Universidad Nacional de La Matanza

Arquitectura de los sistemas /370; /390; System z

Contacto

Lic. José N Castro <jncastro@gylgroup.com>

Lic. Aldo Fernández Villalba <avillalba@gylgroup.com>

1. La arquitectura del sistema y el assembler

Programación en memoria interna

Z1 (1936); **ABC** (1937); **Colosus** (1943); **ENIAC** (1946): Utilizan diferentes formas de programación desde tableros de interruptores.

EDVAC (1948) «*Electronic Discrete-Variable Automatic Computer*»: Arquitectura binaria. Johan von Neumann incorpora el concepto de programa almacenado dentro de la memoria de la computadora.

Lenguaje ensamblador IBM (1957): Programa escrito en codigo legible y traducido a lenguaje de la máquina por un programa.

El lenguaje «Assembler»

- Originalmente desarrollado para no tener que codificar los programas en lenguaje de máquina
- Tiene características diferentes a otros lenguajes de programación:
  - Está fuertemente vinculado a la arquitectura interna de la máquina
  - Puede manejar datos a nivel de bytes o de bits
  - Tiene acceso a bloques de control del sistema
  - Ofrece mayor eficiencia en tiempos de ejecución
  - No se usa generalmente para desarrollo de aplicaciones
  - Es el lenguaje utilizado para desarrollar «Exits»
  - Suele utilizarse para sub-rutinas de alto rendimiento y funciones especiales, que pueden ser llamadas desde programas en algún otro lenguaje de alto nivel.

