Software de base

Notas de clase basadas en el libro
Software de Sistemas, Beck, Addison Wesley

Dr. Francisco Javier Zaragoza Martínez
franz@correo.azc.uam.mx

UAM Azcapotzalco
Departamento de Sistemas

Trimestre 2009 Primavera

Contenido

Evaluación

- Habrá dos exámenes (E) y un proyecto (P).
- Los exámenes valdrán un total de 50 puntos.
- El proyecto valdrá un total de 50 puntos.
- S requiere $E \geq 30$, $P \geq 30$ y $E + P \geq 60$.
- B requiere $E \geq 30$, $P \geq 30$ y $E + P \geq 73$.
- MB requiere $E \geq 30$, $P \geq 30$ y $E + P \geq 87$.

Exámenes

- El primer examen consistirá del tema Introducción y parte del tema Ensambladores.
- El segundo examen consistirá de parte del tema Ensambladores y de los temas Cargadores y ligadores Macroprocesadores.

Part I

Introducción

Contenido

Software de aplicación

- Según su objetivo, existen fundamentalmente dos tipos de programas:
- Los programas de aplicación tienen como objetivo el de resolver algún problema independientemente de la computadora con que se cuente. En este caso se usa a la computadora como una herramienta.

Ejemplos

- Procesadores de texto.
- Hojas de cálculo.
- Navegadores.
- Lectores de correo, etc.

Contenido

Software de base

- Existen otros programas que se consideran básicos para el uso de una computadora determinada independientemente de la aplicación que se le quiera dar.
- A estos los llamaremos software de base.
- La naturaleza de estos programas implica que su diseño está íntimamente ligado a la estructura de la máquina donde se ejecutan.

Contenido

Ejemplos de software de base

Ejemplos

- Ensambladores.
- Cargadores.
- Ligadores.
- Macroprocesadores.
- Compiladores.
- Sistemas operativos.

- En este curso estudiaremos los primeros cuatro y después llevarán un curso de compiladores y dos cursos de sistemas operativos.

Ensambladores

- Los ensambladores traducen instrucciones nemónicas a código máquina.
- La dependencia se debe a que los formatos de instrucciones, modos de direccionamiento, códigos de operación, etc., dependen por completo del procesador de la máquina.
- A pesar de las diferencias, existen también muchas características del diseño de estos programas que no dependen del procesador empleado. En este curso exploraremos ambos lados de la moneda.

Contenido

Las máquinas SIC y SIC/XE

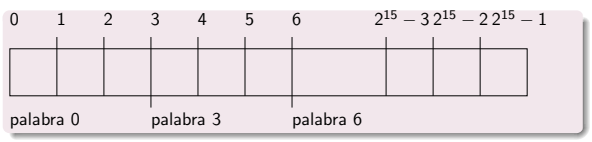
- El SIC y el SIC/XE son dos máquinas imaginarias que tienen características comunes con la mayoría de las máquinas reales.
- Esto incluye la existencia de más de un modelo de la misma máquina con cierta compatibilidad entre sí.
- Tenga en cuenta que los detalles de cada máquina son distintos y que esto afecta el diseño del software de base correspondiente.

Estructura de la máquina SIC

- La memoria consta de bytes de 8 bits.
- Tres bytes consecutivos forman una palabra (word) de 24 bits.
- Cada byte tiene una dirección de 15 bits.
- Esto significa que la memoria tiene 2^{15} bytes.
- Cada palabra tiene como dirección la dirección de su primer byte.

Memoria del SIC

- En este diagrama, cada cuadrito representa un byte de 8 bits y cada tres cuadritos consecutivos representan una palabra de 3 bytes.
- Es similar a un arreglo `char mem[32768]`.



Registros del SIC

- Tiene 5 registros de 24 bits, cada uno de ellos con un uso específico:

Registro	Número	Uso
A	0	Acumulador aritmético
X	1	Registro índice
L	2	Registro de enlace
PC	8	Contador de programa
SW	9	Palabra de estado

- Son similares a variables `int A, X, L, PC, SW`.

Registros PC, L y SW

- El registro PC contiene la dirección de la instrucción a ejecutar.
- El registro L almacena la dirección de regreso cuando se salta a una subrutina.
- El registro SW contiene información en sus bits (llamados banderas).

Tipos de datos del SIC

- Los enteros se almacenan como números binarios de 24 bits, usando el complemento a 2 para los valores negativos.

Ejemplo

Representar los enteros decimales 2007 y -2007 usando 24 bits:

```
00000000 00000111 11010111 = 00 07 D7
11111111 11111000 00101001 = FF F8 29
```

- Los caracteres se almacenan en ASCII usando 8 bits.

Instrucciones del SIC

- Todas las instrucciones del SIC tienen el siguiente formato de 24 bits, donde el bit de bandera x indica el modo de direccionamiento indexado y la D representa el campo de dirección:

8 bits	1 bit	15 bits
Código de operación	x	Dirección D

Ejemplo

La instrucción con nemónico `ADD` tiene código de operación 18 y aplicada a la dirección 2007 tenemos:

```
00011000 x0000111 11010111 = 18 x7 D7
```


Modos de direccionamiento del SIC

- El SIC tiene dos modos de direccionamiento.
- El modo queda indicado por la bandera x de la instrucción y determina la dirección objetivo con la que operará la instrucción.
- La dirección objetivo T se obtiene así:

Modo	Bandera x	Dirección
Directo	$x = 0$	$T = D$
Indexado	$x = 1$	$T = D + X$

Ejemplo de direccionamiento SIC

- Si $x = 0$ tenemos direccionamiento directo. La instrucción `18 07 D7` toma su operando de la dirección $T = D = 07D7$ o 2007 en decimal, es decir, su operando es `mem[2007]`.
- Si $x = 1$ tenemos direccionamiento indexado. La instrucción `18 87 D7` toma su operando de la dirección $T = D+X = 07D7+X$ o $2007+X$ en decimal, es decir, su operando es `mem[2007+X]` que claramente depende del valor actual del registro X.

Tipos de instrucción del SIC

- El SIC proporciona un conjunto básico de instrucciones que contiene los siguientes tipos:
  1. Carga y almacenamiento.
  2. Operaciones aritméticas.
  3. Comparación.
  4. Saltos condicionales.
  5. Subrutinas.
  6. Entrada y salida.

Contenido

Estructura de la máquina SIC/XE

- Esta máquina puede direccionar hasta 2^{20} bytes, lo que provoca un cambio en los modos de direccionamiento (SIC: 2^{15} bytes).
- Además, tiene cuatro registros adicionales:

Registro	Número	Uso
B	3	Registro base
S	4	Sin uso especial
T	5	Sin uso especial
F	6	Punto flotante (48 bits)

Tipos de datos del SIC/XE

- Además de los tipos de datos del SIC, el SIC/XE proporciona un tipo de datos de punto flotante de 48 bits en el siguiente formato:

1 bit	11 bits	36 bits
signo s	exponente e	fracción f

- La fracción f es un número entre 0 y 1; el punto binario está justo antes del primer bit.
- Normalmente el primer bit después del punto binario será 1, pero no es necesario.

Tipo de datos de punto flotante

- El exponente e se interpreta como un número positivo entre 0 y 2047.
- El signo s puede ser 0 para positivo o 1 para negativo.
- El número de punto flotante representado es:

$$x = (-1)^s \cdot f \cdot 2^{e-1024}$$

Ejemplo

El número representado por 01 23 40 00 00 00 está dado por $s = 0$, $e = 18$ y $f = \frac{1}{8} + \frac{1}{16} + \frac{1}{32}$, o sea $(-1)^0 \cdot 0.21875 \cdot 2^{-1006}$.

Formatos de instrucción 1 y 2

- El SIC/XE proporciona cuatro formatos de instrucciones. Dos de estos formatos no hacen referencia a la memoria.
- Formato de 1 byte para modo implícito:

8 bits
Código de operación

- Formato de 2 bytes para registros:

8 bits	4 bits	4 bits
Cód. de operación	Registro 1	Registro 2

Formatos de instrucción 3 y 4

- El formato de 3 bytes es compatible con el SIC:

6 bits	1	1	1	1	1	1	12 bits
Cód. de operación	n	i	x	b	p	e	Dir. D

- El formato de 4 bytes es para direcciones de 20 bits:

6 bits	1	1	1	1	1	1	20 bits
Cód. de operación	n	i	x	b	p	e	Dir. D

Observaciones para el SIC/XE

- En el formato de 2 bytes los campos Registro 1 y Registro 2 corresponden con los números de registro fuente y destino.
- En los formatos de 3 y 4 bytes los códigos de operación son de 6 bits, obtenidos al eliminar los 2 bits más bajos de los códigos del SIC.
- Los bits n, i, x, b, p, e son banderas que (como antes) seleccionan el modo de direccionamiento de la instrucción.

Banderas de direccionamiento

- El bit n indica direccionamiento indirecto.
- El bit i indica direccionamiento inmediato.
- El bit x indica direccionamiento indexado.
- El bit b indica direccionamiento relativo a B.
- El bit p indica direccionamiento relativo a PC.
- El bit e indica direccionamiento extendido.

Direccionamiento relativo a B

- Si b = 1 y p = 0 en el formato 3 se tiene el direccionamiento relativo al registro base.
- En este modo la dirección objetivo se obtiene como $T = D + B$, donde B es el valor actual del registro B y D es el campo de dirección como entero sin signo ($0 \leq D < 2^{12}$).

Ejemplo

La instrucción `1B 47 D7` toma su operando de la dirección T = D+B = `07D7+B` o 2007+B en decimal, es decir, su operando es `mem[2007+B]`.

Direccionamiento relativo a PC

- Si b = 0 y p = 1 en el formato 3 se tiene el direccionamiento relativo al contador de programa.
- En este modo la dirección objetivo se obtiene como $T = D + PC$, donde PC es el valor del contador de programa y D es el campo de dirección con signo ($-2^{11} \leq D < 2^{11}$).

Ejemplo

La instrucción `1B 27 D7` toma su operando de la dirección T = D+PC = `07D7+PC` o 2007+PC en decimal, es decir, su operando es `mem[2007+PC]`.

Observaciones sobre b y p

- Si b = p = 0 en el formato 3 se tiene el direccionamiento directo, es decir T = D.
- En el formato de 4 bytes se debe cumplir siempre que b = p = 0.
- Si además x = 1 entonces al valor obtenido de T se le suma X, por lo que existen varios direccionamientos indexados en los formatos de 3 y 4 bytes.

Direccionamiento inmediato

- Si i = 1 y n = 0 en los formatos 3 y 4 se tiene el direccionamiento inmediato.
- En este modo T se usa como operando, es decir, no se hace referencia a la memoria.

