Son dos instrucciones que difieren del formato general.

Código de operación	Función
31	Llevar el acumulador desde la celda n
32	Llevar desde el acumulador a la celda n
35	Sumar al acumulador contenido de celda n
36	Restar del acumulador contenido de celda n
37	Multiplicar el acumulador por contenido de celda n
38	Dividir el acumulador por contenido de celda n.

Ejemplo 2.

Programar el cálculo de:

$$y = \frac{(r+s)t}{u} - v$$

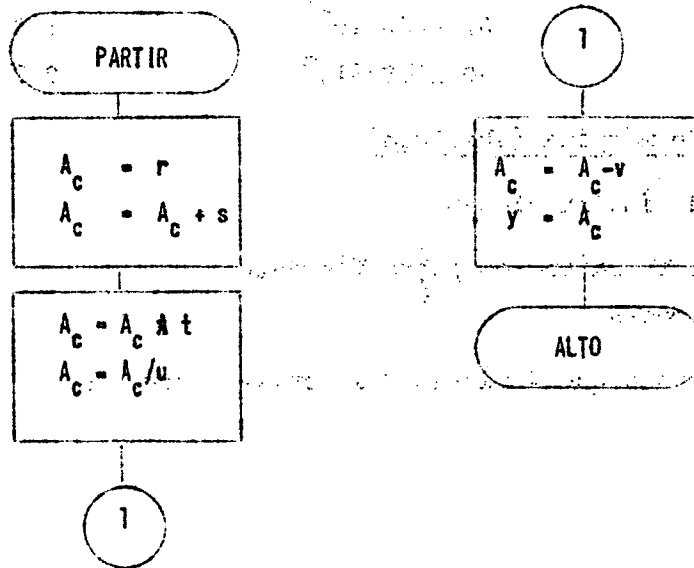
Para simplificar la descripción se utilizará la notación siguiente;

A_c : significa acumulador.

() : significa contenido de

el signo = significa "definido por"

El diagrama de flujo se hará orientado al ER-56.



Para codificar el programa se supondrá la siguiente distribución de memoria:

(1500) = r
 (1502) = s
 (1504) = t
 (1506) = u
 (1508) = v
 (1510) = y

El programa se cargará en memoria a partir de la celda 3000.

Programa codificado:

(3000) = 7200000	Partir, modo punto flotante
(3001) = 1500031	$A_c = (1500)$ $A_c = r$
(3002) = 1502035	$A_c = A_c + (1502)$; $A_c = A_c + s$
(3003) = 1504037	$A_c = A_c * (1504)$; $A_c = A_c * t$
(3004) = 1506038	$A_c = A_c / (1506)$; A_c / u
(3005) = 1508036	$A_c = A_c - (1508)$; $A_c = A_c - v$
(3006) = 1510032	$(1510) = A_c$; $y = A_c$
(3007) = 7900000	ALTO
(3008) = 9999999	ω
(3009) = 9999999	ω

Observación: El dígito 0, en el lugar que corresponde a dígito de índice no tiene ningún efecto.

El programa y los datos se perforan en cinta de papel. Para indicar el término de uno u otro se perfora al final de ellos la doble palabra omega la que junto con ser almacenada pone fin a la lectura. Se utiliza también en salida para poner término a dicha operación.

Los códigos correspondientes a esas funciones son:

- 67 Leer y almacenar en memoria a partir de la celda n
- 68 Perforar desde memoria el contenido de la celda n
- 69 Perforar desde memoria a partir de la celda n

Ejemplo 3. Programar el cálculo de

Programar el cálculo de

$$P(x) = a_0x^3 + a_1x^2 + a_2x + a_3$$

o lo que es lo mismo $P(x) = ((a_0x + a_1)x + a_2)x + a_3$

Distribución de memoria:

$$(1000) = x$$

$$(1002) = a_0$$

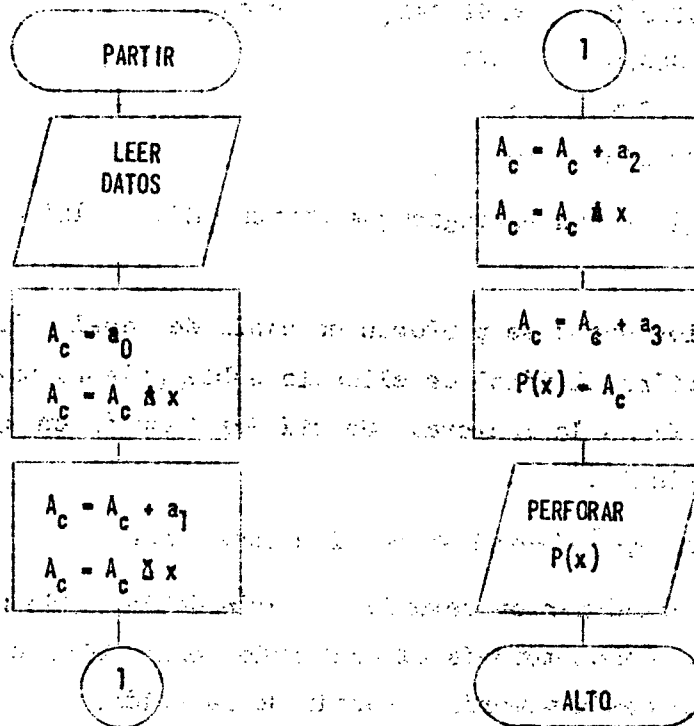
$$(1004) = a_1$$

$$(1006) = a_2$$

$$(1008) = a_3$$

$$(1010) = P(x)$$

El programa se cargará en memoria a partir de la celda 3000.

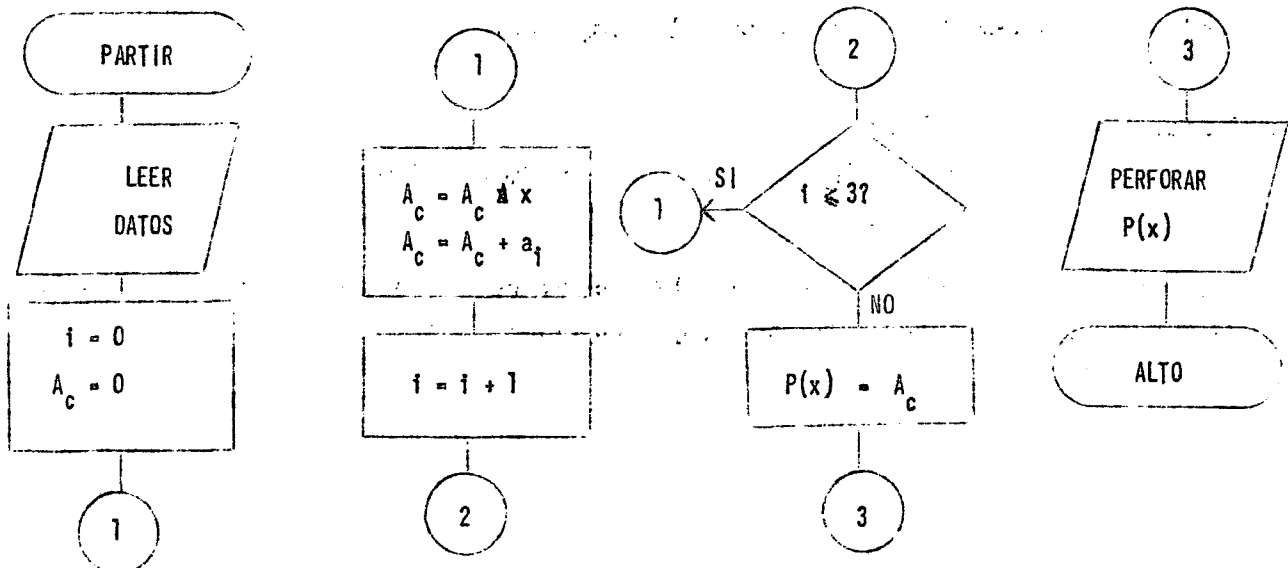


Programa codificado:

(3000)	7200000	Partir, modo punto flotante	
(3001)	1000067	Se almacenan datos a partir de celda 1000	
(3002)	1002031	$A_c = (1002)$	$A_c = a_0$
(3003)	1000037	$A_c = A_c * (1000)$	$A_c = A_c * x$
(3004)	1004035	$A_c = A_c + (1004)$	$A_c = A_c + a_1$
(3005)	1000037	$A_c = A_c * (1000)$	$A_c = A_c * x$
(3006)	1006035	$A_c = A_c + (1006)$	$A_c = A_c + a_2$
(3007)	1000037	$A_c = A_c * (1000)$	$A_c = A_c * x$
(3008)	1008035	$A_c = A_c + (1006)$	$A_c = A_c + a_3$
(3009)	1010032	$(1010) = A_c$	$P(x) = A_c$
(3010)	1010068	Perforar (1010)	
(3011)	1011068	Perforar (1011)	
(3012)	7900000	Alto	
(3013)	9999999	ω	
(3014)	9999999	ω	

Se puede observar en este programa que la instrucción 1000037 se repite tres veces sin ninguna modificación y que la operación de suma se repite también tres veces y que la única modificación que se realiza en ella es en la dirección del operando. Esto significa que ambas operaciones, multiplicación y suma podrían formar parte de un ciclo.

Se analizará el diagrama de flujo siguiente para ver cuales son los elementos que se necesitan para construir un ciclo. Se considerará el mismo problema.



En el diagrama se ha estructurado un ciclo en el que figuran: la instrucción de multiplicación por x, la instrucción de suma del coeficiente a_j y la instrucción que incrementa el índice i que es el que ha permitido en definitiva la construcción del ciclo. Dado que en el programa la única diferencia entre las instrucciones de suma es la dirección del operando, la función que debe cumplir el índice será justamente la de modificar dicha dirección. Se necesitarán además instrucciones que permitan definir el índice con un valor, incrementar el índice, comparar su valor con un parámetro y efectuar una bifurcación en el programa de acuerdo al resultado de la comparación.

El computador tiene cuatro indicadores que reflejan el resultado de una comparación: $<$, $>$, \neq , $=$. Siempre habrá dos indicadores "conectados" (puestos en ON).

Las nuevas operaciones que se utilizarán son:

Código de operación	Función
91	Se define un registro índice (REG) con el valor p
93	Se incrementa un REG con el valor p
98	Se compara el contenido de un REG con el valor p
12	Salto incondicional a la dirección n
13	Salto si el indicador = está conectado
14	Salto si el indicador \neq está conectado
15	Salto si el indicador $>$ está conectado
16	Salto si el indicador $<$ está conectado

Si se tiene una instrucción del tipo siguiente:

n i 37

la dirección efectiva del dato será:

dirección efectiva = dirección n + contenido del REG

dirección efectiva = n + (i)

Las nuevas operaciones permiten obtener el programa que figura a continuación. (Considerar que se ha guardado el valor cero en la celda 1010):

(3000)	7200000	Partir, modo punto flotante
(3001)	1000067	Se almacenan datos a partir de celda 1000
(3002)	0000191	El REG 1 se carga con 0000
(3003)	1010031	Ac = (1010) ; Ac = 0
(3004)	1000037	Ac = Ac * (1000) ; Ac = Ac*x
(3005)	1002135	Ac = Ac + (1002+REG1);Ac=Ac+a ₁
(3006)	0002193	El REG1 se incrementa en 0002
(3007)	0008198	Se compara el contenido de REG1 con 0008
(3008)	3004016	Salto a la dirección 3004 si el indicador < está conectado
(3009)	1010032	(1010) = Ac ; P(x) = Ac
(3010)	1010068	Perforar (1010)
(3011)	1011068	Perforar (1011)
(3012)	7900000	Alto
(3013)	9999999	ω
(3014)	9999999	ω

Aún cuando el programa tiene igual número de instrucciones que en la solución anterior, puede ahora servir para calcular el valor de un polinomio de cualquier grado, para ello lo único que se necesita cambiar es el valor de comparación en la instrucción almacenada en la dirección 3007.

Queda todavía por resolver una situación incómoda y es la de tener que cargar siempre el programa a partir de la celda 3000 (programación absoluta). Si no se hiciera así, al llegar a la instrucción de bifurcación, se producirá el salto a la dirección 3004, que puede contener cualquier cosa que no tenga ninguna relación con el proceso.

Lo contrario a la "programación absoluta" es la "programación relativa" que es aquella que permite cargar los programas en cualquier parte de la memoria sin que se produzca ningún problema. Se hace uso en este caso de un registro especial, el REGISTRO DE INDICE 9 que contiene siempre la dirección de la instrucción que se ejecuta, más 1. Para referirse a direcciones que están más adelante en el programa, se utiliza la fórmula:

$$p = n_2 - n_1 - 1$$

donde:

p = parámetro

n₁ = dirección de la instrucción origen

n₂ = dirección de la instrucción meta de salto

Ejemplo 4.

Se tiene en la dirección 1022 una instrucción de salto a la dirección 1025.

n₁ (1022) 0002 9 16

n₂ (1025) - - - - -

p = 1025 - 1022 - 1

p = 0002

luego la instrucción se puede representar como:

(1022) 0002 9 16

dirección efectiva = (REG9) + 0002

dirección efectiva = 1023 + 0002

dirección efectiva = 1025

Para referirse a direcciones que están más atrás en el programa se utiliza la fórmula:

$$p = 9999 - (n_1 - n_2)$$

Ejemplo 5.

Se tiene en la dirección 3008 una instrucción de salto a la dirección 3004

n₂ (3004) - - - - -

n₁ (3008) - - - - -

(3008) 3004 0 16

p = 9999 - (3008 - 3004)

p = 9999 - 4

p = 9995

luego la instrucción se puede representar como:

(3008) 9995 9 16

dirección efectiva = (REG9) + 9995

dirección efectiva = 3009 + 9995

dirección efectiva = 13004

El primer dígito se pierde pues el registro tiene capacidad para cuatro dígitos solamente; queda entonces la dirección efectiva igual a 3004 que es lo que se estaba buscando.

II. UN ENSAMBLADOR

1. Definición

Los ensambladores son los traductores de los lenguajes orientados a la máquina. Se denominan también lenguajes uno a uno porque existe una correspondencia, instrucción por instrucción, entre éstos y los lenguajes de máquina. Otro término, poco utilizado, para referirse a los ensambladores es el de autocódigo.

La ventaja que tienen los lenguajes orientados a la máquina es la de poder codificar las instrucciones a base de iniciales de palabras, abreviaturas o signos que indican la función a realizar y que son fáciles de recordar por el programador. Además, éste se despreocupa del lugar o dirección donde se almacenarán los datos y programa, pues trabaja con direcciones simbólicas. Para esto existen pseudo-instrucciones que permiten definir zonas de datos, constantes, etc., de tal manera que el ensamblador se preocupa de traducir las expresiones nemotécnicas y substituir las direcciones simbólicas por direcciones reales.

Para poder hacer la substitución de direcciones simbólicas por reales, normalmente el ensamblador necesita de dos pasadas de análisis del programa fuente. En la primera pasada asigna direcciones reales a cada dirección simbólica definida en el programa. Estas direcciones reales las asigna tomando como punto de referencia una dirección que le ha sido entregada a través de una pseudo-instrucción o de constantes propias del ensamblador. Utiliza además, un contador de direcciones cuyo contenido inicial es la dirección de referencia. Este contenido será incrementado de acuerdo al espacio de memoria ocupado por cada instrucción, constante o área de resultados generada, de tal manera que siempre el contador de direcciones contendrá la próxima ubicación disponible. Se efectúa al mismo tiempo la construcción de una tabla de símbolos que contiene los nombres simbólicos creados por el programador y las direcciones reales que le ha asignado el ensamblador. Además se realiza la revisión sintáctica de las instrucciones y en algunos casos la traducción inmediata de los códigos de operación, literales incluidos en la instrucción y todos aquellos elementos que no participan en la descripción de los operandos simbólicos.

En la segunda pasada, el ensamblador traduce los operandos simbólicos de acuerdo a la tabla de símbolos construida en la primera pasada y se hace un análisis de error global, esto es, se determinan errores que involucren la interrelación de proposiciones.

Existen casos particulares en que la tabla de símbolos puede suprimirse y corresponden a problemas que tratan los ensambladores de una pasada. Este tipo de traductores se necesitan en los sistemas de tiempo compartido (time-sharing), en los cuales varios usuarios hacen uso simultáneo del computador desde consolas individuales. La restricción que existe para que puedan funcionar los ensambladores de una pasada es que las áreas de datos y de almacenamiento deben estar definidas antes de que se haga referencia a ellas. Subsiste, sin embargo, el problema de las bifurcaciones, el cual se resuelve transfiriendo el programa generado con las referencias simbólicas sin traducir, al programa cargador o a un sistema intérprete que termine la traducción en el momento de ejecutar el programa objeto.

Hay algunos ensambladores que reconocen y traducen macroinstrucciones, es decir, instrucciones simbólicas que provocan la inclusión, en el programa generado, de subprogramas escritos en lenguaje orientado a la máquina o subprogramas que han tenido la primera pasada del ensamblador. Este hecho permite la traducción conjunta del programa principal (monitor) y de los subprogramas (subrutinas). Se realiza así el ENSAMBLADO de programas que tienen origen distinto, y esta función es la que le ha dado el nombre a este tipo de traductores.

2. El lenguaje de ensamblaje del Sistema IBM/360/370

Llamado comúnmente ASSEMBLER está formado por un conjunto de símbolos nemotécnicos que representan:

- a) Códigos de operación del lenguaje de máquina
- b) Operaciones que van a ser realizadas por el ensamblador.

El programador puede crear también macro instrucciones.

A. Hoja de codificación

La hoja de codificación tiene líneas con capacidad para ochenta caracteres lo que significa que cada línea puede ser vaciada en su totalidad en una tarjeta.

Cada línea queda dividida en dos sectores: el que corresponde a la proposición (columnas 1 a la 71) y el que corresponde a identificación o secuencia (columnas 73 a 80). Si la proposición ocupa más de 71 posiciones, se puede continuar en la línea siguiente. Para ello se especifica cualquier carácter distinto de blanco en la columna 72 y se empieza en la columna 16 de la línea siguiente. Se acepta sólo una línea de continuación y las primeras quince columnas de ella deben estar en blanco.

a) Proposición

Hay dos tipos de proposiciones y son: las instrucciones y los comentarios. Las primeras pueden consistir de uno a cuatro campos que son de izquierda a derecha:

NOMBRE
OPERACION
OPERANDOS
COMENTARIO

i) En el campo de NOMBRE puede figurar un símbolo que se utiliza para identificar la instrucción y debe cumplir las siguientes normas:

Debe estar compuesto de 8 caracteres o menos.
Debe empezar con carácter alfabético
Debe empezar en columna 1
Debe aparecer una sola vez identificando a una instrucción
No debe contener caracteres especiales ni blancos.

ii) En el campo OPERACION figura un código de operación que indica la función que se va a realizar.

Los códigos de operación válidos tienen cinco caracteres o menos
Si no existe nombre debe empezar, al menos, una posición a la derecha de la columna 1
Si existe nombre debe ir separado de éste, al menos, por un blanco
No debe contener caracteres especiales ni blancos.

Formato RR: Registro a Registro (Register to Register)

Formato RX: Registro a Memoria Indexada (Register to Indexed Storage)

Formato RS: Registro a Memoria (Register to Storage)

Formato SI: Memoria y operando inmediato (Storage and Imediate)

Formato S : Memoria (Storage)

Formato SS: Memoria a Memoria (Storage to Storage)

C. Convenciones y símbolos que se adoptarán

- a) [] lo que está encerrado entre los paréntesis [] es optativo de ser colocado.
- b) < > indica "contenido de"
- c) < >_s indica "contenido supuesto de"
- d) = símbolo de definición, significa "definido por"
- e) RUG registro de uso general
- f) RC registro de control
- g) RPF registro de punto flotante
- h) R1,R2,R3 registros de uso general, de control o de punto flotante que actúan como primer, segundo o tercer operando respectivamente
- i) X2,B2,B1 registros de uso general, índice y base respectivamente, que forman parte de la dirección del segundo (primer) operando o del segundo (primer) operando mismo
- j) D2,D1 valor decimal correspondiente a desplazamiento. Este forma parte de la dirección del segundo (primer) operando o del segundo (primer) operando mismo
- k) L longitud del primer y segundo operando
- l) L1,L2 longitud del primer y segundo operando respectivamente
- m) I2 segundo operando, inmediato, esto es, está en la misma instrucción
- n) M1,M3 grupo de cuatro bits utilizado como máscara. Puede ser primer o tercer operando respectivamente
- ñ) Las letras mayúsculas y puntuación representan información que debe ser codificada tal como aparece.
- o) Las letras minúsculas representan información que debe ser proporcionada por el programador.

D. Alineamiento

Se debe tener presente que las direcciones en memoria corresponden a la dirección de un solo byte. Algunas instrucciones que direccionan un byte operan siempre con ese byte y un número fijo de bytes que le sigue. Este número fijo puede ser uno, formando un operando de dos bytes, media palabra (half-word), tres, formando un operando de cuatro bytes, una palabra (full-word) o siete, formando un operando de ocho bytes, doble palabra (double-word). En otras instrucciones debe especificarse la longitud, en bytes, del operando.

El alineamiento (boundary) es una restricción que debe cumplirse en la programación de operandos de longitud fija. Es así como la dirección del byte del extremo izquierdo de un campo fijo de dos bytes (half-word), cuatro bytes (full-word) u ocho bytes (double-word) debe ser múltiplo de dos, cuatro u ocho respectivamente.

E. Definición de constantes y áreas**a) Formato de la pseudo-instrucción DC (Define Constant)**

nombre DC d t L n 'c'

donde:

nombre: identifica al byte del extremo izquierdo del campo ocupado por la constante

DC : código de operación

d : factor de repetición o número de veces que se define la constante.
Puede ser omitido

t : tipo de constante. Debe aparecer

Ln : modificador de longitud donde n representa el número de bytes que tendrá la constante, y puede ser un valor decimal sin signo, o una expresión absoluta positiva encerrada entre paréntesis. Puede omitirse

'c' : constante que se desea generar en memoria.

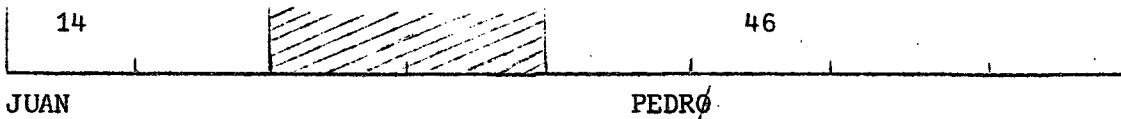
b) Constante de punto fijo F y H

Estas constantes si son definidas sin modificador de longitud quedan con alineamiento de palabra y de media palabra respectivamente. En caso contrario, no se tiene seguridad de que la constante generada quede en el alineamiento correspondiente.

Ejemplo 1.

JUAN DC H'14'
PEDRØ DC F'46'

quedan en memoria como sigue (se supone que JUAN se inicia en una palabra)

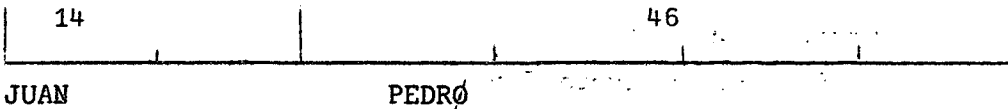


los dos bytes achurados se saltaron para cumplir con el alineamiento que le corresponde a PEDRØ (el primer bit en cada constante representada en binario, indica el signo, 0 si es positivo, 1 si es negativo).

Ejemplo 2.

JUAN DC H'14'
PEDRØ DC FL4'46'

quedan en memoria en la siguiente forma:

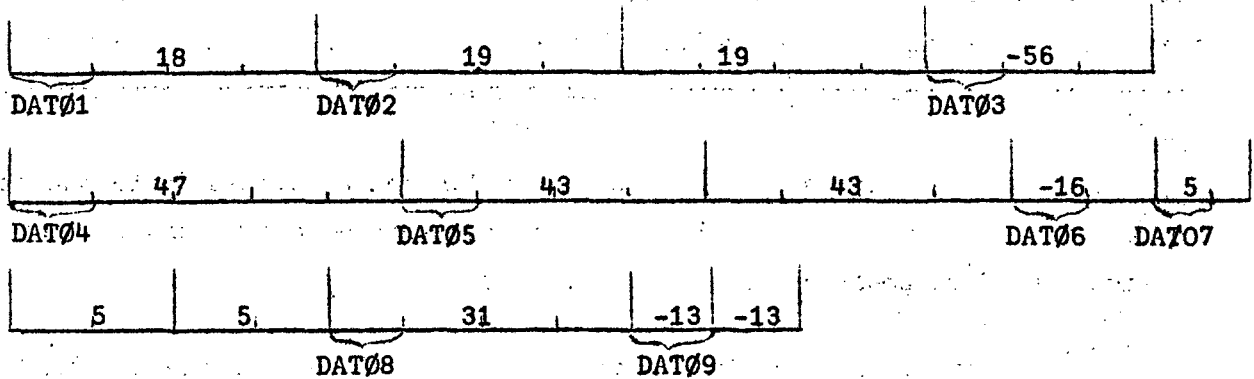


Se pierde el alineamiento por existir el modificador de longitud L4.

Ejemplo 3.

1^a. columna
 DATØ1 DC F'18'
 DATØ2 DC 2F'19'
 DATØ3 DC FL3'-56'
 DATØ4 DC FL5'47'
 DATØ5 DC 2FL4'+43'
 DATØ6 DC H'-16'
 DATØ7 DC 3H'5'
 DATØ8 DC HL4'+31'
 DATØ9 DC 2HL1'-13'

Estas constantes como parte de un programa, quedan definidas en memoria, una a continuación de la otra.



c) Formato de la pseudo-instrucción DS (Define Storage)

Nombre DS d t Ln

Como se observa en el formato, difiere de la proposición DC, sólo en que no tiene el operando 'c', dado que DS permite reservar un área de memoria incluso sin borrar lo que existía en ella anteriormente.

F. Direccionamiento y registros de uso general

El sistema IBM/360/370 tiene un juego de 16 registros (0-15) que sirven como:

- Registros índices
- Registros para control de programas
- Acumuladores para aritmética de punto fijo
- Acumuladores para aritmética de direcciones

por este motivo se denominan "registros de uso general" (RUG).

La capacidad de cada registro es una palabra (full-word) o lo que es lo mismo, 32 bits (0-31 de izquierda a derecha). Cada bit representa un coeficiente de una potencia de dos aumentando ésta de valor, de derecha a izquierda. El bit 0 o bit de orden superior, si se considera el valor de la potencia de dos, representa el signo en las operaciones algebraicas (0 indica signo positivo, 1 indica signo negativo).

En la aritmética de direcciones, se utilizan sólo los 24 bits de orden inferior. Esto permite direccionar hasta 16 777 216 bytes.

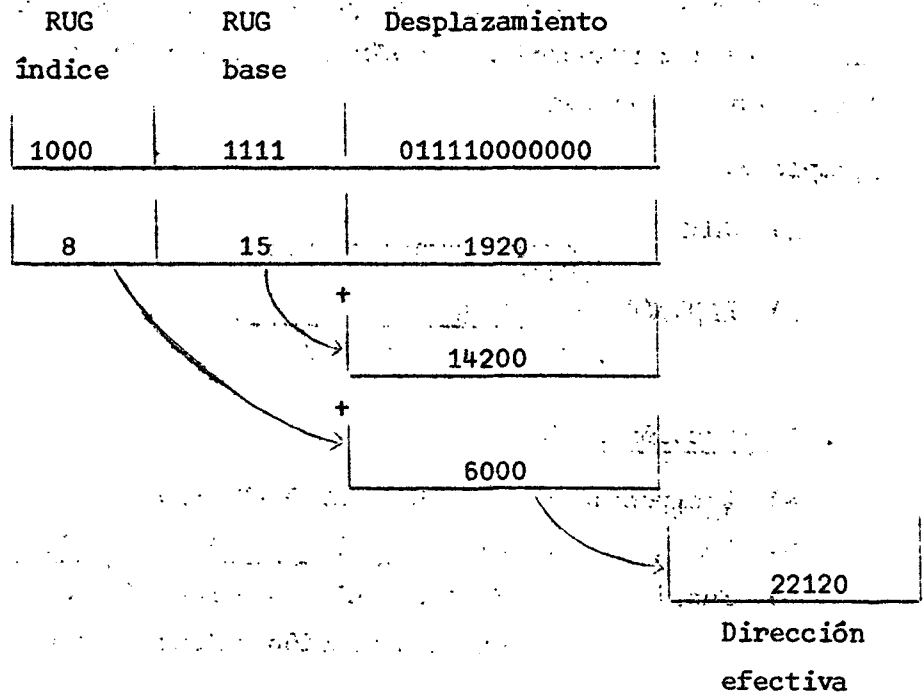
Un campo de cuatro bits en la instrucción permite especificar uno de los 16 registros. Los 24 bits de orden inferior de ese registro contienen una dirección, conocida como dirección base (B). Además, la instrucción contiene un campo de 12 bits cuyo contenido se conoce como desplazamiento (D). Este desplazamiento, al ser sumado a la dirección base, permite direccionar posiciones en memoria de hasta 4095 bytes, más allá de la dirección base.

Muchas instrucciones tienen otro campo de cuatro bits para designar un registro de uso general, denominado registro índice (X). En estas instrucciones la dirección efectiva se obtiene sumando los contenidos de los registros B y X y el desplazamiento D.

$$\text{Dirección efectiva} = \langle \text{RUG B} \rangle + \langle \text{RUG X} \rangle + D$$

Ejemplo 4.

Sea RUG X el RUG 8 y su contenido 6000
 RUG B EL RUG 15 y su contenido 14200
 Desplazamiento D 1920
 la dirección efectiva será 22120



Observación: El RUG 0 no se puede utilizar como registro base o índice. La aparición del 0 en esos campos es interpretado por el sistema como ausencia de registro.

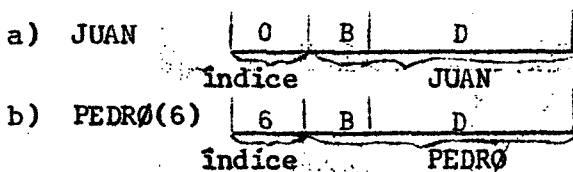
3. Aritmética de punto fijo

En todas las operaciones que se verán en adelante, es necesario tener presente que la información obtenida de un almacenamiento magnético permanece en él sin alteración. Por el contrario, la información que se carga en un almacenamiento magnético borra lo que había antes en él.

Las instrucciones de este capítulo corresponden al grupo de Aritmética de Punto Fijo, sin embargo, en algunos casos será necesario ver instrucciones de otros grupos, lo cual se indicará en la instrucción.

Las direcciones de almacenamiento son representadas generalmente en assembler por símbolos. La dirección efectiva que corresponde a ese símbolo es desglosada por el compilador en: un contenido de registro base y un desplazamiento. En aquellas instrucciones en que puede especificarse registro índice, si se indica sólo el símbolo, el compilador supone que no se hará uso de registro índice, y coloca 0 en la instrucción de máquina. Si se desea indicar registro índice, se especifica a continuación del símbolo, encerrado entre paréntesis (), el número del registro utilizado.

Ejemplo 5.



3.1 Instrucción LOAD

a) Instrucción: L R1, D2(X2, B2)

b) Formato : RX

L	R1	X2	B2	D2
---	----	----	----	----

c) Función : Se carga el RUG especificado en R1 con la palabra contenida en la dirección calculada con D2, X2 y B2.

Ejemplo 6.

Instrucciones válidas y no válidas.

Válidas:

i) L 5,40(7,15)

función, en símbolos: <RUG5> = <40 + <RUG7> + <RUG15>>

Explicación: el contenido del RUG 5 se define por el contenido de la dirección calculada como 40, más contenido de RUG7, más contenido de RUG15.

ii) L 7,JUAN

Se carga el RUG7 con la palabra contenida en la dirección JUAN.

iii) L 2,PEDRØ+4

Se carga el RUG2 con la palabra contenida en la dirección PEDRO+4.

iv) L 1,DIEGØ+2(3)

Se carga el RUG 1 con la palabra contenida en la dirección DIEGØ + 2 + <RUG2>

No válidas:

v) L 15,4096(8,14)

Error en el desplazamiento que, como máximo puede ser 4095.

vi) L 8,AREA(2,15)

Error porque se obliga al compilador a considerar el símbolo AREA como desplazamiento. El compilador rechaza esta instrucción porque todo símbolo de dirección tiene un valor superior a 4095.

vii) L 6,DATØ(,13) o L 6,DATØ(0,13)

Igual motivo al de vi).

viii) Se producirá error por especificación si la dirección no corresponde a un alineamiento de palabra.

3.2. Instrucción LOAD HALF

a) Instrucción: LH R1,D2(X2,B2)

b) Formato : RX | LH | R1 | X2 | B2 | D2

c) Función : Se carga el RUG especificado como primer operando con la media palabra ubicada en la dirección calculada con D2, X2 y B2.

El procedimiento es el que se indica. La media palabra se lleva a la unidad aritmética donde es expandida a palabra completa propagando el bit de signo a través de las 16 posiciones de orden superior. El resultado se carga en el RUG. (Ver ejemplo 7).

3.3. Instrucción LOAD REGISTER

- a) Instrucción: LR RI, R2
- b) Formato : RR

LR	R1	R2
----	----	----
- c) Función : Se carga el RUG especificado como primer operando con el contenido del RUG indicado como segundo operando, (ver ejemplo 7).

Ejemplo 7.

