

LEGv8 avanzado

OdC - 2023

Ejercicio 3: Enunciado

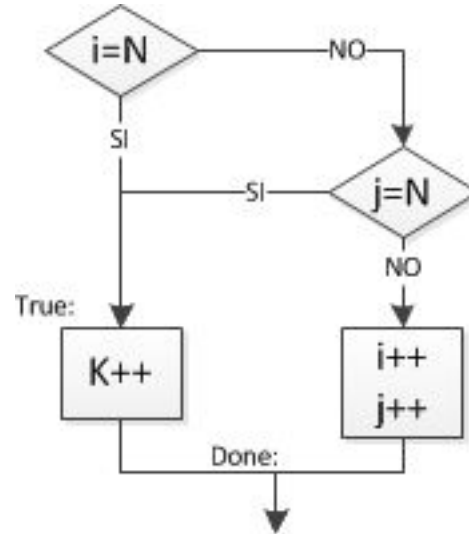
Dado el siguiente programa “C” y la asignación $i, j, k, N \leftrightarrow X0, X1, X2, X9$, escribir el programa LEGv8 que lo implementa. Notar que como se usa el operador `||` la evaluación es por cortocircuito. **Opcional:** hacerlo con el operador `|` que no está cortocircuitado.

```
long long int i,j,k,N;

if (i==N || j==N) {
    ++k;
} else {
    ++i; ++j;
}
```

Ejercicio 3: Cortocircuitado

```
long long int i,j,k,N;  
  
if (i==N || j==N) {  
    ++k;  
} else {  
    ++i; ++j;  
}
```



Ejercicio 3: Cortocircuitado

```
long long int i,j,k,N;
```

```
if (i==N || j==N) {
```

```
    ++k;
```

```
} else {
```

```
    ++i; ++j;
```

```
}
```

$x0 \leftrightarrow i$
$x1 \leftrightarrow j$
$x2 \leftrightarrow k$
$x3 \leftrightarrow N$

True:

end:

```
sub x9, x0, x3
```

```
cbz x9, True
```

```
cmp x1, x3
```

```
b.eq x9, True
```

```
add x0, x0, #1
```

```
add x1, x1, #1
```

```
b end
```

```
addi x2, x2, #1
```

```
// comparo i con N y lo guardo en x9
```

```
// Si son iguales salto a True
```

```
// comparo j con N
```

```
// Si son iguales salto a True
```

```
// ++i
```

```
// ++j
```

```
// ++k
```

Ejercicio 3: No cortocircuitado

```
long long int i,j,k,N;
```

```
if (i==N | j==N) {
```

```
    k = 2;
```

```
} else {
```

```
    ++k;
```

```
}
```

```
return(k);
```

```
sub    sp, sp, #32
```

```
str    x0, [sp, 24]    //X0 = i
```

```
str    x1, [sp, 16]    //X1 = j
```

```
str    x2, [sp, 8]     //X2 = k
```

```
str    x3, [sp]        //X3 = N
```

```
ldr    x1, [sp, 24]    //X1 = i
```

```
ldr    x0, [sp]        //X0 = N
```

```
cmp    x1, x0          //if(i==N){
```

```
cset   w0, eq          //    w0 = 1}
```

```
and    w1, w0, 255     //w1 = w0 and 0xff
```

```
ldr    x2, [sp, 16]    //x2 = j
```

```
ldr    x0, [sp]        //X0 = N
```

```
cmp    x2, x0          //if(j==N){
```

```
cset   w0, eq          //    w0 = 1}
```

```
and    w0, w0, 255     //w0 = w0 and 0xff
```

```
orr    w0, w1, w0       //w0 = w0 or w1
```

```
and    w0, w0, 255     //w0 = w0 and 0xff
```

```
cmp    w0, 0           //if(w0 == 0) {
```

```
beq    .L2             //    goto L2}
```

```
mov    x0, 2
```

```
str    x0, [sp, 8]
```

```
b      .L3
```

