Programación en lenguaje ensamblador

Lenguaje ensamblador y programa ensamblador.
Pseudoinstrucciones.
Implementación de estructuras de decisión e iterativas.
Subrutinas y paso de parámetros.
Representación de estructuras de datos.

---------- ya lo hemos ido viendo ----------

Lenguaje ensamblador y programa ensamblador

```
00001000 Starting Address
Assembler used: EASy68K Editor/Assembler v5.16.01
Created On: 04/04/2019 8:29:13

00000000                             1  *-----------------------------------------------------------
00000000                             2  * Title      :
00000000                             3  * Written by :
00000000                             4  * Date       :
00000000                             5  * Description:
00000000                             6  *-----------------------------------------------------------
00001000                             7      ORG    $1000
00001000                             8  START:                  ; first instruction of program
00001000 41F9 00001028               9      LEA VECTOR,A0
00001006 6104                       10      BSR.s MAX_IN_BYTE_VECTOR
00001008 FFFF FFFF                  11      SIMHALT             ; halt simulator
0000100C                            12
0000100C                            13  MAX_IN_BYTE_VECTOR
0000100C 48E7 4080                  14      movem.l  A0/D1,-(SP)
00001010 4200                       15      clr.b    D0
00001012 3218                       16      move.w   (A0)+,D1
00001014 6008                       17      bra.s    noMayor
00001016                            18  CICLO:
00001016 B018                       19      cmp.b    (A0)+,D0
00001018 6204                       20      bhi.s    noMayor
0000101A 1028 FFFF                  21      move.b   -1(A0),D0
0000101E                            22  noMayor:
0000101E 51C9 FFF6                  23      dbra     d1,CICLO
00001022 4CDF 0102                  24      movem.l  (SP)+,A0/D1
00001026 4E75                       25      rts
00001028                            26
00001028                            27  VECTOR
00001028= 0006                      28      DC.W     6
0000102A= 03 06 04 07 08 03         29      DC.B     3,6,4,7,8,3
00001030                            30
00001030                            31      END      START

No errors detected
No warnings generated

SYMBOL TABLE INFORMATION
Symbol-name         Value
-------------------------
CICLO               1016
MAX_IN_BYTE_VECTOR  100C
NOMAYOR             101E
START               1000
VECTOR              1028
```

comentario

Pseudo-Instruccion

comentario

simulador

Etiqueta
(dirección de memoria)

Direcciones de memoria

Words de extensión

Instrucción 68000

Códigos de operación

Constantes

simulador

Pseudo-Instrucciones

Programa “máquina”

Ensamblador

Del Simulador EASy68k

Disponible en la sección “extras” de la web del curso

---------- ya hemos ido viendo lo básico ----------

Pseudoinstrucciones.

Definitions
- SET and EQU
- SFR and SFRB
- XSFR and YSFR
- LABEL
- BIT
- DBIT
- PORT
- REG
- LIV and RIV
- CHARSET
- CODEPAGE
- ENUM
- STRUCT and ENDSTRUCT
- PUSHV and POPV

```
<name>  MACRO   [parameter list]
        <instructions>
        ENDM
```

Data Definitions
- DC[.Size]
- DS[.Size]
- DB,DW,DD,DQ, and DT
- DS, DS8
- BYT or FCB
- BYTE
- DC8
- ADR or FDB
- WORD
- DW16
- LONG
- SINGLE and EXTENDED
- FLOAT and DOUBLE
- EFLOAT, BFLOAT, and TFLOAT
- Qxx and LQxx
- DATA
- ZERO
- FB and FW
- ASCII and ASCIZ
- STRING and RSTRING
- FCC
- DFS or RMB
- BLOCK
- SPACE
- RES
- BSS
- DSB and DSW
- DS16
- ALIGN
- LTORG

Macro Instructions
- MACRO
- IRP
- IRPC
- REPT
- WHILE
- EXITM
- SHIFT
- MAXNEST
- FUNCTION

Listing Control
- PAGE
- NEWPAGE
- MACEXP
- LISTING
- PRTINIT and PRTEXIT
- TITLE
- RADIX
- OUTRADIX