DATØ1	DC	F' -18'
ALFA	DC	H' 14'
BETA	DC	H' -35'
	L	5, DATØ1
	LH	6, ALFA
	LH	7, BETA
	LR	1, 5
	LR	2, 6

Explicación: Una vez definidas las constantes DATØ1, ALFA y BETA se cargan en los RUG 5, 6 y 7 respectivamente. Para efectuar la operación de carga, se utilizan las instrucciones LOAD y LOAD HALF. A continuación se cargan los RUG 1 y 2 con los contenidos de los RUG 5 y 6 utilizando la instrucción LOAD REGISTER.

3.4. Instrucción LOAD ADDRESS (Instrucción Lógica)

- a) Instrucción: LA R1, D2(X2, B2)
- b) Formato : RX

LA	R1	X2	B2	D2
----	----	----	----	----
- c) Función : Se carga en el RUG especificado en R1 la dirección dada por D2, X2 y B2. La dirección ocupa los bits 8-31 del RUG, los bits 0-7 se ponen en cero.

Ejemplo 8.

```

LA    3,PEDRØ
LA    4,15(0,4)
LA    5,JUAN(6)
LA    0,500

```

Explicación: Se carga el RUG 3 con la dirección PEDRØ. Se carga el RUG4 con la dirección 15+ <RUG4>. Se carga el RUG5 con la dirección JUAN + <RUG6>. Se carga el RUG0 con la dirección 500. Al ser traducida esta última instrucción queda el valor 500 como desplazamiento y 0 en registro índice y registro base.

3.5. Instrucción STORE

- a) Instrucción: ST R1,D2(X2,B2)
- b) Formato : RX

ST	R1	X2	B2	D2
----	----	----	----	----
- c) Función : El contenido del RUG especificado como primer operando se almacena en la dirección calculada con D2,X2 y B2. Esa dirección debe estar en alineamiento de palabra.

Ejemplo 9.

```

CTE1  DC  F'52'
AREA  DS  F
      L   7,CTE1
      ST  7,AREA

```

Explicación: Se carga el RUG7 con el valor 52 y luego se almacena ese contenido en la dirección AREA, ocupando una palabra.

3.6. Instrucción STORE HALF

- a) Instrucción: STH R1,D2(X2,B2)
- b) Formato : RX

STH	R1	X2	B2	D2
-----	----	----	----	----
- c) Función : El contenido de la mitad inferior del RUG especificado como primer operando se almacena en la dirección calculada con D2,X2 y B2. Ocupa en memoria dos bytes. La mitad superior del RUG no interviene en la operación.

Ejemplo 10.

```

    STH     5,AREA
    STH     6,AREA+2
  
```

Explicación: Se almacena el contenido de la mitad inferior del RUG5 en la dirección AREA, y el del RUG6 en la dirección AREA+2.

3.7. Instrucción ADD

- a) Instrucción: A R1,D2(X2,B2)
- b) Formato : RX | A | R1 | X2 | B2 | D2 |
- c) Función : Al contenido del RUG especificado en R1, se le suma la palabra contenida en la dirección entregada por D2,X2 y B2. El resultado queda en el RUG especificado en R1.

Todas las instrucciones de tipo aritmético generan un Código de Condición (CC) de cuatro posibles, el cual queda registrado en la Palabra de Estado del Programa (Program Status Word-PSW) vigente, en los bits 34 y 35, Códigos de Condición generados por la instrucción ADD

<u>CC</u>	<u>Resultado</u>
00	cero 0
01	menor que cero <0
10	mayor que cero >0
11	desborde 0F

El desborde (overflow) se produce cuando el resultado excede la capacidad del registro. Se produce interrupción (interrupt) de programa si el bit de desborde de punto fijo está en uno.

Ejemplo 11.

```

    DATØA     DC     F'18'
    DATØB     DC     F'52'
    DATØC     DC     F'-31'
               L     6,DATØA
               A     6,DATØB
               A     6,DATØC
  
```

Explicación: Se carga el RUG6 con el contenido de DATØA que es la constante 18. A continuación se suma al contenido del RUG6 el contenido de DATØB, queda como resultado en RUG6 el valor 70. Finalmente se suma a ese resultado el contenido de DATØC. Queda en RUG6 el valor 39.

3.8. Instrucción ADD REGISTER

- a) Instrucción: AR R1,R2
- b) Formato : RR

AR	R1	R2
----	----	----
- c) Función : Se suma al contenido del RUG especificado en R1 el contenido del RUG especificado en R2. El resultado queda en el RUG indicado en R1. Se generan los mismos CC que genera la instrucción ADD.

3.9. Instrucción ADD HALF

- a) Instrucción: AH R1,D2(X2,B2)
- b) Formato : RX

AH	R1	X2	B2	D2
----	----	----	----	----
- c) Función : Se suma al contenido del RUG especificado en R1, la media palabra contenida en la dirección calculada con D2, X2 y B2. Para efectuar la suma, se expande la media palabra a palabra completa mediante la propagación del bit de signo a través de las 16 posiciones de orden superior. La expansión se realiza en la Unidad Aritmética.

3.10. Instrucción SUBTRACT

- a) Instrucción: S R1,D2(X2,B2)
- b) Formato : RX

S	R1	X2	B2	D2
---	----	----	----	----
- c) Función : Se resta al contenido del RUG especificado en R1 la palabra contenida en la dirección calculada con D2, X2 y B2. El resultado queda en el RUG especificado en R1.

3.11. Instrucción SUBTRACT REGISTER

- a) Instrucción: SR R1,R2
- b) Formato : RR

SR	R1	R2
----	----	----
- c) Función : Se resta al contenido del RUG especificado en R1 el contenido del RUG especificado en R2. El resultado queda en el RUG indicado en R1.

3.12. Instrucción SUBTRACT HALF

- a) Instrucción: SH R1, D2(X2, B2)
- b) Formato : RX

SH	R1	X2	B2	D2
----	----	----	----	----
- c) Función : Se resta al contenido del RUG especificado en R1 la media palabra contenida en la dirección calculada con D2, X2 y B2. Para efectuar la resta, se expande la media palabra a palabra completa mediante la propagación del bit de signo a través de las 16 posiciones de orden superior. La expansión se realiza en la Unidad Aritmética.

3.13. Instrucción ADD LOGICAL REGISTER

- a) Instrucción: ALR R1, R2
- b) Formato : RR

ALR	R1	R2
-----	----	----
- c) Función : Se suma al contenido del RUG especificado en R1 el contenido del RUG indicado en R2. Se suman los 32 bits de ambos operandos sin que haya cambio posterior en el bit de signo del resultado. Si hay un desborde desde la posición del signo se registra en el Código de Condición.

<u>Código de Condición</u>	<u>Resultado</u>
0	cero (sin desborde)
1	distinto de cero (sin desborde)
2	cero (con desborde)
3	distinto de cero (con desborde)

3.14. Instrucción ADD LOGICAL

- a) Instrucción: AL R1, D2(X2, B2)
- b) Formato : RX

AL	R1	X2	B2	D2
----	----	----	----	----
- c) Función : Se suma al contenido del RUG especificado en R1 la palabra contenida en la dirección calculada con D2, X2 y B2. Se generan los mismos códigos que con ALR.

3.15. Instrucción SUBTRACT LOGICAL REGISTER

- a) Instrucción: SLR R1,R2
- b) Formato : RR

SLR	R1	R2
-----	----	----
- c) Función : Se resta al contenido del RUG especificado en R1, el contenido del RUG indicado en R2. Participan los 32 bits de ambos operandos, sin que haya cambio posterior en el bit de signo del resultado.

<u>Código de Condición</u>	<u>Resultado</u>
0	---
1	distinto de cero (sin desborde)
2	cero (con desborde)
3	distinto de cero (con desborde)

3.16. Instrucción SUBTRACT LOGICAL

- a) Instrucción: SL R1,D2(X2,B2)
- b) Formato : RX

SL	R1	X2	B2	D2
----	----	----	----	----
- c) Función : Se resta al contenido del RUG especificado en R1 la palabra contenida en la dirección calculada con D2,X2 y B2. Se generan los mismos códigos que con SLR.

Ejemplo 12.

A	DC	F'57'
B	DC	F'415'
C	DC	F'-35'
Z	DS	F
L	5,A	<RUG5> = 57
L	7,B	<RUG7> = 415
ALR	5,5	<RUG5> = 114
AL	5,C	<RUG5> = 79
SLR	7,5	<RUG7> = 336
SL	7,A	<RUG7> = 279
ST	7,Z	

El resultado sería el mismo, si se hubieran utilizado las instrucciones AR, A, SR y S.

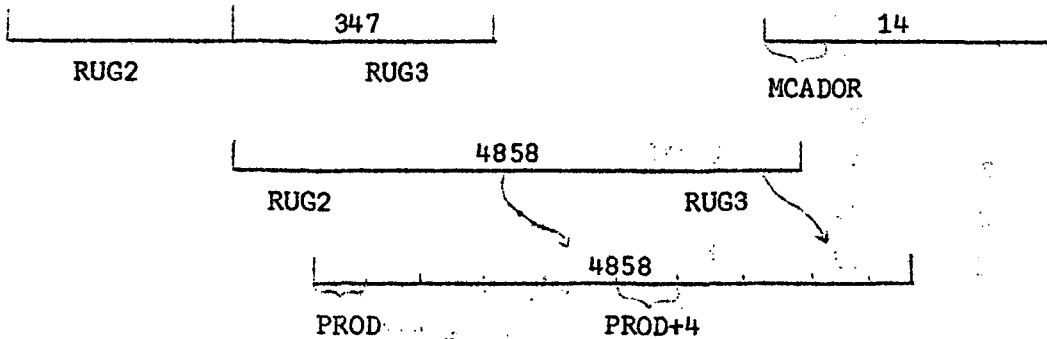
3.17. Instrucción MULTIPLY

- a) Instrucción: M R1,D2(X2,B2)
- b) Formato : RX

M	R1	X2	B2	D2
---	----	----	----	----
- c) Función : El producto del multiplicando (1er operando) y el multiplicador (2ºoperando) reemplaza al primero. Como ambos operandos tienen 32 bits, el producto será un entero de 64 bits que ocupa un par de RUG, PAR e IMPAR siguiente. Debido a que el producto reemplaza al multiplicando, en el campo R1 se especifica el RUG PAR. El contenido de este RUG se ignora, a menos que contenga al multiplicador.

Ejemplo 13.

MCANDØ	DC	F' 347'
MCADØR	DC	F' 14'
PRØD	DS	2F
	L	3,MCANDØ
	M	2,MCADØR
	ST	2,PRØD
	ST	3,PRØD+4



3.18. Instrucción MULTIPLY REGISTER

- a) Instrucción: MR R1,R2
- b) Formato : RR

MR	R1	R2
----	----	----
- c) Función : Se realiza la misma función que efectúa la instrucción Multiply. El multiplicador (2ºoperando) está contenido en un RUG.

Ejemplo 14.

MCANDØ	DC	F' 347'
MCADØR	DC	F' 14'
PRØD	DS	2F
	L	3, MCANDØ
	L	7, MCADØR
	MR	2, 7
	ST	2, PRØD
	ST	3, PRØD+4

Ejemplo 15.

MCANDØ	DC	F' 347'
MCADØR	DC	F' 14'
PRØD	DS	2F
	L	3, MCANDØ
	L	2, MCADØR
	MR	2, 2
	ST	2, PRØD
	ST	3, PRØD+4

En este último ejemplo se utilizó el RUG2 para contener el multiplicador aprovechando que el contenido del RUG PAR se ignora "a menos que contenga el multiplicador".

3.19. MULTIPLY HALF

- a) Instrucción: MH R1, D2(X2, B2)
- b) Formato : RX

MH	R1	X2	B2	D2
----	----	----	----	----
- c) Función : El producto del multiplicando (1er. operando) y el multiplicador (2ºoperando) reemplaza al primero. Antes de la multiplicación, el multiplicador se expande de media palabra a palabra completa mediante la propagación del bit de signo a través de las 16 posiciones de orden superior. Una vez efectuada la multiplicación, el multiplicando es reemplazado por los 32 bits de orden inferior del producto. El RUG especificado en R1 puede ser par o impar.

Ejemplo 16.

MCANDØ	DC	H' 347'
MCADØR	DC	H' 14'
PRØD	DS	F
	LH	5,MCANDØ
	MH	5,MCADØR
	ST	5,PRØD

3.20. Instrucción DIVIDE

a) Instrucción: D R1,D2(X2,B2)

b) Formato	:	R X	D	R1	X2	B2	D2
------------	---	-----	---	----	----	----	----

c) Función : El dividendo (1er operando) es dividido por el divisor (2ºoperando) y reemplazado por el residuo y cuociente. El dividendo ocupa dos RUG, PAR e IMPAR siguiente. El divisor es una palabra contenida en la dirección calculada con D2, X2 y B2. El residuo ocupará el RUG PAR especificado en R1, y el cuociente el RUG IMPAR siguiente, ambos con sus respectivos signos.

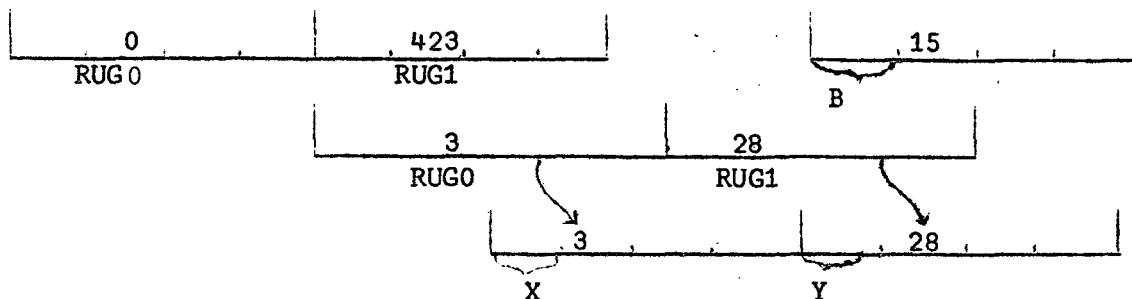
La división por cero, produce una interrupción de programa. No se realiza la división y los operandos no cambian.

Ejemplo 17,

Se desea dividir 423:15

A	DC	F'423'
B	DC	F'15'
X	DS	F
Y	DS	F
	SR	0,0
	L	1,A
	D	0,B
	ST	0,X
	ST	1,Y

Las áreas X e Y se reservan para el residuo y cuociente respectivamente. El RUG0 se deja en cero mediante la instrucción SR, con el objeto de borrar todo dato que hubiera de problemas anteriores. Esta operación debe hacerse porque la instrucción de división considera el dividendo como contenido en los dos RUG, PAR e IMPAR siguiente.



3.21 Instrucción DIVIDE REGISTER

- a) Instrucción: DR R1,R2
- b) Formato : RR | DR | R1 | R2 |
- c) Función : Se realiza la misma función que efectúa la instrucción DIVIDE. El divisor (2ºoperando) está contenido en un RUG.

Ejemplo 18.

El mismo ejemplo 17 utilizando LA y DR.

X	DS	F
Y	DS	F
	LA	3,423
	LA	7,15
	SR	2,2
	DR	2,7
	ST	2,X
	ST	3,Y

Se cargan los RUG 3 y 7 con los valores 423 y 15 utilizando la instrucción Load Address. El RUG2 se deja en cero para borrar resultados anteriores. El residuo que queda en RUG2 se almacena en X, el cuociente que queda en RUG3 se almacena en Y.

Ejemplo 19.

Cálculo de VALOR=2489 + 3% de 2489, se simplifica si se coloca VALOR= $\frac{2489*103}{100}$

VALØR	DS	F
CIEN	DC	F'100'
CØNSTA	DC	F'50'
CØNSTB	DC	F'2489'
CØNSTC	DC	F'103'
	L	5,CØNSTB
	M	4,CØNSTC
	A	5,CØNSTA
	D	4,CIEN
	ST	5,VALØR

Para que se efectúe un redondeo al entero superior, en el resultado final, es necesario sumar, antes de realizar la división, la constante CØNSTA de valor 50.

3.22. Instrucción BRANCH ON CONDITION (Instrucción de bifurcación)

- a) Instrucción: BC M1,D2(X2,B2)
- b) Formato : RX

BC	M1	X2	B2	D2
----	----	----	----	----
- c) Función : Si se cumple el CC correspondiente a la máscara M1, se ejecuta la instrucción que está en la dirección dada por D2,X2 y B2. En caso contrario, se sigue la secuencia normal de ejecución, esto es, se ejecuta la instrucción que viene a continuación.

El "branch" es una interrupción de la secuencia normal de ejecución de un programa, para continuarla en otro punto de él, anterior o posterior al punto de interrupción. También se denominan "saltos" en el programa, porque permiten saltarse una o más instrucciones hacia adelante o atrás.

El branch se realiza normalmente de acuerdo a una condición producida anteriormente, y de ahí el nombre de la instrucción, branch on condition. La mayoría de las instrucciones genera un Código de Condición (CC) de cuatro posibles, el cual queda establecido en la Palabra de Estado del Programa vigente (PSW) en los bits 34 y 35. La instrucción consulta si se ha producido un CC determinado y si ello ha ocurrido, se efectúa el salto.

d) Relación existente entre las máscaras y los Códigos de Condición

M1 (decimal)	M1 (binario)	CC (decimal)	CC (binario)
8	1000	0	00
4	0100	1	01
2	0010	2	10
1	0001	3	11

las anteriores son las máscaras fundamentales que, combinadas, producen lo siguiente:

M1 (decimal)	M1 (binario)	CC (decimal)	CC (binario)
3	0011	2 ó 3	10 ó 11
5	0101	1 ó 3	01 ó 11
6	0110	1 ó 2	01 ó 10
7	0111	1 ó 2 ó 3	01 ó 10 ó 11
9	1001	0 ó 3	00 ó 11
10	1010	0 ó 2	00 ó 10
11	1011	0 ó 2 ó 3	00 ó 10 ó 11
12	1100	0 ó 1	00 ó 01
13	1101	0 ó 1 ó 3	00 ó 01 ó 11
14	1110	0 ó 1 ó 2	00 ó 01 ó 10
15	1111	0 ó 1 ó 2 ó 3	00 ó 01 ó 10 ó 11
0	0000	-----	-----

La máscara 15 corresponde a BRANCH (salto) incondicional dado que cualquier Código de Condición que se produzca se efectúa siempre el salto.

3.23. Instrucción BRANCH ON CONDITION REGISTER (Instrucción de bifurcación)

- a) Instrucción: BCR M1,R2
- b) Formato : RR | BCR | M1 | R2 |
- c) Función : Si se cumple el CC correspondiente a la máscara M1, se ejecuta la instrucción que está en la dirección contenida en el RUG especificado en R2.

3.24. Instrucciones de "Comparación Algebraica"

A. Instrucción COMPARE ALGEBRAIC REGISTER

- a) Instrucción: CR R1,R2
 b) Formato : RR

CR	R1	R2
----	----	----

B. Instrucción COMPARE ALGEBRAIC

- a) Instrucción: C R1,D2(X2,B2)
 b) Formato : RX

C	R1	X2	B2	D2
---	----	----	----	----

C. Instrucción COMPARE HALFWORD ALGEBRAIC

- a) Instrucción: CH R1,D2(X2,B2)
 b) Formato : RX

CH	R1	X2	B2	D2
----	----	----	----	----

 c) Función : Se compara el primer operando con el segundo y el resultado determina el Código de Condición. En el caso de media palabra, el segundo operando, que es la media palabra, se expande a palabra completa antes de la comparación.

<u>Resultado</u>	<u>Código de Condición</u>
(OP = operando)	
OP1 = OP2	00
OP1 < OP2	01
OP1 > OP2	10

Ejemplo 20.

```

DATA   DC   H'425'
HALF   DS   H
        LH   3,HALF
        CH   3,DATA
        BC   8,ØUT
        BC   15,LEER
  
```

En el ejemplo se supone que a la dirección HALF están llegando datos que son cargados en el RUG3 y comparados con 425. La instrucción siguiente a la comparación indica que si se produce el CC correspondiente a la máscara 8, debe saltarse a la dirección ØUT. Ese código es 00 que en el caso de comparación indica que los operandos son iguales. Si no se efectúa el salto, se toma la instrucción siguiente que indica un branch INCONDICIONAL a la dirección LEER.

3.25. Instrucción LOAD MULTIPLE

- a) Instrucción: LM R1,R3,D2(B2)
- b) Formato : RS

LM	R1	R3	B2	D2
----	----	----	----	----
- c) Función : Se cargan los RUGs, desde aquel especificado en R1, hasta el indicado en R3. La información que se carga corresponde a las palabras que están a partir de la dirección dada por D2 y B2.

Ejemplo 21.

DAT1	DC	F'18'
DAT2	DC	F'35'
DAT3	DC	F'-45'
DAT4	DC	F'106'
DAT5	DC	F'-68'
	LM	3,6,DAT1

Se cargan los RUG 3,4,5 y 6 con los datos 18,35,-45 y 106 respectivamente. Notar que basta dar la dirección inicial de la lista de información.

d) Casos especiales:

i) El registro especificado en R1 es mayor que el RUG especificado en R3. Debe considerarse entonces que los RUG forman un ciclo cerrado. Desde el RUG indicado en R1 se inicia la carga hasta encontrar el RUG especificado en R3, siempre en forma ascendente.

Ejemplo 22.

LM 14,2,DATØ

Se cargan los registros 14,15,0,1 y 2.

ii) El registro especificado en R1 es el mismo indicado en R3. En este caso la instrucción tiene la misma función que LOAD, dado que se carga un solo registro.

Ejemplo 23.

LM 8,8,DATØ

Se carga el RUG8 con lo que hay en DATØ.

3.26. Instrucción STORE MULTIPLE

- a) Instrucción: STM R1,R3,D2(B2)
- b) Formato : RS

STM	R1	R3	B2	D2
-----	----	----	----	----
- c) Función : El contenido de los RUGs, desde aquel especificado en R1 hasta el indicado en R3, se almacena en memoria a partir de la dirección dada por D2 y B2.

3.27. Pseudo-Instrucción EQU (Instrucción para el ensamblador, no genera instrucción de máquina)

- a) Formato : Nombre EQU operando
- b) Función : Se asigna al símbolo especificado en nombre, el valor que tiene el operando.

Ejemplo 24.

```
MENOR EQU 4
```

se asigna al símbolo MENOR el valor 4, esto significa que donde aparezca el símbolo MENOR se interpretará como 4. Así se puede tener:

- i) AR MENOR,MENOR que equivale a
AR 4,4
- ii) BC MENOR,OUT que equivale a
BC 4,OUT
- iii) BC 8,NEXT+MENOR que equivale a
BC 8,NEXT+4

3.28. Instrucción LOAD NEGATIVE REGISTER

- a) Instrucción: LNR R1,R2
- b) Formato : RR

LNR	R1	R2
-----	----	----
- c) Función : El contenido del RUG especificado en R2 se carga con signo negativo en el RUG indicado en R1. Los números negativos no sufren alteración. El número cero queda siempre positivo. Se genera CC de acuerdo al resultado.

<u>CC</u>	<u>Resultado</u>
00	Cero
01	Menor que cero

3.29. Instrucción LOAD POSITIVE REGISTER

- a) Instrucción: LPR R1,R2
 b) Formato : RR

LPR	R1	R2
-----	----	----

 c) Función : El contenido del RUG especificado en R2 se carga con signo positivo en el RUG indicado en R1. Los números positivos no sufren alteración. Se genera CC de acuerdo al resultado.

<u>CC</u>	<u>Resultado</u>
00	cero
10	mayor que cero
11	desborde

3.30. Instrucción LOAD COMPLEMENT

- a) Instrucción: LCR R1,R2
 b) Formato : RR

LCR	R1	R2
-----	----	----

 c) Función : El contenido del RUG especificado en R2 se carga con signo contrario en el RUG indicado en R1. El número cero queda siempre positivo. Se genera CC de acuerdo al resultado.

<u>CC</u>	<u>Resultado</u>
00	cero
01	menor que cero
10	mayor que cero
11	desborde

3.31. LOAD AND TEST

- a) Instrucción: LTR R1,R2
 b) Formato : RR

LTR	R1	R2
-----	----	----

 c) Función : El contenido del RUG especificado en R2 se carga en el RUG indicado en R1. El signo y la magnitud determinan el CC. El segundo operando no sufre alteración.

<u>CC</u>	<u>Resultado</u>
00	cero
01	menor que cero
10	mayor que cero

Cuando se especifica el mismo registro en R1 y R2, la operación equivale a una prueba sin movimiento de datos.

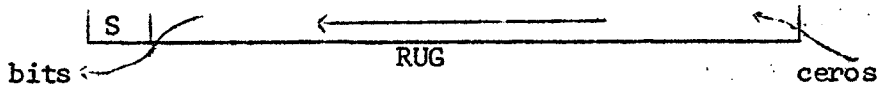
3.32. Instrucción SHIFT LEFT SINGLE

a) Instrucción: SLA R1,D2(B2)

b) Formato : RS | SLA | R1 | B2 | D2

c) Función : El contenido del RUG especificado en R1 se desplaza a la izquierda, el número de posiciones indicado por los seis bits de orden inferior de la representación binaria de la dirección dada por D2 y B2. El resto de la dirección se ignora.

El signo de la información que se desplaza permanece sin cambio. Los lugares vacantes se llenan con ceros. Si se pierde algún bit distinto del signo se produce desborde.



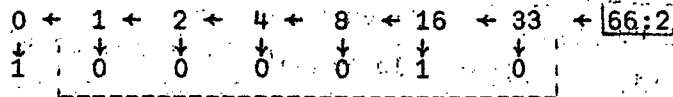
Ejemplo 25.

UNØ	DC	F'1'
RES	DS	F
	L	5,UNØ
	SLA	5,66
	ST	5,RES

Al ser cargado en RUG5 el contenido de UNØ, queda lo siguiente:

0 | 000001 | = RUG5

La instrucción SLA 5,66 indica que la información del RUG5 debe desplazarse a la izquierda el número de posiciones indicado por los 6 bits de orden inferior de la dirección 66, esto es:




los 6 bits de orden inferior indican que la información se desplaza dos lugares, por lo tanto,

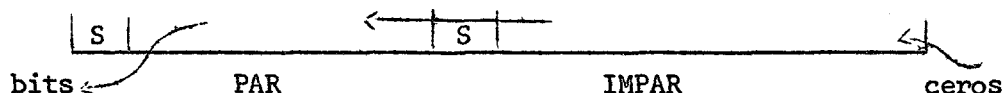
0 | 000100 | = RUG5

el valor resultante en RUG5 es 4.


Desplazar a la izquierda una posición corresponde a multiplicar por 2.
Desplazar n posiciones corresponde a multiplicar por 2^n .

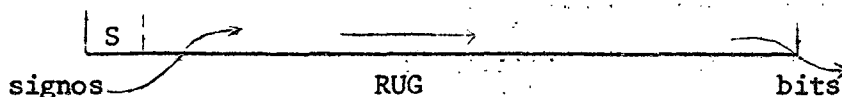
3.33. Instrucción SHIFT LEFT DOUBLE

- a) Instrucción: SLDA R1,D2(B2)
- b) Formato : RS | SLDA | R1 |  | B2 | D2 |
- c) Función : El contenido de los RUG PAR e IMPAR siguiente especificados por el primer operando R1 es desplazado a la izquierda el número de posiciones indicado por los seis bits de orden inferior, de la representación binaria de la dirección dada por D2, y B2. El resto de la dirección se ignora. En R1 se especifica un RUG PAR, lo contrario produce una interrupción de programa por error de especificación. El signo del RUG IMPAR se desplaza como parte de la información, el del RUG PAR permanece sin cambio. Los lugares vacantes se llenan con ceros. Si se pierde algún bit distinto del signo se produce desborde.



3.34. Instrucción SHIFT RIGHT SINGLE

- a) Instrucción: SRA R1,D2(B2)
- b) Formato : RS | SRA | R1 |  | B2 | D2 |
- c) Función : El contenido del RUG especificado en R1 se desplaza a la derecha el número de posiciones indicado por los seis bits de orden inferior de la representación binaria de la dirección dada por D2, y B2. El resto de la dirección se ignora. El signo de la información que se desplaza permanece sin cambio. Los lugares vacantes se llenan con bits iguales al signo.



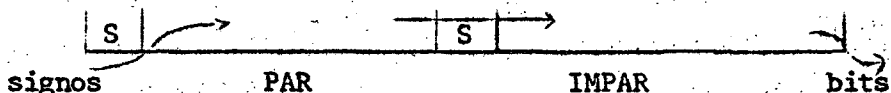
3.35. Instrucción SHIFT RIGHT DOUBLE

a) Instrucción: SRDA R1, D2(B2)

b) Formato : RS

SRDA	R1	//////	B2		D2
------	----	-------------------	----	--	----

c) Función : El contenido de los RUG PAR e IMPAR siguiente especificado por el primer operando R1, es desplazado a la derecha el número de posiciones indicado por los seis bits de orden inferior de la representación binaria de la dirección dada por D2 y B2. El resto de la dirección se ignora. En R1 se especifica un RUG PAR, lo contrario produce una interrupción de programa por error de especificación. El signo del RUG IMPAR se desplaza como parte de la información, el del RUG PAR permanece sin cambio. Los lugares vacantes se llenan con bits iguales al signo.



Ejemplo 26.

```

CTE   DC   F'17'
RES   DS   2F
      L    1,CTE
      SLDA 0,31
      SRA  0,1
      SRDA 0,31
      ST   0,RES
      ST   1,RES+4
    
```

Explicación: Con la instrucción L 1,CTE se tiene:

		00.....010001	RUG1
SLDA	0,31	00.....01000 10.....0000	RUG0 RUG1
SRA	0,1	00.....00100 10.....0000	RUG0 RUG1
SRDA	0,31	00.....00000 00.....1001	RUG0 RUG1

queda así, entonces en RUG0 el valor 0 y en RUG1 el valor 9.

3.36. Modificador de escala para constantes de punto fijo

El modificador de escala especifica la potencia de dos por la cual la constante debe ser multiplicada después que ella ha sido convertida a su representación binaria. Esto significa que una porción fraccionaria puede ser desplazada a representación entera si el exponente es positivo. Por el contrario, si el exponente es negativo, la parte entera puede ser borrada total o parcialmente. El modificador de escala se representa como Sn, donde n es un valor decimal entero con o sin signo o una expresión absoluta encerrada entre paréntesis ().

Ejemplo 27.

DAT01 DC FS5'14.25'

Si no existiera el modificador de escala se representaría solo el valor 14. En este caso la representación corresponderá a $14.25 * 2^5$ que es igual a 456. La representación binaria de 14.25 es:

0.....01110 | 01

punto binario

si esta información se desplaza 5 posiciones a la izquierda queda:

0.....0111001000

que corresponde al valor 456.

Ejemplo 28.

Cálculo de $2489 + \frac{2489 \cdot 3}{100}$, o lo que es lo mismo $2489 * 1.03$

a)	AREA	DS	F
	CIEN	DC	F'100'
	C0NSTA	DC	F'50'
	C0NSTB	DC	F'2489'
	C0NSTC	DC	F'103'
		L	5,C0NSTB
		M	4,C0NSTC
		A	5,C0NSTA
		D	4,CIEN
		ST	5,AREA

b) Otra forma, usando modificador de escala Sn

```

AREA DS F
CONSTA DC FS11'05'
CONSTB DC E'2489'
CONSTC DC FS11'1.03'
        L      5,CONSTB
        M      4,CONSTC
        A      5,CONSTA
SRA    5,11
ST     5,AREA
    
```

3.37. Otros tipos de constantes

a) Constante tipo X.

El tipo X permite definir constantes hexadecimales. Estas constantes consisten de uno o más dígitos hexadecimales. La longitud máxima es de 256 bytes, y el campo ocupado por la constante no se ajusta a alineamiento.

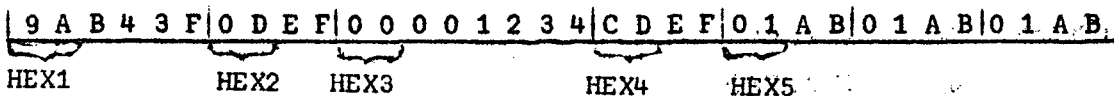
Cada dígito hexadecimal ocupa medio byte. Si la longitud del campo excede al número de dígitos que se genera, se completa el campo con ceros hexadecimales por la izquierda. Si el campo no puede contener a todos los dígitos, se trunca la constante por la izquierda.

Ejemplo 29.

```

HEX1 DC X'9AB43F'
HEX2 DC X' DEF'
HEX3 DC XL4'1234'
HEX4 DC XL2'ABCDEF'
HEX5 DC 3X'1AB'
    
```

Estas constantes quedarán generadas en la siguiente forma:



b) Constante tipo C.

El tipo C permite definir caracteres en el código EBCDIC. Cada carácter por tanto ocupa un byte. La longitud máxima es de 256 bytes y el campo ocupado por la constante no se ajusta a alineamiento.

Si la longitud del campo excede al número de caracteres que se genera, se completa el campo con caracteres blancos, por la derecha. Si el campo no puede contener a todos los caracteres, se trunca la constante por la derecha.

Ejemplo 30.

CAR1	DC	C'ABCD'
CAR2	DCC	CL4'+15'
CAR3	BCC	CL2'TABLA'
CAR4	DC	3C'A*B'

Estas constantes quedarán generadas en la siguiente forma:

A B C D + 1 5 15 T A A * B A * B A * B				
CAR1	CAR2	CAR3	CAR4	

En las tarjetas, la representación de las letras A,B,C,...,I está dada por una perforación en ZONA 12 y una perforación en el dígito 1,2,3,...,6 9 según sea la letra. En memoria, con el tipo C, quedan representadas esas mismas letras como sigue:

A	11000001
B	11000010
C	11000011
I	11001001
	{← 1 byte →}

se puede observar que los cuatro bits de orden superior del byte tienen el valor binario 1100 = 12 decimal, y los cuatro bits de orden inferior tienen el valor 1, 2, 3 ó 9, decimal según sea la letra. Aún cuando no existe la misma relación para el resto de los caracteres se acostumbra decir que la representación en memoria está en la forma ZONA-CARACTER o ZONA DIGITO o ZONA NUMERO, estas dos últimas aplicadas más bien al formato del tipo Z que se verá más adelante.

