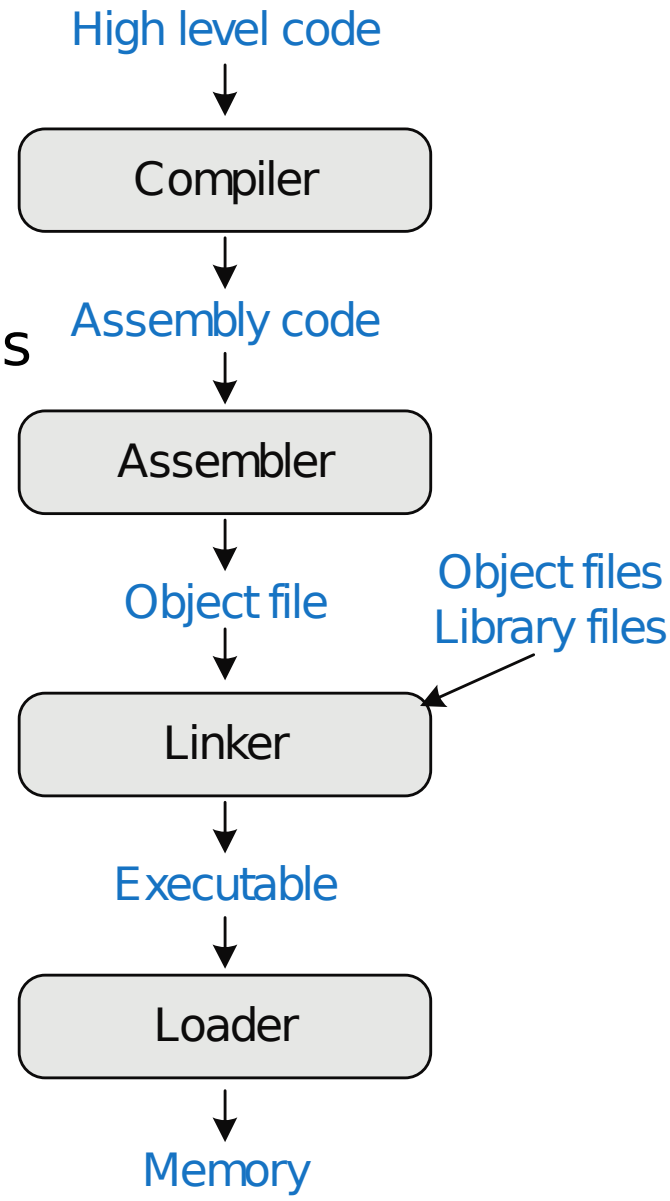
Técnicas Digitales II

Compilado, Ensamblado, Enlazado y Carga

Dr.Ing.Steiner Guillermo

Funciones

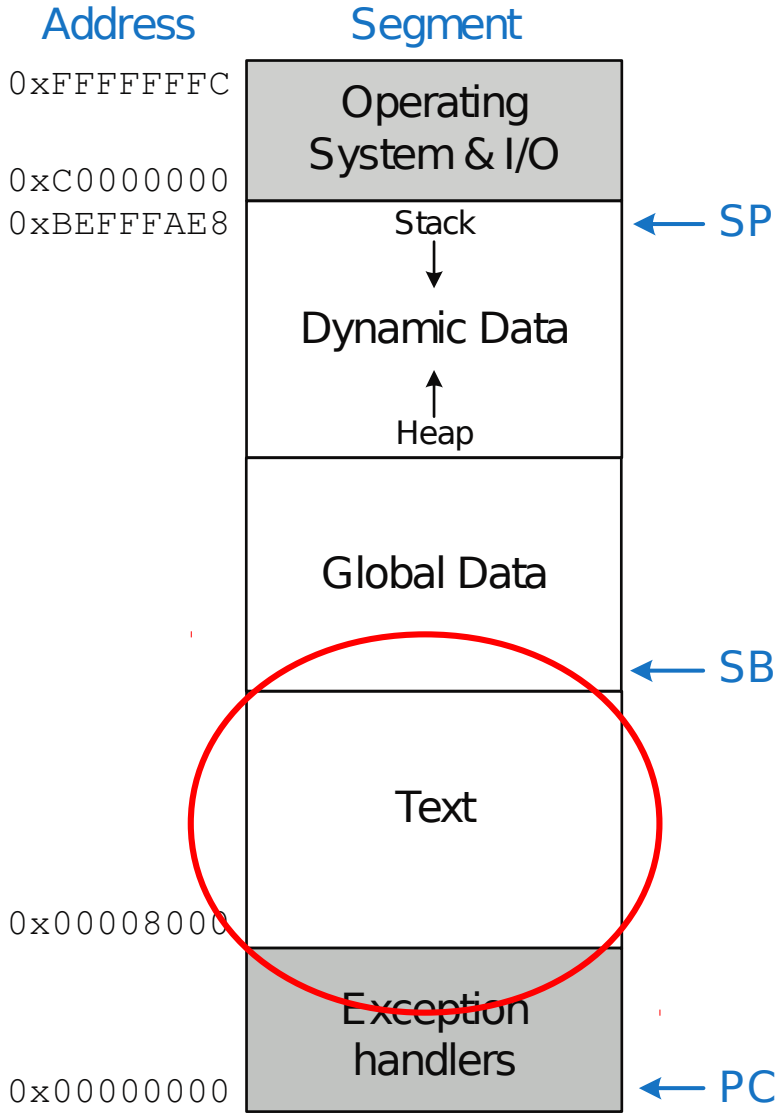
- En clases anteriores, pequeños código en alto nivel fueron trasladado a ensamblador y luego a código maquina .
- Aquí veremos las herramientas necesarias para realizar esta tarea en un programa o proyecto completo.