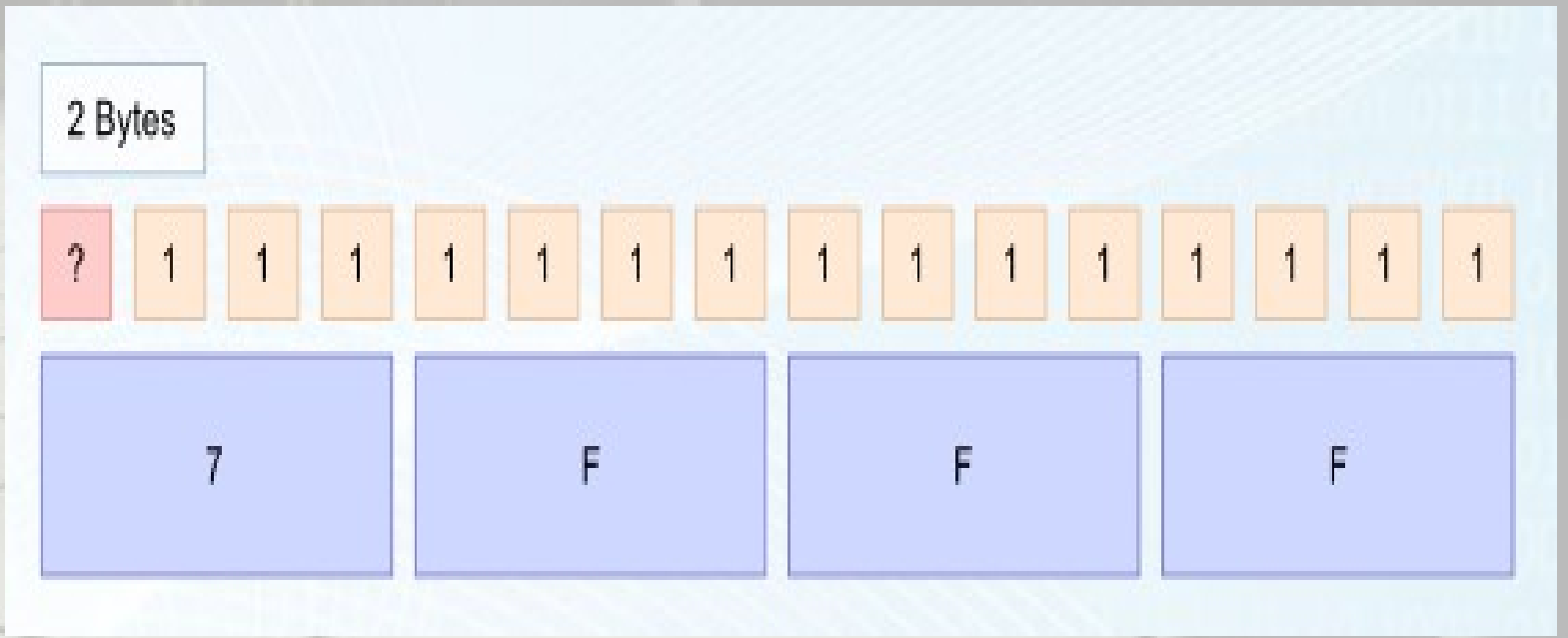
Variables del sistema

Ciertas unidades de información, o *variables*, tienen longitudes específicas y deben estar en posiciones de memoria alineadas. Es decir, la dirección de memoria es múltiplo de la longitud de la unidad de información. El estándar de nomenclatura es el siguiente:

- «**halfword**» Dos bytes consecutivos, alineados en un límite de 2 bytes, es decir, el bit más a la derecha de la dirección contiene un cero. La mayoría de las instrucciones ejecutables tienen un requisito de alineación a media palabra.
- «**fullword**» o «*word*». Cuatro bytes consecutivos, alineados en un límite de 4 bytes. Los dos bits más a la derecha de la dirección contienen ceros.
- «**doubleword**» Ocho bytes consecutivos alineados en un límite de 8 bytes. Los tres bits más a la derecha de la dirección contienen ceros.
- «**Quadword**» Dieciséis bytes consecutivos alineados en un límite de 16 bytes. Los cuatro bits más a la derecha de la dirección contienen ceros.

Aritmética binaria

- Las instrucciones de aritmética binaria trabajan con valores enteros, en «halfwords», «fullwords», etc.
- Los valores con signo, utilizan el primer bit como signo: 0 para positivo y 1 para negativo.
- El mayor valor con signo que puede contener una halfword es +7FFF (32.767); y una Fullword +7FFFFFF (2.147.483.647)



Aritmética binaria

- Los números negativos se almacenan como complemento a 2. El complemento a 2 de un valor es el inverso de los bits más 1.
- Para representar el negativo, primero se invierten los bits, y luego se suma 1)

Registros del sistema

- Son espacios en un área reservada de memoria que funcionan como variables internas del sistema
- Hay ***cuatro tipos de registros*** en el sistema con diferentes usos

Type	General purpose	Access	Control	Floating point
Number	16	16	16	4
Size	64 Bits	32 Bits	64 Bits	64 Bits

Control registers

- Administran el «*virtual address space*» y brindan información sobre interrupciones de E/S y otras cosas. Son de uso interno de la CPU, el programa no tiene acceso a estos

Floating point registers

- Almacenan valores en forma algebraica, exponencial y fraccionaria. Permiten manipular números de punto flotante. Hay cuatro registros de coma flotante numerados 0, 2, 4 y 6. El manejo de punto flotante es programación assembler avanzada

Access registers

- Registros que permiten direccionar datos en otro «*address space*». El uso de estas técnicas es assembler avanzado, es utilizado por productos del sistema operativo

Registros de uso general

- Hay ***16 registros*** de uso general, numerados de **0** a **15**
- Simbólicamente se los referencia como **R0** a **R15**. Suelen utilizarse macros estándar para definir estos símbolos como constantes simbólicas con los valores correspondientes
- Ocupan ***4 bytes (32 bits)*** de memoria cada uno, en los sistemas recientes son de ***8 bytes (64 bits)***
- Son recursos de acceso rápido
- Tienen usos determinados en la arquitectura del sistema para direccionamiento, operaciones aritméticas enteras, y funciones de control del proceso
- Hay instrucciones, como la división, que utilizan un par de registros, una combinación consecutiva par e impar

Registros con usos especiales por convención

- `R0` No se utiliza en instrucciones de programa
- `R1` Puntero a lista de parámetros al llamar a sub programas externos
- `R14` Dirección de memoria para el retorno desde un sub programa externo
- `R12` Registro base del programa
- `R13` Puntero a un área de memoria de 72 bytes para salvar el contenido de otros registros
- `R15` Código de terminación de un programa o sub programa