Z, C	Z, C	Z, C	Z, C	Z, C
------	------	------	------	------

De acuerdo a esta estructura, las constantes del ejemplo 30 se puede decir que quedan como sigue:

Z A Z B Z C Z D	Z + Z 1 Z 5 Z 8	Z T Z A	Z A Z * Z B	Z A Z * Z B	Z A Z * Z B
CAR1	CAR2	CAR3	CAR4		

a) Constante tipo Z.

El tipo Z permite definir valores numéricos en el formato ZONA-DIGITO. Difiere con el formato obtenido con el tipo C, en que el byte de orden inferior tiene la estructura SIGNO-DIGITO.

Z D	Z D	Z D	Z D	S D
-----	-----	-----	-----	-----

D = dígito S = signo

Si la longitud del campo excede al número de dígitos de la constante, se completa el campo con dígitos cero por la izquierda. Si el campo no puede contener a todos los dígitos, se trunca la constante por la izquierda.

Ejemplo 31.

ZØN1	DC	Z ¹ +352'
ZØN2	DC	Z ¹ -437'
ZØN3	DC	Z15'4'
ZØN4	DC	ZL2'-3895'
ZØN5	DC	2ZL3'18'

Estas constantes quedarán generadas en la siguiente forma:

Z 3 Z 5 + 2	Z 4 Z 3 - 7	Z 0 Z 0 Z 0 Z 0 + 4	Z 9 - 5	Z 0 Z 1 + 8	Z 0 Z 1 + 8
ZØN1	ZØN2	ZØN3	ZØN4	ZØN5	

d). Constante tipo P.

El tipo P permite definir valores numéricos en el formato de dígitos empaquetados.

D D	D D	D D	D D	D S
-----	-----	-----	-----	-----

D = dígito S = signo

Si la longitud del campo excede al número de dígitos de la constante, se completa el campo con dígitos cero por la izquierda. Si el campo no puede contener a todos los dígitos, se trunca la constante por la izquierda.

Ejemplo 32.

PAC1	DC	P'+281'
PAC2	DC	P'-695'
PAC3	DC	PL5'7'
PAC4	DC	PL2'-856723'
PAC5	DC	3PL3'36'

Estas constantes quedarán generadas en la siguiente forma:

281+	695-	000000007+	723-	00036+	00036+	00036+
PAC1	PAC2	PAC3	PAC4	PAC5		

No existen instrucciones que permitan realizar operaciones aritméticas con constantes definidas con el tipo C, o el tipo Z. Por otra parte, la información al ser leída a memoria queda en el formato ZONA-CARACTER. Sin embargo, existen instrucciones que permiten convertir del formato ZONA-DIGITO (formato del tipo Z) a formato DIGITOS EMPAQUETADOS y de éste a BINARIO. Existen además, instrucciones que permiten realizar el proceso inverso, esto es, de formato BINARIO a formato de DIGITOS EMPAQUETADOS y de éste a ZONA-DIGITO. Para la información que se lee a memoria, es necesario considerar que la ZONA de los dígitos es 1111 y ésta configuración se interpreta como signo +. Luego los números positivos leídos se pueden considerar como en formato ZONA DIGITO puesto que el último byte tendrá la estructura 1111XXXX, es decir, + XXXX. En los números negativos será tarea del programador, reemplazar con las instrucciones adecuadas la configuración 1111 por 1101 que corresponde al signo -. Se verá a continuación el siguiente ejemplo:

Ejemplo 33.

Caso a) Se lee a memoria el dato 3452. Su estructura será:

Z 3	Z 4	Z 5	Z 2
-----	-----	-----	-----

en binario:

11110011	11110100	11110101	11110010
----------	----------	----------	----------

+

o lo que es lo mismo:

Z 3	Z 4	Z 5	+ 2
-----	-----	-----	-----

Caso b) Se lee a memoria el dato negativo 8564. Debe leerse como positivo.
Su estructura será:

Z	8	Z	5	Z	6	Z	4
---	---	---	---	---	---	---	---

en binario

1	1	1	1	1	0	0	0	1	1	1	1	0	1	0	1	1	1	1	0	1	1	0	1	1	1	0	1	0	0
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

debe reemplazarse por 1101

para que quede:

Z	8	Z	5	Z	6	-	4
---	---	---	---	---	---	---	---

3.38. Instrucción PACK (Aritmética Decimal)

- a) Instrucción: PACK D1(L1,B1),D2(L2,B2)
- b) Formato : SS

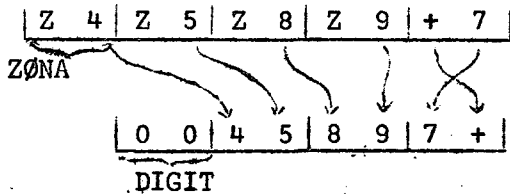
PACK	L1	L2	B1	D1	B2	D2
------	----	----	----	----	----	----
- c) Función : El contenido del campo D2(B2) que se considera en formato de ZONA-DIGITO es convertido a formato PACKED y el resultado almacenado en el campo D1(B1). Para la conversión las zonas son ignoradas, excepto la del byte de orden inferior pues se supone que ella representa el signo. El signo se coloca en los cuatro bits de la derecha del byte de orden inferior, y a la izquierda de él se agrupan los dígitos.

Los campos se procesan de derecha a izquierda y no hay verificación de validez de códigos. Si el campo del primer operando es demasiado corto como para contener todos los dígitos significativos del campo del segundo operando, los dígitos restantes se ignoran. Por el contrario, si el campo es más largo, se rellena con ceros decimales hasta completar el campo.

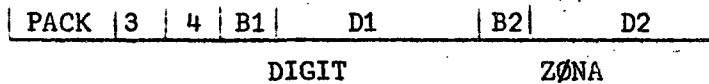
NOTA: En todas las instrucciones en que se especifica longitud, el compilador al hacer la traducción resta 1 a las longitudes.

Ejemplo 34.

ZONA	DC	Z'45897'
DIGIT	DS	PL4
PACK		DIGIT(4),ZONA(5)



La instrucción PACK DIGIT(4), ZØNA(5) al ser generada quedará como:



Dado que al ser generada la constante o área, se asigna al símbolo la longitud en que se ha definido la constante o área, se puede escribir con el mismo resultado

	PACK	DIGIT, ZØNA
6	PACK	DIGIT(4), ZØNA
6	PACK	DIGIT, ZØNA(5)

3.39. Instrucción CONVERT TO BINARY

- a) Instrucción: CVB R1, D2(X2, B2)
- b) Formato : RX | CVB | R1 | X2 | B2 | D2 |
- c) Función : La doble palabra contenida en la dirección D2(X2, B2) y cuyo formato es PACKED es convertida a binario puro y su resultado almacenado en el RUG especificado en R1. El mayor valor que puede convertirse corresponde a la capacidad de un RUG (2147483647 o -2147483648). La dirección D2(X2, B2) debe corresponder a un alineamiento de doble palabra.

Ejemplo 35.

ZØNA	DC	Z' -48963'
BINARIØ	DS	D
RESULT	DS	F
	PACK	BINARIØ, ZØNA
	CVB	5, BINARIØ
	ST	5, RESULT

3.40. Instrucción CONVERT TO DECIMAL

- a) Instrucción: CVD R1,D2(X2,B2)
- b) Formato : RX

CVD	R1	X2	B2	D2
-----	----	----	----	----
- c) Función : El <RUG> especificado en R1 y cuyo formato es binario puro es convertido a PACKED y depositado en la doble palabra indicada con D2(X2,B2). La dirección debe corresponder a un alineamiento de doble palabra.

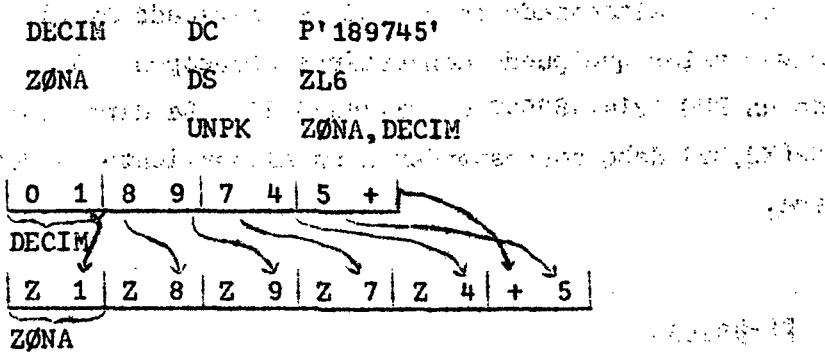
3.41. Instrucción UNPACKED (Aritmética Decimal)

- a) Instrucción: UNPK D1(L1,B1),D2(L2,B2)
- b) Formato : SS

UNPK	L1	L2	B1	D1	B2	D2
------	----	----	----	----	----	----
- c) Función : El contenido del campo D2(B2) que se considera en formato PACKED es convertido a formato ZONA-DIGITO y el resultado almacenado en el campo D1(B1).

Los campos se procesan procediendo de derecha a izquierda. Si el campo del primer operando es demasiado corto como para contener todos los dígitos significativos del segundo operando, los dígitos sobrantes se ignoran.

Ejemplo 36:



3.42. Problemas propuestos

- a) Definir
 - i) un área de 57 bytes de longitud con alineamiento de palabra.
 - ii) una constante de nombre BYTE cuyo contenido sea 0111 1011.

- b) Indicar los contenidos del RUG2 y AREA al terminar el siguiente proceso:

```
JUAN DC F'+35'  
      DC F'-30'  
      DC H'10'  
AREA DS 2F  
      L 4,JUAN  
      LA 2,10(0,4)  
      AR 4,2  
      A 4,JUAN+4  
      AH 2,JUAN+8  
      AR 2,2  
      ST 4,AREA  
      STH 2,AREA+4
```

- c) Ordenar en secuencia ascendente los tres datos que hay en el área TRESDAT (3 palabras completas).

- d) Programar la siguiente expresión:

$$Z = 2(A+B-2(C-D))$$

- e) Programar la siguiente expresión:

$$\text{SUELDON} = \text{SUELDOA} + 21\% \text{ de SUELDOA}$$

- f) Analizar e indicar qué queda en AREA al término del siguiente programa:

```
UNØ   DC F'1'  
DØS   DC F'2'  
CTE1  DC F'7'  
CTE2  DC F'-18'  
AREA  DS F  
      L 1,CTE1  
      L 2,CTE2  
      SR 1,2  
      BC 4,MENØR  
      MR 0,2  
      ST 1,AREA  
      BC 15,ØUT  
MENØR M 0,UNØ  
      D 0,DØS  
      ST 0,AREA
```

ØUT

g) Analizar e indicar qué queda en RES al término del siguiente programa:

```
ALFA DC F'5'  
BETA DC F'-4'  
RES DS F  
LH 1,ALFA  
LA 2,10(0,1)  
AR 2,2  
SH 2,BETA  
ST 2,RES
```

h) Programar el siguiente cálculo:

$$Z = 2 | X + 2 | X - | Y | | - | Y | |$$

suponer que siempre los dígitos significativos caben en un RUG.

i) Analizar e indicar qué queda en AREA al término del siguiente programa:

```
AREA DS F  
DATØ1 DC XL4'1'  
DATØ2 DC PL4'0'  
LM 1,2,DATØ1  
CR 1,2  
BC 4,MM  
AR 2,1  
BC 15,STØRE  
MM SR 2,1  
STØRE ST 2,AREA
```

j) Indicar el contenido de AREA, después del siguiente proceso:

```
AREA DS F  
A DC F'15'  
B DC F'18'  
LM 1,2,A  
SRA 2,1  
AR 1,2  
SLDA 0,1  
AR 1,2  
ST 1,AREA
```

k) Indicar el contenido de DOSB al terminar el siguiente proceso:

DOSB DS H
CUATRØ DC F'4'
ØCHØ DC F'8'
C256 DC F'256'
CERØ DC H'0'
C15 DC H'15'
L 1, CUATRØ
L 15, C256
LA 2, 40(1,15)
L 3, CERØ
AH 3, C15
SLR 2,3
STH 2, DOSB

l) Indicar el contenido de Z después del proceso siguiente:

A DC F'37'
B DC F'-42'
C DC H'-14'
Z DS F
L 1, A
LH 2, C
LH 3, B
MR 0, 2
DR 0, 3
AL 1, B
SL 1, C
ST 1, Z

11) Indicar qué queda en Z después del siguiente proceso:

```

A  DC  F' -18'
   DC  H' 0'
   DC  H' 35'
B  DC  F' 42'
Z  DS  F
   LH  1, B
   AH  1, A
   L   2, A+4
   AR  2, 1
   S   2, B
   LH  3, A+4
   STH 3, Z
   STH 2, Z+2
    
```

m) Programar la expresión:

$$Z = |X| - 2 |X - |Y||$$

n) Indicar lo que queda en Z después del siguiente proceso:

```

A  DC  F' 57'
   DC  F' -38'
B  DC  H' 32'
   DC  H' -25'
C  DC  F' 0'
Z  DS  F
   LM  1, 2, A
   LH  3, B
   CH  1, C
   BC  2, MAYØR
   ALR 1, 3
   SR  1, 2
MAYØR STH 3, C+2
   A   2, C
   M   0, A+4
   D   0, C
   SH  1, B+2
   ST  1, Z
    
```


ñ) Programar el problema siguiente:

$$Z = \frac{A+B}{A-B} * 18 + X \quad \text{si } A \leq 15$$

$$Z = \frac{A-B}{A+B} * 81 - X \quad \text{si } 15 < A \leq 35$$

$$Z = \frac{(A+B)(A-B)}{2A - B} + X \quad \text{si } A > 35$$

o) Indicar el contenido de RUG1 después del proceso siguiente:

A	DC	F'53'
B	DC	F'-48'
C	DC	F'15'
UNØ	EQU	4
	LM	1,3,A
	SR	0,0
	SLDA	0,32
	SRA	0,UNØ
	SLDA	0,32
	AR	2,3
	LPR	2,2
	SRA	2,UNØ
	AR	1,2

p) Indicar el contenido de A después del siguiente proceso:

A	DC	F'15'
B	DC	H'0'
C	DC	H'18'
	LM	1,2,A
	AH	1,B
	LTR	1,1
	BC	8,ALFA
	SLA	1,1
ALFA	ST	1,A

q) Programar el problema siguiente:

$$SL = SB*1.40-DL$$

Si SL <505 desplazar el <RUG 5> = 10 un lugar a la izquierda, en caso contrario, desplazarlo un lugar a la derecha y el resultado almacenarlo en CONTROL.

r) El descuento por Seguro Social en los EEUU es el 3 1/8 por ciento de los ingresos hasta US\$4800 en un año. Dados los ingresos acumulados hasta la fecha, calculados con anterioridad (IAF) y sus entradas de la presente semana, calcular el descuento sobre sus ingresos de esta semana y los nuevos ingresos acumulados hasta el momento (NIA). Deben considerarse las siguientes posibilidades.

i) La persona ha ganado US\$ 4800 o más, antes de esta semana en cuyo caso el descuento es cero.

ii) La persona no ha ganado US\$ 4800 incluso con su ingreso de esta semana, en cuyo caso el descuento es de 3 1/8 por ciento de las ganancias de esta semana.

iii) Antes de esta semana, la persona no ha ganado US\$ 4800, pero sí, al incluir lo que ha ganado en ella. En este caso, el descuento es de 3 1/8 por ciento de la diferencia entre US\$ 4800 y sus ingresos previos acumulados.

s) Se tienen tres datos A, B y C, ocupando ocho bytes cada uno, en el formato ZONA-DIGITO. Se pide calcular:

$$R = A + B - C$$

4. Ensamblado y pseudo instrucciones

4.1. Términos y expresiones

Cada término representa un valor. Este valor puede ser asignado por el ensamblador o puede estar implícito en el término (términos autodefinidos o literales).

Ejemplo 37.

15,4092,X'AB4',X'FF' son términos autodefinidos.

Expresiones: Hay dos tipos de expresiones:

- ABSOLUTA
- REUBICABLE

Una expresión estará formada por un término o un conjunto de términos relacionados entre si por operadores aritméticos.

Reglas para construir expresiones:

- a) No puede empezar con operador aritmético
- b) Los términos deben separarse por operadores o paréntesis
- c) No pueden haber dos o más operadores seguidos
- d) No puede tener más de 16 términos
- e) No pueden haber más de 5 niveles de paréntesis
- f) Una expresión de varios términos no puede contener un literal
- g) La evaluación de las expresiones se realiza de izquierda a derecha, con prioridad de multiplicaciones y divisiones sobre sumas y restas.
- h) El resultado de la división es entero. Si se divide por cero, el resultado es cero.

A. Expresión absoluta

Una expresión es absoluta si su valor no cambia al ser reubicado el programa.

Una expresión absoluta puede tener términos reubicables, solos o en combinación con términos absolutos, con las siguientes restricciones:

- a) Debe haber un número par de términos reubicables (R) en la expresión.
- b) Los términos reubicables deben estar pareados, esto es, debe existir un término reubicable con signo + y un término reubicable con signo -. Los términos pareados no tienen que estar contiguos obligadamente.
- c) Los términos pareados, deben entrar en esa forma, en multiplicaciones o divisiones. Así, $R-R*10$ no es válido en cambio $(R-R)*10$ es válido.

B. Expresión reubicable

Una expresión es reubicable si su valor cambia en n al ser desplazado el programa en memoria (reubicado) n bytes desde su punto de origen.

Una expresión reubicable puede tener términos reubicables, solos o en combinación con términos absolutos, con las siguientes restricciones:

- a) Debe haber un número impar de términos reubicables (R)
- b) Todos los términos reubicables salvo uno, deben estar pareados
- c) El término no pareado no puede estar precedido por signo menos
- d) No pueden haber términos reubicables en una multiplicación o división.

4.2. Seccionamiento de programas

Un programa grande puede ser subdividido en secciones llamadas secciones de control (Control Section). Las secciones pueden ser traducidas en forma independiente y después combinadas formando un solo programa objeto. Una sección de control es un conjunto de instrucciones que puede ser reubicado independientemente de otra sección de control. Normalmente la identificación de una sección de control es realizada a través de la instrucción CSECT. La primera sección de control puede ser identificada también con la instrucción START.

La combinación de las secciones de control se puede realizar debido a que el ensamblador cuenta con un Contador de Direcciones (Location Counter) para cada sección de control. Estas son asignadas a ubicaciones de partida consecutivas, de acuerdo al orden en que se van sucediendo en el programa. Cada sección de control posterior a la primera, empieza en la siguiente doble palabra disponible.

El Contador de Direcciones (CD) es inicializado con la instrucción START. Antes de generarse una instrucción, una constante o un área, el CD se ajusta al alineamiento apropiado para el ítem, si es que el ajuste es necesario. Después de traducido el ítem se incrementa el valor contenido en el CD en la longitud del ítem. Así, siempre indica al ensamblador, la próxima posición disponible. Si la instrucción es nombrada por un símbolo, el valor atribuido del símbolo es el valor contenido en el CD después del ajuste al alineamiento, pero antes de sumar la longitud. Tales símbolos son siempre reubicables.

La primera sección de control de un programa tiene las siguientes propiedades:

- a) El valor inicial de su CD puede ser especificado como un valor absoluto.
- b) Normalmente contiene los literales (ver literales) requeridos en el programa.

Un programa no seccionado es tratado como una sección de control única.

A. Pseudo Instrucción CONTROL SECTION

- a) Código : CSECT
- b) Formato : [símbolo] CSECT sin operandos
- c) Función : Permite identificar el comienzo o la continuación de una sección de control. No genera instrucción de máquina. El símbolo es establecido como el nombre de la sección de control, en caso contrario la sección es considerada sin nombre. Todas las instrucciones que siguen a la CSECT son traducidas como parte de esa sección de control, hasta que se encuentre otra CSECT o una pseudo instrucción DSECT.

Si aparecen varias pseudo instrucciones CSECT con el mismo nombre, se considera que la primera identifica el comienzo de la sección y las demás identifican reanudaciones de ella.

B. Pseudo Instrucción START

- a) Código : START
- b) Formato : [símbolo] START término autodefinido o blanco
- c) Función : Permite identificar la primera o única sección de control. Además inicializa el Contador de Direcciones. No genera instrucción de máquina.

El símbolo es establecido como el nombre de la sección de control, en caso contrario la sección es considerada sin nombre. El valor del término autodefinido es cargado en el CD. Debe ser múltiplo de 8 para partir con almacenamiento de doble palabra. Si no se coloca operando autodefinido se carga 0 en el CD.

Ninguna instrucción que dependa del contenido del CD puede estar antes de START.

C. Secciones de Control sin nombre

Puede haber solo una sección de control sin nombre en un programa. Si aparece una sección de control sin nombre y es seguida por secciones de control con nombre, cualquier otra sin nombre, se considera reanudación de la primera.

D. Pseudo Instrucción USING

Permite indicar al ensamblador: él o los registros de uso general que puede utilizar como Registros Base y además cuales son sus contenidos supuestos.

- a) Código : USING
- b) Formato : blanco USING expresión, r1,r2,...,rn
- c) Función : Indica al ensamblador que puede utilizar como registros base aquellos indicados en los operandos r1,r2,...,rn. Especifica además cuales serán los contenidos supuestos de esos registros.
El RUG indicado en r1 tiene como contenido supuesto el valor de la expresión. El que se indica en r2 tendrá el valor de la expresión + 4096. El que se indica en rn tendrá el valor de la expresión + (n-1)*4096.

Ejemplo 38.

```
USING    FIRST,4
```

Indica al ensamblador que puede utilizar el RUG4 como registro base y el contenido que debe suponer en él es el valor de la expresión FIRST.

Como expresión se puede utilizar el signo * (asterisco) que representa el valor contenido en el CD en el momento de analizarse el signo.

Ejemplo 39.

```
USING    *,15
```

Indica al ensamblador que puede utilizar el RUG15 como registro base y el contenido que debe suponer en él es a su vez el contenido del CD.

E. Pseudo Instrucción DROP

- a) Código : DROP
- b) Formato : blanco DROP r1,r2,...,rn
- c) Función : Indica al ensamblador que no puede utilizar como registros base aquellos indicados en los operandos r1,r2,...,rn. Si el RUG indicado no se estaba utilizando, la instrucción no tiene efecto.

F. Sección Formal (Dummy Section)

Una sección formal representa una sección de control que es ensamblada pero que no es parte del programa objeto. Se puede describir así la estructura de un área de almacenamiento sin que se reserve dicha área (se supone que se reserva área de almacenamiento por alguna parte del mismo o de otro ensamblado).

Pseudo Instrucción DUMMY SECTION

- a) Código : DSECT
- b) Formato : [símbolo] DSECT sin operandos
- c) Función : Permite identificar el comienzo o reanudación de una sección formal. No genera instrucción de máquina. El símbolo es establecido como el nombre de la sección formal. Todas las instrucciones que siguen a la DSECT pertenecen a la sección formal.

Si aparecen varias DSECT con el mismo nombre, se considera que la primera identifica el comienzo de la sección y las demás identifican reanudaciones de ella. Los símbolos que aparecen en el campo de nombre de una DSECT o de las instrucciones de la sección pueden ser utilizados como operandos en pseudo instrucciones USING e instrucciones de máquina. Se utiliza un Contador de Direcciones para determinar la ubicación relativa de los elementos de programa nombrados en la sección. El contenido del CD es siempre cero al comienzo de la sección, y los valores correspondientes a los símbolos que nombran instrucciones en la sección formal son relativos a la instrucción inicial de la misma sección. El objetivo de las secciones formales es permitir al programador describir áreas cuya ubicación en memoria no será determinada hasta que el programa sea ejecutado. El área se describe en la sección formal.

Para efectos de funcionamiento se proporciona al ensamblador, a través de una pseudo instrucción USING, un registro cuyo contenido supuesto será el valor del símbolo que identifica la DSECT.

Ejemplo 40.

Supóngase un programa principal que llama a un subprograma de nombre RUTINA. En el programa principal se carga en RUG11 la dirección de partida del área de entrada (dirección actual). En el subprograma se indica al ensamblador que puede utilizar como RUG base el RUG11 y como contenido supuesto tendrá el valor del símbolo que identifica la sección formal, esto es, cero. En la sección formal se describe el área que utilizará el programador.

```

RUTINA          CSECT
BEGIN          BALR      2,0
              USING     *,2
              --
              --
              USING     AREIN,11
              TM        BYTE,X'C5'
              B0        UN0S
              --
              --
              UN0S      MVC      MENSAJE,MENSA
              --
              --
              MENSA     DC      CL15'WT000SUN0S00'
              MENSAB    DC      CL15'WT000SICER0S00'
              --
              --
              AREIN     DSECT
              BYTE     DS      CL1
              MENSAJE   DS      CL15
              --
              --
              END

```


G. Instrucción BRANCH AND LINK REGISTER (Instrucción de bifurcación)

- a) Instrucción: BALR R1,R2
- b) Formato : RR

BALR	R1	R2
------	----	----
- c) Función : Se almacenan en el RUG especificado en R1 los 32 bits del extremo derecho de la palabra de estado del programa (PSW), esto es, se almacena el "código de longitud de la instrucción" el "código de condición", las "máscaras de programa" y la "dirección de la instrucción siguiente". A continuación se ejecuta la instrucción contenida en la dirección indicada en el RUG especificado en R2 (salto). Si el campo R2 es cero, no se efectúa el salto, se ejecuta la próxima instrucción en secuencia.

Ejemplo 41.

```
LA      5,SALTO
--
--
--
10600 BALR  7,5
--
--
--
SALTO
```

Explicación. Se carga el RUG5 con la dirección SALTO. Se carga en el RUG7 la segunda mitad de la PSW, en la parte de dirección queda 10602. Se efectúa el salto a la dirección indicada en RUG5.

NOTA: La pseudo instrucción USING se vio que permite indicar al ensamblador que registros puede utilizar como BASE y cuáles son sus contenidos supuestos. En la etapa de ejecución, esos contenidos deben ser cargados en los respectivos RUG. La instrucción BALR es una de las que permite cumplir ese objetivo.

Ejemplo 42.

	START	2000
	BALR	12,0
	USING	*,12
	BC	15,SIGA
JUAN	DC	F'0'
SIGA	--	
	--	
	--	

Explicación. En la etapa de traducción en una primera pasada de análisis del programa, se asignan valores a los símbolos de acuerdo con el operando de la pseudo instrucción START.

	START	2000
2000	BALR	12,0
	USING	*,12
2002	BC	15,SIGA
2008 JUAN	DC	F'0'
2012 SIGA	--	
	--	
	--	

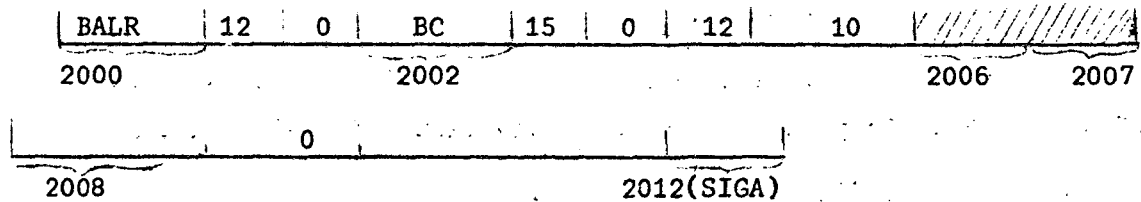
Al asignar el valor al símbolo JUAN, el CD se ajusta al alineamiento de palabra necesario para generar la constante.

En la segunda pasada de análisis del programa se terminan de generar instrucciones constantes y áreas.

- Se genera la instrucción BALR 12,0
- La pseudo instrucción USING indica al ensamblador que puede utilizar como registro BASE el 12, y como contenido supuesto el valor que tiene el CD en ese momento, es decir, 2002.
- Se genera la instrucción BC 15,SIGA
 - la máscara 15 figura en el campo M1.
 - se coloca 0 en el campo X2, porque no se indica en la instrucción uso de registro INDICE.
 - el valor del símbolo SIGA (2012) se distribuye entre <RUG B> y D.

$$\begin{aligned} \text{Valor de SIGA} &= \langle \text{RUG } 12 \rangle + D2 \\ 2012 &= 2002 + D2 \\ D2 &= 10 \end{aligned}$$

iv) Se tiene así:



H. Instrucción BRANCH AND LINK (Instrucción de bifurcación)

- Instrucción: BAL R1,D2(X2,B2)
- Formato : RX BAL|R1|X2|B2|D2
- Función : Se realiza la misma función que con BALR. El salto en este caso lo efectúa a la dirección dada por D2(X2,B2).

I. Normas que sigue el ensamblador para asignar Registro Base

- El contenido supuesto del registro que va a utilizar debe ser siempre menor o igual que la dirección que se va a conformar.
- Si hay dos o más registros, utiliza aquel que permite menor desplazamiento en la instrucción compilada.
- Si hay dos o más registros que tienen igual contenido supuesto, elige el mayor de ellos.

4.3. Constantes de Dirección

Una constante de dirección corresponde a una dirección de memoria almacenada como una constante. Normalmente se utilizan para inicializar registros base.

La constante de dirección es encerrada entre paréntesis. Si hay dos o más constantes, se separan entre sí por coma y el conjunto completo se encierra entre paréntesis.

Hay cuatro tipos de constantes de dirección: A,Y,S y V.

- Tipo A: El valor máximo puede ser $2^{31}-1$, ocupa 4 bytes y se almacena en alineamiento de palabra completa. La constante puede ser especificada como una expresión absoluta, o una expresión reubicable. Lo normal es lo segundo, luego, al ser reubicado el programa, las constantes de dirección cambian de valor en base al nuevo punto de carga del programa.

b) Tipo Y: Similar al tipo A, difiere sólo en que su valor máximo puede ser $2^{15}-1$, o sea, ocupa dos bytes.

c) Tipo S: Se utiliza para almacenar una dirección en la forma BASE-DESPLAZAMIENTO. Ocupa dos bytes con alineamiento de media palabra.

Se puede especificar en dos formas:

i) Como una expresión absoluta o reubicable. Ej. S(BETA)

ii) Como dos expresiones absolutas, siendo la primera el desplazamiento y la segunda el registro base. Ej. S(400(13)).

d) Tipo V: Se utiliza para reservar área para la dirección de un símbolo externo que será meta de salto. El símbolo debe ser especificado como símbolo reubicable y necesita ser declarado explícitamente como externo.

(Ver pseudo instrucción EXTRN). Ocupa 4 bytes con alineamiento de palabra completa.

Ejemplo 41.

```

PRØGA      START      1000
1000      BEGIN      BALR      15,0
          USING      FIRST,15
1002      FIRST      BC        15,SKP
1008      DATA      DC        F'3472'
1012      BASE1      DC        A(FIRST+4096)
          --
          --
1016      SKP        L         14,BASE1
          USING      FIRST+4096,14
          L         13,BASE2
          --
          --
          USING      FIRST+2*4096,13
2000      BC        15,CK8
          --
          --

```

(Continúa)

(Continuación)

```

3000  LOOP  A   4,DATA
      ---
      ---
      ---
4000  LOOPB S   5,DATA
      ---
      ---
      ---
10000 BC   8,LOOP
      ---
      ---
      ---
11000 CK8   BC  8,LOOPB
      END   BEGIN

```

Los números que aparecen a la izquierda del programa corresponden a los valores tentativos asignados por el ensamblador en la primera pasada de análisis. Los pasos de la segunda pasada son los siguientes:

- a) Se genera la instrucción BALR 15,0

BALR	15	0
------	----	---

BEGIN=1000

b) La pseudo-instrucción USING indica al ensamblador que puede utilizar como registro BASE el 15 y como contenido supuesto, el valor de FIRST, esto es, 1002.

- c) Se genera la instrucción BC 15,SKP

BC	15	0	15	100
----	----	---	----	-----

el desplazamiento se obtiene de:

$$\begin{aligned}
 \text{Valor de SKP} &= \langle \text{RUG15} \rangle + D2 \\
 1102 &= 1002 + D2 \\
 D2 &= 100
 \end{aligned}$$

d) Se genera la constante 3472. Se saltan dos bytes para ajustarse al alineamiento adecuado

3472

DATA = 1008

e) Se generan las constantes de dirección

Valor de FIRST + 4096
BASE1= 1012

Valor de FIRST + 2 *4096
BASE2= 1016

f) Para generar la instrucción de nombre SKP se dispone sólo del RUG15 como BASE con contenido supuesto igual a 1002.

$$\begin{aligned} \text{Valor de BASE1} &= \langle \text{RUG 15} \rangle_s + D2 \\ 1012 &= 1002 + D2 \\ D2 &= 10 \end{aligned}$$

L	14	0	15	10
---	----	---	----	----

g) La pseudo instrucción USING indica al ensamblador que puede además utilizar como registro BASE el 14, y como contenido supuesto el valor de FIRST más 4096, esto es, 5098.

h) Para generar la instrucción L 13, BASE2 se dispone de los registros 15 y 14. De acuerdo a la norma a) del punto I se elige el 15.

L	13	0	15	14
---	----	---	----	----

i) La pseudo instrucción USING indica al ensamblador que puede además utilizar como registro BASE el 13 y como contenido supuesto el valor de FIRST más dos veces 4096, esto es, 9194.

j) Para generar la instrucción BC 15, CK8 se dispone de los registros 15, 14 y 13. De acuerdo a la norma a) del punto I se elige el 13.