Ejemplo

La instrucción `19 07 D7` toma como operando el valor de T = D = `07D7` o 2007 en decimal, es decir, su operando es 2007.

Direccionamiento indirecto

- Si i = 0 y n = 1 en los formatos 3 y 4 se tiene el direccionamiento indirecto.
- En este modo el dato almacenado en la dirección T se usa como la dirección del operando.

Ejemplo

La instrucción `1A 07 D7` toma su operando de `mem[mem[2007]]`.

Observaciones sobre i e n

- Los modos de direccionamiento inmediato e indirecto no admiten indexación, es decir, x = 0.
- Si i = n entonces T es simplemente la dirección objetivo. En este caso i = n = 1 en el SIC/XE, i = n = 0 en el SIC.
- Si se hace i = n = 0 en el SIC/XE entonces b, p y e se consideran parte del campo de dirección de la instrucción, lo que hace que el formato de 3 bytes sea compatible con el SIC.

Tipos de instrucción del SIC/XE

- El SIC/XE posee todas las instrucciones del SIC, además de otras nuevas de los siguientes tipos:
  1. Manejo de los registros nuevos.
  2. Operaciones aritméticas de punto flotante.
  3. Operaciones entre registros.
  4. Generación de interrupciones de software.
- También puede usar canales de entrada y salida mientras se ejecutan otras instrucciones (DMA).

Part II

Ensambladores

Contenido

Ensambladores y traductores

- Un ensamblador es un programa que traduce un código fuente en lenguaje ensamblador a un código objeto en lenguaje máquina de un cierto procesador.
- De forma más general, un traductor es un programa que traduce un código fuente escrito en un lenguaje fuente a un código objeto escrito en un lenguaje objeto.

Directivas

- Además, un ensamblador suele interpretar algunas directivas como las siguientes:
  1. . indica el inicio de un comentario.
  2. START indica el nombre y (dir. de inicio).
  3. END indica fin y (dir. de primera instrucción).
  4. BYTE genera constantes de bytes.
  5. WORD genera constantes de un word.
  6. RESB reserva el número indicado de bytes.
  7. RESW reserva el número indicado de words.

Ejemplo de código fuente y objeto

```
COPY    START 1000    COPIA EL ARCHIVO DE LA ENTRADA EN LA SALIDA
FIRST   STL   RETADR GUARDA LA DIRECCION DE RETORNO             141033
CLOOP   JSUB  RDREC  LEE EL REGISTRO DE ENTRADA                 482039
        LDA   LENGTH VERIFICA SI ES FIN DE ARCHIVO (LENGTH = 0)  001036
        COMP  ZERO                                              281030
        JEQ   ENDFIL SALE SI SE ENCONTRO EL FIN DE ARCHIVO      301015
        JSUB  WRREC  ESCRIBE EL REGISTRO DE SALIDA              482061
        J     CLOOP  CICLO                                      3C1003
ENDFIL LDA    EOF    INSERTA MARCA DE FIN DE ARCHIVO            00102A
        STA   BUFFER                                            0C1039
        LDA   THREE  HACE LENGTH = 3                            00102D
        STA   LENGTH                                            0C1036
        ...                                                     ...
OUTPUT BYTE   X'05'  CODIGO DEL DISPOSITIVO DE SALIDA           05
        END   FIRST

HCOPY  00100000107A
T0010001E1410334820390010362810303010154820613C100300102A0C103900102D
T00101E150C10364820610810334C0000454F46000003000000
T0020391E041030001030E0205D30203FD8205D2810303020576490392C205E38203F
T0020571C1010364C0000F001010004103 0E0279302064509039DC20792C1036
T0020730738206 44C00000005
E001000
```


Ensamblador simple para el SIC

- Cada renglón del código fuente puede contar con hasta cuatro campos distintos que están separados por espacios: etiqueta, instrucción, operando y comentario.
- La etiqueta también se llama símbolo.
- La instrucción puede ser un nemónico de la máquina o una directiva del ensamblador.
- El operando puede ser una constante o una etiqueta.

Funciones necesarias para traducir

- Convertir las instrucciones nemónicas a sus equivalentes en lenguaje máquina.
- Convertir los operandos símbolicos a sus direcciones de máquina equivalentes.
- Construir o ensamblar las instrucciones de máquina en el formato adecuado.
- Convertir las constantes en el código fuente a su representación interna de máquina.
- Escribir el código objeto (y posiblemente también el listado de ensamblado).

Observaciones

- Todas estas funciones se pueden realizar procesando el código fuente línea por línea, excepto por la segunda.
- Considere por ejemplo la línea:

  `FIRST STL RETADR`
- En el momento que se lee esta línea no se puede procesar, pues se desconoce el valor del operando RETADR.

Más observaciones

- Es por eso que la mayoría de los ensambladores hacen dos pasos o lecturas por el código fuente.
- En el primero se asignan valores a todas las etiquetas y en el segundo se realizan la mayoría de las funciones antes descritas.
- Además, el ensamblador debe de procesar las directivas, las cuales no se traducen en instrucciones, pero pueden influir en el código objeto.

Descripción general de funciones del primer paso

Primer paso: Definición de símbolos

- Asignar direcciones a todas las proposiciones del programa.
- Guardar los valores asignados a todas las etiquetas.
- Procesar (entre otras) las directivas que afecten la asignación de direcciones.

Descripción general de funciones del segundo paso

Segundo paso: Ensamblado y generación de código objeto

- Ensamblar las instrucciones (traducir los nemónicos y examinar los modos de direccionamiento).
- Generar los valores de los datos definidos.
- Procesar las directivas faltantes.
- Escribir el programa objeto y el listado de ensamblado.

Código objeto

- El ensamblador debe de escribir el código objeto generado en algún dispositivo.
- En nuestro caso, el formato del código objeto tiene tres tipos de registros:
  - Registro de encabezado.
  - Registro de texto.
  - Registro de fin.
- Los detalles de estos registros son irrelevantes pero la información que contienen debe estar en alguna parte.

Registro de encabezado

- Posición 1: una `H` (header).
- Posiciones 2 a 7: nombre del programa.
- Posiciones 8 a 13: dirección del inicio del programa (hexadecimal).
- Posiciones 14 a 19: longitud en bytes del programa (hexadecimal).

Ejemplo

```
HCOPY 001000000107A
```


Registro de texto

- Posición 1: una `T` (text).
- Posiciones 2 a 7: dirección de inicio del código objeto en este registro (hexadecimal).
- Posiciones 8 a 9: longitud en bytes del código objeto en este registro (hexadecimal).
- Posiciones 10 a 69: código objeto, dos posiciones por byte (hexadecimal).

Ejemplo

```
T00101E150C10364820610810334C0000454F46000003000000
```


Registro de fin

- Posición 1: una `E` (end).
- Posiciones 2 a 7: dirección de la primera instrucción ejecutable del programa (hexadecimal).

Ejemplo

```
E001000
```


Contenido

Tablas y lógica del ensamblador

- Un ensamblador simple maneja dos tablas internas:
- La tabla de códigos de operación `TABOP` se usa para examinar los nemónicos y traducirlos a lenguaje máquina.
- La tabla de símbolos `TABSIM` se usa para almacenar valores asignados a etiquetas.
- Además se necesita un contador de localidades `CONTLOC` que se utiliza para llevar la cuenta de las direcciones.

El contador de localidades `CONTLOC`

- El valor inicial de `CONTLOC` se toma del indicado por la directiva `START`.
- Después de procesar cada línea del código fuente se le suma a `CONTLOC` la longitud de la instrucción o datos generados.
- Cuando se encuentra una etiqueta en el código fuente, el valor de `CONTLOC` proporciona su valor.
- No se confunda a `CONTLOC` con PC.

La tabla de códigos de operación

- TABOP debe contener los nemónicos y sus códigos de operación equivalentes en lenguaje máquina.
- Además, puede contener información acerca del formato y la longitud de la instrucción.

Uso de TABOP durante el ensamblado

- Durante el primer paso, TABOP se usa para examinar y confirmar los nemónicos en el código fuente (y posiblemente para encontrar la longitud de la instrucción y así poder actualizar CONTLOC adecuadamente).
- Durante el segundo paso, TABOP se usa para traducir los nemónicos a lenguaje máquina (haciendo uso de la información de formato de la instrucción correspondiente).

Organización de TABOP

- TABOP se suele organizar como una tabla de dispersión cuya clave es el nemónico.
- Esta estructura es adecuada porque proporciona una recuperación rápida de datos con un mínimo de búsqueda.
- Aunque esta tabla es en principio estática, se suele utilizar un método general de dispersión.

La tabla de símbolos TABSIM

- La tabla de símbolos TABSIM incluye el nombre y el valor de cada etiqueta del código fuente.
- También contiene banderas que indican condiciones de error (por ejemplo: etiqueta definida en dos lugares).
- Esta tabla puede contener información adicional.

Uso de TABSIM durante el ensamblado

- Durante el primer paso se introducen las etiquetas en TABSIM a medida que se encuentran en el programa fuente junto con sus direcciones asignadas.
- Durante el segundo paso se buscan en la tabla los símbolos empleados como operandos para obtener las direcciones necesarias para ensamblar las instrucciones.

Organización de TABSIM

- TABSIM también suele estar organizada como una tabla de dispersión y, en principio, podría ser la misma que TABOP.
- En este caso, una buena función de dispersión sería considerar a la etiqueta como si estuviera representando a un número entero que se calcularía módulo P, un número primo igual a la longitud de la tabla.

Paso de información (1)

- Es posible que ambos pasos de ensamblado lean el programa fuente como entrada.
- Sin embargo, cierta información puede o debe comunicarse entre los dos pasos (por ejemplo: los valores de CONTLOC o las banderas de error).
- Para esto el primer paso puede escribir un archivo temporal que contiene tanto el código fuente como esta información adicional.

Paso de información (2)

- Este archivo temporal (llamado archivo intermedio) también puede contener los resultados de las búsquedas de símbolos, banderas de direccionamiento, apuntadores a TABOP y TABSIM, etc.
- De esta forma se puede evitar realizar estas operaciones de nuevo en el segundo paso de ensamblado.

Variables del ensamblador

- En los siguientes listados las variables
  - `ETIQUETA`,
  - `CODOP` y
  - `OPERANDO`

  toman respectivamente los valores del campo de
  - etiqueta,
  - instrucción y
  - operando

  de la línea que se acaba de leer.