Arquitectura del sistema

Direccionamiento por registro Base + Desplazamiento

Ejemplo: Dirección de memoria: `00001246`

Registro Base, ej. R12 contiene: `00001000`

Dirección: `R12` + `246`

Se indica con la notación: **`246(R12)`**

Arquitectura del sistema

Direccionamiento por

registro Base + registro Indice + Desplazamiento

Ejemplo: Dirección de memoria: `00001246`

Registro base R12 contiene: `00001000`

Registro índice R3 contiene: `00000200`

Dirección: `R12` + `R3` + `46`

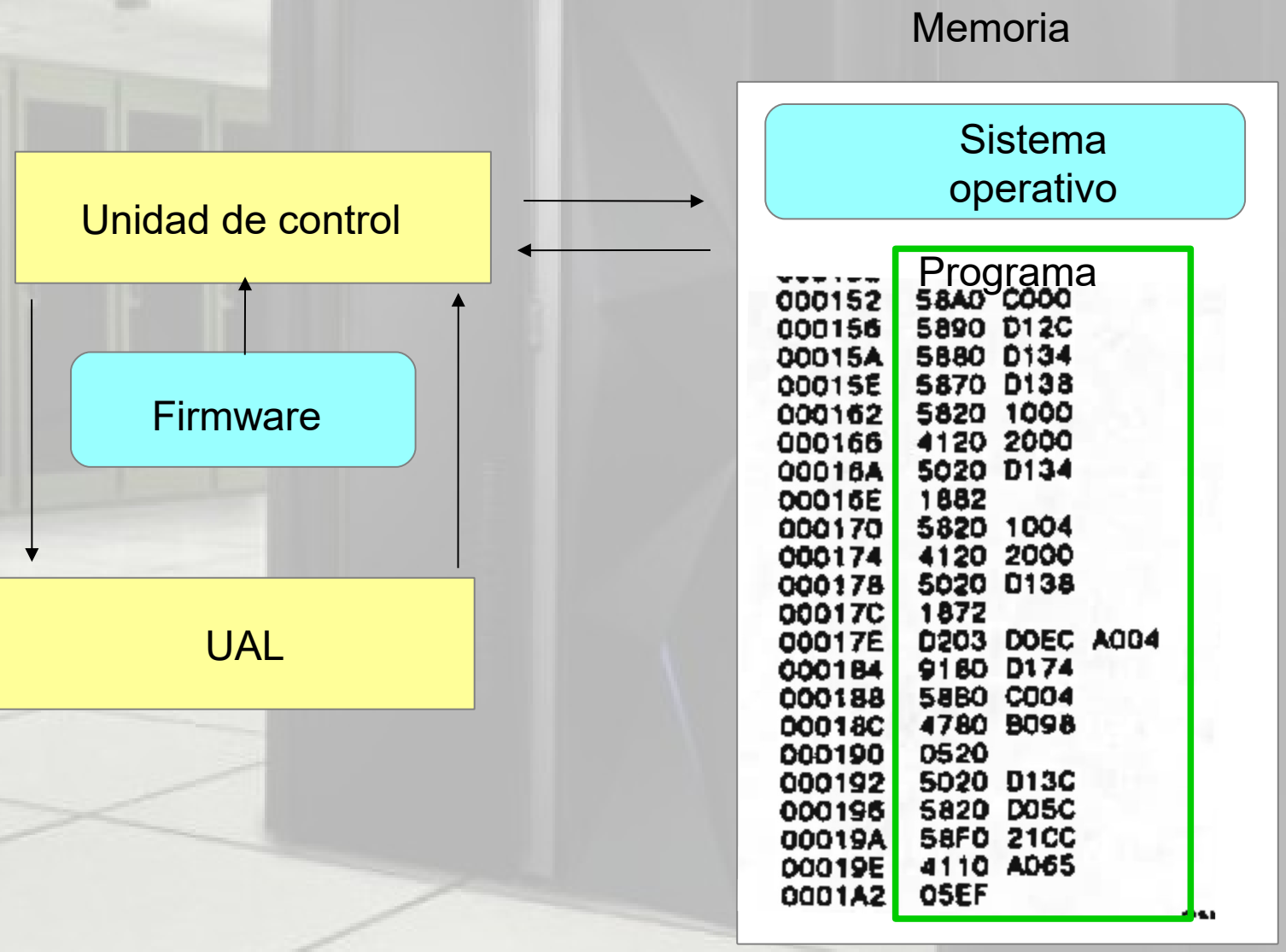
Se indica con la notación: **`46(R3,R12)`**

Documentos públicos de IBM

- z/Architecture Reference Summary
  Cartilla resumida de las instrucciones de la arquitectura del sistema, formatos, códigos, etc.