Mapa de Memoria

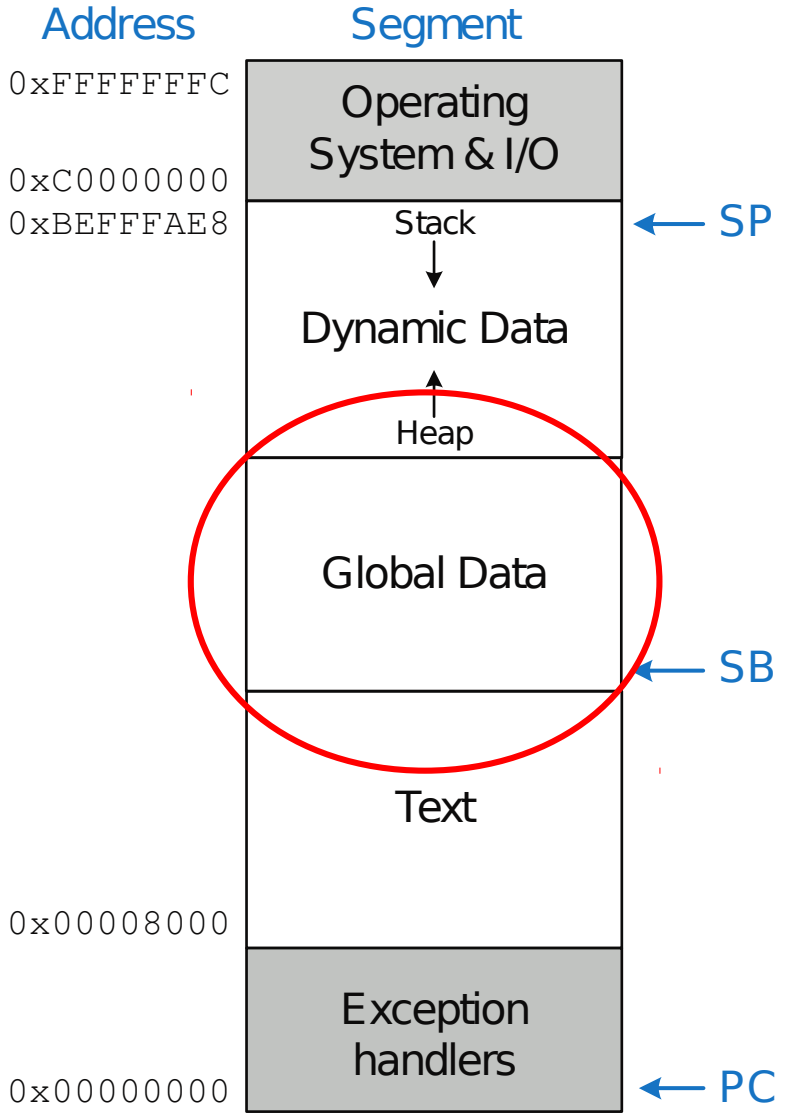
- El espacio de direcciones en ARM posee 32 bit
- Esto permite 2^{32} bytes (4GB)
- Las direcciones de los word entonces serán múltiplo de 4 en posiciones de 0 a 0xFFFFFFFC
- La Arquitectura ARM divide a este espacio de direcciones en 5 partes o segmentos.