Algoritmo del ensamblador I

Primer paso, algoritmo general

1. Lee la primera línea de la entrada.
2. Si `CODOP` = `START` haz:
   1. Guarda `OPERANDO` como dirección inicial.
   2. Asigna a `CONTLOC` la dirección inicial.
   3. Lee la siguiente línea de la entrada.
3. Si no, asigna a `CONTLOC` el valor 0.
4. Mientras `CODOP` $\neq$ `END` haz:
   1. Si no es un comentario, procesa la línea.
   2. Lee la siguiente línea de la entrada.
5. Guarda (`CONTLOC` - dirección inicial) como longitud del programa.

Algoritmo del ensamblador II

Primer paso, proceso de línea

1. Si hay un símbolo en el campo `ETIQUETA` haz:
   1. Busca `ETIQUETA` en `TABSIM`.
   2. Si se encuentra, señala un error (símbolo duplicado).
   3. Si no, inserta (`ETIQUETA`, `CONTLOC`) en `TABSIM`.
2. Busca `CODOP` en `TABOP`.
3. Si se encuentra, suma 3 a `CONTLOC`.
4. Si no, si `CODOP` = `WORD`, suma 3 a `CONTLOC`.
5. Si no, si `CODOP` = `RESW`, suma 3*`OPERANDO` a `CONTLOC`.
6. Si no, si `CODOP` = `RESB`, suma `OPERANDO` a `CONTLOC`.
7. Si no, si `CODOP` = `BYTE` haz:
   1. Encuentra la longitud de la constante en bytes.
   2. Suma la longitud a `CONTLOC`.
8. Si no, señala un error (código de operación inválido).

Algoritmo del ensamblador III

Segundo paso, algoritmo general

1. Lee la primera línea de la entrada.
2. Si `CODOP` = `START`, lee la siguiente línea de la entrada.
3. Escribe el registro de encabezamiento en el programa objeto.
4. Asigna valor inicial al primer registro de texto.
5. Mientras `CODOP` $\neq$ `END` haz:
   1. Si no es un comentario, ensambla la línea.
   2. Lee la siguiente línea de la entrada.
6. Escribe el último registro de texto en el programa objeto.
7. Escribe el registro de fin en el programa objeto.

Algoritmo del ensamblador IV

Segundo paso, ensamblado de línea

1. Busca `CODOP` en `TABOP`.
2. Si se encuentra haz:
   1. Si hay un símbolo en el campo `OPERANDO` haz:
      1. Busca `OPERANDO` en `TABSIM`.
      2. Si se encuentra, guarda el valor del símbolo como dirección del operando.
      3. Si no, señala un error (símbolo indefinido).
   2. Si no, almacena 0 como dirección del operando.
   3. Ensambla la instrucción del código objeto.
3. Si no, si `CODOP` = `BYTE` o `WORD` convierte la constante en código objeto.
4. Si el código objeto no cabe en el registro de texto actual haz:
   1. Escribe el registro de texto en el programa objeto.
   2. Asigna valor inicial a un nuevo registro de texto.
5. Añade el código objeto al registro de texto.

Contenido

Direccionamiento del SIC/XE

- En esta sección se analiza un ensamblador para el SIC/XE y los efectos que tienen en el análisis las características extendidas del mismo.
- En particular indicaremos como se representan los diversos modos de direccionamiento en el código fuente.

Ejemplo de código fuente y objeto

```
COPY   START 1000    COPIA EL ARCHIVO DE LA ENTRADA EN LA SALIDA
FIRST  STL   RETADR GUARDA LA DIRECCION DE RETORNO              17202D
       LDB   #LENGTH DEFINE EL REGISTRO DE BASE                 69202D
       BASE  LENGTH
CLOOP +JSUB  RDREC  LEE EL REGISTRO DE ENTRADA                  4B101036
       LDA   LENGTH VERIFICA SI ES FIN DE ARCHIVO (LENGTH = 0)  032026
       COMP #0                                                  290000
       ...                                                      ...
       J     @RETADR VUELVE A QUIEN LO LLAMO                    3E2003
EOF    BYTE  C'EOF'                                            454F46
       ...                                                      ...
OUTPUT BYTE  X'05'  CODIGO DEL DISPOSITIVO DE SALIDA            05
       END   FIRST

HCOPY  000000001077
T0000001D17202D69292D4B10103603202629000033200 74B10105D3F2FEC032010
T00001D130F2016010003 0F200D4B10105D3E2003454F46
T0010361DB410B400B4407 5101000E32019332FFADB2013A00433200857C003B850
T0010531D3B2FEA1340004 F0000F1B410774000E32011332FFA53C003DF2008B850
T0010700 73B2FEF4F000005
M00000705
M00001405
M00002705
E000000
```

Direccionamiento inmediato

- Indicado en el código fuente por #.
- No se hace referencia a la memoria.
- Basta transformar el operando a su representación interna.
- Formato 3: (op[6]) (01) (0bpe) (desp[12]).

Ejemplo

La instrucción `LDA #3` se ensambla a `01 00 03` ya que

(000000)(01)(0000)(000000000011).

Direccionamiento por registros

- Formato 2: (op[8]) (r1[4]) (r2[4]).
- Se pueden colocar los nombres de los registros en la tabla de símbolos.

Ejemplo

La instrucción `COMPR A,S` se ensambla a `A0 04` ya que

(10100000)(0000)(0100).

Direccionamiento relativo

- Dos modos:
  1. Relativo al contador de programa (PC).
  2. Relativo al registro base (B).
- Calcular el desplazamiento correcto.
- Se debe verificar que el desplazamiento calculado se encuentra en el rango adecuado (0 a 4095 para B y -2048 a 2047 para PC).

Direccionamiento relativo al PC (1)

- Durante la ejecución de una instrucción del SIC el contador de programa avanza justo después de obtener la instrucción y justo antes de ejecutarla, por lo que el registro PC ya contiene la dirección de la siguiente instrucción.
- Formato 3: (op[6]) (11) (x010) (desp[12]).

Direccionamiento relativo al PC (2)

Ejemplo

Si `FIRST` vale 0x000 y `RETADR` vale 0x030 entonces la instrucción

```
FIRST STL RETADR
```

se ensambla a `17 20 2D` ya que

(000101)(11)(0010)(000000101101)

y el desplazamiento es

RETADR - (FIRST+3) = 0x02D.

Direccionamiento relativo al PC (3)

Ejemplo

Si `CLOOP` vale 0x006 entonces

```
0x017 J CLOOP
```

se ensambla a `3F 2F EC` ya que

(001111)(11)(0010)(111111101100)

y el desplazamiento es

CLOOP - (0x017+3) = -0x14 = 0xFEC.

Direccionamiento relativo a B (1)

- Ahora la resta se hace con respecto a B.
- El ensamblador debe saber el valor de B.
- Esto se hace con la directiva `BASE`.
- La directiva `NOBASE` indica que B no se usa como registro base.
- El programador es responsable del uso de B.
- Formato 3: (op[6]) (11) (x100) (desp[12]).

Direccionamiento relativo a B (2)

Ejemplo

Si `BUFFER` vale 0x036 y `LENGTH` vale 0x033 entonces la segunda instrucción en

```
BASE LENGTH
STCH BUFFER, X
```

se ensambla a `57 C0 03` ya que

(010101)(11)(1100)(000000000011)

y el desplazamiento es

BUFFER - LENGTH = 0x003.

Direccionamiento relativo a B (3)

Ejemplo

En el mismo caso, la instrucción

```
STX LENGTH
```

se ensambla a 13 40 00 ya que

(000100)(11)(0100)(000000000000)

y el desplazamiento es

```
LENGTH - LENGTH = 0x000.
```


Otros modos de direccionamiento

- Instrucciones con formato extendido (+): no hay problema, no se requiere un cálculo especial para obtener el campo de dirección.
- Direccionamiento indirecto (@): nada nuevo, el desplazamiento se calcula de la forma anterior para producir la dirección deseada.

Contenido

Relocalización de programas

- Si un procesador tiene más memoria entonces es posible tener en ella varios programas simultaneamente.
- Si supieramos de antemano cuáles programas van a estar en la memoria entonces podríamos ensamblarlos con direcciones de inicio adecuadas.
- Sin embargo, casi nunca sabemos.

Necesidad de relocalización

- Se debe tener la posibilidad de cargar cualquier programa en cualquier espacio de memoria disponible.
- Se debe poder ensamblar un programa sin saber en qué dirección se va a cargar.
- La dirección de carga solamente se sabrá en el momento que inicia el proceso de carga.

Ejemplo de mala relocalización

- Si THREE vale 0x102D entonces

  ```
  0x101B LDA THREE
  ```

  se ensambla con direccionamiento directo a

  ```
  0x101B 00 10 2D
  ```

- Si en lugar de cargar desde 0x1000 se carga desde 0x2000 entonces queda

  ```
  0x201B 00 10 2D
  ```

  y tomará un operando equivocado de una dirección que no pertenece al programa.

Consideraciones de relocalización

- Para poder cargar y ejecutar un programa en una dirección distinta a la de ensamblado se necesitará cambiar a veces el campo de dirección de una instrucción.
- Otras veces no será necesario cambiar nada.
- El ensamblador no conoce la dirección de carga y no puede hacer estos cambios, pero puede escribir la información necesaria en el código objeto para que el cargador los haga.

Ejemplos de relocalización

A partir de 0x0000

```
0x0000 ...
0x0006 4B 10 10 36 (+JSUB RDREC)
0x1036 B4 10       (RDREC)
```

A partir de 0x5000

```
0x5000 ...
0x5006 4B 10 60 36 (+JSUB RDREC)
0x6036 B4 10       (RDREC)
```

A partir de 0x7420

```
0x7420 ...
0x7426 4B 10 84 56 (+JSUB RDREC)
0x8456 B4 10       (RDREC)
```

Direccionamiento directo y relocalización

- Cuando el ensamblador genere el código objeto correspondiente a una instrucción con direccionamiento directo deberá usar una dirección objetivo con respecto al inicio del programa (normalmente igual a 0x0000).
- Además, el ensamblador deberá producir una instrucción para el cargador que le indique sumar la dirección real de inicio del programa al campo de dirección de estas instrucciones.

Registros de modificación (1)

- El código objeto podrá contener registros de modificación que constan de tres campos:
- Posición 1: una `M` (modify).
- Posiciones 2 a 7: dirección de inicio del campo de dirección a modificar con respecto al inicio del programa (hexadecimal).
- Posiciones 8 y 9: longitud del campo de dirección (hexadecimal, en medios bytes).

Ejemplo

```
M00000705
```


Registros de modificación (2)

- La dirección de inicio es la del primer byte que contenga al menos una parte del campo de dirección (que podrían ser los bits más altos o los más bajos dependiendo del procesador con el que se esté trabajando).
- La longitud del campo de dirección podría darse en cualquier unidad que tuviera sentido (bits, bytes o palabras).