- https://www.ibm.com/support/pages/sites/default/files/2021-05/SA22-7871-10.pdf

Programación en memoria interna



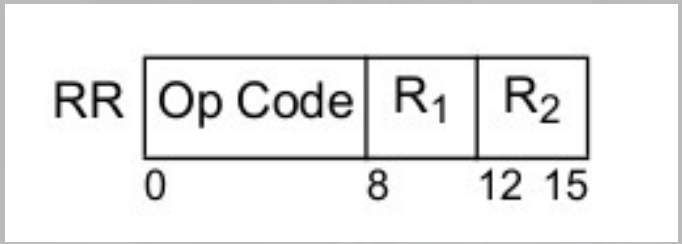
Arquitectura del sistema

Tipo de operación RR, de Registro a Registro

Add Register (AR)

Código de operación 1A: **0001 1010**

1A43



Ciclo interno de proceso

```
000524 5880 D132
000528 1A43
000530 C2C3 D0EC A004
```

Formato RR:

Op Code	R1	R2
0 7	8 12	15

1	A	4	3
0001	1010	0100	0011

Add registers

`AR R1,R2` Formato: **RR**; Código: **1A**

1. Obtenga el código de instrucción desde la dirección actual (+ 528)
2. Como los 1ros bites son cero, el largo es de 2 bytes, sume 2 a la dirección actual para obtener la dirección de la instrucción siguiente
3. La instrucción es 1A (Sumar registros), entonces
4. Copie el contenido del 1er registro (registro 4) al area de trabajo
5. Sumele el contenido del segundo registro (registro 3)
6. Mueva el resultado al primer registro (registro 4)
7. Pase a la siguiente instrucción.

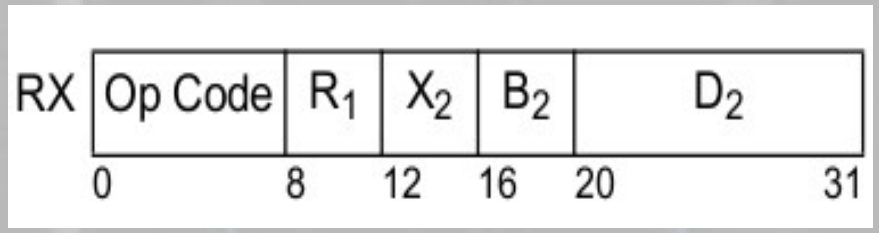
Arquitectura del sistema

Tipo RX-a, Registro a Memoria indexada

A (Instrucción **Add)**

Código de operación **5A**: 0101 1010

5A6340FF

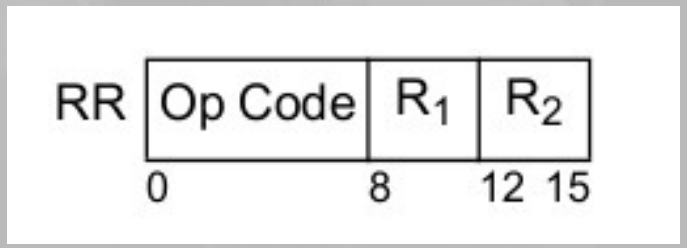


Assembler versus la arquitectura del sistema

Add Register (Suma un registro general a otro registro)

```
AR 6,2
```

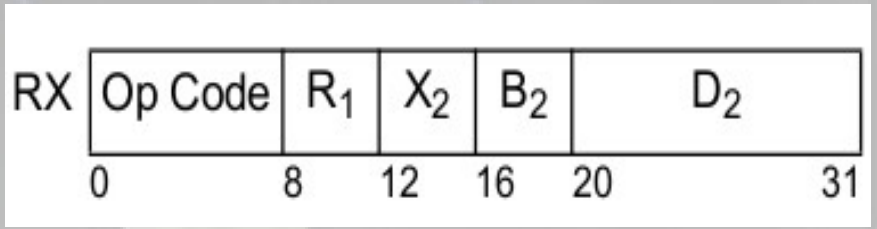
1A62



Add (Suma una variable a un registro general)

```
A 6,VAR02   ==>   A 6,255(3,4)
```