} True

```
.L2:
```

```
ldr    x0, [sp, 8]
```

```
add    x0, x0, 1
```

```
str    x0, [sp, 8]
```

```
.L3:
```

```
ldr    x0, [sp, 8]
```

```
add    sp, sp, 32
```

```
ret
```

} False

x0 ↔ i ↔ [sp, 24]

x1 ↔ j ↔ [sp, 16]

x2 ↔ k ↔ [sp, 8]

x3 ↔ N ↔ [sp, 0]

aarch64-linux-gnu-gcc -O0 -S -o- -xc ej3.c > main.s

Ejercicio 6: Enunciado

Traducir el siguiente programa en “C” a ensamblador LEGv8 dada la asignación de variables a registros X0, X1, X2, X9 \leftrightarrow str, found, i, N. El número 48 se corresponde con el carácter ‘0’ en ASCII, por lo tanto el programa cuenta la cantidad de ‘0’s que aparecen en una cadena de caracteres de longitud N.

```
#define N (1<<10)
char *str;
long found, i;
for (found=0, i=0; i!=N; ++i)
    found += (str[i]==48);
```

Ejercicio 6: Resolución

```
#define N (1<<10)
char *str;
long found, i;
for (found=0, i=0; i!=N; ++i)
    found += (str[i]==48);
```

X0 ↔ str
X1 ↔ found
X2 ↔ i
X9 ↔ N

```
.data
str: .dword 0x754d30616c6f4830, 0x00000000306f646e
N: .dword 16
.text
ldr X0, =str           // X0 = &str[0]
ldr X9, N               // N = 16
add X1, XZR, XZR        // found = 0
add X2, XZR, XZR        // i = 0
for: cmp X2, X9          // comparo i y N
    b.eq end            // Salto si son iguales (termina el for)
    add X11, X0, X2      // X11 = &str[0] + i
    ldurb W12, [X11, #0] // X12 = str[i]
    cmp W12, #48         // Verifico si el byte que traje es un 0
    b.ne skip           // Si son distintos no lo cuento
    add X1, X1, #1       // Si es un cero, found +=1
skip: add X2, X2, #1     // i = i + 1
    B for
end:
```

Ejercicio 7: Enunciado

Traducir el siguiente programa “C” a LEGv8. La asignación de variables a registros X0, X1, X2, X3, X9 ↔ A, s, i, j, N. Notar que en “C” los arreglos bidimensionales se representan en memoria usando un **orden por filas**: $\&A[i][j] = \&A[0][0] + 8*(i*N+j)$.

```
#define N (1<<10)
long A[N][N], s, i, j;
s=0;
for (i=0; i<N; ++i)
    for (j=0; j<N; ++j)
        s += A[i][j];
```

X0 ↔ &A[0]
X1 ↔ s
X2 ↔ i
X3 ↔ j
X9 ↔ N

A[2][3] =

1	7	2
44	3	21

A ->


1	7	2	44	3	21
---	---	---	----	---	----

Ejercicio 7.0: Resuelto

```
.data
    N: .dword 3
    A: .dword 1,7,2,44,3,21,1,2,3 // A[N][N]

.text
    ldr X0, =A           // x0 =&A[0][0]
    ldr X9, N             // N=3
    add X1, XZR, XZR      // s=0
    add X2, XZR, XZR      // i=0
    add X3, XZR, XZR      // j=0
```

X0	↔	&A[0]
X1	↔	s
X2	↔	i
X3	↔	j
X9	↔	N



```
oLoop:    cmp X2,X9           // if(i==N)
          b.eq oEnd          // goto oEnd;

iLoop:    add X3, XZR, XZR     // j=0
          cmp X3,X9           // if(j==N)
          b.eq iEnd          // goto iEnd;
          mul X12, X2, X9      // X12 = i * N
          add X12, X12, X3     // X12 = (i * N) + j
          lsl x12, x12, #3     // X12 = ((i * N) + j) * 8
          add X12, X12, X0     // X12 = &A[0][0] + ((i * N) + j) * 8
          ldur X11, [X12,#0]   // X11=A[i][j]
          add X1, X1, X11      // s+=A[i][j]
          addi X3, X3, #1      // j++;

          b iLoop

iEnd:     addi X2, X2, #1      // i++;

oEnd:
```

Ejercicio 7.1: Enunciado

Traducir el siguiente programa “C” a LEV8. La asignación de variables a registros X0, X1, X2, X3, X9 \leftrightarrow A, s, i, j, N. Notar que en “C” los arreglos bidimensionales se representan en memoria usando un **orden por filas**, es decir $\&A[i][j] = A + 8*(i*N+j)$.

