ESTRUCTURA DE COMPUTADORES
Grado en Ingeniería Informática

Sesión de laboratorio número 3

FUNCIONES DEL PROGRAMA

Objetivos

- Construir funciones simples y llamarlas desde un programa.

Bibliografía

- D.A. Patterson y J. L. Hennessy, *Estructura y diseño de computadores*, Reverté, capítulo 2, 2011.

Introducción teórica

Funciones del programa

Las funciones del programa (*callee functions*) son la traducción de los métodos de Java o las funciones de C. La pareja de instrucciones **`jal eti`** (o llamada a función) y **`jr $ra`** (retorno de función), ligadas al registro **`$ra`** (**`$31`**), dan el soporte básico al flujo de ejecución.

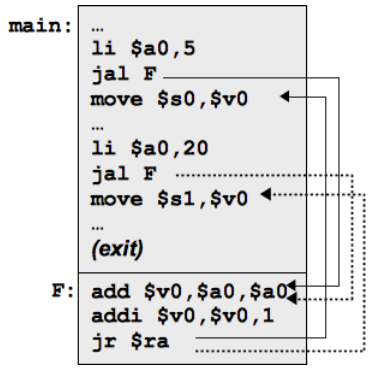

Figura 1. A la izquierda, aparece el esquema de un programa **`main()`** que llama desde dos puntos a una función **`F`** que en alto nivel se expresaría como **`int F(int a){return 2*a+1}`**. En los dos casos, la instrucción **`jal F`** guarda en el registro **`$ra`** la dirección de retorno, y por eso la función **`F`** acaba con una instrucción **`jr $ra`**. Las flechas de la figura muestran el flujo de ejecución: la primera llamada en continuo ⟶ y la segunda a trazos ⋯⋯▶.
El programa principal, por su parte, acaba con la llamada al sistema **`exit`**.

El convenio de uso de los registros también contempla la separación entre registros del programa y registros de la función. Los registros **`$s0`** a **`$s7`** están orientados a servir de variables globales del programa, y los registros **`$t0`** a **`$t9`** a variables locales del procedimiento. El convenio dice:

- Si el programa principal utiliza un registro **`$ti`**, ha de prever que cualquier función que llame podrá cambiar su contenido.
- Si una función necesita escribir en un registro **`$si`**, deberá de preservar su contenido previamente y restaurarlo antes de terminar.
- Si una función utiliza un registro **`$ti`**, deberá tener en cuenta que, entre dos ejecuciones de la misma función, cualquier otra función podrá modificar su contenido.

El convenio prevé, a su vez, la comunicación entre el programa principal y la función, y la regula considerando el número y tipo de datos intercambiados. Per ejemplo, si los argumentos son de tipo entero y no hay más de cuatro, irán por orden en los registros **$a0** a **$a3**. El valor retornado por la función, si es un entero, se escribirá en el registro **$v0**.

En resumen: en los ejercicios de estas prácticas conviene seguir las reglas de la tabla siguiente a la hora de programar.

Registros	Uso
$s0…$s7	El código del programa principal
$a0…$a3	Paso de parámetros del programa a las funciones
$t0…$t9	El código de la función
$v0	Retorno de resultados de las funciones a los programas

Tabla 1. Reglas del convenio de uso de los registros por parte de las aplicaciones

Funciones del programa y funciones del sistema

El uso de las funciones del programa es muy parecido al de las funciones del sistema; la diferencia más notable es que el código de las funciones del sistema está oculto, es independiente de los programas, es común para todos ellos y preserva el contenido de los registros globales y locales. En definitiva, no es necesario conocer la dirección donde se encuentran las funciones del sistema para poder utilizarlas.

Estas son las funciones del sistema útiles para esta pràctica.

Nombre	$v0	Descripción	Argumentos	Resultado
print_int	*1*	Imprime el valor de un entero	**$a0** = entero a imprimir	—
read_int	*5*	Lee el valor de un entero	—	**$v0** = entero leido
exit	*10*	Acaba el proceso	—	—
print_char	*11*	Imprime un carácter	**$a0** = carácter a imprimir	—

Tabla 2. Funciones del sistema que deben utilizarse en esta práctica.