5A6340FF



Compilador Cobol

```
PROCEDURE DIVISION USING X Y.
000-MAIN-LOOP  SECTION.
    ADD X TO Y.
    ADD 1 TO CRASH2.
    IF CRASH2 > 38
       ADD CRASH1 TO CRASH2.
010-MAIN-LOOP-EXIT.
    GOBACK.
```

```
00032   *000-MAIN-LOOP
00033   ADD
     0001CC  D201 D18E 8000          MVC   398(2,13),0(8)
     0001D2  960F D18F               OI    399(13),X'0F'
     0001D6  D202 D195 7000          MVC   405(3,13),0(7)
     0001DC  960F D197               OI    407(13),X'0F'
     0001E0  FA21 D195 D18E          AP    405(3,13),398(2,13)
     0001E6  D202 7000 D195          MVC   0(3,7),405(13)
     0001EC  940F 7000               NI    0(7),X'0F'
     0001F0  960F 7002               OI    2(7),X'0F'
00034   ADD
     0001F4  FA20 900B A031          AP    11(3,9),49(1,10)
     0001FA  940F 900B               NI    11(9),X'0F'
     0001FE  F822 900B 900B          ZAP   11(3,9),11(3,9)
00035   IF
     000204  F921 900B AD2F          CP    11(3,9),47(2,10)
     00020A  5880 C004               L     11,4(0,12)
     00020E  4700 B116               BC    13,278(0,11)
00035   ADD
>>   000212  FA20 900B 900A          AP    11(3,9),10(1,9)
>>   000218  940F 900B               NI    11(9),X'0F'
     00021C  F822 900B 900B          ZAP   11(3,9),11(3,9)
     000222                  GN=3    EQU   *
00036   *010-MAIN-LOOP-EXIT
00037   GOBACK
     000222  47F0 B136               BC    15,310(0,11)
     000226  9120 D054               TM    84(13),X'20'
```

Documentos públicos de IBM

- *High Level Assembler for z/OS & z/VM & z/VSE Language Reference*. Sintaxis de las sentencias assembler y otra información del lenguaje

  https://www-40.ibm.com/servers/resourcelink/svc00100.nsf/pages/zOSV2R3sc264940/$file/

Documentos públicos de IBM y otros

- z/Architecture Principles of Operation (SA22-7832-09).
  Definición detallada de «z/Architecture». Referencia para programadores de assembler. Describe cada función a nivel de detalle

  https://publibfp.dhe.ibm.com/epubs/pdf/dz9zr000.pdf

- Introduction to S/370 Principles of Operation
  Jim Morrison's MVS Goodies. Como entender el "*Principles of Operations*"

  http://cbttape.org/~jmorrison/s370asm/html/tut-POPs-001.html

2. Programando Assembler

Variables en Assembler

- En assembler las variables y constantes deben ser definidas explícitamente
- Para definir las variables se utilizan las instrucciones **DS**, «*Define Storage*» o **DC** «*Define Constant*»

LABEL	DC	Multiplication factor	Type	Type Extension	Program Type	Modifier	Nominal Value	
ABCD	DC	1	X			L1	'23'	X'23'
	DC	1	C			L1	'Hello'	X'C885939396'
	DC	1	F			L1	'00'	X'00000000'
FLD1	DC	1	A			L1	(ABCD)	X'00001249'
FLD2	DC	3	X			L1	'01'	X'010101'
FLD3	DC	1	C			L3	'NO'	X'D5D640'

Variables en Assembler

- Para ciertos tipos de datos, DS y DC el ensamblador también realiza la alineación que se requiera
- Si se especifica un modificador de longitud, no se realiza alineación

Type	Format	Alignment without length
A	Value of address	Fullword
B	Binary digits	Byte
C	Characters	Byte
F	Fixed point binary	Fullword
H	Fixed point binary	Halfword
P	Packed decimal	Byte
V	Externally defined address	Fullword
X	Hexadecimal digits	Byte
Z	Zoned decimal	Byte