Mapa de memoria



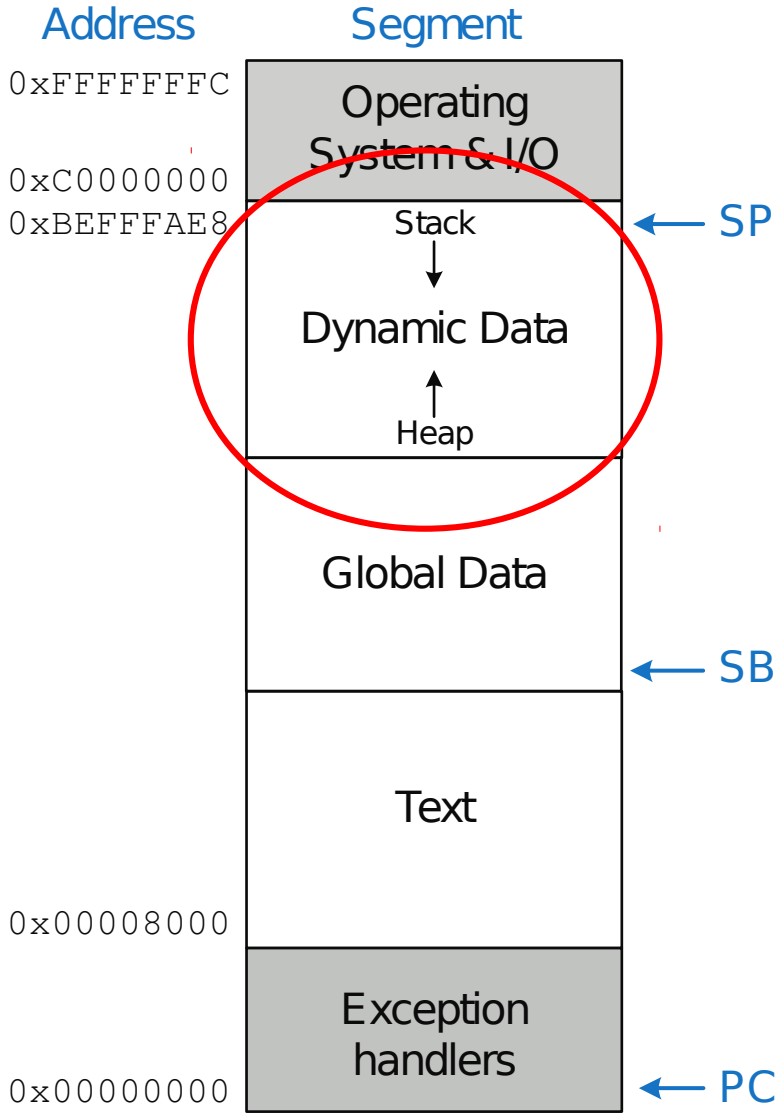
- **Segmento de texto.** segmento RO, guarda el código maquina del programa, puede incluir además literales (constantes) y datos de solo lectura.