Ejercicios de laboratorio

Ejercicio 1: La funciones de los programas

Abra y observe el código siguiente contenido en el fichero "03_exer_01.s". Note que sólo se encuentra el segmento de código (**.text**) y no hay ningún comentario. Los comentarios deberá añadirlos conforme vaya entendiendo lo que hace el programa.

```
          .globl __start
          .text 0x00400000
__start:  li $v0,5
          syscall
          move $a0,$v0
          li $v0,5
          syscall
          move $a1,$v0
          jal Mult
          move $a0,$v0
          li $v0,1
          syscall
          li $v0,10
          syscall
Mult:     li $v0, 0
```

```
          beqz $a1, MultRet
MultFor:  add $v0, $v0, $a0
          addi $a1, $a1, -1
          bne $a1, $zero, MultFor
MultRet:  jr $ra
```

En primer lugar, debe detectar qué instrucciones pertenecen al programa principal y cuáles a una función de nombre **`Mult`**.

- ¿Cuáles son las dos últimas instrucciones del programa principal?
- ¿Cuál es la última instrucción de la función?
- Busque las cuatro llamadas al sistema utilizadas en el programa. ¿Qué hace cada una?
- Busque un bucle dentro de la función. ¿Cuántas veces se ejecuta este bucle?
- ¿Qué hace la función exactamente?

Cargue el programa y ejecútelo. Note que la entrada/salida por la consola es muy pobre.

- ¿Sabe ejecutar el programa completo? Al ejecutarlo, tenga en cuenta que el programa pide la entrada de dos números por el teclado y luego imprime un resultado. Ahora bien, no habrá ningún mensaje que indique que se está esperando una entrada del teclado.
- ¿Sabe hacer una ejecución paso a paso?

Técnica experimental: uso de los *breakpoints*. Son muy útiles para detener el programa en un punto donde conviene inspeccionar los registros o la memoria sin tener que ir paso a paso desde el principio. Simplemente se le indica al simulador la dirección de la instrucción donde ha de detenerse la ejecución. Utilice la técnica anterior para detener la ejecución dentro de **`Mult`** y observar el valor de la dirección de retorno contenida en el registro **`$ra`**. Deberá indicar como punto de ruptura del flujo de ejecución la dirección de la instrucción **`jr $ra`**.

- ¿Cuál es el valor de la dirección de retorno?
- ¿A qué instrucción del programa apunta?

Ejercicio 2: Creación de funciones

Vamos a mejorar el diálogo del programa anterior a través de la consola. Esta mejora consiste en asociar el símbolo de una letra a cada valor que se lea o escriba. Así, puede nombrar el multiplicando como 'M', el multiplicador como 'Q' y el producto como 'R'. Debe escribir dos funciones que añadirá al programa del apartado anterior:

- Para la introducción de valores por el teclado. La función **`Input`** tiene como argumento el símbolo de la letra que vamos a escribir en la consola. La función debe escribir en la consola este símbolo seguido del carácter '**`=`**' y después leer un entero (el multiplicador o el multiplicando). La función debe devolver este valor leído.
- Para la impresión del resultado. La función **`Output`** tiene dos argumentos: la letra y el resultado (número entero) que se debe imprimir. La función debe escribir la letra, el carácter "**`=`**", el valor del resultado y el carácter de final de línea LF (*line feed*, valor 10 del código ASCII).

Para mayor claridad, expresaremos estas dos funciones en pseudocódigo:

```
int Input(char $a0) {
        print_char($a0);
        print_char('=');
        $v0=read_int();
        return($v0); }
```

```
void Output(char $a0, int $a1) {
        print_char($a0);
        print_char('=');
        print_int($a1);
        print_char('\n');
        return; }
```

Nótese que los argumentos recibidos por las funciones están almacenados en registros. Por ejemplo, la función **`Input`** recibe el carácter a imprimir en el registro **`$a0`**; de manera similar, **`Output`** recibe los dos argumentos (un carácter y un entero) en los registres **`$a0`** y **`$a1`**. Este detalle es muy importante: esta manera de pasar los parámetros a las funciones se llama por valor. En una práctica posterior modificaremos este ejemplo haciendo que las variables se ubiquen en la memoria principal y pasando como argumentos su dirección de memoria.

Cuando tenga hecha la codificación de las dos funciones **`Input`** y **`Output`**, deberá de reescribir completamente el cuerpo del programa principal para que rotule el multiplicando con la letra "M", el multiplicador con la "Q" y el resultado del producto con "R". El diálogo resultante debe aparecer en la consola como en la figura siguiente:

```
A=Input('M');
B=Input('Q');
C=Mult(A,B);
Output('R',C);
Exit();
```

```
M=215
Q=875
R=188125
```

Figura 2. A la izquierda el pseudocódigo del programa principal que debe escribir y a la derecha un ejemplo de diálogo resultante. En negrita, aparece el texto escrito por el programa. En cursiva, el texto tecleado por el usuario.