Definiendo variables en assembler

```
PRUEBA1  CSECT
FULL01   DC    F'255'      Constante fullword
FULL02   DS    F           Fullword
X01      DC    C'X'        Literal de un byte
HALF01   DC    H'26'       Constante halfword
HALF02   DC    H'-26'      Constante halfword negativa
HEX01    DC    4XL2'AA'    4 constantes tipo X de dos bytes c/u
*
ADD01    DC    A(FULL02)   Constante de direccion interna
ADD02    DC    V(RUTIN99)  Constante de direccion externa
         END   ,           Fin del codigo
```

Formatos de instrucciones

Hay una gran cantidad de formatos diferentes de instrucciones, sólo veremos como ejemplo algunas de ellas

- `I` Immediate
- `RI` Register and Immediate
- `RR` Register and Register
- `RS` Register and Storage
- `RX` Register and Indexed Storage
- `SI` Storage and Immediate
- `SS` Storage and Storage

Formatos de instrucciones

```
PRUEBA2  CSECT
         SVC    13                 Immediate
         AHI    R1,32              Register and Immediate
         AR     R2,R3              Register and Register
         STM    R14,R12,12(R13)    Register and Storage
         ST     R4,220(R2,R3)      Register and Indexed Storage
         MVI    220(R11),X'FF'     Storage and Immediate
         MVC    0(4,R5),220(R4)    Storage and Storage
         YREGS  ,                  Definir constantes simbolicas Rx
         END    ,                  Fin del codigo
```

Direccionamiento Base + Desplazamiento

Para poder programar assember con esta modalidad, manejada por el ensamblador, es necesario realizar dos acciones:

- 1. Indicarle al ensamblador que se va a utilizar un determinado registro general como base para el programa. Por estándar, el registro 12
- 2. Cargar en el registro que se usará como base la dirección de memoria donde se cargará el programa en el momento de ser ejecutado, para que sirva de dirección base

Direccionamiento Base + Desplazamiento

Se utilizan dos instrucciones en el programa para cumplir con esta tarea:

- `USING` Instrucción al ensamblador, no genera código máquina. Instruye al ensamblador a utilizar un determinado registro general como base del programa asumiendo un determinado valor en el mismo

- `BALR` Instrucción de máquina. Carga en un registro la dirección de memoria, y, opcionalmente, bifurca hacia otra dirección que se indique

Direccionamiento con Base - Desplazamiento

```
PRUEBA3  CSECT
         BALR  R12,0          Cargar el Registro general 12
         USING *,R12          Utilizar el reg 12 como base del pgm
         LA    R13,SAVEAREA   Cargar en R13 la direccion de save area
         A     R2,HALF01      Sumar 26 al contenido de R2
*
RETURN   EQU   *
         BR    R14             Volver al programa iniciador
*
HALF01   DC    H'26'           Constante halfword
SAVEAREA DC    18F'0'
*
         YREGS ,               Definir constantes simbolicas Rx
         END   ,               Fin del codigo
```

Aritmética decimal

En el sistema ***EBCDIC***, los valores hexadecimales x'**F0**' a x'**F9**' representan los caracteres del **0** al **9**

```
 0   1   2   3   4   5   6   7   8   9
F0  F1  F2  F3  F4  F5  F6  F7  F8  F9
```

Cuando los datos se «*empaquetan*», se elimina el primer medio byte o «*nibble*», llamado «*zona*», salvo el de la extrema derecha que pasa a ser el signo del número

```
F0 F1 F4 F3 => 00 14 3F
```

El «nibble», o medio byte reservado para el signo conteniendo el valor «**C**» indica un número con signo positivo, un valor «**D**» es signo negativo.