Mapa de memoria



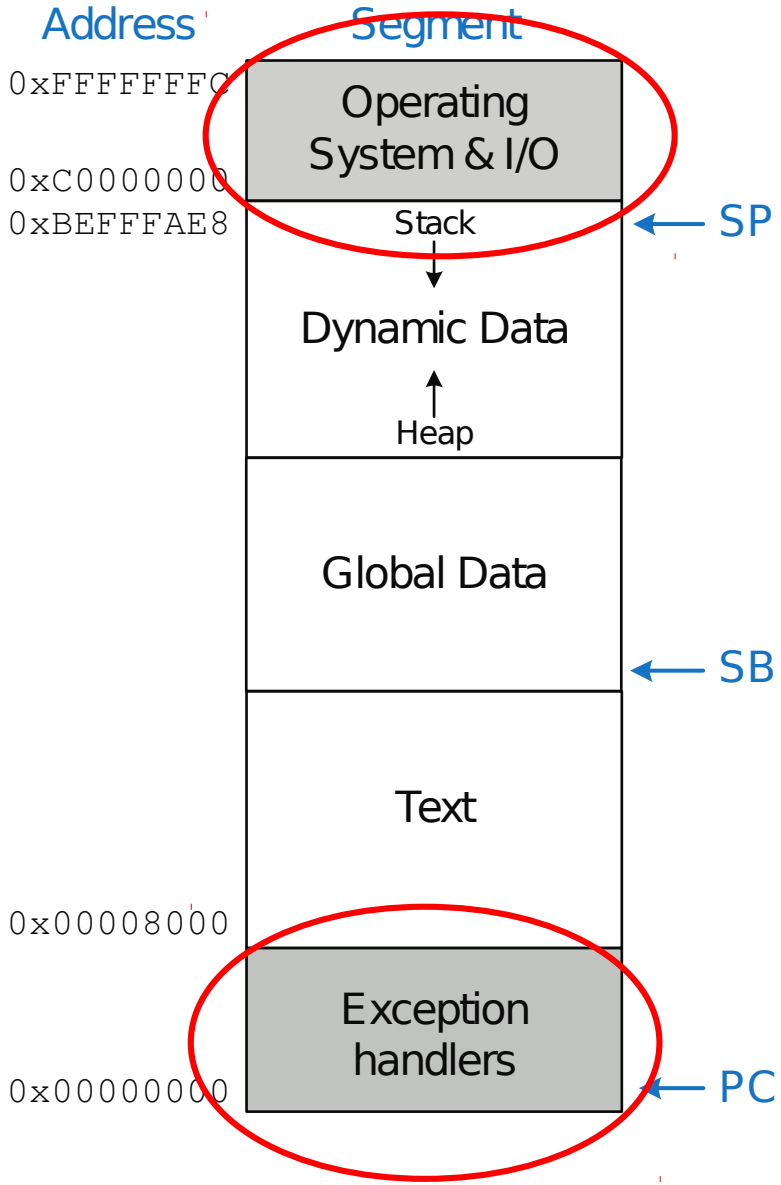
- **Segmento de texto**
- **Segmento de datos globales.** segmento RW, guarda las variables globales, Estas variables son ubicadas en la memoria antes de que el programa comience, ARM comúnmente usa el registro R9 (SB) como *Static Basic Pointer,* o puntero base para acceder a estas variables

Mapa de memoria



- **Segmento de texto**
- **Segmento de datos globales.**
- **Segmento de Datos Dinámicos**. Posee el Stack y el Heap este último encargado de almacenar las variables dinámicas generadas con malloc (C) o new (C++), generalmente crece en forma inversa al stack, el asignador de memoria es el encargado de no corromper la memoria, evitando el cruce con el Stack

Mapa de memoria



- **Segmento de texto**
- **Segmento de datos globales.**
- **Segmento de Datos Dinámicos**.
- **Manejador de Excepciones, OS y segmento de I/O.** La parte mas baja del mapa es reservada para el vector de excepciones y para los manejadores de excepciones, mientras que la parte mas alta para el sistema operativo y los dispositivos de I/O

Compilador

- El compilador es básicamente un traductor, traduce un lenguaje a otro lenguaje.
- En general el compilador traduce un Código escrito en lenguaje de Alto Nivel a Ensamblador.
- En los ejemplos se utilizará el GCC, compilador ampliamente difundido y libre, utilizado para compilar código a utilizarse en una *Raspberry Pi*.