BC	15	0	13	1806
----	----	---	----	------

k) En forma similar se generan las instrucciones siguientes con lo cual se obtiene:

A	4	0	15	6
S	5	0	15	6
BC	8	0	15	1998
BC	8	0	14	1902

Ejemplo 42.

```

                START    256
BEGIN          BALR     15,0
                USING    *,15,14,13,12
DIR           BC       15,GØ
R12           DC       A(DIR+12288)
R13           DC       A(DIR+8192)
R14           DC       A(DIR+4096)
                --
                --
                --
GØ            LM       12,14,R12

```

Con esta última instrucción se cargan los RUG 12,13 y 14 con DIR+12288, DIR+8192 y DIR+4096 respectivamente.

4.4. Literales

El usuario tiene la opción de definir y usar simultáneamente una constante en la instrucción donde es requerida. Esas constantes se denominan literales. Su estructura en la instrucción de assembler es la misma que la de los operandos de los tipos de constantes ya vistos, precedida del signo =.

El ensamblador genera los literales, los agrupa y los almacena en un área llamada Area de Literales (Literal Pool) al final de la primera o única sección de programa.

Los literales se almacenan de acuerdo a su longitud y orden de aparición (sin que esto signifique repetición). En cuanto a la longitud, se almacenan primero los que sean divisibles por ocho, después los que sean divisibles por cuatro, a continuación otros literales de longitud par y finalmente los de longitud impar.

Ejemplo 43.

```

START    0
--
--
--
L      5,= F'-42'
--
--
--

```

(Continúa)

(Continuación)

```

AH      6,=H'15'
--
--
--
L       7,=A(FIRST+4096)
--
--
--
END

```

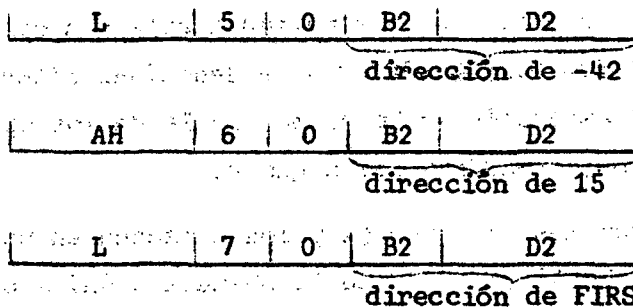
Los literales definidos se agruparán al final del programa en la siguiente forma:

```

=F'-42'
=A(FIRST+4096)
=H'15'

```

y las instrucciones generadas serán:



Se puede observar que las direcciones de los literales no son conocidas por el programador, en otras palabras, no es conocida la dirección de origen del área de literales. Mediante la pseudo instrucción Literal Origen el programador puede controlar la primera posición del área de literales.

A. Pseudo instrucción Literal Origen

- a) Código : LTORG
- b) Formato : [símbolo] LTORG sin operandos
- c) Función : Agrupa todos los literales, definidos desde la instrucción LTORG anterior o desde el comienzo del programa si no existe otra, inmediatamente después de su aparición. Si se especifica símbolo, él corresponde a la dirección del primer byte del área de literales. No genera instrucción de máquina.

Ejemplo 44.

```
START      0
--
--
--
L          7,=F'900'
--
--
--
SH        7,=H'52'
--
--
--
AR        5,2
LTØRG
M         4,DATØS
--
--
--
END
```

Al ser ensamblado el programa anterior, quedará en memoria la siguiente información:

```
AR        5,2
          =F'900'
          =H'52'
M         4,DATØS
```

Esto produce error en la ejecución, dado que, después de la instrucción AR 5,2 aparecen constantes. La forma de solucionar el problema será:

```
AR        5,2
BC        15,SIGA
LTØRG
SIGA M    4,DATØS
```

con lo cual se obtiene:

```

AR      5,2
BC      15,SIGA
        =F'900'
        =H'52'
SIGA M   4,DATØS

```

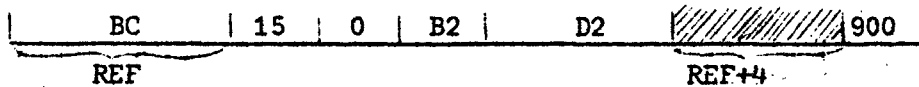
Si en el programa se desea hacer uso de la dirección de la constante 900 se puede utilizar como punto de referencia la instrucción BC 15,SIGA en la forma siguiente:

```

REF BC   15,SIGA
        LTØRG
SIGA M   4,DATØS
--
--
A       9,REF+4

```

Existe sin embargo, la posibilidad del problema siguiente: la instrucción BC 15,SIGA se genera a partir de una dirección múltiplo de dos, pero no de cuatro. Para generarse la constante 900 se deben saltar dos bytes. Luego al especificar A 9,REF+4 se indicará error de DIRECCIONAMIENTO dado que REF+4 no es múltiplo de 4.



Para evitar situaciones de este tipo, se hace uso de la pseudo instrucción Conditional No Operation (CNOP) que permite ajustar el contenido del Contador de Direcciones (Location Counter) a una posición determinada.

B. Pseudo instrucción Conditional no Operation

- a) Código : CNOP
- b) Formato : [símbolo] CNOP b,p
- c) Función : Se ajusta el contenido del Contador de Direcciones al alineamiento especificado por los operandos b y p donde:
 - b: indica el byte de un conjunto p de bytes
 - p: indica una palabra (4 bytes) o una doble palabra (8 bytes)

Los valores posibles son:

b,p	Indicación
0,4	comienzo de una palabra
2,4	comienzo de la 2a. mitad de la palabra
0,8	comienzo de una doble palabra
2,8	comienzo de la 2a. mitad de la 1a. palabra
4,8	comienzo de la 2a. palabra
6,8	comienzo de la 2a. mitad de la 2a. palabra.

Si el Contador de Direcciones está ajustado al alineamiento que se indica con CNOP, la pseudo instrucción no tiene efecto. En caso contrario, se generan tantas instrucciones de "no operación" como sea necesario (BCR 0,0)

Ejemplo 45.

14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29
2	3	0	1	2	3	0	1	2	3	0	1	2	3	0	1
6	7	0	1	2	3	4	5	6	7	0	1	2	3	4	5

Se puede observar que los bytes 0,4 corresponden a direcciones múltiplo de 4, y los bytes 0,8 a direcciones múltiplo de 8.

a) <LC> = 16

CNOP 0,4

la pseudo instrucción no tiene efecto

b) <LC> = 16

CNOP 2,4

se genera, a partir de 16, una instrucción de no operación

BCR	0	0
-----	---	---

16

c) <LC> = 16

CNOP 0,8

la pseudo instrucción no tiene efecto

d) <LC> = 16
 CNOP 6,8

se genera, a partir de 16, tres instrucciones de no operación

BCR	0	0	BCR	0	0	BCR	0	0
16								

La solución al problema planteado al tratar de direccionar literales será:

	CNØP	0,4
REF	BC	15, SIGA
	LTØRG	
SIGA	M	4, DATØS
	--	
	--	
	--	
A		9, REF+4

4.5. Pseudo Instrucciones de control de listado

Son utilizadas para colocar título a los listados de programas fuente, para insertar líneas o páginas en blanco, para imprimir las macro-instrucciones en detalle o sólo su nombre, etc. No generan instrucción de máquina, no se incluyen en el listado, excepto PRINT que aparece impresa.

A. Pseudo instrucción TITLE

- Código : TITLE
- Formato : [símbolo] TITLE 'uno a cien caracteres'
- Función : El símbolo está formado por 1 a 4 caracteres alfanuméricos. Dicho símbolo se perfora en las columnas 73-76 de todas las tarjetas de resultado de ensamble. Puede haber varias pseudo instrucciones TITLE en un programa, pero sólo la primera de ellas puede tener especificado el símbolo en el campo de nombre.

Los caracteres que aparecen entre apóstrofes se imprimen como título en la primera línea de cada página del listado del programa fuente. Si el conjunto de caracteres contiene apóstrofes o ampersands (&) ellos deben colocarse como un par, cada vez. Cada pseudo instrucción TITLE proporciona el encabezamiento para las páginas que le siguen. Exceptuando la primera

pseudo instrucción TITLE, las restantes implican el salto a una nueva página antes de continuar el listado.

Si el título excede la columna 71, se perfora carácter de continuación en la 72, y se reinicia el título en la columna 16 de la tarjeta siguiente.

B. Pseudo instrucción EJECT

- a) Código : EJECT
- b) Formato : blanco EJECT sin operandos
- c) Función : La pseudo instrucción EJECT causa que la siguiente línea del listado sea impresa como primera línea de la página siguiente. Si la siguiente línea es la primera de la página siguiente, la pseudo instrucción no tiene efecto. Para saltar n páginas debe especificarse n+1 EJECT seguidos.

C. Pseudo instrucción SPACE

- a) Código : SPACE
- b) Formato : blanco SPACE blanco o valor decimal
- c) Función : Se utiliza para insertar en el listado un número de líneas en blanco especificado por el valor decimal. Si no se especifica valor se inserta una línea en blanco. Si el valor especificado excede el número de líneas restantes en la página, tiene el efecto de una pseudo instrucción EJECT.

D. Pseudo instrucción PRINT

- a) Código : PRINT
- b) Formato : blanco PRINT $\left[\begin{array}{c} \{ \text{ON} \} \\ \{ \text{OFF} \} \end{array} \right] \left\{ \begin{array}{c} \text{GEN} \\ \text{NOGEN} \end{array} \right\} \left(\begin{array}{c} \text{DATA} \\ \text{NODATA} \end{array} \right)$

Observación: El paréntesis { } indica que hay que elegir uno de los operandos encerrados. Aquellos que aparecen subrayados, si no se colocan, se suponen.

- c) Función : De acuerdo a los operandos especificados se obtiene lo siguiente:
 - i) ON Se imprime el listado del programa fuente
 - ii) OFF No se imprime el listado
 - iii) GEN Se imprimen las macro-instrucciones en detalle, esto es, con todas las instrucciones que la componen

- iv) NOGEN Se imprime sólo la llamada de la macro en la forma siguiente:
[nombre] operación operandos
- v) DATA Se imprimen las constantes definidas en el programa en toda su longitud
- vi) NODATA Se imprimen solamente los 8 bytes de orden superior de las constantes.

4.6. Pseudo instrucciones de control del programa

Las pseudo instrucciones de control de programa permiten indicar final de ensamble, ajustar el contador de direcciones a un valor determinado, etc. Se han visto ya de este grupo, las pseudo instrucciones LTORG y CNOP. Otras pseudo instrucciones de importancia son END, ICTL, ISEQ entre las más usadas.

A. Pseudo instrucción END (Final de Ensamble)

- a) Código : END
- b) Formato : blanco END blanco o expresión reubicable
- c) Función : Indica el término de las instrucciones que van a ser ensambladas. También puede designar a través del operando, un punto en el programa traducido, o en otro, al cual se transferirá el control después que el programa sea cargado a memoria para su ejecución.

B. Pseudo instrucción ICTL. Control del formato de entrada (Input Format Control)

- a) Código : ICTL
- b) Formato : blanco ICTL uno a tres valores decimales en la forma c,t,r
- c) Función : La pseudo instrucción permite al programador, alterar el formato normal de las instrucciones de assembler, esto es, comienzo en columna 1, término en columna 71 y reiniciación en columna 16. Con los operandos c,t,r se redefine el comienzo, término y reiniciación.
 - i) c = comienzo. Debe especificarse siempre con valores 1 a 40
 - ii) t = término. Si se especifica debe tener valores 41 a 80, si no se especifica se supone 71. La columna que sigue a la especificada indica si hay tarjeta de continuación o no. Si se ha indicado 80 como columna de término, el compilador supone siempre que no hay tarjeta de continuación.

iii) r = reiniciación. Si se especifica debe tener valores 2 a 40 y debe ser mayor que c, además debe haberse indicado t.

Si no se coloca ICTL se supone 1,71 y 16 para comienzo, término y reiniciación respectivamente.

C. Pseudo instrucción ISEQ. Verificación de la secuencia de entrada (Input Sequence Checking)

a) Código : ISEQ

b) Formato : blanco ISEQ dos valores decimales en la forma i,d

c) Función : Permite verificar que las tarjetas del programa fuente están ordenadas secuencialmente en forma ascendente. Para ello se analiza el contenido del campo especificado con los dos valores decimales, que representan la columna del extremo izquierdo del campo y la columna del extremo derecho. El campo a verificar no debe estar entre las columnas de comienzo y término de la instrucción. El operando d debe ser mayor o igual que i. Al encontrarse en el programa una pseudo instrucción ISEQ sin operandos se pone término al chequeo iniciado con otro ISEQ, en caso de haberse especificado.

4.7. Detención de proceso

A. Instrucción SUPERVISOR CALL

a) Instrucción: SVC I

b) Formato : RR | SVC | I |

c) Función : La instrucción causa una interrupción de llamada a supervisor, que es el programa que controla el proceso. A pesar de que el formato es RR, los campos de registros son considerados como un solo operando el cual proporciona el código de interrupción. Uno de estos códigos de interrupción puede significar detención de proceso.

B. Macro-instrucción END of JOB

- a) Código de la macro: EOJ
- b) Formato : nombre EOJ sin operandos
- c) Función : Permite al programador especificar detención de proceso, sin necesidad de recordar el código de interrupción respectivo. La macro produce una instrucción SVC con el código adecuado para detención.

Puede especificarse nombre de tal manera que es posible indicar saltos a término de proceso usando como meta de salto el identificador de la macro.

4.8. Problemas propuestos

- a) Compilar el siguiente trozo de programa

```

START 0
BALR 15,0
USING GØ,15,14
GØ BC 15,SIGUE
AREA DS F
A DC F'15'
B DC F'18'
SIGUE LM 1,2,A
SRA 2,1
AR 1,2
SLDA 0,1
AR 1,2
ST 1,AREA

```

- b) Compilar el siguiente trozo de programa:

```

START 56
BALR 8,0
USING *,8
BC 15,BEGIN
JOSE DC 2F'64'
BEGIN L 2,JOSE
SRDA 2,JOSE

```


c) Suponiendo que el programa anterior se ejecute a partir de la dirección 56, ¿Qué queda en los RUGs 2 y 3?

d) Compilar el programa siguiente:

```

START 400
BALR 15,0
USING *,15
BC 15,DIEGØ
DIR1 DC A(DIEGØ+4)
DIR2 DC A(DIEGØ)
DIEGØ LM 0,1,DIR1
BALR 2,0
SR 2,1
ST 2,PEDRØ
LA 1,4(2,0)
BC 15,ØUT
PEDRØ DS F
ØUT EØJ
END

```

e) Al procesar el programa anterior, indicar qué queda en RUG1 y en PEDRØ, al finalizar el proceso, suponiendo que el programa se carga a partir de la dirección 400.

f) ¿Qué queda en los RUGs 1 y 2 al terminar el proceso siguiente?

```

START 256
BALR 15,0
USING *,15
SPACE 3
JUAN EQU 1
LA 1,*+2
LA 2,*+2
SR 2,1
SLA 1,JUAN

```

g) Indicar qué queda en los bytes DATOS+3, DATOS+5 y DATOS+7 al término del proceso siguiente:

```

      ICTL      1
PRØG  TITLE   'PRUEBA'
      START    X'2800'
      BALR     15,0
      ISEQ     73,75
      USING    *,15
      BALR     10,0
      BALR     11,0
      STM      10,11,DATØS
      BC       15,ØUT
DATØS DS      2F
ØUT   EØJ
      END

```

h) Suponiendo que el programa que figura a continuación pudiera ser procesado a partir de la dirección indicada en START, ¿Cual sería el contenido de los RUGs 1,2,3 y 4 al finalizar el proceso?

```

      START    400
      BALR     15,0
      USING    *,15
      BC       15,GØ
JUAN  DS      0H
PEDRØ DC      X'002'
DIEGØ DC      HL2'+3'
GØ    LH      1,DIEGØ
      LH      2,DIEGØ-2
      LH      3,PEDRØ
      LH      4,JUAN
      AR      4,3
      AR      4,2
      A       4,=A(JUAN)
      SR      4,1
      EØJ
      END

```

i) En las mismas condiciones del problema anterior, indicar qué queda en AREA después del siguiente proceso:

```

START 400
BALR 15,0
USING *,15,14
FIRST BC 15,BEGIN
AD1 DC A(FIRST+4096)
AREA DS 2F
BEGIN L 14,AD1
      L 1,=F'5'
      L 2,=A(*)
      LR 3,1
      BC 15 SIGUE
      LTORG
SIGUE AR 1,14
ØCHØ EQU 6
SRL 1,9
SR 2,1
SRL 2,ØCHØ
SRDL 2,96
STM 2,3,AREA
EØJ
END

```

5. Instrucciones lógicas

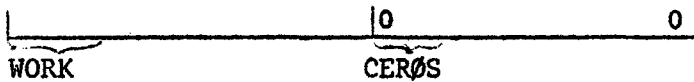
5.1. Instrucción MOVE CHARACTER

- a) Instrucción: MVC D1(L,B1),D2(B2)
- b) Formato : SS | MVC | L | B1 | D1 | B2 | D2 |
- c) Función : La información contenida a partir de la dirección D2(B2) es transferida al campo indicado por D1(B1). Se transfieren L bytes (máximo 256). La operación se realiza de izquierda a derecha a través de cada campo, byte por byte, los cuales no se cambian ni inspeccionan.

Ejemplo 46.

```

WØRK    DS    CL80
CERØS   DC    8CL10'0
        MVC   WØRK(80),CERØS
    
```



Observación: El símbolo WØRK tiene implícita, en su definición la longitud 80. Luego puede especificarse:

```

        MVC   WØRK,CERØS
    
```

con idéntico resultado.

Ejemplo 47.

```

CERØ    DC    X'0'
TABLA   DS    CL80
        MVC   TABLA,CERØ
    
```



En este caso, se propaga el byte CERØ (su contenido) a lo largo de todo el campo TABLA, dado que la instrucción opera de izquierda a derecha, byte por byte. Primero se transfiere el contenido del byte CERØ al byte TABLA, a continuación el contenido de TABLA, que ahora tiene ceros, al byte TABLA+1, y así sucesivamente.

5.2. Instrucción MOVE INMEDIATE

- Instrucción: MVI D1(B1),I2
- Formato : SI

MVI	I2	B1	D1
-----	----	----	----
- Función : El contenido del segundo byte de la instrucción (operando inmediato), se transfiere a la dirección D1(B1).

Ejemplo 48.

```

MVI     PEDRØ,C'A'
MVI     JUAN,X'C1'
MVI     DIEGØ,193
    
```

La primera instrucción transfiere a la dirección PEDRØ, el carácter A, que en binario es 11000001. La segunda instrucción transfiere a la dirección JUAN, el valor hexadecimal C1, que en binario es 11000001. La última instrucción transfiere a la dirección DIEGØ, el valor decimal 193, que en binario es 11000001. Esto es, tres formas distintas para obtener igual resultado.

Ejemplo 49.

Se desea dejar el campo BLANCØS con el carácter Ø. BLANCØS tiene longitud 10 bytes.

```
MVI    BLANCØS,C' Ø '
MVC    BLANCØS+1(9),BLANCØS
```

5.3. Instrucción MOVE NUMERICS

- a) Instrucción: MVN D1(L,B1),D2(B2)
- b) Formato : SS | MVN | L | B1 | D1 | B2 | D2
- c) Función : Los cuatro bits de orden inferior de cada byte del campo indicado por D2(B2) se transfieren a las posiciones de los cuatro bits de orden inferior de los correspondientes bytes del campo D1(B1). Se operan L bytes.

La transferencia es de izquierda a derecha a través de cada campo, byte por byte. Los campos pueden superponerse en cualquier forma que se desee.

La parte que se transfiere, parte numérica, no sufre cambio ni es sometida a verificación de validez. Los cuatro bits de orden superior de cada byte, parte zona, no sufren cambio.

Ejemplo 50.

```
DATØ1  DC    P'45722681'
DATØ2  DC    P'-38975402'
        MVN    DATØ1,DATØ2
```

0	3	8	9	7	5	4	0	2	-
DATØ2									
0	4	5	7	2	2	6	8	1	+
DATØ1									

resultado

0	3	5	9	2	5	6	0	1	-
---	---	---	---	---	---	---	---	---	---

5.4. Instrucción MOVE ZONES

- a) Instrucción: MVZ D1(L,B1),D2(B2)
- b) Formato : SS | MVZ | L | B1 | D1 | B2 | D2
- c) Función : Similar a la instrucción anterior, transfiere los bits de orden superior, parte zona. Los cuatro bits de orden inferior permanecen invariables en ambos operandos.

Ejemplo 51.

DATØ1 DC P'45722681'

DATØ2 DC P'-38975402'

MVZ DATØ1,DATØ2

0	3	8	9	7	5	4	0	2	-
DATØ2									
0	4	5	7	2	2	6	8	1	+
DATØ1									

resultado

0	4	8	7	7	2	4	8	2	+
---	---	---	---	---	---	---	---	---	---

5.5. Instrucción MOVE WITH OFFSET (aritmética decimal)

- a) Instrucción: MVO D1(L1,B1),D2(L2,B2)
- b) Formato : SS | MVO | L1 | L2 | B1 | D1 | B2 | D2
- c) Función : El segundo operando se transfiere al campo que está a la izquierda y contiguo a los cuatro bits de orden inferior del primer operando.

El procesamiento de los campos se realiza de derecha a izquierda, byte por byte. Si el campo del primer operando no contiene al segundo operando, la información excedente

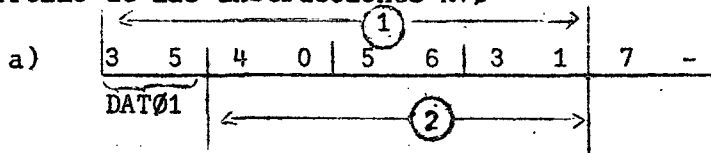
se ignora. El caso contrario, esto es, campo del primer operando más largo, produce la extensión del segundo operando mediante ceros de orden superior. Puede producirse superposición de campos, y su procesamiento se realiza almacenando un byte de resultado tan pronto como hayan sido extraídos los bytes necesarios, de los operandos.

Ejemplo 52.

```

START      0
BALR      15,0
USING     *,15
B         BEGIN
AREA      DS      PL20
DATØ1     DC      P'-354056317'
DATØ3     DC      P'182345903'
BEGIN     MVØ     DATØ1(4),DATØ1+1(3)      a)
          MVC     AREA(5),DATØ1
          MVØ     DATØ1+1(4),DATØ1(3)      b)
          MVC     AREA+5(5),DATØ1
          MVØ     DATØ3(2),DATØ3+1(3)      c)
          MVC     AREA+10(5),DATØ3
          MVØ     DATØ3(3),DATØ3+2(2)      d)
          MVC     AREA+15(5),DATØ3
          SVC     14
          END
    
```

Desarrollo de las instrucciones MVØ



- ① campo receptor
- ② campo que se transfiere

i) el byte del extremo derecho del campo receptor es DATØ1+3, y su contenido es 31.

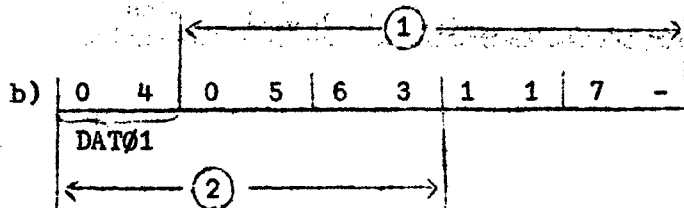
ii) los 4 bits de orden inferior de DATØ1+3 contienen el 1 que permanecerá inalterable.

iii) A la izquierda y adyacente al 1 queda el campo que se transfiere.

iv) resultado:

0	4	0	5	6	3	1	1	7	-
---	---	---	---	---	---	---	---	---	---

DATØ1

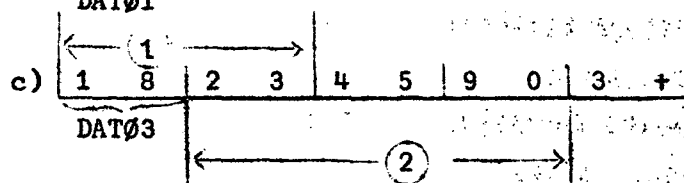


i) Todo el campo que se transfiere se coloca a la izquierda y adyacente al signo menos (byte del extremo derecho del campo receptor).

ii) resultado:

0	4	0	0	4	0	5	6	3	-
---	---	---	---	---	---	---	---	---	---

DATØ1

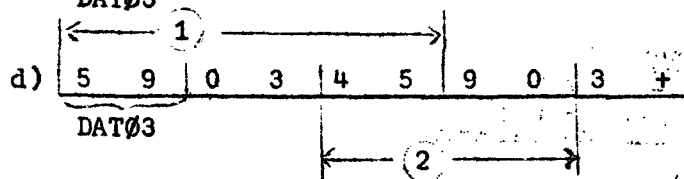


i) El campo receptor puede recibir sólo los dígitos 590, el resto se pierde.

ii) resultado:

5	9	0	3	4	5	9	0	3	+
---	---	---	---	---	---	---	---	---	---

DATØ3



i) El primer byte que se transfiere es DATØ3+3 que se coloca a la izquierda y adyacente al dígito 5.

ii) El segundo byte que se transfiere es DATØ3+2 cuya información original ha sido modificada en el paso anterior.

iii) resultado:

0	0	5	9	0	5	9	0	3	+
---	---	---	---	---	---	---	---	---	---

DATØ3

5.6. Instrucción INSERT CHARACTER

- a) Instrucción: IC R1, D2(X2, B2)
- b) Formato : RX

IC	R1	X2	B2	D2
----	----	----	----	----
- c) Función : El contenido del byte direccionado con D2(X2, B2) se inserta en el RUG especificado en R1, en los bits 24 a 31. Los bits restantes del registro no sufren cambio.

Ejemplo 53.

```

C255   DC   X'FF'
        SR   5,5
        IC   5,C255

```

Análisis:

- i) SR 5,5 deja el RUG en cero
- ii) IC 5,C255 inserta en el RUG5, en los bits 24 a 31 un byte con unos.
- iii) resultado

0				0011111111
0			24	31
- <RUG5> = 255

5.7. Instrucción STORE CHARACTER

- a) Instrucción: STC R1, D2(X2, B2)
- b) Formato : RX

STC	R1	X2	B2	D2
-----	----	----	----	----
- c) Función : El contenido de los bits 24 a 31 del RUG especificado en R1, se almacena en la dirección D2(X2, B2).

Ejemplo 54.

```

DATØS   DS   CL3
        --
        --
        --
        LA   5,150
        LA   6,45
        L    8,=F'247'
        STC  5,DATØS
        STC  6,DATØS+1
        STC  8,DATØS+2

```

Análisis:

i) Las instrucciones LA y L cargan en los RUG 5,6 y 8 los valores 150, 45 y 247 respectivamente. Todos contenidos en un byte.

ii) Las instrucciones STC almacenan esos valores en los tres bytes del área DATØS.

iii) Resultado :

150	45	247
DATØS DATØS+1 DATØS+2		

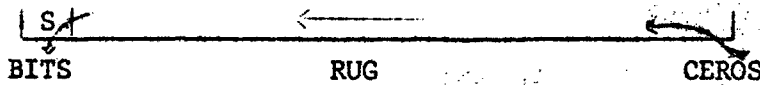
5.8. Instrucción SHIFT LEFT SINGLE LOGICAL

a) Instrucción: SLL R1,D2(B2)

b) Formato : RS

SLL	R1	B2	D2
-----	----	----	----

c) Función : Se desplaza el <RUG> especificado en R1 de acuerdo a los seis bits de orden inferior de la representación binaria de la dirección obtenida con D2(B2). Se desplaza toda la información, incluido el signo. Los bits de orden superior desaparecen. Los bits de orden inferior que quedan libres quedan con ceros.



Ejemplo 55.

LA 5,15
SLL 5,257

Análisis:

i) Se carga en RUG5 el valor 15.

0	0	0	0	0	0	1	1	1	1
---	---	---	---	---	---	---	---	---	---

ii) La representación binaria de 257 es

1 0 0 | 0 0 0 0 0 1
 6 bits

iii) Los seis bits de orden inferior indican una posición de desplazamiento a la izquierda.

iv) Resultado:

0	0	0	0	0	0	1	1	1	1	0
---	---	---	---	---	---	---	---	---	---	---

<RUG5> = 30

5.9. Instrucción SHIFT RIGHT SINGLE LOGICAL

a) Instrucción: SRL R1,D2(B2)

b) Formato : RS

SRL	R1		B2	D2
-----	----	--	----	----

c) Función : Similar a la instrucción SLL, desplaza la información hacia la derecha. Los bits de orden inferior desaparecen. Los bits de orden superior que quedan libres quedan con ceros. El signo se desplaza junto con el resto de la información.



5.10. Instrucción SHIFT LEFT DOUBLE LOGICAL

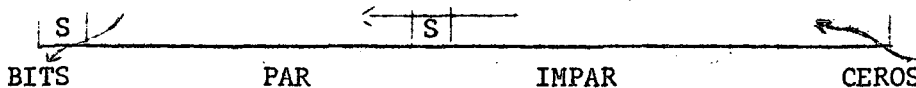
a) Instrucción: SLDL R1,D2(B2)

b) Formato : RS

SLDL	R1		B2	D2
------	----	--	----	----

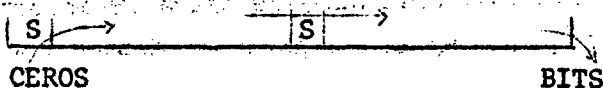
c) Función : El contenido de los RUG PAR e IMPAR siguiente, especificados por el operando R1, es desplazado a la izquierda. En R1 se especifica un RUG PAR. Los lugares de desplazamiento se indican por los seis bits de orden inferior de la representación binaria de la dirección dada por D2(B2).

Los signos de ambos RUG se desplazan como parte de la información. Desaparecen bits por la izquierda del RUG PAR, y los lugares vacantes a la derecha del RUG IMPAR se ocupan con ceros.



5.11. Instrucción SHIFT RIGHT DOUBLE LOGICAL

- a) Instrucción: SRDL R1,D2(B2)
- b) Formato : RS | SRDL | R1 | B2 | D2
- c) Función : Similar a la instrucción SLDL desplaza información hacia la derecha. Los bits de orden inferior del RUG IMPAR desaparecen. Los bits de orden superior del RUG PAR que quedan libres se llenan con ceros. El signo se desplaza junto con el resto de la información.



5.12. Instrucción COMPARE LOGICAL CHARACTER

- a) Instrucción: CLC D1(L,B1),D2(B2)
- b) Formato : SS | CLC | L | B1 | D1 | B2 | D2
- c) Función : Compara el primer operando, ubicado en la dirección D1(D1), con el segundo operando, almacenado en D2(B2). La operación se realiza de izquierda a derecha a través de L bytes. La comparación es binaria y todos los códigos son válidos. Al detectarse desigualdad o término del campo, la operación se detiene y se genera Código de Condición.

<u>Código de Condición</u>	<u>Resultado</u>
0	Operando 1 = Operando 2
1	Operando 1 < Operando 2
2	Operando 1 > Operando 2

Se pueden comparar todo tipo de caracteres y el resultado será de acuerdo al siguiente valor relativo de menor a mayor:

∅ < caract. especiales < caract. alfabéticos < caract. numéricos

5.13. Instrucción COMPARE LOGICAL IMMEDIATE

- a) Instrucción: CLI D1(B1),I2
- b) Formato : SI | CLI | I2 | B1 | D1
- c) Función : Compara el byte indicado por la dirección D1(B1) con el operando inmediato I2. De acuerdo al resultado se genera Código de Condición.*

5.14. Instrucción COMPARE LOGICAL

- a) Instrucción: CL R1,D2(2X,B2)
- b) Formato : RX

CL	R1	X2	B2	D2
----	----	----	----	----
- c) Función : Compara el <RUG> especificado en R1 con la palabra indicada con D2(X2,B2). La comparación es binaria. La operación procede de izquierda a derecha y finaliza al detectar desigualdad o término de los operandos. De acuerdo al resultado se genera Código de Condición.*

5.15. Instrucción COMPARE LOGICAL REGISTER

- a) Instrucción: CLR R1,R2
- b) Formato : RR

CLR	R1	R2
-----	----	----
- c) Función : Se compara el <RUG> especificado en el campo R1 con el <RUG> indicado en R2. No se considera el signo como tal. De acuerdo al resultado se genera Código de Condición.

* Los Códigos de Condición generados son los indicados en la instrucción Compare Logical Character.

Ejemplo 56. Ejemplos de instrucciones.

```
CLC    MASTER1(20),TRANS
CLI    LARGØ,16
CLI    BYTE,X'5'
CLI    ALFA,C'S'
CL     0,MASK
CLR    1,2
```

5.16. Instrucción TEST UNDER MASK

- a) Instrucción: TM D1(B1),I2
- b) Formato : SI

TM	I2	B1	D1
----	----	----	----
- c) Función : El estado de los bits del byte cuya dirección es B1(D1) se investiga de acuerdo al contenido del operando inmediato I2 (máscara). La máscara se construye colocando "unos" en las posiciones que se desea investigar y "ceros" en las que no interesan. El contenido del byte analizado no se altera. De acuerdo al resultado se genera Código de Condición.