El sistema también reconoce como positivos **A**; **E**; y **F**; y **B** como negativo

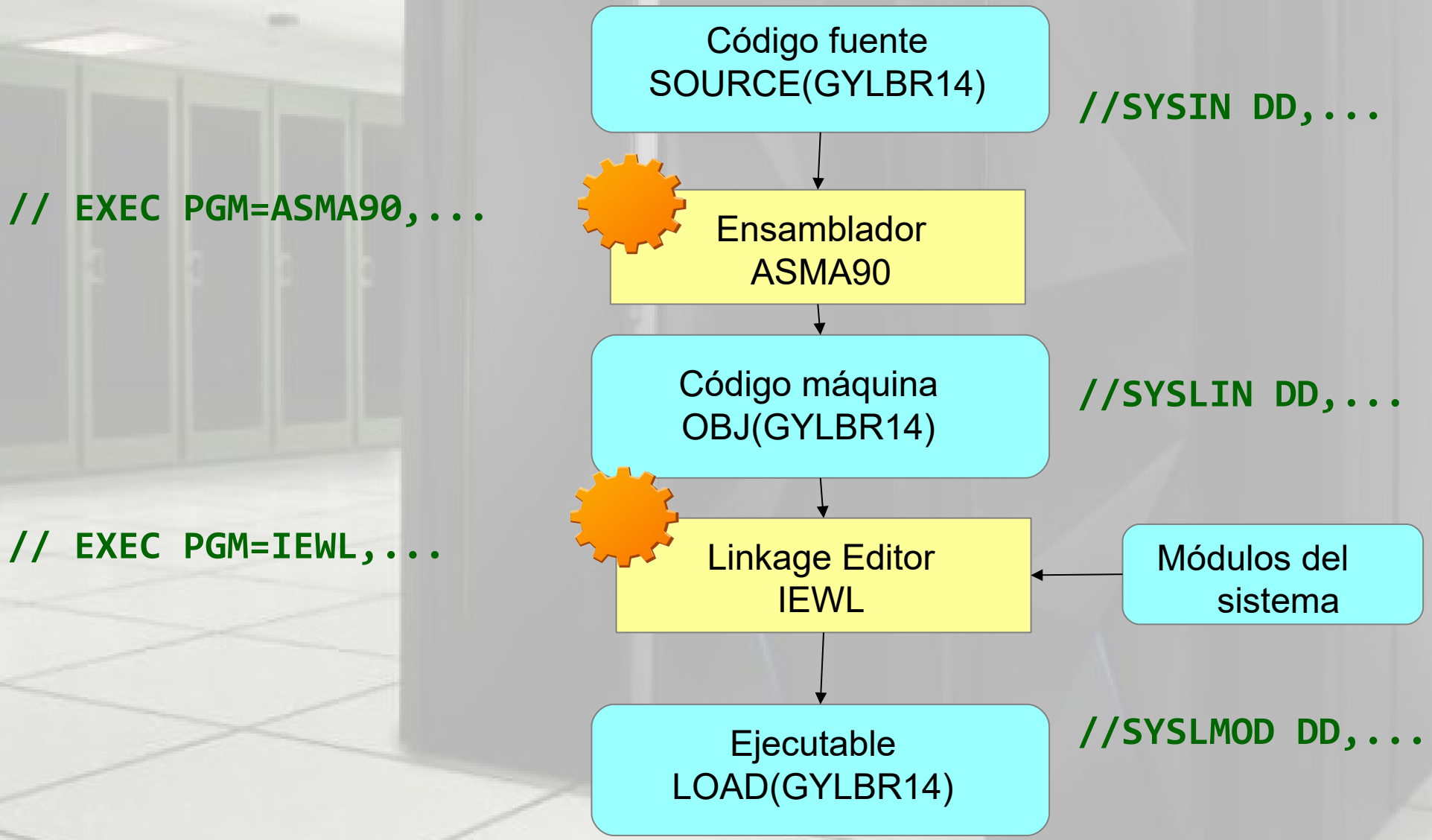
Formato decimal empaquetado

```
PRUEBA4  CSECT
CHR01    DC    CL6'000123'     Formato EBCDIC
PACK01   DC    PL6'123'        Formato empaquetado positivo
PACK02   DC    PL6'-123'       Formato empaquetado negativo
         END   ,               Fin del codigo
```

IEFBR14, El programa que hace nada...

```
IEFBR14  CSECT ,       Comienzo de la seccion de control
         SLR   15,15   Puesta a cero del registro 15
         BR    14      Volver al programa iniciador
         END   ,       Fin del codigo
```

Batch del ensamblador «Assembler»



¿Dudas, Preguntas?

Contactos

José N Castro <jncastro@gylgroup.com>

Aldo Fernández Villalba <avillalba@gylgroup.com>