Compilador

```
int f,g,y;

int sum(int a,int b)
{
   return (a+b);
}

int main(void)
{
   f=2;
   g=3;
   y=sum(f,g);
   return y;
}
```

gcc -O1 -S prog.c -o prog.s

```
        .text
        .global sum
        .type sum, %function
sum:    add r0, r0, r1
        bx lr
        .global main
        .type main, %function
main:   push {r3, lr}
        mov r0, #2
        ldr r3, .L3
        str r0, [r3, #0]
        mov r1, #3
        ldr r3, .L3+4
        str r1, [r3, #0]
        bl sum
        ldr r3, .L3+8
        str r0, [r3, #0]
        pop {r3, pc}
.L3:    .word f
        .word g
        .word y
```

Ensamblado

- Un ensamblador convierte el código en ensamblador en un archivo objeto que contiene el código en lenguaje máquina.
- En el Ensamblado se realizan dos pasadas al código
  - En la primera, se asignan las direcciones de la instrucciones y se encuentran los símbolos (etiquetas y nombre de variables globales), se crea entonces una tabla de estos símbolos con los nombres y sus direcciones.
  - En la Segunda pasada, se genera el código máquina, las direcciones para la etiquetas se toman de la tabla de símbolos. Luego código y tabla se guardan en el archivo objeto.

```
gcc -c prog.s -o prog.o

gcc -O1 -g -c prog.c -o prog.o
```

Ensamblado

```
SYMBOL TABLE:
00000000 l d .text   00000000 .text
00000000 l d .data   00000000 .data
00000000 g F .text   00000008 sum
00000008 g F .text   00000038 main
00000004 O *COM*  00000004 f
00000004 O *COM*  00000004 g
00000004 O *COM*  00000004 y
```

objdump -t prog.o

objdump -S prog.o

```
00000000 <sum>:
    int sum(int a, int b) {
    return (a + b);
    }
0: e0800001     add r0, r0, r1
4: e12fff1e     bx lr
00000008 <main>:
    int f, g, y; // global variables
    int sum(int a, int b);
    int main(void) {
8: e92d4008     push {r3, lr}
    f = 2;
C: e3a00002     mov r0, #2
10:e59f301c     ldr r3, [pc, #28]
14:e5830000     str r0, [r3] ; 34 <main+0x2c>
    g = 3;
18:e3a01003     mov r1, #3
1c:e59f3014     ldr r3, [pc, #20]
20:e5831000     str r1, [r3] ; 38 <main+0x30>
    y = sum(f,g);
24:ebfffffe bl 0 <sum>
28:e59f300c ldr r3, [pc, #12] ; 3c <main+0x34>
2c:e5830000 str r0, [r3]
    return y;
30: e8bd8008 pop {r3, pc}
...
```

Enlazado (*linking*)

- Un proyecto en general están compuesto por varios archivos fuentes, librerías y archivos de inicialización.
- Si se cambia uno de ellos es innecesario re compilar todos.

- El trabajo de el enlazador, es juntar todos los archivos objeto que componen un proyecto en un archivo en código máquina llamado ejecutable y asignar las direcciones de variables globales.
- El enlazador reubica los datos y las funciones de los archivos objetos para que no se superpongan, usa la información de sus tabla de símbolos para ajustar el código según la dirección asignada a las funciones y variables.