Conditional Assembly
- IF / ELSEIF / ENDIF
- SWITCH / CASE / ELSECASE / ENDCASE

Code Modification
- ORG
- CPU
- SUPMODE, FPU, PMMU
- FULLPMMU
- PADDING
- PACKING
- MAXMODE
- EXTMODE and LWORDMODE
- SRCMODE
- BIGENDIAN
- WRAPMODE
- SEGMENT
- PHASE and DEPHASE
- SAVE and RESTORE
- ASSUME
- EMULATED
- BRANCHEXT

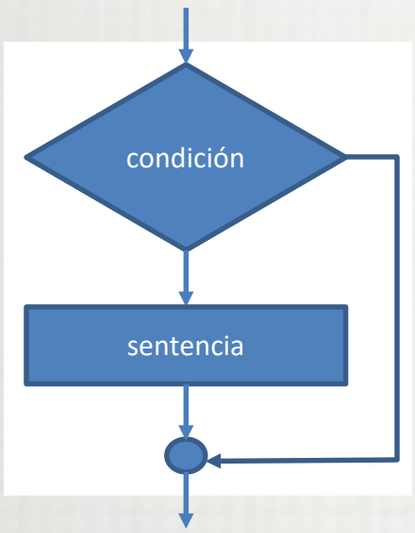
Local Symbols
- Basic Definition (SECTION/ENDSECTION)
- Nesting and Scope Rules
- PUBLIC and GLOBAL
- FORWARD
- Performance Aspects

Miscellaneous
- SHARED
- INCLUDE
- BINCLUDE
- MESSAGE, WARNING, ERROR, and FATAL
- READ
- RELAXED
- END

Del Manual de usuario del macro ensamblador AS

Disponible en la sección “extras” de la web del curso

Implementación de estructuras de decisión e iterativas



If (condición) sentencia

No tiene mayor dificultad. Se utiliza la capacidad de las instrucciones de ruptura de secuencia condicionales.

```
        …               ; evaluación de una expresión que refleje el resultado en los CC
        Bcc     fin     ;
        …               ;segmento de sentencia
fin:
```

* Evaluación de expresiones (notación infija y postfija) -> ver siguiente página

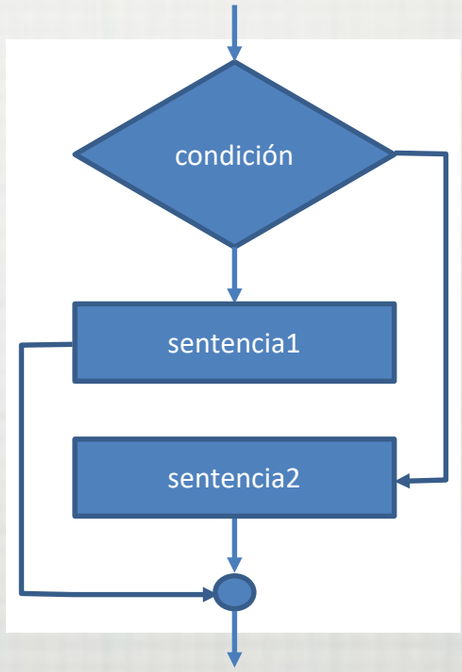
---

If (condición) sentencia1; else sentencia2

Del mismo modo sin dificultad. Se utiliza la capacidad de las instrucciones de ruptura de secuencia condicionales e incondicionales.

```
        …               ; evaluación de una expresión que refleje el resultado en los CC
        Bcc     else    ;
        …               ;segmento de sentencia1
        BRA     fin
else:   …
fin:
```

* Evaluación de expresiones (notación infija y postfija) -> ver siguiente página



Implementación de estructuras de decisión e iterativas

Para evaluar las condiciones es preciso conocer cómo se evalúan expresiones en general

Notacion infija

5+4*3/(2+1)

Se procesa elemento a elemento (operadores y operandos), de izquierda a derecha, con ayuda de una pila.