Registros de modificación (3)

- La instrucción `+JSUB RDREC` se ensambla a `4B 10 10 36` y genera el siguiente código objeto:

```
HCOPY  000000001077
T0000001D17202D69292D4B101036032026...
...                    00000000
M00000705
...
E000000
```


Direccionamiento y relocalización

- Observe que sólo se necesitan modificar las instrucciones con direccionamiento directo.
- En los direccionamientos por registro e inmediato no hay campo de dirección.
- En el direccionamiento relativo los desplazamientos no cambian.
- Por eso, cuando hay opción, se prefiere ensamblar con direccionamiento relativo en lugar de con direccionamiento directo.

Contenido

Características independientes de la máquina

- Hasta ahora hemos estudiado características del ensamblador que dependen fuertemente del procesador empleado.
- Ahora estudiaremos aquellas que sólo dependen de la conveniencia para el programador y el ambiente de software.
  - Literales, símbolos y expresiones.
  - Bloques de programa.
  - Secciones de control y ligado.

Operandos literales

- A veces se quiere utilizar un valor constante como operando sin definir una etiqueta.
- A ese operando se le llama literal puesto que su valor se toma literalmente de la instrucción.
- Usaremos el caracter = para especificar un operando literal.
- Abajo la `C` es caracter y la `X` es hexadecimal.

Ejemplos de literales

```
ENDFIL LDA =C'EOF'   (03 20 10)
WLOOP  TD  =X'05'    (E3 20 11)
```

Operandos literales e inmediatos

Diferencia

Una diferencia entre el uso de un operando literal y el uso de un operando inmediato es que en el segundo el valor constante se ensambla como parte de la instrucción, mientras que con el primero el ensamblador genera el valor especificado como una constante en alguna parte de la memoria y es la dirección de esta constante la que se usa para ensamblar la instrucción: es como si se hubiera usado una etiqueta y el direccionamiento directo o relativo.

Ejemplo de uso de literales

```
0x001A 03 20 10 (LDA =C'EOF')
...             (LTORG)
0x002D 45 4F 46 (=C'EOF')
...
0x1062 E3 20 11 (TD  =X'05')
...
0x106B DF 20 08 (WD  =X'05')
...
0x1076 05       (=X'05')
```


Depósitos de literales

- Los literales se suelen almacenar en depósitos de literales en el código objeto.
- La directiva `LTORG` instruye al ensamblador a crear un depósito que incluya a todos los literales no almacenados hasta el momento.
- Si no hay ninguna directiva `LTORG` entonces los literales se colocan al final del programa.
- La directiva `LTORG` permite que los literales estén cerca del lugar donde se usan y se pueda usar el direccionamiento relativo.

Literales repetidos

- Los literales repetidos se pueden reconocer para no reservar más de un espacio de memoria para la misma constante. ¿Cómo?
- Comparando las cadenas que los definen.
- Se ahorra más espacio (pero se pierde tiempo) si además se revisa si cadenas distintas definen el mismo literal.
- Se debe tener cuidado con los literales que cambian de valor a lo largo del programa (el literal `=*` se refiere al contador de programa).

Tabla de literales

- Los literales se organizan en una tabla de dispersión `TABLIT` que contiene:
  - El nombre o cadena que define al literal.
  - El valor o representación interna del literal.
  - La longitud del literal en bytes.
  - La dirección donde se almacenará.
- La función de dispersión puede usar como clave el nombre del literal o su valor.

Literales durante el primer paso

- Se reconoce un operando literal.
- Se busca en `TABLIT`.
- Si no está entonces se agrega a `TABLIT`.
- Si se encuentra un `LTORG` se asigna una dirección a cada literal en `TABLIT` y se suma a `CONTLOC` la longitud total de estos literales.

Literales durante el segundo paso

- Se genera código objeto correspondiente a los literales que aparecen en `TABLIT`.
- Esto se hace de la misma forma que con las directivas `BYTE` o `WORD`.
- En el caso de un literal que represente al contador de programa (*) se debe de generar el registro de modificación correspondiente.

Contenido

Definición de símbolos

- Muchos ensambladores tienen una directiva para definir símbolos y especificar su valor:

```
símbolo EQU valor (equate)
```

- Estos símbolos tienen dos usos comunes:
  - Mejorar la legibilidad de un programa.
  - Definición de nemónicos para registros.
- La directiva ORG instruye al ensamblador que cambie el valor del contador de localidades:

```
ORG valor (origen)
```


Ejemplos de uso de EQU y ORG

- Las definiciones del lado izquierdo tienen el mismo efecto que las del lado derecho.
- Pero las del lado derecho son más fáciles de interpretar que las del lado izquierdo.

Con EQU

```
TABS      RESB 1100
SIMBOLO   EQU TABS
VALOR     EQU TABS+6
BANDERAS  EQU TABS+9
```

Con ORG

```
TABS      RESB 1100
          ORG TABS
SIMBOLO   RESB 6
VALOR     RESW 1
BANDERAS  RESB 2
          ORG TABS+1100
```


Restricciones para EQU y ORG

- Una restricción importante para las directivas EQU y ORG es que todos los símbolos usados del lado derecho tienen que haber sido definidos con anterioridad.
- Esto se debe a que estamos usando un ensamblador de dos pasos.

Ejemplo de buen uso

```
ALFA RESW 1
BETA EQU ALFA
```

Ejemplo de mal uso

```
BETA EQU ALFA
ALFA RESW 1
```


Contenido

Expresiones

- Los ensambladores suelen permitir el uso de operaciones simples para obtener operandos:
  - Operaciones aritméticas (+, -, *).
  - Operaciones lógicas (AND, OR, ...).
  - Operaciones de bits (AND, OR, ...).
- Los operandos también deben ser simples:
  - Constantes.
  - Símbolos.
  - Etiquetas.
  - Especiales (como * para CONTLOC).

Tipos de operandos y expresiones

- Existen dos tipos de operandos.
  - Absolutos: aquellos cuyo valor no depende de donde comience el programa.
  - Relativos: aquellos cuyo valor sí depende de donde comience el programa.
- Y también dos tipos de expresiones.
  - Absolutas: aquellas cuyo valor no depende de donde comience el programa.
  - Relativas: aquellas cuyo valor sí depende de donde comience el programa.

Ejemplos de expresiones

- Sean A1 y A2 dos operandos absolutos y sean R1 y R2 dos operandos relativos.
- Ejemplos de expresiones absolutas son:

```
10-A1, A1+A2, R1-R2
```

- Ejemplos de expresiones relativas son:

```
10+R1, A1+R2, R1-A2
```

- Ejemplos de expresiones ilegales son:

```
10-R1, R1+R2, A1-R2
```


Más sobre expresiones

- En las expresiones absolutas los operandos relativos aparecen restándose a pares.
- En las expresiones relativas los operandos relativos también aparecen restándose a pares, pero siempre sobra uno sumando.
- Cualquier otra expresión es ilegal.
- TABSIM debe contener una bandera para indicar si un símbolo es absoluto o relativo.

Contenido

Bloques de programa

- Los ensambladores analizados hasta ahora generan su código objeto en el mismo orden en el que aparece el código fuente.
- Algunos ensambladores permiten separar el código fuente en bloques de programa que luego serán reordenados.
- Por ejemplo, podría haber bloques de código, de datos, de literales, de tablas, etc.

Directiva USE

- El parámetro de la directiva USE le indica al ensamblador qué porciones del código fuente pertenecen a qué bloque.
- Normalmente existe un bloque sin nombre. Una porción del código fuente pertenece a este bloque si no está precedida por una directiva USE o si la directiva USE correspondiente no tiene parámetro.

Ejemplo de la directiva USE

```
COPY START 0
     ...          (bloque sin nombre (1) 0x0000 a 0x0026)
     USE   CDATA
     ...          (bloque CDATA (1)      0x0000 a 0x0005)
     USE   CBLKS
     ...          (bloque CBLKS (1)      0x0000 a 0x1000)
     USE
     ...          (bloque sin nombre (2) 0x0027 a 0x004C)
     USE   CDATA
     ...          (bloque CDATA (2)      0x0006 a 0x0006)
     USE
     ...          (bloque sin nombre (3) 0x004D a 0x0065)
     USE   CDATA
     ...          (bloque CDATA (3)      0x0007 a 0x000A)
     END
```


Reordenamiento

- Cada bloque puede estar formado por varios segmentos de código fuente.
- Los segmentos serán reordenados lógicamente por el ensamblador para reunir de forma consecutiva todos los segmentos que pertenecen al mismo bloque.
- Esto es equivalente a si el programador hubiera reordenado a mano el código fuente antes de ensamblarlo.

Un CONTLOC por bloque

- Una forma de poder realizar el reordenamiento es llevar un contador de localidades independiente por cada bloque.
- Cada uno de ellos debe comenzar con el valor 0 y se debe poder almacenar en algún lado cuando se esté procesando otro bloque.

Bloques y primer paso de ensamblado

- Durante el primer paso de ensamblado se asigna a cada etiqueta un valor relativo al inicio del bloque donde se define, por lo que TABSIM deberá contener un indicador de en qué bloque fue definida cada etiqueta.
- Al terminar el primer paso cada uno de los contadores de localidades tiene un valor igual a la longitud del bloque correspondiente.
- En ese momento se le puede asignar a cada bloque una dirección de inicio relativa al inicio del primer bloque (o inicio del programa).

Tabla de bloques

- Toda esta información se puede almacenar en una tabla de bloques como la siguiente:

Nombre	Número	Inicio	Longitud
sin nombre	0	0x0000	0x0066
CDATA	1	0x0066	0x000B
CBLKS	2	0x0071	0x1000

Bloques y segundo paso de ensamblado

- Durante el segundo paso de ensamblado se obtiene el valor de cada símbolo relativo al inicio del programa sumando el valor del símbolo relativo al inicio al bloque que lo contiene con la dirección de inicio de ese bloque relativa al inicio del programa.

Ejemplo

Si `LENGTH` se definió en el bloque `CDATA` con dirección 0x0003 relativa al inicio de `CDATA`, entonces la dirección de `LENGTH` relativa al inicio del programa es

0x0066+0x0003=0x0069

Observaciones sobre bloques

- Colocar datos grandes en un bloque al fin del programa reduce la necesidad de las instrucciones de formato extendido y permite el direccionamiento relativo. Las literales están en el mismo bloque que `LTORG`.
- Los bloques mejoran la legibilidad del programa y evitan el uso de instrucciones de salto para brincar áreas de datos.
- El ensamblador no reordena el código objeto. Sólo asegura que cada registro de texto contiene la dirección de inicio adecuada.