Ejercicio 3: Instrucciones condicionales

Nótese que la función **`Mult`** sólo funciona correctamente si el multiplicador Q es positivo. Pruebe a ejecutar el programa con *Q=–5*: el bucle de la función no acabará y deberá detenerlo mediante la combinación de teclas CTRL+C o bien pulsando el icono del menú rotulado con la palabra *Stop*.

En este apartado se le pide modificar ligeramente el programa principal para que si *Q<0*, en lugar de calcular *R=Mult(M,Q)* calcule *R=Mult(–M,–Q)*, es decir cambie el signo de ambos argumentos antes de llamar a la función a fin de mantener el resultado correcto. Si expresamos esta acción en pseudocódigo para una mayor claridad tenemos el siguiente:

```
M=Input('M');
Q=Input('Q');
If (Q<0)
        M=–M;
        Q=–Q;
R=Mult(M,Q);
Output('R',R);
Exit();
```

Figura 3. Una manera de resolver la limitación de **`Mult`** y poder operar con multiplicadores negativos

El punto fundamental aquí es descubrir cómo cambiar el signo de un número entero.

Ejercicios adicionales con el simulador

Puede hacerlos en el laboratorio, si le sobra tiempo, o acabarlos en casa.

Ejercicio 4: Iteraciones

1. Haga los cambios necesarios en el programa principal para que se repita el cálculo M×Q hasta que alguno de los dos operandos introducidos por el teclado valga cero, es decir, se trata de repetir la multiplicación mientras los dos operandos sean diferentes de cero. Eso mismo expresado en pseudocódigo:

```
repeat
        M=Input('M');
        Q=Input('Q');
        R=Mult(M,Q);
        Output('R',R);
while ((M≠0) && (Q≠0));
exit();
```

2. Diseñe un programa que pida un número *n* y escriba la tabla de multiplicar de *n*, desde *n*×1 hasta *n*×10. Para hacer la programación más sencilla puede utilizar una función `OutputM` el pseudocódigo de la cual se expresa a continuación:

```
void OutputM(int x, int y, int r) {
        print_int(x);
        print_char('x');
        print_int(y);
        print_char('=');
        print_int(r);
        print_char('\n');
}
```

Ejercicio 5: Selector

Escriba la función **`void PrintChar(char c)`**, que imprime en la consola un carácter siguiendo el estilo de C: entre comillas y mostrando los casos especiales **`'\n'`** (carácter ASCII número 10) y **`'\0'`** (carácter ASCII número 0).

```
void PrintChar(int x) {
    putchar('"');  /* comilla */
    switch (x){
        case 0:  print_char('\'); print_char('0'); break;
        case 10:    print_char('\'); print_char('n'); break;
        default: print_char(x);
    }
    putchar('"'); /* comilla */
}
```

Anexo

Ejemplos de control de flujo

En la tabla siguiente,

- Los símbolos *cond*, *cond1*, etc., hacen referencia a las seis condiciones simples (= y ≠, > y ≤, < y ≥) que relacionan dos valores contenidos en registros. El asterisco indica condición contraria; por ejemplo, si *cond* = "**>**" tenemos *cond** = "≤".
- En la columna de alto nivel, los símbolos *A*, *B*, etc. indican sentencias simples o compuestas; en la columna de bajo nivel, los símbolos **A**, **B**, etc. representan los bloques de instrucciones equivalentes en ensamblador.

Condicionales.

Alto nivel	Ensamblador

if (*cond1*)
  A;
else if (*cond2*)
  B;
else
  C;
D;
	`if:  bif (cond1*) elseif`
	`  A`
	`  j endif`
	`elseif:  bif (cond2*) else`
	`  B`
	`  j endif`
	`else:  C`
	`endif:  D`
	
	`if:  bif (cond1) then`
	`  bif (cond2) elseif`
	`  j else`
	`then:  A`
	`  j endif`
	`elseif:  B`
	`  j endif`
	`else:  C`
	`endif:  D`

if (*cond1* && *cond2*)
  A;
B;
	`if:  bif (cond1*) endif`
	`  bif (cond2*) endif`
	`  A`
	`endif:  B`

if (*cond1*||*cond2*)
  A;
B;
	`if:  bif (cond1) then`
	`  bif (cond2*) endif`
	`then:  A`
	`endif:  B`
	