- los operandos se llevan al resultado
- los operadores se enfrentan a la cumbre de la pila y si su precedencia es…
  - …mayor, se apila.
  - …menor, se purga la pila al resultado hasta que no lo sea
  - …igual, se purga la cumbre de la pila al resultado y se apila

Los "(" se apilan y los ")" purgan la pila hasta un "("

Elemento	resultado	pila
5	5	
+	5	+
4	5,4	+
*	5,4	+*
3	5,4,3	+*
/	5,4,3,*	+/
(	5,4,3,*	+/(
2	5,4,3,*,2	+/(
+	5,4,3,*,2	+/(+
1	5,4,3,*,2,1	+/(+
)	5,4,3,*,2,1,+	+/
FIN	5,4,3,*,2,1,+,/,+	

Notacion postfija

5,4,3,2,1,+,*,/,+

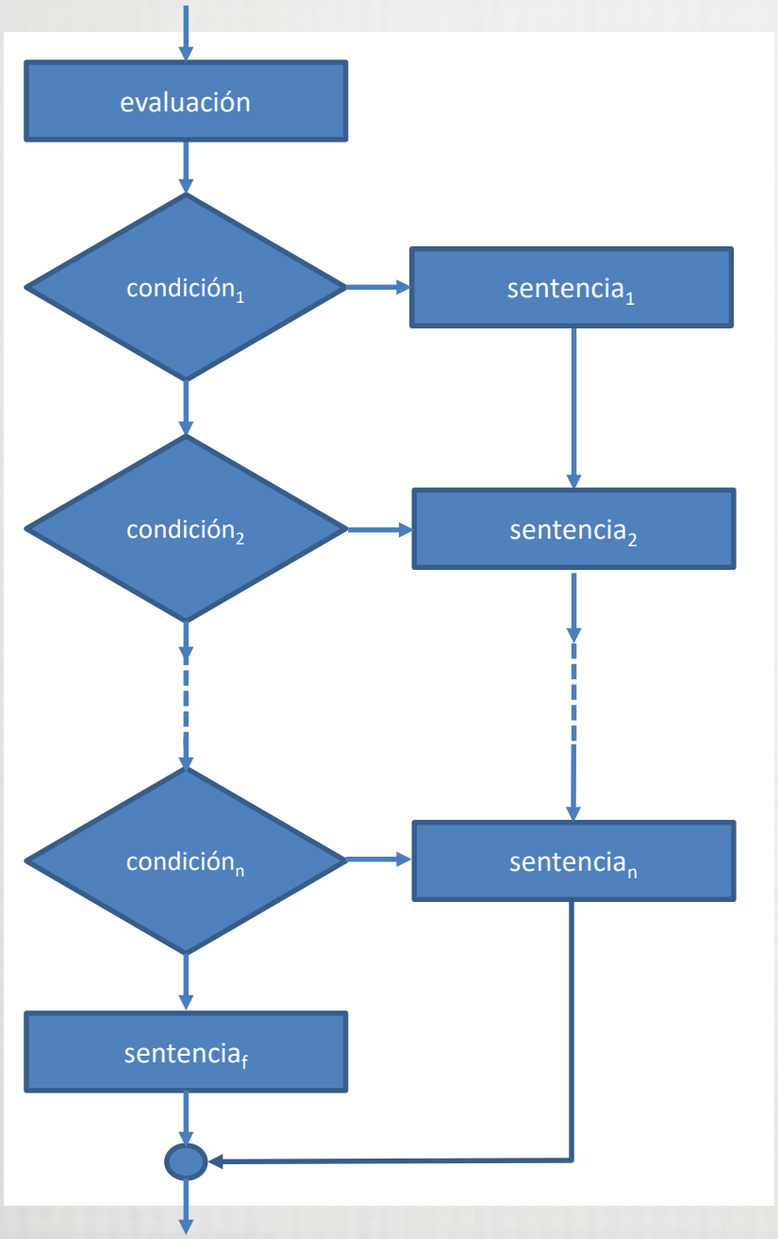
Se procesa elemento a elemento, de izquierda a derecha, con ayuda de una pila.

- los operandos se apilan
- los operadores se ejecutan extrayendo dos elementos de la pila y apilando el resultado.