<u>Código de Condición</u>		<u>Resultado</u>
Decimal	Binario	
0	00	Todos ceros (OFF)
1	01	Mixto
3	11	Todos unos (ON)

Ejemplo 57.

```

BYTE      DC      X'54'
-----
--
--
--
TM        BYTE,X'4'

```

La constante de nombre BYTE tiene como contenido:

```

  0 1 0 1 0 1 0 0
  └────────────────┘
    BYTE

```

la máscara utilizada (X'4') tiene la estructura:

```

  0 0 0 0 0 1 0 0
  └────────────────┘
  0 1 2 3 4 5 6 7

```

esto es, permitirá averiguar el contenido del bit 5. El Código de Condición generado es 3 (todos unos).

5.17. Instrucciones: AND, OR y EXCLUSIVE OR

Todas las instrucciones se realizan bit por bit. El segundo operando actúa sobre el primer operando. El resultado reemplaza al primer operando. De acuerdo al resultado se generan los siguientes posibles Códigos de Condición:

<u>Código de Condición</u>	<u>Resultado</u>
0	Es cero
1	Es distinto de cero

a) AND: Si el bit considerado en el primer operando es 1, y el bit del segundo operando es 1, el bit resultante es 1. En cualquier otro caso es 0.

b) OR: Si el bit considerado en el primer operando es 1, o el bit del segundo operando es 1, o ambos son 1, el bit resultante es 1. Ambos deberán ser 0 para que el resultado sea 0.

c) XOR: (Exclusive OR). Si el bit considerado en el primer operando es 1, 0 el bit del segundo operando es 1, pero NO son 1 ambos, el bit resultante es 1. En caso contrario, o si ambos son 0, el bit resultante es 0.

Ejemplo 58.

	AND	OR	XOR
1er. op.	00000011	00000011	00000011
2ºop.	00000101	00000101	00000101
result.	00000001	00000111	00000110

A. Instrucción AND REGISTER

a) Instrucción: NR R1,R2

b) Formato : RR

NR	R1	R2
----	----	----

B. Instrucción AND

a) Instrucción: N R1,D2(X2,B2)

b) Formato : RX

N	R1	X2	B2	D2
---	----	----	----	----

C. Instrucción AND IMMEDIATE

a) Instrucción: NI D1(B1),I2

b) Formato : SI

NI	I2	B1	D1
----	----	----	----

D. Instrucción AND CHARACTER

a) Instrucción: NC D1(L,B1),D2(B2)

b) Formato : SS

NC	L	B1	D1	B2	D2
----	---	----	----	----	----

E. Instrucción OR REGISTER

a) Instrucción: OR R1,R2

b) Formato : RR

OR	R1	R2
----	----	----

F. Instrucción OR

a) Instrucción: O R1,D2(X2,B2)

b) Formato : RX

O	R1	X2	B2	D2
---	----	----	----	----

G. Instrucción OR INMEDIATE

- a) Instrucción: OI D1(B1), I2
- b) Formato: SI

OI	I2	B1	D1
----	----	----	----

H. Instrucción OR CHARACTER

- a) Instrucción: OC D1(L, B1), D2(B2)
- b) Formato: SS

OC	L	B1	D1	B2	D2
----	---	----	----	----	----

I. Instrucción EXCLUSIVE OR REGISTER

- a) Instrucción: XR R1, R2
- b) Formato: RR

XR	R1	R2
----	----	----

J. Instrucción EXCLUSIVE OR

- a) Instrucción: X R1, D2(X2, B2)
- b) Formato: RX

X	R1	X2	B2	D2
---	----	----	----	----

K. Instrucción EXCLUSIVE OR INMEDIATE

- a) Instrucción: XI D1(B1), I2
- b) Formato: SI

XI	I2	B1	D1
----	----	----	----

L. Instrucción EXCLUSIVE OR CHARACTER

- a) Instrucción: XC D1(L, B1), D2(B2)
- b) Formato: SS

XC	L	B1	D1	B2	D2
----	---	----	----	----	----

Del análisis de las instrucciones se puede concluir lo siguiente:

Instrucciones AND permiten:

- i) Mantener los contenidos de los bits del primer operando, colocando unos en los bits respectivos del segundo operando.
- ii) Dejar en cero (OFF) los bits del primer operando, colocando ceros en los bits respectivos del segundo operando.

Instrucciones OR permiten:

- i) Mantener los contenidos de los bits del primer operando, colocando ceros en los bits respectivos del segundo operando.

Instrucciones Exclusive OR permiten:

i) Mantener los contenidos de los bits del primer operando colocando ceros en los bits respectivos del segundo operando.

ii) Cambiar los contenidos de los bits del primer operando colocando unos en los bits respectivos del segundo operando.

Ejemplo 59.

BYTE DC X'FO'

a) Dejar los bits 1 y 4 en OFF

NI BYTE,X'B7'

b) Dejar los bits 1 y 4 en ON

ØI BYTE,X'48'

c) Cambiar de estado los bits 1 y 4

XI BYTE,X'48'

a) 11110000

b) 11110000

c) 11110000

10110111
10110000

01001000
11111000

01001000
10111000

5.18. Instrucción TRANSLATE AND REPLACE

a) Instrucción: TR D1(L,B1),D2(B2)

b) Formato : SS

TR	L	B1	D1	B2	D2
----	---	----	----	----	----

c) Función : Se traduce la lista de longitud L bytes ubicada a partir de la dirección D1(B1), de acuerdo a una tabla de traducción cuya dirección inicial es D2(B2). Los bytes del primer operando reciben el nombre de bytes del argumento y los bytes de la tabla se denominan bytes de función.

Los bytes del primer operando se seleccionan uno a uno para su traducción, procediendo de izquierda a derecha. Interpretados como número binario se suman a la dirección D2(B2) y el resultado obtenido se usa como dirección del byte de función el cual reemplaza al byte de argumento.

Ejemplo 60.

Convertir todos los caracteres alfabéticos y especiales del código EBCDIC en ceros y los caracteres numéricos en su complemento.

```

CØNVERT START 0
          BALR  15,0
          USING *,15
          --
          --
          TR    DATØS,TABLA
          --
          --
          EØJ
          DATØS DS    CL120
          TABLA DC    240X'F0'
                   DC    X'FAF9F8F7F6F5F4F3F2F1'
                   DC    6X'F0'
          END

```

El valor máximo que puede contener un byte es 255, de tal manera que la tabla debe tener una longitud de 256 bytes. Si aparece un carácter 0, cuya representación es F0, o lo que es lo mismo 11110000, el valor decimal correspondiente a este número binario, que es 240, se sumará a TABLA y dará la dirección del byte que contiene FA, valor que reemplazará al F0. Si aparece un carácter 9, cuya representación es F9, o lo que es lo mismo 11111001, el valor decimal correspondiente a este número binario que es 249 se sumará a TABLA y dará la dirección del byte que contiene F1, valor que reemplazará al F9. Cualquier otro carácter será reemplazado por 0.

5.19. Instrucción TRANSLATE AND TEST

a) Instrucción: TRT D1(L,B1),D2(B2)

b) Formato : SS | TRT | L | B1 | D1 | B2 | D2 |

c) Función : Se analiza la lista de longitud L bytes ubicada a partir de la dirección D1(B1), de acuerdo a una tabla cuya dirección inicial es D2(B2). Los bytes del primer operando reciben el nombre de bytes del argumento y los bytes de la tabla se denominan bytes de función.

Los bytes del primer operando se seleccionan uno por uno para su análisis, procediendo de izquierda a derecha. Interpretados como número binario se suman a la dirección D2(B2) y el resultado obtenido se usa como dirección del byte de función. Si el byte de función es cero, el análisis continúa, en caso contrario, la dirección del argumento correspondiente se inserta en el RUG1 y el byte de función en el RUG2, en los ocho bits de orden inferior. Los bits 0-23 del RUG2 no sufren cambio como tampoco los ocho bits de orden superior del RUG1.

Se pueden generar los siguientes códigos de condición:

<u>Código de Condición</u>	<u>Resultado</u>
0	Todos los bytes de función son ceros
1	Se encuentra un byte de función distinto de cero y quedan bytes del argumento por analizar.
2	Se encuentra un byte de función distinto de cero y no quedan bytes del argumento por analizar.

Ejemplo 61.

Contabilizar los blancos contenidos en la información perforada en una tarjeta.

```

ANALIS  START  0
          BALR  15,0
          USING *,15
          --
          --
          SR    3,3
          LA    4,80
          LA    5,TARJ
STORE    STC   4,TRANS+1
TRANS    TRT   5(0),TABLA
          BC    8,ALMAC
          LA    3,1(0,3)

```

(Continúa)

(Continuación)

```

BC      2,ALMAC
SR      1,5
LA      1,1(0,1)
LR      5,1
SR      4,1
BC      15,STØRE
ALMAC  ST      3,RESULT
EQJ
RESULT DS      F
TARJ   DS      CL80
TABLA  DC      64X'0'
        DC      X'40'
        DC      191X'0'
END

```

Se define una TABLA de 256 bytes de longitud que tiene ceros en todas las posiciones excepto en la 65 que contiene un carácter blanco (podría ser cualquier carácter distinto de cero). El RUG3 que llevará la cuenta de los blancos se deja en cero inicialmente; en el RUG4 se carga la longitud del campo que se analizará y en el RUG5 se carga la dirección de partida TARJ de dicho campo. A continuación se almacena la longitud contenida en el byte inferior del RUG4 en la dirección TRANS+1 que corresponde al byte que especifica la longitud en la instrucción TRANSLATE AND TEST.

Si todos los bytes de función son ceros, se salta inmediatamente a almacenar el resultado, en caso contrario, se aumenta el RUG en 1 y se modifica longitud y dirección del primer operando para continuar el análisis del resto del campo si se ha producido código de condición 4 o terminar el proceso si el código ha sido 2.

5.20. Instrucción EDIT

a) Instrucción: ED D1(L,B1),D2(B2)

b) Formato : SS

ED	L	B1	D1	B2	D2
----	---	----	----	----	----

c) Función : Los datos ubicados a partir de la dirección D2(B2) que están en formato empaquetado, se imprimen de acuerdo a los caracteres contenidos en el campo D1(B1) denominado PATRON. En la impresión es posible suprimir ceros no significativos, insertar comas y puntos decimales, insertar el signo menos o el símbolo de crédito, etc. El resultado reemplaza al patrón. Los caracteres fundamentales de éste son:

<u>Carácter</u>	<u>Significado</u>	<u>Representación</u>	
		<u>Binaria</u>	<u>Hexadecimal</u>
d	Seleccionador de dígitos	00100000	X'20'
(Iniciador de significación	00100001	X'21'
)	Separador de campos	00100010	X'22'

i) El carácter seleccionador de dígitos determina que un dígito del dato o un carácter de relleno sea insertado en el campo de resultado.

ii) El carácter iniciador de significación cumple la misma función del anterior, pero deja indicado que los dígitos que le siguen son significativos.

iii) El carácter separador de campos identifica cada uno de los campos debiendo construirse el patrón como para un campo nuevo. Se reemplaza por carácter de relleno.

iv) El carácter de relleno puede ser cualquiera y es el primer carácter del área patrón. Normalmente se utiliza:

∅ = blanco, cuya representación en binario es 01000000 y en hexadecimal es X'40'. Otro carácter posible es:

* = asterisco, cuya representación en binario es 01011100 y en hexadecimal es X'5C'.

v) El proceso se realiza de izquierda a derecha, un carácter por vez. El carácter a almacenar depende de tres factores:

El dígito de datos

El carácter del área patrón

El estado de un interruptor (switch) llamado trigger S

De acuerdo a ellos se puede obtener:

Expansión del dígito a formato zona

Dejar sin cambio el carácter del área patrón

Almacenar un carácter de relleno

vi) El trigger S se pone en cero al comienzo de la operación y luego cambia según los dos primeros factores indicados en el punto v).

vii) Cualquier código de signo positivo pone el trigger S en 0. Los códigos negativos lo dejan sin cambio.

La tabla que figura a continuación resume todos los aspectos anteriores:

AREA PATRON	d d d d ((((N N))	O O T T R R O O
DATO FUENTE	0 0 ≠ ≠ 0 0 ≠ ≠ N N N N N N 0 0 A A A A A A	
TRIGGER S	0 1 0 1 0 1 0 1 0 1 0 1 0 1	
ACCION	/ / / / / / / / / / / / / / / /	
ALMACENA DIGITO FUENTE EN PATRON	X X X X X X X	
DEJA CARACTER PATRON SIN CAMBIO		X X
ALMACENA CARACTER DE RELLENO	X X X X X X X	
ESTADO DEL TRIGGER DESPUES DE LA OP.	0 1 1 1 1 1 1 1 0 1 0 0 0 1	

Ejemplo 62.

Se tienen tres datos en formato empaquetado, cada uno ocupa cuatro bytes. Se pide eliminar los ceros no significativos, editar cada dato con punto decimal separando dos dígitos de la parte fraccionaria, de la parte entera, además, si el dato es negativo editar a continuación de él la palabra NEG.

Usar el carácter blanco como carácter de relleno.

```
EDIC   START 0
      BALR 15,0
      USING *,15
      ---
      ED PATRON(42),DATOS
      ---
      EOL
DATOS  DC PL4'-10541'
      DC PL4'48539'
      DC PL4'-3276814'
PATRON DC 3X'4020202020214B202040D5C5C722'
      END
```

El resultado que se obtiene es el siguiente:

~~000~~105.41~~0000~~NEG~~0000~~485.39~~0000~~32768.14~~0000~~NEG

5.21. Instrucción EDIT AND MARK

- a) Instrucción: EDMK D1(L,B1),D2(B2)
- b) Formato : SS

EDMK	L	B1	D1	B2	D2
------	---	----	----	----	----
- c) Función : La operación es idéntica a la de la instrucción EDIT pero tiene una función adicional que consiste en insertar en el RUG1 la dirección del primer dígito significativo. Se puede colocar un carácter de protección de cifra en la ubicación dada por contenido de RUG1 menos uno. Si la significación es forzada por un carácter de iniciación de significación, la dirección del primer dígito significativo no se almacena.

Debe colocarse entonces, como medida de seguridad, la dirección del carácter que sigue al carácter de iniciación de significación en el RUG1, antes de ejecutar la instrucción EDIT AND MARK. Si se fuerza la significación, la dirección cargada permanecerá en el RUG1, en caso contrario, la dirección del primer dígito significativo se almacenará en el RUG1.

Si se utiliza una sola instrucción para editar varios datos, como en el ejemplo 62, sólo la dirección del primer dígito significativo del último dato estará disponible en el RUG1 después de ejecutar la instrucción.

Ejemplo 63.

Editar los dos primeros datos del ejemplo 62 con el símbolo \$ como carácter protector de cifra.

```

EDIM   START 0
      BALR 15,0
      USING *,15
      LA 2,DATØS
      LA 3,0
MUEVE  MVC PATRØ1,PATRØN
      LA 1,PATRØ1+6
      EDMK PATRØ1,0(2)
      S 1,UNØ
      MVC 0(1,1),PESØ
      --
      --
      LA 2,4(0,2)
      LA 3,1(0,3)
      C 3,DØS
      BC 4,MUEVE
      EQU
UNØ    DC F'1'
DØS    DC F'2'
PESØ   DC C'$'

```

(Continúa)

(Continuación)

```

DATØS  DC  PL4' -10541'
        DC  PL4' 48539'
PATRØN DC  X'4020202020214B202040D5C5C722'
PATRØ1 DS  XL14
        END

```

Dado que es necesario editar un dato por vez, se carga inicialmente el RUG2 con la dirección DATOS con el objeto de poder modificar dicha dirección programando una sola instrucción EDMK. El RUG3 se carga con cero y servirá de control de ciclo. El PATRON es necesario moverlo a PATRO1 pues el resultado que se obtiene lo destruye cada vez que se ejecuta EDMK. En RUG1 se carga la dirección donde está ubicado el carácter de significación más uno. Esto permite, después de ejecutar EDMK, restar uno al contenido de RUG1 y mover a esa dirección el carácter protector de cifra.

Las instrucciones siguientes permiten reiniciar el proceso para editar la información contenida en DATOS+4.

6. Códigos mnemotécnicos ampliados

En las instrucciones de bifurcación utilizadas, de acuerdo al formato de ella, es necesario especificar como primer operando, la máscara relacionada con los códigos de condición producidos, después de instrucciones aritméticas o de comparación. Esto obliga al programador a memorizar las relaciones existentes entre máscara y código de condición y el significado de éste. Una facilidad proporcionada por el Sistema IBM/360/370 es el conjunto de códigos mnemotécnicos ampliados que permiten al programador recordar con mayor facilidad las instrucciones correspondientes a distintas condiciones de bifurcación.

A. Salto incondicional y no operación

<u>Instrucción</u>	<u>Significado</u>	<u>Código ampliado</u>
BC 15,D2(X2,B2)	Salto incondicional	B D2(X2,B2)
BCR 15,R2	Salto incondicional	BR R2
BC 0,D2(X2,B2)	No operación	NOP D2(X2,B2)
BCR 0,R2	No operación	NOPR R2

B. Instrucciones después de operaciones aritméticas. (Se analiza el resultado)

<u>Instrucción</u>	<u>Significado</u>	<u>Código ampliado</u>
BC 13, D2(X2, B2)	Salto si no es > 0	BNP D2(X2, B2)
BC 11, D2(X2, B2)	Salto si no es < 0	BNM D2(X2, B2)
BC 8, D2(X2, B2)	Salto si es = 0	BZ D2(X2, B2)
BC 7, D2(X2, B2)	Salto si no es = 0	BNZ D2(X2, B2)
BC 4, D2(X2, B2)	Salto si es < 0	BM D2(X2, B2)
BC 2, D2(X2, B2)	Salto si es > 0	BP D2(X2, B2)
BC 1, D2(X2, B2)	Salto si overflow	BO D2(X2, B2)
BCR 13, R2	Salto si no es > 0	BNPR R2
BCR 11, R2	Salto si no es < 0	BNMR R2
BCR 8, R2	Salto si es = 0	BZR R2
BCR 7, R2	Salto si no es = 0	BNZR R2
BCR 4, R2	Salto si es < 0	BMR R2
BCR 2, R2	Salto si es > 0	BPR R2
BCR 1, R2	Salto si overflow	BOR R2

C. Instrucciones después de comparación. (A comparado con B)

<u>Instrucción</u>	<u>Significado</u>	<u>Código ampliado</u>
BC 13, D2(X2, B2)	Salto si A no mayor que B	BNH D2(X2, B2)
BC 11, D2(X2, B2)	Salto si A no menor que B	BNL D2(X2, B2)
BC 8, D2(X2, B2)	Salto si A es igual a B	BE D2(X2, B2)
BC 7, D2(X2, B2)	Salto si A no es igual a B	BNE D2(X2, B2)
BC 4, D2(X2, B2)	Salto si A es menor que B	BL D2(X2, B2)
BC 2, D2(X2, B2)	Salto si A es mayor que B	BH D2(X2, B2)
BCR 13, R2	Salto si A no mayor que B	BNHR R2
BCR 11, R2	Salto si A no menor que B	BNLR R2
BCR 8, R2	Salto si A es igual a B	BER R2
BCR 7, R2	Salto si A no es igual a B	BNER R2
BCR 4, R2	Salto si A es menor que B	BLR R2
BCR 2, R2	Salto si A es mayor que B	BHR R2

D. Instrucciones después de TEST UNDER MASK

	<u>Instrucción</u>	<u>Significado</u>	<u>Código ampliado</u>
BC	14,D2(X2,B2)	Salto si no hay unos	BNO D2(X2,B2)
BC	8,D2(X2,B2)	Salto si hay ceros	BZ D2(X2,B2)
BC	4,D2(X2,B2)	Salto si resultado mixto	BM D2(X2,B2)
BC	1,D2(X2,B2)	Salto si hay unos	BO D2(X2,B2)
BCR	14,R2	Salto si no hay unos	BNOR R2
BCR	8,R2	Salto si hay ceros	BZR R2
BCR	4,R2	Salto si resultado mixto	BMR R2
BCR	1,R2	Salto si hay unos	BOR R2

7. Aritmética Decimal

La aritmética decimal opera con datos que están en el formato "empaquetado" (PACKED) en el cual cada byte contiene dos dígitos decimales, excepto el byte del extremo derecho del campo que contiene un dígito y el signo. De acuerdo al formato, cada dato se interpreta como si fuera entero, de tal manera que, si el programador desea resultados reales, esto es, con parte entera y parte fraccionaria, es de su responsabilidad ubicar correctamente el punto decimal, imaginario, y operar de acuerdo a dicha ubicación.

7.1. Instrucción ZERO AND ADD

- a) Instrucción: ZAP D1(L1,B1),D2(L2,B2)
- b) Formato : SS

ZAP	L1	L2	B1	D1	B2	D2
-----	----	----	----	----	----	----
- c) Función : El segundo operando, que se encuentra en la dirección D2(B2) y tiene una longitud de L2 bytes, se ubica en la dirección D1(B1). La operación es equivalente a sumar una cantidad a cero.

Si el campo del primer operando es demasiado corto como para contener todos los dígitos significativos del segundo operando, se produce un desborde (overflow) decimal y como consecuencia de esto una interrupción de programa. Si el campo del primer operando es más largo que el necesario para contener el segundo operando, se rellena con ceros por la izquierda.

La operación se realiza de derecha a izquierda, byte por byte, de tal manera que puede haber superposición de campos siempre que los respectivos bytes del extremo derecho coincidan o que el del primer operando quede a la derecha del byte extremo del segundo operando.

Ejemplo 64.

```

--
AREA1 DS PL4
DATØ DC PL5' -4897'
--
ZAP AREA1, DATØ
--
EØJ
END

```

i) Areas antes de ejecutar ZAP

				0	0	0	4	8	9	7	-
AREA1				DATØ							

ii) Areas después de ejecutar ZAP

0	0	0	4	8	9	7	-	0	0	0	0	0	4	8	9	7	-
AREA1								DATØ									

Aún cuando la longitud del primer operando es menor que la del segundo, puede contener todos los dígitos significativos de éste, luego no hay desborde y tampoco interrupción de programa.

7.2. Instrucción ADD DECIMAL

- a) Instrucción: AP D1(L1,B1),D2(L2,B2)
- b) Formato : SS

AP	L1	L2	B1	D1	B2	D2
----	----	----	----	----	----	----
- c) Función : El segundo operando se suma al primer operando, el resultado reemplaza al primer operando. La suma es algebraica.

Si el campo del primer operando es demasiado corto como para contener todos los dígitos significativos del resultado, se produce un desborde (overflow) decimal y como consecuencia de esto una interrupción de programa. Si el campo del

primer operando es más largo que el necesario para contener el resultado, se rellena con ceros por la izquierda.

Los campos pueden superponerse siempre que los respectivos bytes del extremo derecho coincidan. Esto significa que es posible sumar una cantidad a sí misma.

Los códigos de condición que se generan son los mismos que se producen en "Aritmética de punto fijo".

<u>Código de Condición</u>	<u>Resultado</u>
0	igual a cero
1	menor que cero
2	mayor que cero
3	desborde (overflow)

Ejemplo 65.

Suma de un número a sí mismo

```

--
AP ALFA,ALFA
--
EØJ
ALFA DC PL5'179253'
END
    
```

El resultado en ALFA será:

0	0	0	3	5	8	5	0	6	+
ALFA									

7.3. Instrucción SUBTRACT DECIMAL

- a) Instrucción: SP D1(L1,B1),D2(L2,B2)
- b) Formato : SS

SP	L1	L2	B1	D1	B2	D2
----	----	----	----	----	----	----
- c) Función : Se resta al primer operando, el segundo operando, el resultado reemplaza al primer operando.

La instrucción SUBTRACT DECIMAL opera igual que ADD DECIMAL, salvo que se invierte el signo del segundo operando después de haberse extraído del almacenamiento y antes de realizar la operación aritmética.

7.4. Instrucción MULTIPLY DECIMAL

- a) Instrucción: MP D1(L1,B1),D2(L2,B2)
- b) Formato : SS

MP	L1	L2	B1	D1	B2	D2
----	----	----	----	----	----	----
- c) Función : El producto entre multiplicador (segundo operando) y multiplicando (primer operando) reemplaza al multiplicando. La longitud máxima del multiplicador puede ser de ocho bytes (siete en la instrucción de máquina) y corresponde a quince dígitos y signo, además, debe ser inferior a la del multiplicando, en caso contrario se produce una interrupción de programa.

Como la cantidad de dígitos del producto puede ser, como máximo, igual a la suma de la cantidad de dígitos de los operandos, el multiplicando debe tener tantos ceros de orden superior, no significativos, como dígitos tenga el multiplicador. Esta definición del campo que contiene al multiplicando da la seguridad de que no se producirá desborde.

La longitud máxima del producto es de dieciseis bytes (quince en la instrucción de máquina) y corresponde a treinta y un dígitos y signo.

Puede haber superposición de campos, siempre que los respectivos bytes del extremo derecho coincidan, lo que permite multiplicar un dato consigo mismo.

Ejemplo 66.

Se tienen tres variables:

COSTO	cuya configuración es DDD,DDS	5 dígitos y signo
DESCUENTO	cuya configuración es DDS	2 dígitos y signo
CANTIDAD	cuya configuración es DDDS	3 dígitos y signo

Calcular:

$$TOTAL = COSTO * DESCUENTO * CANTIDAD$$

El resultado se pide ajustado a un dígito de parte fraccionaria (la coma que aparece en las configuraciones de los datos no se almacena).

i) La cantidad máxima de dígitos que puede tener el resultado es 10, luego el campo TOTAL debe definirse con longitud de 6 bytes.

ii) La cantidad de dígitos que tendrá la parte fraccionaria será igual a 4, luego, si se quiere redondear a un dígito debe sumarse el valor 5 al dígito de orden inmediatamente inferior. En este caso se define una constante de valor 5000 en la que el dígito cero del extremo derecho es reemplazado por el signo del resultado total.

```

MULTI  START 0
        BALR 15,0
        USING *,15
        --
        --
        ZAP TØTAL,CØSTØ
        MP  TØTAL,DESCTØ
        MP  TØTAL,CANTID
        MVN AJUSTE+1(1),TØTAL+5
        AP  TØTAL,AJUSTE
        MVØ TØTAL(6),TØTAL(4)
        ED  PATRØN,TØTAL+2
        --
        --
        EØJ
AJUSTE DC X' 5000'
TØTAL  DS CL6
CØSTØ  DS CL3
DESCTØ DS CL2
CANTID DS CL2
PATRØN DC X'402020202020214B204022'
        END
    
```

A continuación se describe el efecto producido por cada instrucción:

ZAP	TØTAL,CØSTØ
0 0 0 0 0 0	D D D D D S

TOTAL

MP	TØTAL,DESCTØ
0 0 0	D D D D D D D S

TOTAL

MP TOTAL, CANTID

O	D	D	D	D	D	D	D	D	D	D	S
---	---	---	---	---	---	---	---	---	---	---	---

TOTAL

MVN AJUSTE+1(1), TOTAL+5

5	0	0	0
---	---	---	---

AJUSTE

AP TOTAL, AJUSTE

O	D	D	D	D	D	D	D	D	D	D	S
---	---	---	---	---	---	---	---	---	---	---	---

5	0	0	S
---	---	---	---

AJUSTE

SE REDONDEA

MVØ TOTAL(6), TOTAL(4)

O	D	D	D	D	D	D	D	D	D	D	S
---	---	---	---	---	---	---	---	---	---	---	---

O	O	O	O	D	D	D	D	D	D	D	S
---	---	---	---	---	---	---	---	---	---	---	---

TOTAL

ED PATRØN, TOTAL+2

Ø	D	D	D	D	D	D	.	D	Ø	Ø
---	---	---	---	---	---	---	---	---	---	---

PATRØN

7.5. Instrucción DIVIDE DECIMAL

a) Instrucción: DP D1(L1, B1), D2(L2, B2)

b) Formato : SS

DP	L1	L2	B1	D1	B2	D2
----	----	----	----	----	----	----

c) Función : El dividendo (primer operando) es dividido por el divisor (segundo operando) y reemplazado por el cociente y el residuo. El campo del cociente va a la izquierda y el campo del residuo va a la derecha, en el área del primer operando. La longitud del residuo es igual a la del divisor y su signo es el mismo del dividendo.

La longitud máxima del divisor puede ser de 8 bytes (7 en la instrucción de máquina) y corresponde a 15 dígitos y signo, además, debe ser inferior a la del dividendo, en caso contrario se produce una interrupción de programa.

Los campos de dividendo y divisor pueden superponerse únicamente si sus bytes de orden inferior coinciden.

Para evitar que se produzca desborde, como consecuencia de un cociente mayor que el campo que lo puede contener, que significaría interrupción de programa, es conveniente generar el campo del dividendo con una longitud igual a la suma de los bytes que ocuparía el dividendo definido en forma independiente, más L2.

Ejemplo 67.

Se tienen las variables:

ALFA que ocupa 5 bytes

BETA que ocupa 3 bytes

Calcular:

$$GAMA = \frac{ALFA}{BETA}$$

- i) Suponiendo que ALFA y BETA son enteras y se desea GAMMA sin decimales.
- ii) Suponiendo que ALFA y BETA son enteras y se desea GAMMA redondeado a dos decimales.
- iii) Suponiendo que ALFA tiene dos decimales, BETA tiene un decimal y se desea GAMMA redondeado a dos decimales.
- iv) Suponiendo que ALFA tiene 5 decimales, BETA un decimal y se desea GAMMA redondeado a un decimal.
- v) Suponiendo que ALFA tiene un decimal, BETA dos decimales y se desea GAMMA redondeado a dos decimales.

Solución de i):

```
START 0
BALR 15,0
USING *,15
--
--
ZAP GAMMA,ALFA
```

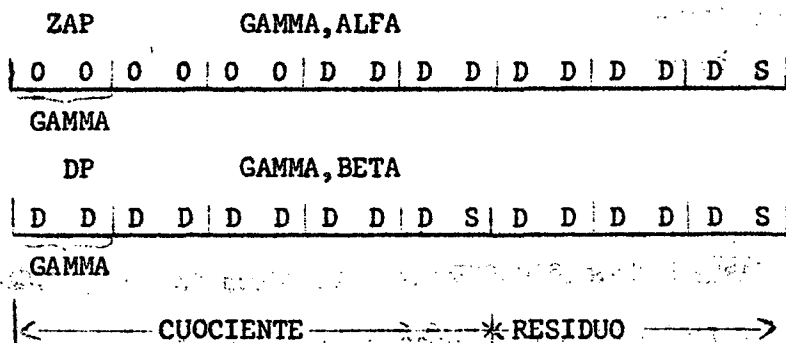
(Continúa)

(Continuación)

```

DP  GAMMA,BETA
ED  PATRØN,GAMMA
--
--
EØJ
GAMMA DS PL8
ALFA  DS PL5
BETA  DS PL3
PATRØN DC X'40202020202020202020214B4022'
END
    
```

Efecto de las instrucciones ZAP y DP:



Para solucionar los casos ii), iii), iv) y v) es conveniente hacer un análisis previo para determinar un procedimiento general. Deben considerarse los tres siguientes elementos:

DR = Dígitos de la parte fraccionaria que es necesario obtener en el resultado, para efectuar el redondeo adecuado.

DN = Dígitos de la parte fraccionaria del numerador (dividendo).

DD = Dígitos de la parte fraccionaria del denominador (divisor).

Se presentan tres casos posibles:

Primer caso: Numerador y denominador tienen igual número de dígitos en la parte fraccionaria (o no tiene ninguno de los dos) y se desea un resultado con parte fraccionaria.

La solución es amplificar el numerador por 10^{DR} .

El almacenamiento para el resultado debe contemplar: total de dígitos del numerador (TDN), DR y bytes que ocupa el denominador (L2).

$L1 = \frac{TDN+DR+SIGNO}{2} + L2$ si el resultado de la fracción no es entero exacto, se aproxima al entero superior inmediato.

Segundo caso: El numerador tiene mayor cantidad de dígitos en la parte fraccionaria que el denominador y se desea un resultado con parte fraccionaria.

Primero será necesario amplificar la fracción completa por 10^{DN} para eliminar las partes fraccionarias y luego amplificar el numerador por 10^{DR} . Pero se sabe que el punto decimal no está representado en el dato, luego, no es necesario amplificar el numerador por 10^{DN} , al mismo tiempo y por la misma causa, el denominador bastará con amplificarlo por 10^{DN-DD} .

Se tendrá así que el factor de amplificación está dado por:

$$\frac{10^{DR}}{10^{DN-DD}}$$

el cual se puede simplificar aún más, cuando se conozcan los datos DR, DN y DD.

Tercer caso: El numerador tiene menor cantidad de dígitos en la parte fraccionaria que el denominador, y se desea un resultado con parte fraccionaria.

Igual que en el segundo caso, primero será necesario amplificar la fracción completa, pero ahora por 10^{DD} , y luego amplificar el numerador por 10^{DR} . Sin embargo, el denominador no es necesario amplificarlo por 10^{DD} y el numerador se amplifica sólo por 10^{DD-DN} y luego por 10^{DR} .

Se tendrá así que el factor de amplificación está dado por:

$$10^{DD-DN} * 10^{DR} \quad \text{o lo que es lo mismo} \quad \frac{10^{DR}}{10^{DN-DD}}$$

y se determina así que esta fórmula es aplicable en todos los casos, incluso en el primero en que la diferencia $DN-DD=0$, y se tiene por lo tanto:

$$\frac{10^{DR}}{10^{DN-DD}} = 10^{DR}$$

Solución de ii):

```

START 0
BALR 15,0
USING *,15
--
ZAP GAMMA,ALFA
MP GAMMA,MIL
DP GAMMA,BETA
MVN AJUSTE(1),GAMMA+6
AP GAMMA(7),AJUSTE
MVØ GAMMA(7),GAMMA(6)
ED PATRØN,GAMMA+1
--
--
EØJ

```

```

AJUSTE DC X' 50'
MIL DC P' 1000'
PATRØN DC X' 402020202020202020214B20204022'
ALFA DS PL5
BETA DS PL3
GAMMA DS PL10
END

```

Solución de iii):

DN = 2 DD = 1 DR = 3

$$\frac{10^3}{10^{2-1}} = 10^2$$

La solución es similar a la de ii), cambia solamente el factor de multiplicación que es CIEN en vez de MIL y las modificaciones que de esto se deriven para las instrucciones siguientes.