Ensamblado

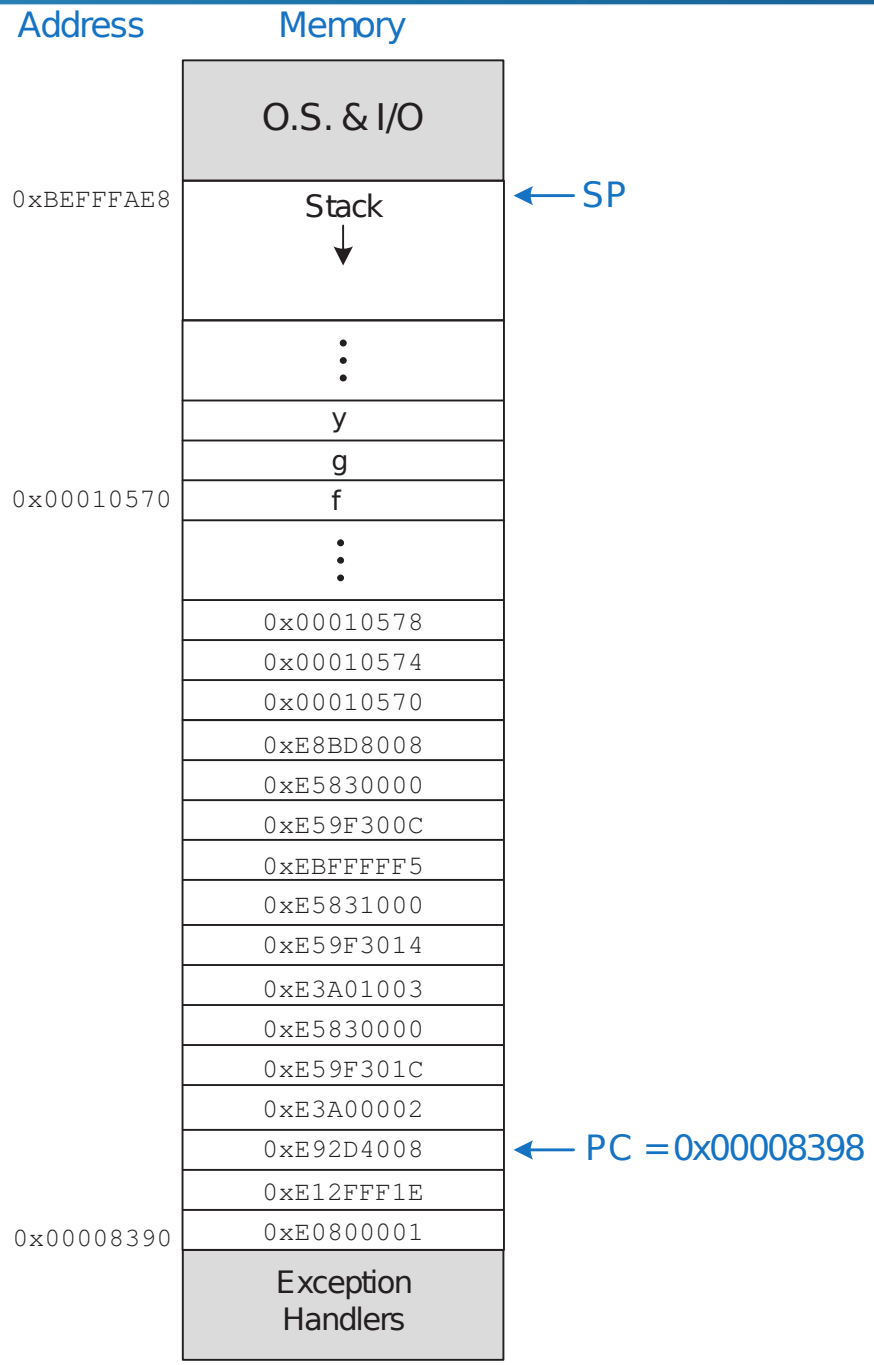
```
SYMBOL TABLE:
000082e4 l d   .text 00000000 .text
00010564 l d   .data 00000000 .data
00008390 g F   .text 00000008 sum
00008398 g F   .text 00000038 main
00010570 g O   .bss  00000004 f
00010574 g O   .bss  00000004 g
00010578 g O   .bss  00000004 y
```

objdump -S -t prog

```
00008390 <sum>:
int sum(int a, int b) {
return (a + b); }
8390: e0800001 add r0, r0,
8394: e12fff1e bx lr
00008398 <main>:
int f, g, y; // global variables
int sum(int a, int b);
int main(void) {
8398: e92d4008 push {r3, lr}
f = 2;
839c: e3a00002 mov r0, #2
83a0: e59f301c ldr r3, [pc, #28] ; 83c4 <
83a4: e5830000 str r0, [r3]
g = 3;
83a8: e3a01003 mov r1, #3
83ac: e59f3014 ldr r3, [pc, #20] ; 83c8 <main+0x30>
83b0: e5831000 str r1, [r3]
y = sum(f,g);
83b4: ebfffff5 bl 8390 <sum>
83b8: e59f300c ldr r3, [pc, #12] ; 83cc <main+0x34>
83bc: e5830000 str r0, [r3]
return y;
}
83c0: e8bd8008 pop {r3, pc}
83c4: 00010570 .word 0x00010570
83c8: 00010574 .word 0x00010574
83cc: 00010578 .word 0x00010578
```

Enlazado (*linking*)

- El sistema operativo carga un porgrama, leyendo el segmento text del ejecutable y lo copia al segmento correspondiente en memoria.
- Luego el SO salta al comienzo del programa.



- Bibliografía

  Harris & Harris. Digital design and computer architecture: ARM edition. Elsevier, 2015. Capítulo 6.

¿ Preguntas ?