5	4	3	*	2	1	+	/	+
					1			
		3		2	2	3		
	4	4	12	12	12	12	4	
5	5	5	5	5	5	5	5	9

La notación postfija se denomina también sufija o "polaca inversa" en referencia a la notación prefija introducida en matemáticas por Lukasiewick en 1920

Implementación de estructuras de decisión e iterativas



switch (expresión) { [case e_i: $sentencia_i$;]* default: $sentencia_f$;}

Ver esta estructura en ensamblador clarifica su funcionamiento en alto nivel.

```
        …               ; evaluación de una expresión con valor enumerable
        …               ; chequeo del caso 1
        Bcc     s1      ;
        …               ; chequeo del caso 2
        Bcc     s2      ;
        …               ; --- chequeos y saltos intermedios---
        …               ; chequeo del caso n
        Bcc     sn      ;
        …               ;segmento de sentencia final (default)
        BRA     fin
s1:     …               ; segmento de sentencia 1
s2:     …               ; segmento de sentencia 2
        …               ;---segmentos de sentencias intermedias---
sn:     …               ; segmento de sentencia n
fin:
```

Introducir un “break” supone insertar un “BRA fin”

Implementación de estructuras de decisión e iterativas

while (condición) sentencia;
do sentencia; while (condición);
for (inicialización, mantenimiento, ciclo) sentencia;
foreach (variable: conjuto) sentencia;

****respuesta a una pregunta sobre uso del "for" (en su versión más ámplia, p.ej. Java) más allá de ser un ciclo con un contador.**

El "for" encapsula un "while" con variables (posiblemente) locales previas y con (posiblemente) una(s) acción(es) de iteración concreta(s). En sentido estricto no es una estructura "canónica", si bien facilita la posibilidad de que las variables previas sean locales.

```
for (inicialización;condición;iteración)
        sentencia
```
Equivale a
```
        { inicialización;
          while (condición) {
            sentencia;
            iteración;
          }
        }
```

- Su aplicación trivial es la ejecución de una sentencia un número predeterminado de veces:

```
for (int i=0;i<10;i++) System.out.println("Hola, mundo");
```

- En todo caso, y en general, la estructura "for" puede usarse a conveniencia, como en el siguiente ejemplo:

```
static void quickSort(int[] m, int start, int end){
        int i=start, j=end;
        for (int pivote=m[(i+j)/2];i<j;){
                while (m[i]<pivote) i++; while (pivote<m[j]) j--;
                if (i<=j) {int tmp=m[i];m[i++]=m[j];m[j--]=tmp;}
        }
        if (start<j) quickSort(m,start,j);
        if (i<end) quickSort(m,i,end);
}
```

donde el "for" no es otra cosa que una inicialización y un "while", pero aportando la restricción sintáctica de hacer que la variable "pivote" sea local al mismo "for". Las restricciones sintácticas son una aportación del lenguaje de alto nivel. Los ensambladores pueden aportar alguna pero nada comparable, y en todo caso, en el momento de la ejecución (obviamente) ese tipo de restricciones no existen.

- Muchos lenguajes proporcionan una utilidad específica bajo la misma sentencia "for" (conocida en ocasiones como "foreach"), como es el recorrido de una colección de datos.

```
String[] c= {"Ana","Benito","Carlos","Diana"});
for (String s:c) System.out.println("Hola,"+s);
```

que es en realidad un "recubrimiento" del uso del patrón "Iterador"