Solución de iv):

DN = 5 DD = 1 DR = 2

$$\frac{10^2}{10^{5-1}} = 10^{-2}$$

El signo menos del exponente indica que hay que amplificar el denominador por 10^2 .

```

START 0
BALR 15,0
USING *,15
--
--
ZAP GAMMA,ALFA
MP BETA,CIEN
DP GAMMA,BETA
MVN AJUSTE(1),GAMMA+3
AP GAMMA,AJUSTE
MVØ GAMMA(4),GAMMA(3)
ED PATRØN,GAMMA
--
--
EØJ

AJUSTE DC X'50'
CIEN DC P'100'
PATRØN DC X'402020202020214B204022'
ALFA DS PL5
BETA DS PL4
GAMMA DS PL8

END

```

El campo BETA se define de tal manera que acepte el producto de BETA*100. Para definir el campo GAMMA se hace el siguiente análisis: Si BETA tiene en su campo la cantidad máxima, esto es, 99999, al multiplicarse por 100 quedará 9999900, en total siete dígitos. Luego el cuociente debe tener $9-7=2$ dígitos (dos bytes). Si por el contrario tiene la cantidad mínima, esto es, 1, al multiplicarse por 100 quedará 100, en total tres dígitos que se reducen a dos pues solo se contabilizan los ceros. Luego el cuociente debe tener $9-2=7$ dígitos (cuatro bytes). Para GAMMA, entonces, será necesario ocho bytes, cuatro para el cuociente y cuatro para el divisor.

Solución de v):

$$DN = 1$$

$$DD = 2$$

$$DR = 3$$

$$\frac{10^3}{10^{-2}} = 10^4$$

Solución similar a la de ii):

7.6. Instrucción COMPARE DECIMAL

- a) Instrucción: CP D1(L1,B1),D2(L2,B2)
- b) Formato : SS

CP	L1	L2	B1	D1	B2	D2
----	----	----	----	----	----	----
- c) Función : Se compara el primer operando con el segundo operando y se genera código de condición de acuerdo al resultado.

Si los campos son de longitud desigual, el más corto se extiende mediante ceros de orden superior hasta igualar la longitud del más largo. Un cero positivo da comparación igual con un cero negativo.

Es posible comparar un número consigo mismo.

Código de Condición	Resultado
0	OP1 = OP2
1	OP1 < OP2
2	OP1 > OP2

OP1 = primer operando

OP2 = segundo operando

7.7. Problemas propuestos

- a) ¿Cuál es el contenido del RUG1 y de AREA después del proceso siguiente?

```

START 0
BALR 14,0
USING *,14
B FIRST
EQUIS DC X'AB'
AREA DS 5F
FIRST L 1,=F'160'
STC 1,AREA
MVC AREA+1(19),AREA
IC 1,EQUIS
EØJ
END
    
```

b) ¿Cuál es el contenido del RUG1 al término del proceso siguiente?

```

START 300
BALR 15,0
USING *,15
L 1,=F'300'
MVI *+4,X'5B'
A 1,=F'-300'
EØJ
END

```

Observación: 5B es el código de operación correspondiente a SUBTRACT.

c) Indicar cuál es el contenido de los RUGs 2 y 3 al finalizar el proceso siguiente:

```

ICTL 1
START 256
BALR 8,0
USING *,8
EJECT
B BEGIN
JØSE DC 2F'+15'
BEGIN LM 2,3,JØSE
SRDL 2,64
EØJ
END

```

d) ¿Qué queda en los campos MAK2 y JUAN después de ejecutar el programa siguiente?

```

START 256
BALR 7,0
USING *,7
B GØ
MAK1 DC X'F6'
MAK2 DC X'06'
JUAN DC 2X'77'
GØ NC *-1(1),MAK2
NC JUAN,MAK1
EØJ
END

```

e) ¿Qué queda en el área PEPE después del proceso siguiente?

```
START 400
BALR 8,0
USING *,8
B BEGIN
PEPE DC PL4'567'
BEGIN MVØ PEPE(3),PEPE
NI PEPE+2,X'FO'
MI PEPE+3,X'OF'
EØJ
END
```

f) ¿Cuál es el contenido de las áreas M1,M2 y PEPE al término del proceso siguiente?

```
START 0
BALR 7,0
USING *,7
B GØ
M1 DC X'F6'
M2 DC X'16'
PEPE DC 2X'30'
GØ NC PEPE+1(1),M2
XC M1(2),PEPE
EØJ
END
```

g) ¿Cuál es el contenido del área BEGIN después del proceso siguiente?

```
START 0
BALR 15,0
USING *,15
B SIGUE
BEGIN DC 3F'2'
TABLA DC 3X'OF'
SIGUE TR BEGIN(3),TABLA
EØJ
END
```


h) Indicar qué queda en DIGIT después del siguiente proceso:

```
START 800
BALR 14,0
USING *,14
B BEGIN
ZØNE DC ZL8'-18954312'
DIGIT DS PL6
BEGIN PACK DIGIT(6),ZØNE(8)
MVC DIGIT(1),DIGIT+5
MVC DIGIT+1(5),DIGIT
EØJ
END
```

i) ¿Qué queda en el campo A al finalizar el siguiente proceso?

```
START X'2800'
BALR 2,0
USING *,2
B GØ
A DC PL6'4378000'
B DC P'26'
C DC X'50'
DP A,B
MVN C(1),A+3
AP A(4),C
MVØ A(4),A(3)
EØJ
END
```

j) Indicar qué queda en el área Z después del proceso siguiente:

```
START 400
BALR 15,0
USING *,15
B GØ
A DC P'1545631'
B DC P'27'
```

(Continúa)

(Continuación)

```

C   DC   P'42'
Z   DS   PL8
    ZAP  Z,A
    DP   Z,B
    DP   Z(6),C
    SVC  14
    END

```

k) Programar la siguiente expresión:

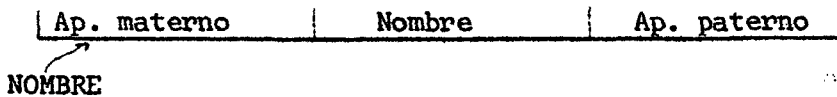
$$RES = \frac{DATO1 * DATO2}{DATO3}$$

donde:

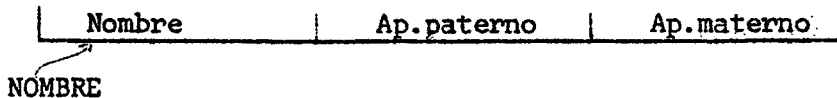
DATO1 tiene dos enteros y dos decimales DD,DDS
 DATO2 tiene un entero y un decimal D,DS
 DATO3 tiene dos enteros y dos decimales DD,DDS

Se desea el resultado redondeado al centésimo.

1) Se tiene en el área NOMBRE, el nombre y apellidos de un empleado, ordenados en la siguiente forma:



y se desea:

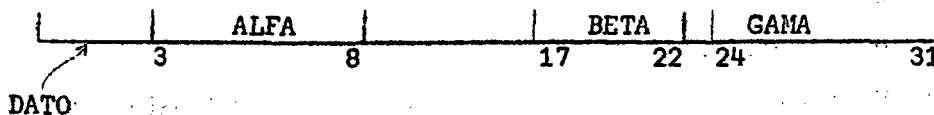


Nombre tiene doce caracteres.

Apellido paterno tiene diez y siete caracteres.

Apellido materno tiene diez y siete caracteres.

m) En un campo de 32 bits de nombre DATO, se tiene agrupada la siguiente información binaria:



Se pide dejar la información ALFA, BETA y GAMA empaquetada, ocupando una doble palabra cada una.

n) Una cantidad variable ha sido cargada en el RUG3. Esa cantidad indicará el desplazamiento de la información almacenada en la doble palabra DOBLEW. Si el signo del RUG3 es positivo, el desplazamiento es a la derecha y si es negativo, a la izquierda. El desplazamiento debe ser el mismo a la derecha o a la izquierda.

ñ) Se tiene la siguiente información binaria:

S	ALFA	S	BETA
0 1	9	11	31

Se pide multiplicar ALFA*BETA y dejar el resultado en un campo RES en binario.

o) El mismo problema ñ) resolverlo con aritmética decimal.

p) Se tiene en un byte de nombre MIXTO la siguiente información:

0 1 0 1 0 1 1 0
MIXTO

Se pide dejar los ceros a la izquierda y los unos a la derecha.

q) Se tiene en un área de cuatro bytes información binaria. Se pide dejar los unos agrupados a la derecha del área.

r) Se tienen tres datos A, B y C en EBCDIC. Cada uno ocupa ocho bytes. Se pide programar el siguiente cálculo:

$$Z = A * B + C \quad \text{si } C \leq 5$$

$$Z = (A + B) * C \quad \text{si } 5 < C \leq 10$$

$$Z = A * B/C \quad \text{si } C > 10$$

El resultado debe quedar en formato ZONA-DIGITO.

8. Instrucciones de BIFURCACION

La secuencia normal de ejecución de las instrucciones de un programa consiste en ejecutar una instrucción y a continuación la instrucción que le sigue en el almacenamiento, para esto se cuenta con la palabra de estado del programa (Program Status Word-PSW) la cual contiene, entre otros campos, el que registra la dirección de la instrucción que se ejecuta. Esta dirección es incrementada en el número de bytes de la instrucción, obteniéndose así la dirección de la instrucción siguiente en secuencia, llamada dirección actualizada. La dirección actualizada reemplaza a la dirección anterior, lo que permite ejecutar la instrucción siguiente.

Máscara de

Sistema		Llave	AMWP	Código de Interrupción	
0		7 8	11 12	15 16	31
ILC	CC	Máscara Program.		Dirección de la Instrucción(IA)	
32 33	34 35 36		39 40		63

ILC = Código de longitud de la instrucción (Instruction Length Code)

CC = Código de condición (Condition Code)

IA = Dirección de la instrucción (Instruction Address)

En la bifurcación se opera fundamentalmente a base de los tres campos mencionados ILC, CC e IA. El primero da la longitud de la instrucción que se ejecuta, longitud que al ser sumada al valor del tercer campo permite obtener la dirección actualizada. El segundo campo indicará si la próxima instrucción se extrae de la dirección actualizada o de la dirección de bifurcación.

8.1 Instrucción BRANCH ON COUNT REGISTER

a) Instrucción: BCTR R1, R2

b) Formato : RR

BCTR	R1	R2
------	----	----

c) Función : Al contenido del RUG especificado en R1 se le resta uno, algebraicamente. Si el resultado obtenido es cero se ejecuta la siguiente instrucción en secuencia. Si el resultado es distinto de cero, la instrucción que se va a ejecutar se extrae de la dirección de bifurcación (se efectúa el salto a la dirección contenida en el RUG indicado en R2). Si el campo R2 es cero, se realiza la resta, pero, no se ejecuta salto.

8.2 Instrucción BRANCH ON COUNT

a) Instrucción: BCT R1, D2(X2, B2)

b) Formato : RX

BCT	R1	X2	B2	D2
-----	----	----	----	----

c) Función : Cumple igual función que BRANCH ON COUNT REGISTER. La dirección de bifurcación se calcula antes de ejecutar la resta.

Ejemplo 68.

Se tiene un registro de cien caracteres. Se pide contabilizar los caracteres blanco.

```

CUENTABL  START 0
          BALR 2,0
          USING *,2
          SR 5,5
          L 6,DIRTABLA
          LA 6,99(6)
TEST      CLI 0(6),C'Ø'
          BE IGUAL
          BCT 6,TEST
          B STØRE
IGUAL     LA 5,1(5)
          BCT 6,TEST
STØRE    ST 5,CUENTA
          EØJ
CUENTA   DS F
TABLA    DS 100CL1
BLANCØ   DC C'Ø'
DIRTABLA DC A(TABLA)
          END

```

8.3. Instrucción BRANCH ON INDEX HIGH

- a) Instrucción: BXH R1,R3,D2(B2)
- b) Formato : RS | BXH | R1 | R3 | B2 | D2 |
- c) Función : Se suma un incremento a un valor inicial y la suma resultante se compara algebraicamente con un valor final o término de comparación. Si el resultado de la suma es mayor (HIGH) que el término de comparación, se efectúa el salto a la dirección dada por D2(B2), en caso contrario, se ejecuta la siguiente instrucción en secuencia.

El valor inicial debe estar cargado en el RUG especificado en R1, el incremento en el RUG dado en R3 que debe ser PAR y el valor de comparación en el RUG IMPAR inmediatamente posterior al indicado en R3.

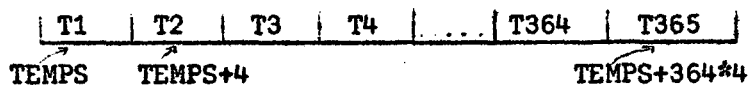
Si R3 es un RUG IMPAR se considera el mismo incremento, como valor de comparación. Si R1 es igual al RUG que contiene el valor de comparación se considera como valor final el contenido original del registro, esto es, antes de que sea incrementado.

La dirección de bifurcación se calcula antes de las operaciones de suma y comparación.

Ejemplo 69.

Se tiene una tabla en la que están registradas las temperaturas máximas ocurridas en cada día de un año. Se pide obtener el promedio anual de temperaturas.

La información está en formato "empaquetado" ocupando cuatro bytes cada temperatura y con un dígito en la parte fraccionaria.



```

CØNTEMP START 0
        BALR 6,0
        USING *,6
        LA 7,TEMPS VALØR INICIAL
        LA 8,4 INCREMENTØ
        LA 9,TEMPS+364*4 VALOR FINAL
SUMA    AP CØNTA,0(4,7)
        BXH 7,8,ØUT
        B SUMA
ØUT     DP CØNTA,C365
        ED PATRØ,CØNTA
        EØJ
CØNTA   DC PL4'0'
PATRØ   DC X'4020214B204022'
C365    DC P'365'
TEMPS   DS 365PL4
        END
    
```

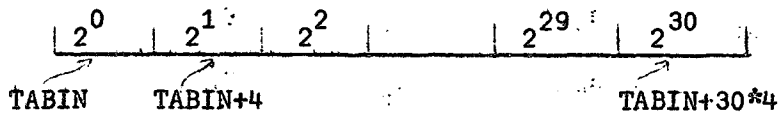
Se carga en el RUG7 la dirección de comienzo de la tabla que será el valor inicial. En el RUG8 se carga el valor 4 como incremento y en el RUG9 se carga la dirección del último elemento de la tabla que será el valor final o término de comparación.

Se suman en CONTA las temperaturas, utilizando el RUG7 que tiene la dirección de cada elemento.

En la instrucción BRANCH ON INDEX HIGH se suma al <RUG7> el <RUG8> y el resultado se compara con el <RUG9>. Si el resultado de la comparación es mayor se efectúa un salto a OUT, en caso contrario, continúa la secuencia normal, pero esta secuencia normal significa, en este caso, volver a la dirección SUMA.

Ejemplo 70.

Generar una tabla de potencias de 2, en binario, desde 2^0 hasta 2^{30} .



```

START 0
BALR 15,0
USING *,15
L 0,UNØ
SR 1,1
LA 2,4
LA 3,31*4
LA 4,TABIN-4
LØØP BXH 1,2,FIN
ST 0,0(1,4)
SLA 0,1
B LØØP
FIN EØJ
UNØ DC F'1'
TABIN DS 31F
END
    
```

8.4. Instrucción BRANCH ON INDEX LOW OR EQUAL

a) Instrucción: BXLE R1,R3,D2(B2)

b) Formato : RS BXLE | R1 | R3 | B2 | D2

c) Función : La función es similar a la de la instrucción BRANCH ON INDEX HIGH. En esta instrucción la bifurcación se ejecuta cuando la suma del contenido del RUG especificado en R1 y el incremento, resulta menor o igual que el valor final.

Ejemplo 71.

Se tienen cinco tablas diferentes, cada tabla tiene cien elementos y cada elemento tiene nueve dígitos y signo, en formato empaquetado. Se pide calcular la suma de cada quintupla de elementos.

TAB1	TAB2	TAB3	TAB4	TAB5	TSUM
a ₁₁	a ₂₁	a ₃₁	a ₄₁	a ₅₁	a ₁₁
a ₁₂	a ₂₂	a ₃₂	a ₄₂	a ₅₂	a ₁₂
.
.
.
a ₁₁₀₀	a ₂₁₀₀	a ₃₁₀₀	a ₄₁₀₀	a ₅₁₀₀	a ₁₁₀₀

```

START 0
BALR 15,0
USING *,15
LM 6,11,REG
LOOP ZAP 0(6,6),ZERØ
AP 0(6,6),0(5,9)
AP 0(6,6),500(5,9)
AP 0(6,6),1000(5,9)
AP 0(6,6),1500(5,9)
AP 0(6,6),2000(5,9)
BXLE 6,7,*+4
BXLE 9,10,LOOP
EØJ

```

(Continúa)

(Continuación)

REG	DC	A(TSUM)
	DC	F' 6'
	DC	F' 0'
	DC	A(TAB1)
	DC	F' 5'
	DC	A(TAB1+99*5)
ZERØ	DC	P' 0'
TAB1	DS	500CL5
TSUM	DS	100CL6
	END	

La instrucción BXLE 6,7,*+4 se utiliza con el objeto de incrementar el contenido del RUG6 que ha sido cargado anteriormente con la dirección TSUM. Cualquiera que sea el resultado de la comparación siempre se ejecutará la instrucción siguiente, dado que la dirección de bifurcación corresponde a la instrucción que sigue en secuencia.

La instrucción BXLE 9,10,LOOP incrementa el contenido del RUG9 con el contenido del RUG10 y el resultado lo compara con el contenido del RUG11. Mientras dicho resultado sea menor o igual, la ejecución se reiniciará en la dirección LOOP.

8.5. Instrucción EXECUTE

- a) Instrucción: EX R1,D2(X2,B2)
- b) Formato : RX

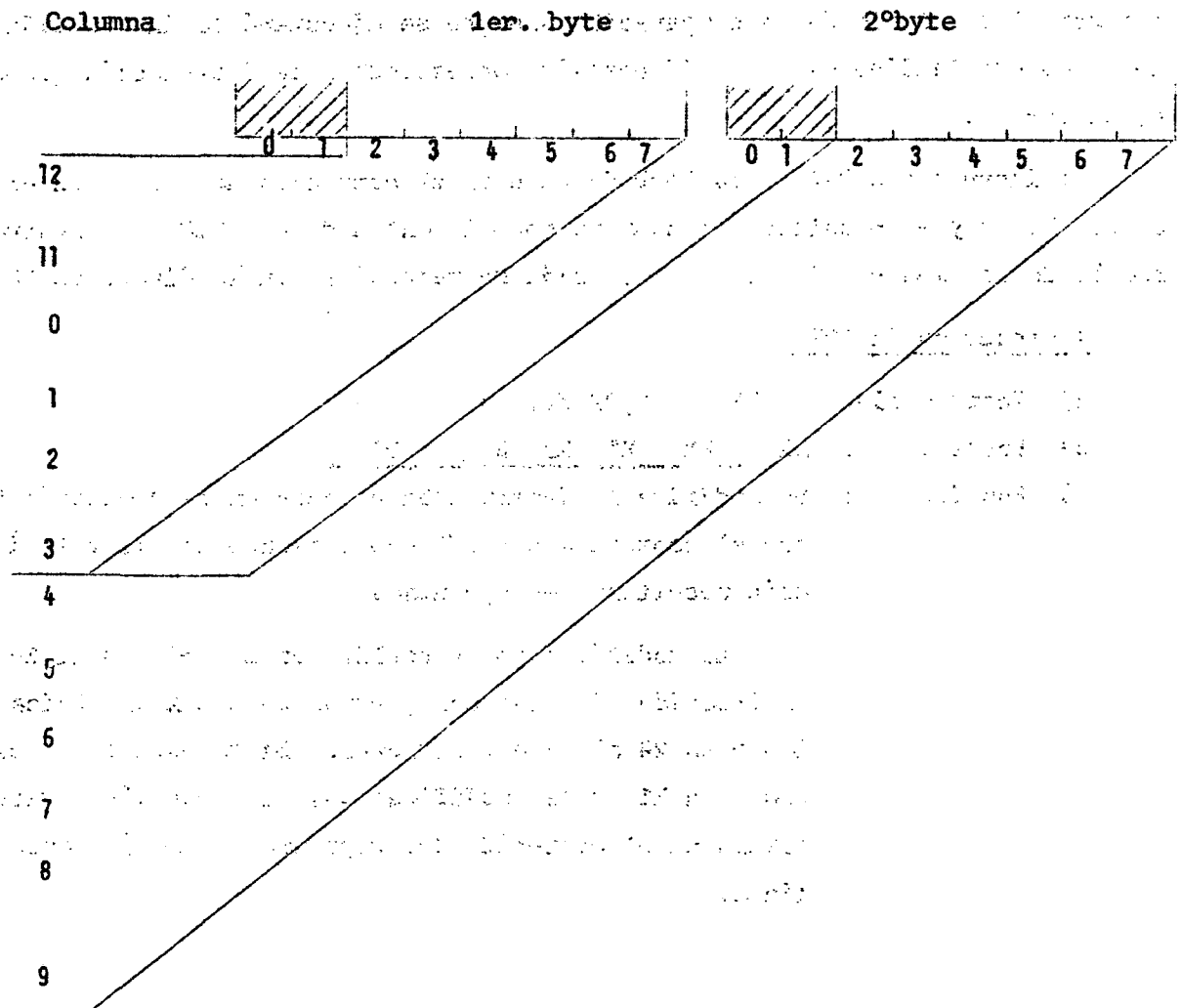
EX	R1	X2	B2	D2
----	----	----	----	----
- c) Función : Se modifica la instrucción ubicada en la dirección D2(X2,B2) por el contenido del RUG especificado en R1, y la instrucción resultante es ejecutada.

La modificación se realiza en los bits 8 al 15 de la instrucción, los que son puestos en conexión lógica OR con los bits 24 al 31 del registro. Si se especifica el registro 0 en R1 no hay modificación. La conexión lógica OR no altera ni el contenido del registro, ni la instrucción original.

Una vez que la instrucción resultante ha sido ejecutada se retorna a la instrucción siguiente a EXECUTE, salvo que la instrucción sujeta a modificación sea de bifurcación, en cuyo caso la dirección de salto reemplaza en la PSW a la dirección actualizada. Si la instrucción que se modifica es otra instrucción EXECUTE se produce una excepción de ejecución y como consecuencia una interrupción de programa.

Ejemplo 72.

Existe la posibilidad de leer tarjetas multiperforadas, esto es, que pueden tener hasta las doce posiciones perforadas, dentro de cada columna. La información se almacena en dos bytes seguidos, por cada columna leída, como se indica a continuación.



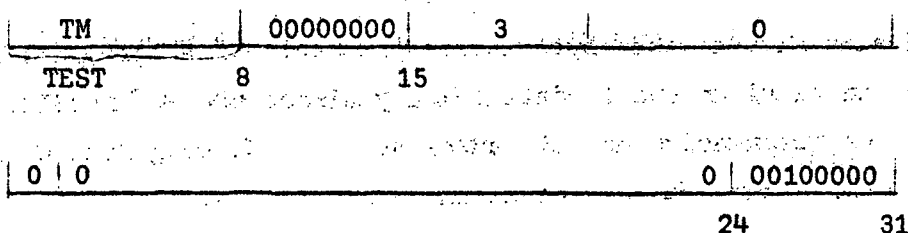
Los dos primeros bits de cada byte no se utilizan. En el problema que figura a continuación se ha simulado una tarjeta leída y almacenada en TARJBIN. Se pide analizar los bits de información de tal manera que si es 1 se guarda un carácter uno en MATRIZ, en caso contrario se guarda un carácter cero.

```

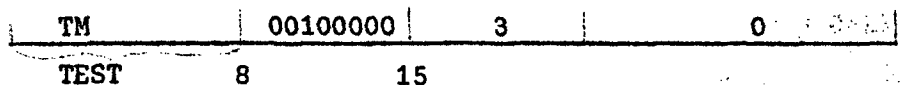
PRØG2  START 0
        BALR  2,0
        USING *,2
        LM    3,6,REG
BEGIN  LM    7,9,ØTRØCER
        LA    10,32
CØMPA  EX    10,TEST
        BØ    UNØS
        MVC  0(1,6),CERØCAR
        B    MØDIF
UNØS   MVC  0(1,6),UNØCAR
MØDIF  A    6,UNØBIN
        SRL  10,1
        BXLE 7,8,CØMPA
        BXLE 3,4,BEGIN
        EØJ
REG    DC    A(TARJBIN)
UNØBIN DC    F'1'
        DC    A(TARJBIN+159)
        DC    A(MATRIZ)
ØTRØCER DC    F'0,1,5'
UNØCAR  DC    C'1'
CERØCAR DC    C'0'
TARJEIN DC    20X'0123456789ABCDEF'
TEST   TM    0(3),X'00'
MATRIZ DS    12CL80
        END

```

La instrucción EX 10,TEST pone en conexión lógica OR los bits 8 a 15 de la instrucción que hay en TEST con los bits 24 a 31 del RUG 10.



La instrucción resultante será:



la que permitirá analizar el bit 2 del byte cuya dirección está dada por el contenido del RUG3, esto es, TARJBIN. Después de efectuado el análisis y hechas las modificaciones necesarias para almacenar el siguiente carácter cero o uno, se desplaza el contenido del RUG10 un lugar a la derecha, de tal manera que, al empezar de nuevo el ciclo se pueda investigar el bit 3 del byte que está en la dirección que contiene el RUG3, y así sucesivamente.

9. Subrutinas y Subprogramas

Se define como subrutina al segmento de programa al cual se hace referencia en distintos puntos de éste y se compagina (ensambla) junto con él.

Se define como subprograma al programa independiente que es llamado por un programa principal y que se compagina aparte de él.

En el primer caso surgen las preguntas siguientes: ¿Cómo sabe la subrutina donde debe volver? y ¿de dónde toma los datos y dónde deja los resultados? El problema que plantea la primera pregunta se resuelve con la instrucción BRANCH AND LINK REGISTER, que se ha utilizado anteriormente con el segundo operando igual a cero, antes de la pseudo-instrucción USING. La instrucción, carga en el RUG especificado en R1 la dirección de la próxima instrucción y salta a la dirección contenida en el RUG indicado en R2. Esta última dirección será la de la subrutina la cual tendrá como instrucción de retorno un salto incondicional a la dirección contenida en el RUG especificado en R2, que debe ser el mismo que figura en R1 en la instrucción BALR.

Ejemplo 73.

<u>Rutina principal</u>	<u>Subrutina</u>
---	USING *,14
---	---
LA, 14,SUBRUT	--->SUBRUT
---	---
---	---
BALR 15,14	---
---	---
---<	BR 15
---	---

Los problemas que plantea la segunda pregunta los resuelve el propio programador estableciendo el o los registros donde entregará los resultados y en los que recibirá resultados. Es posible que en vez de registros, utilice un área de memoria con acceso por parte de la rutina principal y de la subrutina.

Ejemplo 74.

```

BEGIN   START 0
        BALR  2,0
        USING *,2
        L     3,UNØ      <RUG 3> = 1
        L     14,DIRSR   <RUG 14>= SR1
        BALR  13,14     <RUG 13>= ST1
ST1     ST     3,RESP1   <RESP1> = <RUG 3>
        L     3,CUATRØ   <RUG 3> = 4
        L     14,DIRSR   <RUG 14>= SR1
        BALR  13,14     <RUG 13>= ST2
ST2     ST     3,RESP2   <RESP2> = <RUG 3>
        EØJ
DIRSR   DC     A(SR1)
UNØ     DC     F'1'
CUATRØ  DC     F'4'
RESP1   DS     F
RESP2   DS     F
* SUBROUTINA
        USING *,14
SR1     SLA    3,1
        BR     13
        END   BEGIN

```

La subrutina desplaza el contenido del RUG3 un lugar a la izquierda y retorna a la rutina principal.

Ejemplo 75.

```

PRØGA  START 0
        BALR  2,0
        USING *,2
        L     14,DIREC
        CNØP  2,4
        BALR  13,14

CØNST  DC   A(LISTA)
        DC   F'4'
        DC   A(PRØM1)
        L    14,A
        A    14,A+4
        ST   14,C
        L    14,DIREC
        CNØP  2,4
        BALR  13,14
        DC   A(LISTB)
        DC   F'6'
        DC   A(PRØM2)
        EØJ

A       DC   F'95,81'
C       DS   F

LISTA  DC   F'31,42,27,18'
LISTB  DC   F'11,24,-5,-91,57,77'
PRØM1  DS   F
PRØM2  DS   F
DIREC  DC   A(PRØM)

* SUBROUTINA PRØMEDIØ
        USING *,14
PRØM   STM  2,7,SALVA
        L    5,0(13)
        LA   6,4

```

(Continúa)

(Continuación)

	L	4,4(13)
	LR	7,4
	S	7,UNØ
	SLA	7,2
	AR	7,5
	SR	2,2
	SR	3,3
SUMA	A	3,0(5)
	BXLE	5,6,SUMA
	DR	2,4
	L	5,8(13)
	ST	3,0(5)
	LM	2,7,SALVA
	B	12(13)
UNØ	DC	F'1'
SALVA	DS	6F
	END	BEGIN

En el caso de subprogramas, el problema que es necesario resolver es el "enganche" entre la rutina principal y el subprograma, dado que ambos constituyen módulos independientes que pueden ser compaginados en momentos distintos y que en el momento de ser cargados en memoria para su ejecución no tienen porque serlo en las mismas ubicaciones que ocuparon inicialmente. En otras palabras, cuando se compaginó la rutina principal, la referencia al subprograma, que corresponde a la dirección donde debe estar cargado el mismo, quedó sin resolver pues no era conocida dicha dirección. Solo será resuelto ese problema cuando se carguen ambos módulos para ser ejecutados. El sistema debe saber cuál o cuales símbolos son externos, o lo que es lo mismo, debe conocer cuales símbolos están definidos en otro módulo y eso se consigue con la pseudo-instrucción EXTERNAL que tiene como operandos los símbolos externos. Además, el sistema debe saber dónde buscar la definición de dichos símbolos y eso se logra con la pseudo-instrucción ENTRY que tiene como operandos los símbolos utilizados en otros módulos y que están definidos en el módulo en que ella se encuentra.

9.1. Pseudo-Instrucción EXTERNAL

- a) Código : EXTRN
- b) Formato : blanco EXTRN operando
- c) Función : Declara aquellos símbolos que son usados en el módulo pero que están definidos fuera de él. Si hay más de un símbolo se separa del resto por coma. Los símbolos declarados no se pueden utilizar como identificadores de proposición dentro del mismo módulo.

9.2. Pseudo-Instrucción ENTRY

- a) Código : ENTRY
- b) Formato : blanco ENTRY operando
- c) Función : Declara aquellos símbolos que son definidos en el módulo y que son utilizados por otros módulos. Si hay más de un símbolo se separa del resto por coma. Los símbolos utilizados aparecen como identificadores de proposición dentro del mismo módulo. Si el símbolo es el nombre de la sección de control no necesita ser declarado con la pseudo-instrucción ENTRY.

Otra forma de declarar los símbolos externos es definiéndolos en la rutina llamadora, con constantes de dirección tipo V y cargando las constantes en registros de uso general antes de hacer un BRANCH AND LINK REGISTER que produciría el salto al subprograma respectivo.

Ejemplo 76.

El mismo problema del ejemplo 75, resuelto ahora, con un subprograma.

```

PRØGA  START 0
        EXTRN PRØM
        BALR 2,0
        USING *,2
        L   14,DIREC
        CNØP 2,4
        BALR 13,14

```

(Continúa)

(Continuación)

```

CØNST  DC  A(LISTA)
        DC  F'4'
        DC  A(PRØM1)
        L   14,A
        A   14,A+4
        ST  14,C
        L   14,DIREC
        CNØP 2,4
        BALR 13,14
        DC  A(LISTB)
        DC  F'6'
        DC  A(PRØM2)
        EØJ
A       DC  F'95,81'
C       DS  F
LISTA  DC  F'31,42,27,18'
LISTB  DC  F'11,24,-5,-91,57,77'
PRØM1  DS  F
PRØM2  DS  F
DIREC  DC  A(PRØM)
        END  PRØGA

PRØM   START 0
        ENTRY PRØM
        USING *,14
        STM  2,7,SALVA
        L   5,0(13)
        LA  6,4
        L   4,4(13)
        LR  7,4
        S   7,UNØ
        SLA 7,2
        AR  7,5
        SR  2,2
        SR  3,3

```

(Continúa)

(Continuación)

```

SUMA  A    3,0(5)
      BXLE 5,6,SUMA
      DR   2,4
      L   5,8(13)
      ST  3,0(5)
      LM  2,7,SALVA
      B   12(13)
UNØ   DC  F'1'
SALVA DS  6F
      END

```

Observaciones:

a) No es necesario colocar ENTRY PROM dado que PROM es punto de entrada por derecho propio, por ser el nombre de la sección de control.

b) Si se especifica DIREC DC V(PROM) no es necesario colocar EXTRN PROM.

10. Instrucciones nuevas de Assembler para el Sistema/370

10.1. Instrucción COMPARE AND SWAP

- a) Instrucción: CS R1,R3,D2(B2)
- b) Formato : RS

CS	R1	R3	B2	D2
----	----	----	----	----
- c) Función : Se comparan el primer y el segundo operandos. Si son iguales, el tercer operando es almacenado en la ubicación del segundo. Si son distintos, el segundo operando es cargado en la ubicación del primero.