7.1) Hacer lineal el acceso al arreglo y recorrerlo con un solo lazo.

```
#define N (1<<10)
long A[N][N], s, i, j;
s=0;
newN = N * N
for (i=0; i<newN; ++i)
    s += A[i];
```

X0	\leftrightarrow	$\&A[0]$
X1	\leftrightarrow	s
X2	\leftrightarrow	i
X3	\leftrightarrow	j
X9	\leftrightarrow	N

A[2][3] =

1	7	2
44	3	21

A ->

1	7	2	44	3	21
---	---	---	----	---	----

Ejercicio 7.1: Resuelto


```
.data
N: .dword 3
A: .dword 1,7,2,44,3,21,1,2,3 // A[N][N]

.text
ldr X0, =A           // x0 =&A[0][0]
ldr X9, N             // N=3
add X1, XZR, XZR      // s=0
add X2, XZR, XZR      // i=0
mul X9, X9, X9         // newN = N * N
```

X0 ↔ &A[0]
X1 ↔ s
X2 ↔ i
~~X3~~ ↔ j
X9 ↔ N



oLoop:



```
cmp X2,X9             // if(i==N)
b.eq oEnd             // goto oEnd;
add X12, XZR, X2       // X12 = i
lsl x12, x12, #3       // X12 = i * 8
add X12, X12, X0       // X12 = &A[0] + i * 8
ldur X11, [X12,#0]     // X11=A[i]
add X1, X1, X11        // s+=A[i]
addi X2, X2, #1        // i++
b oLoop
```

oEnd:

Ejercicio 7.2: Enunciado

Traducir el siguiente programa “C” a LEV8. La asignación de variables a registros X0, X1, X2, X3, X9 \leftrightarrow A, s, i, j, N. Notar que en “C” los arreglos bidimensionales se representan en memoria usando un **orden por filas**, es decir $\&A[i][j] = A + 8*(i*N+j)$.

7.2) Se puede hacer lo mismo sin usar ninguna variable índice i, j.

```
#define N (1<<10)
long A[N][N], s, i, j;
s=0;
finalAddress = &A[0][0] + (N * N)
for (i=0; i!=finalAddress; ++i)
    s += A[i];
```

X0 \leftrightarrow &A[0]
X1 \leftrightarrow s
X2 \leftrightarrow i
X3 \leftrightarrow j
X9 \leftrightarrow N

A[2][3] =

1	7	2
44	3	21

A ->

1	7	2	44	3	21
---	---	---	----	---	----

Ejercicio 7.2: Resuelto

```
.data
N: .dword 3
A: .dword 1,7,2,44,3,21,1,2,3 // A[N][N]

.text
ldr X0, =A           // x0 =&A[0][0]
ldr X9, N             // N=3
add X1, XZR, XZR      // s=0
mul X9, X9, X9         // X9= N * N
lsl x9, x9, 3         // X9= N * N * 8
add X9, x9, X0        // finalAddr = &A[0][0] + (N*N*8)
```

X0 ↔ &A[0]
X1 ↔ s
~~X2 ↔ i~~
~~X3 ↔ j~~
X9 ↔ N



oLoop:



```
cmp X0,X9             // if(i==finalAddr)
b.eq oEnd             // goto oEnd;
ldur X11, [X0,#0]     // X11=A[i]
add X1, X1, X11        // s+=A[i]
addi X0, X0, #8        // i++
b oLoop
```

oEnd:

QEMU - Memory map