Continuación del ejemplo

```
...         (bloque sin nombre (1) 0x0000 a 0x0026)
...         (bloque sin nombre (2) 0x0027 a 0x004C)
...         (bloque sin nombre (3) 0x004D a 0x0065)
...         (bloque CDATA (1)      0x0000 a 0x0005)
...         (bloque CDATA (2)      0x0006 a 0x0006)
...         (bloque CDATA (3)      0x0007 a 0x000A)
...         (bloque CBLKS (1)      0x0000 a 0x1000)

HCOPY  000000001071
T0000001E1720634B20210320602900003320064B203B3F2FEE0320550F2056010003
T00001E090F20484B20293E203F
T0000271DB400B44075101000E32038332FFADB2032A00433200857A02FB850
T000044093B2FEA13201F4F0000
T00006C01F1
T00004D19B410772017E3201B332FFA53A016DF2012B8503BEFEF4F0000
T00006D04454F4605
E000000
```


Contenido

Secciones de control y ligado

- Una sección de control es una parte de un programa que puede cargarse y relocalizarse independientemente del resto.
- Ejemplo: Una subrutina.
- Una sección de control (o subrutina) puede hacer referencia (llamar) a otra (subrutina). A esto se le llama referencia externa.
- El ensamblador no sabe dónde se cargaran las secciones de control. Un método para manejar las referencias externas es el ligado.

Directiva CSECT

- Las directivas `START` y `CSECT` señalan el inicio de una sección de control.

Ejemplo

```
COPY   START 0
       ...        (seccion de control COPY)
RDREC CSECT
       ...        (seccion de control RDREC)
WRREC CSECT
       ...        (seccion de control WRREC)
       END FIRST
```


Directivas EXTDEF y EXTREF

- A diferencia de los bloques, el ensamblador trata a las secciones de control por separado.
- En particular, las secciones de control se pueden ensamblar por separado.
- Esto obliga a señalar los símbolos externos (definidos en una sección y usados en otra).
- Con `EXTDEF` en la sección que los define.
- Con `EXTREF` en la sección que los usa.
- Los nombres de sección son externos.

Ejemplo de EXTDEF y EXTREF

```
COPY   START  0
       EXTDEF BUFFER,BUFEND,LENGTH
       EXTREF RDREC,WRREC
       ...        (seccion de control COPY)
RDREC CSECT
       EXTREF BUFFER,LENGTH,BUFEND
       ...        (seccion de control RDREC)
WRREC CSECT
       EXTREF LENGTH,BUFFER
       ...        (seccion de control WRREC)
       END FIRST
```

Instrucciones con referencias externas

- Cuando se ensambla una instrucción que hace una referencia a un símbolo externo se desconoce el valor de este símbolo.
- El ensamblador colocará el valor cero y un registro de modificación en el código objeto.
- Si la referencia es una dirección no se podrá usar el direccionamiento relativo. ¿Porqué?
- Todas las demás instrucciones se ensamblan de la forma usual.

TABSIM y símbolos externos

- TABSIM debe almacenar la sección en la que se define cada símbolo.
- Se considera un error hacer una referencia a un símbolo de otra sección que no esté marcado como externo (tanto con EXTDEF como con EXTREF).
- Por otro lado, se puede definir el mismo símbolo en dos o más secciones siempre y cuando no esté marcado como externo.

Nuevos tipos de registros

- Para poder indicar en el código objeto la información necesaria acerca de los símbolos externos y las modificaciones correspondientes se necesitan los siguientes tipos de registros:
  - Registro de definición.
  - Registro de referencia.
  - Registro de modificación (revisado).

Registro de definición

- Posición 1: una D (define).
- Posiciones 2 a 7: símbolo externo definido en esta sección de control.
- Posiciones 8 a 13: dirección relativa del símbolo externo dentro de esta sección de control (hexadecimal).
- Posiciones 14 a 73: la misma información para otros símbolos externos.

Ejemplo

```
DBUFFER000033BUFEND001033LENGTH00002D
```


Registro de referencia

- Posición 1: una R (reference).
- Posiciones 2 a 7: símbolo externo al que se hace referencia en esta sección de control.
- Posiciones 8 a 73: la misma información para otros símbolos externos.

Ejemplo

```
RRDREC WRREC
```


Registro de modificación (revisado)

- Posición 1: una M (modify).
- Posiciones 2 a 7: dirección de inicio del campo a modificar relativa al inicio de la sección de control (hexadecimal).
- Posiciones 8 y 9: longitud del campo de dirección (hexadecimal, en medios bytes).
- Posición 10: bandera de modificación (+ ó −).
- Posiciones 11 a 16: símbolo externo que se sumará o restará al campo indicado.

Ejemplo

```
M00000305+LENGTH
```


Código objeto de secciones de control

```
HCOPY  000000001033
DBUFFER000033BUFEND001033LENGTH00002D
RRDREC WRREC
...

HWRREC 00000000001C
RLENGTHBUFFER
T0000000CB41077100000E32012332FFA53900000DF2008B8503B2FEE4F000005
M00000305+LENGTH
M00000D05+BUFFER
```


Uso de registros de modificación

- Los registros de modificación revisados se pueden usar para relocalizar un programa.
- En lugar de sumar la dirección de inicio del programa se suma la dirección de inicio de la sección correspondiente.

Ejemplo

El registro de modificación

```
M00000705
```

se cambia por el registro revisado

```
M00000705+COPY
```

Nuevas expresiones válidas

- No basta que los valores relativos aparezcan restados a pares, sino que estos pares deben de pertenecer a la misma sección de control.
- Si los operandos son símbolos externos entonces el ensamblador no podrá determinar si una expresión es válida o no.
- El ensamblador evalúa los términos que pueda y entrega un valor inicial junto con los registros de modificación correspondientes, dejando que el cargador termine el trabajo incluyendo la detección de errores.

Contenido

Dos pasos con superposición

- Sólo algunos datos y subrutinas (relativos a `TABSIM`) se usan en los dos pasos.
- El ensamblador se puede separar en tres segmentos: un segmento raíz que contiene lo compartido y dos segmentos hijos que contienen lo que sólo se usa en un paso.
- Solamente dos estarán a la vez en memoria: el segmento raíz y un segmento hijo.
- Esto reduce el requerimiento de memoria durante la ejecución del ensamblador.

Contenido

Ensambladores de un paso

- El problema de ensamblar un programa en un paso son las referencias hacia adelante, es decir, a símbolos que no se han definido.
- Las referencias a datos hacia adelante se pueden eliminar colocándolos al inicio.
- Las referencias a código hacia adelante son incómodas o imposibles de eliminar.
- Hay al menos dos formas distintas de resolver este problema en un paso.

Carga y ejecución (1)

- Los ensambladores de carga y ejecución no generan código objeto sino que escriben el código directamente en la memoria para su ejecución inmediata.
- Si el operando de una instrucción es un símbolo no definido, se agrega a `TABSIM` y se marca como no definido.
- La dirección del campo del operando de la instrucción se agrega a una lista de referencias hacia adelante asociada a su entrada en `TABSIM`.

Carga y ejecución (2)

- Cuando se encuentra la definición de un símbolo se revisa su lista de referencias y se inserta su valor en los lugares indicados.
- Al llegar al final del código fuente, cualquier entrada de `TABSIM` que no esté definida se debe señalar como error.
- Si no hubo errores, el ensamblador brinca a la primera instrucción del programa.
- Nota: Se debe conocer la dirección de inicio del programa antes del ensamblado.

Código objeto en un paso (1)

- Los ensambladores de un paso que generan código objeto se necesitan cuando no se tiene un dispositivo externo para almacenar el código intermedio.
- Cuando se descubre una referencia hacia adelante se almacena en una lista.
- Cuando se encuentra la definición de un símbolo puede ser que ya no se tengan disponibles las instrucciones que hacían referencia a este símbolo (por ejemplo, si ya se escribió el registro de texto).

Código objeto en un paso (2)

- En este caso, el ensamblador debe generar otro registro de texto con la dirección y valor correcto del operando.
- De esta forma el cargador inserta el valor correcto en la dirección adecuada.
- Esto implica que los registros de texto deben cargarse en el mismo orden en el que se generan. ¿Porqué?

Contenido

Ensambladores de varios pasos (1)

- Recordemos que se pidió que los lados derechos de las directivas `EQU` y `ORG` estuvieran definidas previamente.
- Una lista de definiciones que no se puede resolver con un ensamblador de dos pasos:

```
ALFA  EQU BETA+1
BETA  EQU DELTA+1
DELTA RESW 1
```

- ¿Cómo podemos permitir estas definiciones?

Ensambladores de varios pasos (2)

- Una posibilidad es hacer tantos pasos por el código fuente como sean necesarios para resolver todas las referencias.
- Sin embargo, basta que se hagan varios pasos sólo sobre las partes involucradas (que se identifican en el primer paso) y que se resuelven antes del segundo paso.
- Una forma de hacerlo es almacenando en `TABSIM` las definiciones que usen referencias hacia adelante y las dependencias de valor entre símbolos.

Ejemplo de varios pasos

```
ALFA  EQU BETA+1
BETA  EQU DELTA+1
DELTA RESW 1      (CONTLOC vale 0x1000 aqui)
```

1. `ALFA` depende de `BETA`.
2. `BETA` depende de `DELTA`.
3. `DELTA = 0x1000`. ¿Se puede hacer algo?
4. Sí: `BETA = 0x1001`. ¿Se puede hacer algo?
5. Sí: `ALFA = 0x1002`. ¿Se puede hacer algo?
6. No, se conocen todos los valores (3 pasos).

Part III

Cargadores y ligadores

Contenido

Cargadores, ligadores y traductores

- Un cargador es un programa de sistema que realiza la función de llevar un programa objeto a la memoria para su ejecución.
- Un ligador es un programa de sistema que combina dos o más programas objeto y proporciona la información necesaria para resolver las referencias entre ellos.
- Casi todos los traductores del mismo sistema generan código objeto en el mismo formato, así se usa el mismo cargador y ligador independientemente del lenguaje original.

Contenido

Algoritmo para un cargador básico

- Un cargador básico puede cargar programas objeto en direcciones absolutas.

1. Lee el registro de encabezado.
2. Verifica el nombre del programa y longitud.
3. Mientras lea un registro de texto lo convierte a su representación interna y lo copia a la localidad de memoria especificada.
4. Salta a la dirección especificada en el registro de fin.

Consideraciones prácticas

- En la práctica el código objeto se puede guardar en binario (y no en hexadecimal y ASCII como con el SIC u otros).
- En este caso, cada byte en el código objeto corresponde con un byte en la memoria (y no dos a uno, como con el SIC u otros).
- Además, los formatos de registros deben cambiar puesto que cualquier byte puede formar parte de un registro (normalmente, los registros deberán indicar claramente su longitud en un encabezado).