Todos los operandos tienen una palabra de longitud. El primero y el tercero están en los RUGs especificados en R1 y R3 respectivamente y el segundo está en la dirección D2(B2) que debe cumplir con alineamiento de palabra.

Si el resultado de la comparación es distinto no se efectúa almacenamiento en memoria y por lo tanto no se toman acciones de protección de memoria y cambio de bit.

Cuando el resultado de la comparación es igual, no se permite el acceso de otra Unidad Central de Proceso (UCP) a la ubicación del segundo operando. El acceso es impedido prácticamente desde el momento en que el segundo operando es cargado para comparación hasta el momento en que el tercer operando es almacenado en la ubicación del segundo.

Una función de tipo serial es realizada antes de que el operando esté cargado y lo mismo ocurre si aparece el código de condición 0 después que el resultado es almacenado. La operación de la UCP es demorada hasta que todos los accesos previos de ella, a memoria principal han sido terminados, lo mismo es observado para canales y otras UCPs y después que eso ocurre, el segundo operando es cargado.

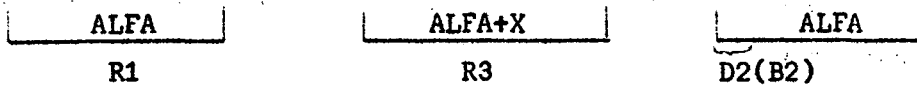
A ninguna instrucción posterior o a sus operandos tiene acceso la UCP, hasta que la ejecución de la instrucción COMPARE AND SWAP está terminada, incluida la colocación del valor de resultado, si lo hay, en la memoria principal, lo mismo es observado para canales y otras UCPs.

<u>Código de Condición</u>	<u>Resultado</u>
0	Primer y segundo operandos son iguales
1	Primer y segundo operandos son distintos

Observación: La instrucción COMPARE AND SWAP puede ser usada por programas que comparten áreas de almacenamiento común, ya sea en multiprogramación o en multiproceso. Por ejemplo, un programa puede modificar el contenido de una ubicación aún cuando exista la posibilidad de que otra UCP pueda actualizar simultáneamente la ubicación. En este caso, primero se carga la palabra que se va a actualizar, en un RUG. En seguida, el valor actualizado es computado y colocado en otro RUG. Después es ejecutada la instrucción COMPARE AND SWAP con el valor original en el RUG especificado en R1 y con el valor actualizado en el RUG indicado en R3. Si se produce el código de condición 0 la actualización se ha producido (los valores son iguales), en caso contrario, la ubicación de memoria ya no contiene el valor original, o sea, no se ha producido la actualización deseada, el RUG indicado en R1 tiene un nuevo valor obtenido por la intervención de otro programa u otra UCP. Luego se puede repetir el procedimiento con los mismos valores.

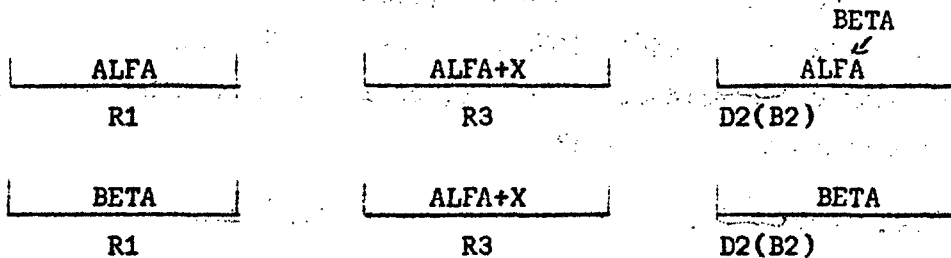
Gráficamente el problema se puede representar en la siguiente forma:

1° Se cargan los valores original y actualizado en los RUGs. Sea ALFA el valor original y ALFA+X el valor actualizado.



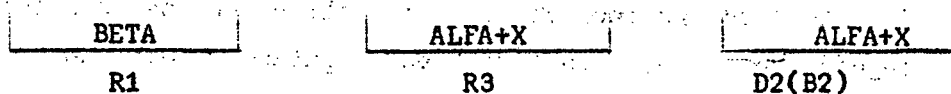
2° Se ejecuta la instrucción COMPARE AND SWAP.

Si el primer y el segundo operando son iguales queda en D2(B2) el valor ALFA+X. Si el contenido de D2(B2) fue modificado por otro programa u otra UCP, el resultado de la comparación será distinto y en el RUG indicado en R1 y en D2(B2) quedará el nuevo valor, por ejemplo BETA.



3° Si el código de condición producido es 1, lo que significa que primer y segundo operandos eran distintos, se repite la instrucción COMPARE AND SWAP con los mismos valores que se tienen en D2(B2) y en el RUG indicado en R1.

Suponiendo que no hay una nueva modificación del contenido de la dirección D2(B2) se tendrá finalmente:



10.2. Instrucción COMPARE DOUBLE AND SWAP

- a) Instrucción: CDS R1,R3,D2(B2)
- b) Formato : RS

CDS	R1	R3	B2	D2
-----	----	----	----	----
- c) Función : Cumple la misma función que la instrucción COMPARE AND SWAP.

Todos los operandos tienen una doble palabra de longitud. En los campos R1 y R3 se especifican RUGs pares, dado que el primer y el tercer operando ocupan dos RUGs, PAR e IMPAR siguiente, cada uno. El segundo operando está en la dirección D2(B2) que debe cumplir con alineamiento de doble palabra.

10.3. Instrucción COMPARE LOGICAL CHARACTERS UNDER MASK

- a) Instrucción: CLM R1,M3,D2(B2)
- b) Formato : RS

CLM	R1	M3	B2	D2
-----	----	----	----	----
- c) Función : Se compara el segundo operando con el primero en función de una máscara. Se genera código de condición de acuerdo al resultado.

Se utiliza el campo M3 como máscara, haciendo corresponder cada bit del campo con cada byte del RUG especificado en R1, partiendo de izquierda a derecha en ambos casos. Los bytes del RUG que corresponden a bits uno de la máscara se consideran contiguos y se comparan con igual número de bytes a partir de la dirección D2(B2). Los bytes que corresponden a bits cero no participan en la operación.

La comparación es realizada considerando los operandos como cantidades binarias sin signo. Ningún operando es cambiado.

<u>Código de Condición</u>	<u>Resultado</u>
0	Los bytes seleccionados son iguales o la máscara es cero.
1	El campo del primer operando es menor que el segundo operando.
2	El campo del primer operando es mayor que el segundo operando.

10.4. Instrucción COMPARE LOGICAL LONG

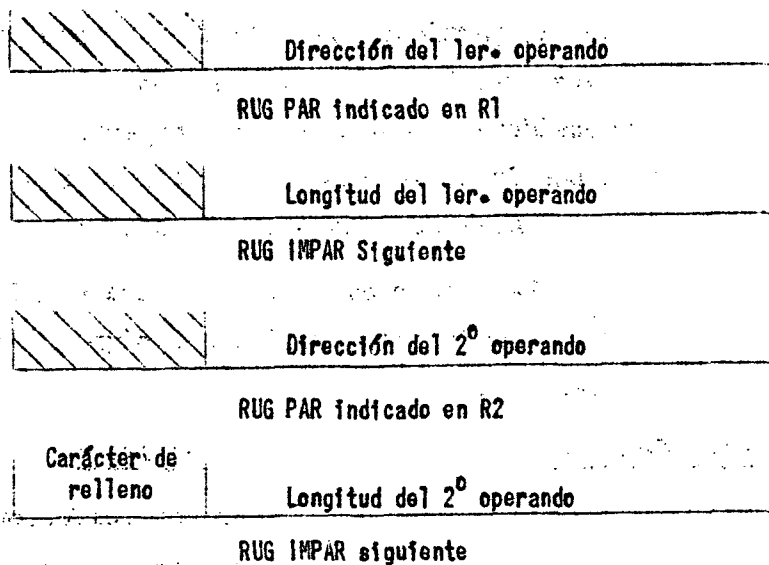
- a) Instrucción: CLCL R1,R2
- b) Formato : RR

CLCL	R1	R2
------	----	----
- c) Función : Se compara el primer operando con el segundo. Se genera código de condición de acuerdo al resultado.

En los campos R1 y R2 se especifican RUGs pares, dado que ambos operandos ocupan dos RUGs, PAR e IMPAR siguiente, cada uno. El primer byte o byte de orden superior de cada operando, es designado por el contenido de los bits 8-31 del

RUG IMPAR respectivo. Los bits 0-7 del RUG IMPAR que corresponde al segundo operando contiene un carácter de relleno que se ocupa para extender el operando más corto hasta completar la longitud del operando más largo. Los bits 0-7 de los RUGs pares y del impar que corresponde al primer operando, se ignoran.

El contenido de cada RUG se describe a continuación en forma gráfica:



La comparación se realiza de izquierda a derecha, byte por byte. La operación termina cuando se detecta una desigualdad o cuando se detecta el final de los campos. Ningún operando es cambiado. Si se especifica longitud cero para ambos operandos, se consideran iguales.

En el caso de encontrar bytes desiguales durante la comparación, el campo de longitud (contador) y el de dirección al término de la operación identifican al byte de la desigualdad, para ello, el contenido de los bits 8 a 31 de los RUGs impares es disminuido en el número de bytes en que hubo igualdad, a menos que la desigualdad haya ocurrido con el carácter de relleno, en cuyo caso, el campo de longitud para el operando más corto es puesto en cero. El contenido de los bits 8 a 31 de los RUGs pares es incrementado en el número de bytes en que hubo igualdad. Si los dos operando son iguales, incluido el carácter de relleno si es necesario, los dos campos de longitud son puestos en cero y las direcciones son incrementadas

en los valores de longitud correspondientes. El contenido de los bits 0 a 7 de los RUGs pares es puesto en cero y el de los RUGs impares permanece sin cambio.

El control que se tiene sobre la cantidad de bytes comparados permite que la instrucción sea interrumpida por un evento externo y reiniciada a partir del punto de interrupción. En este caso la dirección de la instrucción en la PSW aparece como si la instrucción no hubiera sido aún ejecutada.

Código de Condición

Resultado

0

Los operandos son iguales o ambos campos tienen longitud cero.

1

El primer operando es menor.

2

El primer operando es mayor.

10.5. Instrucción INSERT CHARACTERS UNDER MASK

a) Instrucción: ICM R1, M3, D2(B2)

b) Formato : RS

ICM	R1	M3	B2	D2
-----	----	----	----	----

c) Función : Se almacenan en el RUG especificado en R1, bytes tomados a partir de la dirección D2(B2).

La cantidad de bytes que se almacena corresponde a los unos que contiene la máscara M3.

Los bytes que se llenan del RUG son los que corresponden a los unos de la máscara, comenzando de izquierda a derecha. Los bytes que corresponden a ceros de la máscara permanecen sin cambio.

El código de condición resultante depende de la máscara y de los bits almacenados. Si la máscara es cero o si todos los bits almacenados son iguales a cero, el código es cero. Si no todos los bits son iguales a cero se considera el bit almacenado de orden superior (bit del extremo izquierdo del campo D2(B2)). Si el bit es uno, el código es uno, si el bit es cero, el código es dos.

<u>Código de Condición</u>	<u>Resultado</u>
0	Máscara cero o bits insertados son todos ceros.
1	Bit insertado de orden superior es uno.
2	Bit insertado de orden superior es cero.

10.6. Instrucción MOVE LONG

- a) Instrucción: MVCL R1,R2
- b) Formato : RR

MVCL	R1	R2
------	----	----
- c) Función : El segundo operando es movido a la ubicación del primer operando, siempre que no haya traslapo de direcciones de operando que afecten al contenido final del resultado. Si quedan bytes de orden inferior de la ubicación del primer operando que no han sido llenados, se transfieren a ellas caracteres de relleno.

En los campos R1 y R2 se especifican RUGs pares, dado que ambos operandos ocupan dos RUGs, PAR e IMPAR siguiente, cada uno. El primer byte o byte de orden superior de cada operando es designado por el contenido de los bits 8-31 del RUG PAR respectivo. La longitud de cada operando es especificada por el contenido de los bits 8-31 del RUG IMPAR respectivo. Los bits 0-7 del RUG IMPAR que corresponde al segundo operando contiene un carácter de relleno. Los bits 0-7 de los RUGs pares y del impar que corresponde al primer operando, se ignoran.

El movimiento parte en el extremo de orden superior de ambos campos y sigue hacia la derecha. No hay cambio ni inspección de los operandos. La operación es finalizada cuando el número de bytes especificados en el RUG impar del primer operando ha sido movido a la dirección del primer operando. A medida que se realiza la transferencia, el contenido de los RUGs impares va siendo disminuido. Si el RUG impar que corresponde al segundo operando llega a cero primero, se continúan transfiriendo caracteres de relleno.

Como parte de la ejecución de la instrucción, los contenidos de los RUGs impares (contadores) son comparados para establecer el código de condición, además se hace un chequeo de las direcciones de los operandos para determinar si hay traslapo destructivo. Se entiende por traslapo destructivo de los operandos cuando

la ubicación del primero de ellos es utilizada como fuente (segundo operando), después de que un dato ha sido movido a ella. Cuando hay traslapo destructivo, no se produce movimiento y el código de condición es puesto en 3.

Dependiendo de si el segundo operando abarca desde la posición 16.777.215 a la posición 0, el movimiento tiene lugar en los siguientes casos:

a) Cuando el segundo operando no abarca esas posiciones, el movimiento es realizado cuando el byte de orden superior del primer operando coincide con o está a la izquierda del byte de orden superior del segundo operando, o si el byte de orden superior del primer operando está a la derecha del byte de orden inferior del segundo operando que está participando en la operación.

b) Cuando el segundo operando abarca esas posiciones, el movimiento es realizado de acuerdo a las mismas condiciones expresadas en a) cambiando solamente la o por y.

El byte de orden inferior del segundo operando se determina por la menor de las longitudes (contadores) de los operandos.

Cuando la longitud especificada en los bits 8 a 31 del RUG IMPAR que corresponde al primer operando es cero, no se realiza movimiento, pero se genera código de condición para indicar los valores relativos de las longitudes.

El control que se tiene sobre la cantidad de bytes transferidos permite que la instrucción sea interrumpida por un evento externo y reiniciada a partir del punto de interrupción. En este caso, la dirección de la instrucción en la PSW aparece como si la instrucción no hubiera sido aún ejecutada.

Código de Condición

Resultado

0

Las longitudes de los operandos son iguales

1

La longitud del primer operando es menor

2

La longitud del primer operando es mayor

3

No hay transferencia a causa de traslapo destructivo.

10.7. Instrucción SHIFT AND ROUND DECIMAL

- a) Instrucción: SRP D1(L1, B1), D2(B2), I3
- b) Formato : SS | SRP | L1 | I3 | B1 | D1 | B2 | D2
- c) Función : El primer operando ubicado en la dirección D1(B1) es desplazado de acuerdo a los seis bits de orden inferior de la representación binaria de la dirección D2(B2). Cuando se especifica un desplazamiento a la derecha, el resultado es redondeado con el factor I3.

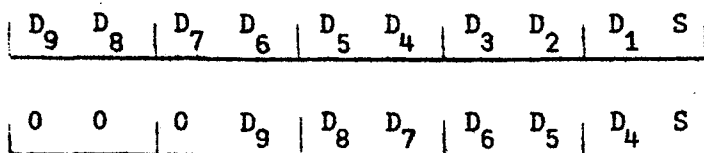
Los valores que figuran a continuación indican la interpretación de los seis bits que especifican el desplazamiento.

<u>Contenido de los seis bits</u>	<u>Interpretación</u>
011111	31 dígitos se desplazan a la izquierda
000001	1 dígito se desplaza a la izquierda
000000	No hay desplazamiento.
111111	1 dígito se desplaza a la derecha
100000	32 dígitos se desplazan a la derecha.

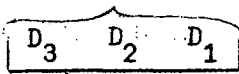
Se considera que el primer operando está en formato decimal empaquetado y se verifica la validez de los dígitos decimales y del código de signo, este último no participa en el desplazamiento. Se introducen ceros en las posiciones de dígitos que quedan desocupadas. Un resultado cero es considerado positivo.

Si un dígito significativo es desplazado fuera de la posición del dígito de orden superior durante un desplazamiento a la izquierda, se produce desborde (overflow) decimal. La operación se termina ignorando el desborde.

Durante el desplazamiento a la derecha, el contenido del campo I3 se utiliza como un factor de redondeo. Este factor se suma al último dígito que salió fuera del campo por efecto del desplazamiento y propagando el dígito de desborde si lo hay, hacia la izquierda. Tanto el primer operando como el factor de redondeo son considerados positivos sólo para efectuar la suma.



Dígitos Desplazados



ult. Dígito Desp.

La validez del primer operando es verificada y se establece código de condición aún cuando se especifique desplazamiento cero.

<u>Código de Condición</u>	<u>Resultado</u>
0	Cero
1	Menor que cero
2	Mayor que cero
3	Desborde (overflow)

10.8. Instrucción STORE CHARACTERS UNDER MASK

- a) Instrucción: STCM R1, M3, D2(B2)
- b) Formato : RS | STCM | R1 | M3 | B2 | D2
- c) Función : Se seleccionan bytes del primer operando de acuerdo a una máscara y se almacenan en la dirección dada por el segundo operando.

Se utiliza el campo M3 como máscara, haciendo corresponder cada bit del campo, con cada byte del RUG especificado en R1, partiendo de izquierda a derecha en ambos casos. Los bytes del RUG que corresponden a bits uno de la máscara se almacenan uno a continuación del otro conservando el orden original, a partir de la dirección D2(B2).

El número de bytes almacenados es igual al número de unos en la máscara. El contenido del RUG no se altera. No se genera código de condición.

11. Entrada/salida de información (Input/Output)

Las operaciones de entrada/salida se ejecutan a través de dispositivos llamados canales los cuales simplemente conectan unidades de entrada/salida a la unidad de procesamiento.

Puede considerarse el canal como un pequeño computador independiente para manejar operaciones de entrada/salida. Tiene un conjunto limitado de instrucciones llamados comandos. Un conjunto de comandos forma un "programa de canal". Tiene además sus propios registros internos para operaciones y por lo tanto no requiere del uso de los registros de la UCP, aún cuando comparte memoria con ella.

Pocas veces se realizan programas de canales, fundamentalmente porque existen macro-instrucciones que liberan al programador de aquella tarea. Sin embargo, con el objeto de dar una visión más completa se analizarán algunos aspectos y elementos de programación relacionados con los programas de canales aún cuando se utilicen siempre algunas macro-instrucciones que permitirán darle más claridad a los ejemplos.

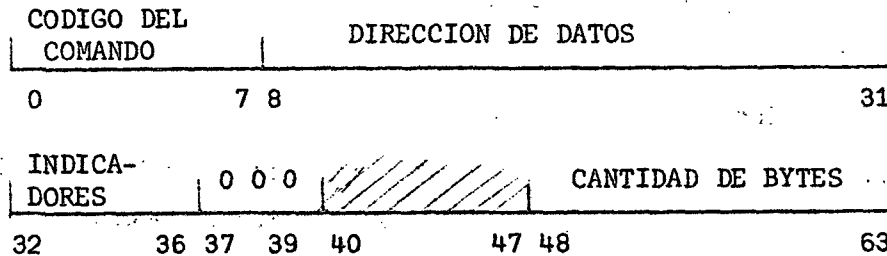
Una entrada típica de datos es la lectura de una tarjeta como asimismo una salida típica es la impresión de una línea. Al aparecer una instrucción de entrada o salida en el programa del usuario y ser analizada esa instrucción por la UCP, ésta notifica al canal que corresponde que debe iniciar su programa. La comunicación la realiza a través de una instrucción privilegiada START INPUT OUTPUT (SIO) cuya ejecución genera una cadena de hechos, el primero de los cuales es transferir la palabra de dirección del canal (Channel Address Word-CAW) desde los bytes 72 a 75 de la memoria principal al canal designado por SIO. El formato de la palabra CAW es el siguiente:

LLAVE	0	0	0	0	DIRECCION DEL COMANDO
0	3	4	7	8	31

Existe la posibilidad de "proteger" la memoria principal de posibles destrucciones de información inadvertidas. Con este objeto, se divide en bloques de 2048 bytes cada uno y a cada bloque se le asigna un registro de cuatro bits. Las combinaciones de cuatro bits pueden considerarse "llaves" de almacenamiento.

El almacenamiento se efectúa solamente si las combinaciones de la llave de protección proporcionada por la CAW (o la PSW) y la llave del almacenamiento coinciden o cuando la llave de la CAW (PSW) tiene un valor cero.

La dirección del comando es la dirección efectiva del primer comando en el programa de canal. El formato de la palabra de comando de canal (Channel Command Word - CCW) es el siguiente:



Los comando son seis:

- READ
- READ BACKWARD
- SENSE
- WRITE
- CONTROL y
- TRANSFER IN CHANNEL

Read, causa la transferencia de información desde un dispositivo de entrada, a memoria. Read Backward, permite leer información desde una cinta magnética que se mueve en dirección contraria a la que se utilizó para grabarla. Sense, transfiere información de estados de un dispositivo a la memoria principal. Write, causa la transferencia de información desde memoria a un dispositivo de salida. Control, se utiliza para acciones como: rebobinar cinta magnética, saltarse archivos o registros físicos, etc. Transfer in Channel, es una instrucción de bifurcación

Los comandos se definen en forma similar a las instrucciones de Assembler:

[nombre] CCW operando1,operando2,operando3,operando4

donde:

- nombre: es optativo y permite identificar el comando
- operando1: código del comando, especifica qué función se realizará. Ocupa los bits 0-7
- operando2: expresión reubicable que identifica el área I/O. Ocupa los bits 8-31

operando3 : indicadores (flags), permiten establecer encadenamiento de datos o comandos, saltarse áreas, etc. Ocupa los bits 32-36

operando4 : cantidad de bytes (Count), expresión absoluta que indica la cantidad de bytes que se transfieren o saltan. Ocupa los bits 48-63.

Significado de los bits indicadores (FLAGS)

- BIT 32. Encadenamiento de datos (Chain Data-CD). Si está en ON (1) indica que el área designada por el próximo comando, utiliza la operación indicada en el primer comando del último grupo con encadenamiento de datos.
- BIT 33. Encadenamiento de comandos (Chain Command-CC). Si está en ON (1) indica que quedan comandos por procesar. Si está en OFF (0) indica que ese es el último comando que se ejecuta.
- BIT 34. Suprime error de longitud (Suppress Length Informacion-SLI). Si está en ON (1) y el bit CD está en OFF (0) en la última CCW usada queda suprimida la indicación de longitud incorrecta. Si están en ON los bits CC y SLI tiene lugar el encadenamiento de comandos.

En la tabla que sigue se indican los efectos y acciones que produce SLI en combinación con CD y CC. La entrada "longitud incorrecta (LI)" significa que la indicación está disponible para el programa en la CSW, un doble guión significa que la indicación se suprime, parada, que se detiene la operación en el subcanal.

<u>Bits Indicadores</u>			<u>Acción e Indicación</u>	
CD	CC	SLI	OP.NORMAL	OP.INMEDIATA
0	0	0	Parada, LI	Parada, --
0	0	1	Parada, --	Parada, --
0	1	0	Parada, LI	Encad.comandos
0	1	1	Encad.comandos	Encad.comandos
1	0	0	Parada, LI	Parada, --
1	0	1	Parada, LI	Parada, --
1	1	0	Parada, LI	Parada, --
1	1	1	Parada, LI	Parada, --

BIT 35. Salta (Skip-SK). Si está en ON (1) especifica la supresión de transferencia de información al almacenamiento principal durante una operación de lectura, lectura hacia atrás, o consulta. Si está en OFF (0), tiene lugar la transferencia normal de datos.

BIT 36. Interrupción controlada por programa (Program Controlled Interruption-PCI). Si está en ON (1), determina que el canal genera una interrupción una vez extraída la CCW. El comando no se ejecuta, y la condición PCI queda establecida en el bit 40 de la CSW.

Se consideran las operaciones fundamentales READ y WRITE cuyos códigos de operación correspondientes son:

	<u>Binario</u>	<u>Hexadecimal</u>
READ	00000010	X'02'
WRITE	00000001	X'01' no produce espacio. X'09' un espacio después de imprimir.

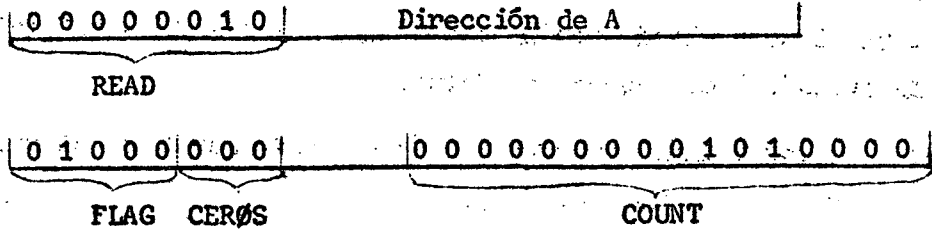
Ejemplo 77.

Leer 3 tarjetas y ubicarlas en las zonas A,B,y C de memoria:

R	EQU	X'02'
A	DS	CL80
B	DS	CL80
C	DS	CL80

CCW	R,A,X'40',80
CCW	R,B,X'40',80
CCW	R,C,0,80

Los distintos operandos ocuparán los lugares respectivos en la CCW como se indica en el siguiente ejemplo:



Ejemplo 78.

Se desea leer un registro de 80 caracteres de tal forma que:

- Los primeros 20 vayan a A
- Los siguientes 50 vayan a B
- Los siguientes 5 vayan a C
- Los siguientes 5 vayan a D

R	EQU	X'02'	
CD	EQU	X'80'	
A	DS	CL20	1 0 0 0 0 0 0 0 0 0
B	DS	CL50	↑ CD
C	DS	CL5	
D	DS	CL5	

CCW	R,A,CD,20
CCW	O,B,CD,50
CCW	O,C,CD,5
CCW	O,D,X'00',5

Si hay encadenamiento de datos, el código que aparece en el lugar de operando 1 es ignorado en los comandos que siguen al primero del grupo con CD.

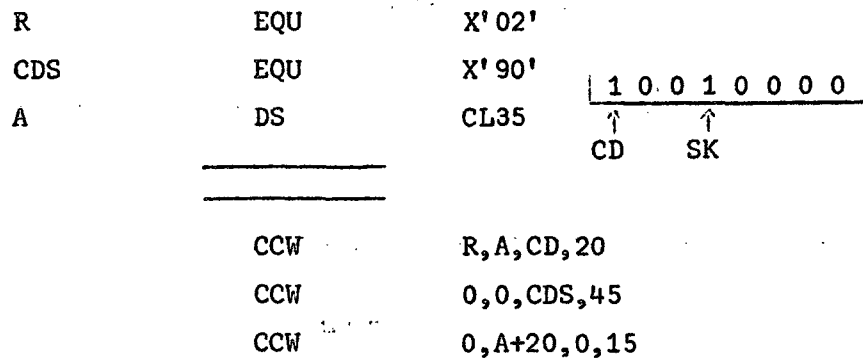
Combinaciones de CD y CC:

<u>CD</u>	<u>CC</u>	<u>Acción</u>
0	0	No hay encadenamiento. La CCW corriente es la última.
0	1	Encadenamiento de comandos
1	0	Encadenamiento de datos
1	1	Encadenamiento de datos

Ejemplo 79.

Se quiere leer un registro de 80 caracteres de tal forma que:

Los primeros 20 vayan a A
 los últimos 15 vayan a A+20



Ejemplo 80.

Leer un registro de 100 caracteres de tal forma que:

Los primeros 10 se salten
 los siguientes 15 vayan a A
 los siguientes 20 se salten
 los siguientes 25 vayan a A+15
 los últimos se salten

) 150 (

R	EQU	X'02'
CD	EQU	X'80'
S	EQU	X'10'
CDS	EQU	X'90'
A	DS	CL40

CCW	R,0,CDS,10
CCW	0,A,CD,15
CCW	0,0,CDS,20
CCW	0,A+15,CD,25
CCW	0,0,S,30

Variante. Definiendo SLI EQU X'AO' 1 0 1 0 0 0 0 0 se puede colocar en lugar del penúltimo comando y siguiente

CCW 0,A+15,SLI,25

el error que se produzca por longitud incorrecta se suprime.

Ejemplo 81.

Grabar 132 caracteres que están en EDIT:

W	EQU	X'01'
EDIT	DS	CL132

CCW W,EDIT,0,132

Ejemplo 82.

Grabar tres registros A, B y C de 132 caracteres cada uno, como un solo registro de cinta.

CCW	W,A,CD,132
CCW	0,B,CD,132
CCW	0,C,0,132

Resultado:

I R G	132 caracteres de A	132 caracteres de B	132 caracteres de C	I R G
-------------	------------------------	------------------------	------------------------	-------------

Ejemplo 83.

Grabar tres registros A, B y C de 132 caracteres cada uno, como tres registros físicos de cinta.

```

CCW      W,A,CC,132
CCW      W,B,CC,132
CCW      W,C,O,132
    
```

Resultado:

I R G	132 caracteres de A	I R G	132 caracteres de B	I R G	132 caracteres de C	I R G
-------------	------------------------	-------------	------------------------	-------------	------------------------	-------------

Las macro instrucciones que se utilizarán son:

Macro CCB (Command Control Block)

Formato:

Nombre del bloque CCB SYSnnn, nombre del programa de canal

donde:

- nombre del bloque: es el nombre que identifica la CCB
- SYSnnn: nombre simbólico de la unidad a la que está asociada la CCB
- nombre del programa del canal: nombre que identifica la primera CCW.

Macro EXCP (Execute Channel Program)

Formato:

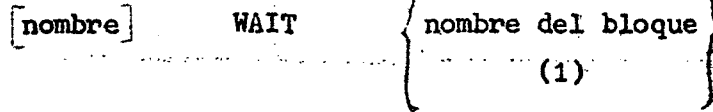
[nombre] EXCP { nombre del bloque
(1) }

donde:

- nombre: es el identificador de la macro
 - nombre de bloque: es el identificador de la macro CCB. Puede darse directamente como expresión simbólica o a través del RUG1.
- La macro permite iniciar una operación de entrada/salida.

Macro WAIT

Formato:



donde:

nombre: es el identificador de la macro

nombre del bloque: es el identificador de la macro CCB.

Se utiliza para permitir que una operación de entrada/salida, iniciada con la macro EXCP, sea terminada antes de proseguir el proceso:

Observación: Estas macro-instrucciones son válidas para el Sistema de Operación DOS (Disk Operating System)

Tal como la PSW mantiene estados del programa, se tiene para los canales la palabra de estado del canal (Channel Status Word - CSW). El formato de la CSW es el siguiente:

LLAVE	0000		
0	3 4	7 8	31
BITS DE ESTADO		CANTIDAD DE BYTES	
32	47	48	63

En los bits de estado quedan registradas situaciones como: dispositivo de entrada o salida ocupado, término del programa de canal, término de la operación de entrada o salida, diferencia entre la longitud de un registro y la cantidad de bytes especificada en la CCW, etc.

La CSW ocupa los bytes 64 a 71 de memoria principal.

Ejemplo 84.

```

PROGX  START 0
        BALR 2,0
        USING *,2
LECT    EXCP LEET1
        WAIT LEET1
        CLC  A(2),=C*/*
        BE   FIN
    
```

(Continúa)

(Continuación)

```

EXCP  IMPL1
WAIT  IMPL1
MVC   A,A1
      B    LECT
FIN   EØJ
LEET1 CCB  SYSIN,LEER
IMPL1 CCB  SYSLST,IMPRE
LEER  CCW  R,A,CD,20
      CCW  0,0,CDS,20
      CCW  0,A+20,CD,15
      CCW  0,0,SK,25
IMPRE CCW  W,A1,0,133
A1    DC  C' '
A     DC  CL40' '
      DC  CL92' '
R     EQU  2
W     EQU  1
CD    EQU  X'80'
CDS   EQU  X'90'
SK    EQU  X'10'
      END  PRØGX

```

La macro-instrucción EXCP LEET1 inicia la ejecución del programa de canales que permitirá leer tarjetas. El operando LEET1 es el identificador de la macro-instrucción CCB cuyos operandos son: SYSIN que es el nombre simbólico asignado al dispositivo que permitirá la lectura y, LEER que es el identificador del primer comando del programa de canales.

El primer comando lee información (20 bytes) que se almacena en el área A, indica al mismo tiempo que hay encadenamiento de datos (CD), esto es, que el comando siguiente mantendrá la misma orden de lectura. En el segundo comando se especifica nuevamente encadenamiento de datos, pero además, se indica que se saltarán 20 bytes (CDS). En el tercer comando continúa el encadenamiento de datos y se leen 15 bytes que se almacenan en A+20. En el último comando se indica que se saltarán (SK) 25 bytes y además que ahí termina el programa de canales.

La operación de lectura se realiza sin que el proceso continúe debido a la macro-instrucción WAIT LEET1. Al finalizar la lectura se ejecuta la instrucción de assembler CLC A(2),=C'/*' que permite detectar el término de los datos, esto es, que se ha leído la tarjeta que tiene /* en las columnas 1 y 2 respectivamente. Si eso no ha ocurrido, el proceso continúa imprimiendo la información de la tarjeta leída e iniciando nuevamente el ciclo.