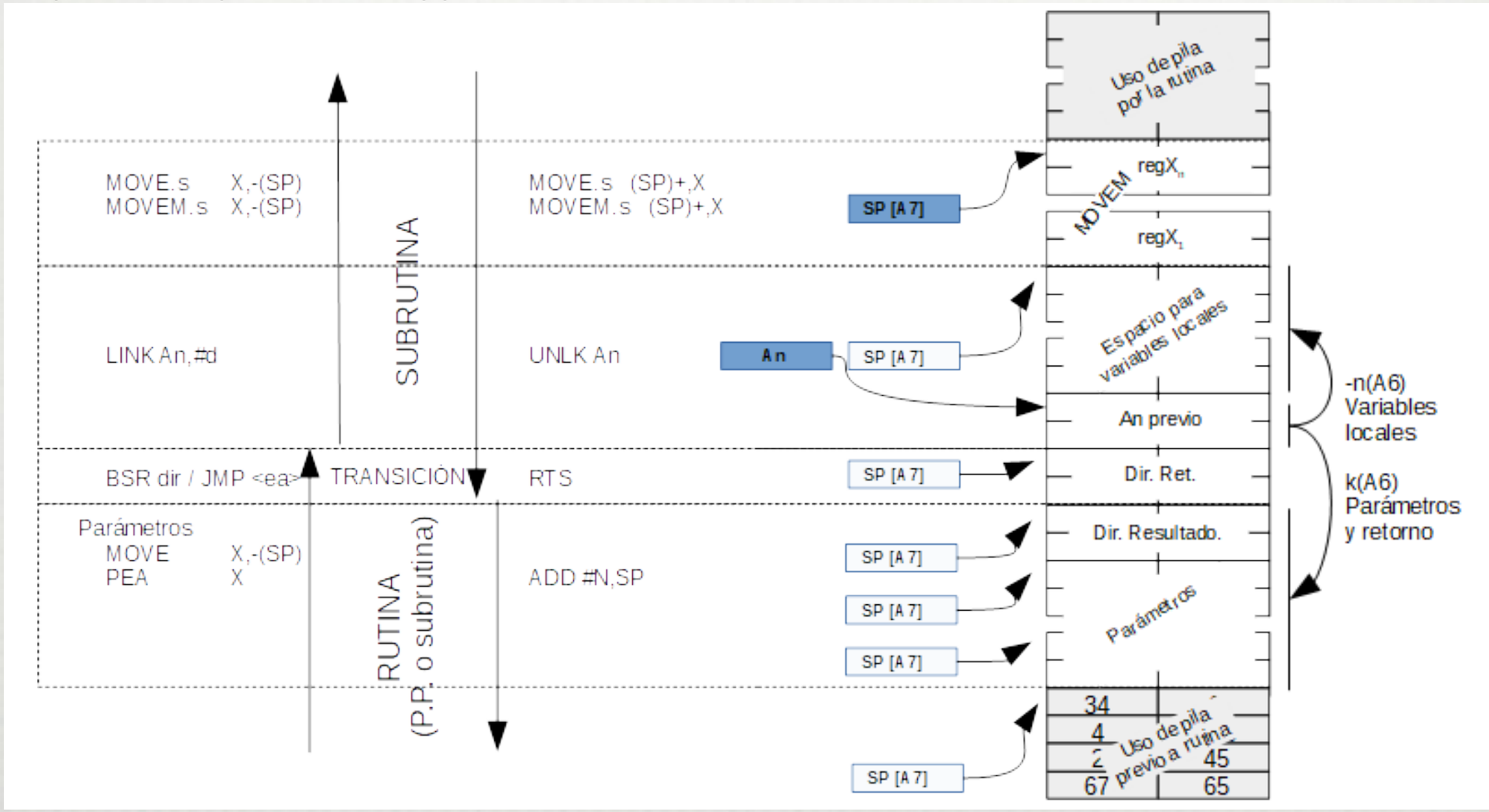
```
Iterator<String> c=Arrays.asList(new String[]{"Ana","Benito","Carlos","Diana"}).iterator();
for (String s; c.hasNext();) {s=c.next(); System.out.println("Hola,"+s);}
```

Subrutinas y paso de parámetros.

---------- ya lo hemos ido viendo ----------

Paso de parámetros por registro → uso de MOVEM
por pila → uso de LINK… variables locales
por memoria → acceso a variables “globales”… el antiguo “common” de Fortran

Paso de parámetros por referencia y por valor.