Contenido

Limitaciones del cargador absoluto

- El cargador absoluto tiene la desventaja de que la dirección real de carga debe ser especificada en el código fuente.
- Esto no es un gran problema en una máquina con poca memoria, pero es completamente inadecuado en los demás casos.
- Además, se dificulta el uso de subrutinas de biblioteca ya que no se podrían relocalizar.
- Ahora estudiaremos el diseño de un cargador que también haga relocalización y ligado.

Relocalización

- Un cargador que permite la relocalización de programas es un cargador relocalizador, también conocido como cargador relativo.
- Hay dos métodos comunes para representar la relocalización del código objeto.
- El primero es el ya estudiado, basado en registros de modificación.
- Este método es útil cuando la mayoría de las instrucciones usan direccionamiento relativo o inmediato. ¿Porqué?

Problema de los registros

- Sin embargo, cuando la mayoría de las instrucciones usan direccionamiento directo se necesitarían casi tantos registros de modificación como instrucciones.
- En este caso se requiere otro mecanismo más eficiente en el uso de espacio en el código objeto para especificar la relocalización.

Contenido

Máscaras de bits

- Alteremos el formato del registro de texto para que contenga un bit de relocalización asociado a cada instrucción (o byte, etc.).
- Estos bits se agrupan en una máscara de bits después del indicador de longitud del registro.
- La longitud de la máscara depende de la longitud máxima de un registro de texto.
- Si el bit de relocalización vale 1 se le suma al parámetro de la instrucción la dirección de inicio del programa (si vale 0, nada).

Ejemplo de máscara de bits

Programa relocalizable para el SIC

```
H COPY    000000 00107A
T 000000 1E FFC 140033 481039 000036 280030 300015
                481061 3C0003 00002A 0C0039 00002D
T 00001E 15 E00 0C0036 481061 080033 4C0000 454F46
                000003 000000
T 001039 1E FFC 040030 000030 E0105D 30103F D8105D
                280030 301057 548039 2C105E 38103F
T 001057 0A 800 100036 4C0000 F1     001000
T 001061 19 FE0 040030 E01079 301064 508039 DC1079
                2C0036 381064 4C0000 05
E 000000
```


Contenido

Ligado de programas

- Cuando un programa tiene más de una sección de control, éstas se pueden ensamblar juntas o por separado.
- Las secciones de control aparecerán como segmentos separados de código objeto.
- El cargador no puede (ni necesita) saber cuáles secciones de control se ensamblaron al mismo tiempo.

Ejemplo de ligado y relocalización (1)

```
0000 PROGA START  0
           EXTDEF LISTA,ENDA
           EXTREF LISTB,ENDB,LISTC,ENDC
           ...
0020 REF1  LDA    LISTA                   03201D
0023 REF2 +LDT    LISTB+4                 77100004
0027 REF3  LDX    #ENDA-LISTA             050014
           ...
0040 LISTA EQU    *
           ...
0054 ENDA  EQU    *
0054 REF4  WORD   ENDA-LISTA+LISTC        000014
0057 REF5  WORD   ENDC-LISTC-10           FFFFF6
005A REF6  WORD   ENDC-LISTC+LISTA-1      00003F
005D REF7  WORD   ENDA-LISTA-(ENDB-LISTB) 000014
0060 REF8  WORD   LISTB-LISTA             FFFFC0
           END    REF1
```


Explicación de cálculos (1)

- REF1: el operando LISTA se puede calcular completamente y vale 1D.
- REF2: el operando LISTB+4 se preevalúa a 4 y necesita un registro de modificación M00002405+LISTB.
- REF3: el operando ENDA-LISTA se puede calcular completamente y vale 14.
- REF4: el operando ENDA-LISTA+LISTC se preevalúa a 14 y necesita un registro de modificación M00005406+LISTC.

Ejemplo de ligado y relocalización (2)

```
0000 PROGB START  0
           EXTDEF LISTB,ENDB
           EXTREF LISTA,ENDA,LISTC,ENDC
           ...
0036 REF1 +LDA    LISTA                   03100000
003A REF2  LDT    LISTB+4                 772027
003D REF3 +LDX    #ENDA-LISTA             05100000
           ...
0060 LISTB EQU    *
           ...
0070 ENDB  EQU    *
0070 REF4  WORD   ENDA-LISTA+LISTC        000000
0073 REF5  WORD   ENDC-LISTC-10           FFFFF6
0076 REF6  WORD   ENDC-LISTC+LISTA-1      FFFFFF
0079 REF7  WORD   ENDA-LISTA-(ENDB-LISTB) FFFFF0
007C REF8  WORD   LISTB-LISTA             000060
           END
```


Explicación de cálculos (2)

- REF1: el operando LISTA se preevalúa a 0 y necesita un registro de modificación M00003705+LISTA.
- REF2: el operando LISTB+4 se puede calcular completamente y vale 27.
- REF3: el operando ENDA-LISTA se preevalúa a 0 y necesita dos registros de modificación M00003E05+ENDA y M00003E05-LISTA.
- REF4: el operando ENDA-LISTA+LISTC se preevalúa a 0 y necesita tres registros de modificación.

Ejemplo de ligado y relocalización (3)

```
0000 PROGC START  0
           EXTDEF LISTC,ENDC
           EXTREF LISTA,ENDA,LISTB,ENDB
           ...
0018 REF1 +LDA    LISTA                   03100000
001C REF2 +LDT    LISTB+4                 77100004
0020 REF3 +LDX   #ENDA-LISTA              05100000
           ...
0030 LISTC EQU    *
           ...
0042 ENDC  EQU    *
0042 REF4  WORD   ENDA-LISTA+LISTC        000030
0045 REF5  WORD   ENDC-LISTC-10           000008
0048 REF6  WORD   ENDC-LISTC+LISTA-1      000011
004B REF7  WORD   ENDA-LISTA-(ENDB-LISTB) 000000
004E REF8  WORD   LISTB-LISTA             000000
           END
```


Explicación de cálculos (3)

- REF1: el operando LISTA se preevalúa a 0 y necesita un registro de modificación M00001905+LISTA.
- REF2: el operando LISTB+4 se preevalúa a 4 y necesita un registro de modificación M00001D05+LISTB.
- REF3: el operando ENDA-LISTA se preevalúa a 0 y necesita dos registros de modificación M00002105+ENDA y M00002105-LISTA.
- REF4: el operando ENDA-LISTA+LISTC se preevalúa a 30 y necesita dos registros de modificación.

Ejemplo de ligado y relocalización (4)

- Si PROGA se carga a partir de la dirección 0x4000 entonces PROGB se carga a partir de la dirección 0x4063 y PROGC se carga a partir de la dirección 0x40E2.
- Para terminar de calcular el operando de REF4 en PROGA se deben de sumar:
  - el valor 0x14 ya precalculado,
  - el valor 0x30 de LISTC relativo a PROGC y
  - la dirección de inicio 0x40E2 de PROGC
- para obtener al final el valor 0x4126.

Contenido

Algoritmo del cargador ligador

- Ya podemos presentar el algoritmo de un cargador ligador con relocalización que utiliza registros de modificación.
- La entrada consta de un conjunto de secciones de control a cargar y ligar.
- Una sección de control puede hacer referencias externas a símbolos que se definirán más adelante, por lo que a veces no se puede realizar de inmediato el ligado.
- Por eso, un cargador ligador suele hacer dos pasos: en el primero asigna direcciones a todos los símbolos externos y en el segundo realiza la carga, relocalización y ligado.

Tablas y variables del cargador ligador

- Se necesita una tabla de símbolos externos TABSE que almacena para cada símbolo externo su nombre, dirección y sección de control en donde se define (dispersión).
- Una variable DIRPROG que contiene la dirección de carga del programa ligado.
- Una variable DIRSC que contiene la dirección de carga de la sección de control actual.
- Este valor se suma a las direcciones relativas para volverlas direcciones absolutas.

Paso uno del cargador ligador

- El cargador sólo se ocupa de procesar los registros de encabezado y los registros de definición de las secciones de control.
- Normalmente, el valor de DIRPROG se toma del sistema operativo.
- Al final del paso uno, TABSE contiene todos los símbolos externos definidos en el conjunto de secciones de control junto con las direcciones asignadas a cada uno de ellos.
- Esto también se llama mapa de carga.

Paso dos del cargador ligador

- En este paso se realizan las operaciones de carga, relocalización y ligado del programa.
- Al final del paso dos, el cargador suele transferir el control al programa cargado (posiblemente indicándole al sistema operativo adonde transferir el control).

Contenido

Búsqueda automática en biblioteca

- Los cargadores suelen poder incluir de forma automática rutinas de bibliotecas (estándares o especificadas por el usuario).
- Los programadores las pueden usar como si fueran parte del lenguaje de programación.
- Las rutinas usadas se leen automáticamente de la biblioteca, se ligan y se cargan.
- El programador sólo necesita mencionar el nombre de las rutinas. A esto se le llama búsqueda automática en biblioteca.

Símbolos externos y bibliotecas (1)

- Durante el primer paso, los símbolos externos referenciados en las bibliotecas se tratan igual que antes.
- Al final del primer paso, los símbolos que aún estén sin definir se buscan primero en las bibliotecas especificadas y luego en las bibliotecas estándares.
- Se continúa procesando las subrutinas encontradas como si fueran parte de la entrada.

Símbolos externos y bibliotecas (2)

- Este proceso se repite con cualquier referencia externa que se encuentre en las bibliotecas. Si al final de este proceso aún quedan referencias sin resolver, éstas deben señalarse como errores.
- Esto permite al programador usar sus propias versiones de algunas rutinas cuyos nombres aparezcan en las bibliotecas, pues al final de la primera etapa del primer paso los símbolos correspondientes ya estarán definidos.

Directorio de subrutinas

- Las bibliotecas suelen estar en un archivo de código objeto. Para buscar una rutina se podría examinar todo el código objeto, pero esto es muy ineficiente.
- El sistema podría tener un directorio de subrutinas que contiene sus nombres y una referencia a su posición en el código objeto.
- La búsqueda en la biblioteca es entonces una búsqueda en este directorio seguida de la lectura de la parte indicada del código objeto.

Contenido

Opciones del cargador

- Muchos cargadores permiten especificar opciones que modifican el comportamiento estándar del cargador.
- Esas opciones se pueden pasar a través de un lenguaje especial, ya sea en un archivo independiente o como parte del flujo de entrada (incluso como parte del código fuente en ensamblador).
- A continuación describimos cinco opciones comunes.