12. Definiciones de macros^{1/}

Una definición de macro consiste de:

- a) Una proposición de encabezamiento de la definición de la macro, cuyo formato es:

Nombre	Operación	Operandos
Blanco	MACRO	Blanco

- b) Una proposición prototipo de la macro-instrucción cuyo formato es:

Nombre	Operación	Operandos
Parámetro simbólico o blanco	Símbolo	Ninguno, uno o varios parámetros simbólicos

El objeto de la proposición prototipo es especificar el código de operación mnemotécnico y el formato de todas las macro-instrucciones que se refieren a la definición de la macro. Los parámetros simbólicos se utilizan en la definición de la macro-instrucción para representar el campo de nombre y los operandos en la macro-instrucción correspondiente. En el campo operando pueden existir desde 0 hasta 200 parámetros simbólicos separados entre sí por coma.

c) Ninguna, una o varias proposiciones modelo, instrucciones de ensamble condicional, etc. Una proposición modelo consta de uno a cuatro campos que son de izquierda a derecha: nombre, operación, operando y comentario. De las proposiciones modelo que están en la definición de la macro se generan las secuencias deseadas de proposiciones del lenguaje de ensamble.

^{1/} Traducción resumida de la Parte II del Manual Assembler Language de IBM.

d) Una proposición de salida de la definición de la macro, cuyo formato es:

Nombre	Operación	Operandos
Blanco	MEND	Blanco

Ejemplo 85.

Se tiene la siguiente definición de macro:

	MACRO	Encabezamiento
NAME	MOVE &T0,&FROM	Prototipo
NAME	ST 2,SAVE	Modelo
	L 2,&FROM	Modelo
	ST 2,&T0	Modelo
	L 2,SAVE	Modelo
	MEND	Salida

y la macro-instrucción

HERE MOVE FIELD A, FIELD B

de acuerdo a ella se generarán las siguientes proposiciones de lenguaje de ensamble:

HERE	ST	2,SAVE	Generada
	L	2,FIELD B	Generada
	ST	2,FIELD A	Generada
	L	2,SAVE	Generada

Ejemplo 86.

	MACRO	Definición
NAME	MOVE &TY, &P, &T0, &FROM	de la macro
NAME	ST &TY 2,SAVEAREA	
	L &TY 2,&P,&FROM	
	ST &TY 2,&P,&T0	
	L &TY 2,SAVEAREA	
	MEND	

(Continúa)

(Continuación)

HERE	MØVE	D, FIELD, A, B	}	macro-inst.
HERE	STD	2, SAVEAREA		
	LD	2, FIELDB	}	instrucciones generadas
	STD	2, FIELDA		
	LD	2, SAVEAREA		

A. Instrucciones de ensamble condicional

Las instrucciones de ensamble condicional permiten al programador:

- Definir y asignar valores a símbolos SET que pueden ser usados para variar partes de proposiciones generadas y
- Variar la secuencia de proposiciones generadas. Así el programador puede usar esas instrucciones para generar diferentes secuencias de proposiciones desde la misma definición de macro.

Hay 13 instrucciones de ensamble condicional:

LCLA	SETA	AIF	ANOP	.	GBLA
LCLB	SETB	AGO		.	GBLB
LCLC	SETC	ACTR		.	GBLC

LCLA, LCLB y LCLC se usan para definir y asignar valores iniciales a símbolos SET locales.

SETA, SETB y SETC se usan para asignar valores Aritméticos, Binarios y Caracteres a símbolos SET.

AIF, AGO y ANOP pueden ser utilizados en conjunto con símbolos de secuencia para variar la secuencia en la cual las proposiciones son compiladas.

El programador puede chequear atributos asignados por el compilador a símbolos u operandos de macro-instrucciones para determinar cuales proposiciones van a ser procesadas.

ACTR puede ser usada para limitar el número de saltos AIF y AGO ejecutados en alguna compilación.

B. Símbolos SET

Difieren de los parámetros simbólicos en tres aspectos:

- a) Dónde pueden ser utilizados en un programa fuente en lenguaje assembler.
- b) Cómo se les asigna valores.
- c) Cómo los valores asignados se pueden cambiar.

Pueden ser utilizados dentro y fuera de una definición de macro.

Se les asigna valores por las instrucciones de ensamble condicional SETA, SETB y SETC y por declaraciones locales y globales.

Los valores asignados a cada símbolo SETA, SETB y SETC no son restrictivos.

1) Definición de símbolos SET

Los símbolos SET deben ser definidos por el programador, antes que ellos sean usados. Cuando un símbolo SET es definido, se le asigna un valor inicial. Los símbolos SET pueden recibir nuevos valores por medio de las instrucciones SETA, SETB y SETC. Un símbolo SET es definido cuando aparece como operando de una instrucción LCLA, LCLB o LCLC.

∅	LCLA	uno o más símbolos variables (que van a ser usados como
	LCLB	símbolos SET), separados por comas
	LCLC	

LCLA, LCLB y LCLC se usan para definir y asignar valores iniciales a símbolos SETA, SETB y SETC respectivamente, los que reciben valores 0, 0 y carácter nulo.

No deben definirse símbolos que empiecen con ϵ ^{1/}SYS. Todas las instrucciones LCLA, LCLB o LCLC en una definición de macro deben aparecer inmediatamente después de la proposición prototipo y de todas las instrucciones GBLA, GBLB o GBLC. Todas las instrucciones LCLA, LCLB o LCLC externas a definiciones de macros deben aparecer después de todas ellas, en el programa fuente, después de todas las instrucciones GBLA, GBLB o GBLC, externas a definiciones de macros, antes de todas las instrucciones de ensamble condicional y de proposiciones PUNCH y REPRO externas a definiciones de macros y antes de la primera sección de control del programa.

1/ Se ha usado ϵ en vez de $\&$

C. Atributos

El compilador asigna atributos a operandos de macro-instrucciones y a símbolos en el programa. A esos atributos se puede referir sólo en instrucciones de ensamblaje condicional. Hay seis clases de atributos. Ellos son: tipo, longitud, escala, entero, cuenta y número.

Si un operando de una macro-instrucción externa es un símbolo antes de la sustitución, entonces los atributos del operando son los mismos que los correspondientes atributos del símbolo. Este debe aparecer en la entrada nombre de una proposición del lenguaje de ensamblaje o como operando de un EXTRN en el programa. La proposición debe estar fuera de las definiciones de macros y no debe contener símbolos variables.

Si un operando de una macro-instrucción interna es un parámetro simbólico, entonces los atributos del operando son los mismos que los correspondientes atributos del operando de la macro-instrucción externa.

Cada atributo tiene una notación asociada:

Tipo	Type	T'
Longitud	Length	L'
Escala	Scaling	S'
Entero	Integer	I'
Cuenta	Count	K'
Número	Number	N'

Si un operando de una macro-instrucción es una sublista, el programador puede referirse a los atributos de la sublista o de cada operando de la sublista. Los atributos de tipo, longitud, escala y entero de una sublista son los mismos que los correspondientes atributos del primer operando en la sublista.

El programador puede referirse a un atributo en la siguiente forma:

a) En una proposición que está fuera de las definiciones de macros, debe escribir la notación del atributo seguida de un símbolo. Ejemplo: T'NAME se refiere al atributo tipo del símbolo NAME.

b) En una proposición que está dentro de una definición de macro, debe escribir la notación del atributo seguida por un parámetro simbólico. Ejemplo: L'εNAME se refiere al atributo longitud de los caracteres, en la macro-instrucción, correspondientes al parámetro simbólico εNAME. L'εNAME(2) se refiere al atributo longitud del segundo operando de la sublista correspondiente a εNAME.

1) Atributo tipo (T')

El atributo tipo de un operando de una macro-instrucción o un símbolo es una letra.

Ejemplo 87.

A	constante de dirección tipo A, longitud implícita, alineada
C	constante carácter
F	constante de punto fijo, palabra completa, longitud implícita, alineada
I	instrucción de máquina
M	macro-instrucción
W	instrucción CCW
	etc.

2) Atributos longitud (L'), escala (S') y entero (I'). Los atributos longitud, escala y entero de operandos de macro-instrucciones y símbolos son valores numéricos.

Ejemplo de atributo de longitud:

Ejemplo 88.

A1	DS	CL8
B2	DC	CL2'AB'
HIORD	MVC	A1(L'B2),B2
LOORD	MVC	A1+L'A1-L'B2(L'B2),B2

El atributo longitud de A1 es 8 y el de B2 es 2. La proposición HIORD mueve 2 bytes de B2 a la dirección A1. La proposición LOORD mueve 2 bytes a la dirección $A1+8-2=A1+6$ (dos bytes del extremo derecho).

El atributo longitud de * es igual a la longitud de la instrucción donde aparece, excepto en EQU * donde es 1.

3) Los atributos de escala y enteros son proporcionados para símbolos que identifican proposiciones DC o DS de punto fijo, punto flotante y decimal. Para punto fijo y flotante el atributo de escala lo da el modificador de escala. El atributo entero es una función de S' y L' .

a) Punto fijo $I' = 8 * L' - S' - 1$

Ejemplo 89.

HALFCØN	DC	HS6'-25.93'
ØNECØN	DC	FS8'100.3E-2'
L' de HALFCØN = 2		S' = 6
$I' = 8 * 2 - 6 - 1 = 9$		

L' de ONECON = 4		S' = 8
$I' = 8 * 4 - 8 - 1 = 23$		

b) Punto flotante $I' = 2 * (L' - 1) - S'$

c) Para punto decimal S' es el número de dígitos decimales a la derecha del punto decimal (empaquetado y zona dígito). I' es el número de dígitos decimales a la izquierda del punto (zona-dígito).

$I' = 2 * L' - S' - 1$ (empaquetado)

$I' = L' - S'$ (Zona-dígito)

Ejemplo 90.

FIRST	DC	P' +1.25'
SECOND	DC	Z' -543'
THIRD	DC	Z' 79.68'
FOURTH	DC	P' 79.68'

FIRST	L'=2	S'=2	I'=1
SECOND	L'=3	S'=0	I'=3
THIRD	L'=4	S'=2	I'=2
FOURTH	L'=3	S'=2	I'=3

4) Atributo cuenta (K'). El programador puede referirse al atributo cuenta sólo de operandos de macro-instrucciones.

El atributo cuenta es un valor igual al número de caracteres en el operando de una macro-instrucción después de la sustitución por símbolos variables, excluyendo las comas. Si el operando es una sublista, el atributo cuenta incluye los paréntesis de comienzo y término y las comas.

Si una macro-instrucción contiene como operandos, símbolos variables, los caracteres que reemplazan los símbolos variables son usados para determinar el atributo cuenta.

5) Atributo número (N'). El programador puede referirse al atributo número de operandos de macro-instrucciones solamente.

El atributo número es un valor igual al número de operandos en una sublista de operandos. El número de operandos en una sublista es igual al número de comas más uno.

Ejemplo 91.

(A,B,C,D,E)	5 op.
(A, ,C,D,E)	5 op.
(A,B,C,D,,)	6 op

D. Símbolos de secuencia

Consiste de un punto seguido por 1 a 7 caracteres alfanuméricos el primero de los cuales debe ser alfabético.

Ejemplo 92.

.READER	.A23456
.LØØP2	.X4F2
.N	.S4

La entrada nombre de una proposición puede contener un símbolo de secuencia. Ellos permiten al programador variar la secuencia en la cual las proposiciones son procesadas por el compilador.

Si un símbolo de secuencia aparece en la entrada nombre de una macro-instrucción y la correspondiente proposición prototipo contiene un parámetro simbólico en la entrada nombre, el símbolo de secuencia no reemplaza al parámetro simbólico donde él es usado, en la definición de la macro.

Ejemplo 93.

```

MACRO
&NAME MOVE &T0,&FROM
&NAME ST 2,SAVEAREA
L 2,&FROM
ST 2,&T0
L 2,SAVEAREA
MEND

.SYM MOVE FIELDA,FIELDDB
ST 2,SAVEAREA
L 2,FIELDDB
ST 2,FIELDA
L 2,SAVEAREA
    
```

E. SETA

La instrucción SETA puede ser usada para asignar un valor aritmético a un símbolo SETA.

Un símbolo	SETA	Una expresión aritmética
SETA		SETA

La expresión es evaluada como un valor aritmético de 32 bits con signo el cual se asigna al símbolo SETA en la entrada nombre. Los valores mínimo y máximo permitidos son -2^{31} y $+2^{31}-1$.

La expresión puede consistir de un término o una combinación aritmética de términos.

Los operandos aritméticos que pueden utilizarse son + - * y /

Ejemplo 94.

```

&AREA+X' 2D' I' &N/25
&BETA*10 &EXIT-S' &ENTRY+1
L' &HERE+32 29
    
```

1) Evaluación de expresiones aritméticas:

- a) A cada término se le da su valor numérico
- b) La operación se realiza de izquierda a derecha. Pero, la multiplicación y/o división se realizan antes de suma o resta.
- c) El resultado computado se asigna al símbolo SETA.

Ejemplo 95.

```
(L' &HERE+32)*29
&AREA+X' 2D' /((&EXIT-S' &ENTRY+1)
&BETA*10*(I' &N/25/((&EXIT-S' &ENTRY+1))
```

El valor aritmético asignado a un símbolo SETA es sustituido para el símbolo SETA cuando él se usa en una relación aritmética. Si el símbolo SETA no se utiliza en una expresión aritmética, el valor aritmético es convertido a un entero sin signo, con eliminación de ceros no significativos.

Ejemplo 96.

	MACRØ	
&NAME	MØVE	&TØ, &FRØM
	LCLA	&A, &B, &C, &D
&A	SETA	10
&B	SETA	12
&C	SETA	&A- &B
&D	SETA	&A+ &C
&NAME	ST	2, SAVEAREA
	L	2, &FROM&C
	ST	2, &TØ&D
	L	2, SAVEAREA
	MEND	
HERE	MØVE	FIELDA, FIELDB
HERE	ST	2, SAVEAREA
	L	2, FIELDB2
	ST	2, FIELDA8
	L	2, SAVEAREA

} valor absoluto de &C y &D

Ejemplo 97.

```

MACRO
&NAME MOVE &TO, &FROM
      LCLA &A
&A SETA 5
&NAME ST 2,SAVEAREA
      L 2,&FROM&A
&A SETA 8
      ST 2,&TO&A
      L 2,SAVEAREA
MEND

HERE MOVE FIELDA, FIELDB
HERE ST 2,SAVEAREA
      L 2, FIELDDB5
      ST 2, FIELDDB8
      L 2,SAVEAREA
    
```

Un símbolo SETA se puede utilizar con un parámetro simbólico para referirse a un operando en una sublista de operandos. Si un símbolo SETA se utiliza para este objeto, debe tener asignado un valor entre 1 y 100, ambos inclusivos.

Ejemplo 98.

```

MACRO
ADDX &NUMBER, &REG
LCLA &LAST
&LAST SETA N' &NUMBER
      L &REG, &NUMBER(1)
      A &REG, &NUMBER(&LAST)
      ST &REG, &NUMBER(1)
MEND

ADDX (A,B,C,D,E),3
      L 3,A
      A 3,E
      ST 3,A
    
```


F. SETC

La instrucción SETC es usada para asignar un valor carácter a un símbolo SETC.

Un símbolo SETC Un operando
 SETC

El operando puede consistir de: atributo tipo, una expresión carácter, o una concatenación de notaciones de subcadenas y expresiones carácter. Un símbolo SETA puede aparecer en el operando de una proposición SETC. El resultado es la representación carácter del valor decimal, sin signo, sin ceros no significativos. Si el valor es cero, se usa el cero decimal.

1) Atributo tipo

Debe aparecer sólo en el campo de operando.

Ejemplo 99.

εTYPE SETC T'εABC

se asigna a εTYPE la letra que es el atributo tipo del operando de macro-instrucción que corresponde al parámetro simbólico εABC.

2) Expresión carácter

Una expresión carácter consiste de cualquier combinación de caracteres encerrados entre apóstrofos (máx. 127 caracteres).

El valor carácter encerrado entre apóstrofos en el campo operando es asignado al símbolo SETC. La longitud máxima de valor carácter es ocho caracteres. Si se asigna un valor mayor se consideran los 8 caracteres de la izquierda.

Evaluación de expresiones carácter.

Asignación del valor carácter AB%4 al símbolo εALPHA

εALPHA SETC 'AB%4'

Se puede concatenar más de una expresión carácter en una expresión única colocando un punto entre el apóstrofo que termina una expresión y el que inicia la siguiente.

Ejemplo 100.

EBETA SETC 'ABCDEF'

puede escribirse como

EBETA SETC 'ABC','DEF'

Símbolos variables se concatenan de acuerdo a las reglas vistas anteriormente.

Ejemplo 101.

Si se asignó AB%4 a &ALPHA, se tiene que:

&GAMMA SETC '&ALPHA.RST' o

&GAMMA SETC '&ALPHA'..'RST'

asignan a &GAMMA el valor AB%4RST.

Para representa un & que no es parte de un símbolo variable es necesario colocar dos &. Ambos formarán parte del valor carácter asignado al símbolo SETC.

Ejemplo 102.

&AND SETC 'HALF&&'

se asigna HALF&& a &AND

3) Notación de subcadenas

Las subcadenas de valores carácter permiten al programador asignar parte de un valor carácter a un símbolo SETC. Se debe indicar:

- a) El valor carácter
- b) La parte del valor carácter que se desea asignar (subcadena de valor carácter).

La notación consiste de una expresión carácter seguida por dos expresiones aritméticas separadas por coma y encerradas entre paréntesis. Las expresiones aritméticas deben ser permitidas como operandos de SETA.

La primera expresión aritmética indica el primer carácter de la expresión carácter que va a ser asignado al símbolo SETC. La segunda expresión aritmética indica el número de caracteres consecutivos que va a ser asignado.

Ejemplo 103.

```
' &ALPHA' (2,5)
'AB%4' (&AREA+2,1)
' &ALPHA'.'RST' (6, &A)
'ABC&GAMA' (&A, &AREA+2)
```

4) Concatenación de notaciones subcadena y expresiones carácter.

Si una notación subcadena sigue una expresión carácter, ambas pueden ser concatenadas colocando un punto entre ellas.

Ejemplo 104.

```
&GAMMA      SETC      ' &ALPHA'.' &BETA' (2,3)
```

Si a la notación subcadena le sigue otra, o una expresión carácter, no es necesario el punto (opcional).

Ejemplo 105.

```
&WORK      SETC      ' &ALPHA' (1,4)' &ABC'
&WORD      SETC      ' &ALPHA' (1,4)' &ABC' (1,3)
```

Uso de símbolos SETC

Ejemplo 106.

```
MACRØ
&NAME      MØVE      &TØ, &FRØM
           LCIC      &PREFIX
&PREFIX    SETC      'FIELD'
&NAME      ST        2,SAVEAREA
           L        2, &PREFIX&FRØM
           ST        2, &PREFIX&TØ
           L        2,SAVEAREA
MEND
HERE       MØVE      A,B
HERE       ST        2,SAVEAREA
           L        2, FIELDB
           ST        2, FIELDA
           L        2,SAVEAREA
```

Ejemplo 107.

```

MACRO
&NAME MOVE &T0, &FROM
LCLC &PREFIX
&PREFIX SETC 'FIELD'
&NAME ST 2,SAVEAREA
L 2,&PREFIX&FROM
&PREFIX SETC 'AREA'
ST 2,&PREFIX&T0
L 2,SAVEAREA
MEND

HERE MOVE A,B
HERE ST 2,SAVEAREA
L 2,FIELD B
ST 2,AREAA
L 2,SAVEAREA

```

Ejemplo 108.

```

MACRO
&NAME MOVE &T0, &FROM
LCLC &PREFIX
&PREFIX SETC '&T0' (1,5)
&NAME ST 2,SAVEAREA
L 2,&PREFIX&FROM
ST 2,&T0
L 2,SAVEAREA
MEND

HERE MOVE FIELD A, B
HERE ST 2,SAVEAREA
L 2,FIELD B
ST 2,FIELD A
L 2,SAVEAREA

```

G. SETB

La instrucción SETB puede ser usada para asignar el valor binario 0 ó 1 a un símbolo SETB.

Un símbolo SETB	SETB	0 ó 1, (0) o (1) o una expresión lógica encerrada entre paréntesis.
--------------------	------	---

La expresión lógica es evaluada para determinar si es "verdad" o es "falsa". De acuerdo a ello se asigna al símbolo SETB el valor 1 ó 0 respectivamente.

Una expresión lógica consiste de un término o una combinación lógica de términos. Los términos que pueden utilizarse solos, son relaciones aritméticas, relaciones de carácter y símbolos SETB. Los operadores lógicos son AND, ØR y NØT. Pueden haber dos operadores lógicos consecutivos, sólo si el primero es AND u ØR y el segundo NØT.

Una relación aritmética consiste de dos expresiones aritméticas conectadas por operadores de relación. Una relación de carácter consiste de dos cadenas de caracteres conectados por operadores de relación. Los operadores son:

EQ, NE, LT, GT, LE, GE

Si dos valores carácter son de distinta longitud, se compara el más corto con el más largo. Los operadores lógicos y de relación deben ir precedidos y seguidos por un blanco, al menos.

Ejemplo 109.

```
( &AREA+2      GT      29 )
('AB%4'        EQ      ' &ALPHA' )
(T' &ABC        NE      T' &XYZ )
(T' &P12        EQ      'F' )
(&AREA+2      GT      29      ØR      &B )
(NØT          &B      AND      &AREA+X' 2D'      GT      29 )
```

1) Evaluación de expresiones lógicas

a) Cada término es evaluado y se le asigna su valor lógico (verdadero o falso).

b) Las operaciones lógicas son realizadas de izquierda a derecha. Sin embargo los NOT se realizan antes de los AND y éstos antes de los OR.

c) El resultado computado es el valor asignado al símbolo SETB en el campo nombre.

2) Uso de símbolos SETB

Si un símbolo SETB se utiliza en el operando de una instrucción SETA o en relaciones aritméticas en los operandos de instrucciones AIF y SETB los valores binarios 1 (verdad) y 0 (falso) son convertidos a los valores aritméticos +1 y +0 respectivamente. Si un símbolo SETB se utiliza en el operando de una instrucción SETC, o en relaciones de carácter, en los operandos de instrucciones AIF y SETB o en otras proposiciones, los valores binarios 1 y 0 son convertidos a los caracteres 1 y 0 respectivamente.

Ejemplo 110.

MACRO		
NAME	MOVE	EQ, &FROM
	LCLA	&A1
	LCLB	&B1, &B2
	LCLC	&C1
&B1	SETB	(L' &T0 EQ 4)
&B2	SETB	(S' &T0 EQ 0)
&A1	SETA	&B1
&C1	SETC	' &B2'
	ST	2, SAVEAREA
	L	2, &FROM&A1
	ST	2, &T0&C1
	L	2, SAVEAREA
	MEND	
HERE	MOVE	FIELDA, FIELDB
HERE	ST	2, SAVEAREA
	L	2, FIELDDB1
	ST	2, FIELDABO
	L	2, SAVEAREA

H. AIF (Salto condicional)

Se utiliza para alterar, de acuerdo a una condición, la secuencia en la cual las proposiciones del programa fuente serán procesadas.

Símbolo de secuencia	AIF	Una expresión lógica encerrada entre paréntesis, seguida por un símbolo de secuencia.
----------------------	-----	---

La expresión lógica se evalúa para determinar si es verdadera o falsa. Si la expresión es verdadera, la proposición nombrada con el símbolo de secuencia es la siguiente procesada. En caso contrario continúa la secuencia normal.

Si AIF está en una definición de macro, el símbolo debe aparecer en el campo nombre de una proposición de la definición. Si AIF está fuera de la definición de macro, debe ocurrir lo mismo con el símbolo de secuencia.

Ejemplo 111.

```

MACRØ
EN MOVE      ET, &F
           AIF      (T' &T      NE      T' &F).END
           AIF      (T' &T      NE      'F').END
EN ST       2,SAVEAREA
           L        2, &F
           ST       2, &T
           L        2,SAVEAREA
.END MEND
    
```

I. AGO (Salto incondicional)

Similar a la anterior pero efectúa el salto sin que sea necesario una condición anterior.

Símbolo de secuencia	AGO	Símbolo de secuencia
----------------------	-----	----------------------

Ejemplo 112.

```

MACRO
ENAME MOVE      ET, &F
      AIF      (T' &T   EQ   'F').FIRST
      AGO      .END
      .FIRST   AIF      (T' &T   NE   T' &F).END
ENAME ST        2, SAVEAREA
      L        2, &F
      ST       2, &T
      L        2, SAVEAREA
      .END     MEND

```

J. ACTR (Contador de ciclos del ensamble condicional)

Se utiliza para limitar el número de saltos AGO y AIF ejecutados en una definición de macro o en el programa fuente principal.

⌘ ACTR Una expresión SETA válida

Esta proposición debe seguir inmediatamente a declaraciones globales o locales si las hay. Un contador se coloca (inicializa) con el valor del operando. Cada vez que se ejecuta un salto AGO o AIF, el contador es disminuido en 1. Si el contador es cero antes de la resta, el compilador realiza una de las dos acciones siguientes:

a) Si se está procesando una macro-definición, el procesamiento de ella y cualquiera anterior en el nido es terminado y la proposición siguiente en el programa principal es procesada.

b) Si el programa principal está siendo procesado, el ensamble condicional es terminado, y la porción del programa generada hasta ahí, es compilada.

K. ANOP (No operación de ensamble)

Se utiliza para realizar saltos a proposiciones nombradas por símbolos o símbolos variables.

Un símbolo de secuencia ANOP ⌘

Ejemplo 113.

```

MACRØ
&NAME MØVE    &T, &F
      LCLC    &TYPE
      AIF     (T' &T    EQ    'F').FTYPE
&TYPE SETC   'E'
.FTYPE ANØP
&NAME ST&TYPE 2,SAVEAREA
      L&TYPE  2, &F
      ST&TYPE 2, &T
      L&TYPE  2,SAVEAREA
MEND

```

L. Características adicionales

Permiten al programador:

- a) Terminar el procesamiento de una macro definición
 - b) Generar mensajes de error
 - c) Definir símbolos SET globales
 - d) Definir símbolos SET subindicados
 - e) Usar símbolos variables del sistema
 - f) Preparar palabras clave y macro definiciones de modo mixto y escribir palabras clave y macro-instrucciones de modo mixto.
- 1) MEXIT (Salida de una macro definición)

Se utiliza para indicar al compilador que deberá terminar el procesamiento de una macro definición. La forma típica de esta instrucción es:

```

Símbolo de secuencia      MEXIT      Ø

```

No debe confundirse con MEND que indica el término de una macro definición y debe estar al final de ella.

Ejemplo 114.

```

MACRO
ENAME MOVE    ET, &F
      AIF    (T' &T    EQ    'F').ØK
      MEXIT
.ØK   ANØP
ENAME ST      2,SAVEAREA
      L      2,&F
      ST    2,&T
      L      2,SAVEAREA
      MEND
    
```

2) Proposición MNØTE

Se utiliza para generar un mensaje y para indicar qué código de severidad de error, si existe, va a ser asociado con el mensaje. El código de error es sólo información para el programador.

Símbolo de secuencia MNØTE operando

Operando puede ser:

- a) código de error, 'mensaje'
- b) , 'mensaje'
- c) 'mensaje'

En b) y c) se supone código de error = 1. La proposición MNØTE sólo puede ser usada en una macro definición. Se pueden utilizar símbolos variables para generar la proposición MNØTE.

Si el código de severidad es un *, MNØTE no es considerado un mensaje de error, sino un comentario.

La proposición MNØTE aparece en el listado con un número de proposición en el punto donde fue generada. Si el código de severidad era un entero o un Ø, el número de proposición es colocado en una lista de MNØTE y otras proposiciones de error. No ocurre lo mismo en caso de *.

Ejemplo 115.

```

MACRØ
ENAME MØVE      ET, &F
      AIF      (T' &T      NE      T' &F).M1
      AIF      (T' &T      NE      'F').M2
ENAME  ST      2,SAVEAREA
      L        2, &F
      ST      2, &T
      L        2,SAVEAREA
      MNØTE    *, 'MØVE GENERATED'
      MEXIT
.M1    MNØTE    8, 'TYPE NØT SAME'
      MEXIT
.M2    MNØTE    8, 'TYPE NØT F'
      MEND

```

M. Símbolos variables locales y globales

Los siguientes son símbolos variables locales

- a) Parámetros simbólicos
- b) símbolos SET locales
- c) símbolos variables del sistema.

Los símbolos SET globales son los únicos símbolos variables globales.

GBLA,GBLB y GBLC definen símbolos SET globales

LCLA,LCLB y LCLC definen símbolos SET locales.

Los símbolos SET globales pueden comunicar valores entre proposiciones en una o más macro-definiciones y proposiciones fuera de las macro definiciones. Si un símbolo SET local es definido en dos o más macro-definiciones, o en una macro definición y fuera de macro-definiciones, el símbolo es considerado diferente en cada caso. Sin embargo, un símbolo SET global es el mismo en todos los lugares.

1) Definición de símbolos locales y globales.

∅	GBLA	uno o más símbolos variables
	GBLB	que van a ser usados como símbolos
	GBLC	SET globales, separados por comas.

Deben seguir inmediatamente a la instrucción prototipo dentro de una macro-definición. Fuera de la macro-definición debe seguir después de todas ellas y antes de la primera sección de control. Dentro de la macro-definición deben aparecer antes de las proposiciones LCLA, LCLB y LCLC.

2) Uso de símbolos SET globales y locales.

Cada ejemplo de los que se verán consiste de dos parte. La primera parte es un programa fuente en lenguaje de ensamble. La segunda muestra las proposiciones que serán generadas por el ensamblador después que él procese las proposiciones del programa fuente.

Ejemplo 116.

	MACRO	
ENAME	LØADA	
	LCLA	EA
ENAME	LR	15,EA
EA	SETA	EA+1
	MEND	
	LCLA	EA
FIRST	LØADA	
	LR	15,EA
	LØADA	
	LR	15,EA
	END	FIRST
FIRST	LR	15,0
	LR	15,0
	LR	15,0
	LR	15,0
	END	FIRST

Los símbolos EA de la macro-definición y fuera de ella son distintos, luego EA SETA EA+1 no afecta al valor del símbolo EA para las instrucciones generadas.

Ejemplo 117.

```

MACRØ
&NAME LØADA
      GBLA      &A
&NAME LR      15, &A
&A    SETA      &A+1
      MEND

      GBLA      &A
FIRST LØADA
      LR      15, &A
      LØADA
      LR      15, &A
      END      FIRST

FIRST LR      15, 0
      LR      15, 1
      LR      15, 1
      LR      15, 2
      END      FIRST

```

Ejemplo 118.

```

MACRØ
&NAME LØADA
      LCLA      &A
&NAME LR      15, &A
&A    SETA      &A+1
      MEND

MACRØ
LØADB
      LCLA      &A
      LR      15, &A
&A    SETA      &A+1
      MEND

```

(Continúa)

(Continuación)

```

FIRST  LØADA
        LØADB
        LØADA
        LØADB
        END      FIRST
FIRST  LR      15,0
        LR      15,0
        LR      15,0
        LR      15,0
        END      FIRST

```

Ejemplo 119.

```

MACRØ
&NAME  LØADA
        GBLA   &A
&NAME  LR     15, &A
&A     SETA   &A+1
        MEND
MACRØ
&NAME  LØADB
        GBLA   &A
        LR     15, &A
&A     SETA   &A+1
        MEND
FIRST  LØADA
        LØADB
        LØADA
        LØADB
        END      FIRST
FIRST  LR     15,0
        LR     15,1
        LR     15,2
        LR     15,3
        END      FIRST

```

Ejemplo 120.

	MACRØ	
ENAME	LØADA	
	GBLA	EA
ENAME	LR	15, EA
EA	SETA	EA+1
	MEND	
	MACRØ	
	LØADB	
	GBLA	EA
	LR	15, EA
EA	SETA	EA+1
	MEND	
	LCLA	EA
FIRST	LØADA	
	LØADB	
	LR	15, EA
	LØADA	
	LØADB	
	LR	15, EA
	END	FIRST
FIRST	LR	15, 0
	LR	15, 1
	LR	15, 0
	LR	15, 2
	LR	15, 3
	LR	15, 0
	END	FIRST

BIBLIOGRAFIA

1. Computer Usage Company, Programación del Sistema IBM/360, México, Editorial Limusa-Wiley, S.A., 1968, 365 pp.
2. Struble, George, Assembler Language Programming: The IBM System/360, USA, Addison-Wesley, 1969, 434 p.
3. Steinhart, Robert F. y Pollack, Seymour V., Programming the IBM System/360, USA, Holt Rinehart and Winston, 1970, 576 pp.
4. Thomas Alex, Jr., System 360 Programming, USA, Rinehart Press San Francisco, 1971, 273 pp.
5. Vickers, Frank, D., Introduction to Machine and Assembly Language: System/360/370, USA, Holt Rinehart and Winston, 1971, 303 pp.
6. Kardonsky de F., Adriana y Sánchez C., Víctor, Curso de Programación, Santiago, Universidad de Chile, Facultad de Ciencias Físicas y Matemáticas, 1967, 167 pp.
7. Sánchez C., Víctor, Apuntes de Assembler, Santiago, Universidad de Chile, Facultad de Ciencias Físicas y Matemáticas, 1967, 170 pp.
8. Sánchez C., Víctor, Manual de Assembler, Tomo I, Santiago, Universidad de Chile, Facultad de Ciencias Físicas y Matemáticas, 1970, 102 pp.
9. IBM System Products Division, Introducción para Programadores a la Arquitectura las Instrucciones y el Lenguaje Compaginador del Sistema IBM/360, Argentina, 1971, 274 pp.
10. IBM System Products Division, Assembler Language, USA, 1967, 155 pp.
11. IBM System Products Division, IBM System/370 Principles of Operation, USA, 1974, 326 pp.