	`if:  bif (cond1*) endif`
	`  bif (cond2*) endif`
	`  A`
	`endif:  B`

Selectores

Alto nivel	Ensamblador

switch (exp){
case X :
A;
break;
case Y :
case Z :
B;
break;
default:
C;
}
D;
	`bif (exp != X) caseY`
	`caseX: A`
	`j endSwitch`
	`caseY: bif (exp != Y) default`
	`caseZ: bif (exp != Z) default`
	`B`
	`j endSwitch`
	`default: C`
	`endSwitch: D`
	
	`bif (exp == X) caseX`
	`bif (exp == Y) caseY`
	`bif (exp == Z) caseZ`
	`j default`
	`caseX: A`
	`j endSwitch`
	`caseY:`
	`caseZ: B`
	`j endSwitch`
	`default: C`
	`endSwitch: D`

Iteraciones

Alt nivel	Ensamblador

while (*cond*)
A;
B;
	`while: bif (cond*) endwhile`
	`A`
	`j while`
	`endwhile B`

do
A;
while (*cond*)
B;
	`do: A`
	`bif (cond) do`
	`B`

do
A;
if(*cond1*) continue;
B;
if(*cond2*) break;
C;
while (*cond3*)
D;
	`do: A`
	`bif (cond1) while`
	`B`
	`bif (cond2) enddo`
	`C`
	`while: bif (cond3) do`
	`enddo: D`

iterar *n* veces /* *n*>0 */
A;
B;
	`li $r,n`
	`loop: A`
	`addi $r,$r,-1`
	`bgtz $r,loop`
	`B`

Llamadas al sistema del PCSpim

$v0	Nombre	Descripción	Argumentos	Resultado	Equivalente Java	Equivalente C
1	*print_integer*	Imprime (*) el valor de un entero	**$a0** = entero a imprimir	—	`System.out.print(int $a0)`	`printf("%d",$a0)`
2	*print_float*	Imprime (*) el valor de un *float*	**$f12** = float a imprimir	—	`System.out.print(float $f0)`	`printf("%f",$f0)`
3	*print_double*	Imprime (*) el valor de un *double*	**$f12** = double a imprimir	—	`System.out.print(double $f0)`	`printf("%Lf",$f0)`
4	*print_string*	Imprime una cadena de caracteres acabada en *nul* ('\0')	**$a0** = puntero a la cadena	—	`System.out.print(int $a0)`	`printf("%s",$a0)`
5	*read_integer*	Lee (*) el valor de un entero	—	**$v0** = entero leído		
6	*read_float*	Lee (*) el valor de un *float*	—	**$f0** = *float* leído		
7	*read_double*	Lee (*) el valor de un *double*	—	**$f0** = *double* leído		
8	*read_string*	Lee una cadena de caracteres (de longitud limitada) hasta encontrar un '\n' y la deja en el buffer acaba en *nul* ('\0')	**$a0** = puntero al buffer de entrada **$a1** = nombre máximo de caracteres de la cadena			
9	*sbrk*	Reservar un bloque de memoria del *heap*	**$a0** = longitud del bloque en bytes	**$v0** = dirección base del bloque de memoria		`malloc(integer n);`
10	*exit*		—	—		`exit(0);`
11	*print_character*		**$a0** = carácter a imprimir			`putc(char c);`
12	*read_character*			**$v0** = carácter leído		`getc();`

NOTAS

(*) El asterisco en las funciones 1, 2, 3, 5, 6 y 7 indica que, además de la operación de entrada/salida, hay un cambio de representación de binario a alfanumérico o de alfanumérico a binario

(**) En *pcspim-ES*, la función 12 lee un carácter del teclado sin producir un eco en la consola. En otras versiones del simulador sí escribe el eco

Codificación ASCII (ISO/IEC 8859-1)

Esta es la codificación utilizada por la consola y el teclado del simulador.