Opciones `INCLUDE`, `DELETE` y `CHANGE`

`INCLUDE nombre`

Esta opción le indica al cargador que lea el programa objeto llamado `nombre` (posiblemente una biblioteca) y que lo trate como parte de la entrada.

`DELETE nombre`

Esta opción le indica al cargador que elimine la sección de control llamada `nombre` del conjunto de programas que se están cargando.

`CHANGE fuente destino`

Esta opción le indica al cargador que cambie el símbolo `fuente` por el símbolo `destino` cada vez que aparezca en el código objeto.

Opciones LIBRARY y NOCALL

LIBRARY nombre

Esta opción le indica al cargador que use la biblioteca no estándar llamada `nombre` para resolver las referencias externas en lugar de la biblioteca estándar.

NOCALL nombre

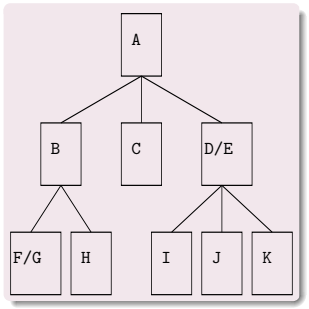
Esta opción le indica al cargador que la rutina llamada `nombre` no se usará en esta ejecución del programa, por lo que no es necesario cargarla.

Contenido

Programas con superposiciones

- El método de superposiciones sirve para reducir la cantidad de memoria requerida durante la ejecución de un programa.
- Muchos sistemas de administración de superposiciones requieren que éstas tengan una estructura de árbol.
- Cada nodo del árbol se llama segmento.
- Cada segmento puede constar de una o más secciones de control.

Ejemplo de un árbol de segmentos



Sección	y longitud
A	1000
B	1800
C	4000
D	2800
E	800
F	1000
G	400
H	800
I	1000
J	800
K	2000

Carga de segmentos superpuestos

- El árbol tiene un segmento raíz que se carga en la memoria al iniciar el programa. En el ejemplo se carga el segmento raíz `A` que mide `0x1000` bytes.
- Los demás segmentos se cargan en la memoria a medida que se llaman.
- Cada segmento puede llamar a sus hijos como rutinas, pero nunca a dos al mismo tiempo. En el ejemplo los hijos del segmento `A` son los segmentos `B`, `C` y `D/E`.

Segmentos activos

- Un segmento presente en la memoria se llama segmento activo.
- En un árbol de superposiciones, si en algún momento se está usando un cierto segmento entonces también estarán activos todos los segmentos en su trayectoria hacia la raíz.
- Esto significa que si un segmento llama a otro más abajo en el árbol puede ser necesario cargarlo, pero si llama a otro más arriba en el árbol entonces no se necesita hacer nada.

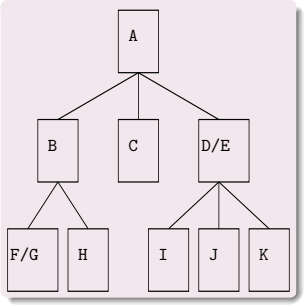
Superposición de segmentos

- Como los segmentos en el mismo nivel del árbol sólo pueden llamarse desde un nivel superior, entonces no pueden requerirse al mismo tiempo.
- Por lo tanto, si se carga un segmento debido a una transferencia de control, éste se puede superponer a cualquier segmento del mismo nivel o inferior que esté en la memoria.
- Así se puede ejecutar todo el programa usando poca memoria.

Definición de segmentos

- La estructura del árbol de superposiciones se define usando instrucciones para el cargador.
- La instrucción `SEGMENT nombre(lista)` define un segmento llamado `nombre` y la `lista` de secciones de control que lo forman.
- El primer segmento definido será la raíz.
- Cada segmento que se defina posteriormente tendrá como padre al último segmento definido. Este comportamiento se puede cambiar con la instrucción `PARENT padre`.

Ejemplo de definición de segmentos



Segmentos

```
SEGMENT SEG1(A)
SEGMENT SEG2(B)
SEGMENT SEG3(F,G)
PARENT  SEG2
SEGMENT SEG4(H)
PARENT  SEG1
SEGMENT SEG5(C)
PARENT  SEG1
SEGMENT SEG6(D,E)
SEGMENT SEG7(I)
PARENT  SEG6
SEGMENT SEG8(J)
PARENT  SEG6
SEGMENT SEG9(K)
```


Dirección inicial de segmentos

- Ya definida la estructura de superposiciones es muy fácil encontrar la dirección inicial de los segmentos.
- Cada uno empieza inmediatamente después del final de su padre.

Segmento	Secciones	Inicio	Longitud	Final+1
1	A	0000	1000	1000
2	B	1000	1800	2800
3	F/G	2800	1400	3C00
4	H	2800	0800	3000
5	C	1000	4000	5000
6	D/E	1000	3000	4000
7	I	4000	1000	5000
8	J	4000	0800	4800
9	K	4000	2000	6000

Segmentos ausentes

- Como el cargador puede asignar las direcciones iniciales de todas las secciones de control entonces se puede relocalizar y ligar de la forma usual con una excepción:
- Se debe permitir la posibilidad de que al llamar a un segmento éste no se encuentre en la memoria.
- Esto se puede hacer de formas distintas. Sólo veremos una forma sencilla.

Archivo de segmentos

- Se realiza la relocalización y el ligado pero en lugar de colocar el resultado en la memoria se guarda en un archivo `ARCHSEG`.
- El segmento raíz siempre estará en la memoria. Se le agrega de forma automática una sección de control especial llamada manejador de superposiciones (`MANSUP`).
- `MANSUP` debe tener información sobre la estructura de superposiciones del programa.

Tabla de segmentos

- Esta estructura se almacena en una tabla de segmentos (`TABSEG`) creada por el cargador y también agregada de forma automática como sección de control al segmento raíz.
- `TABSEG` contiene la dirección de carga del segmento, la dirección de su punto de entrada (suponemos que sólo tiene uno) y su localización en `ARCHSEG`.

Área de transferencia

- `TABSEG` también incluye un área de transferencia para cada segmento (excepto el segmento raíz) que contiene instrucciones usadas para pasarle el control al segmento.
- Si el segmento ya está en la memoria, el área de transferencia contiene un salto al punto de entrada del segmento.
- Y si no está en la memoria, el área de transferencia contiene instrucciones que le dicen a `MANSUP` que segmento debe cargar.

Transferencia de control

- Las transferencias de control de un segmento a otro son convertidas por el cargador en saltos al área de transferencia.
- Si el segmento ya está en la memoria, de allí se pasa el control al segmento pedido.
- Y si no está en la memoria, de allí se pasa el control al manejador de superposiciones.
- En este caso, `MANSUP` pasará el control al segmento pedido después de cargarlo y de actualizar `TABSEG`.

Consideraciones finales

- El regreso del control de un segmento llamado al segmento que lo llamó se hace de la forma usual y no es necesario involucrar al manejador de superposiciones.
- Las formas más usuales son brincos (`JMP`) y llamadas a subrutinas (`CALL/JSR/RET`).
- Frecuentemente, las funciones del manejador de superposiciones son realizadas por una componente del sistema operativo.

Contenido

Editores de ligado

- A diferencia de un cargador ligador, un editor de ligado sólo genera una versión ligada del programa (llamada módulo de carga) que se escribe en un archivo o biblioteca para su posterior ejecución.
- Al momento de la carga lo único que se tiene que hacer es la relocalización (que se puede hacer con un cargador más sencillo).

Ventajas de los editores de ligado

- Una ventaja es que si un programa se va a ejecutar muchas veces sólo se resuelven una vez las referencias externas, reduciendo considerablemente el tiempo de carga.
- Otra ventaja es que si ya se conoce de antemano la dirección de carga, entonces se puede generar código objeto que no requiera de relocalización durante la carga.

Contenido

Ligado dinámico

- Los editores de ligado realizan las operaciones de ligado antes de la carga.
- Los cargadores ligadores realizan estas operaciones durante la carga.
- Una tercera opción es realizar el ligado después de la carga, es decir, cargando y ligando subrutinas en el momento que se llaman por primera vez.
- A esto se le llama ligado dinámico.

Ventaja del ligado dinámico

- Una ventaja del ligado dinámico es que si un programa contiene muchas subrutinas que se usan con poca frecuencia entonces éstas sólo se cargan si son utilizadas.
- De esta forma se ahorra tiempo de carga y espacio en la memoria considerablemente.

Carga y ligado dinámicos (1)

- Hay varias formas de realizar la carga y ligado dinámicos.
- Una de ellas es que las rutinas que se carguen dinámicamente deben llamarse por medio de una solicitud de servicio al cargador o sistema operativo en lugar de usar un brinco a subrutina.
- El parámetro de esta solicitud es el nombre simbólico de la rutina. El sistema revisa si la rutina ya se ha cargado o no.

Carga y ligado dinámicos (2)

- En caso necesario, la rutina se carga desde la biblioteca adecuada y el sistema le pasa el control a la rutina llamada.
- Cuando la rutina termina su ejecución le devuelve el control al sistema y éste a su vez se lo devuelve al programa que la llamó.
- Esto se hace con el propósito de que el sistema sepa que el espacio de memoria asignado a la rutina se puede reutilizar.
- ¿Porqué no se libera inmediatamente?

Contenido

Cargador de arranque (1)

- ¿Y cómo se carga el cargador?
- La respuesta a esta pregunta no es sencilla.
- Se requiere que otro programa (el cargador de arranque) cargue en la memoria al cargador o al sistema operativo.
- ¿Y cómo se carga el cargador de arranque?
- ¿Con un cargador del cargador de arranque?
- ¿Qué fué primero, el huevo o la gallina?

Cargador de arranque (2)

- El cargador de arranque no puede estar en la RAM al encender una computadora, pero puede estar por ejemplo en una ROM.
- En cualquier caso, éste comienza en una dirección fija y su única función es la de cargar otro programa en otra dirección fija para pasarle el control inmediatamente.
- Este segundo programa es invariablemente más complicado (desde un cargador relativo hasta un sistema operativo completo).

Part IV

Macroprocesadores

Contenido

Macroinstrucciones

- Una macroinstrucción (o macro) representa un grupo de proposiciones usadas frecuentemente en el código fuente.
- El macroprocesador reemplaza cada macro con el código fuente correspondiente, a lo que se llama expansión de macros.
- Las macros permiten escribir versiones abreviadas de programas, dejándole al macroprocesador los detalles mecánicos.
- También se usan con lenguajes de alto nivel.

