ESTRUCTURA DE COMPUTADORS
Grau en Enginyeria Informàtica

Sessió de laboratori número 3

FUNCIONS DEL PROGRAMA

Objectius

- Construir funcions simples i cridar-les des d'un programa.

Bibliografia

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

Introducció teòrica

Funcions del programa

Les funcions del programa (*callee functions*) són la traducció dels mètodes de Java o les funcions de C. El parell d'instruccions **`jal eti`** (o crida a la funció) i **`jr $ra`** (el retorn de la funció), lligades pel registre **`$ra`** (**`$31`**), donen el suport bàsic al flux d'execució.

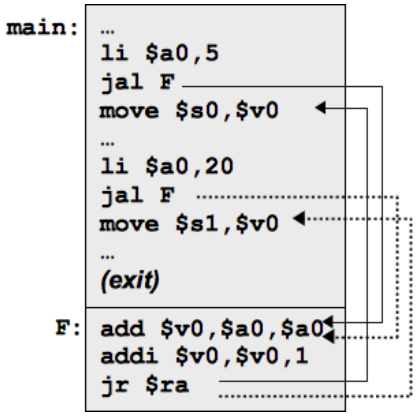

Figura 1. A l'esquerra, teniu l'esquema d'un programa **`main()`** que crida des de dos punts a una funció **`F`** que en alt nivell s'expressaria com **`int F(int a){return 2*a+1}`**. En tots dos casos, la instrucció **`jal F`** guarda en el registre **`$ra`** l'adreça de retorn, i per això la funció **`F`** acaba amb una instrucció **`jr $ra`**. Les fletxes de la figura en mostren el flux d'execució: la primera crida en continu ——▸ i la segona a traços ·····▸. El programa principal, per la seua banda, acaba amb la crida al sistema **`exit`**.

El conveni d'ús del registres també preveu la separació entre registres del programa i registres de la funció. Els registres **`$s0`** a **`$s7`** estan orientats a servir de variables globals del programa, i els registres **`$t0`** a **`$t9`** a variables locals del procediment. El conveni diu:

- Si el programa principal utilitza un registre **`$ti`**, ha de preveure que qualsevol funció que cride podrà canviar-ne el contingut.
- Si una funció necessita escriure en un registre **`$si`**, haurà de preservar-ne el contingut previ i restaurar-lo abans d'acabar.
- Si una funció fa servir un registre **`$ti`**, haurà de tindre en compte que, entre dues execucions de la mateixa funció, qualsevol altra funció podrà modificar-ne el contingut.

El conveni també preveu la comunicació entre el programa principal i la funció, i la regula en funció del nombre i tipus de les dades intercanviades. Per exemple, si els arguments són de tipus enter i no hi ha més de quatre, aniran per ordre en els registres **`$a0`** a **`$a3`**. El valor retornat per la funció, si és un enter, s'escriurà en el registre **`$v0`**.

En resum: en els exercicis d'aquestes pràctiques, us convé seguir les regles de la taula següent a l'hora de programar.

Registres	Ús
`$s0...$s7`	El codi del programa principal
`$a0...$a3`	Pas de paràmetres del programa a les funcions
`$t0...$t9`	El codi de la funció
`$v0`	Retorn de resultats de les funcions als programes

Taula 1. Regles del conveni d'ús dels registres per part de les aplicacions

Funcions del programa i funcions del sistema

L'ús de les funcions del sistema és ben paregut al de les funcions del programa; la diferència més notable és que el codi de les funcions del sistema està amagat, és independent dels programes, és comú per a tots ells i preserva el contingut dels registres globals i locals. En definitiva, no cal conèixer l'adreça on es troben les funcions del sistema per a poder fer-ne ús.

Estas son las funciones del sistema útiles para esta práctica:

Nom	$v0	Descripció	Arguments	Resultat
print_int	*1*	Imprimeix el valor d'un enter	**`$a0`** = enter per a imprimir	—
read_int	*5*	Llig el valor d'un enter	—	**`$v0`** = enter llegit
exit	*10*	Acaba el procés	—	—
print_char	*11*	Imprimeix un caràcter	**`$a0`** = caràcter per a imprimir	—

Taula 2. Funcions del sistema que cal utilitzar en aquesta pràctica.

Exercicis de laboratori

Exercici 1: Les crides al sistema i les funcions dels programes

Obriu i observeu el codi següent contingut al fitxer "03_exer_01.s". Noteu que només hi ha el segment de codi (**`.text`**) i no trobem cap comentari. Els comentaris els haureu d'afegir segons aneu entenent què fa 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 lloc, heu de detectar quines instruccions pertanyen al programa principal i quines a una funció de nom **`Mult`**.

- Quines són les dues últimes instruccions del programa principal?
- Quina és l'última instrucció de la funció?
- Busqueu-hi les quatre crides al sistema emprades en el programa. Què fa cadascuna?
- Busqueu un bucle dins de la funció. Quantes vegades s'executa aquest bucle?
- Busqueu un bucle dins de la funció. ¿Cuántas veces se ejecuta este bucle?
- Què fa la funció exactament?