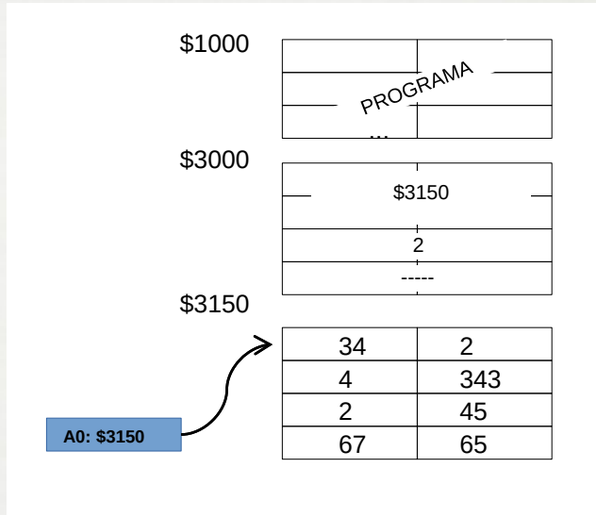


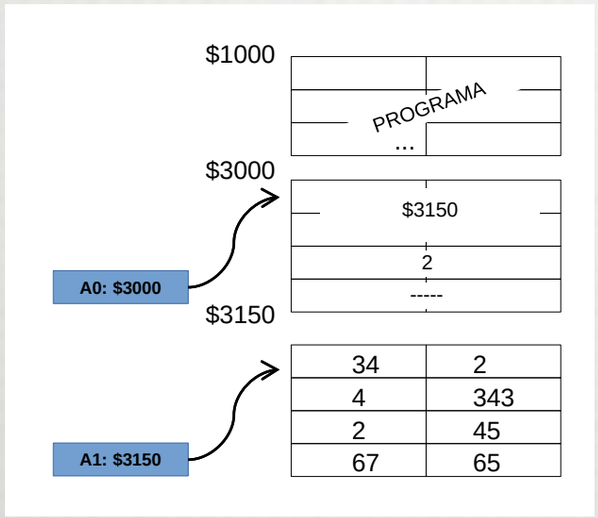
Este es el mecanismo general. Puede haber casos particulares (p.ej. retornos en registros) y optimizaciones en compiladores y máquinas virtuales.

Tiempos de ejecución de instrucciones

Cálculo de tiempos de ejecución (ciclos)
(1) vs. (3) y (2) vs. (4)

```
00003000                                10          ORG $3000
00003000                                11  STRUCT
00003000= 00003150                      12      DC.L    ARRAY
00003004= 0002                          13      DC.W    2
00003006                                14      DS.W    1
00003008                                15
00001000                                16      ORG $1000
00001000                                17  PROG
00001000  2078 3000                     18      MOVE.L  STRUCT,A0           ; (1)
00001004  3038 3004                     19      MOVE.W  STRUCT+4,D0         ; (1)
00001008  C0FC 0002                     20      MULU    #2,D0               ; (2)
0000100C  31F0 0000 3006                21      MOVE.W  (A0,D0.W),STRUCT+6  ; (1)
00001012  4E4F                          22      TRAP    #15
00001014= 0000                          23      DC.W    0
00001016                                24
00003150                                25      ORG $3150
00003150                                26  ARRAY
00003150= 0022 0002 0004 0...           27      DC.W    34,2,4,343,23,45,67,65
```






```
00003000                                10      ORG $3000
00003000                                11  STRUCT
00003000= 00003150                      12      DC.L    ARRAY
00003004= 0002                          13      DC.W    2
00003006                                14      DS.W    1
00003008  =00000004                     15  INDICE  EQU 4
00003008  =00000006                     16  RESULT  EQU 6
00003008                                17
00001000                                18      ORG $1000
00001000                                19  PROG
00001000  41F8 3000                     20      LEA STRUCT,A0                   ; (3)
00001004  2250                          21      MOVE.L  (A0),A1                 ; (3)
00001006  3028 0004                     22      MOVE.W  INDICE(A0),D0           ; (3)
0000100A  E348                          23      LSL.W   #1,D0                   ; (4)
0000100C  3171 0000 0006                24      MOVE.W  (A1,D0.W),RESULT(A0)    ; (3)
00001012  4E4F                          25      TRAP    #15
00001014= 0000                          26      DC.W    0
00001016                                27
00003150                                28      ORG $3150
00003150                                29  ARRAY
00003150= 0022 0002 0004 0...           30      DC.W    34,2,4,343,23,45,67,65
```

Representación de estructuras de datos.

Las diferentes estructuras de datos que podemos definir/usar en un sistema tienen relación directa con la Gestión de Memoria. En particular la Asignación de Memoria para una estructura de datos puede ser de tres tipos:

Asignación estática: es la que se define directamente en posiciones fijas en tiempo de programación.

Asignación automática: se denomina así la que utiliza la pila del sistema. Es típica de variables locales.

Asignación dinámica: es la más general y requiere de mecanismos complejos para estructurar y optimizar la gestión del espacio disponible. Uno de los mecanismos más usados para gestionar el espacio (denominado Heap –no confundir con la estructura de datos-) es el denominado algoritmo Buddy.