Definición de macros

- Se usan dos directivas para definir macros.
- La directiva `etiqueta MACRO parámetros` indica el inicio de la definición de una macro de nombre `etiqueta` junto con su lista de `parámetros` (cada uno comienza con `&`).
- A esta línea se le llama prototipo de la macro.
- Después viene el cuerpo de la macro.
- Finalmente, la directiva `MEND` indica el final de la definición de la macro.
- Las definiciones no generan código objeto.

Ejemplo de macrodefinición

```
WRBUFF MACRO &OUTDEV,&BUFADR,&RECLTH
.
.       MACRO QUE ESCRIBE EL REGISTRO DEL BUFFER
.
        CLEAR X          LIMPIA EL CONTADOR DEL CICLO
        LDT   &RECLTH
        LDCH  &BUFADR,X  TOMA EL CARACTER DEL BUFFER
        TD    =X'&OUTDEV' PRUEBA DEL DISPOSITIVO DE SALIDA
        JEQ   *-3        REPITE HASTA QUE ESTE LISTO
        WD    =X'&OUTDEV' ESCRIBE EL CARACTER
        TIXR  T          REPITE HASTA QUE SE HAYAN
        JLT   *-14       ESCRITO TODOS LOS CARACTERES
        MEND
```

Invocaciones a macros

- En el programa principal (y en otras partes) pueden aparecer macrollamadas o invocaciones a macros en las cuales se indica el nombre de la macro que se invoca y los argumentos que se utilizarán en la expansión de la macro.

Ejemplos de macrollamadas

```
        WRBUFF 05,BUFFER,LENGTH

ENDFIL WRBUFF 05,EOF,THREE
```


Ejemplo de expansión de macro

```
.ENDFIL WRBUFF 05,EOF,THREE

ENDFIL CLEAR X          LIMPIA EL CONTADOR DEL CICLO
       LDT   THREE
       LDCH  EOF,X  TOMA EL CARACTER DEL BUFFER
       TD    =X'05' PRUEBA DEL DISPOSITIVO DE SALIDA
       JEQ   *-3        REPITE HASTA QUE ESTE LISTO
       WD    =X'05' ESCRIBE EL CARACTER
       TIXR  T          REPITE HASTA QUE SE HAYAN
       JLT   *-14       ESCRITO TODOS LOS CARACTERES
```


Expansión de macros

- Las definiciones de macros desaparecen.
- Las invocaciones se vuelven comentarios y se agrega el cuerpo de la macro correspondiente.
- Los argumentos han reemplazado a los parámetros, según la posición en la que se hayan listado.
- Se eliminan las líneas de comentarios, pero se conservaron los campos de comentarios.

Etiquetas y expansión

- La etiqueta de la macrollamada se coloca como etiqueta de la primera línea de la macroexpansión, lo que permite usar a las macros como si fueran nemónicos.
- Observe que el cuerpo de la macro no tiene etiquetas, en su lugar hay referencias al contador de localidades.
- Esto evita que las etiquetas aparezcan múltiples veces generando errores de redefinición. Sin embargo, obliga a obtener a mano las referencias a CONTLOC.

Macrodefiniciones anidadas

- Un macroprocesador de un paso puede trabajar con definiciones de macros que contengan a su vez definiciones de macros.
- Lo único que se necesita es que la definición de cada macro aparezca antes de cualquier invocación a la misma macro.

Contenido

Tabla de definiciones

- Las definiciones de las macros se almacenan en una tabla de definiciones (TABDEF) que contiene el prototipo y el cuerpo de la macro (sin incluir comentarios y remplazando los parámetros por una notación posicional).

```
        WRBUFF &OUTDEV,&BUFADR,&RECLTH
        CLEAR X
        LDT   ?3
        LDCH  ?2,X
        TD    =X'?1'
        JEQ   *-3
        WD    =X'?1'
        TIXR  T
        JLT   *-14
        MEND
```


Tablas de nombres y argumentos

- Los nombres de las macros se introducen en la tabla de nombres (TABNOM) que es un índice para TABDEF.
- Cuando se reconoce una invocación a macros se almacenan sus argumentos en una tabla de argumentos (TABARG) de acuerdo a la posición en que aparezcan.

Algoritmo del macroprocesador (1)

- Consta de cuatro procedimientos llamados desde el siguiente programa principal:

Programa principal

1. Haz `EXPANSION = FALSO`.
2. Mientras `CODOP` no sea `END`:
   1. `TOMA-LINEA`.
   2. `PROCESA-LINEA`.

Algoritmo del macroprocesador (2)

Procedimiento `PROCESA-LINEA`

1. Busca `CODOP` en `TABNOM`.
2. Si lo encuentra entonces `EXPANDE`.
3. Si no, si `CODOP` es `MACRO` entonces `DEFINE`.
4. Si no, escribe la línea fuente en el archivo expandido.

Algoritmo del macroprocesador (3)

Procedimiento `DEFINE`

1. Introduce el nombre en `TABNOM` y el prototipo en `TABDEF`.
2. Haz `NIVEL = 1`. Mientras `NIVEL > 0`:
   1. `TOMA-LINEA`.
   2. Si no es una línea de comentario:
      1. Sustituye parámetros por posicional.
      2. Introduce la línea en `TABDEF`.
      3. Si `CODOP` es `MACRO`, incrementa `NIVEL`.
      4. Si `CODOP` es `MEND`, decrementa `NIVEL`.

Algoritmo del macroprocesador (4)

Procedimiento `EXPANDE`

1. Haz `EXPANSION = VERDADERO`.
2. Toma el prototipo de `TABDEF`.
3. Copia los argumentos a `TABARG`.
4. Escribe la invocación como comentario.
5. Mientras no termine la definición de macro:
   1. `TOMA-LINEA`.
   2. `PROCESA-LINEA`.
6. Haz `EXPANSION = FALSO`.

Algoritmo del macroprocesador (5)

Procedimiento `TOMA-LINEA`

1. Si `EXPANSION` es `VERDADERO`:
   1. Toma la siguiente línea de la definición de la macro en `TABDEF`.
   2. Sustituye parámetros por posicional.
2. Si no:
   1. Lee la siguiente línea del archivo fuente.

Contenido

Concatenación de parámetros

- Suponga que una macro tiene el parámetro `&PAR` y que en el cuerpo de la macro aparece el pseudosímbolo `CAD&PAR@ENA`.
- Es muy fácil modificar el macroprocesador para que reemplace `&PAR@` por el valor del parámetro `&PAR`.
- A la `@` se le llama operador de concatenación.
- Esto es útil cuando se van a procesar de forma similar dos o más series de variables que se llaman de forma parecida.

Contenido

Generación de etiquetas únicas (1)

- Anteriormente habíamos prohibido el uso de etiquetas dentro de una macrodefinición.
- Esto es poco conveniente pues produce código propenso a errores y difícil de leer.
- Una forma de permitir etiquetas dentro de una macro es la de asegurar durante la macroexpansión que estas etiquetas se reemplazarán por otras etiquetas únicas, es decir, por etiquetas que no puedan aparecer en otra parte.

Generación de etiquetas únicas (2)

- Un mecanismo simple para lograr esto es agregar un caracter especial (uno que no pueda aparecer en una etiqueta normal) y un contador a cada etiqueta que se genere durante la macroexpansión.
- Este contador se incrementa cada vez que se haga una macroexpansión.
- El contador puede ser numérico (si las etiquetas permiten números) o alfabético (en caso de que sólo se permitan letras en las etiquetas).

Contenido

Expansión condicional de macros

- La mayoría de los macroprocesadores contienen macro instrucciones que permiten cambiar su comportamiento de expansión.
- A esta característica se le llama expansión de macros condicional.
- En general, se tienen macroinstrucciones de asignación (`SET`), de decisión (`IF`, `ELSE`, `ENDIF`) y de ciclos (`WHILE`, `ENDW`).
- Con frecuencia no se permite anidar macro instrucciones.

Contenido

Parámetros con palabras clave

- A veces es conveniente usar parámetros con palabras clave, donde a cada parámetro se le asigna el valor de un argumento con una expresión de asignación.
- Esto permite que los parámetros estén en cualquier orden cuando se invoca una macro.
- Además, en la macrodefinición se podrían asociar valores por omisión a los parámetros, de modo que en una invocación sólo se den como argumentos los valores que sean distintos a los valores por omisión.

Contenido

Expansión de macros recursiva (1)

- El algoritmo del procesador de macros estudiado es capaz de procesar definiciones de macros dentro de otras. Sin embargo, no es capaz de procesar invocaciones de macros dentro de otras.
- Hay dos razones por las cuales ocurre esto:
- La variable `EXPANSION` se pone en falso cuando se sale de la segunda expansión (olvidándose que se estaba a la mitad de la primera expansión).

Expansión de macros recursiva (2)

- Por otro lado, la segunda expansión sobreescribe `TABARG` (perdiendo los argumentos de la primera expansión).
- Estos problemas se pueden resolver fácilmente si el procesador de macros se escribe en un lenguaje de programación que permita llamadas recursivas, de modo que `EXPANSION` y `TABARG` se vuelvan variables locales de los procedimientos del algoritmo.

Contenido

Macroprocesadores generales

- Con frecuencia los procesadores de macros corresponden con un lenguaje de programación específico. Sin embargo, tener un procesador de macros general tiene la ventaja de que el programador no debe aprender un mecanismo de macros especial para cada lenguaje que conozca.
- Además, aunque la escritura de uno de estos macroprocesadores generales es más complicada que la de un macroprocesador específico, sólo se escribiría una vez.

Macroprocesadores generales (2)

- Algunas de las dificultades a resolver en la escritura de un macroprocesador general son:
  - Ignorar los comentarios (que se señalan de forma distinta en cada lenguaje).
  - Identificación de constantes, operadores, identificadores y espacios en blanco.
  - Simulación de la sintaxis para invocación de macros, etc.

Contenido

Macroprocesamiento en traductores (1)

- A los procesadores de macros estudiados hasta ahora también se les llama preprocesadores: estos procesan definiciones de macros y expanden invocaciones de macros, produciendo una versión expandida del código fuente, la cual se usará como entrada para un ensamblador o compilador.
- Sin embargo, también existe la posibilidad de combinar las funciones de procesamiento de macros con el traductor de lenguaje.

Macroprocesamiento en traductores (2)

- El método más simple es el de un macroprocesador línea a línea, en el que la salida del macroprocesador se pasa al traductor conforme se va produciendo en lugar de escribirse en un archivo temporal.
- La cooperación puede ser más estrecha aún: se pueden compartir datos y rutinas, mejorar los mensajes de error emitidos, etc.
- Una desventaja de estos macroprocesadores es que se deben diseñar para un traductor específico que se volverá más complejo.

Esto es todo

Fin