Carregueu el programa i executeu-lo. Notareu que l'entrada/eixida per la consola és molt pobre.

- Sabeu executar el programa complet? En executar-lo, tingueu en compte que el programa demana l'entrada de dos nombres pel teclat i després imprimeix un resultat. Ara bé, no hi haurà cap missatge que us indique que s'està esperant una entrada del teclat.
- Sabeu fer una execució pas a pas?

Tècnica experimental: ús dels *breakpoints*. És molt útil per a detenir el programa en un punt on convé inspeccionar els registres o la memòria sense haver-hi d'anar pas a pas des del principi. Hi ha prou a indicar-li al simulador l'adreça de la instrucció on ha d'aturar-se l'execució. Feu servir la tècnica anterior per a detenir l'execució dins de **`Mult`** i observar el valor de l'adreça de retorn contingut en el registre **`$ra`**. Haureu d'indicar com a punt de trencament del flux d'execució l'adreça de la instrucció **`jr $ra`**.

- Quin és el valor de l'adreça de retorn?
- A quina instrucció del programa apunta?

Exercici 2: Creació de funcions

Anem a millorar el diàleg del programa anterior a través de la consola fent que tot just abans de cada lectura o escriptura la consola mostre un text format per dos caràcters: una lletra arbitrària i un símbol '='. Així, el programa demanarà el multiplicand escrivint 'M=', el multiplicador amb 'Q=' i retolarà el resultat com 'R=...'.

Heu d'escriure dues funcions que haureu d'afegir al programa de l'apartat anterior:

- Per a la lectura de valors pel teclat: La funció **`Input`** té com a argument un caràcter. La funció ha d'escriure aquest caràcter en la consola seguit pel caràcter '**`=`**' i després llegir un enter. La funció ha de tornar aquest enter llegit.
- Per a la impressió del resultat: La funció **`Output`** té dos arguments: el caràcter i un valor enter. La funció ha d'escriure aquest caràcter, el símbol "**`=`**", el valor del resultat i el caràcter de final de línia LF (*line feed*, valor 10 del codi ASCII).

Per a major claredat, expressarem aquestes dues funcions en pseudocodi:

```
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; }
```

Noteu que els arguments rebuts per les funcions estan emmagatzemats en registres. Per exemple, la funció **`Input`** rep el caràcter a imprimir en el registre **`$a0`**; de manera similar, **`Output`** rep els dos arguments (un caràcter i un enter) en els registres **`$a0`** i **`$a1`**. Aquest detall és molt important: aquesta manera de passar els paràmetres a les funcions s'anomena **per valor**. En una pràctica posterior modificarem aquest exemple fent que les variables s'ubiquen en la memòria principal i passant com a arguments la seua adreça de memòria.

Quan tingueu feta la codificació de les dues funcions **`Input`** i **`Output`**, haureu de reescriure completament el cos del programa principal perquè retole el multiplicand amb la lletra "M", el multiplicador amb la "Q" i el resultat del producte amb "R". El diàleg resultant ha d'aparèixer en la consola com en la figura 2:

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

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

Figura 2. A l'esquerra teniu l'esquema en pseudocodi del programa principal que heu d'escriure i a la dreta un exemple de diàleg resultant. En negreta, apareix el text escrit pel programa. En cursiva, el text teclejat per l'usuari.

Exercici 3: Instruccions condicionals

Noteu que la funció **`Mult`** només funciona correctament si el multiplicador Q és positiu. Proveu a executar el programa amb *Q=–5*: el bucle de la funció s'allargarà i caldrà detenir-lo mitjançant la combinació de tecles Ctrl+C o bé polsar la icona del menú retolada amb la paraula *Stop*.

En aquest apartat us demanem modificar lleugerament el programa principal per tal que si *Q<0*, en comptes de calcular *R=Mult(M,Q)* calcule *R=Mult(–M,–Q)*, això és, canvie el signe d'ambdós arguments abans de cridar la funció per tal de mantenir el resultar correcte. Si expressem aquesta acció en pseudocodi per a major claredat tenim el següent:

```
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 resoldre la limitació de **`Mult`** i poder operar amb multiplicadors negatius

El punt fonamental ací és esbrinar com canviar el signe d'un nombre enter.

Exercicis addicionals amb el simulador

Podeu fer-los en el laboratori, si us sobra temps, o acabar-los a casa.

Exercici 4: Iteracions

1. Feu els canvis necessaris en el programa principal que tal de repetir el càlcul M×Q fins que algun dels dos operands introduïts pel teclat valga zero, això és, es tracta de repetir la multiplicació mentre els dos operands siguen diferents de zero. Això mateix expressat en pseudocodi:

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

2. Dissenyeu un programa que demane per un nombre *n* i escriga la taula de multiplicar de *n*, des de *n*×1 fins *n*×10. Per tal de fer la programació més senzilla podeu emprar la funció `OutputM` que descriu el pseudocodi següent:

```
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');
}
```

Exercici 5: Selector

Escriviu la funció **`void PrintChar(char c)`**, que imprimeix en la consola un caràcter seguint l'estil de C: entre cometes i mostrant els casos especials **'\n'** (caràcter ASCII número 10) i **'\0'** (caràcter ASCII número 0).

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

Annex

Exemples de control de flux

En les taules següents,

- Els símbols *cond*, *cond1*, etc., fan referència a les sis condicions simples (= i ≠, > i ≤, < i ≥) que relacionen dos valors continguts en registres. L'asterisc indica condició contrària; per exemple, si *cond* = "**>**" tenim *cond** = "≤".
- En la columna d'alt nivell, els símbols *A*, *B*, etc. indiquen sentències simples o compostes; en la columna de baix nivell, els símbols **A**, **B**, etc. representen els blocs d'instruccions equivalents en assemblador.

Condicionals.

<table>
<tr><th>Alt nivell</th><th>Assemblador</th></tr>
<tr><td>if (cond1)
  A;
else if (cond2)
  B;
else
  C;
D;</td><td>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</td></tr>
<tr><td>if (cond1 && cond2)
  A;
B;</td><td>if: bif (cond1*) endif
    bif (cond2*) endif
    A
endif: B</td></tr>
<tr><td>if (cond1||cond2)
  A;
B;</td><td>if: bif (cond1) then
    bif (cond2*) endif
then: A
endif: B

if: bif (cond1*) endif
    bif (cond2*) endif
    A
endif: B</td></tr>
</table>

Selectors

Alt nivell	Assemblador

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`

Iteracions

Alt nivell	Assemblador

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

Crides al sistema del PCSpim

$v0	Nom	Descripció	Arguments	Resultat	Equivalent Java	Equivalent C
1	*print_integer*	Imprimeix (*) el valor d'un enter	`$a0` = enter per a imprimir	—	`System.out.print(int $a0)`	`printf("%d",$a0)`
2	*print_float*	Imprimeix (*) el valor d'un *float*	`$f12` = float per a imprimir	—	`System.out.print(float $f0)`	`printf("%f",$f0)`
3	*print_double*	Imprimeix (*) el valor d'un *double*	`$f12` = double per a imprimir	—	`System.out.print(double $f0)`	`printf("%Lf",$f0)`
4	*print_string*	Imprimeix una cadena de caràcters acabada en nul ('\0')	`$a0` = punter a la cadena	—	`System.out.print(int $a0)`	`printf("%s",$a0)`
5	*read_integer*	Llig (*) el valor d'un enter	—	`$v0` = enter llegit	`$f0 = scan.nextInt()`	`scanf("%d",&$v0)`
6	*read_float*	Llig (*) el valor d'un *float*	—	`$f0` = *float* llegit	`$f0 = scan.nextFloat()`	`scanf("%f",&$f0)`
7	*read_double*	Llig (*) el valor d'un *double*	—	`$f0` = *double* llegit	`$f0 = scan.nextDouble()`	`scanf("%Lf",&$f0)`
8	*read_string*	Llig una cadena de caràcters (de llargària limitada) fins trobar un '\n' i la desa en el buffer acaba en nul ('\0')	`$a0` = punter al buffer d'entrada `$a1` = nombre màxim de caràcters de la cadena	—	`$v0 = scan.nextLine()`	`fgets($a0,$a1,STDIN)`
9	*sbrk*	Reservar un bloc de memòria del *heap*	`$a0` = longitud del bloc en bytes	`$v0` = adreça base del bloc de memòria	`new` (**)	`$v0 = malloc($a0);`
10	*exit*	Fi de procés	—	—	`System.exit(0);`	`exit(0);`
11	*print_character*	Imprimeix un caràcter	`$a0` = caràcter per a imprimir	—	`System.out.print(char $a0)`	`putc($a0);`
12	*read_character*	Llig un caràcter		`$v0` = caràcter llegit	`$v0 = System.in.read()`	`$v0 = getc();`

NOTES

(*) L'asterisc en *Imprimeix** i *Llig** indica que, a més a més de l'operació d'entrada/eixida, hi ha un canvi de representació de binari a alfanumèric o d'alfanumèric a binari.

(**) En *pcspim-ES*, la funció 12 llig un caràcter del teclat sense produir un eco en la consola. En altres versions del simulador sí escriu l'eco.

Codificació ASCII (ISO/IEC 8859-1)

Aquesta és la codificació utilitzada per la consola i el teclat de PC-SPIM.

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

Las cel·les ombrejades corresponen a caràcters de control no imprimibles. (**SP)** denota l'espai entre paraules, (**NBSP)** significa *non-breaking space* y (**SHY)** *syllabe hyphen*.