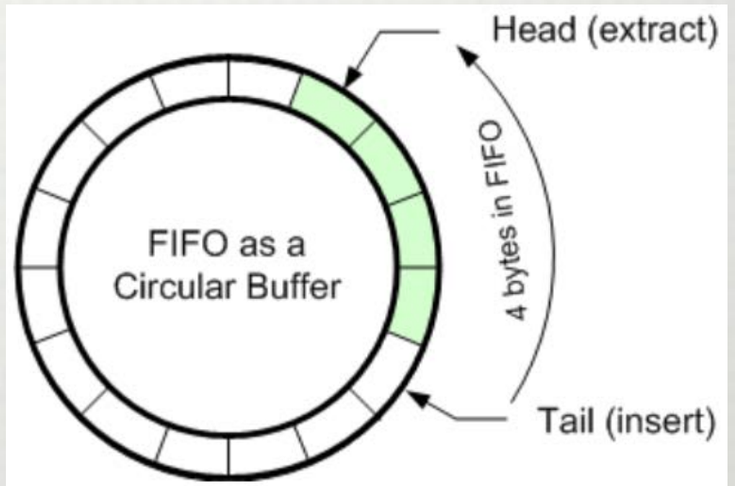
Estructuras sencillas. Un par de ejemplos…

Pilas (LIFO – Last In First Out) -> trivial (o no)

Hemos visto que los modos de direccionamiento –(An) y (An)+ nos permiten gestionar un espacio de memoria directamente como una pila facilitándonos sus operaciones “Push” y “Pop”. Obviamente si no hay ningún control de que el espacio usado no excede del reservado para tal estructura pueden surgir problemas. Un sistema robusto establecerá algún control en ese sentido.

Colas (FIFO – First In First Out) -> no tan trivial

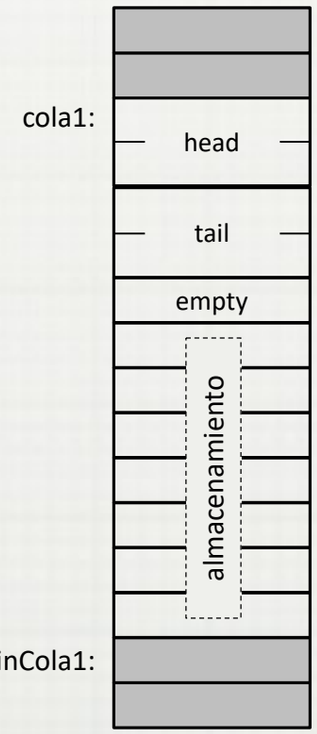
Los modos de direccionamiento –(An) y (An)+ también nos permiten realizar las operaciones de una cola con sencillez ya que sus operaciones son esencialmente las mismas de la pila, ahora denominadas “enqueue” y “dequeue”. Pero hay un problema añadido al del control del espacio reservado: las colas “se desplazan” (hacia direcciones bajas si usamos –(An) o, alternativamente, altas si usamos (An)+. La solución a este problema es la implementación de colas “circulares”



Representación de estructuras de datos. -cola circular-

*nota.- Cambiada respecto a la inicial porque le faltaba el marcador "empty" (puede incluirse "truculentamente" en el byte alto de "head" o "tail", pero optamos por la claridad)

```
    ORG    $1000  ; Sección de inicialización del sistema
START:
    ;inicializaciones
    ;instrucciones de inicialización de cola1
    ;entrada en modo de servicio

    SIMHALT    ; halt simulator



    ORG $1500  ;sección de programa (con uso de estructuras de datos)
cola1Len:    EQU 128
cola1:       DS.W cola1Len
finCola1:

    ORG $2000   ;sección de definición de estructuras de datos
ccHead:      EQU 0
ccTail:      EQU 4
Ccempty:     EQU 8
ccData:      EQU 10

;RUTINA  ccQUEUE
;FUNCION encola una word
;ENTRADA en la pila: referencia a la cola, la word a encolar
;SALIDA  noZ si se ha realizado con exito, Z en caso contrario.

;RUTINA  ccDEQUEUE
;FUNCION desencola una word
;ENTRADA en la pila: referencia a la cola
;SALIDA  en la pila: la word desencolada
;       noZ si se ha realizado con exito, Z en caso contrario.

;RUTINA  ccISFULL
;FUNCION indica si la cola está llena
;SALIDA  noZ sila pila está llena, Z en caso contrario.

;RUTINA  ccISEMPTY
;FUNCION indica si la cola está vacía
;SALIDA  noZ sila pila está vacía, Z en caso contrario.

END   START
```



Operaciones a implementar:
[inicialización]
quewe
dequeue
isFull
isEmpty