	_0	_1	_2	_3	_4	_5	_6	_7	_8	_9	_A	_B	_C	_D	_E	_F
0_	NUL 00 0	SOH 01 1	STX 02 2	ETX 03 3	EOT 04 4	ENQ 05 5	ACK 06 6	BEL 07 7	BS 08 8	HT 09 9	LF 0A 10	VT 0B 11	FF 0C 12	CR 0D 13	SO 0E 14	SI 0F 15
1_	DLE 10 16	DC1 11 17	DC2 12 18	DC3 13 19	DC4 14 20	NAK 15 21	SYN 16 22	ETB 17 23	CAN 18 24	EM 19 25	SUB 1A 26	ESC 1B 27	FS 1C 28	GS 1D 29	RS 1E 30	US 1F 31
2_	(SP) 20 32	! 21 33	" 22 34	# 23 35	$ 24 36	% 25 37	& 26 38	' 27 39	(28 40	) 29 41	* 2A 42	+ 2B 43	, 2C 44	- 2D 45	. 2E 46	/ 2F 47
3_	0 30 48	1 31 49	2 32 50	3 33 51	4 34 52	5 35 53	6 36 54	7 37 55	8 38 56	9 39 57	: 3A 58	; 3B 59	< 3C 60	= 3D 61	> 3E 62	? 3F 63
4_	@ 40 64	A 41 65	B 42 66	C 43 67	D 44 68	E 45 69	F 46 70	G 47 71	H 48 72	I 49 73	J 4A 74	K 4B 75	L 4C 76	M 4D 77	N 4E 78	O 4F 79
5_	P 50 80	Q 51 81	R 52 82	S 53 83	T 54 84	U 55 85	V 56 86	W 57 87	X 58 88	Y 59 89	Z 5A 90	[5B 91	\ 5C 92	] 5D 93	^ 5E 94	_ 5F 95
6_	` 60 96	a 61 97	b 62 98	c 63 99	d 64 100	e 65 101	f 66 102	g 67 103	h 68 104	i 69 105	j 6A 106	k 6B 107	l 6C 108	m 6D 109	n 6E 110	o 6F 111
7_	p 70 112	q 71 113	r 72 114	s 73 115	t 74 116	u 75 117	v 76 118	w 77 119	x 78 120	y 79 121	z 7A 122	{ 7B 123	| 7C 124	} 7D 125	~ 7E 126	DEL 7F 127
8_	PAD 80 128	HOP 81 129	BPH 82 130	NBH 83 131	IND 84 132	NEL 85 133	SSA 86 134	ESA 87 135	HTS 88 136	HTJ 89 137	VTS 8A 138	PLD 8B 139	PLU 8C 140	RI 8D 141	SS2 8E 142	SS3 8F 143
9_	DCS 90 144	PU1 91 145	PU2 92 146	STS 93 147	CCH 94 148	MW 95 149	SPA 96 150	EPA 97 151	SOS 98 152	SGCI 99 153	SCI 9A 154	CSI 9B 155	ST 9C 156	OSC 9D 157	PM 9E 158	APC 9F 159
A_	(NBSP) A0 160	¡ A1 161	¢ A2 162	£ A3 163	¤ A4 164	¥ A5 165	¦ A6 166	§ A7 167	¨ A8 168	© A9 169	ª AA 170	« AB 171	¬ AC 172	(SHY) AD 173	® AE 174	 AF 175
B_	° B0 176	± B1 177	² B2 178	³ B3 179	´ B4 180	µ B5 181	¶ B6 182	· B7 183	¸ B8 184	¹ B9 185	º BA 186	» BB 187	¼ BC 188	½ BD 189	¾ BE 190	¿ BF 191
C_	À C0 192	Á C1 193	Â C2 194	Ã C3 195	Ä C4 196	Å C5 197	Æ C6 198	Ç C7 199	È C8 200	É C9 201	Ê CA 202	Ë CB 203	Ì CC 204	Í CD 255	Î CE 206	Ï CF 207
D_	Ð D0 208	Ñ D1 209	Ò D2 210	Ó D3 211	Ô D4 212	Õ D5 213	Ö D6 214	× D7 215	Ø D8 216	Ù D9 217	Ú DA 218	Û DB 219	Ü DC 220	Ý DD 221	Þ DE 222	ß DF 223
E_	à E0 224	á E1 225	â E2 226	ã E3 227	ä E4 228	å E5 229	æ E6 230	ç E7 231	è E8 232	é E9 233	ê EA 234	ë EB 235	ì EC 236	í ED 237	î EE 238	ï EF 239
F_	ð F0 240	ñ F1 241	ò F2 242	ó F3 243	ô F4 244	õ F5 245	ö F6 246	÷ F7 247	ø F8 248	ù F9 249	ú FA 250	û FB 251	ü FC 252	ý FD 253	þ FE 254	ÿ FF 255
	_0	_1	_2	_3	_4	_5	_6	_7	_8	_9	_A	_B	_C	_D	_E	_F

Las celdas sombreadas corresponden a caracteres de control no imprimibles. (**SP)** denota el espacio entre palabras, (**NBSP)** significa *non-breaking space* y (**SHY)** *syllabe hyphen*.